Parallel Algorithm for Solution of Three Dimensional Hyperbolic Partial Differential Equation

A Dissertation submitted to Jawaharlal Nehru University
in partial fulfillment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY
in
COMPUTER SCIENCE & TECHNOLOGY

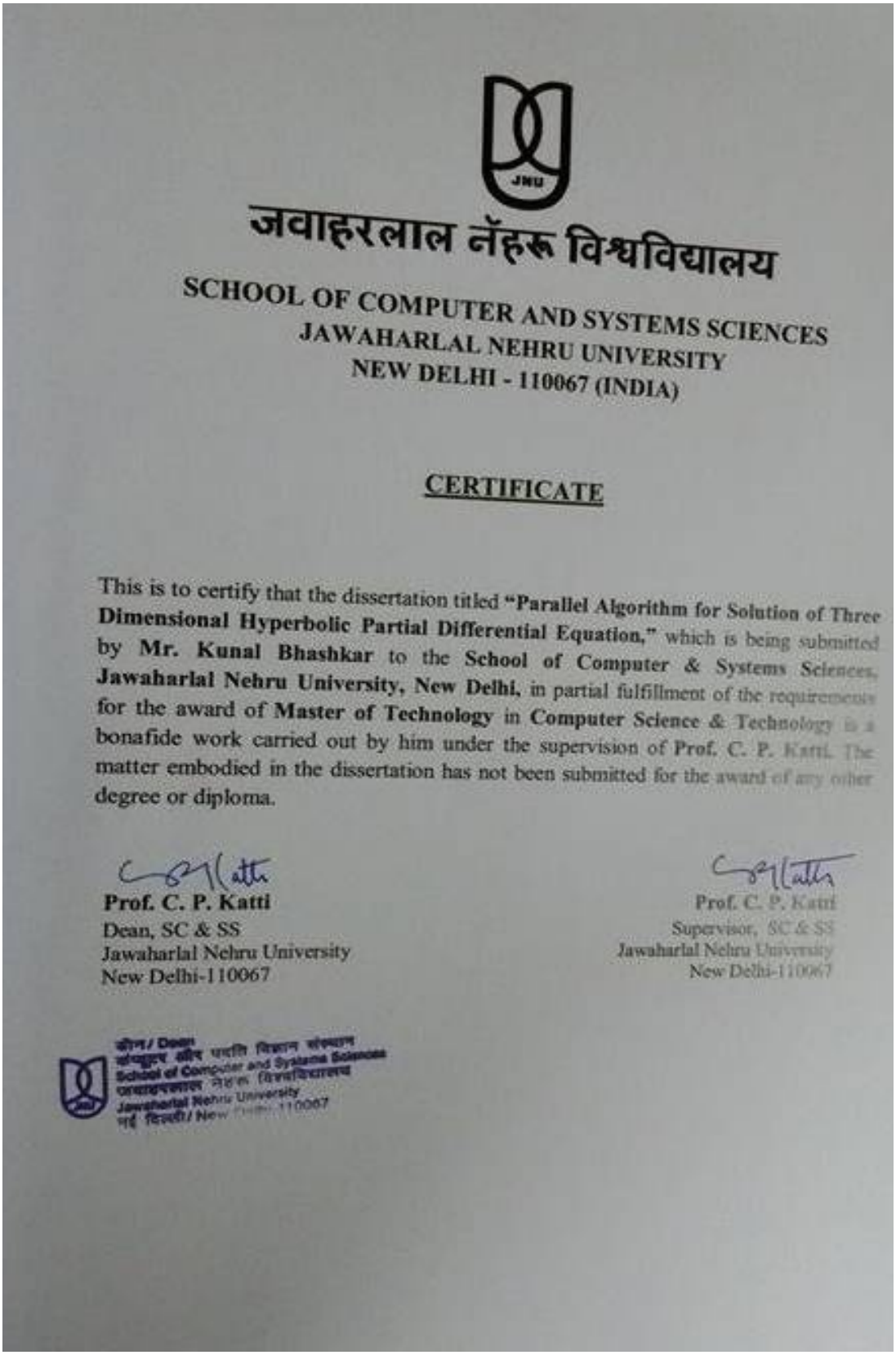
Submitted
by

KUNAL BHASHKAR

under the guidance of

Prof. C. P. KATTI

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI - 110067 (INDIA)
JULY 2015



जवाहरलाल नेहरू विश्वविद्यालय

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI - 110067 (INDIA)

CERTIFICATE

This is to certify that the dissertation titled **"Parallel Algorithm for Solution of Three Dimensional Hyperbolic Partial Differential Equation,"** which is being submitted by **Mr. Kunal Bhashkar** to the **School of Computer & Systems Sciences, Jawaharlal Nehru University, New Delhi,** in partial fulfillment of the requirements for the award of **Master of Technology** in **Computer Science & Technology** is a bonafide work carried out by him under the supervision of **Prof. C. P. Katti.** The matter embodied in the dissertation has not been submitted for the award of any other degree or diploma.

Prof. C. P. Katti
Dean, SC & SS
Jawaharlal Nehru University
New Delhi-110067

Prof. C. P. Katti
Supervisor, SC & SS
Jawaharlal Nehru University
New Delhi-110067

डीन / Dean
कंप्यूटर और पद्धति विज्ञान संस्थान
School of Computer and Systems Sciences
जवाहरलाल नेहरू विश्वविद्यालय
Jawaharlal Nehru University
नई दिल्ली / New Delhi-110067

जवाहरलाल नेहरू विश्वविद्यालय

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI - 110067 (INDIA)

DECLARATION

This is to certify that the dissertation titled **"Parallel Algorithm for Solution of Three Dimensional Hyperbolic Partial Differential Equation,"** which is being submitted to the **School of Computer & Systems Sciences, Jawaharlal Nehru University, New Delhi,** in partial fulfillment of the requirements for the award of **Master of Technology** in **Computer Science & Technology** is a bonafide work carried out by me.

The matter embodied in the dissertation has not been submitted for the award of any other degree or diploma.

Kunal Bhashkar
M.Tech (2013 - 2015)
School of Computer & Systems Sciences,
Jawaharlal Nehru University,
New Delhi - 110067.

Dedicated to my parents

ACKNOWLEDGEMENT

I wish to express my sincere appreciation to my honourable supervisor **Prof. C.P Katti,** School of Computer & Systems Sciences, Jawaharlal Nehru University, New Delhi for his valuable guidance, enthusiasm and encouragement. He has been inspiration to me and a great teacher. It was really unforgettable experience to work under his guidance. I would like to express gratefulness towards him for developing me independent problem solving approach. I would like to appreciate his interest in my queries and their practical suggestions. It would have been really impossible to complete this dissertation without his invaluable support and patience.

I express my heartfelt indebtedness to my friends and appreciate their help in all manners they could. I wish to express my admiration to all of them for their love and being with me through thick and thin.

At last, I would like to thank to my friends who formed part of my vision and taught me the good things that really matters in life. I would like to share this moment of happiness with them.

Kunal Bhashkar

ABSTRACT

In this thesis, we discussed the solution of three dimensional partial differential equations (PDEs) using *Finite Difference Method*. The selected three-dimensional PDE to solve in this thesis are of *Hyperbolic type*. Parallel Virtual Machine (PVM) is used in support of the communication among all microprocessors of Parallel Computing System. PVM is well known as a software system that enables a collection of heterogeneous computers to be used as coherent and flexible concurrent computational resource. In *Finite Difference Method* we get a Block Tridiagonal Matrix (BTM) in middle step. The Block Tridiagonal systems of linear equations use in variety of scientific and engineering applications. My Proposed Recursive doubling algorithm (RDA) is a well-known prefix computation based numerical algorithm that requires complexity less than $O(M^3(N/P + \log P))$ work to compute the solution of a Block Tridiagonal system with N block rows and block size M on P processors. Here, we show that a Proposed RDA is sub-optimal when computing solutions of Block Tridiagonal systems with multiple right hand sides and present a novel algorithm, called the *accelerated recursive doubling algorithm*. To the best of our knowledge, this algorithm has not been reported before in the literature. The parallel implementation of a new explicit group iterative scheme is proposed for the solution of a three dimensional second order partial differential equation. The Proposed explicit group method is derived from the standard centred seven-point finite difference discretisation formula. We utilize the new domain decomposition technique on this group scheme to divide the tasks involved in solving the equation. The aim of this study is to describe the development of the parallel group iterative scheme under Open MP programming environment as a way to reduce the computational costs of the solution processes using multiple core technologies. Numerical experiments are conducted together with their detailed performance analysis. The results will be discussed in tabular format.

TABLE OF CONTENT

LIST OF TABLE

LIST OF FIGURE

Chapter 1

Introduction of Parallelism

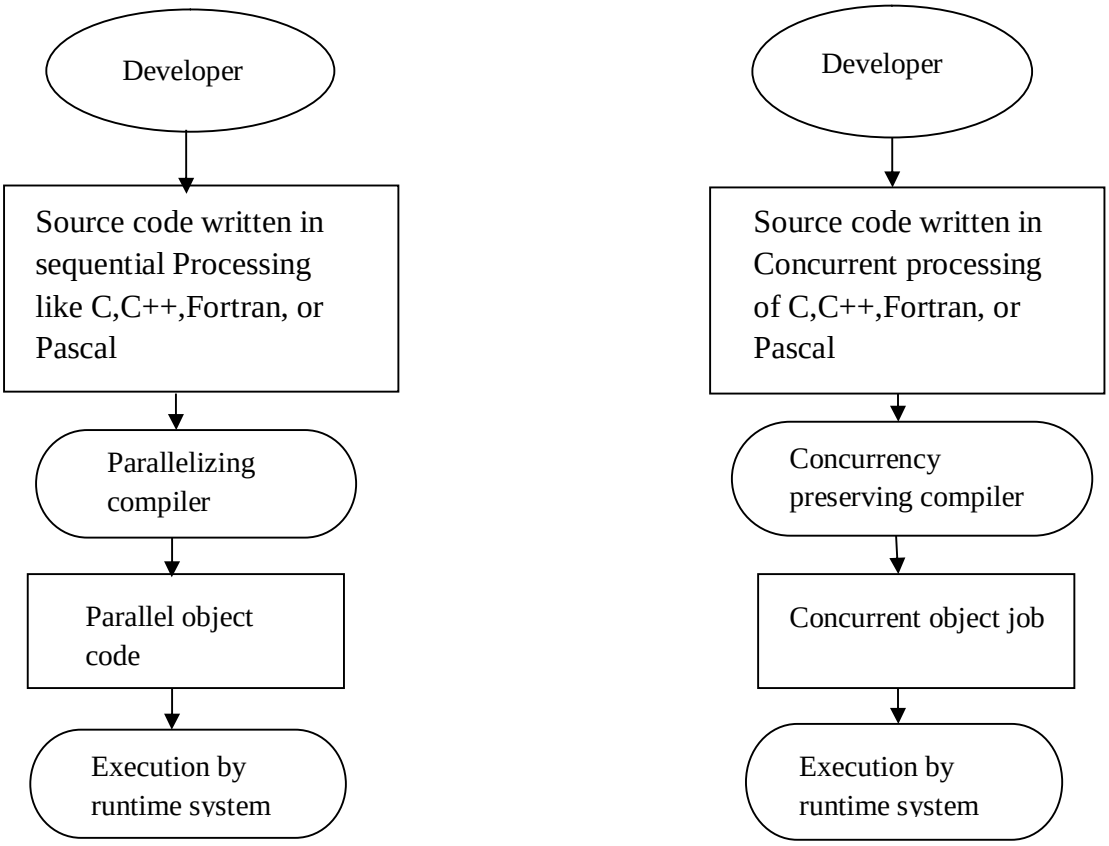
1.1 INTRODUCTION

For using a parallel computers, we require a parallel environment where parallelism is automatically oppressed. The operating system (OS) must also be extended to support parallel processing. The OS must have a competency to manage resources behind parallelism. Parallelism is divided into two categories.

- **Implicit Parallelism** In implicit parallelism, we approach a conventional language, such as C, C++, FORTRAN or Pascal to write the source program. If a source program is coded in a sequential manner is translated into parallel object code by parallelizing compiler. As given below in Fig. 1.1a, this compiler must be able to detect parallelism and allocate target machine resources.

  For implicit parallelism, success depends heavily on intelligence of parallelizing compiler. This parallelizing technique requires less effort on the part of programmer.

- **Explicit Parallelism** It is a second approach requires more effort to the Developer to develop a source programme using C, C++, FORTRAN, or Pascal like parallel languages. In the user programs parallelism is specified in explicit way.



(i) Implicit parallelism (ii) Explicit parallelism

Fig 1.1 Flow Chart

1.2 PARALLEL COMPUTATION

- **Definition:** The Parallel Computing is simultaneous execution of the same task on multiple processors in order to obtain faster results.

  i.e Processing of multiple tasks simultaneously on multiprocessor system using divide and conquer technique is called Parallel Computing.

1.3 WHY PARALLEL COMPUTING?

In general, software has been written for *serial* computation, to be run on a single computer having a single Central Processing Unit (CPU), a problem is broken into discrete series of instructions. Instructions are broken into one after another, only one instruction execute at any moment of time.

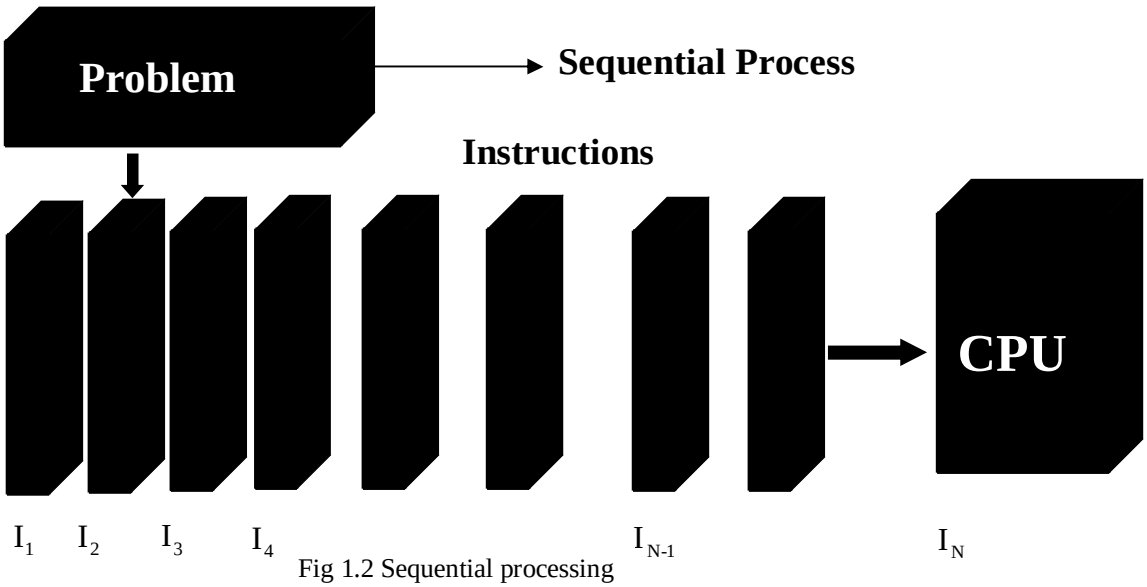


Fig 1.2 Sequential processing

In the simplest sense, *parallel computing* is the simultaneous use of multiple compute resources to solve a computational problem. It is to be run on multiple CPUs and problem is broken into discrete parts that can be solved concurrently. After that, each part is further broken down in the series of instructions and instructions from each part are further execute simultaneously on different CPUs.

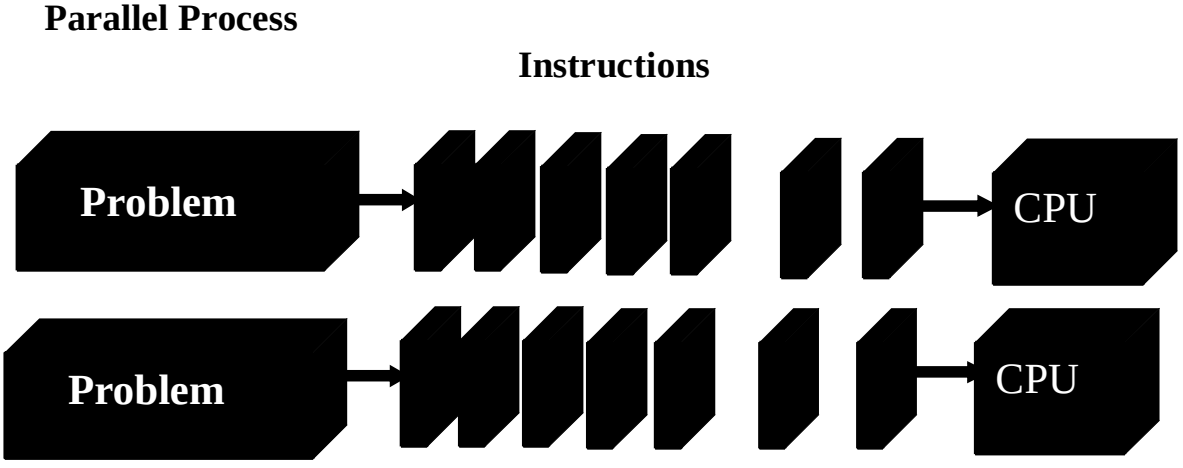


Figure 1.3 Parallel Processing

1.4 APPLICATION OF PARALLEL COMPUTING

- Finite element analysis of structural design involves a large system of algebraic equations which have to be solved. Such computations these days demand supercomputing power.
- Genetic engineers demand fast computers for studying molecular biology, artificial synthesis of protein and for gel matching in the estimation of mutation rate of human spices.
- Many areas as-pollution reduction through computational modelling, image processing, and design of computational biologist follow the concept of parallel computing.
- In Electrical Engineering, Circuit design, Microelectronics.
- Today commercial applications provide equal and opposite driving force in the development of faster computer. These applications requires a large amount of data in sophisticated ways. For example: Databases, Data mining, and Oil Exploration.
- Nuclear reactor safety analysis is another area demanding supercomputer facilities.
- Supercomputers are needed in diagnostic equipment such as computer assisted tomographic scanners.

1.5 BENEFITS OF USING PARALLEL COMPUTING

- **Save time and/or Memory:** In theory, throwing more resources at task will shorten it's time to completion, with potential cost saving. Parallel clusters can be built from cheap, commodity and component.
- **Solve larger problem:** Many problems are so large and/or complex that is impractical or impossible to solve them on a single computer, especially given limited computer memory. For example: Web search engines/databases processing millions of transaction per second.
- **Provide Concurrency:** A single compute can only do one thing at a time.

1.6 PARALLEL ARCHITECTURE

1.6.1 Flynn's Classification

Michael Flynn (1972) introduced a various computer architectures based on notion of instruction and data streams. One can classify computers into four categories according to the whether the instruction or data streams are single or multiple. A stream is defined as a sequence of items-instructions or data which is operated by the processor.

- **SISD**: Single Instruction & Single Data Stream
- **SIMD**: Single Instruction & Multiple Data Stream
- **MISD**: Multiple Instruction & Single Data Stream
- **MIMD**: Multiple Instruction & Multiple Data Stream

➢ **SISD:** Single instruction & Single Data Stream is one of the conventional sequential machines which is irrelevant whether pipelining is used to speed up the processing or not. This type of machine is sometimes referred to as a scalar computer.

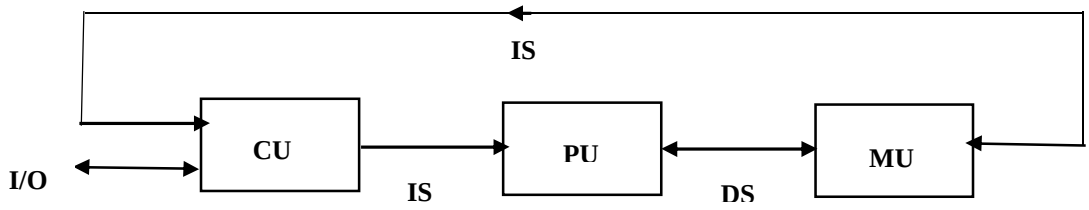


Figure 1.4 (a) SISD uniprocessor architecture

➢ **SIMD:** Single instruction stream & Multiple Data stream type of classification include all machines with vector instructions and machines belonging to this class are often called vector computers.

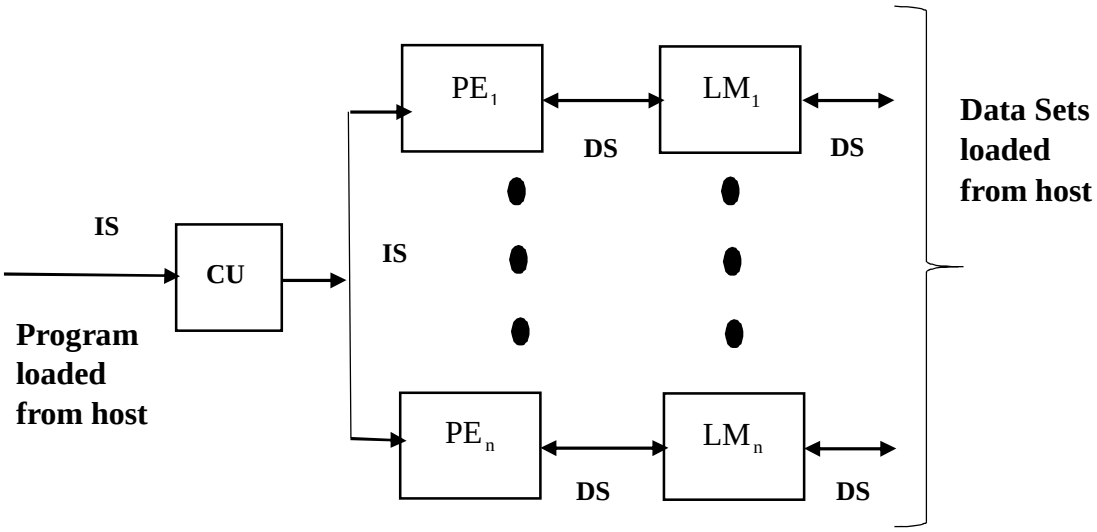


Figure 1.4 (b) SIMD architecture (with distributed memory)

- **MISD:** Multiple Instruction Stream & Single Data Stream in which the same data stream flows through the linear array of processors which is executing different instruction streams.

