Analysis and Implementation of Asynchronous Finite Difference Scheme for Advection Diffusion Equation

A THESIS

submitted by

KUMAR SAURABH
(MA14M004)

in partial fulfillment of the requirement
for the award of the degree of

MASTER OF TECHNOLOGY
in
INDUSTRIAL MATHEMATICS AND SCIENTIFIC COMPUTING

A DAAD-IIT MASTER SANDWICH PROJECT

DEPARTMENT OF MATHEMATICS, IIT MADRAS, INDIA

&

MATHCCES, DEPARTMENT OF MATHEMATICS, RWTH AACHEN, GERMANY

MAY 2016

DECLARATION

This is to declare that the thesis entitled **"Analysis and Implementation of Asynchronous Finite Difference Scheme for Advection Diffusion Equation"** is the result of investigations carried out by me at Indian Institute of Technology, Madras, India and RWTH Aachen, Germany under the supervision of Prof. Dr. Martin Frank and Prof. S. Sundar. In keeping with the general practice of reporting scientific observations, due acknowledgement has been made whenever the work described is based on the findings of other investigations. Any omission which might have occured by oversight or error in judgement is regretted.

Saurabh

Chennai - 600 036
May 2016

(Kumar Saurabh)
(MA14M004)

THESIS CERTIFICATE

This is to certify that the thesis titled **"Analysis and Implementation of Asynchronous Finite Difference Scheme for Advection Diffusion Equation"**, submitted by **Kumar Saurabh**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Technology**, is a bonafide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. S. Sundar
Project Guide
Professor
Dept. of Mathematics
IIT-Madras, 600 036
India

Dr. S. Sundar
Professor
Department of Mathematics
Indian Institute of Technology Madras
Chennai - 600 036, INDIA.

Prof. Dr. Martin Frank
Project Guide
Professor
MATH-CCES, Dept. of Mathematics
RWTH Aachen, 52062
Germany

Date: 03 May 2016

ACKNOWLEDGEMENT

I cannot help mentioning those people who have had great influence on me both intellectually and physically while pursuing my Masters Degree in Industrial Mathematics and Scientific Computing. I admit that without their great support and help, I would not finish this journey to obtain my Masters degree. First of all, I am very grateful to Prof. Dr. Martin Frank who accepted me as a DAAD exchange student and guided me during my research work. His brilliant ideas have always inspired me whenever I could not find a way to look at the problems in broader vision. I would also like to thank Prof. S. Sundar for his great inspirations and support which made me purely concentrate on the research problems as a graduate student.

I would also like to thank from the bottom of my heart to Thomas Camminady for his guidance and brilliant ideas that helped me to complete my work within the limited span of time. I would like to thank Philipp Otte for sharing his expertise on High Performance Computing and other members of Math CCES, RWTH Aachen along with International Office, RWTH Aachen, Jigar Parekh, Viany Patil and Bhupesh Verma who made my whole stay in Germany feel like " home away from home." I would like to acknowledge DAAD for providing the funds to carry out the Research work, RWTH-IT center for providing the computing resources and PPCES to make the video lectures on parallel computing available to all.

ABSTRACT

This thesis addresses the problems related to the Synchronization in context with the parallel computers. The numerical solution of partial differential equations in large-scale problems might require the usage of parallel computing. In this context, the computational domain is often split into smaller sub-domains that are handled by separate processors. Due to the non-local nature of the underlying problem, communication as well as synchronization between these processing elements (PEs) is required. Finite Difference is one of such techniques to solve the PDEs. Since the discretization by finite- difference schemes results in mostly simple arithmetic operations, the overhead due to communication and synchronization might become a bottleneck. Current trends in massively parallel computing systems suggest that the number of processors will continue to grow over time, which in turn, increases the overhead associated with communication and synchronization. Simulation at extreme levels of parallelism will then require an elimination, or at least a tight control of these overheads. While stability is conserved when these schemes are used asynchronously, accuracy is greatly degraded. Message arrivals at PEs are essentially random processes, so is the behavior of the error. Within a statistical framework the average errors drop always to first-order regardless of the statistics of delay. A new scheme is proposed to robust the asynchrony. The analytical results are compared against numerical simulations. The main purpose of the thesis is the implementation of the Asynchronous Scheme on parallel computers and study the error-time trade off.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Many natural and engineered systems and processes can be accurately modelled by partial differential equations (PDEs). In a number of real applications, however, due to the complexity of the equations themselves as well as the geometrical aspects of the problem, analytical solutions are not known and numerical simulations provide invaluable information to understand these systems.

Even with numerical simulations, the complexity of systems at realistic conditions typically requires massive computational resources. In the last few decades, this computational power has been realized through increasing levels of parallelism. When a problem is decomposed into a number of processing elements (PEs), solving the PDE typically requires communication between PEs to compute spatial derivatives. Each communication takes at least a few microseconds, and on a common network, tens of microseconds. This minimum communication time, regardless of how much information is communicated, is called the network latency. If network latency exceeds the computing time between consecutive communications, reducing the computation of each node does not accelerate the simulation. This barrier to scaling, called the latency barrier, is necessarily encountered as PDE-based simulation is deployed to more PEs. Hence, as the number of PEs increase, this communication become more challenging. The speed of communication and computation have increased over time but the communication speed relative to the

computation have been dropping overtime. Network latency improves slowly over time compared to the Moores law. On average, network bandwidth doubles every two years; but over the same amount of time, latency improves on average by no more than 20 to 40 percent [1]. In fact, this may well be a major bottleneck at the next generation of computing systems which may comprise an extremely large number of PEs. At those extreme levels of parallelism, even small imbalances due to noise in otherwise perfectly balanced codes can represent enormous penalties as PEs idle, waiting to receive data from other PEs. This is especially critical when, as commonly done, a global synchronization is imposed at each time step to finalize all communications as well as to obtain information to determine the time-step size in unsteady calculations subjected to a so-called Courant Friedrichs Lewy condition. Thus, in order to take advantage of computational systems at extreme levels of parallelism, relaxing all (especially global)synchronizations is of prime necessity. Fig 1.1 plots the time required for the simulation of one time step of Advection - diffusion Equation with 65536 grid points with varying number of processor. We observe that as the number of processors increased, the time begin to increase rather than decrease after a certain point. The latency cost exceeds the computation cost in such cases. The ideal scaling is observed for the asynchronous case. In asynchronous case, different processors finish at different time. So, the plot of average time becomes necessary.

Previous work by Donzis & Aditya [2] have mainly been limited to theoretical aspects. In practice, when the asynchronous scheme is implemented using parallel libraries, the time required for the overall computation decreases but the error increases. Hence, there is a time - error trade-off. This is the main thrust of this work. In practice, the asynchronous scheme can be implemented using standard parallel libraries like MPI and Open-MP. In this work, we discuss the implementation of asynchronous schemes using these standard libraries functionality in a Deterministic and Stochastic way. In Deterministic way, we exchange the information only after a fixed amount of time and in Stochastic implementation, we exchange data as and when available. We try to study the delay statistics when the communication is carried out via MPI two-sided communication Vs one-sided and the impact of Computer Architecture on asynchronous scheme.

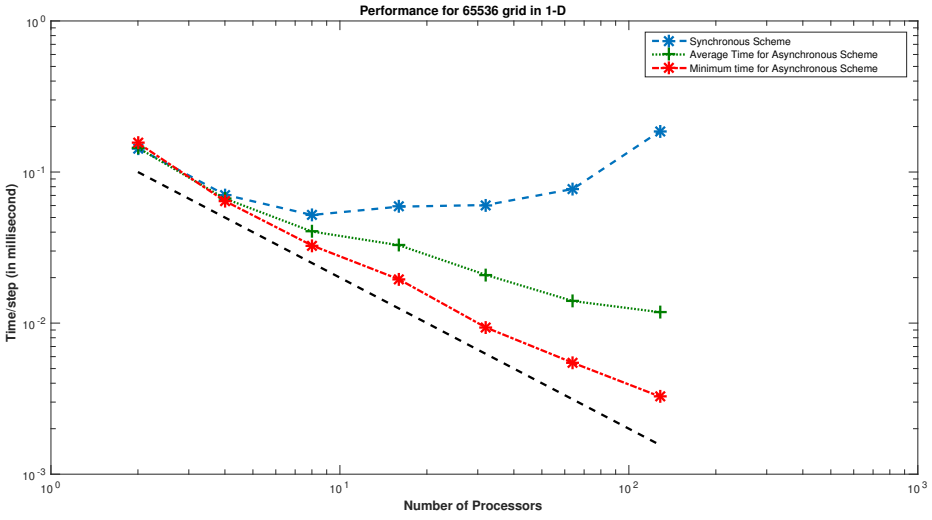


Figure 1.1: Time taken in case of Synchronous and Asynchronous Scheme

1.2 Organization of Chapter

We begin our discussion with the Numerical Analysis for the Synchronous and Asynchronous Case. In order to use the asynchronous scheme, it needs to be stable and consistent. Chapter 2 deals with the stability, convergence and consistency of asynchronous case. In Chapter 2, we will observe that the order of Convergence for standard Central Difference drops down to First order in space. Chapter 3 deals with the methods to recover the order. Chapter 4 deals with the implementation of Asynchronous Scheme using parallel libraries MPI and OpenMP.

Chapter 2

Numerical Analysis of Asynchronous Scheme

The current chapter deals with the theoretical analysis of the Asynchronous Scheme. First, we will consider the Diffusion Equation and extend the analysis to the Advection - Diffusion Equation. We will observe that the Asynchronous Scheme remains stable under the regime of stability of Synchronous Scheme. However the order of convergence drops to first order, when the standard Central Difference Scheme is used to discretize the Diffusion equation. The majority of the analysis presented in this Chapter has been taken from [2, 3]. The theoretical analysis are then validated using the Numerical Simulation by introducing the artificial delay.

2.1 Concept

Finite difference schemes are one way to solve partial differential equations. Consider for example the one dimensional heat equation in its simplest case:

$$\frac{\partial u(x,t)}{\partial t} = \alpha \frac{\partial^2 u(x,t)}{\partial x^2} \tag{2.1}$$

where $u(x,t)$ is the temperature at a spatial location $x \in [0,l]$ and time t and α is the thermal diffusivity of the medium. With N uniformly distributed grid points, Eq. (2.1) can be discretized using a second-order central difference in space and first-order forward difference in time to obtain a numerical scheme

with well-know characteristics.

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2} + \mathcal{O}(\Delta x^2, \Delta t) \tag{2.2}$$

where u_n^i is the temperature at a point $x = x_i$ at the time level n. Here $x_i = i\Delta x$ with $\Delta x = l/N$ being the grid spacing and i = 1 , . . . , N. The time step size is Δt. The last term represents the order of the truncation error in time and space for this approximation. The above Eq. (2.2) can be re-written as

$$u_i^{n+1} = u_i^n + \frac{\alpha \Delta t}{\Delta x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n) \tag{2.3}$$

which shows that to advance the solution from time level n to $n+1$, one needs the value of the function at neighbouring points at time level n. This is trivially implemented in a serial code where all the values u_i^n are available in the PE memory

The stencil corresponding to Eqn (2.3) is shown in Figure 2.1.

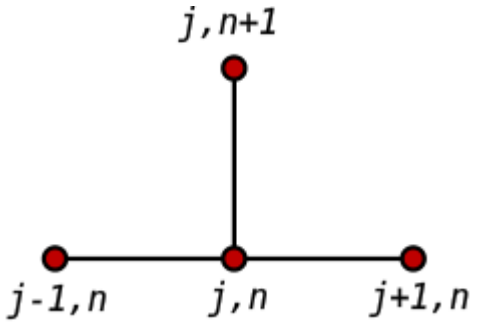


Figure 2.1: Standard explicit stencil from heat equation.

Since the error is $(\Delta x^2, \Delta t)$ we could increase the number of spatial and temporal nodes to increase the accuracy of the solution. The corresponding system can be written in matrix form as $u^{n+1} = Mu^n$ with corresponding matrix $M \in \mathbb{R}^{N_x \times N_x}$. Extending the problem into 3 dimensions the problem size grows since the linear system is of size $N_x^3 \times N_x^3$. Therefore, it might me helpful, to solve these systems on multiple processors. One common approach to split the workload onto multiple processors, is to decompose the computational domain accordingly to the physical domain which can be seen in Figure 2.2.

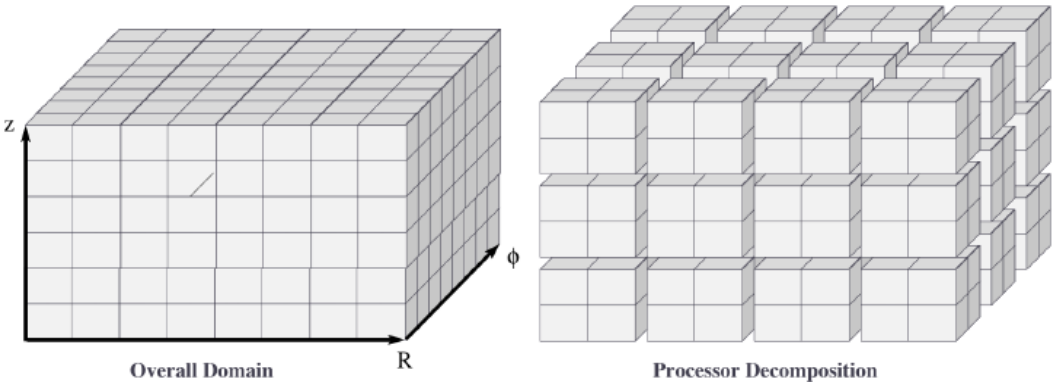


Figure 2.2: Domain Decomposition for the 3D case. Split every part of the domain into different PEs

The problem, that arises in the context of domain decomposition is the need for information exchange on the boundary elements of each PE.

2.1.1 Synchronous Finite Difference Scheme

Consider this problem in greater detail. In Figure 2.3 we we can see grid points of a one dimensional domain decomposition. Computations at interior points remain trivial as the required information is available locally to the PEs. Updating the values at grid points close to PE boundaries, however, require values from other PEs, that is, either u_{i+1}^n or u_{i-1}^n from the corresponding neighbouring PEs. These values are typically communicated over the network into buffer (or "ghost") arrays. Computations are halted until all PEs receive data in these so-called halo exchanges.

By using asynchronous finite-difference schemes, Donzis & Aditya [2] propose a way to overcome this synchronization overhead and reduce the communication. Asynchronous schemes eliminate the synchronization after each time layer by allowing the different PEs to run asynchronous in time. The question arises, under which conditions these methods are still convergent and whether the accuracy remains the same.

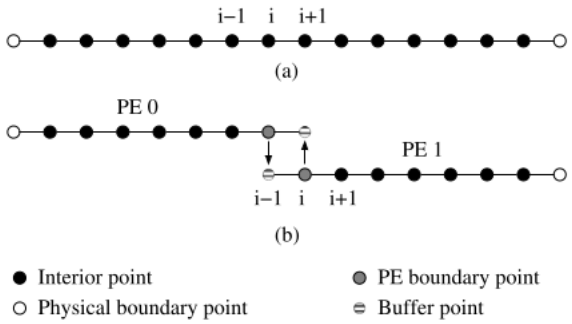


Figure 2.3: Discretized one-dimensional domain. (a) Domain in serial codes. (b) Same domain decomposed into two PEs.

