

ALGORITHMS FOR RHOTRIX MULTIPLICATION ON TWO-
DIMENSIONAL PROCESS GRID TOPOLOGIES

BY

EZUGWU EL-SHAMIR ABSALOM, B.Sc. (ABU, 2006)
MSC/SCIEN/00043/2008-2009

A THESIS SUBMITTED TO THE
POSTGRADUATE SCHOOL,
AHMADU BELLO UNIVERSITY, ZARIA
NIGERIA

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE AWARD OF
THE DEGREE OF MASTER OF SCIENCE (M.Sc.) IN COMPUTER SCIENCE

DEPARTMENT OF MATHEMATICS
AHMADU BELLO UNIVERSITY, ZARIA
NIGERIA

JANUARY, 2011

DECLARATION

I declare that the work in the thesis entitled '*Algorithms for Rhotrix Multiplication on Two-Dimensional Process Grid Topologies*' has been carried out by me in the Department of Mathematics under the supervision of Prof. S. B. Junaidu (Director IACC, Ahmadu Bello University, Zaria-Nigeria).

The information derived from the literature has been duly acknowledged in the text and a list of references provided. No part of this thesis was previously presented for another degree or diploma at any other University to the best of my knowledge.

Ezugwu El-Shamir Absalom

Date

CERTIFICATION

This thesis entitled “ALGORITHMS FOR RHOTRIX MULTIPLICATION ON TWO-DIMENSIONAL PROCESS GRID TOPOLOGIES” by EZUGWU, Absalom El-Shamir meets the regulations governing the award of the degree of Master of Science of Ahmadu Bello University, Zaria, and is approved for the contribution to knowledge and literary presentation.

Prof. S. B. Junaidu
Major Supervisor

Date

Dr. S. E. Abdullahi
Minor Supervisor

Date

External Examiner

Date

Dr. A. A. Tijani
Head of Department

Date

Prof. A. A. Joshua
Dean of Postgraduate School

Date

ACKNOWLEDGEMENT

First of all I would like to thank **God Almighty** for the grace and strength provided to me for completing this course. The research work would not have been completed without the help of few intelligent scholars. Also it is a pleasure to acknowledge the free rein given to me by Prof. Sahalu Balarabe Junaidu (Director Iya Abubakar Computer Centre Ahmadu Bello University, Zaria-Nigeria) whom against all odds has sacrificed time and pleasure in pursuing this Study. I thank him as well as Dr. S. E. Abdullahi, Dr.A.O Ajibade, Dr. A. A. Obiniyi, Dr. A. M. Ibrahim, Dr. B. Sani, Dr. A. Mohammed and Dr M. I. Buhari (of Department of Computer Engineering Brunel University) for their discussions, suggestions and encouragement.

Many colleagues chipped in with words of wisdom at different stages of my studies and it was a great time with them around: special mention goes to Mr. Abdullahi Mohammed, Mr. Peter C. Matthew, Dr. Peter Yusuf Ofemile (General Hospital Gombe State), Mr. Donfack Kana Armand Florentin, Mr. Ngene Chibuike, Mr. Ibrahim Ayodele Kadir, (Department of Computer Science, Federal Polytechnic Kaduna), Mr. Shehu Mohammed Tukur, Mr. Baroon Ismail Mohammad, Mr. Mustafa Bagiwa, Miss Akudo Anyawu, Miss Nneoma Ayanwu and Miss Chidnma Nnaji,.

My special thanks goes to Baba Yunus Mohammed (CEO Open Press Ghana) who sponsored this programme right from its inception and stood by me to the end.

Finally, I dedicate this thesis to my parents, brothers and sister, Pius and Suzan Ezugwu, Celestine Pius, Sunday Pius, Ikechukwu Pius, Ahmed Pius, and Abdullamid Pius and Ngozi Pius, Aisha Pius and Abigail Pius, who have jovially supported me and my endeavours and whose vision in fact shaped me.

ABSTRACT

This research work is focused on designing multiple sequential and parallel row-column and heart-oriented rhotrix multiplication algorithms. The algorithms have been implemented on process array and heterogeneous master-worker platforms. The first implementation is the row-column rhotrix multiplication on processor arrays. The second is the heart-oriented rhotrix multiplication on dynamic master-worker with allocation of rhotrix vectors where the master distributes rhotrix vectors (data) and computations to the workers. We hypothesised that no single algorithm always achieves the best possible performance for multiplying rhotrices with different sizes on arbitrary process grids. The performance models for these algorithms and the experimental results on clusters of workstations support the research hypothesis. Furthermore, initial heuristics for the poly-algorithmic selection for parallel arbitrary rhotrix multiplication was provided. The two multiplication methods identified were experimentally implemented using C, Java and Delphi programming languages. The obtained experimental results on both platforms demonstrate the possible parallelisation of rhotrices, and idealistic views of their future applications in research and industries.

TABLE OF CONTENTS

TITLE PAGE.....	I
DECLARATION.....	II
CERTIFICATION.....	III
ACKNOWLEDGEMENT.....	IV
ABSTRACT.....	V
TABLE OF CONTENTS.....	VI
CHAPTER ONE: INTRODUCTION.....	1
1.1. Background to the study.....	1
1.2. Research Motivation.....	4
1.3. Research Objectives.....	4
1.4. Research Methodology.....	5
1.5. Organization of the Thesis.....	5
1.6. Contribution to Knowledge.....	6
1.7. Benefits of this Research Work to Society.....	6
CHAPTER TWO: LITERATURE REVIEW.....	7
2.1. Introduction.....	7
2.2. Parallel Computing Overview.....	7
2.2.1. Some Concepts and Terminologies.....	10
2.2.2. Message Passing Model.....	13
2.2.3. Data Parallel Model.....	14
2.3. Analysing Sequential Algorithm.....	15
2.4. Analysing Parallel Algorithms.....	16
2.5. Review of Cannon's Algorithms and Block Rhotrix Operations.....	17
2.5.1. Rhotrix-Rhotrix Multiplication: Cannon's Algorithm Method.....	21
2.5.2. Communication Steps in Cannon's Algorithm.....	22
2.5.3. Systolic Array.....	27
2.6. Background Concept from Group Theory to Complex Analysis.....	30
2.7. A Review of Strassen's Algorithm.....	30
2.8. Conversion of Rhotrix to Coupled Matrix.....	36
2.8.1. Solving $(m \times n) \times (n \times m)$ and $(m-1) \times (n-1)$ Coupled Matrix Problems.....	37

2.8.2. Parallel Computing with Processor Array Overview	41
2.8.3. Architectural Considerations in Array Processor Design	41
2.8.4. Mapping Parallel Algorithms onto Locally Interconnected Computing Networks	42
2.9. Conclusion	43
CHAPTER THREE: SEQUENTIAL AND PARALLEL ALGORITHMIC DESIGN FOR RHOTRIX MULTIPLICATION	
3.1. Introduction.....	44
3.2. The Algebra of Vector dot-product	44
3.2.1. Heart-Oriented Rhotrix Multiplication	45
3.2.2. Row-Wise Heart-Oriented Rhotrix Multiplication	56
3.3. Row-Column Multiplication of N-Dimensional Rhotrices	61
3.3.1. Row-Column Definition and Formula Expression for N-Dimensional Rhotrix Justification of adopted method: from multiset point of view	64
3.3.2. Row-Column Multiplication Techniques.....	65
3.3.3. Generalized Form of Row-Column Multiplication Technique.....	69
3.4. Application of Cannon's Algorithm in Rhotrix Row-Column Multiplication ...	71
3.4.1. Communication Steps in Cannon's Algorithm.....	75
3.4.2. Analysis of Cannon Algorithm.....	79
3.4.3. Theoretical Computation of Cannon Algorithm.....	79
3.5. Verification of Strassen's Algorithm by Divide-and-Conquer Technique	83
3.5.1. Strassen's Algorithm and Rhotrix Row-Column Multiplication	85
3.5.2. Strassen's Algorithm Computation Approach.....	91
3.5.3. Strassen's Algorithm Fixup Case.....	93
3.5.4. Program Task Graph for Strassen's Algorithm.....	94
3.6. Contribution Summary	99
3.7. Parallel Multiplication of Rhotrices.....	99
3.8. Architecture and Data-parallel Operations for Processor Array.....	100
3.8.1. Structure of Processing Interconnection Network	101
3.8.2. Architectural Features of Processor Arrays (Enabling and Disabling processors)	102
3.9. The Mapping Problem	104
3.9.1. Design of Processor Array Mapping Techniques	104

3.9.2. Linear Mapping of Processing Element.....	106
3.9.3. Communication Vectors	106
3.9.4. Linear Mapping.....	107
3.9.5. Selection of Scheduling Vector S^T Based on Scheduling Inequalities	107
3.9.6. Rhotrix Multiplication and 2-D Processor Array Design	108
3.10. Rhotrix Multiplication on Process Array Grid	111
3.10.1. Pseudocode for N-Dimension Rhotrix Multiplication on Process Array Grid	111
3.10.2. Input Data Movement Pattern for (Coupled matrix).....	112
3.10.3. Input Data Movement Pattern for Rhotrix Elements	114
3.11. Row-column Multiplication on Master-Worker Platform (Coupled Matrix).....	117
3.11.1. Application Process and Input Data Communication	118
3.11.2. Master-Worker Network Topology Platform	119
3.11.3. Minimization of the Communication Volume	120
3.11.4. The Maximum Re-use Algorithm	122
3.12. Algorithms for Master-worker Platforms (Coupled Matrix)	123
3.12.1. Principle of the Algorithm Design	124
3.13. Program Task Graph for Heart-Oriented Rhotrix Algorithm	127
3.14. Memory Usage.....	130
3.15. Heart-Oriented MPI Experimental Performance Model.....	130
3.16. Heart-Oriented Scalability Analysis Performance Model	132
3.17. Task-Parallel Approach for Heart-Oriented Algorithm Using MPI	133
3.18. A Framework of Task-Parallel Approach for Strassen's Algorithm Using MPI	135
3.19. Data Structures.....	136
3.20. Computational Time Complexity Analysis for Rhotrix Multiplication.....	139
3.20.1. Computational Time Complexity for Row-Column Multiplication	139
3.20.2. Computational Time Complexity for Heart-Oriented Multiplication.....	140
3.21. Conclusion	141
CHAPTER FOUR: IMPLEMENTATION AND PERFORMANCE RESULTS	142
4.1. Introduction.....	142
4.2. MPI Performance Results	142
4.3. Performance Analyzer	147
4.4. Process Array Simulation for Coupled Matrix Implementation	152

4.5. Process Array Simulation for Row-Column Algorithm Implementation	157
4.5. Conclusion	162
CHAPTER FIVE: SUMMARY, CONCLUSION AND FUTURE WORK	163
5.1. Summary and Conclusion	163
5.2. Future Work	164
REFERENCES	165

LIST OF FIGURES

Figure 2.1: Parallel computing process structure	7
Figure. 2.2a: Parallel computing case studies	9
Figure. 2.2b: Areas of application of parallel computing	10
Figure 2.3: Parallel programming model (message passing model)	14
Figure 2.4: Parallel programming model (data parallel model).....	15
Figure 2.5a: Blocking process for fifteen dimensional rhotrix R_{15} with main entries a and hearts d	20
Figure 2.5b: blocking process for fifteen dimensional rhotrix q_{15} with main entries b and hearts e	21
Figure 2.6a: First block rhotrix multiplication step. A, B and D, E after initial alignment	23
Figure 2.6b: Final block rhotrix multiplication step. <i>Sub-rhotrix location after first shift</i>	23
Figure 2.6c: Final block rhotrix multiplication step). <i>Sub-rhotrix location after second shift</i> ...	24
Figure 2.6d: Final block rhotrix	24
Figure 2.7a: Example of blocking process of fifteen dimensional rhotrix, R	25
Figure 2.7b: Example of blocking process of fifteen dimensional rhotrix Q	25
Figure 2.8a: Block of matrices of main and hearts entries of rhotrix R	26
Figure 2.8b: Block of matrices of main and hearts entries of rhotrix Q	26
Figure 2.9: Matrix multiplication using systolic array	29
Figure 3.4a: Movement of A and B elements	74
Figure 3.4b: Step 2-Alignment of elements of A and B	74
Figure 3.4c: Step 2-Alignment of elements of A and B	74
Figure 3.4.3a: Blocked rhotrix of dimensional fifteen, R_{15}	80
Figure 3.4.3b: Blocked rhotrix of dimensional fifteen, Q_{15}	80
Fig. 3.5a: Dependency graph for Winograd's variant of Strassen's algorithm	93
Fig. 3.5b: Task graph for Winograd's variant of Strassen's algorithm	96
Figure 3.8a: Architecture of a generic processor array	101
Figure 3.8b Two-dimensional mesh interconnection networks.....	102
Figure 3.8c Execution of an if-then-else statement	103
Figure. 3.8d Inactive processors states	103
Figure 3.8e This Figure Indicates the active and inactive processors switch roles	104

Figure 3.10a: Uniformly indexed variables of coupled matrices	112
Figure 3.10b: Flipped $R_5^{T/2}$	113
Figure 3.10c: Flipped $Q_5^{T/2}$	113
Figure 3.10d General array processor computation representation	113
Figure 3.10e: Staggered input data of the coupled matrices	114
Fig 3.10f: Flipped R_5	115
Fig 3.10g: Flipped Q_5	115
Figure 3.10h: Staggered input data of the two rhotrices	115
Figure 3.11a: Partition of the three rhotrices R, Q and C	119
Figure 3.11b: Example showing that min-min is not optimal	121
Figure 3.11c: Memory usage for the maximum reuse algorithm	121
Figure 3.11d: Four steps of the maximum re-use algorithm	122
Figure 3.12: Task/Channel graph of the algorithm.....	123
Figure 3.13: Task Graph for Heart-Oriented Rhotrix Algorithm.....	128
Fig 3.17: Using six MPI processes to implement the heart-oriented rhotrix multiplication algorithm	134
Figure 3.18: A model of MPI task processes graph to implement Strassen's algorithm	136
Figure 3.19a: Fortran mapping of double indices rhotrix to single array memory.....	137
Figure 3.19b: C's way of mapping the rhotrix to memory	137
Figure 3.19c: C's way of mapping the rhotrix to memory.....	138
Figure 3.19d: C's way of mapping the rhotrix to memory	139
Figure 4.2a: The real execution times for heart-oriented rhotrix algorithm	144
Figure 4.2b: The real execution times for row-column rhotrix algorithm	145
Figure 4.2c. The parallel execution times for heart-oriented rhotrix algorithm	146
Figure 4.2d. The real execution times for row-column rhotrix algorithm	147
Figure 4.3a: MPI timeline Screenshot Process view for P0 to P3	148
Figure 4.3b: MPI timeline screenshot process view for p0 to p13	148
Figure 4.3c: MPI timeline screenshot process view for P0 to P19.....	149
Figure 4.3d: timeline processes view legend	150
Figure 4.3e: Screenshot of TimeLine Communicator View for process P0 to P1.....	151

Figure 4.3f: Screenshot of TimeLine Communicator View for process P0 to P19	151
Figure 4.4a: Staggering of rhotrix vector into processing elements	153
Figure 4.4b: Screenshot process array simulator input interface.....	153
Figure 4.4c: Process array computation result	156
Figure 4.4d: Screenshot of process array simulator output result.....	156
Figure 4.5: Row-column process array structure	157

LIST OF TABLES

Table 2.1 Strassen’s Algorithm	32
Table 3.2a Setting conditions for λ over a three dimensional rhotrix entries	53
Table 3.2b Setting conditions for λ over a five dimensional rhotrix entries	53
Table 3.4a Representation of λ In blocked form with missing rows and column padded with zeros.....	72
Table 3.4b $A_{i,j}$ represents $2n-1 \times 2n-1$ matrix of entries	73
Table 3. 4.1 The communication steps in Cannon’s algorithm on 16 processes	77
Table 3.5 Strassen’s algorithm for rhotrix multiplication.....	91
Table 3.17a Heart-oriented data distribution among processes	134
Table 3.17b Heart-oriented computation among processes	135
Table 4.2a Sequential execution time for seven sets of heart-oriented rhotrices computation	144
Table 4.2b Sequential execution time for seven sets of row-column rhotrices computation	144
Table 4.2c Parallel execution time of four sets of heart-oriented rhotrices computation	145
Table 4.2d Parallel speedup of four sets of heart-oriented rhotrices computation	146

CHAPTER ONE: INTRODUCTION

1.1. Background to the study

Linear algebra is an area which involves heavy computations and therefore is in need of high-performance routines to decrease the total computation time. In effort to make this possible, parallel computers have evolved with different architectures. Parallel computer systems consisting of multiple processors provide much more raw computing power (by orders of magnitude) than traditional uni-processor computer systems. Powerful parallel computer systems open up new frontiers in parallel applications that strive to utilize such systems effectively. The challenges for parallel applications, particularly in the field of scientific computing, lie in what the new parallel programming model should be and what kinds of parallelism an algorithm should possess to utilize the parallel computer system effectively.

One of the models of parallel computers is the multicomputer model. A multicomputer is a concurrent distributed-memory system with message-passing among processors (Foster, 1995). A multicomputer consists of a number of von Neumann processors that have their own memory and an interconnection network that links processors through message interchange. Hence, the MIMD architectural model, which stands for multiple instruction streams and multiple data streams (Kumar *et al.*, 1994). Together with message-passing, forms an ideal parallel programming model suitable for multicomputer. This combined hardware, software architecture is capable of executing different programs concurrently on processors and exchanging information among processors via message passing.

Among the key requirements for effective parallel computation, the performance requirement is always the most critical, because the primary motivation of parallel applications is to achieve higher performance that cannot be accomplished by using traditional sequential computer systems. However, it is difficult for a single algorithm to achieve high performance for a wider range of problem sizes on computer systems with different characteristics. A non-trivial example is the well-known sorting problem. The “quick-sort” algorithm works well for sorting lists while the “bubble-sort” algorithm works well for short lists. In order to accomplish the sorting problem optimally, application developers must take the responsibility for formulating the problem and

choosing an appropriate algorithm according to the length of the list. However, this approach puts a heavy burden on application developers for complex problems, as well as imposing a need for predictive performance models. Furthermore, in order to utilize different parallel computer systems effectively, application developers must have deep knowledge of underlying systems. The better approach to overcome this dilemma is to use the “poly-algorithmic approach.” The poly-algorithmic approach, a concept introduced by Rice and Rosen, (1966), refers to the use of two or more algorithms to solve the same problems with a high level decision-making process determining which of a set of algorithm performs best in a given situation. The poly-algorithm approach helps the parallel libraries to achieve performance scalability and portability while hiding the complexity of performance-oriented software and the multiplicity of algorithms in the interface.

Consequently, the research hypothesis is that no single parallel rhotrix multiplication algorithm can always achieve the best performance for multiplying rhotrices with different sizes on process grids with different shapes; the poly-algorithmic approach is able to achieve the requirement of high performance over a range of system and problems parameters. In the present study, we will specifically target the problem of rhotrix multiplication on systolic array parallel architecture. We will build a collection of general purpose parallel rhotrix multiplication algorithms. We will measure performance of each algorithm for rhotrices with different sizes on simulated process grids (systolic array). By using the poly-algorithmic approach, we will show that it is possible to achieve high and uniform performance by choosing an appropriate algorithm from a collection of parallel rhotrix multiplication algorithms suitable for a particular situation. This will require multiple algorithms and associated performance models and measurements for representative case.

The concept of rhotrices was introduced by Ajibade (2003) as an extension of ideas on matrix-tertions and matrix-noitrets discussed in Attanasov and Shannon (1998). Ajibade presented the initial algebra and analysis on rhotrices and established some interesting relationships between rhotrices and their hearts. He defined the multiplication of two rhotrices

$$R = \left\langle \begin{array}{ccc} & a & \\ b & h(R) & d \\ & e & \end{array} \right\rangle \text{ and } Q = \left\langle \begin{array}{ccc} & f & \\ g & h(Q) & j \\ & k & \end{array} \right\rangle$$

as follows:

$$\left\langle \begin{array}{ccc} & ah(Q) + fh(R) & \\ bh(Q) + gh(R) & h(R)h(Q) & dh(Q) + jh(R) \\ & eh(Q) + kh(R) & \end{array} \right\rangle$$

This multiplication procedure gives initial interesting results but may not allow further development regarding the question posed in (Ajibade, 2003). Some of those questions were answered in (Attanasov and Shannon, 1998), in relation to matrix-tertions and matrix-noitrits but to the best of our knowledge there has not been any extension regarding generalization of this concept of rhotrix multiplication with regards to n-dimensional rhotrices. This is why we propose an extension by means of generalization and derivation of mathematical models and algorithms for the heart-oriented (an acronym for the multiplication method given by Ajibade) rhotrix multiplication.

An alternative multiplication method was proposed by Sani (2004). According to this method, since rhotrices lie somewhere between 2×2 and 3×3 matrices, it is reasonable to define their multiplication similarly to matrix multiplication. To do that he defined the rows and columns of a rhotrix

$$\left\langle \begin{array}{ccc} & a & \\ b & h(R) & d \\ & e & \end{array} \right\rangle \text{ as } \begin{array}{ccc} b & a & \\ & e & d \end{array} \text{ and } \begin{array}{ccc} & a & d \\ b & & e \end{array}$$

respectively. He then defined multiplication of any two rhotrices using the row-column multiplication method of matrices, except for the hearts which is multiplied directly. The multiplication is defined as follows:

$$R \circ Q = \left\langle \begin{array}{ccc} & a & \\ b & h(R) & d \\ & e & \end{array} \right\rangle \circ \left\langle \begin{array}{ccc} & f & \\ g & h(Q) & j \\ & k & \end{array} \right\rangle = \left\langle \begin{array}{ccc} & af + dg & \\ bf + eg & h(R)h(Q) & aj + dk \\ & bj + ek & \end{array} \right\rangle$$

This multiplication, like that of matrices, is non commutative, but it is associative. Again as part of this work, we also propose a new approach to this multiplication method by a way of designing new generalization case, mathematical model and algorithms for the row-column rhotrices multiplication different from what was initially given by the author.

Rhotrix multiplication can be seen as a fundamental operation in most arbitrary numerical linear algebra applications. Its efficient implementation on parallel high performance computers, together with the implementation of other basic linear algebra operations, could therefore be considered an issue of prime importance when providing these systems with scientific software libraries. Consequently, considerable effort has been devoted in this research work towards developing efficient parallel rhotrix multiplication algorithms, and this will remain a task in the future as well.

This research work presents some sets of new poly algorithms approach for multiplying higher dimensional rhotrices in both sequential and parallel environment using conventional programming languages like C, Message Passing Interface (MPI) and the process array architecture for the parallelization execution.

1.2. Research Motivation

The Motivation of the research work was based on two fundamental multiplicative concepts on rhotrices introduced in (Ajibade, 2003), (Sani, 2004), and (Sani, 2008). In each of the aforementioned journal publications, multiplication of rhotrices was introduced and addressed by these researchers in a unique but separate ways. The initial driving force behind the conception of the research work was to extend the work done by these scholars and to equally obtain high performance by taking advantage of grid computer architectures and exploiting inherent parallel techniques such as those found in high-performance computation.

1.3. Research Objectives

The primary objective of this thesis is to devise a distributed computational framework that would allow the parallelization and execution of our proposed sequential and parallel rhotrices multiplication algorithms, based on the rhotrices multiplication

methods developed in (Ajibade, 2003) and (Sani, 2004). Particular effort will be placed on achieving this in an affordable way, which is by avoiding the unnecessary use of costly hybrid machines and taking advantage of existing computational infrastructure available in the university computer laboratory. More specifically, several mathematical models will be developed and new sequential rhotrix multiplication algorithms designed over the parallelization framework of a functional grid. This study will produce a set of the most effective strategies of parallelization of these and similar algorithms.

In order to validate the usefulness and effectiveness of the implemented framework, two main problems will be discussed:

- Heart-oriented multiplication: i.e direct multiplication of rhotrix elements with corresponding array indices.
- Row-column multiplication: i.e multiplication of rhotrix elements row by column wise.

1.4. Research Methodology

The method used in this study is as follows:

A review of different rhotrix multiplication methods from related literatures was carried out. This includes row-column multiplication akin to the traditional row-column matrix multiplication, similar to this also is the rhotrix conversion to coupled matrix by padding, referred to as rhotrix transformation. A collection of algorithms were designed to implement each of the aforementioned methods. The implementation was done using C and Java programming Languages for the sequential algorithm test bed while Delphi API and MPI message passing were used to simulate the parallel algorithms.

1.5. Organization of the Thesis

The rest of the thesis is organized as follows: Chapter II reviews the relevant literature in the research areas that covers different ideology defining rhotrix multiplication that pertains to this study and discusses the methodology of scalable parallel platform models adopted for the study, Chapter III discusses rhotrix data representation format, algorithm design for rhotrices row-column and heart-oriented rhotrices application, data mapping problem, rhotrix data mapping design and parallel platform performance model. Chapter IV describes the research implementation strategy of our parallel rhotrix

multiplication algorithms and provides the experimental results of each algorithm on the adopted platform. Chapter V summarises the present study and presents the conclusion and future work.

1.6. Contribution to Knowledge

The main contributions of this work are in twofold:

On the theoretical side, we have proposed and designed four sequential and three parallel algorithms implementation of row-column and heart-oriented computation of rhotrices on computer systems. Furthermore, these algorithms have been checked against experimental results generated from sequential C programs and MPI message passing used as test bed to further buttress the validity of the algorithms presented in our study.

On the practical side, we have implemented and evaluated two parallel implementations of rhotrix row-column and dot product computation on process array and heterogeneous master - worker platforms using MPI message passing library. These implementations are based on the data allocation and dynamic load balancing strategies.

1.7. Benefits of this Research Work to Society

Considering the sparseness and arbitrary representation of some scientific and engineering problems, rhotrix multiplication can be regarded as a better candidate for the computational chemists for representing problems in terms of states of a chemical system. Each rhotrix index corresponds to a different basis state, and the rhotrix approximates the Hamiltonian of the system. A change of basis is accomplished through rhotrix multiplication. Rhotrix could be termed as a super matrix (for every set of rhotrix elements, there exist two other matrices embedded within these sets). Rhotrix-Vector multiplication could be applied in some transforms used in image processing, signal processing for the electronics and telecommunication industries (convolution). Some other possible application domain areas of large rhotrix multiplication are in molecular biology, genetic engineering and cellular computing. In the aforementioned areas, changes in states and trends monitoring pattern can be accomplished through derived sets of system of linear equations from rhotrix multiplication.

CHAPTER TWO: LITERATURE REVIEW

2.1. Introduction

The purpose of this chapter is to review the relevant literature in the research of parallel computing, parallel programming models, message passing, and some related parallel matrix algorithms that pertain to the present study. The methodology of scalable process array and Message Passing Implementation design issues were also reviewed which guides the design of our parallel matrix multiplication on similar topology.

2.2. Parallel Computing Overview

In the simplest sense, parallel computing is the simultaneous use of multiple compute resources to solve a computational problem which are to be run using multiple central processing units (CPUs). In essence, a problem is broken into discrete parts that can be solved concurrently, each part is further broken down to a series of instructions and finally, instructions from each part execute simultaneously on different CPUs

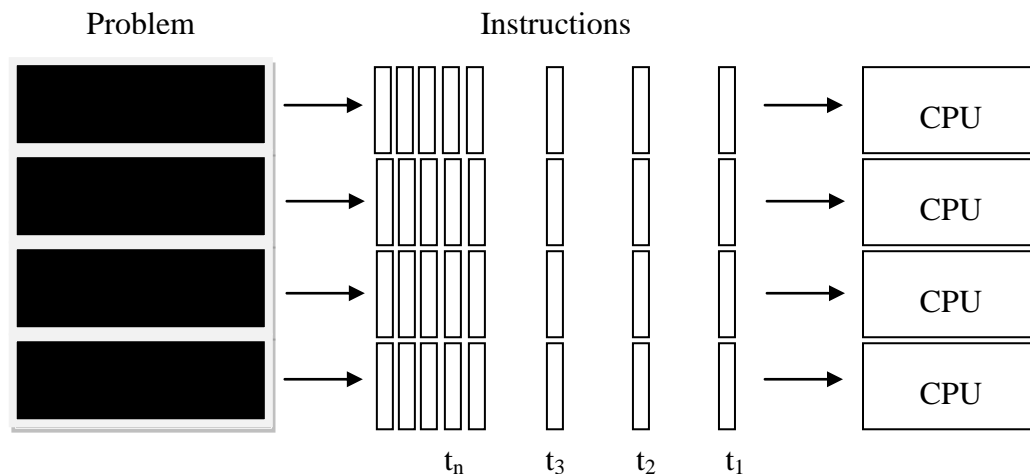


Figure 2.1: Parallel computing process structure

The computer resources can include a single computer with multiple processors, an arbitrary number of computers connected by a network and / or a combination of both. The computational problem usually demonstrates characteristics such as the ability to be:

- Broken apart into discrete pieces of work that can be solved simultaneously;
- Execute multiple program instructions at any moment in time;

- Solved in less time with multiple compute resources than with a single computer resource.

Uses for parallel computing:

Historically, parallel computing has been considered to be "the high end of computing", and has been used to model difficult scientific and engineering problems found in the real world. Some examples:

- Atmosphere - Earth, Environment
- Physics - applied, nuclear, particle, condensed matter, high pressure, fusion, photonics
- Bioscience - Biotechnology, Genetics
- Chemistry - Molecular Sciences
- Geology - Seismology
- Mechanical Engineering - from prosthetics to spacecraft
- Electrical Engineering, Circuit Design, Microelectronics
- Computer Science and Mathematics

Why use parallel computing?

Main Reasons:

- a. Save time and/or money: In theory, throwing more resources at a task will shorten its time to completion, with potential cost savings. Parallel clusters can be built from cheap, commodity components.
- b. Solve larger problems: Many problems are so large and/or complex that it is impractical or impossible to solve them on a single computer, especially given limited computer memory. For example:

"Grand Challenge" (en.wikipedia.org/wiki/Grand_Challenge) problems requiring PetaFLOPS and PetaBytes of computing resources.

- c. Web search engines/databases processing millions of transactions per second
- d. Provide concurrency: A single compute resource can only do one thing at a time. Multiple computing resources can be doing many things simultaneously. For example, the Access Grid (<http://www.accessgrid.org/>) provides a global

collaboration network where people from around the world can meet and conduct work "virtually".

e. Use of non-local resources: Using computer resources on a wide area network, or even the Internet when local computer resources are scarce. For example:

SETI@home (setiathome.berkeley.edu) uses over 330,000 computers for a computer power over 528 TeraFLOPS (as of August 04, 2008) and Folding@home (folding.stanford.edu) uses over 340,000 computers for a compute power of 4.2 PetaFLOPS (as of November 4, 2008)

f. Limits to serial computing: Both physical and practical reasons pose significant constraints to simply building ever faster serial computers:

- Transmission speeds - the speed of a serial computer is directly dependent upon how fast data can move through hardware. Absolute limits are the speed of light (30 cm/nanosecond) and the transmission limit of copper wire (9 cm/nanosecond). Increasing speeds necessitate increasing proximity of processing elements.
- Limits to miniaturization - processor technology is allowing an increasing number of transistors to be placed on a chip. However, even with molecular or atomic-level components, a limit will be reached on how small components can be.
- Economic limitations - it is increasingly expensive to make a single processor faster. Using a larger number of moderately fast commodity processors to achieve the same (or better) performance is less expensive.

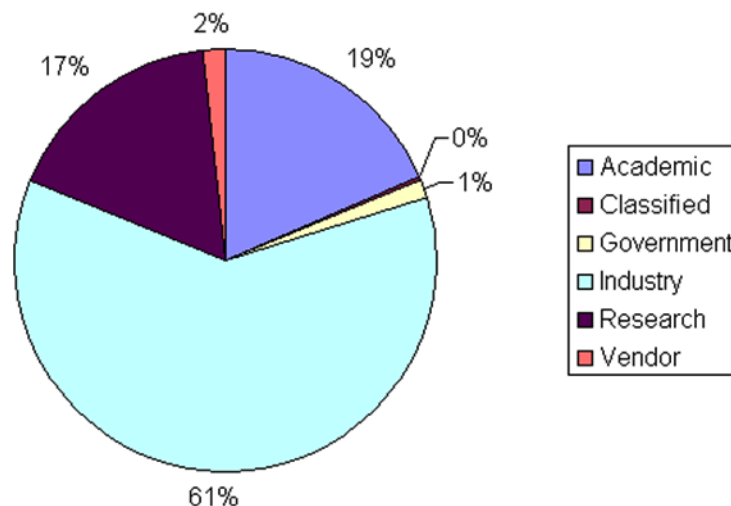


Figure. 2.2a: Parallel computing case studies

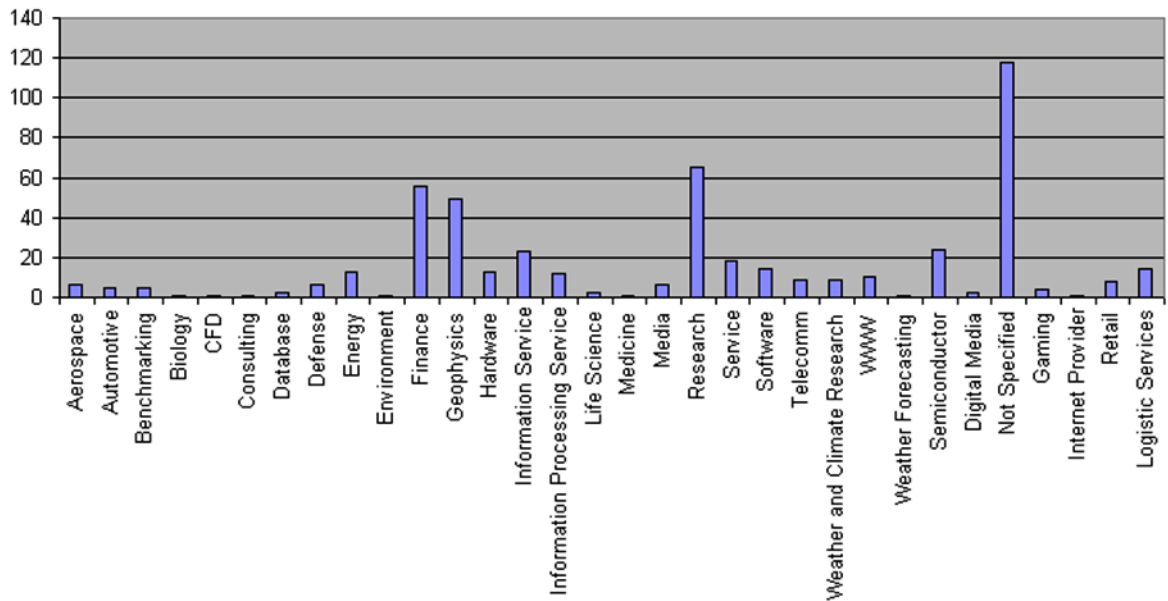


Figure. 2.2b: Areas of application of parallel computing

2.2.1. Some Concepts and Terminologies

Like everything else, parallel computing has its own "jargon". Some of the more commonly used terms associated with parallel computing and used in this thesis are listed below.

Task

A logically discrete section of computational work, a task is typically a program or program-like set of instructions that is executed by a processor.

Parallel task

A task that can be executed by multiple processors safely (yields correct results)

Serial execution

Execution of a program sequentially, one statement at a time: In the simplest sense, this is what happens on a one processor machine. However, virtually all parallel tasks will have sections of a parallel program that must be executed serially.

Parallel execution

Execution of a program by more than one task, with each task being able to execute the same or different statement at the same moment in time.

Pipelining

Breaking a task into steps performed by different processor units, with inputs streaming through, much like an assembly line; a type of parallel computing.

Shared memory

From a strictly hardware point of view, describes a computer architecture where all processors have direct (usually bus based) access to common physical memory. In a programming sense, it describes a model where parallel tasks all have the same "picture" of memory and can directly address and access the same logical memory locations regardless of where the physical memory actually exists.

Symmetric multi-processor (SMP)

Hardware architecture where multiple processors share a single address space and access to all resources; shared memory computing.

Distributed memory

In hardware, refers to network based memory access for physical memory that is not common. As a programming model, tasks can only logically "see" local machine memory and must use communications to access memory on other machines where other tasks are executing.

Communications

Parallel tasks typically need to exchange data. There are several ways this can be accomplished, such as through a shared memory bus or over a network, however the actual event of data exchange is commonly referred to as communications regardless of the method employed.

Synchronization

The coordination of parallel tasks in real time, very often associated with communications. Often implemented by establishing a synchronization point within an application where a task may not proceed further until another task(s) reaches the same or logically equivalent point. Synchronization usually involves waiting by at least one task, and can therefore cause a parallel application's wall clock execution time to increase.

Granularity

In parallel computing, granularity is a qualitative measure of the ratio of computation to communication. Coarse: relatively large amounts of computational work are done between communication events. Fine: relatively small amounts of computational work are done between communication events

Observed speedup

Observed speedup of a code which has been parallelized, defined as: Wall-clock time of serial execution and wall-clock time of parallel execution. This is one of the simplest and most widely used indicators for a parallel program's performance.

Parallel overhead

The amount of time required to coordinate parallel tasks, as opposed to doing useful work. Parallel overhead can include factors such as: task start-up time, synchronizations, data communications, software overhead imposed by parallel compilers, libraries, tools, operating system, and task termination time

Massively parallel

This refers to the hardware that comprises a given parallel system having many processors. The meaning of "many" keeps increasing, but currently, the largest parallel computers can be comprised of processors numbering in the hundreds of thousands.

Scalability

Refers to a parallel system's (hardware and/or software) ability to demonstrate a proportionate increase in parallel speedup with the addition of more processors. Factors that contribute to scalability include: hardware particularly memory-cpu bandwidths and network communications, application algorithm, parallel overhead related, characteristics of your specific application and coding

Multi-core processors

Multiple processors (cores) on a single chip.

Cluster computing

Use of a combination of commodity units (processors, networks or SMPs) to build a parallel system.

2.2.2. Message Passing Model

The message passing model demonstrates the following characteristics:

- a. A set of tasks that use their own local memory during computation. Multiple tasks can reside on the same physical machine and/or across an arbitrary number of machines.
- b. Tasks exchange data through communications by sending and receiving messages.
- c. Data transfer usually requires cooperative operations to be performed by each process. For example, a send operation must have a matching receive operation.

Implementations:

From a programming perspective, message passing implementations commonly comprise a library of subroutines that are imbedded in source code. The programmer is responsible for determining all parallelism. Historically, a variety of message passing libraries have been available since the 1980s. These implementations differed substantially from each other making it difficult for programmers to develop portable applications. In 1992, the MPI Forum was formed with the primary goal of establishing a standard interface for message passing implementations. MPI is now the "de facto" industry standard for message passing, replacing virtually all other message passing implementations used for production work. Most, if not all of the popular parallel computing platforms offer at least one implementation of MPI. For shared memory architectures, MPI implementations usually don't use a network for task communications. Instead, they use shared memory (memory copies) for performance reasons.

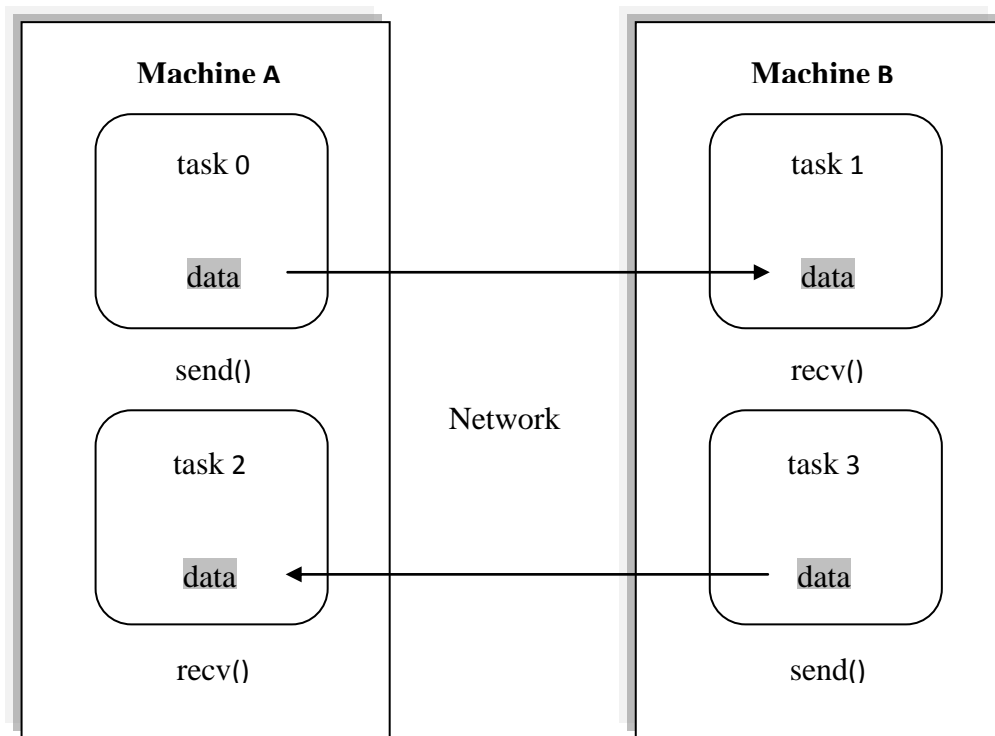


Figure 2.3: Parallel programming model (message passing model)

2.2.3. Data Parallel Model

The data parallel model demonstrates the following characteristics:

- a. Most of the parallel work focuses on performing operations on a data set. The data set is typically organized into a common structure, such as an array or cube.
- b. A set of tasks work collectively on the same data structure, however, each task works on a different partition of the same data structure.
- c. Tasks perform the same operation on their partition of work, for example, "add 4 to every array element".
- d. On shared memory architectures, all tasks may have access to the data structure through global memory. On distributed memory architectures the data structure is split up and resides as "chunks" in the local memory of each task.

Implementations:

Programming with the data parallel model is usually accomplished by writing a program with data parallel constructs. The constructs can be calls to a data parallel subroutine library or, compiler directives recognized by a data parallel compiler. The compiler directives, allows the programmer to specify the distribution and alignment of

data. Fortran and C implementations are available for most common parallel platforms. Distributed memory implementations of this model usually have the compiler convert the program into standard code with calls to a message passing library (MPI usually) to distribute the data to all the processes. All message passing is done invisibly to the programmer.

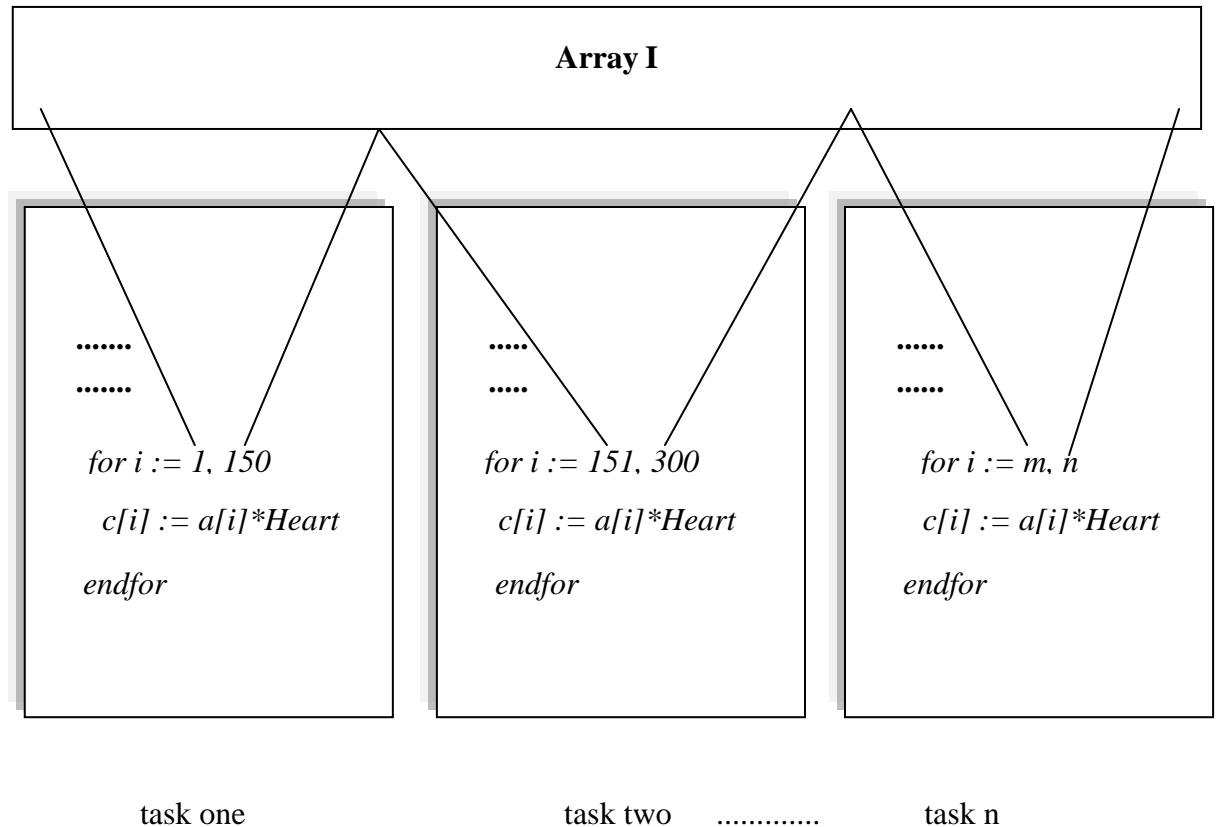


Figure 2.4: Parallel programming model (data parallel model)

2.3. Analysing Sequential Algorithm

The design and analysis of sequential algorithms is a well developed field, with a large body of commonly accepted results and techniques. This consensus is built upon the fact that the methodology and notation of asymptotic analysis (the so-called “big-O” notation) deliver results which are applicable across all sequential computers, programming languages, compilers and so on. This generality is achieved at the expense of a certain degree of blurring, in which constant factors and non-dominating terms in the analysis are simply ignored. In spite of this, the approach produces results which allow useful comparisons of the essential performance characteristics of different algorithms which are reflected in practice when implemented on real machines, in real

languages through real compilers. For example, merge-sort with its $\Theta(n \log n)$ run time is (in the worst case) an asymptotically better sorting algorithm than insertion sort ($\Theta(n^2)$) on any normal sequential machine (although the actual problem size at which the dominance becomes apparent will vary from implementation to implementation). Underpinning this work is the “Random Access Machine” (RAM) model which is an abstraction of the essential capabilities and cost characteristics which unite all sequential machines. The notation allows the description of “upper bounds” (with $O()$), “lower bounds” (with $\Omega()$) and “tight bounds” (with $\Theta()$) on the behaviour of functions representing the time or space requirements of an algorithm as its input problem size grows.

2.4. Analysing Parallel Algorithms

The sequential world benefits from a single universal abstract machine model (the RAM) which accurately (enough) characterizes all sequential computers and from a simple criterion of “better” for algorithm comparison (“less is better”, usually of run time, and occasionally of memory space). Thinking parallel, we immediately encounter two complications. Firstly, and fundamentally, there is no commonly agreed model of parallel computation. The diversity of proposed and implemented parallel architectures is such that it is not clear that such a model will ever emerge. Worse than this, the variations in architecture capabilities and associated costs mean that no such model can emerge, unless we are prepared to forgo certain tricks or shortcuts exploitable on one machine but not another. An algorithm designed in some abstract model of parallelism may have asymptotically different performance on two different architectures (rather than just the varying constant factors of different sequential machines). Secondly, our notion of “better” even in the context of a single architecture must surely take into account the number of processors involved, as well as the run time. The trade-offs here will need careful consideration. In this research work we will not attempt to unify the irretrievably diverse. Thus we will have a small number of machine models and will design algorithms for our chosen problems for some or all of these, however, in doing so we still hope to emphasize common principles of design which transcend the differences in architecture. Equally, in some instances, we will exploit particular features of one model where that leads to a novel or particularly effective algorithm.

Parallel algorithms are developed using independences among data to be processed. For example, we can easily develop a parallel algorithm to compute a rhotrix-vector multiplication. The reader should note that the linear system of equation of the form $R_n x = b$ has been studied in Aminu (2010), where R_n is an n-dimensional rhotrix and b is the right hand side rhotrix vector:

For instance, given a rhotrix R_n and a vector x , find b such that $R_n x = b$.

Obviously, each entry of x is independent entries of y , so we can compute all entries in parallel. On the other hand, it is difficult to parallelize algorithms which process data with dependences. For example, we consider algorithm which computes x_{10000} , defined by recurring formula:

$$x_{n+1} = \begin{cases} \alpha x_n & x_n < \frac{1}{2} \\ \alpha(1-x_n) & \frac{1}{2} \leq x_n \end{cases} \quad \text{for } 0 \leq x_0 \leq 1 \text{ and } 0 < \alpha$$

To compute x_{n+1} , it is necessary to know x_n . So, this algorithm is sequential, or we have to find other property of x_n to parallelize.

2.5. Review of Cannon's Algorithms and Block Rhotrix Operations

Cannon (1969) presented a systolic approach for parallel multiplication of two square dense matrices distributed on square processor grids. The principle of Cannon's algorithm is that the computing of the sub-matrices C_{pq} on processor-(p,q) needs all the sub-matrices of $A_{p,k}$ in the pth processor row and all the sub-matrices of $B_{k,p}$ in the qth process column, where $0 \leq k < P$. Cannon's algorithm works circularly to shift matrix A in the row dimension and matrix B in the column dimension so that each process gets fresh sub-matrices $A_{p,k}$ and $B_{k,q}$ after each shift step and then updates the partial results of sub-matrices C_{pq} in each process. The repeating update and shift process is systolic. However, in order to multiply sub-matrices of A and B correctly in each process, Cannon's algorithm need to align matrices A and B initially so that the indices of sub-matrices A and B match canonically. Cannon only considered square matrix multiplication on square process grids. The Cannon algorithm is sketched as follows:

Step 1: Initially align matrix A.

Step 2: Initially align matrix B.

Step 3: Update partial C by multiplying sub-matrices A and B in each process.

Step 4: Circularly shift matrix A along the row dimension.

Step 5: Circularly shift matrix B along the column dimension.

Step 6: Repeat Step 3 through Step 5 P times (where $P = Q$).

Step 7: Align matrix A to restore the initial distribution.

Step 7: Align matrix B to restore the initial distribution.

Cannon algorithm, on the other hand is a memory-efficient version of a simple algorithm. We can equally extend the application of this algorithm on rhotrix multiplication by simply partitioning rhotrices R and Q into p square blocks of submatrices as shown in Figure 2.5a and 2.5b respectively, Abdullahi *et al* (2010).

The processes are labelled from $P_{0,0}$ to $P_{\sqrt{p}-1, \sqrt{p}-1}$ and initially assign partitioned submatrices A_{ij} and B_{ij} to process P_{ij} .

Although every process in the i-th row requires \sqrt{p} submatrices A_{ik} ($0 \leq k < \sqrt{p}$) it is possible to schedule the computation of the \sqrt{p} processes in the i-th row such that, at any given time, each process is using a different A_{ik} .

These blocks can be systematically rotated among the processes after every submatrix multiplication so that every process gets a fresh A_{ik} after each rotation.

If an identical schedule is applied to the columns, then no process holds more than one block of each matrix at any time, and the total memory requirement of the algorithm over all the processes is $\Theta(n^2)$. Cannon algorithm is based on this idea. The first communication step of the algorithm aligns the blocks of A and B in such a way that each process multiplies its local submatrices. This alignment is achieved for matrix A by shifting all submatrices A_{ij} to the left (with wraparound) by i steps. All submatrices B_{ij} are shifted up by j steps.

These are circular shift operation:

After a submatrix multiplication step, each block of A moves one step left and each

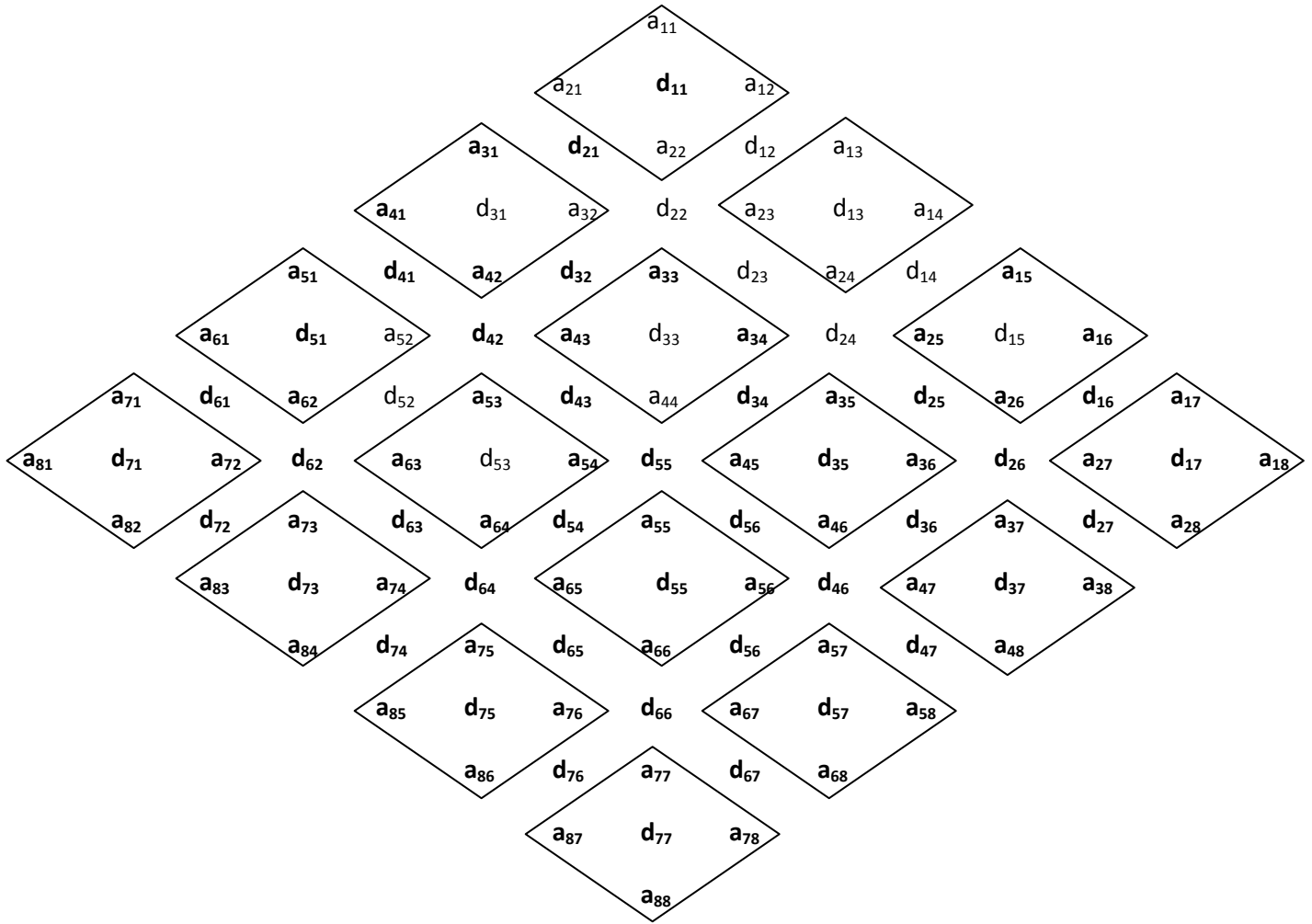


Figure 2.5a: Blocking process for fifteen dimensional rotatrix R_{15} with main entries a and hearts d

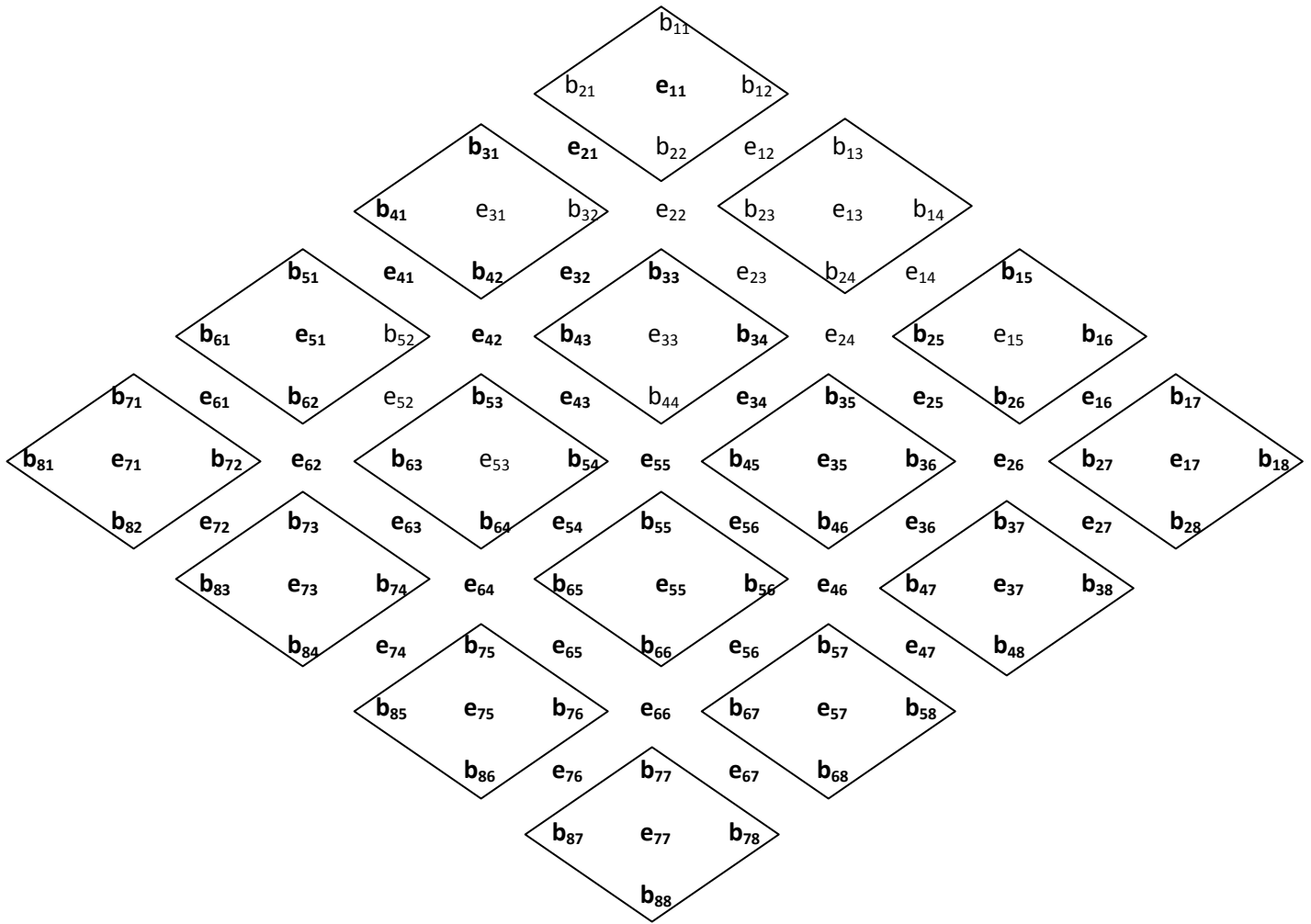


Figure 2.5b: Blocking process for fifteen dimensional rhotrix Q_{15} with main entries b and hearts e

The blocking process in Figure 2.5a and 2.5b is similar to those found in matrix blocking technique obtained using cannon algorithm. The fifteen dimensional rhotrix above was blocked into sixteen partitions comprising of rhotrix main entries and the heart elements denoted by a , b and d , e . Those elements outside the blocks are mapped to their corresponding elements having the same index numbering, thereby allowing them to belong to the same partitioned blocks.

2.5.1. Rhotrix-Rhotrix Multiplication: Cannon's Algorithm Method

- Consider n dimensional rhotrices R_n and Q_n partitioned into p blocks $a_{i,j}$ and $b_{i,j}$ ($0 \leq i, j < \sqrt{p}$) each.
- Process $P_{i,j}$ initially stores $a_{i,j}$, $b_{i,j}$ and $d_{i,j}$, $e_{i,j}$ and computes block $c_{i,j}$ and $h_{i,j}$, of the resulting rhotrix.

- Computing subrhatrix $c_{i,j}$ requires all subrhatrices $a_{i,k}$, $b_{k,j}$ and $d_{i,k}$, $e_{k,j}$ for $0 \leq k < \sqrt{p}$.
- Every process in the i^{th} row requires all subrhatrices, the all-to-all broadcast can be avoided by scheduling the computations of the processes of the i^{th} row such that, at any given time, each process is using a different block $a_{i,k}$ and $d_{i,k}$ by systematically rotating these blocks among the processes after every subrhatrix multiplication so that every process gets a fresh $a_{i,k}$, and $d_{i,k}$ after each rotation.
- If an identical schedule is applied to the columns of B and E, then no process holds more than one block of each rhatrix at any time.

2.5.2. Communication Steps in Cannon's Algorithm

Applying the block row-column operation using the cannon's algorithm method for rhatrices with main entries $A_{i,j}$, $B_{i,j}$ and with corresponding hearts $D_{i,j}$, $E_{i,j}$, we consider cannon's matrix multiplication algorithm from the point of view of process $P_{i,j}$. The processes are organized into a $2-D$ mesh, and each process has a block of A , B , D and E needed for computing C and H respectively. Figure 2.6a depicts the first block multiplication step. After each block multiplication process $P_{i,j}$ sends its block of A to the process on its left and receives a new block of A from the process on its right. Similarly, it sends its block of B to the process above it and receives a new block of B from the process below it. Figure 2.6b, 2.6c and 2.6d shows the second, third and final block rhatrix multiplication respectively. Summing the results of all block rhatrix multiplications yields $C_{i,j}$ and $H_{i,j}$.

The block rhatrix multiplication algorithm for $n \times n$ matrices with a block size of $(n/q) \times (n/q)$

```

procedure RHOTRIX_MULT (A, B, C, D, E, H)
begin
  for  $i := 0$  to  $t - 1$  do
    for  $j := 0$  to  $t - 1$  do
      begin
        {Initialize all elements of  $C_{i,j}$  and  $H_{k,j}$  to zero}
         $C[i, j] := 0$ ;

```

```

         $H[i, j] := 0;$ 
    for  $k := 0$  to  $t - 1$  do
        {Receive message from slave process (worker)  $i, j$ }

         $C_{i,j} := C_{i,j} + A_{i,k} \times B_{k,j};$ 
         $H_{i,j} := H_{i,j} + D_{i,k} \times E_{k,j};$ 

    endfor;

endfor;

{Write  $C$  and  $H$  to output file}
end RHOTRIX_MULT

```

The following figures illustrates the algorithm for 16 processors:

A. First Alignment

$A_{0,0} D_{0,0}$ $B_{0,0} E_{0,0}$	$A_{0,1} D_{0,1}$ $B_{1,1} E_{1,1}$	$A_{0,2} D_{0,2}$ $B_{2,2} E_{2,2}$	$A_{0,3} D_{0,3}$ $B_{3,3} E_{3,3}$
$A_{1,1} D_{1,1}$ $B_{1,0} E_{1,0}$	$A_{1,2} D_{1,2}$ $B_{2,1} E_{2,1}$	$A_{1,3} D_{1,3}$ $B_{3,2} E_{3,2}$	$A_{1,0} D_{1,0}$ $B_{0,3} E_{0,3}$
$A_{2,2} D_{2,2}$ $B_{2,0} E_{2,0}$	$A_{2,3} D_{2,3}$ $B_{3,1} E_{3,1}$	$A_{2,0} D_{2,0}$ $B_{0,2} E_{0,2}$	$A_{2,1} D_{2,1}$ $B_{1,3} E_{1,3}$
$A_{3,3} D_{3,3}$ $B_{3,0} E_{3,0}$	$A_{3,0} D_{3,0}$ $B_{0,1} E_{0,1}$	$A_{3,1} D_{3,1}$ $B_{1,2} E_{1,2}$	$A_{3,2} D_{3,2}$ $B_{2,3} E_{2,3}$

Figure 2.6a: First block rhotrix multiplication step. A , B and D , E after initial alignment

B. Second Alignment

$A_{0,0} D_{0,0}$ $B_{0,0} E_{0,0}$	$A_{0,1} D_{0,1}$ $B_{0,1} E_{0,1}$	$A_{0,2} D_{0,2}$ $B_{0,2} E_{0,2}$	$A_{0,3} D_{0,3}$ $B_{0,3} E_{0,3}$
$A_{1,0} D_{1,0}$ $B_{1,0} E_{1,0}$	$A_{1,1} D_{1,1}$ $B_{1,1} E_{1,1}$	$A_{1,2} D_{1,2}$ $B_{1,2} E_{1,2}$	$A_{1,3} D_{1,3}$ $B_{1,3} E_{1,3}$
$A_{2,0} D_{2,0}$ $B_{2,0} E_{2,0}$	$A_{2,1} D_{2,1}$ $B_{2,1} E_{2,1}$	$A_{2,2} D_{2,2}$ $B_{2,2} E_{2,2}$	$A_{2,3} D_{2,3}$ $B_{2,3} E_{2,3}$
$A_{3,0} D_{3,0}$ $B_{3,0} E_{3,0}$	$A_{3,1} D_{3,1}$ $B_{3,1} E_{3,1}$	$A_{0,2} D_{0,2}$ $B_{0,2} E_{0,2}$	$A_{0,3} D_{0,3}$ $B_{0,3} E_{0,3}$

Figure 2.6b: Final block rhotrix multiplication step. *Sub-rhotrix location after first shift*

C. Third Alignment

$A_{0,0}D_{0,0}$ $B_{0,0}E_{0,0}$	$A_{0,1}D_{0,1}$ $B_{1,1}E_{1,1}$	$A_{0,2}D_{0,2}$ $B_{2,2}E_{2,2}$	$A_{0,3}D_{0,3}$ $B_{3,3}E_{3,3}$
$A_{1,1}D_{1,1}$ $B_{1,0}E_{1,0}$	$A_{1,2}D_{1,2}$ $B_{2,1}E_{2,1}$	$A_{1,3}D_{1,3}$ $B_{3,2}E_{3,2}$	$A_{1,0}D_{1,0}$ $B_{0,3}E_{0,3}$
$A_{2,2}D_{2,2}$ $B_{2,0}E_{2,0}$	$A_{2,3}D_{2,3}$ $B_{3,1}E_{3,1}$	$A_{2,0}D_{2,0}$ $B_{0,2}E_{0,2}$	$A_{2,1}D_{2,1}$ $B_{1,3}E_{1,3}$
$A_{3,3}D_{3,3}$ $B_{3,0}E_{3,0}$	$A_{3,0}D_{3,0}$ $B_{0,1}E_{0,1}$	$A_{3,1}D_{3,1}$ $B_{1,2}E_{1,2}$	$A_{3,2}D_{3,2}$ $B_{2,3}E_{2,3}$

Figure 2.6c: Final block rhotrix multiplication step. *Sub-rhotrix location after second shift*

$A_{0,0}D_{0,0}$ $B_{0,0}E_{0,0}$	$A_{0,1}D_{0,1}$ $B_{1,1}E_{1,1}$	$A_{0,2}D_{0,2}$ $B_{2,2}E_{2,2}$	$A_{0,3}D_{0,3}$ $B_{3,3}E_{3,3}$
$A_{1,1}D_{1,1}$ $B_{1,0}E_{1,0}$	$A_{1,2}D_{1,2}$ $B_{2,1}E_{2,1}$	$A_{1,3}D_{1,3}$ $B_{3,2}E_{3,2}$	$A_{1,0}D_{1,0}$ $B_{0,3}E_{0,3}$
$A_{2,2}D_{2,2}$ $B_{2,0}E_{2,0}$	$A_{2,3}D_{2,3}$ $B_{3,1}E_{3,1}$	$A_{2,0}D_{2,0}$ $B_{0,2}E_{0,2}$	$A_{2,1}D_{2,1}$ $B_{1,3}E_{1,3}$
$A_{3,3}D_{3,3}$ $B_{3,0}E_{3,0}$	$A_{3,0}D_{3,0}$ $B_{0,1}E_{0,1}$	$A_{3,1}D_{3,1}$ $B_{1,2}E_{1,2}$	$A_{3,2}D_{3,2}$ $B_{2,3}E_{2,3}$

Figure 2.6d: Final block rhotrix

First, align the blocks of A , D and B , E in such a way that each process multiplies its local submatrices:

- Shift sub matrices $A_{i,j}$ and $D_{i,j}$ to the left (with wraparound) by i steps.
- Shift sub matrices $B_{i,j}$ and $E_{i,j}$ up (with wraparound) by j steps.
- Perform local block matrix multiplication.
- Next, each block of A , D moves one step left and each block of B , E moves one step up (again with wraparound).

Perform next block multiplication; add to partial result, repeat until all the \sqrt{p} blocks have been multiplied as shown in Figure 2.6d.