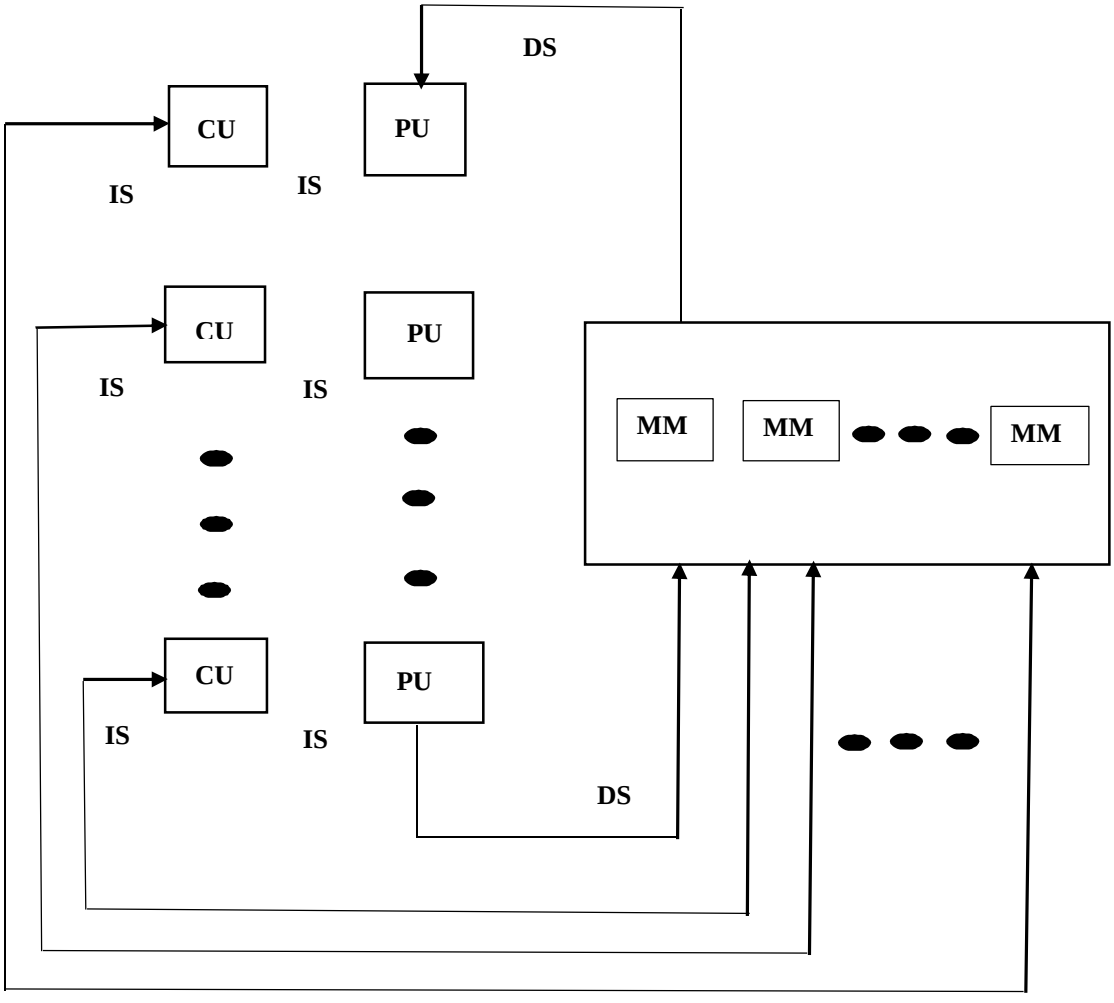


Figure 1.4 (c) MISD architecture (the systolic array)

- **MIMD:** Multiple instruction stream over multiple data streams. Parallel computers are reserved for MIMD machines. This include all form of multiprocessor configuration from LAN (Local area network) and WAN (Wide area network) to the large arrays of multiprocessors.

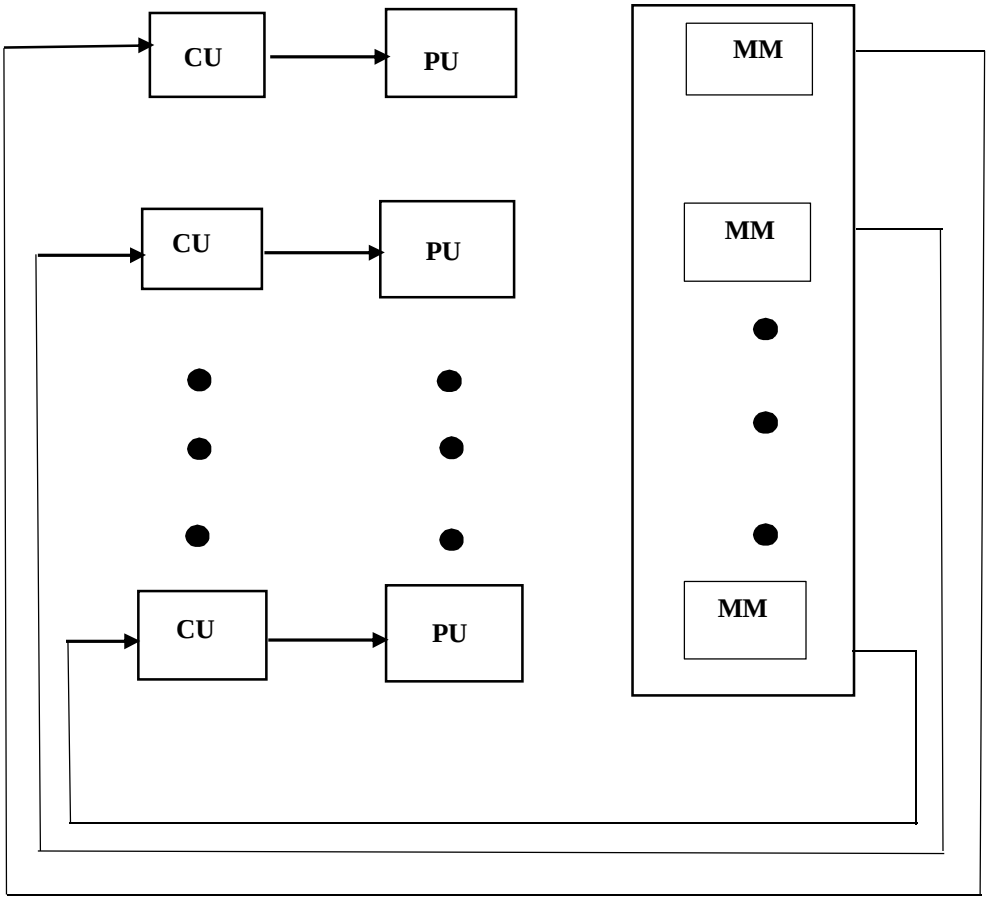


Fig 1.4 (d) MIMD architecture

Captions:

CU= Control unit **PU**= Processing unit

MU= Memory unit **MM**= Memory Module

IS= Instruction Stream **DS**= Data Stream

PE= Processing Element **LM**= Local Memory

I/O= Input /Output

Figure. 1.4 Flynn's classification of computer architectures

In the above discussed machine models, the SIMD and MIMD models are more suitable for special purpose computations, and MIMD model is used for general purpose computations. For this reason, MIMD is most popular model.

The various processors in MIMD machines are linked together. This link has been established in two ways which will be describe as below.

1.7 COUPLING BETWEEN PROCESSING ELEMENTS

The autonomy enjoyed by the PEs while cooperating with one another during problem solving determines degree of coupling between them. In this, each workstations works independently. If they want to cooperate they will exchange a message. Hence we can say that logically they are autonomous and physically they don't share any memory and communication is via I/O channels. On the other hand, a *tightly coupled* parallel computer shares a common main memory. In **Fig.1.5 (a)** we summarize this discussion.

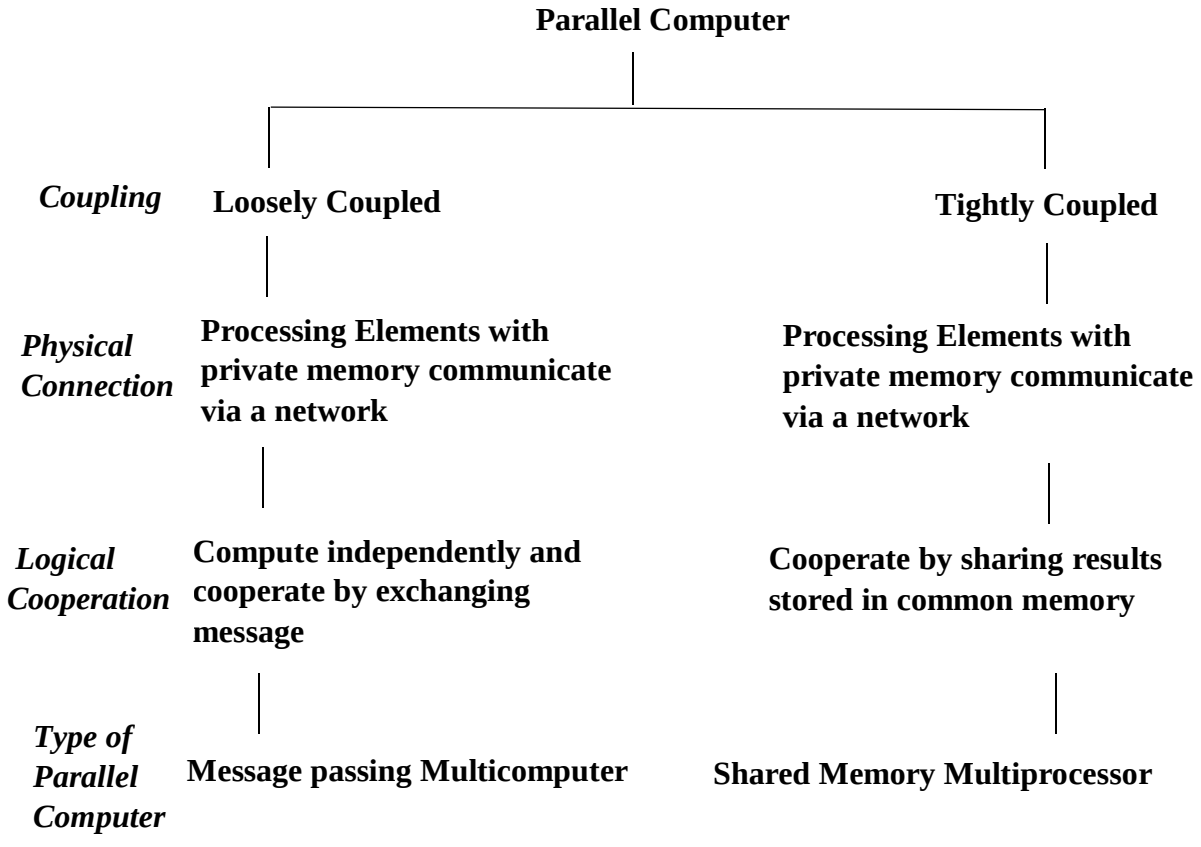


Figure. 1.5 Classification as loosely or tightly coupled system

- **Tightly coupled systems:** In *tightly coupled system* the processors share a global memory. In general, these systems consist of N processors connecting M memory banks through a communication network. Such systems has many processor in closed communication and sharing computer bus, clock, sometimes memory and peripheral devices.

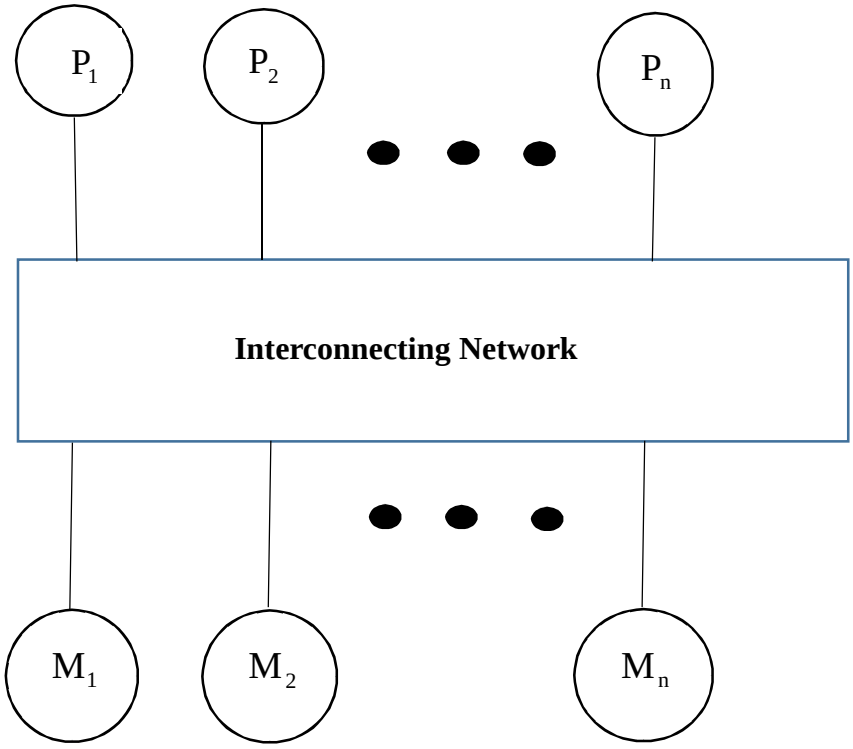
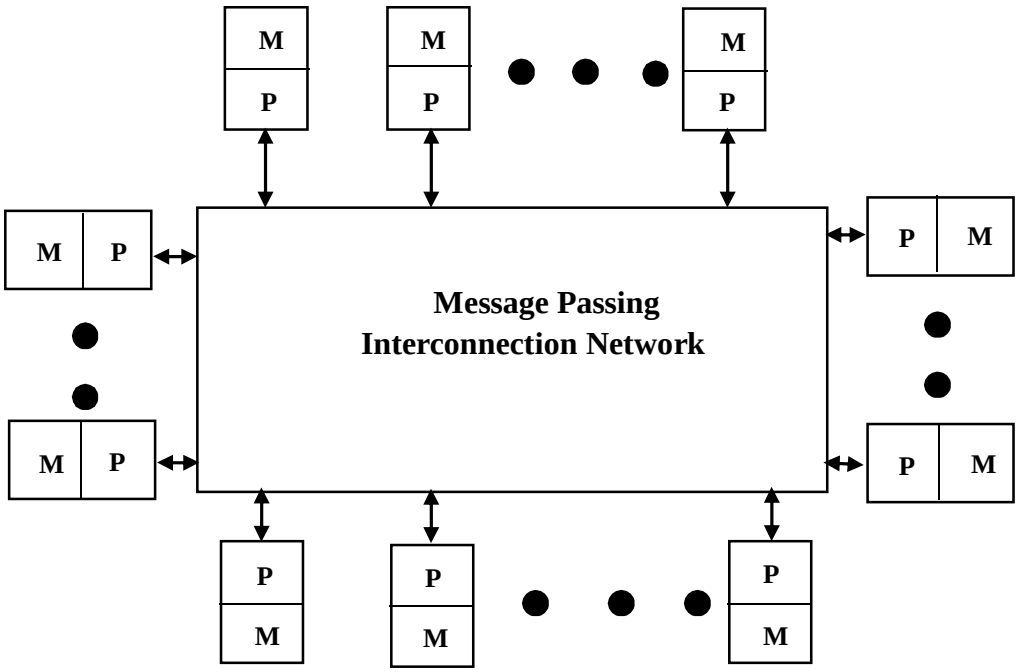


Figure 1.5 (a) Tightly Coupled System

- **Loosely Coupled system:** The computer networks used in these applications consists of a collection of processors that do not share memory or clock. Here each processor has its own local memory. In this arrangement processors communicate with one another through various communication lines like high speed buses or telephone lines.



M= Memory

P=Processor

Fig 1.5 (b) Loosely Coupled System

1.8 MESH CONNECTED COMPUTERS

The **Figure 1.6** shows the basic 2D mesh architecture. Each processor, other than the ones located on the boundary, has degree 4. The free links of the boundary processors can be used for input/output or to establish row and column wraparound connections to form the 2D torus. Here a $k \times k$ mesh has a diameter 2k-2 and bisection width k or k+1. A $k \times k$ torus has diameter k or k-1 and bisection width 2k or 2k+2. A $k \times k$ torus is sometimes referred as a k-ray 2- cube (2D "cube" of size k). The general form of this architecture is known as k-ary or q-cube (q-D cube of size k). In particular, for k=2, we get the class of 2-ary (or binary) q-cubes, also known as (binary) hypercube. Thus, 2D torus and binary hypercube represent the two extremes of the k-ary q-cube architecture; fixing q at 2 gives us the 2D torus architecture with fixed node degree and $\Theta(\sqrt{p})$ diameter, while fixing k at 2 gives us the binary hypercube with logarithmic node degree and $\Theta(\log p)$ diameter.

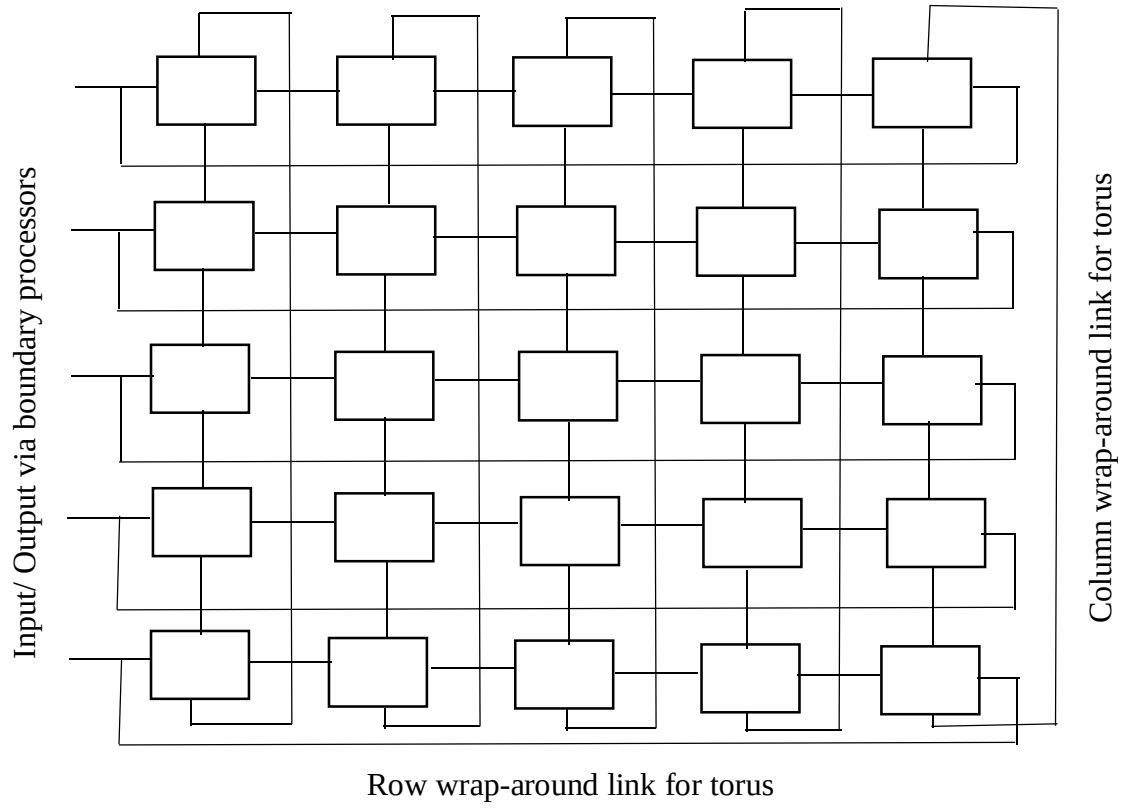


Figure 1.6 Two dimensional Mesh Connected Computer

1.9 HYPERCUBE ARCHITECTURE

There are 2^n nodes in n-dimensional Boolean cube. There are two coordinate points in each direction. The node can be given addresses such that the addresses of adjacent nodes differ in precisely one bit. The Boolean cube is a recursive structure. An n-dimensional can extended to an (n+1) dimensional cube by connecting corresponding vertices of two n-dimensional cubes. One has highest order bit 0 and another has highest order bit 1. The recursive nature of Boolean cube is illustrated in **Figure 1.7**

Here each node has n neighbours. The maximum distance between arbitrary pair of nodes is n and average distance is n/2. The number of nodes at distance k from node is $\binom{n}{k}$. The total of

Internode connection is $n2^{n-1}$. There are n disjoint paths between any pair of processors. Of these paths k are of length k and n-k of length k+2. It is highly fault-tolerant. Its capacity is too good for considering rich connection. It is homogeneous graph without any special node.

In hypercube, two processors are directly connected with communication link if and only if their *Hamming distance* is unity *i.e* their identity numbers differ in exactly one bit, where the *Hamming distance* between two processors is the number of bits in which their identity number differs.

For a given N, the diameter of hypercube is being equal to $\log_2 N$, which is nothing but the number of dimensions. Most of the problem specific to the interconnection network scheme such as mesh, tree, ring etc. can be easily mapped to the hypercube provided the number of nodes, dimension matched.