2.1.2 Asynchronous Finite Difference Scheme

The problem of synchronization can be avoided by allowing all PEs to continue calculations regardless of the status of the messages that are to be received by the corresponding PEs. In the context of Figure 2.3, PE 1 is not required to wait for the most updated value u_{i-1}^n, but instead, can compute derivatives using $u_{i-1}^{\tilde{n}}$ where $\tilde{n}$ is the latest time level available to PE 1 at x_{i-1}. This modifies the finite difference equation (2.3) for points close to PEs boundaries. In particular, the expressions for the leftmost and rightmost grid points in each PE are, respectively:

$$u_i^{n+1} = u_i^n + \frac{\alpha \Delta t}{\Delta x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^{\tilde{n}}) \tag{2.4}$$

$$u_i^{n+1} = u_i^n + \frac{\alpha \Delta t}{\Delta x^2}(u_{i+1}^{\tilde{n}} - 2u_i^n + u_{i-1}^n) \tag{2.5}$$

Hereby, $\tilde{n}$ is the introduced asynchronicity. In the synchronous case, we have $\tilde{n} = n$, but now we allow $\tilde{n} = n, n-1, n-2, ..., n-(L-1)$ to be any of the prior $(L-1)$ time layers.The occurrence of a particular level for $\tilde{n}$ at a particular location and time depends on how fast the communications take place, which in turn depends on a number of factors like hardware, network topology, network traffic, message size, etc. some of which are unpredictable and turn the process into a random one. Indeed, it is known that communication times possess random characteristics in real systems. Since $\tilde{n}$ has to be modelled as a discrete random variable that takes the

values between n and $n-(L-1)$, let us rewrite $\tilde{n} = n - \tilde{k}$, then the random variable becomes $\tilde{k}$ with probabilities $p(\tilde{k})$ such that

$$\sum_{\tilde{k}=0}^{L-1} p(\tilde{k}) = 1 \tag{2.6}$$

2.2 Stability

For the stability analysis of the synchronous scheme, one of the well known technique is Von Neuman Stability Analysis, which makes use of Fourier Series to represent the error. Using Von Neuman Analysis, one can easily determine the condition, under which one of the mode (wavenumber) will grow unbound in time, rendering the scheme to be unstable.

But for the stability analysis of Asynchronous Scheme, Von Neuman Analysis has a disadvantage on account of two factors. Firstly, it requires the discretized equation to be uniform across the entire domain and secondly, the value of $\tilde{k}$ or $\tilde{n}$ at the boundaries is essentially random.

Matrix formulation, on the other hand, incorporate these difficulties in a uniform framework for the analysis of both Synchronous and Asynchronous case. Thus, this method will be used for our further analysis.

2.2.1 Synchronous Case

To prove the stability of the Asynchronous case, let us start with the formulation for Synchronous case. Eqn (2.3) can be extended to compute the value at all the nodes of u_i^{n+1} at all the nodes $i = 1, 2, ..., N$ for the time layer $n+1$. For a periodic domain, we can explicitly write:

$$\begin{pmatrix} u_1^{n+1} \\ u_2^{n+1} \\ \vdots \\ u_N^{n+1} \end{pmatrix} = \begin{pmatrix} 1-2r_\alpha & r_\alpha & 0 & 0 & \dots & 0 & 0 & r_\alpha \\ r_\alpha & 1-2r_\alpha & r_\alpha & 0 & \dots & 0 & 0 & 0 \\ & & \ddots & & & & & \\ r_\alpha & 0 & 0 & \dots & 0 & 0 & r_\alpha & 1-2r_\alpha \end{pmatrix} \begin{pmatrix} u_1^n \\ u_1^n \\ \vdots \\ u_I^n \end{pmatrix} \tag{2.7}$$

or

$$u^{n+1} = Mu^n \tag{2.8}$$

with M being the matrix of Eqn. (2.7) and $r_\alpha = \frac{\alpha \Delta t}{\Delta x^2}$. Since M is time independent, we can recursively write:

$$u^{n+1} = Mu^n = M^2 u^{n-1} = ... = M^{n+1} u^0 \tag{2.9}$$

Thus stability results from $\|M\| \leq 1$. Let $\|.\|_\infty$ be the infinity norm, defined as $\|A\|_\infty = \max_i \sum_j |A_{ij}|$. Given the matrix M, above method is stable, if

$$|r_\alpha| + |1 - 2r_\alpha| + |r_\alpha| \leq 1 \tag{2.10}$$

which is fulfilled for $0 \leq r_\alpha \leq 1/2$. This result is known as **CFL** condition for the given problem and its discretizaion.

2.2.2 Asynchronous Case

The bound on the maximal delay is needed for the theoretical analysis and the existence of such a bound can be considered a reasonable assumption. For a more simpler case, let us assume that the scheme is either synchronized or has a delay 1, i.e. $\tilde{n} = n$ or $n - 1$. Furthermore we will assume that we only need data exchange for the rightmost boundary nodes. Additionally we will assume, that the number of PEs is the number of spatial points, which is no realisitic assumption, but does not influence the validity of the proof. Under the given assumtions, we can write:

$$\begin{pmatrix} u^{n+2} \\ u^{n+1} \end{pmatrix} = \tilde{C} \begin{pmatrix} u^{n+1} \\ u^n \end{pmatrix} \tag{2.11}$$

where $(u^{n+2}, u^{n+1})^T$ is of dimension $2N$ as well as $(u^{n+1}, u^n)^T$. The matrix C is of the size $2N\,X\,2N$ and defined as

$$C = \begin{pmatrix} A_0 & A_1 \\ I & 0 \end{pmatrix} \tag{2.12}$$

with the identity matrix I and

$$A_0 = \begin{pmatrix} (1 - 2r_\alpha) & (1 - \tilde{k_2})r_\alpha & 0 & \dots & r_\alpha \\ r_\alpha & (1 - 2r_\alpha) & (1 - \tilde{k_3})r_\alpha & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ (1 - \tilde{k_1})r_\alpha & 0 & \dots & r_\alpha & (1 - 2r_\alpha) \end{pmatrix} \tag{2.13}$$

as well as

$$A_1 = \begin{pmatrix} 0 & \tilde{k_2} r_\alpha & 0 & \dots & 0 \\ 0 & 0 & \tilde{k_3} r_\alpha & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \\ \tilde{k_1} & 0 & \dots & 0 & 0 \end{pmatrix} \tag{2.14}$$

The $\tilde{k_i} \in \{0, 1\}$ enables the switch between $\tilde{n} = n$ and $\tilde{n} = n - 1$. Given this formulation and time dependency of $\tilde{C}$, Eqn (2.11) can be re-written as

$$\begin{pmatrix} u^{n+2} \\ u^{n+1} \end{pmatrix} = \tilde{C} \begin{pmatrix} u^{n+1} \\ u^{n} \end{pmatrix} = \dots = \tilde{C}^{n+1} \begin{pmatrix} u^{1} \\ u^{0} \end{pmatrix} \tag{2.15}$$

Let us define $W^n = (u^{n+1}, u^n)^T$. For stability we want $\|W^n\| \leq \|W^0\|$ for a given norm. Thus the norm of $\tilde{C}$ has to be bounded by unity. Considering the infinity norm as for the synchronous case, from the structure of $\tilde{C}$, we get:

$$\left\|\tilde{C}\right\|_\infty \leq 1 \leftrightarrow |r_\alpha| + |1 - 2r_\alpha| + |(1 - \tilde{k_i}) r_\alpha| + |\tilde{k_1} r_\alpha| \leq 1 \tag{2.16}$$

Since $\tilde{k_i}$ is either 0 or 1, this simplifies to

$$|r_\alpha| + |1 - 2r_\alpha| + |r_\alpha| \leq 1 \tag{2.17}$$

which is same exactly the stability criterion for the synchronous case in Eq. (2.10). Therefore, in this simplified case, the asynchronous scheme is stable, if the initial scheme is stable.

The theoretical analysis, showed that the question of convergence does not depend on the statistics of the delay $\tilde{k_i}$ nor on the number of PEs. The first fact is important, since the $\tilde{k_i}$ are in principle random variables and assumptions on their statistics can not be made in the general case. The latter ensures, that the method is stable independent of the domain decomposition, as well of the underlying architecture. Furthermore, stability is kept when the problem is scaled to a finer grid decomposition and a higher number of PEs.

2.2.3 General Case

Due to the choice of infinity norm for our analysis of the scheme, only those rows that include delay (the corresponding grid points close to the

boundaries) are relevant, because all other rows are bounded by unity as they are essentially synchronous.

Each finite difference stencil of size $2S + 1$, that includes only two time layers can be written as:

$$u_i^{n+1} = \sum_{j=-S}^{S} c_j u_{i+j}^n \tag{2.18}$$

Therefore the stability criteria when using the infinity norm turns out to be:

$$\sum_{j=-S}^{S} |c_j| \leq 1 \tag{2.19}$$

which is a sufficient condition for stability.

We now introduce the asynchronicity by adding an additional sum over all possible delays from 0 to $L-1$. Since only one of these terms is non-zero, we use Kronecker delta to accomplish this. Let Kronecker delta be given by:

$$\delta_{k_i,m} = \begin{cases} 1, & \text{iff } k_i = m, \\ 0, & \text{iff } k_i \neq m. \end{cases} \tag{2.20}$$

We can now write:

$$u_i^{n+1} = \sum_{m=0}^{L-1} \sum_{j=-S}^{S} c_j \delta_{k_i,m} u_{i+j}^{n-m} \tag{2.21}$$

Generalising the formulation of Eq. (2.19) to all possible time layers we can deduce the stability criteria for the general asynchronous case, i.e. our scheme is stable, if

$$\sum_{m=0}^{L-1} \sum_{j=-S}^{S} |c_j \delta_{k_i,m}| = 1 \tag{2.22}$$

Since the Kronecker delta is either 0 or 1, and adds upto unity when summed over m we get

$$\begin{aligned}\sum_{m=0}^{L-1}\sum_{j=-S}^{S}|c_j\delta_{k_i,m}| &= \sum_{m=0}^{L-1}\sum_{j=-S}^{S}|c_j|\delta_{k_i,m} \\ &= \sum_{j=-S}^{S}|c_j|\sum_{m=0}^{L-1}\delta_{k_i,m} \\ &= \sum_{j=-S}^{S}|c_j| \qquad (2.23)\end{aligned}$$

which is the same condition for stability of synchronous scheme as in Eq. (2.19). Therefore asynchronous scheme is stable, if the synchronous scheme is stable.

Hence for the Advection Diffusion equation, using second order central difference for both first and second derivatives, we can write:

$$\frac{1}{\Delta t}(u_i^{n+1}-u_i^n)+\frac{c}{2\Delta x}(u_{i+1}^n-u_{i-1}^n)=\frac{\alpha}{\Delta x^2}(u_{i+1}^n-2u_i^n+u_{i-1}^n) \qquad (2.24)$$

Using $r_c=\frac{c\Delta t}{\Delta x}$ and $r_\alpha=\frac{\alpha\Delta t}{\Delta x^2}$, we can write Eqn (2.24) as:

$$u_i^{n+1}=u_{i+1}^n(r_\alpha-r_c/2)+u_i^n(1-r_\alpha)+u_{i+1}^n(r_\alpha-r_c/2) \qquad (2.25)$$

According to Eqn (2.23), the stability condition states:

$$|r_\alpha-r_c/2|+|1-2r_\alpha|+|r_\alpha+r_c/2|\leq 1 \qquad (2.26)$$

which implies for $r_c<1$

$$r_c/2\leq r_\alpha\leq 1/2 \qquad (2.27)$$

for both synchronous and asynchronous schemes.

2.3 Consistency

By performing Taylor expansion, one can easily show, that for the synchronous case and by using the above stencil, the error between the correct solution and the approximated solution can is given by

$$E_i^n=-\frac{u_{tt}}{2}\Delta t+\frac{\alpha u_{xxxx}}{12}\Delta x^2+O(\Delta t^2,\Delta x^4) \qquad (2.28)$$

For $(\Delta t, \Delta x) \to 0$ the error vanishes, therefore the numerical approximation is consistent with the analytical problem with first order convergence in time and second order in space.

Let us consider the asynchronous case. For arbitrary delay k at a grid point $i+1$, we deduce with the help of Taylor expansion:

$$\tilde{E}_i^n|_{\tilde{k}_{i+1}} = -\frac{u_{tt}}{2}\Delta t + \frac{\alpha u_{xxxx}}{12}\Delta x^2 - \alpha k u_t \frac{\Delta t}{\Delta x^2} + \alpha k u_{tx}\frac{\Delta t}{\Delta x} - \alpha k u_{txx}\frac{\Delta t}{2} + O(\Delta x^3, \Delta t^2, \Delta x^p \Delta t^q) \tag{2.29}$$

where the parameter p and q are bounded from below by -2 and 1 respectively. By introducing the Courant number $r_\alpha = \alpha \Delta t/\Delta x^2$, we can rewrite the equation and get

$$\begin{aligned}\tilde{E}_i^n|_{\tilde{k}_{i+1}} = -\frac{r_\alpha u_{tt}}{2\alpha}\Delta x^2 + \frac{\alpha u_{xxxx}}{12}\Delta x^2 - \underline{r_\alpha k u_t} + r_\alpha k u_{tx}\Delta x \\ - r_\alpha k u_{txx}\frac{\Delta x^2}{2} + O(\Delta x^3, \Delta t^2, \Delta x^p \Delta t^q)\end{aligned} \tag{2.30}$$

We see, that this scheme, because of the underlined term in (2.30) is not consistent with the initial scheme, when r_α is kept constant. Since the error contains zeroth order contributions, the error does not vanish under grid refinement. However, this error analysis is only valid for nodes where two PEs work asynchronous. For the majority of points, this is not the case. It is therefore necessary to perform a statistical error analysis, that includes both, the statistics of the delays as well as the number of PEs.

In the following discussion, let I_B denote the set of boundary nodes and I_I the set of interior nodes. That is, $i \in I_B$, if we need communication between PEs for the computation of u_i^{n+1} and $i \in I_I$ otherwise. Clearly I_B and I_I are disjoint. We introduce a spatial average $\langle \overline{f} \rangle_S = 1/N_S \sum_{i \in S} f_i$ for a set S of size N_S. Furthermore let $\bar{f}$ denote the ensemble average over the statistical properties. The combination of the spatial and ensemble average for a given time layer n is

$$\langle \overline{E} \rangle = \frac{1}{N}\sum_{i=1}^{N} \bar{E}_i^n \tag{2.31}$$

Since we only need to take an ensemble average over nodes that belong to

I_B this is equivalent to

$$\langle \overline{E} \rangle = \frac{1}{N} \Bigg[\underbrace{\sum_{i \in I_I} {E_i}^n}_{\text{Synchronous part}} + \underbrace{\sum_{i \in I_B} \bar{E_i^n}}_{\text{Asynchronous part}} \Bigg] \tag{2.32}$$

First, lest us consider the synchronous part of Eqn. (2.32). From Eqn. (2.28), we know that

$$\begin{aligned} \sum_{i \in I_I} E_i^n &\approx \sum_{i \in I_I} \left(-\frac{u_{tt}}{2}\Delta t + \frac{\alpha u_{xxxx}}{12}\Delta x^2 \right) \\ &= \sum_{i \in I_I} \left(-\frac{u_{tt}}{2\alpha} r_\alpha + \frac{\alpha u_{xxxx}}{12} \right) \Delta x^2 \\ &= \Delta x^2 \sum_{i \in I_I} K_S \\ &= \Delta x^2 N_I \langle \overline{K_S} \rangle_{I_I} \end{aligned} \tag{2.33}$$

where space average K_S is given by

$$K_S = \left(-\frac{u_{tt}}{2\alpha} r_\alpha + \frac{\alpha u_{xxxx}}{12} \right) \Delta x^2 \tag{2.34}$$

Now, consider the asynchronous part of Eqn (2.32). Let $p_{k[i+1]}$ denotes the probability of facing delay $= k$ at the node $i+1$. Thus, the asynchronous part of delay can be given as:

$$\begin{aligned} \sum_{i \in I_B} \overline{\tilde{E}_i^n}|_{\tilde{k}_{i+1}} &\approx \sum_{k=0}^{L-1} p_{k[i+1]} \tilde{E}_i^n|_{\tilde{k}_{i+1}=k} \\ &\approx \left(-\frac{r_\alpha u_{tt}}{2\alpha}\Delta x^2 + \alpha \frac{u_{xxxx}}{12}\Delta x^2 \right) + \left(-r_\alpha u_t + r_\alpha u_{tx}\Delta x - \frac{r_\alpha u_{txx}}{2}\Delta x^2 \right) \overline{\tilde{k}_{i+1}} \\ &\approx \left(-\frac{r_\alpha u_{tt}}{2\alpha}\Delta x^2 + \alpha \frac{u_{xxxx}}{12}\Delta x^2 \right) + \left(-r_\alpha u_t + r_\alpha u_{tx}\Delta x - \frac{r_\alpha u_{txx}}{2}\Delta x^2 \right) \overline{\tilde{k}} \\ &\approx N_B \langle K_S \rangle_{I_B} \Delta x^2 + N_B \left(-r_\alpha \langle \dot{u} \rangle_{I_B} + r_\alpha \langle u_{tx} \rangle_{I_B} \Delta x - \frac{r_\alpha \langle u_{txx} \rangle_{I_B}}{2} \Delta x^2 \right) \overline{\tilde{k}} \end{aligned} \tag{2.35}$$

where $\overline{\tilde{k}_{i+1}} = \sum_{k=0}^{L} p_{k[i+1]} k$. Here we have assumed that k_i is independent of i, i.e. the delay statistics is same for all the boundary points, which later we will see in Chapter 4 is not the case in practice. To account for the dependence of k_i on i, $\overline{\tilde{k}_{i+1}}$ has to be replaced by spatial average $\langle \bar{\tilde{k}} \rangle_{N_B}$.