Two rhotrices Q and R are considered for illustration in Figure 2.7a and 2.7b:

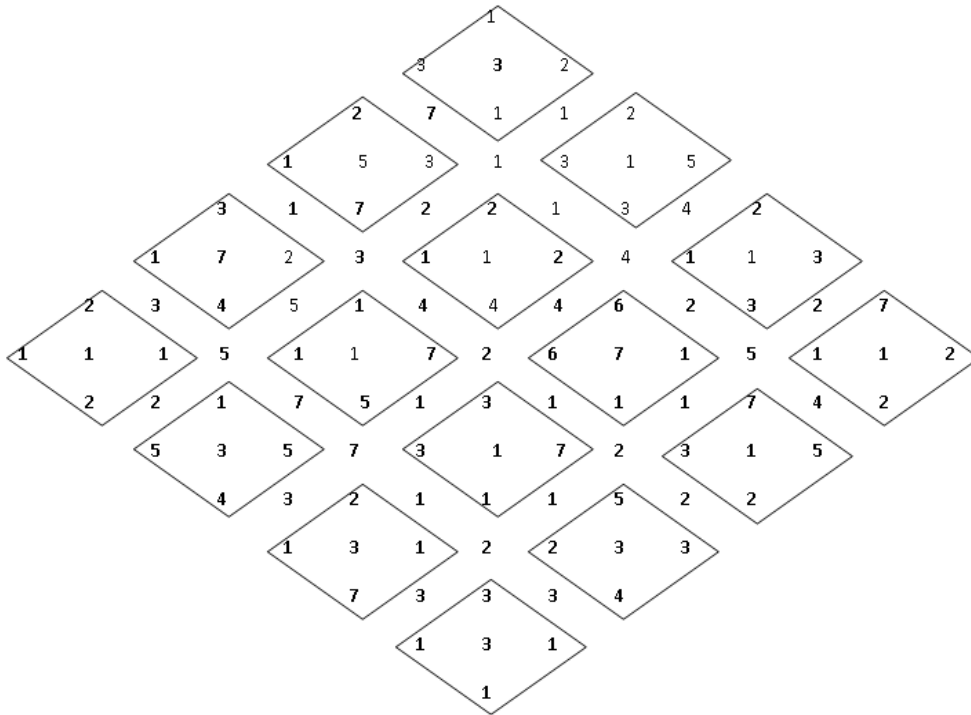


Figure 2.7a: Example of blocking process of fifteen dimensional rhotrix, R

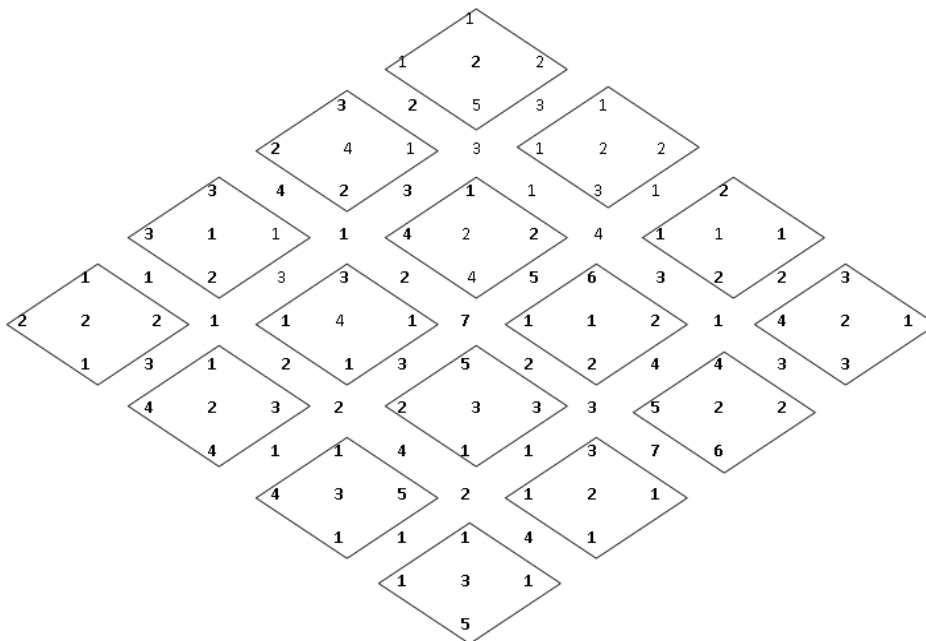


Figure 2.7b: Example of blocking process of fifteen dimensional rhotrix Q

$$A = \begin{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix} & \begin{bmatrix} 2 & 5 \\ 3 & 3 \end{bmatrix} & \begin{bmatrix} 2 & 3 \\ 1 & 3 \end{bmatrix} & \begin{bmatrix} 7 & 2 \\ 1 & 2 \end{bmatrix} \\ \begin{bmatrix} 2 & 3 \\ 1 & 7 \end{bmatrix} & \begin{bmatrix} 2 & 2 \\ 1 & 4 \end{bmatrix} & \begin{bmatrix} 6 & 1 \\ 6 & 1 \end{bmatrix} & \begin{bmatrix} 7 & 5 \\ 3 & 2 \end{bmatrix} \\ \begin{bmatrix} 3 & 2 \\ 1 & 4 \end{bmatrix} & \begin{bmatrix} 1 & 7 \\ 1 & 5 \end{bmatrix} & \begin{bmatrix} 3 & 7 \\ 3 & 1 \end{bmatrix} & \begin{bmatrix} 5 & 3 \\ 2 & 4 \end{bmatrix} \\ \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} & \begin{bmatrix} 1 & 5 \\ 5 & 4 \end{bmatrix} & \begin{bmatrix} 2 & 1 \\ 1 & 7 \end{bmatrix} & \begin{bmatrix} 3 & 1 \\ 1 & 1 \end{bmatrix} \end{bmatrix} \quad D = \begin{bmatrix} \begin{bmatrix} 3 & 1 \\ 7 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 4 \\ 1 & 4 \end{bmatrix} & \begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix} & \begin{bmatrix} 1 \\ 4 \end{bmatrix} \\ \begin{bmatrix} 5 & 2 \\ 1 & 3 \end{bmatrix} & \begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} & \begin{bmatrix} 7 & 1 \\ 1 & 2 \end{bmatrix} & \begin{bmatrix} 1 \\ 2 \end{bmatrix} \\ \begin{bmatrix} 7 & 5 \\ 3 & 5 \end{bmatrix} & \begin{bmatrix} 1 & 5 \\ 7 & 7 \end{bmatrix} & \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} & \begin{bmatrix} 3 \\ 3 \end{bmatrix} \\ \begin{bmatrix} 1 & 2 \\ 2 & 3 \end{bmatrix} & \begin{bmatrix} 2 & 3 \\ 3 & 3 \end{bmatrix} & \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix} & \begin{bmatrix} 3 \\ 3 \end{bmatrix} \end{bmatrix}$$

Figure 2.8a: Block of matrices of main and hearts entries of rhotrix R

$$B = \begin{bmatrix} \begin{bmatrix} 1 & 2 \\ 1 & 5 \end{bmatrix} & \begin{bmatrix} 1 & 2 \\ 1 & 3 \end{bmatrix} & \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} & \begin{bmatrix} 3 & 1 \\ 4 & 3 \end{bmatrix} \\ \begin{bmatrix} 3 & 1 \\ 2 & 2 \end{bmatrix} & \begin{bmatrix} 1 & 2 \\ 4 & 4 \end{bmatrix} & \begin{bmatrix} 6 & 2 \\ 1 & 1 \end{bmatrix} & \begin{bmatrix} 4 & 2 \\ 5 & 6 \end{bmatrix} \\ \begin{bmatrix} 3 & 1 \\ 3 & 2 \end{bmatrix} & \begin{bmatrix} 3 & 1 \\ 1 & 1 \end{bmatrix} & \begin{bmatrix} 5 & 3 \\ 2 & 1 \end{bmatrix} & \begin{bmatrix} 3 & 1 \\ 1 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 3 \\ 4 & 4 \end{bmatrix} & \begin{bmatrix} 1 & 5 \\ 4 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 1 \\ 1 & 5 \end{bmatrix} \end{bmatrix} \quad E = \begin{bmatrix} \begin{bmatrix} 2 & 3 \\ 2 & 3 \end{bmatrix} & \begin{bmatrix} 2 & 1 \\ 1 & 4 \end{bmatrix} & \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix} & \begin{bmatrix} 2 \\ 3 \end{bmatrix} \\ \begin{bmatrix} 4 & 3 \\ 4 & 1 \end{bmatrix} & \begin{bmatrix} 2 & 5 \\ 2 & 7 \end{bmatrix} & \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix} & \begin{bmatrix} 2 \\ 7 \end{bmatrix} \\ \begin{bmatrix} 1 & 3 \\ 1 & 1 \end{bmatrix} & \begin{bmatrix} 4 & 3 \\ 2 & 2 \end{bmatrix} & \begin{bmatrix} 2 & 1 \\ 4 & 2 \end{bmatrix} & \begin{bmatrix} 2 \\ 4 \end{bmatrix} \\ \begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} & \begin{bmatrix} 2 & 1 \\ 3 & 1 \end{bmatrix} & \begin{bmatrix} 3 & 1 \\ 3 & 1 \end{bmatrix} & \begin{bmatrix} 3 \\ 3 \end{bmatrix} \end{bmatrix}$$

Figure 2.8b: Block of matrices of main and hearts entries of rhotrix Q

Computation at Process P_{00} , C_{00} and H_{00}

After the first alignment

$$C_{00} = A_{00} + B_{00} = \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 1 & 5 \end{bmatrix} = \begin{bmatrix} 3 & 12 \\ 4 & 11 \end{bmatrix}$$

$$H_{00} = D_{00} + E_{00} = \begin{bmatrix} 3 & 1 \\ 7 & 1 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 8 & 12 \\ 16 & 24 \end{bmatrix}$$

After the first shift

$$\begin{aligned} C_{00} &= C_{00} + A_{01} \times B_{10} = \begin{bmatrix} 3 & 12 \\ 4 & 11 \end{bmatrix} + \begin{bmatrix} 2 & 5 \\ 3 & 3 \end{bmatrix} \times \begin{bmatrix} 3 & 1 \\ 2 & 2 \end{bmatrix} \\ &= \begin{bmatrix} 3 & 12 \\ 4 & 11 \end{bmatrix} + \begin{bmatrix} 16 & 12 \\ 15 & 9 \end{bmatrix} = \begin{bmatrix} 19 & 24 \\ 19 & 20 \end{bmatrix} \\ H_{00} &= H_{00} + D_{00} \times E_{00} = \begin{bmatrix} 8 & 12 \\ 16 & 24 \end{bmatrix} + \begin{bmatrix} 1 & 4 \\ 1 & 4 \end{bmatrix} \times \begin{bmatrix} 4 & 3 \\ 4 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 8 & 12 \\ 16 & 24 \end{bmatrix} + \begin{bmatrix} 20 & 7 \\ 20 & 7 \end{bmatrix} = \begin{bmatrix} 28 & 19 \\ 36 & 31 \end{bmatrix} \end{aligned}$$

After the second shift

$$\begin{aligned} C_{00} &= C_{00} + A_{02} \times B_{20} = \begin{bmatrix} 19 & 24 \\ 19 & 20 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 1 & 3 \end{bmatrix} \times \begin{bmatrix} 3 & 1 \\ 3 & 2 \end{bmatrix} \\ &= \begin{bmatrix} 19 & 24 \\ 19 & 20 \end{bmatrix} + \begin{bmatrix} 15 & 8 \\ 12 & 7 \end{bmatrix} = \begin{bmatrix} 34 & 32 \\ 31 & 27 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} H_{00} &= H_{00} + D_{02} \times E_{20} = \begin{bmatrix} 28 & 19 \\ 36 & 31 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix} \times \begin{bmatrix} 1 & 3 \\ 1 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 28 & 19 \\ 36 & 31 \end{bmatrix} + \begin{bmatrix} 3 & 5 \\ 7 & 11 \end{bmatrix} = \begin{bmatrix} 31 & 24 \\ 43 & 42 \end{bmatrix} \end{aligned}$$

After the third shift

$$\begin{aligned} C_{00} &= C_{00} + A_{03} \times B_{30} = \begin{bmatrix} 34 & 32 \\ 31 & 27 \end{bmatrix} + \begin{bmatrix} 7 & 2 \\ 1 & 2 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 34 & 32 \\ 31 & 27 \end{bmatrix} + \begin{bmatrix} 11 & 16 \\ 5 & 4 \end{bmatrix} = \begin{bmatrix} 45 & 48 \\ 36 & 31 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} H_{00} &= H_{00} + D_{03} \times E_{30} = \begin{bmatrix} 31 & 24 \\ 43 & 42 \end{bmatrix} + \begin{bmatrix} 1 \\ 4 \end{bmatrix} \times \begin{bmatrix} 2 & 3 \end{bmatrix} \\ &= \begin{bmatrix} 31 & 24 \\ 43 & 42 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 8 & 12 \end{bmatrix} = \begin{bmatrix} 33 & 27 \\ 51 & 54 \end{bmatrix} \end{aligned}$$

2.5.3. Systolic Array

Similar in operation to the Cannon's algorithm is the systolic array or two-dimensional pipeline, as it is often referred to. The word systolic has been borrowed from the medical field, just as the heart pumps blood, information is pumped through a systolic array in various directions, and at regular intervals (Wilkinson and Allen, 2005). In the two-dimensional systolic array considered in this thesis, information is pumped from left to right and from top to bottom. The information meets at interval nodes where the processing occurs. The same information passes onward (left to right or downward).

Consider a systolic array used to multiply two 4 x 4 matrices, A and B . The elements of A enters from left and the elements of B enter from the top. The final product terms of C will be held in the processors, as shown in Figure 2.9. A suitable numbering of processors uses x and y coordinates starting at $(0,0)$ in the top-left corner. Each processor, $P_{i,j}$, repeatedly performs the same algorithm (after c is initialized to zero) which accumulates the required summations.

```

recv(&a, pi,j-1);           // receive from left
recv(&b, pi-1,j);           // receive from right
c = c + a * b;              // accumulate value for ci,j
send(&a, pi,j+1);          // send to right
send(&b, pi+1,j);          // send downwards

```

At $P_{0,0}$, first, $a_{0,0}$ and $b_{0,0}$ enter $P_{0,0}$, to produce $c_{0,0} = a_{0,0}b_{0,0}$. Both $a_{0,0}$ and $b_{0,0}$ are passed onward, $a_{0,0}$ right to $P_{0,1}$, and $b_{0,0}$ down to $P_{1,0}$. Next, $P_{0,0}$ receives $a_{0,1}$ from the left and $b_{1,0}$ from above. After these numbers are multiplied, the product is added to $c_{0,0}$, and $a_{0,1}$ and $b_{1,0}$ are passed onward. Continuing with each of the other numbers that enter from left and top, finally after four “cycles,” we get the required results:

$$c_{0,0} = a_{0,0}b_{0,0} + a_{0,1}b_{1,0} + a_{0,2}b_{2,0} + a_{0,3}b_{3,0}$$

$P_{0,1}$ operates in a similar manner but receives numbers from $P_{0,0}$ to the left one cycle later. Hence, the numbers from the top are also delayed by one cycle. $P_{0,1}$ starts with multiplying $a_{0,0}$ and $b_{0,1}$ and after four cycles obtains the value for $c_{0,1}$ as:

$$c_{0,1} = a_{0,0}b_{0,1} + a_{0,1}b_{1,1} + a_{0,2}b_{2,1} + a_{0,3}b_{3,1}$$

A similar computation occurs with each of the other processors. The computation operates in a synchronous fashion, timed by the *send()*s and *recv()*s. The method is also applicable to submatrix decomposition, in such a way the submatrices are used in place of the matrix elements.

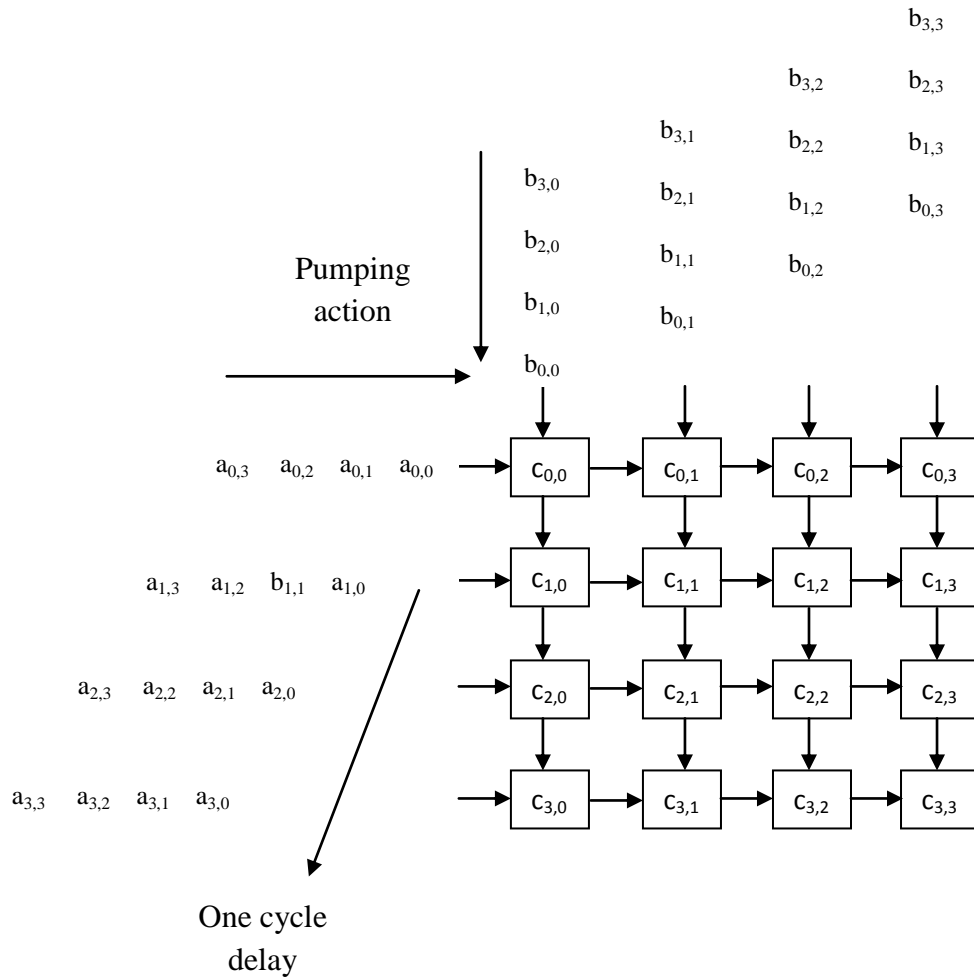


Figure 2.9: Matrix multiplication using systolic array

2. 6. Background Concept from Group Theory to Complex Analysis

The exponent of matrix multiplication has proved so elusive that it has been given a name ω , which in essence is described as the smallest number such that $O(n^{\omega+\varepsilon})$ multiplications suffice for all $\varepsilon > 0$. Similarly, since all n^2 must be part of any computation, ω is clearly at least 2.

The road to the current best upper bound on ω was built on clever ideas and increasingly complex combinatorial techniques. Many researchers long believed that the standard, $O(n^3)$ algorithm was the best possible. Then, Strassen in 1969 stunned the research world with his $O(n^{2.81})$ algorithm for multiplying matrices. Still sometimes used in practice, Strassen's algorithm is reminiscent of a shortcut, first observed by Gauss, for multiplying complex numbers. Though finding the product $(a + bi)(c + di) = ac - bd + (bc + ad)i$ seems to require four multiplications, Gauss observed that it can actually be done with three multiplication ac , bd , and $(a + b)(c + d)$, because $bc + ad = (a + b)(c + d) - ac - bd$.

Let $A, B \in C = \mathfrak{R}[i]/(i^2 + 1)$, where $A = (a + ib)$ and $B = (c + id)$, then $AB = (ac - bd) + (ad + bc)i$.

This process uses 4 multiplications. How possible is optimality? Certainly by definition:

$$m_1 = (a + b)(c + d) = (ac - bd) + bc - ad$$

$$m_2 = bc$$

$$m_3 = ad$$

$$AB = (m_1 - m_2 + m_3) + (m_2 + m_3)i$$

Thus, complex multiplication is accomplished with only 3 multiplications over \mathfrak{R} .

2.7. A Review of Strassen's Algorithm

Strassen in 1969 introduced an algorithm to compute matrix multiplications. This algorithm reduces the number of multiplications required for computing the product of two 2×2 matrices A and B from eight to seven, by expressing the entries of AB as linear combinations of products of linear combinations of the entries of A and B. This trick, applied to four blocks of $2^n \times 2^n$ matrices, reduces the problem to seven multiplications of $2^{n-1} \times 2^{n-1}$ matrices.

Let $A, B, C \in \mathfrak{R}^{2 \times 2}$

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \quad (2.1)$$

$$M1 := (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2})$$

$$M2 := (A_{2,1} + A_{2,2})B_{1,1}$$

$$M3 := A_{1,1}(B_{1,2} - B_{2,2})$$

$$M4 := A_{2,2}(B_{2,1} - B_{1,1})$$

$$M5 := (A_{1,1} + A_{1,2})B_{2,2}$$

$$M6 := (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2})$$

$$M7 := (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})$$

$$C_{1,1} = M1 + M4 - M5 + M7$$

$$C_{1,2} = M3 + M5$$

$$C_{2,1} = M2 + M4$$

$$C_{2,2} = M1 - M2 + M3 + M6$$

In essence, instead of using the normal 8 multiplications of trivial approach, Strassen's algorithm only uses 7. We can continue to apply Strassen's algorithm recursively to achieve the time complexity of $O(n^{\log_2 7}) = O(n^{2.807})$. Considering $A_{i,j}$, $B_{i,j}$, and $C_{i,j}$ as matrices instead of scalars allows matrix multiplication over $n = 2^N$ size matrices to be performed using only $7^N = 7^{\log_2 n} = n^{\log_2 7} = O(n^{2.81})$. Which means it will run faster than the conventional algorithm for sufficiently large matrices.

Unless the matrix has a dimension of 2^k you cannot apply Strassen's algorithm directly. A method called dynamic peeling is able to solve this problem effectively (Huss-Lederman and Jacobson, 1996). Dynamic peeling makes matrix dimensions even by stripping off an extra row or column as needed, and putting their contributions back to the final result later.

This interesting feature motivated us to incorporate this algorithm into the present research area. From our earlier definition of rhotrices, we related its row-column multiplication to be similar in every aspect with that of matrices multiplication and hence the importance of the present study. We believe that the Strassen's algorithm can also be applied to the row-column multiplication of rhotrices even though; rhotrix is always of odd dimension.

Huss-Lederman pioneered the design of an efficient and portable serial implementation of Strassen's algorithm called DGEFMM (Huss-Lederman and Jacobson, 1996). DGEFMM is designed to replace DGEMM and obtained better performance for all matrix sizes while minimizing the temporary storage. DGEMM, $C = op(A) \times op(B)$ (IBM, 2004).

There are several parallel implementation of Strassen's algorithm on distributed memory architectures. The methods typically belong to three classes. The first class is to use the conventional algorithm at the top level (across processors) and Strassen's algorithm at the bottom level (within a processor). One basic application of this is that of the Fox's Broadcast-Multiply-Roll (BMR) methods (Fox *et al.*, 1987). Another similar implementation approach is found in (Ohtaki *et al.*, 2004) but is more focused on a distribution scheme particularly for heterogeneous clusters. The second class is to use Strassen's algorithm at both the top and bottom level. Chou *et al.* (1995) decomposes the matrix A into 2 x 2 blocks of submatrices, then further decomposes each submatrix into four 2 x 2 blocks (i.e., 4 x 4 blocks). This way he is able to identify 49 multiplications and uses 7 or 49 processors to perform multiplication concurrently.

Table 2.1 Strassen's Algorithm

Phase 1	
$T1 = A11 + A22$	$T6 = B11 + B22$
$T2 = A21 + A22$	$T7 = B12 - B22$
$T3 = A11 + A12$	$T8 = B21 - B11$
$T4 = A21 - A11$	$T9 = B11 + B12$
$T5 = A12 - A22$	$T10 = B21 + B22$

Phase 2	
$Q1 = T1 \times T6$	$Q5 = T3 \times B22$
$Q2 = T2 \times B11$	$Q6 = T4 \times T9$
$Q3 = A11 \times T7$	$Q7 = T5 \times T10$
$Q4 = A22 \times T8$	
Phase 3	
$T1 = Q1 + Q4$	$T3 = Q3 + Q1$
$T2 = Q5 - Q7$	$T4 = Q2 - Q6$
Phase 4	
$C11 = T1 - T2$	$C12 = Q3 + Q5$
$C21 = Q2 + Q4$	$C22 = T3 - T4$

We equally believe that if such task can be achieved by portioning matrices into sub-blocks of 2 x 2, we can equally extend this method to rhotrices, by a way of partitioning the heart entries into 2 x 2 sub-blocks and then applying the odd dimension techniques of solving the Strassen's algorithm as discussed in (Huss-Lederman *et al.*, 1996). The last class is using Strassen's algorithm at the top level and the conventional one at the extra matrix additions imposed by Strassen's algorithm becomes less compared to the saved matrix multiplication cost. Therefore Strassen's algorithm is better used across processors on the top level.

Finally, for matrices with odd dimensions, some technique must be applied to make the dimensions even, apply Strassen's algorithm to the altered matrix, and then correct the results. Originally, Strassen suggested padding the input matrices with extra rows and columns of zeros, so that the dimensions of all the matrices encountered during the recursive calls are even. After the product has been computed, the extra rows and columns are removed to obtain the desired result. We call this approach static padding, since padding occurs before any recursive calls to Strassen's algorithm. Alternatively, each time Strassen's algorithm is called recursively, an extra row of zeros

can be added to each input with an odd row-dimension and an extra column of zeros can be added for each input with an odd column-dimension. This approach to padding is called dynamic padding since padding occurs throughout the execution of Strassen's algorithm. A version of dynamic padding is used in (Huss-Lederman and Jacobson, 1996).

Let $C = AB$ be the desired matrix product, and let be a matrix with an even dimension, where * denotes an unspecified value. Then Strassen's algorithm can be used to compute

$$\tilde{A} = \left(\begin{array}{ccc|c} & & & 0 \\ & A & & \vdots \\ & & & 0 \\ \hline * & \dots & * & * \end{array} \right) \text{ and } \tilde{B} = \left(\begin{array}{ccc|c} & & & * \\ & B & & \vdots \\ & & & * \\ \hline * & \dots & * & * \end{array} \right)$$

$$\tilde{A}\tilde{B} = \left(\begin{array}{ccc|c} & & & * \\ & AB & & \vdots \\ & & & * \\ \hline * & \dots & * & * \end{array} \right)$$

The product $C = AB$ appears in the upper-left block independent of the values substituted for the * 's in A and B. The same would be true if the zeros would have been placed in the bottom row of B instead of the last column of A. In the above diagram one extra row and column has been inserted; however, if desired, additional rows and columns can be inserted so long as the resulting matrix has even dimension. Furthermore, the extra rows and columns could have been inserted in arbitrary locations so long as the column index in the first matrix matches up with the row index in the second matrix. Typically the extra rows and columns contain zeros and are inserted so that A and B are in the top-left or bottom-right blocks (Douglas *et al.*, 1994).

Another approach, called dynamic peeling, deals with odd dimensions by stripping off the extra rows and/or columns as needed, and adding their contributions to the final result in a later round of fixup work. The fixup is required to include the contribution of the rows and columns that were

deleted. More specifically, let A be an $m \times k$ matrix and B be a $k \times n$ matrix. Assuming that m , k , and n are all odd, A and B are partitioned into the block matrices

$$A = \left(\begin{array}{c|c} A_{11} & a_{12} \\ \hline a_{21} & a_{22} \end{array} \right),$$

where A_{11} has even dimensions, a_{12} is a column vector, a_{21} is a row vector, and a_{22} is a single element. Similarly,

$$B = \left(\begin{array}{c|c} B_{11} & b_{12} \\ \hline b_{21} & b_{22} \end{array} \right),$$

where B_{11} has even dimensions, b_{12} is a column vector, b_{21} is a row vector, and b_{22} is a single element.

$$\left(\begin{array}{c|c} C_{11} & c_{12} \\ \hline c_{21} & c_{22} \end{array} \right) = \left(\begin{array}{c|c} A_{11}B_{11} + a_{12}b_{21} & A_{11}b_{12} + a_{12}b_{22} \\ \hline a_{21}B_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{array} \right),$$

where $A_{11}B_{11}$ is computed using Strassen's algorithm, and the other computations constitute the fixup work. In Section 3, we discussed how we actually applied similar Strassen's technique to design our own rhotrix multiplication algorithm and coded its computations.

2.8. Conversion of Rhotrix to Coupled Matrix

It is clear that transposing any rhotrix or matrix is equivalent to rotating its columns through 90^0 in an anti-clockwise direction. In the same way, if columns of a rhotrix are rotated through 45^0 , what we get is a special form of matrix with missing values. This can further be illustrated by considering the rhotrix R_5 for instance

$R_5 = \left\langle \begin{array}{cccc} & & a_{11} & & \\ & a_{21} & c_{11} & a_{12} & \\ a_{31} & c_{21} & a_{22} & c_{12} & a_{13} \\ & a_{32} & c_{22} & a_{23} & \\ & & a_{33} & & \end{array} \right\rangle$, using half transpose of rhotrices defined in (Sani, 2007), we

have the half transpose of R_5 denoted by $R_5^{T/2}$, as

$$R_5^{T/2} = \begin{pmatrix} a_{11} & & a_{12} & & a_{13} \\ & c_{11} & & c_{12} & \\ a_{21} & & a_{22} & & a_{23} \\ & c_{21} & & c_{22} & \\ a_{31} & & a_{32} & & a_{33} \end{pmatrix} \quad (2.2)$$

$R_5^{T/2}$ is known as coupled matrix (Sani, 2007).

where $T/2$ indicates a rotation through 45^0 in an anti-clockwise direction and may be termed as ‘half transpose’. The special matrix in (2.2) is like coupling a 3×3 matrix with a 2×2 matrix, which is why we can call it ‘a coupled matrix’. In the general case therefore, we have

$$\Rightarrow R_n^{T/2} = \left\langle a_{ij}, c_{lk} \right\rangle^{T/2} = \left\langle a_{ij}, c_{lk} \right\rangle = [AC]_n \quad (2.1)$$

which is a coupled matrix, coupling a $t \times t$ matrix with $a(t-1) \times (t-1)$ matrix, where $t = (n+1)/2$ as shown in (Sani, 2007). To multiply any two coupled matrices, $[Ac]_n$ and $[Bd]_n$, we simply fill the missing spaces with zeros and multiply as usual for any two matrices. After the multiplication, we remove the zeros so as to end up with a coupled matrix.

Thus for two coupled matrices, coupling $t \times t$ matrices with $(t - 1) \times (t - 1)$ matrices, we get,

$$\Rightarrow R_n \circ Q_n = \left\langle a_{i_1 j_1}, c_{l_1 k_1} \right\rangle \circ \left\langle b_{i_2 j_2}, d_{l_2 k_2} \right\rangle = \sum_{j_1=1}^t (a_{i_1 j_1} b_{i_2 j_2}), \sum_{k_1=1}^{t-1} (c_{l_1 k_1} d_{l_2 k_2}) \quad (2.2)$$

which is the same method of multiplication for rhotrices (Sani, 2007). That means, the $t \times t$ matrices in the two coupled matrices are multiplied together using row–column multiplication and the $(t - 1) \times (t - 1)$ matrices in the two coupled matrices are also multiplied together using row–column multiplication.

2.8.1. Solving $(m \times n) \times (n \times m)$ and $(m-1) \times (n-1)$ Coupled Matrix Problems

By filling coupled matrix with zeroes, we can have a generalized representation as shown below:

$$\begin{bmatrix} a_{11} & 0 & a_{12} & 0 & \dots & \dots & 0 & a_{1t} \\ 0 & c_{11} & 0 & c_{12} & \dots & \dots & c_{1t-1} & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & c_{t-11} & 0 & c_{t-12} & \dots & \dots & c_{t-1t-1} & 0 \\ a_{t1} & 0 & a_{t2} & 0 & \dots & \dots & 0 & a_{tt} \end{bmatrix}$$

One great advantage of this conversion process could be traced to finding application areas for rhotrices especially in the field of engineering and industries. The coupled matrix is generally seen as a super matrix, and hence makes rhotrix multiplication suitable for application in the domain of scientific and engineering oriented problems.

For the simple fact that, coupled matrix assumes all the properties of a matrix, then every computation extended on a matrix is equally applicable to it, and similar representation in a computer system is assumed possible. Consider the multiplication of any two rhotrices, say R_{15} and Q_{15} . The two rhotrices are first coupled to form coupled matrices, $[AD]$ and $[BE]$. If we fill the missing spaces of these matrices with zeros as explained in the last section, we get an $m \times n$ and $n \times m$ square matrix. The two rhotrix below could then be coupled as:

2.8.2. Parallel Computing with Processor Array Overview

The increasing demands for high-performance scientific computations indicate the need for tremendous computing capability, in terms of both volume and speed. The availability of low-cost, high-density, fast processing/memory devices will presage a major breakthrough in future supercomputer designs, especially in the design of highly concurrent processors. Currently parallel computers can be characterized into three structural classes: vector processors, multiprocessor systems, and array processors (Hwang and Briggs, 1984). The first two classes belong to the general-purpose computer domain. The development of these systems requires a complicated design of control units and optimized schemes for allocation of machine resources. The third class, however, belongs to the domain of special-purpose computers. The design of such systems requires a broad knowledge of the relationship between parallel computing algorithms and optimal-computing hardware and software structures.

It is this last class that we shall focus our parallel algorithm design upon, since it offers a promising solution to meet real-time processing requirements. Especially, locally interconnected computing networks, such as systolic and wavefront arrays, are well suited to efficiently implement a major class of signal processing and large matrix multiplication algorithms due to their massive parallelism and regular data flow (Kung, 1982). Such architectures promise real-time solutions to a large variety of advanced computational tasks.

2.8.3. Architectural Considerations in Array Processor Design

There are many important issues in designing array processor systems, such as processor interconnection, system clocking, and modularity (Kung and Gal-Ezer, 1982). Interconnection in massively parallel array processors is the most critical issue of the system design, since communication is very expensive in terms of area, power, and time consumption (Mead and Conway, 1980). Therefore, communication has to be restricted to localized interconnections. To avoid global interconnections, local and regular data movements are strongly preferred. This is the most salient characteristic of systolic and wavefront arrays.

The clocking scheme is also a very critical issue. In the globally synchronous scheme, there is a global clock network which distributes the clock signal over the entire array. For very large systems, the clock skew incurred in global clock distribution is a nontrivial factor, causing

unnecessary slowdown in the clock rate (Mead and Conway, 1980). Under this circumstance, the self-timed scheme is more preferable.

Large design of layout costs suggests using repetitive modular structures, that is, a few different types of simple (and often standard) cells. Thus we have to identify the primitives that can be implemented efficiently and optimally realize the potential of new device technologies. Programmable processor modules (as opposed to dedicated modules) are favoured due to cost-effectiveness considerations. The high cost of designing such modules may be amortized over a broader application domain indeed; a major portion of scientific computations can be reduced to a basic set of matrix operations and related algorithms. These should be exploited in order to simplify the hardware module.

2.8.4. Mapping Parallel Algorithms onto Locally Interconnected Computing Networks

An array processor is composed of an array of processor elements (PE) with direct (static) or direct (dynamic) interconnections, including linear, orthogonal, hexagonal, tree, perfect-shuffle, or other types of structure. The most critical issue is communication, that is, moving data between PE's in a large-scale interconnection network. Correspondingly, a communication-oriented analysis on parallel algorithms will be most useful for mapping algorithms onto the arrays. To conform with the constraints imposed by VLSI, this research work will emphasize a special class of algorithm, that is, recursive and locally dependent algorithms (in a recursive algorithm, all processors do nearly identical tasks and each processor repeats a fixed set of tasks on sequentially available data).

For proper communication in an interconnected computing network, each PE in the array should know:

1. Where to send (or fetch) data, and
2. When to send (or fetch) data

When mapping a locally recursive algorithm onto a computing network, it allows a simple solution to the question "where to send the data?" because the data movements can be confined to nearest neighbour PE's. Therefore, locally interconnected computing networks will suffice to execute the algorithm with high performance. The conventional approach to the second question

“when to send the data?” is to use a globally synchronous scheme, where the timing is controlled by a sequence of “beats” (Ullman, 1984). A prominent example is the systolic array (Kung, 1982). However, locality can have two meanings in array processor designs: localized data transactions and /or localized timing scheme (that is, using self-timed, data-driven control). In fact, the class of locally recursive algorithms permits both locality features; these should be exploited in the architectural designs. An example for such a design is the wavefront array (Kung *et al.*, 1982).

2.9. Conclusion

In this section we presented a review work on the possibility of parallelising rhotrix multiplication using Cannon algorithm for equation (1.3), the aim was to achieve a faster computational result through partitioning of rhotrix elements in block form. This method seems not to leave rhotrix with any uniqueness as its still relate the procedure to that of existing matrix multiplication, thus, we intend to build upon this lapses found during partitioning and movement of the data entries for matrix which might not be feasible for a rhotrix. In the next section, we explore a more promising algorithmic approach for multiplying n-dimensional rhotrix in both sequential and parallel environment.

CHAPTER THREE: SEQUENTIAL AND PARALLEL ALGORITHMIC DESIGN FOR RHOTRIX MULTIPLICATION

3.1. Introduction

In this chapter, we discuss algorithm design for both sequential and parallel rhotrix multiplication. The design issues will cover the two well known method of rhotrix multiplication which are known as the heart-oriented multiplication by Ajibade (2003) and row-column multiplication by Sani (2004) from literature. It is cognisant to note that both methods actually differ by their approach, logic in which rhotrix multiplication is performed and likewise result obtained. First, we present Ajibade's method; in this case we consider a direct multiplication of rhotrix main elements with just a single heart of the opposite multiplicand. Then we also consider Sani's method, whose approach likened rhotrix multiplication to the well known conventional row-column square matrix multiplication. Finally, algorithmic designs for these methods are presented.

3.2. The Algebra of Vector dot-product

This section is almost a digression. It is devoted to the study of the simplest variant of the problem. In mathematics, the dot product, also known as the scalar product, is an operation which takes two vectors over the real numbers, R , and returns a real-valued scalar quantity. It is the standard inner product of the Euclidean space. In Euclidean geometry, the dot product, length, and angle are related: For a vector a , $a.a$ is the square of its length, and more generally, if b is another vector

$$a . b = |a| . |b| \cos \theta$$

Where

- $|a|$ and $|b|$ denote the length (magnitude) of a and b ,
- θ is the angle between them.

Since $|a|\cos(\theta)$ is the scalar projection of a onto b , the dot product can be understood geometrically as the product of the length of this projection and the length of b .

In linear algebra, the dot product of two vectors $a = [a_1, a_2, \dots, a_n]$ and $b = [b_1, b_2, \dots, b_n]$ is defined as follows:

$$a.b = \sum_{i=1}^n a_i \cdot b_i = a_1.b_1 + a_2.b_2 + \dots + a_n.b_n$$

The rhotrix main entries (vectors) or elements to heart multiplication could in some ways be likened to the same mathematical scalar dot product described above. In this paper, we aim to find an efficient implementation of this product computation on computers system. This implementation can also be considered an issue of prime importance when providing such systems with scientific software libraries or applications which use the dot product computation as data processing, especially for such area as rhotrices algebra which is a relatively new field of study.

3.2.1. Heart-Oriented Rhotrix Multiplication

A set of rhotrices of dimension three was defined in (Ajibade, 2003) and extension in the dimension was considered possible. For instance a set of rhotrices of dimension five can be defined as

$$R = \left\{ \left\langle \begin{array}{ccccc} & & a & & \\ & b & c & d & \\ e & f & h(R) & h & i \\ & j & k & l & \\ & & m & & \end{array} \right\rangle : a, b, c, \dots, m \in \mathfrak{R} \right\}, \quad (3.2.1)$$

where $h(R)$ is called the heart of R . The heart of a rhotrix is always defined as the element at the perpendicular intersection of the two diagonals of a rhotrix.

The first method for multiplication of three dimensional rhotrices was defined in (Ajibade, 2003), and can be extended to n dimensional rhotrix. Of interest in this paper for the sake of illustration and clarity is five dimensional rhotrices depicted as follows:

$$R_5 \circ Q_5 = \left\langle \begin{array}{ccccc} & & a & & \\ & b & c & d & \\ e & f & h(R) & h & i \\ & j & k & l & \\ & & m & & \end{array} \right\rangle \circ \left\langle \begin{array}{ccccc} & & n & & \\ & o & p & q & \\ r & s & h(Q) & t & u \\ & v & w & x & \\ & & y & & \end{array} \right\rangle \quad (3.2.2)$$

It should be noted that, the general concept of rhotrices is still at its elementary stage of development, as we are yet to establish a generalized form for its multiplication. Having ascertained this belief, the multiplication denoted by ‘ \circ ’, can be defined in many ways, but in this section we defined rhotrix multiplication based on the initial method given by (Ajibade, 2003), but with an extension to n dimension. The multiplication of the two rhotrices above is given by;

$$= \left\langle \begin{array}{cccc} & & ah(Q) + nh(R) & \\ & bh(Q) + oh(R) & ch(Q) + ph(R) & dh(Q) + qh(R) \\ eh(Q) + rh(R) & fh(Q) + sh(R) & h(R)h(Q) & hh(Q) + th(R) & ih(Q) + uh(R) \\ & jh(Q) + vh(R) & kh(Q) + wh(R) & lh(Q) + xh(R) & \\ & & mh(Q) + yh(R) & & \end{array} \right\rangle \quad (3.2.3)$$

Presented as part of this research work, is a generalized multiplication algorithm of n-dimensional rhotrix; which is an extension of work presented in (Ajibade, 2003). The main objective of this section is to establish an ideal multiplicative concept and algorithm design suitable for higher dimensional heart-oriented rhotrix multiplication and present them for both sequential and parallel computation (Ezugwu *et al*, 2011a).

Ajibade (2003), indicated that the dimension of a rhotrices can be increased although a rhotrix would always have an odd dimension. He also indicated that a rhotrix R_n of dimension n will have $|R_n|$ entries where $|R_n| = \frac{1}{2}(n^2 + 1)$. Let's consider generalizing any given rhotrix R_n with entries $a_1, a_2, \dots, a_{\frac{1}{2}(n^2+1)}$, and we assume that the following holds:

- If we denote the number of entries in a rhotrix by N , then the middle entry, known as the heart element, can be expressed as $H = \frac{1}{2}(N + 1)$ from statistical distribution expression for median. In our case, the value of H indicates the index of the heart entry.
- Similarly if $N = |R_n|$ then $N = \frac{1}{2}(n^2 + 1)$, hence, $H = \frac{\frac{1}{2}(n^2 + 1) + 1}{2} \equiv \frac{1}{4}[n^2 + 3]$.

Some results: we can derive a generalization for high dimensional rhotrix by considering some sequence of 3, 5, 7, 9 etc dimensional rhotrices as illustrated below:

For a rhotrix of dimension 3, we have $2 \times 1 + 1 = 3$

For a rhotrix of dimension 5, we have $2 \times 2 + 1 = 5$

For a rhotrix of dimension 7, we have $2 \times 3 + 1 = 7$

For a rhotrix of dimension 9, we have $2 \times 4 + 1 = 9$

...

....

...

....

For a rhotrix of dimension n , we have $2 \times k + 1 = n$

Where k is the k^{th} term of the incremental value and n is the dimension of the rhotrix(which is always an odd number), and then we have: $k = \frac{1}{2}(n-1)$ or $\lfloor \frac{n}{2} \rfloor$

- Again, if we denote the direction of the leftmost entry of a rhotrix by L and the number of entries by $|R_n| = \frac{1}{2}(n^2 + 1)$ then the rhotrix entry at L is given by $L = \frac{1}{4}[n^2 + 3] - \frac{n-1}{2}$

or $L = H - K$

- Similarly, the rightmost rhotrix entry denoted by R , is given by $R = \frac{1}{4}[n^2 + 3] + \frac{n-1}{2}$

or $L = H + K$

- It is important to note that L and R denote the leftmost and the rightmost indices of a and b respectively in the rhotrix. This is represented as:

$$R_n = \left(\begin{array}{cccccccc} & & & & a_1 & & & \\ & & & & a_2 & a_3 & a_4 & \\ & & & & a_5 & a_6 & a_7 & a_8 & a_9 \\ & & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ & & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_{\frac{1}{4}(n^2+3)-\frac{n-1}{2}} & \dots & \dots & \dots & \dots & a_{\frac{1}{4}(n^2+3)} & \dots & \dots & \dots & a_{\frac{1}{4}(n^2+3)+\frac{n-1}{2}} \\ & & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \\ & & & & a_{\frac{1}{2}(n^2+1)-8} & a_{\frac{1}{2}(n^2+1)-7} & a_{\frac{1}{2}(n^2+1)-6} & a_{\frac{1}{2}(n^2+1)-5} & a_{\frac{1}{2}(n^2+1)-4} & \\ & & & & a_{\frac{1}{2}(n^2+1)-3} & a_{\frac{1}{2}(n^2+1)-2} & a_{\frac{1}{2}(n^2+1)-1} & & & \\ & & & & & & a_{\frac{1}{2}(n^2+1)} & & & \end{array} \right)$$

$$Q_n = \left(\begin{array}{cccccccc}
& & & & b_1 & & & \\
& & & & b_2 & b_3 & b_4 & \\
& & & b_5 & b_6 & b_7 & b_8 & b_9 \\
& \dots & \dots & \dots & \dots & \dots & \dots & \dots \\
b_{\frac{1}{4}(n^2+3)-\frac{n-1}{2}} & \dots & \dots & \dots & \dots & b_{\frac{1}{4}(n^2+3)} & \dots & \dots & b_{\frac{1}{4}(n^2+3)+\frac{n-1}{2}} \\
& \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\
& & b_{\frac{1}{2}(n^2+1)-8} & b_{\frac{1}{2}(n^2+1)-7} & b_{\frac{1}{2}(n^2+1)-6} & b_{\frac{1}{2}(n^2+1)-5} & b_{\frac{1}{2}(n^2+1)-4} & \\
& & & b_{\frac{1}{2}(n^2+1)-3} & b_{\frac{1}{2}(n^2+1)-2} & b_{\frac{1}{2}(n^2+1)-1} & & \\
& & & & b_{\frac{1}{2}(n^2+1)} & & & \\
\end{array} \right) \quad (3.2.4)$$

We further simplify equation (3.2.4) as follow:

$$\text{Since } R = \frac{\frac{1}{2}(n^2+1)+1}{2} + \frac{n-1}{2} \text{ then } R = \frac{\frac{1}{2}(n^2+1)+1+n-1}{2} = \frac{\frac{1}{2}n^2 + \frac{1}{2} + n}{2} \text{ which implies that}$$

$$R = \frac{1}{4}n^2 + \frac{1}{4} + \frac{n}{2} = \frac{n^2 + 2n + 1}{4}$$

We also do the same for L

$$\text{Since } L = \frac{\frac{1}{2}(n^2+1)+1}{2} - \frac{n-1}{2} \text{ then } L = \frac{\frac{1}{2}(n^2+1)+1-n+1}{2} = \frac{\frac{1}{2}n^2 + \frac{1}{2} + 1 - n + 1}{2}$$

$$L = \frac{1}{4}n^2 + \frac{5}{4} - \frac{2n}{4} = \frac{n^2 - 2n + 5}{4}$$

$$\begin{aligned}
R_n &= \left\langle \begin{array}{cccccccc} & & & & a_1 & & & \\ & & & & a_2 & a_3 & a_4 & \\ & & a_5 & a_6 & a_7 & a_8 & a_9 & \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_{\frac{n^2-2n+5}{4}} & \dots & \dots & \dots & a_{\frac{1}{4}(n^2+3)} & \dots & \dots & a_{\frac{n^2+2n+1}{4}} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ & a_{\frac{1}{2}(n^2+1)-8} & a_{\frac{1}{2}(n^2+1)-7} & a_{\frac{1}{2}(n^2+1)-6} & a_{\frac{1}{2}(n^2+1)-5} & a_{\frac{1}{2}(n^2+1)-4} & & \\ & & a_{\frac{1}{2}(n^2+1)-3} & a_{\frac{1}{2}(n^2+1)-2} & a_{\frac{1}{2}(n^2+1)-1} & & & \\ & & & a_{\frac{1}{2}(n^2+1)} & & & & \end{array} \right\rangle \\
Q_n &= \left\langle \begin{array}{cccccccc} & & & & b_1 & & & \\ & & & & b_2 & b_3 & b_4 & \\ & & b_5 & b_6 & b_7 & b_8 & b_9 & \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ b_{\frac{n^2-2n+5}{4}} & \dots & \dots & \dots & b_{\frac{1}{4}(n^2+3)} & \dots & \dots & b_{\frac{n^2+2n+1}{4}} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ & b_{\frac{1}{2}(n^2+1)-8} & b_{\frac{1}{2}(n^2+1)-7} & b_{\frac{1}{2}(n^2+1)-6} & b_{\frac{1}{2}(n^2+1)-5} & b_{\frac{1}{2}(n^2+1)-4} & & \\ & & b_{\frac{1}{2}(n^2+1)-3} & b_{\frac{1}{2}(n^2+1)-2} & b_{\frac{1}{2}(n^2+1)-1} & & & \\ & & & b_{\frac{1}{2}(n^2+1)} & & & & \end{array} \right\rangle
\end{aligned}
\tag{3.2.5}$$

Substituting the values $n = 3$, $n = 5$ and $n = 7$ in equation (3.4.1) we have the following rhotrices:

$$R_3 = \left\langle \begin{array}{ccc} a_1 \\ a_2 & a_3 & a_4 \\ a_5 \end{array} \right\rangle, \left\langle \begin{array}{cccc} a_1 & & & \\ a_2 & a_3 & a_4 & \\ a_5 & a_6 & a_7 & a_8 & a_9 \\ a_{10} & a_{11} & a_{12} & \\ a_{13} \end{array} \right\rangle \text{ and } \left\langle \begin{array}{cccccc} a_1 & & & & & \\ a_2 & a_3 & a_4 & & & \\ a_5 & a_6 & a_7 & a_8 & a_9 & \\ a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ a_{17} & a_{18} & a_{19} & a_{20} & a_{21} & \\ a_{22} & a_{23} & a_{24} & & & \\ a_{25} \end{array} \right\rangle, \text{ respectively.}$$

Similarly, from (3.2.4), we can express the multiplication of any two heart-oriented rhotrices in the following way.

$$R_n \circ Q_n = \left(\begin{array}{cccccccc} & & & & a_1 & & & \\ & & & & a_2 & a_3 & a_4 & \\ & & & a_5 & a_6 & a_7 & a_8 & a_9 \\ & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ & \dots & \dots & \dots & \dots & a_{\frac{1}{4}(n^2+3)} & \dots & \dots & a_{\frac{1}{4}(n^2+3)+\frac{n-1}{2}} \\ & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ & & & a_{\frac{1}{2}(n^2+1)-8} & a_{\frac{1}{2}(n^2+1)-7} & a_{\frac{1}{2}(n^2+1)-6} & a_{\frac{1}{2}(n^2+1)-5} & a_{\frac{1}{2}(n^2+1)-4} \\ & & & a_{\frac{1}{2}(n^2+1)-3} & a_{\frac{1}{2}(n^2+1)-2} & a_{\frac{1}{2}(n^2+1)-1} & & \\ & & & & a_{\frac{1}{2}(n^2+1)} & & & \end{array} \right) \circ$$

$$\left(\begin{array}{cccccccc} & & & & b_1 & & & \\ & & & & b_2 & b_3 & b_4 & \\ & & & b_5 & b_6 & b_7 & b_8 & b_9 \\ & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ & \dots & \dots & \dots & b_{\frac{1}{4}(n^2+3)} & \dots & \dots & \dots & b_{\frac{1}{4}(n^2+3)+\frac{n-1}{2}} \\ & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ & & & b_{\frac{1}{2}(n^2+1)-8} & b_{\frac{1}{2}(n^2+1)-7} & b_{\frac{1}{2}(n^2+1)-6} & b_{\frac{1}{2}(n^2+1)-5} & b_{\frac{1}{2}(n^2+1)-4} \\ & & & b_{\frac{1}{2}(n^2+1)-3} & b_{\frac{1}{2}(n^2+1)-2} & b_{\frac{1}{2}(n^2+1)-1} & & \\ & & & & b_{\frac{1}{2}(n^2+1)} & & & \end{array} \right) =$$

$$R_n \circ Q_n = \left(\begin{array}{cccccccc} & & & & c_1 & & & \\ & & & & c_2 & c_3 & c_4 & \\ & & & c_5 & c_6 & c_7 & c_8 & c_9 \\ & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ & \dots & \dots & \dots & \dots & c_{\frac{1}{4}(n^2+3)} & \dots & \dots & c_{\frac{1}{4}(n^2+3)+\frac{n-1}{2}} \\ & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ & & & c_{\frac{1}{2}(n^2+1)-8} & c_{\frac{1}{2}(n^2+1)-7} & c_{\frac{1}{2}(n^2+1)-6} & c_{\frac{1}{2}(n^2+1)-5} & c_{\frac{1}{2}(n^2+1)-4} \\ & & & c_{\frac{1}{2}(n^2+1)-3} & c_{\frac{1}{2}(n^2+1)-2} & c_{\frac{1}{2}(n^2+1)-1} & & \\ & & & & c_{\frac{1}{2}(n^2+1)} & & & \end{array} \right)$$

$$\begin{aligned}
& \text{OR} \\
R_n \circ Q_n = & \left\langle \begin{array}{cccccccc} & & & & c_1 & & & \\ & & & & c_2 & c_3 & c_4 & \\ & & & c_5 & c_6 & c_7 & c_8 & c_9 \\ & & \dots & \dots & \dots & \dots & \dots & \dots \\ & c_{\frac{n^2-2n+5}{4}} & \dots & \dots & \dots & c_{\frac{1}{4}(n^2+3)} & \dots & \dots & c_{\frac{n^2+2n+1}{4}} \\ & & \dots & \dots & \dots & \dots & \dots & \dots & \\ & & & c_{\frac{1}{2}(n^2+1)-8} & c_{\frac{1}{2}(n^2+1)-7} & c_{\frac{1}{2}(n^2+1)-6} & c_{\frac{1}{2}(n^2+1)-5} & c_{\frac{1}{2}(n^2+1)-4} \\ & & & & c_{\frac{1}{2}(n^2+1)-3} & c_{\frac{1}{2}(n^2+1)-2} & c_{\frac{1}{2}(n^2+1)-1} & \\ & & & & & c_{\frac{1}{2}(n^2+1)} & & \end{array} \right\rangle \\
& (3.2.6)
\end{aligned}$$

where C_i is an expression in terms of a_i and b_i

Definition (heart-oriented multiplication)

$$\text{Let } R_5 = \left\langle \begin{array}{cccc} & a_1 & & \\ & a_2 & a_3 & a_4 \\ a_5 & a_6 & a_7 & a_8 & a_9 \\ & a_{10} & a_{11} & a_{12} \\ & & a_{13} & \end{array} \right\rangle \text{ and } Q_5 = \left\langle \begin{array}{cccc} & b_1 & & \\ & b_2 & b_3 & b_4 \\ b_5 & b_6 & b_7 & b_8 & b_9 \\ & b_{10} & b_{11} & b_{12} \\ & & b_{13} & \end{array} \right\rangle \text{ be two rhotrices of}$$

dimension 5 with entries from \mathbb{R} , the set of real numbers. We follow the multiplication (called *heart-oriented multiplication* in this thesis) which was first defined in (Ajibade, 2003) on rhotrices of third dimension as follows:

$$R_5 \circ Q_5 = \left\langle \begin{array}{cccc} & a_1 & & \\ & a_2 & a_3 & a_4 \\ a_5 & a_6 & a_7 & a_8 & a_9 \\ & a_{10} & a_{11} & a_{12} \\ & & a_{13} & \end{array} \right\rangle \circ \left\langle \begin{array}{cccc} & b_1 & & \\ & b_2 & b_3 & b_4 \\ b_5 & b_6 & b_7 & b_8 & b_9 \\ & b_{10} & b_{11} & b_{12} \\ & & b_{13} & \end{array} \right\rangle =$$

$$\begin{aligned}
& b_7 \circ \left\langle \begin{array}{cccc} & a_1 & & \\ & a_2 & a_3 & a_4 \\ a_5 & a_6 & a_7 & a_8 \\ & a_{10} & a_{11} & a_{12} \\ & & & a_{13} \end{array} \right\rangle + a_7 \circ \left\langle \begin{array}{cccc} & b_1 & & \\ & b_2 & b_3 & b_4 \\ b_5 & b_6 & b_7 & b_8 \\ & b_{10} & b_{11} & b_{12} \\ & & & b_{13} \end{array} \right\rangle \\
& = \left\langle \begin{array}{cccc} & a_1 b_7 + a_7 b_1 & & \\ & a_2 b_7 + a_7 b_2 & a_3 b_7 + a_7 b_3 & a_4 b_7 + a_7 b_4 \\ a_5 b_7 + a_7 b_5 & a_6 b_7 + a_7 b_6 & a_7 b_7 & a_8 b_7 + a_7 b_8 \\ & a_{10} b_7 + a_7 b_{10} & a_{11} b_7 + a_7 b_{11} & a_{12} b_7 + a_7 b_{12} \\ & & & a_{13} b_7 + a_7 b_{13} \end{array} \right\rangle \quad (3.2.7)
\end{aligned}$$

This can be expressed in the following way:

$$\begin{aligned}
c_1 &= a_1 b_7 + b_1 a_7 & c_2 &= a_2 b_7 + b_2 a_7 & c_3 &= a_3 b_7 + b_3 a_7 & c_4 &= a_4 b_7 + b_4 a_7 \\
c_5 &= a_5 b_7 + b_5 a_7 & c_6 &= a_6 b_7 + b_6 a_7 & c_7 &= b_7 \times a_7 & c_8 &= a_8 b_7 + b_8 a_7 \\
c_9 &= a_9 b_7 + b_9 a_7 & c_{10} &= a_{10} b_7 + b_{10} a_7 & c_{11} &= a_{11} b_7 + b_{11} a_7 & c_{12} &= a_{12} b_7 + b_{12} a_7 \\
c_{13} &= a_{13} b_7 + b_{13} a_7 & & & & & & \quad (3.2.8)
\end{aligned}$$

where a_7 and b_7 denote the hearts indices defined by $\frac{1}{4}[n^2 + 3]$ of the two rhotrices, the resulting value of c_7 is the product of the two hearts from the two rhotrices. We can now extend this to accommodate rhotrix of arbitrary dimension. Going by (3.2.8), we can represent the rhotrix's heart entry as $a_{\frac{1}{4}[n^2+3]}$ and $b_{\frac{1}{4}[n^2+3]}$, respectively. Equation (3.2.8) can be represented in rhotrix for as:

$$R_5 \circ Q_5 = C = \left\langle \begin{array}{cccc} & c_1 & & \\ & c_2 & c_3 & c_4 \\ c_5 & c_6 & c_7 & c_8 \\ & c_{10} & c_{11} & c_{12} \\ & & & c_{13} \end{array} \right\rangle \quad (3.2.9)$$

In general, given two rhotrices R_n and Q_n of dimension n , the entries of the heart-oriented product C_n of R_n and Q_n can be expressed as follows:

$$C_i = \lambda(b_{\frac{1}{4}[n^2+3]} a_i + a_{\frac{1}{4}[n^2+3]} b_i) + (1 - \lambda)(a_{\frac{1}{4}[n^2+3]} b_{\frac{1}{4}[n^2+3]}), \text{ for } i = 1, 2, \dots, \frac{1}{2}(n^2 + 1) \quad (3.2.10a)$$

or

$$C_i = b_{\frac{1}{4}[n^2+3]} \times a_i + a_{\frac{1}{4}[n^2+3]} \times b_i \cdot \lambda, \quad \lambda \in \{0,1\}$$

(3.2.10b)

where $\lambda = 0$ when the index value corresponds to that of the heart and $\lambda = 1$ where otherwise. This condition is further illustrated in Table 3.2a and 3.2b.

Alternatively, suppose that we now represent the n-dimensional rhotrix in equation (3.2.5) by $R_n = \langle a_i, a_h \rangle$ where of course a_i and a_h represents the a_i entries and its heart respectively, with $i = 1, 2, 3, \dots, |R_n|$ and $h = 3, 7, 13, \dots, \frac{1}{4}[n^2 + 3]$.

Consider an i dimensional rhotrix, having number of entries $|R_i| = \frac{1}{2}(i^2 + 1)$. Let \bar{i} be the mean of these entries computed as shown in Table 3.2a and Table 3.2b:

Table 3.2a Setting conditions for λ over a three dimensional rhotrix entries

i	\bar{i}	$i - \bar{i}$	$ i - \bar{i} $	
1	3	-2	2	$\lambda = 0$
2	3	-1	1	
3	3	0	0	$\lambda = 1$
4	3	1	1	$\lambda = 0$
5	3	2	2	

Table 3.2b Setting conditions for λ over a five dimensional rhotrix entries

i	\bar{i}	$i - \bar{i}$	$ i - \bar{i} $	
1	7	-6	6	
2	7	-5	5	
3	7	-4	4	

4	7	-3	3	$\lambda = 0$
5	7	-2	2	
6	7	-1	1	
7	7	0	0	$\lambda = 1$
8	7	1	1	$\lambda = 0$
9	7	2	2	
10	7	3	3	
11	7	4	4	
12	7	5	5	
13	7	6	6	

Let the heart entry of rhotrix Q_n denoted by b_h multiply all the entries of rhotrix R_n denoted by a_i and vice versa for the heart entry of the rhotrix R_n denoted by a_h . Subsequently, the two corresponding hearts of R_n and Q_n multiply each other as a single resulting product. Based on the derived value of λ , we can then define a function on λ as:

$$\lambda = \begin{cases} 0, & |i - \bar{i}| > 0 \\ 1, & |i - \bar{i}| = 0 \end{cases}$$

We define multiplication thus, of any two heart-oriented rhotrices of the same dimension as follow:

$$R_n \circ Q_n = b_h \langle a_i \rangle + a_h \langle b_i \rangle \circ (1 - \lambda)$$

It follows that,

$$C_i = \begin{cases} b_h \langle a_i \rangle & \text{for } i = h, \lambda = 1 \\ b_h \langle a_i \rangle + a_h \langle b_i \rangle & \text{for } i \neq h, \lambda = 0 \end{cases}$$

we can thus generalize this as.

$$C_i = b_h \langle a_i \rangle + a_h \langle b_i \rangle (1 - \lambda) \quad (3.2.11)$$

Where $\lambda = 1$ for $i = h$ and $\lambda = 0$ for $i \neq h$

It can also be verified further, though not discussed in this work that multiplication of any two rhotrices is commutative, associative, and distributive over addition. Hence we say that the set R of all rhotrices is a commutative algebra (Ajibade, 2003).

Let us consider the multiplication of the following pair of rhotrices.

$$R_5 \circ Q_5 = \left\langle \begin{array}{ccccc} & & 2 & & \\ & 3 & 1 & 4 & \\ 4 & 2 & 4 & 5 & 3 \\ & 1 & 5 & 4 & \\ & & 6 & & \end{array} \right\rangle \circ \left\langle \begin{array}{ccccc} & & 2 & & \\ & 4 & 5 & 2 & \\ 1 & 3 & 10 & 5 & 3 \\ & 2 & 1 & 2 & \\ & & 4 & & \end{array} \right\rangle$$

$$R_5 \circ Q_5 = R = \left\langle \begin{array}{cccccc} & & & & & 2*10+2*4 \\ & & & & & 3*10+4*4 & 1*10+5*4 & 4*10+2*4 \\ 4*10+1*4 & 2*10+3*4 & & 4*10 & 5*10+5*4 & 3*10+3*4 \\ & 1*10+2*4 & 5*10+1*4 & 4*10+2*4 & & \\ & & & & & 6*10+4*4 \end{array} \right\rangle$$

Substituting this computation into (3.2.8), we then have:

$$\begin{aligned} R_1 &= 2*10+2*4 & R_2 &= 3*10+4*4 & R_3 &= 1*10+5*4 & R_4 &= 4*10+2*4 \\ R_5 &= 4*10+1*4 & R_6 &= 2*10+3*4 & R_7 &= 4*10 & R_8 &= 5*10+5*4 \\ R_9 &= 3*10+3*4 & R_{10} &= 1*10+2*4 & R_{11} &= 5*10+1*4 & R_{12} &= 4*10+2*4 \\ R_{13} &= 6*10+4*4 \end{aligned}$$

$$R_5 \circ Q_5 = \left\langle \begin{array}{ccccc} & & 28 & & \\ & 46 & 30 & 48 & \\ 44 & 32 & 40 & 70 & 42 \\ & 18 & 54 & 48 & \\ & & 76 & & \end{array} \right\rangle$$

The obtained result differs with that of Sani's row-column method of rhotrices multiplication (Sani, 2004) and (Sani, 2007); this might be, of course, due to the simple facts that the approach given in (Ajibade, 2003) preferred to identify rhotrices as a unique mathematical algebraic objects different from the traditional matrix algebra. The two hearts-oriented multiplication algorithm for single and double array indexing is as shown below

Algorithm 1: Algorithm for Heart-oriented Multiplication of Rhotrices

Input:

$$h = \frac{1}{4}[n^2 + 3]$$

$a[0..k]$

$b[0..k]$

Output:

$c[0..k]$

for $i =: 1$ to k

if ($i = \frac{1}{4}[n^2 + 3]$)

$$c[i] \leftarrow a[i] * b[i];$$

else

$$c[i] \leftarrow b[h] \times a[i] + a[h] \times b[i];$$

endif

endfor

3.2.2. Row-Wise Heart-Oriented Rhotrix Multiplication

The row-wise rhotrix multiplication tends to put into consideration the position and direction of each entry in the cause of the multiplicative operations. The operation is performed in a row-wise direction indicated by the entry indices i and j . The index i indicate the row entry position while the index j indicates the row direction. The general representation of the row-wise rhotrix is as depicted in (3.2.12). It is important to note that division in this case are all integer division, since we are less concerned with the resulting decimal values.

Definition: The row of any given rhotrix is an array of entries running diagonally from the top-most left to the bottom rightmost direction of the rhotrix.

$$R_n = \left(\begin{array}{cccccccc} & & & & a_{11} & & & \\ & & & & a_{31} & a_{21} & a_{12} & \\ & & a_{5,1} & a_{4,1} & a_{32} & a_{22} & a_{13} & \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n,1} & \dots & \dots & \dots & a_{\frac{n+1}{2}, \lfloor \frac{n+3}{4} \rfloor} & \dots & \dots & a_{1, \frac{n+1}{2}} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ & a_{n, \frac{n+1}{2}-2} & a_{n-1, \frac{n+1}{2}-2} & a_{n-2, \frac{n+1}{2}-1} & a_{n-3, \frac{n+1}{2}-1} & a_{n-4, \frac{n+1}{2}} & & \\ & & a_{n, \frac{n+1}{2}-1} & a_{n-1, \frac{n+1}{2}-1} & a_{n-2, \frac{n+1}{2}} & & & \\ & & & a_{n, \frac{n+1}{2}} & & & & \end{array} \right) \quad (3.2.12)$$

We use commas in (3.2.12) to avoid any ambiguity with respect to the array indexes separation.

Further simplification of equation 3.2.12 gives:

$$R_n = \left(\begin{array}{cccccccc} & & & & a_{11} & & & \\ & & & & a_{31} & a_{21} & a_{12} & \\ & & a_{5,1} & a_{4,1} & a_{32} & a_{22} & a_{13} & \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n,1} & \dots & \dots & \dots & a_{\frac{n+1}{2}, \lfloor \frac{n+3}{4} \rfloor} & \dots & \dots & a_{1, \frac{n+1}{2}} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ & a_{n, \frac{n-3}{2}} & a_{n-1, \frac{n-3}{2}} & a_{n-2, \frac{n-1}{2}} & a_{n-3, \frac{n-1}{2}} & a_{n-4, \frac{n+1}{2}} & & \\ & & a_{n, \frac{n-1}{2}} & a_{n-1, \frac{n-1}{2}} & a_{n-2, \frac{n+1}{2}} & & & \\ & & & a_{n, \frac{n+1}{2}} & & & & \end{array} \right) \quad (3.2.13)$$

$$R_n \circ Q_n = \left(\begin{array}{cccccccc} & & & a_{11} & & & & \\ & & & a_{31} & a_{21} & a_{12} & & \\ & & a_{51} & a_{41} & a_{32} & a_{22} & a_{13} & \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n1} & \dots & \dots & \dots & a_{\frac{n+1}{2}, \lfloor \frac{n+3}{4} \rfloor} & \dots & \dots & \dots & a_{1, \frac{n+1}{2}} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n, \frac{n-3}{2}} & a_{n-1, \frac{n-3}{2}} & a_{n-2, \frac{n-1}{2}} & a_{n-3, \frac{n-1}{2}} & a_{n-4, \frac{n+1}{2}} & & & & \\ & a_{n, \frac{n-1}{2}} & a_{n-1, \frac{n-1}{2}} & a_{n-2, \frac{n+1}{2}} & & & & & \\ & & a_{n, \frac{n+1}{2}} & & & & & & \end{array} \right) \circ \left(\begin{array}{cccccccc} & & & b_{11} & & & & \\ & & & b_{31} & b_{21} & b_{12} & & \\ & & b_{51} & b_{41} & b_{32} & b_{22} & b_{13} & \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ b_{n1} & \dots & \dots & \dots & b_{\frac{n+1}{2}, \lfloor \frac{n+3}{4} \rfloor} & \dots & \dots & \dots & b_{1, \frac{n+1}{2}} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ b_{n, \frac{n-3}{2}} & b_{n-1, \frac{n-3}{2}} & b_{n-2, \frac{n-1}{2}} & b_{n-3, \frac{n-1}{2}} & b_{n-4, \frac{n+1}{2}} & & & & \\ & b_{n, \frac{n-1}{2}} & b_{n-1, \frac{n-1}{2}} & b_{n-2, \frac{n+1}{2}} & & & & & \\ & & b_{n, \frac{n+1}{2}} & & & & & & \end{array} \right) \quad (3.2.14)$$

$$C_n = \left(\begin{array}{cccccccc} & & & & c_{11} & & & \\ & & & c_{31} & c_{21} & c_{12} & & \\ & & c_{51} & c_{41} & c_{32} & c_{22} & c_{13} & \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ c_{n1} & \dots & \dots & \dots & c_{\frac{n+1}{2}, \lfloor \frac{n+3}{4} \rfloor} & \dots & \dots & \dots & c_{1, \frac{n+1}{2}} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ c_{n, \frac{n+1}{2}-2} & c_{n-1, \frac{n+1}{2}-2} & c_{n-2, \frac{n+1}{2}-1} & c_{n-3, \frac{n+1}{2}-1} & c_{n-4, \frac{n+1}{2}} & & & & \\ & c_{n, \frac{n+1}{2}-1} & c_{n-1, \frac{n+1}{2}-1} & c_{n-2, \frac{n+1}{2}} & & & & & \\ & & c_{n, \frac{n+1}{2}} & & & & & & \end{array} \right) \quad (3.2.15)$$

Definition: (Row-wise heart-oriented rothrix multiplication)

Let $a, b, c, \dots, c \in \mathfrak{R}$, $R_n \circ Q_n = a_{11} \times b_{\frac{n+1}{2}, \lfloor \frac{n+3}{4} \rfloor} + b_{11} \times a_{\frac{n+1}{2}, \lfloor \frac{n+3}{4} \rfloor} \cdot \cdot \cdot a_{n, \frac{n+1}{2}} \times b_{\frac{n+1}{2}, \lfloor \frac{n+3}{4} \rfloor} + b_{n, \frac{n+1}{2}} \times a_{\frac{n+1}{2}, \lfloor \frac{n+3}{4} \rfloor}$.

Where $a_{\frac{n+1}{2}, \lfloor \frac{n+3}{4} \rfloor}$ and $b_{\frac{n+1}{2}, \lfloor \frac{n+3}{4} \rfloor}$ are the hearts of the two rothrices. This is illustrated as in (3.2.16)

and (3.2.17).

$$R_5 \circ Q_5 = \left\langle \begin{array}{cccc} & & a_{11} & \\ & a_{31} & a_{21} & a_{12} \\ a_{51} & a_{41} & a_{32} & a_{22} & a_{13} \\ & a_{52} & a_{42} & a_{33} \\ & & a_{53} & \end{array} \right\rangle \circ \left\langle \begin{array}{cccc} & & b_{11} & \\ & b_{31} & b_{21} & b_{12} \\ b_{51} & b_{41} & b_{32} & b_{22} & b_{13} \\ & b_{52} & b_{42} & b_{33} \\ & & b_{53} & \end{array} \right\rangle \quad (3.2.16)$$

$$R_5 \circ Q_5 = C_5 = \left\langle \begin{array}{cccc} & & c_{11} & \\ & c_{31} & c_{21} & c_{12} \\ c_{51} & c_{41} & c_{32} & c_{22} & c_{13} \\ & c_{52} & c_{42} & c_{33} \\ & & c_{53} & \end{array} \right\rangle \quad (3.2.17)$$

Similarly, multiplication of the rhotrices in (3.2.16) is commutative. The product function and operation is similar to that of the well known vector dot product, the only differences is that the resultant product for the two rhotrices shown (3.2.17) is a rhotrix of the same order as the two multiplicands in (3.2.16).

The multiplication process of the two rhotrices in (3.2.16) above can thus be expressed as:

$$b_{32} \circ \left\langle \begin{array}{cccc} & & a_{11} & \\ & a_{31} & a_{21} & a_{12} \\ a_{51} & a_{41} & a_{32} & a_{22} & a_{13} \\ & a_{52} & a_{42} & a_{33} \\ & & a_{53} & \end{array} \right\rangle + a_{32} \circ \left\langle \begin{array}{cccc} & & b_{11} & \\ & b_{31} & b_{21} & b_{12} \\ b_{51} & b_{41} & b_{32} & b_{22} & b_{13} \\ & b_{52} & b_{42} & b_{33} \\ & & b_{53} & \end{array} \right\rangle =$$

$$\left\langle \begin{array}{cccc} & & a_{11}b_{32} + b_{11}a_{32} & \\ & a_{31}b_{32} + b_{31}a_{32} & a_{21}b_{32} + b_{21}a_{32} & a_{12}b_{32} + b_{12}a_{32} \\ a_{51}b_{32} + b_{51}a_{32} & a_{41}b_{32} + b_{41}a_{32} & a_{32}b_{32} & a_{22}b_{32} + b_{22}a_{32} & a_{13}b_{32} + b_{13}a_{32} \\ & a_{52}b_{32} + b_{52}a_{32} & a_{42}b_{32} + b_{42}a_{32} & a_{33}b_{32} + b_{33}a_{32} \\ & & a_{53}b_{32} + b_{53}a_{32} & \end{array} \right\rangle \quad (3.2.18)$$

$$\begin{aligned}
c_{11} &= a_{11}b_{32} + b_{11}a_{32} & c_{12} &= a_{12}b_{32} + b_{12}a_{32} & c_{13} &= a_{13}b_{32} + b_{13}a_{32} & c_{21} &= a_{21}b_{32} + b_{21}a_{32} \\
c_{22} &= a_{22}b_{32} + b_{22}a_{32} & c_{31} &= a_{31}b_{32} + b_{31}a_{32} & c_{32} &= a_{32}b_{32} & c_{33} &= a_{33}b_{32} + b_{33}a_{32} \\
c_{41} &= a_{41}b_{32} + b_{41}a_{32} & c_{42} &= a_{42}b_{32} + b_{42}a_{32} & c_{51} &= a_{51}b_{32} + b_{51}a_{32} & c_{52} &= a_{52}b_{32} + b_{52}a_{32} \\
c_{53} &= a_{53}b_{32} + b_{53}a_{32}
\end{aligned} \tag{3.2.19}$$

Where a_{32} and b_{32} denote the hearts of the two rhotrices whose indices is defined by $\frac{n+1}{2}, \left\lfloor \frac{n+3}{4} \right\rfloor$, the resulting value of c_{32} is the product of the two hearts from the two rhotrices. We can equally extend this to accommodate for rhotrix of n-dimension. Then going by equation (3.2.12), we can represent the row-wise rhotrix hearts as $a_{\frac{n+1}{2}, \left\lfloor \frac{n+3}{4} \right\rfloor}$ and $b_{\frac{n+1}{2}, \left\lfloor \frac{n+3}{4} \right\rfloor}$, then, we have:

$$c_{i,j} = b_{\frac{n+1}{2}, \left\lfloor \frac{n+3}{4} \right\rfloor} * a_{i,j} + a_{\frac{n+1}{2}, \left\lfloor \frac{n+3}{4} \right\rfloor} * b_{i,j} (1 - \lambda) \tag{3.2.20}$$

Where λ denotes a constant value that lies between 1 and 0, such that if the indices of $a_{i,j}$ and $b_{i,j}$ takes the hearts position, then $\lambda = 0$ and $\lambda = 1$ otherwise.

