High Performance Computing for Multiphase Fluid Flows

Bipin Kumar
Master of Science(Research) in Mathematics

A thesis submitted in the fulfilment
of the requirement for the degree
of
Doctor of Philosophy (Ph.D.) in Computing
to the

Dublin City University

Faculty of Engineering and Computing, School of Computing

Supervisors:

Dr. Martin Crane
School of Computing

Dr. Yann Delaure
School of Mechanical and Manufacturing Engineering

October 2009

High Performance Computing for Multiphase Fluid Flows

Bipin Kumar
School of Computing,
Dublin City University, Ireland

Abstract

Multiphase fluid flows are very common in engineering and science applications. Examples include air flow on water surface, metallurgical flow and blood flow in the body. In these flows, fluids are separated by a sharp interface and form different phases. The flow is characterized by the movement of this interface. Accurate modelling of the interface movement is a fundamental problem in the numerical simulation of these flows. Velocities for the movement are provided by the numerical solution of the Navier-Stokes (N-S) equations. These equations are discretized and converted into linear systems of equations. Research in the direction towards solving these systems efficiently has been the main focus of many researchers in the field of Computational Fluid Dynamics (CFD).

A modified Volume of Fluid (VOF) method for modelling two phase flows is implemented using an analytic relation for its reconstruction step. The Finite Volume Method (FVM) is utilized, by incorporating a staggered grid, to discretize the two-dimensional (2-D) N-S equations. A preconditioned Krylov-Subspace iterative method, namely, the Bi Conjugate Gradient Stabilized (Bi-CGSTAB) method is employed to solve the linear systems of equations. Solving the linear system usually consumes most of the simulation time for multiphase flow problems. Novel algorithms for the Incomplete LU Threshold (ILUT) preconditioner, forward and backward substitution and other matrix operations for penta-diagonal matrices are proposed here by adopting a diagonal sparse matrices format. The novel algorithm for ILUT reduces the computational complexity from $\mathcal{O}(n^3 - n^2)$ to $\mathcal{O}(n)$ in comparison to dense format. Further, it brings down the communication overhead, consequently facilitating parallelization. Parallel versions of these algorithms are developed using a new load balancing scheme. The MPI C++ communication library is utilized to develop the parallel version.

The 2-D VOF code is applied to shape advection problems and results are found

in good agreement with those available in literature. In the case of translation of a square box, it provides more accurate results than other VOF methods. The code for the VOF method and the parallel iterative solvers are integrated with 2-D N-S code in C++. The whole code is then implemented to simulate several two phase flow problems: dam breaking with and without an obstacle, rising of an air bubble and lid driven cavity flows. Speedup data from parallel programs implemented on these problems are generated.

Acknowledgements

First of all, I would like to express heartfelt gratitude and sincere appreciation to my supervisors Dr. Martin Crane and Dr. Yann Delaure for their guidance full with many valuable suggestions, simulating ideas and invaluable advice throughout my research period at Dublin City University (DCU). Their wide knowledge and logical way of thinking have been of great value for me from the initial to the final level and enabled me to develop an understanding of the subject. It has been my pleasure being their research student.

Special thanks are due to Abhijeet Battacharya, Jai Prakash Mehta, Sourabh Mukhopadhyay and Rattna Datta for devoting their precious time and excellent advice. I am grateful to Suresh Uppal and his wife Priyanka Uppal who have made available their support in a number of various ways. I also wish to thank all my friends for their assistance. Talks held with them will always remain in my thoughts.

My deepest gratitude goes to my parents Ram Chandra and Shusheel Devi Porwal and elder brother Pavan Kumar who always encouraged me and kept away from family responsibilities. My special thanks are due to my sister Jyoti Porwal for her loving support.

I am very grateful to the funding sources Faculty of Engineering, DCU to support me for three years. My thanks are due to Computer Science Department to give access to parallel computer that made my Ph.D. research possible.

Finally, I would like to express my loving thanks to my wife Preeti Porwal for her love and encouragement. She always helped me to concentrate on completing this dissertation and supported mentally throughout this course. Without her help and encouragement, it would have not been possible for me to complete this study.

October 2009
Dublin City University

Bipin Kumar

List of abbreviations and symbols

CFD :	Computational Fluid Dynamics
ρ :	density
$\mathbf{V}$:	velocity vector
u :	x component of velocity vector
v :	y component of velocity vector
x, y :	components of two dimensional Cartesian coordinate system
p :	pressure
t :	time
Γ :	viscosity
S_v :	body forces (source term)
C :	a scaler function (called here as colour function)
$C_{i,j}$:	value of volume of fluid (i.e. colour function) in a cell at location i, j
C_r :	Courant number
δt :	time step
δx :	length of cell in x-direction
δy :	length of cell in y-direction
ϕ :	unknown variable in flow problem
CV :	Control Volume
FVM :	Finite Volume Method
FEM :	Finite Element Method
$P_{i,j}$:	pressure at a grid point(i, j)
SIMPLE :	Semi Implicit Method for Pressure Linked Equation
n^{th} :	represents the iteration level
n :	size of a vector
$\mathbf{A}$:	n×n matrix
$\mathbf{x}$:	vector of length 'n'
$\mathbf{B}$:	vector of length 'n'
CSC :	Compressed Sparse Column
CSR :	Compressed Sparse Row
$\mathbf{r}$:	residual vector (defined as $\mathbf{r} = \mathbf{B} - \mathbf{Ax}$)
K_m :	Krylov Subspace
CGM :	Conjugate Gradient Method
Bi-CGM :	Bi-conjugate Gradient Method

Bi-CGSTAB :	Bi-conjugate Gradient Stabilized
$\mathbf{A^T}$:	Transpose of matrix A
λ_{max} :	Maximum eigen value of a matrix
λ_{min} :	Minimum eigen value of a matrix
$\kappa(\mathbf{A})$:	condition number of matrix A
$\mathbf{L}$:	lower trianguler matrix
$\mathbf{U}$:	upper trianguler matrix
ILU :	Incomplete Lower and Upper trianguler Factorization
ICH :	Incomplete Cholesky Factorization
DS :	Diagonal Scaling
SSOR :	Symmetric Successive Over Relaxation
τ :	threshold value for fill-in ILU Factorization
FCT :	Flux Corrected Transport
PDE :	Partial Differential Equation
N-S :	Navier-Stokes Equations
CICSAM :	Compressive Interface Capturing Scheme for Arbitrary Meshes
SLIC :	Simple Line Interface Calculation
PLIC :	Piecewise Linear Interface Calculation
DDR :	Defined Donating Region
DDAR :	Defined Donating and Accepting Region
Re :	Reynolds number
PCM :	Pseudo Concentration Method
CPU :	Central Processing Unit(of a computer processor)
GHz :	Gega Hertz (represents the speed of CPU)
S_n :	Speedup
T_s :	Time taken by serial version of a computer code
T_p :	Time taken by parallel version of a code when run of p processors
E_n :	Eefficiency of a parallel algorithm
np :	number of processors
MPI :	Message Passing Interface
DMS :	Distributed Memory System
λ :	fraction of volume of fluid in a computational cell

∇ :	nabla operator $\nabla = (\partial/\partial x, \partial/\partial y, \partial/\partial z)$
m1 :	component of normal vector of a straight line in x-direction
m2 :	component of normal vector of a straight line in y-direction
m :	minimum of m1 and m2
α :	line constant
Δ_1, Δ_2, Δ_{big}:	symbols for triangles
$h1, h2$:	value of Heaviside step function
Area :	Area of polygon
(**VV**) :	Dyadic product of two vectors
∇.**V** :	divergence of vector field **V**
∇ **V** :	gradient of vector field
U_w :	Velocity at west face of a cell
U_e :	Velocity at east face of a cell
U_n :	Velocity at north face of a cell
U_s :	Velocity at south face of a cell
Denom :	Denominator
FX^e :	change in flux at east face of a CV
FX^w :	change in flux at west face of a CV
$FX_{i,j}$:	change of flux in x-direction in a cell at (i, j) location
$FY_{i,j}$:	change of flux in y-direction in a cell at (i, j) location
H :	iteration matrix
'**diag**' :	main diagonal of matrix **A**
'**ldiag**' :	lower diagonal (just below the main diagonal) of matrix **A**
'**lldiag**' :	lower lower diagonal (just below the lower diagonal) of matrix **A**
'**udiag**' :	upper diagonal (just above the main diagonal) of matrix **A**
'**uudiag**' :	upper upper diagonal (just above the upper diagonal) of matrix **A**
'**Ldiag**' :	main diagonal of matrix **L**
'**Udiag**' :	main diagonal of matrix **U**
'**Lldiag**', '**Llldiag-**' :	similar definition to-
'**Uldiag**', '**Ulldiag**' :	'**ldiag**' and '**lldiag**'
'**Ludiag**', '**Luudiag-**' :	similar definition to-
'**Uudiag**', '**Uuudiag**' :	'**udiag**' and '**uudiag**'
'offset1' :	distance of lower(or upper) diagonal from main diagonal

'offset2' :	distance of lower lower(or upper upper) diagonal from main diagonal
$\mathbf{z}$:	intermediate vector required to solve the preconditioned system
$\tilde{\mathbf{p}}$:	solution vector to the preconditioned system
$\mathcal{O}$(n) :	complexity of an algorithm
$\lceil\ \rceil$:	ceiling function
$\lfloor\ \rfloor$:	floor function
$\sum_{i\,j}$:	summation over the indexes i and j
(L) :	height of the water column in dam breaking simulation
(a) :	width of the water column in dam breaking simulation
μ_f :	viscosity of the fluid (water)
μ_g :	viscosity of the gas (air)
ρ_f :	density of the fluid (water)
ρ_g :	density of the gas (air)
g :	gravity acceleration
σ :	surface tension of the water
D :	diameter of the air bubble
Mo :	Morton number
Eo :	Eotvos number
u^* :	dimension less velocity of rising air bubble
t^* :	dimension less time
'Time 1_node' :	time required when program run on 1 node using multicores
'Time n_node' :	time required when program run on n node using single core

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The study of fluid dynamics has applications in many areas of engineering and science. These include the understanding of environmental, biological and chemical flows. Laboratory experiments and computer simulations are the two main approaches for the study of complex flow problems. Computer simulations can help one to understand and analyse the complexity involved in these flows more clearly without having to perform time consuming, expensive and complicated experiments. Advancements in modern computers have facilitated the solution and analysis of fluid flows with high accuracy (i.e. close to reality) and reduced computational time. This simulation approach is known as Computational Fluid Dynamics (CFD). In CFD, one has to employ basic physical principles to develop mathematical models and thereafter obtain accurate numerical solutions. Development and improvement of numerical schemes have encouraged researchers to investigate almost every branch of fluid dynamics and its application to real life problems.

Multiphase flows occur in many industrial and natural phenomena such as petroleum refining (Mayer and Lenhard, 1998), biological flows (Christopher, 2005) and interac-

tion of air with the sea surface (Melville, 1996). The simulation of multiphase fluid flows is one of the most challenging problems in CFD as it involves the modelling of sharp interfaces separating multiple fluids. The numerical simulation of multiphase fluid flow can be divided into fluid flow modelling and multiphase modelling as shown in Fig.(1.1).

The fluid and flow properties (velocity, pressure etc.) can be represented by Partial Differential Equations (PDEs) such as the Navier-Stokes (N-S) equations (Section 2.3). Numerical solution of these equations constitute fluid flow modelling. Multiphase

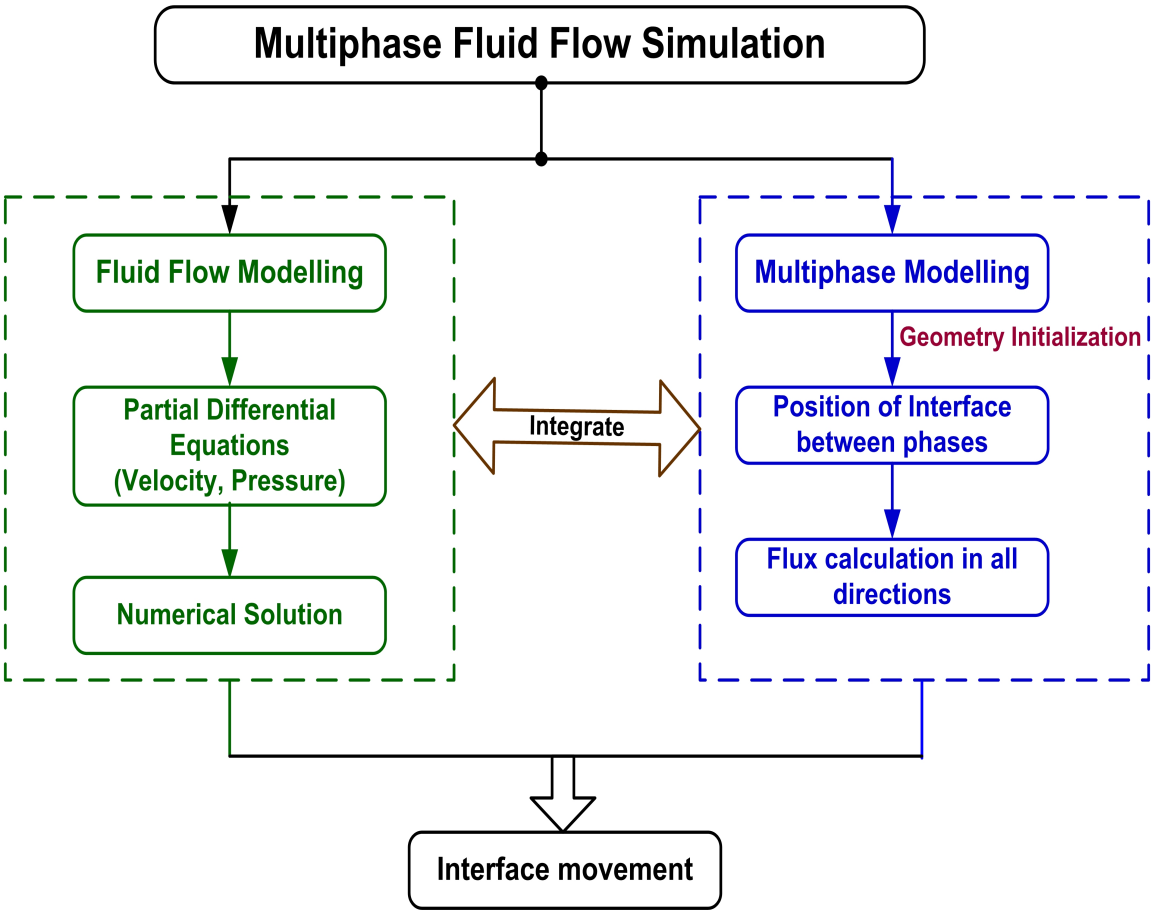


Figure 1.1: Different steps of the multiphase fluid flow simulation

modelling involves defining the interface between various phases and then calculating the flux in all directions by using the solution from the fluid flow model (Section 3.4). Discontinuity in the fluid velocity, pressure and other fluid properties across the interface is one of the characteristics of multiphase fluid flow. Further, the interface motion is integrated with the fluid properties and gradients which may be discontinuous near the interface. The movement of the interface between two phases affects the solution of the PDEs in both phases where the interface is tracked as a moving boundary (Welch, 1995). Since the fluid flow is characterized by the movement of the interface, this step completes the multiphase fluid flow simulation. More details of this procedure are given in the next two sections.

1.1 Multiphase Modelling

Maintaining a sharp interface during fluid transportation is a difficult task in the modelling of interfacial flows (Rider and Kothe, 1998). The interface between two phases can be modelled by a scalar transport equation (Greaves, 2004). The modelling involves the description (or construction) and movement of the interface. An effective approach for interface modelling is interface capturing (see Section 2.4.1). In this thesis, a Volume of Fluid (VOF) method based on interface capturing approach has been used (Section 2.4.1.2). This method has two steps: reconstruction and advection of the interface between two fluids (Rudman, 1997; Denis et al., 1999). First, the interface is defined by calculating the volume fractions of each fluid. Then a transportation algorithm is employed for the movement of the interface. The main challenge in the modelling of the interfacial fluid flow is the implementation of a method which can efficiently move the sharp interface without stretching and wrinkling (Aliabadi and

Shujaee, 2001; Rudman, 1997).

In the reconstruction step, the interface is approximated by a straight line (or a plane in 3D) (Scardovelli and Zaleski, 2000). Next, the area of the geometrical shape (represented by triangles or rectangles) below the lines is calculated to evaluate the volume of fluid based on the position of the interface. The interface is approximated in the subsequent time steps by using the volume of fluid data of the previous time step and this is where the main difficulty (maintaining the sharpness of the interface) in the interface approximation arises (Ruben and Zaleski, 2003). In this thesis, an analytical relation between the fractional volumes and interface position has been implemented to overcome this difficulty (Section 3.3).

1.2 Fluid Flow Modelling

The numerical solution of certain PDEs, representing the physical properties of fluids, involves the discretization of those PDEs and solving large systems of linear equations. The solution of the linear system of equations (involving the matrix-vector product) is the kernel in CFD and consumes most of the computational time (Buttari et al., 2007; Pichel et al., 2009). Development of efficient algorithms for the numerical solution of PDEs, hence, falls into the category of high performance computing (Suda et al., 2009).

The solution procedure for linear systems of equations has two issues: the memory requirement for storing sparse matrices involved and the computational time. The storage formats for the non-zero entries of the matrices, are decided based on MATRIX structures (Buttari et al., 2007; Shahnaz et al., 2006). For un-structured matrices, separate arrays are required for pointing to the locations of the non-zero entries. The memory required by pointer arrays can be saved in the case of well-structured matrices.

A special matrix storage format has been adopted in this work in order to save memory (Section 4.2).

Large sparse linear systems are solved by advanced iterative methods such as Krylov Subspace methods (Behra and Mittal, 2009; Nordsveen and Moe, 1999; Saad, 1989; van der Vorst, 2003). Since such solvers may take a large number of iterations to converge for a given tolerance value, preconditioning techniques are employed to accelerate the convergence rate (Birken et al., 2008). In order to reduce the computational time for the entire simulation of the fluid flows, parallel algorithms are developed for the available parallel computer (Behra and Mittal, 2009; Liu et al., 2005).