Thus overall statistical error is given by:

$$\begin{aligned} \overline{E} = N_I \left\langle \overline{K_S} \right\rangle_{I_I} \Delta x^2 + N_B \left\langle \overline{K_S} \right\rangle_{I_B} \Delta x^2 + \Bigg(& - r_\alpha N_B \langle u_t \rangle_{I_B} \\ & + r_\alpha N_B \langle u_{xt} \rangle_{I_B} \Delta x - \frac{N_B}{2} \langle u_{txx} \rangle_{I_B} \Bigg) \bar{\tilde{k}} \end{aligned} \tag{2.36}$$

Taking spatial average over the entire domain, we get:

$$\left\langle \overline{E} \right\rangle = \left\langle \overline{K_S} \right\rangle \Delta x^2 + \frac{N_B}{N} \left(- r_\alpha \langle u_t \rangle_{I_B} + r_\alpha \langle u_{xt} \rangle_{I_B} \Delta x - \frac{1}{2} \langle u_{txx} \rangle_{I_B} \right) \bar{\tilde{k}} \tag{2.37}$$

From the above discussion, we can draw some interesting conclusion. For the synchronous case, $\bar{\tilde{k}} = 0$ and we have second order consistency in space. As soon as we consider the asynchronous case, even for the small amount of delay, there is a decrease in convergence, and we are left with

$$\left\langle \overline{E} \right\rangle \approx -\frac{N_B}{N} \bar{\tilde{k}} r_\alpha \left\langle \overline{u_t} \right\rangle_{I_B} = -S \frac{P}{N} \bar{\tilde{k}} \left\langle \overline{u_t} \right\rangle_{I_B} \tag{2.38}$$

Here we have used $N_B = SP$. Keeping all the parameters constant we can write $\Delta x = l/N$ and we get the following approximation:

$$\left\langle \overline{E} \right\rangle \sim \frac{P}{N} \sim P \Delta x \tag{2.39}$$

The above result shows that the rate of convergence drops to first order, when the asynchrony is present, for constant PEs. However, one important point to note that in case of weak scaling, where the problem size grows with the number of processors (i.e. $\frac{P}{N}$ = constant), the scheme becomes inconsistent. In our discussion, we will be mainly concerned about the Strong Scaling until and unless especially mentioned.

2.4 Implementation and Numerical Result

The test problem for the simulation is advection-diffusion equation:

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = \alpha\frac{\partial^2 u}{\partial x^2} \tag{2.40}$$

with $u = u(t, x)$. Further we assume a periodic domain of length 2π and the initial condition are superimposed sinusoidal waves, i.e.

$$u(x, 0) = \sum_{\kappa} A(\kappa) sin(\kappa x) \tag{2.41}$$

which has the analytical result:

$$u_a(x, t) = \sum_{\kappa} \exp^{(-\alpha\kappa^2 t)} A(\kappa) sin(\kappa(x - ct)) \tag{2.42}$$

which means that the waves are shifted by $-ct$ and damped by $\exp^{-\alpha\kappa^2 t}$.
The discretized form of advection diffusion for the interior nodes are:

$$\frac{1}{\Delta t}(u_i^{n+1} - u_i^n) + \frac{c}{2\Delta x}(u_{i+1}^n - u_{i-1}^n) = \frac{\alpha}{\Delta x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n) \tag{2.43}$$

whereas for the left boundary nodes is:

$$\frac{1}{\Delta t}(u_i^{n+1} - u_i^n) + \frac{c}{2\Delta x}(u_{i+1}^n - u_{i-1}^{\tilde{n}}) = \frac{\alpha}{\Delta x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^{\tilde{n}}) \tag{2.44}$$

and for the right boundary nodes is:

$$\frac{1}{\Delta t}(u_i^{n+1} - u_i^n) + \frac{c}{2\Delta x}(u_{i+1}^{\tilde{n}} - u_{i-1}^n) = \frac{\alpha}{\Delta x^2}(u_{i+1}^{\tilde{n}} - 2u_i^n + u_{i-1}^n) \tag{2.45}$$

The parameter used for simulation are:

Listing 2.1: Parameter for numerical simulation

```
len = 2*pi; % length of domain
A = 1;
k = 2;
alpha = 1;
r_alpha = 0.1; % CFL
c = 1;
final_t = 0.08*len/c
```

where

$$r_\alpha = \frac{\alpha \Delta t}{\Delta x^2} \tag{2.46}$$

To reproduce the numerical results, Matlab code was used to simulate the problem with artificially simulating the delay.

Listing 2.2: Matlab code for numerical simulation

```
function [ err ] = async( delay, N, PE )
% delay: Delay in time steps
% N: Number of Grid Points
% PE: Number of Processors
U_old = U_initial
while(t < final_t)
   %Calculate the value at the boundary points and store it in
       array bp
   %by taking the corresponding value from U_old.
   for proc = 1:PE
      for k = 1: (delay + 1)
         for i = 1:ne
             %ne: number of element in each processor
            %Calculate boundary nodes by using value from bp ...
            %and other nodes according to scheme
            U_new = Calculation from U_old values.
         end
         U_old = U_new;
      end
   end
end
end
```

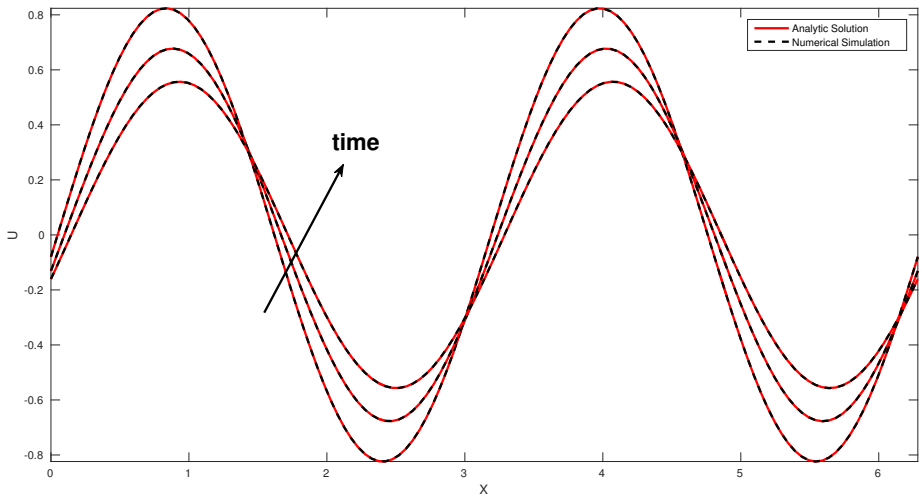
The above Matlab code is designed so that there is a communication between different processor after a certain `delay` number of time steps. `delay` $= 0$ corresponds to synchronous case. Recalling, from the analysis that we carried out, L represents the maximum allowable delay and p_k denotes the probability that that a particular boundary node will face `delay` $= k$. In particular, `delay` $= 0$, given to the above Matlab code as an input parameter represents the synchronous case, whereas `delay` $= 1$ corresponds to $L = 1$ with $p_0 = 0.5$ and $p_1 = 0.5$. Similarly $delay = 2$ corresponds to $L = 2$ with $p_0 = 0.33$, $p_1 = 0.33$ and $p_2 = 0.33$ and so on.

The legends in the figure and table entries presented below represents the value of delay given as an input parameter to the above code.

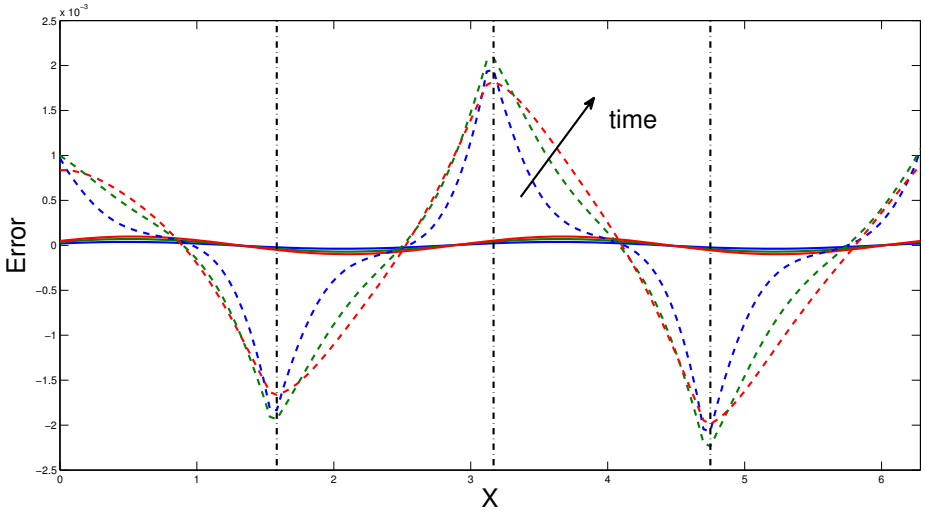
2.4.1 Results

The results of the numerical simulations are presented in Fig 2.4. During the numerical experiments, we vary the number of PEs, the statistics of delay as well as the number of grid points. Figure 2.4b represents the error, defined as $E_i = u_a - u_i(x,t)$ for the case $L = 2, p_0 = 0.5$ and $p_1 = 0.5$, where u_a represents the analytical solution and $u_i(x,t)$ represents the numerical solution. As expected, the error observed in the asynchronous case is more than the synchronous case. Also, it is to be worth noting that in case of asynchronous case, maximum error is observed at the PE boundaries, unlike Synchronous Case in which the error also follows the difference equation. [4]

From log-log plot in Figure 2.5a and Figure 2.5b, we see the second order convergence in space for the synchronous case which drops to first order convergence for the asynchronous case irrespective of the number of processor or the statistics of delay. The actual order of Convergence in case of Synchronous and Asynchronous case is compared in Table 2.1. Also, it is to be noted that larger the delay, larger is the error.

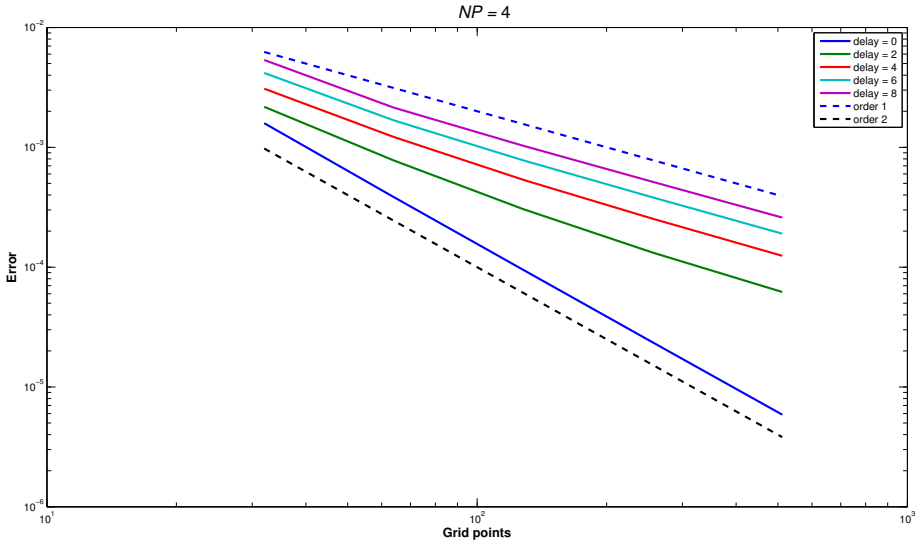


(a) Number of PEs = 4

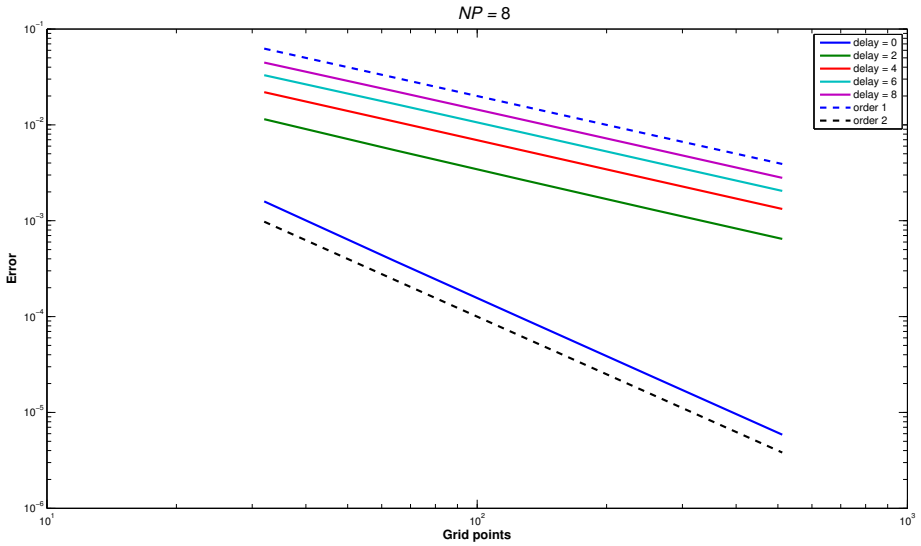


(b) Dashed line represent Asynchronous Case and Solid Lines Synchronous Case. Vertical Lines represent PE boundaries

Figure 2.4: Asynchronous Scheme with Order Recovery. For the simulation $L = 1$ with $p_0 = 0.5$ and $p_1 = 0.5$.



(a) Number of PEs = 4

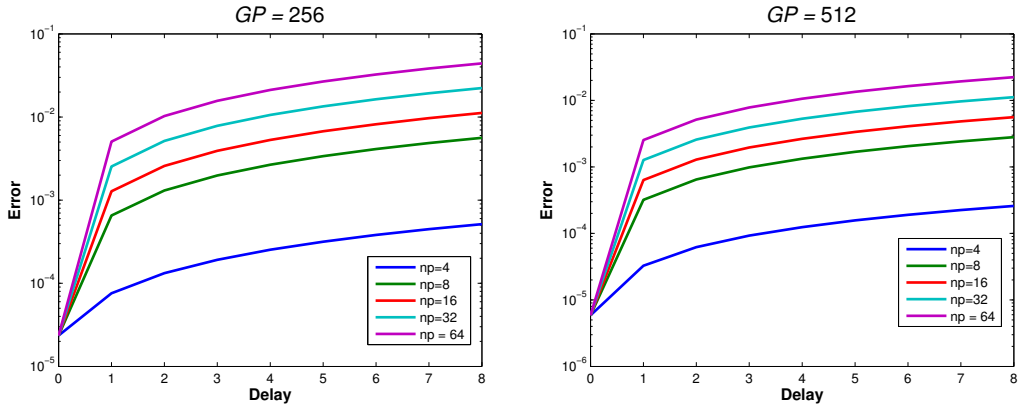


(b) Number of PEs = 8

Figure 2.5: Synchronous vs Asynchronous Schemes. The legend entries in the above figure, `delay` $= 1$ corresponds to $L = 1$ with $p_0 = 0.5$ and $p_1 = 0.5$. Similarly `delay` $= 2$ corresponds to $L = 2$ with $p_0 = 0.33$, $p_1 = 0.33$ and $p_2 = 0.33$ and so on.