Algorithm 2: Heart-oriented Row-wise rhotrix multiplication

Input:

$$p \leftarrow \frac{n+1}{2}$$

$$q \leftarrow \left\lfloor \frac{n+3}{4} \right\rfloor$$

$a[0 \dots \text{row_upperbound}, 0 \dots \text{column_upperbound}]$

$b[0 \dots \text{row_upperbound}, 0 \dots \text{column_upperbound}]$

Output:

$c[0 \dots \text{row_upperbound}, 0 \dots \text{column_upperbound}]$

```

For i ← 0 to row_upperbound
For j ← 0 to column_upperbound
{
    if (i == p && j == q){
        c[i,j] ← a[i,j] * b[i,j];
    }
    else {
        c[i,j] ← a[i][j] * b[p][q] + b[i,j] * a[p,q];
    }
}
Endfor

```

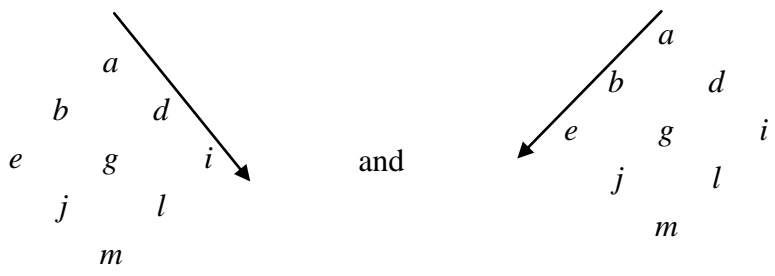
We have presented an extended heart-oriented rhotrix multiplication method, derivation and algorithm for higher order rhotrices multiplication. The method presented is quite unique and directed only to rhotrix multiplication of the form discussed in (Ajibade, 2003). In particular we have shown that a generalized equation as in our case could be used to implement n-dimensional rhotrices multiplication as against the row-column multiplication method presented in (Sani, 2004).

3.3. Row-Column Multiplication of N-Dimensional Rhotrices

In this section, a generalized formula for an alternative row-column multiplication method for rhotrices is presented. The concept established in this section follows the basic rudiment of mathematical axioms that prove the existing relationships between rhotrices and matrices. This is an extension of the same work presented on rhotrices of order three.

An alternative row-column multiplication method for rhotrices was suggested in (Sani, 2004). His work considered a comparative relationship on some common properties that relate and link rhotrices multiplication to that of the well known row-column matrices multiplication. In this section we present a generalized expression representing the row-column multiplication of some objects which are in some ways between $n \times n$ dimensional matrices and $(2n-1) \times (2n-1)$ dimensional matrices for the purpose of this work. Rows and columns definition of rhotrix was initially given in (Sani, 2004), and similarly defined in the following order of arrangements.

$$R_5 = \left\langle \begin{array}{ccccc} & & a & & \\ & b & c & d & \\ e & f & g & h & i \\ & j & k & l & \\ & & m & & \end{array} \right\rangle$$



respectively.

3.3.1. Row-Column Definition and Formula Expression for N-Dimensional Rhotrix

In an attempt to answer the question of “devising a suitable method of solving rhotrices multiplication without having to first perform any form of transformation to coupled matrix and vice-versa” posed in (Sani, 2007), an alternative method for multiplication of n-dimensional rhotrices is therefore presented (Ezugwu *et al*, 2011b), and thus recorded as follows:

$$R_n \circ Q_n = \left\langle \begin{array}{cccccc} & & & a_{11} & & \\ & & & a_{31} & a_{21} & a_{12} \\ & & & a_{51} & a_{41} & a_{32} & a_{22} & a_{13} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n1} & \dots & \dots & \dots & \dots & \dots & \dots & \dots & a_{1t} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ & & & a_{nt-2} & a_{n-1t-2} & a_{n-2t-1} & a_{n-3t-1} & a_{n-4t} \\ & & & a_{nt-1} & a_{n-1t-1} & a_{n-2t} \\ & & & a_{nt} & & & & \end{array} \right\rangle \circ \left\langle \begin{array}{cccccc} & & & b_{11} & & \\ & & & b_{31} & b_{21} & b_{12} \\ & & & b_{51} & b_{41} & b_{32} & b_{22} & b_{13} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ b_{n1} & \dots & \dots & \dots & \dots & \dots & \dots & \dots & b_{1t} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ & & & b_{nt-2} & b_{n-1t-2} & b_{n-2t-1} & b_{n-3t-1} & b_{n-4t} \\ & & & b_{nt-1} & b_{n-1t-1} & b_{n-2t} \\ & & & b_{nt} & & & & \end{array} \right\rangle \quad (3.3.1a)$$

The subscript n in equation (3.3.1a) denotes the dimension of the rhotrix, where $t = (n+1)/2$ and t is the column order of rhotrices R_n and Q_n respectively, so that if $n=3$, then $t=(3+1)/2 = 2$. If $n=5$, $t=3$ and if $n=9$, $t=5$. For example consider a five dimensional rhotrix, this is represented as follows:

$$\left\langle \begin{array}{cccc} & & a_{11} & \\ & a_{31} & a_{21} & a_{12} \\ a_{51} & a_{41} & a_{32} & a_{22} & a_{13} \\ & a_{52} & a_{42} & a_{33} \\ & & a_{53} & \end{array} \right\rangle \quad (3.3.1b)$$

If we view a rhotrix as having odd rows as the main entries and even rows as the heart entries all positioned side by side along each other, then we can generalize, by defining the rows and columns of rhotrix main entries as:

$$\left\langle \begin{array}{cccc} a_{n1} & a_{n-11} & \dots & a_{11} \\ & a_{n2} & a_{n-12} & \dots & a_{12} \\ & & a_{n3} & a_{n-13} & \dots & a_{13} \\ & & & \dots & \dots & \dots & \dots \\ & & & & \dots & \dots & \dots & \dots \\ & & & & & a_{nt} & a_{n-1t} & \dots & a_{1n} \end{array} \right\rangle \text{ and } \left\langle \begin{array}{cccc} & & & b_{11} & b_{13} & \dots & b_{1t} \\ & & & b_{31} & b_{32} & \dots & b_{3t} \\ & & & b_{51} & b_{52} & \dots & b_{5t} \\ & & & \dots & \dots & \dots & \dots \\ & & & \dots & \dots & \dots & \dots \\ & & & \dots & \dots & \dots & \dots \\ b_{n1} & b_{n2} & \dots & b_{nt} \end{array} \right\rangle$$

respectively and similarly for the hearts entries.

Suppose we now represent the n-dimensional rhotrices in equation (3.3.1a) by $R_n = \langle a_{ik} \rangle$ and $Q_n = \langle b_{kj} \rangle$ where a_{ik} and b_{kj} represent the a_{ik} and b_{kj} elements, respectively. The multiplication will then be as follow:

$$R_n \circ Q_n = \langle a_{i,j} \circ b_{i,j} \rangle = \lambda \sum_{k=1}^t (a_{i,k} b_{2k-1,j}) + (1-\lambda) \sum_{k=1}^{t-1} (a_{i,k} b_{2k,j}) \quad (3.3.2)$$

Where:

$$n = 3,5,7,\dots,(2m+1), \text{ where } m \in \mathbb{Z}^+$$

$$t = (n+1)/2$$

$$i, j = 1,2,3,\dots,n \text{ and}$$

$$k = 1,2,3,\dots,t, \text{ for } i \bmod 2 = 1: \lambda = 1$$

$$k = 2,4,6,\dots,t-1, \text{ for } i \bmod 2 = 0: \lambda = 0$$

Hence the row-column rhotrix multiplication method is similar to matrix multiplication. However, in this case odd rows of a_{ij} multiply odd columns of b_{ij} , while even rows of a_{ij} will then multiply even columns of b_{ij} .

Justification of adopted method: from multiset point of view

A multiset representation argument:

$$\left\langle \begin{array}{cccc} & & a_{11} & & \\ & a_{21} & c_{11} & a_{12} & \\ a_{31} & c_{21} & a_{22} & c_{12} & a_{13} \\ & a_{32} & c_{22} & a_{23} & \\ & & a_{33} & & \end{array} \right\rangle$$

The above rhotrix representation was presented by Sani (2007) and we believe is a suitable choice for him. But consider its multiset permutation

$$v = \left[[a_{11}, a_{12}, a_{13}], [c_{11}, c_{12}], [a_{21}, a_{22}, a_{23}], [c_{21}, c_{22}], [a_{31}, a_{32}, a_{33}] \right]$$

The letter and indices do not tell anything of the position of the element-multiset V.

Sincerely speaking, this method of rhotrix representation seems to point out two matrices. It is as good as working with two separate matrices after which results are combined, and makes less sense of the idea of a rhotrix as an entity.

Consider the multiset permutation

$$B = \{ [1,0,0], [1,0], [0,1,0], [0,1], [0,0,1] \}$$

B is a basis for all rhotrices of the fifth dimension, in the rhotrix vector space R^5 , where in general we write the multiset permutation of an arbitrary rhotrix of the fifth dimension given in (3.3.1b) as

$$\begin{aligned} v' &= \left[[a_{11}, a_{12}, a_{13}], [a_{21}, a_{22}], [a_{31}, a_{32}, a_{33}], [a_{41}, a_{42}], [a_{51}, a_{52}, a_{53}] \right] \\ &= \left[[a_{ij}], i = 1,2,3, j = 1, \dots, \frac{1}{2} [5 - (-1)^i] \right] \end{aligned}$$

Note that V' is an element of the rhotrix vector space R^5 of the fifth dimension.

$$\begin{aligned}
C_{11} &= a_{11} b_{11} + a_{12} b_{31} + a_{13} b_{51} + a_{14} b_{71} + a_{15} b_{91} \\
C_{12} &= a_{11} b_{12} + a_{12} b_{32} + a_{13} b_{52} + a_{14} b_{72} + a_{15} b_{92} \\
C_{13} &= a_{11} b_{13} + a_{12} b_{33} + a_{13} b_{53} + a_{14} b_{73} + a_{15} b_{93} \\
C_{14} &= a_{11} b_{14} + a_{12} b_{34} + a_{13} b_{54} + a_{14} b_{74} + a_{15} b_{94} \\
C_{15} &= a_{11} b_{15} + a_{12} b_{35} + a_{13} b_{55} + a_{14} b_{75} + a_{15} b_{95}
\end{aligned}
\left. \vphantom{\begin{aligned} C_{11} \\ C_{12} \\ C_{13} \\ C_{14} \\ C_{15} \end{aligned}} \right\} i = 1, j, k = 1, 2, 3, 4, 5 \text{ where } i \text{ is odd}$$

$$\begin{aligned}
C_{31} &= a_{31} b_{11} + a_{32} b_{31} + a_{33} b_{51} + a_{34} b_{71} + a_{35} b_{91} \\
C_{32} &= a_{31} b_{12} + a_{32} b_{32} + a_{33} b_{52} + a_{34} b_{72} + a_{35} b_{92} \\
C_{33} &= a_{31} b_{13} + a_{32} b_{33} + a_{33} b_{53} + a_{34} b_{73} + a_{35} b_{93} \\
C_{34} &= a_{31} b_{14} + a_{32} b_{34} + a_{33} b_{54} + a_{34} b_{74} + a_{35} b_{94} \\
C_{35} &= a_{31} b_{15} + a_{32} b_{35} + a_{33} b_{55} + a_{34} b_{75} + a_{35} b_{95}
\end{aligned}
\left. \vphantom{\begin{aligned} C_{31} \\ C_{32} \\ C_{33} \\ C_{34} \\ C_{35} \end{aligned}} \right\} i = 3, j, k = 1, 2, 3, 4, 5 \text{ where } i \text{ is odd}$$

$$\begin{aligned}
C_{51} &= a_{51} b_{11} + a_{52} b_{31} + a_{53} b_{51} + a_{54} b_{71} + a_{55} b_{91} \\
C_{52} &= a_{51} b_{12} + a_{52} b_{32} + a_{53} b_{53} + a_{54} b_{72} + a_{55} b_{92} \\
C_{53} &= a_{51} b_{13} + a_{52} b_{33} + a_{53} b_{53} + a_{54} b_{73} + a_{55} b_{93} \\
C_{54} &= a_{51} b_{14} + a_{52} b_{34} + a_{53} b_{54} + a_{54} b_{74} + a_{55} b_{94} \\
C_{55} &= a_{51} b_{15} + a_{52} b_{35} + a_{53} b_{55} + a_{54} b_{75} + a_{55} b_{95}
\end{aligned}
\left. \vphantom{\begin{aligned} C_{51} \\ C_{52} \\ C_{53} \\ C_{54} \\ C_{55} \end{aligned}} \right\} i = 5, j, k = 1, 2, 3, 4, 5 \text{ where } i \text{ is odd}$$

$$\begin{aligned}
C_{71} &= a_{71} b_{11} + a_{72} b_{31} + a_{73} b_{51} + a_{74} b_{71} + a_{75} b_{91} \\
C_{72} &= a_{71} b_{12} + a_{72} b_{32} + a_{73} b_{52} + a_{74} b_{72} + a_{75} b_{92} \\
C_{73} &= a_{71} b_{13} + a_{72} b_{33} + a_{73} b_{53} + a_{73} b_{72} + a_{75} b_{93} \\
C_{74} &= a_{71} b_{14} + a_{72} b_{34} + a_{73} b_{54} + a_{74} b_{74} + a_{75} b_{94} \\
C_{75} &= a_{71} b_{15} + a_{72} b_{35} + a_{73} b_{55} + a_{74} b_{75} + a_{75} b_{95}
\end{aligned}
\left. \vphantom{\begin{aligned} C_{71} \\ C_{72} \\ C_{73} \\ C_{74} \\ C_{75} \end{aligned}} \right\} i = 7, j, k = 1, 2, 3, 4, 5 \text{ where } i \text{ is odd}$$

$$\begin{aligned}
C_{91} &= a_{91} b_{11} + a_{92} b_{31} + a_{93} b_{51} + a_{94} b_{71} + a_{95} b_{91} \\
C_{92} &= a_{91} b_{12} + a_{92} b_{32} + a_{93} b_{52} + a_{94} b_{72} + a_{95} b_{92} \\
C_{93} &= a_{91} b_{13} + a_{92} b_{33} + a_{93} b_{53} + a_{94} b_{73} + a_{95} b_{93} \\
C_{94} &= a_{91} b_{14} + a_{92} b_{34} + a_{93} b_{54} + a_{94} b_{74} + a_{95} b_{94} \\
C_{95} &= a_{91} b_{15} + a_{92} b_{35} + a_{93} b_{55} + a_{94} b_{75} + a_{95} b_{95}
\end{aligned}
\left. \vphantom{\begin{aligned} C_{91} \\ C_{92} \\ C_{93} \\ C_{94} \\ C_{95} \end{aligned}} \right\} i = 9, j, k = 1, 2, 3, 4, 5 \text{ where } i \text{ is odd}$$

$$\begin{aligned}
C_{21} &= a_{21} b_{21} + a_{22} b_{41} + a_{23} b_{61} + a_{24} b_{81} \\
C_{22} &= a_{21} b_{22} + a_{22} b_{42} + a_{23} b_{62} + a_{24} b_{82} \\
C_{23} &= a_{21} b_{23} + a_{22} b_{43} + a_{23} b_{63} + a_{24} b_{83} \\
C_{24} &= a_{21} b_{24} + a_{22} b_{44} + a_{23} b_{64} + a_{24} b_{84}
\end{aligned}
\left. \vphantom{\begin{aligned} C_{21} \\ C_{22} \\ C_{23} \\ C_{24} \end{aligned}} \right\} i = 2, j, k = 1, 2, 3, 4, \text{ where } i \text{ is even}$$

$$\begin{aligned}
C_{41} &= a_{41} b_{21} + a_{42} b_{41} + a_{43} b_{61} + a_{44} b_{81} \\
C_{42} &= a_{41} b_{22} + a_{42} b_{42} + a_{43} b_{62} + a_{44} b_{82} \\
C_{43} &= a_{41} b_{23} + a_{42} b_{43} + a_{43} b_{63} + a_{44} b_{83} \\
C_{44} &= a_{41} b_{44} + a_{42} b_{44} + a_{43} b_{64} + a_{44} b_{84}
\end{aligned}
\left. \vphantom{\begin{aligned} C_{41} \\ C_{42} \\ C_{43} \\ C_{44} \end{aligned}} \right\} i = 4, j, k = 1, 2, 3, 4, \text{ where } i \text{ is even}$$

$$\begin{aligned}
C_{61} &= a_{61} b_{21} + a_{62} b_{41} + a_{63} b_{61} + a_{64} b_{81} \\
C_{62} &= a_{61} b_{22} + a_{62} b_{42} + a_{63} b_{62} + a_{64} b_{82} \\
C_{63} &= a_{61} b_{23} + a_{62} b_{43} + a_{63} b_{63} + a_{64} b_{83} \\
C_{64} &= a_{61} b_{44} + a_{62} b_{44} + a_{63} b_{64} + a_{64} b_{84}
\end{aligned}
\left. \vphantom{\begin{aligned} C_{61} \\ C_{62} \\ C_{63} \\ C_{64} \end{aligned}} \right\} i = 6, j, k = 1, 2, 3, 4, \text{ where } i \text{ is even}$$

$$\begin{aligned}
C_{81} &= a_{81} b_{21} + a_{82} b_{41} + a_{83} b_{61} + a_{84} b_{81} \\
C_{82} &= a_{81} b_{22} + a_{82} b_{42} + a_{83} b_{62} + a_{84} b_{82} \\
C_{83} &= a_{81} b_{23} + a_{82} b_{43} + a_{83} b_{63} + a_{84} b_{83} \\
C_{84} &= a_{81} b_{44} + a_{82} b_{44} + a_{83} b_{64} + a_{84} b_{84}
\end{aligned}
\left. \vphantom{\begin{aligned} C_{81} \\ C_{82} \\ C_{83} \\ C_{84} \end{aligned}} \right\} i = 8, j, k = 1, 2, 3, 4, \text{ where } i \text{ is even}$$

The resulting multiplicand of the two rhotrices, $R_9 \circ Q_9$ is also given in the form shown below:

$$\text{Let } R_9 \circ Q_9 = C_9 = \left(\begin{array}{cccccccc} & & & & C_{11} & & & & \\ & & & & C_{31} & C_{21} & C_{12} & & \\ & & & & C_{51} & C_{41} & C_{32} & C_{22} & C_{13} \\ C_{71} & C_{61} & C_{52} & C_{42} & C_{33} & C_{23} & C_{14} & & \\ C_{91} & C_{81} & C_{72} & C_{62} & C_{53} & C_{43} & C_{34} & C_{24} & C_{15} \\ & & & & C_{92} & C_{82} & C_{73} & C_{63} & C_{54} & C_{44} & C_{35} \\ & & & & C_{93} & C_{83} & C_{74} & C_{64} & C_{55} & & \\ & & & & C_{94} & C_{84} & C_{75} & & & & \\ & & & & & & & & & & C_{95} \end{array} \right)$$

By comparing entries, we have the following resultant equations:

Those entries for which i assume odd values are calculated as:

$$C_{11} = \sum_{K=1}^5 a_{1,K} b_{2k-1,1}, C_{12} = \sum_{K=1}^5 a_{1,K} b_{2k-1,2}, C_{13} = \sum_{K=1}^5 a_{1,K} b_{2k-1,3}, C_{14} = \sum_{K=1}^5 a_{1,K} b_{2k-1,4}, C_{15} = \sum_{K=1}^5 a_{1,K} b_{2k-1,5}$$

$$C_{31} = \sum_{K=1}^5 a_{3,K} b_{2k-1,1}, C_{32} = \sum_{K=1}^5 a_{3,K} b_{2k-1,2}, C_{33} = \sum_{K=1}^5 a_{3,K} b_{2k-1,3}, C_{34} = \sum_{K=1}^5 a_{3,K} b_{2k-1,4}, C_{35} = \sum_{K=1}^5 a_{3,K} b_{2k-1,5}$$

$$C_{51} = \sum_{K=1}^5 a_{5,K} b_{2k-1,1}, C_{52} = \sum_{K=1}^5 a_{5,K} b_{2k-1,2}, C_{53} = \sum_{K=1}^5 a_{5,K} b_{2k-1,3}, C_{54} = \sum_{K=1}^5 a_{5,K} b_{2k-1,4}, C_{55} = \sum_{K=1}^5 a_{5,K} b_{2k-1,5}$$

etc., while those entries for which i takes even values are similarly calculated as:

$$C_{21} = \sum_{K=1}^4 a_{2,K} b_{2k,1}, C_{22} = \sum_{K=1}^4 a_{2,K} b_{2k,2}, C_{23} = \sum_{K=1}^4 a_{2,K} b_{2k,3}, C_{24} = \sum_{K=1}^4 a_{2,K} b_{2k,4}$$

$$C_{41} = \sum_{K=1}^4 a_{4,K} b_{2k,1}, C_{42} = \sum_{K=1}^4 a_{4,K} b_{2k,2}, C_{43} = \sum_{K=1}^4 a_{4,K} b_{2k,3}, C_{44} = \sum_{K=1}^4 a_{4,K} b_{2k,4} \text{ etc.}$$

Algorithm 3: Row column rhotrix multiplication

for $i \leftarrow 1$ to n

 if $i \bmod 2 \neq 0$

 col: upperbound $\leftarrow (n+1) / 2$

```

    β ← 1
    for j ← 1 to upperbound
    for k ← 1 to upperbound
c[i,j] ← c[i,j] + β*a[i,k] * b[2k-1,j] + (1- β) *a[i,k]*b[2k,j]
    endfor
endfor
}
else{
    col: upperbound ← (n-1) / 2
    β ← 0
    for j ← 1 to upperbound
    for k ← 1 to upperbound
c[i,j] ← c[i,j] + β*a[i,k] * b[2k-1,j] + (1- β) *a[i,k]*b[2k,j]
    endfor
endfor
}
endfor

```

3.3.3. Generalized Form of Row-Column Multiplication Technique

Splitting equation (3.3.2), the summations are as follows:

$$C_{i,j} = \sum_{k=1}^t a_{i,k} b_{2k-1,j} \quad \text{Entries for the odd matrix of } R. \quad (3.3.4)$$

$$C_{i,j} = \sum_{k=1}^{t-1} a_{i,k} b_{2k,j} \quad \text{Entries for the even matrix of } R. \quad (3.3.5)$$

The differences between these two summations are reflected in the two upper bounds (i.e. t , and $t-1$), and for index b (i.e. $2k-1$ and $2k-0$). We want to devise a generalized formula to handle these two sets of entries. Consider the function:

$$f(x) = \begin{cases} 1, & \text{if } x \text{ is odd} \\ 0, & \text{if } x \text{ is even} \end{cases}, \text{ defined by } f(x) = \frac{1}{2}[(-1)^{x+1} + 1], \quad x = 1, 2, \dots, n$$

Replacing 1 and 0 with $f(i)$ in the index of b in equations (3.3.4) and (3.3.5) respectively, we have:

$$C_{i,j} = \sum_{k=1}^t a_{i,k} b_{2k - \frac{1}{2}[(-1)^{i+1} + 1], j} \quad \text{Entries for the order 3 rhotrix of } R_{i,j}. \quad (3.3.6)$$

$$C_{i,j} = \sum_{k=1}^{t-1} a_{i,k} b_{2k - \frac{1}{2}[(-1)^{i+1} + 1], j} \quad \text{Entries for the order 2 rhotrix of } R_{i,j}. \quad (3.3.7)$$

Similarly consider these two functions:

$$g(x) = \begin{cases} 3, & \text{if } x \text{ is odd} \\ 2, & \text{if } x \text{ is even} \end{cases} \quad \text{defined by } g(x) = \frac{1}{2}[5 - (-1)^x], \quad x = 1, 2, \dots, n$$

$$g(x) = \begin{cases} 5, & \text{if } x \text{ is odd} \\ 4, & \text{if } x \text{ is even} \end{cases} \quad \text{defined by } g(x) = \frac{1}{2}[9 - (-1)^x], \quad x = 1, 2, \dots, n$$

Replacing only the differences in equations (3.3.6) and (3.3.7), for the upper bounds 3 and 2, and 5 and 4 respectively we obtain the equations:

For $n = 5$

$$\langle C_{ij} \rangle = \sum_{k=1}^{\frac{1}{2}[5 - (-1)^i]} a_{i,k} b_{2k - \frac{1}{2}[(-1)^{i+1} + 1], j} \quad i = 1, 2, \dots, 5 \quad j, k = 1, 2, \dots, \frac{1}{2}[5 - (-1)^i] \quad (3.3.8)$$

For $n = 9$

$$\langle C_{ij} \rangle = \sum_{k=1}^{\frac{1}{2}[9 - (-1)^i]} a_{i,k} b_{2k - \frac{1}{2}[(-1)^{i+1} + 1], j} \quad i = 1, 2, \dots, 9 \quad j, k = 1, 2, \dots, \frac{1}{2}[9 - (-1)^i] \quad (3.3.9)$$

This approach can further be extended to n -dimensional rhotrices, finally let us consider the function

$$h(x) = \begin{cases} t, & \text{if } x \text{ is odd} \\ t-1, & \text{if } x \text{ is even} \end{cases} \quad \text{defined by } h(x) = \frac{1}{2}[2t - 1 - (-1)^x], \quad x = 1, 2, 3, \dots, n$$

Hence from equation (3.3.1), we know that $t = (n+1)/2$, therefore, $n = 2t-1$, by substituting t and $t-1$ for the function $h(i)$, we obtain this equation.

$$\langle C_{ij} \rangle = \sum_{k=1}^{\frac{1}{2}[n-(-1)^i]} a_{i,k} b_{2k-\frac{1}{2}[(-1)^{i+1}+1],j} \quad i = 1, 2, \dots, n \quad j, k = 1, 2, \dots, \frac{1}{2}[n-(-1)^i] \quad (3.3.10)$$

Equation (3.3.10) can further be expressed as:

Let $p = \frac{1}{2}[n-(-1)^i]$ and by substituting the value of p into equation (3.3.10), we have

$$\langle C_{ij} \rangle = \sum_{k=1}^p a_{i,k} b_{2k-\frac{1}{2}[(-1)^{i+1}+1],j} \quad i = 1, 2, \dots, n \quad j, k = 1, 2, \dots, p \quad (3.3.11)$$

Algorithm 4: Modified Row-Column Sequential Algorithm

for $i \leftarrow 1$ to n

$P \leftarrow \frac{1}{2}((-1)^{i+1}+1)$

for $j \leftarrow 1$ to t

for $k \leftarrow 1$ to t

$c[i,j] \leftarrow c[i,j] + a[i,k]*b[2k-p,j]$

endfor

endfor

endfor

We have shown a more simplified method of rhotrix row-column multiplication by taking a holistic approach in deriving a new form of a generalized expression, the approach used in this case does not require any form of rhotrix transformation to either matrix or coupled matrix before any possible multiplication could be achieved. The approach and results analyzed shows that the equations (3.10) and (1.19) can be used to solve rhotrices of any dimension and yet give the same result equivalent to the initial method proposed (Sani, 2004).

3.4. Application of Cannon's Algorithm in Rhotrix Row-Column Multiplication

In previous chapter, we presented a summary review of Cannon's algorithm. in similar issue, we also mentioned that the algorithm uses a mesh of processors with wraparound connections (a

$a_{11,1}$	$a_{11,2}$	$a_{11,3}$	$a_{11,4}$	$a_{11,5}$	$a_{11,6}$	$a_{11,7}$	$a_{11,8}$
$a_{12,1}$	$a_{12,2}$	$a_{12,3}$	$a_{12,4}$	$a_{12,5}$	$a_{12,6}$	$a_{12,7}$	0
$a_{13,1}$	$a_{13,2}$	$a_{13,3}$	$a_{13,4}$	$a_{13,5}$	$a_{13,6}$	$a_{13,7}$	$a_{13,8}$
$a_{14,1}$	$a_{14,2}$	$a_{14,3}$	$a_{14,4}$	$a_{14,5}$	$a_{14,6}$	$a_{14,7}$	0
$a_{15,1}$	$a_{15,2}$	$a_{15,3}$	$a_{15,4}$	$a_{15,5}$	$a_{15,6}$	$a_{15,7}$	$a_{15,8}$
0	0	0	0	0	0	0	0

Table 3.4b $A_{i,j}$ represents $2^{n-1} \times 2^{n-1}$ matrix of R_{15} entries

A0,0	A0,1	A0,2	A0,3
A1,0	A1,1	A1,2	A1,3
A2,0	A2,1	A2,2	A2,3
A3,0	A3,1	A3,2	A3,3
A4,0	A4,1	A4,2	A4,3
A5,0	A5,1	A5,2	A5,3
A6,0	A6,1	A6,2	A6,3
A7,0	A7,1	A7,2	A7,3

The movement pattern of data elements in Cannon’s algorithm is as illustrated in Figure 3.4a. All the shifts are with wraparound. For clarity, we will refer to elements of the arrays A and B , though submatrices would normally be used. The algorithm can be described by the following steps:

- i. Initially processor $P_{i,j}$ has elements $a_{i,j}$ and $b_{i,j}$ ($0 \leq i < n, 0 \leq j < n$).
- ii. Elements are moved from their initial position to an “aligned” position. The complete i^{th} row of A is shifted i places left, and the complete j^{th} column of B is shifted j places upward. This has the effect of placing the element $a_{i,j+i}$ and the element $b_{i+j,j}$ in processor

$P_{i,j}$, as illustrated in Figure 3.4b. these elements are a pair of those required in the accumulation of $c_{i,j}$.

- iii. Each processor, $P_{i,j}$, multiplies its elements.
- iv. The i^{th} row of A is shifted one place left, and the j^{th} column of B is shifted one place upward. This has the effect of bringing together the adjacent elements of A and B , which will also be required in the accumulation, as illustrated in Figure 3.4c.
- v. Each processor, $P_{i,j}$, multiplies the elements brought to it and adds the results to the accumulating sum.
- vi. Steps *iv* and *v* are represented until the final results obtained ($n-1$ shifts with n rows and n columns of elements).

The advantage of this algorithm is that, additional storage for partial results is unnecessary.

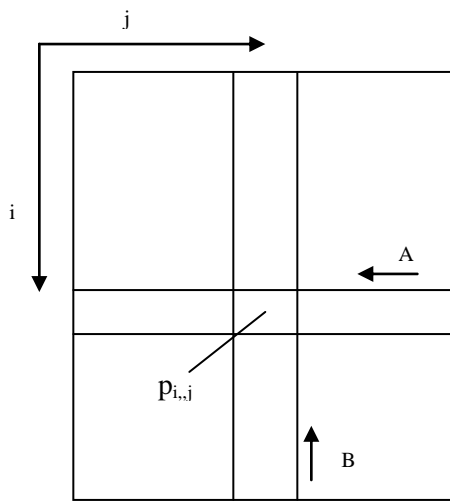


Figure 3.4a: Movement of A and B elements

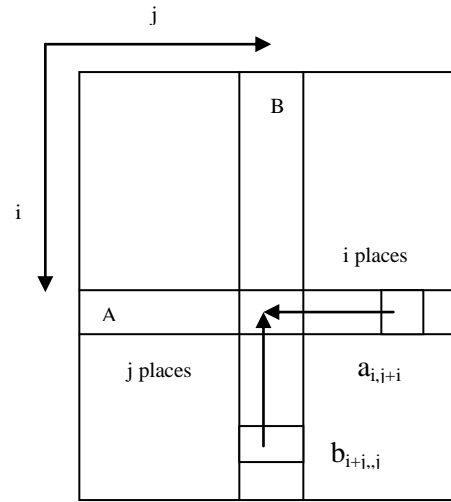


Figure 3.4b: Step 2-Alignment of elements of A and B

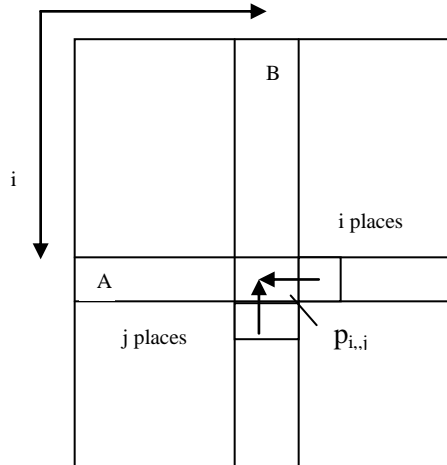


Figure 3.4c: Step 2-Alignment of elements of A and B

3.4.1. Communication Steps in Cannon's Algorithm

Applying the block row-column operation using the cannon's algorithm method for rhotrices with entries $A_{i,j}$ and $B_{i,j}$, we consider cannon's matrix multiplication algorithm from the point of view of process $P_{i,j}$. The processes are organized into a 2-D mesh, and each process has a block of A, and B needed for computing C respectively. Table 3.5c depicts the first block multiplication step. After each block multiplication process $P_{i,j}$ sends its block of A to the process on its left and receives a new block of A from the process on its right. Similarly, it sends its block of B to the process above it and receives a new block of B from the process below it. Table 3.4.1 *d*, *e* and *f* shows the remaining communication steps in Cannon's algorithm on 16 processes for rhotrix multiplication respectively. Summing the results of all block rhotrix multiplications yields $C_{i,j}$. Cannon's algorithm will reorder the summation in the inner loop of block rhotrix multiplication as follows:

$$c(i,j) = c(i,j) + \sum_{k=0}^{s-1} A(i,k)*B(k,j)$$

$$= C(i,j) + \sum_{k=0}^{s-1} A(i, (i+j+k) \bmod s)*B((i+j+k) \bmod s, j)$$

Cannon's rhotrix multiplication algorithm

for all $(i=0 \text{ to } s-1)$... "skew" A

Left-circular-shift row i of A by i ,

so that $A(i,j)$ is overwritten by $A(i, (j+i) \bmod s)$

end for
for all (i=0 to s-1) ... "skew" B
 Up-circular-shift column i of B by i,
 so that B(i,j) is overwritten by B((i+j) mod s, j)
end for
for all((i=0 to s-1, j=0 to s-1, k=0 to s-1) and (i%2≠0))
 $c(i,j) = c(i,j) + a(i,j)*b(2k-1,j)$
 Left-circular-shift each row of A by 1,
 so that A(i,j) is overwritten by A(i, (j+1) mod s)
 Up-circular-shift each column of B by 1,
 so that B(i,j) is overwritten by B((i+1) mod s, j)
end for
for all((i=0 to s-1, j=0 to s-1, k=0 to s-1) and (i%2==0))
 $c(i,j) = c(i,j) + a(i,j)*b(2k,j)$
 Left-circular-shift each row of A by 1,
 so that A(i,j) is overwritten by A(i, (j+1) mod s)
 Up-circular-shift each column of B by 1,
 so that B(i,j) is overwritten by B((i+1) mod s, j)
end for
end for

Table 3. 4.1 The communication steps in Cannon's algorithm on 16 processes

A ₀₀	A ₀₁	A ₀₂	A ₀₃
A ₁₀	A ₁₁	A ₁₂	A ₁₃
A ₂₀	A ₂₁	A ₂₂	A ₂₃
A ₃₀	A ₃₁	A ₃₂	A ₃₃
(a) Initial alignment of A			
A ₀₀ B ₀₀	A ₀₁ B ₁₁	A ₀₂ B ₂₂	A ₀₃ B ₃₃
A ₁₁ B ₁₀	A ₁₂ B ₂₁	A ₁₃ B ₃₂	A ₁₀ B ₀₃
A ₂₀ B ₂₀	A ₂₁ B ₃₁	A ₂₂ B ₀₂	A ₂₃ B ₁₃
A ₃₃ B ₃₀	A ₃₀ B ₀₁	A ₃₁ B ₁₂	A ₃₂ B ₂₃
(c) A and B after initial alignment			
A ₀₂ B ₂₀	A ₀₃ B ₃₁	A ₀₀ B ₀₂	A ₀₁ B ₁₃
A ₁₃ B ₃₀	A ₁₀ B ₀₁	A ₁₁ B ₁₂	A ₁₂ B ₂₃
A ₂₀ B ₀₀	A ₂₁ B ₁₁	A ₂₂ B ₂₂	A ₂₃ B ₃₃
A ₃₁ B ₁₀	A ₃₂ B ₂₁	A ₃₃ B ₃₂	A ₃₀ B ₀₃
(e) Submatrix location after second shift			

B ₀₀	B ₀₁	B ₀₂	B ₀₃
B ₁₀	B ₁₁	B ₁₂	B ₁₃
B ₂₀	B ₂₁	B ₂₂	B ₂₃
B ₃₀	B ₃₁	B ₃₂	B ₃₃
(b) Initial alignment of B			
A ₀₁ B ₁₀	A ₀₂ B ₂₁	A ₀₃ B ₃₂	A ₀₀ B ₀₃
A ₁₂ B ₂₀	A ₁₃ B ₃₁	A ₁₀ B ₀₂	A ₁₁ B ₁₃
A ₂₃ B ₃₀	A ₂₀ B ₀₁	A ₂₁ B ₁₂	A ₂₂ B ₂₃
A ₃₀ B ₀₀	A ₃₁ B ₁₁	A ₃₂ B ₂₂	A ₃₃ B ₃₃
(d) Submatrix location after first shift			
A ₀₃ B ₃₀	A ₀₀ B ₀₁	A ₀₁ B ₁₂	A ₀₂ B ₂₃
A ₁₀ B ₁₀₀	A ₁₁ B ₁₁	A ₁₂ B ₂₂	A ₁₃ B ₃₃
A ₂₁ B ₁₀	A ₂₂ B ₂₁	A ₂₃ B ₃₂	A ₂₀ B ₀₃
A ₃₂ B ₂₀	A ₃₃ B ₃₁	A ₃₀ B ₀₂	A ₃₁ B ₁₃
(f) Submatrix location after third shift			

Algorithm 5: Cannon's Algorithm for Row-Column Multiplication of Rhotrices

procedure RHOTRIX_MULT (a, b, c)

begin

for $i := 0$ to $t - 1$ *do*

for $j := 0$ to $t - 1$ *do*

begin

{Initialize all elements of $c_{i,j}$ to zero}

$c[i, j] := 0;$

if ($i \% 2 \neq 0$)

for $k := 0$ to $t - 1$ *do*

{Receive message from slave process (worker) i, j }

$c[i, j] = c[i, j] + a[i, k] \times b[2k - 1, j];$

endfor;

else

if ($i \% 2 == 0$)

for $k := 0$ to $t - 1$ *do*

{Receive message from slave process (worker) i, j }

$c[i, j] = c[i, j] + a[i, k] \times b[2k, j];$

endfor;

endfor;

endfor;

{Write C to output file}

end RHOTRIX_MULT

3.4.2. Analysis of Cannon Algorithm

Communication: in retrospect to submatrices, given v^2 submatrices, each of size $m \times m$, the initial alignment requires a maximum of $v-1$ shift (communication) operations for A and B each. After that, there will be $v-1$ shift operations for A and B each. Each shift operation involves $m \times m$ elements. Hence

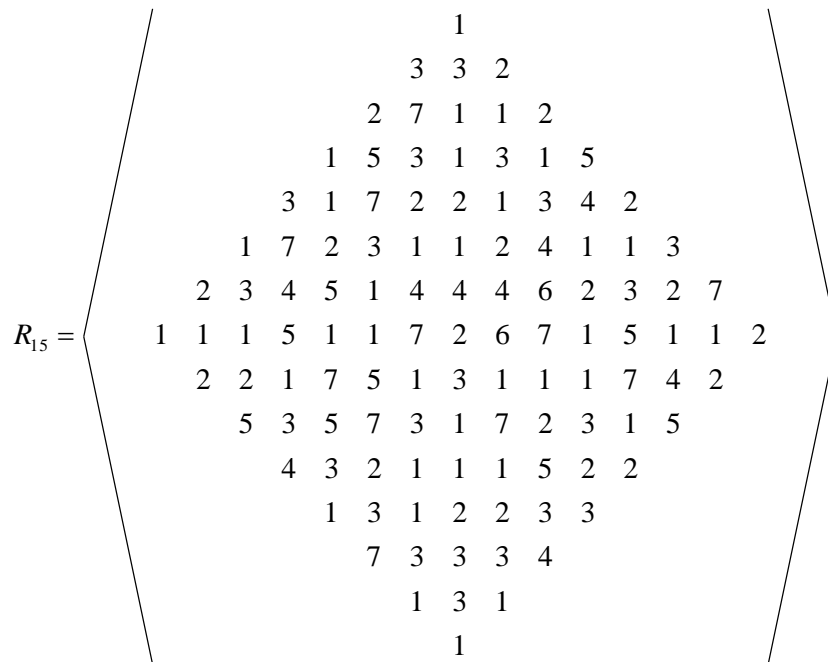
$$t_{comm} = 4(v-1)(t_{startup} + m^2 t_{data}) \text{ or a communication time complexity of } O(vm^2) \text{ or } (mn).$$

Computation: Each submatrix multiplication requires m^3 multiplications and m^3 additions. Therefore, with $v-1$ shifts,

$$t_{comm} = 2vm^3 = 2m^2n \text{ or a computation time complexity of } O(m^2n).$$

3.4.3. Theoretical Computation of Cannon Algorithm

We can now apply the above algorithm to compute for rhotrices of order R_{15} and Q_{15} , in essence we want to verify whether the derived algorithm follows and can equally be applied to multiplication of rhotrices of order N . Figure 3.4.3 illustrates the blocking method adopted from Cannon's algorithm for matrix multiplication.



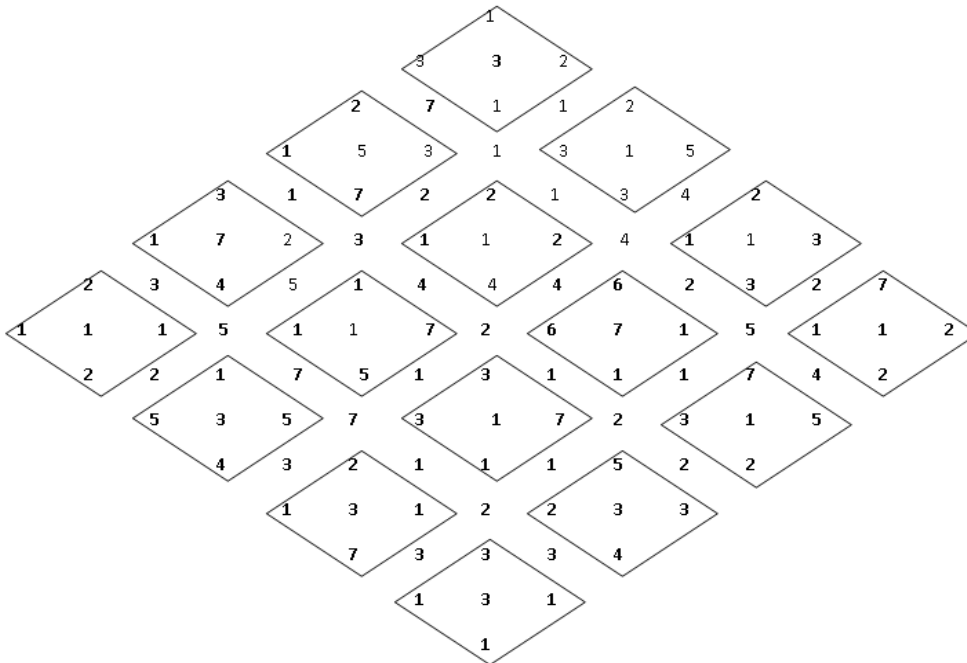
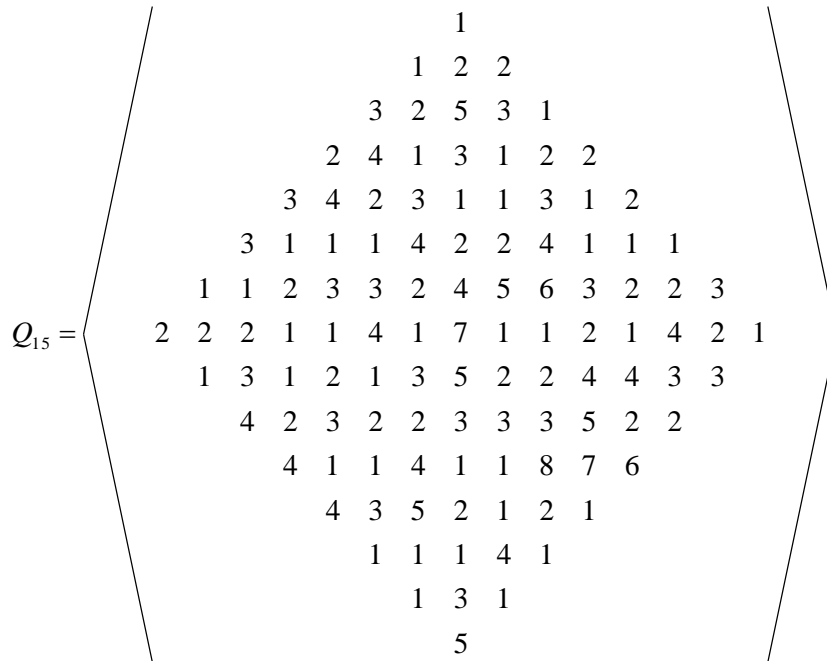


Figure 3.4.3a: Blocked rhotrix of dimensional fifteen, R_{15}

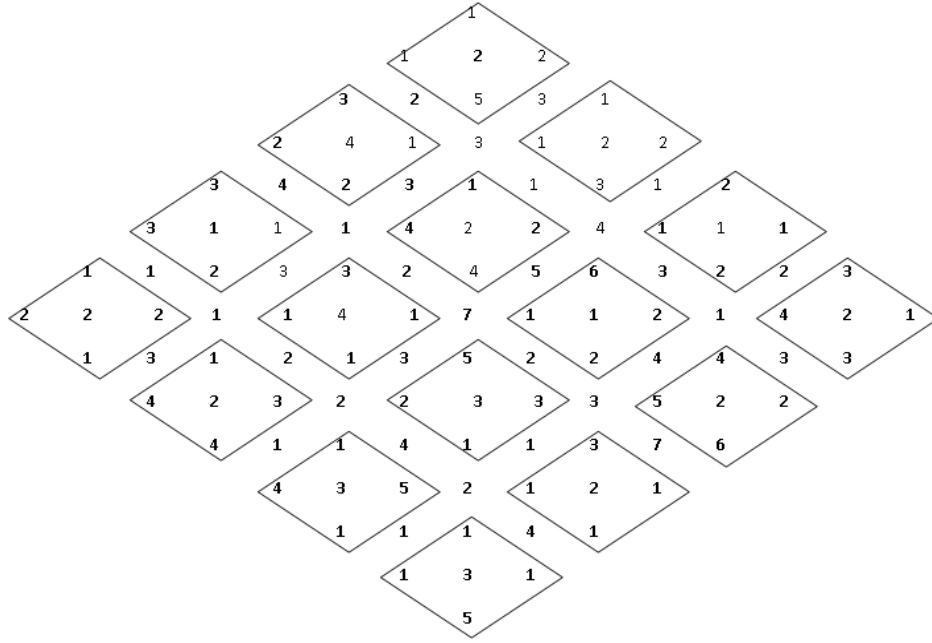


Figure 3.4.3b: Blocked rhotrix of dimensional fifteen, Q_{15}

$$A = \begin{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix} & \begin{bmatrix} 2 & 5 \\ 3 & 3 \end{bmatrix} & \begin{bmatrix} 2 & 3 \\ 1 & 3 \end{bmatrix} & \begin{bmatrix} 7 & 2 \\ 1 & 2 \end{bmatrix} \\ \begin{bmatrix} 2 & 3 \\ 1 & 7 \end{bmatrix} & \begin{bmatrix} 2 & 2 \\ 1 & 4 \end{bmatrix} & \begin{bmatrix} 6 & 1 \\ 6 & 1 \end{bmatrix} & \begin{bmatrix} 7 & 5 \\ 3 & 2 \end{bmatrix} \\ \begin{bmatrix} 3 & 2 \\ 1 & 4 \end{bmatrix} & \begin{bmatrix} 1 & 7 \\ 1 & 5 \end{bmatrix} & \begin{bmatrix} 3 & 7 \\ 3 & 1 \end{bmatrix} & \begin{bmatrix} 5 & 3 \\ 2 & 4 \end{bmatrix} \\ \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} & \begin{bmatrix} 1 & 5 \\ 5 & 4 \end{bmatrix} & \begin{bmatrix} 2 & 1 \\ 1 & 7 \end{bmatrix} & \begin{bmatrix} 3 & 1 \\ 1 & 1 \end{bmatrix} \end{bmatrix}, \quad B = \begin{bmatrix} \begin{bmatrix} 1 & 2 \\ 1 & 5 \end{bmatrix} & \begin{bmatrix} 1 & 2 \\ 1 & 3 \end{bmatrix} & \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} & \begin{bmatrix} 3 & 1 \\ 4 & 3 \end{bmatrix} \\ \begin{bmatrix} 3 & 1 \\ 2 & 2 \end{bmatrix} & \begin{bmatrix} 1 & 2 \\ 4 & 4 \end{bmatrix} & \begin{bmatrix} 6 & 2 \\ 1 & 1 \end{bmatrix} & \begin{bmatrix} 4 & 2 \\ 5 & 6 \end{bmatrix} \\ \begin{bmatrix} 3 & 1 \\ 3 & 2 \end{bmatrix} & \begin{bmatrix} 3 & 1 \\ 1 & 1 \end{bmatrix} & \begin{bmatrix} 5 & 3 \\ 2 & 1 \end{bmatrix} & \begin{bmatrix} 3 & 1 \\ 1 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 3 \\ 4 & 4 \end{bmatrix} & \begin{bmatrix} 1 & 5 \\ 4 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 1 \\ 1 & 5 \end{bmatrix} \end{bmatrix}$$

Blocks of rhotrix's submatrices of main entries of rhotrix R and Q

$$C = \begin{bmatrix} \begin{bmatrix} 3 & 1 \\ 7 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 4 \\ 1 & 4 \end{bmatrix} & \begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 4 & 0 \end{bmatrix} \\ \begin{bmatrix} 5 & 2 \\ 1 & 3 \end{bmatrix} & \begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} & \begin{bmatrix} 7 & 1 \\ 1 & 2 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 2 & 0 \end{bmatrix} \\ \begin{bmatrix} 7 & 5 \\ 3 & 5 \end{bmatrix} & \begin{bmatrix} 1 & 5 \\ 7 & 7 \end{bmatrix} & \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix} & \begin{bmatrix} 3 & 0 \\ 3 & 0 \end{bmatrix} \\ \begin{bmatrix} 1 & 2 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 2 & 3 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 3 & 3 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 3 & 0 \\ 0 & 0 \end{bmatrix} \end{bmatrix}, D = \begin{bmatrix} \begin{bmatrix} 2 & 3 \\ 2 & 3 \end{bmatrix} & \begin{bmatrix} 2 & 1 \\ 1 & 4 \end{bmatrix} & \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix} & \begin{bmatrix} 2 & 0 \\ 3 & 0 \end{bmatrix} \\ \begin{bmatrix} 4 & 3 \\ 4 & 1 \end{bmatrix} & \begin{bmatrix} 2 & 5 \\ 2 & 7 \end{bmatrix} & \begin{bmatrix} 1 & 4 \\ 2 & 3 \end{bmatrix} & \begin{bmatrix} 2 & 0 \\ 7 & 0 \end{bmatrix} \\ \begin{bmatrix} 1 & 3 \\ 1 & 1 \end{bmatrix} & \begin{bmatrix} 4 & 3 \\ 2 & 2 \end{bmatrix} & \begin{bmatrix} 2 & 1 \\ 4 & 2 \end{bmatrix} & \begin{bmatrix} 2 & 0 \\ 4 & 0 \end{bmatrix} \\ \begin{bmatrix} 2 & 3 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 2 & 1 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 3 & 1 \\ 0 & 0 \end{bmatrix} & \begin{bmatrix} 3 & 0 \\ 0 & 0 \end{bmatrix} \end{bmatrix}$$

Blocks of rhotrix's submatrices of heart entries of R and Q.

We choose to represent the rhotrix's submatrix with A and B for i rows with odd numbers and A' and B' for i rows with even numbers

Computation at process $P_{0,0}$, for $C_{0,0}$ and $C'_{0,0}$

After the first alignment

$$C_{0,0} = A_{0,0} + B_{0,0} = \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 1 & 5 \end{bmatrix} = \begin{bmatrix} 3 & 12 \\ 4 & 11 \end{bmatrix} \quad C'_{0,0} = A'_{0,0} + B'_{0,0} = \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 1 & 5 \end{bmatrix} = \begin{bmatrix} 3 & 12 \\ 4 & 11 \end{bmatrix}$$

After the first alignment

$$C_{0,0} = C_{0,0} + A_{0,1} \times B_{1,0} = \begin{bmatrix} 3 & 12 \\ 4 & 11 \end{bmatrix} + \begin{bmatrix} 2 & 5 \\ 3 & 3 \end{bmatrix} \times \begin{bmatrix} 3 & 1 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 3 & 12 \\ 4 & 11 \end{bmatrix} + \begin{bmatrix} 16 & 12 \\ 15 & 9 \end{bmatrix} = \begin{bmatrix} 19 & 24 \\ 19 & 20 \end{bmatrix},$$

$$C'_{0,0} = C'_{0,0} + C'_{0,1} \times A'_{1,0} = \begin{bmatrix} 8 & 12 \\ 16 & 24 \end{bmatrix} + \begin{bmatrix} 1 & 4 \\ 1 & 4 \end{bmatrix} \times \begin{bmatrix} 4 & 3 \\ 4 & 1 \end{bmatrix} = \begin{bmatrix} 8 & 12 \\ 16 & 24 \end{bmatrix} + \begin{bmatrix} 20 & 7 \\ 20 & 7 \end{bmatrix} = \begin{bmatrix} 28 & 19 \\ 36 & 31 \end{bmatrix},$$

After the second shift

$$C_{0,0} = C_{0,0} + A_{0,2} \times B_{2,0} = \begin{bmatrix} 19 & 24 \\ 19 & 20 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 1 & 3 \end{bmatrix} \times \begin{bmatrix} 3 & 1 \\ 3 & 2 \end{bmatrix} = \begin{bmatrix} 19 & 24 \\ 19 & 20 \end{bmatrix} + \begin{bmatrix} 15 & 8 \\ 12 & 7 \end{bmatrix} = \begin{bmatrix} 34 & 32 \\ 31 & 27 \end{bmatrix},$$

$$C'_{0,0} = C'_{0,0} + A'_{0,2} \times B'_{2,0} = \begin{bmatrix} 28 & 19 \\ 36 & 31 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix} \times \begin{bmatrix} 1 & 3 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 28 & 19 \\ 36 & 31 \end{bmatrix} + \begin{bmatrix} 3 & 5 \\ 7 & 11 \end{bmatrix} = \begin{bmatrix} 31 & 24 \\ 43 & 42 \end{bmatrix},$$

After the third shift

$$C_{0,0} = C_{0,0} + A_{0,3} \times B_{3,0} = \begin{bmatrix} 34 & 32 \\ 31 & 27 \end{bmatrix} + \begin{bmatrix} 7 & 2 \\ 1 & 2 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 34 & 32 \\ 31 & 27 \end{bmatrix} + \begin{bmatrix} 11 & 16 \\ 5 & 4 \end{bmatrix} = \begin{bmatrix} 45 & 48 \\ 36 & 31 \end{bmatrix},$$

$$C'_{0,0} = C'_{0,0} + A'_{0,3} \times B'_{3,0} = \begin{bmatrix} 31 & 24 \\ 43 & 42 \end{bmatrix} + \begin{bmatrix} 1 & 0 \\ 4 & 0 \end{bmatrix} \times \begin{bmatrix} 2 & 3 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 31 & 24 \\ 43 & 42 \end{bmatrix} + \begin{bmatrix} 2 & 3 \\ 8 & 12 \end{bmatrix} = \begin{bmatrix} 33 & 27 \\ 51 & 54 \end{bmatrix},$$

3.5. Verification of Strassen's Algorithm by Divide-and-Conquer Technique

Interestingly, Strassen's algorithm organizes arithmetic involving the sub-arrays A through H so that we can compute I , J , K , and L using just seven recursive matrix multiplications. We can easily verify that they work correctly.

Let us assume that n a power of two and let us partition P , Q , and R each into four $\frac{n}{2} \times \frac{n}{2}$ matrices, so that we can write $R = PQ$ as

$$\begin{pmatrix} W & X \\ Y & Z \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} E & F \\ G & H \end{pmatrix} \text{ Then,}$$

Let's defines seven sub-matrix products:

$$S_1 = A(F - H)$$

$$S_2 = (A + B)H$$

$$S_3 = (C + D)E$$

$$S_4 = D(G - E)$$

$$S_5 = (A + D)(E + H)$$

$$S_6 = (B - D)(G + H)$$

$$S_7 = (A - C)(E + F)$$

Given this seven sub-matrix products, we can compute W as

$$\begin{aligned} W &= S_5 + S_6 + S_4 - S_2 \\ &= (A + D)(E + H) + (B - D)(G + H) + D(G - E) - (A + B)H \\ &= AE + DE + AH + DH + BG - DG + BH - DH + DG - DE - AH - BH \\ &= AE + BG \end{aligned}$$

Similarly we can compute X as

$$\begin{aligned} X &= S_1 + S_2 \\ &= A(F - H) + (A + B)H \end{aligned}$$

$$\begin{aligned}
&= AF - AH + AH + BH \\
&= AF + BH
\end{aligned}$$

Again we can compute Y as

$$\begin{aligned}
Y &= S_3 + S_4 \\
&= (C + D)E + D(G - E) \\
&= CE + DE + DG - DE \\
&= CE + DG
\end{aligned}$$

Finally, we can compute Z as

$$\begin{aligned}
Z &= S_1 - S_7 - S_3 + S_5 \\
&= A(F - H) - (A - C)(E + F) - (C + D)E + (A + D)(E + H) \\
&= AF - AH - AE + CE - AF + CF - CE - DE + AE + DE + AH + DH \\
&\quad CF + DH
\end{aligned}$$

Thus, we can compute $R = PQ$ using seven recursive multiplications of matrices of size $\frac{n}{2} \times \frac{n}{2}$. Similarly, if we let a function $t(n)$ denote the running time of the algorithm on an input of size n , and characterize $t(n)$ using an equation that relates $t(n)$ to values of the function t for problem sizes smaller than n . We get recursion equation

$$t(n) = \begin{cases} b & \text{if } n < 2 \\ 7t(\frac{n}{2}) + kn & \text{if } n > 2, \end{cases} \quad (3.5)$$

If we use the iterative substitution method and we assume that the matrix size is fairly large enough, then by substituting (3.5) of the recurrence for each occurrence of the function t on the right-hand side, we can characterize the running time $t(n)$ as

$$t(n) = 7t(\frac{n}{2}) + kn^2,$$

for some constant $k > 0$. Hence by the master theorem, we have the following:

Theorem 1: we can multiply two $n \times n$ matrices in $O(n^{\log_2 7})$ time.

Thus, with a fair bit of additional computation, we can perform the multiplication for $n \times n$ matrices in time $O(n^{2.808})$, which is $o(n^3)$ time.

3. 5.1. Strassen’s Algorithm and Rhotrix Row-Column Multiplication

In accordance with the Winograd’s variant of Strassen’s algorithm, which uses seven matrix multiplications and 15 matrix additions we extend and implement similar algorithmic computation on rhotrix multiplication. It is well-known that this is the minimum number of multiplications and additions possible for any recursive matrix multiplication algorithm based on division into quadrants. The division of the matrices into quadrants follows equation (2.1) in Section 2.7.

Our major difficulty is on partitioning rhotrix elements into 2×2 blocks of quadrants knowing that rhotrix has an odd dimension. In actual sense this might seem complicated, but we can still combine both Strassen’s algorithm with the standard multiplication technique of matrix to achieve our goal. In this section we do not discuss the theoretical proof behind this algorithm as they are covered elsewhere. The following steps illustrate the splitting process of rhotrices into main and heart entries.

Definition: Sifting is the process of separating main entries or heart entries from the entire given entries of the rhotrix. It is denoted by the symbol \oplus with the main entries on the left and the heart entries on the right.

The above definition can be illustrated by the following example.

$$R_5 = \left\langle \begin{array}{cccc} & & a_{11} & \\ & a_{21} & c_{11} & a_{12} \\ a_{31} & c_{21} & a_{22} & c_{12} & a_{13} \\ & a_{32} & c_{22} & a_{23} \\ & & a_{33} & \end{array} \right\rangle = \left\langle \begin{array}{ccc} & & a_{11} \\ & a_{21} & a_{12} \\ a_{31} & a_{22} & a_{13} \\ & a_{32} & a_{23} \\ & & a_{33} \end{array} \right\rangle \oplus \left\langle \begin{array}{cc} & c_{11} \\ c_{21} & c_{12} \\ & c_{22} \end{array} \right\rangle$$

Note that the following relations is true from the above definition

$$\text{where } \left\langle \begin{array}{ccccc} & & a_{11} & & \\ & a_{21} & & a_{12} & \\ a_{31} & & a_{22} & & a_{13} \\ & a_{32} & & a_{23} & \\ & & a_{33} & & \end{array} \right\rangle \subset \left\langle \begin{array}{ccccc} & & a_{11} & & \\ & a_{21} & c_{11} & a_{12} & \\ a_{31} & & c_{21} & & a_{13} \\ & a_{32} & c_{22} & a_{23} & \\ & & a_{33} & & \end{array} \right\rangle$$

$$\text{and } \left\langle \begin{array}{ccc} & c_{11} & \\ c_{21} & & c_{12} \\ & c_{22} & \end{array} \right\rangle \subset \left\langle \begin{array}{ccccc} & & a_{11} & & \\ & a_{21} & c_{11} & a_{12} & \\ a_{31} & & c_{21} & & a_{13} \\ & a_{32} & c_{22} & a_{23} & \\ & & a_{33} & & \end{array} \right\rangle$$

Where \subset denotes rhotrix embedment and \oplus denotes rhotrix sifting.

Similarly,

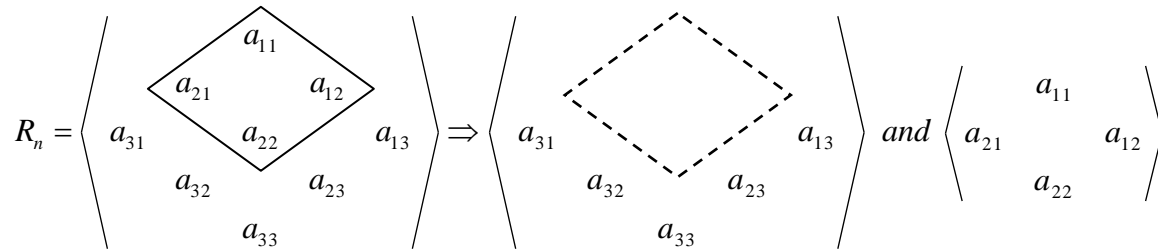
$$Q_5 = \left\langle \begin{array}{ccccc} & & b_{11} & & \\ & b_{21} & d_{11} & b_{12} & \\ b_{31} & & d_{21} & & b_{13} \\ & b_{32} & d_{22} & b_{23} & \\ & & b_{33} & & \end{array} \right\rangle = \left\langle \begin{array}{ccccc} & & b_{11} & & \\ & b_{21} & & b_{12} & \\ b_{31} & & b_{22} & & b_{13} \\ & b_{32} & & b_{23} & \\ & & b_{33} & & \end{array} \right\rangle \oplus \left\langle \begin{array}{ccc} & d_{11} & \\ d_{21} & & d_{12} \\ & d_{22} & \end{array} \right\rangle$$

$$\text{where } \left\langle \begin{array}{ccccc} & & b_{11} & & \\ & b_{21} & & b_{12} & \\ b_{31} & & b_{22} & & b_{13} \\ & b_{32} & & b_{23} & \\ & & b_{33} & & \end{array} \right\rangle \subset \left\langle \begin{array}{ccccc} & & b_{11} & & \\ & b_{21} & d_{11} & b_{12} & \\ b_{31} & & d_{21} & & b_{13} \\ & b_{32} & d_{22} & b_{23} & \\ & & b_{33} & & \end{array} \right\rangle$$

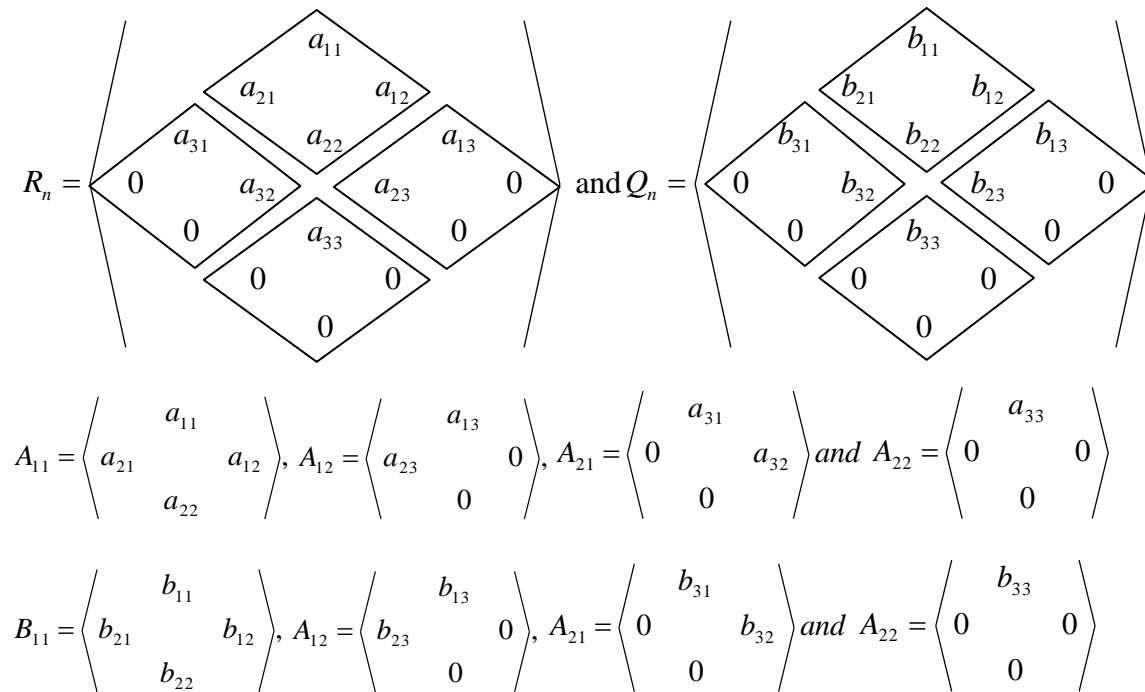
$$\text{and } \left\langle \begin{array}{ccc} & d_{11} & \\ d_{21} & & d_{12} \\ & d_{22} & \end{array} \right\rangle \subset \left\langle \begin{array}{ccccc} & & b_{11} & & \\ & b_{21} & d_{11} & b_{12} & \\ b_{31} & & d_{21} & & b_{13} \\ & b_{32} & d_{22} & b_{23} & \\ & & b_{33} & & \end{array} \right\rangle$$

It is important to observe that the heart entries of R_n and Q_n from the above extraction are not themselves rhotrices, but represents computational step among group of steps required in performing a multiplication tasks.

Once we are able to split this, then the next step is to find a way of partitioning these entries into blocks of 2×2 , as required by the desired algorithm.



If the rhortrices R_n, Q_n are not of type $2^n \times 2^n$, we have two options, first option is to either fill the missing rows and columns with zeros (a method referred to as padding approach) and partition R_n and Q_n into equally sized block matrices or we delete the un-partitioned rows and columns (a method referred to as peeling approach) and then later incorporate them into the computation process. But first let's consider the padding approach as shown below.



and

$$C_{11} = \begin{pmatrix} c_{11} & & \\ c_{21} & c_{12} & \\ & c_{22} & \end{pmatrix}, C_{12} = \begin{pmatrix} c_{13} & & \\ c_{23} & 0 & \\ & 0 & \end{pmatrix}, C_{21} = \begin{pmatrix} c_{31} & & \\ 0 & c_{32} & \\ & 0 & \end{pmatrix} \text{ and } C_{22} = \begin{pmatrix} c_{33} & & \\ 0 & & 0 \\ & 0 & \end{pmatrix}$$

with $A_{i,j}, B_{i,j}$ and $C_{i,j} \in \mathfrak{R}^{2^{n-1} \times 2^{n-1}}$ then

$$C_{1,1} = A_{1,1} B_{1,1} + A_{1,2} B_{2,1}$$

$$C_{1,2} = A_{1,1} B_{1,2} + A_{1,2} B_{2,2}$$

$$C_{2,1} = A_{2,1} B_{1,1} + A_{2,2} B_{2,1}$$

$$C_{2,2} = A_{2,1} B_{1,2} + A_{2,2} B_{2,2}$$

With this construction we have not reduced the number of multiplications. We still need 8 multiplications to calculate the $C_{i,j}$ matrices, the same number of multiplications we need when using standard matrix multiplication.

Now comes the important part. We define new matrices

$$M_1 := (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2})$$

$$M_2 := (A_{2,1} + A_{2,2})B_{1,1}$$

$$M_3 := A_{1,1}(B_{1,2} - B_{2,2})$$

$$M_4 := A_{2,2}(B_{2,1} - B_{1,1})$$

$$M_5 := (A_{1,1} + A_{1,2})B_{2,2}$$

$$M_6 := (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2})$$

$$M_7 := (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})$$

which are then used to express the $C_{i,j}$ in terms of M_k . Because of our definition of the M_k we can eliminate one matrix multiplication and reduce the number of multiplications to 7 (one multiplication for each M_k) and express the $R_{i,j}$ as

$$C_{1,1} = M_1 + M_4 - M_5 + M_7$$

$$C_{1,2} = M_3 + M_5$$

$$C_{2,1} = M_2 + M_4$$

$$C_{2,2} = M_1 - M_2 + M_3 + M_6$$

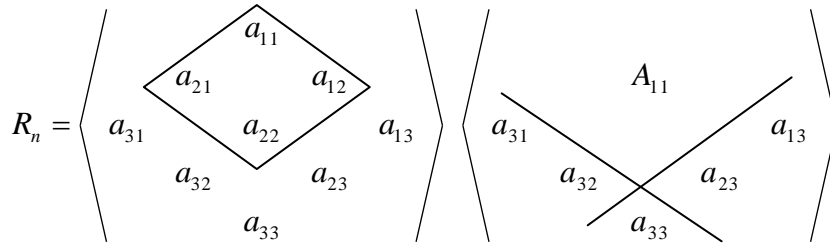
We iterate this division process n times until the sub matrices degenerate into numbers (elements of the ring R). At the end of the recursive computation we now do away with the zeros added to augments the missing rows and columns to get back our rhotrix.

Alternatively we consider one of the key approaches to dealing with matrices with odd dimension, the peeling method discussed in (Huss-Lederman *et al.*, 1996). The peeling approach discussed can be used to apply Strassen's algorithm to smaller rhotrices obtained by deleting rows and columns. In this case, after applying the Strassen's algorithm, still we need to do some additional work, referred to as fixup. The fixup has to be taken into consideration in other to include the contribution of the initial rhotrix's rows and columns that were deleted earlier on.

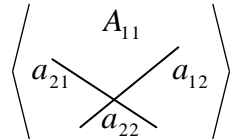
The peeling approach corresponds to blocking methods found in Cannon's algorithm and is strictly applied to only the rhotrix main entries whose dimension is usually odd, and not to the heart entries whose dimension would always be even. The input rhotrices are partitioned into blocks, possibly of different sizes, such that the rhotrix products is compatible with the dimensions of the blocks, that is, if $A_{i,j}$ is the i, j -block of the rhotrix R_n and $B_{j,k}$ is the j, k -blocks of the rhotrix Q_n , then the column dimension of $A_{i,j}$ must be equal to the row dimension of $B_{j,k}$ for all i, j , and k . If the dimensions of the blocks are compatible in this sense, then $C_{i,k}$, the i, k -block of

$C = AB$, is equal to $\beta \sum_{k=1}^n a_{i,k} b_{2k-1,j} + (1-\beta) \sum_{k=1}^{n-1} a_{i,k} b_{2k,j}$ where j runs over the blocks in the i -th columns of the block rhotrix obtained by partitioning A . After partitioning the input rhotrices into blocks, Strassen's algorithm can then be applied to compute the block products of rhotrices whose dimensions are even. The fixup work includes the block products that are not computed with Strassen's algorithm and the work required to combine the block products.