1.3 Objectives

This thesis is focused on finding solutions of the challenging tasks mentioned in the above two Sections. The first objective of this research is to investigate and implement preconditioned iterative solvers for the linear system of equations which can be run on modern parallel computers. In order to achieve this goal, the main tasks are:

- Investigation of algorithms for suitable preconditioners for Krylov Subspace methods for matrices arising from multifluid flow problems.
- Analysis of the memory requirements of the sparse matrix formats.
- Investigation of the computational complexity of preconditioners based on the chosen matrix format.
- Development of a novel time and memory efficient algorithm for sparse matrices arising from the numerical solution of multiphase flows.

- Investigation of parallel algorithms for Krylov Subspace methods and appropriate preconditioners to run on a parallel computer.

The second objective is to implement the VOF method to ensure that the interface should not wrinkle or smear. The main tasks for this goal are:

- Investigation of the algorithm for the reconstruction step of the VOF method.
- Development of 2D VOF code using analytic relations for the interface reconstruction.
- Implementation of the VOF transportation algorithm for the movement of the reconstructed interface.
- Validation of the developed VOF code.
- Integration of the developed VOF code, linear solver code and N-S solver code.

1.4 Thesis Organisation and Overview of Chapters

Chapter 2 provides a background study and general overview of multiphase flow problems. It includes a discussion of the methods for modelling the multiphase flow problems applicable to immiscible fluids separated by a sharp interface. A development history of the versions of the VOF methods and their merits/demerits for different flow problems is presented. A short overview of the discretization methods for PDEs and two different grid structures applicable to multiphase flow problems is provided. Categories of the linear solvers, fundamental concepts and types of the Krylov Subspace methods as well as preconditioners for them are discussed in detail. Strategies for parallelization of these methods and related issues are also highlighted. This chapter also

includes details of the benchmark problems used to validate the numerical schemes and computer code.

The mathematical formulation of both steps of the VOF method and difficulties related to the calculation of the interface position from the known values of the volume of fluid fractions is explained in **Chapter 3**. The interface reconstruction step involves the estimation of the normal vector to the straight line and the calculation of the area of polygons. The detailed discussion of the geometrical formula for this area calculation and its comparison with an analytical relation is presented in this chapter. An extended form of the analytical relation in 2D, introduced by Zaleski (Scardovelli and Zaleski, 2000), is explained and documented. The geometrical and mathematical details of the advection algorithm are explained in a flow chart. A short discussion on staggered grid and pressure velocity coupling are also provided in this chapter. At the end of the chapter, the algorithm for the Semi Implicit Pressure Linked Equations (SIMPLE) method of solution of the N-S equations is presented.

In **Chapter 4**, details of the linear systems solvers, parallelization strategies and integration of the parallel solvers with the N-S solvers are provided. A comparison of the memory requirements of different matrix formats is given and the advantages of diagonal format, adopted in this work, is documented. Algorithms for matrix operations, the BiCGSTAB method and the ILUT preconditioner in diagonal format are presented and their computational complexities are discussed. Details of the implementation procedure of the parallel solvers are described by providing the parallel algorithms. At the end of the chapter, the details of the integration of the N-S solver and developed parallel solver as well as the VOF method are outlined. The whole integration procedure is delineated in a flow chart.

The developed VOF code is integrated with the N-S code and tested using benchmark simulation problems. The results of these simulation problems obtained from sequential and parallel code are presented in **Chapter 5**. At the beginning of this chapter, validation of the VOF code is accomplished for shape advection problems. The results from these problems are found to be in good agreement with those in the literature (Harvie and Fletcher, 2001; Ketabdari et al., 2008). Further, a new benchmark test to validate the VOF code is presented.

The next section of this chapter is devoted to the validation of the N-S solver. We have chosen three benchmark problems to validate the N-S code:

(1) lid driven cavity flow (Erturk, 2008; Gjesdal and Lossius, 1997)
(2) dam breaking (Greaves, 2006; Ketabdari et al., 2008; Qian et al., 2003) and
(3) the rising of an air bubble (Hua and Lou, 2007; Lawson et al., 1999).

The performance of various preconditioners is measured by comparing the number of iterations required for different problems. This comparison indicates the variation in the condition number of the matrix generated at different time steps. These variations are expalined by figures of the fluid locations at different time steps. Moreover, the time taken by different preconditioners are also recorded and compared to observe the computational cost of them.

Finally, the summary of the important findings are presented in **Chapter 6**. The next research steps to be taken from these finding are also pointed out in this chapter.

Chapter 2

Literature Survey and Proposed Research

2.1 Introduction

This chapter provides a literature review on the simulation of the multiphase flow problems. As outlined in Section 1.1, modelling of multiphase fluid flow may be divided into (i) multiphase models and (ii) fluid flow models. In the multiphase model, the interface is identified by the value of a scalar function or marker points on the mesh. Advection of the interface is accomplished either by moving the mesh (the interface being used as a deforming boundary) or by applying advection schemes to the interface indicator on the fixed mesh. In the fluid flow models, the numerical solution procedure requires discretization of the computational domain as well as of the differential equations, thereby generating linear systems of equations. These systems are solved by employing suitable iterative solvers. The solvers are typically implemented on parallel computers, for large scale problems, in order to speed up the solution procedure and to

deal with memory constraints. To run the simulation on parallel computers, parallel algorithms are designed according to the computer topologies.

The topics covered in this context are as follows:

- a general overview of multiphase flow,
- the computational modelling issues and mathematical equations for these flows,
- the numerical methods utilized for modelling,
- the methods for equation discretization and linear systems thereby obtained,
- a discussion on matrix data-structures and non-stationary iterative solvers for solving linear systems,
- a description of preconditioners applied to these iterative solvers, in particular the ILUT preconditioner,
- the need for parallelization of the solvers and network topologies for the parallel computer,
- the components of the solvers which can be parallelized,
- some issues on load balancing schemes for parallel algorithms,
- a discussion on parallel preconditioners and
- some benchmark problems for validation of both multiphase and fluid flow models.

While discussing these topics, the proposed research directions for particular topics have also been mentioned.

2.2 Multiphase Flow: An Overview

Multiphase phenomena can be found in everyday life: the interaction of air flow with water at the sea surface is an example of this phenomenon in our environment (Rahman et al., 2006; Zhi and hai, 2007); the flow of two metals in liquid form in an alloy represents a multiphase flow system in the metallurgical industry (Liovic et al., 2001); rising of an air bubble in viscous fluid represents multiphase fluid flow in the petroleum and chemical industry, (Chen et al., 1999; Hua and Lou, 2007).

Depending upon the nature of the system, these flows can be classified as (a) liquid-liquid flows, (b) gas-liquid flows and (c) liquid-solid flows. The simplest form the multiphase fluid flow is two-phase flow, for example the flow of two immiscible fluids[1] such as air and water. In this case, two fluids are separated by a sharp interface (Aliabadi and Shujaee, 2001; Navti et al., 1998; Zhao et al., 2004). This thesis focuses on the flow of two immiscible fluids in 2D and investigates the modelling approaches as well as numerical methods required for the motion of the fluid.

2.3 Mathematical Equations for Physical Phenomena

The physical behaviour of two-phase flows, e.g., flow of two immiscible fluids, can be predicted by computer modelling (Singhal et al., 2002). A general classification of recent advances in computational methods for multiphase flow involving gas, solid and liquid phases is discussed in articles by (Loth, 2000; Moses, 2007). The properties of the fluid flow (such as velocity and pressure) are governed by the continuity and momentum equations which are the mathematical interpretations of the mass and

[1] Two immiscible fluids are separated by sharp interfaces, so that the mixture properties are discontinuous across the interface.

momentum conservation principles (Anderson, 1995). These governing equations are represented by a set of PDEs as follows (Versteeg and Malalasekera, 1995):

(a) The Continuity Equation: The general continuity equation at a point in a compressible fluid is given by

$$\frac{\partial}{\partial t}(\rho) + \nabla \cdot (\rho \mathbf{V}) = 0 \quad , \tag{2.1}$$

where t stands for time, ρ denotes density, $\mathbf{V}$ is the velocity vector and ∇ represents the nabla operator.

(b) The Momentum Equation: The transportation of fluid is described by the momentum equation as follows

$$\Big(\underbrace{\frac{\partial \rho \mathbf{V}}{\partial t}}_{\substack{\text{Unsteady} \\ \text{acceleration}}} + \underbrace{\nabla \cdot (\rho \mathbf{V} \mathbf{V})}_{\substack{\text{Convective} \\ \text{acceleration}}} \Big) = \underbrace{-\nabla p}_{\substack{\text{Pressure} \\ \text{gradient}}} + \underbrace{\nu \nabla^2 \mathbf{V}}_{\text{Viscosity}} + \underbrace{\mathbf{S_v}}_{\substack{\text{Other} \\ \text{body} \\ \text{forces}}} \tag{2.2}$$

where ν denotes the viscosity coefficient, S_V is the source term.

This set of PDEs (equations 2.1 and 2.2) represents the flow of each viscous fluids and are known as the Navier-Stokes (N-S) equations (see (Galdi, 2000)). These non-linear and coupled equations are difficult to solve analytically, and numerical solutions are generally used for simulation.

In the simulation, the properties of the fluid change abruptly across the interface whose location must be known at every time step. This position is determined by interface advection methods and the fluid motion is predicted using N-S solvers (Chen and Kahrif, 1999; Henrie and Stanley, 1997). There are two main approaches to the

solution of the multiphase fluid flows such as air and water flow which involves large viscosity and density ratios:

(1) In the first approach, only the liquid phase (assumed to be inviscid) is modelled. In this approach, the N-S equations can be converted into Laplace's equation (Delaure and Lewis, 2003).

(2) Viscous effects are considered in the second approach (Singhal et al., 2002) and both the liquid and gas are modelled with the full N-S equations.

This thesis is focused on the second approach for multiphase flow, in particular the flow of the two immiscible fluids (which is also known as multifluid flow). In order to simulate different phases of the flow, a multifluid model accounting for the motion of the interface between two fluids is required. The numerical solution of the PDEs involves the discretization in a specified solution domain and thereafter solving a system of linear algebraic equations. This numerical solution provides the fluid flow properties such as velocity and pressure. The numerical methods required (to solve the PDEs and multifluid model) for the computer simulation are presented in the next section.

2.4 Numerical Methods

As discussed above, the simulation of multiphase flow problems requires the numerical solution of the PDEs and a model for specifying and moving the interface between the two phases. The numerical solution calls for the discretization of the computational domain using a grid arrangement to store variables for the fluid properties. Thereafter, the PDEs are discretized according to the chosen grid resulting in a linear system of equations which is then solved to obtain the numerical solution. This section presents

the overview of the different steps involved in the whole simulation procedure. An overview of existing methods for multifluid modelling and discretization of the PDEs is provided in the first three subsections. The linear solvers required to solve the system of equations, and their convergence behaviour are presented in the fourth part of the section. An overview of the model validation completes section 2.4.

2.4.1 Methods for Multiphase Flow

Modelling the advection of the sharp interface numerically is a challenging task (Delaure, 2001) and a range of methods (see(Tryggvason et al., 2001)) have been developed over the last 30 years to achieve this. Among other numerical methods to simulate two-fluid flows, those applied to immiscible fluids can be broadly classified into two categories: (i) interface-tracking and (ii) interface-capturing (Chen and Kahrif, 1999; Ferziger and Peric, 1999; Hogg et al., 2006). Another classification of these methods, as given in the reference (Ubbink, 1997), divides them into (a) surface methods and (b) volume methods. The surface method category is similar to the interface tracking methods while the interface capturing methods are similar to volume methods. In this thesis, we follow the former classification.

2.4.1.1 Interface-Tracking Methods

In *interface-tracking methods*, the interface between two fluids is explicitly tracked during the fluid motion to maintain a sharp discontinuity in the fluid viscosity and density. The motion of the interface is tracked either by marker points located on the surface or by attaching it to a mesh boundary surface. A short description of two commonly-used methods is provided below:

(i) In the *front tracking method* (Unverdi and Tryggvason, 1992), a separate front, on a fixed grid, is used to identify the interface for each phase with the help of additional computational elements introduced explicitly. These elements are known as marker particles on the surface and form a moving internal boundary. Both phases are treated as one fluid with variable material properties and one set of the N-S equations is solved over the whole computational domain. Further, fluid properties such as density and viscosity are calculated with respect to the interface position.

This method uses a piecewise linear, higher order polynomial to fit the interface (by joining the marker particles), which is advected with the flow fields in a Lagrangian manner (Greaves, 2004; Hua et al., 2008). An irregular grid is constructed in the vicinity of the interface and finite difference stencil is used to calculate fluid properties (Tryggvason et al., 2001). Spacing between the marker particles is one of the complexities of this method. The interface is not well-resolved if the particles are placed far apart while in the case of too closely-spaced particles, very high interface curvature may arise which generates high surface tension (Ubbink, 1997).

An application of this method for the simulation of free surface flow can be found in (Navti et al., 1998). Nguyen and co-workers (Nguyen et al., 2009) presented a front tracking method and applied it successfully to a two-phase flow problem with surface tension. However, the main problem in the front tracking methods is the treatment of the topology of the front due to fact that the interaction of the interface grid with the stationary grid is very complicated. The major drawbacks of the front tracking method as reported in (Unverdi and Tryggvason, 1992) are:

- it requires dynamic restructuring of interface grid,
- it requires the addition of grid points where the grid stretches and the removal of points in the compression region,
- the interaction of one front to another in a cell, forces two interfaces to merge in one.

As a result this method is computationally expensive and also may not simulate the interface accurately.

(ii) Another interface-tracking method is the *moving mesh method* (Hyman, 1984; Welch, 1995), wherein a boundary fitted grid is employed and the grid points are embedded in the fluid and these points are moved in a Lagrangian manner. The same fluid elements are kept in all computational cells which are adjacent to the interface and the fluid always coincides with the specified region facilitating the precise tracking of the fluid surface (Kim and Lee, 2003). The interface is integrated with these points and is tracked by the nodes affixed on both phases. The movement of the interface is determined utilizing the knowledge of velocities known at the current time step (Tao, 2005). A simple interpolation between the fluid boundaries and the interface determines the motion of the mesh (Welch, 1995).

The N-S equations are solved for both fluids (liquid and gas). For the liquid phase, these equations are solved on a deforming unstructured mesh (Hyman, 1984). One of the advantages of this method is that it permits the accurate prescription of the boundary conditions of the interface (Ubbink, 1997). A detailed discussion on the moving mesh methods can be found in the reference (Tao, 2005). Quan and Schmidt (Quan and Schmidt, 2007) implemented this method for incompressible

two-phase flow using an unstructured mesh. Recently, a model for merging and breakup in the moving mesh method has also been presented by Quan et al. (Quan et al., 2009).

Since the grid (or marker) points move in the Lagrangian manner, these methods are also known as **Lagrangian methods** (Hyman, 1984; Tornberg, 2000). One of the difficulties of Lagrangian methods is the numerical instability due to the irregular nature of the mesh (Navti et al., 1998).

In interface-tracking methods, the computational mesh is adjusted to fit the interface. The advantage of such methods is that they maintain a sharp interface for which the exact position is known throughout the calculation. However, they require special treatment when the interface is subjected to large deformation or stretching as mentioned above. In the case of large deformation, a numerical inaccuracy in the solution of the flow field may occur due to the high mesh distortion, hence, this method is not suitable for free surface problems such as dam breaking and wave propagation where complex interface breakup and merging occurs (Kim and Lee, 2003).

The severe problem of mesh deformation in the interface-tracking method has motivated the author to explore the interface-capturing method in which the mesh is treated as a fixed reference frame of the fluid movement.

2.4.1.2 Interface-Capturing Methods

In these methods, the sharp interface between two fluids is identified by an indicator function. Using the fluid velocities, the interface advection across the mesh is modelled by advecting the indicator function with the flowing fluid. The indicator function can be chosen as:

(i) *a level set function* (Sussman and Smereka, 1994) or

(ii) *a volume fraction function* (Hirt and Nichols, 1981).

(i) The level set function is a smooth (i.e. each has derivatives of all orders) but arbitrary function which takes different values for both fluids. The values of this function identify the interface. One way of choosing its values is to take a positive value in one fluid and a negative one in the other as defined originally in (Sussman and Smereka, 1994). Thus, the interface is defined at those points where the function has a zero value. This method is known as the *Level Set Method* (LSM). According to(Ubbink, 1997), this method can also be put in the category of surface methods. The LSM automatically takes care of the merging and breaking of the interface. Gibou et al. (Gibou et al., 2007) used a LSM based model to simulate multiphase incompressible flow. They implemented the model successfully to simulate the 2D boiling film for different grid sizes. One drawback of this method, however, is that either a higher-order scheme or a mesh refinement is required to obtain accurate solutions of the advection equation as it does not conserve the mass intrinsically. This makes it a computationally expensive method (Devals et al., 2007).
(ii) The volume fraction function (which is also known as the *colour function*) is a step function and represents the fraction of volume occupied by one fluid.

The interface is reconstructed from the value of the colour function (Hirt and Nichols, 1981; Ruben and Zaleski, 2003). The volume fraction method solves a scalar transport equation (equation 2.3) in an Eulerian manner[2], thereby satisfying the conservation laws; in other words, the volume fraction of a mesh cell is conserved. An example of this method is illustrated in Fig.(2.1) which shows that the value of the colour function C is unity in the fluid 2 and zero in the fluid 1 unity while its lies between zero and unity

[2]The computational grid is fixed and the continuum moves with respect to it.

at the interface. The mesh is kept fixed and a suitable technique is chosen to locate the interface in the interface capturing methods (Greaves, 2004; Scardovelli and Zaleski, 2000). Since the interface is reconstructed at each time step from the known value

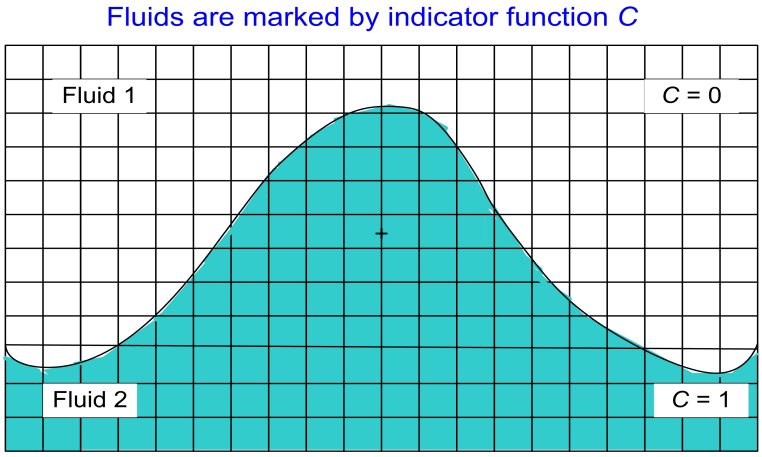


Figure 2.1: Interface capturing method

of the indicator function, the interface capturing methods are able to cope with large stretching and deformation of the interface. Consequently, these methods can be used for modelling large-scale deformations of the interface including breakup and merging (Chen and Kahrif, 1999). The volume fraction methods which use the colour function to identify the interface are known as *Volume Of Fluid* (VOF) methods. The VOF methods have been widely used for the numerical simulation of viscous flows (having non-zero viscosity) with moving interfaces.