Delay	Order of Convergence for asynchronous case
0(sync)	-2.0195
1	-1.0764
2	-1.0371
4	-1.0117
6	-1.0033
8	-0.9995

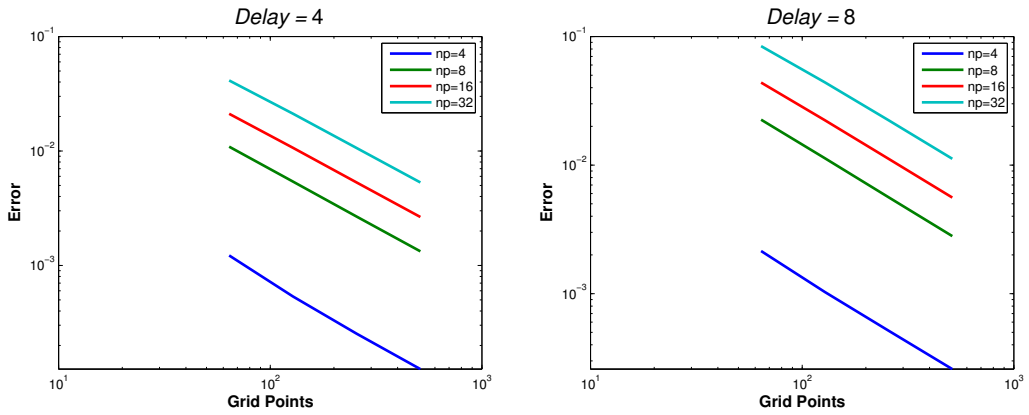
Table 2.1: Order of convergence in case of Synchronous and Asynchronous Case.



(a) Number of Grid Point = 256 (b) Number of Grid Point = 512

Figure 2.6: Influence of Delay on Grid Point

In Figure 2.6a and 2.6b, the number of grid point is kept constant at 256 and 512 respectively. For each case, a simulation with different number of PEs is performed with varying delay from 0 to 8 for each case. As expected, the error is same for all number of PEs if the delay is 0 because we are essentially simulating the synchronous case. As soon as we increase the delay, we can observe two things. Firstly, the higher the delay, the larger the error as higher delay results in increasing the number of steps where the computation is carried out asynchrony. Secondly the error is larger for a greater number of PEs which is in agreement with $\langle \overline{E} \rangle \in O(P\Delta x)$. Also larger number of PEs, keeping a fixed problem size N implies more boundary



(a) Delay = 4 (b) Delay = 8

Figure 2.7: Influence of number of Grid Point for different number of PEs

nodes which results in greater error as these are the points which faces the error due to asynchrony.

In Figures 2.7a and 2.7b the delay is fixed to 4 and 8 respectively and simulation is performed with varying number of PEs with varying number of grid points. Here again we observe the first degree convergence and error increase as we increase the number of PEs.

For all the cases, presented above we have used the concept of Strong Scaling, i.e., the problem size (N) is kept constant and number of processor is varied. In case of weak scaling, where the problem size and the number of processor varies such that $\frac{N}{P} = constant$, we observe in Fig 2.8 that the order of convergence drops down to 0 which is consistent with our discussion presented in the previous section. Table 2.2 represents the comparison between Synchronous and Asynchronous case for Weak Scaling.

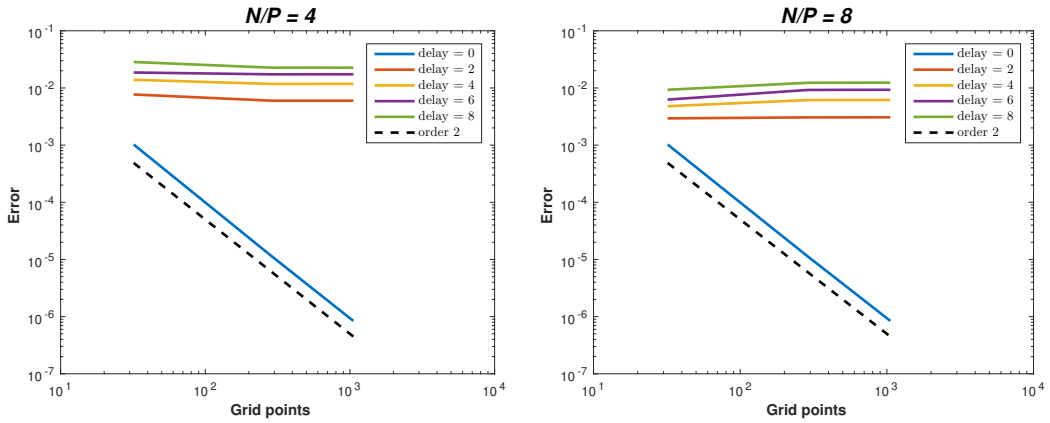


Figure 2.8: Synchronous Vs Asynchronous Case in case of Weak Scaling

Delay	Order of Convergence in case of Weak Scaling
0(sync)	-2.0034
2	-0.0749
4	-0.0490
6	-0.0214
8	-0.0685

Table 2.2: Order of convergence in case of Synchronous and Asynchronous Case for Weak Scaling.

2.5 Concluding Remarks

In this Chapter, we have discussed the concept of Asynchronous computation and carried out the analysis of the Asynchronous Scheme in terms of Stability, Consistency and Convergence. The Asynchronous Scheme is found to be stable in the regime in which Synchronous Scheme is found to be stable. Furthermore, we have observed that the Order of convergence drops to first order, in case of Asynchronous Scheme, irrespective of the statistics of delay or the number of PEs.

Based on the results of the Numerical Simulations carried out, following observation can be made:

- The points at PE boundaries faces the maximum error, in case of

Asynchronous Scheme, unlike the Synchronous Scheme for which the error too follows the difference equation.

- The order of Convergence drops down to 1 in case of Asynchronous Scheme, if standard Central Difference Stencil is used for Advection - Diffusion Equation in case of Strong Scaling.
- In case of weak Scaling, the Order of Convergence drops down to 0 rendering the scheme to be inconsistent.
- Higher delays means Higher error.
- As we increase the number of PE, the number of boundary points increases and more error can be observed.

Hence, in order to use the Asynchronous Scheme for the practical purpose, we need to recover the Order of Convergence. The next Chapter deals with the methods to do the same.

Chapter 3

Order Recovery

In the previous Chapter, we discussed the Asynchronous Scheme for the Advection - Diffusion Equation using Standard Central Difference Stencil in terms of its Stability, Consistency and Order of Convergence. The Asynchronous Scheme was found to be stable and consistent if Synchronous Scheme is stable but the Order of Convergence drops down to first order. In this Chapter we will discuss two of the Order Recovery techniques, based on the analysis of the truncation error. First technique deals with the changing the time step Δt while the second technique deals with the use of a wider Stencil.

3.1 Order Recovery by Changing the value of Time Step

Based on the Truncation Error analysis, presented in the previous Chapter, rewriting Eqn. (2.29), we get:

$$\tilde{E}_i^n|_{\tilde{k}_{i+1}} = -\frac{u_{tt}}{2}\Delta t + \frac{\alpha u_{xxxx}}{12}\Delta x^2 - \alpha k u_t \frac{\Delta t}{\Delta x^2} + \alpha k u_{tx}\frac{\Delta t}{\Delta x} - \alpha k u_{txx}\frac{\Delta t}{2} + O(\Delta x^3, \Delta t^2, \Delta x^p \Delta t^q) \tag{3.1}$$

On approximating, $\Delta t \sim \Delta x^3$, we get the truncation error of the Order of Δx. Recalling from the previous Chapter, during the analysis of Asynchronous case, we have the asynchronous part multiplied by a factor of $\frac{N_B}{N}$ which we have proved to be approximately of the order of Δx, results in the overall scheme to be order of Δx^2.

Delay	Order of Recovery for Asynchronous Scheme	Order Recovery by Changing Δt
0(sync)	-2.0195	-1.9008
1	-1.0764	-1.9712
2	-1.0371	-2.0048
4	-1.0117	-2.0313
6	-1.0033	-2.0488
8	-0.9995	-2.0609

Table 3.1: Comparison of the Order Recovery Scheme with the Original Scheme by Changing Δt

3.1.1 Numerical Simulation

For the Numerical Simulation, the parameters are decided as:

Listing 3.1: Parameters for Order Recovery by Changing Time Step

```
len = 2*pi; % length of domain
A = 1;
k = 2;
alpha = 1;
delta_t = delta_x^3;
r_alpha = alpha*delta_t/delta_x^2;
c = 1;
final_t = 0.08*len/c
```

The results of the numerical Simulation is presented in Figure 3.1. The actual Order is presented in the Table 3.1. We observe that the results of numerical simulation is consistent with the analysis and we recover a Second Order Accurate Scheme. But decreasing the value of Δt according to Δx doesn't solve our purpose. Implementing the current method in practice means that if space resolution is increased by a factor of 2, the time step is to decreased by a factor of 8. This renders the simulation quite expensive with the increasing space resolution. Thus, we now present a different approach which retains a desired order of accuracy without the need to decrease the time step size by using a different type of Stencil.

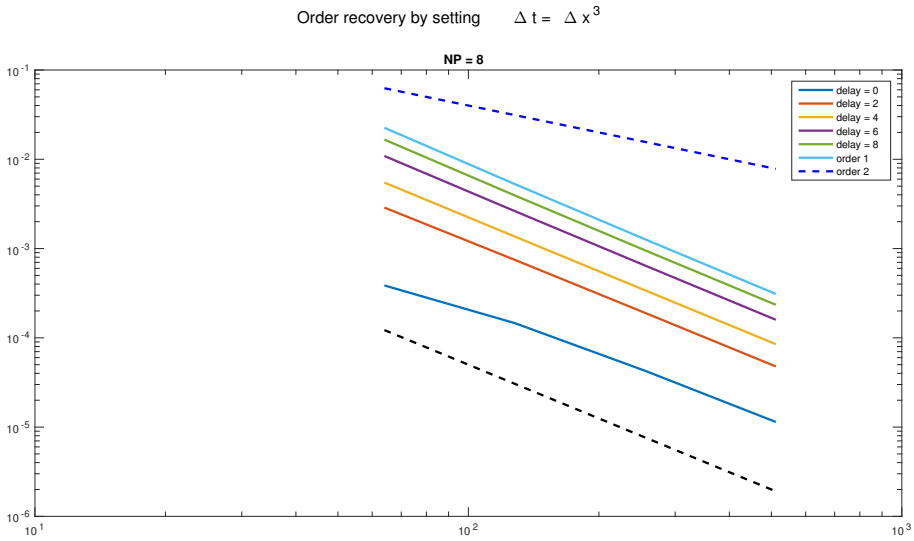


Figure 3.1: Number of PEs = 8. The Order Recovery Scheme obtained by changing the value of time step Δt

3.2 Order Recovery by using Modified Stencil

Let us consider the modified stencil of the form given by Eqn (3.2). Our aim is to study the truncation error in presence of asynchrony and obtain a consistent, stable scheme of Order of Δx because of a pre-factor of N_B/N multiplied with the asynchronous part.

$$\frac{\partial^2 u}{\partial x^2} = \frac{b_{-2}u^n_{i-2} + b_{-1}u^n_{i-1} + b_0 u^n_i + b_1 u^{\tilde{n}}_{i+1} + b_2 u^{\tilde{n}}_{i+2}}{\Delta x^2} \tag{3.2}$$

Considering the delay only on the right boundary node, the resulting stable and consistent scheme is given by Eqn (3.3). The details are mentioned in [2].

$$\frac{1}{\triangle t}(u^{n+1}_i - u^n_i) = \frac{\alpha}{2\triangle x^2}(u^{\tilde{n}}_{i+2} - u^{\tilde{n}}_{i+1} - u^n_i + u^n_{i-1}) + \mathcal{O}(k\Delta t, \Delta x, k\Delta t/\Delta x) \tag{3.3}$$

When used in context with Advection-diffusion Scheme, the scheme results in Eqn (3.4). The scheme is conditionally stable for $r_\alpha < 1/2$ and

$r_c \leq \sqrt{2r_\alpha} - r_\alpha (r_c = \frac{c\Delta t}{\Delta x})$. [2]

$$\frac{1}{\triangle t}(u_i^{n+1} - u_i^n) + \frac{c}{2\triangle x}(u_{i+1}^{\tilde{n}} - u_{i-1}^n) = \frac{\alpha}{2\triangle x^2}(u_{i+2}^{\tilde{n}} - u_{i+1}^{\tilde{n}} - u_i^n + u_{i-1}^n) \quad (3.4)$$

The scheme proposed by Donzis & Aditya [2] do not tell anything about the scheme to be used for the left boundary node. So, as a part of our original work, we tried to devise a scheme capable of recovering order for left boundary node too. This is essential as the delay can be faced at either of the two ends or both.

The above scheme is not symmetric. It has two points on the right and one point on the left. Also, in general, asynchrony can be observed at both right and left boundary nodes. There are two possibilities in order to form scheme for the left boundary nodes. One way is to keep the stencil as same as Eqn (3.4) and consider the asynchrony at the points which do not fall within the PE whereas the second approach is to shift the stencil such that it too have the two points which are outside the PE boundaries. These two schemes are discussed below in detail. It must be noted that, we are changing the scheme only for the boundary nodes, whereas for the central nodes that has its neighbour within the PE, it is still discretized by the Standard Central Difference Scheme.

3.2.1 Scheme I for Left Boundary Nodes

Considering the Scheme same as Eqn (3.4) and accounting for the left boundary node and the nodes that do not fall within the PE, the equation for asynchronous case takes the form of Eqn (3.5).

$$\frac{1}{\triangle t}(u_i^{n+1} - u_i^n) + \frac{c}{2\triangle x}(u_{i+1}^n - u_{i-1}^{\tilde{n}}) = \frac{\alpha}{2\triangle x^2}(u_{i+2}^n - u_{i+1}^n - u_i^n + u_{i-1}^{\tilde{n}}) \quad (3.5)$$

After performing the truncation analysis and collecting the terms of highest order of Δx and Δt, the truncation error results to:

$$\frac{1}{\triangle t}(u_i^{n+1} - u_i^n) = \frac{\alpha}{2\triangle x^2}(u_{i+2}^n - u_{i+1}^n - u_i^n + u_{i-1}^{\tilde{n}}) + \mathcal{O}(k\Delta t, \Delta x, k\Delta t/\Delta x^2) \quad (3.6)$$

Here, we observe that the truncation error contains the term $\Delta t/\Delta x^2$ pre-multiplied by the factor k (delay) which in our formulation at constant

r_α leads to the same problem as discussed in Section 2.3 and the overall order drops to first order. The results for the numerical simulation indicates that the scheme remains consistent and stable but the order of convergence drops down to first order with the increasing delay. However, we observe that the magnitude of error decreases with the new Scheme as indicated by Fig 3.2 and Fig 3.3. However, still the maximum error appears on the boundaries. The actual Order of the Scheme is represented in Table 3.2. Here, we observe that we are able to recover the order to some extent. But, as the delay is increased the Order of Convergence falls back to the first order, mainly because of the term $\Delta t/\Delta x^2$ being pre-multiplied by a factor of k, as discussed above.