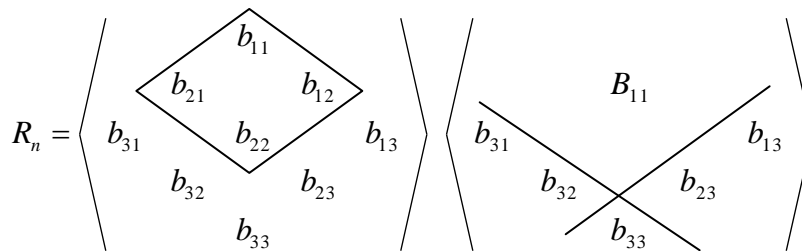
One partitioning scheme chooses the upper left-hand block to be the largest sub-rhotrix, with even dimensions, containing the element in the upper left-hand corner. In this scheme there can be at most four block, which occurs when both the row and column dimension are odd. In this case



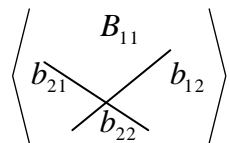
if we assume $a_{33} \Leftrightarrow a_{22}$ then



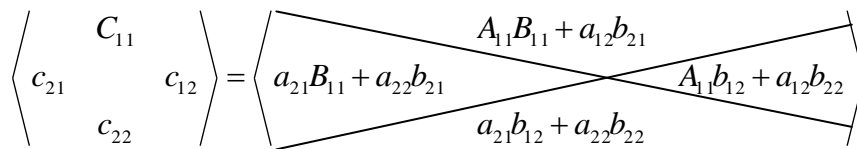
where A_{11} is an $(m - 1) \times (k - 1)$ matrix, a_{12} is a $(m - 1) \times 1$ matrix, a_{21} is a $1 \times (k - 1)$ matrix, a_{22} is a 1×1 matrix



if we assume $b_{33} \Rightarrow b_{22}$, $b_{23}, b_{13} \Rightarrow b_{12}$ and $b_{31}, b_{32} \Rightarrow b_{21}$ then,



and B_{11} is an $(k - 1) \times (n - 1)$ matrix, b_{12} is a $(k - 1) \times 1$ matrix, b_{21} is a $1 \times (n - 1)$ matrix, b_{22} is a 1×1 matrix. The product $C = AB$ is computed as



The above peeling approach for which we applied to solve for the rhotrix main entries as stated before do not apply to the heart elements with even dimension unless otherwise, hence the good news is that we can directly apply the Strassen's algorithm for the computation of the heart entries

as shown below. The algorithm basically describes how to perform a single level recursion on 2×2 blocks:

$$\left\langle \begin{array}{cc} R_{11} & \\ R_{21} & R_{12} \\ & R_{22} \end{array} \right\rangle = \left\langle \begin{array}{cc} C_{11} & \\ C_{21} & C_{12} \\ & C_{22} \end{array} \right\rangle \left\langle \begin{array}{cc} D_{11} & \\ D_{21} & D_{12} \\ & D_{22} \end{array} \right\rangle$$

where C_{11} , C_{12} , C_{21} , and C_{22} are $(m - 1) \times (k - 1)$ matrices and D_{11} , D_{12} , D_{21} , and D_{22} are $(k - 1) \times (n - 1)$ matrices respectively of the heart entries and $R_{i,j}$ denotes the cumulative partial sum.

3.5.2. Strassen's Algorithm Computation Approach

The computation process can be splinted into two major parts; the first part handles the recursive computation of 2×2 blockable matrices with even dimensions while the latter parts handles the fixup (i.e., multiplication resulting from the recursive computation with the peeled off row and column) case resulting from the earlier peeling processes carried out. For the first part, the computation can proceeds as shown in Table 3.5. for both rhotrix main entries and heart entries.

Table 3.5 Strassen's Algorithm for Rhotrix Multiplication

Computation for rhotrix main entries	Computation for rhotrix heart entries
Phase 1 and 2	Phase 1 and 2
$S1 = A21 + A22$ $T1 = B12 - B11$	$S1 = C21 + A22$ $T1 = D12 - D11$
$S2 = S1 + A11$ $T2 = B22 - T1$	$S2 = S1 + C11$ $T2 = D22 - T1$
$S3 = A11 - A21$ $T3 = B22 - B12$	$S3 = C11 - C21$ $T3 = D22 - D12$
$S4 = A12 - S2$ $T4 = B21 - T2$	$S4 = C12 - S2$ $T4 = D21 - T2$
Phase 3	Phase 3
$P1 = A11 \times B11$ $P5 = S3 \times T3$	$P1 = C11 \times D11$ $P5 = S3 \times T3$
$P2 = A12 \times B21$ $P6 = S4 \times B22$	$P2 = C12 \times D21$ $P6 = S4 \times D22$
$P3 = S1 \times T1$ $P7 = A22 \times T4$	$P3 = S1 \times T1$ $P7 = C22 \times T4$
$P4 = S2 \times T2$	$P4 = S2 \times T2$
Phase 4	Phase 4

$C11 = U1 = P1 + P2$	$R11 = U1 = P1 + P2$
$U2 = P1 + P4$	$U2 = P1 + P4$
$U3 = U2 + P5$	$U3 = U2 + P5$
$C21 = U4 = U3 + P7$	$R21 = U4 = U3 + P7$
$C22 = U4 = U3 + P7$	$R22 = U4 = U3 + P7$
$U6 = U2 + P3$	$U6 = U2 + P3$
$C12 = U7 = U6 + P6$	$R12 = U7 = U6 + P6$

Fig 3.5a shows the computation graph for the Winograd variant of Strassen's algorithm. The sub-blocks rhotrices A and B, and C and D are the initial inputs, shown labelling the vertices at the top of the two graphs. All edges are directed from inputs, higher vertices to lower ones below. Thus, for example, in fig. 3.5a A11 and A21 are initial inputs, which are combined to produce S3. S3 is subsequently combined with T3 to produce P5, and so on, until the final results, sub-blocks of R, are produced at the bottom level.

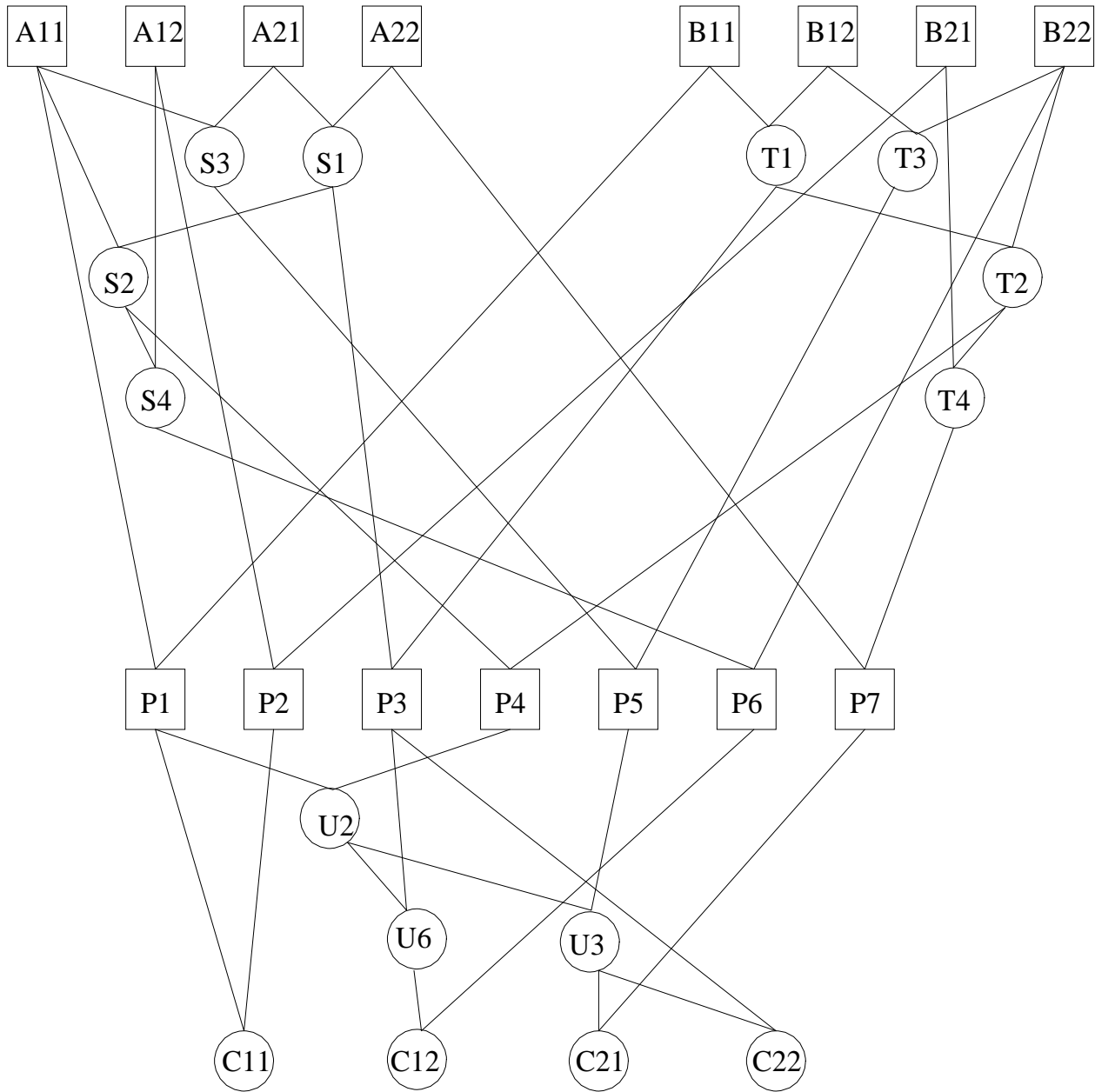


Fig. 3.5a. Dependency graph for Winograd's variant of Strassen's Algorithm

3.5.3. Strassen's Algorithm Fixup Case

The second part of our computation deals with the dynamic peeling approach explained earlier on, based on the previous analysis, it has been shown that the dynamic peeling could be a useful method, but it has not been previously tested through actual implementation for rhortices. We wrote a java program to implement the essentially data-driven computation and presented a framework model for the parallel MPI implementation.

After combining operations, there are potentially three fixup steps:

$$\text{Step 1:} \quad C_{11} = a_{12}b_{21} + C_{11},$$

$$\text{Step 2:} \quad c_{12} = \left\langle \begin{array}{c} A_{11} \\ a_{12} \end{array} \right\rangle \circ \left\langle \begin{array}{c} b_{12} \\ b_{22} \end{array} \right\rangle + c_{12},$$

$$\text{Step 3:} \quad \left\langle \begin{array}{c} c_{21} \\ c_{22} \end{array} \right\rangle = \left\langle \begin{array}{c} a_{21} \\ a_{22} \end{array} \right\rangle \circ \left\langle \begin{array}{cc} b_{21} & B_{11} \\ b_{22} & b_{12} \end{array} \right\rangle + \left\langle \begin{array}{c} c_{21} \\ c_{22} \end{array} \right\rangle.$$

The three steps can be computed using the standard rhotrix multiplication method. It should be noted that, A_{11} and B_{11} are the result obtained from our initial recursive computation for even dimensioned blockable rhotrices and a_{12} , a_{21} , a_{22} , b_{12} , b_{21} and b_{22} are the peeled off row and column elements.

3.5.4. Program Task Graph for Strassen's Algorithm

It is straightforward to represent Strassen's algorithm in a task graph. A task graph is a directed graph whose edges represent dependence relationship and nodes represent tasks (either sequential or parallel programs). The node located at the arc tail should be executed before the node at the arc head. Fig. 3.5b displays the corresponding task graph of one-level recursion of Strassen's algorithm. Winograd's variant of Strassen's algorithm uses the required 7 multiplications and minimal number of 15 additions/subtractions. This is simplified into the following 4 stages of computations.

Stages (1) and (2)

$$S1 = A21 + A22,$$

$$S2 = S1 - A11 = A21 + A22 - A11,$$

$$S3 = A11 - A21,$$

$$S4 = A12 - S2 = A12 - A21 - A22 + A11,$$

$$T1 = B12 - B11,$$

$$T2 = B22 - T1 = B22 - B12 + B11,$$

$$T3 = B22 - B12,$$

$$T4 = B21 - T2 = B21 - B22 + B12 - B11.$$

Stages (3) compute the seven products

$$P1 = A11B11,$$

$$P2 = A12B21,$$

$$P3 = S1T1,$$

$$P4 = S2T2,$$

$$P5 = S3T3,$$

$$P6 = S4B22,$$

$$P7 = A22T4,$$

And stage (4) computes

$$U1 = P1 + P2,$$

$$U2 = P1 + P4,$$

$$U3 = U2 + P5,$$

$$U4 = U3 + P7,$$

$$U5 = U3 + P3,$$

$$U6 = U2 + P3,$$

$$U7 = U6 + P6.$$

Hence we can easily verify that $C11 = U1$, $C12 = U7$, $C21 = U4$, and $C22 = U5$

We can now apply the above task notation to produce a simplified task graph as shown below.

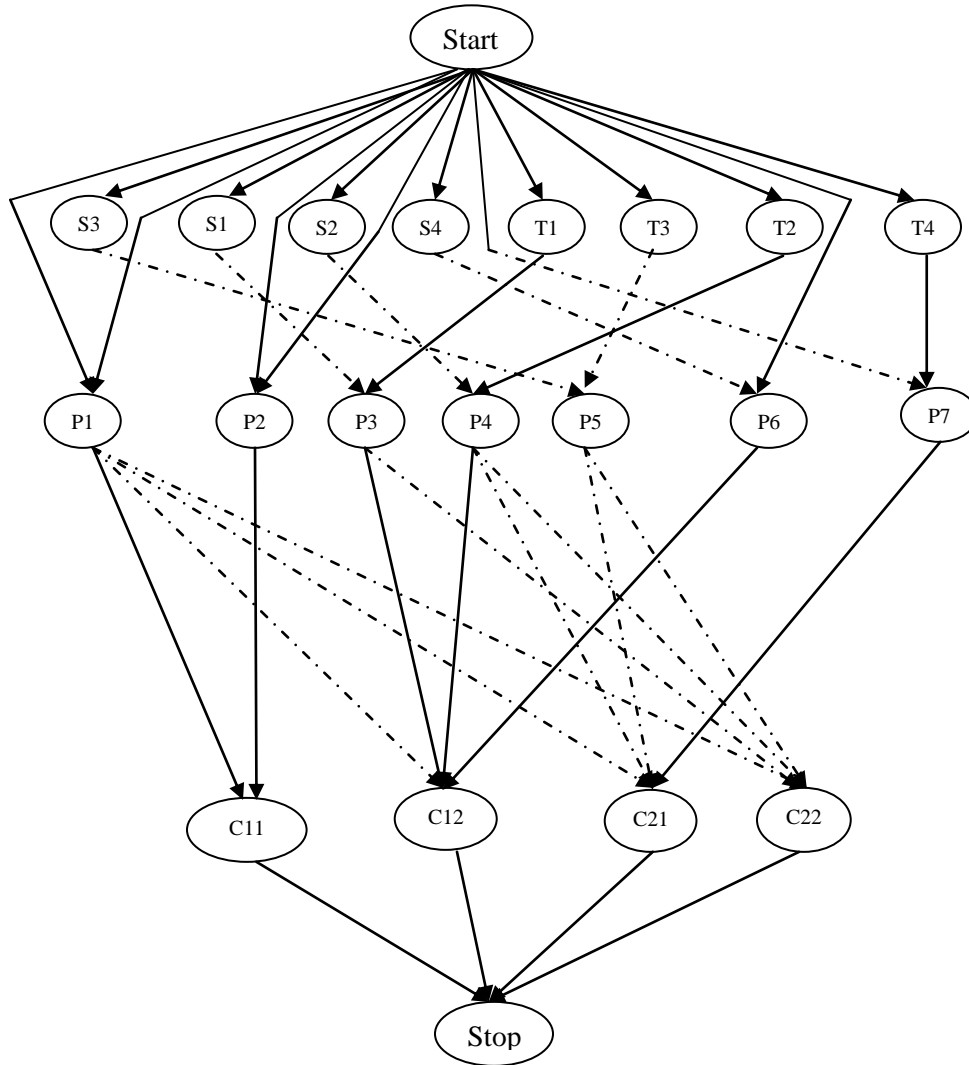


Fig. 3.5b Task graph for Winograd's variant of Strassen's Algorithm

Let consider rhotrix padding sample computation of the above algorithm based on the given rhotrices R_5 and Q_5 given in Section 3.5.1 as shown below:

$$A_{11} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, \quad A_{12} = \begin{bmatrix} a_{13} & 0 \\ a_{23} & 0 \end{bmatrix}$$

$$A_{21} = \begin{bmatrix} a_{31} & a_{32} \\ 0 & 0 \end{bmatrix}, \quad A_{22} = \begin{bmatrix} a_{33} & 0 \\ 0 & 0 \end{bmatrix}$$

$$B_{11} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}, \quad B_{12} = \begin{bmatrix} b_{13} & 0 \\ b_{23} & 0 \end{bmatrix}$$

$$B_{21} = \begin{bmatrix} b_{31} & b_{32} \\ 0 & 0 \end{bmatrix}, \quad B_{22} = \begin{bmatrix} b_{33} & 0 \\ 0 & 0 \end{bmatrix}$$

Phase one

$$T_1 = A_{11} + A_{22} = \begin{bmatrix} a_{11} + a_{33} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

$$T_2 = A_{21} + A_{22} = \begin{bmatrix} a_{31} + a_{33} & a_{32} \\ 0 & 0 \end{bmatrix}$$

$$T_3 = A_{11} + A_{12} = \begin{bmatrix} a_{11} + a_{13} & a_{12} \\ a_{21} + a_{23} & a_{22} \end{bmatrix}$$

$$T_4 = A_{21} - A_{11} = \begin{bmatrix} a_{31} - a_{11} & a_{32} + a_{12} \\ -a_{21} & -a_{22} \end{bmatrix}$$

$$T_5 = A_{12} - A_{22} = \begin{bmatrix} a_{13} - a_{33} & 0 \\ a_{23} & 0 \end{bmatrix}$$

$$T_6 = B_{11} + B_{22} = \begin{bmatrix} b_{11} + b_{33} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$T_7 = B_{12} - B_{22} = \begin{bmatrix} b_{13} - b_{33} & 0 \\ b_{23} & 0 \end{bmatrix}$$

$$T_8 = B_{21} - B_{11} = \begin{bmatrix} b_{31} - b_{11} & b_{32} - b_{12} \\ -b_{21} & -b_{22} \end{bmatrix}$$

$$T_9 = B_{11} + B_{12} = \begin{bmatrix} b_{11} + b_{13} & b_{12} \\ b_{21} + b_{23} & b_{22} \end{bmatrix}$$

$$T_{10} = B_{21} + B_{222} = \begin{bmatrix} b_{31} + b_{33} & b_{32} \\ 0 & 0 \end{bmatrix}$$

Phase two

$$Q_1 = T_1 \times T_6 = \begin{bmatrix} a_{13}b_{11} + a_{11}b_{33} + a_{33}b_{11} + a_{33}b_{33} + a_{12}b_{21} & a_{11}b_{12} + a_{33}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{21}b_{33} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

$$Q_2 = T_2 \times B_{11} = \begin{bmatrix} a_{31}b_{11} + a_{33}b_{11} + a_{32}b_{21} & a_{31}b_{12} + a_{33}b_{22} \\ 0 & 0 \end{bmatrix}$$

$$Q_3 = A_{11} \times T_7 = \begin{bmatrix} a_{11}b_{13} - a_{11}b_{33} + a_{12}b_{23} & 0 \\ a_{21}b_{13} - a_{21}b_{33} + a_{22}b_{23} & 0 \end{bmatrix}$$

$$Q_4 = A_{22} \times T_8 = \begin{bmatrix} a_{33}b_{31} - a_{33}b_{11} & a_{33}b_{32} - a_{33}b_{12} \\ 0 & 0 \end{bmatrix}$$

$$Q_5 = T_3 \times B_{22} = \begin{bmatrix} a_{11}b_{33} + a_{13}b_{33} & 0 \\ a_{21}b_{33} + a_{23}b_{33} & 0 \end{bmatrix}$$

$$Q_6 = T_4 \times T_9 =$$

$$\begin{bmatrix} a_{31}b_{11} - a_{31}b_{13} - a_{11}b_{11} - a_{11}b_{13} + a_{32}b_{21} + a_{32}b_{23} - a_{12}b_{21} - a_{12}b_{23} & a_{31}b_{12} - a_{11}b_{12} + a_{32}b_{22} - a_{12}b_{22} \\ -a_{21}b_{11} - a_{21}b_{13} - a_{22}b_{21} - a_{22}b_{23} & -a_{21}b_{12} - a_{22}b_{22} \end{bmatrix}$$

$$Q_7 = T_5 \times T_{10} = \begin{bmatrix} a_{13}b_{31} + a_{13}b_{33} - a_{33}b_{31} - a_{33}b_{33} & a_{13}b_{32} - a_{33}b_{32} \\ a_{23}b_{31} + a_{23}b_{33} & a_{23}b_{32} \end{bmatrix}$$

Phase three

$$T_1 = Q_1 + Q_4 = \begin{bmatrix} a_{11}b_{11} + a_{11}b_{23} + a_{33}b_{33} + a_{12}b_{21} + a_{33}b_{31} & a_{11}b_{12} + a_{12}b_{22} + a_{33}b_{32} \\ a_{21}b_{11} + a_{21}b_{33} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

$$T_2 = Q_5 - Q_7 = \begin{bmatrix} a_{11}b_{33} - a_{13}b_{31} + a_{33}b_{31} + a_{33}b_{33} & -a_{13}b_{33} - a_{33}b_{32} \\ a_{21}b_{33} - a_{23}b_{31} & -a_{23}b_{32} \end{bmatrix}$$

$$T_3 = Q_3 + Q_1 = \begin{bmatrix} a_{11}b_{13} + a_{12}b_{23} + a_{11}b_{11} + a_{33}b_{11} + a_{33}b_{33} + a_{12}b_{21} & a_{11}b_{12} + a_{33}b_{12} + a_{12}b_{22} \\ a_{21}b_{13} + a_{22}b_{23} + a_{21}b_{11} - a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

$$T_4 = Q_2 - Q_6 = \begin{bmatrix} a_{33}b_{11} + a_{31}b_{13} + a_{11}b_{11} + a_{11}b_{13} - a_{32}b_{23} + a_{12}b_{21} + a_{12}b_{23} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{21}b_{13} + a_{22}b_{21} + a_{22}b_{23} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

Phase four

$$C_{11} = T_1 - T_2 = \begin{bmatrix} a_{11}b_{11} + a_{11}b_{33} + a_{33}b_{33} + a_{12}b_{21} + a_{33}b_{31} - a_{11}b_{33} + a_{13}b_{31} - a_{33}b_{31} - a_{33}b_{33} & a_{11}b_{12} + a_{12}b_{22} + a_{33}b_{32} + a_{13}b_{33} + a_{33}b_{32} \\ a_{21}b_{11} + a_{21}b_{33} + a_{22}b_{21} - a_{21}b_{33} + a_{23}b_{31} & a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \end{bmatrix}$$

$$C_{12} = Q_3 + Q_5 = \begin{bmatrix} a_{11}b_{13} + a_{12}b_{23} + a_{13}b_{33} & 0 \\ a_{21}b_{13} + a_{22}b_{23} + a_{23}b_{33} & 0 \end{bmatrix}$$

$$C_{21} = Q_2 - Q_4 = \begin{bmatrix} a_{33}b_{11} + a_{32}b_{21} + a_{33}b_{31} & a_{31}b_{12} + a_{33}b_{22} + a_{33}b_{32} - a_{33}b_{12} \\ 0 & 0 \end{bmatrix}$$

$$C_{22} = T_3 - T_4 = \begin{bmatrix} a_{11}b_{13} + a_{12}b_{23} + a_{11}b_{11} + a_{33}b_{11} + a_{33}b_{33} + a_{12}b_{21} - a_{33}b_{31} + a_{33}b_{11} & a_{11}b_{12} + a_{33}b_{12} + a_{12}b_{22} - a_{33}b_{32} + a_{33}b_{12} \\ a_{21}b_{13} + a_{22}b_{23} + a_{21}b_{11} - a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

3.6. Contribution Summary

In Section 3.2.1 and 3.3, we presented two distinctive approaches of rhotrix multiplication. The first approach is an initial extension of work done by A. O. Ajibade, who pioneered the concept of rhotrices in 2003. His method takes cognizance of the central element of the rhotrix elements referred to as rhotrix hearts. In our study, we have tagged this with the keyword heart-oriented operation. The second approach is the modification of the generalization of row-column method introduced by B. Sani, in 2004 and 2007. Our major contribution to these two approaches could be found in the generalization techniques presented above and also the algorithm design for representing rhotrices multiplication on computer systems.

3.7. Parallel Multiplication of Rhotrices

Abdullahi *et al* (2010) investigated the possibility of parallelising the multiplication of n -dimensional rhotrices, using Cannon's algorithm. Their aim was to partition rhotrix into blocks of square submatrices of the form $2^{n-1} \times 2^{n-1}$. The challenges that might be encountered

with this method is that, rhotrix have odd dimension, which makes it difficult to apply Cannon's algorithm designed purposely for square matrices to rhotrix. In this work, we explore a more promising method of multiplying higher order rhotrices in parallel environment using processor array or systolic array architecture.

3.8. Architecture and Data-parallel Operations for Processor Array

The architecture of a generic processor array consists of a collection of processing elements controlled by a front-end computer. The front-end computer is a standard uni-processor. Its primary memory contains the instructions being executed as well as data that are manipulated sequentially by the front end. The processor array is divided into many individual processor-memory pairs. Data that are manipulated in parallel are distributed among these memories. In order to perform a parallel operation, the front-end computer transmits the appropriate instruction to the processors in the processor array, which simultaneously execute the instruction on operand stored in their local memories. A controlled path (indicated by a dashed line as shown in the figure below) allows the front-end computer to broadcast instructions to the back-end processors.

One key features of this architecture is that, consideration is made for the cases were the number of input vectors outnumber processing elements. In a scenario when the size of the input vector exceeds the number of processors in processor array, some or all the processors need to store and manipulate multiple vector elements. For example, a 10,000-element vector can be stored on a 1024-processor system by giving 784 processor 10 elements and 240 processors 9 elements: $784 \times 10 + 240 \times 9 = 10,000$. One important observation is that, depending upon the architecture and operating system invoke the mapping of vector elements to processor may or may not have to be managed by the programmer.

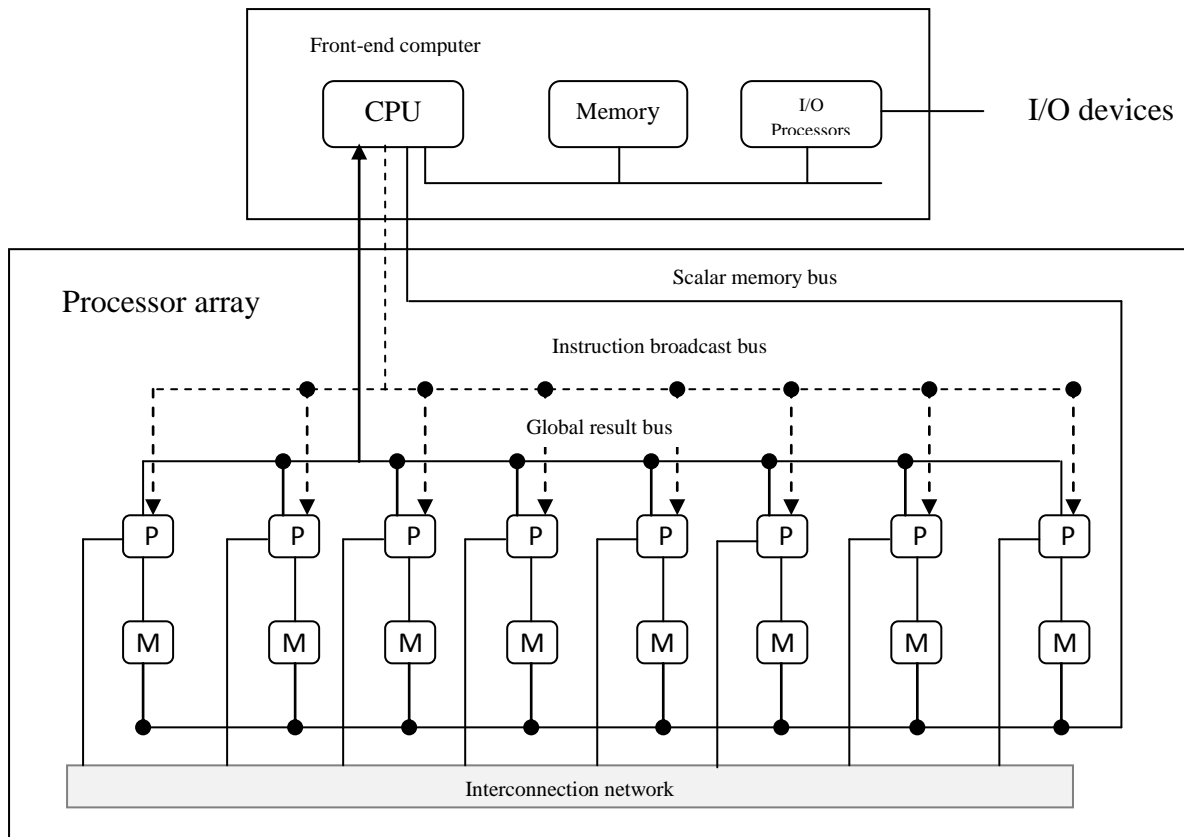


Figure 3.8a: Architecture of a generic processor array (adopted from Quinn, 2003)
 The processor array contains many primitive processors (shown by boxes labelled with a P).
 Each processor has its own local memory (shown by boxes labelled with an M)

3.8.1. Structure of Processing Interconnection Network

To bring together operands stored in the memories of different processors, the processors can pass data through an interconnection network. For the purpose of this research work, we pay attention to the two-dimensional mesh. The two dimensional mesh has the advantage of relatively straightforward implementation in VLSI, where a single chip may contain a large number of processors. The interconnection network supports concurrent message passing. For example, in the two-dimensional mesh shown in Figure 3.8b, each processing element can simultaneously send a value to the processing element to its “North” or “East.”

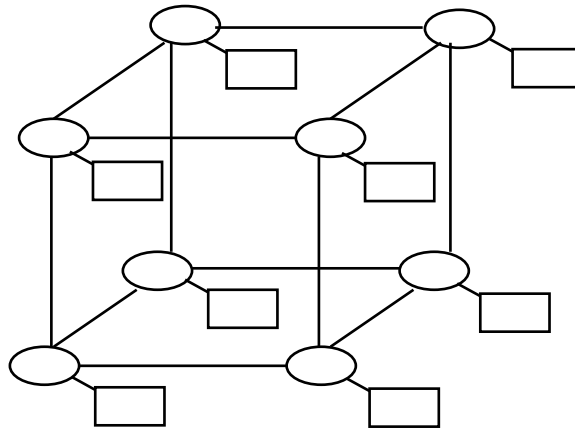


Figure 3.8b Two-dimensional mesh interconnection networks

3.8.2. Architectural Features of Processor Arrays (Enabling and Disabling processors)

One major characteristic of the processor array is its synchronous execution ability. Simply put, all the individual processors work in lockstep. However, it is possible for only a subset of the processors to perform an instruction. Each processor has a masking bit that allows it to “opt out” of performing an instruction. Masking is useful if the number of data items being manipulated is not an exact multiple of the size of the processor array. Masking also enables the processor array to support conditional executed parallel operations.

Considering an instance of how the masking operation is being effected, supposing rhotrix element A is distributed across the processor array from the north and another rhotrix element B distributed otherwise from “east”, since we cannot multiply the heart elements with the main entries of a rhotrix, we can assign mask bit of 1 to the task of assigning and multiplying the rhotrix main entries and then assign a mask bit of 0 to the tasks of assigning and multiplying the heart of the rhotrix entries. This could further be view this way, supposing at the onset we distribute an integer vector A consisting of just 0s and 1s across the processor array, one element per processor, first, every processor tests to see if its element of A has the value 0, if so, the processor set its mask bit, indicating it is not executing the next instruction. The unmasked processor set their elements of A to 1. At this point the mask bits are flipped, so that previously active processors become inactive, and vice versa. Now the unmasked processors set their elements of A to -1. Finally, all the mask bits are erased.

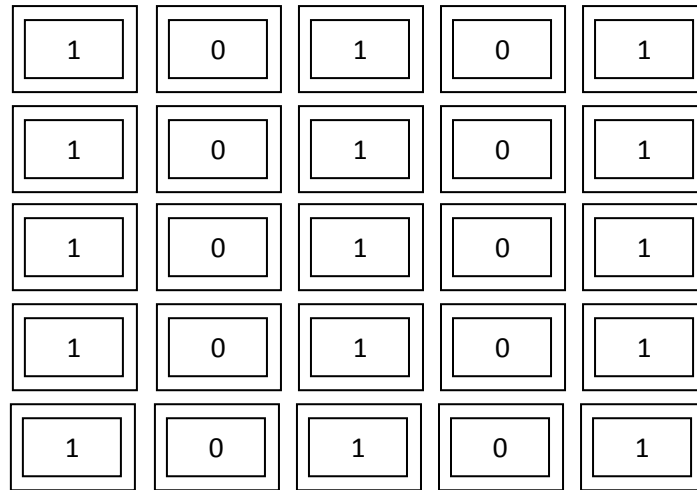


Figure 3.8c Execution of an if-then-else statement

If processor mask id or bit is 1, computation of rhotrix main entries is performed else, computations of the heart elements are performed. Best imagined, Rhotrix main entries are assigned job id of “1” which are mapped to processors with masks bit of “1” and rhotrix heart entries assigned job id of “0” which are mapped to processors with mask bit of “0”. The unmasked process is carried out as follows, for the first step of processing; if the value of the processor id is zero then it is set to -1. This marks the first lockstep process, indicating an inactive process state. For the second step, if the processor id is 1, it is set to -1, indicating an inactive process state for the second lockstep. The active processors set their values of A to -1. The active and inactive processors switch role is shown in Figure 3.8d.

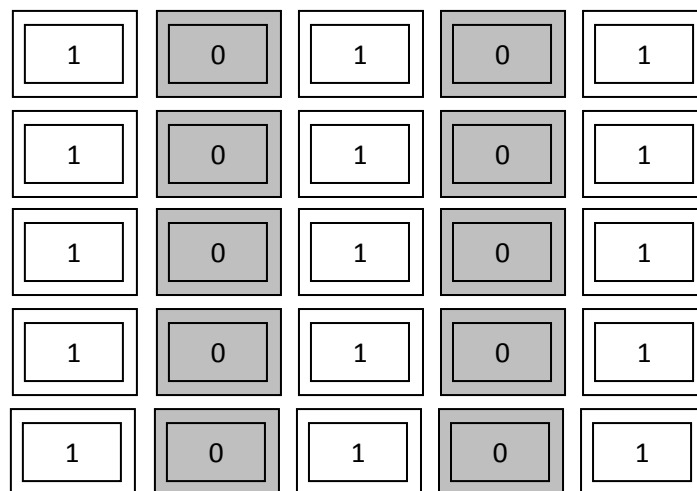


Figure. 3.8d Inactive processors states

Shading indicates those processors that are masked out (inactive) because their elements of A are 0. The remaining processors set their values of A to 1.

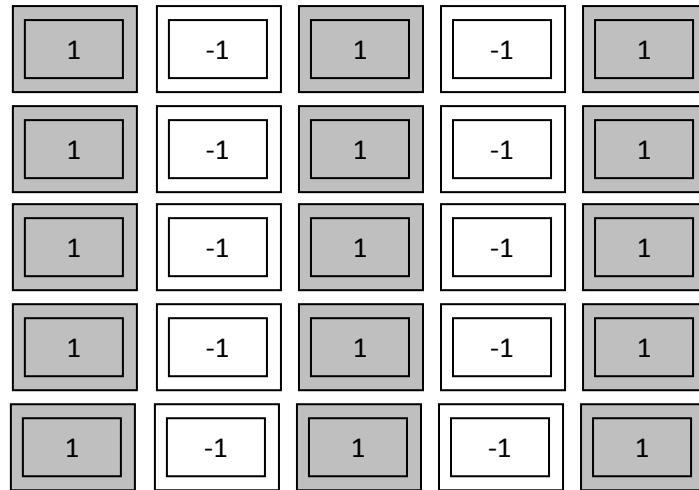


Figure 3.8e This figure indicates the active and inactive processors switch roles. The active processors set their values of A to -1.

3.9. The Mapping Problem

This section is almost a digression. It is devoted to the study of the simplest variant of the problem. The mapping problem involves the decision of how the concurrently executable units determine by the partitioning method are assigned to the processors of the multicomputer system in order to achieve a maximal degree of parallelism in an efficient manner. Since the mapping will affect the overall execution time and thus the speedup attainable by the parallel program, it must be chosen carefully. In particular, it should keep the computational load of the processors balanced and should minimize the communication overhead among the processors. Because of its importance, the mapping problem has attracted a great amount of research efforts during the last few years (Norman and Thanisch, 1993). Section 3.9.1 to Section 3.9.6 is solemnly devoted to find a more suitable mapping pattern for rothrix elements (vectors) on participating processors for every given tasks allocation, independent of the grid topology.

3.9.1. Design of Processor Array Mapping Techniques

The processor array architecture design can be summarized in the following points.

- The processor array architectures are designed by using linear mapping techniques on regular dependence graphs (DG).
- Regular Dependency Graph: the presence of an edge in a certain direction at any node in the DG represents presence of an edge in the same direction at all nodes in the DG

- DG corresponds to space representation \rightarrow no time instance is suggested to any computation $\Rightarrow T = 0$.
- Systolic architectures have a space-time representation where each node is mapped to a certain processing element (PE) and is schedule at a particular time instance.
- Systolic design methodology maps an N-dimensional DG to a lower dimensional systolic architecture

Definitions:

- Projection vector (also referred to as iteration vector), $d = \begin{pmatrix} d_1 \\ d_2 \end{pmatrix}$ two nodes that are displaced by d or multiples of d are executed by the same processor
- Processor space vector, $P^T = (P_1 \ P_2)$ any node with index $I^T=(i,j)$ would be executed by processor;

$$P^T I = (P_1 \ P_2) \begin{pmatrix} i \\ j \end{pmatrix}$$

- Scheduling vector, $S^T = (S_1 S_2)$. Any node with index I would be executed at time, $S^T I$.
- Hardware Utilization Efficiency, $HUE = 1/|S^T d|$. This is because two tasks executed by the same processor are spaced $|S^T d|$ time units apart.
- Processor space vector and projection vector must be orthogonal to each other $\Rightarrow P^T d=0$.
- If A and B are mapped to the same processor, then they cannot be executed at the same time, that is, $S^T I_A \neq S^T I_B$ that is, $S^T d \neq 0$.
- Edge mapping: if an edge e exists in the space representation or DG , then an edge $P^T e$ is introduce in the systolic array with $S^T e$ delays.
- A DG can be transformed to a space-time representation by interpreting one of the dimensions as temporal dimension. For a 2-D DG , the general transformation is described by $i' = T = 0$, $j' = P^T I$, and $t' = S^T I$, that is,

$$\begin{pmatrix} i' \\ j' \\ t' \end{pmatrix} = T \begin{pmatrix} i \\ j \\ t \end{pmatrix} = \begin{pmatrix} 0 & 0 & 0 \\ \cdot & P' & 0 \\ \cdot & S' & 0 \end{pmatrix} \begin{pmatrix} i \\ j \\ t \end{pmatrix}$$

j' = processor axis

T = scheduling time instance

3.9.2. Linear Mapping of Processing Element

Dependency vectors:

Let $f(i_1, i_2, \dots, i_n)$ be a recurrent equation defined on an n -dimensional integral space Z^n :

$$f(i_1, i_2, \dots, i_n) = \Phi(f(j_1, j_2, \dots, j_n), \dots, f(j'_1, j'_2, \dots, j'_n)) \quad (3.9.1)$$

denote the domain of f by Ω . Each point in Ω represents a unit computation which is called a computation point. (Computation points are comparable to the names of process or objects in other models of computation). Based on the definition of f , a dependency relation among all the computation points can be defined.

Definition: A computation point U is said to depend on a computation point V (denoted by $U \succ V$), if $f(v)$ appears on the right hand side of equation (3.9.1), when $F(U)$ is on the left hand side. The difference of the two points $d = u - v$, is called a dependency vector (associated with U).

Denote the set of dependency vectors associated with a computation point U by $D(u)$. For a general recurrent equation, $D(u)$ may depend on U . For example, in the following equation, at every odd- k point $D(u) = \{(1, 1)\}$, while at every other point $D(u) = \{(-1, 1)\}$

$$f(i, j) = \begin{cases} f(i-1, k-1) + 1 & \text{if } k \text{ odd} \\ f(i+1, k-1) + 1 & \text{otherwise} \end{cases} \quad (3.9.2)$$

3.9.3. Communication Vectors

We consider the class of parallel machine having regular multi-dimensional network structures. Processing elements are inter-connected by communication channels or links, typical network topologies including hexagonal networks and torus-connected k -ary n -cubes (hypercube fall into this group). In such networks every node has the same set of communication links. For example, in a $2D$ square mesh, every node will have four links connecting to its nearest neighbour in north, south, east and west directions, respectively.

Communication links are represented by communication vectors. If the dimension of the network is k , then each communication vector is of length $k+1$. The first component corresponds to k spatial dimension and the last component represents time. We assume that passing a message through a communication link takes one unit of time. Hence, the last component of every communication vector is 1. Furthermore, we represent time delays explicitly. A unit delay is represented by a communication vector $(0, \dots, 0, 1)$. A topology

matrix can be defined for each network such that each communication vector is a column vector. For convenience, we assume that the delay vector $(0, \dots, 0, 1)$ is always the last column vector in a topology matrix.

3.9.4. Linear Mapping

Let Ω denote the domain of an n dimensional QURE. Let F denote a linear mapping function from Z^n , and let Ft denote the time component of F . The function F maps each point in Ω to a point in the range R which represents a process to be executed on a certain node in the network. Linear mapping preserve the dependency relation among the computation points. We call the set of points in the range R with dependency relation “ \succ ” a design of the QURE. Any meaningful mapping function F has to satisfy the causality constraint in the time domain.

Definition: A linear mapping satisfying the following two conditions is called a basic linear mapping.

A1. For all x, y in Ω , if $x \neq y$, then $F(x) \neq F(y)$;

A2. For all x, y in Ω , if y depends on x , then $Ft(x) < Ft(y)$.

A linear mapping from Z^n to Z^n can be represented by $n \times n$ square matrix. Let T be such a matrix. T may contain elements that are rational. Let D be the dependency matrix of the network.

Definition 3: Given a square with a dependency matrix D and a network with topology C , a square matrix T satisfying the following three conditions is called a valid linear mapping of D with respect to C

B1. T is non-singular;

B2: For every column vector d in D , $t.d > 0$, where t is the last row vector of T , which represents time (\cdot is the inner product of vectors).

B3: (validity w.r.t topology C) for each column vector d in D , the resulting space-time vector Td must be a composition of communication vectors given by matrix C , where a composition is a linear combination with non-negative integral coefficients.

3.9.5. Selection of Scheduling Vector S^T Based on Scheduling Inequalities

For a dependence relation $X \rightarrow Y$, where $I_x^T = (i_x, j_x)^T$ and $I_y^T = (i_y, j_y)^T$ are respectively the indices of the nodes X and Y . The scheduling inequality for this dependence is given by.

$$S_y \geq S_x + T_x$$

Where T_x is the computation time of the node X . The scheduling equations can be classified into the following two types:

- Linear scheduling, where

$$S_x = S^T I_x = (S_1 \ S_2)(i_x j_x)^T$$

$$S_y = S^T I_y = (S_1 \ S_2)(i_y j_y)^T$$

- Affine scheduling, where

$$S_x = S^T I_x + \mathcal{V}_x = (S_1 \ S_2)(i_x j_x)^T + \mathcal{V}_x$$

$$S_y = S^T I_y + \mathcal{V}_y = (S_1 \ S_2)(i_y j_y)^T + \mathcal{V}_y$$

So scheduling equation for affine scheduling is as follows:

$$S^T I_x + \mathcal{V}_y \geq S^T I_x + \mathcal{V}_X + T_x$$

Each edge of a DG leads to an inequality for selection of the scheduling vectors which consists of 2 steps.

- Capture all fundamental edges. The reduced dependence graph (RDG) is used to capture the fundamental edges and the regular iterative algorithm (RIA) description of the corresponding problem is used to construct RDGs.
- Construct the scheduling inequalities according to

$$S^T I_x + \mathcal{V}_y \geq S^T I_x + \mathcal{V}_X + T_x$$

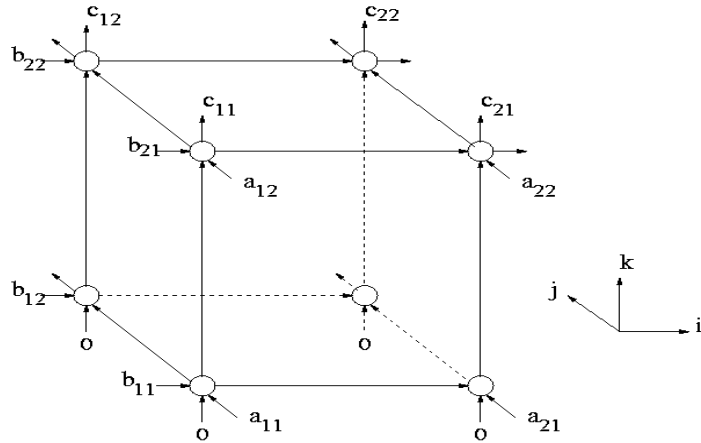
and solve them for feasible S^T .

3.9.6. Rhotrix Multiplication and 2-D Processor Array Design

If true that rhotrix is a supper matrix, then we can extract separately the elements of rhotrix hearts from the main entries in form of a square matrix with resultant products values of C_{11} , C_{12} , C_{21} , and C_{22} for a five-dimensional rhotrix as shown below. Again the main entries could equally be considered as square matrix.

$$C_{11} = a_{11}b_{11} + a_{12}b_{31} \quad C_{12} = a_{11}b_{12} + a_{12}b_{32}$$

$$C_{21} = a_{21}b_{11} + a_{22}b_{21} \quad C_{22} = a_{21}b_{12} + a_{22}b_{22}$$



The iteration in standard output RIA form is as follows:

$$a(i,j,k) = a(i,j-i,k), \quad b(i,j,k) = b(i-i,j,k), \quad c(i,j,k) = c(i,j,k-i) + a(i,j,k) * b(i,j,k)$$

- Applying scheduling inequality with

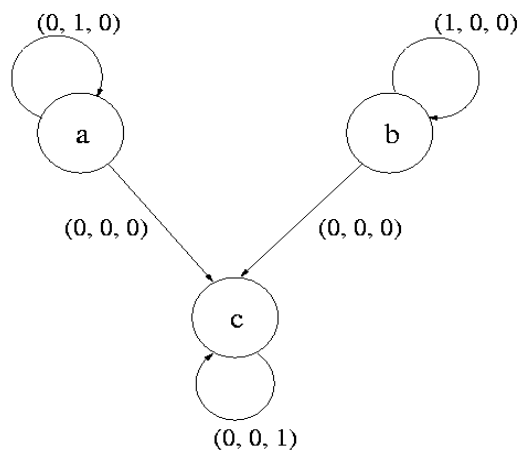
$T_{mult-add} = 1$, and $T_{com} = 0$ we get

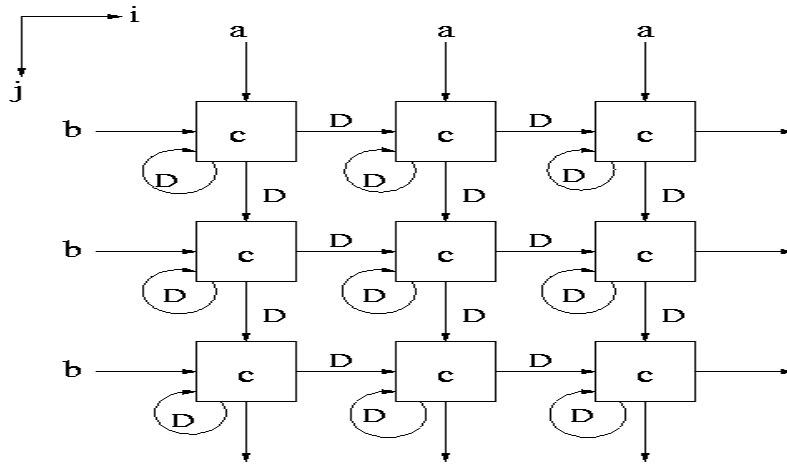
$S_2 \geq 0$, $S_1 \geq 0$, $S_3 \geq 1$, $\gamma_c - \gamma_a \geq 0$ and $\gamma_c - \gamma_b \geq 0$. Take $\gamma_a = \gamma_b = \gamma_c = 0$ for linear scheduling.

- Solution 1:

$$S^T = (1, 1, 1), \quad d^T = (0, 0, 1), \quad P_1 = (1, 0, 0)$$

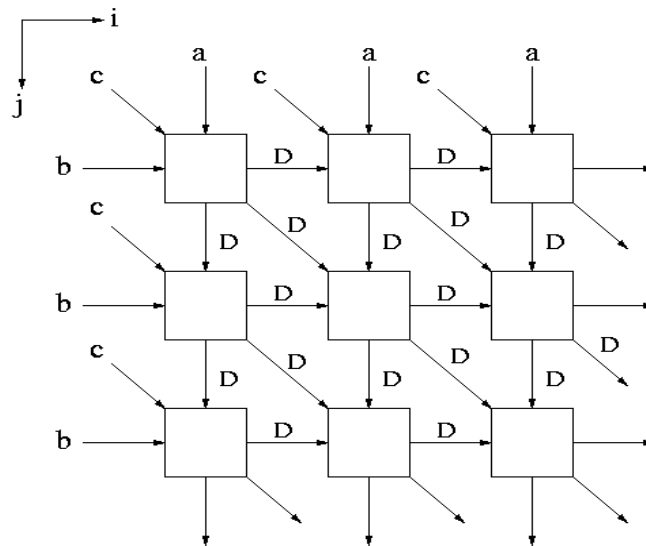
$$P_2 = (0, 1, 0), \quad P^T = (P_1 \ P_2)^T$$





- Solution two:

$$S^T = (1, 1, 1), d^T = (1, 1, -1), P_1 = (1, 0, 1), P_2 = (0, 1, 1), P^T = (P_1 P_2)^T$$



Solution one			Solution two		
e	$p^T e$	$s^T e$	E	$P^t e$	$s^T e$
a(0, 1, 0)	(0, 1)	1	a(0, 1, 0)	(0, 1)	1
a(1, 0, 0)	(1, 0)	1	b(1, 0, 0)	(1, 0)	1
a(0, 0, 1)	(0, 0)	1	c(0, 0, 1)	(1, 1)	1

3.10. Rhotrix Multiplication on Process Array Grid

multiplication algorithm of an $n \times m$ rhotrix A with $n \times p$ matrix B results in new matrix denoted by C of dimension $m \times p$. Matrix C is given by $C = A \cdot B$ where the elements are defined as:

$$c_{i,j} = \sum_{k=1}^n a_{i,k} b_{2k-1,j} \quad (3.10.1)$$

$$c_{i,j} = \sum_{k=1}^{n-1} a_{i,k} b_{2k,j} \quad (3.10.2)$$

$$c_{i,j} = \sum_{k=1}^n a_{i,k} b_{k,j} \quad (3.10.3)$$

The thesis scope considered the parallel computation of the above three equations under two different parallel platform, in which case (3.10.1) and (3.10.2) are dealt with using the process array computational platform, while (3.10.3) was considered under both process array and message passing interface parallel platform. The message passing parallel programming interface is considered in Section 3.11 and its algorithm design structure presented in Section 3.12.

This method can be realized with the array of processors of dimension $m \times p$. The principle is the same as in Figure. 3.3a. The connections are realized in horizontal and in vertical directions. Therefore the mesh connections of Linear processor Array structure is convenient for this operation where the data stream of rhotrix A is flowing to the right and the data stream of rhotrix B is flowing top down. The elements of rhotrix R are stored in the appropriate processors of the array. In the case of the rhotrix-rhotrix computation the expected speedup is.

$$O\left(\frac{n^3}{\log n}\right) \quad (3.10.4)$$

3.10.1. Pseudocode for N-Dimension Rhotrix Multiplication on Process Array Grid

This section outlines the algorithm that governs the parallel multiplication of two n -dimensional rhotrices on processor array architecture. The pseudocode of the algorithm follows. The main algorithm uses three auxiliary algorithm $colFlip(A_n)$, $rowFlip(B_n)$, and $Stagger(A_n, B_n)$.

Consider n -dimensional rhotrices R_5 and Q_5

Step 1: Modify R_n by flipping its columns 1, 2,.... & j , where j is the last column of rhotrix R_n

Step 2: Modify Q_n by flipping its rows 1, 2,.... & i , where i is the last row of rhotrix Q_n

Step 3: Stagger the data sets for input

Step 4: Feed the staggered input into the systolic array architecture for parallel multiplication.

3.10.2. Input Data Movement Pattern for (Coupled matrix)

Let $R_5 = \left\langle \begin{array}{ccccc} & & a_{11} & & \\ & a_{21} & c_{11} & a_{12} & \\ a_{31} & c_{21} & a_{22} & c_{12} & a_{13} \\ & a_{32} & c_{22} & a_{23} & \\ & & a_{33} & & \end{array} \right\rangle$, using definition of half transpose of rhotrices defined in

(Sani, 2007), we have the half transpose of R_5 denoted by $R_5^{T/2}$, as

$$R_5^{T/2} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ & c_{11} & c_{12} \\ a_{21} & a_{22} & a_{23} \\ & c_{21} & c_{22} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \sim \begin{pmatrix} a_{11} & 0 & a_{12} & 0 & a_{13} \\ 0 & c_{11} & 0 & c_{12} & 0 \\ a_{21} & 0 & a_{22} & 0 & a_{23} \\ 0 & c_{21} & 0 & c_{22} & 0 \\ a_{31} & 0 & a_{32} & 0 & a_{33} \end{pmatrix}$$

This $R_5^{T/2}$ is known as coupled matrix (Sani, 2007). Now supposing we have two coupled matrices

$R_5^{T/2}$ and $Q_5^{T/2}$ represented by uniform indexed variable as shown below:

$$R_5^{\frac{T}{2}} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{pmatrix} \quad \text{and} \quad Q_5^{\frac{T}{2}} = \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \\ b_{51} & b_{52} & b_{53} & b_{54} & b_{55} \end{pmatrix}$$

Figure 3.10a: Uniformly indexed variables of coupled matrices

We need to modify the matrices input data.

Flipping $R_5^{T/2}$ row-wise and $Q_5^{T/2}$ column-wise, thus, the resulting rhotrices are as shown in figure 3.10b and 3.10c:

$$\begin{array}{ccccc}
 a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\
 a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\
 a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\
 a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \\
 a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & b_{51} & b_{52} & b_{53} & b_{54} & b_{55}
 \end{array}$$

Flipping columns 1, 2, 4 & 5 and Flipping row 1, 2, 4 & 5 gives:

$$\begin{array}{ccccc}
 a_{15} & a_{14} & a_{13} & a_{12} & a_{11} & b_{51} & b_{52} & b_{53} & b_{54} & b_{55} \\
 a_{25} & a_{24} & a_{23} & a_{22} & a_{21} & b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \\
 a_{35} & a_{34} & a_{33} & a_{32} & a_{31} & b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\
 a_{45} & a_{44} & a_{43} & a_{42} & a_{41} & b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\
 a_{55} & a_{54} & a_{53} & a_{52} & a_{51} & b_{11} & b_{12} & b_{13} & b_{14} & b_{15}
 \end{array}$$

Figure 3.10b: Flipped $R_5^{T/2}$

Figure 3.10c: Flipped $Q_5^{T/2}$

We now proceed to stagger the input data into the individual processing units, following the steps given in Section 3.10.1 as shown in Figure 3.10e.

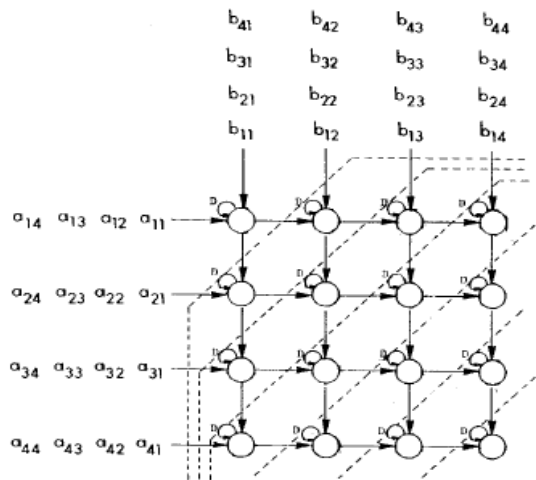


Figure 3.10d General array processor computation representation

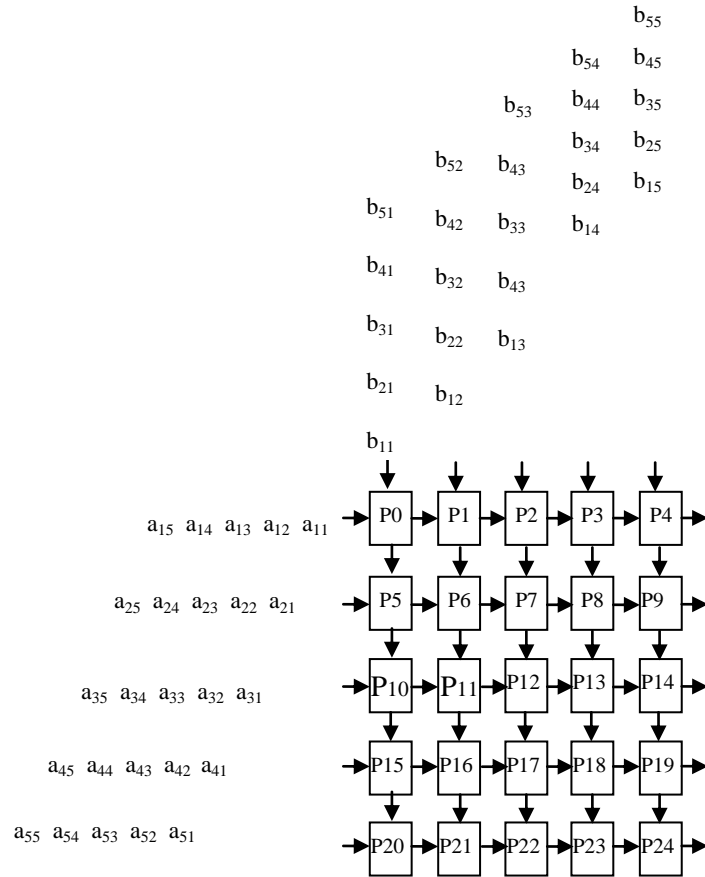


Figure 3.10e: Staggered input data of the coupled matrices

3.10.3. Input Data Movement Pattern for Rhotrix Elements

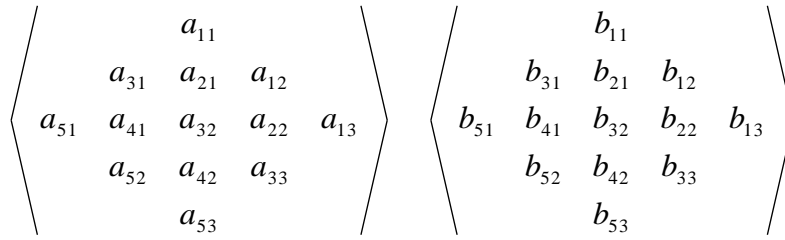
In this section we tried to extend similar input data modification of the above pseudocode to rhotrix multiplication without transforming the rhotrix into coupled matrix. This approach will be based on previous definition of the derived techniques presented in Section 3.3.1

$$\text{Let } R_9 \circ Q_9 = \left\langle \begin{array}{ccccc} & & a_{11} & & \\ & & a_{31} & a_{21} & a_{12} \\ a_{51} & a_{41} & a_{32} & a_{22} & a_{13} \\ & & a_{52} & a_{42} & a_{33} \\ & & & a_{53} & \end{array} \right\rangle \circ \left\langle \begin{array}{ccccc} & & & & b_{11} \\ & & b_{31} & b_{21} & b_{12} \\ b_{51} & b_{41} & b_{32} & b_{22} & b_{13} \\ & & b_{52} & b_{42} & b_{33} \\ & & & & b_{53} \end{array} \right\rangle$$

Now suppose we have the above two rhotrices R_5 and Q_5 as follows:

We need to modify the rhotrices input data as shown below.

Flipping R_5 row-wise and R_5 column-wise, thus, the resulting rhotrices are as shown in fig 3.10f and 3.10.g:



Flipping columns 1 & 3 and Flipping rows 1 & 3 gives:

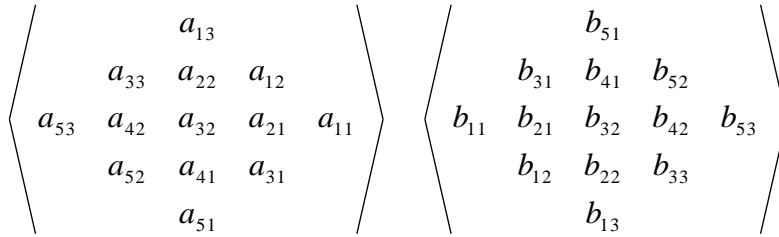


Fig 3.10f: Flipped R_5

Fig 3.10g: Flipped Q_5

We now proceed to stagger the input data into the individual processing units, following the steps given in Section 3.10.1 as shown below in Figure 3.10h.

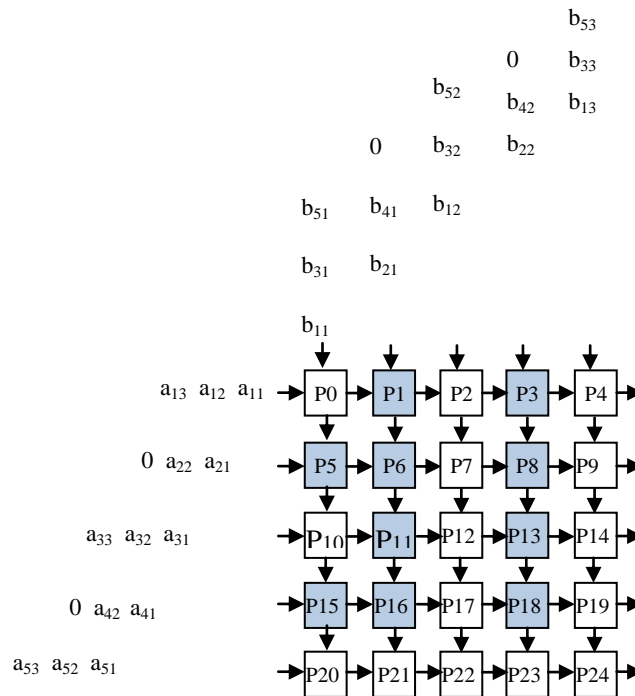


Figure 3.10h Staggered input data of the two rhotrices

The shaded processes indicate processors in inactive states, while the un-shaded indicates processors active states. These states fluctuate depending on input data id's and processor mask bit. This feature has been expatiated in Section 3.8.2.

Next, we implement a parallel algorithm for the pseudocode presented in Section 3.10.1, this algorithm is as shown below.

Algorithm 6: Rhotrix elements column flip operation module

```
colFlip(a[n, n]){
    col ← n;
    for i ← 1 to n{
        for j ← 1 to n{
            a[i, j] ← a[i, col];
            col ← col-1;
        }
        col ← n;
    }
}
```

Algorithm 7: Rhotrix element row flip operation module

```
rowFlip(b[n, n]){
    row ← n;
    for i ← 1 to n{
        for j ← 1 to n{
            a[i, j] ← a[row, j];
        }
        row ← row-1;
    }
}
```

Algorithm 8: Staggering operation module

```
Stagger(a[n, n], b[n, n]){
    for i:1 to n{
        for j:1 to n{
            pi,1 ← a[i, j];
            p1,j ← b[j, i];
        }
    }
}
```

Algorithm 9: Rhotrix multiplication module

```
RhotMul( $A_n, B_n$ ){
Input: Two n-d rhotrices;
Output:  $R_n$  product of rhotrices  $A_n$  and  $B_n$ ;
colFlip( $A_n$ );
rowFlip( $B_n$ );
Stagger ( $A_n, B_n$ );
c ← 0;
Pi,j receive a from pi,j-1 and b from pi-1,j;
c:=c + a*b;
Pi,j send a from pi,j+1 and b from pi+1,j;
}
```

Implementing n-dimensional rhotrices in parallel requires $n \times n$ processing units, which will run in $O(n)$ times.

3.11. Row-column Multiplication on Master-Worker Platform (Coupled Matrix)

This section is aimed at selecting a scalable platform for our proposed parallel row-column rhotrix-multiplication algorithm implementable on clusters of workstations. The algorithm is devised to handle resource selection (deciding which workers to enrol) and communication ordering (both for input and result messages). In the next chapter we present an MPI implementation of the parallel algorithm based on this platform.

In this research work, we do not intend to limit our work only to the 2D processor grid strategy to networks of workstations. Instead, we adapt a realistic application scenario, where input files are read from a fixed repository (disk on a data server). Computations will be delegated to available resources in the target architecture, and results will be returned to the repository. This calls for a master-worker paradigm or more precisely for a computational

scheme where the master (the processor holding the input data) assigns computations to other resource, the workers. In this centralised approach, all rhotrix files originate from, and must be returned to the master. The master distributes both data and computations to the worker.

3.11.1. Application Process and Input Data Communication

We deal with the computational kernel $C \leftarrow C + RQ$ stated in the form

$$C_{i,j} = \beta \sum_{k=1}^n a_{i,k} \times b_{2k-1,j} + (1-\beta) \sum_{k=1}^{n-1} a_{i,k} \times b_{2k,j},$$

where β denote rhotrix row index assuming value odd or even ie $\beta = 0$ and $\beta = 1$. We partition the three rhotrices R , Q , and C as illustrated in Figure 3.11a. More precisely:

a. We use a block-oriented approach (reason being for the fact that we can actually divide a rhotrix into blocks of main entries and block of hearts as a set of separate data elements). The atomic elements that we manipulate are not rhotrix coefficients but instead square blocks of size $q \times q$ (hence with q^2 coefficients).

b. The input Rhotrix C is of size $n_C \times n_{RQ}$:

- we split R into r horizontal stripes A_i , $1 \leq i \leq r$, where $r = n_C / q$;
- we split each stripe $R(A_i)$ into t square $q \times q$ blocks $R(A_{i,k})$, $1 \leq k \leq t$, where $t = n_{RQ} / q$.

c. Again we consider the input rhotrix Q of size $n_{R(A)Q(B)} \times n_{Q(B)}$:

- we split B into s vertical stripes R_j , $1 \leq j \leq s$, where $s = n_Q / q$;
- we split stripe $Q(B_j)$ into t square $q \times q$ blocks $Q_{2k-1,j} | Q_{2k,j}$, $1 \leq k \leq t$.

d. We compute $C_{i,j} = \beta \sum_{k=1}^n a_{i,k} \times b_{2k-1,j} + (1-\beta) \sum_{k=1}^{n-1} a_{i,k} \times b_{2k,j}$ for $\beta = 0$ or $\beta = 1$. Rhotrix C is

accessed (both for input and output) by square $q \times q$ blocks $C_{i,j}$, $1 \leq i \leq r$, $1 \leq j \leq s$. There are $r \times s$ such blocks. We point out that with such a decomposition all stripes and blocks have same size. This will greatly simplify the analysis of communication costs.

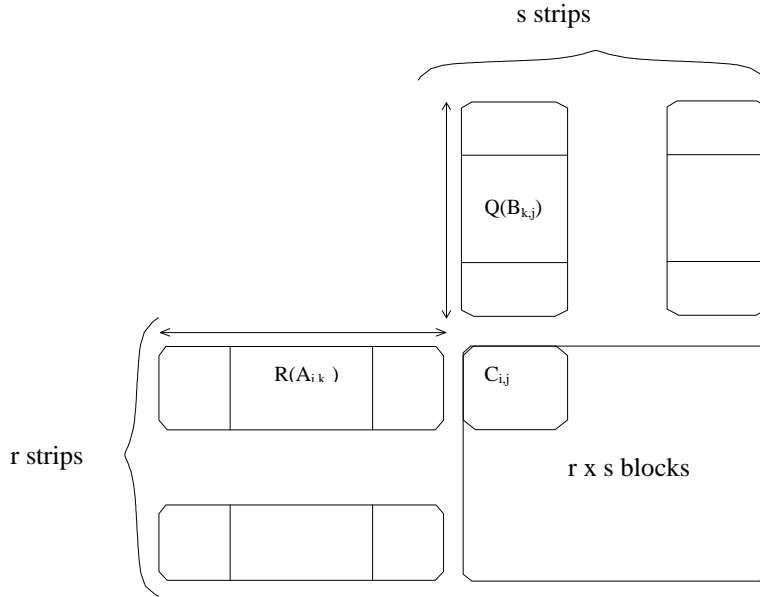


Figure 3.11a Partition of the three matrices R, Q and C

3.11.2. Master-Worker Network Topology Platform

We target a heterogeneous network topology comprising of clusters of workstations $D = \{P_0, P_1, P_2, \dots, P_p\}$, composed of a master P_0 and of p identical workers P_i , $1 \leq i \leq p$. Because we manipulate large data blocks, we adopt a linear cost model, both for computations and communications (i.e., we neglect start-up overheads). We have the following notations:

- It takes $N.w_i$ time-units to execute a task of size N on P_i ;
- It takes $N.c_i$ time-units for the master P_0 to send a message of size N to P_i or to receive a message of size N from P_i .

A fully homogeneous network platform would be a platform with identical workers and identical communication links: $w_i = w$ and $c_i = c$ for each worker P_i , $1 \leq i \leq p$. Without loss of generality, we assume that the master has no processing capability (otherwise, add a fictitious extra worker paying no communication cost to simulate computation at the master). Next, we need to define the communication model. We adopt the one-port model (Bhat and *et al.*, 1999) and (Bhat and *et al.*, 2003), which is defined as follows: (i) the master can only send data to, and receive data from, a single worker at a given time-step; (ii) a given worker cannot start execution before it has terminated the reception of the message from the master; similarly, it cannot start sending the results back to the master before finishing the computation. In fact, this one-port model naturally comes in two flavours,

depending upon whether we allow the master to simultaneously send and receive messages or not. If we do allow for simultaneous sends and receives, we have actually the two-port model. Here we concentrate on the true one-port model, where the master cannot be enrolled in more than one communication at any time-step. The one-port model is realistic. Bhat, Raghavendra, and Prasanna (2003) advocate its use because “current hardware and software do not easily enable multiple messages to be transmitted simultaneously.” Even if non-blocking multi-threaded communication libraries allow for initiating multiple send and receive operations, they claim that all these operations “are eventually serialized by the single hardware port to the network.” Experimental evidence of this fact has recently been reported by (Saif and Parashar, 2004), who report that asynchronous MPI sends get serialized as soon as message sizes exceed a hundred kilobytes. Their result holds for two popular MPI implementations, MPICH on Linux clusters and IBM MPI on the SP2. Note that all the MPI experiments in Chapter 4 obey the one-port model.

Our final assumption is related to memory capacity; we assume that a worker P_i can only store m blocks (either from R , Q , or C). For large problems, this memory limitation will considerably impact the design of the algorithms, as data reuse will be greatly dependent on the amount of available buffers.