As reported in the article by (Kaceniauskas, 2005), the numerical implementation of the LSM is very complicated and requires large computational resources. Further, the interface reconstruction technique of the VOF method provides better approximation of the interface (by following the conservation of the mass fraction) than the LSM. Hence, the VOF methods have been chosen in this work for modelling the two-phase

flow. Details of this method are provided in the following subsection.

2.4.2 The VOF Methods

The VOF method, first introduced by Hirt and Nichols (Hirt and Nichols, 1981), uses the colour function to model the displacement of a fluid surface. The volume conservation of one of the fluid can be expressed as (Hirt and Nichols, 1981):

$$\frac{\partial C}{\partial t} + \mathbf{V} \cdot \nabla C = 0 \tag{2.3}$$

In this method, the computational domain is divided into a number of computational cells which can be referenced by the index pair (i, j) in 2D. The interface is represented by the value of the colour function C in each cell and denoted by $C_{i,j}$. As outlined in Section (2.4.1.2), $C_{i,j}$ takes the value of the volume fraction of one of the two immiscible fluids. There are two steps in the VOF methods: the reconstruction of the interface and the advection of the interface. Various techniques have been proposed for reconstructing and advecting a well-defined interface using $C_{i,j}$. These are based on either an algebraic or a geometric approach.

2.4.2.1 The Algebraic Approach

In the algebraic approach, the convective scalar transport equation for the volume fraction is discretized by the values of C at cell faces in such a way that preserves the position of the interface (Ubbink and Issa, 1999). One such method takes the interface orientation into account while calculating the amount of volume fraction moved across the face of a computational cell (Hogg et al., 2006). The algebraic approach has the boundedness problem where the amount of the volume fraction that initially lies in

the neighbouring cells is not necessarily preserved properly during the advection (in the absence of shear). If the amount is not preserved properly, it leads to non-physical deformations of the interface (Ashgriz and Poo, 1991; Rudman, 1997). Advection of the interface without diffusing, dispersing or wrinkling-due to numerical approximation- is one of the main difficulties of the VOF methods (Hogg et al., 2006), because the interface position is not known in advance. Ubbink and Issa (Ubbink and Issa, 1999) proposed a scheme for the interface advection, named *Compressive Interface Capturing Scheme for Arbitrary Meshes* (CICSAM). This scheme combines two other advection schemes described by Leonard (Leonard, 1991). In flows where the interface takes a shape such as in the case of bubble formation, CICSAM imposes a limit on the *Courant* number defined as (e.g. in x-direction) $C_r = |u|\frac{\delta t}{\delta x}$ (Ubbink, 1997) , where u, δt and δx are the velocity magnitude in x-direction, time step and length interval respectively (Greaves, 2004). Due to this limitation, the geometric approach becomes the altrnative choice for the VOF method. Details of this method are provided below.

2.4.2.2 The Geometric Approach

In this approach, the interface is represented by a series of line segments connecting the sides of the cells. The interface may be approximated either by a straight line or a polynomial which divides the computational cell into two parts containing the calculated volume of each fluid. The main methods in this category include:

(i) The *Simple Line Interface Calculation* (SLIC) method (Greaves, 2004; Rudman, 1997) in which the straight line approximating the interface is selected as parallel to the coordinate axes. The SLIC method was the first VOF method and was found to smear the interface during advection (Morris, 2000). One of the

difficulties here is that the simple representation of the interface may allow more than one interface of each fluid in the computational domain.

(ii) Another method in the geometric approach is the *Piecewise Linear Interface Calculation* (PLIC) method (Greaves, 2004; Kumar et al., 2008; Rider and Kothe, 1998) wherein the interface is approximated by a straight line which may be at an angle to the coordinate axes. In this method, the interface is reconstructed by a predefined set of rules based on the value of the colour function. This method relies on the explicit advection of reconstructed interface and overcomes the difficulties encountered in the SLIC method (Rudman, 1997).

Due to problems of smearing and overlapping of the interface in the SLIC method, the PLIC method has been found to be more suitable for the reconstruction step. Advantages of the interface-capturing methods over the interface-tracking methods and the accuracy of the geometric approach (for capturing the interface) motivated the author to implement the VOF(PLIC) method in this thesis. Further details of this method are continued below.

The VOF(PLIC) Method:

The reconstruction step of the VOF(PLIC) method has two substeps:

(a) normal vector estimation and
(b) volume fraction calculation based on the interface position.

The estimation of normal vectors can be performed by a first order scheme for example using a nine points stencil as given in the document (Rudman, 1997). During the VOF advection, the volume fractions are updated by advecting the reconstructed interface using the local velocities to approximate the motion of the interface. Different components of this method are depicted in Fig.(2.2).

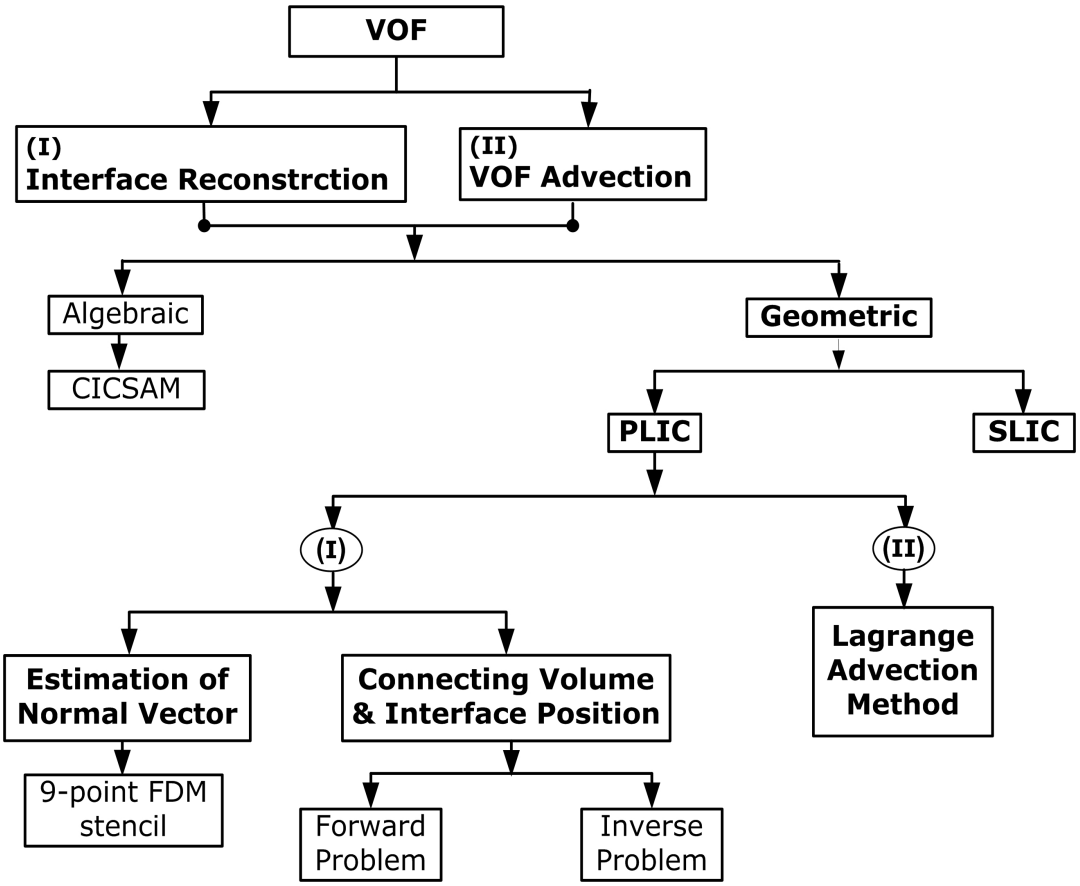


Figure 2.2: Components of VOF (PLIC) method

The main difficulty with the PLIC method is that the cell shapes are implicitly used in the interface reconstruction. Therefore, it is difficult to extend the PLIC method to arbitrarily complex geometries and three dimensional (3D) problems (Scardovelli and Zaleski, 2000). To deal with the difficulties concerning 3D problems, Scardovelli and Zaleski (Scardovelli and Zaleski, 2000) introduced an analytic relation between the interface and the volume fraction in a normalized unit square domain. They documented the formula for the relation only for half of the normalised domain i.e. $[0, \frac{1}{2}]$.

This relation is re-derived in this work and has been extended to the whole

domain, i.e. $[0, 1]$. The re-derived formula has been used to implement the VOF(PLIC) method. Two problems namely"forward" and "inverse" have been formulated in order to establish this relation. The forward problem is used to find the volume fraction in a cell. In the inverse problem, the equation of the interface is obtained using the value of the colour function and normal directions. One of the main features of this relation is that it provided a one to one mapping between the colour function and the interface position, facilitating the accurate calculation of the interface position. Furthermore, it can be extended for the 3D problems. The normal vector of the line representing the interface is estimated during the reconstruction step. A detailed mathematical description of VOF(PLIC) method is provided in chapter 3.

2.4.3 Discretization Methods for PDEs

The PDEs representing the flow and physical phenomena of multiphase flow problems, as described in equations (2.1) and (2.2) in Section (2.3), are solved numerically for the computer simulation (Aliabadi and Shujaee, 2001; Elman et al., 2008; Zhao et al., 2004). In the numerical studies of fluid flows, an approximate solution of these PDEs can be obtained using certain discretization methods such as:

- *The Finite Difference Method* (FDM) (Peiró and Sherwin, 2005),
- *The Finite Element Method* (FEM) (Ferziger and Peric, 1999) ,
- *The Finite Volume Method* (FVM) (Versteeg and Malalasekera, 1995).

These methods convert the differential equations into system of linear algebraic equations. A brief description of these methods is provided below.

§● The FDM is the oldest method (Peiró and Sherwin, 2005) for the discretization of the PDEs and describes the unknowns ϕ of the flow problem by means of grid points on the coordinate lines (Ansorge, 2003). Historical details of this method can be found in (Ernst, 2009). The differential equation is approximated by replacing the derivatives of ϕ using truncated Taylor series expansions in terms of nodal values of ϕ at each grid point and its neighbours (Ferziger and Peric, 1999). Liovic et al. (Liovic et al., 2001) employed FDM along with VOF(PLIC) for the simulation of multiple immiscible fluids. They reported that this method is not suitable for problems involving high density differences because it is difficult to perform high resolution simulation due to a requirement for a robust efficient Poisson equation solver. Therefore, this method can not be used for multifluid flow problem having high density differences such as flows of water and air.

§● In FEM (Petrov-Galerkin approach), the domain is divided into a set of finite elements, and simple piecewise functions valid on elements are used to approximate the local variations of the unknown flow variable ϕ (Zienkiewicz and Morgan, 1983). The exact solution ϕ precisely satisfies the governing equations, but the piecewise approximation of it does not hold exactly and leaves a residual. This residual, which serves as an error measure, is minimised by integrating its values over the domain after multiplying by a set of weighting functions (Ferziger and Peric, 1999). These weighting functions are used to approximate the solution within each element in such a way which ensures that continuity of the solution is satisfied across the boundaries of elements. The FEM is able to deal with arbitrary geometries and facilitates error analysis (Zienkiewicz and Morgan, 1983).

Kaceniauskas (Kaceniauskas, 2005) has implemented successfully the FEM with the VOF method for the simulation of breaking wave phenomena. The authors in (Navti et al., 1998) applied the FEM for the free-surface flow simulation and results were found to be in good agreement with the literature. Devals et al. (Devals et al., 2007) introduced a FEM based interface-capturing method in conjunction with a filtering technique for the colour function and an automatic mesh refinement procedure for two-phase flow simulation. The efficiency of this method was assessed by solving the Rayleigh-Taylor flow problem and the proposed scheme was shown to produce solutions that are of comparable accuracy as those that can be obtained using a fine mesh. Aulisa et al. (Aulisa et al., 2007) presented a numerical model using a FEM with the front tracking method, to simulate two-phase flows. They tested the model for flows where a droplet is surrounded by another fluid. In (Grille et al., 1999), authors employed the FEM to model a visco-elastic lid driven cavity flow and investigated the effect of elasticity on the velocity field.

The main drawback of the FEM is that in the case of an unstructured grid, the matrices generated by the discretization are not well structured and it is difficult to find the efficient solution methods to solve such matrices. The authors in (Liovic et al., 2001) reported that for re-meshing of the moving interfaces, FEM suffers from a heavy computational load. Due to this extra computational load and unstructured matrices, this method has not been adopted in the present work.

§● In the FVM the solution domain is subdivided into finite numbers of contiguous *Control Volume* (CVs) in the FVM. A typical sketch of CVs in a 2D grid is de-

picted in Fig.(2.3). The fundamentals of this method are introduced by Patankar and Spalding (Patankar and Spalding, 1972). The conservation equations are applied to each CV and the integral form of the conservation equation is used as a starting point. The computational nodes, at which the variable values are calculated, are situated at the centroid of each CV as shown in Fig.(2.3). A variety of finite-difference-type approximations can be employed for the terms in the integrated equations which represent flow processes such as convection, diffusion and sources (Patankar, 1980). The variable values at the surface of CVs of the nodal values are calculated using interpolation of the nodal values. Integration over the CVs is the special discretization component of FVM which distinguishes it from all other CFD techniques.

Since the conservation laws are applied to each CV, the resulting systems express the conservation of relevant properties within them and make a clear relationship between the numerical scheme and underlying physical conservation principle. This is one of the main reasons to choose the FVM in this study for the discretization of PDEs.

The CVs in the FVM are defined by a suitable grid. It is important that these CVs should not overlap in order to maintain the conservation in the whole computational domain. This process is known as *domain discretization* (Anderson, 1995; Patankar, 1980). Equation discretization is carried out for all the momentum equations as well as for the continuity equation as described in Section 2.3. In 2D, there are three equations, viz., two momentum equations and one continuity equation. Since the momentum equation is a vector equation and involves both pressure and velocity terms as indicated in equation(2.2), it requires special attention during discretization. Because of the coupling of the pressure and velocity in the momentum equations, the distribution of

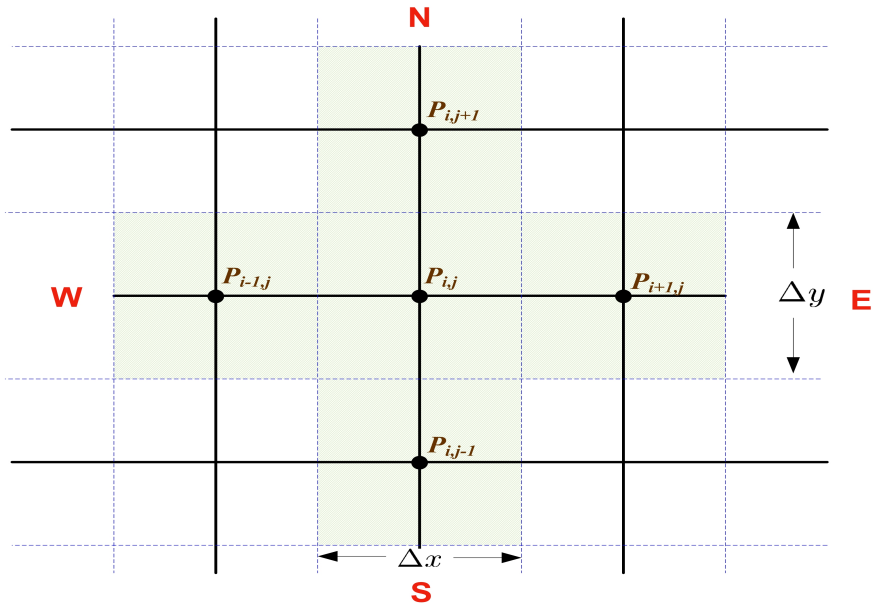


Figure 2.3: Sketch of control volumes in a 2D grid

computational points/nodes in the domain becomes more complex and leaves a choice for various storage arrangement for the variables (Patankar, 1980). The difficulties related to the storage arrangement choice lie in the *pressure correction method* as described in chapter 3. This method is required to deal with the continuity equation for incompressible flows. An overview of two such choices is given here.

§• One of the choices for the arrangement of variables is to store all variables at the same set of grid points. In this case, the same CV is used for all variables. This type of arrangement of grid points is known as a *collocated grid* (Yu et al., 2005). The obvious advantages of this grid are: (a) a smaller number of coefficients have to be stored as only one set of CVs is required and (b) an easier programming implementation. However, collocated grids create difficulties with pressure-velocity coupling and can generate oscillations in pressure unless corrective schemes are introduced (Versteeg and Malalasekera, 1995). Moreover, these

grids require interpolation at boundaries to approximate boundary conditions (Ferziger and Peric, 1999).

§• Another widely-used arrangement of grid points is the *staggered grid* in which all variables are stored at different nodes of the CV (Fletcher, 1991; Patankar, 1980). The pressure variables are generally stored at the centre of the CV while the x and y-momentum variables are at horizontal and vertical locations respectively. Therefore, in this case, the whole grid can be viewed as three different grids in 2D, viz., the *x-momentum grid*, the *y-momentum grid* and the *pressure grid*. Each grid defines the CVs for respective field variables. The main advantages of the staggered grid over the collocated grid can be summarized as done by authors in reference (Ferziger and Peric, 1999):

- In contrast to the collocated grid, several terms can be calculated without interpolation.
- A strong coupling of the pressure and velocity fields reduces the oscillations in pressure and velocity as well as avoids other convergence problems.
- The integral of the momentum equation over the domain is preserved. In other words, the numerical approximation is conservative.

Peric and co-workers (Peric et al., 1988) compared the performances of staggered and collocated grids with FVM for the solution of 2D incompressible flows and documented that for flows such as (a) in the lid drive cavity problem, (b) over a backward-facing step problem and (c) in the circular pipe with sudden contraction, the colocated grid has importance only when non-orthogonal grids are considered.