Figure (a) 0-D Hypercube

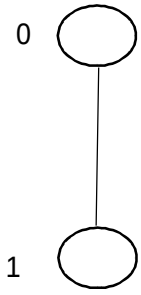


Figure (b) 1-D Hypercube

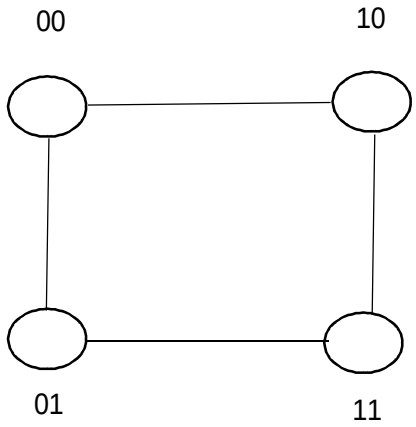


Figure (c) 2-D Hypercube

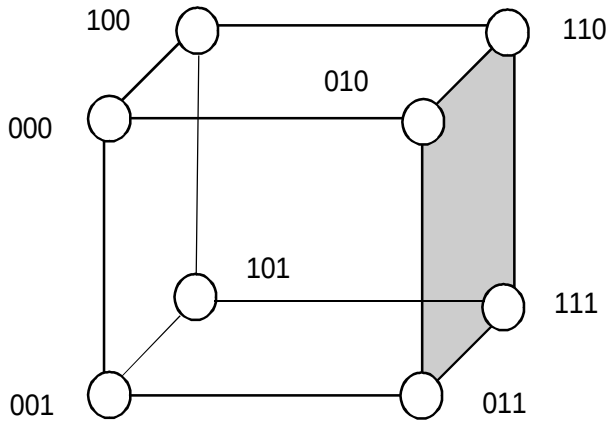


Figure (d) 3-D Hypercube

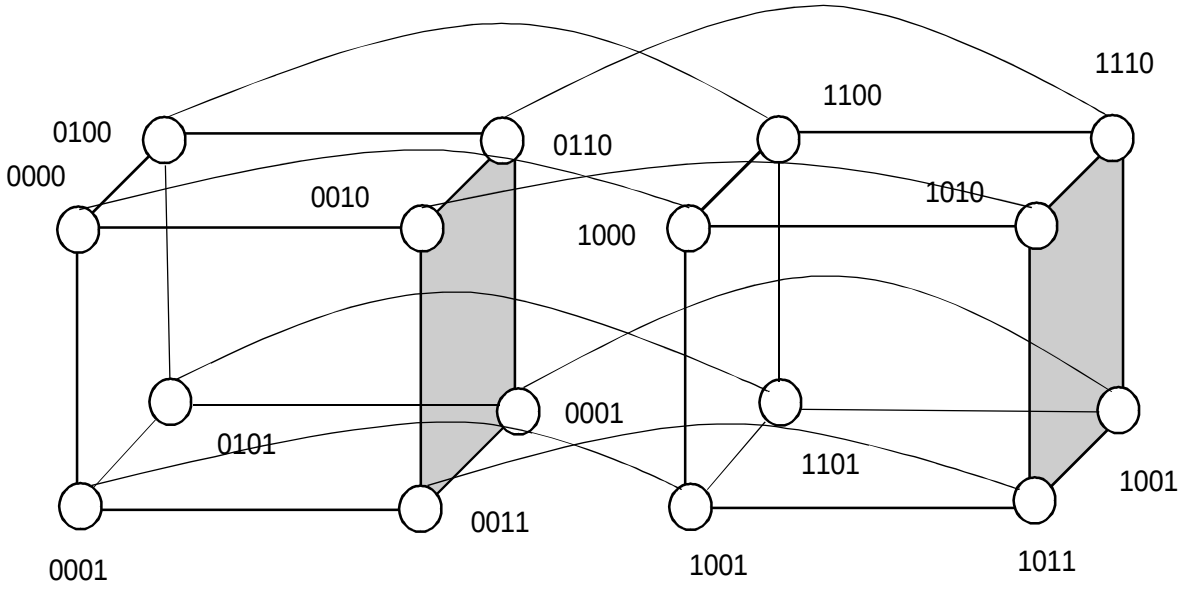


Figure (e) 4-D Hypercube

Figure 1.7 Hypercube Architecture

Chapter 2

Some Parallel Numerical Methods in Solving Partial Differential Equation

2.1 INTRODUCTION

It is lavishly clear that many important scientific problems are governed by partial differential equations according to [4-5]. The difficulty in obtaining exact solution arises from the governing partial differential equations and the complexities of the geometrical configuration of physical problems [10,11,12]. For example, imagine a metal rod insulated along its length with no heat can escape for its surface. If the temperature along the rod is not constant, then heat conduction takes place. In such situations, the numerical method is used to obtain the numerical solutions [2]. These partial differential equations may have boundary value problems as well as initial value problems. First, the PDEs will be written in matrix form to ease the work. Then, parallel algorithm for all types of the PDEs will be developed and run in parallel computing environment to provide the numerical solution. Finally, the speed of convergences of using the numerical methods will be compared. In general, the transient particle diffusion or heat conduction is Partial Differential Equations (PDE) of the parabolic type and Laplace's equation for temperature, diffusion, electrostatic conduction is elliptic and wave equation or transport equation is the PDE of hyperbolic type [4,5,12]. The parabolic partial differential equations are normally used in such fields like molecular diffusion, heat transfer, nuclear reactor analysis, and fluid flow [1,6].

Partial differential equations (PDEs) widely used as mathematical models for phenomena in all branches of engineering and science.

A Parabolic Equation

$$\frac{\partial u}{\partial t} = a_1(x,y,t)\frac{\partial^2 u}{\partial x^2} + a_2(x,y,t)\frac{\partial^2 u}{\partial y^2} + b_1(x,y,t)\frac{\partial u}{\partial x} + b_2(x,y,t)\frac{\partial u}{\partial y} - c(x,y,t) \qquad \text{(i)}$$

where $a < 0, c \geq 0$ and $b^2 - 4ac = 0$. The PDE is said to be parabolic if det(Z) = 0 . The heat conduction equation and other diffusion equation are examples. The heat equation is as $\frac{\partial U}{\partial T} = \kappa \frac{\partial^2 U}{\partial X^2}.$, κ is a constant. Initial boundary conditions are used to give

$$\begin{cases} u(x,\ t) = g(x,\ t) \text{ for } x \in \partial\Omega\ ,\ t > 0 \\ u(x,0)\ = (x) \text{ for } x \in \Omega, \\ \text{where } ux\ x\ = f(ux,\ uy,\ u,\ x,\ y) \text{ holds in } \Omega\ . \end{cases}$$

B Hyperbolic Equation

$$\frac{\partial^2 u}{\partial t^2} - \left(a\frac{\partial^2 u}{\partial x^2} + 2b\frac{\partial^2 u}{\partial x \partial y} + c\frac{\partial^2 u}{\partial y^2} \right) + d\frac{\partial u}{\partial t} + e\frac{\partial u}{\partial x} + f\frac{\partial u}{\partial y} + gu = 0 \qquad \text{(ii)}$$

Where $b^2 - 4ac > 0.$ The PDE is said to be hyperbolic if det(Z) < 0 .The wave equation is an example of a hyperbolic partial differential equation. The wave equation is as

$\frac{\partial^2 u}{\partial x^2} - \frac{1}{\beta}\frac{\partial^2 u}{\partial t^2} = 0$, β is constant. Initial-boundary conditions are used to give

$$\begin{cases} u(x,y,t) = g(x,y,t) \text{ for } x \in \partial\Omega,\ t > 0 \\ u(x,y,0) = v_0(x) \text{ for } x \in \Omega, \\ ut(x,y,0) = v_1(x) \text{ for } x \in \Omega, \\ \text{where} ux\,y = f(ux, ut, x, y) \text{ holds in } \Omega. \end{cases}$$

C Elliptic Equation

$$a(x,y)\frac{\partial^2 u}{\partial x^2} + 2b(x,y)\frac{\partial^2 u}{\partial x \partial y} + c(x,y)\frac{\partial^2 u}{\partial y^2} = d\left(x, y, u, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}\right) \quad \text{(iii)}$$

where $b^2 - 4ac > 0.$ The PDE is said to be elliptic if Z is a positive definite matrix with det(Z) < 0 . Laplace's equation and poisson's equation are examples. The Laplace's equation is $\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = 0.$ Boundary conditions are used to give the constraint u(x,y) on $\partial\Omega$.

where $ux\,x + ux\,y = f(ux, uy, u, x, y)$

D Finite Difference Method

Finite Difference Method is a classical and straightforward way to solve the partial difference equation [1,2] numerically. It consists of transforming the partial derivatives in difference equations over a small interval and the continuous domain of the state variables by a network or mesh of discrete points. The partial differential equation is converted into a set of finite difference equations so that it can be solved subject to the appropriated boundary conditions. Assuming that *u* is function of the independent variables x and y, then divided the x-y plan in mesh points equal to $\delta x = h$ and $\delta y = k$,

Evaluate u at point P by:

$$u_p = u(ih, jk) = u_{i,j} \quad \text{(iv)}$$

The value of the second derivative at P could also be evaluated by:

$$\begin{cases} \left(\frac{\partial^2 u}{\partial x^2}\right)_p = \left(\frac{\partial^2 u}{\partial x^2}\right)_{i,j} \cong \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} \\ \left(\frac{\partial^2 u}{\partial y^2}\right)_p = \left(\frac{\partial^2 u}{\partial y^2}\right)_{i,j} \cong \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{k^2} \end{cases} \quad \text{(v)}$$

2.2 METHOD TO SOLVE PARTIAL DIFFERENTIAL EQUATIONS

The most common method to solve the Partial Differential Equations is the Finite Difference Method.

2.2.1: Finite Difference Method: According to heat flow equation we know that $\frac{\partial u}{\partial y} = \frac{\partial^2 u}{\partial x^2}$.

Now let us assume that our region of interest is $[a \le x \le b] \times [0, T]$ and h and k are the mesh sizes in the x and t directions. A simple difference replacement of derivatives at the node (m,n) in given equation gives the difference equation as

$$\frac{u(x_{m,}t_{n+1}) - u(x_{m,}t_n)}{k} + 0(k) = \frac{u(x_{m+1,}t_n) - 2u(x_{m,}t_n) + u(x_{m-1,}t_n)}{h^2} + 0(h^2) \qquad \text{(vi)}$$

Here we put y=t.

If we neglect the truncation error then we can write equation (vi) as

$$u_m^{n+1} = u_m^n + \left(\frac{k}{h^2}\right)\left(u_m^{n+1} - 2u_m^n + u_{m-1}^n\right) \qquad \text{(vii)}$$

Here u_m^n represents the value of u(x,t) at

X=a+mh, t=nk

In schematic form eq'n (xv) can be written as

	-1	
r	1-2r	r

u=0

Where r= k/h^2 .The difference scheme is used to compute u_m^n line by line, starting from given data in the t-direction. Hence we can say that the order of accuracy of difference scheme or the truncation error of order $(k + h^2)$.

If we subtract (xiv) from (xv) then we can get

$$\in_m^{n+1} = (1-2r) \in_m^n + (\in_{m+1}^n + \in_{m-1}^n) + 0(k^2 + kh^2) \qquad \text{(viii)}$$

Where $\in_m^n = u_m^n - u(x_m, t_n)$ and the effect of the round-off error is neglected.

2.3 DIFFERENCE SCHEME FOR EQUATIONS IN ONE SPACE DIMENSION WITH CONSTANT COEFFICIENTS OR SOLUTION OF PARABOLIC PARTIAL DIFFERENTIAL EQUATIONS:

Consider an example heat flow equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$$

Here t and x are the initial and time space coordinates respectively, in the region

$$R = [a \le x \le b] \times [t \ge 0],$$

With initial and boundary conditions.

The region R is replaced by a set of points R_k which r the vertices of the grid of points (m,n) where x=a+mh,t=nk with Mh=b-a,M is an integer. The quantities k and h are the mesh sizes in the time and space directions respectively. The difference approximation at the nodal point (m,n) can be written as

$$G(\nabla_t)u_m^{n+1} = r\delta_x^2 u_m^n \qquad \text{(xvii)}$$

$$\text{where } r = \frac{k}{h^2}, \left(\frac{\partial^2 u}{\partial x^2}\right)_m^n = h^{-2}\delta_x^2 u(x_m, t_n) + O(h^2) \qquad \text{(xviii)}$$

here approximate value of u at (x_m, t_n) is denoted by u_m^n.

In the implicit difference scheme it involves grid values at more then one grid points at time grid t=(n+1)k. Thus implicit difference scheme becomes

$$F(\nabla_t)u_m^{n+1} = r(1+\sigma\delta_x^2)^{-1}\delta_x^2 u_m^{n+1} \qquad \text{(ix)}$$

$$\text{Here} \quad \left(\frac{\partial^2 u}{\partial x^2}\right)_m^n = h^{-2}(1+\sigma\delta_x^2)^{-1}\delta_x^2 u(x_m, t_{n+1}) + \begin{cases} O(h^2), \sigma.\text{arbitrary} \\ O(h^4), \sigma = \frac{1}{12} \end{cases}$$

- **Example:** Use the Schmidt scheme to determine the numerical solution of the initial boundary value problem

$$\begin{cases} u_t = u_{xx} \\ u(x,0) = \sin \Pi x, 0 < x < 1 \\ u(0,t) = u(1,t) = 0, t \ge 0 \end{cases}$$

The Schmidt difference scheme is given by

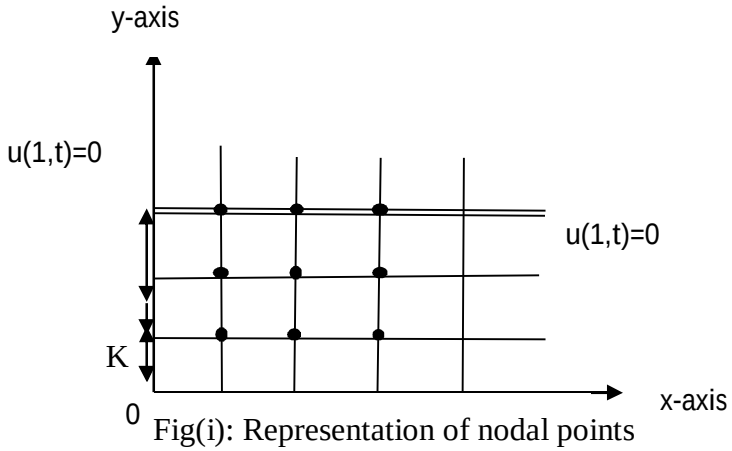
$$u_m^{n+1} = u_m^n + r(u_{m+1}^n - 2u_m^n + u_{m-1}^n)$$

Where u_m^n is the approximate value of solution u(x,t) at $x_m = x_0 + mh, t_n = nk$ and $r = k/h^2$. If we choose $h = 1/4$ and $r = 1/6$. The Difference scheme maybe written as

$$u_m^{n+1} = u_m^n + \frac{1}{6}(u_{m+1}^n + 4u_m^n + u_{m-1}^n)$$

Where m=1(1)3 and $n \geq 0$. The nodal points are shown below. By using the boundary conditions, we have

$u_0^n = 0, u_4^n = 0, n = 0,1,2,...$



Fig(i): Representation of nodal points

Now we can obtain as,

For n=0,m=1,2,3;

$$u_1^1 = \frac{1}{6}\left(u_0^0 + 4u_1^0 + u_2^0\right) = \frac{1}{6}\left(0 + 4\sin\frac{\Pi}{4} + \sin\frac{\Pi}{2}\right)$$

$$u_1^1 = .6380711$$

$$u_2^1 = \frac{1}{6}\left(u_1^0 + 4u_2^0 + u_3^0\right) = \frac{1}{6}\left(\sin\frac{\Pi}{4} + 4\sin\frac{\Pi}{2} + \sin\frac{3\Pi}{4}\right)$$

$$u_2^1 = 9023689$$

$$u_3^1 = \frac{1}{6}\left(u_2^0 + 4u_3^0 + u_4^0\right) = \frac{1}{6}\left(\sin\frac{\Pi}{2} + 4\sin\frac{3\Pi}{2}\right)$$

$$u_3^1 = .6380711$$

For n=1,m=1,2,3;

$$u_1^2 = \frac{1}{6}\left(u_0^1 + 4u_1^1 + u_2^1\right) = \frac{1}{6}(4(.6380711) + .9023689)$$

$$u_1^2 = .5757755$$

$$u_2^2 = \frac{1}{6}\left(u_1^1 + 4u_2^1 + u_3^1\right) = \frac{1}{6}(.6380711 + 4(.9023689) + .6380711)$$

$$u_2^2 = .8142696$$

$$u_3^2 = \frac{1}{6}\left(u_2^1 + 4u_3^1 + u_4^1\right) = \frac{1}{6}(.9023689 + 4(.6380711) + 0)$$

$$u_3^2 = .5757755$$

The solution u(x,t) is symmetric about the line $x=\frac{1}{2}$.

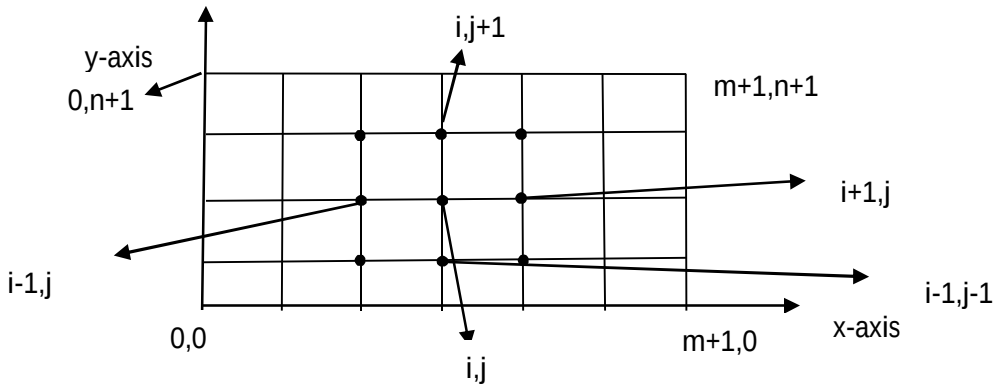
- **NOTE:** Similarly we have Difference Methods for Hyperbolic partial differential Equation and Elliptic Partial Differential Equation both.

❖ **Example of Difference Method for Elliptic Partial Differential Equation:**

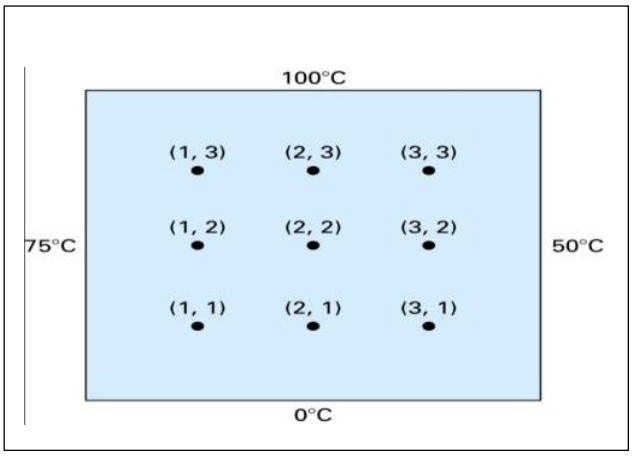
- Solution Technique:
- Elliptic equations in engineering are typically used to characterize steady-state, boundary value problems.
- For numerical solution of elliptic PDEs, the PDE is transformed into an algebraic difference equation.
- Because of its simplicity and general relevance to most areas of engineering, we will use a heated plate as an example for solving elliptic PDEs.

Here equation as:

$$\begin{cases} \dfrac{\partial^2 T}{\partial x^2}+\dfrac{\partial^2 T}{\partial y^2}=0\,,\text{Laplace Equation} \\ \dfrac{\partial^2 T}{\partial x^2}=\dfrac{T_{i+1,j}-2T_{i,j}+T_{i-1,j}}{\Delta x^2};\ O\left(\Delta x^2\right) \\ \dfrac{\partial^2 T}{\partial x^2}=\dfrac{T_{i,j+1}-2T_{i,j}+T_{i,j-1}}{\Delta y^2};\ O\left(\Delta y^2\right) \\ T_{i+1,j}-4T_{i,j}+T_{i-1,j}+T_{i,j+1}+T_{i,j-1}=0\,;\text{Laplace Difference Equation} \end{cases}$$



➢ **Example :**



- Solution :
- In addition, boundary conditions along the edges must be specified to obtain a unique solution.
- The simplest case is where the temperature at the boundary is set at a fixed value, Dirichlet boundary condition.

Now at point (1,1) the given Laplace Equation becomes:

$$\begin{cases} T_{21} + T_{01} + T_{12} + T_{10} - 4T_{11} = 0 \\ T_{10} = 75 \\ T_{10} = 0 \end{cases}$$

Hence the final equations becomes:

$$\begin{cases} \text{at } (1,1): -T_{21} - T_{12} + 4T_{11} = 75 \\ \text{at } (2,1): 4T_{21} - T_{13} - T_{11} - T_{22} = 0 \\ \text{at } (1,2): -T_{11} - T_{22} + 4T_{12} - T_{13} = 75 \\ \text{at } (3,1): -T_{21} - T_{32} + 4T_{31} = 50 \\ \text{at } (2,2): -T_{21} - T_{12} + 4T_{22} - T_{32} - T_{23} = 0 \\ \text{at } (3,2): -T_{31} - T_{22} + 4T_{32} + T_{33} = 50 \\ \text{at } (1,3): -T_{12} + 4T_{13} - T_{23} = 175 \\ \text{at } (2,3): -T_{21} - T_{13} + 4T_{23} - T_{33} = 100 \\ \text{at } (3,3): -T_{31} - T_{23} + 4T_{33} = 150 \end{cases}$$

We can write above equation in Matrix form as:

$$\begin{bmatrix} 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & 4 & -1 & 0 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 4 & -1 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & -1 & 4 & 1 \\ 0 & -1 & 4 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & 0 & 4 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 4 \end{bmatrix} \begin{bmatrix} T_{11} \\ T_{12} \\ T_{13} \\ T_{21} \\ T_{22} \\ T_{23} \\ T_{31} \\ T_{32} \\ T_{33} \end{bmatrix} = \begin{bmatrix} 75 \\ 0 \\ 75 \\ 50 \\ 0 \\ 50 \\ 175 \\ 100 \\ 150 \end{bmatrix}$$

2.4 TWO-DIMENSIONAL PDE SOLVER

A Hyperbolic Partial Differential Equations

Hyperbolic differential equations, includes the "wave equation" which is fundamental to the study of vibrating systems. It is instructive to outline the derivation of the simple wave equation in one dimension problem.