3.11.3. Minimization of the Communication Volume

In this section, we derive a lower bound on the total number of communications (sent from, or received by, the master) that are needed to execute any rhotrix multiplication algorithm satisfying our hypotheses (centralized data and limited memory). Since we aim at minimizing the total communication volume, we can simulate any parallel algorithm on a single worker. Therefore, we only need to consider the one-worker case. We deal with the original, and realistic, formulation of the problem as follows:

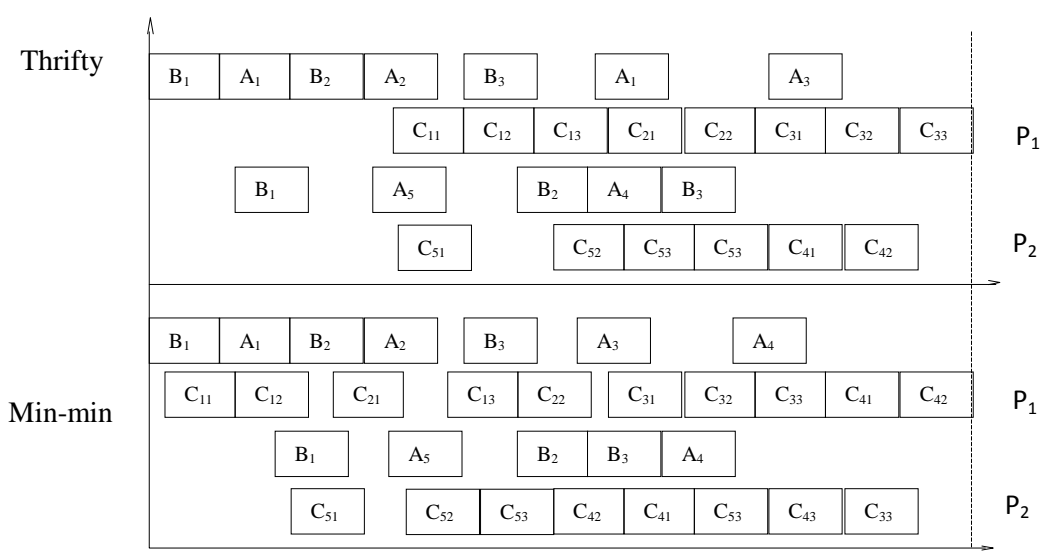


Figure 3.11b Example showing that Min-min is not optimal: with $p = 2$, $c = 8$, $w = 9$, $r = 6$, and $s = 3$, Thrifty has a lower makespan.

We consider two greedy algorithm for p workers:

Thrifty: This algorithm “spares” resource as it aims at keeping each enrolled worker fully active. It work as follows:

- Send enough blocks to the first worker so that it is never idle,
- Send blocks to a second worker during spare communication slots, and
- Enroll a new worker (and send block to it) only if this does not delay previously enrolled workers

Min-min: This algorithm is based on the well known min-min heuristic (Maheswaran *et al.*, 1999). At each step, all tasks are considered. For each of them, we compute their possible starting time on each worker, given files that have already been sent to this worker and all decisions taken previously: we select the best worker, hence the first min in the heuristic. We take the minimum of starting time over all tasks, hence the second min. It turns out that neither greedy algorithm is optimal. This further illustrated in Figure 3.11b above.

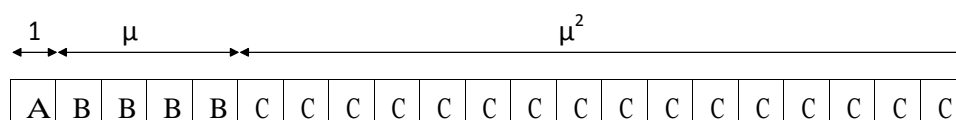


Figure 3.11c Memory usage for the maximum reuse algorithm when $m = 21$: $\mu = 4$; 1 block is used for A, μ for B, and μ^2 for C.

- The master sends blocks $R(A_{ik})$, $R(B_{2k-1,j})$ | $R(B_{2k,j})$, and C_{ij} ,
- The master retrieves final values of blocks C_{ij} , and
- We enforce limited memory on the worker; only m buffers are available, which means that at most m blocks of $R(A)$, $Q(B)$, and/or C can simultaneously be stored on the worker.

First, we describe an algorithm that aims at reusing R blocks as much as possible after they have been loaded. Next, we assess the performance of this algorithm.

3.11.4. The Maximum Re-use Algorithm

Below we introduce and analyze the performance of the maximum re-use algorithm, whose memory management is illustrated in Figure 3.11c. Four consecutive execution steps are shown in Figure 3.11d. Assume that there are m available buffers. First we find μ as the largest integer such that $1 + \mu + \mu^2 \leq m$. The idea is to use one buffer to store $R(A)$ blocks, μ buffers to store $Q(B)$ blocks, and μ^2 buffers to store C blocks.

	B11	B12	B13	B14
A11	C11	C12	C13	C14
	C21	C22	C23	C24
	C31	C32	C33	C34
	C41	C42	C43	C44

	B11	B12	B13	B14
	R11	R12	R13	R14
A21	C21	C22	C23	C24
	C31	C32	C33	C34
	C41	C42	C43	C44

	B11	B12	B13	B14
	C11	C12	C13	C14
	C21	C22	C23	C24
A31	C31	C32	C33	C34
	C41	C42	C43	C44

	B11	B12	B13	B14
	C11	C12	C13	C14
	C21	C22	C23	C24
	C31	C32	C33	C34
A41	C41	C42	C43	C44

Figure 3.11d Four steps of the maximum re-use algorithm, with $m = 21$ and $\mu = 4$. The elements of C updated are displayed in white on black.

In the outer loop of the algorithm, a $\mu \times \mu$ square of C blocks is loaded. Once these μ^2 blocks have been loaded, they are repeatedly updated in the inner loop of the algorithm until their final value is computed. Then the blocks are returned to the master, and μ^2 new C blocks are sent by the master and stored by the worker. As illustrated in Figure 3.11c, we need μ buffers to store a row of $Q(B)$ blocks, but only one buffer for $R(A)$ blocks: $R(A)$ blocks are sent in sequence, each of them is used in combination with a row of μB blocks to update the corresponding row of C blocks. This leads to the following sketch of the algorithm:

Outer loop: while there remain C blocks to be computed

- Store μ^2 blocks of C in worker's memory:

send a $\mu \times \mu$ square $\{C_{i,j} / i_0 \leq i < i_0 + \mu, j_0 \leq j < j_0 + \mu\}$

- Inner loop: For each k from 1 to t :
 1. Send a row of μ elements $\{R(B_{2k-1,j}) \mid Q(B_{2k,j}) / j_0 \leq j < j_0 + \mu\}$;
 2. Sequentially send μ elements of column $\{R(A_{i,k}) / i_0 \leq i < i_0 + \mu\}$. For each $R(A_{i,k})$, update μ elements of C
- Return results to master.

3.12. Algorithms for Master-worker Platforms (Coupled Matrix)

In this section, we adapt the maximum re-use algorithm to fully homogeneous platforms. We must first decide which part of the memory will be used to stock which part of the original rhotrix, in order to maximize the total number of computations per time unit. Relative to Cannon's algorithm (Cannon, 1969) and the ScaLAPACK outer product algorithm (Blackford *et al.*, 1999) both distribute square blocks of matrix product C to the processors, likewise, we could equally do so with resultant rhotrix C . Intuitively, squares are better than elongated rectangles because their perimeter (which is proportional to the communication volume) is smaller for the same area. We use the same approach here, but we have not been able to assess any optimal result.

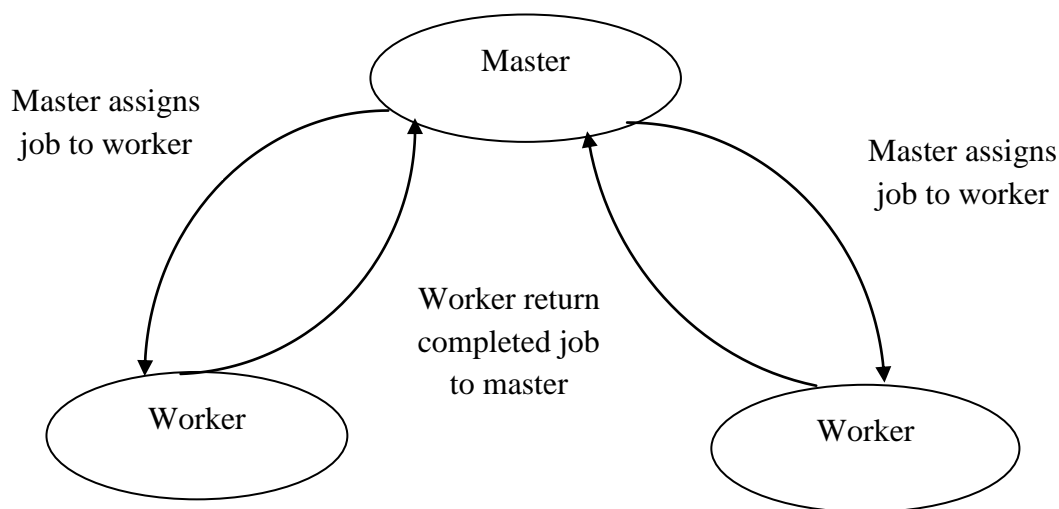


Figure 3.12: Task/Channel Graph of the Algorithm

The master interacts with each worker and on the process the worker initiates a communication cycle by sending a message to the master indicating its readiness to start work.

Manager Pseudocode:

Identify n jobs in host system or user-specified directory

Receive size k , the number of elements in each job

Allocate $n \times k$ matrix to store job vector

Repeat

Send message or task to worker

Receive message or result from worker

if message contains result

store result

endif

if job remains then send worker task

else Send worker termination message

endif

until all workers terminated

Worker Pseudocode:

Repeat

Send first request to master

Receive task from master

Perform computation on task

Send result back to master

forever

3.12.1. Principle of the Algorithm Design

We load into the memory of each worker $\mu \times q$ blocks of R and $\mu \times q$ blocks of Q to compute $\mu^2 \times q$ blocks of C. In addition, we need 2μ extra buffers, split into μ buffers for R and μ for Q, in order to overlap computation and communication steps. In fact, μ buffers for R and μ for Q would suffice for each update, but we need to prepare for the next update while computing. Overall, the number of C blocks that we can simultaneously load into memory is the largest integer μ such that $\mu^2 + 4\mu \leq m$.

We have to determine the number of participating workers P. For that purpose, we proceed as follows. On the communication side, we know that in a round (computing a C block entirely), the master exchanges with each worker $2\mu^2$ blocks of R (μ^2 sent and μ^2 received), and sends μt blocks of R and μt blocks of Q. Also during this round, on the computation side, each worker computes $\mu^2 t$ block updates. If we enrol too many processors, the

communication capacity of the master will be exceeded. There is a limit on the number of blocks sent per time unit, hence on the maximal processor number \mathcal{W}_n , which we compute as follows: \mathcal{W}_n is the smallest integer such that $2\mu c \times \mathcal{W}_n \geq \mu^2 tw$.

Indeed, this is the smallest value to saturate the communication capacity of the master required to sustain the corresponding computations. We derive that

$$W_n = \left\lceil \frac{\mu^2 tw}{2c\mu t} \right\rceil = \left\lceil \frac{\mu w}{2c} \right\rceil \quad (3.12)$$

In the context of rhotrix multiplication, we have $c = q^2 \tau_c$ and $w = q^3 \tau_a$, where τ_c and τ_a respectively represent the speed of the communication link and the speed of the processor.

Hence $\left\lceil \frac{\mu q \tau_a}{2 \tau_c} \right\rceil$. Moreover, we need to enforce that $\mathcal{W}_n \leq p$, hence we finally obtain $\mathcal{W}_n =$

$$\min \left\{ p, \left\lceil \frac{\mu q \tau_a}{2 \tau_c} \right\rceil \right\}.$$

For the sake of simplicity, we suppose that r is divisible by μ , and that s is divisible by $\mathcal{W}_n \mu$. We allocate μ block columns (i.e., $q\mu$ consecutive columns of the original matrix) of R to each processor. The algorithm is decomposed into two parts. The first algorithm outlines the program of the master, while the second algorithm is the program of each worker.

Algorithm 10: Master algorithm.

Read rhotrix elements from file

Allocate memory to buffer

Split the rhotrix into squares $R_{irow, icol}$ of μ^2 blocks (of size $q \times q$):

$R_{irow, icol} = \{ R_{irow, icol} \mid (irow - 1)\mu + 1 \leq i \leq irow \times \mu, (icol - 1)\mu + 1 \leq j \leq icol \times \mu \};$

for icol' $\leftarrow 0$ to $\frac{s}{\mathcal{W}_n \mu}$ by step \mathcal{W}_n do

for irow' $\leftarrow 1$ to $\frac{r}{\mu}$ do

for id_{worker} $\leftarrow 1$ to \mathcal{W}_n do

icol $\leftarrow icol' + id_{worker}$;

send block $R_{irow, icol}$ to worker id_{worker} ;

```

endfor
for k ← 1 to t do
    for idworker ← 1 to Wn do
        icol ← icol' + idworker;
        for j ← (icol - 1) μ + 1 to icol x μ do
            send R(b2k-1,j) / R(b2k,j);
        endfor
        for i ← (irow - 1) μ + 1 to irow x μ do
            send R(ai,k);
        endfor
    endfor
endfor

for idworker ← 1 to Wn do
    icol ← icol' + idworker;
    receive Rirow, icol from worker idworker;
endfor

endfor
endfor

```

Algorithm 11: Worker Algorithm.

```

for all blocks do
    Receive Rirow, icol from master;
    for k ← 1 to t do
        for j ← (icol - 1) μ + 1 to icol x μ do receive R(b2k-1,j) / R(b2k,j);
        for i ← (irow - 1) μ + 1 to irow x μ do receive R(ai,k);
        for j ← (icol - 1) μ + 1 to icol x μ do
            if (i mod 2 ≠ 0) {
                R(ci,k) ← R(ci,j) + R(ai,k) x R(b2k-1,j)
            }
            else{
                R(ci,k) ← R(ci,j) + R(ai,k) x R(b2k,j)
            }
        endfor
    endfor
endfor

```

```

        }
    endfor
endfor
endifor
return  $R_{irow, icol}$  to master;
endifor

```

3.13. Program Task Graph for Heart-Oriented Rhotrix Algorithm

The heart-oriented rhotrix algorithm can easily be represented in a task graph. Similar graph model was initially presented in Section 3.5.4 for Strassen's algorithm, hence, we only present structural representation of similar graph for the heart-oriented rhotrix multiplication. Figure 3.13 illustrates the corresponding task graph which has been simplified into the following two stages of computation.

Let consider the following rhotrices

$$R_3 = \left\langle \begin{array}{ccc} & a_1 & \\ a_2 & a_3 & a_4 \\ & a_5 & \end{array} \right\rangle \text{ and } Q_3 = \left\langle \begin{array}{ccc} & b_1 & \\ b_2 & b_3 & b_4 \\ & b_5 & \end{array} \right\rangle$$

The computation can be expressed into

Stage one:

$$\begin{aligned}
 A_1 &= b_3 a_1 & B_1 &= a_3 b_1 \\
 A_2 &= b_3 a_2 & B_2 &= a_3 b_2 \\
 & & H_3 &= a_3 b_3 \\
 A_4 &= b_3 a_4 & B_4 &= a_3 b_4 \\
 A_5 &= b_3 a_5 & B_5 &= a_3 b_5
 \end{aligned}$$

Stage two:

$$\begin{aligned}
 C_1 &= A_1 + B_1 \\
 C_2 &= A_2 + B_2 \\
 C_3 &= H \\
 C_4 &= A_4 + B_4 \\
 C_5 &= A_5 + B_5
 \end{aligned}$$

We can now apply the above task notation to produce a simplified task graph as shown below.

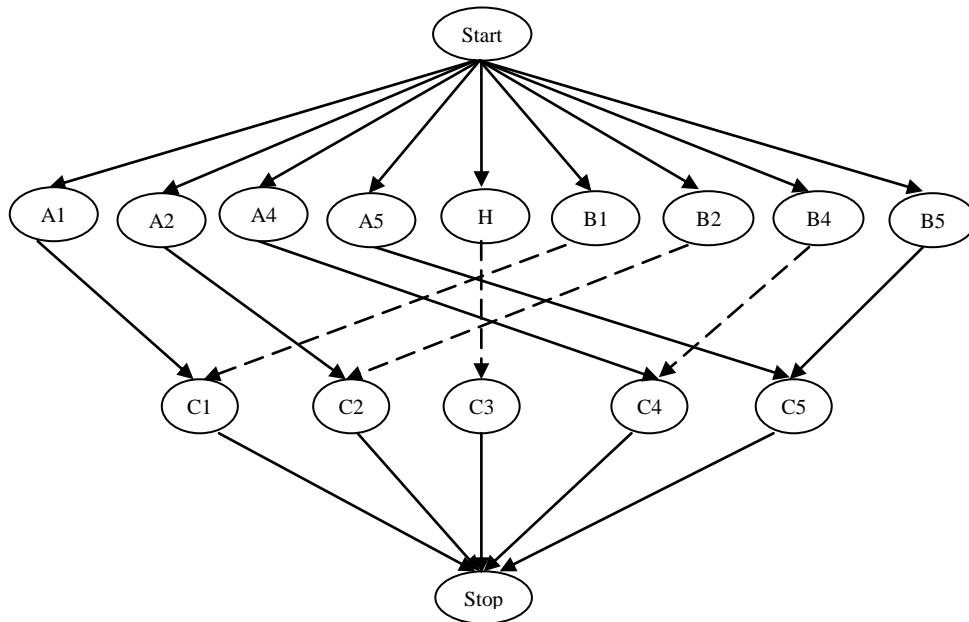


Figure 3.13: Task Graph for Heart-Oriented Rhotrix Algorithm

Algorithm 12: Master-Worker algorithm for heart-oriented rhotrix multiplication.

$ha \leftarrow x$

$hb \leftarrow y$

Part One

find out if I am MASTER or WORKER

if I am MASTER

offset \leftarrow 0; active \leftarrow 0;

for $i = 1$ to p do

read current block of the rhotrices A_i and B_i of size sb

elements starting from the position offset of file;

send current block of the rhotrices A_i and B_i to worker P_i ;

active \leftarrow active + 1;

offset \leftarrow offset + sb ;

endfor

```

do until no more jobs
  receive results from WORKER  $\rightarrow R_i$ 
  active  $\leftarrow$  active - 1;
  sender  $\leftarrow P_{anyworker}$ ;

if offset is not at end of the rhotrix files then
  read current block of the rhotrix a and b of size sb
  elements starting from the position offset of file;
  send current block of the rhotrix a and b to worker
   $P_{sender}$ ;
  active  $\leftarrow$  active + 1;
  offset  $\leftarrow$  offset + sb;
endif
end do

```

```

send a terminator message to worker  $P_{sender}$ ;
print global result;

```

Part Two

```

else if I am WORKER
  do until no more jobs
    receive from MASTER next job
    if there is terminator message then break;
    calculate  $C(i) \leftarrow Rhotrix\_Mult(block\_a, block\_b)$ 
    send  $C(i)$  to MASTER
  end do

```

```

begin Rhotrix_Mult (block_a, block_b)
for i  $\leftarrow$  0 to n
  if  $i := (1/2(n^2+1)+1)/2$ 
     $C[i] = ha \times hb$ 
  else
     $C[i] = a[i] \times hb + b[i] \times ha$ 

```

```

    endif
return (ans[i])
end FCN(input)
endif

```

3.14. Memory Usage

As shown in Figure 3.17 and Table 3.17, process P0 needs more memory than every other process, since it stores all the blocks of rhotrix entries and computed results. While processes P1, P2, P4 and P5 stores a complete set of rhotrix entries comprising main entries and the heart element required to perform a unit set of computation. Process P3 only stores the heart elements. We consider the memory cost required by process P0-P5. We noticed that the memory usage of processes P1, P2, P4 and P5 are identical in sizes. Each of them needs $4 \times \frac{1}{2}(n^2 + 1)$ memory elements to store two main entries and two heart entries from the two rhotrices. Process P3 needs $2 \times \frac{1}{2}(n^2 + 1)$ and process P0 needs $10 \times \frac{1}{2}(n^2 + 1)$ elements of memory. Thus considering the space to store the two rhotrices R and Q entries, the total, amount of memory requirement is equal to $8(n^2 + 1)$. If we denote P0 to be the master processor, then, its memory requirement will be $\lfloor n^2 + 1 \rfloor$ while those of the slaves processes will have $\left\lceil \frac{n^2 + 1}{p} + 2 \right\rceil$ each. The number 2 stands for heart of first and second rhotrices. Thus, in general we have $\left\lceil \frac{n^2 + 2p + 1}{p} \right\rceil$ as the memory usage for the slave processes.

3.15. Heart-Oriented MPI Experimental Performance Model

In this section we present a time prediction models for the heart-oriented rhotrix algorithm implementations discussed in Section 3.10. We give the execution time for each step of the algorithm implementation. Therefore, the execution time can be broken up into four terms:

- i. **T_I** : It is the total I/O time to read the elements of the vectors **a** and **b** into several blocks of size $s \times b \times \text{sizeof}(\text{int})$ bytes from the local disk of the master processor. The **sb** is the number of elements of the block. Therefore, the master reads $2 \times n \times \text{sizeof}(\text{int})$ bytes totally of the vectors **a** and **b** . Then, the time **T_I** is given by:

$$T_1 = \frac{2 \times n \times \text{sizeof}(\text{int})}{(V_{I/O})_{\text{master}}} \quad (3.15.1)$$

where $(V_{I/O})_{\text{master}}$ is the I/O speed of the master workstation.

- ii. T_2 : It is the total communication time to send all blocks of the vectors a and b to all workers. The size of each block is $sb \times sizeof(int)$ bytes. Therefore, the master sends $2 \times n \times sizeof(int)$ bytes totally. Then, the time T_2 is given by:

$$T_2 = \frac{2 \times n \times sizeof(int)}{V_{comm}} \quad (3.15.2)$$

where V_{comm} is the communication speed

- iii. T_3 : It is the average computation time across the cluster. Each worker performs a dot product between the block of the vector a and the block of the vector b with size $sb \times sizeof(int)$ bytes. It requires sb steps. Then, the time T_3 is given by:

$$T_3 = \frac{\left(\left\lceil \frac{n}{sb} \right\rceil - p \right) (sb \times sizeof(int))}{\sum_{j=1}^p (V_{comp})_j} + \max_{j=1}^p \left\{ \frac{sb \times sizeof(int)}{(V_{comp})_j} \right\} \quad (3.15.3)$$

where $\sum_{j=1}^p (V_{comp})_j$ is the computation capacity of the cluster (homogeneous or heterogeneous) when p processors are used. We include the second max term in the equation (3.10.3) which defines the worse load imbalance at the end of the execution when there are not enough blocks of the rhatrix main entries left to keep all the processors busy.

- iv. T_4 : It includes the communication time to receive $\left\lceil \frac{n}{sb} \right\rceil$ results from all workers. Each worker sends the local sum back $1 \times sizeof(int)$ bytes. Therefore, the master will receive $\left\lceil \frac{n}{sb} \right\rceil \times sizeof(int)$ bytes totally. Therefore, the time T_4 is given by:

$$T_4 = \frac{\left\lceil \frac{n}{sb} \right\rceil \times sizeof(int)}{V_{comm}} \quad (3.15.4)$$

where V_{comm} is the communication speed

Further, we note that for this dynamic rhatrix implementation in practice there is parallel communication and computation and for this reason we take the maximum value between the communication time i.e. $T_2 + T_4$ and the computation time i.e. T_3 .

Therefore, the total execution time of our rhotrix product implementation, T_p , using p processors, is given by:

$$T_p = T_1 + \max\{T_2 + T_4, T_3\} \quad (3.15.5)$$

3.16. Heart-Oriented Scalability Analysis Performance Model

The scalability analysis of our parallel algorithm is considered by examining how the program execution time and speedup varies with some of the specified program parameters such as rhotrix size, order and number of processors exploited during the program run.

a. The parallel program execution time T_e can be examined within the following time components

- Computation time T_{comp} : it is the time a single processor (worker p_i) spends doing its part of a computation. It depends on the problem size n and specifics of the processor p . Ideally, this is just T_{serial}/P

$$T_{comp} = T_{serial}/P \quad (3.16.1)$$

- Idle time T_{idle} : When a processor is not computing or communicating, it is idle. In our parallel algorithms we tried to minimize a processor's idle time by ensuring proper load balancing and efficient coordination of processor computation and communication.
- Communication time T_{comm} : it is the time it takes for processes to send and receive messages from master to worker and vice versa. This time is usually broken up into two parts,

$$T_{comm} = T_1 + T_m. \quad (3.16.2)$$

- The first part is the time associated with initializing the communication and is called the latency, T_1 .
- The second part is the time it takes to send a message of length m , T_m . T_m is given by m/B where B is the physical bandwidth of the channel (usually given in megabytes per second).
- We assume that communication cannot be overlapped with other operations, thus, $T_{comm} = N_{messages} \times (T_1 + \langle m \rangle / B)$ where $\langle m \rangle$ is the average message size and $N_{messages}$ is the number of messages required by our algorithm.

The execution time, T_e , is given by, $T_e = T_{\text{comp}} + T_{\text{comm}} + T_{\text{idle}}$, where the execution time is divided between computing, communicating, or sitting idle, respectively as explained above.

- b. The relative parallel algorithm Speedup T_{speedup} is defined as the execution time on one processor over the execution time on P processors. Absolute speedup is obtained by replacing the execution time on one processor with the execution time of the fastest sequential algorithm and it is given as:

$$\text{Speed-Up Factor} = \frac{\text{Processing Time in a Single Processor}}{\text{Processing Time in the Array Processor}} \quad (3.16.3)$$

3.17. Task-Parallel Approach for Heart-Oriented Algorithm Using MPI

We consider a very simple method of parallelizing the heart-oriented algorithm. Based on the algorithm, there exist two kind of rhotrix multiplications (i.e., rhotrix heart by main entries and heart by heart multiplication). For instance, consider three dimensional rhotrices and employ six processes to compute the multiplications, the program design becomes very simple. The basic idea is that we divide the computation into five parts where each part is being handled by a process. Figure 3.13 above is a task/channel graph for the heart-oriented algorithm while Figure 3.17 depicts the computation partitioning across six processes. Again, based on this logical task partitioning, we can easily write an MPI program to implement our program design. For example, process P0 corresponding to the red node performs the task of data distribution and receives computed results from each processes. Processes P1, P2, P4 and P5 perform the task of heart to main entries computation, while process P3 corresponding to the dark red node performs the task of heart to heart computation. Please note that we can equally choose to assign computation to process P0, either ways, follows.

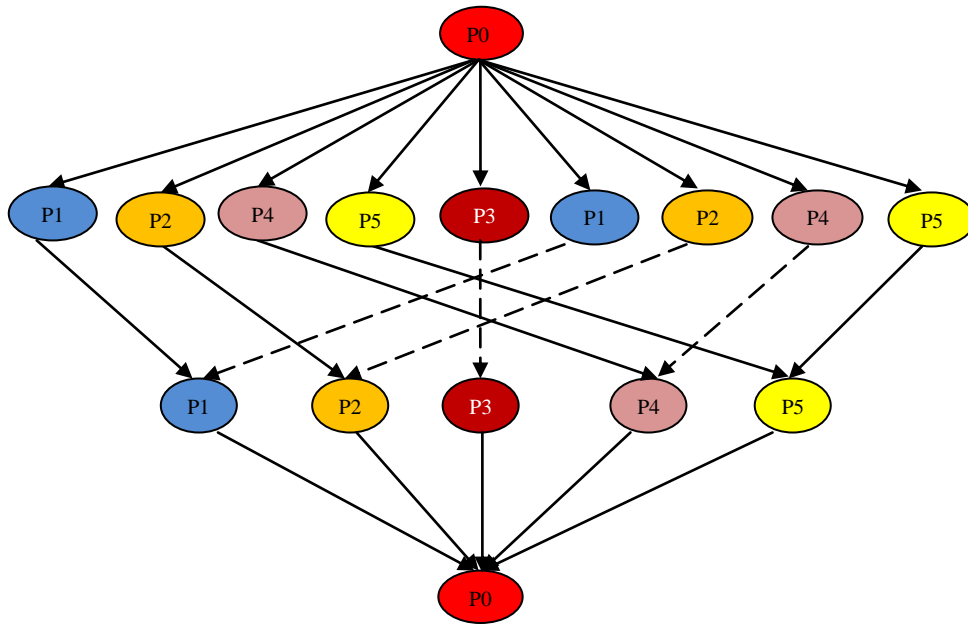


Fig 3.17 Using six MPI processes to implement the heart-oriented rhotrix multiplication algorithm. Nodes with the same colour belong to a single process.

Table 3.17a Heart-oriented data distribution among processes

P0	$a_1, a_2, a_3, a_4, a_5, b_1, b_2, b_3, b_4, b_5$
P1	a_1, b_1, a_3, b_3
P2	a_2, b_2, a_3, b_3
P3	a_3, b_3
P4	a_4, b_4, a_3, b_3
P5	a_5, b_5, a_3, b_3
P0	c_1, c_2, c_3, c_4, c_5

The dynamic allocation of the main entries is based on the following assumptions: First, the number of workstations in the cluster is denoted by p and we assume that p is power of 2. First, the workstations are numbered from 1 to p . Second, we load into the local disk of the master workstation P0 all main entries (or blocks) of the rhotrix main entries a and b . The algorithm of the heart-oriented rhotrix multiplication comprises of two phase: the first phase outlines the program of the master, while second phase is the program of each worker. We must note that the *offset* is a pointer that shows the current position in the file. Further, the master terminates when there are not any blocks of the rhotrix main entries a and b . Note that

in order to terminate, the number of tasks outstanding in the workers is counted (*active*). It is also possible simply to count the number of results returned as shown in Table 3.17b.

Table 3.17b Heart-oriented computation among processes

P0	$a_1, a_2, a_3, a_4, a_5, b_1, b_2, b_3, b_4, b_5$
P1	$A1 = [b_3 \times a_1]$
P2	$A2 = [b_3 \times a_2]$
P4	$A4 = [b_3 \times a_4]$
P5	$A5 = [b_3 \times a_5]$
P3	$H = [a_3 \times b_3]$
P1	$B1 = [a_3 \times b_1]$
P2	$B2 = [a_3 \times b_2]$
P4	$B4 = [a_3 \times b_4]$
P5	$B5 = [a_3 \times b_5]$
P1	$C1 = A1 + B1$
P2	$C2 = A2 + B2$
P3	$C3 = H$
P4	$C4 = A4 + B4$
P5	$C5 = A5 + B5$
P0	$C1, C2, C3, C4, C5$

3.18. A Framework of Task-Parallel Approach for Strassen's Algorithm Using MPI

We can use a straightforward method to parallelize Strassen's algorithm. Based on the algorithm, there exist seven matrix multiplications (i.e., P1-P7). If we employ seven processes to compute the multiplications, the program design becomes very simple. The basic idea is that we divide the task graph in Fig. 3.5a into seven parts and each part is handled by a process. Fig. 3.18 depicts the task partitioning across the seven processes. Nodes with the same colour belong to a single process. It is important to note that the positions of nodes in Fig. 3.18 are identical to those in Fig. 3.5a where computations are carried out for step 1 and 2 of S and T, step 3 of U and step 4 of R. Since Figure 3.18 directly reflects our program design model, it is straightforward to write an MPI program based on the graph. For instance,

process P3 performs computations of S1, T1, U4 and R21 which correspond to the brown nodes on the right.

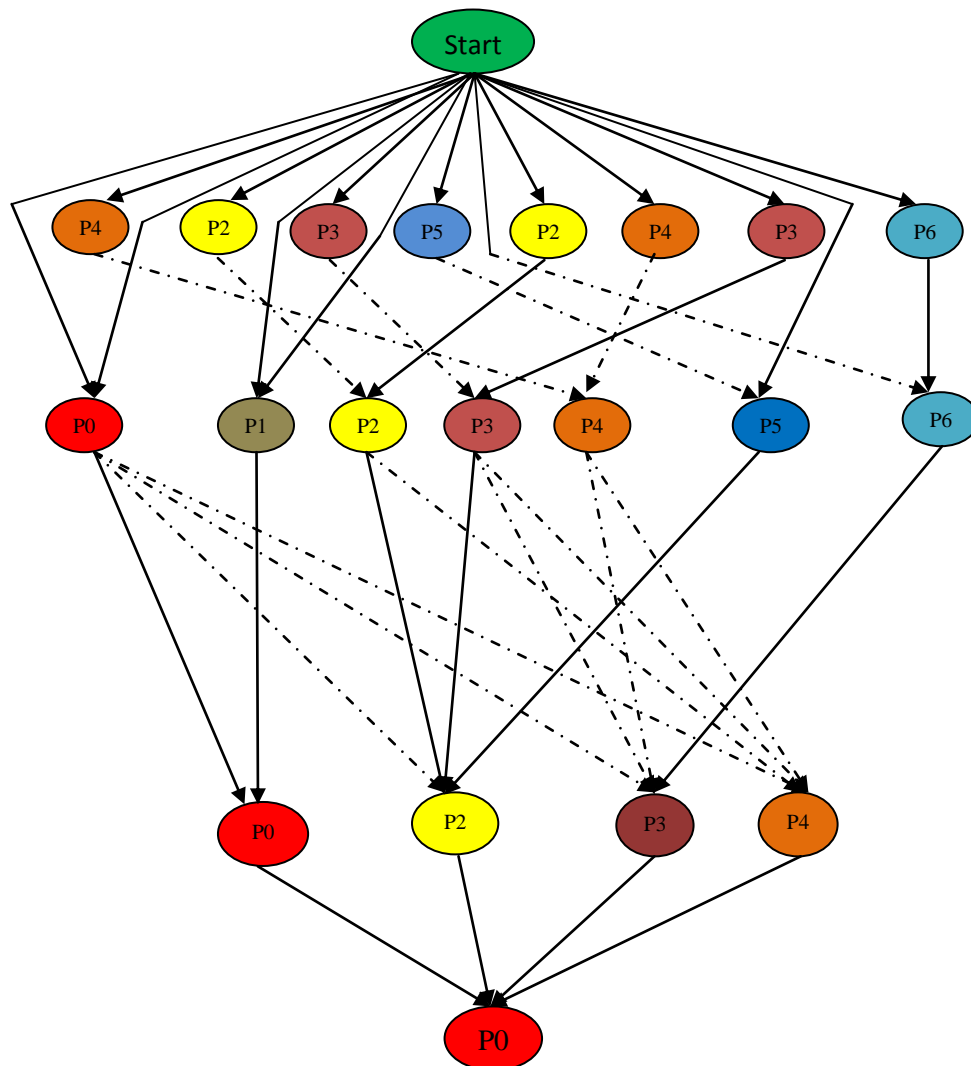


Figure 3.18: A model of MPI task processes graph to implement Strassen's algorithm

3.19. Data Structures

In this section we will discuss on the different ways of storing matrices, that is, where should an element be placed in memory. Naturally, when storing a two-dimensional objects such as the double indices (row-column) matrix into a one-dimensional objects such as the memory, we are facing the problem of choosing a mapping of the different set of coordinates to each other. Traditionally, there has been two alternatives, store by row or store by column. For instance in Fortran, the column-major order is used, in which columns are stored after each other, that is, we are going by stride 1 (the distance between subsequent accesses) for increasing row coordinates, see Figure 3.19a and 3.19b. In order to calculate the memory

address for an element in a row-column rhotrix we need the number of rows for the rhotrix.

The address is then

$$add(A, lda, r, c) = A + (r - 1) + (c - 1).lda \quad (3.19.1)$$

Where A is the address of the first element in A and lda is the number of rows, or leading dimension, of A . r and c are one-based, that is, they start numbering from 1, if we are going along a row, we are going by stride lda . For example, let consider how a five dimensional rhotrix is represented in a computer memory.

$$R_5 = \left\langle \begin{array}{cccc} & & a_{11} & & \\ & a_{31} & a_{21} & a_{12} & \\ a_{51} & a_{41} & a_{32} & a_{22} & a_{13} \\ & a_{52} & a_{42} & a_{33} & \\ & & a_{53} & & \end{array} \right\rangle$$

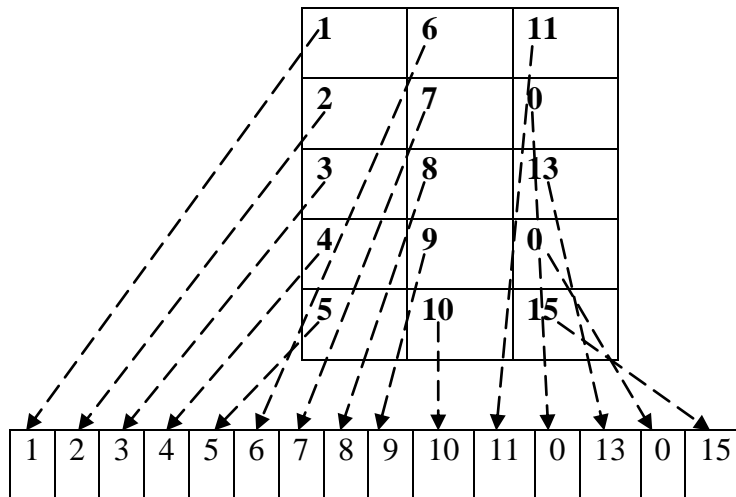


Figure 3.19a: Fortran mapping of double indices rhotrix to single array memory

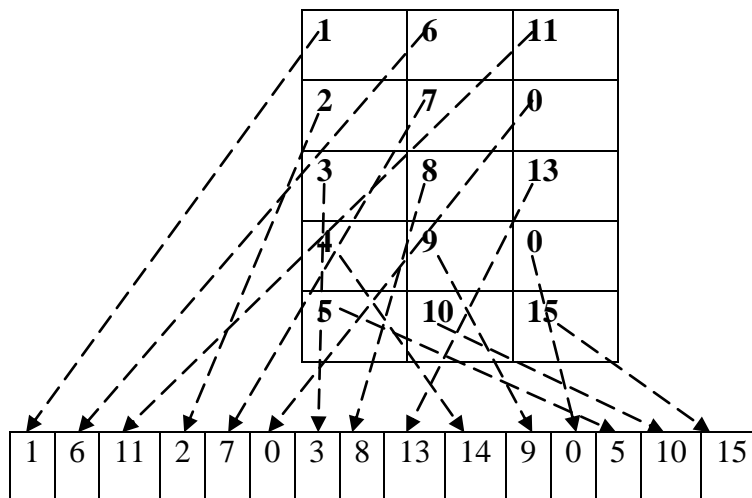


Figure 3.19b: C's way of mapping the rhotrix to memory

In C, the rows are stored after each order, that is, we are going by stride 1 if we follow the same row. In order to calculate the memory address for an element in a double indices rhotrix stored in the C way, we need the number of columns for the rhotrix, *lda*.

$$\text{address}(A, \text{lad}, r, c) = A + (r - 1) \bullet \text{lda} + (c - 1) \quad 3.19.2$$

We can also make use of the column-major matrices for row-column rhotrices in C, by using the pre-processor example and calculating the offset manually. The following example shows how to use the pre-processor to hide explicit offset calculation.

```

Void fillRhotrix(double *A, int lda, int r, int c, double value)
#define A(r,c) (A[(r)*lda + (c)])
{
    int i, j;

    for (i = 0; i < r; i++)
        for (j = 0; j < r; j++)
            A(i,j) = value;      /* Expands to A[i * lda + j] = value */
}

#undef A      /* If we need another A somewhere else */

```

Also shown in Figure 3.19c and 3.19d are the mapping of rhotrix elements to computer memory, but in this case we considered the heart-oriented single indices data-to-memory mapping. The elements are mapped in row-major order as in C.

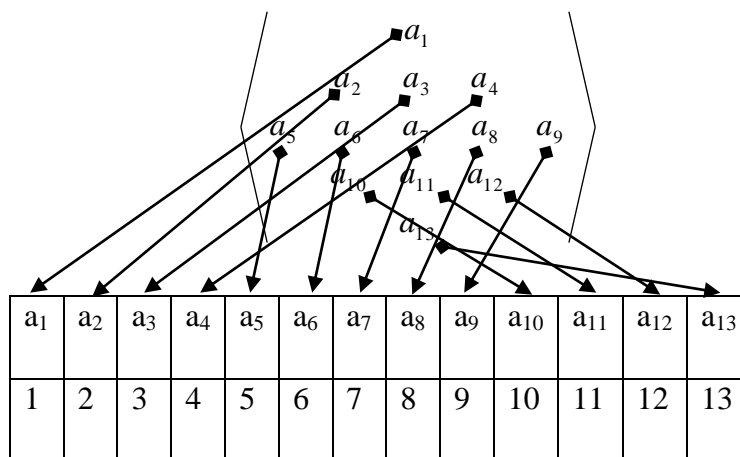


Figure 3.19c: C's way of mapping rhotrix entries to memory

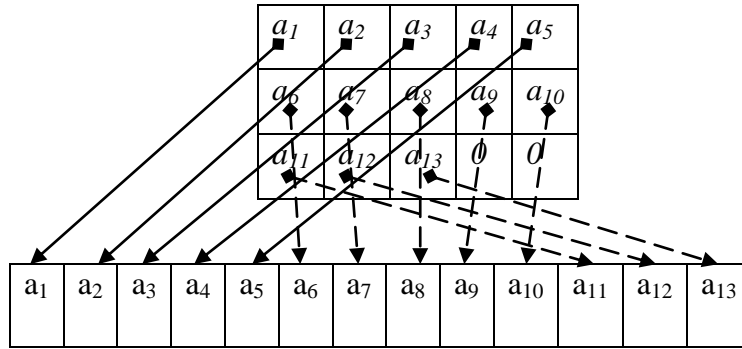


Figure 3.19d C's way of mapping the rhotrix to memory (blocked)

3.20. Computational Time Complexity Analysis for Rhotrix Multiplication

In this section, we establish an optimal way of multiplying rhotrices, a fundamental operation that is a bottleneck for many important algorithms. Faster rhotrix multiplication would give more efficient algorithms for many standard linear algebraic problems that will in the nearest future be driven from rhotrices, such as inverting rhotrices or simply put the inverse of a rhotrix, solving systems of linear equations, and finding determinants. Even some basic graph algorithms is anticipated to run only as fast as rhotrix multiplication. From matrices perspective, it has been established that the standard method for multiplying $n \times n$ matrices requires $O(n^3)$ multiplications. This fundamental concept can be of help to us as we integrate similar notion in determining the computational time complexity for rhotrices multiplication.

3.20.1. Computational Time Complexity for Row-Column Multiplication

Theorem 2: The computational time sequence for any $n \times n$ rhotrices multiplication is given by $\frac{1}{4}(n^3 + 3n)$, which also belongs to the class of multiplication time complexity function of $O(n^3)$ classes, where n is the dimension of the rhotrix.

Proof:

Recall that the computational time complexity for matrix multiplication is $O(n^3)$ where n is the order of the square matrix. We note that in rhotrix there exist two matrices of orders t and $t-1$ respectively.

Thus, the computation time sequence of the square matrix of order $t-1$ is given by:

$T_{com} = (t-1)^3 = t^3 - 3t^2 + 3t - 1$. Since the total number of multiplications in both the outer and inner matrices equals the number of multiplication in the host rhotrix then

$$T_{comp} = t^3 + (t-1)^3$$

Therefore, the time complexity for the two matrices combined is

$$T_{comp} = t^3 + (t-1)^3 = t^3 + t^3 - 3t^2 + 3t - 1 = 2t^3 - 3t^2 + 3t - 1 \quad (3.21)$$

However, there exist a relationship between the dimension of a rhotrix and the orders of its outer and inner matrices, for example, if n is the dimension of the host rhotrix, t the order of the outer matrix and $t-1$ the order of the inner matrix then $n = t + t - 1 = 2t - 1$. Which implies

that $t = \left(\frac{n+1}{2}\right)$. Thus substituting t in equation (3.21), we have

$$T_{comp} = 2\left(\frac{n+1}{2}\right)^3 - 3\left(\frac{n+1}{2}\right)^2 + 3\left(\frac{n+1}{2}\right) - 1$$

$$T_{comp} = 2\left(\frac{n^3 + 3n^2 + 3n + 1}{8}\right) - 3\left(\frac{n^2 + 2n + 1}{4}\right) + 3\left(\frac{n+1}{2}\right) - 1$$

$$T_{comp} = \frac{1}{4}(n^3 + 3n^2 + 3n + 1) - \frac{3}{4}(n^2 + 2n + 1) + \frac{3}{2}(n+1) - 1$$

$$T_{comp} = \frac{1}{4}(n^3 + 3n^2 + 3n + 1) - \frac{3}{4}(n^2 + 2n + 1) + \frac{6}{4}(n+1) - 1$$

$$T_{comp} = \frac{1}{4}n^3 + \frac{3}{4}n^2 - \frac{3}{4}n^2 + \frac{3}{4}n - \frac{6}{4}n + \frac{6}{4}n + \frac{1}{4} - \frac{3}{4} + \frac{6}{4} - 1$$

$$T_{comp} = \frac{1}{4}(n^3 + 3n). \text{ Hence the proof.}$$

Based on the above derivation, we conclude that the computational time complexity for any row-column rhotrix multiplication is given by $O(n^3)$.

3.20.2. Computational Time Complexity for Heart-Oriented Multiplication

Theorem 3: we can multiply two $n \times n$ dimensional rhotrices in $O(n^2)$ times.

Let consider say 3, 5, 9 etc dimensional rhotrices with $n = 3, 5, 9$. etc

For $n = 3$, there are total of 9 multiplication, and $9 = 3^2$

For $n = 5$, there are total of 25 multiplication, and $25 = 5^2$

For $n = 7$, there are total of 49 multiplication, and $49 = 7^2$

Thus we can generalize that, $T_{comp} = 2\left[\frac{1}{2}(n^2 + 1)\right] - 1 = n^2$

or a computational time complexity of $O(n^2)$

3.21. Conclusion

In this chapter we have presented a number of sequential and parallel rhotrix multiplication algorithm based on either the row-column or heart-oriented methods. But prior to this, we had to derive three new formula expressions for the heart-oriented and one for the row-column rhotrix multiplication and also a generalized representation format for both cases. Similarly, we described a theoretical computational model for rhotrix multiplication using the Cannon's algorithm, Process array and Strassen's algorithm, in which case we were able to implement a parallel version of each algorithm and presented a parallel framework for the process array computational model and MPI implementation on distributed memory systems. The Strassen's algorithm we believe is scalable and portable to other platforms because most of the desired called subroutines already exist in a standard library. Hence it is worthwhile to put more effort into this method so that the basic operation of rhotrix multiplication can be improved further.

CHAPTER FOUR: IMPLEMENTATION AND PERFORMANCE RESULTS

4.1. Introduction

In this chapter, we first describe the implementation of the parallel rhotrix multiplication algorithms given in previous sections. Then, the experimental results of each algorithm are presented. The experimental tests were conducted in two phases. In phase I: comprising of Section 4.2 and Section 4.3, we tested the parallel rhotrix heart-oriented multiplication algorithms on homogenous and heterogeneous process grids in order to reveal the scalability and general performance characteristics of each algorithms with various problem sizes and process count. We deliberately designed four sets of test cases, which were guided by the performance models presented in Chapter three. In phase II: comprising of Section 4.3 and Section 4.4, we tested the parallel row-column rhotrix multiplication algorithm by means of simulator designed using Delphi API. The simulator serves as process array emulator. All the row-column rhotrix multiplication algorithms were tested on rhotrix with different sizes. The combination of these two phases' experimental results was used to test the research hypothesis.

4.2. MPI Performance Results

For our experiments we used DeinoMPI which is an implementation of the MPI-2 standard for Microsoft Windows originally derived from the MPICH2 distribution from Argonne National Laboratory. MPI is software for message passing, proposed as a standard by a broad committee of vendors, implementers, and users (Huss–Lederman *et al.*, 1993). MPI is portable and flexible software that is used widely. It is considered as a standard for writing message-passing programs in which higher level routines and abstractions are built upon lower level message passing routines.

All the experiments were performed on a cluster of HP Pavilion dv6500 Notebook PC Intel(R) Core(TM) Duo CPU T5550 @1.83GHz 1.83GHz, Memory (RAM): 2.50GB 32 bit running the Windows Vista operating system. Each node has four Gigabytes of memory, but we only use 1.25GB of memory. The nodes are connected with a switched 100Mbps Fast Ethernet network. In order to build a master-worker platform, we arbitrarily choose one processor as the master, and the other processors will serve as workers. Finally we used MPI WTime as timer in all experiments.

In our studies, we have discovered that the operations performed by our parallel algorithms can be put into three categories:

- v. Computations that are performed sequentially
- vi. Computations that are performed in parallel
- vii. Parallel overhead (communication operations and reduction computations)

One of the basic fundamental concepts behind our parallel algorithm design was to come up with a suitable parallel algorithm that will execute larger tasks of rhotrix block faster than their sequential counterparts. In this section we try to analyze the performance of our algorithms by considering the two basic metrics that are commonly used to measure performance, that is the program execution time and speedup. An obvious performance parameter is a parallel program's execution time, or what is commonly referred to as the *wall-clock time*. The execution time is defined as the time elapsed from when the first processor starts executing a problem to when the last processor completes execution.

We start by determining the program execution time for both sequential and parallel computation of the heart-oriented algorithm followed by estimation of execution speedup factor, which is defined as the ration between the sequential execution time and parallel execution time give in (3.34).

In our experiments we implemented the two different algorithmic methods of performing rhotrix multiplication using MPI. These two implementations of the rhotrix algorithm were tested using rhotrices of dimensions ranging from 5 up to 27 with strides of 2. In order to avoid overflow exceptions for large rhotrix dimension, small-valued non negative rhotrix elements were used. The experiments have been repeated using 2 up to 20 hosts in steps of 2 for both implementations with a total of 4 test runs.

The employed cluster environment is a typical academic installation with dedicated local area network. In order to avoid pointless scenarios, the experiments were conducted after midnights where no unknown loads on the hosts. Furthermore, unnecessary time stealing daemons were blocked at the time of experimentation.

Table 4.2a Sequential execution time for seven sets of heart-oriented rhotrices computation

Rhotrix Size	13	41	61	85	113	145	181	365
Dimension (n)	5	7	9	13	15	17	19	27
Time of Execution (s)	17.095	19.739	22.534	24.803	25.393	26.234	27.204	35.825

We can interpret from the data in Tables 4.2a and 4.2b the real performance pattern of our two sequential algorithms for the heart-oriented and row column computations depicted in Figures 4.2a and 4.2b, that increases in each of the relative execution time for both algorithm are seen to be directly proportional to the number of rhotrix size. Although we expect to see a progressive increase in the execution time, the abnormalities seen in the two graphs are due to some experimental factors relating to system capacity such as speed size, memory and the state of the system as of the time of program runs.

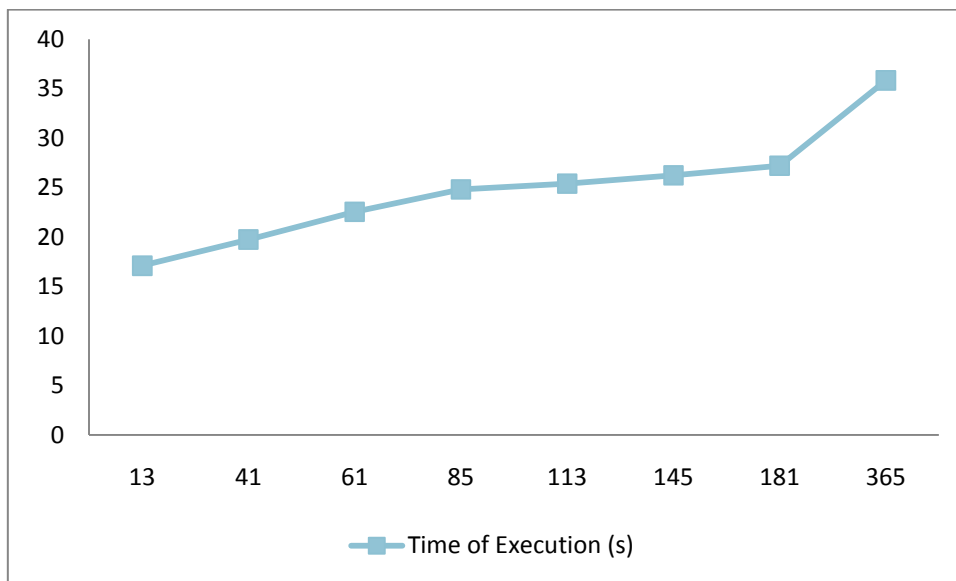


Figure. 4.2a: The real execution times for heart-oriented rhotrix algorithm

Table 4.2b Sequential execution time for seven sets of row-column rhotrices computation

Rhotrix Size	13	41	61	85	113	145	181	365
Dimension (n)	5	7	9	13	15	17	19	27
Time of Execution (s)	19.640	23.431	24.534	25.803	27.393	28.612	30.204	35.412

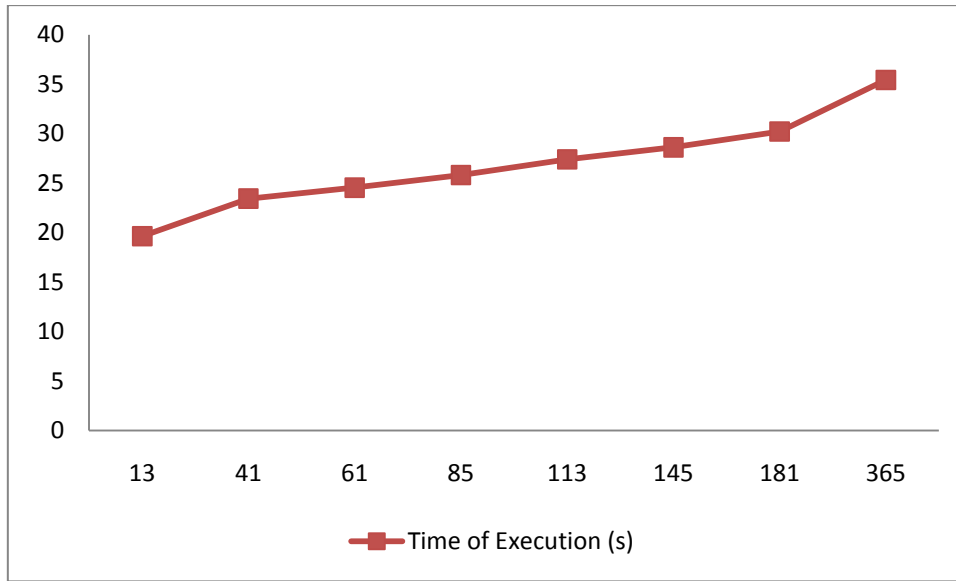


Fig. 4.2b: The real execution times for row-column rhotrix algorithm.

Table 4.2c Parallel execution time of four sets of heart-oriented rhotrices computation

p	2	4	6	8	10	12	14	17	20
n = 113	0.784309	0.581550	0.499357	0.295838	0.305847	0.199402	0.133509	0.117245	0.086092
n = 145	0.793727	0.636489	0.444367	0.341249	0.273908	0.183845	0.187662	0.170244	0.097259
n = 181	0.843268	0.834887	0.546062	0.377213	0.343849	0.303510	0.260287	0.236591	0.170666
n = 365	1.241481	1.031977	0.620731	0.581010	0.530133	492034	0.418176	0.333949	0.370984

Figure.4.2c. shows the parallel execution time for MPI-based heart-oriented rhotrix multiplication algorithm on 2, 4, 8, 10, 12, 14, 17 and 20 hosts and for rhotrix orders ranging from 113, 145, 181 and 365 elements. We can infer from Figure 4.3. that the algorithm runs faster on a larger number of hosts, the gain in the speedup factor is slower for smaller number of processors. For instance, the difference in execution time between 2, 4, 6, 8 10 and 12 hosts is smaller than the difference between 14, 17 and 20 hosts. This is due to the reduction in execution time. This reduction becomes even more apparent for a large number of hosts.

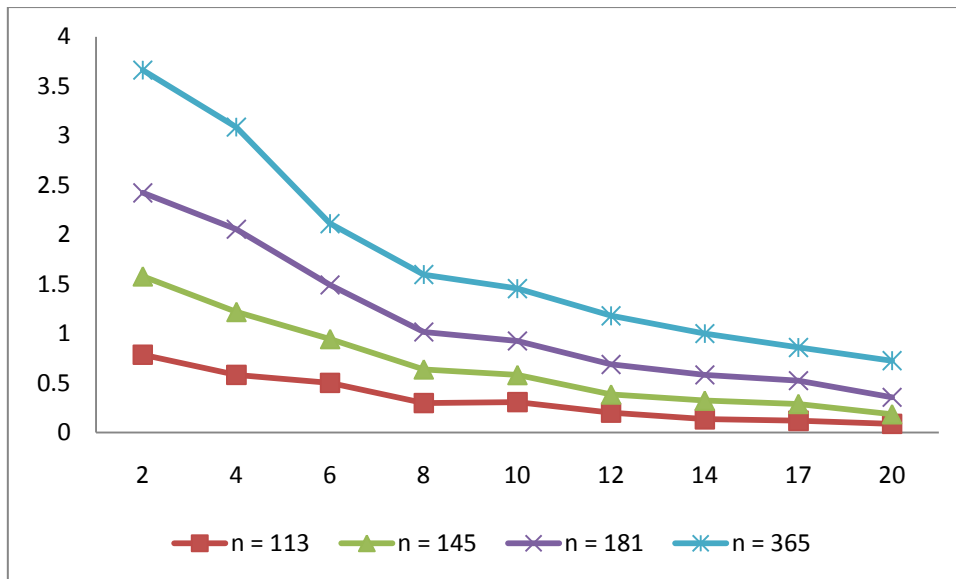


Figure. 4.2c: The parallel execution times for heart-oriented rhotrix algorithm.

Table 4.2d Parallel speedup of four sets of heart-oriented rhotrices computation

p	2	4	6	8	10	12	14	17	20
n = 113	0.086092	0.117245	0.133509	0.199402	0.305847	0.295838	0.499357	0.581550	0.784309
n = 145	0.097259	0.170244	0.187662	0.183845	0.273908	0.341249	0.444367	0.636489	0.793727
n = 181	0.170666	0.236591	0.260287	0.303510	0.343849	0.377213	0.546062	0.834887	0.843268
n = 365	0.370984	0.333949	0.418176	492034	0.530133	0.581010	0.620731	1.031977	1.241481

Figure.4.2d. shows the estimated program speedup for MPI-based heart-oriented rhotrix algorithm on 2, 4, 8, 10, 12, 14, 17 and 20 hosts and for rhotrix sizes of 113, 145, 181 and 365 blocks. Similarly Figure 4.2d. can also be interpreted to mean that our parallel algorithm runs faster on a larger number of hosts than on smaller number of hosts, the gain in the speedup factor is seen to increase with an increasing number in processor size.

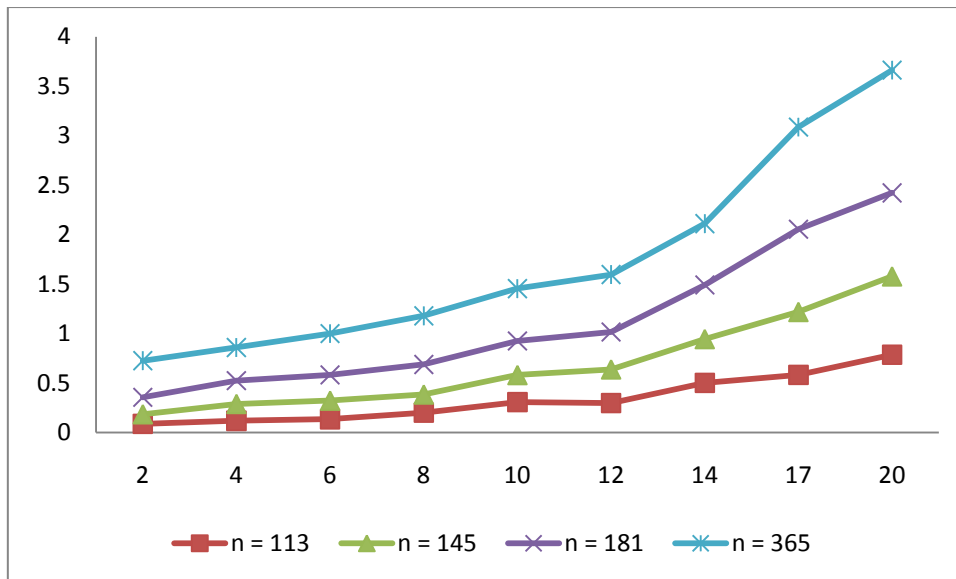


Figure. 4.2d: Program speedup for MPI-based heart-oriented rhotrix algorithm

4.3. Performance Analyzer

MPI Timeline: The MPI Timeline for heart-oriented rhotrix algorithm graphically displays the MPI activity that occurred during an application's run. For each MPI process you can look horizontally to see what the process is doing as a function of elapsed time. Figures 4.3a, 4.3b and 4.3c are screen shots of the Sun Studio Analyzer window which shows processes P0 to P3, over a time span of 0.225 seconds, P0 to P13 over a time span of 0.60 seconds, and P0 to P19 over a time span of 0.80 seconds.

The primary view of logfile data is the “timeline window” in which time (in seconds) is indicated from left to right and MPI process ranks are shown from top to bottom. Coloured rectangles spanning section of the timelines indicates that a particular process was in a particular state during the indicated time interval. These states are defined by logging library and typically consist of MPI function call durations and durations of user-defined states. Clicking with the mouse on such a rectangle pops up a small windows containing detailed information (state name, process duration, etc). Arrows show messages, and details about a particular message (length, tag, etc) appearing in one of the circles associated with the origin of an arrow.

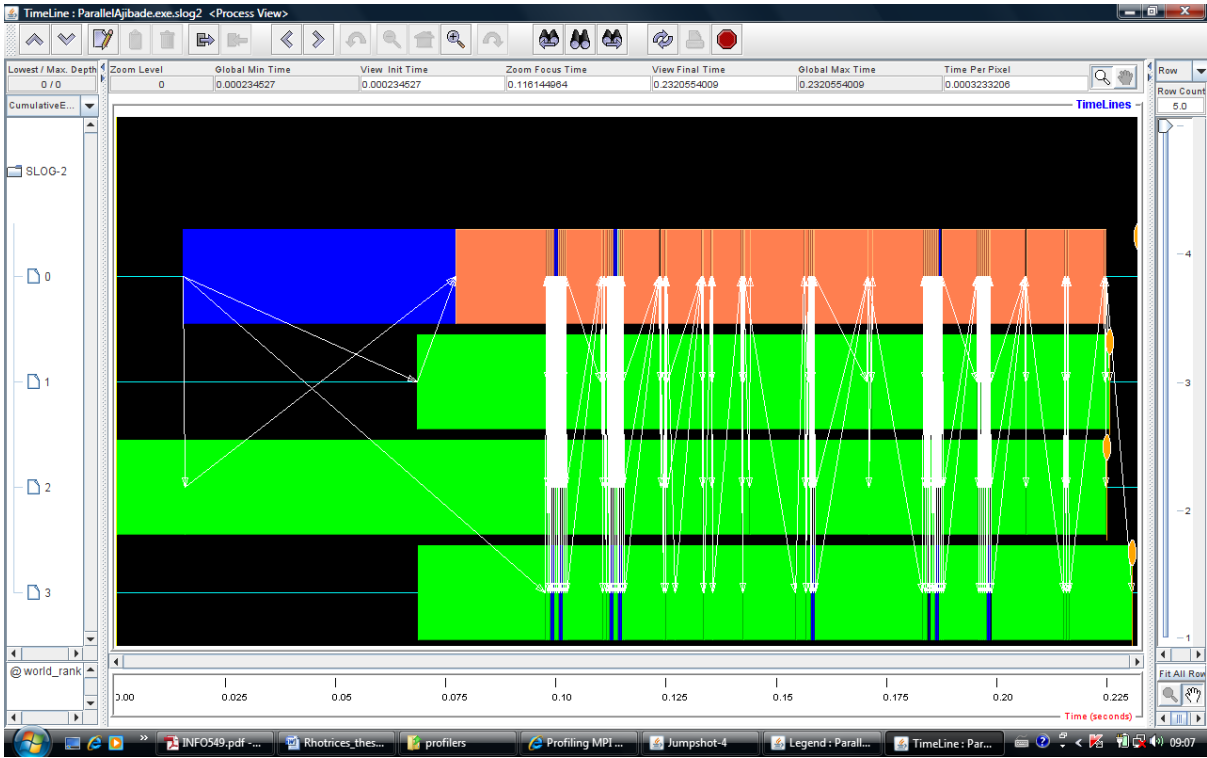


Figure 4.3a: MPI TimeLine Screenshot Process View for P0 to P3

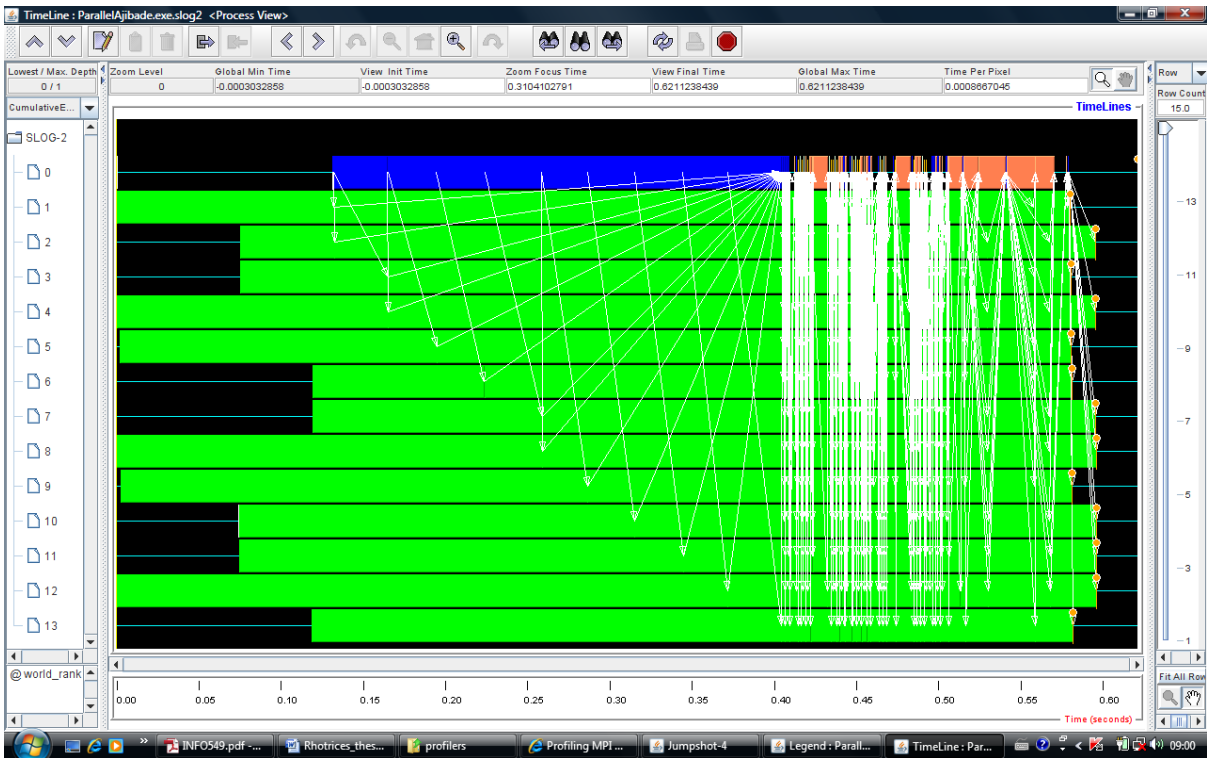


Figure 4.3b: MPI TimeLine Screenshot Process View for P0 to P13

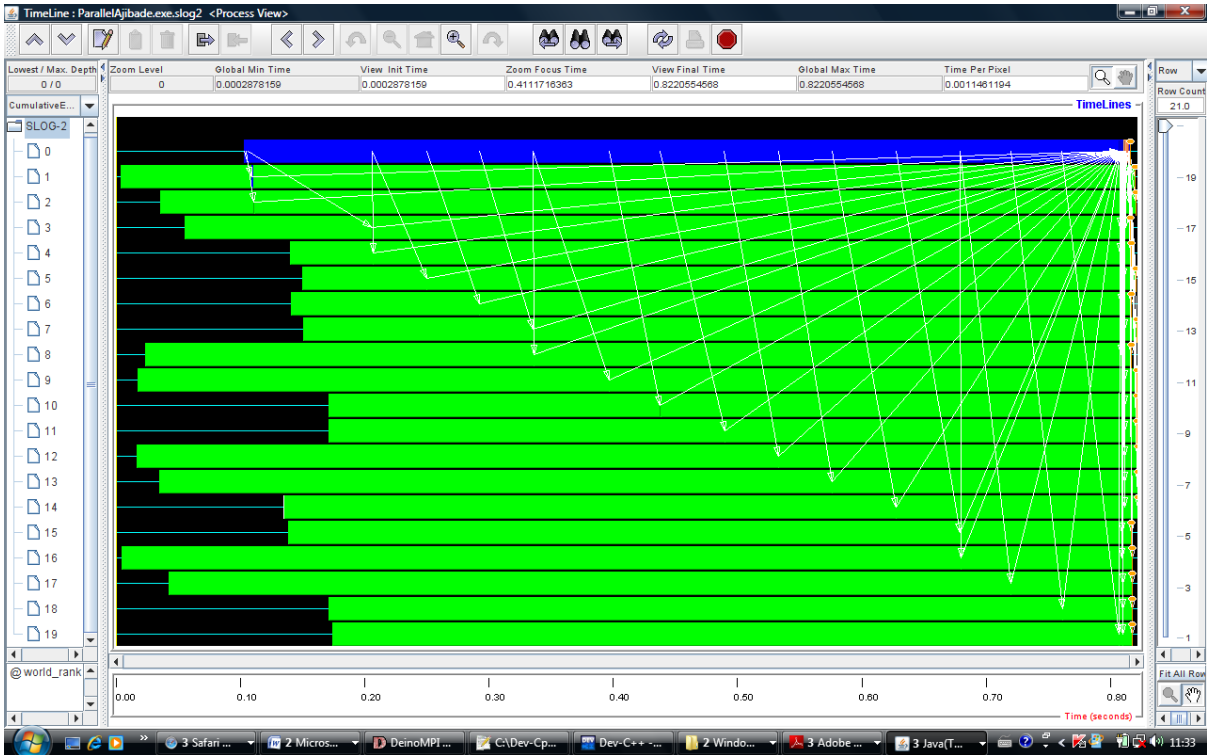


Figure 4.3c: MPI TimeLine Screenshot Process View for P0 to P19

Gantt view: It shows event that are aligned on the paralleled timelines of individual processes, with processes on Y-axis and time on X-axis. Each states are represented by one colour. The Absolute Time, measured in seconds, is shown at the top along the horizontal axis. The Relative Time, measured in milliseconds, is shown at the bottom along the horizontal axis. At the left, the MPI process ranks from P0 to P20 are listed. At this level of zoom, the names of MPI API functions are not visible. Figure 4.3d is a snapshot of process view legends.

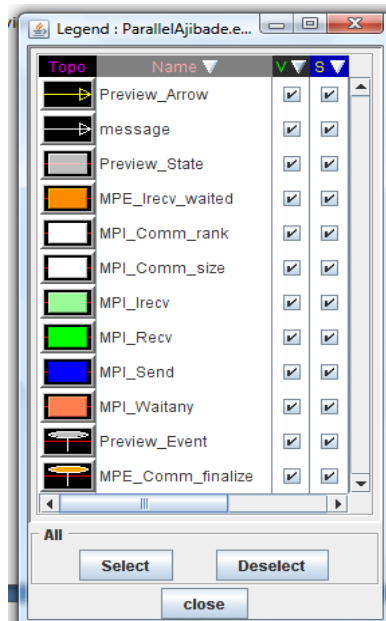


Figure 4.3d: TimeLine Processes View Legend

Some of the basic MPI function calls used in our program and also reflecting on the TimeLine process view legend as shown in Figure 4.3d are briefly explained as follow:

- i. **MPI_Send**: this function call is responsible for allowing process to send message to another process.
- ii. **MPI_Reduce**: this function call is responsible for performing the task of reduction operation on values submitted by all process in a communicator.
- iii. **MPI_Recv**: this calls method allows a participating process to receive a message sent by another member process.
- iv. **MPI_Bcast**: this message passing function call performs the task of broadcasting messages to all participating processes in a communicator.