The presence of the non-linearity in the velocity term and the coupling of the pressure term in the N-S equations implies that an iterative scheme is required for the solution procedure. In the case of incompressible fluids, a pressure correction (Ferziger and Peric, 1999) iterative scheme is employed in which some innovative physical reasoning is used to construct the next iteration from the results of the previous iterations. The effective and widely used iterative scheme for this purpose is the *Semi Implicit Method for Pressure Linked Equation* (SIMPLE) developed by Patankar (Patankar, 1980) which has been shown to be very stable for a wide range of flow problems.

In this scheme, the momentum equation (2.2) for an intermediate velocity is solved using a previously-generated pressure. In the next step, the continuity equation is solved using the intermediate velocity to calculate the pressure update which is then used to update the velocity components. The scheme perturbs the pressure terms in the momentum equation and does not affect the velocities components (Elman et al., 2008). More details of the SIMPLE method are provided in chapter 3.

Since the FVM has a direct resemblance to the physical situation– the conservation of quantities such as mass, momentum, energy, and species are satisfied in each CV and whole domain– it has been widely used for modelling multiphase flow problems. Skuratovsky and Levy (Skuratovsky and Levy, 2004) successfully implemented the FVM with SIMPLE for solving two-fluid flows with heat and mass transfer of wet particulate materials through a pneumatic dryer. FVM along with modified VOF is employed to simulate the bubble rising in a viscous liquid by Chen et al. (Chen et al., 1999). A 3D Navier-Stokes flow solver using a second order cell-centred FVM which has been used for the simulation of a NASA space shuttle launch vehicle (Mavriplis et al., 2007). Cowles presents the FVM for the simulation of ocean processes in coastal

areas (Cowles, 2008).

In the present problem of the multiphase fluid flow, it is particularly important that the conservation laws are satisfied in order to calculate the exact amount of the volume fraction of one of the fluids in the the computational domain. To fulfil this requirement, the FVM with a staggered grid arrangement has been shown to be well suited (Fletcher, 1991). Hence, this method has been adopted in this research for the discretization of PDEs.

Discretization steps (domain discretization and equation discretization) in the numerical solution procedure convert the PDEs into linear algebraic system of equations. Fig. (2.4) summarizes the procedure of discretization and illustrates the steps involved in converting the PDEs that represent the physical model into an algebraic systems of equations. The next step after discretization is solving these systems of equations. The next subsection provides the details of methods for solving the linear system of equations.

2.4.4 Methods for Solving Linear Systems

As indicated above, FVM discretization converts the governing PDEs representing the momentum equations (i.e. equation 2.2) into linear systems of equations which can be solved by linear algebraic solvers. Fig.(2.5) categorizes these solvers into two types, viz., direct solvers and iterative solvers (Golub and Van Loan, 1996). For large scale problems, a consideration of direct solvers can be ruled out because of memory constraints (Benzi, 2002; Benzi et al., 2005). Iterative solvers have been the viable means of solution processes in such cases. A detailed history of iterative solvers can be found in (Saad and van der Vorst, 2000).

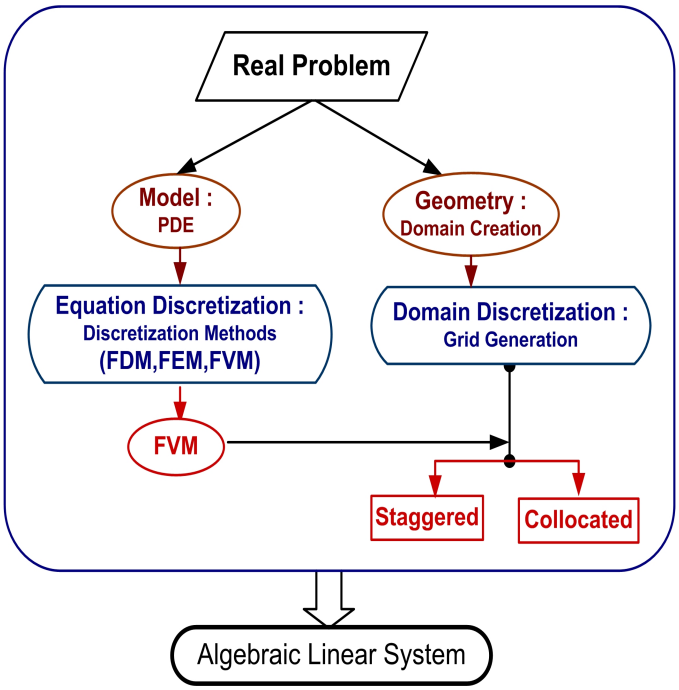


Figure 2.4: PDE to linear system

Iterative solvers may again be classified into stationary and non-stationary solvers (Saad, 1989). In the non-stationary category, current variable values are updated based on several of the previous iteration values, e.g. the value of a residual vector r at the $(n+1)^{th}$ iteration level is updated based on the values of the variables at the n^{th} and $(n-1)^{th}$ levels. Whereas in the stationary methods, the values at the $(n+1)^{th}$ level is updated based only on the values at the n^{th} level. Prior knowledge of the matrix structure generated by the discretization methods should be available in order to decide the matrix data structure and to choose the appropriate linear solver. The convergence rate of non-stationary K-S methods is related to the matrix properties and are accelerated using preconditioning techniques (Benzi, 2002; Sun et al., 2009). All of these issues are explored in the following three subsections.

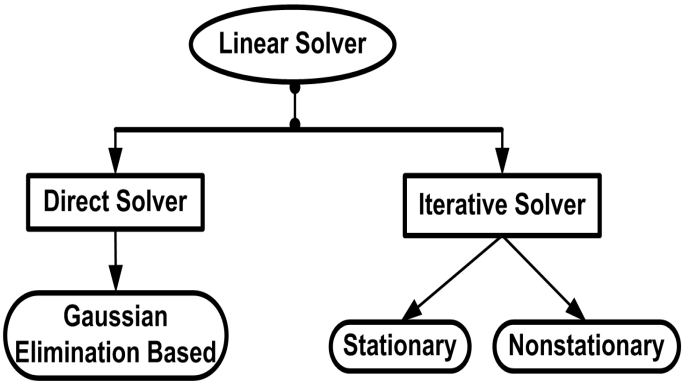


Figure 2.5: Categories of linear solvers

2.4.4.1 Matrix Data Structure

Non-stationary methods are commonly employed to solve large sparse linear systems of equations (Benzi et al., 2005). These systems are typically represented as $\mathbf{A}\,\mathbf{x} = \mathbf{B}$, where $\mathbf{A}$ is an $n \times n$ *sparse* matrix, and $\mathbf{x}$ and $\mathbf{B}$ are the n-dimensional vectors. A matrix having mostly zero elements is called a *sparse* matrix.

In sparse matrices, the number of non-zero entries is quite small in comparison to the number of total entries therefore, there is a need for a special data structure to store these entries in order to save on memory. Literature reveals (Kumar, B.V.R. and Kumar, 2005a; Shahnaz et al., 2006) several common sparse matrix formats such as *Compressed Sparse Column* (CSC), *Compressed Sparse Row* (CSR), *index format* and *diagonal format*. In order to choose a specific matrix data structure, one needs to analyse the storage requirements of these formats (Straubhaar, 2008). Matrices arising from FVM discertization are commonly found to be of diagonal form, in particular penta-diagonal (for 2D problems) or septa-diagonal (for 3D problems) form (Versteeg and Malalasekera, 1995).

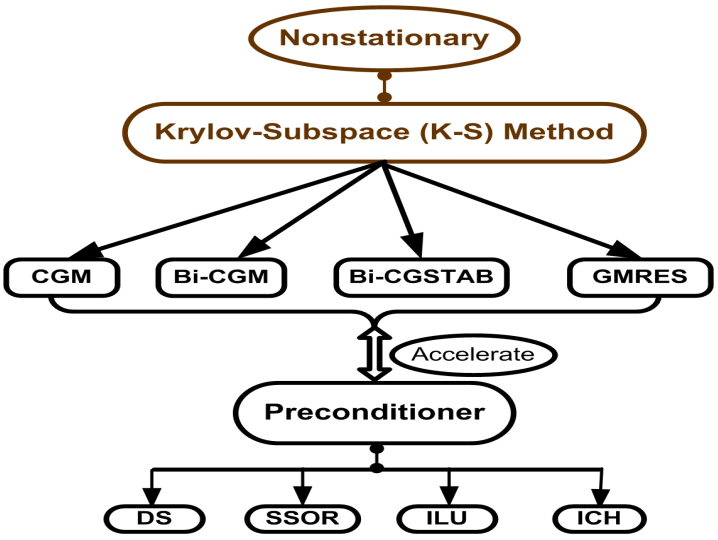


Figure 2.6: Nonstationary iterative methods

2.4.4.2 Krylov-Subspace (K-S) Methods

Krylov Subspace methods (Axelsson, 1996; Saad, 1989) are some of the most efficient iterative methods applied to the large sparse linear system of equations obtained from PDEs representing multiphase flow problems (Greaves, 2004). In these methods, a residual vector $\mathbf{r} = \mathbf{B} - \mathbf{A}\,\mathbf{x}$ is calculated initially, and is minimized or is set to satisfy some other constraints in each iteration.

This residual vector on multiplication by powers of the matrix A in successive iterations, generates a subspace which is called a *Krylov Subspace* ($\mathcal{K}_m$) (Saad, 1996; van der Vorst, 2003), where m denotes the dimension of the subspace. Krylov-Subspace methods belong to the non-stationary category (Benzi et al., 2005), as depicted in Fig.(2.6) and have been widely adopted as a good choice for solving large sparse linear systems of equations (Bergamaschi and Àngeles, 2005; Dag, 2007; Gu et al., 2009; van der Vorst, 2002).

The *Conjugate Gradient Method* (CGM) is probably the most well-known of these

methods but is only applicable to symmetric positive-definite matrices (Golub and Van Loan, 1996). Some other powerful K-S methods which are not restricted to symmetric positive definite matrices are: *Bi-Conjugate Gradient Method*(Bi-CGM), *Bi-Conjugate Gradient Stabilized* (Bi-CGSTAB) and the *Generalized Minimal RESidual* (GMRES) (Liu and Li, 2002; Topsakal et al., 2001).

The GMRES method was developed in 1986 by Saad & Schultz (Saad and Schultz, 1986). The main advantage of this method is that it guarantees to compute approximate solutions with minimal residual norm. The minimal residual is defined as $\mathbf{r}_n = \mathbf{b} - \mathbf{A}\mathbf{x}_n$, where the symbols have the same meaning as defined previously and the subscript n denotes the n^{th} iterations step. A disadvantage of GMRES is overhead costs per iteration which increases linearly with the iteration count and requires more memory to store all the basis vectors of Krylov Subspaces (Saad, 1989, 1996). In order to overcome these difficulties, a restart version was developed and termed as GMRES(l) (Saad, 1996). This version was found to decrease the robustness of the method because it does not guarantee convergence, as well as the risk of slowing the convergence in the case of large value of "l" (Saad and van der Vorst, 2000).

The Bi-CGM, developed by Fletcher (Fletcher, 1976), solves both the primal ($\mathbf{Ax} = \mathbf{B}$) and the dual ($\mathbf{A^T x} = \mathbf{B}$), and therefore requires transpose matrix operations. The transpose operations are difficult in many data structures of sparse matrices (Sleijpen and Fokkema, 1993) and therefore, it is difficult to implement this method. Moreover, the convergence of this method is not smooth and it suffers from several breakdown conditions; a detailed discussion of which is provided in (Saad and van der Vorst, 2000; Pommerell, 1999). To remedy the breakdown problem of Bi-CGM, van der Vorst (van der Vorst, 2003) in 1992 proposed the Bi-CGSTAB method.

Bi-CGSTAB produces smooth as well as faster convergence than Bi-CGM. In the literature (Sleijpen and Fokkema, 1993), it has been pointed out that Bi-CGSTAB shows stagnant convergence for problems in which the coefficient matrix has almost purely imaginary eigenvalues. In order to remove this problem, Sleijpen and Fokkema (Sleijpen and Fokkema, 1993) propose the generalized version of this method, viz., Bi-CGSTAB(l) which takes the benefits from the GMRES(l) and Bi-CGM. In this method, the large value of 'l' may give faster convergence but then the computational cost becomes higher because the cost of the GMRES part increases as the value of 'l' increases (Itoh and Namekawa, 2003).

The convergence behaviour of K-S methods has been shown to depend on the spectral properties of the coefficient matrix of the problem under consideration (Benzi et al., 2005). One such property is the *condition number* of the matrix. The condition number $\kappa(\mathbf{A})$ of a matrix $\mathbf{A}$ can be calculated from its maximum and minimum eigenvalues λ_{max} and λ_{min} respectively and is defined as $\kappa(\mathbf{A}) = \lambda_{max}/\lambda_{min}$ (Elman, 1992; Horn and Johnson, 1985; Jennings and McKeown, 1992). Matrices with high condition numbers are known as *ill-conditioned*. Discretization of the PDEs representing the multiphase flow problem may result in non-symmetric and ill-conditioned *sparse* matrices (Versteeg and Malalasekera, 1995). K-S methods can converge slowly when applied to these ill-conditioned matrices (Kumar, B.V.R. et al., 2004; Jennings and McKeown, 1992; Castillo et al., 2009).

2.4.4.3 Preconditioning

It has been shown that preconditioning techniques (Arabshahi and Dehghan, 2006; Sheen and Wu, 1998; Sun et al., 2009; Zucchini, 2000) can improve the rate of con-

vergence and robustness of iterative solvers. In these techniques, the original linear system of equations, viz., $\mathbf{Ax} = \mathbf{b}$ may be transformed into a preconditioned system $\tilde{\mathbf{A}}\mathbf{x} = \tilde{\mathbf{b}}$ by a linear transformation. The preconditioned system is constructed in such a way that the characteristics of the matrix $\tilde{\mathbf{A}}$ are close to the characteristics of the identity matrix i.e. having unit values on the diagonal and zero every where else. The main aim here is that the preconditioned system should be much easier to solve than the original system and the computational cost of the construction of preconditioned system should not be probihibitive (Duff and van der Vorst, 1998).

The condition number of the preconditioned matrix $\tilde{\mathbf{A}}$ is less than that of the original matrix $\mathbf{A}$. Krylov Subspace solvers converge faster when applied to preconditioned systems than when applied to un-preconditioned systems (Benzi, 2002). Among other preconditioners applied to the K-S solvers, the most effective are (Axelsson, 1996; Golub and Van Loan, 1996; Birken et al., 2008; Saad, 1996; van der Vorst, 2003)

- *Incomplete L-U* (ILU) factorization,
- *Incomplete CHolesky* factorization (ICH),
- *Symmetric Successive Over Relaxation* (SSOR) and
- *Diagonal Scaling* (DS).

Diagonal Scaling is one of the easiest to apply to Krylov Subspace solvers because in this case matrix entries in a row are scaled by the diagonal entry. In general, ILU and ICH have been found to produce faster convergence rates than SSOR when applied to matrices obtained from CFD problems as noted in (Kumar, B.V.R. and Kumar, 2005b). In ILU factorization, the matrix is decomposed as $\mathbf{A} = \mathbf{LU}$, where $\mathbf{L}$ is the lower triangular and U is the upper triangular matrix by dropping some elements of these matrices in order to reduce computational expensive. Due to this cancellation

of non-zero matrix entries, this factorization is called *incomplete factorization*. The ILU has different versions which can perform more effectively to different sparse data structures depending upon the criteria for the elimination/dropping of matrix entries.

One of ways of dropping the entries is based on their location (Saad, 1996; Chow and Saad, 1997). In this version, the rows of matrices **L** and **U** can be computed one at a time and accumulated easily in a row-oriented data structure such as the CSR format of sparse matrices. In the simplest form of ILU, the nonzero entries of **L** and **U** matrices are retained at the same location as in the original matrix **A**. This version is known as *no-fill* ILU or ILU(0) and is better to suitedto well conditioned systems (Diosady and Darmofal, 2009). The ILU(0) version generates a crude approximation of the precondition matrix as explained in (Saad, 1996). But it is useful in some CFD problems as discussed in (Birken et al., 2008). Several other versions of ILU are available which allow more fill-in of the **L** and **U** matrices, and provide more accurate factorization to take fewer iterations for converging K-S methods. However, the computational cost of these ILU algorithms is higher than ILU(0) because in these cases, more operations are required in order to avoid possible break-downs. (Saad, 1992, 1996; Benzi, 2002). It has been noticed that in this approach (i.e., dropping the matrix entries based on locations), the numerical values of the entries are not considered to make it difficult to predict the locations of the largest entries (Malas and Gürel, 2007).

One of the other approaches to generate ILU factorization is based on the magnitude of the elements rather than their locations. In this approach, which is known as the *threshold approach*, the small magnitude elements are dropped or treated as zero (Chow and Saad, 1997). ILU factorization based on this approach has been found to be quite

effective in preconditioning the K-S methods and is known as ILUT (Castillo et al., 2009). It is the simplest form of the threshold ILU algorithm (Saad, 1996). The general form of the threshold approach is ILUT(p,τ) in which the elements whose magnitudes are less than the given value of τ in a row are dropped (Saad, 1997). Further, the maximum number of non-zero entries in the row should not be greater than p; thus, this version keeps track of the storage requirements beforehand. The ILUT is known to be more accurate and stable than the fill-level versions (Chow and Saad, 1997; Saad, 1992). The incomplete Choleski factorization (Golub and Van Loan, 1996) may be seen to decompose the matrix into a lower triangular matrix $\mathbf{L}$ and its transpose. The matrix $\mathbf{L}$ is called the Choleski triangle and can be generated only for symmetric positive definite matrices (Golub and Van Loan, 1996).

2.4.5 Model Validation and Other Numerical Schemes

The whole model for multiphase flow problems consists of two parts (as shown in Fig.(1.1). These parts are (i) PDE solvers which includes linear system solvers and (ii) multifluid methods. For computer simulations, both these parts are integrated together and executed within a time iteration (Anderson, 1995). In order to validate this integrated model, many benchmark problems are available in previous literature on the subject (Erturk, 2008; Hua and Lou, 2007; Ketabdari et al., 2008; Rudman, 1997). A brief overview of these benchmark problems (in the sequence of increasing complexity) and other numerical schemes applied to them, is provided below.