The wave equation is given by the differential equation

$$\frac{\partial^2 u}{\partial t^2}(x,t) - \alpha^2 \frac{\partial^2 u}{\partial x^2}(x,t) = 0 \,, 0 < x < L, t > 0 \qquad \text{(x)}$$

Subject to the boundary conditions

$$u(0,t) = u(L,t) = 0 \,, t > 0 \text{ and initial conditions } u(x,0) = f(x), 0 \le x \le L,$$

$$\frac{\partial u}{\partial t}(x,0) = g(x) \,, 0 \le x \le L \quad \text{where } \alpha \text{ is constant.}$$

To set up the finite difference method, assume u=f(x) is the function of the independent variables x and t. Subdivide the x-plane into set of equal rectangles if sides $\delta x = h$ and $\delta t = k$. We introduce a time grid $t_n = n\nabla t$ for n= 0,1,2,3,4.. and ∇t is the time step size. We set

$p^n(x) = p(x,t_n)$ as the nth iterate of the pressure of the global point x. The time derivative in (4) are discretised by cantered second order finite difference, which gives the semi-discrete scheme:

$$\frac{P^{n+1} - 2P^n + P^{n-1}}{\Delta t^2} + \chi^2 \frac{P^{n+1} - P^{n-1}}{2\Delta t} = c^2 \Delta^2 P^n \qquad \text{(xi)}$$

B Two Dimensional Parabolic Equations

A forward finite difference is used to approximate the time derivative. Consider the two-dimensional of parabolic equations

$$\frac{\partial u}{\partial t} = c\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right), \text{ c is constant} \qquad \text{(xii)}$$

Applying the crank Nicolson scheme to the two-dimensional heat equation results in

$$\frac{u^{(n+1)} - u^{(n)}}{\Delta t} = \frac{c}{2}\left[\frac{\partial^2 u^{(n+1)}}{\partial x^2} + \frac{\partial^2 u^{(n+1)}}{\partial y^2} + \frac{\partial^2 u^{(n)}}{\partial x^2} + \frac{\partial^2 u^{(n)}}{\partial y^2}\right] \qquad \text{(xiii)}$$

This leads to the following finite difference equation

$$\begin{aligned}
&N_{ij}(t+\Delta t) = \\
&N_{ij}(t) + \Delta t[P^{j}_{i-1,i} N_{i-1,j}(t) + R^{j-1,j}_{i} N_{i,j-1}(t) - P^{j}_{i,i+1} N_{ij}(t) \\
&- R^{j,j+1}_{i} N_{ij}(t) + Q^{j}_{i-1,i} N_{i-1,j}(t) + Q^{j}_{i+1,i} N_{i+1,j}(t) \\
&+ Q^{j-1,j}_{i} N_{i,j-1}(t) + Q^{j+1,j}_{i} N_{i,j+1}(t) \\
&- \left(Q^{j}_{i,i-1} + Q^{j}_{i,i+1} + Q^{j,j-1}_{i} + Q^{j,j+1}_{i}\right) N_{ij}(t) + \Gamma_{i,j} - L_{ij} N_{ij}(t)]
\end{aligned}$$

where $\Gamma_{i,j}$ and L_{ij} are the generation and death rates, respectively. Under suitable regularity assumption one can expand N,P,Q, and R, use $N_{i,j}(t) \approx u(t, x_{i,j})\Delta V_{i,j}$ and write the word equation above mathematically as:

$$\frac{\partial u}{\partial t} = -\frac{\partial (Pu)}{\partial x} - \frac{\partial}{\partial x}\left(Q\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(Q\frac{\partial u}{\partial y}\right) + \Gamma - Lu, \qquad \text{(xiv)}$$

With $\Gamma(t, x_i, y_j) = \Gamma_{i,j}(t) / \Delta V_{i,j}$ and where the indices (i,j) have been substituted with the dependence of u and of all coefficients on the space variable. We can also write as

$$\frac{\partial u}{\partial t} + \nabla.(Wu) = \nabla.(Q\nabla u) + \Gamma - Lu, \qquad \text{(xv)}$$

Where, in two dimensions, W = (P, R). The general advection-diffusion model (xv) requires the specification of the drift, diffusion, proliferation, and death coefficient in the terms W, Q, Γ and L in particular of their dependence of the state variables. Based on central finite difference method, the discretization is shown as follow:

Let $\frac{\partial u}{\partial t} = \frac{N_{ij}(t+\Delta t) - N_{ij}(t)}{\Delta t}$ and applying the discretization to the right side, the equation

(xiv) becomes

$$
\begin{aligned}
&\frac{N_{ij}(t+\Delta t)-N_{ij}(t)}{\Delta t}\\
&=\left[P^{j}_{i-1,i}N_{i-1,j}(t)-P^{j}_{i,j+1}N_{ij}(t)\right]+\left[R_{i}^{j-1,j}N_{i,j-1}(t)-R_{i}^{j,j+1}N_{ij}(t)\right]\\
&+\left[Q^{j}_{i+1,i}N_{i+1,j}(t)-Q^{j}_{i,i-1}N_{ij}(t)\right]+\left[Q^{j}_{i-1,i}N_{i-1,j}(t)-Q^{j}_{i,i+1}N_{ij}(t)\right]\\
&+\left[Q_{i}^{j+1,j}N_{i,j+1}(t)-Q_{i}^{j,j-1}N_{i,j}(t)\right]+\left[Q_{i}^{j-1,j}N_{i,j-1}(t)-Q_{i}^{j,j+1}N_{ij}(t)\right]\\
&+\Gamma_{i,j}-L_{ij}N_{ij}(t)\\
&=\left[P^{j}_{i-1,i}N_{i-1,j}(t)-P^{j}_{i,i+1}N_{ij}(t)\right]+\left[R_{i}^{j-1,j}N_{i,j-1}(t)-R_{i}^{j,j+1}N_{ij}(t)\right]\\
&+\left[Q^{j}_{i-1,i}N_{i-1,j}(t)-\left(Q^{j}_{i,i-1}+Q^{j}_{i,i+1}\right)N_{ij}(t)+Q^{j}_{i+1,i}N_{i+1,j}(t)\right]\\
&+\left[Q_{i}^{j-1,j}N_{i,j-1}(t)-(Q_{i}^{j,j-1}+Q_{i}^{j,j+1})N_{ij}(t)+Q_{i}^{j+1,j}N_{i,j+1}(t)\right]\\
&+\Gamma_{i,j}-L_{ij}N_{ij}(t)
\end{aligned}
$$

C Two Dimensional Elliptic Equation

The two dimensional elliptic equation $\frac{\partial^2 r}{\partial x^2}+\frac{\partial^2 r}{\partial y^2}=0$ can be further implemented to solve the large scale mathematical problem. Generally, finite-difference approximation to two dimensional elliptic equation is given by

$$\frac{r_{i-1,j}-2r_{i,j}+r_{i+1,j}}{(\Delta x)^2}+\frac{r_{i,j-1}-2r_{i,j}+r_{i,j+1}}{(\Delta y)^2}=0 \qquad \text{(xvi)}$$

where $\Delta x = h$, $\Delta y = k$

$$\frac{r_{i-1,j}-2r_{i,j}+r_{i+1,j}}{(h)^2}+\frac{r_{i,j-1}-2r_{i,j}+r_{i,j+1}}{(k)^2}=0 \qquad \text{(xvii)}$$

by multiplying each side with h^2 , we have

$$r_{i-1,j}-2r_{i,j}+r_{i+1,j}+\frac{h^2}{k^2}\left(r_{i,j-1}-2r_{i,j}+r_{i,j+1}\right)=0 \qquad \text{(xviii)}$$

If we assume $\theta=\frac{h^2}{k^2}$, then we will have the finite-difference approximation equation is as follows

$$\theta\, r_{i,j-1}+r_{i-1,j}-(2+2\theta)r_{i,j}+r_{i+1,j}+\theta\, r_{i,j+1}=0 \qquad \text{(xix)}$$

$$\text{For } 0\le\theta\le 1$$

The discretization of the mathematical model based on the finite-difference approximation to equation (28) can be written as,

$$\left(\frac{\partial^2 r}{\partial x^2}+\frac{\partial^2 r}{\partial y^2}+k^{-2}(r)\right)e(r)=0 \qquad \text{(xx)}$$

After applying the finite-difference approximation to equation (xx) is given by

$$\left(\frac{r_{i+1,j}-2r_{i,j}+r_{i-1,j}}{(\Delta x)^2}+\frac{r_{i,j+1}-2r_{i,j}+r_{i,j-1}}{(\Delta y)^2}+k^{-2}\left(r_{i,j}\right)\right)e\left(r_{i,j}\right)=0 \qquad \text{(xxi)}$$

From equation (xxi) , it becomes

$$\left(\frac{r_{i+1,j}-2r_{i,j}+r_{i-1,j}}{(h)^2}+\frac{r_{i,j+1}-2r_{i,j}+r_{i,j-1}}{(k)^2}+k^{-2}\left(r_{i,j}\right)\right)e\left(r_{i,j}\right)=0 \qquad \text{(xxii)}$$

where $\Delta x = h, \Delta y = k$. If we bring the $e\left(r_{i,j}\right)$ terms to the right-hand side, it become $e\left(r_{i,j}\right)=0$. Thus,

$$\frac{r_{i+1,j}-2r_{i,j}+r_{i-1,j}}{(h)^2}+\frac{r_{i,j+1}-2r_{i,j}+r_{i,j-1}}{(k)^2}+k^{-2}\left(r_{i,j}\right)=0 \qquad \text{(xxiii)}$$

By multiplying each side by h^2 equation (35) becomes

$$r_{i+1,j}-2r_{i,j}+r_{i-1,j}+\left(\frac{h^2}{k^2}\right)\left[r_{i,j+1}-2r_{i,j}+r_{i,j-1}\right]+h^2k^{-2}r_{i,j}=0 \qquad \text{(xxiv)}$$

The exact solution to the discretised problem obeys the equation

$$r_{i+1,j}+r_{i-1,j}-\left[2+2\left(\frac{h^2}{k^2}\right)-h^2k^{-2}\right]r_{i,j}+\left(\frac{h^2}{k^2}\right)r_{i,j+1}+\left(\frac{h^2}{k^2}\right)r_{i,j-1}=0 \qquad \text{(xxv)}$$

$$\left[2+2\left(\frac{h^2}{k^2}\right)-h^2k^{-2}\right]r_{i,j}=r_{i+1,j}+r_{i-1,j}+\left(\frac{h^2}{k^2}\right)r_{i,j+1}+\left(\frac{h^2}{k^2}\right)r_{i,j-1} \qquad \text{(xxvi)}$$

Thus,

$$r_{i,j} = \frac{r_{i+1,j} + r_{i-1,j} + \left(\frac{h^2}{k^2}\right) r_{i,j+1} + \left(\frac{h^2}{k^2}\right) r_{i,j-1}}{\left[2 + 2\left(\frac{h^2}{k^2}\right) - h^2 k^{-2}\right]} \qquad \text{(xxvii)}$$

This equation cannot be solved explicit for fixed $r_{i,j}$ because there are five unknowns involved. Thus, if the n^{th} iterate is denoted $r_{i,j}^{n}$.

2.5 PARALLEL PERFORMANCE EVALUATION

The performance of the parallel algorithm will be analyzed in terms of the time execution, speedup, efficiency, effectiveness and temporal performance. The measurements are defined as follows:

Speedup: $$S(P) = \frac{t_1}{t_p} \qquad \text{(i)}$$

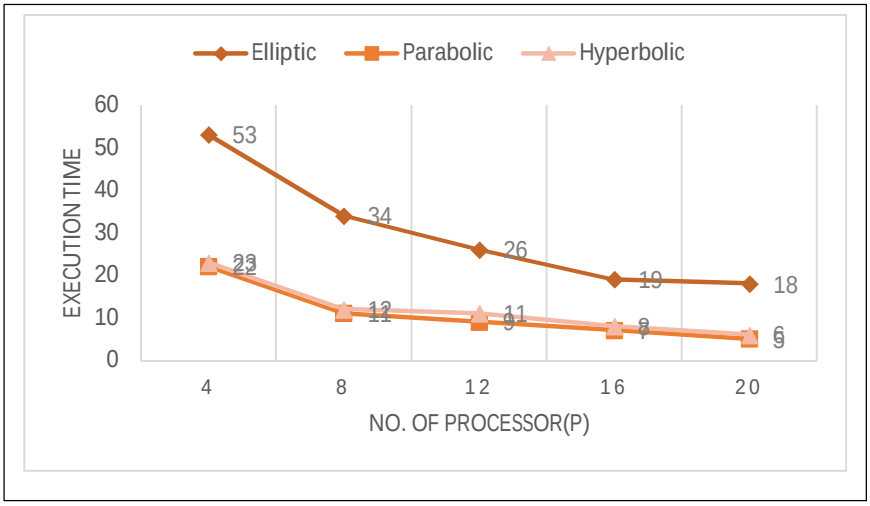
Effectiveness: $$E(P) = \frac{S(P)}{Pt_p} = \frac{E_p}{t_p} \qquad \text{(ii)}$$

Temporal performance: $$L(P) = t_p^{-1} = \frac{1}{t_p} \qquad \text{(iii)}$$

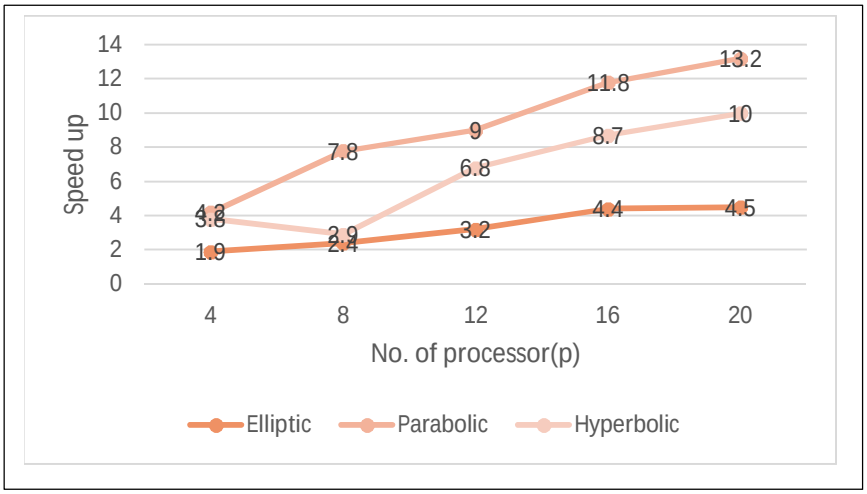
Here t_1 is execution time for a single processor and t_p is execution time for a p parallel processor

Figure 1(a) shows that the execution time is decreasing with the increasing of the number of processors. The reduction of execution time as number of processors increase can also be seen in solving parabolic and hyperbolic problem. Figure 1(b) shows that the speedup increases when the number of processors is added. It is because the distributed memory hierarchy reduces the time consuming access to a cluster of workstations. The efficiency of a parallel program is a measure of processor utilization. Figure 1(c) shows that the efficiency decreases with the increasing of number of processors. As known, efficiency is the ratio of speedup with number of processors. So, efficiency is a performance closely related to speedup. The effectiveness is escalating with the increasing of the number of processors. The formula of the effectiveness is depending on the speedup, when the speedup increases, the effectiveness will also increase. Figure 1(e) shows that the temporal performance graph is proportional to the number of processors increase. This is because the execution time is decreasing versus the number of processors. It can be conclude that, from the aspect of execution time, speedup, efficiency,

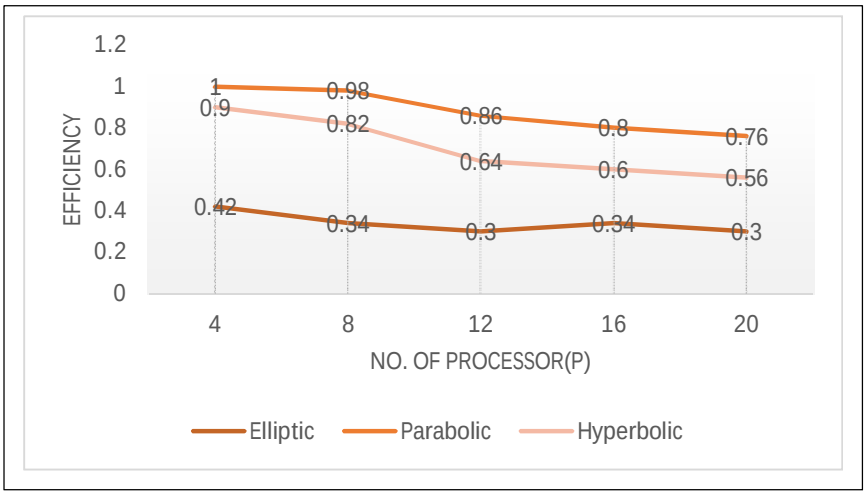
effectiveness and temporal performance shows the performance of parallel algorithm is improved by the increasing of the number of processors. Communication and execution times is always affecting the performance of parallel computing. The Red Black Gauss Seidel which is effective is found to be well suited for parallel implementation on PVM where data decomposition is run synchronously and concurrently at every time level. The PVM system has been used for applications such as molecular dynamics simulations, superconductivity studies, distributed fractal computations, matrix algorithms, and in the classroom as the basis for teaching concurrent computing.



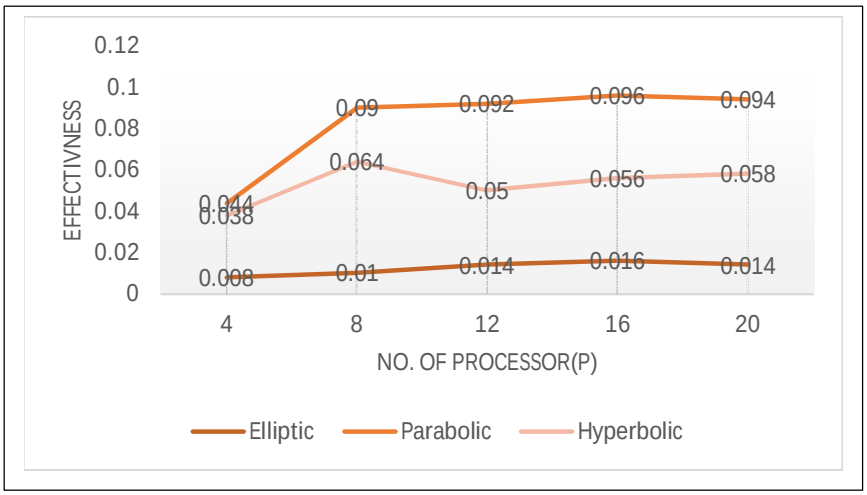
(a)



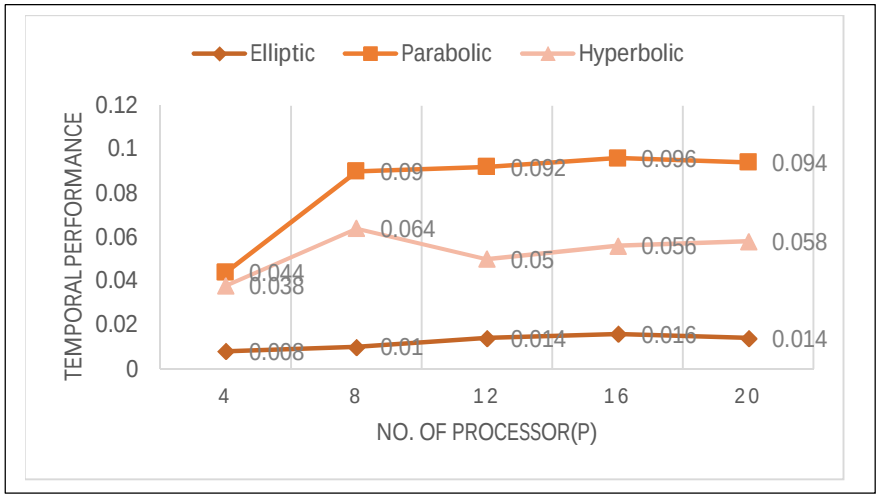
(b)


1.2
1
0.8
0.6
0.4
0.2
0
EFFICIENCY
1
0.98
0.86
0.8
0.76
0.9
0.82
0.64
0.6
0.56
0.42
0.34
0.3
0.34
0.3
4
8
12
16
20
NO. OF PROCESSOR(P)
Elliptic
Parabolic
Hyperbolic

(c)


0.12
0.1
0.08
0.06
0.04
0.02
0
EFFECTIVNESS
0.09
0.092
0.096
0.094
0.064
0.05
0.056
0.058
0.044
0.038
0.008
0.01
0.014
0.016
0.014
4
8
12
16
20
NO. OF PROCESSOR(P)
Elliptic
Parabolic
Hyperbolic

(d)



. (e)

Figure 2.1 Parallel Performance Evaluation (a) Execution Time (b) Speedup (c) Efficiency (d) Effectiveness (e) Temporal Performance

Chapter 3

Proposed Algorithm to solve Block Tridiagonal Systems

Block Tridiagonal systems of linear equations arise in a wide variety of scientific and engineering applications. Recursive doubling algorithm is a well-known prefix computation based numerical algorithm that requires $O(M^3(N/P + \log P))$ work to compute the solution of a block Tridiagonal system with N block rows and block size M on P processors. In real-world applications, solutions of Tridiagonal systems are most often sought with multiple, often hundreds and thousands, of different right hand sides but with the same Tridiagonal matrix. Here, we show that a proposed recursive doubling algorithm is sub-optimal when computing solutions of block Tridiagonal systems with multiple right hand sides and present a novel algorithm, called the accelerated recursive doubling algorithm, that delivers O(R) improvement when solving block Tridiagonal systems with R distinct right hand sides.

A**Z** = **b** is the form of block Tridiagonal system of equations which is represented by a matrix-vector equation of the form in which the block Tridiagonal matrix A has N × N array of blocks where each block is an M × M array of numbers and the elements other than its three central block diagonals are each identically equal to zero. Vectors **x** and **y** are each of length NM. We will often refer to a block Tridiagonal system with N rows of M×M blocks simply as an (M,N) system. When M = 1, then the Tridiagonal system is called as a point Tridiagonal system.
Here, we present a short review of this parallel primitive followed by a brief discussion of the theoretical complexity and empirical performance effects of directly applying the RDA, originally designed for point systems [27], to block Tridiagonal systems.