3.2.2 Scheme II for Left Boundary Nodes

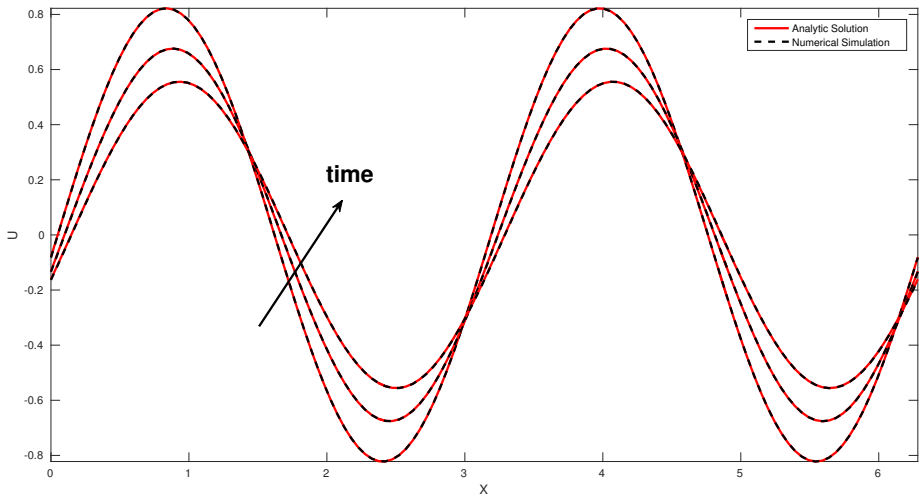
Shifting the Scheme given by Eqn (3.4) by one to the left, i.e. by replacing i by $i-1$, such that the left boundary nodes too have two points from outside of the PE. The resulting approximation to the diffusion equation becomes:

$$\frac{1}{\triangle t}(u_i^{n+1}-u_i^n) = \frac{\alpha}{2\triangle x^2}(u_{i+1}^n-u_i^n-u_{i-1}^{\tilde{n}}+u_{i-2}^{\tilde{n}})+\mathcal{O}(k\Delta t,\Delta x,k\Delta t/\Delta x) \quad (3.7)$$

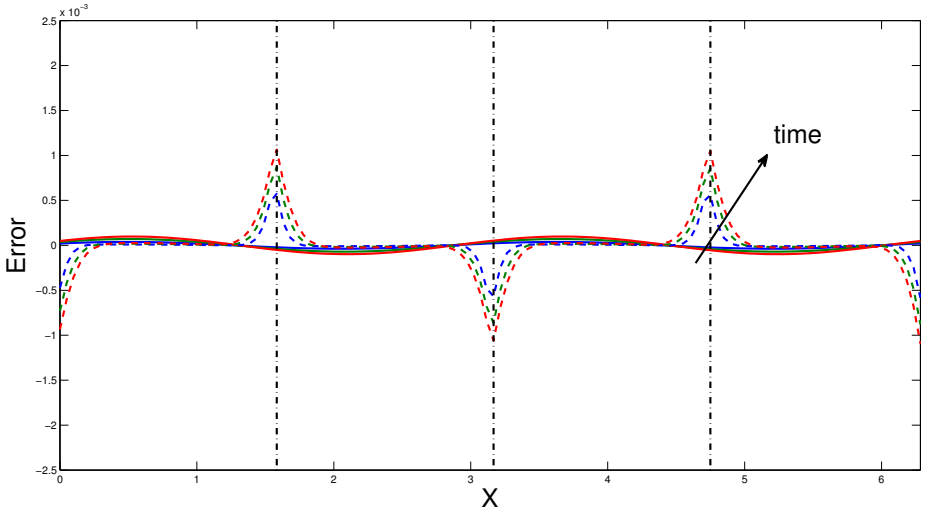
This scheme seems to be perfect as the truncation error is of the Order of Δx. But when used in conjunction with Eqn (3.4), this overall way of discretization faces a intrinsic difficulty.

Consider a point ne on the right boundary of a particular PE. Then, $ne+1$ will be the left boundary of the next PE. Consider the Synchronous case. If we discretize ne according to Eqn (3.4) and $ne+1$ according to Eqn (3.7), the both points lead to the same approximation of the diffusion term, which is given according to the Eqn (3.8). This formulation is clearly inconsistent, as we can also observe from the numerical results displayed in Fig 3.5. We see a certain jump close to the PE boundaries, irrespective of the space resolution and delay statistics, which is due to above mentioned reason.

$$\frac{\partial^2 u}{\partial x^2} = \frac{u_{ne+2}^n - u_{ne+1}^n - u_{ne}^n + u_{ne-1}^n}{2\Delta x^2} \quad (3.8)$$

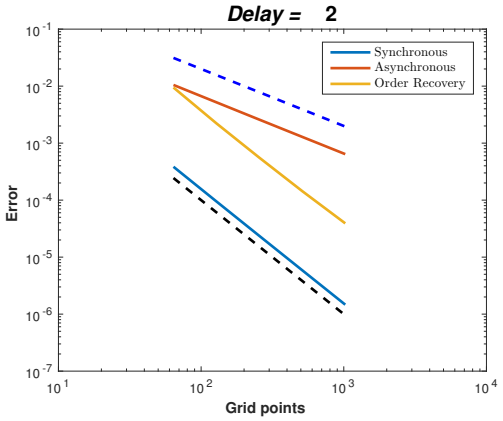


(a) Number of PEs = 4

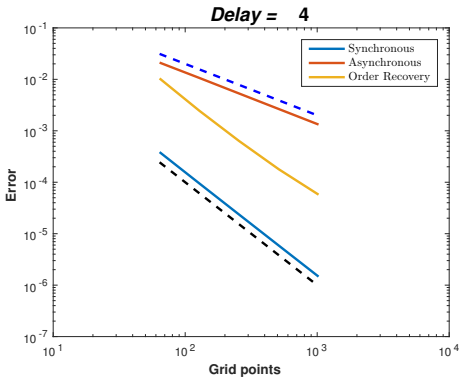


(b) Dashed line represent Asynchronous Case and Solid Lines Synchronous Case. Vertical Lines represent PE boundaries

Figure 3.2: Asynchronous Scheme with Order Recovery. For the simulation $L = 1$ with $p_0 = 0.5$ and $p_1 = 0.5$.



(a) Delay = 2 $p_0 = p_1 = p_2 = 0.33$



(b) Delay = 4, $p_0 = p_1 = p_2 = p_3 = p_4 = 0.2$

Figure 3.3: Comparison of Error in case of Synchronous, Asynchronous and Order Recovery Scheme with 16 processor

Delay	Order of Convergence for Asynchronous Scheme	Order Recovery by Changing Δt	Order Recovery by Scheme I
0(sync)	-2.0195	-1.9008	-2.0160
1	-1.0764	-1.9712	-1.8141
2	-1.0371	-2.0048	-1.6396
4	-1.0117	-2.0313	-1.4558
6	-1.0033	-2.0488	-1.3719
8	-0.9995	-2.0609	-1.3311

Table 3.2: Comparison of Order in case of Asynchronous Scheme by Standard Central Difference Scheme and Order Recovery by Changing Δt and Scheme I

.

3.2.3 Modified Scheme

This Scheme is based on the assumption that only one of the neighbouring PEs will face delay. To make it clear, let's say that PE1 is evaluating a time step n and PE2 is evaluating a time step $\widehat{n}$. The right boundary node of PE1 is facing delay (i.e. it uses the value at time level $\tilde{n}$ to calculate the value at the time n, where $(\tilde{n} < n)$), then the left boundary node of the PE2 will have the information required at the time level it is currently

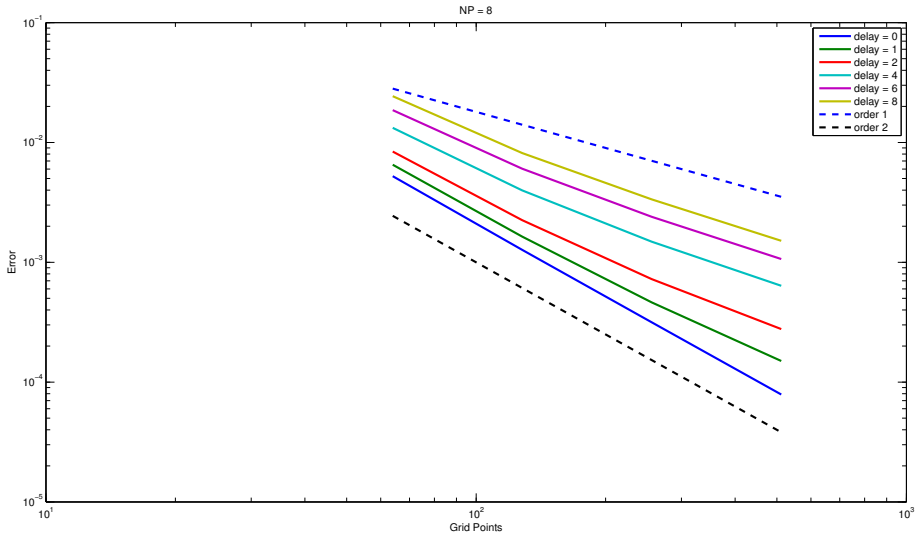


Figure 3.4: Resuts from Discretizing left boundary nodes by (3.5) and right boundary nodes by Eqn (3.4) and Standard Central Difference Scheme for interior nodes.

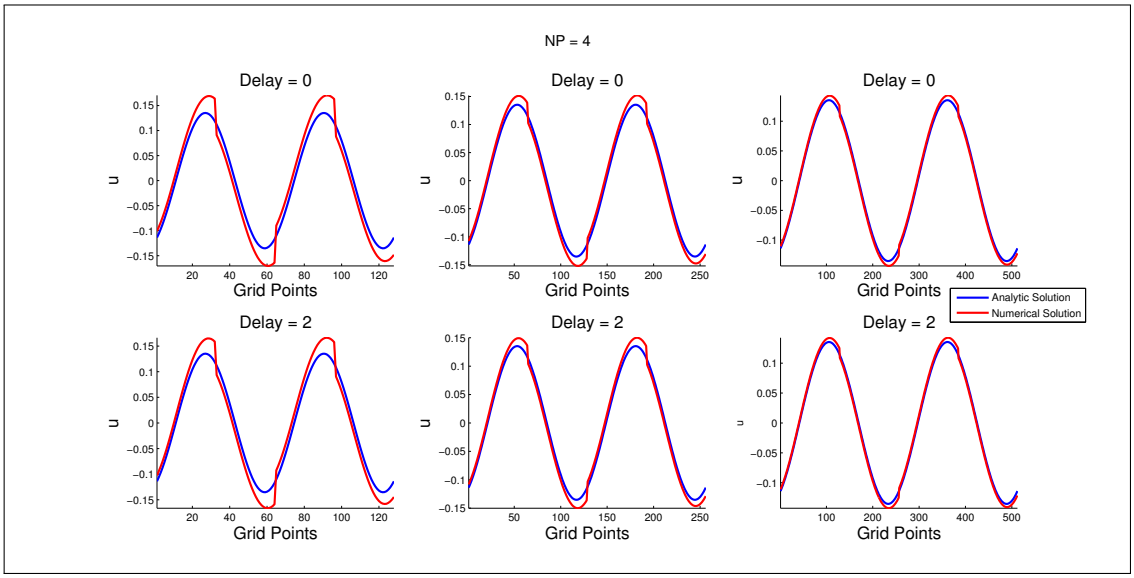


Figure 3.5: Resuts from Discretizing left boundary nodes by (3.7) and right boundary nodes by Eqn (3.4) and Standard Central Difference Scheme for interior nodes.

evaluating ($\widehat{n}$,$\widehat{n} < n$) as the next PE has the information that it requires for the evaluation of time step $\widehat{n}$ in its memory. Similarly if the left node of a particular PE is facing delay, then the right boundary node of previous PE will not face delay. (By next and prev PE, we mean the PE which contains the neighbouring boundary point. For example, if l is a particular left boundary point for PE1, the PE previous to PE1 will be the PE which has $l-1$ as its right boundary point. Similarly, for the next PE.) The basis of the assumption, in practical sense means that there is no buffering of message, i.e. if the computation is finished for a particular time step, the neighbouring PE has received its value. There is no temporary buffer which stores the value to be sent at ta later stage. In the next Chapter, we will see that communication via `MPI_Send` and `MPI_Irecv` will allow the buffering of message to take place, whereas, the communication via `MPI_Put` or `MPI_Get` will not allow the buffering of the same.

Under this assumption, we can develop a framework to decide the Discretization Scheme according to delay statistics. If a left boundary node is facing delay, we discretize it according to Eqn.(3.7), otherwise using the standard Central Difference Scheme as it has the required information at

Current Time Step	Time Step data used for Evaluation of Current Time Step for			
	PE0 (no delay)	PE1 (delay)	PE2 (no delay)	PE3 (delay)
n+1	n	n	n	n
n+2	n+1	n	n+1	n
n+3	n+2	n	n+2	n
n+4	n + 3	n	n + 3	n
	Synchronization			
n+5	n+4	n+4	n+4	n+4

Table 3.3: Time step data used for the evaluation when `delay = 4`

the current time step which it is evaluating and thus the scheme will be second order accurate. Similarly, if the right boundary node is facing delay, it is discretized according to Eqn. (3.4), otherwise according to Standard Central Difference Scheme. The problem discussed in Section 3.2.2 will not be present because of our assumption, as either the left boundary point of a PE or right boundary point of previous PE will be facing delay but not both.

In order to artificially simulate the delay with 4 processors, we chose two processors (say PE1 and PE3) which faced delay. To calculate the value of boundary nodes of PE1 and PE3 the delayed values from PE0 and PE2 are used. But to calculate the value of PE0 and PE2 , the value at current time level is used. All the processors are synchronized after `delay` number of steps. To make it clear, the time step value used for evaluation for `delay = 4` is mentioned in Table 3.3. The results of the numerical simulation using the modified scheme is shown in Fig. 3.6. The actual Order of the modified scheme is listed in Table 3.4, where we observed the recovered order in the last column.

Delay	Order of Convergence for Asynchronous Scheme	Order Recovery by Changing Δt	Order Recovery by Modified Scheme
0(sync)	-2.0195	-1.9008	-2.0195
1	-1.0764	-1.9712	-2.0050
2	-1.0371	-2.0048	-2.0049
4	-1.0117	-2.0313	-2.0044
6	-1.0033	-2.0488	-2.0039
8	-0.9995	-2.0609	-2.0029

Table 3.4: Comparison of Orders of Various Scheme.

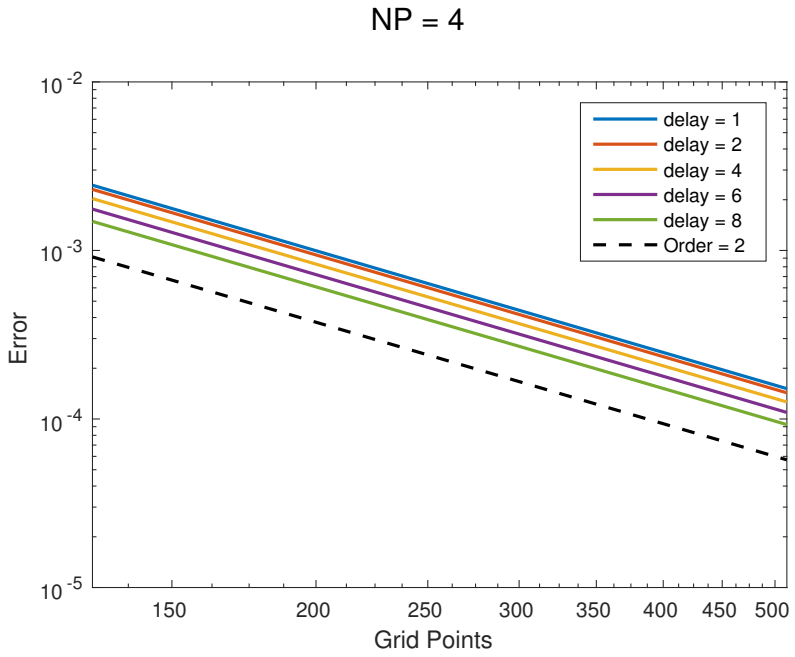


Figure 3.6: Resuts from Discretizing Advection-Diffusion Equation according to Modified Scheme

3.3 Concluding Remarks

In this Chapter, we discussed the two techniques in order to recover the Order - Changing the value of Δt and modifying the Stencil. Though able to

recover the order, the choice of Δt according to the value of Δx doesn't seems to be advantageous as with increasing the space resolution, the time step decreases. The Second method under the assumption of no buffering seems to recover the order without changing the value of Δt. But the question that a scheme, irrespective of delay statistics and able to recover the order is still not answered and this is part of our ongoing effort. In the next Chapter, we will deal with the issues related to the actual implementation of Asynchronous Scheme using the parallel libraries, namely MPI and OpenMP.

Chapter 4

Implementation using parallel libraries

As we have discussed in Section 2.1, there is a need to divide the domain and distribute it across different computers, a technique known as Domain Decomposition. The two commonly used libraries that can be used to accomplish the task are MPI and OpenMP. We begin this chapter by brief introduction of these two libraries, followed by the algorithms used for the implementation of the Asynchronous Scheme. This is the part of our Original work. Finally, we will introduce the effect of Computer Architecture and its role in determining the most effective way of data splitting over the various processors.