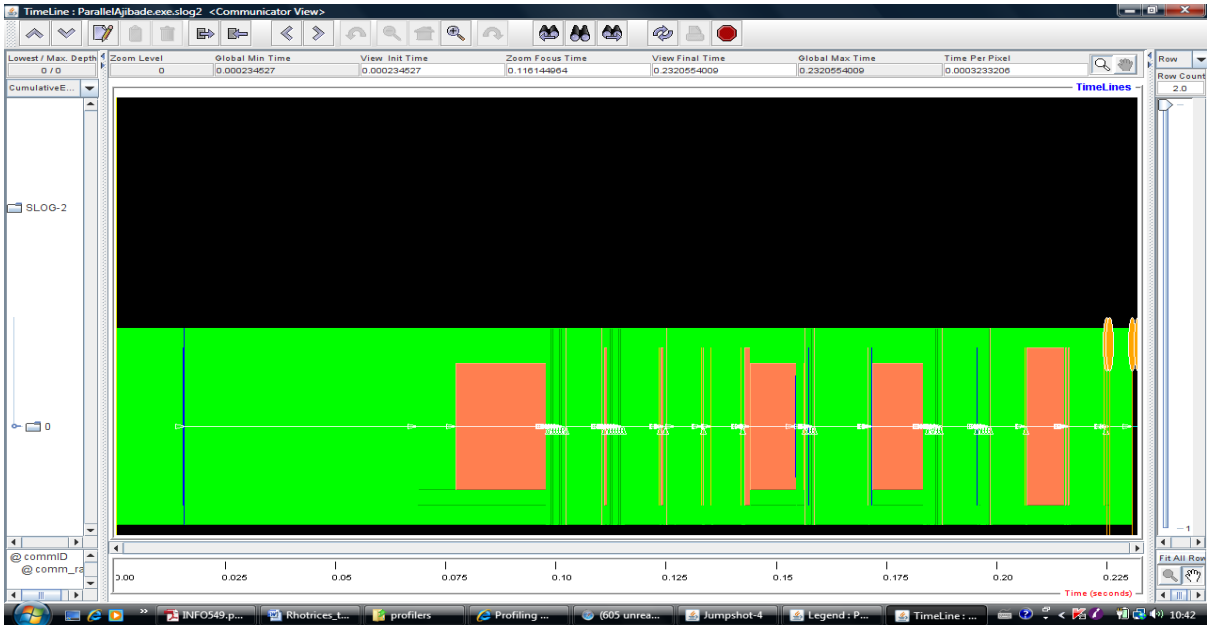


Figure 4.3e: Screenshot of TimeLine Communicator View for process P0 to P13

It is equally crucial to note that while only a single process

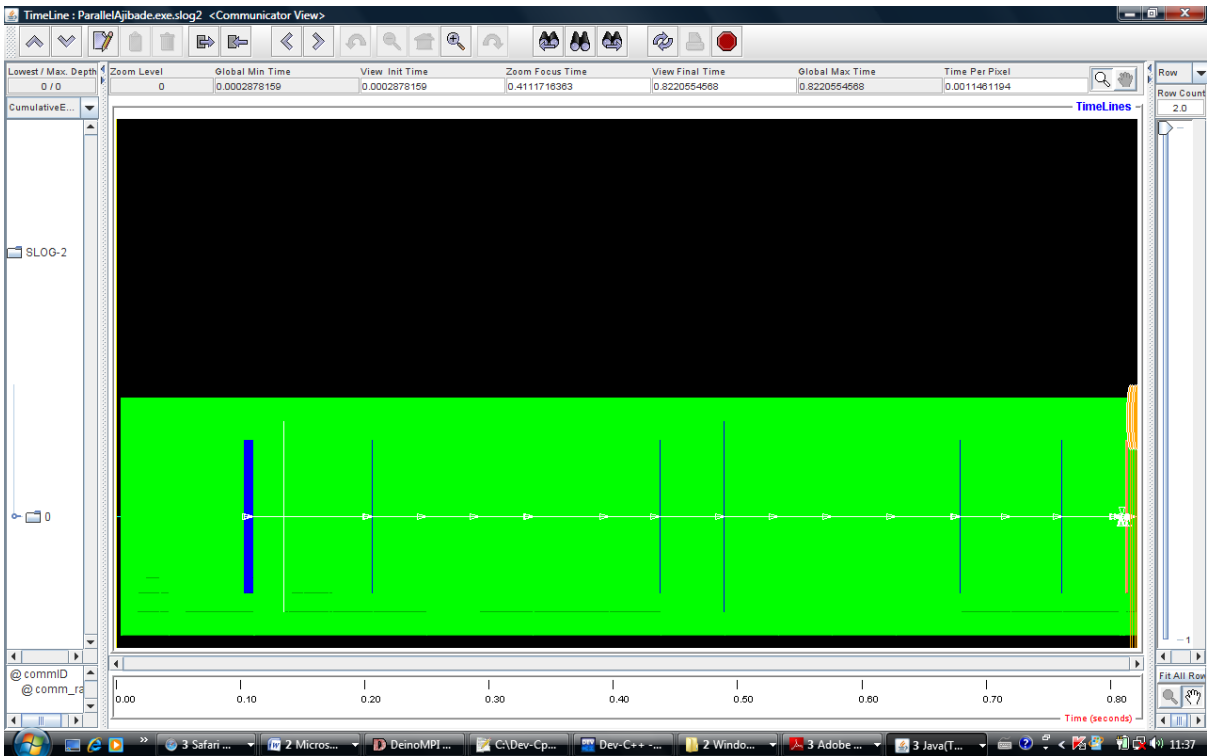


Figure 4.3f: Screenshot of TimeLine Communicator View for process P0 to P19

4.4. Process Array Simulation for Coupled Matrix Implementation

In this section we considered the parallel multiplication scenario of two coupled matrices, which computation process is as shown below. The processing elements are numbered from P0 to P24. The computation is done in such a way that, at every clock tick some sets of computations is carried out, this task is repeated until the n^{th} tasks is executed (Ezugwu *et al*, 2010b).

$$R_5(A) = \left\langle \begin{array}{ccccc} & & 2 & & \\ & & 3 & 1 & 4 \\ 4 & 2 & 4 & 5 & 3 \\ & & 1 & 5 & 4 \\ & & 6 & & \end{array} \right\rangle^{T/2} \quad \text{and} \quad Q_5 = \left\langle \begin{array}{ccccc} & & 2 & & \\ & & 3 & 1 & 4 \\ 4 & 2 & 4 & 5 & 3 \\ & & 1 & 5 & 4 \\ & & 6 & & \end{array} \right\rangle^{T/2}$$

$$R_5^{T/2} = \begin{pmatrix} 2 & 4 & 2 \\ 4 & 5 & 1 \\ 4 & 4 & 3 \\ 5 & 5 & 2 \\ 6 & 1 & 4 \end{pmatrix} \quad \text{and} \quad Q_5^{T/2} = \begin{pmatrix} 4 & 1 & 6 \\ 2 & 5 & 4 \\ 3 & 4 & 4 \\ 1 & 5 & 4 \\ 2 & 4 & 2 \end{pmatrix}$$

Filling the missing spaces with zero's, we obtain

$$R_5^{T/2} = \begin{pmatrix} 2 & 0 & 4 & 0 & 2 \\ 0 & 5 & 0 & 1 & 0 \\ 4 & 0 & 4 & 0 & 3 \\ 0 & 5 & 0 & 2 & 0 \\ 6 & 0 & 1 & 0 & 4 \end{pmatrix} \quad \text{and} \quad Q_5^{T/2} = \begin{pmatrix} 4 & 0 & 1 & 0 & 6 \\ 0 & 2 & 0 & 5 & 0 \\ 3 & 0 & 4 & 0 & 4 \\ 0 & 1 & 0 & 5 & 0 \\ 2 & 0 & 4 & 0 & 2 \end{pmatrix}$$

we flip columns of the first coupled matrix and rows of the second coupled matrix to get

$$R_5^{T/2} = \begin{pmatrix} 2 & 0 & 4 & 0 & 2 \\ 0 & 5 & 0 & 1 & 0 \\ 4 & 0 & 4 & 0 & 3 \\ 0 & 5 & 0 & 2 & 0 \\ 6 & 0 & 1 & 0 & 4 \end{pmatrix} \quad \text{and} \quad Q_5^{T/2} = \begin{pmatrix} 4 & 0 & 1 & 0 & 6 \\ 0 & 2 & 0 & 5 & 0 \\ 3 & 0 & 4 & 0 & 4 \\ 0 & 1 & 0 & 5 & 0 \\ 2 & 0 & 4 & 0 & 2 \end{pmatrix}$$

and finally, we do the staggering of the flipped coupled matrix, as illustrated in Figure 4.4a.

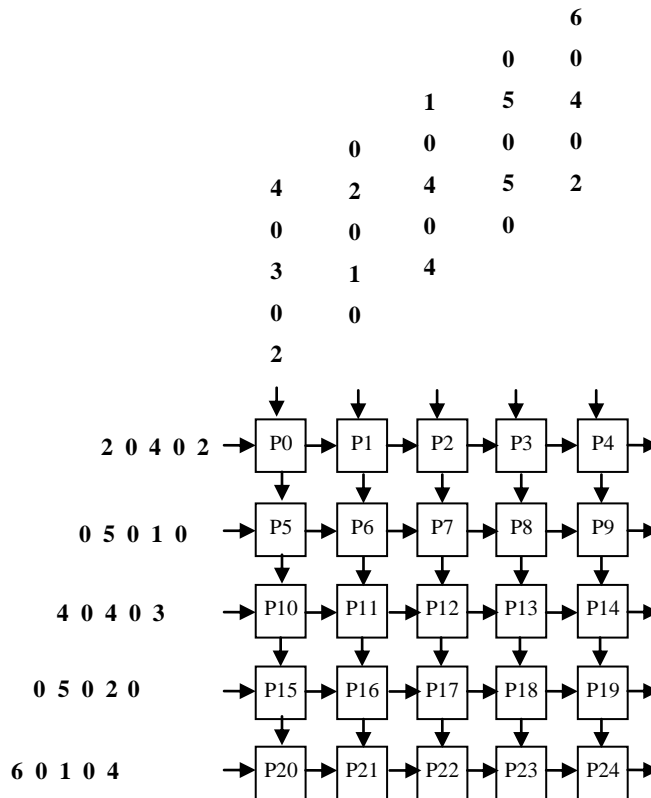


Figure 4.4a: Staggering of rhotrix vector into processing elements

The staggering process of the rhotrix elements into each block of processing elements presents an internal architectural grunching structure adopted by the systolic array as a model of computational processes involved in implementing the last stages of our parallel algorithm given in Section 3.5.1 above. The process model is as shown in Figure 4.4b bellow.



Figure 4.4b: Screenshot process array simulator input interface

The following processing blocks depicts the step by step computation method used in the process array simulation computation for the coupled matrix.

Clock Tick One

P0 <---- 2*2

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21	P22	P23	P24
4																								

Clock Tick two

P0 <---- 0*0, P1 <---- 0*2, P5 <---- 2*0

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21	P22	P23	P24
4	0				0																			

Clock Tick three

P0 <---- 3*4, P1 <---- 0*1, P2 <---- 4*2, P5 <---- 1*0, P6 <---- 0*0, P10 <---- 2*3

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21	P22	P23	P24
16	0	8			0	0				6														

Clock Tick four

P0 <---- 0*0, P1 <---- 0*4, P2 <---- 0*0, P3 <---- 0*2, P5 <---- 3*0, P6 <---- 1*1, P7 <---- 0*4, P10 <---- 0*0, P11 <---- 0*3, P15 <---- 2*0

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21	P22	P23	P24
16	0	8	0		0	1	0			6	0				0									

Clock Tick five

P0 <---- 4*2, P1 <---- 0*2, P2 <---- 4*4, P3 <---- 0*5, P4 <---- 2*2, P5 <---- 0*5, P6 <---- 0*0,

P7 <---- 0*1, P8 <---- 0*0, P10 <---- 3*4, P11 <---- 0*1, P12 <---- 3*4, P15 <---- 0*2,

P16 <---- 0*0, P20 <---- 2*4

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21	P22	P23	P24
24	0	24	0	4	0	1	0	0		18	0	12			0	0				8				

Clock Tick six

P1 <---- 2*0, P2 <---- 0*0, P3 <---- 0*4, P4 <---- 0*0, P5 <---- 0*4, P6 <---- 2*5,

P7 <---- 0*4, P8 <---- 5*1, P9 <---- 0*2, P10 <---- 0*0, P11 <---- 0*4, P12 <---- 0*0,

P13 <---- 0*3, P15 <---- 3*0, P16 <---- 1*2, P17 <---- 0*4, P20 <---- 0*0, P21 <---- 4*0

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21	P22	P23	P24
24	0	24	0	4	0	11	0	5	0	18	0	12	0		0	2	0			8	0			

Clock Tick seven

P2 <---- 2*1, P3 <---- 0*5, P4 <---- 4*4, P6 <---- 0*0, P7 <---- 0*5, P8 <---- 0*0, P9 <----

0*1, P10 <---- 4*4, P11 <---- 0*2, P12 <---- 4*4, P13 <---- 0*5, P14 <---- 2*3, P15 <---- 0*5,

P16 <---- 0*0, P17 <---- 0*2, P18 <---- 0*0, P20 <---- 3*1, P21 <---- 1*0, P22 <---- 4*4

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21	P22	P23	P24
24	0	26	0	20	0	11	0	5	0	34	0	28	0	6	0	2	0	0		11	0	16		

Clock Tick eight

P3 <---- 0*2, P4 <---- 0*0, P7 <---- 0*1, P8 <---- 5*5, P9 <---- 0*4, P11 <---- 0*4, P12 <---- 0*0, P13 <---- 0*4, P14 <---- 0*0, P15 <---- 0*4, P16 <---- 2*5, P17 <---- 0*4, P18 <---- 2*5, P19 <---- 0*2, P20 <---- 0*0, P21 <---- 0*1, P22 <---- 0*0, P23 <---- 0*4

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21	P22	P23	P24
24	0	26	0	20	0	11	0	30	0	34	0	28	0	6	0	12	0	10	0	11	0	16	0	

Clock Tick nine

P4 <---- 6*2, P8 <---- 0*0, P9 <---- 0*5, P12 <---- 1*4, P13 <---- 5*0, P14 <---- 4*4, P16 <---- 0*0, P17 <---- 0*5, P18 <---- 0*0, P19 <---- 0*2, P20 <---- 4*6, P21 <---- 2*0, P22 <---- 4*1, P23 <---- 5*0, P24 <---- 2*4

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21	P22	P23	P24
24	0	26	0	32	0	11	0	30	0	34	0	32	0	24	0	12	0	10	0	35	0	20	0	8

Clock Tick ten

P9 <---- 0*6, P13 <---- 0*4, P14 <---- 0*0, P17 <---- 0*1, P18 <---- 5*5, P19 <---- 0*4, P21 <---- 0*6, P22 <---- 0*0, P23 <---- 1*0, P24 <---- 0*0

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21	P22	P23	P24
24	0	26	0	32	0	11	0	30	0	34	0	32	0	24	0	12	0	35	0	35	0	20	0	8

Clock Tick eleven

P14 <---- 4*6, P18 <---- 0*0, P19 <---- 0*5, P22 <---- 1*6, P23 <---- 5*0, P24 <---- 4*1

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21	P22	P23	P24
24	0	26	0	32	0	11	0	30	0	34	0	32	0	46	0	12	0	35	0	35	0	26	0	12

Clock Tick twelve

P19 <---- 0*6, P23 <---- 0*6, P24 <---- 0*0

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21	P22	P23	P24
24	0	26	0	32	0	11	0	30	0	34	0	32	0	46	0	12	0	35	0	35	0	26	0	12

Clock Tick thirteen

P24 <---- 6*6

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21	P22	P23	P24
24	0	26	0	32	0	11	0	30	0	34	0	32	0	46	0	12	0	35	0	35	0	26	0	48

The final result obtained is same as in $2n+3$ times, though we believe that optimization is very much possible.

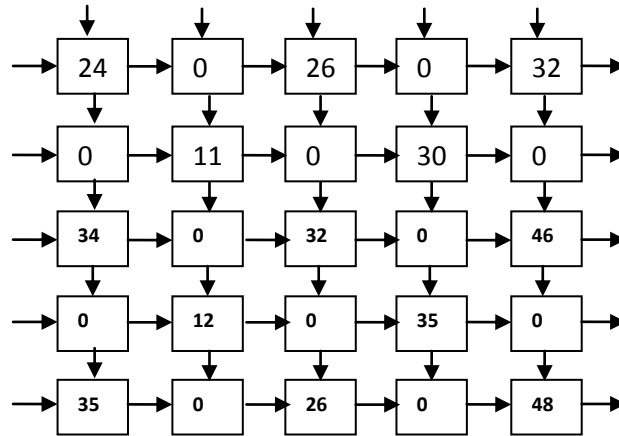


Figure 4.4c: Process array computation result.

The simulation result can be compared with the theoretical result shown below; the two solutions gave the same out values to indicate that our simulation approach is in line with the initial proposed parallel algorithms in Section 3.5.3.

$$C_5^{T/2} = \begin{pmatrix} 24 & 26 & 32 \\ & 11 & 30 \\ 34 & 32 & 46 \\ & 12 & 35 \\ 55 & 26 & 48 \end{pmatrix} \quad C_5 = \begin{pmatrix} & 24 & & & \\ & 34 & 11 & 26 & \\ 55 & 12 & 32 & 30 & 32 \\ & 26 & 35 & 46 & \\ & & & & 48 \end{pmatrix}$$

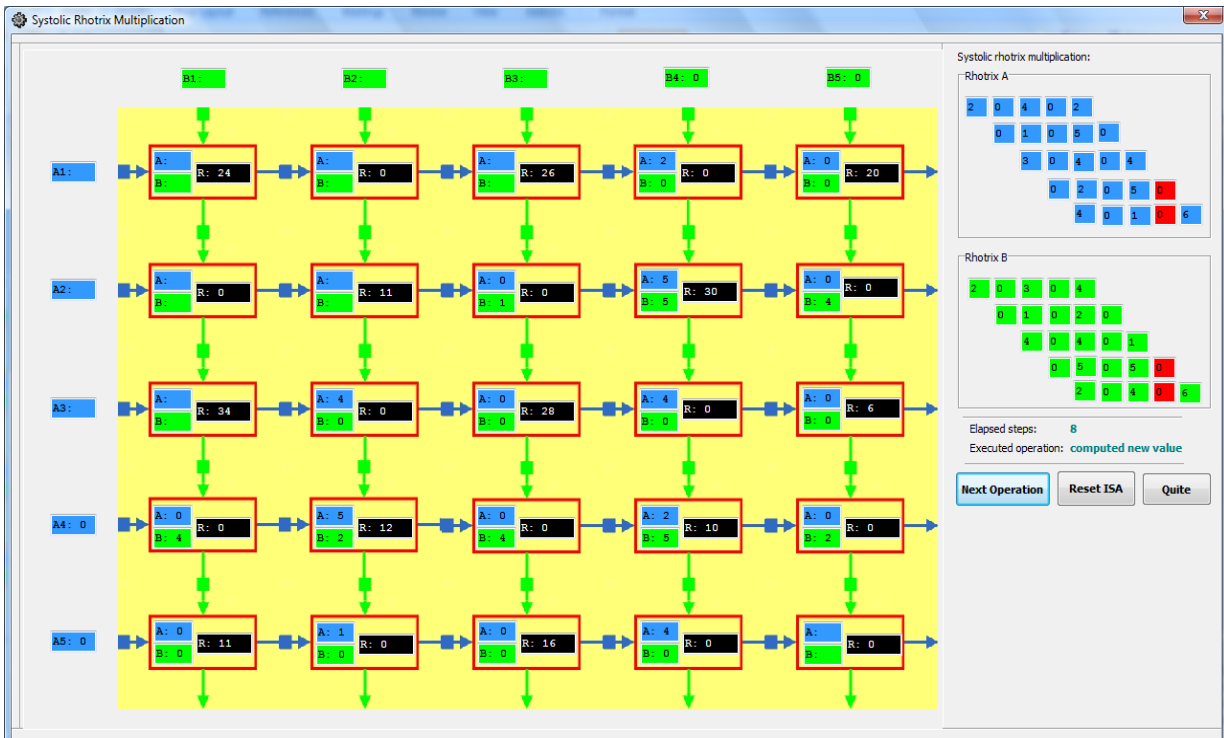


Figure 4.4d: Screenshot of process array simulator output result

4.5. Process Array Simulation for Row-Column Algorithm Implementation

We have presented below, our new process array design model to accommodate the processing imbalance found in the rhotrix structure. It can be recaptured that for the rhotrix row-column computation discussed in Section 3.3, the hearts elements are considered separately for multiplication in terms of their row-column cross multiplication even though they are embedded in-between the rhotrix main entries. Our new design structure presented in Figure 4.5 caters for this new computation procedure. One important observation with this approach is that, we have completely overcome the issue of rhotrix padding (coupled matrix) transformation. In essence, what we have is just a step-in step-out computation process. At every clock tick, a set of computation is performed by the processing elements in pipeline order, this processing format is further clarified from the lists of figures presented below.

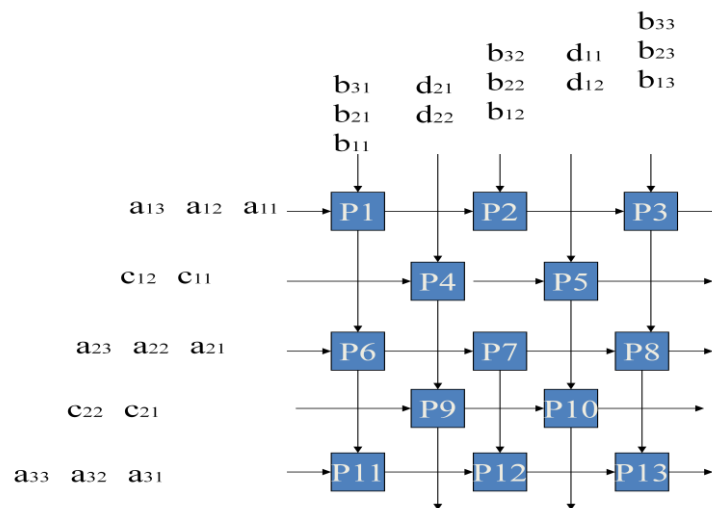
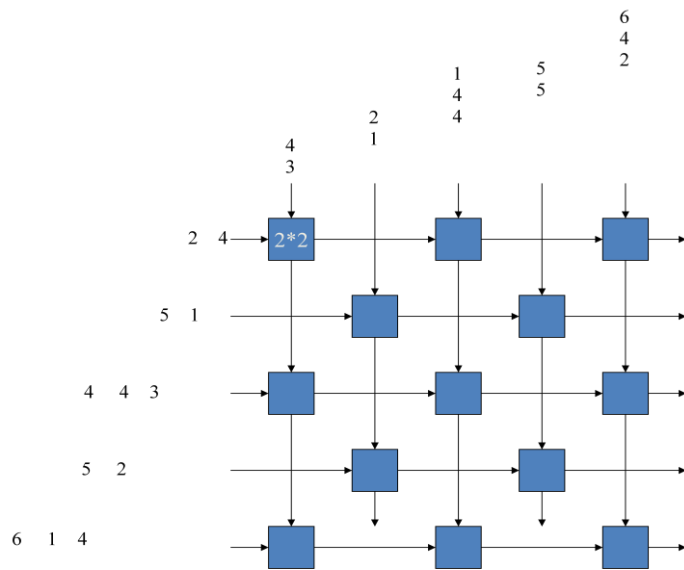


Figure 4.5a: Row-column process array structure

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13
0	0	0	0	0	0	0	0	0	0	0	0	0

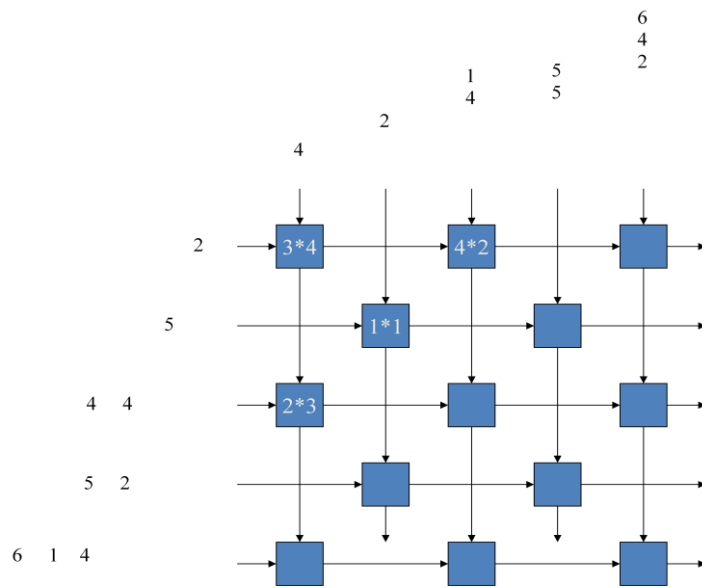
Step one: At clock tick one, we have the computation



P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13
4	0	0	0	0	0	0	0	0	0	0	0	0

Figure 4.5b Computation performed by Process P1

Step two: At clock tick two, we have the computation



P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13
16	8	0	1	0	6	0	0	0	0	0	0	0

Figure 4.5c Computation performed by Process P1, P2, P4

Step three: At clock tick three, we have the computation

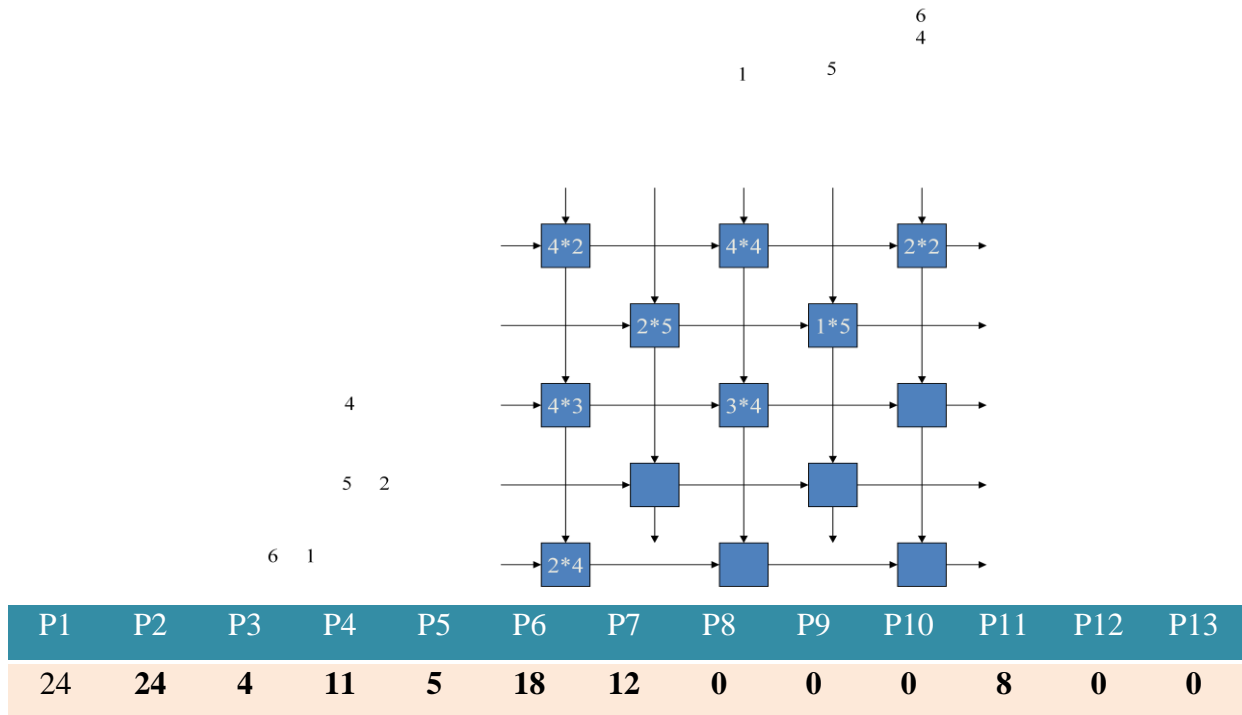


Figure 4.5d: Computation performed by Process P1, P2, P3, P4, P5, P6, P7, P11

Step four: At clock tick four, we have the computation

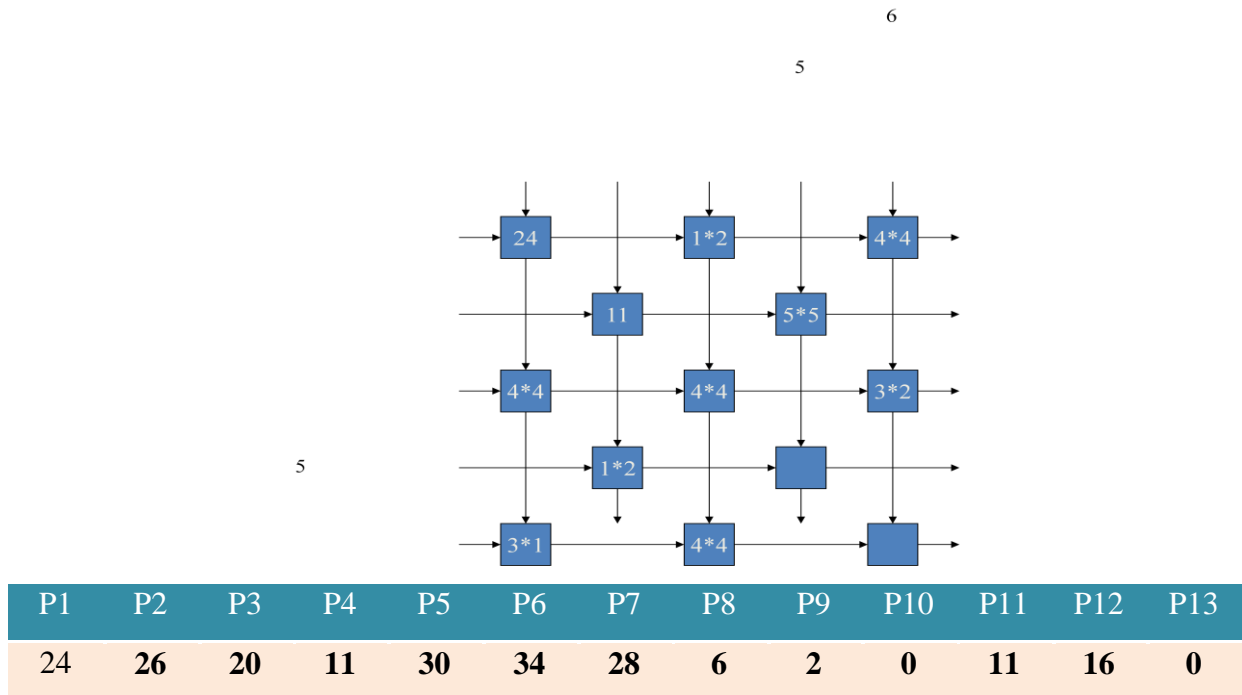


Figure 4.5e Computation performed by Process P2,P3, P5, P6, P7, P8, P9, P11

Step five: At clock tick five, we have the computation

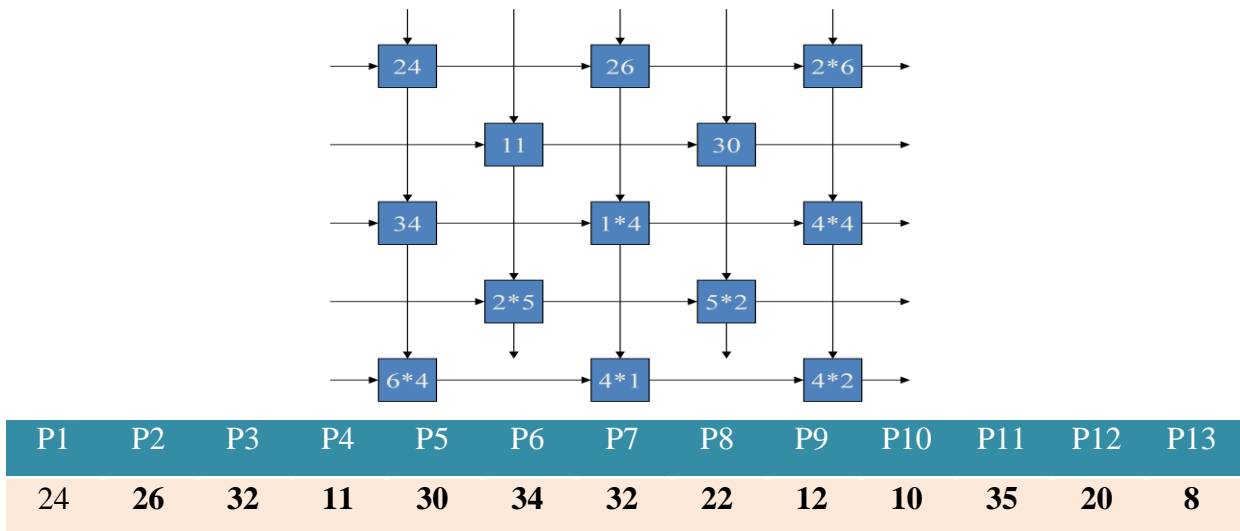


Figure 4.5f: Computation performed by Process P3, P7, P8, P9, P11, P10, P12, P13

Step six: At clock tick six, we have the computation

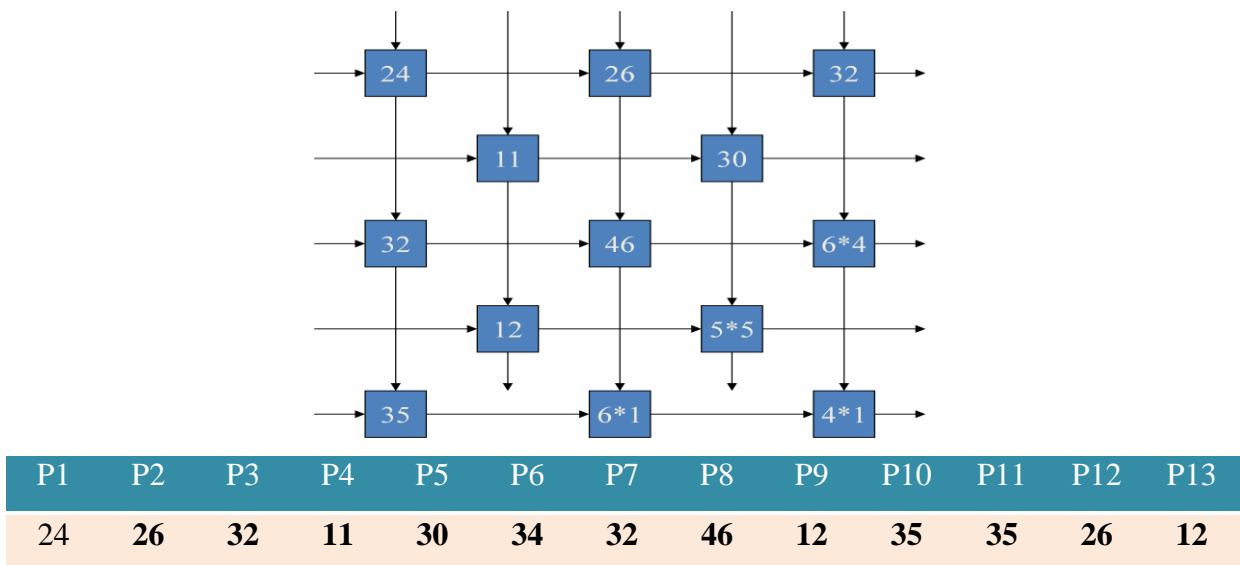


Figure 4.5 g: Computation performed by Process P8, P10, P12, P13

Step seven: At clock tick seven, we have the computation

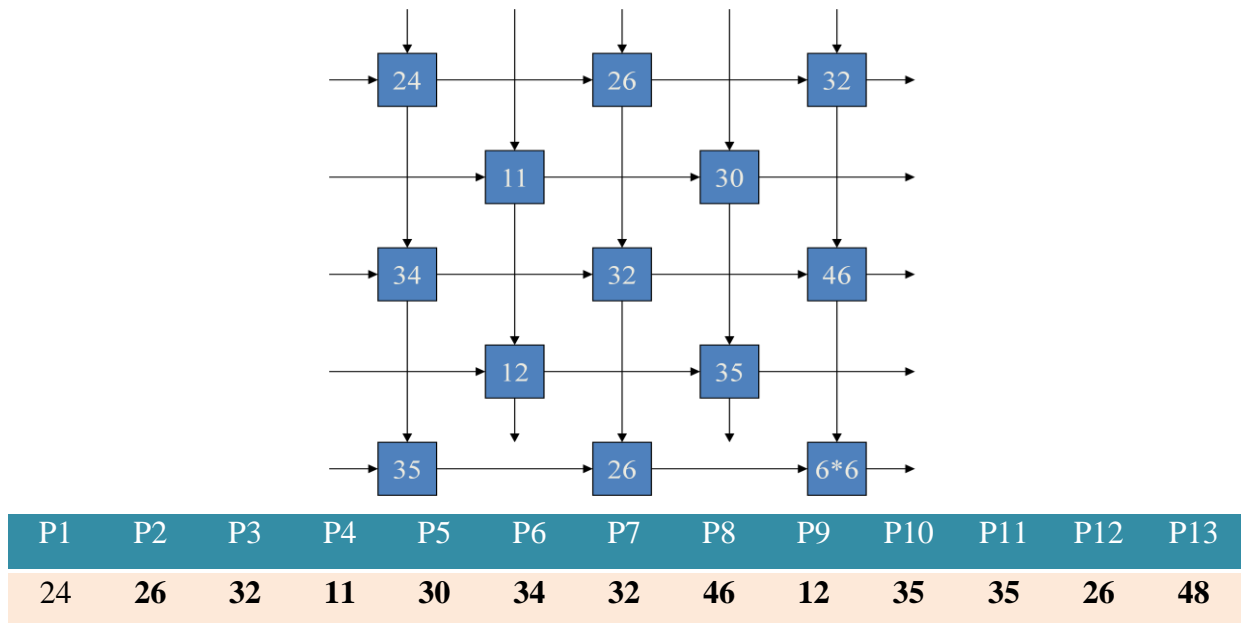


Figure 4.5h: Computation performed by Process P13

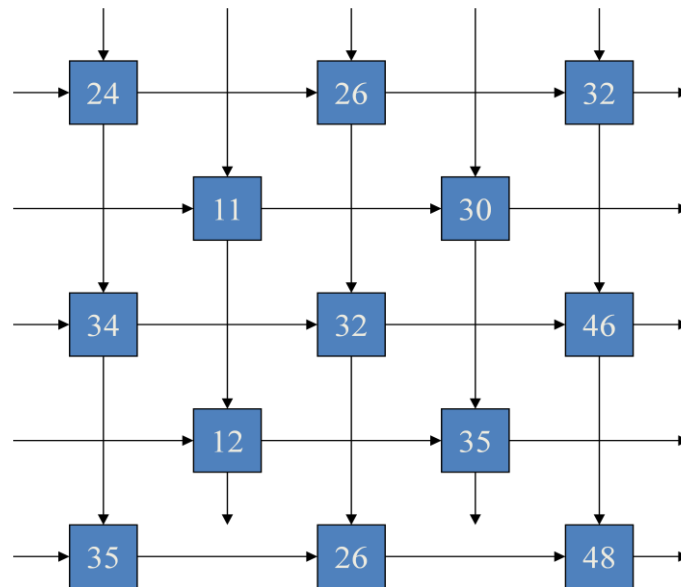


Figure 4.5 i: Row-column process array structure

Remark: by our new model the final result obtained is same as in $2n+3$ times, with an added advantage of reduction in the number of processing elements as compared to the model in Section 4.4 where we have 25 processing elements as against 13 in our case, which is a total slash of 12 processors.

4.5. Conclusion

In this chapter we have shown that it is realistic to carry out n-dimensional rhotrix-rhotrix multiplication by means of process array approach, which we believe offers a better and a faster computational solution to the row-column rhotrix multiplication compared to other existing approaches in parallel programming.

The concept used in this work can easily be extended to compute more complex functions, since it offers a way to make certain exponential algorithms and use hardware to make them linear. As the case may be, we believe that optimisation is very much possible, though not covered extensively within the scope of our work, we still believe it can be considered as a reference point for future works.

CHAPTER FIVE: SUMMARY, CONCLUSION AND FUTURE WORK

5.1. Summary and Conclusion

In this research, we have introduced some key mathematical models based on the generalization cases presented for each of the rhotrix multiplication methods adopted from Ajibade (2003) and Sani (2004). These models are of the form $C_i = b_{\lfloor \frac{n^2+3}{4} \rfloor} \times a_i + a_{\lfloor \frac{n^2+3}{4} \rfloor} \times b_i + (1-\lambda)(a_{\lfloor \frac{n^2+3}{4} \rfloor} \times b_{\lfloor \frac{n^2+3}{4} \rfloor})$, $C(i) = b_h \langle a_i \rangle + a_h \langle b_i \rangle (1-\lambda)$,

$$c_{i,j} = a_{i,j} \times a_{\lfloor \frac{n+1}{2} \lfloor \frac{n+3}{4} \rfloor \rfloor} + b_{i,j} \times a_{\lfloor \frac{n+1}{2} \lfloor \frac{n+3}{4} \rfloor \rfloor} (1-\lambda), \text{ and } C_{i,j} = \langle a_{i,j} \circ b_{i,j} \rangle = \lambda \sum_{k=1}^t (a_{i,k} \times b_{2k-1,j}) + (1-\lambda) \sum_{k=1}^{t-1} (a_{i,k} \times b_{2k,j}) \text{ for}$$

heart-oriented and row-column rhotrices multiplication. We presented several new sequential and parallel rhotrix multiplication algorithms that have no restriction on either rhotrix size or process grids and that support the data-distribution-independence (van de Velde and Lorenz 1989; Van de Valde 1994). In order to accomplish the research goal for rhotrix parallelization we formally defined and introduced some new concepts regarding rhotrix row-column and heart-oriented generalization expression. Similarly, knowing fully well, the complexity involved with data-distribution and data mapping for the object of interest, we defined several mapping functions and architectural design topology for rhotrix vectors to processing elements and also, we presented data structures for the traditional mapping of rhotrix entries to computer memories .

To fully understand the characteristics of the parallel rhotrix multiplication algorithms, performance model is built for the family of algorithms discussed in the thesis. The detailed analysis of the performance model revealed the correctness of the algorithms presented and also showed that no single algorithm always achieves the best performance for different rhotrix sizes and grid shapes, whose assessment and multiplication are viewed from different perspectives. Thus, the poly-algorithmic approach (Rice and Rosen, 1966) is needed to achieve high performance in terms of speed and space depending upon factors such as the problem size, grid shape, mapping pattern and amount of memory available.

We implemented a collection of sequential and parallel rhotrix multiplication algorithms using C language and exploited the object-based programming in Delphi for the systolic array simulator model. Some of the parallel algorithms were tested on the Intel duo core processor running MPI. The experimental results revealed the performance characteristics of each algorithm.

The research has presented several contributions to the study of rhotrix algebra and parallel library design. First, we introduced a generalized form of rhotrix structural representation covering both the heart-oriented and row-column multiplication expression and provided several algorithms for sequential and parallel rhotrix multiplication. The program user is left with the options of selecting suitable algorithm based on problem orientation. Secondly, we implemented a collection of parallel rhotrices multiplication algorithms, that have no restriction on rhotrices sizes and parallel platform on which they are executed upon. Furthermore, a fundamental concept of rhotrix parallelization has been established which would serve as a useful formalism for future parallel algorithm design extension.

5.2. Future Work

Rhotrix is a new branch of mathematical algebra similar to matrix in some way but having a unique property different from that of matrices; it is often referred to as super matrix. Thus, several aspects of research can be continued in the study of parallel rhotrices multiplication and its associated features. One of the algorithms with potential advantage for large rhotrix multiplication is the parallel row-column multiplication algorithm. This algorithm poses some fundamental challenges common to those of matrix multiplication, which we anticipate might have similar or even better application areas.

Another approach for parallel rhotrix multiplication is to use a three-dimensional process topology (Agarwal *et al.*, 1995). They claimed that the advantage of the 3D algorithm approach moves less data than the known 2D approach, thus the 3D approach can achieve better performance. However the 3D algorithm evidently requires incremental memory to replicate local rhotrices than those algorithms presented in this study do. Therefore there is a trade-off of speed versus space. There is need for future investigation of these algorithms beyond the scope covered in the present study.

Two important application areas of representing rhotrices as given in the row-column or 2-D indices nomenclature are rhotrix vector space where each row in a rhotrix is a vector, and in the information dissemination. The heart-oriented rhotrix is fast becoming an immense tool in computational parlance, especially in the area of queuing models of shared-memory parallel applications.

References

- Abdullahi, M. Ezugwu, E. A, Barroon, I. A. and Shehu, M. T., (2010), "Feasibility Analysis of using Cannon Method to Perform Block Rhotrix Multiplication," *International Journal of Computer Science and Information Security*, Vol. 8 No. 5, pp:231-239.
- Agarwal, R. C., S. M. Balle, F. G. Gustavson, M. Joshi, and P. Palker., (1995), "A Three-Dimensional Approach to Parallel Matrix Multiplication." *IBM Journal of Research and Development* 39 (5) 575-582.
- Aminu, A., (2010). "The equation $R_n x = b$ over rhotrices". *Int. J. Math. Educ. Sci. Technol.* 41, No. 1, 98-105.
- Ajibade, A. O. (2003), "The Concept of Rhotrix in Mathematical Enrichment," *International Journal of Mathematical Education in Science and Technology*, Vol. 34:2s, 175-179.
- Atanassov, K. T., and Shannon, A. G., (1998), *International Journal of Math. educ. Sci. Technol.*, 29, 898-903
- Bhat P.B., Raghavendra C.S., and Prasanna V.K., (1999), "Efficient Collective Communication in Distributed Heterogeneous Systems." In *ICDCS'99 19th International Conference on Distributed Computing Systems*, pages 15–24. IEEE Computer Society Press.
- Bhat P.B., Raghavendra C.S., and Prasanna V.K., (2003), "Efficient Collective Communication in Distributed Heterogeneous Systems." *Journal of Parallel and Distributed Computing*, 63:251–263.
- Blackford L.S., Choi J., Cleary A., D'Azevedo E., Demmel J., Dhillon I., Dongarra J., Hammarling S., Henry G, Petitet A., Stanley K., Walker D., and Whaley R.C., (1999) *ScaLAPACK Users' Guide*. SIAM. pp.122.
- Cannon L.E. (1969), 'A Cellular Computer to Implement the Kalman Filter Algorithm.' Ph.D. Thesis. Montana State University.
- Chou C., Deng Y., Li G. and Wang Y., (1995), "Parallelizing Strassen's Method for Matrix Multiplication on Distributed-Memory MIMD Architectures," *Computers for Mathematics with Applications*, 30(2):49.

- Douglas C., Heroux M., Sliselman G., and Smith R. M. (1994), A portable level 3 BLAS Winograd variant of Strassen's matrix-matrix multiply algorithm. *GEMMW: Journal of Computational Physics*, 110:1-10.
- Ezugwu E. Absalom., Ajibade O. A., and Sahalu B. Junaidu., (2011), "Algorithm Design for Row-Column Multiplication of N-Dimensional Rhotrices," *Global Journal of Computer Science and Technology*, Vol. 11, pp:58-72. USA, January.
- Ezugwu E. Absalom., Sani B., and Sahalu B. Junaidu., (2011), "The Concept of Heart-Oriented Rhotrix Multiplication," *Global Journal of Science Frontier Research*, Vol.11, pp:35-46. USA, March.
- Ezugwu E. Absalom., Abdullahi M. Ibrahim K., Mohammed A., and Sahalu B. Junaidu., (2010), "Parallel Multiplication of Rhotrices Using Systolic Array Architecture," *International Journal of Computer Information System*, Vol.1 pp:57-62, November.
- Foster, I.T., 1995, "Designing and building parallel programs: Concepts and tools for parallel software engineering." Reading, MA: Addison-Wesley Publishing Company.
- Fox G. C., Hey A. I. and Otto S., (1987), "Matrix Algorithms on the Hypercube I: Matrix Multiplication," *Parallel Computing* 4:17-31.
- Huss-Lederman, S., E. Jacobson and A. Tsao (1993). Comparison of scalable parallel matrix libraries. In *Proceedings of the Scalable Parallel Libraries Conference*, Starksville, MS. pp. 142-149.
- Huss-Lederman S. and Jacobson E., (1996), "Implementation of Strassen's Algorithm for Matrix Multiplication", *Proceedings of the 1996 ACM/IEEE conference on Supercomputing*, Pittsburgh.
- Hwang, K. And Briggs, F. A., (1984), "Computer Architecture and Parallel Processing" A graduate-level textbook by McGraw-Hill Book Co New York.
- IBM Engineering and Scientific Subroutine Library Version 3 Release 3 Guide and Reference. Document Number SA22-7272-04, IBM.

- Kumar, V., Grama, A., Gupta, A., and Karypis, G., 1994, "Introduction to parallel computing: Design and analysis of algorithms." Redwood City, CA: The Benjamin/Cummings Publishing Company.
- Kung, Y. S., and Gal-Ezer, J. R., "Synchronous and asynchronous computing in VLSI array processors," in Proc SPIE Conf (Arlington, VA), 1982.
- Kung, S. Y., Arun, K. S., Gal-Ezel, R. J., and Bhaskar Rao, D. V., "Wavefront array Processor: Language, architecture, and applications," IEEE Trans Comput (Special Issue on Parallel and Distributed Computers), Vol C-31, no 11, pp. 1054-1066, 1982
- Mead, C. and Conway I., "Introduction to VSLI Systems." Reading, MA: Addison-Wesley, 1980.
- Norman M.G., Thanisch P., (1993), "Models of Machines and Computation for mapping in multicomputer." ACM computing Surveys, 25(3); 263-302.
- Ohtaki Y., Takahashi D., Boku T., and Sato M., (2004), "Parallel Implementation of Strassen's Matrix Multiplication Algorithm for Heterogeneous Clusters," Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04), Santa Fe.
- Quin, Michael J. "Parallel Programming in C with MPI and OpenMP" New York: McGraw-Hill, 2003. pp.
- Rice, J. R., and S. Rosen. "Numerical Analysis Problem Solving System." In proceedings of 21st ACM National Conference, by the Association for Computing Machinery, 51-56. New York, NY: ACM Publications.
- Sani, B., (2004), An alternative method of multiplication of rhotrices, International Journal of Mathematical Education in Science and Technology, Vol.35:5, 777-781.
- Sani, B., (2007), "Conversion of a rhotrix to a 'coupled matrix'", International Journal of Mathematical Education in Science and Technology, Vol. 39:2, 244-249.
- Sani, B., (2008), "The Row-Column Multiplication of High Dimensional Rhotrices", International Journal of Mathematical Education in Science and Technology, Vol.35:5, 657-662.

- Singh, D., Ibrahim, A. M., Yohanna, T., Singh, J. N. (2007), An overview of the applications of multiset, Novi Sad J. Math, Vol. 37, No. 2, 73-92.
- Strassen,V., (1969), "Gaussian Elimination is not Optimal", Numerische Mathematik, 13:354-356.
- Ullman, J. D, Computational Aspects of VLSI. Rockville, MD: Computer Science Press, 1984.
- Van de Velde, E.F. and J. Lorenz., (1989, "Adaptive data distribution for concurrent continuation." Pasadena, C.A: California Institute of Technology, Caltech/ Rice Center for Research in Parallel Computation. Technical Report CRPC: 89-4.
- Van de Velde, E.F., (1994), Concurrent Scientific Computing. New York, NY: Springer-Verlag New York, Inc. pp162.
- Wilkinson B. And Allen M. (2005), "Parallel Programming, Techniques and Applications Using Networked Workstations and Parallel Computers." Second Edition. Pearson Prentice Hall. pp350-351.

Appendix A

Source Code

In this appendix, we present the research program source codes we developed. We implemented the program based on C and MPI Library. Section A.1 explained the organization of the source code. In Section A.2, the listing of the source code was presented.

A.1 Organization of Source Code

Each implementation is organized as shown in Table A.1.

Table A.1: Organization of source code

Name	Source Code
Sequential Row Column Rhotrix Multiplication (BS1)	Rhotrix_coupled.c
Sequential Row Column Rhotrix Multiplication (BS2)	Row_col.c
Sequential Heart-Oriented Multiplication (SAJ)	Rhotrix_prod.c
Strassen's Row-column Multiplication (BS3)	Strassen.java
Parallel Heart-Oriented Rhotrix Multiplication in MPI (PAJ)	MPI_rhot_prod.c
Parallel Coupled Matrix Multiplication in MPI (PBS)	MPI_coupled.c
Parallel Row Column Rhotrix Multiplication in Process Array	SystolicRhotrixMultiplication.dfm, SystolicRhotrixMultiplication.pas

A.2 Source Code

A.2.1 Rhotrix_coupled.c

```
/*-----  
SEQUENTIAL RHOTRIX PROBLEM IN C LANGUAGE  
  
PROGRAMMED BY EZUGWU, El-Shamir Absalom  
Department of Mathematics  
Faculty of Sciences  
Ahmadu Bello University  
Samarua, Zaria  
Kaduna State  
  
SUPERVISED BY PROF. S. Juniadu  
AHMADU BELLO UNIVERSITY  
PROGRAMMED ON FEB, 2010  
  
PROGRAM MODE: COUPLED MATRIX  
-----*/  
  
#include<stdlib.h>  
#include<stdio.h>
```

```

#include <ctype.h>
#include<time.h>
#include <conio.h>
int ra[100][100],rb[100][20],r[100][100],i,j,k,m,n,p,q;
const char name;
char SW;
FILE *fp;
int IERROR;
char ERR_MSG[100];

/*****
TITLE
prints the program title and asks whether
interactive mode or file mode is used for input.
*****/
void TITLE()
{
    printf(" *****\n");
    printf("    WELCOME TO SEQUENTIAL RHOTRIX SOLVER\n");
    printf("    B. SANI'S METHOD BY PADDING\n");
    printf(" *****\n");
    do{
        printf(" YOU CAN ENTER YOUR DATA BY\n\n");
        printf(" > INTERACTIVE MODE ----- I\n\n");
        printf(" > FILE MODE ----- F\n\n");
        printf(" IF YOU WANT FILE MODE, \n");
        printf(" MAKE A FILE CONTAINING DATA BEFORE SELECTING F. \n");
        printf(" ARE YOU READY ? NOW\n\n");
        printf(" SELECT I OR F ! ");
        SW = getchar();
        if (SW != 'i' && SW != 'I' && SW != 'f' && SW != 'F'){
            printf("YOU TYPED A WRONG CHARACTER !\n");
            printf("SELECT AGAIN WITH CARE !\n");
        }
    } while(SW != 'i' && SW != 'I' && SW != 'f' && SW != 'F');
}

/*****
FILE MODE
reads data from a data file
*****/
void FILEMODE() /* function FILE INPUT */
{
    int i, j;
    char filename[60];
    char filename2[60];

    FILE *fin;
    printf("\n\n Write the first input file name ? ");

```

```

scanf("%s",filename);
if ( (fin=fopen(filename,"rt"))==NULL ) {
    puts("\n\n  Caution !!! \n");
    sprintf(ERR_MSG,"%s Cannot open file ",filename);
    IERROR=2; return;
}
else {
    fscanf(fin, "%d %d", &m,&n);
    for (i=1 ; i <= m+1; i++)
        for (j=1 ; j <= n+1; j++)
            fscanf(fin, "%d", &ra[i][j]);

}
printf("\n\n Write the scond input file name ? ");
scanf("%s",filename2);
if ( (fin=fopen(filename2,"rt"))==NULL ) {
    puts("\n\n  Caution !!! \n");
    sprintf(ERR_MSG,"%s Cannot open file ",filename2);
    IERROR=2; return;
}
else {
    fscanf(fin, "%d %d", &p,&q);
    for (i=1 ; i <= p+1; i++)
        for (j=1 ; j <= q+1; j++)
            fscanf(fin, "%d", &rb[i][j]);
}

fclose(fin);
}
/*****
INTERACTIVE MODE
reads data interactively.
*****/
void INTEMODE() /* function INTERACTIVE INPUT */
{
    int i, j;
    char response;
    printf("\n\n\n");
    printf(" ***** \n");
    printf(" INTERACTIVE INPUT \n");
    printf(" ***** \n\n\n");
    do{
printf("Enter The Rows And Cloumns Of The First Rhotrix:");
scanf("%d %d",&m,&n);
printf("\n IS THE DATA CORRECT ? <Y/N>\n");
    getchar();response = getchar();
    }while (response == 'N' || response == 'n');
do{
printf("\nEnter Elements Of The First Rhotrix:\n");

```

```

for(i=1;i< m+1;i++)
{
for(j=1;j< n+1;j++)
scanf("%d",&ra[i][j]);
}
printf("\n IS THE DATA CORRECT ? <Y/N> \n");
getchar();response = getchar();
}while ( response == 'N' || response == 'n');
do{
printf("\nEnter The Rows And Cloumns Of The Second Rhotrix:");
scanf("%d %d",&p,&q);
printf("\n IS THE DATA CORRECT ? <Y/N> \n");
getchar();response = getchar();
}while ( response == 'N' || response == 'n');
do{
printf("\nEnter Elements Of The Second Rhotrix:\n");
for(i=1;i< p+1;i++)
{
for(j=+1;j< q+1;j++)
scanf("%d",&rb[i][j]);
}
printf("\n IS THE DATA CORRECT ? <Y/N> \n");
getchar();response = getchar();
}while ( response == 'N' || response == 'n');
}

/*****
ERROR_HANDLE
*****/
void error_handle()
{
/*****/
/* Error Code */
/*****/
/* IERROR=0 No Error */
/* IERROR=1 Normal Exit */
/* IERROR=2 File Open Error */
/* IERROR=3 problem is infeasible */
/*****/

if (IERROR==0) return;
else {
printf("\n%s\n",ERR_MSG);
fprintf(fp,"\n%s\n",ERR_MSG);
fclose(fp); exit(0);
}
}
int main(void)
{

```

```

//clrscr();
fp =fopen("rhotrixpad.out","wt");
printf("\nSerial Processing : Rhotrix Multiplication\n");
    fprintf(fp, "\nEZUGWU, El-Shamir Absalom MSC/SCIE/00043/2008-2009\n");
    fprintf(fp, "\nDate : %s", __DATE__);
    fprintf(fp, "\nTime : %s", __TIME__);

fprintf(fp, "\n=====
=====\n");
do{
    TITLE();
    if (SW == 'T' || SW == 'i') { INTEMODE(); break; }
    else if(SW == 'F' || SW == 'f') { FILEMODE(); break; }
} while(1);
error_handle();

fprintf(fp, "The First Rhotrix R(A)[%1d,%1d] Is:\n",m, n);
for(i=1;i< m+1;i++)
{
for(j=1;j< n+1;j++)
fprintf(fp, " R(A)[%1d, %1d] = %6d ", i, j, ra[i][j]); //print the first rhotrix
fprintf(fp, "\n");
}
fprintf(fp, "The Second Rhotrix R(B)[%1d,%1d]Is:\n", m, n);
for(i=1;i< p+1;i++) // print the second rhotrix
{
for(j=1;j< q+1;j++)
fprintf(fp, " R(B)[%1d, %1d] = %6d ", i, j, rb[i][j]);
fprintf(fp, "\n");
}
if(n!=p)
{
printf("Aborting!!!!/nMultiplication Of The Above Rhotrices Not Possible.");
exit(0);
}
else
{
for(i=1;i< m+1;i++)
{
for(j=1;j< q+1;j++)
{
r[i][j] = 0;
for(k=1;k< n+1;k++)
{
r[i][j] = r[i][j] + ra[i][k] * rb[k][j];
}
}
}
}
}

```

```

fprintf(fp, "\nMultiplication Of The Above Two Rhotrices Are:\n\n");
for(i=1; i < m+1; i++)
{
for(j=1; j < q+1; j++)
{
fprintf(fp, " R(C)[%1d, %1d] = %6d ", i, j, r[i][j]);
}
}
fprintf(fp, "\n");
}
}
getch();
return 0;
}

```

A.2.2 Row_col.c

```

/*-----
SEQUENTIAL RHOTRIX PROBLEM IN C LANGUAGE

PROGRAMMED BY EZUGWU, El-Shamir Absalom
Department of Mathematics
Faculty of Sciences
Ahmadu Bello University
Samarua, Zaria
Kaduna State

SUPERVISED BY PROF. S. Juniadu
AHMADU BELLO UNIVERSITY
PROGRAMMED ON FEB, 2010
ROW-COLUMN RHOTRIX PRODUCT
-----*/

#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <time.h>
#include <conio.h>
#define START if ( (startm = clock()) == -1) {printf("Error calling clock");exit(1);}
#define STOP if ( (stopm = clock()) == -1) {printf("Error calling clock");exit(1);}
#define PRINTTIME fprintf(fp, "%6.3f seconds used by the processor.", ((double)stopm-
startm)/CLOCKS_PER_SEC);
int ra[100][100], rb[100][100], r[100][100], i, j, k, n, p, t, s;
const char name;
long int time_count = 0;
clock_t startm, stopm;
char SW;
FILE *fp;
int IERROR;
char ERR_MSG[100];
clock_t clock_start;
/*****

```

TITLE

prints the program title and asks whether interactive mode or file mode is used for input.

```
*****/
void TITLE()
{
    printf(" *****\n");
    printf("  WELCOME TO SEQUENTIAL RHOTRIX SOLVER \n");
    printf("    RHOTRIX ROW-COLUMN SOLVER  \n");
    printf(" *****\n\n");
    do{
        printf(" YOU CAN ENTER YOUR DATA BY \n\n");
        printf("  > INTERACTIVE MODE ----- I \n\n");
        printf("  > FILE MODE ----- F \n\n");
        printf(" IF YOU WANT FILE MODE, \n");
        printf(" MAKE A FILE CONTAINING DATA BEFORE SELECTING F. \n");
        printf(" ARE YOU READY ? NOW \n\n");
        printf(" SELECT I OR F ! ");
        SW = getchar();
        if (SW != 'i' && SW != 'I' && SW != 'f' && SW != 'F'){
            printf("YOU TYPED A WRONG CHARACTER !\n");
            printf("SELECT AGAIN WITH CARE !\n");
        }
    } while(SW != 'i' && SW != 'I' && SW != 'f' && SW != 'F');
}

```

```
*****/
```

FILE MODE

reads data from a data file

```
*****/
void FILEMODE() /* function FILE INPUT */
{
    int i, j;
    char filename[100];
    char filename2[100];

    FILE *fin;
    printf("\n\n Write the input file name ? ");
    scanf("%s",filename);
    if ( (fin=fopen(filename,"rt"))==NULL ) {
        puts("\n\n Caution !!! \n");
        sprintf(ERR_MSG,"%s Cannot open file ",filename);
        IERROR=2; return;
    }
    else {
        fscanf(fin, "%d %d", &n,&t);
        for (i=1 ; i <= n; i++)
            for (j=1 ; j <= t; j++)
                fscanf(fin, "%d", &ra[i][j]);
    }
}

```

```

}
printf("\n\n Write the sccond input file name ? ");
scanf("%s",filename2);
if ( (fin=fopen(filename2,"rt"))==NULL ) {
    puts("\n\n Caution !!! \n");
    sprintf(ERR_MSG,"%s Cannot open file ",filename2);
    IERROR=2; return;
}
else {
    fscanf(fin, "%d %d", &n,&t);
    for (i=1 ; i <= n; i++)
        for (j=1 ; j <= t; j++)
            fscanf(fin, "%d",&rb[i][j]);
}

fclose(fin);
}
/*****
INTERACTIVE MODE
reads data interactively.
*****/
void INTEMODE() /* function INTERACTIVE INPUT */
{
    int i, j;
    char response;
    printf("\n\n\n");
    printf(" ***** \n");
    printf(" INTERACTIVE INPUT \n");
    printf(" ***** \n\n");
    do{
printf("Enter The Rows And Cloumns Of The First Rhotrix:");
scanf("%d %d",&n,&t);
printf("\n IS THE DATA CORRECT ? <Y/N>\n");
    getchar();response = getchar();
    }while (response == 'N' || response == 'n');
do{
printf("\nEnter Elements Of The First Rhotrix:\n");
for(i=1;i< n+1;i++)
{
for(j=1;j< t+1;j++)
scanf("%d",&ra[i][j]);
}
printf("\n IS THE DATA CORRECT ? <Y/N> \n");
getchar();response = getchar();
}while ( response == 'N' || response == 'n');
do{
printf("\nEnter The Rows And Cloumns Of The Second Rhotrix:");
scanf("%d %d",&n,&t);
printf("\n IS THE DATA CORRECT ? <Y/N> \n");

```

```

    getchar();response = getchar();
}while ( response == 'N' || response == 'n');
do{
printf("\nEnter Elements Of The Second Rhotrix:\n");
for(i=1;i< n+1;i++)
{
for(j=1;j< t+1;j++)
scanf("%d",&rb[i][j]);
}
printf("\n IS THE DATA CORRECT ? <Y/N> \n");
getchar();response = getchar();
}while ( response == 'N' || response == 'n');
}

/*****
ERROR_HANDLE
*****/
void error_handle()
{
/*****/
/* Error Code          */
/*****/
/* IERROR=0 No Error      */
/* IERROR=1 Normal Exit   */
/* IERROR=2 File Open Error */
/* IERROR=3 problem is infeasible */
/*****/

if (IERROR==0) return;
else {
printf("\n%s\n",ERR_MSG);
fprintf(fp,"\n%s\n",ERR_MSG);
fclose(fp); exit(0);
}
}
int main(void)
{

//clrscr();
START;
fp =fopen("rhotbs1.out","wt");
printf("\nSerial Processing : Rhotrix Multiplication\n");
fprintf(fp,"\nEZUGWU, El-Shamir Absalom MSC/SCIE/00043/2008-2009\n");
fprintf(fp,"\nDate : %s",__DATE__);
fprintf(fp,"\nTime : %s",__TIME__);

fprintf(fp,"\n=====
=====\n");
do{

```

```

    TITLE();
    if (SW == 'T' || SW == 'i') { INTEMODE(); break; }
    else if (SW == 'F' || SW == 'f') { FILEMODE(); break; }
} while(1);
error_handle();

fprintf(fp, "The First Rhotrix R(A)[%1d,%1d] Is:\n", n, t);
for(i=1; i < n+1; i++)
{
    for(j=1; j < t+1; j++)
    fprintf(fp, " R(A)[%1d, %1d] = %6d ", i, j, ra[i][j]); //print the first rhotrix
    fprintf(fp, "\n");
}
fprintf(fp, "The Second Rhotrix R(B)[%1d,%1d]Is:\n", n, t);
for(i=1; i < n+1; i++) // print the second rhotrix
{
    for(j=1; j < t+1; j++)
    fprintf(fp, " R(B)[%1d, %1d] = %6d ", i, j, rb[i][j]);
    fprintf(fp, "\n");
}
/* count loops for one second, used to time rhotrix operation*/
    clock_start = clock();
while(((clock() - clock_start) / CLOCKS_PER_SEC) < 1)
    time_count++;
    fprintf(fp, "\ntime_count = %ld", time_count/1000000);
for(i=1; i < n+1; i++)
{
    if(i%2==1){
    for(j=1; j < t+1; j++)
    {
        r[i][j] = 0;
        for(k=1; k < t+1; k++)
        {
            r[i][j] = r[i][j] + ra[i][k] * rb[2*k-1][j];
        }
    }
}
for(i=1; i < n+1; i++)
{
    if(i%2==0){
    for(j=1; j < t+1; j++)
    {
        r[i][j] = 0;
        for(k=1; k < t; k++)
        {
            r[i][j] = r[i][j] + ra[i][k] * rb[2*k][j];
        }
    }
}
}
}

```

```

}
}
STOP;
fprintf(fp, "\nMultiplication Of The Above Two Rhotrices Are:\n\n");
for(i=1; i < n+1; i++)
{
for(j=1; j < t+1; j++)
{
fprintf(fp, "R(C)[%1d, %1d] = %6d ", i, j, r[i][j]);
}
fprintf(fp, "\n");
}
PRINTTIME;
getch();
return 0;
}

```

A.2.3 Rhotrix_prod.c

```

/*-----
SEQUENTIAL RHOTRIX PROBLEM IN C LANGUAGE

PROGRAMMED BY EZUGWU, El-Shamir Absalom
Department of Mathematics
Faculty of Sciences
Ahmadu Bello University
Samarua, Zaria
Kaduna State

SUPERVISED BY PROF. S. Juniadu
AHMADU BELLO UNIVERSITY
PROGRAMMED ON FEB, 2010

PROGRAM MODE: HEART-ORIENTED RHOTRIX MULTIPLICATION
-----*/
#include<stdlib.h>
#include<stdio.h>
#include<ctype.h>
#include<math.h>
#include<time.h>
#include <conio.h>
#define START if ( (startm = clock()) == -1) {printf("Error calling clock");exit(1);}
#define STOP if ( (stopm = clock()) == -1) {printf("Error calling clock");exit(1);}
#define PRINTTIME fprintf(fp, "%6.3f seconds used by the processor.", ((double)stopm-
startm)/CLOCKS_PER_SEC);
int ra[1000],rb[1000],r[1000],ha,hb,i,j,k,m,n;
clock_t startm, stopm;
//const char name;
char SW;
FILE *fp;

```

```

int IERROR;
char ERR_MSG[100];

/*****
TITLE
prints the program title and asks whether
interactive mode or file mode is used for input.
*****/
void TITLE()
{
    printf(" *****\n");
    printf("  WELCOME TO SEQUENTIAL RHOTRIX PROBLEM SOLVER \n");
    printf("  RHOTRIX SINGLE TO ALL MULTIPLICATION SOLVER  \n");
    printf(" *****\n\n");
    do{
        printf("  YOU CAN ENTER YOUR DATA BY \n\n");
        printf("    > INTERACTIVE MODE ----- I \n\n");
        printf("    > FILE MODE ----- F \n\n");
        printf("  IF YOU WANT FILE MODE, \n");
        printf("  MAKE A FILE CONTAINING DATA BEFORE SELECTING F. \n");
        printf("  ARE YOU READY ? NOW \n\n");
        printf("  SELECT I OR F ! ");
        SW = getchar();
        if (SW != 'i' && SW != 'I' && SW != 'f' && SW != 'F'){
            printf("YOU TYPED A WRONG CHARACTER !\n");
            printf("SELECT AGAIN WITH CARE !\n");
        }
    } while(SW != 'i' && SW != 'I' && SW != 'f' && SW != 'F');
}

/*****
FILE MODE
reads data from a data file
*****/
void FILEMODE() /* function FILE INPUT */
{
    int i, j;
    char filename[60];
    char filename2[60];

    FILE *fin;
    printf("\n\n Write the input file name ? ");
    scanf("%s",filename);
    if ( (fin=fopen(filename,"rt"))==NULL ) {
        puts("\n\n Caution !!! \n");
        sprintf(ERR_MSG,"%s Cannot open file ",filename);
        IERROR=2; return;
    }
    else {

```

```

    fscanf(fin, "%d %d", &n,&ha);
    for (i=1 ; i <= (0.5*(pow(n,2)+1))+1; i++)
        fscanf(fin, "%d", &ra[i]);

}
printf("\n\n Write the scond input file name ? ");
scanf("%s",filename2);
if ( (fin=fopen(filename2,"rt"))==NULL ) {
    puts("\n\n Caution !!! \n");
    sprintf(ERR_MSG,"%s Cannot open file ",filename2);
    IERROR=2; return;
}
else {
    fscanf(fin, "%d %d", &n,&hb);
    for (i=1 ; i <= (0.5*(pow(n,2)+1))+1; i++)
        fscanf(fin, "%d", &rb[i]);
}

fclose(fin);
}
/*****
    INTERACTIVE MODE
reads data interactively.
*****/
void INTEMODE() /* function INTERACTIVE INPUT */
{
    int i;
    char response;
    printf("\n\n\n");
    printf(" ***** \n");
    printf(" INTERACTIVE INPUT \n");
    printf(" ***** \n\n");
    do{
        printf("Enter The Dimension and heart Of The First Rhotrix:");
        scanf("%d %d",&n, &ha);
        printf("\n IS THE DATA CORRECT ? <Y/N>\n");
        getchar();response = getchar();
        }while (response == 'N' || response == 'n');
    do{
        printf("\nEnter Elements Of The First Rhotrix:\n");
        for(i=1;i < (0.5*(pow(n,2)+1))+1;i++)
        {
            scanf("%d",&ra[i]);
        }
        printf("\n IS THE DATA CORRECT ? <Y/N> \n");
        getchar();response = getchar();
        }while ( response == 'N' || response == 'n');
    do{
        printf("\nEnter The Dimension and heart Of The Second Rhotrix:");

```

```

scanf("%d %d",&n, &hb);
    printf("\n IS THE DATA CORRECT ? <Y/N> \n");
    getchar();response = getchar();
    }while ( response == 'N' || response == 'n');
do{
printf("\nEnter Elements Of The Second Rhotrix:\n");
for(i=1;i< (0.5*(pow(n,2)+1))+1;i++)
{
scanf("%d",&rb[i]);
}
    printf("\n IS THE DATA CORRECT ? <Y/N> \n");
    getchar();response = getchar();
    }while ( response == 'N' || response == 'n');
}

/*****
ERROR_HANDLE
*****/
void error_handle()
{
/*****/
/* Error Code */
/*****/
/* IERROR=0 No Error */
/* IERROR=1 Normal Exit */
/* IERROR=2 File Open Error */
/* IERROR=3 problem is infeasible */
/*****/

if (IERROR==0) return;
else {
    printf("\n%s\n",ERR_MSG);
    fprintf(fp,"\n%s\n",ERR_MSG);
    fclose(fp); exit(0);
}
}
int main(void)
{
int i;
START;
fp =fopen("rhotAji.out","wt");
printf("\nSerial Processing : Rhotrix Multiplication\n");
    fprintf(fp,"\nEZUGWU, El-Shamir Absalom MSC/SCIE/00043/2008-2009\n");
    fprintf(fp,"\nDate : %s",__DATE__);
    fprintf(fp,"\nTime : %s",__TIME__);

fprintf(fp,"\n=====
=====\n");
do{