3.1 PARALLEL SCAN

Given N data items $Z_1, Z_2, \cdots, Z_N$ and a binary associative operator Θ that operates on any two of these data items to produce another data item of the same type, a parallel scan (prefix) computes the N partial scans $S_1, S_2, \ldots, S_N$ defined by

$$S_i = Z_1 \Theta Z_2 \Theta Z_3 \ldots \Theta Z_i \qquad \text{(i)}$$

on $P \leq N$ processors. Denoting the complexity of the binary operation Θ by κ, the complexity

Of a parallel scan operation can be shown to be $O\left(\kappa \frac{N}{P} + \kappa \log P + (\tau + \mu\ell)\log P\right)$ in

which a permutation network is used as the model of parallel computation. In a permutation network, the cost of each round of communication is modelled as $\tau + \mu\ell$ and each processor is allowed to send and receive at most one message during a communication step. Here, τ is the start-up cost for a communication step, μ is the transfer bandwidth of the communication network and ℓ is the size of the largest message. Permutation networks closely model the behaviour of most multistage interconnection networks. The total scan S_N can be simultaneously computed on each processor along with the partial scans without incurring any additional communication overhead. Only the computation cost per processor is doubled. Thus,

the total cost of a parallel scan operation that computes both the total scan s_N as well as the partial scans of N data items on P processors is also $O\left(\kappa \frac{N}{P} + \kappa \log P + (\tau + \mu\ell)\log P\right)$.

3.2 RECURSIVE DOUBLING ALGORITHM (RDA) FOR (M, N) SYSTEM

3.2.1 Numerical Formulation

For a (M , N) block Tridiagonal system, let Li, Di and Ui denote the lower, main and upper diagonal blocks, respectively, in block row i, With $L_1 = U_N = I$ and $z_0 = z_{N+1} = 0$ at the boundaries, block row i ($1 \le i \le N$) can be written as:

$$z_{i+1} = - U_i^{-1} D_i z_i - U_i^{-1} L_i z_{i-1} + U_i^{-1} b_i \qquad \text{(ii)}$$

Which can be rewritten as

$$Z_{i+1} = B_i Z_i \qquad \text{(iii)}$$

where

$$Z_{i+1} = \begin{bmatrix} z_{i+1} \\ z_i \\ 1 \end{bmatrix}, \quad B_i = \begin{bmatrix} B_i^{11} & B_i^{12} & B_i^{13} \\ I & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and

$$\begin{cases} B_i^{11} = - U_i^{-1} D_i \\ B_i^{12} = - U_i^{-1} L_i \\ B_i^{13} = - U_i^{-1} b_i \end{cases} \qquad \text{(iv)}$$

The dimension of B_i is (2M + 1) × (2M + 1). Recursive substitution of Z_i yields:

$$Z_{i+1} = B_i Z_i = \cdots = B_i B_{i-1} B_{i-2} \cdots B_1 Z_1 = s_i Z_1 \qquad \text{(v)}$$

where

$$s_i = \begin{bmatrix} s_i^{11} & s_i^{12} & s_i^{13} \\ s_i^{21} & s_i^{21} & s_i^{23} \\ 0 & 0 & 1 \end{bmatrix} = B_i B_{i-1} B_{i-2} \cdots B_1 \qquad \text{(vi)}$$

Note that s_i is a partial prefix (matrix-matrix product) scan that can be computed in parallel using a parallel prefix scan using Eq^n (i). Also, the dimensions of blocks $s_i^{11}, s_i^{12}, s_i^{21}$ and s_i^{22} are M×M while those of s_i^{13} and s_i^{23} are M×1. To compute Z_i locally using the partial scans s_i, Z_1

has to be computed first and made locally available on each processor (see Eq^n (v)). Z_1 is computed on each processor using the boundary conditions $z_0 = z_{N+1} = 0$ as follows:

$$Z_{N+1} = \; S_i \; Z_1 \tag{vii}$$

$$\Rightarrow \begin{bmatrix} z_{N+1} \\ z_N \\ 1 \end{bmatrix} = \begin{bmatrix} s_N^{11} & s_N^{12} & s_N^{13} \\ s_N^{21} & s_N^{22} & s_N^{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_0 \\ 1 \end{bmatrix}$$

$$\Rightarrow z_{N+1} = s_N^{11} z_1 + s_N^{12} z_0 + s_N^{13} \tag{vii}$$

$$\Rightarrow z_1 = -[s_N^{11}]^{-1} s_N^{13}$$

This yields $Z_1 = [z_1 \; 0 \; 1]$ where **0** is a M×1 zero-vector. It follows from Eq^n (vii) that the total prefix product S_N is needed on each processor to compute Z_1 locally. As put forth in Remark 1, a parallel prefix matrix-matrix product computes both the partial as well as the total products on each processor, thus making S_N available on each processor. Algorithm 1 is an outline of the RDA.

Algorithm 3.1: Proposed RDA for Block Tridiagonal Systems

(i) Assign $\left\lceil \frac{N}{P} \right\rceil$ block rows to each processor.

(ii) For each Local block row i , calculate $U_i^{-1}, B_i^{11}, B_i^{12}$ and B_i^{13} using Eq^n (iv).

(iii) Each processor computes the Local prefix matrix products $s_1, s_2, \ldots, s_{\frac{N}{P}}$ in sequence manner.

(iv) On each processor *k,* initialize the partial parallel prefix product (PPP) $t = s^k_{\left\lceil \frac{N}{P} \right\rceil}$ and the Total product (TP) $T = s^k_{\left\lceil \frac{N}{P} \right\rceil}$.

(v) Do a parallel scan on the matrices t_k on processors $1 \le k \le P$ using matrix-matrix product as the binary, associative operator.

(vi) For each Local block row i , compute s_i using the partial prefix product *t* computed in the previous step and locally available matrices.

(vii) By scanning, assign $S_N = T$ (available) on each processor to calculate Z_1 using Eq^n (vii).

(viii) For each Local block row i, calculate Z_i using Eq^n (v).

3.2.2 Algorithmic Complexity

The algorithm executes in four stages, namely, (a) initialization (b) local serial prefix computation (c) non-local parallel prefix computation, and (d) finalization, as briefly described next.

(a) Initialization: In steps (i) and (ii), the block Tridiagonal system is partitioned amongst P processors and the B_i matrices are computed for the local block row indices. Based on Eq^n (iii), this involves matrix inversion of one block-sized, matrix-matrix multiplications of two block sized and one block size matrix-vector multiplication for each block row index incurring a total computation cost of less than $O\left(M^3 \frac{N}{P}\right)$.

(b) Local serial prefix computation: Computing the local prefix products on $\left\lceil \frac{N}{P} \right\rceil$ block rows incurs a total computation cost of less than $\left(M^3 \frac{N}{P}\right)$ in step (iii).

(c) Parallel Prefix Computation: Each processor keeps track of two matrices, t and T. The matrix t stores the partial matrix prefix product while the matrix T stores the total matrix product. Both are initialized to the total prefix product as computed sequentially on each processor computed in the previous step. This is shown in Fig. 3.1 using an example with 12 block rows and 4 processors. The parallel prefix operation in step (v) has $\log P$ stages. In each stage i, only the matrix T is exchanged between every pair of processors whose ranks vary in the bit position i . The matrix t is updated as $t \leftarrow t \Theta T$ only if T is received from a lower ranked processor. T_{loc} is always updated as $T_{loc} \leftarrow T_{loc} \Theta T_{read}$, where T_{loc} and T_{read} are the local and received copies of T, respectively. Note that though matrix-matrix product is binary and associative, it is not commutative. As such, care should be exercised to preserve the order of matrix-matrix multiplication of Eq^n (iv). In each step, at most two block-sized matrix-matrix products are computed incurring a cost of $O(M^3)$. In addition, one block-sized matrix T is communicated between unique sender-receiver pairs in each of the $\log P$ stages for a total communication cost of $O((\tau + \mu M^2)\log P)$. Therefore, the complexity of this nonlocal step is $O(M^3 \log P + (\tau + \mu M^2)\log P)$.

(d) Finalization: The local portion of the final solution is recovered using steps (vi) through step (viii) as shown in Fig. 3.1. It involves an inversion of a block-sized matrix and a block sized matrix-matrix multiplication for each local block row for a total cost of $\left(M^3 \frac{N}{P}\right)$.

- **Total cost:** Summing up the costs of the four stages, the total runtime of Proposed RDA to compute the solutions of an (*M, N*) system with *R* different right hand sides on *P* processors is:

$$T(R) = R.\left(M^3 \frac{N}{P} + M^3 \log P + (\tau + \mu M^2)\log P\right) \quad \text{(viii)}$$

Which can be re-written as:

$$T(R) = c_1 . R . f(M, N, P) \quad \text{(ix)}$$

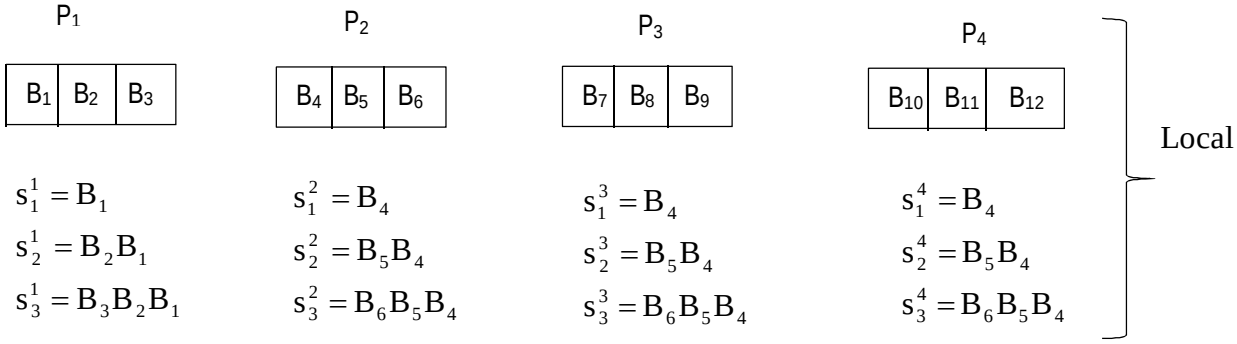
Where c_1 is some positive constant and

$$f(M, N, P) = M^3 \frac{N}{P} + M^3 \log P + M^2 \log P \qquad (x)$$

- **Performance:** Fig. 3.2 highlights the performance bottleneck of RDA-based block Tridiagonal solvers. The computing platform is described in Section V. The top curve of Fig. 3.2(a), which plots the total runtime as the number of processors is doubled, illustrates its poor strong-scaling characteristics. To understand this poor scalability, it is convenient to split the computations in the RDA algorithm into two types local and nonlocal computations (see Fig. 3.1). In the local computation phase, no inter-processor communications are involved. The non-local computation phase, on the other hand, is characterized by log P stages of communications akin to a parallel reduction operation. The ratio of time spent in the non-local phase to that spent in the local phase increases rapidly as the number of processors is increased for a fixed problem size, as shown in the lower curve of Fig. 3.2(a). This is because the non-local phase scales as $\approx M^3 \log P$ and adding more processors yields no improvements in the parallel runtime of the non-local computation phase, despite near linear speed-up in the local phases of the RDA (see Fig. 3.2(b)). This imbalance between the time spent in the local and non-local phases get further magnified as P → N and the number of right-hand sides increase (see Fig. 3.3). One of the primary motivations of the new algorithm presented next is to reduce this imbalance.

3.3 PROPOSED ACCELERATING RECURSIVE DOUBLING ALGORITHM

The accelerated recursive doubling algorithm presented here is based on a mathematical formulation that separates the overall computations into two phases, one that is independent of the right hand side **b**, which we call the independent phase, and another that depends on it, which we call the dependent phase. This style of execution is similar to the forward-backward two-phase execution pattern of CRA.



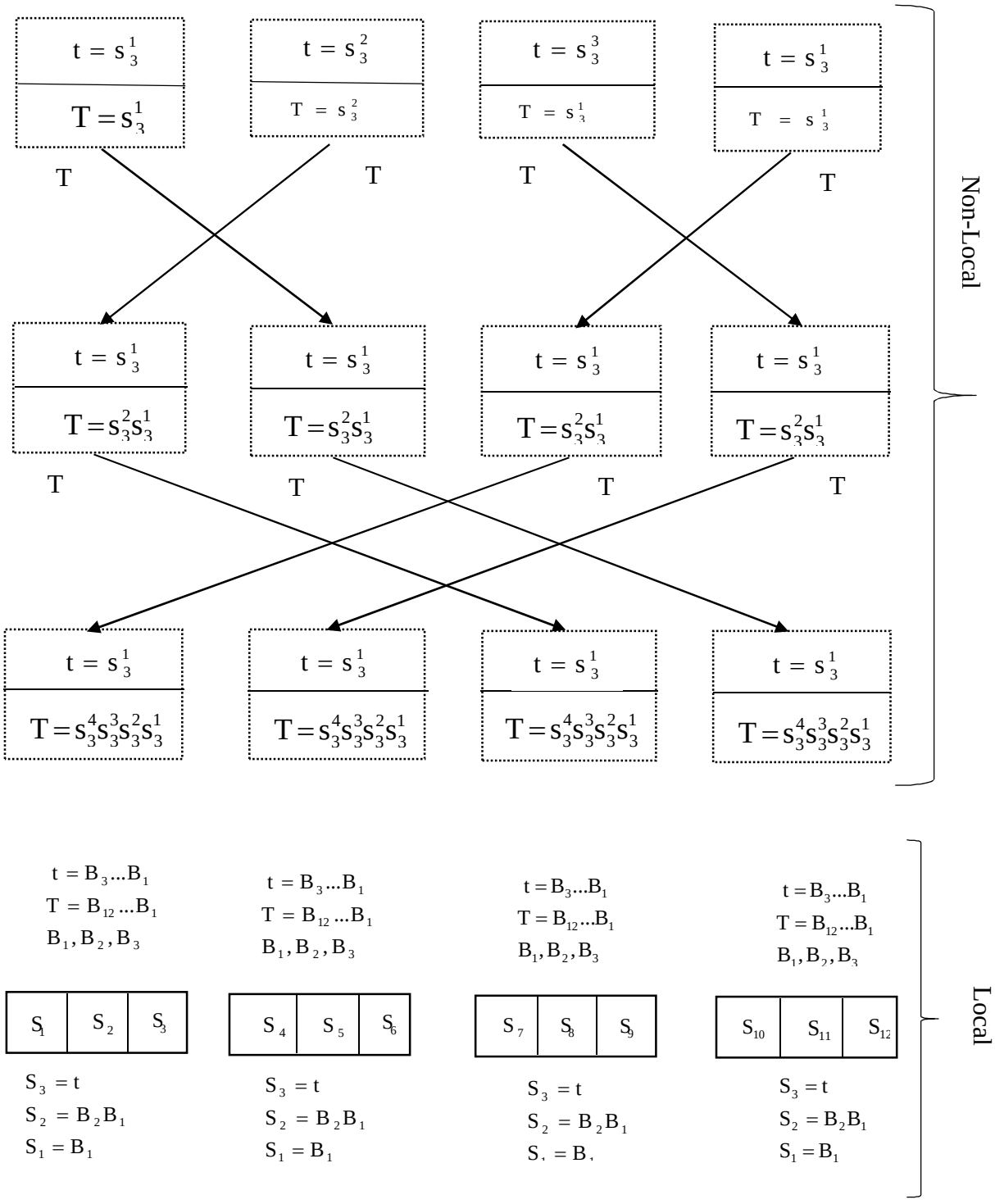


Figure 3.1 Parallel Proposed RDA on a Tridiagonal system with 12 block rows on 4 processors.

Described in Section I. The algorithmic complexity of the right hand side independent phase of computations will be shown to be larger than the second phase.
The proposed accelerated recursive doubling algorithm is presented next in three sub-parts. The first sub-part describes the numerical formulation of separating the original block Tridiagonal system of equations into the independent and dependent components. The second and third sub-parts describe the formulations of the independent and dependent phases, respectively. This new algorithm is ideal for computing solutions of block Tridiagonal systems with multiple right hand sides, especially for classes of problems for which it is computationally more efficient to use a recursive doubling algorithm than a cyclic reduction algorithm [15].

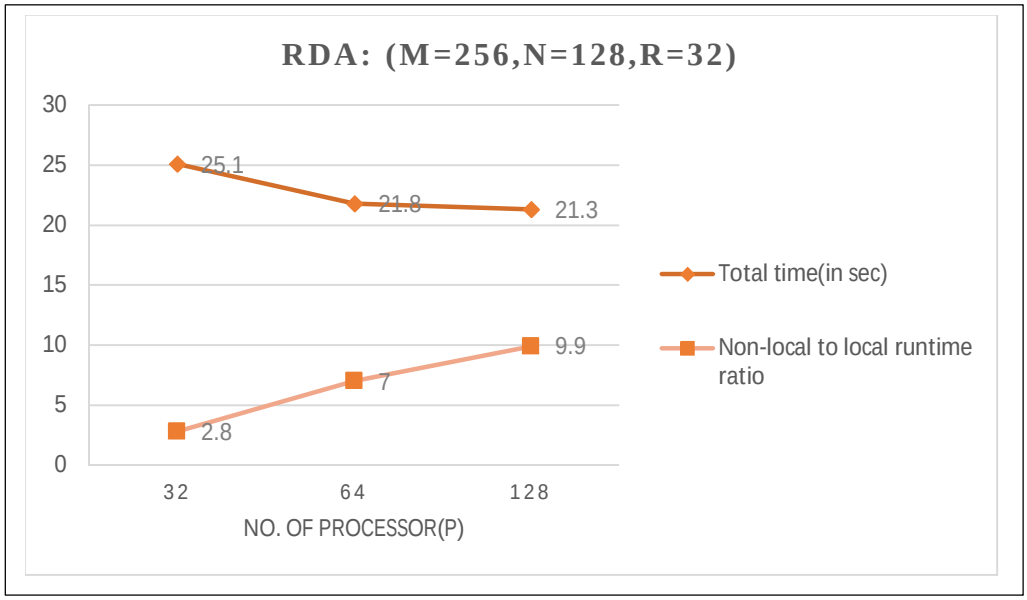


Figure 3.2 (a): Strong scaling behaviour of RDA is plotted in the orange (top) curve. The corresponding ratios of non-local to local runtimes are plotted in the blue (bottom) curve.

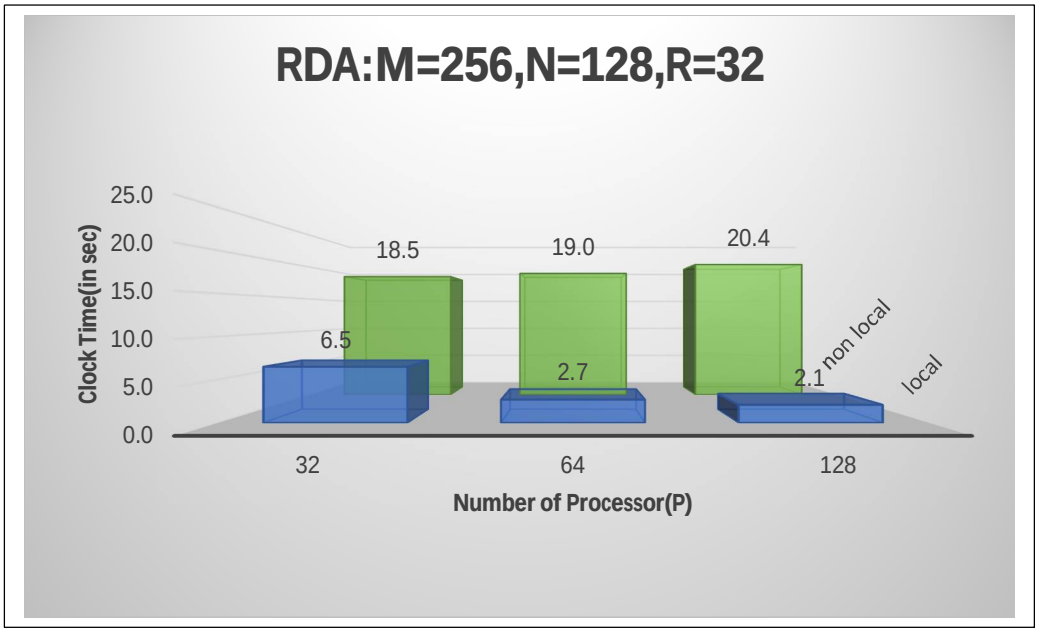


Figure 3.2 (b): Time spent in the local and non-local phases of the RDA when the number of right hand sides is increased.

3.3.1 Numerical Reformulation

Based on Eq^{n} (iv), it is apparent that the right hand side **b** enters the computation through the B_i^{13} component of each matrix B_i Guided by this observation, we decompose each B_i matrix as $B_i = C_i + F_i$
Where

$$C_i = \begin{bmatrix} B_i^{11} & B_i^{12} & 0 \\ I & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad F_i = \begin{bmatrix} 0 & 0 & B_i^{13} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \text{(xi)}$$

Note that the non-zero components B_i^{11}, B_i^{12} and I of B_i are matrices of dimensions M × M while the non-zero component B_i^{13} of F_i is only an M×1 vector. The following two results, presented without proofs, can be easily shown to hold true:

$$\begin{cases} F_i C_j = F_i \ , \forall \ 1 \le i, j \le N \\ F_i F_j = 0 \ , \forall \ 1 \le i, j \le N \end{cases} \qquad \text{(xii)}$$

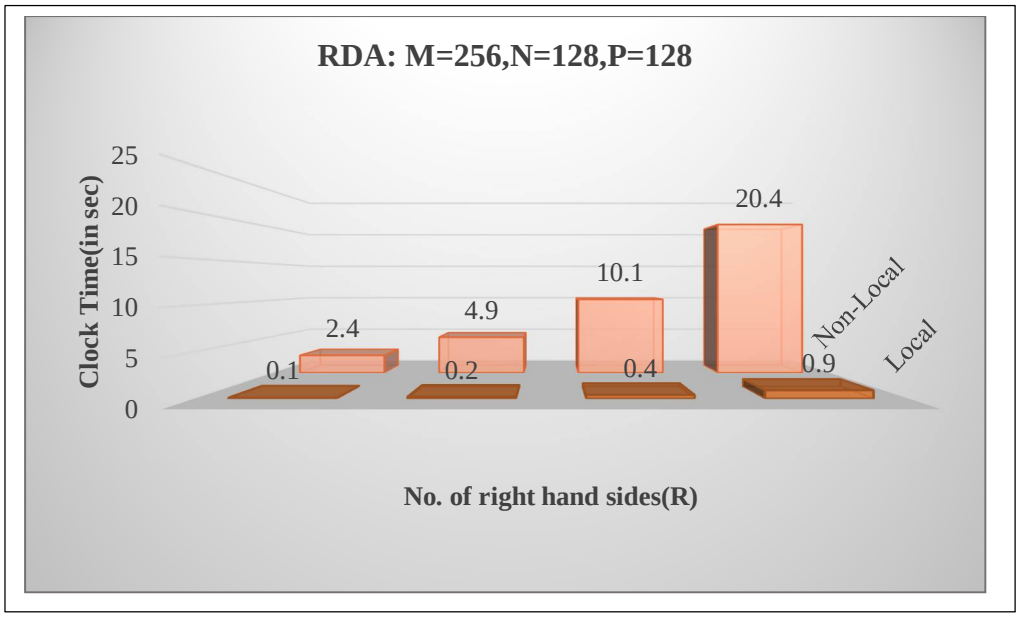


Figure 3.3: Local and Non-local runtimes of RDA with varying number of right hand sides

3.3.2 Right Hand Side Independence Phase

In order to divide the overall computations into a set of right hand side independent and a set of right hand side dependent computations, let a *modified partial product* be defined as:

$$Q_i = C_i C_{i-1} \ldots C_1 \qquad \text{(xii)}$$

Unlike the partial product s_i defined in Eq^n (vi), the modified partial product Q_i does not depend on the right hand side. As such, when computing the solutions for R different right hand sides, the partial products Q_i can be computed just once, stored and reused for each right hand side b^j for all $1 \leq j \leq R$.