(a) Problems for testing the VOF Method:

The most common test problems for testing VOF methods are the translation and rotation of geometrical shapes. These are pure advection problems and

no N-S solver is required. During the advection, the geometric shape should remain intact or minimally deformed. One such problem is the translation of a square box in a fluid with uniform constant velocity. The main complexity in this problem is to maintain the sharp corners of the square box filled with fluid. The reconstruction algorithm should be able to approximate these corners efficiently. At the sharp corners, the estimation of normal vectors becomes complicated (see Section 3.3.1). Consequently, the calculation of the volume fraction advection may cause numerical inaccuracies. Hence, this is a good benchmark problem to test the reconstruction and advection steps of the VOF methods.

The performance of this method is measured by calculating the error based on the exact solution and the calculated solution (Greaves, 2004; Rudman, 1997; Scardovelli and Zaleski, 2003). Rudman (Rudman, 1997) proposed a VOF method based on the "*Flux Corrected Transport*" (FCT) algorithm (Zalesak, 1979) for the interface advection and Young's method for the interface reconstruction. His method has been implemented for the translation of geometrical shapes such as a hollow circle and a square box and gave better results than methods such as SLIC, Hirt-Nicolas VOF (Hirt and Nichols, 1981).

Another scheme for interface reconstruction and flux update is supplied by Rider and Kothe (Rider and Kothe, 1998) which uses the operator-split time integration approach. Harvie and Fletcher (Harvie and Fletcher, 2000, 2001) present two schemes, viz., the *Stream Scheme* and the *Defined Donating Region*(DDR) for advecting the reconstructed interface. The former scheme uses semi-continuous velocity fields and the latter one defines the donating region in all boundaries with some restriction. In the DDR scheme, the flux is calculated based on volume

intersection between the donating regions and the position of the reconstructed interface. These schemes have been implemented on the geometrical shape movement (translation and rotation) problems. The stream scheme has been found to be in good agreement with literature for box translation, but the DDR gave poor performance for the same problem.

A scheme known as *Defined Donating and Accepting Region*(DDAR) has been proposed by Nobari et al. (Nobari et al., 2009) which was based on the DDR scheme of Harvie and Fletcher (Harvie and Fletcher, 2001). In this scheme, the flux calculation is performed using information of cell volume fraction and its face velocities. For the square box translation problem, it has provided better performance than any other scheme.

Another well-known benchmark test for comparison of scalar advection algorithm is solid body rotation. In this test problem, which is known as the *Zalesak* problem (Zalesak, 1979), a slotted circle is rotated through one or more revolutions. Since this is a rotation problem, it tests the accuracy of the advection scheme as well as the reconstruction algorithm of the VOF method. The stream and DDR schemes have been found to be in good agreement with published numerical results while the DDAR scheme did not perform better than Young's and PLIC methods for the solid body rotation problem (Nobari et al., 2009).

(b) Driven Cavity Flow:

Driven cavity flow serves as a test problem for N-S solvers modelling single phase flows and is one of the most studied problems in CFD due to its simple geometry (Bruneau and Saad, 2001). This problem tests the accuracy and efficiency of the numerical methods due to the ambiguity of the boundary conditions (Erturk,

2008). Lid driven cavity flows provides a model for understanding more complex flows with closed recirculation regions, such as flows over a slit, contraction flows and roll coating flows (Grille et al., 1999).

A detailed discussion of the driven cavity flow problem is provided in the article (Erturk, 2008) and the fluid mechanics underplanning this flow is reviewed in (Shankar and Deshpande, 2000). A solution of the 2D incompressible N-S equations and its validation on lid driven cavity flow problem is presented in (Erturk et al., 2001). Ghia et al. (Ghia et al., 1982) have presented a solution to the lid driven cavity flows using the multigrid method. Their results are widely used as a benchmark by researchers to validate N-S solvers because they have provided extensive experimental data as well as a comparison of higher order numerical schemes.

Gjesdal & Lossius (Gjesdal and Lossius, 1997) have introduced two pressure correction algorithms for multigrid solvers and applied them to lid driven cavity flow problems. They have compared the performance of their algorithms with SIMPLE and SIMPLEC as given in (Versteeg and Malalasekera, 1995). Wu & Shao (Wu and Shao, 2004) have investigated a near-incompressible steady lid-driven cavity flow for Reynolds Numbers (Re) 100-7500 using a multi-relaxation time model, and have compared results with Ghia et al. for grid size 64×64 and 256×256. Results for grid size 256×256 are in better agreement than the grid size 64×64.

(c) The Dam Breaking Problem:

A benchmark problem to test N-S solvers with multiphase flow is the dam breaking problem. This problem contains large deformations of the free surface hence

the surface tension is unimportant. The large surface deformation requires accurate reconstruction of the interface including the efficient estimation of normal vectors and must account for complex deformation such as interface breaking and merging. Since, at every time step a new linear system of equations (obtained from the PDE discretization) is generated, the matrix entries (which are calculated based on the value of the colour function) are updated at every time step subsequently changing the properties of the matrix. This problem measures the accuracy of the VOF method as well as the linear solver and Navier Stokes solver.

Martin & Moyce (Martin and Moyce, 1952b,a) have carried out experiments on the collapse of a fluid column on a rigid plane. In their work, they have provided data for surge height, distance from the axis of symmetry and velocity of the top column of the experiment along with the details of experimental setup which can be used to test the simulation results.

Numerical simulation of free surface problems is helpful in understanding the hydrodynamics of free-surface flows that cause impact loads on maritime structures (Abdolmaleki et al., 2004). The authors in (Abdolmaleki et al., 2004) have validated a N-S solver with a VOF method for dam breaking problems using experimental results. They have concluded that simulation results are much more sensitive to grid size than time step.

Kaceniauska (Kaceniauskas, 2005) has implemented the *Pseudo-Concentration Method* (PCM) for interface capturing, wherein a pseudo-concentration function is defined on the entire domain whose value value indicates the presence of one fluid. The movement of the interface between two fluids is followed by these concentrations. He implemented this PCM in conjunction with the FEM, for

simulating dam breaking problems. The numerical results were found to be in good agreement with experimental data but the computational cost was found to be more than the other methods. Ketabdari and others (Ketabdari et al., 2008) implemented a DDAR scheme for the dam breaking problem and the results were found to be in good agreement with experimental data.

(d) The Rising Bubble Problem :

The numerical simulation of a single air bubble rising in a viscous liquid provides another benchmark problem to test all three parts of the multiphase flow model. In this problem, the surface tension force on the bubble surface has to be considered as it has significant impact on the shape of the bubble. The important properties characterising the behaviour of rising bubbles include, density and viscosity of both phases (liquid and air) and the surface tension on the interface between these phases.

The main difficulty in the simulation of this problem arises due to the rise history of the bubble and the non-linear coupling of some factors such as surface tension, buoyancy force, viscosity etc. (Hua et al., 2008). A bubble rising in a liquid driven by a buoyancy force may create deformation of the bubble, resulting, in some cases, in a toroidal shape (Cerne et al., 1998). The surface tension, in particular, can lead to the large variations in the fluid flow characteristics over a narrow region (in the vicinity of the interface). The variation in the bubble shape changes the properties of the matrices, as discussed in the previous problem, consequently it tests the performance of both the VOF method and the N-S solver.

The study of this problem has great importance in engineering applications such

as rise of air bubbles in boiler tubes, gas bubbles in oil wells (Hua and Lou, 2007). Gibou et al. (Gibou et al., 2007) developed a Level Set Method-based model to simulate the multiphase incompressible flow. They implemented the model to simulate the 2D boiling film problem and presented results for different grid sizes. Hua & Lou (Hua and Lou, 2007) introduced a new front tracking method to simulate the rising of bubble in a viscous liquid. Further, they reported details of parameters, such as mesh size, volume correction, Re, which affect the bubble shape. These parameters can be used to validate the numerical results. Lawson et al. (Lawson et al., 1999) have presented the experimental details of the break-up of large bubble and compared the results from numerical simulation using the VOF method. Their results can be used for direct validation of numerical results. More experimental details of this problem can be found in the reference (Martinez-Bazaz et al., 1999b,a).

2.5 Parallel Numerical Algorithms

As seen in previous sections, computer simulation of multiphase flow problems involves the implementation of complex algorithms which are as intensive in both time and memory requirements. Executing these algorithms may take days when run on a single processor. For instance, in the present project simulation of some problems such as the dam breaking, the rising of an air bubble, for fine grid resolution (e.g. 512×512) takes 6-7 days when executed on single processor with clock speed 2.8 GHz and 8GB RAM. Furthermore, for higher resolution simulations of CFD problems, the matrix size may become too big to fit in the memory and thus, it is difficult to solve the linear system on the available computer (Cowles, 2008; Shang, 2009).

To deal with memory constraints and to reduce computational time, these complex algorithms need to be implemented on parallel computers consisting of clusters of processors (Beltrán and Guzmán, 2009; Li, 2005; Mavriplis et al., 2007). An idea of the importance of the parallel computers can be found from the Top 500 site (www.top500.org), which details the 500 fastest machines in the world on their performance on various benchmark problems In this approach, one can avail of the huge memory and computational power of many processors. The parallel implementation of CFD code has been carried out for the simulation of a space shuttle by authors in (Mavriplis et al., 2007). The development of parallel numerical algorithms has been the focus of intense research in the area of parallel matrix computation (Mezher and Philippe, 2002; Straubhaar, 2008; Shang, 2009).

The quality of parallel algorithms can be measured by using common performance evaluators, viz, speed-up S_n and efficiency E_n which are defined in (Hecquet et al., 2007; Tao et al., 2008) as follows:

$$S_n = T_s/T_p \ , \quad E_n = S_n/n, \tag{2.4}$$

where T_p is the time taken by a parallel algorithm on n processors and T_s denotes the time taken by its serial version. One needs to develop the appropriate parallel algorithm in order to implement it on a parallel computer based on the data structure used for matrix storage. In the present research work, the simulation has been carried out on a linux cluster which has 56 nodes, each having 2 quad core processors with clock speed 2.8 Ghz and 8 GB RAM. In the programs related to parallel matrix computation, processors are required to send the executed data to each other within a network (Beaumont et al., 2001). These data are sent in the form of messages by using certain parallel communication libraries such as *Message Passing Interface* (MPI)

(Dalcin et al., 2008; Le and Rejeb, 2006; Quinn, 2004; Wu et al., 2009).

For the parallel computation of the linear solvers, one processor is required to manage the processor communication such as gathering the parts of matrix-vector products calculated by other processors (Mezher and Philippe, 2002). This processor acts as the master processor and therefore, the master-slave paradigm (Duncan, 1990; Mavriplis et al., 2007) is adopted in this research to design the parallel algorithms for the different steps of the linear solvers.

2.5.1 Parallel K-S Methods

The communication overhead is the bottleneck in the development of efficient parallel algorithms for K-S solvers (Bergamaschi and Àngeles, 2005; Behra and Mittal, 2009; Giraud et al., 2008; Ouarraui and Kaeli, 2004). Any attempt at implementation of the parallel preconditioned K-S solvers is aimed at shortening the computation time and solving very large problems that may not be solved (or take too long to solve) on a single processor (Sun et al., 2009). The specific versions of parallel algorithms are formulated for particular problems which can work optimally on a given hardware such as the Distributed Memory System (DMS) (Behra and Mittal, 2009; Li, 2005). The main components of K-S methods, which need to be parallelized, are (Giraud et al., 2008; Kumar, B.V.R. et al., 2005; Gu et al., 2009; Secher et al., 2009): (i) inner product, (ii) matrix-vector product and (iii) vector updates. Out of these three, the matrix-vector product consumes most of the computational time (Buttari et al., 2007; Dag, 2007; Mellor-Crummey and Gravin, 2004; Pichel et al., 2009; Shahnaz et al., 2006; Williams et al., 2009).

In order to develop the parallel version of these solvers, either the matrix has to be generated at each node or it has to be distributed by a node using an efficient

load balancing scheme (Section 2.5.2). For a particular solver, researchers need to build the parallel version by their own way of parallelization according to the memory architecture of the parallel computer and parallel libraries.

A parallel version of CGM can be found in (Bycul et al., 2002). Dag (Dag, 2007) has presented the implementation of the parallel preconditioned CGM and documented results in the form of execution time and speed-up factors. Zucchini (Zucchini, 2000) has also reported the speed-up factors obtained from his version of parallel CGM. Parallel versions of Bi-CGM have been discussed in (Kumar, B.V.R. et al., 2004, 2005).

Ouarraui et al. (Ouarraui and Kaeli, 2004) have presented the parallel Bi-CGSTAB algorithm using object oriented MPI. An improvement in this algorithm has been provided by Gu and co-workers (Gu et al., 2009). They have proposed an idea to reduce the data dependency in the inner products. Authors in (Wang et al., 2009) presented a parallelization of the Bi-CGSTAB method with diagonal preconditioner. They have implemented the code on the CRAY XT3 and SUN X4600 computers for simulation of thermo-hydro-mechanical problems. Liu and Li (Liu and Li, 2002) have presented a comparison of results obtained from parallel Bi-CGSTAB running on three different machines.

2.5.2 Load Balancing

One of the steps in the development of a parallel algorithm is to distribute the data on a different processors on a cluster using an efficient load balancing scheme. A discussion on the different load balancing schemes for heterogeneous clusters has been provided by Beltrán and Guzmán (Beltrán and Guzmán, 2009). They have proposed a general load balancing scheme for heterogeneous clusters related to different hardware and software.

More load balancing schemes for parallel matrix computations for the heterogeneous clusters have been discussed by Beaumont and co-workers (Beaumont et al., 2001).

In the current project, the parallel algorithms have to be implemented on a homogeneous cluster (Wu et al., 2009) as pointed out above (in the beginning of the section). The designing of load balancing schemes also depends on the data distribution strategies, for instance, the partitioning of matrices and vectors from the linear systems of equations (Kahou et al., 2008). A note on the load balancing for the row-wise matrix partition has been provided in (Petersen et al., 2009).

Most of these traditional load balancing schemes use a formula to distribute the loads which divides the total number of processors into two parts called '*lowered numbered half*' and '*upper numbered half*' (Jordan and Alaghband, 2003). Hence, there is an equal distribution of load within these two parts but different in both the parts. As discussed in Section 4.9.1, this equal load on both parts can be less efficient for parallel algorithms for preconditioners. In the present research, a new load balancing scheme has been devised which provides equal distribution of data to all processors except one processor, consequently facilitating the parallelization of the preconditioners. More details about the new load balancing scheme have been provided in Section 4.9.3.

2.5.3 Parallel Preconditioner

The major computational components of preconditioned K-S methods which are to be parallelized are (Dag, 2007):

(i) the computation of the preconditioner,

(ii) the sparse matrix-vector multiplication,

(iii) the sparse inner product and

(iv) the application of the preconditioner.

Parallelization of the ILU and ICH preconditioners is a challenging task because of data dependency (Henon et al., 2008). The application of these preconditioners involves the development of parallel algorithms for forward and backward substitution. In these algorithms, the data available in one node is needed by another connected node in the network. The parallel implementation of threshold based ILU factorization is known to be quite effective because of dynamic creation of fill elements (Osei-Kuffuor and Saad, 2009). In general, the parallel versions of these preconditioners are developed with the help of highly parallel graph partitioning algorithms (Karypis and Kumar, 1997). Using graph partitioning algorithms, the original matrix $\mathbf{A}$ is partitioned among processors and then each processor generates factors for its part of the matrix. In the next step, all the processors exchange the data in order to complete the factorization.

An overview of the development of the parallel preoconditioner can be found in (Saad and van der Vorst, 2000). Parallelism involved in ILU is discussed in (Vuik et al., 1998; Shen et al., 2003). The literature in (Benzi, 2002) provides the details of parallelization ILU and ILU(0) factorization. Bassermann (Basermann, 2000) has presented the parallelization of block ILUT. Parallelization strategies for these preconditioners also depend on the matrix data structure. As indicated in Section (2.4.4.1), matrices arising from the 2D FVM discretization are penta-diagonal. Therefore, one may need to develop new schemes for the parallelization of preconditioners for these matrices. In this research, a parallel version for penta-diagonal ILUT in diagonal format has been developed based on the master-slave paradigm. More details of the parallel ILUT in the diagonal format have been provided in Section (4.9.5).

2.6 Chapter Conclusions

The main conclusions of the literature review are presented in this section. The overview of the multifluid model is presented first, followed by the conclusions from other sections.

§● Computer simulation of multifluid flow requires the numerical solution of PDEs. This numerical solution is used to move the interface between two phases of fluids in the multifluid model. The key observations in this model are:

(i) the interface-capturing method is more suitable for large deformation and stretching of the interface.

(ii) the VOF(PLIC) method maintains the sharpness of the interface.

(iii) an analytic relation between the interface position and the volume fraction facilitates extension to 3D problems.

§● PDEs are solved numerically in a computational domain which is divided into parts using suitable grid schemes. The following points are noted for steps from domain discretization to linear system of equations.

(a) the FVM enforces conservation of the momentum and mass in each CV as well as in the whole domain.

(b) the staggered grid is more suitable to deal with the problem involving velocity-pressure coupling.

(c) the non-linearity in the velocity terms can be dealt with by the SIMPLE method.

(d) a special data structure is required to store sparse matrices.

(e) the non-stationary K-S solver Bi-CGSTAB provides smooth convergence.

(f) effective preconditioners for K-S solvers are DS, ILUT and SSOR.

§● Large scale problems are solved on the parallel computer. Parallel algorithms are needed to develop to carry out the simulation on these computers. The overview of the parallel algorithms can be concluded as :

(1) matrices derived from FVM (in 2D) are pentadiagonal. In order to solve these matrices, new algorithms for K-S methods and preconditioners are required to reduce computational cost and save memory.

(2) the load balancing scheme can be modified to facilitate the parallelization.

A detailed discussion of the implementation of components of the VOF(PLIC) method such as the analytical relation between interface position and volume fraction and the Lagrange advection algorithm are provided in the next chapter. Also the next chapter provides details of staggered grid arrangement, the pressure correction method and the SIMPLE algorithm.

Chapter 3

Solving Multifluid Flow Models

3.1 Introduction

This chapter provides the implementation details of the methods for the modelling of two immiscible fluids. As seen in the previous chapter, the interface between two fluids can be modelled accurately by the geometric approach of the interface-capturing methods. The VOF (PLIC) method belongs to this category and is applied, in this study, to a fixed Eulerian mesh to evaluate the interface position.

The main challenge in the first step is the calculation of the interface position. In this work, this has been dealt with by using an analytic relation. This relation is split into two parts: *forward relation* and *inverse relation*. Advecting the reconstructed interface, without smearing and wrinkling, is another challenging part of the VOF method.