4.1 MPI

MPI stands for Message Passing Interface. MPI provides a mature, capable, and efficient programming model for parallel computation. Parallel computing, with any programming model, involves two actions: transferring data among workers and coordinating the workers. In message passing, a parallel program comprises a number of separate processes that communicate by calling routines. In general, the communication can be either two-sided (via `MPI_Send` and `MPI_Recv`) or one-sided (via `MPI_Put` and `MPI_Get`). The two sided communication involves the both the *source* and the *target* processor during the communication, i.e. the consent of both is necessary to finish the request. (This is accomplished through the

matching tag and rank). In one sided communication, a process may either put data into another process or get data from another process. The process performing the operation is called the origin process; the other process is the target process. The data movement hap- pens without explicit cooperation between the origin and target processes. The origin process specifies both the source and destination of the data. (For details, refer [5, 6].)

4.2 OpenMP

OpenMP parallelism uses the concept of shared memory parallelism. Programming in shared memory can be done in a number of ways, some based on threads, others on processes. The main difference, is that threads share the same process construct and memory, whereas multiple processes do not share memory. MPI is a multiple process based programming model. Overall, thread-based models have some advantages. Creating an additional thread of execution is usually faster than creating another process, and synchronization and context switches among threads are faster than among processes.

OpenMP uses an execution model of fork and join in which the "master" thread executes sequentially until it reaches instructions that essentially ask the runtime system for additional threads to do concurrent work. Once the concurrent scope of execution has completed, these extra threads simply go away, and the master thread continues execution serially. The details of the underlying threads of execution are compiler dependent and system dependent. (For details of OpenMP, [7])

4.3 Algorithmic implementation using MPI/OpenMP

Different approaches were used to simulate the asynchronous case using MPI and OpenMP. We have tried three different approaches to simulate the case. Brief algorithm are are mentioned in the following Section.

4.3.1 Master Work Paradigm Using MPI

The basic idea behind the Master Work paradigm is that there will be a *Master* processor which will coordinate the activities of all the other processors whereas the other processors will carry out the part of the work as instructed by the *Master* Processor. `Processor 0` acts as a "Master Processor" while the other processor carries the main part of the work. All the processor ("except the Master Processor") computes the values for all the nodes and send the boundary nodes to the master processor. The master processor then receives the values of the boundary nodes from the $processor_i$ and then sends the values that is needed by the $processor_i$ to carry out the further computation.

"Master processor" maintains a buffer `Vector<queue> bp` which contains the boundary points at the different time instants. The data-structure `queue` is capable to resize the array according to the need. This is needed as one processor may be more than one step ahead of the another process. Let's say $processor_i$ has finished computing at the time layer m and needed the value of the boundary nodes at time layer $m$to start the computation at time layer $m+1$. If the neighbouring processor is carrying out the computation at time layer $m+n, n>1$, then it becomes necessary to store the value at different time layers. But once the value at a particular time layer is used, it is of no more use. So, we have incorporated `push` and `pop` function to the queue data-structure which take cares of the above mentioned scenerio.

Approach 1: Master Work Paradigm using MPI

```
if(rank == 0){  // Master Processor
   // Do initialization and distribute the corresponding boundary
      values at initial condition
   Vector<queue> bp[2*(numprocs - 1)]; // Vector of queues
      corresponding to each boundary points
   int count = 0;
   Vector<double> temp_bp[2*(numprocs - 1)];
   // Initialize temp_bp with initial condition at respective point
   while(1){
      MPI_Probe(,..,MPI_ANY_SOURCE,MPI_ANY_TAG,stats) ;
      if(stats.MPI_TAG == 'f'){
         count++;
      }
```

```
      else{
      // Calculate left and right boundary point index in Vector bp
          corresponding to stats.MPI_SOURCE
      if(bp[left_pop_index].get_size() > 0){
         temp_bp[left_pop_index] = bp[left_pop_index].pop();
      }
      if(bp[right_pop_index].get_size() > 0){
         temp_bp[right_pop_index] = bp[right_pop_index].pop();
      }
      // Copy the value of temp_bp[left_index] and
          temp_bp[right_index] to the a temporary array.
      MPI_Send(...);// Send the value to stats.MPI_SOURCE
      MPI_Receive(...);// Complete the receive process;
      // Calculate left and right boundary point index in Vector bp
          corresponding to stats.MPI_SOURCE
      bp[left_push_index].push(left_val);
      bp[right_push_index].push(right_val);
      }
      if (count == (numprocs - 1)){
         break;
      }
   }

}
else{
   // Accept the value of boundary points from rank 0 at the
       initial condition
   for (t = 0; t < final_t; t += delta_t)
   {
      // Calculate boundary nodes value.
      MPI_Send(...) // Send boundary node to rank 0 with tag = rank
      Calculate the value at the interior nodes.
      MPI_Recv(..) // Receive the value from rank 0
   }
   MPI_Send(..) // Send value with a special tag 'f' to rank 0
}
```

The time-line generated with the help of Vampir Trace following this approach is shown in Fig (4.1).

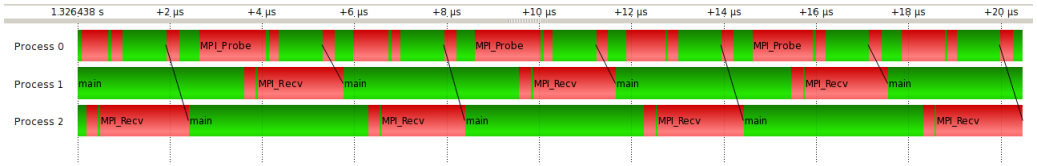


Figure 4.1: Vampir Trace with Grid Point 512. The arrow in the Trace indicates the line from Sender to Receiver. The communication between any two processor for the boundary nodes occur through the Master Processor.

4.3.2 Asynchronous Computation Using OMP

Here, the idea is the same as in the Section (4.3.1), except in this case the role that is played by the *Master* processor will be played by the Critical Section. In the above multi-threaded application the `Vector<queue> bp` is in the shared region. This contains the values of the boundary points at different time layer as in the previous case. Since different processor may try to read and write the shared memory buffer `bp` at the same time, it forms a critical section.

Approach 2:Asynchronous Computation Using Shared Memory

```
//Calculate for the initial Condition
Vector<queue> bp[2*(numprocs - 1)]; // Vector of queues
    corresponding to each boundary points in the shared section.
//Push all the boundary nodes to the Vector bp at the time level 0
#pragma omp parallel
{
   //Calculate left_push, right_push, left_pop ,right_pop index
   #pragma omp critical
   {
      if(bp[left_pop].get_size() > 0){
         left = bp[left_pop].pop();
      }
      if(bp[right_pop].get_size() > 0){
         right = bp[right_pop].pop();
      }
   }
   for(t = 0; t < final_t; t++){
      for(i = 0; i < ne; i++){
```

```
            // Carry out the calculation according to scheme
        }
        #pragma omp critical
        {
            if(bp[left_pop].get_size() > 0){
            left = bp[left_pop].pop();
        }
        if(bp[right_pop].get_size() > 0){
            right = bp[right_pop].pop();
        }
        bp[left_push].push(left_val);
        bp[right_push].push(right_val);
        }
    }
}
```

In both of the approach mentioned in Section 4.3.1 and Section 4.3.2, the problem of scalability arises. If the simulation is ran with a large amount of processors, the delay is going to increase. In Section 4.3.1 more processes will try to write to the master process due to which there will be more waiting time incorporated by the processor. In Section 4.3.2, with increase in the number of processor more processor will try to enter the critical section leaving the other process to wait till it completes the process. Thus, it is becomes important to avoid such a situation.

4.3.3 Point to Point Communication using MPI

In this approach there is a direct data transfer between the two processor without the involvement of any third processor as was in Section (4.3.1). The time-line generated with the help of Vampir Trace following this approach is shown in figure (4.2).

```
// Carry out the initial condition calculation
if(rank == 0){
   step = 0;
   for(t = 0; t < final_t; t+= delta_t){
      step++;
      MPI_Irecv(...) // non-blocking receive for the nodes with tag
          = 'step'
      // Calculate only for the boundary nodes
      MPI_Send(...) // writes in the buffer with tag = 'step'
      calculate() // calculate for the other nodes.
      MPI_Test(For request to complete);
      if(requset not completed){
         MPI_IProbe(..); // non-blocking probe with tag = step
         if(flag == true){ // Data in buffer is available
            MPI_Wait();
         }
      }
   }
}
else{
   // Same for the other processors
}
```

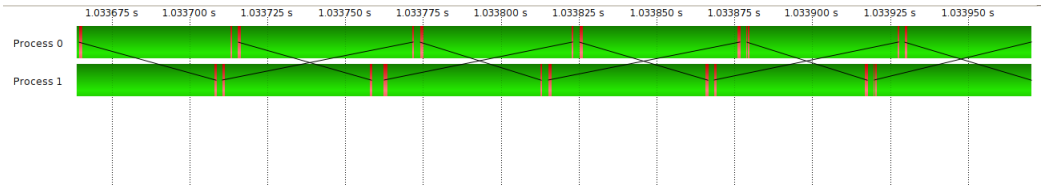


Figure 4.2: Vampir Trace with Grid Point 512. The arrow in the Trace indicates the line from Sender to Receiver. There is a direct transfer between the two processor for the exchange of the boundary nodes between the two processors without the involvement of the third processor.

This approach remove the drawback of the previous two approaches as this doesn't involve anything such as "critical section" or a common processor through which all different processors communicate. As the

number of processor increases, only two of the neighbouring boundary nodes will be involved. So, we can get rid of the problem of scalability that we discussed earlier. From now on, we will use this approach for our analysis and discussion.

4.4 Asynchronous Implementation

Asynchrony can be implemented in two ways. The first method is based on the idea that we do not exchange the value at the boundary nodes at each step rather than we exchange the values after the certain amount of step, known as `SYNC_STEP`. This, in turn, will decrease the time required for the synchronization. The another way to implement the asynchronous scheme is that we do not wait for the communication to finish rather than continue the calculation with the latest available values. The former can be called as Deterministic Asynchronous Implementation while the later can be called as the Stochastic Asynchronous Implementation. The details are discussed in the following Section.

4.4.1 Deterministic Asynchronous Implementation

In this type of implementation, the exchange of data takes place after a certain `SYNC_STEP`. `SYNC_STEP` = 1 refers to the Synchronous Case. It is called Deterministic because of the fact that the error do not defer if we run the same simulation different time. The error is not dependent on the hardware or network statistics. In this approach our main aim is to decrease the overall time needed for synchronization (as now we do not enforce the Synchronization after every time step rather than after certain `SYNC_STEP`) and study the effect on error. The study is performed by considering the various test cases. All the simulations are performed on MPI-l or MPI-s of RWTH IT Cluster [8]. Before proceeding further, let us define a parameter Speedup as the ratio of time taken by synchronous scheme to the time taken by the time taken under relaxed synchronization, given by Eqn.(4.1). The objective of defining the parameter in such a way is to study the effect of synchronization.

$$Speedup = \frac{\text{Time taken by Synchronous Scheme}}{\text{Time taken by scheme under relaxed synchronization}} \tag{4.1}$$

Test case 1

Our aim is to study the effect of `SYNC_STEP` on time and error. For this purpose, we have used 4096 grid points and distribute it onto 8 processors and varied the value of `SYNC_STEP`. Final time is kept as $0.08 * len/c$ and Courant number r_α is taken to be 0.1. The results are shown in Figure 4.3.

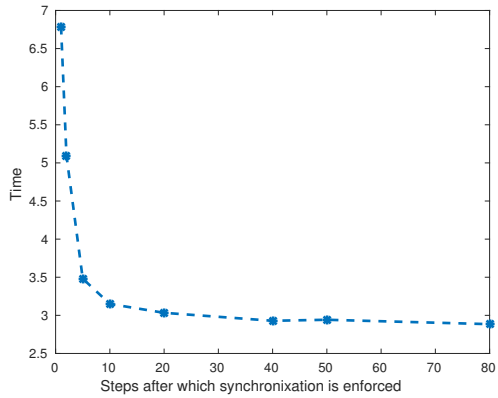
As can be seen from Figure 4.3a, the time first decreases rapidly and then after a certain time it has minimal effect on increasing the synchronization step. To formally describe the reason for this, let us consider that the computation time for carrying out the calculation only (ignoring the time required for synchronization and message passing) is **X** seconds and the Synchronization cost associated with each Synchronization is **Y** seconds. If the total number of time step is N_T, then number of times the Synchronization takes place is $\frac{N_T}{\texttt{SYNC\_STEP}}$ and the total cost of synchronization will be $\frac{N_T}{\texttt{SYNC\_STEP}}Y$. Therefore total cost of computation equals to $X + \frac{N_T}{\texttt{SYNC\_STEP}}Y$. Hence the speedup (defined according to Eqn (4.1)) will be given by Eqn (4.2).

$$Speedup = \frac{X + N_T Y}{X + \frac{N_T}{SYNC\_STEP}Y} \tag{4.2}$$

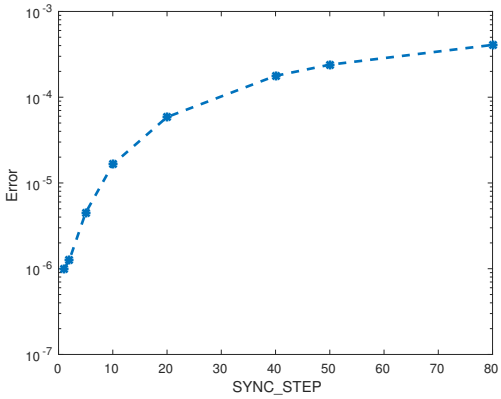
The following observation can be made from our discussion:

- Higher the number of SYNC_STEP, lesser will be the cost of Synchronization.
- Higher the value of X means the calculation cost is high, i.e. more load on the processor.
- If X is very high as compared to Y, i.e. the calculation cost is very high compared to synchronization cost, then increasing the value of SYNC_STEP do not do not solve much of our purpose. We will not receive much Speedup in such case even by increasing the value of SYNC_STEP.

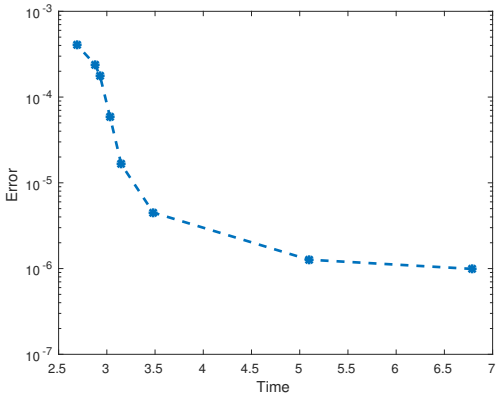
As we increase the value of SYNC_STEP, the computation cost remains same and synchronization cost decreases. But after a particular amount of time, when computation cost becomes much higher than the synchronization cost, increasing the SYNC_STEP do not contribute much to the Speedup. Thus, we can only receive speedup upto a particular value.



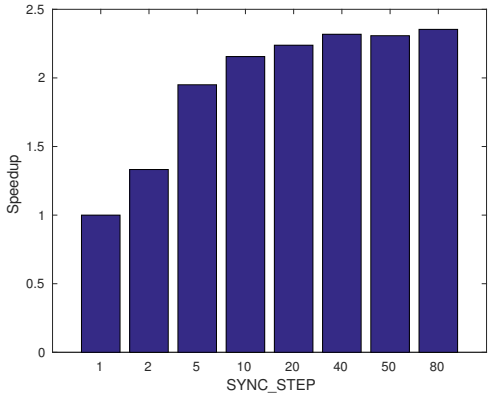
(a) Variation of Time with respect to SYNC_STEP



(b) Variation of Error with respect to SYNC_STEP



(c) Time Vs Error Plot



(d) Speedup Vs SYNC_STEP

Figure 4.3: Result of Deterministic Implementation with Grid Points 4096 on 8 processor.