- **Complexity:** The algorithm to compute the modified partial products is identical to steps (i) through (vi) of Algorithm 3.1.Therefore, the complexity of computing the modified partial products is

$$O\left(M^3 \frac{N}{P} + M^3 \log P + (\tau + \mu M^2) \log P \right)$$

3.3.3. Dependent Phase on Right Hand Side

Based on Eq^n (iv) and Eq^n (xi), it is clear that when the right hand side b_j, then $B_i = C_i$ implying that $s_i = Q_i$ for all $1 \leq i \leq N$ and the solution can be computed by executing the finalization steps (vii-viii) in Algorithm 3.1. When $b_j = 0$ contributions to the original partial sums s_i that are missing from the modified partial sums Q_i, due to their definitions (see Eq^n (xi) and Eq^n (xiv)), need to be consistently aggregated with Q_i to recover the original partial sums s_i^j, superscripted here on with the index j to indicate that it is evaluated with respect to the right hand side b_j. Let E_i^j denote this contribution on block row i from right hand side b_j .The proposed accelerated recursive doubling algorithm is guided by these observations and outlined in Algorithm 3.2 where $A(Q_i, E_i^j)$ denotes the aggregation function. If the complexity of computing the matrices E_i^j can be proven to be smaller than the complexity of computing the modified partial products Q_i the overall complexity of Algorithm 3.2.

Algorithm 3.2 Proposed Accelerated RDA

(i) Calculate and store Q_i using steps (i)-(vi) of Algorithm 3.1.
(ii) **for all** right hand side b^j do
(iii) Calculate E_i^j for each local block row *i* .
(iv) $s_i^j \leftarrow A(Q_i, E_i^j)$ For each local block row *i.*
(v) Calculate solution x^j using Eq^n (v).
(vi) **end for**

Can be shown to be smaller than that of Algorithm 3.1. To prove the smaller complexity, it is necessary to establish the following:

- Definitions of E_i^j and the function $A(Q_i, E_i^j)$.
- Complexity of computing E_i^j and $A(Q_i, E_i^j)$.

- Correctness of the aggregation function $A(Q_i, E_i^j)$.

Note (i): In the remainder of this section, the superscript j will be suppressed for ease of presentation with the understanding that all definitions and analyses hold true for each right hand side b^j.

▪ **Definition:** Let E_i be defined as follows:

$$E_i = \begin{cases} F_1, & i = 1 \\ Q_i Z_{i-1} + F_i, & i \geq 2 \end{cases} \qquad \text{(xiii)}$$

Where $Z_{i-1} = V_1 + V_2 + V_3 + \ldots + V_{i-1}$

and $V_i = Q_i^{-1} F_i$

and the aggregation function be defined as:

$$A(Q_i, E_i) = Q_i + E_i \qquad \text{(xiv)}$$

- **Complexity**: From the definitions above, it is clear that prior to computing the matrices E_i, matrices V_i and then Z_i need to be computed in that order. The steps to compute E_i are the following:

(i): For each local block row i, calculate V_i using Eq^n (xiii)
(ii): Do a parallel scan to calculate Z_i using Eq^n (xiii)
(iii): For each local block row i, calculate E_i using Eq^n (xiii).

In analysing the complexity of computing E_i, it is assumed that the modified partial products Q_i and their inverses Q_i^{-1} have already been computed and stored during the preceding right hand side independent phase.

Step i: By calculating V_i involves a block-sized matrix-matrix multiplication (see Eqn (xiii)) that requires complexity $O(M^3)$ work, it can be shown that V_i can be computed using $O(M^2)$ work as follows. Consider the following matrix-matrix product required to compute V_i.

$$V_i = Q_i^{-1} F_i = \begin{bmatrix} q_i^{11} & q_i^{12} & q_i^{13} \\ q_i^{21} & q_i^{22} & q_i^{23} \\ q_i^{31} & q_i^{32} & q_i^{33} \end{bmatrix} \begin{bmatrix} 0 & 0 & B_i^{13} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 0 & q_i^{11} B_i^{13} \\ 0 & 0 & q_i^{21} B_i^{13} \\ 0 & 0 & q_i^{31} B_i^{13} \end{bmatrix} \equiv \begin{bmatrix} q_i^{11} B_i^{13} \\ q_i^{21} B_i^{13} \\ q_i^{31} B_i^{13} \end{bmatrix} \qquad \text{(xv)}$$

Note that $q_i^{11} B_i^{13}$ and $q_i^{21} B_i^{13}$ are both matrix-vector multiplications, each costing $O(M^2)$ and $q_i^{11} B_i^{13}$ is a vector-vector multiplication that costs O(M) computations. Thus, due to the block

structure of D_i defined through Eq^n (xi), the computation of V_i requires $O(M^2)$ computation, even if Q_i^{-1} is dense, implying that the real cost of computing V_i for all local block rows is only $O\left(M^2 \frac{N}{P}\right)$ and not $O\left(M^3 \frac{N}{P}\right)$.

Step ii: It is clear from step (i) that each V_i can be implemented as a vector of length 2M + 1, as shown in Eqn (xv). Note that is a prefix sum as shown in Eq^n (xiii).
Since Z_i is a prefix sum over the vector forms of V_i (see Eq^n (xv)), they can be computed using steps (i)-(vi) of the parallel scan algorithm as described in Algorithm 3.1 with appropriate modifications to reflect that the binary, associate operator is a vector addition of length 2M + 1 and the message size that is communicated in the non-local phase is of size *M*. The complexity of this step is, therefore, $O\left(M \frac{N}{P} + M \log P + (\tau + \mu M)\log P\right)$.

Step iii: In this step, each matrix-matrix multiplication $Q_i Z_j$ is of the form:

$$Q_i Z_j = \begin{bmatrix} q_i^{11} & q_i^{12} & q_i^{13} \\ q_i^{21} & q_i^{22} & q_i^{23} \\ q_i^{31} & q_i^{32} & q_i^{33} \end{bmatrix} \begin{bmatrix} 0 & 0 & Z_j^{13} \\ 0 & 0 & Z_j^{23} \\ 0 & 0 & Z_j^{33} \end{bmatrix}$$
$$= \begin{bmatrix} 0 & 0 & q_i^{11} Z_j^{13} + q_i^{12} Z_j^{23} + q_i^{13} Z_j^{33} \\ 0 & 0 & q_i^{21} Z_j^{13} + q_i^{22} Z_j^{23} + q_i^{23} Z_j^{33} \\ 0 & 0 & q_i^{31} Z_j^{13} + q_i^{32} Z_j^{23} + q_i^{33} Z_j^{33} \end{bmatrix} \quad \text{(xvi)}$$

Since Z_i^{13} and Z_i^{23} are vectors with dimensions M ×1 and Z_i^{33} is a number, each non-zero sub-matrix of the product $Q_i Z_j$ is a sum of three vectors of length M of which the first two are computed using a matrix-vector product of an M × M matrix with a vector of length M. This is shown in Eqn (xvi). Therefore, computation of the product $Q_i Z_j$ in Eqn (xvi) requires $O\left(M^2 \frac{N}{P}\right)$ work when aggregated over all local block-rows. Since the aggregation function $A(Q_i, E_i^j)$ is a simple sum of two $O(M^2)$ sixed matrices, step (iv) of Algorithm 3.2 requires $O(M^2)$work. A matrix-vector multiplication that requires $O(M^2)$work computes the solution in step (v). Therefore, the cost of aggregating the contribution from R right hand sides is

$$R.O\left(M^2 \frac{N}{P} + M \log P + (\tau + \mu M)\log P\right)$$

3.3.4 Correctness: The correctness is proven using following lemma:

Lemma 1: For all $1 \le i \le N$

$$S_i = Q_i + E_i \quad \text{(xvii)}$$

Proof: The base case with $i = 1$ is trivially true from the definitions in Eq^n (iv), Eq^n (xii) and Eq^n (xiii). Let the inductive hypothesis be true for $i = k - 1$. Then:

$$\begin{aligned}
S_k &= B_k S_{k-1} = (C_k + F_k)(Q_{k-1} + E_{k-1}) \\
&= C_k Q_{k-1} + C_k E_{k-1} + F_k Q_{k-1} + F_k E_{k-1} \\
&= Q_k + C_k (Q_{k-1} Z_{k-2} + F_{k-1}) + F_k + F_k E_{k-1} \\
&\quad \{\text{using the inductive hypothesis}\} \\
&= Q_k + (Q_k Z_{k-2} + F_k) + C_k F_{k-1} + F_k (Q_{k-1} Z_{k-2} + F_{k-1})
\end{aligned}$$

$$\begin{aligned}
&= Q_k + (Q_k Z_{k-1} + F_k) - Q_k V_{k-1} + C_k F_{k-1} + F_k Q_{k-1} Z_{k-2} \quad \{\text{using Eqn (xii)}\} \\
&= Q_k + E_k - Q_k Q_{k-1}^{-1} F_{k-1} + C_k F_{k-1} + F_k Q_{k-1} Z_{k-2} \quad \{\text{using Eqn (xiii)}\} \\
&= Q_k + E_k - C_k F_{k-1} + C_k F_{k-1} + F_k Q_{k-1} Z_{k-2} \\
&= Q_k + E_k + F_k Q_{k-1} Z_{k-2} \\
&= Q_k + E_k + F_k Q_{k-1} [Q_1^{-1} F_1 + Q_2^{-1} F_2 + \ldots + Q_{k-2}^{-1} F_{k-2}] \\
&= Q_k + E_k + F_k [C_{k-1} \ldots C_2 F_1 + C_{k-1} \ldots + C_3 F_2 + \ldots + C_{k-1}^{-1} F_{k-2}] \\
&= Q_k + E_k + [F_k F_1 + F_k F_2 + \ldots + F_k F_{k-2}] \quad \{\text{using equation (xii)}\} \\
&= Q_k + E_k \quad \{\text{using Eqn (xii)}\} \\
&\text{which proves the lemma.}
\end{aligned} \tag{xviii}$$

3.3.5 Total Cost

Step (i) of Algorithm 3.2 executes steps (i)-(vi) of Algorithm 3.1 $O\left(M^3 \frac{N}{P} + M^3 \log P + (\tau + \mu M^2) \log P \right)$ using work. Summing up the individual costs of steps (iii)-(v) of Algorithm 3.2, it follows that the work required to add the contribution from each right hand side to the modified partial products Q_i is $O\left(M^3 \frac{N}{P} + M^3 \log P + (\tau + \mu M^2) \log P \right)$ work. Therefore, the complexity of proposed ARDA to solve an (*M*,*N*) block Tridiagonal system with *R* right hand sides on *P* processors is:

$$\begin{aligned}
T'(R) &= O(M^3 \frac{N}{P} + M^3 \log P + (\tau + \mu M^2) \log P) + \\
&\quad R.O(M^2 \frac{N}{P} + M \log P + (\tau + \mu M) \log P) \\
&= c_2 f(M, N, P) + c_3 \frac{R}{M} f(M, N, P)
\end{aligned} \tag{xix}$$

with positive constants c_2 and c_3 and f defined by Eqn (x).

3.4 RELATIVE SPEEDUP

We define a performance metric, called relative speedup $S(R)$, to compare the relative performances of the original and the accelerated algorithms. Using Eqn (ix) and Eqn (xix), the relative speedup of the Proposed ARDA with respect to the RDA for a (M, N) block Tridiagonal system with R right hand sides is defined as $S(R)=\frac{T(R)}{T'(R)}$. It follows that:

$$S(R)=\frac{T(R)}{T'(R)}=\frac{c_1 RM}{c_2 M+c_3 R}=\frac{1}{c_1'/R+c_2'/M} \qquad \text{(xx)}$$

where $c_1'=c_2/c_1$ and $c_2'=c_3/c_1$ are positive constants. Therefore, as $M \to \infty$, $S(R) = O(R)$.

3.5 RESULTS

To validate these results, both algorithms were implemented and executed on a Cray XT5 machine with 2 twelve-core AMD MagnyCours 2.1 GHz processors per node and 32 GB of memory in each compute node. The nodes are connected via a high-bandwidth Cray Gemini interconnect. We study the performance improvements due to the new algorithm based on how the relative speedup and scalability vary with respect to the number of right hand sides and the block sixe.

3.5.1 Relative speedup

Fig. 3.4 (a) shows the speed-up $S(R)$ (see Eq^n (xx)) when solving for a block Tridiagonal system with block-sixe M =256 and number of block rows N = 128 with changing number of right hand sides using P = 128 processors. The relative speedup grows rapidly with increasing number of right hand sides. This can be understood from the observation that as the number of right hand sides increase, the time spent in the local computing phases (see Fig. 3.1) remains nearly the same in both the RDA and Proposed ARDA algorithms (since the granularity N/P remains the same) while the net gain in the time spent in the non-local phase keeps adding with increasing number of right hand sides. This is shown in Fig. 3.4 (b). Compare this with Fig. 3.3. The dramatic reduction in the runtimes of the non-local phase of the new algorithm is a result of two important modifications to the RDA, namely:

- The binary, associative operator in the non-local phase executed for each right hand side in the RDA is a matrix-matrix product that requires $O(M^2)$ work while it is a vector addition that requires $O(M)$ in the Proposed ARDA.

- The message sixes exchanged reduces from $O(M^2)$ in the RDA to O(M) in the Proposed ARDA.

3.5.2 Scalability

Strong-scaling results are presented in Fig.3.5. As the number of processors increase, the amount of computations in the local phases, which scales as $\sim \frac{N}{P}$ in both algorithms, decreases and becomes comparable. As such, the runtime difference between the two algorithms is influenced primarily by the computations in the non-local phase, which scales as $\sim RM^3 \log P$

in the RDA and $\sim RM \log P$ in the Proposed ARDA for R right hand sides, an improvement that only gets magnified with increasing block sixe as seen in Fig. 3.5.

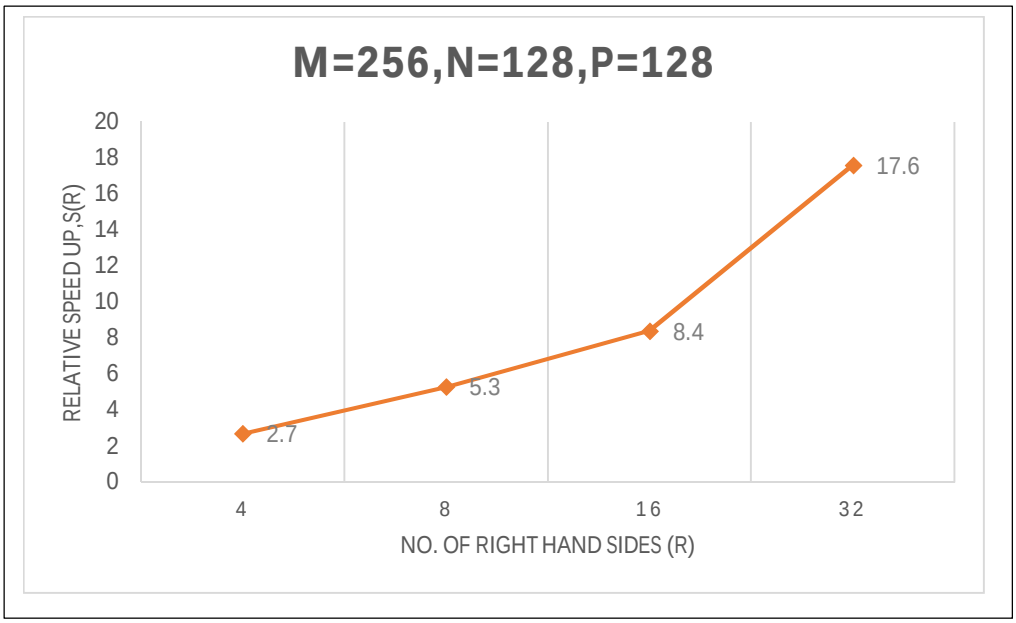


Figure 3.4 (a) Relative speedup

Figure 3.6 shows the effect of changing the granularity and the block sixe on the relative speedup for different numbers of right hand sides on $P = 32$. Consider a column group in any one of the plots, say, the column group for $R = 32$ of Fig.3.6 (a). As the block sixe, M, is doubled, the relative speedup, $S(R)$, increases to a maximum and then starts decreasing. This behaviour holds true for all column groups in Fig. 3.6 and can be understood from the fact that as the block size increases for fixed N, P and R, the time spent in the local phase of the computation which scales as $\sim M^3$ in both algorithms dominates and offsets the runtime gains from the non-local computations.

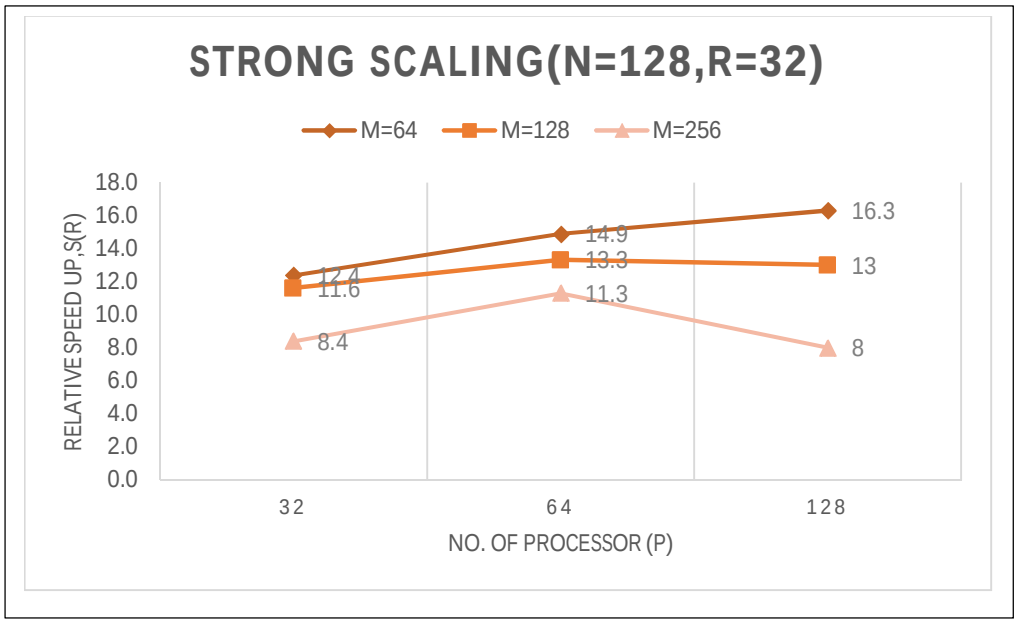


Figure 3.5 Strong scaling speedup with N=128 and R=32 for three different block sizes.

Now, consider the column group for $R = 32$ for all three granularities shown in Fig. 3.6. Clearly, the relative speedup increases as the granularity $\frac{N}{P} \rightarrow 1$ in all cases. This is the strong-scaling effect shown in Fig. 3.5. To summarize, the effect of granularity is that it controls the ratio of time spent in the local and non-local phases in both algorithms. When the time spent in the local phase

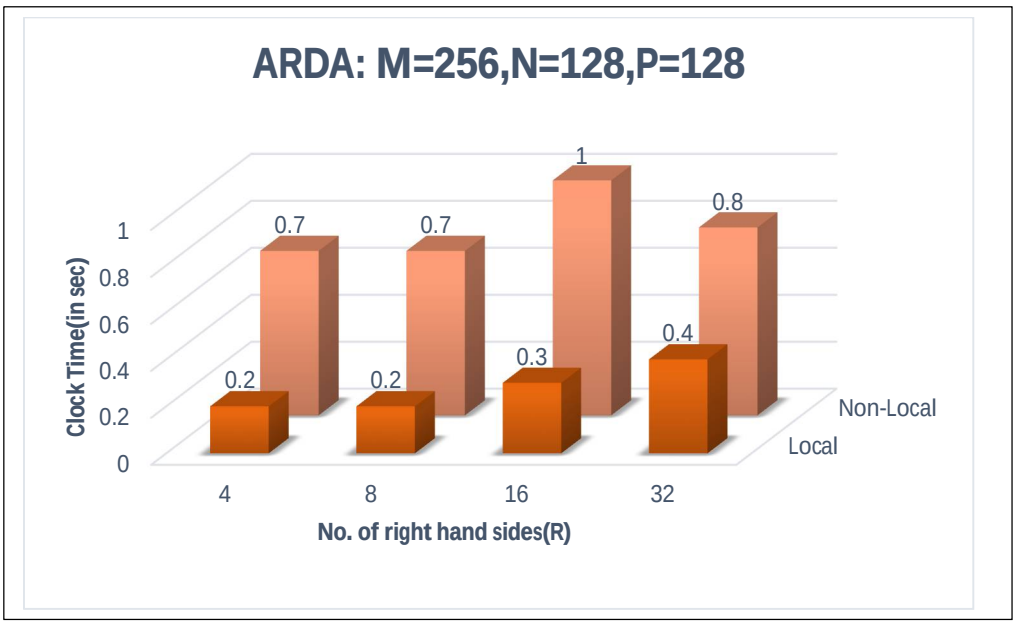


Figure 3.4 (b) Local and non-local computation

Dominates (large granularity), the new algorithm performs at least as well as the original algorithm and the relative speedup tends to unity. On the other hand, when the granularity of the problem tends to unity, the time spent within the non-local phase increases. The larger the time spent in the non-local phase, the greater the relative speedup and the new algorithm delivers vastly superior performance (compare Fig. 3.3 and Fig. 3.4(b)).