The previous chapter, it has been observed that the Finite Volume Method(FVM) is more suitable method than FDM and FEM for solving the momentum equations for the multifluid flow problems in a specified discertized computational domain, because, in

this method, the conservation laws are automatically satisfied over the whole domain. Moreover, the *staggered* grid has been found to be more suitable than the *collocated* grid to store the field variables. The pressure-velocity coupling in the momentum equations can be dealt with by applying the pressure-correction technique such as the SIMPLE algorithm. In the context of these issues, the following topics are covered in the present chapter:

- a definition of the "colour function" used in the VOF method,
- an estimation of the normal vector and a calculation of the line constant needed for the reconstruction step of the VOF method,
- forward and inverse analytic relations for the line constant and the volume fraction used in this step,
- the correspondence between this analytic relation and the geometric formula for calculation of the volume fraction,
- advection of the interface in the computational domain using the Lagrangian method,
- the staggered grid implementation for the domain discretization,
- the discretization of the momentum equations in the specified discretized domain,
- the pressure-correction method to deal with the non-linearity of the equations and
- details of the SIMPLE algorithm of the pressure-correction method.

A flow chart of the SIMPLE algorithm is provided at the end of this chapter. All these topics are crucial computational tools for simulating the multiphase flow phenomena.

3.2 Implementation of VOF(PLIC) Method

As mentioned in the previous chapter, interface-capturing methods can be classified into two approaches, algebraic and geometric. The latter approach includes the VOF method which is the topic of this section. It has been noted in the previous chapter that the VOF(PLIC) method gives more accurate approximations of the interface than the SLIC method. The components of the VOF(PLIC) method as illustrated in Fig. (2.2) are discussed here.

It has been seen in section (2.4.2) that VOF methods use a colour function which is a *step function* and provides the value of volume fraction of one fluid (in a mixture of two fluids) in a control volume (CV). The colour function C in each cell can in 2D be defined as

$$C_{i,j} = \begin{cases} 0 & \text{fluid 1} \\ \lambda \in (0,1) & \text{both fluids} \\ 1 & \text{fluid 2} \end{cases} \tag{3.1}$$

Equation (3.1) states that the value of the colour function in a cell at location (i, j), i.e, $C_{i,j}$ is equal to unity if the cell is filled with fluid one, and it is equal to zero in the case when cell is filled with another fluid. The value of $C_{i,j}$ lies between zero and unity if the interface is present in the cell indicating that both fluids are present. The relationship between the topology of the interface and the distribution of C is illustrated in Fig.(3.1).

As mentioned before in section 2.4.2.2, the VOF(PLIC) method involves two steps.

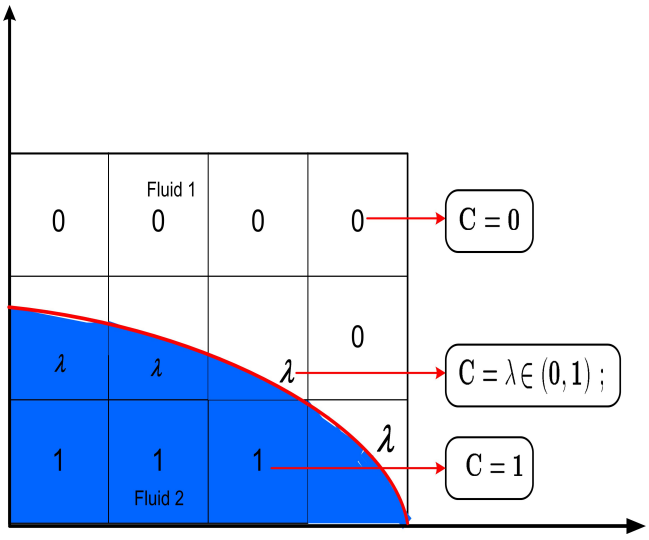


Figure 3.1: Values of colour function in particular CVs

The first step is known as the interface reconstruction algorithm in which the interface position and shape are approximated from the known distribution of the volume fraction. Both the values of the colour function and its gradient are used to evaluate the interface position and orientation (Kumar et al., 2008; Rudman, 1997; Scardovelli and Zaleski, 2003). In the second step, the volume fraction at a new time level is determined from the VOF fluxes derived for the known velocity field at the reconstructed front. This step is known as the *VOF advection algorithm*. Both of these steps are required during the simulation in order to determine the movement of the fluid. Detailed information of both these is provided in the next two sections.

3.3 Interface Reconstruction

In the VOF(PLIC) method, the interface between two fluids in a grid cell is approximated by a line segment which intersects the cell's faces. The line segment divides the

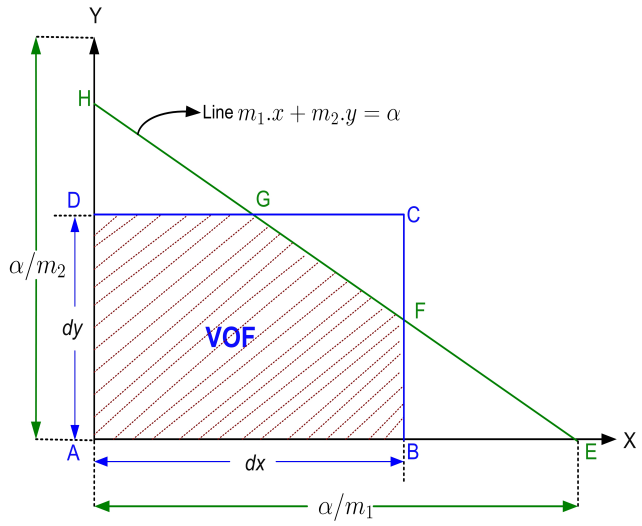


Figure 3.2: Cell ABCD is cut by the straight line EH and contains fluid 2 in region ABFGD and fluid 1 in region FCG

cell into two parts, each of them containing one of the two fluids as shown in polygon ABFCGD in Fig.(3.2). In this figure, the notations are as follows:

- rectangle ABCD: represents a grid cell,
- dx: the length of cell in x-direction,
- dy: the length of cell in y-direction,
- Line EH: approximation of the interface,
- polygon ABFGD: volume of one fluid in the cell and
- $\frac{\alpha}{m_1}$: projection of the line segment EH on the X axis.

The general equation of a straight line (say $L1$) with normal $\tilde{\mathbf{m}}$ may be written as

$$m_1.x + m_2.y = \alpha \tag{3.2}$$

where m_1, m_2 are the components of the normal vector in the x and y directions respectively and α is the line constant which is related to the distance of the line from the origin. The coordinates of the points at which the line intersects the axes X and Y are respectively $(\frac{\alpha}{m_1}, 0)$ and $(0, \frac{\alpha}{m_2})$ which corresponds to the points E and H in Fig.(3.2).

In the simulation, the values of the volume fractions are provided initially for all the cells. But, at the next time step, the fluid mixture moves and the interface changes its position, and hence new values of the volume fraction have to be calculated. In order to evaluate the volume fraction of one fluid in a cell, one has to calculate the area below the line $L1$ which is the area of the polygon ABFGD, as depicted in Fig.(3.2). For calculating the polygon area, the position of the line is determined by estimating the normal vector from the known value of colour function and the line constant α. The next subsection presents the procedure for normal vector estimation.

3.3.1 The Estimation of the Normal Vector

The reconstruction of the interface is based on the idea that a normal vector $\mathbf{m}$ together with the colour function values determine a unique linear interface cutting the cell. In the first part of the reconstruction, a normal vector is estimated by a nine-point finite difference formula which is defined as (Rudman, 1997),

$$\mathbf{m} = \nabla C \tag{3.3}$$

Formula (3.3) represents the gradient of the colour function C in the direction of coordinate axes. The discrete approximation to equation (3.3) is given by

$$\nabla C = \begin{pmatrix} \nabla^x C \\ \nabla^y C \end{pmatrix} \equiv \begin{pmatrix} m_1 \\ m_2 \end{pmatrix} \tag{3.4}$$

where, ∇^x is the gradient in the x direction and ∇^y is the gradient in the y direction. For approximating the values of these gradient terms, we choose eight nearest neighbours in 2D (i.e. all the neighbours sharing the vertex). This scheme is known as the *nine-point stencil* or *centred column scheme* (Scardovelli and Zaleski, 2003). For a uniform mesh, in a grid cell at location (i, j), the gradient terms of equation (3.4) in the coordinate form can be expressed as (Rudman, 1997),

$$(m_1)_{i,j} = \frac{1}{\delta x}\left(C_{i+1,j+1} + 2C_{i+1,j} + C_{i+1,j-1} - C_{i-1,j+1} - 2C_{i-1,j} - C_{i-1,j-1}\right) \tag{3.5}$$

$$(m_2)_{i,j} = \frac{1}{\delta y}\left(C_{i+1,j+1} + 2C_{i,j+1} + C_{i-1,j+1} - C_{i+1,j-1} - 2C_{i,j-1} - C_{i-1,j-1}\right) \tag{3.6}$$

Equations (3.5) & (3.6) represent the normal vector estimation formula for the x and y directions respectively. Denis and others (Denis et al., 1999) reported that this scheme produces good estimation of the normal vector. An assessment of the accuracy test of different normal estimation schemes has been carried out by Scardovelli and Zaleski (Scardovelli and Zaleski, 2003). They documented that the linear fit (using the nine point stencil as given above) produce the same order error as other methods such as quadratic fit which requires more computations.

3.3.2 The Calculation of The VOF from the normal vector and the line constant

The values of the normal vector components m_1 and m_2 along with the line constant α provide the exact position of the interface in the grid cell. As mentioned before, the value of the colour function $C_{i,j}$ in a grid cell (i, j) is determined from the area of polygon ABFGD as shown in Fig. (3.2). This area may be calculated geometrically as follows,

$$\pentagon\ ABFGD = \underbrace{\triangle\ AEH}_{\triangle_{big}} - \underbrace{\triangle\ BFE}_{\triangle_1} - \underbrace{\triangle\ DGH}_{\triangle_2} \tag{3.7}$$

Mathematically, this may be shown to be (Greaves, 2004)

$$Area = \underbrace{\frac{\alpha^2}{2m_1m_2}}_{\triangle_{big}} \left\{ 1 - \underbrace{H(\alpha - m_1dx)\left(\frac{\alpha - m_1dx}{\alpha}\right)^2}_{\triangle_1} - \underbrace{H(\alpha - m_2dy)\left(\frac{\alpha - m_2dy}{\alpha}\right)^2}_{\triangle_2} \right\} \tag{3.8}$$

where $H(x)$ is the Heaviside step function defined as

$$H(x) = \begin{cases} 0 & x < 0 \\ 1 & x > 0 \end{cases}$$

This formula calculates the area below the line as shown in Fig.(3.2) The line segment representing the interface can cut the grid cell in two different ways (in a particular coordinate direction) depending upon its slope. The procedure for calculating the area below the line in the two cases for the x direction using equation (3.8)is described here.

(I) Consider the case of a line with positive slope intersecting the grid cell at the axes as shown in Fig.(3.3). This figure represents three triangles,

- $\triangle_{big}$: triangle BEH
- $\triangle_1$: triangle AEF and
- $\triangle_2$: triangle CGH

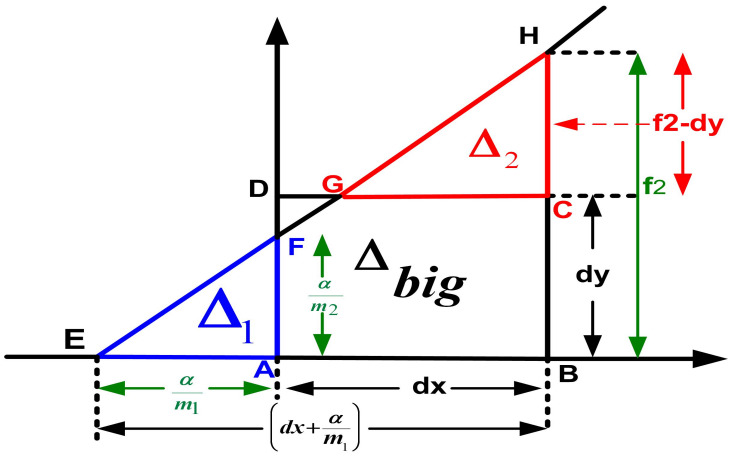


Figure 3.3: Line of positive slope intersects the cell faces

From this figure it is clear that the area of polygon ⬠ ABCGF is required to calculate the actual volume of the fluid contained in the cell.

$$
\begin{aligned}
\triangle_{big} &= \left| 0.5 \times \left(dx + \frac{\alpha}{m_1} \right) \times f_2 \right| \\
\triangle_1 &= \left| \left(0.5 \times \frac{\alpha}{m_1} \times \frac{\alpha}{m_2} \right) \right| \\
\triangle_2 &= \triangle_{big} \frac{(CH)^2}{(BH)^2} \\
\Rightarrow \quad \triangle_2 &= \triangle_{big} \left\{ \frac{(f_2 - dy)^2}{f_2^2} \right\} = \triangle_{big} \left(1 - \frac{dy}{f_2} \right)^2 \\
\left(\frac{\alpha}{m_1} < 0 \right) &\Rightarrow h1 = 1 \quad \text{and} \quad (f_2 - dy) > 0 \Rightarrow h2 = 1 \\
& \boxed{Area = \triangle_{big} \{1 - h1.\ \triangle_1 - h2.\triangle_2\}}
\end{aligned}
\tag{3.9}
$$

Equation(3.7) states that to calculate the area of polygon ⬠ ABCGF, one needs to calculate the area of these triangles and then apply equation (3.8). The values of the area of these triangles and the complete calculation procedure is presented in equation (3.9). In equation (3.9), variables $h1$ and $h2$ represent the Heaviside step function $H(\alpha - m_1 dx)$ and $H(\alpha - m_2 dy)$ respectively.

(II) Another situation occurs when a line with negative slope intersects the cell as shown in Fig(3.2). A clearer picture of this case is depicted in Fig.(3.4). The different steps involved in the procedure of area calculation are described below,

$$
\begin{aligned}
\triangle_{big} &= 0.5 \times \left(\frac{\alpha}{m_1} \times \frac{\alpha}{m_2} \right) \\
\triangle_1 = \triangle_{big} \times \frac{(BE)^2}{(AE)^2} \quad &= \quad \triangle_{big} \times \frac{\left(\frac{\alpha}{m_1} - dx \right)^2}{(\alpha/m_1)^2} \\
\Rightarrow \quad \triangle_1 &= \triangle_{big} \left(\frac{\alpha - m_1.dx}{\alpha} \right)^2 \\
\triangle_2 = \triangle_{big} \frac{(DH)^2}{(AH)^2} &= \triangle_{big} \times \frac{\left(\frac{\alpha}{m_2} - dy \right)^2}{(\alpha/m_2)^2} \\
\Rightarrow \quad \triangle_2 &= \triangle_{big} \left(\frac{\alpha - m_2.dy}{\alpha} \right)^2 \\
(\alpha - m_1 dx) > 0 \Rightarrow h1 = 1 \quad &\text{and} \quad (\alpha - m_2 dy) > 0 \Rightarrow h2 = 1 \\
\boxed{Area = \triangle_{big} \{1 - h1.\ \triangle_1 - h2.\triangle_2\}}
\end{aligned}
\tag{3.10}
$$

The variables $h1$ and $h2$ in equation (3.10) have the same meaning as described in the previous case. The above two situation are more general case when line intersects the cell such that it generates two small triangles. Other cases may arise when there is only one triangle (i.e. $h1 = 0$ or $h2 = 0$) or there is no triangle present at all (i.e.

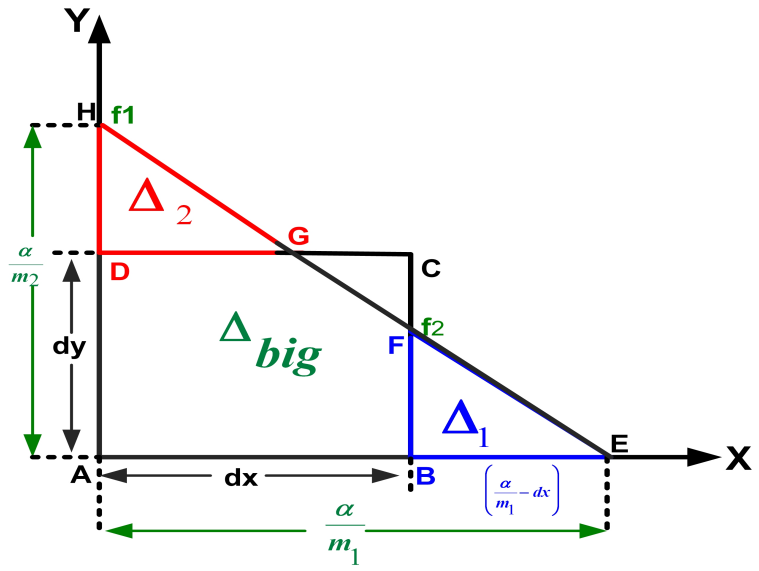


Figure 3.4: Line of negative slope intersects the cell faces

$h1 = h2 = 0$) as shown in Fig.(3.5). For the latter case the area will be equal to $\triangle_{big}$ (i.e., $Area = \triangle_{big}$).

During advection, the value of α has to be calculated from the known values of the colour function in order to know the interface position. It is difficult to calculate the exact value of α from equation (3.9) or (3.10), and the calculation procedure becomes even more complicated in 3D. This is due to the fact that it requires a numerical iterative approximation which can converge slowly in 3D case in order to solve the cube root as given in (Denis et al., 1999; Rider and Kothe, 1998). To mitigate this difficulty, Scardovelli and Zaleski (Scardovelli and Zaleski, 2000) proposed an analytic relation between the values of the colour function and the line constant α in a normalized domain $[0, 1]$. They provided the details of their analytical solution and its properties in the half domain $[0, \frac{1}{2}]$ only. In the present research, this relation has been further investigated and an extension to the full domain in 2D has been documented. One of the main advantages of this formula is that it can be easily extended to 3D problems.

We have provided the details of all the analytical steps involved in its derivation.

This formula has two parts, viz., a *forward relation* and an *inverse relation*. The forward relation computes the value of the colour function while the value of line constant α is obtained from inverse relation. Both relations and their correspondence to the geometric equation (3.8) are presented below.