```

```

    TITLE();
    if (SW == 'T' || SW == 'i') { INTEMODE(); break; }
    else if(SW == 'F' || SW == 'f') { FILEMODE(); break; }
} while(1);
error_handle();

fprintf(fp, "The First Rhotrix R(A)[%1d] Is:\n", m);
for(i=1; i < (0.5*(pow(n,2)+1))+1; i++)
{
fprintf(fp, " R(A)[%1d] = %6d ", i, ra[i]); //print the first rhotrix
fprintf(fp, "\n");
}
fprintf(fp, "The Second Rhotrix R(B)[%1d]Is:\n", m);
for(i=1; i < (0.5*(pow(n,2)+1))+1; i++) // print the second rhotrix
{
fprintf(fp, " R(B)[%1d] = %6d ", i, rb[i]);
fprintf(fp, "\n");
}
for(i=1; i < (0.5*(pow(n,2)+1))+1; i++)
{
if(i==0.5*(pow(n,2)+1)/2)
{
r[i] = hb*ha;
}
else
{
r[i] = ra[i]*hb + rb[i]*ha;
}
}
fprintf(fp, "\nMultiplication Of The Above Two Rhotrices Are:\n\n");
for(i=1; i < (0.5*(pow(n,2)+1))+1; i++)
{
fprintf(fp, " R[%1d] = %6d ", i, r[i]);
fprintf(fp, "\n");
}
STOP;
PRINTTIME;
getch();
return 0;
}

```

A.2.4 Strassen.java

```

/* -----
* Rhotrix Multiplication - Java Application Version
* Algorithm Implementation: Strassen's
* Multiplication Method: Peeling
*

```

```

* Version    : Still in Progress
* Written: Date: January, 2nd 2011
*
* Author: Ezugwu El-Shamir Absalom
*      MSC/SCI/0043/2008-2009
*      Department of Mathematics
*
* Institution: Ahmadu Bello University, Zaria
* Supervised under Prof. S.B. Junaidu
*
* -----
*/

```

```

import java.io.*;
import java.util.Calendar;
import java.util.StringTokenizer;
public class Strassen
{
    public static String [][] matrix;
    public Strassen()
    {
        String stringone="i";
        String stringtwo="I";
        String stringthree="f";
        String stringfour="F";
        String SW;
        int n;
        int k=1;
        int[][] a, b, c;
        do{
            title();
            SW = readInput();

```

```

        if (stringone.compareTo(SW)== 0 // stringtwo.compareTo(SW)== 0) { interMode();
break; }

        else if(stringthree.compareTo(SW)== 0 // stringfour.compareTo(SW)== 0) { fileMode();
break; }

    } while(k==1);
    System.exit(-1);

}

public void title()
{
    String stringone="i";
    String stringtwo="I";
    String stringthree="f";
    String stringfour="F";
    String SW;
    do{
System.out.println(" *****\n");
System.out.println("    WELCOME TO SEQUENTIAL RHOTRIX SOLVER \n");
System.out.println("    STRASSEN'S ALGORITHM  \n");
System.out.println(" *****\n\n");
System.out.println(" YOU CAN ENTER YOUR DATA BY \n\n");
System.out.println(" > INTERACTIVE MODE ----- I \n\n");
System.out.println(" > FILE MODE ----- F \n\n");
System.out.println(" IF YOU WANT FILE MODE, \n");
System.out.println(" MAKE A FILE CONTAINING DATA BEFORE SELECTING F.
\n");
System.out.println(" ARE YOU READY? NOW \n\n");
System.out.println(" SELECT I OR F ! ");

        SW = readInput();

        if (stringone.compareTo(SW)!= 0 && stringtwo.compareTo(SW)!= 0 &&
stringthree.compareTo(SW)!= 0 && stringfour.compareTo(SW)!= 0)
        {
            System.out.println("YOU TYPED A WRONG CHARACTER !\n");

```

```

        System.out.println("SELECT AGAIN WITH CARE !\n");
    }
}while(stringone.compareTo(SW)!= 0 && stringtwo.compareTo(SW)!= 0 &&
stringthree.compareTo(SW)!= 0 && stringfour.compareTo(SW)!= 0);
}

public void interMode()
{
    int[][] a, b, c;
    String br;
    System.out.println("Enter the number of rows and columns: ");
    String colRow = readInput();
    int n=Integer.parseInt(colRow);
    a = new int[n][n];
    b = new int[n][n];
    c = new int[n][n];

    System.out.println("\nEnter entries for first Rhotrix:\n");
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
        {
            System.out.println("Enter the value of
cell("+(i+1)+" "+(j+1)+"): ");
            String cell = readInput();
            a[i][j]=Integer.parseInt(cell);
        }
    }
    System.out.println("\nEnter entries for second Rhotrix:\n");
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
        {

```

```

        System.out.println("Enter the value of cell
        ("+(i+1)+", "+(j+1)+"): ");
        String cell = readInput();
        b[i][j]=Integer.parseInt(cell);
    }
}
Calendar t1 = Calendar.getInstance();
System.out.println("\n Rhotrix 1 X Rhotrix 2 is:\nBy Standard method:\n");
standardRhotrixMultiplication(a, b, c);
printArray(c);
Calendar t2 = Calendar.getInstance();
System.out.println("Time taken to solve: "+(t2.getTimeInMillis() -
t1.getTimeInMillis() ));

    t1 = Calendar.getInstance();
    System.out.println("\nBy Strassen Rhotrix Multiplication method:\n");
    c = strassenRhotrixMultiplication(a, b);
    printArray(c);
    t2 = Calendar.getInstance();
    System.out.println("Time taken to solve: "+(t2.getTimeInMillis() -
t1.getTimeInMillis() ));
}
public void fileMode()
{
try {
    BufferedReader fileName = new BufferedReader(new FileReader("C:/mdata.txt"));

String line;

    int order = 0;

    int rowIndex = 0;
    int counter = 0;

```

```

while ((line = fileName.readLine()) != null) {
    counter++;
    if(counter == 1){
        order = Integer.parseInt(line);
        matrix = new String [order][order];
        System.out.println("order: " + order);
    }

    if(counter == 2){

        String source = line;
        System.out.println("source: " + source);
    }

    if(counter !=2 && counter !=1){
        StringTokenizer theLine = new StringTokenizer(line);
        int colIndex = 0;
        while(theLine.hasMoreTokens()){
            String st = theLine.nextToken();//.trim();
            matrix [rowIndex][colIndex] = st;
            colIndex = colIndex + 1;
        }
        rowIndex = rowIndex + 1;
    }

}

for(int x = 0; x<matrix.length; x++){
    for(int p = 0; p<matrix.length; p++){
        System.out.println(matrix[x][p] + " ");
    }
}

```

```

fileName.close();
}
catch (IOException e) {}
}

private static String readInput() {
    try {
        BufferedReader in = new BufferedReader(
            new InputStreamReader(System.in));
        return in.readLine();
    }
    catch (IOException e) {}
    return "";
}

public void standardRhotrixMultiplication(int[][] a, int[][] b, int[][] c)
{
    int n = a.length;
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            for(int k=0; k<n; k++)
                c[i][j] += a[i][k] * b[k][j];
}

public int [][] strassenRhotrixMultiplication(int [][] A, int [][] B)
{
    int n = A.length;
    int [][] result = new int[n][n];

    if((n%2 != 0) && (n != 1))
    {
        int[][] a1, b1, c1;
        int n1 = n+1;
        a1 = new int[n1][n1];
    }
}

```

```

    b1 = new int[n1][n1];
    c1 = new int[n1][n1];

    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            {
                a1[i][j] =A[i][j];
                b1[i][j] =B[i][j];
            }
    c1 = strassenRhotrixMultiplication(a1, b1);
    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            result[i][j] =c1[i][j];
    return result;
}

if(n == 1)
{
    result[0][0] = A[0][0] * B[0][0];
}
else
{
    int [][] A11 = new int[n/2][n/2];
    int [][] A12 = new int[n/2][n/2];
    int [][] A21 = new int[n/2][n/2];
    int [][] A22 = new int[n/2][n/2];

    int [][] B11 = new int[n/2][n/2];
    int [][] B12 = new int[n/2][n/2];
    int [][] B21 = new int[n/2][n/2];
    int [][] B22 = new int[n/2][n/2];

    divideArray(A, A11, 0 , 0);

```

```

    divideArray(A, A12, 0, n/2);
    divideArray(A, A21, n/2, 0);
    divideArray(A, A22, n/2, n/2);

    divideArray(B, B11, 0, 0);
    divideArray(B, B12, 0, n/2);
    divideArray(B, B21, n/2, 0);
    divideArray(B, B22, n/2, n/2);

    int [][] P1 = strassenRhotrixMultiplication(addRhotrices(A11, A22),
addRhotrices(B11, B22));
    int [][] P2 = strassenRhotrixMultiplication(addRhotrices(A21, A22),
B11);
    int [][] P3 = strassenRhotrixMultiplication(A11,
subtractRhotrices(B12, B22));
    int [][] P4 = strassenRhotrixMultiplication(A22,
subtractRhotrices(B21, B11));
    int [][] P5 = strassenRhotrixMultiplication(addRhotrices(A11, A12),
B22);
    int [][] P6 = strassenRhotrixMultiplication(subtractRhotrices(A21,
A11), addRhotrices(B11, B12));
    int [][] P7 = strassenRhotrixMultiplication(subtractRhotrices(A12,
A22), addRhotrices(B21, B22));

    int [][] C11 = addRhotrices(subtractRhotrices(addRhotrices(P1, P4),
P5), P7);
    int [][] C12 = addRhotrices(P3, P5);
    int [][] C21 = addRhotrices(P2, P4);
    int [][] C22 = addRhotrices(subtractRhotrices(addRhotrices(P1, P3),
P2), P6);

    copySubArray(C11, result, 0, 0);
    copySubArray(C12, result, 0, n/2);

```

```

        copySubArray(C21, result, n/2, 0);
        copySubArray(C22, result, n/2, n/2);
    }
    return result;
}

public int [][] addRhotrices(int [][] A, int [][] B)
{
    int n = A.length;

    int [][] result = new int[n][n];

    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            result[i][j] = A[i][j] + B[i][j];

    return result;
}

public int [][] subtractRhotrices(int [][] A, int [][] B)
{
    int n = A.length;

    int [][] result = new int[n][n];

    for(int i=0; i<n; i++)
        for(int j=0; j<n; j++)
            result[i][j] = A[i][j] - B[i][j];

    return result;
}

public void divideArray(int[][] parent, int[][] child, int iB, int jB)

```

```

    {
        for(int i1 = 0, i2=iB; i1<child.length; i1++, i2++)
            for(int j1 = 0, j2=jB; j1<child.length; j1++, j2++)
                {
                    child[i1][j1] = parent[i2][j2];
                }
    }

public void copySubArray(int[][] child, int[][] parent, int iB, int jB)
{
    for(int i1 = 0, i2=iB; i1<child.length; i1++, i2++)
        for(int j1 = 0, j2=jB; j1<child.length; j1++, j2++)
            {
                parent[i2][j2] = child[i1][j1];
            }
}

public void printArray(int [][] array)
{
    int n = array.length;

    System.out.println();
    for(int i=0; i<n; i++)
    {
        for(int j=0; j<n; j++)
        {
            System.out.print("R["+(i+1)+", "+(j+1)+"] = "+array[i][j] +
"\t");
        }
        System.out.println();
    }
    System.out.println();
}

```

```

        public static void main(String[] args) throws IOException
        {
            new Strassen();
        }
    }

```

A.2.5 MPI_rhot_prod.c

```

#include <stdio.h>
#include <math.h>
#include <mpi.h>
#include <malloc.h>
// Global variables
int processor_rank = 0;
int processor_count = 1;
int main(int argc, char **argv )
{
    int myid, numprocs;
    int ans,i,w,n,index, ha,hb;
    int complete = 0;
    //int iter = 13;
    int *a,*b,*r,*worker;
    int dispatched = 0;
    int returned = 0;
    int Rhotrix[2];
    int *result;
    int Work = 1;
    int idle = -1;
    FILE *fp;
    int RhotrixA_FileStatus = 1, RhotrixB_FileStatus = 1;
    double time0, time1;

    MPI_Request *recv_req;
    MPI_Status status;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    if (numprocs >= 2)
    {
        if (myid == 0) // Master
        {
            // Record starting time

            time0 = MPI_Wtime();
            if ((fp = fopen ("C:/data/rdata113_1.txt", "r")) == NULL){

```

```

        RhotrixA_FileStatus = 0;
    }

    if(RhotrixA_FileStatus != 0) {
        fscanf(fp, "%d %d\n", &n, &ha);
        a = (int *) malloc(n*sizeof(int));
        for (i = 0; i < n; i++){
            fscanf(fp, "%d", &a[i]);
        }
        fclose(fp);
    }

    if((fp = fopen ("C:/data/rdata113_2.txt", "r")) == NULL){
        RhotrixB_FileStatus = 0;
    }

    if(RhotrixB_FileStatus != 0) {
        fscanf(fp, "%d %d\n", &n, &hb);

        b = (int *) malloc(n*sizeof(int));
        for(i = 0; i < n; i++){
            fscanf(fp, "%d", &b[i]);
        }
        fclose(fp);
    }

    r = (int *) malloc(n*sizeof(int));
    worker = (int *) malloc(numprocs*sizeof(int));
    result = (int *) malloc(numprocs*sizeof(int));
    recv_req = (MPI_Request *) malloc(numprocs*sizeof(MPI_Request));

    for (i=0; i<n; i++) { a[i] = a[i]; b[i] = b[i]; r[i]=0; }
    // initialize worker status
    for (w=1; w<numprocs; w++) { worker[w] = idle;}
    while(complete < n)
    {
        for (w=1; w<numprocs; w++)
        {
            if ((worker[w] == idle) && (complete < n))
            {
                printf ("Master sending Rhotrix_A[%d]=%d Rhotrix_B[%d]=%d to Worker
%d\n",complete,a[complete],complete,b[complete],w);
                Rhotrix[0] = a[complete];
                Rhotrix[1] = b[complete];
                // Send the rhotrix elements
                MPI_Send(Rhotrix,2,MPI_INT,w,0,MPI_COMM_WORLD);
                // Post a non-blocking Recv for that Rhotrix Elements
                MPI_Irecv(&result[w],1,MPI_INT,w,0,MPI_COMM_WORLD,&recv_req[w]);
                worker[w] = complete;
            }
        }
    }

```

```

    dispatched++;
    complete++; // next rhotrix elements to send out
}
} //foreach idle worker
// Collect returned results
returned = 0;
for(w=1; w<=dispatched; w++)
{
    MPI_Waitany(dispatched, &recv_req[1], &index, &status);
    printf("Master receiving a result back from Worker %d
R[%d]=%d\n",status.MPI_SOURCE,worker[status.MPI_SOURCE],result[status.MPI_SOU
RCE]);
    r[worker[status.MPI_SOURCE]] = result[status.MPI_SOURCE];
    worker[status.MPI_SOURCE] = idle;
    returned++;
}
dispatched -= returned;
} //while work to be done
//signal each worker to that there is no more work to be done
Rhotrix[0] = -1;
Rhotrix[1] = -1;
for (w=1; w<numprocs; w++)
{
    printf("Master telling worker %d to shut down\n",w);
    MPI_Send(Rhotrix,2,MPI_INT,w,0,MPI_COMM_WORLD);
}
for (i=0; i < n; i++)
{
    if(i==(n+1)/2)
{
printf("hb x ha => %d x %d = %d\n",hb,ha,hb*ha);
}
    printf("a[%d] x hb + b[%d] x ha => %d x %d + %d x %d =
%d\n",i,i,a[i],hb,b[i],ha,r[i]);
}
} // Master
else // Workers
{
    printf("Worker %d Ready\n",myid);
    while (Work == 1)
    {
        MPI_Recv(Rhotrix,2,MPI_INT,0,0,MPI_COMM_WORLD,&status);

        printf("Worker %d received message: Rhotrix_A = %d and Rhotrix_B =
%d\n",myid,Rhotrix[0],Rhotrix[1]);
        if ((Rhotrix[0] == -1) && (Rhotrix[1] == -1))
        {
            Work = -1;
            printf("Worker %d Shutting Down\n",myid);

```

```

    }
    else
    {
        ans = Rhotrix[0]*hb + Rhotrix[1]*ha;
        printf("Worker %d Returning an answer for Rhotrix_A x hb = %d x %d + Rhotrix_B x
ha = %d x %d = %d\n",myid,Rhotrix[0],hb,Rhotrix[1],ha,ans);
        MPI_Send(&ans,1,MPI_INT,0,0,MPI_COMM_WORLD);
    }
}
} // Workers
// Record finish time

    time1 = MPI_Wtime();

// Print time statistics

    printf("Total time using processors [%d] : [%f] seconds\n", myid, time1 - time0);

}
else
printf("ERROR:Must have at least 2 processes to run\n");

MPI_Finalize();
return 0;
}

```

A.2.6 MPI_coupled.c

```

/*-----
PARALLEL RHOTRIX PROBLEM IN C LANGUAGE

PROGRAMMED BY EZUGWU, El-Shamir Absalom
    Department of Mathematics
    Faculty of Sciences
    Ahmadu Bello University
    Samarua, Zaria
    Kaduna State
SUPERVISED BY PROF. S. Juniadu
AHMADU BELLO UNIVERSITY
PROGRAMMED ON FEB, 2010

ROW-COLUMN RHOTRIX MULTIPLICATION
-----*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include "mpi.h"

```

```

/* Timer for the *C*omputation */
double cstartwtime,cstopwtime;
/* Timer for how long the whole procedure *A* actually took */
double astartwtime,astopwtime;

void printrhotrix(FILE *outfp, int m, int n, float *rhot) {

    int i,j;

    for(i=0; i<m; i++) {
        for(j=0; j<n; j++) fprintf(outfp, "%6g ", *(rhot+(i*n)+j));
        fprintf(outfp, "\n");
    }
}

void doparamult(int Am, int An, int Bm, int Bn,
                float *rhotA, float *rhotB, float *rhotC) {
    float *currA, *currB, *temprhotA, *temprhotB, *sumrhot;
    int i, j, trank, nprocs, num;
    int *distlist=NULL;
    MPI_Status stats;

    MPI_Comm_size (MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &trank);

    // Allocate space on all processes
    temprhotA = (float *) malloc(Am * An * sizeof(float));
    temprhotB = (float *) malloc(Bm * Bn * sizeof(float));
    sumrhot = (float *) malloc(An * Bn * sizeof(float));

    for(i=0; i < (An*Bn); i++) rhotC[i] = 0;
    for(i=0; i < (An*Bn); i++) sumrhot[i] = 0;
    for(i=0; i < (Am*An); i++) temprhotA[i] = 0;
    for(i=0; i < (Bm*Bn); i++) temprhotB[i] = 0;

    if (trank == 0) {

        distlist = (int *) malloc((nprocs-1)*sizeof(int));
        for(i=0; i<(nprocs-1); i++) distlist[i] = 0;
        for(i=0; i<Am; i++) (*(distlist + (i%(nprocs-1))))++;

        currA=rhotA;
        currB=rhotB;

        for(i=1; i<nprocs; i++) {
            MPI_Send(distlist+(i-1), 1, MPI_INT, i, 0, MPI_COMM_WORLD);
            MPI_Send(currA, An*distlist[i-1], MPI_FLOAT, i, 0, MPI_COMM_WORLD);

```

```

    MPI_Send(currB, Bn*distlist[i-1], MPI_FLOAT, i, 0, MPI_COMM_WORLD);
    currA+=An*distlist[i-1];
    currB+=Bn*distlist[i-1];
}

free(distlist);
} else {
    MPI_Recv(&num, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &stats);
    MPI_Recv(tempshotA, Am*An, MPI_FLOAT, 0, 0, MPI_COMM_WORLD, &stats);
    MPI_Recv(tempshotB, Bm*Bn, MPI_FLOAT, 0, 0, MPI_COMM_WORLD, &stats);
}

if (trank != 0) {
    currA = tempshotA;
    currB = tempshotB;
    while(num) {

        for(i=0; i<An; i++)
            for(j=0; j<Bn; j++)
                *(sumrhot+(i*Bn)+j) += currA[i] * currB[j];

        currA += An;
        currB += Bn;
        num--;
    }
}

MPI_Reduce(sumrhot, rhotC, An*Bn, MPI_FLOAT, MPI_SUM, 0, MPI_COMM_WORLD);

free(tempshotA);
free(tempshotB);
free(sumrhot);

}

int main(int argc, char **argv){
    int Am=0, An=0, Bm=0, Bn=0;
    FILE *fpA, *fpB, *fpC;
    float *transrhotA=NULL, *rhotB=NULL, *rhotC=NULL;
    int res,trank;
    char *f1=NULL, *f2=NULL, *f3=NULL;
    int i, j,nprocs;
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD, &trank);
    MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
    MPI_Comm_rank(MPI_COMM_WORLD, &trank);

    astartwtime = MPI_Wtime();
    if (trank == 0) {

```

```

fpA = fopen("C:/data/mdata1.inp", "r");
if (fpA == NULL) {
    fprintf(stdout, "UNABLE TO OPEN FILE 1\n");
}
fpB = fopen("C:/data/mdata2.inp", "r");
if (fpB == NULL) {
    fprintf(stdout, "UNABLE TO OPEN FILE 2\n");
}

fscanf(fpA, "%d ", &Am);
fscanf(fpA, "%d ", &An);
fscanf(fpB, "%d ", &Bm);
fscanf(fpB, "%d ", &Bn);

if (Am<=0 || An<=0 || Bm<=0 || Bn<=0) {
    fprintf(stdout, "UNABLE TO PARSE DIMENSIONS!\n");
}

if (An != Bm) {
    fprintf(stdout, "MATRICES ARE INCOMPATIBLE!\n");
    return 0;
}

transrhotA = (float *) malloc(sizeof(float) * Am * An);
rhotB = (float *) malloc(sizeof(float) * Bm * Bn);

if(transrhotA==NULL || rhotB==NULL) {
    fprintf(stdout, "INSUFFICIENT MEMORY!\n");
    return 0;
}

for (i=0; i<Am; i++)
    for (j=0; j<An; j++)
        fscanf(fpA, "%f ", (transrhotA+(Am*j)+i) );

for (i=0; i<(Bn * Bm); i++) fscanf(fpB, "%f ", rhotB+i);

fclose(fpA);
fclose(fpB);

}

MPI_Bcast(&An, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(&Am, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(&Bn, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(&Bm, 1, MPI_INT, 0, MPI_COMM_WORLD);

/* Allocate space on all processes for the rhotrix result.
   This makes it easy to code cause we can use Allreduce */

```

```

rhotC = (float *) malloc(Am * Bn * sizeof(float));

/* Start the timer */
cstartwtime = MPI_Wtime();

/* Notice the Am and An have been swapped because of the transposition */
doparamult(An, Am, Bm, Bn, transrhotA, rhotB, rhotC);

/* Stop the timer */
cstopwtime = MPI_Wtime();

if (trank == 0) {
    fpC = fopen("C:/data/absalom.out", "w");
    if (fpC == NULL) {
        fprintf(stdout, "UNABLE TO OPEN FILE FOR WRITING!\n");
        return 0;
    }
    fprintf(fpC, "%d %d\n", Am, Bn);
    printrhotrix(fpC, Am, Bn, rhotC);
    fclose(fpC);
} else {
    printf("A * B =\n");
    printrhotrix(stdout, Am, Bn, rhotC);
}
astopwtime = MPI_Wtime();
if (trank == 0) {
    if (res == 0) {
        fprintf(stdout, "\nRhotrix Multiplication Failed...\n");
    } else {
        fprintf(stdout, "\nRhotrix multiplication done.\n");
        fprintf(stdout, "%f seconds of computation, %f total seconds\n",
                cstopwtime-cstartwtime, astopwtime-astartwtime);
    }
}
MPI_Finalize();
return 0;
}

```

A.2.7 SystolicRhotrixMultiplication.pas

```

unit SystolicRhotrixMultiplication;
interface
uses
    Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
    Dialogs, ExtCtrls, jpeg, StdCtrls;
type
    TCell = record
        A : integer;
        B : integer;

```

```
R : integer;  
AIn : integer;  
BIn : integer;  
end;
```

```
TfrmMain = class(TForm)
```

```
  pnTop: TPanel;  
  pnRight: TPanel;  
  pnDown: TPanel;  
  pnLeft: TPanel;  
  pnCenter: TPanel;  
  pnImage: TPanel;  
  img11: TImage;  
  img12: TImage;  
  img13: TImage;  
  img41: TImage;  
  img21: TImage;  
  img22: TImage;  
  img23: TImage;  
  img42: TImage;  
  img31: TImage;  
  img32: TImage;  
  img33: TImage;  
  img43: TImage;  
  edB1: TEdit;  
  edB2: TEdit;  
  edB3: TEdit;  
  edB4: TEdit;  
  edB5: TEdit;  
  edA1: TEdit;  
  edA2: TEdit;  
  edA3: TEdit;  
  edA4: TEdit;  
  edA5: TEdit;  
  C11A: TEdit;  
  C11B: TEdit;  
  C11R: TEdit;  
  C12A: TEdit;  
  C12B: TEdit;  
  C12R: TEdit;  
  C13B: TEdit;  
  C13R: TEdit;  
  C13A: TEdit;  
  C14B: TEdit;  
  C14R: TEdit;  
  C14A: TEdit;  
  C15B: TEdit;  
  C15R: TEdit;  
  C15A: TEdit;
```

C43B: TEdit;
C43R: TEdit;
C43A: TEdit;
C44B: TEdit;
C44R: TEdit;
C44A: TEdit;
C45B: TEdit;
C45R: TEdit;
C45A: TEdit;
C41B: TEdit;
C41R: TEdit;
C41A: TEdit;
C32B: TEdit;
C32R: TEdit;
C32A: TEdit;
C22B: TEdit;
C22R: TEdit;
C22A: TEdit;
C21B: TEdit;
C21R: TEdit;
C21A: TEdit;
C31B: TEdit;
C31R: TEdit;
C31A: TEdit;
C33B: TEdit;
C33A: TEdit;
C23B: TEdit;
C23R: TEdit;
C23A: TEdit;
C24B: TEdit;
C24R: TEdit;
C24A: TEdit;
C25B: TEdit;
C25R: TEdit;
C25A: TEdit;
C42B: TEdit;
C42R: TEdit;
C42A: TEdit;
C52B: TEdit;
C52R: TEdit;
C52A: TEdit;
C53B: TEdit;
C53R: TEdit;
C53A: TEdit;
btStep: TButton;
btReset: TButton;
gbMatrix1: TGroupBox;
A55: TEdit;
A54: TEdit;

A52: TEdit;
A51: TEdit;
A53: TEdit;
A45: TEdit;
A44: TEdit;
A43: TEdit;
A42: TEdit;
A41: TEdit;
A35: TEdit;
A34: TEdit;
A33: TEdit;
A32: TEdit;
A31: TEdit;
A25: TEdit;
A24: TEdit;
A23: TEdit;
A22: TEdit;
A21: TEdit;
A15: TEdit;
A14: TEdit;
A13: TEdit;
A12: TEdit;
A11: TEdit;
gbMatrixB: TGroupBox;
B11: TEdit;
B12: TEdit;
B13: TEdit;
B14: TEdit;
B15: TEdit;
B23: TEdit;
B22: TEdit;
B21: TEdit;
B24: TEdit;
B25: TEdit;
B31: TEdit;
B32: TEdit;
B33: TEdit;
B34: TEdit;
B35: TEdit;
B41: TEdit;
B42: TEdit;
B43: TEdit;
B44: TEdit;
B45: TEdit;
B51: TEdit;
B52: TEdit;
B53: TEdit;
B54: TEdit;
B55: TEdit;

Label1: TLabel;
Bevel1: TBevel;
Label2: TLabel;
Label3: TLabel;
edStep: TLabel;
edOperation: TLabel;
Bevel2: TBevel;
Image1: TImage;
Image2: TImage;
Image3: TImage;
C34A: TEdit;
C34B: TEdit;
C34R: TEdit;
C33R: TEdit;
Image4: TImage;
Image5: TImage;
C51B: TEdit;
C51A: TEdit;
C51R: TEdit;
Image6: TImage;
Image7: TImage;

C54A: TEdit;
C54B: TEdit;
C54R: TEdit;

C35A: TEdit;
C35B: TEdit;
C35R: TEdit;
C55A: TEdit;
C55B: TEdit;
C55R: TEdit;
Image8: TImage;
Image9: TImage;
Image10: TImage;
Image11: TImage;
Image12: TImage;
Image13: TImage;
Button1: TButton;
{ B51: TEdit; }
{B52: TEdit; }
{B53: TEdit;}
{B54: TEdit;}
{B55: TEdit; }

{A53: TEdit; }

procedure btStepClick(Sender: TObject);
procedure btResetClick(Sender: TObject);

```

procedure FormCreate(Sender: TObject);
procedure Button1Click(Sender: TObject);
private
  BuffA : array [1..5] of integer;
  BuffB : array [1..5] of integer;
  ISA : array [1..5, 1..5] of TCell;
  Tact : integer;
  Step : integer;

  procedure BindBuffA;
  procedure BindBuffB;
  procedure InitializeBuffers;
  procedure InitializeISA;
  procedure BindISA;
  procedure PopulateBuffA(ATact : integer);
  procedure PopulateBuffB(ATact : integer);
  procedure Advance;
  procedure InitializeAll;
public
end;

var
  frmMain: TfrmMain;

implementation

{$R *.dfm}

procedure TfrmMain.InitializeBuffers;
var
  i : integer;
begin
  for i := 1 to 5 do
    BuffA[i] := -1;
  for i := 1 to 5 do
    BuffB[i] := -1;
end;

procedure TfrmMain.BindBuffA;
begin
  edA1.Text := 'A1: ';
  if BuffA[1] <> -1 then
    edA1.Text := edA1.Text + IntToStr(BuffA[1]);
  edA2.Text := 'A2: ';
  if BuffA[2] <> -1 then
    edA2.Text := edA2.Text + IntToStr(BuffA[2]);
  edA3.Text := 'A3: ';
  if BuffA[3] <> -1 then
    edA3.Text := edA3.Text + IntToStr(BuffA[3]);

```

```

edA4.Text := 'A4: ';
if BuffA[4] <> -1 then
  edA4.Text := edA4.Text + IntToStr(BuffA[4]);
edA5.Text := 'A5: ';
if BuffA[5] <> -1 then
  edA5.Text := edA5.Text + IntToStr(BuffA[5]);
end;

```

```

procedure TfrmMain.BindBuffB;
begin
  edB1.Text := 'B1: ';
  if BuffB[1] <> -1 then
    edB1.Text := edB1.Text + IntToStr(BuffB[1]);
  edB2.Text := 'B2: ';
  if BuffB[2] <> -1 then
    edB2.Text := edB2.Text + IntToStr(BuffB[2]);
  edB3.Text := 'B3: ';
  if BuffB[3] <> -1 then
    edB3.Text := edB3.Text + IntToStr(BuffB[3]);
  edB4.Text := 'B4: ';
  if BuffB[4] <> -1 then
    edB4.Text := edB4.Text + IntToStr(BuffB[4]);
  edB5.Text := 'B5: ';
  if BuffB[5] <> -1 then
    edB5.Text := edB5.Text + IntToStr(BuffB[5]);
end;

```

```

procedure TfrmMain.InitializeISA;
var
  i: integer;
  j: integer;
begin
  for i := 1 to 5 do
    for j := 1 to 5 do
      begin
        ISA[i,j].A := -1;
        ISA[i,j].B := -1;
        ISA[i,j].R := 0;
        ISA[i,j].AIn := -1;
        ISA[i,j].BIn := -1;
      end;
    end;
end;

```

```

procedure TfrmMain.BindISA;
var
  i: integer;
  j: integer;
begin
  C11A.Text := 'A: ';

```

```

if (ISA[1,1].A <> -1) then
  C11A.Text := C11A.Text + IntToStr(ISA[1,1].A);
C12A.Text := 'A: ';
if (ISA[1,2].A <> -1) then
  C12A.Text := C12A.Text + IntToStr(ISA[1,2].A);
C13A.Text := 'A: ';
if (ISA[1,3].A <> -1) then
  C13A.Text := C13A.Text + IntToStr(ISA[1,3].A);
C14A.Text := 'A: ';
if (ISA[1,4].A <> -1) then
  C14A.Text := C14A.Text + IntToStr(ISA[1,4].A);
C15A.Text := 'A: ';
if (ISA[1,5].A <> -1) then
  C15A.Text := C15A.Text + IntToStr(ISA[1,5].A);
C21A.Text := 'A: ';
if (ISA[2,1].A <> -1) then
  C21A.Text := C21A.Text + IntToStr(ISA[2,1].A);
C22A.Text := 'A: ';
if (ISA[2,2].A <> -1) then
  C22A.Text := C22A.Text + IntToStr(ISA[2,2].A);
C23A.Text := 'A: ';
if (ISA[2,3].A <> -1) then
  C23A.Text := C23A.Text + IntToStr(ISA[2,3].A);
C24A.Text := 'A: ';
if (ISA[2,4].A <> -1) then
  C24A.Text := C24A.Text + IntToStr(ISA[2,4].A);
C25A.Text := 'A: ';
if (ISA[2,5].A <> -1) then
  C25A.Text := C25A.Text + IntToStr(ISA[2,5].A);
C31A.Text := 'A: ';
if (ISA[3,1].A <> -1) then
  C31A.Text := C31A.Text + IntToStr(ISA[3,1].A);
C32A.Text := 'A: ';
if (ISA[3,2].A <> -1) then
  C32A.Text := C32A.Text + IntToStr(ISA[3,2].A);
C33A.Text := 'A: ';
if (ISA[3,3].A <> -1) then
  C33A.Text := C33A.Text + IntToStr(ISA[3,3].A);
C34A.Text := 'A: ';
if (ISA[3,4].A <> -1) then
  C34A.Text := C34A.Text + IntToStr(ISA[3,4].A);
C35A.Text := 'A: ';
if (ISA[3,5].A <> -1) then
  C35A.Text := C35A.Text + IntToStr(ISA[3,5].A);
C41A.Text := 'A: ';
if (ISA[4,1].A <> -1) then
  C41A.Text := C41A.Text + IntToStr(ISA[4,1].A);
C42A.Text := 'A: ';
if (ISA[4,2].A <> -1) then

```

```

C42A.Text := C42A.Text + IntToStr(ISA[4,2].A);
C43A.Text := 'A: ';
if (ISA[4,3].A <> -1) then
  C43A.Text := C43A.Text + IntToStr(ISA[4,3].A);
C44A.Text := 'A: ';
if (ISA[4,4].A <> -1) then
  C44A.Text := C44A.Text + IntToStr(ISA[4,4].A);
C45A.Text := 'A: ';
if (ISA[4,5].A <> -1) then
  C45A.Text := C45A.Text + IntToStr(ISA[4,5].A);
C51A.Text := 'A: ';
if (ISA[5,1].A <> -1) then
  C51A.Text := C51A.Text + IntToStr(ISA[5,1].A);
C52A.Text := 'A: ';
if (ISA[5,2].A <> -1) then
  C52A.Text := C52A.Text + IntToStr(ISA[5,2].A);
C53A.Text := 'A: ';
if (ISA[5,3].A <> -1) then
  C53A.Text := C53A.Text + IntToStr(ISA[5,3].A);
C54A.Text := 'A: ';
if (ISA[5,4].A <> -1) then
  C54A.Text := C54A.Text + IntToStr(ISA[5,4].A);
C55A.Text := 'A: ';
if (ISA[5,5].A <> -1) then
  C55A.Text := C55A.Text + IntToStr(ISA[5,5].A);
C11B.Text := 'B: ';
if (ISA[1,1].B <> -1) then
  C11B.Text := C11B.Text + IntToStr(ISA[1,1].B);
C12B.Text := 'B: ';
if (ISA[1,2].B <> -1) then
  C12B.Text := C12B.Text + IntToStr(ISA[1,2].B);
C13B.Text := 'B: ';
if (ISA[1,3].B <> -1) then
  C13B.Text := C13B.Text + IntToStr(ISA[1,3].B);
C14B.Text := 'B: ';
if (ISA[1,4].B <> -1) then
  C14B.Text := C14B.Text + IntToStr(ISA[1,4].B);
C15B.Text := 'B: ';
if (ISA[1,5].B <> -1) then
  C15B.Text := C15B.Text + IntToStr(ISA[1,5].B);
C21B.Text := 'B: ';
if (ISA[2,1].B <> -1) then
  C21B.Text := C21B.Text + IntToStr(ISA[2,1].B);
C22B.Text := 'B: ';
if (ISA[2,2].B <> -1) then
  C22B.Text := C22B.Text + IntToStr(ISA[2,2].B);
C23B.Text := 'B: ';
if (ISA[2,3].B <> -1) then
  C23B.Text := C23B.Text + IntToStr(ISA[2,3].B);

```

```

C24B.Text := 'B: ';
if (ISA[2,4].B <> -1) then
  C24B.Text := C24B.Text + IntToStr(ISA[2,4].B);
C25B.Text := 'B: ';
if (ISA[2,5].B <> -1) then
  C25B.Text := C25B.Text + IntToStr(ISA[2,5].B);
C31B.Text := 'B: ';
if (ISA[3,1].B <> -1) then
  C31B.Text := C31B.Text + IntToStr(ISA[3,1].B);
C32B.Text := 'B: ';
if (ISA[3,2].B <> -1) then
  C32B.Text := C32B.Text + IntToStr(ISA[3,2].B);
C33B.Text := 'B: ';
if (ISA[3,3].B <> -1) then
  C33B.Text := C33B.Text + IntToStr(ISA[3,3].B);
C34B.Text := 'B: ';
if (ISA[3,4].B <> -1) then
  C34B.Text := C34B.Text + IntToStr(ISA[3,4].B);
C35B.Text := 'B: ';
if (ISA[3,5].B <> -1) then
  C35B.Text := C35B.Text + IntToStr(ISA[3,5].B);
C41B.Text := 'B: ';
if (ISA[4,1].B <> -1) then
  C41B.Text := C41B.Text + IntToStr(ISA[4,1].B);
C42B.Text := 'B: ';
if (ISA[4,2].B <> -1) then
  C42B.Text := C42B.Text + IntToStr(ISA[4,2].B);
C43B.Text := 'B: ';
if (ISA[4,3].B <> -1) then
  C43B.Text := C43B.Text + IntToStr(ISA[4,3].B);
C44B.Text := 'B: ';
if (ISA[4,4].B <> -1) then
  C44B.Text := C44B.Text + IntToStr(ISA[4,4].B);
C45B.Text := 'B: ';
if (ISA[4,5].B <> -1) then
  C45B.Text := C45B.Text + IntToStr(ISA[4,5].B);
C51B.Text := 'B: ';
if (ISA[5,1].B <> -1) then
  C51B.Text := C51B.Text + IntToStr(ISA[5,1].B);
C52B.Text := 'B: ';
if (ISA[5,2].B <> -1) then
  C52B.Text := C52B.Text + IntToStr(ISA[5,2].B);
C53B.Text := 'B: ';
if (ISA[5,3].B <> -1) then
  C53B.Text := C53B.Text + IntToStr(ISA[5,3].B);
C54B.Text := 'B: ';
if (ISA[5,4].B <> -1) then
  C54B.Text := C54B.Text + IntToStr(ISA[5,4].B);
C55B.Text := 'B: ';

```

```

if (ISA[5,5].B <> -1) then
  C55B.Text := C55B.Text + IntToStr(ISA[5,5].B);

C11R.Text := 'R: ' + IntToStr(ISA[1,1].R);
C12R.Text := 'R: ' + IntToStr(ISA[1,2].R);
C13R.Text := 'R: ' + IntToStr(ISA[1,3].R);
C14R.Text := 'R: ' + IntToStr(ISA[1,4].R);
C15R.Text := 'R: ' + IntToStr(ISA[1,5].R);
C21R.Text := 'R: ' + IntToStr(ISA[2,1].R);
C22R.Text := 'R: ' + IntToStr(ISA[2,2].R);
C23R.Text := 'R: ' + IntToStr(ISA[2,3].R);
C24R.Text := 'R: ' + IntToStr(ISA[2,4].R);
C25R.Text := 'R: ' + IntToStr(ISA[2,5].R);
C31R.Text := 'R: ' + IntToStr(ISA[3,1].R);
C32R.Text := 'R: ' + IntToStr(ISA[3,2].R);
C33R.Text := 'R: ' + IntToStr(ISA[3,3].R);
C34R.Text := 'R: ' + IntToStr(ISA[3,4].R);
C35R.Text := 'R: ' + IntToStr(ISA[3,5].R);
C41R.Text := 'R: ' + IntToStr(ISA[4,1].R);
C42R.Text := 'R: ' + IntToStr(ISA[4,2].R);
C43R.Text := 'R: ' + IntToStr(ISA[4,3].R);
C44R.Text := 'R: ' + IntToStr(ISA[4,4].R);
C45R.Text := 'R: ' + IntToStr(ISA[4,5].R);
C51R.Text := 'R: ' + IntToStr(ISA[5,1].R);
C52R.Text := 'R: ' + IntToStr(ISA[5,2].R);
C53R.Text := 'R: ' + IntToStr(ISA[5,3].R);
C54R.Text := 'R: ' + IntToStr(ISA[5,4].R);
C55R.Text := 'R: ' + IntToStr(ISA[5,5].R);
end;

```

```

procedure TfrmMain.PopulateBuffA(ATact: integer);
begin
  case ATact of
  1:
    begin
      BuffA[1] := StrToInt(A11.Text);
      BuffA[2] := -1;
      BuffA[3] := -1;
      BuffA[4] := -1;
      BuffA[5] := -1;
      A11.Color := clRed;
      A12.Color := clHighlight;
      A13.Color := clHighlight;
      A14.Color := clHighlight;
      A15.Color := clHighlight;
      A21.Color := clHighlight;
      A22.Color := clHighlight;
      A23.Color := clHighlight;
      A24.Color := clHighlight;
    end;
  end;

```

```

A25.Color := clHighlight;
A31.Color := clHighlight;
A32.Color := clHighlight;
A33.Color := clHighlight;
A34.Color := clHighlight;
A35.Color := clHighlight;
A41.Color := clHighlight;
A42.Color := clHighlight;
A43.Color := clHighlight;
A44.Color := clHighlight;
A45.Color := clHighlight;
A51.Color := clHighlight;
A52.Color := clHighlight;
A53.Color := clHighlight;
A54.Color := clHighlight;
A55.Color := clHighlight;
end;
2:
begin
  BuffA[1] := StrToInt(A12.Text);
  BuffA[2] := StrToInt(A21.Text);
  BuffA[3] := -1;
  BuffA[4] := -1;
  BuffA[5] := -1;
  A11.Color := clHighlight;
  A12.Color := clRed;
  A13.Color := clHighlight;
  A14.Color := clHighlight;
  A15.Color := clHighlight;
  A21.Color := clRed;
  A22.Color := clHighlight;
  A23.Color := clHighlight;
  A24.Color := clHighlight;
  A25.Color := clHighlight;
  A31.Color := clHighlight;
  A32.Color := clHighlight;
  A33.Color := clHighlight;
  A34.Color := clHighlight;
  A35.Color := clHighlight;
  A41.Color := clHighlight;
  A42.Color := clHighlight;
  A43.Color := clHighlight;
  A44.Color := clHighlight;
  A45.Color := clHighlight;
  A51.Color := clHighlight;
  A52.Color := clHighlight;
  A53.Color := clHighlight;
  A54.Color := clHighlight;
  A55.Color := clHighlight;

```

```

end;
3:
begin
  BuffA[1] := StrToInt(A13.Text);
  BuffA[2] := StrToInt(A22.Text);
  BuffA[3] := StrToInt(A31.Text);
  BuffA[4] := -1;
  BuffA[5] := -1;
  A11.Color := clHighlight;
  A12.Color := clHighlight;
  A13.Color := clRed;
  A14.Color := clHighlight;
  A15.Color := clHighlight;
  A21.Color := clHighlight;
  A22.Color := clRed;
  A23.Color := clHighlight;
  A24.Color := clHighlight;
  A25.Color := clHighlight;
  A31.Color := clRed;
  A32.Color := clHighlight;
  A33.Color := clHighlight;
  A34.Color := clHighlight;
  A35.Color := clHighlight;
  A41.Color := clHighlight;
  A42.Color := clHighlight;
  A43.Color := clHighlight;
  A44.Color := clHighlight;
  A45.Color := clHighlight;
  A51.Color := clHighlight;
  A52.Color := clHighlight;
  A53.Color := clHighlight;
  A54.Color := clHighlight;
  A55.Color := clHighlight;
end;

```

```

4:
begin
  BuffA[1] := StrToInt(A14.Text);
  BuffA[2] := StrToInt(A23.Text);
  BuffA[3] := StrToInt(A32.Text);
  BuffA[4] := StrToInt(A41.Text);
  BuffA[5] := -1;
  A11.Color := clHighlight;
  A12.Color := clHighlight;
  A13.Color := clHighlight;
  A14.Color := clRed;
  A15.Color := clHighlight;
  A21.Color := clHighlight;
  A22.Color := clHighlight;
  A23.Color := clRed;

```

```

A24.Color := clHighlight;
A25.Color := clHighlight;
A31.Color := clHighlight;
A32.Color := clRed;
A33.Color := clHighlight;
A34.Color := clHighlight;
A35.Color := clHighlight;
A41.Color := clRed;
A42.Color := clHighlight;
A43.Color := clHighlight;
A44.Color := clHighlight;
A45.Color := clHighlight;
A51.Color := clHighlight;
A52.Color := clHighlight;
A53.Color := clHighlight;
A54.Color := clHighlight;
A55.Color := clHighlight;
end;
5:
begin
  BuffA[1] := StrToInt(A15.Text);
  BuffA[2] := StrToInt(A24.Text);
  BuffA[3] := StrToInt(A33.Text);
  BuffA[4] := StrToInt(A42.Text);
  BuffA[5] := StrToInt(A51.Text);
  A11.Color := clHighlight;
  A12.Color := clHighlight;
  A13.Color := clHighlight;
  A14.Color := clHighlight;
  A15.Color := clRed;
  A21.Color := clHighlight;
  A22.Color := clHighlight;
  A23.Color := clHighlight;
  A24.Color := clRed;
  A25.Color := clHighlight;
  A31.Color := clHighlight;
  A32.Color := clHighlight;
  A33.Color := clRed;
  A34.Color := clHighlight;
  A35.Color := clHighlight;
  A41.Color := clHighlight;
  A42.Color := clRed;
  A43.Color := clHighlight;
  A44.Color := clHighlight;
  A45.Color := clHighlight;
  A51.Color := clRed;
  A52.Color := clHighlight;
  A53.Color := clHighlight;
  A54.Color := clHighlight;

```

```

    A55.Color := clHighlight;
end;
6:
begin
    BuffA[1] := -1;
    BuffA[2] := StrToInt(A25.Text);
    BuffA[3] := StrToInt(A34.Text);
    BuffA[4] := StrToInt(A43.Text);
    BuffA[5] := StrToInt(A52.Text);
    A11.Color := clHighlight;
    A12.Color := clHighlight;
    A13.Color := clHighlight;
    A14.Color := clHighlight;
    A15.Color := clHighlight;
    A21.Color := clHighlight;
    A22.Color := clHighlight;
    A23.Color := clHighlight;
    A24.Color := clHighlight;
    A25.Color := clRed;
    A31.Color := clHighlight;
    A32.Color := clHighlight;
    A33.Color := clHighlight;
    A34.Color := clRed;
    A35.Color := clHighlight;
    A41.Color := clHighlight;
    A42.Color := clHighlight;
    A43.Color := clRed;
    A44.Color := clHighlight;
    A45.Color := clHighlight;
    A51.Color := clHighlight;
    A52.Color := clRed;
    A53.Color := clHighlight;
    A54.Color := clHighlight;
    A55.Color := clHighlight;
end;
7:
begin
    BuffA[1] := -1;
    BuffA[2] := -1;
    BuffA[3] := StrToInt(A35.Text);
    BuffA[4] := StrToInt(A44.Text);
    BuffA[5] := StrToInt(A53.Text);
    A11.Color := clHighlight;
    A12.Color := clHighlight;
    A13.Color := clHighlight;
    A14.Color := clHighlight;
    A15.Color := clHighlight;
    A21.Color := clHighlight;
    A22.Color := clHighlight;

```

```

A23.Color := clHighlight;
A24.Color := clHighlight;
A25.Color := clHighlight;
A31.Color := clHighlight;
A32.Color := clHighlight;
A33.Color := clHighlight;
A34.Color := clHighlight;
A35.Color := clRed;
A41.Color := clHighlight;
A42.Color := clHighlight;
A43.Color := clHighlight;
A44.Color := clRed;
A45.Color := clHighlight;
A51.Color := clHighlight;
A52.Color := clHighlight;
A53.Color := clRed;
A54.Color := clHighlight;
A55.Color := clHighlight;
end;
8:
begin
  BuffA[1] := -1;
  BuffA[2] := -1;
  BuffA[3] := -1;
  BuffA[4] := StrToInt(A45.Text);
  BuffA[5] := StrToInt(A54.Text);
  A11.Color := clHighlight;
  A12.Color := clHighlight;
  A13.Color := clHighlight;
  A14.Color := clHighlight;
  A15.Color := clHighlight;
  A21.Color := clHighlight;
  A22.Color := clHighlight;
  A23.Color := clHighlight;
  A24.Color := clHighlight;
  A25.Color := clHighlight;
  A31.Color := clHighlight;
  A32.Color := clHighlight;
  A33.Color := clHighlight;
  A34.Color := clHighlight;
  A35.Color := clHighlight;
  A41.Color := clHighlight;
  A42.Color := clHighlight;
  A43.Color := clHighlight;
  A44.Color := clHighlight;
  A45.Color := clRed;
  A51.Color := clHighlight;
  A52.Color := clHighlight;
  A53.Color := clHighlight;

```

```

A54.Color := clRed;
A55.Color := clHighlight;
end;
9:
begin
  BuffA[1] := -1;
  BuffA[2] := -1;
  BuffA[3] := -1;
  BuffA[4] := -1;
  BuffA[5] := StrToInt(A55.Text);
  A11.Color := clHighlight;
  A12.Color := clHighlight;
  A13.Color := clHighlight;
  A14.Color := clHighlight;
  A15.Color := clHighlight;
  A21.Color := clHighlight;
  A22.Color := clHighlight;
  A23.Color := clHighlight;
  A24.Color := clHighlight;
  A25.Color := clHighlight;
  A31.Color := clHighlight;
  A32.Color := clHighlight;
  A33.Color := clHighlight;
  A34.Color := clHighlight;
  A35.Color := clHighlight;
  A41.Color := clHighlight;
  A42.Color := clHighlight;
  A43.Color := clHighlight;
  A44.Color := clHighlight;
  A45.Color := clHighlight;
  A51.Color := clHighlight;
  A52.Color := clHighlight;
  A53.Color := clHighlight;
  A54.Color := clHighlight;
  A55.Color := clRed;
end;
else
begin
  BuffA[1] := -1;
  BuffA[2] := -1;
  BuffA[3] := -1;
  BuffA[4] := -1;
  BuffA[5] := -1;
  A11.Color := clHighlight;
  A12.Color := clHighlight;
  A13.Color := clHighlight;
  A14.Color := clHighlight;
  A15.Color := clHighlight;
  A21.Color := clHighlight;

```

```

A22.Color := clHighlight;
A23.Color := clHighlight;
A24.Color := clHighlight;
A25.Color := clHighlight;
A31.Color := clHighlight;
A32.Color := clHighlight;
A33.Color := clHighlight;
A34.Color := clHighlight;
A35.Color := clHighlight;
A41.Color := clHighlight;
A42.Color := clHighlight;
A43.Color := clHighlight;
A44.Color := clHighlight;
A45.Color := clHighlight;
A51.Color := clHighlight;
A52.Color := clHighlight;
A53.Color := clHighlight;
A54.Color := clHighlight;
A55.Color := clHighlight;
end;
end;
end;

procedure TfrmMain.PopulateBuffB(ATact: integer);
begin
  case ATact of
    1:
      begin
        BuffB[1] := StrToInt(B11.Text);
        BuffB[2] := -1;
        BuffB[3] := -1;
        BuffB[4] := -1;
        BuffB[5] := -1;
        B11.Color := clRed;
        B12.Color := clLime;
        B13.Color := clLime;
        B14.Color := clLime;
        B15.Color := clLime;
        B21.Color := clLime;
        B22.Color := clLime;
        B23.Color := clLime;
        B24.Color := clLime;
        B25.Color := clLime;
        B31.Color := clLime;
        B32.Color := clLime;
        B33.Color := clLime;
        B34.Color := clLime;
        B35.Color := clLime;
        B41.Color := clLime;

```

```

    B42.Color := clLime;
    B43.Color := clLime;
    B44.Color := clLime;
    B45.Color := clLime;
    B51.Color := clLime;
    B52.Color := clLime;
    B53.Color := clLime;
    B54.Color := clLime;
    B55.Color := clLime;
end;
2:
begin
    BuffB[1] := StrToInt(B12.Text);
    BuffB[2] := StrToInt(B21.Text);
    BuffB[3] := -1;
    BuffB[4] := -1;
    BuffB[5] := -1;
    B11.Color := clLime;
    B12.Color := clRed;
    B13.Color := clLime;
    B14.Color := clLime;
    B15.Color := clLime;
    B21.Color := clRed;
    B22.Color := clLime;
    B23.Color := clLime;
    B24.Color := clLime;
    B25.Color := clLime;
    B31.Color := clLime;
    B32.Color := clLime;
    B33.Color := clLime;
    B34.Color := clLime;
    B35.Color := clLime;
    B41.Color := clLime;
    B42.Color := clLime;
    B43.Color := clLime;
    B44.Color := clLime;
    B45.Color := clLime;
    B51.Color := clLime;
    B52.Color := clLime;
    B53.Color := clLime;
    B54.Color := clLime;
    B55.Color := clLime;
end;
3:
begin
    BuffB[1] := StrToInt(B13.Text);
    BuffB[2] := StrToInt(B22.Text);
    BuffB[3] := StrToInt(B31.Text);
    BuffB[4] := -1;

```

```

BuffB[5] := -1;
B11.Color := clLime;
B12.Color := clLime;
B13.Color := clRed;
B14.Color := clLime;
B15.Color := clLime;
B21.Color := clLime;
B22.Color := clRed;
B23.Color := clLime;
B24.Color := clLime;
B25.Color := clLime;
B31.Color := clRed;
B32.Color := clLime;
B33.Color := clLime;
B34.Color := clLime;
B35.Color := clLime;
B41.Color := clLime;
B42.Color := clLime;
B43.Color := clLime;
B44.Color := clLime;
B45.Color := clLime;
B51.Color := clLime;
B52.Color := clLime;
B53.Color := clLime;
B54.Color := clLime;
B55.Color := clLime;
end;
4:
begin
  BuffB[1] := StrToInt(B14.Text);
  BuffB[2] := StrToInt(B23.Text);
  BuffB[3] := StrToInt(B32.Text);
  BuffB[4] := StrToInt(B41.Text);
  BuffB[5] := -1;
  B11.Color := clLime;
  B12.Color := clLime;
  B13.Color := clLime;
  B14.Color := clRed;
  B15.Color := clLime;
  B21.Color := clLime;
  B22.Color := clLime;
  B23.Color := clRed;
  B24.Color := clLime;
  B25.Color := clLime;
  B31.Color := clLime;
  B32.Color := clRed;
  B33.Color := clLime;
  B34.Color := clLime;
  B35.Color := clLime;

```

```

    B41.Color := clRed;
    B42.Color := clLime;
    B43.Color := clLime;
    B44.Color := clLime;
    B45.Color := clLime;
    B51.Color := clLime;
    B52.Color := clLime;
    B53.Color := clLime;
    B54.Color := clLime;
    B55.Color := clLime;
end;
5:
begin
    BuffB[1] := StrToInt(B15.Text);
    BuffB[2] := StrToInt(B24.Text);
    BuffB[3] := StrToInt(B33.Text);
    BuffB[4] := StrToInt(B42.Text);
    BuffB[5] := StrToInt(B51.Text);
    B11.Color := clLime;
    B12.Color := clLime;
    B13.Color := clLime;
    B14.Color := clLime;
    B15.Color := clRed;
    B21.Color := clLime;
    B22.Color := clLime;
    B23.Color := clLime;
    B24.Color := clRed;
    B25.Color := clLime;
    B31.Color := clLime;
    B32.Color := clLime;
    B33.Color := clRed;
    B34.Color := clLime;
    B35.Color := clLime;
    B41.Color := clLime;
    B42.Color := clRed;
    B43.Color := clLime;
    B44.Color := clLime;
    B45.Color := clLime;
    B51.Color := clRed;
    B52.Color := clLime;
    B53.Color := clLime;
    B54.Color := clLime;
    B55.Color := clLime;
end;
6:
begin
    BuffB[1] := -1;
    BuffB[2] := StrToInt(B25.Text);
    BuffB[3] := StrToInt(B34.Text);

```

```

BuffB[4] := StrToInt(B43.Text);
BuffB[5] := StrToInt(B52.Text);
B11.Color := clLime;
B12.Color := clLime;
B13.Color := clLime;
B14.Color := clLime;
B15.Color := clLime;
B21.Color := clLime;
B22.Color := clLime;
B23.Color := clLime;
B24.Color := clLime;
B25.Color := clRed;
B31.Color := clLime;
B32.Color := clLime;
B33.Color := clLime;
B34.Color := clRed;
B35.Color := clLime;
B41.Color := clLime;
B42.Color := clLime;
B43.Color := clRed;
B44.Color := clLime;
B45.Color := clLime;
B51.Color := clLime;
B52.Color := clRed;
B53.Color := clLime;
B54.Color := clLime;
B55.Color := clLime;
end;
7:
begin
BuffB[1] := -1;
BuffB[2] := -1;
BuffB[3] := StrToInt(B35.Text);
BuffB[4] := StrToInt(B44.Text);
BuffB[5] := StrToInt(B53.Text);
B11.Color := clLime;
B12.Color := clLime;
B13.Color := clLime;
B14.Color := clLime;
B15.Color := clLime;
B21.Color := clLime;
B22.Color := clLime;
B23.Color := clLime;
B24.Color := clLime;
B25.Color := clLime;
B31.Color := clLime;
B32.Color := clLime;
B33.Color := clLime;
B34.Color := clLime;

```

```

B35.Color := clRed;
B41.Color := clLime;
B42.Color := clLime;
B43.Color := clLime;
B44.Color := clRed;
B45.Color := clLime;
B51.Color := clLime;
B52.Color := clLime;
B53.Color := clRed;
B54.Color := clLime;
B55.Color := clLime;
end;
8:
begin
  BuffB[1] := -1;
  BuffB[2] := -1;
  BuffB[3] := -1;
  BuffB[4] := StrToInt(B45.Text);
  BuffB[5] := StrToInt(B54.Text);
  B11.Color := clLime;
  B12.Color := clLime;
  B13.Color := clLime;
  B14.Color := clLime;
  B15.Color := clLime;
  B21.Color := clLime;
  B22.Color := clLime;
  B23.Color := clLime;
  B24.Color := clLime;
  B25.Color := clLime;
  B31.Color := clLime;
  B32.Color := clLime;
  B33.Color := clLime;
  B34.Color := clLime;
  B35.Color := clLime;
  B41.Color := clLime;
  B42.Color := clLime;
  B43.Color := clLime;
  B44.Color := clLime;
  B45.Color := clRed;
  B51.Color := clLime;
  B52.Color := clLime;
  B53.Color := clLime;
  B54.Color := clRed;
  B55.Color := clLime;
end;
9:
begin
  BuffB[1] := -1;
  BuffB[2] := -1;

```

```

BuffB[3] := -1;
BuffB[4] := -1;
BuffB[5] := StrToInt(B55.Text);
B11.Color := clLime;
B12.Color := clLime;
B13.Color := clLime;
B14.Color := clLime;
B15.Color := clLime;
B21.Color := clLime;
B22.Color := clLime;
B23.Color := clLime;
B24.Color := clLime;
B25.Color := clLime;
B31.Color := clLime;
B32.Color := clLime;
B33.Color := clLime;
B34.Color := clLime;
B35.Color := clLime;
B41.Color := clLime;
B42.Color := clLime;
B43.Color := clLime;
B44.Color := clLime;
B45.Color := clLime;
B51.Color := clLime;
B52.Color := clLime;
B53.Color := clLime;
B54.Color := clLime;
B55.Color := clRed;
end;
else
begin
BuffB[1] := -1;
BuffB[2] := -1;
BuffB[3] := -1;
BuffB[4] := -1;
BuffB[5] := -1;
B11.Color := clLime;
B12.Color := clLime;
B13.Color := clLime;
B14.Color := clLime;
B15.Color := clLime;
B21.Color := clLime;
B22.Color := clLime;
B23.Color := clLime;
B24.Color := clLime;
B25.Color := clLime;
B31.Color := clLime;
B32.Color := clLime;
B33.Color := clLime;

```

```

    B34.Color := clLime;
    B35.Color := clLime;
    B41.Color := clLime;
    B42.Color := clLime;
    B43.Color := clLime;
    B44.Color := clLime;
    B45.Color := clLime;
    B51.Color := clLime;
    B52.Color := clLime;
    B53.Color := clLime;
    B54.Color := clLime;
    B55.Color := clLime;
    end;
end;
end;

procedure TfrmMain.btStepClick(Sender: TObject);
begin
    Advance;
end;

procedure TfrmMain.FormCreate(Sender: TObject);
begin
    InitializeAll;
end;

procedure TfrmMain.btResetClick(Sender: TObject);
begin
    InitializeAll;
    btStep.Enabled := true;
end;

procedure TfrmMain.Advance;
var
    i: integer;
    j: integer;
    S: integer;
    E: integer;
begin
    case Step of
        1:
            begin
                PopulateBuffA(Tact);
                PopulateBuffB(Tact);
                BindBuffA;
                BindBuffB;
                for i := 1 to 5 do
                    ISA[1,i].BIn := BuffB[i];
                for i := 1 to 5 do

```

```

    ISA[i,1].AIn := BuffA[i];
    edOperation.Caption := 'refreshed buffers';
end;
2:
begin
    //move AIn to A and BIn to B
    for i := 1 to 5 do
        for j := 1 to 5 do
            begin
                ISA[i,j].A := ISA[i,j].AIn;
                ISA[i,j].B := ISA[i,j].BIn;
            end;
        //export A and B
        for i := 1 to 5 do
            for j := 1 to 5 do
                begin
                    S := i + 1;
                    E := j + 1;
                    if E < 6 then
                        ISA[i,E].AIn := ISA[i,j].A;
                    if S < 6 then
                        ISA[S,j].BIn := ISA[I,J].B;
                    end;
                endOperation.Caption := 'exec. communication';
            end;
        3:
        begin
            //sum A and B
            for i := 1 to 5 do
                for j := 1 to 5 do
                    begin
                        if (ISA[i,j].A <> -1) and (ISA[i,j].B <> -1) then
                            ISA[i,j].R := ISA[i,j].R + ISA[i,j].A * ISA[i,j].B;
                        end;
                    endOperation.Caption := 'computed new value';
                    Inc(Tact);
                    Step := 0;
                end;
            end;
            Inc(Step);
            edStep.Caption := IntToStr(Tact-1);
            BindISA;
            if tact = 15 then
                btStep.Enabled := false;
            end;

        procedure TfrmMain.InitializeAll;
        begin
            Tact := 1;

```

```

Step := 1;
InitializeBuffers;
InitializeISA;
BindBuffA;
BindBuffB;
BindISA;
edStep.Caption := "";
edOperation.Caption := "";
end;

```

```

procedure TfrmMain.Button1Click(Sender: TObject);
begin
Application.MainForm.close;
end;
end.