It is very important to understand that the relative speedup defined in Eq^{n} (xx) is valid *even for* $P = 1$. However, for computational problems of relevance to today's breed of large-scale computing, both N and P are typically very large. For such problems, the relative speedup $S(R)$ delivered by the new accelerated algorithm proposed is significant.

Numerical solutions obtained by both algorithms mutually agreed in all cases, though not every solution was stable. A detailed investigation of the numerical stabilities of the two algorithms is outside the scope of this study, which is focussed exclusively on their parallel runtime performance and scalability issues. To make RDA and Proposed ARDA numerically stable for the same classes of Tridiagonal matrices for which cyclic reduction algorithms are stable,

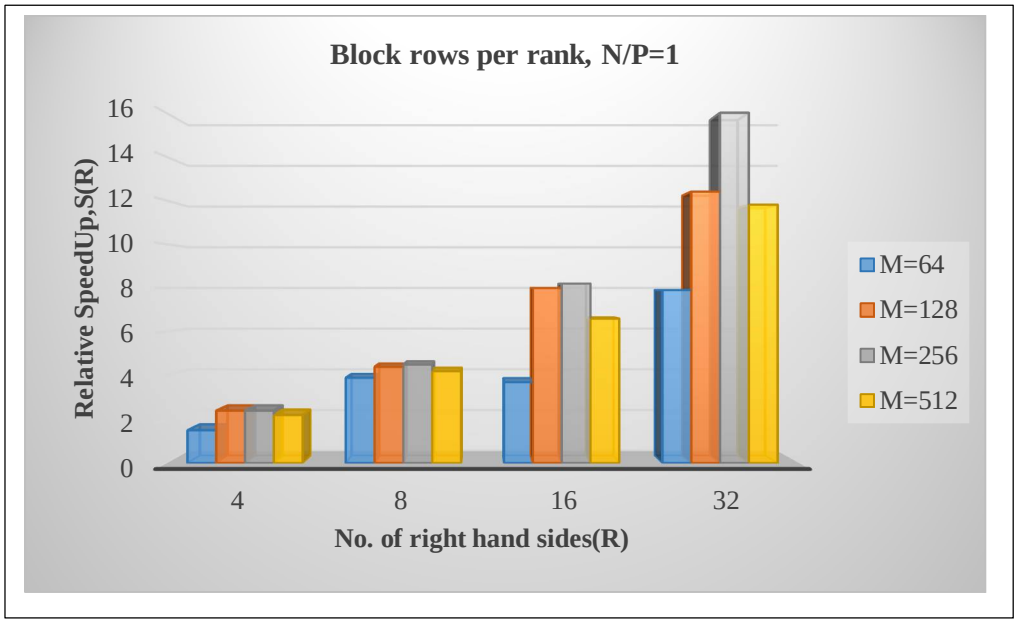


Figure 3.6 (a) Granularity N/P=1

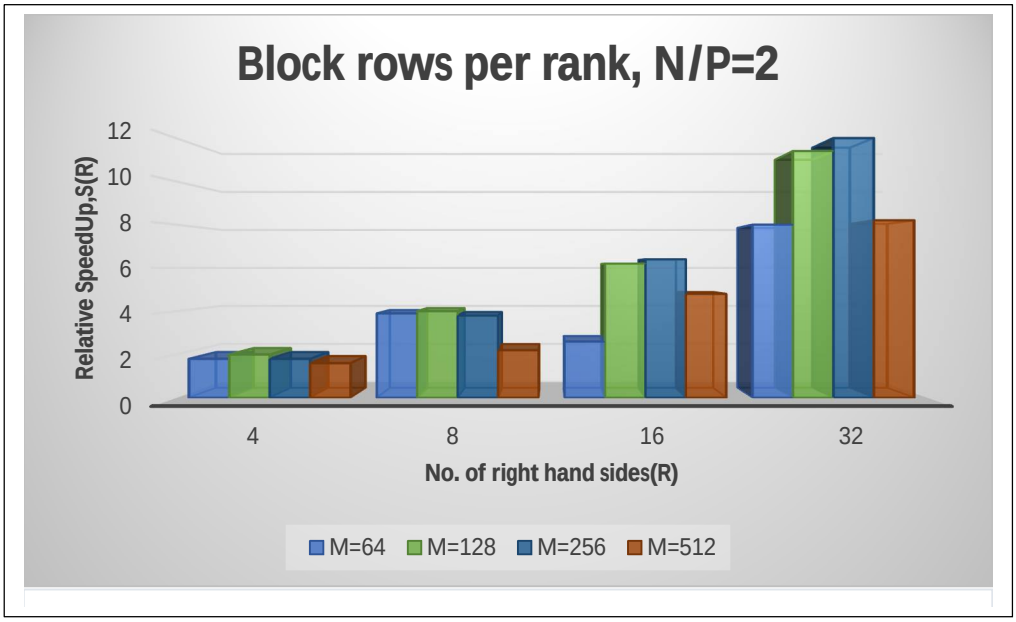


Figure 3.6 (b) Granularity N/P=2

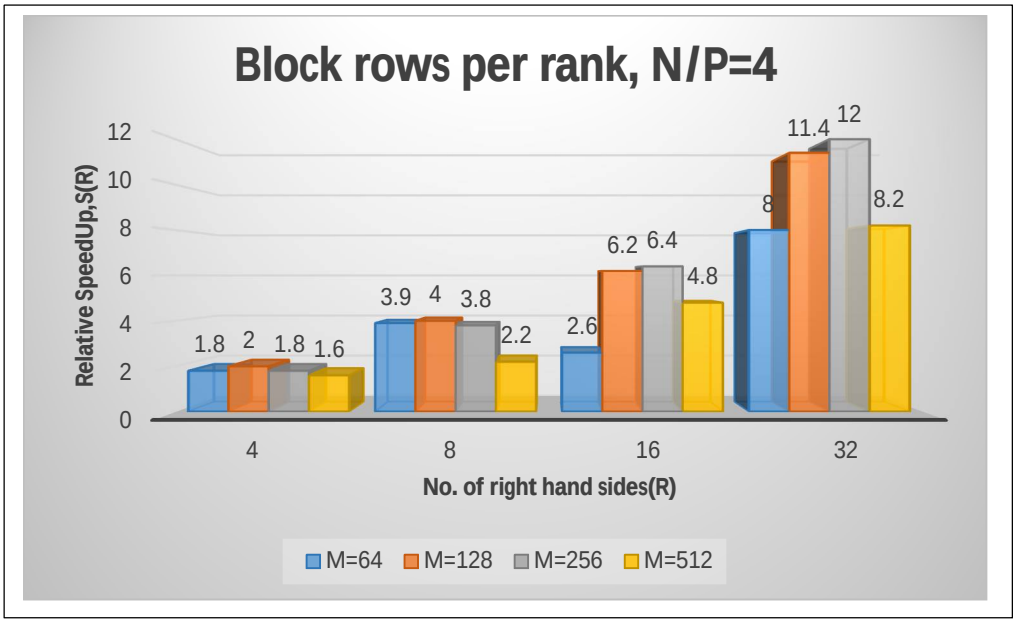


Figure 3.6 (c) Granularity N/P=4

Figure 3.6 Relative speedup of ARDA with respect to the RDA with varying M,N and R on P=32

an LU-decomposition of the matrix A is computed and the resulting equation A**x** = LU**x** = **b** is computed in two steps [29]. In the first step, the system L**y** = **b** is solved, where **y** = U**x**. In the second, the system U**x** = **y** is solved. Each of these steps can be cast into the same prefix computation based algorithm as the one presented here. As a result, these pre-processing steps do not add to the overall complexity of either the original (RDA) or the Proposed ARDA algorithms presented in this chapter. However, it can change the overall constant in both.

Chapter 4

A Proposed Parallel Algorithm for solution of 3D Hyperbolic PDE

Consider the 3D second order hyperbolic equations defined

in the region $\Omega = \{(x,y,z,t) \mid 0 < x,y,z < 1, t > 0\}$ of the following form as

$$\frac{\partial^2 U}{\partial t^2} + 2\alpha \frac{\partial U}{\partial t} + \beta^2 U = \frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} + \frac{\partial^2 U}{\partial z^2} + F(x,y,z,t) \tag{i}$$

Where $\alpha(x,y,t) > 0,\ \beta(x,y,t) \geq 0$.

The initial condition consist of

$$U(x,y,z,0) = f_1(x,y,z);\ U_1(x,y,z,0) = f_2(x,y,z) \tag{ii}$$

and the boundary conditions consist of

$$\begin{cases} U(0,y,z,t) = g_1(y,z,t);\ U(1,y,z,t) = g_2(y,z,t) \\ U(x,0,z,t) = g_3(x,z,t);\ U(x,1,z,t) = g_4(x,z,t) \\ U(x,y,0,t) = g_5(x,y,t);\ U(x,y,1,t) = g_6(x,y,t) \end{cases} \tag{iii}$$

This equation is commonly encountered in physics and engineering mathematics such as vibration of structures and signal analysis. In recent years, various numerical schemes have been developed for solving one, two and three dimensional hyperbolic equation [30-39]. The scheme is proven to require lesser execution time than the others explicit group methods [40]. As an extension to these works, Kew and Ali [41, 42] presented the utilization of domain decomposition techniques on explicit group methods and parallelized it using OpenMP programming environment. The method is unconditionally stable and applicable to singular problem. In this chapter, we present a new explicit group relaxation method derived from the standard seven-point difference approximation for the solution of (i). This explicit group method is developed using small fixed size group strategy which require lesser execution times than the classic point iterative method. The method is then parallelized using OpenMP environment with the utilization of domain decomposition technique. In the next section, a brief overview will be given on the formulation of explicit group method for the three dimensional telegraph equations. The parallelization using domain decomposition technique under OpenMP programming environment will be discussed in Section 3. Section 4 presented the numerical experiments and the results. Finally, concluding remarks are given in Section 5.

4.1 METHOD FORMULATION

In solving problem (i) using finite difference approximations, we let the spatial domain, Ω be discretised uniformly in x-,y- and z- directions with mesh size $h = \Delta x = \Delta y = \Delta z = 1/n$ where n is an arbitrary positive integer. The grid points are given by ,

$(x_i, y_i, z_i, t_m) \equiv (ih, jh, lh, mk)$ Where m=1,2,3….. and k>0 be the time steps. Let $U^m_{i,j,l}$ be the exact solution of the different equation and $u^m_{i,j,l}$ be the computed solution of the approximation method at the grid point (1) can be approximated by various finite difference schemes. One commonly formula is the standard seven point difference approximation

$$
\begin{aligned}
&-\left(r/2\right)\left(u_{i+1,j,l,m+1} + u_{i-1,j,l,m+1}\right) + \left(1 + 3r + a + b/2\right)u_{i,j,l,m+1} - \left(r/2\right)\left(u_{i,j,l,m+1} + u_{i,j-1,l,m+1}\right) \\
&-\left(r/2\right)\left(u_{i,j,l+1,m+1} + u_{i,j,l-1,m+1}\right) \\
&= \left(r/2\right)\left(u_{i+1,j,l,m} + u_{i-1,j,l,m}\right) + \left(r/2\right)\left(u_{i,j+1,l,m} + u_{i,j-1,l,m}\right) + \left(r/2\right)\left(u_{i,j,l+1,m} + u_{i,j,l-1,m}\right) \qquad \text{(iv)} \\
&+ \left(2 - 3r - b/2\right)u_{i,j,l,m} + (a-1)u_{i,j,l,m-1} + \Delta t^2 F_{i,j,l,m+1/2}
\end{aligned}
$$

Where $r = \Delta t^2 / h^2 \; ; a = \alpha \Delta t \; ; b = \beta^2 \Delta t^2$

The iterations for this standard centred seven-point difference scheme are generated at any time level on all grid point using (2) until convergence is achieved before proceeding to the next time level. The process continues until the desired time level is reached. Consider the standard seven-point formula (ii) which was derived from the centred finite difference discretisation. The mesh points are grouped in cubes of eight points (Fig. 4.1) and applying (ii) to each of these points will produce the (8x8) systems of equations in the form

$$
\begin{bmatrix}
k_1 & -k_2 & 0 & -k_2 & -k_2 & 0 & 0 & 0 \\
-k_2 & k_1 & -k_2 & 0 & 0 & -k_2 & 0 & 0 \\
0 & -k_2 & k_1 & -k_2 & 0 & 0 & -k_2 & 0 \\
-k_2 & 0 & -k_2 & k_1 & 0 & 0 & 0 & -k_2 \\
-k_2 & 0 & 0 & 0 & k_1 & -k_2 & 0 & -k_2 \\
0 & -k_2 & 0 & 0 & -k_2 & k_1 & -k_2 & 0 \\
0 & 0 & -k_2 & 0 & 0 & -k_2 & k_1 & -k_2 \\
0 & 0 & 0 & -k_2 & -k_2 & 0 & -k_2 & k_1
\end{bmatrix}
\begin{bmatrix}
u_{i,j,l,m+1} \\ u_{i+1,j,l,m+1} \\ u_{i+1,j+1,l,m+1} \\ u_{i,j+1,l,m+1} \\ u_{i,j,l+1,m+1} \\ u_{i+1,j,l+1,m+1} \\ u_{i+1,j+1,l+1,m+1} \\ u_{i,j+1,l+1,m+1}
\end{bmatrix}
=
\begin{bmatrix}
rhs_{i,j,l} \\ rhs_{i+1,j,l} \\ rhs_{i+1,j+1,l} \\ rhs_{i,j+1,l} \\ rhs_{i,j,l+1} \\ rhs_{i+1,j,l+1} \\ rhs_{i+1,j+1,l+1} \\ rhs_{i,j+,l+1}
\end{bmatrix}
\qquad \text{(v)}
$$

where $k_1 = 1 + 3r + a + b/2 \; ; k_2 = r/2 \; ; k_3 = 2 - 3r - b/2 \; ; k_4 = a - 1$

$$rhs_{i,j,l} = k_2(u_{i-1,j,l,m+1} + u_{i,j-1,l,m+1} + u_{i,j,l-1,m+1} + u_{i-1,j,l,m} + u_{i+1,j,l,m} + u_{i,j-1,l,m} + u_{i,j+1,l,m} + u_{i,j,l+1,m}) + k_3 u_{i,j,l,m} + k_4 u_{i,j,l,m-1} + \Delta t^2 F_{i,j,l,m+1/2}$$

$$rhs_{i+1,j,l} = k_2(u_{i+2,j,l,m+1} + u_{i+1,j-1,l,m+1} + u_{i+1,j,l-1,m+1} + u_{i,j,l,m} + u_{i+2,j,l,m} + u_{i+1,j-1,l,m} + u_{i+1,j+1,l,m} + u_{i+1,j,l+1,m}) + k_3 u_{i+1,j,l,m} + k_4 u_{i+1,j,l,m-1} + \Delta t^2 F_{i+1,j,l,m+1/2}$$

$$rhs_{i+1,j+1,l} = k_2(u_{i+2,j+1,l,m+1} + u_{i+1,j+2,l,m+1} + u_{i+1,j+1,l-1,m+1} + u_{i,j+1,l,m} + u_{i+2,j+1,l,m} + u_{i+1,j,l,m} + u_{i+1,j+2,l,m} + u_{i+1,j+1,l-1,m} + u_{i+1,j+1,l+1,m}) + k_3 u_{i+1,j+1,l,m} + k_4 u_{i+1,j+1,l,m-1} + \Delta t^2 F_{i+1,j+1,l,m+1/2}$$

$$rhs_{i+1,j+1,l} = k_2(u_{i-1,j+1,l,m+1} + u_{i,j+2,l,m+1} + u_{i,j+1,l-1,m+1} + u_{i-1,j+1,l,m} + u_{i+1,j+1,l,m} + u_{i,j,l,m} + u_{i,j+2,l,m} + u_{i,j+1,l-1,m} + u_{i,j+1,l+1,m}) + k_3 u_{i,j+1,l,m} + k_4 u_{i,j+1,l,m-1} + \Delta t^2 F_{i,j+1,l,m+1/2}$$

$$rhs_{i,j,l+1} = k_2(u_{i-1,j,l+1,m+1} + u_{i,j-1,l+1,m+1} + u_{i,j,l+2,m+1} + u_{i-1,j,l+1,m} + u_{i+1,j,l+1,m} + u_{i,j-1,l+1,m} + u_{i,j+1,l+1,m} + u_{i,j,l,m} + u_{i,j,l+2,m}) + k_3 u_{i,j,l+1,m} + k_4 u_{i,j,l+1,m-1} + \Delta t^2 F_{i,j,l+1,m+1/2}$$

$$rhs_{i+1,j,l+1} = k_2(u_{i+2,j,l+1,m+1} + u_{i+1,j-1,l+1,m+1} + u_{i+1,j,l+2,m+1} + u_{i,j,l+1,m} + u_{i+2,j,l+1,m} + u_{i+1,j,l,m} + u_{i+1,j-1,l+1,m} + u_{i+1,j+1,l+1,m} + u_{i+1,j,l+2,m}) + k_3 u_{i+1,j,l+1,m} + k_4 u_{i+1,j,l+1,m-1} + \Delta t^2 F_{i+1,j,l+1,m+1/2}$$

$$rhs_{i+1,j+1,l+1} = k_2(u_{i+2,j+1,l+1,m+1} + u_{i+1,j+2,l+1,m+1} + u_{i+1,j+1,l+2,m+1} + u_{i,j+1,l+1,m} + u_{i+2,j+1,l+1,m} + u_{i+1,j,l+1,m} + u_{i+1,j+2,l+1,m} + u_{i+1,j+1,l,m} + u_{i+1,j+1,l+2,m}) + k_3 u_{i+1,j+1,l+1,m} + k_4 u_{i+1,j+1,l+1,m-1} + \Delta t^2 F_{i+1,j+1,l+1,m+1/2}$$

$$rhs_{i,j+1,l+1} = k_2(u_{i-1,j+1,l+1,m+1} + u_{i,j+2,l+1,m+1} + u_{i,j+1,l+2,m+1} + u_{i-1,j+1,l+1,m} + u_{i+1,j+1,l+1,m} + u_{i,j,l+1,m} + u_{i,j+2,l+1,m} + u_{i,j+1,l,m} + u_{i,j+1,l+2,m}) + k_3 u_{i,j+1,l+1,m} + k_4 u_{i,j+1,l+1,m-1} + \Delta t^2 F_{i,j+1,l+1,m+1/2}$$

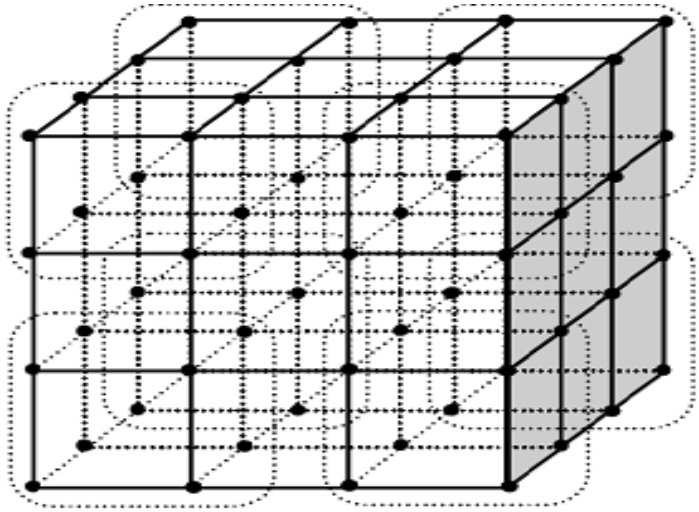


Figure 1: Computational molecule for Explicit Group

This matrix (3) can be inverted to produce an eight points explicit group (EG) equation

$$\begin{bmatrix} u_{i,j,l,m+1} \\ u_{i+1,j,l,m+1} \\ u_{i+1,j+1,l,m+1} \\ u_{i,j+1,l,m+1} \\ u_{i,j,l+1,m+1} \\ u_{i+1,j,l+1,m+1} \\ u_{i+1,j+1,l+1,m+1} \\ u_{i,j+1,l+1,m+1} \end{bmatrix} = A \begin{bmatrix} m_1 & m_2 & m_3 & m_2 & m_2 & m_3 & m_4 & m_3 \\ m_2 & m_1 & m_2 & m_3 & m_3 & m_2 & m_3 & m_4 \\ m_3 & m_2 & m_1 & m_2 & m_4 & m_3 & m_2 & m_3 \\ m_2 & m_3 & m_2 & m_1 & m_3 & m_4 & m_3 & m_2 \\ m_2 & m_3 & m_4 & m_3 & m_1 & m_2 & m_3 & m_2 \\ m_3 & m_2 & m_3 & m_4 & m_2 & m_1 & m_2 & m_3 \\ m_4 & m_3 & m_2 & m_3 & m_3 & m_2 & m_1 & m_2 \\ m_3 & m_4 & m_1 & m_2 & m_2 & m_3 & m_2 & m_1 \end{bmatrix} \begin{bmatrix} rhs_{i,j,l} \\ rhs_{i+1,j,l} \\ rhs_{i+1,j+1,l} \\ rhs_{i,j+1,l} \\ rhs_{i,j,l+1} \\ rhs_{i+1,j,l+1} \\ rhs_{i+1,j+1,l+1} \\ rhs_{i,j+1,l+1} \end{bmatrix} \qquad \text{(vi)}$$

Where $A = 1/\left(k_1^4 - 10 * k_2^2 * k_1^2 + 9 * k_2^4\right)$

$m_1 = k_1^3 - 7 * k_1 * k_1^2; m_2 = k_1^2 * k_2 - 3 * k_2^3; m_3 = 2 * k_2^2 * k_1; m_4 = 6 * k_2^3$

The iterations are generated on these groups of eight mesh points and it is treated explicitly similar to the way where the single point is treated in the point iterative method. Similarly, the process is repeated until the desired time level is achieved.