The Forward Relation

From this relation, the value of the colour function can be calculated using the known values of α, m_1 and m_2. The values of these variables are known for the line which has been advected at the current time step. Therefore, the *forward relation* is used to update the volume fraction after advection. It is directly related to equation (3.8) and is defined as,

$$C(\alpha) = \begin{cases} \dfrac{\alpha^2}{2.m.(1-m)} & 0 \leq \alpha < m \\ \dfrac{\alpha}{(1-m)} - \lambda & m \leq \alpha < 1/2 \\ 1 - \left(\dfrac{1-\alpha}{1-m} - \lambda \right) & 1/2 < \alpha \leq (1-m) \\ 1 - \dfrac{(1-\alpha)^2}{2.m.(1-m)} & (1-m) < \alpha \leq 1 \end{cases} \tag{3.11}$$

In the equation (3.11), $m = \min(m_1, m_2)$, $m_1 + m_2 = 1$, $\lambda = \dfrac{m}{2.(1-m)}$ and $C \in [0, 1]$. Since the formula is derived for normalized form, we consider each grid cell as unit square (therefore, dx and dy are assumed to be unity) but all the calculations has been done using the actual values of dx and dy by normalizing the cell length as elucidated later in Fig. (3.11). The relation between equations (3.8) and (3.11) can be examined

by investigating each inequality of (3.11) one by one.

The First inequality

The condition $0 \leq \alpha < m$ implies that both the small triangles ($\triangle BFE$ and

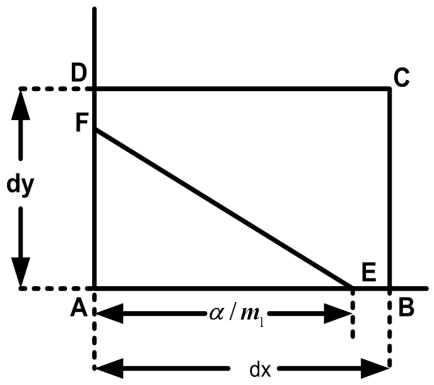


Figure 3.5: Interface position corresponding to the first inequality in equation (3.11)

$\triangle DHG$) in Fig.(3.2) are absent. The position of the interface for this inequality is shown in Fig. (3.5). This means that the values of Heaviside step functions $H(\alpha - m_1)$ and $H(\alpha - m_2)$ are zero and the equation (3.8) becomes

$$Area = \frac{\alpha^2}{2.m_1 m_2}$$

This is same as in the first condition of equation (3.11).

The Second inequality

In this case let $m = m_1$ then the meaning of the condition $m_1 \leq \alpha < 1/2$ is that the small triangle in the y direction, i.e., $\triangle DGH$ is absent as depicted in Fig. (3.6). Hence the value of $H(\alpha - m_2) = 0$ and $H(\alpha - m_1) = 1$. The formula for

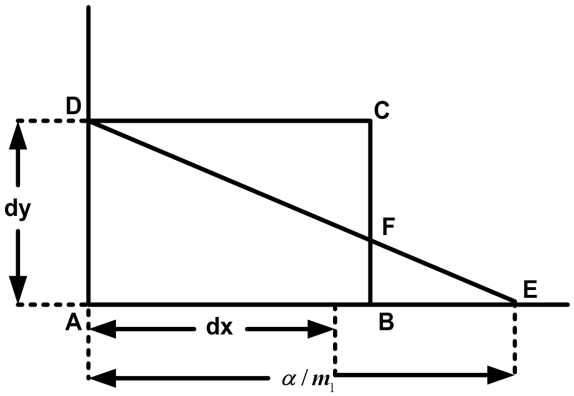


Figure 3.6: Interface position corresponding to the second inequality in equation(3.11)

the area below the line DE in the cell $ABCD$ in Fig. (3.6) can be expressed as

$$\begin{aligned} Area &= \frac{\alpha^2}{2.m_1m_2}\left\{1-\frac{(\alpha-m_1)^2}{\alpha^2}\right\} \\ &= \frac{\alpha}{m_2}-\frac{m_1}{2m_2} \\ &= \frac{\alpha}{(1-m_1)}-\frac{m_1}{2.(1-m_1)} \\ &= \frac{\alpha}{(1-m)}-\lambda \end{aligned}$$

Hence, the value of the area is same as in the second condition of equation (3.11).

The Third inequality

The condition $1/2 < \alpha \leq (1-m)$ states that the small triangle ($\triangle BEF$) on the x axis is absent(so $m = m_2$). This position of the interface is illustrated in Fig.(3.7) which shows that the value of $H(\alpha - m_2) = 1$ and $H(\alpha - m_1) = 0$. Now equation (3.8) can be shown to be equal to $1 - \left(\frac{1-\alpha}{1-m} - \lambda\right)$.

The Fourth inequality

The condition in this inequality refers to the position of the interface as shown in Fig (3.2). Here, both Heaviside step functions have unit value and the formula

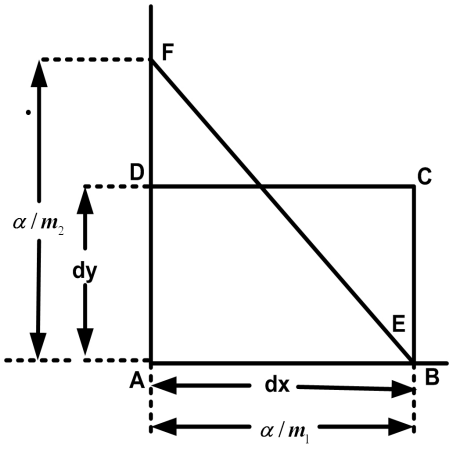


Figure 3.7: Interface position corresponding to the third inequality in equation (3.11)

of area is written as

$$\begin{aligned} Area &= \frac{1}{2.m_1m_2}\{-m_1^2 - m_2^2 - \alpha^2 + 2m_1\alpha + 2m_2\alpha\} \\ &= \frac{-1}{2.m_1m_2}\{(m_1+m_2)^2 - 2\alpha(m_1+m_2) - 2m_1m_2 + \alpha^2\} \\ (m_1+m_2) = 1 \quad &\Rightarrow \\ Area &= \frac{-1}{2.m_1m_2}\{1 - 2\alpha + \alpha^2 - 2m_1m_2\} \\ &\equiv 1 - \frac{(1-\alpha)^2}{2m(1-m)} \end{aligned} \tag{3.12}$$

The properties of equation (3.11) can be summarized as,

- Function $C(\alpha) : [0,1] \longrightarrow [0,1]$ is a continuous monotonically increasing function of α.
- It is one-to-one and onto mapping from $[0,1]$ to $[0,1]$.

Since $C(\alpha)$ is a one-to-one and onto and continuous function, its inverse function exists (Fischer, 1983) and can be used for calculating the value of the line constant α from the value of C. This inverse relation is discussed in the next section.

3.3.3 Calculation of the Line Constant (Inverse Relation)

At a certain time step, after advection, the position of the interface (i.e. the value of α) needs to be calculated using the updated values of C, m_1 and m_2. The inverse relation of equation (3.11) is used to calculate the value of α which has been constructed here and can be defined as,

$$\alpha(C) = \begin{cases} \sqrt{2.m.(1-m).C} & 0 \leq C < \lambda \\ (C+\lambda)(1-m) & \lambda \leq C \leq 1/2 \\ 1-(1-C+\lambda)(1-m) & 1/2 < C < 1-\lambda \\ 1-\sqrt{2(1-C).m.(1-m)} & 1-\lambda \leq C \leq 1 \end{cases} \tag{3.13}$$

Similar to the forward relation, the last two inequalities of this formula have been derived in this work based on the first two inequalities and their symmetric properties documented in (Scardovelli and Zaleski, 2000).Equations (3.11) and (3.13) define an analytical relation between volume fraction values C and line constant α. Equation (3.13) determines the exact value of α for a given value of C which is quite difficult to find using equation (3.8). The interface is reconstructed using these two equations at each time step during simulation.

3.4 Advection of the Interface

Once the interface is reconstructed, the advection algorithm is used to model its motion in the underlying velocity field. Equation(2.3) represents the volume conservation of one fluid provided that the velocity is divergence free (for incompressible flows) i.e.,

$\nabla \cdot \mathbf{V} = 0$ (Greaves, 2004; Hieu et al., 2004). However, it is to be noted here that the discrete velocity divergence is not necessarily zero therefore, *divergence correction*, $\nabla \cdot \mathbf{V}$, is added to both side of this equation. Hence, equation(2.3) can be expressed in 2D as (Rider and Kothe, 1998),

$$\frac{\partial C}{\partial t} + \nabla \cdot (\mathbf{V}C) = C \nabla \cdot \mathbf{V} \tag{3.14}$$

Equation (3.14) represents the movement of "C". For advection of the interface, a Lagrangian advection method is implemented in this work and is discussed in the next section.

3.4.1 The Lagrangian Advection Method

With the Lagrangian method outlined in (Denis et al., 1999), the interface is located by tracking the motion of line segments representing the interface between two fluids. The advection of the interface can be carried out in two ways: one of the ways is to move the interface in both x and y directions simultaneously; the second way is to split the advection process in both directions. The latter has more advantages over the former as discussed in (Rider and Kothe, 1998) and thus has been utilized in this work. At each time step, the solution of the N-S equations provides the velocity fields in the x and y directions. To understand the procedure in the x direction, consider the position of the interface at a time step n as shown in Fig.(3.8). In this figure, $u(x)$ denotes the x-component of the velocity ($\mathbf{V} \equiv (u, v)$) and the interface is represented by the line "ab". The general equation of a line at time step n can be written as

$$m_1^{(n)}x + m_2^{(n)}y = \alpha^{(n)} \tag{3.15}$$

During the x-sweep, the y component of velocity is ignored. Assume that the velocity at the west face (line CD) is U_w and at the east face (line AB) is U_e. Then the x-component of velocity within the cell can be described as a linear interpolation of U_w and U_e as shown in equation (3.16).

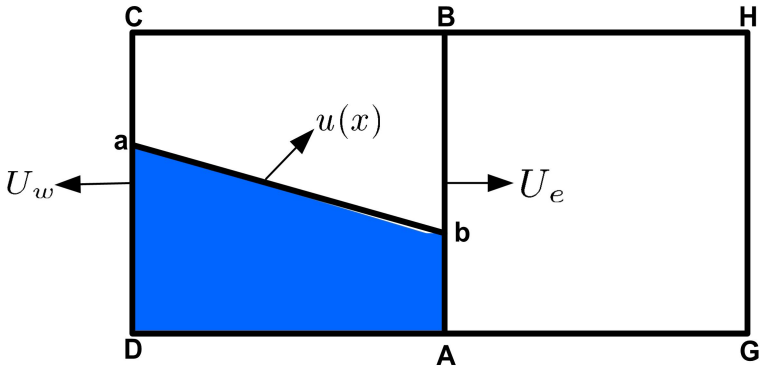


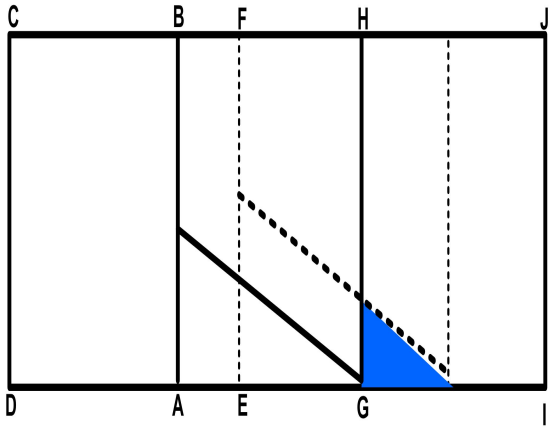
Figure 3.8: Position of the interface at a time step n

$$u(x) = U_w\left(1 - \frac{x}{\delta x}\right) + U_e\frac{x}{\delta x} \tag{3.16}$$

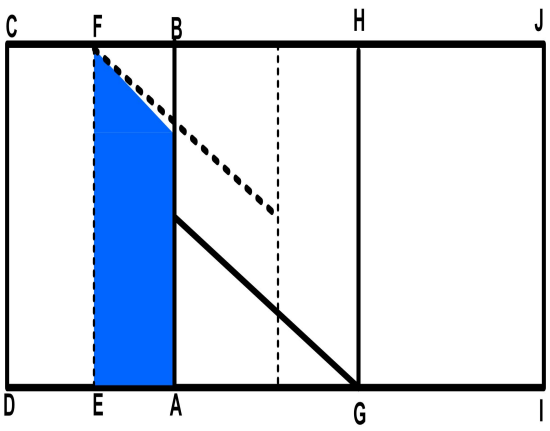
During advection, in the next time step, the fluid moves to other cells and thereby the position of the interface changes. The fluid can move to any one of four directions in 2D depending upon the direction of the velocity field as illustrated in Fig. (3.9).

A mathematical description of the movement in the x-direction is presented here. Suppose the new position of the interface at the next time step is represented by the line "cd" as shown in Fig.(3.10) An arbitrary point on the line "cd" can be described as

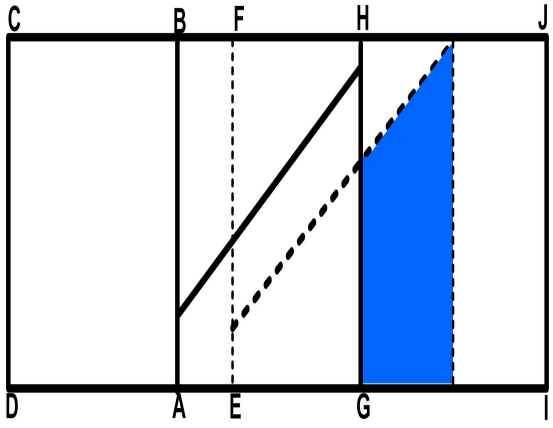
$$x^{n+\frac{1}{2}} = x^n + u(x^n)\delta t \tag{3.17}$$



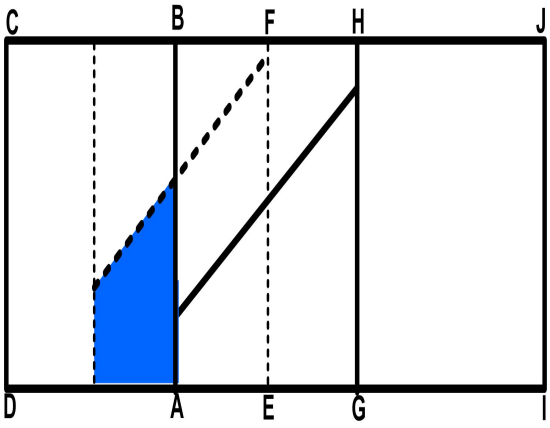
(a) Movement of fluid in +ve x direction for α>0



(b) Movement of fluid in -ve x direction for α>0



(c) Movement of fluid in +ve x direction for α<0



(d) Movement of fluid in -ve x direction for α<0

Figure 3.9: Movement of fluid in x direction

Which, on substituting the value of $u(x^n)$ from equation (3.16), becomes

$$\Rightarrow \quad x^{n+\frac{1}{2}} = x^n + \left[U_w\left(1 - \frac{x^n}{\delta x}\right) + U_e\frac{x^n}{\delta x}\right]\delta t \tag{3.18}$$

$$\Rightarrow \quad x^{n+\frac{1}{2}} = \left[1 + \left(\frac{U_e - U_w}{\delta x}\right)\delta t\right]x^n + U_w\delta t$$

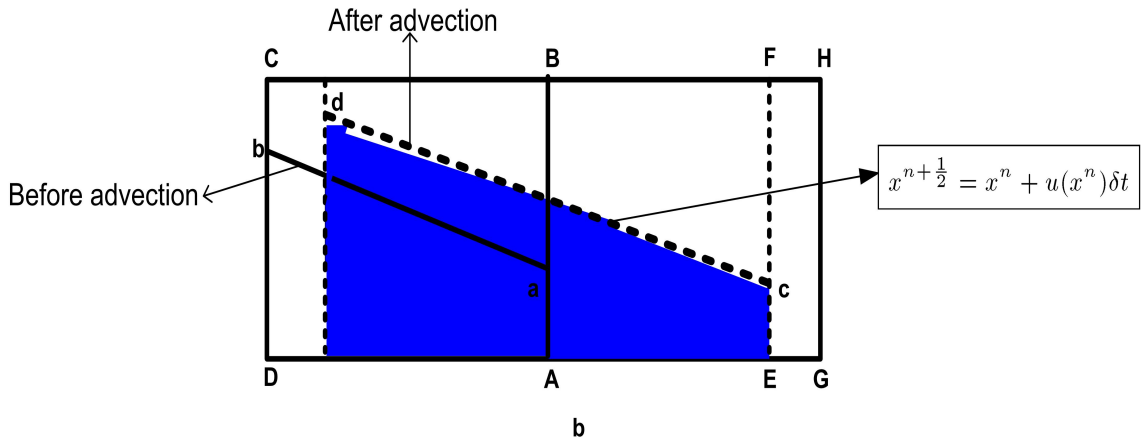


Figure 3.10: New position of the interface at next time step in +ve x direction

The superscript $(n+\frac{1}{2})$ represents the partial time step in one spatial direction x. Therefore the point x^n can be written as

$$x^n = \frac{\left(x^{n+\frac{1}{2}} - U_w \delta t\right)}{\left[1 + \left(\frac{U_e - U_w}{\delta x}\right)\delta t\right]} \tag{3.19}$$

On substituting the value of x^n from equation (3.19), the equation of the new line (i.e. equation 3.15) becomes

$$m_1^n . \frac{\left(x^{n+\frac{1}{2}} - U_w \delta t\right)}{\left[1 + \left(\frac{U_e - U_w}{\delta x}\right)\delta t\right]} + m_2^n y^n = \alpha^n \tag{3.20}$$

The coordinate y^n remains constant during the x-sweep in the advection process. Equation (3.20) can be rearranged as

$$m_1^{n+\frac{1}{2}} x^{n+\frac{1}{2}} + m_2^n y^n = \alpha^{n+\frac{1}{2}} \tag{3.21}$$

where

$$m_1^{n+\frac{1}{2}} = \frac{m_1^n}{\left[1 + \left(\frac{U_e - U_w}{\delta x}\right)\delta t\right]} \tag{3.22}$$

and

$$\alpha^{n+\frac{1}{2}} = \alpha^n + \frac{m_1^n U_w \delta t}{\left[1 + \left(\frac{U_e - U_w}{\delta x}\right) \delta t\right]} \tag{3.23}$$

Equation (3.21) defines the new line after the advection (line "cd" in Fig. (3.10)). This new line can move to the left or right of the cell faces and hence the volume fraction under line (3.15) will move to those cells. In particular, suppose $\left(\alpha^{n+\frac{1}{2}}/m_1^{n+\frac{1}{2}}\right) > \delta x$ then some portion of the volume fraction has moved to the right hand cell as depicted in Fig.(3.10).