```

B.2 Program Output Results

B.2.1 Sequential Rhotrix Product [$R_5(A) \times R_5(B)$]

EZUGWU, El-Shamir Absalom MSC/SCIE/00043/2008-2009

Date : Nov 30 2010

Time : 06:08:47

=====
The First Rhotrix $R(A)[13]$ Is:

```

R(A)[1] = 2
R(A)[2] = 3
R(A)[3] = 1
R(A)[4] = 4
R(A)[5] = 4
R(A)[6] = 2
R(A)[7] = 4
R(A)[8] = 5
R(A)[9] = 3
R(A)[10] = 1
R(A)[11] = 5
R(A)[12] = 4
R(A)[13] = 6

```

The Second Rhotrix $R(B)[13]$ Is:

```

R(B)[1] = 2
R(B)[2] = 4
R(B)[3] = 5
R(B)[4] = 2
R(B)[5] = 1
R(B)[6] = 3

```

$R(B)[7] = 10$
 $R(B)[8] = 5$
 $R(B)[9] = 3$
 $R(B)[10] = 2$
 $R(B)[11] = 1$
 $R(B)[12] = 2$
 $R(B)[13] = 4$

Multiplication Of The Above Two Rhotrices Are:

$R[1] = 28$
 $R[2] = 46$
 $R[3] = 30$
 $R[4] = 48$
 $R[5] = 44$
 $R[6] = 32$
 $R[7] = 40$
 $R[8] = 70$
 $R[9] = 42$
 $R[10] = 18$
 $R[11] = 54$
 $R[12] = 48$
 $R[13] = 76$

B.2.2 Parallel Rhotrix Product [$R_5(A) \times R_5(B)$]

NNumber of Processor = 3

Master sending Rhotrix_A[0]=2 Rhotrix_B[0]=2 to worker 1
Master sending Rhotrix_A[1]=3 Rhotrix_B[1]=4 to worker 2
Master receiving a result back from worker 1 R[0]=28
Master receiving a result back from worker 2 R[1]=46
Master sending Rhotrix_A[2]=1 Rhotrix_B[2]=5 to worker 1
Master sending Rhotrix_A[3]=4 Rhotrix_B[3]=2 to worker 2
Master receiving a result back from worker 1 R[2]=30
Master receiving a result back from worker 2 R[3]=48
Master sending Rhotrix_A[4]=4 Rhotrix_B[4]=1 to worker 1
Master sending Rhotrix_A[5]=2 Rhotrix_B[5]=3 to worker 2
Master receiving a result back from worker 1 R[4]=44
Master receiving a result back from worker 2 R[5]=32
Master sending Rhotrix_A[6]=4 Rhotrix_B[6]=10 to worker 1
Master sending Rhotrix_A[7]=5 Rhotrix_B[7]=5 to worker 2
Master receiving a result back from worker 1 R[6]=80
Master receiving a result back from worker 2 R[7]=70
Master sending Rhotrix_A[8]=3 Rhotrix_B[8]=3 to worker 1
Master sending Rhotrix_A[9]=1 Rhotrix_B[9]=2 to worker 2
Master receiving a result back from worker 1 R[8]=42
Master receiving a result back from worker 2 R[9]=18
Master sending Rhotrix_A[10]=5 Rhotrix_B[10]=1 to worker 1
Master sending Rhotrix_A[11]=4 Rhotrix_B[11]=2 to worker 2
Master receiving a result back from worker 1 R[10]=54
Master receiving a result back from worker 2 R[11]=48
Master sending Rhotrix_A[12]=6 Rhotrix_B[12]=4 to worker 1
Master receiving a result back from worker 1 R[12]=76
Master telling worker 1 to shut down
Master telling worker 2 to shut down

Worker 1 Ready
Worker 1 received message: Rhotrix_A = 2 and Rhotrix_B = 2
Worker 1 Returning an answer for Rhotrix_A x hb = 2 x 10 + Rhotrix_B x ha = 2 x 4 = 28
Worker 1 received message: Rhotrix_A = 1 and Rhotrix_B = 5
Worker 1 Returning an answer for Rhotrix_A x hb = 1 x 10 + Rhotrix_B x ha = 5 x 4 = 30
Worker 1 received message: Rhotrix_A = 4 and Rhotrix_B = 1
Worker 1 Returning an answer for Rhotrix_A x hb = 4 x 10 + Rhotrix_B x ha = 1 x 4 = 44
Worker 1 received message: Rhotrix_A = 4 and Rhotrix_B = 10
Worker 1 Returning an answer for Rhotrix_A x hb = 4 x 10 + Rhotrix_B x ha = 10 x 4 = 80
Worker 1 received message: Rhotrix_A = 3 and Rhotrix_B = 3
Worker 1 Returning an answer for Rhotrix_A x hb = 3 x 10 + Rhotrix_B x ha = 3 x 4 = 42

```

worker 1 received message: Rhotrix_A = 5 and Rhotrix_B = 1
worker 1 Returning an answer for Rhotrix_A x hb = 5 x 10 + Rhotrix_B x ha = 1 x 4 = 54
worker 1 received message: Rhotrix_A = 6 and Rhotrix_B = 4
worker 1 Returning an answer for Rhotrix_A x hb = 6 x 10 + Rhotrix_B x ha = 4 x 4 = 76
worker 1 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 1 Shutting Down

```

```

worker 2 Ready
worker 2 received message: Rhotrix_A = 3 and Rhotrix_B = 4
worker 2 Returning an answer for Rhotrix_A x hb = 3 x 10 + Rhotrix_B x ha = 4 x 4 = 46
worker 2 received message: Rhotrix_A = 4 and Rhotrix_B = 2
worker 2 Returning an answer for Rhotrix_A x hb = 4 x 10 + Rhotrix_B x ha = 2 x 4 = 48
worker 2 received message: Rhotrix_A = 2 and Rhotrix_B = 3
worker 2 Returning an answer for Rhotrix_A x hb = 2 x 10 + Rhotrix_B x ha = 3 x 4 = 32
worker 2 received message: Rhotrix_A = 5 and Rhotrix_B = 5
worker 2 Returning an answer for Rhotrix_A x hb = 5 x 10 + Rhotrix_B x ha = 5 x 4 = 70
worker 2 received message: Rhotrix_A = 1 and Rhotrix_B = 2
worker 2 Returning an answer for Rhotrix_A x hb = 1 x 10 + Rhotrix_B x ha = 2 x 4 = 18
worker 2 received message: Rhotrix_A = 4 and Rhotrix_B = 2
worker 2 Returning an answer for Rhotrix_A x hb = 4 x 10 + Rhotrix_B x ha = 2 x 4 = 48
worker 2 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 2 Shutting Down

```

```

a[0] x hb + b[0] x ha => 2 x 10 + 2 x 4 = 28
a[1] x hb + b[1] x ha => 3 x 10 + 4 x 4 = 46
a[2] x hb + b[2] x ha => 1 x 10 + 5 x 4 = 30
a[3] x hb + b[3] x ha => 4 x 10 + 2 x 4 = 48
a[4] x hb + b[4] x ha => 4 x 10 + 1 x 4 = 44
a[5] x hb + b[5] x ha => 2 x 10 + 3 x 4 = 32
a[6] x hb + b[6] x ha => 4 x 10 + 10 x 4 = 80
a[7] x hb + b[7] x ha => 5 x 10 + 5 x 4 = 70
a[8] x hb + b[8] x ha => 3 x 10 + 3 x 4 = 42
a[9] x hb + b[9] x ha => 1 x 10 + 2 x 4 = 18
a[10] x hb + b[10] x ha => 5 x 10 + 1 x 4 = 54
a[11] x hb + b[11] x ha => 4 x 10 + 2 x 4 = 48
a[12] x hb + b[12] x ha => 6 x 10 + 4 x 4 = 76

```

Total parallel time : [0.066955] seconds

B.2.3 Parallel Rhotrix Product [$R_9(A) \times R_9(B)$]

Number of Processor = 6

```

Master sending Rhotrix_A[0]=2 Rhotrix_B[0]=2 to worker 1
Master sending Rhotrix_A[1]=3 Rhotrix_B[1]=4 to worker 2
Master sending Rhotrix_A[2]=1 Rhotrix_B[2]=5 to worker 3
Master sending Rhotrix_A[3]=4 Rhotrix_B[3]=2 to worker 4
Master sending Rhotrix_A[4]=4 Rhotrix_B[4]=1 to worker 5
Master receiving a result back from worker 1 R[0]=28
Master receiving a result back from worker 2 R[1]=46
Master receiving a result back from worker 3 R[2]=30
Master receiving a result back from worker 4 R[3]=48
Master receiving a result back from worker 5 R[4]=44
Master sending Rhotrix_A[5]=2 Rhotrix_B[5]=3 to worker 1
Master sending Rhotrix_A[6]=4 Rhotrix_B[6]=10 to worker 2
Master sending Rhotrix_A[7]=5 Rhotrix_B[7]=5 to worker 3
Master sending Rhotrix_A[8]=3 Rhotrix_B[8]=3 to worker 4
Master sending Rhotrix_A[9]=1 Rhotrix_B[9]=2 to worker 5
Master receiving a result back from worker 1 R[5]=32
Master receiving a result back from worker 3 R[7]=70
Master receiving a result back from worker 4 R[8]=42
Master receiving a result back from worker 2 R[6]=80
Master receiving a result back from worker 5 R[9]=18
Master sending Rhotrix_A[10]=5 Rhotrix_B[10]=1 to worker 1
Master sending Rhotrix_A[11]=4 Rhotrix_B[11]=2 to worker 2
Master sending Rhotrix_A[12]=6 Rhotrix_B[12]=4 to worker 3
Master receiving a result back from worker 1 R[10]=54
Master receiving a result back from worker 3 R[12]=76
Master receiving a result back from worker 2 R[11]=48
Master telling worker 1 to shut down
Master telling worker 2 to shut down
Master telling worker 3 to shut down
Master telling worker 4 to shut down
Master telling worker 5 to shut down

```

```

a[0] x hb + b[0] x ha => 2 x 10 + 2 x 4 = 28
a[1] x hb + b[1] x ha => 3 x 10 + 4 x 4 = 46
a[2] x hb + b[2] x ha => 1 x 10 + 5 x 4 = 30

```

```

a[3] x hb + b[3] x ha => 4 x 10 + 2 x 4 = 48
a[4] x hb + b[4] x ha => 4 x 10 + 1 x 4 = 44
a[5] x hb + b[5] x ha => 2 x 10 + 3 x 4 = 32
a[6] x hb + b[6] x ha => 4 x 10 + 10 x 4 = 80
a[7] x hb + b[7] x ha => 5 x 10 + 5 x 4 = 70
a[8] x hb + b[8] x ha => 3 x 10 + 3 x 4 = 42
a[9] x hb + b[9] x ha => 1 x 10 + 2 x 4 = 18
a[10] x hb + b[10] x ha => 5 x 10 + 1 x 4 = 54
a[11] x hb + b[11] x ha => 4 x 10 + 2 x 4 = 48
a[12] x hb + b[12] x ha => 6 x 10 + 4 x 4 = 76
Total time using processors [0] : [0.173322] seconds

```

```

worker 4 Ready
worker 4 received message: Rhotrix_A = 4 and Rhotrix_B = 2
worker 4 Returning an answer for Rhotrix_A x hb = 4 x 10 + Rhotrix_B x ha = 2 x 4 = 48
worker 4 received message: Rhotrix_A = 3 and Rhotrix_B = 3
worker 4 Returning an answer for Rhotrix_A x hb = 3 x 10 + Rhotrix_B x ha = 3 x 4 = 42
worker 4 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 4 Shutting Down

```

```

worker 3 Ready
worker 3 received message: Rhotrix_A = 1 and Rhotrix_B = 5
worker 3 Returning an answer for Rhotrix_A x hb = 1 x 10 + Rhotrix_B x ha = 5 x 4 = 30
worker 3 received message: Rhotrix_A = 5 and Rhotrix_B = 5
worker 3 Returning an answer for Rhotrix_A x hb = 5 x 10 + Rhotrix_B x ha = 5 x 4 = 70
worker 3 received message: Rhotrix_A = 6 and Rhotrix_B = 4
worker 3 Returning an answer for Rhotrix_A x hb = 6 x 10 + Rhotrix_B x ha = 4 x 4 = 76
worker 3 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 3 Shutting Down

```

```

worker 1 Ready
worker 1 received message: Rhotrix_A = 2 and Rhotrix_B = 2
worker 1 Returning an answer for Rhotrix_A x hb = 2 x 10 + Rhotrix_B x ha = 2 x 4 = 28
worker 1 received message: Rhotrix_A = 2 and Rhotrix_B = 3
worker 1 Returning an answer for Rhotrix_A x hb = 2 x 10 + Rhotrix_B x ha = 3 x 4 = 32
worker 1 received message: Rhotrix_A = 5 and Rhotrix_B = 1
worker 1 Returning an answer for Rhotrix_A x hb = 5 x 10 + Rhotrix_B x ha = 1 x 4 = 54
worker 1 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 1 Shutting Down

```

```

worker 5 Ready
worker 5 received message: Rhotrix_A = 4 and Rhotrix_B = 1
worker 5 Returning an answer for Rhotrix_A x hb = 4 x 10 + Rhotrix_B x ha = 1 x 4 = 44
worker 5 received message: Rhotrix_A = 1 and Rhotrix_B = 2
worker 5 Returning an answer for Rhotrix_A x hb = 1 x 10 + Rhotrix_B x ha = 2 x 4 = 18
worker 5 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 5 Shutting Down

```

```

worker 2 Ready
worker 2 received message: Rhotrix_A = 3 and Rhotrix_B = 4
worker 2 Returning an answer for Rhotrix_A x hb = 3 x 10 + Rhotrix_B x ha = 4 x 4 = 46
worker 2 received message: Rhotrix_A = 4 and Rhotrix_B = 10
worker 2 Returning an answer for Rhotrix_A x hb = 4 x 10 + Rhotrix_B x ha = 10 x 4 = 80
worker 2 received message: Rhotrix_A = 4 and Rhotrix_B = 2
worker 2 Returning an answer for Rhotrix_A x hb = 4 x 10 + Rhotrix_B x ha = 2 x 4 = 48
worker 2 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 2 Shutting Down

```

B.2.4 Sequential Rhotrix Product [$R_9(A) \times R_9(B)$]

EZUGWU, El-Shamir Absalom MSC/SCIE/00043/2008-2009

Date : Nov 30 2010

Time : 06:08:47

=====

The First Rhotrix $R(A)[0]$ Is:

```

R(A)[1] = 1
R(A)[2] = 2
R(A)[3] = 3
R(A)[4] = 4
R(A)[5] = 1
R(A)[6] = 2
R(A)[7] = 7

```

$R(A)[8] = 1$
 $R(A)[9] = 4$
 $R(A)[10] = 1$
 $R(A)[11] = 3$
 $R(A)[12] = 8$
 $R(A)[13] = 1$
 $R(A)[14] = 2$
 $R(A)[15] = 7$
 $R(A)[16] = 4$
 $R(A)[17] = 1$
 $R(A)[18] = 3$
 $R(A)[19] = 7$
 $R(A)[20] = 4$
 $R(A)[21] = 2$
 $R(A)[22] = 1$
 $R(A)[23] = 6$
 $R(A)[24] = 1$
 $R(A)[25] = 7$
 $R(A)[26] = 2$
 $R(A)[27] = 1$
 $R(A)[28] = 3$
 $R(A)[29] = 1$
 $R(A)[30] = 4$
 $R(A)[31] = 1$
 $R(A)[32] = 6$
 $R(A)[33] = 2$
 $R(A)[34] = 4$
 $R(A)[35] = 1$
 $R(A)[36] = 3$
 $R(A)[37] = 5$
 $R(A)[38] = 3$
 $R(A)[39] = 2$
 $R(A)[40] = 1$
 $R(A)[41] = 4$

The Second Rhotrix $R(B)[0]$ is:

$R(B)[1] = 6$
 $R(B)[2] = 3$
 $R(B)[3] = 9$
 $R(B)[4] = 4$
 $R(B)[5] = 7$
 $R(B)[6] = 2$
 $R(B)[7] = 7$
 $R(B)[8] = 3$
 $R(B)[9] = 4$
 $R(B)[10] = 5$
 $R(B)[11] = 3$
 $R(B)[12] = 8$
 $R(B)[13] = 1$
 $R(B)[14] = 2$
 $R(B)[15] = 7$
 $R(B)[16] = 4$
 $R(B)[17] = 1$
 $R(B)[18] = 3$

$R(B)[19] = 7$
 $R(B)[20] = 4$
 $R(B)[21] = 5$
 $R(B)[22] = 1$
 $R(B)[23] = 6$
 $R(B)[24] = 1$
 $R(B)[25] = 7$
 $R(B)[26] = 2$
 $R(B)[27] = 1$
 $R(B)[28] = 3$
 $R(B)[29] = 1$
 $R(B)[30] = 4$
 $R(B)[31] = 1$
 $R(B)[32] = 6$
 $R(B)[33] = 2$
 $R(B)[34] = 4$
 $R(B)[35] = 1$
 $R(B)[36] = 3$
 $R(B)[37] = 5$
 $R(B)[38] = 3$
 $R(B)[39] = 2$
 $R(B)[40] = 1$
 $R(B)[41] = 4$

time_count = 10701083

Multiplication Of The Above Two Rhotrices Are:

$R[1] = 17$
 $R[2] = 16$
 $R[3] = 33$
 $R[4] = 28$
 $R[5] = 19$
 $R[6] = 14$
 $R[7] = 49$
 $R[8] = 11$
 $R[9] = 28$
 $R[10] = 15$
 $R[11] = 21$
 $R[12] = 56$
 $R[13] = 7$
 $R[14] = 14$
 $R[15] = 49$
 $R[16] = 28$
 $R[17] = 7$
 $R[18] = 21$
 $R[19] = 49$
 $R[20] = 28$
 $R[21] = 10$
 $R[22] = 7$
 $R[23] = 42$
 $R[24] = 7$
 $R[25] = 49$
 $R[26] = 14$

```

R[27] = 7
R[28] = 21
R[29] = 7
R[30] = 28
R[31] = 7
R[32] = 42
R[33] = 14
R[34] = 28
R[35] = 7
R[36] = 21
R[37] = 35
R[38] = 21
R[39] = 14
R[40] = 7
R[41] = 28

```

B.2.5 Parallel Heart-Oriented Rhotrix Output[$R_9(A) \times R_9(B)$]

Number of processor = 12

```

Master sending Rhotrix_A[0]=6 Rhotrix_B[0]=6 to worker 1
Master sending Rhotrix_A[1]=3 Rhotrix_B[1]=3 to worker 2
Master sending Rhotrix_A[2]=9 Rhotrix_B[2]=9 to worker 3
Master sending Rhotrix_A[3]=4 Rhotrix_B[3]=4 to worker 4
Master sending Rhotrix_A[4]=7 Rhotrix_B[4]=7 to worker 5
Master sending Rhotrix_A[5]=2 Rhotrix_B[5]=2 to worker 6
Master sending Rhotrix_A[6]=7 Rhotrix_B[6]=7 to worker 7
Master sending Rhotrix_A[7]=3 Rhotrix_B[7]=3 to worker 8
Master sending Rhotrix_A[8]=4 Rhotrix_B[8]=4 to worker 9
Master sending Rhotrix_A[9]=5 Rhotrix_B[9]=5 to worker 10
Master sending Rhotrix_A[10]=3 Rhotrix_B[10]=3 to worker 11
Master receiving a result back from worker 1 R[0]=42
Master receiving a result back from worker 2 R[1]=21
Master receiving a result back from worker 3 R[2]=63
Master receiving a result back from worker 4 R[3]=28
Master receiving a result back from worker 5 R[4]=49
Master receiving a result back from worker 6 R[5]=14
Master receiving a result back from worker 7 R[6]=49
Master receiving a result back from worker 8 R[7]=21
Master receiving a result back from worker 9 R[8]=28
Master receiving a result back from worker 10 R[9]=35
Master receiving a result back from worker 11 R[10]=21
Master sending Rhotrix_A[11]=8 Rhotrix_B[11]=8 to worker 1
Master sending Rhotrix_A[12]=1 Rhotrix_B[12]=1 to worker 2
Master sending Rhotrix_A[13]=2 Rhotrix_B[13]=2 to worker 3
Master sending Rhotrix_A[14]=7 Rhotrix_B[14]=7 to worker 4
Master sending Rhotrix_A[15]=4 Rhotrix_B[15]=4 to worker 5
Master sending Rhotrix_A[16]=1 Rhotrix_B[16]=1 to worker 6
Master sending Rhotrix_A[17]=3 Rhotrix_B[17]=3 to worker 7
Master sending Rhotrix_A[18]=7 Rhotrix_B[18]=7 to worker 8
Master sending Rhotrix_A[19]=4 Rhotrix_B[19]=4 to worker 9
Master sending Rhotrix_A[20]=5 Rhotrix_B[20]=5 to worker 10
Master sending Rhotrix_A[21]=1 Rhotrix_B[21]=1 to worker 11
Master receiving a result back from worker 3 R[13]=14
Master receiving a result back from worker 5 R[15]=28
Master receiving a result back from worker 7 R[17]=21
Master receiving a result back from worker 10 R[20]=35
Master receiving a result back from worker 11 R[21]=7
Master receiving a result back from worker 1 R[11]=56
Master receiving a result back from worker 2 R[12]=7
Master receiving a result back from worker 4 R[14]=49
Master receiving a result back from worker 6 R[16]=7
Master receiving a result back from worker 8 R[18]=49
Master receiving a result back from worker 9 R[19]=28
Master sending Rhotrix_A[22]=6 Rhotrix_B[22]=6 to worker 1
Master sending Rhotrix_A[23]=1 Rhotrix_B[23]=1 to worker 2
Master sending Rhotrix_A[24]=7 Rhotrix_B[24]=7 to worker 3
Master sending Rhotrix_A[25]=2 Rhotrix_B[25]=2 to worker 4
Master sending Rhotrix_A[26]=1 Rhotrix_B[26]=1 to worker 5
Master sending Rhotrix_A[27]=3 Rhotrix_B[27]=3 to worker 6
Master sending Rhotrix_A[28]=1 Rhotrix_B[28]=1 to worker 7

```

```

Master sending Rhotrix_A[29]=4 Rhotrix_B[29]=4 to worker 8
Master sending Rhotrix_A[30]=1 Rhotrix_B[30]=1 to worker 9
Master sending Rhotrix_A[31]=6 Rhotrix_B[31]=6 to worker 10
Master sending Rhotrix_A[32]=2 Rhotrix_B[32]=2 to worker 11
Master receiving a result back from worker 1 R[22]=42
Master receiving a result back from worker 2 R[23]=7
Master receiving a result back from worker 4 R[25]=14
Master receiving a result back from worker 6 R[27]=21
Master receiving a result back from worker 8 R[29]=28
Master receiving a result back from worker 9 R[30]=7
Master receiving a result back from worker 3 R[24]=49
Master receiving a result back from worker 5 R[26]=7
Master receiving a result back from worker 7 R[28]=7
Master receiving a result back from worker 10 R[31]=42
Master receiving a result back from worker 11 R[32]=14
Master sending Rhotrix_A[33]=4 Rhotrix_B[33]=4 to worker 1
Master sending Rhotrix_A[34]=1 Rhotrix_B[34]=1 to worker 2
Master sending Rhotrix_A[35]=3 Rhotrix_B[35]=3 to worker 3
Master sending Rhotrix_A[36]=5 Rhotrix_B[36]=5 to worker 4
Master sending Rhotrix_A[37]=3 Rhotrix_B[37]=3 to worker 5
Master sending Rhotrix_A[38]=2 Rhotrix_B[38]=2 to worker 6
Master sending Rhotrix_A[39]=1 Rhotrixworker 6 Ready
worker 6 received message: Rhotrix_A = 2 and Rhotrix_B = 2
worker 6 Returning an answer for Rhotrix_A x hb = 2 x 5 + Rhotrix_B x ha = 2 x 2 = 14
worker 6 received message: Rhotrix_A = 1 and Rhotrix_B = 1
worker 6 Returning an answer for Rhotrix_A x hb = 1 x 5 + Rhotrix_B x ha = 1 x 2 = 7
worker 6 received message: Rhotrix_A = 3 and Rhotrix_B = 3
worker 6 Returning an answer for Rhotrix_A x hb = 3 x 5 + Rhotrix_B x ha = 3 x 2 = 21
worker 6 received message: Rhotrix_A = 2 and Rhotrix_B = 2
worker 6 Returning an answer for Rhotrix_A x hb = 2 x 5 + Rhotrix_B x ha = 2 x 2 = 14
worker 6 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 6 Shutting Down

worker 8 Ready
worker 8 received message: Rhotrix_A = 3 and Rhotrix_B = 3
worker 8 Returning an answer for Rhotrix_A x hb = 3 x 5 + Rhotrix_B x ha = 3 x 2 = 21
worker 8 received message: Rhotrix_A = 7 and Rhotrix_B = 7
worker 8 Returning an answer for Rhotrix_A x hb = 7 x 5 + Rhotrix_B x ha = 7 x 2 = 49
worker 8 received message: Rhotrix_A = 4 and Rhotrix_B = 4
worker 8 Returning an answer for Rhotrix_A x hb = 4 x 5 + Rhotrix_B x ha = 4 x 2 = 28
worker 8 received message: Rhotrix_A = 4 and Rhotrix_B = 4
worker 8 Returning an answer for Rhotrix_A x hb = 4 x 5 + Rhotrix_B x ha = 4 x 2 = 28
worker 8 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 8 Shutting Down

worker 9 Ready
worker 9 received message: Rhotrix_A = 4 and Rhotrix_B = 4
worker 9 Returning an answer for Rhotrix_A x hb = 4 x 5 + Rhotrix_B x ha = 4 x 2 = 28
worker 9 received message: Rhotrix_A = 4 and Rhotrix_B = 4
worker 9 Returning an answer for Rhotrix_A x hb = 4 x 5 + Rhotrix_B x ha = 4 x 2 = 28
worker 9 received message: Rhotrix_A = 1 and Rhotrix_B = 1
worker 9 Returning an answer for Rhotrix_A x hb = 1 x 5 + Rhotrix_B x ha = 1 x 2 = 7
worker 9 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 9 Shutting Down

worker 7 Ready
worker 7 received message: Rhotrix_A = 7 and Rhotrix_B = 7
worker 7 Returning an answer for Rhotrix_A x hb = 7 x 5 + Rhotrix_B x ha = 7 x 2 = 49
worker 7 received message: Rhotrix_A = 3 and Rhotrix_B = 3
worker 7 Returning an answer for Rhotrix_A x hb = 3 x 5 + Rhotrix_B x ha = 3 x 2 = 21
worker 7 received message: Rhotrix_A = 1 and Rhotrix_B = 1
worker 7 Returning an answer for Rhotrix_A x hb = 1 x 5 + Rhotrix_B x ha = 1 x 2 = 7
worker 7 received message: Rhotrix_A = 1 and Rhotrix_B = 1
worker 7 Returning an answer for Rhotrix_A x hb = 1 x 5 + Rhotrix_B x ha = 1 x 2 = 7
worker 7 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 7 Shutting Down

worker 3 Ready
worker 3 received message: Rhotrix_A = 9 and Rhotrix_B = 9
worker 3 Returning an answer for Rhotrix_A x hb = 9 x 5 + Rhotrix_B x ha = 9 x 2 = 63
worker 3 received message: Rhotrix_A = 2 and Rhotrix_B = 2
worker 3 Returning an answer for Rhotrix_A x hb = 2 x 5 + Rhotrix_B x ha = 2 x 2 = 14
worker 3 received message: Rhotrix_A = 7 and Rhotrix_B = 7
worker 3 Returning an answer for Rhotrix_A x hb = 7 x 5 + Rhotrix_B x ha = 7 x 2 = 49
worker 3 received message: Rhotrix_A = 3 and Rhotrix_B = 3
worker 3 Returning an answer for Rhotrix_A x hb = 3 x 5 + Rhotrix_B x ha = 3 x 2 = 21
worker 3 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 3 Shutting Down

worker 1 Ready

```

```

worker 1 received message: Rhotrix_A = 6 and Rhotrix_B = 6
worker 1 Returning an answer for Rhotrix_A x hb = 6 x 5 + Rhotrix_B x ha = 6 x 2 = 42
worker 1 received message: Rhotrix_A = 8 and Rhotrix_B = 8
worker 1 Returning an answer for Rhotrix_A x hb = 8 x 5 + Rhotrix_B x ha = 8 x 2 = 56
worker 1 received message: Rhotrix_A = 6 and Rhotrix_B = 6
worker 1 Returning an answer for Rhotrix_A x hb = 6 x 5 + Rhotrix_B x ha = 6 x 2 = 42
worker 1 received message: Rhotrix_A = 4 and Rhotrix_B = 4
worker 1 Returning an answer for Rhotrix_A x hb = 4 x 5 + Rhotrix_B x ha = 4 x 2 = 28
worker 1 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 1 Shutting Down

worker 5 Ready
worker 5 received message: Rhotrix_A = 7 and Rhotrix_B = 7
worker 5 Returning an answer for Rhotrix_A x hb = 7 x 5 + Rhotrix_B x ha = 7 x 2 = 49
worker 5 received message: Rhotrix_A = 4 and Rhotrix_B = 4
worker 5 Returning an answer for Rhotrix_A x hb = 4 x 5 + Rhotrix_B x ha = 4 x 2 = 28
worker 5 received message: Rhotrix_A = 1 and Rhotrix_B = 1
worker 5 Returning an answer for Rhotrix_A x hb = 1 x 5 + Rhotrix_B x ha = 1 x 2 = 7
worker 5 received message: Rhotrix_A = 3 and Rhotrix_B = 3
worker 5 Returning an answer for Rhotrix_A x hb = 3 x 5 + Rhotrix_B x ha = 3 x 2 = 21
worker 5 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 5 Shutting Down
Total time using processors [5] : [74919.148762] seconds
worker 4 Ready
worker 4 received message: Rhotrix_A = 4 and Rhotrix_B = 4
worker 4 Returning an answer for Rhotrix_A x hb = 4 x 5 + Rhotrix_B x ha = 4 x 2 = 28
worker 4 received message: Rhotrix_A = 7 and Rhotrix_B = 7
worker 4 Returning an answer for Rhotrix_A x hb = 7 x 5 + Rhotrix_B x ha = 7 x 2 = 49
worker 4 received message: Rhotrix_A = 2 and Rhotrix_B = 2
worker 4 Returning an answer for Rhotrix_A x hb = 2 x 5 + Rhotrix_B x ha = 2 x 2 = 14
worker 4 received message: Rhotrix_A = 5 and Rhotrix_B = 5
worker 4 Returning an answer for Rhotrix_A x hb = 5 x 5 + Rhotrix_B x ha = 5 x 2 = 35
worker 4 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 4 Shutting Down
Total time using processors [4] : [74919.144545] seconds
worker 2 Ready
worker 2 received message: Rhotrix_A = 3 and Rhotrix_B = 3
worker 2 Returning an answer for Rhotrix_A x hb = 3 x 5 + Rhotrix_B x ha = 3 x 2 = 21
worker 2 received message: Rhotrix_A = 1 and Rhotrix_B = 1
worker 2 Returning an answer for Rhotrix_A x hb = 1 x 5 + Rhotrix_B x ha = 1 x 2 = 7
worker 2 received message: Rhotrix_A = 1 and Rhotrix_B = 1
worker 2 Returning an answer for Rhotrix_A x hb = 1 x 5 + Rhotrix_B x ha = 1 x 2 = 7
worker 2 received message: Rhotrix_A = 1 and Rhotrix_B = 1
worker 2 Returning an answer for Rhotrix_A x hb = 1 x 5 + Rhotrix_B x ha = 1 x 2 = 7
worker 2 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 2 Shutting Down
Total time using processors [2] : [74919.144336] seconds
worker 10 Ready
worker 10 received message: Rhotrix_A = 5 and Rhotrix_B = 5
worker 10 Returning an answer for Rhotrix_A x hb = 5 x 5 + Rhotrix_B x ha = 5 x 2 = 35
worker 10 received message: Rhotrix_A = 5 and Rhotrix_B = 5
worker 10 Returning an answer for Rhotrix_A x hb = 5 x 5 + Rhotrix_B x ha = 5 x 2 = 35
worker 10 received message: Rhotrix_A = 6 and Rhotrix_B = 6
worker 10 Returning an answer for Rhotrix_A x hb = 6 x 5 + Rhotrix_B x ha = 6 x 2 = 42
worker 10 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 10 Shutting Down
Total time using processors [10] : [74919.149089] seconds
_B[39]=1 to worker 7
Master sending Rhotrix_A[40]=4 Rhotrix_B[40]=4 to worker 8
Master receiving a result back from worker 1 R[33]=28
Master receiving a result back from worker 2 R[34]=7
Master receiving a result back from worker 4 R[36]=35
Master receiving a result back from worker 6 R[38]=14
Master receiving a result back from worker 3 R[35]=21
Master receiving a result back from worker 5 R[37]=21
Master receiving a result back from worker 7 R[39]=7
Master receiving a result back from worker 8 R[40]=28
Master telling worker 1 to shut down
Master telling worker 2 to shut down
Master telling worker 3 to shut down
Master telling worker 4 to shut down
Master telling worker 5 to shut down
Master telling worker 6 to shut down
Master telling worker 7 to shut down
Master telling worker 8 to shut down
Master telling worker 9 to shut down
Master telling worker 10 to shut down
Master telling worker 11 to shut down
a[0] x hb + b[0] x ha => 6 x 5 + 6 x 2 = 42
a[1] x hb + b[1] x ha => 3 x 5 + 3 x 2 = 21

```

```

a[2] x hb + b[2] x ha => 9 x 5 + 9 x 2 = 63
a[3] x hb + b[3] x ha => 4 x 5 + 4 x 2 = 28
a[4] x hb + b[4] x ha => 7 x 5 + 7 x 2 = 49
a[5] x hb + b[5] x ha => 2 x 5 + 2 x 2 = 14
a[6] x hb + b[6] x ha => 7 x 5 + 7 x 2 = 49
a[7] x hb + b[7] x ha => 3 x 5 + 3 x 2 = 21
a[8] x hb + b[8] x ha => 4 x 5 + 4 x 2 = 28
a[9] x hb + b[9] x ha => 5 x 5 + 5 x 2 = 35
a[10] x hb + b[10] x ha => 3 x 5 + 3 x 2 = 21
a[11] x hb + b[11] x ha => 8 x 5 + 8 x 2 = 56
a[12] x hb + b[12] x ha => 1 x 5 + 1 x 2 = 7
a[13] x hb + b[13] x ha => 2 x 5 + 2 x 2 = 14
a[14] x hb + b[14] x ha => 7 x 5 + 7 x 2 = 49
a[15] x hb + b[15] x ha => 4 x 5 + 4 x 2 = 28
a[16] x hb + b[16] x ha => 1 x 5 + 1 x 2 = 7
a[17] x hb + b[17] x ha => 3 x 5 + 3 x 2 = 21
a[18] x hb + b[18] x ha => 7 x 5 + 7 x 2 = 49
a[19] x hb + b[19] x ha => 4 x 5 + 4 x 2 = 28
a[20] x hb + b[20] x ha => 5 x 5 + 5 x 2 = 35
a[21] x hb + b[21] x ha => 1 x 5 + 1 x 2 = 7
a[22] x hb + b[22] x ha => 6 x 5 + 6 x 2 = 42
a[23] x hb + b[23] x ha => 1 x 5 + 1 x 2 = 7
a[24] x hb + b[24] x ha => 7 x 5 + 7 x 2 = 49
a[25] x hb + b[25] x ha => 2 x 5 + 2 x 2 = 14
a[26] x hb + b[26] x ha => 1 x 5 + 1 x 2 = 7
a[27] x hb + b[27] x ha => 3 x 5 + 3 x 2 = 21
a[28] x hb + b[28] x ha => 1 x 5 + 1 x 2 = 7
a[29] x hb + b[29] x ha => 4 x 5 + 4 x 2 = 28
a[30] x hb + b[30] x ha => 1 x 5 + 1 x 2 = 7
a[31] x hb + b[31] x ha => 6 x 5 + 6 x 2 = 42
a[32] x hb + b[32] x ha => 2 x 5 + 2 x 2 = 14
a[33] x hb + b[33] x ha => 4 x 5 + 4 x 2 = 28
a[34] x hb + b[34] x ha => 1 x 5 + 1 x 2 = 7
a[35] x hb + b[35] x ha => 3 x 5 + 3 x 2 = 21
a[36] x hb + b[36] x ha => 5 x 5 + 5 x 2 = 35
a[37] x hb + b[37] x ha => 3 x 5 + 3 x 2 = 21
a[38] x hb + b[38] x ha => 2 x 5 + 2 x 2 = 14
a[39] x hb + b[39] x ha => 1 x 5 + 1 x 2 = 7
a[40] x hb + b[40] x ha => 4 x 5 + 4 x 2 = 28
Total time using processors [0] : [0.344110] seconds
Worker 11 Ready
Worker 11 received message: Rhotrix_A = 3 and Rhotrix_B = 3
Worker 11 Returning an answer for Rhotrix_A x hb = 3 x 5 + Rhotrix_B x ha = 3 x 2 = 21
Worker 11 received message: Rhotrix_A = 1 and Rhotrix_B = 1
Worker 11 Returning an answer for Rhotrix_A x hb = 1 x 5 + Rhotrix_B x ha = 1 x 2 = 7
Worker 11 received message: Rhotrix_A = 2 and Rhotrix_B = 2
Worker 11 Returning an answer for Rhotrix_A x hb = 2 x 5 + Rhotrix_B x ha = 2 x 2 = 14
Worker 11 received message: Rhotrix_A = -1 and Rhotrix_B = -1
Worker 11 Shutting Down

```

B.2.5 Parallel Heart-Oriented Rhotrix Multiplication[$R_9(A) \times R_9(B)$]

Number of processor = 20

```

Master sending Rhotrix_A[0]=6 Rhotrix_B[0]=6 to worker 1
Master sending Rhotrix_A[1]=3 Rhotrix_B[1]=3 to worker 2
Master sending Rhotrix_A[2]=9 Rhotrix_B[2]=9 to worker 3
Master sending Rhotrix_A[3]=4 Rhotrix_B[3]=4 to worker 4
Master sending Rhotrix_A[4]=7 Rhotrix_B[4]=7 to worker 5
Master sending Rhotrix_A[5]=2 Rhotrix_B[5]=2 to worker 6
Master sending Rhotrix_A[6]=7 Rhotrix_B[6]=7 to worker 7
Master sending Rhotrix_A[7]=3 Rhotrix_B[7]=3 to worker 8
Master sending Rhotrix_A[8]=4 Rhotrix_B[8]=4 to worker 9
Master sending Rhotrix_A[9]=5 Rhotrix_B[9]=5 to worker 10
Master sending Rhotrix_A[10]=3 Rhotrix_B[10]=3 to worker 11
Master sending Rhotrix_A[11]=8 Rhotrix_B[11]=8 to worker 12
Master sending Rhotrix_A[12]=1 Rhotrix_B[12]=1 to worker 13
Master sending Rhotrix_A[13]=2 Rhotrix_B[13]=2 to worker 14
Master sending Rhotrix_A[14]=7 Rhotrix_B[14]=7 to worker 15
Master sending Rhotrix_A[15]=4 Rhotrix_B[15]=4 to worker 16
Master sending Rhotrix_A[16]=1 Rhotrix_B[16]=1 to worker 17
Master sending Rhotrix_A[17]=3 Rhotrix_B[17]=3 to worker 18
Master sending Rhotrix_A[18]=7 Rhotrix_B[18]=7 to worker 19
Master receiving a result back from worker 1 R[0]=42
Master receiving a result back from worker 2 R[1]=21
Master receiving a result back from worker 3 R[2]=63
Master receiving a result back from worker 4 R[3]=28
Master receiving a result back from worker 5 R[4]=49
Master receiving a result back from worker 6 R[5]=14

```

```

Master receiving a result back from worker 7 R[6]=49
Master receiving a result back from worker 8 R[7]=21
Master receiving a result back from worker 9 R[8]=28
Master receiving a result back from worker 10 R[9]=35
Master receiving a result back from worker 11 R[10]=21
Master receiving a result back from worker 12 R[11]=56
Master receiving a result back from worker 13 R[12]=7
Master receiving a result back from worker 14 R[13]=14
Master receiving a result back from worker 15 R[14]=49
Master receiving a result back from worker 16 R[15]=28
Master receiving a result back from worker 17 R[16]=7
Master receiving a result back from worker 18 R[17]=21
Master receiving a result back from worker 19 R[18]=49
Master sending Rhotrix_A[19]=4 Rhotrix_B[19]=4 to worker 1
Master sending Rhotrix_A[20]=5 Rhotrix_B[20]=5 to worker 2
Master sending Rhotrix_A[21]=1 Rhotrix_B[21]=1 to worker 3
Master sending Rhotrix_A[22]=6 Rhotrix_B[22]=6 to worker 4
Master sending Rhotrix_A[23]=1 Rhotrix_B[23]=1 to worker 5
Master sending Rhotrix_A[24]=7 Rhotrix_B[24]=7 to worker 6
Master sending Rhotrix_A[25]=2 Rhotrix_B[25]=2 to worker 7
Master sending Rhotrix_A[26]=1 Rhotrix_B[26]=1 to worker 8
Master sending Rhotrix_A[27]=3 Rhotrix_B[27]=3 to worker 9
Master sending Rhotrix_A[28]=1 Rhotrix_B[28]=1 to worker 10
Master sending Rhotrix_A[29]=4 Rhotrix_B[29]=4 to worker 11
Master sending Rhotrix_A[30]=1 Rhotrix_B[30]=1 to worker 12
Master sending Rhotrix_A[31]=6 Rhotrix_B[31]=6 to worker 13
Master sending Rhotrix_A[32]=2 Rhotrix_B[32]=2 to worker 14
Master sending Rhotrix_A[33]=4 Rhotrix_B[33]=4 to worker 15
Master sending Rhotrix_A[34]=1 Rhotrix_B[34]=1 to worker 16
Master sending Rhotrix_A[35]=3 Rhotrix_B[35]=3 to worker 17
Master sending Rhotrix_A[36]=5 Rhotrix_B[36]=5 to worker 18
Master sending Rhotrix_A[37]=3 Rhotrix_B[37]=3 to worker 19
Master receiving a result back from worker 2 R[20]=35
Master receiving a result back from worker 3 R[21]=7
Master receiving a result back from worker 4 R[22]=42
Master receiving a result back from worker 6 R[24]=49
Master receiving a result back from worker 8 R[26]=7
Master receiving a result back from worker 9 R[27]=21
Master receiving a result back from worker 13 R[31]=42
Master receiving a result back from worker 15 R[33]=28
Master receiving a result back from worker 18 R[36]=35
Master receiving a result back from worker 1 R[19]=28
Master receiving a result back from worker 5 R[23]=7
Master receiving a result back from worker 7 R[25]=14
Master receiving a result back from worker 10 R[28]=7
Master receiving a result back from worker 11 R[29]=28
Master receiving a result back from worker 12 R[30]=7
Master receiving a result back from worker 14 R[32]=14
Master receiving a result back from worker 16 R[34]=7
Master receiving a result back from worker 17 R[35]=21
Master receiving a result back from worker 19 R[37]=21
Master sending Rhotrix_A[38]=2 Rhotrix_B[38]=2 to worker 1
Master sending Rhotrix_A[39]=1 Rhotrix_B[39]=1 to worker 2
Master sending Rhotrix_A[40]=4 Rhotrix_B[40]=4 to worker 3
Master receiving a result back from worker 2 R[39]=7
Master receiving a result back from worker 3 R[40]=28
Master receiving a result back from worker 1 R[38]=14
Master telling worker 1 to shut down
Master telling worker 2 to shut down
Master telling worker 3 to shut down
Master telling worker 4 to shut down
Master telling worker 5 to shut down
Master telling worker 6 to shut down
Master telling worker 7 to shut down
Master telling worker 8 to shut down
Master telling worker 9 to shut down
Master telling worker 10 to shut down
Master telling worker 11 to shut down
Master telling worker 12 to shut down
Master telling worker 13 to shut down
Master telling worker 14 to shut down
Master telling worker 15 to shut down
Master telling worker 16 to shut down
Master telling worker 17 to shut down
Master telling worker 18 to shut down
Master telling worker 19 to shut down

```

```

a[0] x hb + b[0] x ha => 6 x 5 + 6 x 2 = 42
a[1] x hb + b[1] x ha => 3 x 5 + 3 x 2 = 21
a[2] x hb + b[2] x ha => 9 x 5 + 9 x 2 = 63

```

```

a[3] x hb + b[3] x ha => 4 x 5 + 4 x 2 = 28
a[4] x hb + b[4] x ha => 7 x 5 + 7 x 2 = 49
a[5] x hb + b[5] x ha => 2 x 5 + 2 x 2 = 14
a[6] x hb + b[6] x ha => 7 x 5 + 7 x 2 = 49
a[7] x hb + b[7] x ha => 3 x 5 + 3 x 2 = 21
a[8] x hb + b[8] x ha => 4 x 5 + 4 x 2 = 28
a[9] x hb + b[9] x ha => 5 x 5 + 5 x 2 = 35
a[10] x hb + b[10] x ha => 3 x 5 + 3 x 2 = 21
a[11] x hb + b[11] x ha => 8 x 5 + 8 x 2 = 56
a[12] x hb + b[12] x ha => 1 x 5 + 1 x 2 = 7
a[13] x hb + b[13] x ha => 2 x 5 + 2 x 2 = 14
a[14] x hb + b[14] x ha => 7 x 5 + 7 x 2 = 49
a[15] x hb + b[15] x ha => 4 x 5 + 4 x 2 = 28
a[16] x hb + b[16] x ha => 1 x 5 + 1 x 2 = 7
a[17] x hb + b[17] x ha => 3 x 5 + 3 x 2 = 21
a[18] x hb + b[18] x ha => 7 x 5 + 7 x 2 = 49
a[19] x hb + b[19] x ha => 4 x 5 + 4 x 2 = 28
a[20] x hb + b[20] x ha => 5 x 5 + 5 x 2 = 35
a[21] x hb + b[21] x ha => 1 x 5 + 1 x 2 = 7
a[22] x hb + b[22] x ha => 6 x 5 + 6 x 2 = 42
a[23] x hb + b[23] x ha => 1 x 5 + 1 x 2 = 7
a[24] x hb + b[24] x ha => 7 x 5 + 7 x 2 = 49
a[25] x hb + b[25] x ha => 2 x 5 + 2 x 2 = 14
a[26] x hb + b[26] x ha => 1 x 5 + 1 x 2 = 7
a[27] x hb + b[27] x ha => 3 x 5 + 3 x 2 = 21
a[28] x hb + b[28] x ha => 1 x 5 + 1 x 2 = 7
a[29] x hb + b[29] x ha => 4 x 5 + 4 x 2 = 28
a[30] x hb + b[30] x ha => 1 x 5 + 1 x 2 = 7
a[31] x hb + b[31] x ha => 6 x 5 + 6 x 2 = 42
a[32] x hb + b[32] x ha => 2 x 5 + 2 x 2 = 14
a[33] x hb + b[33] x ha => 4 x 5 + 4 x 2 = 28
a[34] x hb + b[34] x ha => 1 x 5 + 1 x 2 = 7
a[35] x hb + b[35] x ha => 3 x 5 + 3 x 2 = 21
a[36] x hb + b[36] x ha => 5 x 5 + 5 x 2 = 35
a[37] x hb + b[37] x ha => 3 x 5 + 3 x 2 = 21
a[38] x hb + b[38] x ha => 2 x 5 + 2 x 2 = 14
a[39] x hb + b[39] x ha => 1 x 5 + 1 x 2 = 7
a[40] x hb + b[40] x ha => 4 x 5 + 4 x 2 = 28

```

Total time using processors [0]: [1.008339] seconds

```

worker 8 Ready
worker 8 received message: Rhotrix_A = 3 and Rhotrix_B = 3
worker 8 Returning an answer for Rhotrix_A x hb = 3 x 5 + Rhotrix_B x ha = 3 x 2 = 21
worker 8 received message: Rhotrix_A = 1 and Rhotrix_B = 1
worker 8 Returning an answer for Rhotrix_A x hb = 1 x 5 + Rhotrix_B x ha = 1 x 2 = 7
worker 8 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 8 Shutting Down

```

```

worker 2 Ready
worker 2 received message: Rhotrix_A = 3 and Rhotrix_B = 3
worker 2 Returning an answer for Rhotrix_A x hb = 3 x 5 + Rhotrix_B x ha = 3 x 2 = 21
worker 2 received message: Rhotrix_A = 5 and Rhotrix_B = 5
worker 2 Returning an answer for Rhotrix_A x hb = 5 x 5 + Rhotrix_B x ha = 5 x 2 = 35
worker 2 received message: Rhotrix_A = 1 and Rhotrix_B = 1
worker 2 Returning an answer for Rhotrix_A x hb = 1 x 5 + Rhotrix_B x ha = 1 x 2 = 7
worker 2 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 2 Shutting Down

```

```

worker 12 Ready
worker 12 received message: Rhotrix_A = 8 and Rhotrix_B = 8
worker 12 Returning an answer for Rhotrix_A x hb = 8 x 5 + Rhotrix_B x ha = 8 x 2 = 56
worker 12 received message: Rhotrix_A = 1 and Rhotrix_B = 1
worker 12 Returning an answer for Rhotrix_A x hb = 1 x 5 + Rhotrix_B x ha = 1 x 2 = 7
worker 12 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 12 Shutting Down

```

```

worker 3 Ready
worker 3 received message: Rhotrix_A = 9 and Rhotrix_B = 9
worker 3 Returning an answer for Rhotrix_A x hb = 9 x 5 + Rhotrix_B x ha = 9 x 2 = 63
worker 3 received message: Rhotrix_A = 1 and Rhotrix_B = 1
worker 3 Returning an answer for Rhotrix_A x hb = 1 x 5 + Rhotrix_B x ha = 1 x 2 = 7
worker 3 received message: Rhotrix_A = 4 and Rhotrix_B = 4
worker 3 Returning an answer for Rhotrix_A x hb = 4 x 5 + Rhotrix_B x ha = 4 x 2 = 28
worker 3 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 3 Shutting Down

```

```

worker 5 Ready
worker 5 received message: Rhotrix_A = 7 and Rhotrix_B = 7
worker 5 Returning an answer for Rhotrix_A x hb = 7 x 5 + Rhotrix_B x ha = 7 x 2 = 49

```

```

worker 5 received message: Rhotrix_A = 1 and Rhotrix_B = 1
worker 5 Returning an answer for Rhotrix_A x hb = 1 x 5 + Rhotrix_B x ha = 1 x 2 = 7
worker 5 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 5 Shutting Down

worker 11 Ready
worker 11 received message: Rhotrix_A = 3 and Rhotrix_B = 3
worker 11 Returning an answer for Rhotrix_A x hb = 3 x 5 + Rhotrix_B x ha = 3 x 2 = 21
worker 11 received message: Rhotrix_A = 4 and Rhotrix_B = 4
worker 11 Returning an answer for Rhotrix_A x hb = 4 x 5 + Rhotrix_B x ha = 4 x 2 = 28
worker 11 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 11 Shutting Down

worker 9 Ready
worker 9 received message: Rhotrix_A = 4 and Rhotrix_B = 4
worker 9 Returning an answer for Rhotrix_A x hb = 4 x 5 + Rhotrix_B x ha = 4 x 2 = 28
worker 9 received message: Rhotrix_A = 3 and Rhotrix_B = 3
worker 9 Returning an answer for Rhotrix_A x hb = 3 x 5 + Rhotrix_B x ha = 3 x 2 = 21
worker 9 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 9 Shutting Down

worker 13 Ready
worker 13 received message: Rhotrix_A = 1 and Rhotrix_B = 1
worker 13 Returning an answer for Rhotrix_A x hb = 1 x 5 + Rhotrix_B x ha = 1 x 2 = 7
worker 13 received message: Rhotrix_A = 6 and Rhotrix_B = 6
worker 13 Returning an answer for Rhotrix_A x hb = 6 x 5 + Rhotrix_B x ha = 6 x 2 = 42
worker 13 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 13 Shutting Down

worker 15 Ready
worker 15 received message: Rhotrix_A = 7 and Rhotrix_B = 7
worker 15 Returning an answer for Rhotrix_A x hb = 7 x 5 + Rhotrix_B x ha = 7 x 2 = 49
worker 15 received message: Rhotrix_A = 4 and Rhotrix_B = 4
worker 15 Returning an answer for Rhotrix_A x hb = 4 x 5 + Rhotrix_B x ha = 4 x 2 = 28
worker 15 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 15 Shutting Down

worker 18 Ready
worker 18 received message: Rhotrix_A = 3 and Rhotrix_B = 3
worker 18 Returning an answer for Rhotrix_A x hb = 3 x 5 + Rhotrix_B x ha = 3 x 2 = 21
worker 18 received message: Rhotrix_A = 5 and Rhotrix_B = 5
worker 18 Returning an answer for Rhotrix_A x hb = 5 x 5 + Rhotrix_B x ha = 5 x 2 = 35
worker 18 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 18 Shutting Down

worker 14 Ready
worker 14 received message: Rhotrix_A = 2 and Rhotrix_B = 2
worker 14 Returning an answer for Rhotrix_A x hb = 2 x 5 + Rhotrix_B x ha = 2 x 2 = 14
worker 14 received message: Rhotrix_A = 2 and Rhotrix_B = 2
worker 14 Returning an answer for Rhotrix_A x hb = 2 x 5 + Rhotrix_B x ha = 2 x 2 = 14
worker 14 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 14 Shutting Down

worker 16 Ready
worker 16 received message: Rhotrix_A = 4 and Rhotrix_B = 4
worker 16 Returning an answer for Rhotrix_A x hb = 4 x 5 + Rhotrix_B x ha = 4 x 2 = 28
worker 16 received message: Rhotrix_A = 1 and Rhotrix_B = 1
worker 16 Returning an answer for Rhotrix_A x hb = 1 x 5 + Rhotrix_B x ha = 1 x 2 = 7
worker 16 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 16 Shutting Down

worker 19 Ready
worker 19 received message: Rhotrix_A = 7 and Rhotrix_B = 7
worker 19 Returning an answer for Rhotrix_A x hb = 7 x 5 + Rhotrix_B x ha = 7 x 2 = 49
worker 19 received message: Rhotrix_A = 3 and Rhotrix_B = 3
worker 19 Returning an answer for Rhotrix_A x hb = 3 x 5 + Rhotrix_B x ha = 3 x 2 = 21
worker 19 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 19 Shutting Down

worker 1 Ready
worker 1 received message: Rhotrix_A = 6 and Rhotrix_B = 6
worker 1 Returning an answer for Rhotrix_A x hb = 6 x 5 + Rhotrix_B x ha = 6 x 2 = 42
worker 1 received message: Rhotrix_A = 4 and Rhotrix_B = 4
worker 1 Returning an answer for Rhotrix_A x hb = 4 x 5 + Rhotrix_B x ha = 4 x 2 = 28
worker 1 received message: Rhotrix_A = 2 and Rhotrix_B = 2
worker 1 Returning an answer for Rhotrix_A x hb = 2 x 5 + Rhotrix_B x ha = 2 x 2 = 14
worker 1 received message: Rhotrix_A = -1 and Rhotrix_B = -1
worker 1 Shutting Down

worker 10 Ready

```

worker 10 received message: Rhotrix_A = 5 and Rhotrix_B = 5
 worker 10 Returning an answer for Rhotrix_A x hb = 5 x 5 + Rhotrix_B x ha = 5 x 2 = 35
 worker 10 received message: Rhotrix_A = 1 and Rhotrix_B = 1
 worker 10 Returning an answer for Rhotrix_A x hb = 1 x 5 + Rhotrix_B x ha = 1 x 2 = 7
 worker 10 received message: Rhotrix_A = -1 and Rhotrix_B = -1
 worker 10 Shutting Down

Worker 4 Ready
 worker 4 received message: Rhotrix_A = 4 and Rhotrix_B = 4
 worker 4 Returning an answer for Rhotrix_A x hb = 4 x 5 + Rhotrix_B x ha = 4 x 2 = 28
 worker 4 received message: Rhotrix_A = 6 and Rhotrix_B = 6
 worker 4 Returning an answer for Rhotrix_A x hb = 6 x 5 + Rhotrix_B x ha = 6 x 2 = 42
 worker 4 received message: Rhotrix_A = -1 and Rhotrix_B = -1
 worker 4 Shutting Down

worker 6 Ready
 worker 6 received message: Rhotrix_A = 2 and Rhotrix_B = 2
 worker 6 Returning an answer for Rhotrix_A x hb = 2 x 5 + Rhotrix_B x ha = 2 x 2 = 14
 worker 6 received message: Rhotrix_A = 7 and Rhotrix_B = 7
 worker 6 Returning an answer for Rhotrix_A x hb = 7 x 5 + Rhotrix_B x ha = 7 x 2 = 49
 worker 6 received message: Rhotrix_A = -1 and Rhotrix_B = -1
 worker 6 Shutting Down

worker 17 Ready
 worker 17 received message: Rhotrix_A = 1 and Rhotrix_B = 1
 worker 17 Returning an answer for Rhotrix_A x hb = 1 x 5 + Rhotrix_B x ha = 1 x 2 = 7
 worker 17 received message: Rhotrix_A = 3 and Rhotrix_B = 3
 worker 17 Returning an answer for Rhotrix_A x hb = 3 x 5 + Rhotrix_B x ha = 3 x 2 = 21
 worker 17 received message: Rhotrix_A = -1 and Rhotrix_B = -1
 worker 17 Shutting Down

worker 7 Ready
 worker 7 received message: Rhotrix_A = 7 and Rhotrix_B = 7
 worker 7 Returning an answer for Rhotrix_A x hb = 7 x 5 + Rhotrix_B x ha = 7 x 2 = 49
 worker 7 received message: Rhotrix_A = 2 and Rhotrix_B = 2
 worker 7 Returning an answer for Rhotrix_A x hb = 2 x 5 + Rhotrix_B x ha = 2 x 2 = 14
 worker 7 received message: Rhotrix_A = -1 and Rhotrix_B = -1
 worker 7 Shutting Down

B.2.5 Sequential Row-column Rhotrix Multiplication [R₂₁(A) x R₂₁(B)]

EZUGWU, El-Shamir Absalom MSC/SCIE/00043/2008-2009

Date : Nov 28 2010

Time : 11:00:56

=====

The First Rhotrix R(A)[21,11] Is:

R(A)[1, 1] =	2	R(A)[1, 2] =	3	R(A)[1, 3] =	5	R(A)[1, 4] =	6
R(A)[1, 5] =	3	R(A)[1, 6] =	6	R(A)[1, 7] =	7	R(A)[1, 8] =	5
R(A)[1, 9] =	3	R(A)[1, 10] =	6	R(A)[1, 11] =	2		
R(A)[2, 1] =	1	R(A)[2, 2] =	4	R(A)[2, 3] =	9	R(A)[2, 4] =	7
R(A)[2, 5] =	3	R(A)[2, 6] =	5	R(A)[2, 7] =	4	R(A)[2, 8] =	2
R(A)[2, 9] =	6	R(A)[2, 10] =	5	R(A)[2, 11] =	0		
R(A)[3, 1] =	2	R(A)[3, 2] =	3	R(A)[3, 3] =	3	R(A)[3, 4] =	5
R(A)[3, 5] =	4	R(A)[3, 6] =	6	R(A)[3, 7] =	5	R(A)[3, 8] =	0
R(A)[3, 9] =	5	R(A)[3, 10] =	6	R(A)[3, 11] =	3		
R(A)[4, 1] =	2	R(A)[4, 2] =	6	R(A)[4, 3] =	2	R(A)[4, 4] =	2
R(A)[4, 5] =	7	R(A)[4, 6] =	1	R(A)[4, 7] =	5	R(A)[4, 8] =	2
R(A)[4, 9] =	7	R(A)[4, 10] =	6	R(A)[4, 11] =	0		
R(A)[5, 1] =	1	R(A)[5, 2] =	5	R(A)[5, 3] =	3	R(A)[5, 4] =	7
R(A)[5, 5] =	3	R(A)[5, 6] =	1	R(A)[5, 7] =	4	R(A)[5, 8] =	4
R(A)[5, 9] =	7	R(A)[5, 10] =	5	R(A)[5, 11] =	4		
R(A)[6, 1] =	1	R(A)[6, 2] =	4	R(A)[6, 3] =	5	R(A)[6, 4] =	5
R(A)[6, 5] =	2	R(A)[6, 6] =	0	R(A)[6, 7] =	3	R(A)[6, 8] =	3
R(A)[6, 9] =	2	R(A)[6, 10] =	3	R(A)[6, 11] =	0		

R(A)[7, 1] =	3	R(A)[7, 2] =	2	R(A)[7, 3] =	6	R(A)[7, 4] =	4
R(A)[7, 5] =	3	R(A)[7, 6] =	2	R(A)[7, 7] =	7	R(A)[7, 8] =	6
R(A)[7, 9] =	4	R(A)[7, 10] =	2	R(A)[7, 11] =	5		
R(A)[8, 1] =	0	R(A)[8, 2] =	4	R(A)[8, 3] =	2	R(A)[8, 4] =	2
R(A)[8, 5] =	2	R(A)[8, 6] =	5	R(A)[8, 7] =	5	R(A)[8, 8] =	3
R(A)[8, 9] =	6	R(A)[8, 10] =	4	R(A)[8, 11] =	0		
R(A)[9, 1] =	2	R(A)[9, 2] =	3	R(A)[9, 3] =	2	R(A)[9, 4] =	4
R(A)[9, 5] =	2	R(A)[9, 6] =	6	R(A)[9, 7] =	1	R(A)[9, 8] =	7
R(A)[9, 9] =	0	R(A)[9, 10] =	5	R(A)[9, 11] =	7		
R(A)[10, 1] =	2	R(A)[10, 2] =	3	R(A)[10, 3] =	5	R(A)[10, 4] =	3
R(A)[10, 5] =	7	R(A)[10, 6] =	3	R(A)[10, 7] =	5	R(A)[10, 8] =	6
R(A)[10, 9] =	1	R(A)[10, 10] =	4	R(A)[10, 11] =	0		
R(A)[11, 1] =	9	R(A)[11, 2] =	4	R(A)[11, 3] =	2	R(A)[11, 4] =	7
R(A)[11, 5] =	5	R(A)[11, 6] =	4	R(A)[11, 7] =	3	R(A)[11, 8] =	2
R(A)[11, 9] =	8	R(A)[11, 10] =	1	R(A)[11, 11] =	2		
R(A)[12, 1] =	8	R(A)[12, 2] =	6	R(A)[12, 3] =	1	R(A)[12, 4] =	4
R(A)[12, 5] =	3	R(A)[12, 6] =	0	R(A)[12, 7] =	3	R(A)[12, 8] =	7
R(A)[12, 9] =	6	R(A)[12, 10] =	2	R(A)[12, 11] =	0		
R(A)[13, 1] =	7	R(A)[13, 2] =	1	R(A)[13, 3] =	4	R(A)[13, 4] =	3
R(A)[13, 5] =	2	R(A)[13, 6] =	6	R(A)[13, 7] =	4	R(A)[13, 8] =	3
R(A)[13, 9] =	9	R(A)[13, 10] =	8	R(A)[13, 11] =	3		
R(A)[14, 1] =	2	R(A)[14, 2] =	1	R(A)[14, 3] =	8	R(A)[14, 4] =	6
R(A)[14, 5] =	3	R(A)[14, 6] =	0	R(A)[14, 7] =	9	R(A)[14, 8] =	7
R(A)[14, 9] =	5	R(A)[14, 10] =	2	R(A)[14, 11] =	0		
R(A)[15, 1] =	0	R(A)[15, 2] =	9	R(A)[15, 3] =	1	R(A)[15, 4] =	2
R(A)[15, 5] =	6	R(A)[15, 6] =	2	R(A)[15, 7] =	9	R(A)[15, 8] =	1
R(A)[15, 9] =	4	R(A)[15, 10] =	3	R(A)[15, 11] =	1		
R(A)[16, 1] =	5	R(A)[16, 2] =	6	R(A)[16, 3] =	4	R(A)[16, 4] =	3
R(A)[16, 5] =	0	R(A)[16, 6] =	8	R(A)[16, 7] =	4	R(A)[16, 8] =	2
R(A)[16, 9] =	7	R(A)[16, 10] =	9	R(A)[16, 11] =	0		
R(A)[17, 1] =	6	R(A)[17, 2] =	8	R(A)[17, 3] =	4	R(A)[17, 4] =	3
R(A)[17, 5] =	9	R(A)[17, 6] =	8	R(A)[17, 7] =	3	R(A)[17, 8] =	2
R(A)[17, 9] =	5	R(A)[17, 10] =	7	R(A)[17, 11] =	9		
R(A)[18, 1] =	4	R(A)[18, 2] =	8	R(A)[18, 3] =	2	R(A)[18, 4] =	4
R(A)[18, 5] =	2	R(A)[18, 6] =	1	R(A)[18, 7] =	1	R(A)[18, 8] =	6
R(A)[18, 9] =	0	R(A)[18, 10] =	9	R(A)[18, 11] =	0		
R(A)[19, 1] =	6	R(A)[19, 2] =	5	R(A)[19, 3] =	3	R(A)[19, 4] =	9
R(A)[19, 5] =	1	R(A)[19, 6] =	2	R(A)[19, 7] =	5	R(A)[19, 8] =	4
R(A)[19, 9] =	3	R(A)[19, 10] =	7	R(A)[19, 11] =	7		
R(A)[20, 1] =	9	R(A)[20, 2] =	8	R(A)[20, 3] =	3	R(A)[20, 4] =	2
R(A)[20, 5] =	7	R(A)[20, 6] =	3	R(A)[20, 7] =	9	R(A)[20, 8] =	3
R(A)[20, 9] =	2	R(A)[20, 10] =	5	R(A)[20, 11] =	0		
R(A)[21, 1] =	1	R(A)[21, 2] =	2	R(A)[21, 3] =	6	R(A)[21, 4] =	5
R(A)[21, 5] =	4	R(A)[21, 6] =	3	R(A)[21, 7] =	2	R(A)[21, 8] =	9
R(A)[21, 9] =	6	R(A)[21, 10] =	7	R(A)[21, 11] =	1		

The Second Rhotrix R(B)[21,11]Is:

R(B)[1, 1] =	7	R(B)[1, 2] =	3	R(B)[1, 3] =	2	R(B)[1, 4] =	6
R(B)[1, 5] =	3	R(B)[1, 6] =	6	R(B)[1, 7] =	7	R(B)[1, 8] =	1
R(B)[1, 9] =	3	R(B)[1, 10] =	6	R(B)[1, 11] =	1		
R(B)[2, 1] =	4	R(B)[2, 2] =	4	R(B)[2, 3] =	0	R(B)[2, 4] =	7
R(B)[2, 5] =	3	R(B)[2, 6] =	5	R(B)[2, 7] =	4	R(B)[2, 8] =	3
R(B)[2, 9] =	6	R(B)[2, 10] =	5	R(B)[2, 11] =	0		

R(B)[3, 1] =	2	R(B)[3, 2] =	3	R(B)[3, 3] =	0	R(B)[3, 4] =	5
R(B)[3, 5] =	4	R(B)[3, 6] =	6	R(B)[3, 7] =	5	R(B)[3, 8] =	6
R(B)[3, 9] =	5	R(B)[3, 10] =	6	R(B)[3, 11] =	2		
R(B)[4, 1] =	5	R(B)[4, 2] =	6	R(B)[4, 3] =	1	R(B)[4, 4] =	2
R(B)[4, 5] =	7	R(B)[4, 6] =	1	R(B)[4, 7] =	5	R(B)[4, 8] =	4
R(B)[4, 9] =	7	R(B)[4, 10] =	6	R(B)[4, 11] =	0		
R(B)[5, 1] =	7	R(B)[5, 2] =	5	R(B)[5, 3] =	2	R(B)[5, 4] =	7
R(B)[5, 5] =	3	R(B)[5, 6] =	1	R(B)[5, 7] =	4	R(B)[5, 8] =	9
R(B)[5, 9] =	7	R(B)[5, 10] =	5	R(B)[5, 11] =	3		
R(B)[6, 1] =	2	R(B)[6, 2] =	4	R(B)[6, 3] =	4	R(B)[6, 4] =	5
R(B)[6, 5] =	2	R(B)[6, 6] =	0	R(B)[6, 7] =	3	R(B)[6, 8] =	9
R(B)[6, 9] =	2	R(B)[6, 10] =	3	R(B)[6, 11] =	0		
R(B)[7, 1] =	8	R(B)[7, 2] =	2	R(B)[7, 3] =	2	R(B)[7, 4] =	4
R(B)[7, 5] =	3	R(B)[7, 6] =	2	R(B)[7, 7] =	7	R(B)[7, 8] =	9
R(B)[7, 9] =	4	R(B)[7, 10] =	2	R(B)[7, 11] =	5		
R(B)[8, 1] =	9	R(B)[8, 2] =	4	R(B)[8, 3] =	3	R(B)[8, 4] =	2
R(B)[8, 5] =	2	R(B)[8, 6] =	5	R(B)[8, 7] =	5	R(B)[8, 8] =	5
R(B)[8, 9] =	6	R(B)[8, 10] =	4	R(B)[8, 11] =	0		
R(B)[9, 1] =	5	R(B)[9, 2] =	3	R(B)[9, 3] =	9	R(B)[9, 4] =	4
R(B)[9, 5] =	2	R(B)[9, 6] =	6	R(B)[9, 7] =	1	R(B)[9, 8] =	7
R(B)[9, 9] =	0	R(B)[9, 10] =	5	R(B)[9, 11] =	7		
R(B)[10, 1] =	3	R(B)[10, 2] =	3	R(B)[10, 3] =	8	R(B)[10, 4] =	3
R(B)[10, 5] =	7	R(B)[10, 6] =	3	R(B)[10, 7] =	5	R(B)[10, 8] =	4
R(B)[10, 9] =	1	R(B)[10, 10] =	4	R(B)[10, 11] =	0		
R(B)[11, 1] =	7	R(B)[11, 2] =	4	R(B)[11, 3] =	3	R(B)[11, 4] =	7
R(B)[11, 5] =	5	R(B)[11, 6] =	4	R(B)[11, 7] =	3	R(B)[11, 8] =	3
R(B)[11, 9] =	8	R(B)[11, 10] =	1	R(B)[11, 11] =	5		
R(B)[12, 1] =	3	R(B)[12, 2] =	6	R(B)[12, 3] =	7	R(B)[12, 4] =	4
R(B)[12, 5] =	3	R(B)[12, 6] =	0	R(B)[12, 7] =	3	R(B)[12, 8] =	7
R(B)[12, 9] =	6	R(B)[12, 10] =	2	R(B)[12, 11] =	0		
R(B)[13, 1] =	6	R(B)[13, 2] =	1	R(B)[13, 3] =	0	R(B)[13, 4] =	3
R(B)[13, 5] =	2	R(B)[13, 6] =	6	R(B)[13, 7] =	4	R(B)[13, 8] =	2
R(B)[13, 9] =	9	R(B)[13, 10] =	8	R(B)[13, 11] =	1		
R(B)[14, 1] =	8	R(B)[14, 2] =	1	R(B)[14, 3] =	1	R(B)[14, 4] =	6
R(B)[14, 5] =	3	R(B)[14, 6] =	0	R(B)[14, 7] =	9	R(B)[14, 8] =	8
R(B)[14, 9] =	5	R(B)[14, 10] =	2	R(B)[14, 11] =	0		
R(B)[15, 1] =	3	R(B)[15, 2] =	9	R(B)[15, 3] =	1	R(B)[15, 4] =	2
R(B)[15, 5] =	6	R(B)[15, 6] =	2	R(B)[15, 7] =	9	R(B)[15, 8] =	4
R(B)[15, 9] =	4	R(B)[15, 10] =	3	R(B)[15, 11] =	1		
R(B)[16, 1] =	5	R(B)[16, 2] =	6	R(B)[16, 3] =	1	R(B)[16, 4] =	3
R(B)[16, 5] =	0	R(B)[16, 6] =	8	R(B)[16, 7] =	4	R(B)[16, 8] =	2
R(B)[16, 9] =	7	R(B)[16, 10] =	9	R(B)[16, 11] =	0		
R(B)[17, 1] =	8	R(B)[17, 2] =	8	R(B)[17, 3] =	3	R(B)[17, 4] =	3
R(B)[17, 5] =	9	R(B)[17, 6] =	8	R(B)[17, 7] =	3	R(B)[17, 8] =	1
R(B)[17, 9] =	5	R(B)[17, 10] =	7	R(B)[17, 11] =	1		
R(B)[18, 1] =	1	R(B)[18, 2] =	8	R(B)[18, 3] =	5	R(B)[18, 4] =	4
R(B)[18, 5] =	2	R(B)[18, 6] =	1	R(B)[18, 7] =	1	R(B)[18, 8] =	4
R(B)[18, 9] =	0	R(B)[18, 10] =	9	R(B)[18, 11] =	0		
R(B)[19, 1] =	6	R(B)[19, 2] =	5	R(B)[19, 3] =	4	R(B)[19, 4] =	9
R(B)[19, 5] =	1	R(B)[19, 6] =	2	R(B)[19, 7] =	5	R(B)[19, 8] =	1
R(B)[19, 9] =	3	R(B)[19, 10] =	7	R(B)[19, 11] =	1		

R(B)[20, 1] =	2	R(B)[20, 2] =	8	R(B)[20, 3] =	2	R(B)[20, 4] =	2
R(B)[20, 5] =	7	R(B)[20, 6] =	3	R(B)[20, 7] =	9	R(B)[20, 8] =	2
R(B)[20, 9] =	2	R(B)[20, 10] =	5	R(B)[20, 11] =	0		
R(B)[21, 1] =	1	R(B)[21, 2] =	2	R(B)[21, 3] =	1	R(B)[21, 4] =	5
R(B)[21, 5] =	4	R(B)[21, 6] =	3	R(B)[21, 7] =	2	R(B)[21, 8] =	1
R(B)[21, 9] =	6	R(B)[21, 10] =	7	R(B)[21, 11] =	1		

Multiplication Of The Above Two Rhotrices Are:

R(C)[1, 1] =	279	R(C)[1, 2] =	195	R(C)[1, 3] =	111	R(C)[1, 4] =	244
R(C)[1, 5] =	172	R(C)[1, 6] =	183	R(C)[1, 7] =	228	R(C)[1, 8] =	203
R(C)[1, 9] =	256	R(C)[1, 10] =	236	R(C)[1, 11] =	127		
R(C)[2, 1] =	187	R(C)[2, 2] =	235	R(C)[2, 3] =	166	R(C)[2, 4] =	167
R(C)[2, 5] =	158	R(C)[2, 6] =	90	R(C)[2, 7] =	211	R(C)[2, 8] =	252
R(C)[2, 9] =	171	R(C)[2, 10] =	211	R(C)[2, 11] =	0		
R(C)[3, 1] =	252	R(C)[3, 2] =	157	R(C)[3, 3] =	116	R(C)[3, 4] =	225
R(C)[3, 5] =	153	R(C)[3, 6] =	182	R(C)[3, 7] =	169	R(C)[3, 8] =	162
R(C)[3, 9] =	216	R(C)[3, 10] =	219	R(C)[3, 11] =	119		
R(C)[4, 1] =	153	R(C)[4, 2] =	208	R(C)[4, 3] =	137	R(C)[4, 4] =	141
R(C)[4, 5] =	179	R(C)[4, 6] =	88	R(C)[4, 7] =	206	R(C)[4, 8] =	177
R(C)[4, 9] =	134	R(C)[4, 10] =	211	R(C)[4, 11] =	0		
R(C)[5, 1] =	242	R(C)[5, 2] =	189	R(C)[5, 3] =	101	R(C)[5, 4] =	205
R(C)[5, 5] =	180	R(C)[5, 6] =	185	R(C)[5, 7] =	205	R(C)[5, 8] =	185
R(C)[5, 9] =	211	R(C)[5, 10] =	237	R(C)[5, 11] =	105		
R(C)[6, 1] =	132	R(C)[6, 2] =	135	R(C)[6, 3] =	77	R(C)[6, 4] =	97
R(C)[6, 5] =	99	R(C)[6, 6] =	75	R(C)[6, 7] =	142	R(C)[6, 8] =	141
R(C)[6, 9] =	118	R(C)[6, 10] =	138	R(C)[6, 11] =	0		
R(C)[7, 1] =	237	R(C)[7, 2] =	183	R(C)[7, 3] =	90	R(C)[7, 4] =	200
R(C)[7, 5] =	171	R(C)[7, 6] =	175	R(C)[7, 7] =	206	R(C)[7, 8] =	181
R(C)[7, 9] =	236	R(C)[7, 10] =	236	R(C)[7, 11] =	100		
R(C)[8, 1] =	132	R(C)[8, 2] =	179	R(C)[8, 3] =	115	R(C)[8, 4] =	119
R(C)[8, 5] =	120	R(C)[8, 6] =	62	R(C)[8, 7] =	160	R(C)[8, 8] =	165
R(C)[8, 9] =	130	R(C)[8, 10] =	167	R(C)[8, 11] =	0		
R(C)[9, 1] =	182	R(C)[9, 2] =	166	R(C)[9, 3] =	86	R(C)[9, 4] =	204
R(C)[9, 5] =	147	R(C)[9, 6] =	127	R(C)[9, 7] =	191	R(C)[9, 8] =	148
R(C)[9, 9] =	193	R(C)[9, 10] =	177	R(C)[9, 11] =	98		
R(C)[10, 1] =	169	R(C)[10, 2] =	178	R(C)[10, 3] =	133	R(C)[10, 4] =	144
R(C)[10, 5] =	146	R(C)[10, 6] =	110	R(C)[10, 7] =	203	R(C)[10, 8] =	191
R(C)[10, 9] =	161	R(C)[10, 10] =	182	R(C)[10, 11] =	0		
R(C)[11, 1] =	290	R(C)[11, 2] =	188	R(C)[11, 3] =	125	R(C)[11, 4] =	220
R(C)[11, 5] =	199	R(C)[11, 6] =	234	R(C)[11, 7] =	220	R(C)[11, 8] =	186
R(C)[11, 9] =	211	R(C)[11, 10] =	238	R(C)[11, 11] =	129		
R(C)[12, 1] =	178	R(C)[12, 2] =	206	R(C)[12, 3] =	90	R(C)[12, 4] =	157
R(C)[12, 5] =	132	R(C)[12, 6] =	143	R(C)[12, 7] =	179	R(C)[12, 8] =	155
R(C)[12, 9] =	187	R(C)[12, 10] =	240	R(C)[12, 11] =	0		
R(C)[13, 1] =	311	R(C)[13, 2] =	229	R(C)[13, 3] =	129	R(C)[13, 4] =	269
R(C)[13, 5] =	207	R(C)[13, 6] =	221	R(C)[13, 7] =	227	R(C)[13, 8] =	148
R(C)[13, 9] =	249	R(C)[13, 10] =	271	R(C)[13, 11] =	107		
R(C)[14, 1] =	208	R(C)[14, 2] =	186	R(C)[14, 3] =	120	R(C)[14, 4] =	176
R(C)[14, 5] =	113	R(C)[14, 6] =	117	R(C)[14, 7] =	214	R(C)[14, 8] =	234
R(C)[14, 9] =	172	R(C)[14, 10] =	212	R(C)[14, 11] =	0		

R(C)[15, 1] =	193	R(C)[15, 2] =	129	R(C)[15, 3] =	92	R(C)[15, 4] =	171
R(C)[15, 5] =	134	R(C)[15, 6] =	200	R(C)[15, 7] =	149	R(C)[15, 8] =	159
R(C)[15, 9] =	196	R(C)[15, 10] =	226	R(C)[15, 11] =	101		
R(C)[16, 1] =	176	R(C)[16, 2] =	276	R(C)[16, 3] =	146	R(C)[16, 4] =	181
R(C)[16, 5] =	184	R(C)[16, 6] =	96	R(C)[16, 7] =	233	R(C)[16, 8] =	228
R(C)[16, 9] =	198	R(C)[16, 10] =	235	R(C)[16, 11] =	0		
R(C)[17, 1] =	326	R(C)[17, 2] =	241	R(C)[17, 3] =	185	R(C)[17, 4] =	344
R(C)[17, 5] =	235	R(C)[17, 6] =	283	R(C)[17, 7] =	250	R(C)[17, 8] =	239
R(C)[17, 9] =	297	R(C)[17, 10] =	340	R(C)[17, 11] =	178		
R(C)[18, 1] =	161	R(C)[18, 2] =	209	R(C)[18, 3] =	76	R(C)[18, 4] =	114
R(C)[18, 5] =	163	R(C)[18, 6] =	129	R(C)[18, 7] =	209	R(C)[18, 8] =	135
R(C)[18, 9] =	181	R(C)[18, 10] =	201	R(C)[18, 11] =	0		
R(C)[19, 1] =	279	R(C)[19, 2] =	191	R(C)[19, 3] =	99	R(C)[19, 4] =	266
R(C)[19, 5] =	182	R(C)[19, 6] =	198	R(C)[19, 7] =	263	R(C)[19, 8] =	200
R(C)[19, 9] =	255	R(C)[19, 10] =	277	R(C)[19, 11] =	113		
R(C)[20, 1] =	229	R(C)[20, 2] =	226	R(C)[20, 3] =	135	R(C)[20, 4] =	212
R(C)[20, 5] =	217	R(C)[20, 6] =	125	R(C)[20, 7] =	279	R(C)[20, 8] =	241
R(C)[20, 9] =	229	R(C)[20, 10] =	232	R(C)[20, 11] =	0		
R(C)[21, 1] =	264	R(C)[21, 2] =	241	R(C)[21, 3] =	125	R(C)[21, 4] =	225
R(C)[21, 5] =	190	R(C)[21, 6] =	165	R(C)[21, 7] =	233	R(C)[21, 8] =	203
R(C)[21, 9] =	210	R(C)[21, 10] =	222	R(C)[21, 11] =	116		