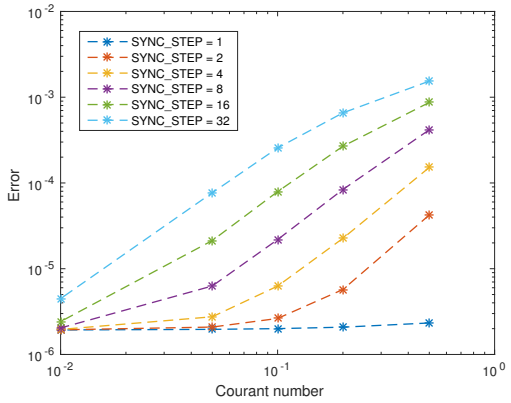
This effect is observed in Fig 4.3d, where we observe the speedup increases on increasing the SYNC_STEP and becomes constant after a particular value. Figure 4.3c gives the plot of the Time required for the computation Vs the Error. This indicates the trade-off between error and time. In practice, it depends on the practical purpose of how much accuracy is desired. For example if the desired accuracy is of order 10^{-5}, then instead of Synchronizing at every step we can go for SYNC_STEP ≈ 10, as seen from 4.3b, resulting in a speed up of ≈ 2.

Test case 2

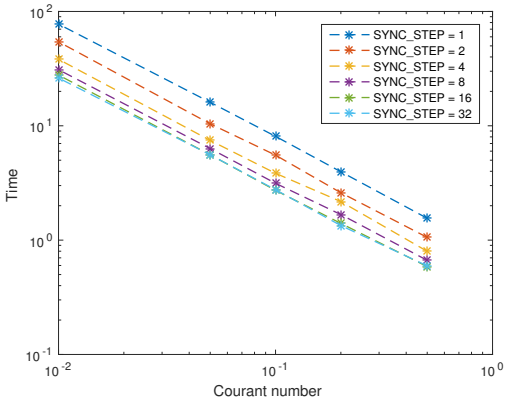
Through this test case, our objective is to observe the effect of Courant number r_α for Synchronous and Asynchronous Case using Deterministic Asynchronus Implementation and the effect of increasing the load on each processor. Changing r_α keeping the grid size Δx fixed means changing the value of Δt. For this purpose, we used 4096 and 16384 grid points and distribute it on 16 processors. Final time is kept as $0.08 * len/c$.

The results of this test case is shown in Fig 4.4 and Fig 4.5. The following observation can be made

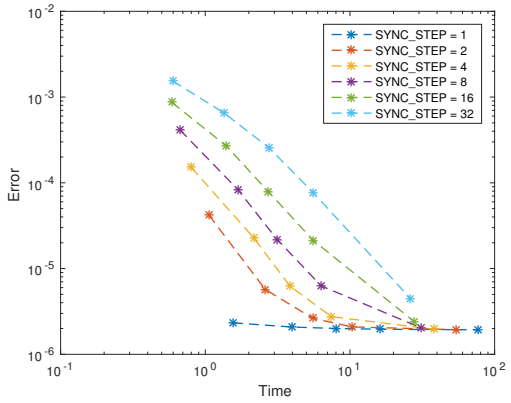
- Higher the value of SYNC_STEP, higher the error, as seen from Fig 4.4a and Fig 4.5a.
- As we know the standard Central Difference Scheme, the Order of Convergence is O(Δt, Δx^2). In our formulation, $O(\Delta t) \leq O(\Delta x^2)$, as $\Delta t \leq r_\alpha \frac{\Delta x^2}{\alpha}$. Due to this the overall truncation error, given by Eqn (2.28) is governed by O(Δx^2) and changing the value of Courant number doesn't have much effect for Synchronous Scheme.
- For Asynchronous Scheme, higher the Courant Number, higher will be the error.
- Higher the load on each processor, i.e. more is the value of $\mathbf{X}$, less is the speedup observed, as seen from Fig 4.4d and Fig 4.5d beacuse of the reason discussed in the previous Test Case.
- Higher the value of Courant number, less is the time required, as seen from Fig 4.4c and Fig 4.5c. It is simply because of the reason that higher value of r_α means less number of total time step N_T.
- Speedup is independent of r_α.



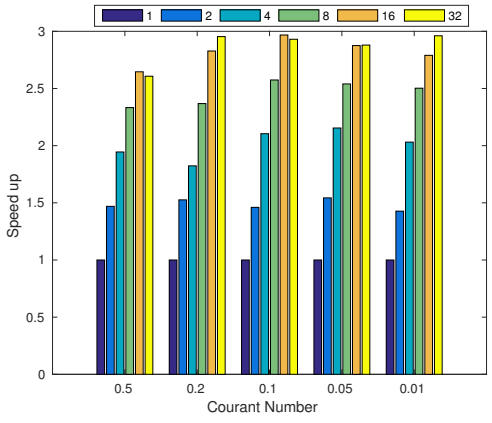
(a) Error Vs Courant Number for various values of SYNC_STEP



(b) Time Vs Courant Number for various values of SYNC_STEP

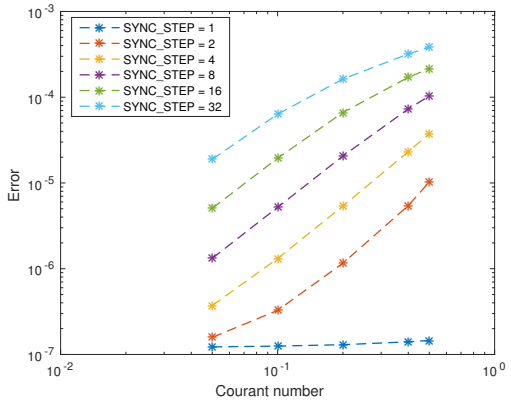


(c) Error Vs Time Plot for various values of SYNC_STEP

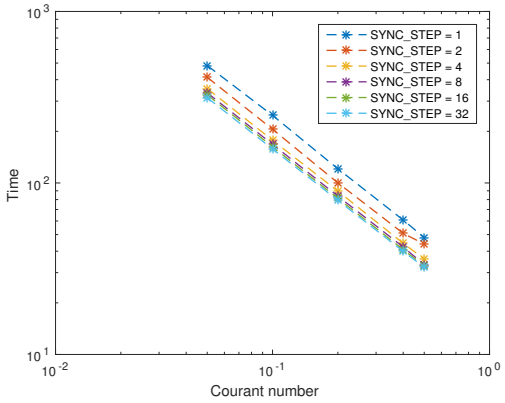


(d) Speedup Vs SYNC_STEP for various Courant Number

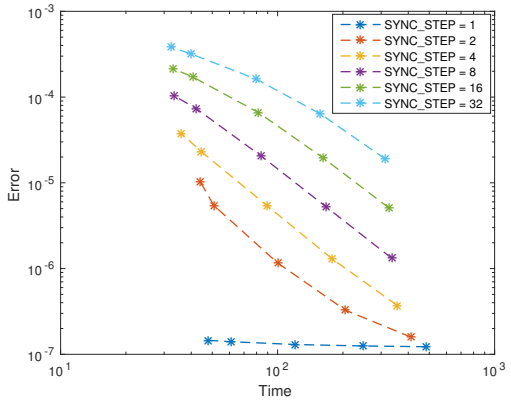
Figure 4.4: Result of Deterministic Implementation with 4096 Grid Points on 16 processor.



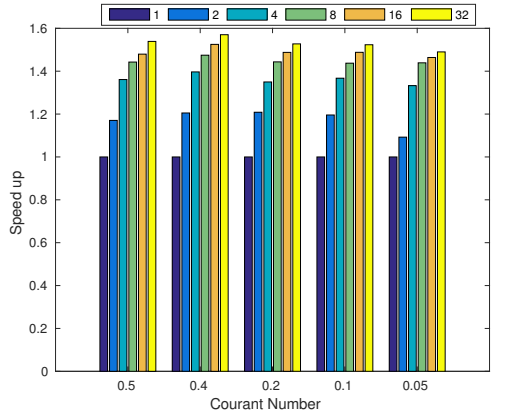
(a) Error Vs Courant Number for various values of SYNC_STEP



(b) Time Vs Courant Number for various values of SYNC_STEP



(c) Error Vs Time Plot for various values of SYNC_STEP



(d) Speedup Vs SYNC_STEP for various Courant Number

Figure 4.5: Result of Deterministic Implementation with 16384 Grid Points on 16 processor.

4.4.2 Stochastic Asynchronous Implementation

In this Section, we will discuss the second method to implement the Asynchronous Case. In this method, we do not enforce any synchronization and the computation for the next time step begins on the latest value available. Before discussing the implementation into detail, let us discuss about the Delay Statistics.

Delay Statistics

MPI provide two ways for the communication to happen as discussed in the Section 4.1, via `MPI_Send` and `MPI_Recv` or `MPI_Put` and `MPI_Get`. In this Section, we will discuss how these two implementation effect Delay Statistics. Before we discuss our discussion about Delay Statistics, let us define a parameter Mean Delay k^* given by Eqn (4.3), where k_i is defined as the ratio of the number of Time Steps that faced Delay = i and total number of Time Steps.

$$k^* = \sum_{i=0}^{i=\infty} i * k_i \tag{4.3}$$

Test Case

In this test case, our aim is to study the effect of communication on the Delay statistics. For this purpose, we took 1073741824 grid points and distributed among 4 and 8 processors and observed the delay statistics on both left and right boundary nodes. The results presented are shown in Fig: 4.6 - 4.9. It must be noted that all the processors are placed on same node. The details of the impact of Computer Architecture is discussed in Section 4.5. Based on the results, the following observations can be made:

- Delay statistics is random. Different processors faces different nature of Delay.
- Buffering of message is observed in case of two-sided communication. The buffering nature is clear from the fact that both the two boundary of the neighbouring processor faces delay. For example in Fig 4.8a, left boundary of PE5 and right boundary of PE4 both faces delay. Since, at one time we are just sending 16 bytes of data (2 double values),

according to MPI documentation, MPI_Send behaves as MPI_Bsend [5]. It packs the data into a buffer that is sent at a later stage.

- Buffering is not observed in case of RMA operation. If the left boundary of any processor faces delay, then right boundary of the previous processor do not faces delay. This was the basis that we considered in Section 3.2.3.

- As the load decreases and communication starts happening more frequently, more asynchrony is observed. This can be observed if we compare the delay statistics of Fig 4.6b with Fig 4.8b and Fig 4.7b with Fig 4.9b.

- MPI RMA operation outcast the MPI two sided communication in terms of mean delay that is observed on different PEs.

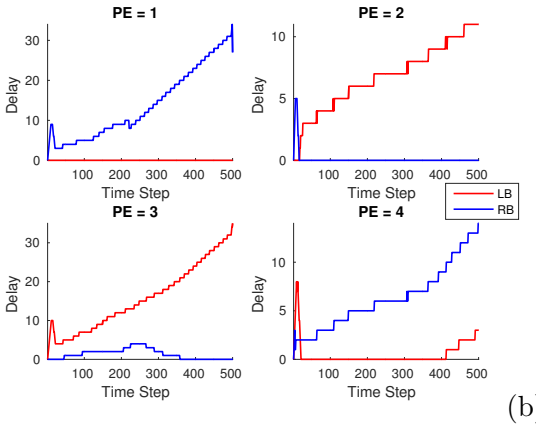
- Some of the delay just go on increasing with the number of time step. This can be due to the difference in relative speed of the neighbouring processor for the particular run.

Based on the above made observation, in order to implement the Stochastic Asynchronous Scheme, there is a need of following requirements:

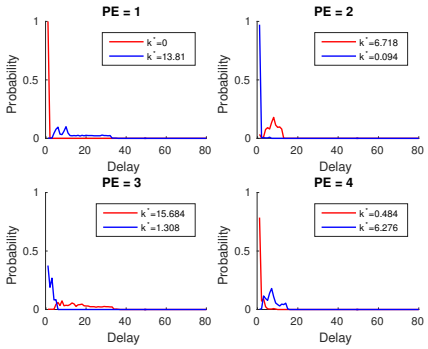
- Since, at any particular time, different processor may be calculating the value at different time level. If the processor is facing delay, then it means that it is calculating at a certain time step ahead of what its neighbouring processors is calculating. Let us consider a scenario, Processor i and Processor j are neighbouring processor, namely P_i and P_j. P_i is calculating the value at n_1 time step and P_j at n_2 ($n_1 < n_2$),i.e P_j is facing Delay. To calculate the value at current time step n_2, P_j requires the latest value from P_i but for P_i the required value is at time step n_1 which is already been calculated and sent to it at any earlier stage. Thus, we need a buffer to accommodate such values at previous time step. We do not want any future values to calculate. This can be accomplished in case of two sided communication by array of `MPI_Request` and in RMA operation by assigning a buffer in the Window by `MPI_Alloc_Mem` and `MPI_Win_Create`.

- Since some of the error grows as the number of time step increases, it becomes a necessity to construct a control knob, such that the

calculation will only proceed when the value of $\tilde{n} \geq n - L$, where $\tilde{n}$ is the latest time step value for which the value from neighbouring PE is available and n is the current time step for which that particular PE is calculating. This ensures that the calculation will only proceed if the value from any of the previous L step is available. If not, partial/total synchronization must be enforced. Here L can be seen as the Maximum Allowable Delay. The decision of the value of L depends on the practical application of our interest.



(a) Evolution of Delay for 500 time step.



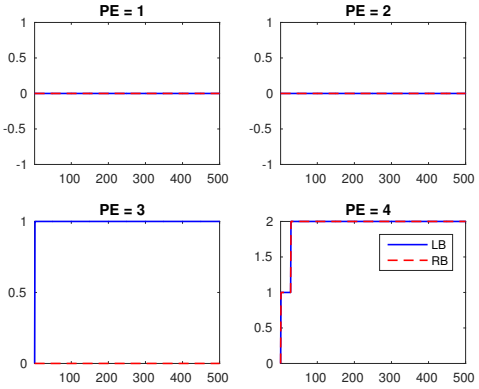
(b) Delay Distribution faced by left and right boundary points.

Figure 4.6: Number of Processor = 4. Number of Grid Points = 1073741824. Nature of Communucation: MPI two-sided. Each process has two boundary point. LB denotes the delay faced by left boundary point and RB the right boundary point

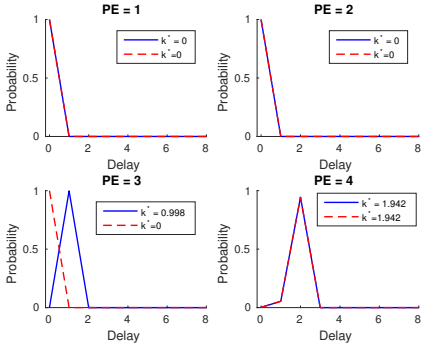
Test Case

Through this test case, we tend to study the effect of Maximum Allowable Delay on Error and time and measure the Speedup in the similar way we described for the Deterministic Case. For this purpose, we considered 65536 Grid Points and distributed on 32 processors. Final time was kept to be $1e-4$ seconds. Courant number r_α was set to be 0.1.

The result is shown in Fig 4.10. Since this is a Stochastic Case, different run leads to different results. For the averaging purpose, 10 runs of each case was carried out. Based on the result, following observation can be made:



(a) Evolution of Delay for 500 time step.



(b) Delay Distribution faced by left and right boundary points.

Figure 4.7: Number of Processor = 4. Number of Grid Points = 1073741824. Nature of Communucation: MPI Two-Sided. Each process has two boundary point. LB denotes the delay faced by left boundary point and RB the right boundary point.

- Higher the value of Maximum Allowable Delay, higher will be the error and lesser will be the time.
- In an ideal case, when computation and communication overlap perfectly, the maximum speedup that can be gained is $L + 1$, where L represents the Maximum Allowable Delay. [9]
- From the Time Vs Error plot Fig 4.10a, it can be observed that time first decreases abruptly and later it assumes a constant value. Increasing the Maximum Allowable Delay do not serve the purpose. This is the same behaviour we observed in the Deterministic Case.

4.5 Impact of Computer Architecture

Let us define us some terminologies before going further.

- **Chasis**: The "chassis" is that thing that houses one or more compute nodes.
- **Nodes**: A certain amount of memory (RAM) is physically allocated on each node. Nodes are typically separated by network connections. Each node has a unique network address (host name/IP number).
- **Socket**: Socket refers to collection of cores with a direct pipe to memory. A socket holds one processor.
- **Core**: Core refers to a single processing unit capable of performing computations. A core is the smallest unit of resource allocation. Each core has its own L1 cache chip.

With these formal definition, let us consider the implementation of Stochastic Asynchronous Scheme with data being distributed across various nodes.