4.2 PROPOSED DECOMPOSITION TECHNIQUE

Most domain decomposition methods (DDM) have been developed for solving elliptic [43, 44] parabolic [45, 46] and hyperbolic problems [41, 42]. They have been considered as very efficient methods for solving partial differential equations on parallel computers [46]. They can be classified into two classes; overlapping and non-overlapping methods with respect to the decomposition of the domain. In [41, 42], Kew and Ali have demonstrated the use of DDM for the explicit group methods by using the overlapping subdomain and Schwarz alternating procedure (SAP). This SAP operates between two overlapping sub-domains; solving the Dirichlet problem on one sub-domain in each iteration by taking the boundary conditions based on the most recent solution obtained from the other sub-domain. The details of the SAP can be obtained in [47]. In order to implement this domain decomposition algorithm, ordering strategies need to be considered for each finite difference discretization scheme due to the shared boundaries between sub-domains [41, 42]. The solution domain is decomposed into blocks as shown in Fig. 4.2. Referring to Fig. 4.2, when the point 1 in 1 Ω is computing, points 2 – 7 at the same time level needs to be used if (3) is used. However, points 1 – 6 are from subdomain 1 Ω while point 7 is from sub-domain 2 Ω . In the case of parallelization, the sub-domains 1 Ω and 2 Ω are computed concurrently. There is a possibility that the solutions at the points 7 is updating on the respective sub-domains when the point 1 are being computed. This may cause inaccuracy in the numerical results. Thus, we need to organize the ordering strategies to prevent any conflict on the usage of points among sub-domains. With this in mind, a red black group ordering strategy is introduced to this EG scheme. The algorithm of this scheme is presented in Algorithm 4.1. The same concept of proposed domain decomposition ordering strategy can also be implemented for the standard centred seven-point scheme. Algorithm 4.2 presents the syntax of implementing the program on multiple-core processor. It is observed that Steps (vii) – (xiv) in Algorithm 4.1 is the most expensive part of the algorithm and therefore stands to gain the most advantage from the parallelization process.

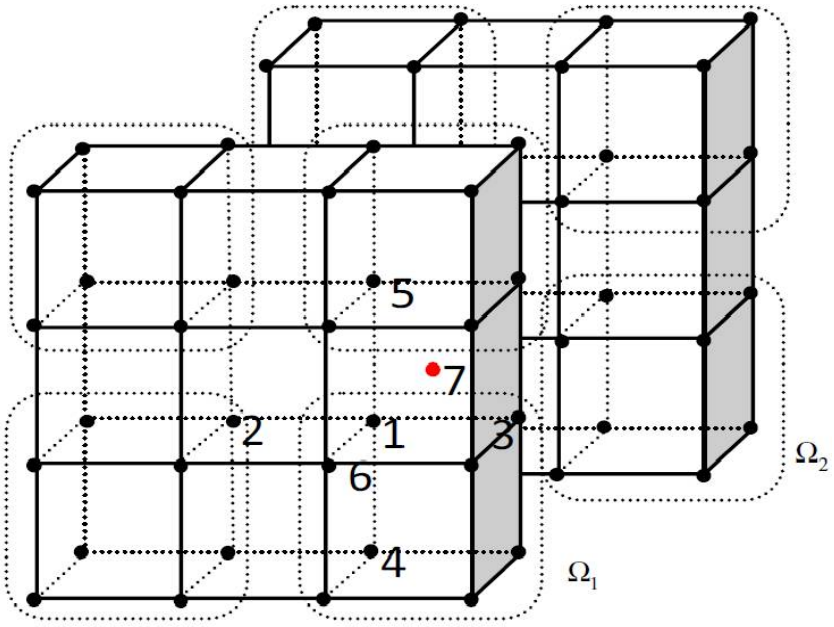


Figure 4.2 Proposed Explicit group scheme method

TABLE 4.1
ALGORITHM 4.1: Algorithm for Proposed Explicit Group Method

(i) Choose an initial guess s to the solution
(ii) For each time step:
    a. Firstly set a boundary condition
    b. Up to convergence level, do(in global):
    c. Recognize the boundaries values of subdomain
    d. Up to convergence level , do(in global):
(iii) For each subdomain:
    a. Solve at the black group points
    b. End do
(iv) For each subdomain:
    a. Solve the red group points
    b. End do
(v) Check the local convergence test
    a. End do
(vi) Check the local convergence test
    a. End do

TABLE 4.2

ALGORITHM 4.2. Line of Code for Implementing the Program on Multicore Processor

```
#include<omp.h>
void main()
{    int num_threads;
omp_set_num_threads(omp_num_procs());
#pragma omp parallel for
{
compute the points in each sub-domain
```

4.3 **NUMERICAL EXPERIMENTS AND RESULTS**

In order to demonstrate the viability of the proposed method in solving the three dimensional second order hyperbolic equation (i), experiments were carried out on a quad core i7 CPU 2.0 GHz, 4GB of RAM with Window 7 operating system using Microsoft Visual Studio 2010. This experiment is to solve the hyperbolic problem (i) with the analytical solution [34]

$$u(x, y, z, t) = \exp(-t) \times \sinh\ x \times \sinh\ y \times \sinh\ z \tag{vii}$$

$$f(x, y, z, t) = (\beta^2 - 2\alpha - 2) \times \exp(t) \times \sinh\ x \times \sinh\ y \times \sinh\ z \tag{viii}$$

The boundary and initial conditions can be obtained from the analytical solution. The proposed group method is a three level scheme. The starting values of $u(x, y, z)$ at the first time level need to be obtained before any computation starts. The values may be obtained using the Taylor series expansion

$$u^1_{i,j,l} = u^0_{i,j,l} + k(u_t)u^0_{i,j,l} + \left(k^2/2\right)(u_{tt})u^0_{i,j,l} + O(k^3) \tag{ix}$$

Where u and t , u are known explicitly at $t = 0$ .The values of relaxation factor for the various mesh sizes are set equal to 1.0. The convergence criteria used throughout the experiment was the l_∞ norm with the local and global error tolerances were set equal to 10^{-6} and 10^{-7} , respectively. Throughout the computation, the values of $\alpha = 10.0$ and $\beta = 5.0$. The RMS errors are tabulated at $T = 2$ for a fixed $\lambda = k / h = 3.2$ for several mesh sizes of 16, 32, 64 and 128 and are listed in Table 4.3. The speedup is used to measure the performance of the parallel algorithms compared to the corresponding sequential algorithms. The speedup formula used is in the form of

$$\text{speedup} = \frac{\text{Execution time for a single thread}(t_1)}{\text{Execution time using 4 threads}(T_4)}$$

It can be observed that the computational results obtained from Proposed explicit group method(EG) maintained the same degree of accuracies with the standard point method. The Proposed EG method requires lesser computing times compared to point method due to its lower computational complexity. As shown in Table III, the execution times of the parallel EG can be saved up to about 34% compared to the sequential EG and 39% for the standard centred seven-point method for the mesh size of 128. The percentages vary for difference schemes.

4.4 EXPERIMENTAL RESULTS:

Table 4.3:

	Non Parallel (1 Thread)					Parallel (4 Thread)			
	h^{-1}	Iter	RMS Error	Elapsed Time		Iter	RMS Error	Elapsed Time	Speed-up
Standard Point Method	16	44	6.55E-04	0.226		44	6.55E-04	0.221	1.023
	32	66	3.17E-04	5.573		66	3.17E-04	4.683	1.19
	64	87	1.58E-04	155.341		87	1.57E-04	118.445	1.312
	128	99	8.34E-05	4536.632		99	8.35E-05	2771.018	1.637
Explicit Group Method	16	25	6.56E-04	0.141		25	6.56E-04	0.137	1.029
	32	37	3.16E-04	2.95		32	3.16E-04	2.615	1.128
	64	49	1.55E-04	77.821		49	1.55E-04	62.095	1.253
	128	55	7.97E-05	1988.437		55	7.96E-05	1312.725	1.515

Chapter 5

CONCLUSION

Solving block Tridiagonal linear systems with multiple right hand sides R arise in a wide range of scientific applications. In this thesis, we presented a parallel, accelerated recursive doubling algorithm that delivers $O(R)$ speedup improvement over the original recursive doubling algorithm. Since the number of different right hand sides is typically very large, this speedup translates to significant overall performance improvements in practice. The numerical formulation, its algorithmic complexity as well as the performance advantage of the new algorithm are discussed in detail. Numerical stability of both algorithms in the context of block Tridiagonal systems is the subject of ongoing investigation. To our knowledge, the proposed accelerated recursive doubling algorithm, has not been reported before in the literature. In this thesis, the parallel implementation of a new explicit group relaxation method, derived from the standard focused seven-point difference formula has been presented in solving the 3D telegraph equations. The parallel implementation utilizes the proposed domain decomposition technique on the discretized solution domain using OpenMP programming environment. For comparison purposes, we also include the RMS error and the execution timings of the point-wise scheme; the standard centred seven-point method. It can also be observed that the parallel algorithms manage to save up approximately 33% of the computational costs compared to their sequential algorithms. The explicit group relaxation method is able to take advantage from parallelism implemented on multi-core technology environment. Research on other explicit group method of the same class like the Explicit Decoupled Group and its variants are under investigation and will be reported soon.

REFERENCES

[1] Nakamura, Shoichiro , "Applied Numerical Methods In C", United State of America PTR Prentice Hall Inc, 1993,

[2] Smith G.D, Numerical Solution of Partial Differential Equations, Oxford Universities Press, 1965.

[3] Bhat B, Rama, Chakraverty, Snehasnish, United Kingdom Numerical Analysis in Engineering, Alpha Science International Ltd, 2004.

[4] Norma Alias, Md. Rajibul Islam, Nur Syazana Rosly "A Dynamic PDE Solver for Breasts' Cancerous Cell Visualization on Distributed Parallel Computing Systems", in Proc. The 8th International Conference on Advances in Computer Science and Engineering (ACSE 2009), Phuket, Thailand, Mar. 16-18, 2009.

[5] Evans D.J, Sukon K.S, "The Alternating Group Explicit (AGE) Iterative Method for Variable coefficient Parabolic Equations", Intern. J. Computer Math. Vol 59, pp 107-121. 1995.

[6] Smith, G.D, United Kingdom, "Numerical Solution of Partial Differential Equations: Finite Difference Methods", Oxford Universities Press, 1985.

[7] Zhang Y. , Cohen J. , and Owens J. D. , "Fast Tridiagonal Solvers on the GPU", in Procs. of the ACM Symposium on Principles and Practice of Parallel Programming, vol. 45, no. 5, pp. 127–136, 2010.

[8] Hirschman S. P. ,Perumalla K. S. , Lynch V. E. , and Sanchez R. , "BCYCLIC: A Parallel Block Tridiagonal Matrix Cyclic Solver", Journal of Computational Physics, vol. 229, pp. 6392–6404, 2010.

[9] Hirshman S. P. , Sanchez R. , and Cook C. R. , "SIESTA: A Scalable Iterative Equilibrium Solver for Toroidal Applications", Physics of Plasmas, vol. 18, p. 062504, 2011.

[10] Alias, N., Sahimi, M.S., and Abdullah, A.R., "The AGEB Algorithm for Solving the Heat Equation in Two Space Dimensions and Its Parallelization on a Distributed Memory Machine", Proceedings of the 10th European PVM/ MPI User's Group Meeting: Recent Advances In Parallel Virtual Machine and Message Passing Interface, Vol. 7, pp. 214–221, 2003.

[11] Alias N. , Sahimi M.S. , Abdullah A.R. , "Parallel Strategies for the Iterative Alternating Decomposition Explicit Interpolation-Conjugate Gradient Method In solving Heat Conductor Equation on a Distributed Parallel Computer Systems", Proceedings of the 3rd International Conference on Numerical Analysis in Engineering. 3: 31-38. 2003.

[12] Norma Alias, Rosdiana Shahril, Md. Rajibul Islam, Noriza Satam, Roziha Darwis, "3D parallel algorithm parabolic equation for simulation of the laser glass cutting using parallel computing platform", The Pacific Rim Applications and Grid Middleware Assembly (PRAGMA15), Penang, Malaysia. Oct 21-24, 2008.

[13] Seal S. K. , Perumalla K. P., and Hirshman S. P. , "Scaling the SIESTA Magneto hydro dynamics Equilibrium Code", Concurrency and Computation: Practice and Experience, vol. 25, no. 15, pp. 2207–2223, 2013.

[14] Leighton F. T. , "An Introduction to Parallel Algorithms and Architectures: Arrays, Trees and Hypercube", Morgan Kaufmann Publishers, 1992.

[15] Seal S. K. , Perumalla K. P. , and Hirshman S. P. , "Revisiting Parallel Cyclic Reduction and Parallel Prefix-based Algorithms for Block Tridiagonal Systems of Equations", Journal of Parallel and Distributed Computing, vol. 73, pp. 273–280, 2013.

[16] Harris M. , Sengupta S., and Owens J. D. , "GPU Gems 3. Addison-Wesley Professional", ch. Parallel Prefix Sum (Scan) with CUDA, 2007.

[17] Thomas L. H. , "Elliptic Problems In Linear Difference Equations Over A Network ," Watson Sci. Comput. Lab. Rep., Columbia University, 1949.

[18] Hockney R. W. , “A Fast Direct Solution Of Poisson’s Equation Using Fourier Analysis”, Journal of the ACM, vol. 12, no. 1, pp. 95–113, 1965.

[19] Heller D. , “Some Aspects Of The Cyclic Reduction Algorithm For Tridiagonal Linear System”, SIAM Journal of Numerical Analysis, vol. 13, no. 4, pp. 484–496, 1976.

[20] Stone H. S. , “An Efficient Parallel Algorithm For The Solution Of A Tridiagonal Linear System of Equations”, Journal of the ACM, vol. 20, no. 1, pp. 27–38, 1973.

[21] Wang H. H. , “A Parallel Method For Tridiagonal Equations”, ACM Trans. on Mathematical Software, vol. 7, no. 2, pp. 170–183, 1981.

[22] Sun X. H. , Zhang H. , and Ni L. M. , “Efficient Tridiagonal Solvers on Multicomputer”, IEEE Trans. on Computers, vol. 41, no. 3, pp. 286–296, 1992.

[23] Sun X. H. and Zhang W. , “A Parallel Two-level Hybrid Method for Tridiagonal Systems and its Application to Fast Poisson Solvers”, IEEE Trans. on Parallel and Distributed Systems, vol. 15, no. 2, pp. 97–106, 2004.

[24] Bini D. A. and Meini B. , “The Cyclic Reduction Algorithm: From Poisson Equation To Stochastic Processes And Beyond”, Numerical Algorithms, vol. 51, no. 1, pp. 23–60, 2008.

[25] Zhang Y. , Cohen J. , and Owens J. D. , “Fast Tridiagonal Solvers on the GPU”, in Procs. of Principles and Practice of Parallel Programming, 2010.

[26] Sengupta S. , Harris M. , Zhang Y. , and Owens J. D. , “Scan primitives for GPU computing”, in Procs. of the ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware, pp. 97–106, 2007.

[27] Egecioglu O. , Koc C. K. , and Laub A. J. , “A Recursive Doubling Algorithm for Solution of Tridiagonal Systems on Hypercubes Multiprocessors”, Journal of Computational and Applied Mathematics, vol. 27, pp. 95 – 108, 1989.

[28] Grama A. ,Gupta A. , Karypis G. , and Kumar V. , “Introduction to Parallel Computing”, Addison Wesley, 2003.

[29] Kincaid D. and Cheney W. , “Numerical Analysis: Mathematics of Scientific Computing”, Brooks/Cole Publishing Company, 1996.

[30] Ali N.H.M. and Kew L.M. , “New explicit group iterative methods in the solution of two dimensional hyperbolic equations”, Journal of Computational Physics 231, pp.6953–6968.

[31] Mohanty R.K. , “New unconditionally stable difference scheme for the solution of multi-dimensional telegraphic equations”, International Journal of Computer Mathematics 86 (12), pp.2061–2071,2009.

[32] Dehghan M. , Shokri A. , “A mesh less method for numerical solution of a linear hyperbolic equation with variable coefficients in two space dimensions”, Numerical Methods for Partial Differential Equations 25, pp.494–506, 2008.

[33] Gao F. , Chi C. , “Unconditionally stable difference schemes for a one space-dimensional linear hyperbolic equation”, Applied Mathematics and Computation 187, pp.1272–1276, 2007.

[34] Mohanty R.K. , “An operator splitting technique for an unconditionally stable difference method for a linear three space dimensional hyperbolic equation with variable coefficients”, Applied Mathematics and Computation 162, pp.549–557, 2005.

[35] Mohanty R.K. , “An operator splitting method for an unconditionally stable difference scheme for a linear hyperbolic equation with variable coefficients in two space dimensions”, Applied Mathematics and Computation 152, pp.799–806, 2004.

[36] Mohanty R.K. , “An unconditionally stable difference scheme for the one-space-dimensional linear hyperbolic equation”, Applied Mathematics Letters 17, pp.101–105, 2004.

[37] Mohanty R.K. , Jain M.K. , Arora U. , “An unconditionally stable ADI method for the linear hyperbolic equation in three space dimensions”, International Journal of Computer Mathematics 79, pp.133–142, 2002.

[38] Evans D.J. , “UK Group Explicit Methods for the Numerical Solution of Partial Differential Equations”, Loughborough University of Technology, Gordon and Breach Science Publisher, The Netherlands, 1997.

[39] Evans D.J. , “Group explicit methods for the numerical solution of first-order hyperbolic problems in one dependent variable”, International Journal of Computer Mathematics 56 (3), pp.245–252, 1995.

[40] Kew L.M. , Ali N.H.M. , “Explicit group iterative methods for the solution of telegraph equations”, in: The 2010 International Conference of Applied and Engineering Mathematics World Congress on Engineering 2010 , London, UK, Lecture Notes In Engineering and Computer Science, pp. 1770–1775, 30 Jun–2 July, 2010.

[41] Kew L.M. , Ali N.H.M. , “Parallel Explicit Group Domain Decomposition Methods for the Telegraph Equation”, in: International Conference on Applied Mathematics and Engineering Mathematics , World Academy of Science, Engineering and Technology 60, Phuket, Thailand, 21-23 December 2011.

[42] Kew L.M. , Ali N.H.M. , “OpenMP Technology In the Parallelization Of New Hyperbolic Group Solver,” in: 12th WSEAS International Conference on Applied Computer Science (WSEAS 2012), Singapore, Latest Advances in Information Science and Applications, pp. 136-141, 11-13 May 2012.

[43] Dryja M. , and Widlund O.B. , “Some Domain Decomposition Algorithms for Elliptic Problems”, in: L. Hayes, D. Kincaid (Eds.), Iterative Methods for Large Linear Systems, Academic Press, San Diego, CA. 1989.

[44] Cai X.-C. , and Widlund O.B. , “Domain Decomposition Algorithms for Indefinite Elliptic Problems”, SIAM J. Sci. Statist., pp. 243–258, 1992.

[45] Dawson C.N. , Du Q. , and Dupont T.F. ,“A Finite Difference Domain Decomposition Algorithm for Numerical Solution of the Heat Equation”, Mathematics of Computation, 57(195) , pp. 63-71, 1991.

[46] Jun, Y. and Mai, T.Z., “IPIC Domain Decomposition Algorithm for Parabolic Problems”, Applied Mathematics and Computation, 177, pp. 352-364, 2006.

[47] Saad, Y. “ Iterative Methods for Sparse Linear Systems”, 2nd Edition. pp. 382-421.

[48] P. Amodio, and L. Brugnano. "Parallel factorizations and parallel solvers for tridiagonal linear systems." Linear algebra and its applications vol 172 1992, pp 347-364.

[49] R. W. Hockney, and J. W. Eastwood. "Computer simulation using particles, 1988." Hilger, Bristol.

[50] Roger W Hockney., and Chris R. Jesshope. “Parallel Computers 2: architecture, programming and algorithms” Vol. 2. CRC Press, pp 1988.

[51] Nathan Mattor, Timothy J. Williams, and Dennis W. Hewett. "Algorithm for solving tridiagonal matrix problems in parallel." Parallel Computing, vol 21, no. 11, pp 1769-1782, 1995.

[52] Wang H. H. "A parallel method for tridiagonal equations", ACM Transactions on Mathematical Software (TOMS) 7, vol 2 ,pp 170-183, 1981.

[53] Press William H. , Flannery Brian P. , Teukolsky Saul A. , and Vetterling William T. “Numerical Recipes: The art of scientific computing”, Vol. 2. London: Cambridge University Press, 1987.

[54] Conte S. D, “A stable implicit finite difference approximation to a fourth order parabolic equation”, J. Assoc.Comp. Mach.,vol 4, 18– 23, 1957.

[55] Fairweather G. and Gourlay A. R. “Some stable difference approximations to a fourth order parabolic partial differential equations”, Maths. Comput., vol 2, pp. 1–11, 1967.

[56] Mohanty R. K. , “A fourth order finite difference method for the general one dimensional nonlinear biharmonic problems of first kind”, J. Comp. Appl. Math., pp. 114, 275–290, 2000.

[57] Mohanty R.K. , “An implicit high accuracy variable mesh scheme for 1-D nonlinear singular parabolic partial differential equations”, Appl. Math. Comput. , pp. 186, 219–229, 2007.

[58] Jain M. K. ,Jain R. K. , and Mohanty R. K. , “Fourth order deference method for three dimensional elliptic equations with non-linear first derivative terms”, Numerical Methods of Partial Diff Eq^n 8 , 575–591, 1992.

[59] Krishnaiah U. A. , Manohar R. P. and Stephenson J.W. , “Fourth order finite difference methods for three dimensional general linear elliptic problems with variable coefficients”, Numerical Methods Partial Diff Eq^n 3 , 229–240, 1987