Test Case

Consider a 1-D case with 1024 grid points distributed among 8 processors. The simulation is run 150000 time steps with RMA communication being used. Various distribution of processors across various nodes are considered. The results is observed in the Table 4.1. Average Delay is defined as the Average of Mean Delay (Eqn: 4.3) observed by all the processor.

Processor per node	Minimum Time	Maximum Time	Average Time	Maximum Delay	Average Delay
8	1.632993	1.898601	1.747674	26729	1152.118219
4	1.306848	2.515238	1.935257	63719	595.315498
2	2.310587	2.394653	2.357168	5074	23.086279
1	2.575294	2.636949	2.604869	1912	49.434085

Table 4.1: Result of Test Case with 1024 grid points distributed among 8 processor

We observe that if we consider the processors that are not on the same node, the total time for simulation increases. This is due to the fact that communication across the nodes is costly as compared to the communication within the same node.

Test Case

In this case, we cinsider a 1-D problem with 268435456 grid points distributed among 8 processors. The simulation ran for 500 time steps with RMA and two-sided non blocking communication. Various distribution of processors across various nodes are considered. The results is observed in the Table 4.2 and Table 4.3.

Processor per node	Minimum Time	Maximum Time	Average Time	Maximum Delay	Average Delay
8	196.022283	210.897438	204.288266	38	0.152375
4	157.093253	168.151056	162.511951	42	0.013750
2	112.703603	126.833151	122.102558	44	0.075125
1	91.371902	96.440170	93.667287	20	0.132000

Table 4.2: Result of Test Case with 268435456 grid points distributed among 8 processor with RMA communication

Processor per node	Minimum Time	Maximum Time	Average Time	Maximum Delay	Average Delay
8	143.154700	158.406454	151.922679	65	10.797625
4	95.791654	137.700051	112.257639	123	20.124625
2	85.375067	141.165011	111.381462	137	17.850750
1	68.627403	105.315875	84.915095	110	13.006375

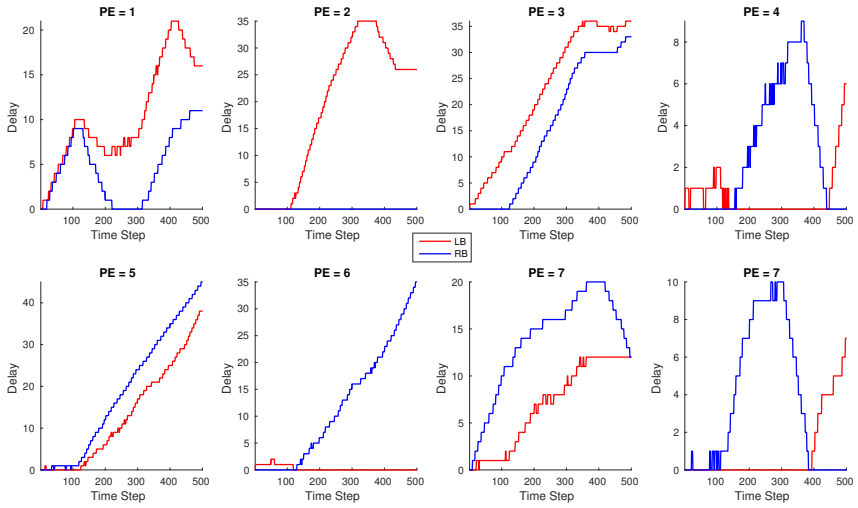
Table 4.3: Result of Test Case with 268435456 grid points distributed among 8 processor with MPI two sided non-blocking communication

In this case, we observe that the trends get reverse if compared to the previous case. The answer for this lies in the memory bandwidth that we can get if all the process is distributed on one node. Each MPI process holds a copy of several memory objects, and if two MPI processes run on the same node, more memory will be allocated on that node, compared to

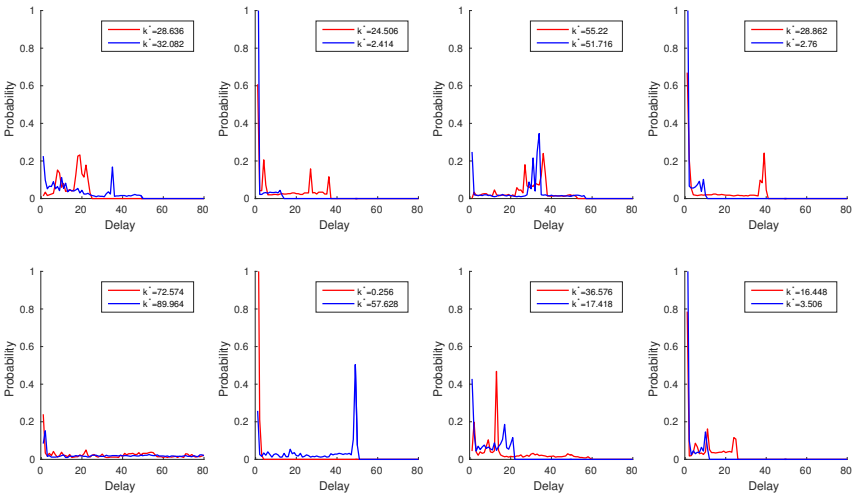
if the job was run in serial. For example, if a certain calculation uses 1.0 Gb of RAM in serial, assigning four MPI processes to a node will use 4.0 Gb of RAM on that node. Thus, running independent MPI processes on all the cores on a particular node at once will force them to compete for memory access in a way that actually can cause the calculation to run slower. Thus, it is recommended for the large data to split the solution over various nodes to gain the performance. [10, 11] With respect to average delay, here again we observe that the MPI RMA operation outperforms MPI two sided non-blocking communication. However, no formal conclusion can be drawn regarding the effect of distribution of processor across various nodes on Delay Statistics.

4.6 Concluding Remarks

In this chapter we discussed about the implementation aspects of Asynchronous Scheme via two ways: Deterministic and Stochastic Asynchronous Scheme using Point to Point communication via MPI. There can be two ways for the communication in MPI to happen. One is through two-sided non-blocking whereas other is through RMA operation. Based upon the observation from our test cases, RMA operation outperforms the two sided non-blocking communications. In last we have seen the impact of Computer Architecture on Performance. The decision of the processes to the distributed over various nodes depend upon the size of the data that a particular node is going to handle.

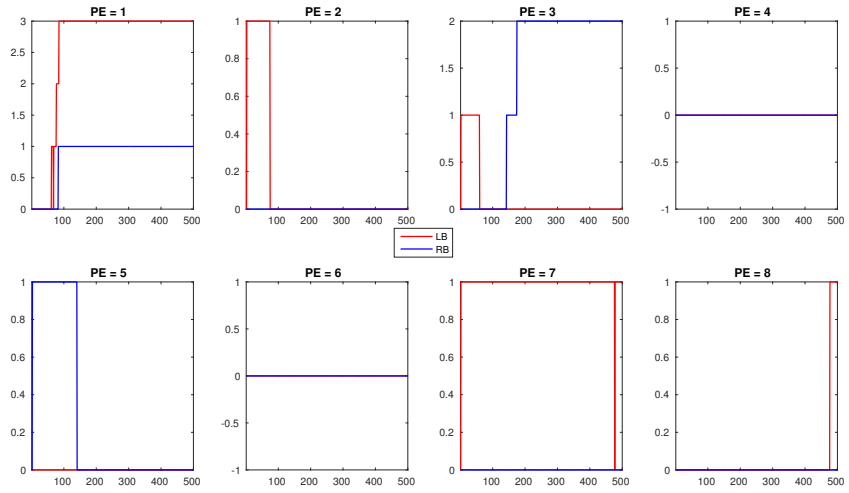


(a) Evolution of Delay for 500 time step. Each process has two boundary point. LB denotes the delay faced by left boundary point and RB the right boundary point

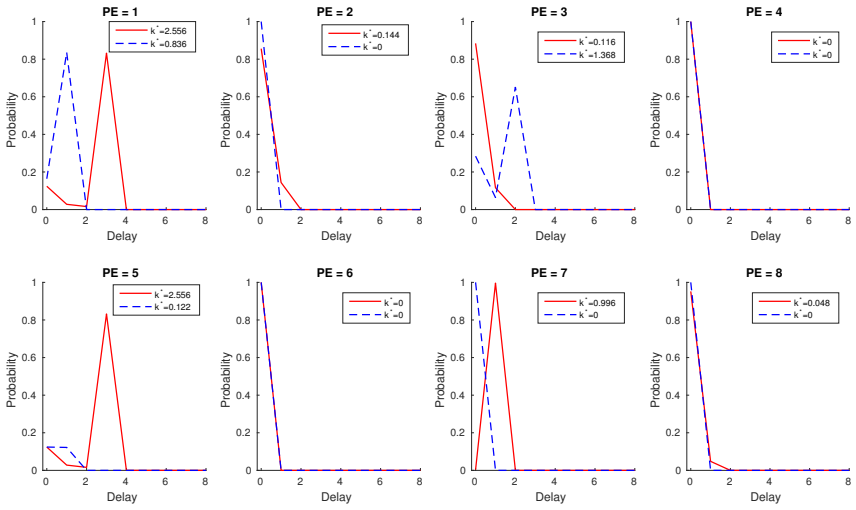


(b) Delay Distribution faced by left and right boundary points.

Figure 4.8: Number of Processor = 8. Number of Grid Points = 1073741824. Nature of Communucation: MPI RMA.

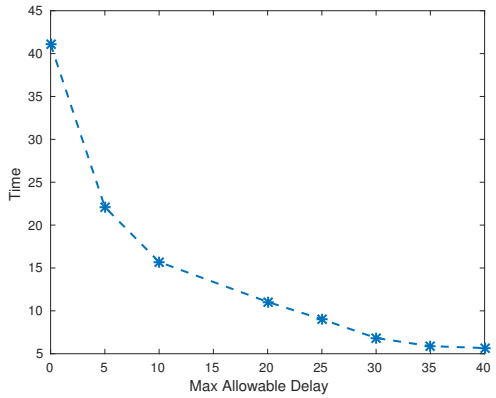


(a) Evolution of Delay for 500 time step. Each process has two boundary point. LB denotes the delay faced by left boundary point and RB the right boundary point

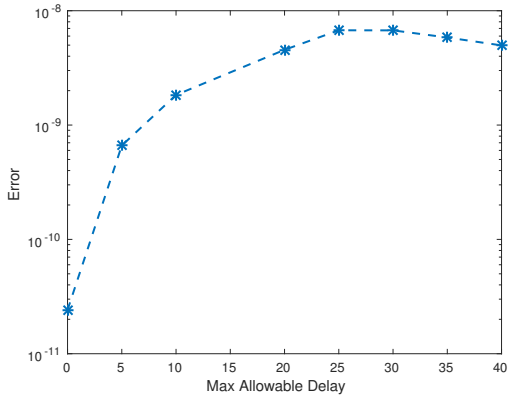


(b) Delay Distribution faced by left and right boundary points.

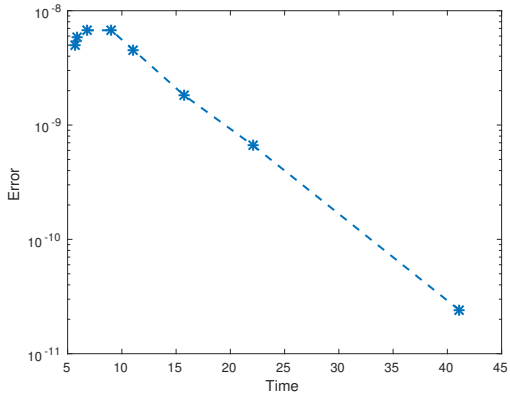
Figure 4.9: Number of Processor = 8. Number of Grid Points = 1073741824. Nature of Communucation: MPI One-Sided



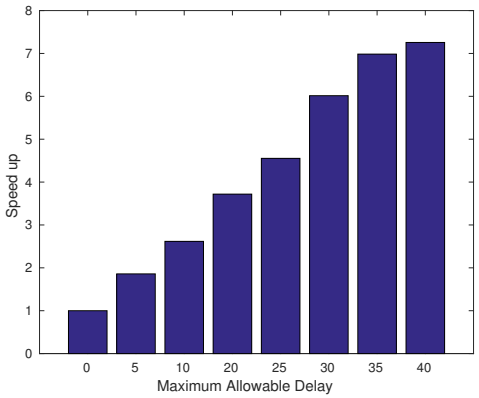
(a) Effect of time on Max Allowable Delay



(b) Variation of Error with respect to Max Allowable Delay



(c) Time Vs Error Plot



(d) Speedup observed with respect to Max Allowable Delay

Figure 4.10: Result of Stochastic Implementation with Grid Points 65536 on 32 processor.

Chapter 5

Conclusion

This thesis investigated the Asynchronous Scheme in case of Advection-Diffusion equation. In massively parallel computing, the problem of synchronization forms a major bottleneck. If the current trend continues, the number of processors tends to grow at a much faster rate as compared with the speed of communication between the processor. Thus to eliminate the ideal timings that the processor faces due to the communication delay, we used the asynchronous framework where we tend to reduce the waiting time on the processor.

The test cases were used to validate the theoretical analysis by simulating the artificial delay and later, the implementation on the real computers using parallel libraries were performed. The major contribution from the thesis can be summarized as follows:

- The Asynchronous Scheme remains consistent if the original scheme is consistent. However, if the standard Central Difference Scheme is used to discretize the Advection-Diffusion Equation, the order of Convergence drops to first order in case of strong scaling and zero order in case of weak scaling.

- To recover back the order, the new Scheme is proposed which used a new Discretization Scheme only for the boundary nodes. The resulting Scheme is capable to recover order under the condition of no message buffering. In case of buffering, it is capable of recovering back the second order to some extent but it falls back to first order with increasing order of delay. However, the magnitude of error falls with the use of this new scheme at the boundary nodes.

- Asynchrony can be implemented as Deterministic and Stochastic Asynchronous Scheme. The speedup obtained in both the case depends on the load that is available for each of the processors. If each of the processor has high load, i.e. it has more computations to perform between each synchronization than the asynchronous scheme gives less speedup as compared when each of the processor has lower load. In the later case, the effect of Synchronization becomes dominant and thus the Asynchronous Scheme proves to be advantageous.

- Stochastic Asynchronous Scheme takes into account the delay statistics which is dependent on hardware, network topology and other factors. The implementation of this scheme is performed using two-sided non-blocking communication and RMA operation. RMA operation was found to outcast two-sided non-blocking operation in all case.

- The communication cost among the processor lying within the same node is low as compared to the case when the communication happens among the different processors. In case of very high load, the problem of memory latency too arises and it dominates the communication across nodes. Thus, for cases where the memory requirement is high it becomes advantageous to distribute the process on different nodes rather than a single node.

Bibliography

[1] Maitham Alhubail, Qiqi Wang. The swept rule for breaking the latency barrier in time advancing PDEs.

[2] Diego A. Donzis and Konduri Aditya. Asynchronous Finite Difference Scheme for Partial Difference Equations *Journal of Computational Physics*. 274(0):370-392,2014

[3] Thomas Camminady. CES Seminar Paper on Asynchronous Finite Difference Scheme for Partial Difference Equation. January 9,2015

[4] Anderson, John D., Jr., Computational Fluid Dynamics, McGraw-Hill, New York, 1995

[5] MPICH , http://www.mpich.org/, 4 12 2015.

[6] Ricky A. Kendall *et al.*, Parallel Programming Models Applicable to Cluster Computing and Beyond

[7] Using OpenMP Portable Shared Memory Parallel Programming, Barbara Chapman, Gabriele Jost and Ruud van der Pas, MIT Press

[8] RWTH IT Cluster, https://doc.itc.rwth-aachen.de/display/CC/Hardware+of+the+RWTH+Compute+Cluster

[9] Dheevatsa Mudigere *et al.* Delayed Difference Scheme for Large Scale Scientific Simulations. *PHYSICAL REVIEW LETTERS*

[10] https://www.quantumwise.com/documents/tutorials/latest/ParallelGuide/XHTML/chap.strategy.html#sect2.strategy.technologies

[11] Dominique LaSalle and George Karypis, MPI for Big Data: New Tricks for an Old Dog.