To update the volume fraction of that cell, the volume contained under the line cd in $\pentagon$ AEFB has to be calculated. By using the coordinate transformation $x^{n+\frac{1}{2}} = (\hat{x})^{n+\frac{1}{2}} + \delta x$, where $(\hat{x})^{n+\frac{1}{2}}$ is the distance from the left face of the right cell, equation (3.21) gives

$$m_1^{n+\frac{1}{2}} \left[(\hat{x})^{n+\frac{1}{2}} + \delta x\right] + m_2^n y^n = \alpha^{n+\frac{1}{2}} \tag{3.24}$$

$$\Longrightarrow m_1^{n+\frac{1}{2}} (\hat{x})^{n+\frac{1}{2}} + m_2^n y^n = \alpha^{n+\frac{1}{2}} - m_1^{n+\frac{1}{2}} \delta x \tag{3.25}$$

which can be written as

$$m_1^{n+\frac{1}{2}} (\hat{x})^{n+\frac{1}{2}} + m_2^n y^n = \alpha' \tag{3.26}$$

where

$$\alpha' = \alpha^{n+\frac{1}{2}} - m_1^{n+\frac{1}{2}} \delta x \tag{3.27}$$

The portion of volume which has moved to the right hand cell is the area below the line defined by equation (3.26) covered in that cell. This area can be calculated by the forward relation (3.11) using the coefficients $m_1^{n+\frac{1}{2}}$, m_2 and α'.

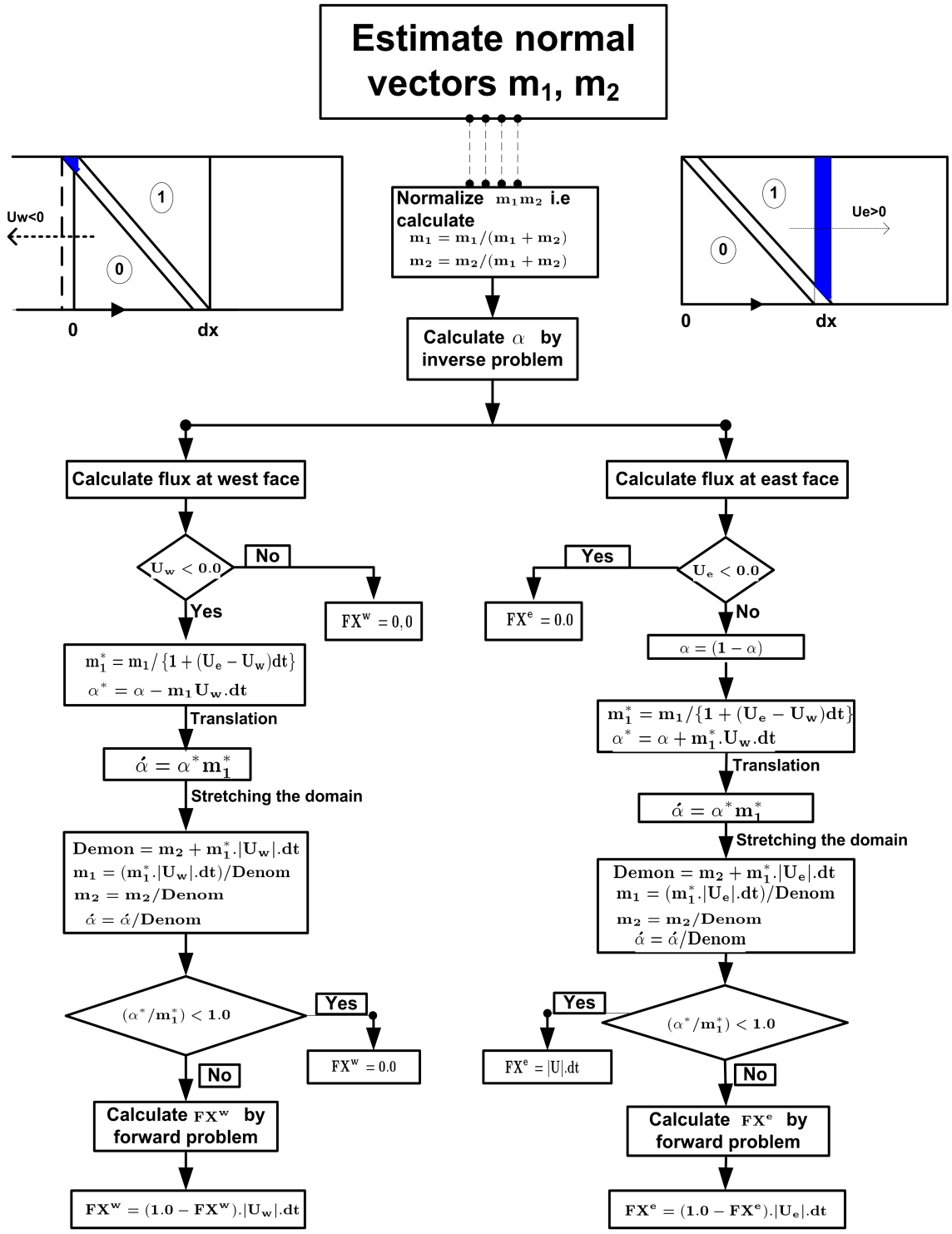


Figure 3.11: Flow chart of calculation of area in X-direction

A flow chart of the algorithm for calculating the area in x-direction for a particular normal vector position is presented in Fig. (3.11). The incoming and outgoing volume fractions are calculated for each cell in the computational domain during the x-sweep of time step. A similar procedure is applied for the y-sweep of the time step to calculate both fluxes for each cell.

Under the Courant Friedrichs Lewy (CFL) condition $\left[(max|\mathbf{V}|)\frac{\delta t}{\delta x} < \Upsilon \in (0,1)\right]$, the Lagrangian method is stable and satisfies the physical condition on the volume fraction $0 \leq C \leq 1$ (Greaves, 2004; Denis et al., 1999). The values of the volume fractions in all the cells are updated in the x direction during the x-sweep. After completing the x-sweep, the calculation of the y-sweep is carried out and the cell values in the y-direction are updated. More details of the cell values updates have been provided in the next subsection.

3.4.2 Flux Update

The fluxes (the amount of the volume fraction flow into a cell in a particular direction) in each cell are updated by time integration of equation (3.14), whereby the volume fraction of a fluid at time step n is marched forward to time step $n+1$. In this research, we have implemented the operator-split time integration scheme in which the fluxes are updated in one spatial direction first followed by the other direction in a certain time step and then the directions are changed in the next time. This scheme has been found to be more accurate than an un-split scheme (Rider and Kothe, 1998).

To understand this scheme, consider the flux update during the x-sweep and let $C^n_{i,j}$ be the volume fraction of the fluid at time step n in the cell at location (i,j). After calculating the fluxes of C into and out of this cell, the updated value of volume

fraction, due to flow in the x direction, can be defined as (Rider and Kothe, 1998)

$$\boxed{C_{i,j}^{n+\frac{1}{2}} = C_{i,j}^{n} + FX_{i,j} + dt\,(U_e - U_w)\,C_{i,j}^{n}} \tag{3.28}$$

where $FX_{i,j}$ is the change of flux at the east and the west faces in the x-direction and is defined as

$$FX_{i,j} = \left(FX_{i,j}^{e} + FX_{i,j}^{w}\right)$$

U_e is the velocity at the east face, U_w is the velocity at the west face of the cell and dt is the time step. The quantities $FX_{i,j}^{e}$ and $FX_{i,j}^{w}$ are the net flux changes at east and west faces of the cell as shown in Fig(3.12). During the flux updation, the value of $FX_{i,j}^{e}$ is calculated as the flux difference at east face cells at locations (i, j) and similar definition applied for the quality $FX_{i,j}^{w}$. The extra term in the right hand side of equation(3.28) is due to the divergence correction as explained in equation(3.14).

After completing the x-sweep, a cell is updated from the north and the south faces during the y-sweep. The updated value of the volume fraction in this case can expressed as

$$\boxed{C_{i,j}^{n+1} = C_{i,j}^{n+\frac{1}{2}} + FY_{i,j} + dt\,(U_n - U_s)\,C_{i,j}^{n}} \tag{3.29}$$

In equation (3.29), U_n, U_s are the velocities at north and south faces. $FY_{i,j}$ is the change of flux at north and south faces whose values may be defined in the similar way as for the $FX_{i,j}$. It should be noted in this equation that the updated value of the volume fraction is represented by the superscript $(n+1)$. The initial sweep direction is altered at every time step in order to minimize the numerical error that arises due

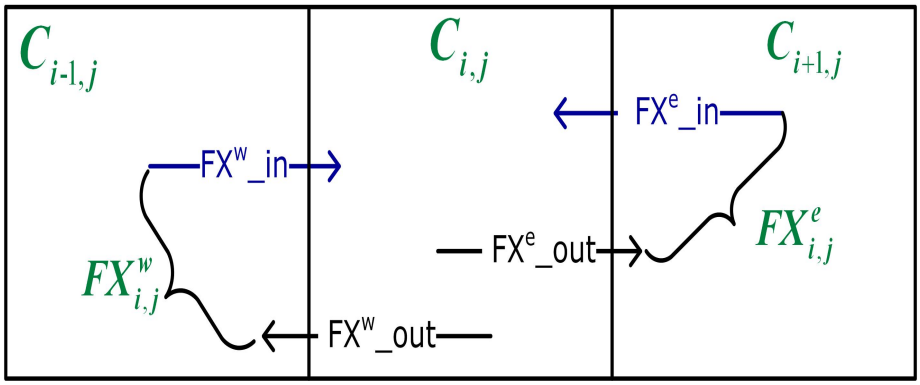


Figure 3.12: VOF update during X-sweep

to the operator splitting scheme.

3.5 Modelling the Fluid Flow

Viscous incompressible flows are governed by the Navier-Stokes equations as mentioned in chapter 2. Equations (2.1) and (2.2) can be expressed in 2D as follows,

u-momentum equation:

$$\rho\left(\frac{\partial u}{\partial t}+\frac{\partial(u^2)}{\partial x}+\frac{\partial(uv)}{\partial y}\right)=-\frac{\partial p}{\partial x}+\Gamma\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)+S_u \tag{3.30}$$

v-momentum equation:

$$\rho\left(\frac{\partial v}{\partial t}+\frac{\partial(uv)}{\partial x}+\frac{\partial(v^2)}{\partial y}\right)=-\frac{\partial p}{\partial y}+\Gamma\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right)+S_v \tag{3.31}$$

continuity equation:

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0 \tag{3.32}$$

In chapter 2, it was observed that the Finite Volume Method(FVM) is more suitable for multiphase flow problems modelling. Its advantages are summarized there. The first step in the FVM is to discretize the computational domain into a finite number of CVs.

3.5.1 Domain Discretization

In the FVM, CVs are defined in the vicinity of the nodal points of the grid where values of field variables are stored. One needs to decide first the location of these variables in the domain. One of the ways is to store all variables for velocity and pressure at the same location (i.e. using the same CV for all the variales). However, this arrangement may generate an un-physical situation, in particular, where the influence of pressure is not properly represented in the discretized momentum equation (Anderson, 1995; Versteeg and Malalasekera, 1995). This problem can be avoided by using a staggered grid for velocity components and defining the pressure variable at the nodal points. A detailed description of the staggered grid is provided below.

The Staggered Grid

The staggered grid is centred around the cell faces as depicted in Fig(3.13). The pressure variables are stored at grid nodes marked by "●" and labelled as $(I-1,J),(I,J),(I+1,J)$. The velocity components are calculated at the cell faces in between nodes and labelled using the half cell width. Specifically the u velocity components are labelled as $(i-1\,,J),(i+1\,,J)$ and marked by "▲". The symbol "★" represents the v velocity components which are labelled as $(I\,,j-1),(I\,,j+1),(I\,,j+2)$.

Therefore, three different grids (one pressure grid and two velocity grids) can be viewed within same domain. A sketch of the staggered grid is depicted in Fig.(3.13). All these three grids are clearly shown by using different colour CV. More clearly, the black colour CV represents the pressure grid which is surrounded over the symbol "●". Similarly, the u velocity grid is represented by the blue CV's and the red colour CV represents the v velocity grid. One of the advantages of this grid is that the pressure gradient is based on *adjacent* pressure points (as shown in Fig.(3.13), which eliminates the possibility of pressure oscillations which is a non-physical situation (Fletcher, 1976).

Next, the solution procedure using a staggered grid will be discussed. As pointed out before in chapter 2, one of the complexities in the mathematical model of multiphase flow problems is the coupling of pressure and velocities in the governing equations (3.30) to (3.32) . In the next subsection some details of this issue have been provided.

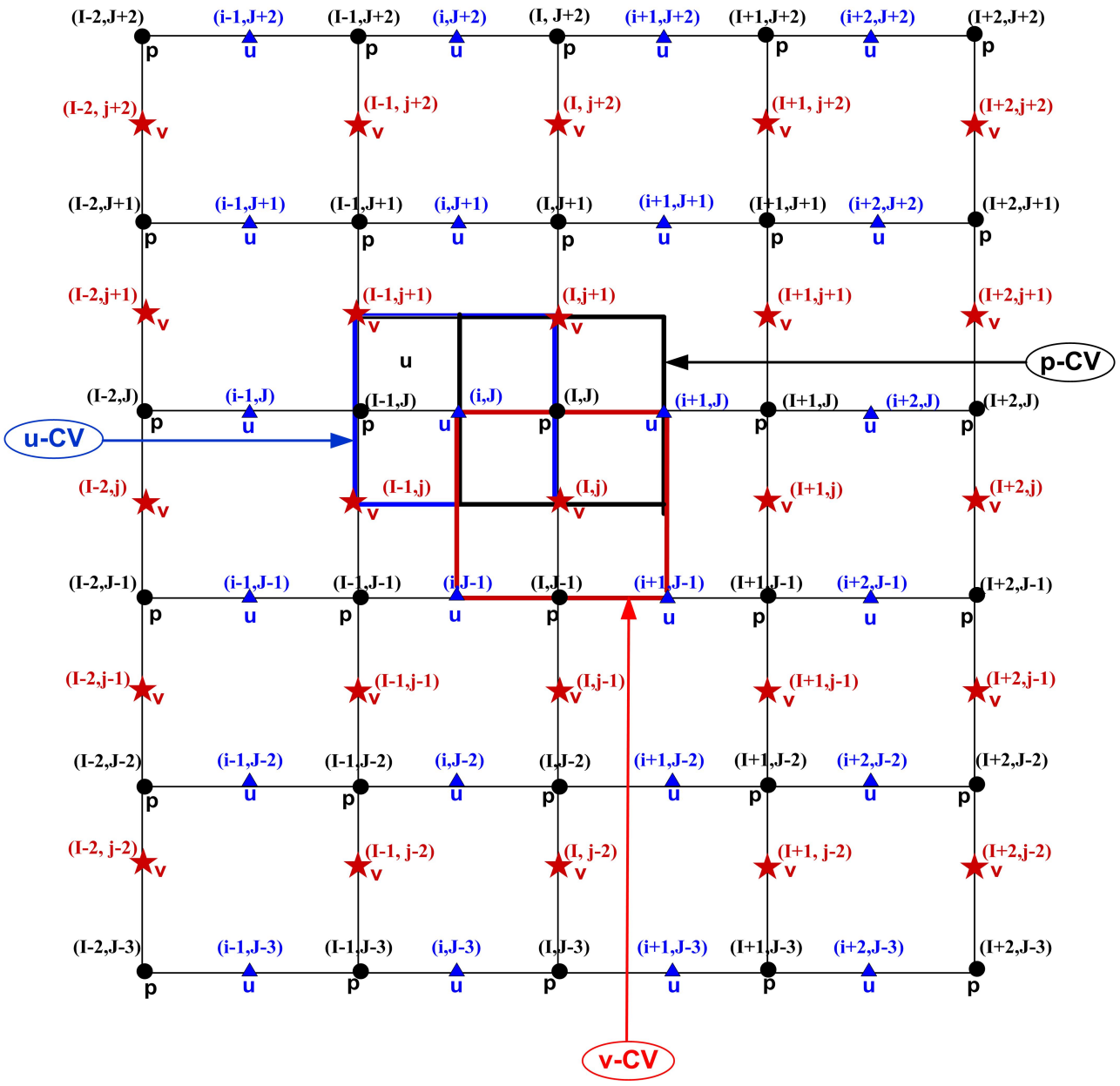


Figure 3.13: Staggered Grid

3.5.2 Pressure-Velocity Coupling

On a closer look at the N-S equations (equations (3.30) to (3.32)), the following facts may be observed:

- There is a non-linearity in the convective term of the momentum equation, for example, equation (3.30) contains the x-derivative of ρu^2 and y-derivative of ρuv.
- Every velocity component is present in each momentum equation and in the continuity equation, which means that all three equations are coupled.
- Another complex issue is the presence of a pressure term in both the momentum equations but not in the continuity equation so that there is no separate equation for the pressure term.

To get the solution of the momentum equations, the pressure should be known in advance. However, the coupling between velocity and pressure imposes a condition that the velocity field would satisfy the continuity if the correct pressure field is applied to the momentum equation.

3.6 Equation Discretization

The second step in the FVM is discretization of the governing equations according to the grid arrangement. The advantages of the staggered grid as mentioned above have led us to utilize it in this work, for discretizing the momentum and continuity equations. The discretization steps are briefly discussed in this section. First, considering the u-momentum equation, the location of u-velocity variable in Fig.(3.13) is (i, J). The

integral form of this equation imposes a conservation condition over the CV and can be expressed as:

$$\boxed{\begin{aligned}\rho\left(\frac{\partial}{\partial t}\int_{CV} u\,d\Omega \;+\; \frac{\partial}{\partial x}\int_{CV} u^2\,d\Omega \;+\; \frac{\partial}{\partial y}\int_{CV}(uv)\,d\Omega\right) \;&=\\ -\frac{\partial}{\partial x}\int_{CV} p\,d\Omega \;+\; \int_{CV}\Gamma\left(\frac{\partial^2 u}{\partial x^2} \;+\; \frac{\partial^2 u}{\partial y^2}\right)d\Omega + \int_{CV} S_u d\Omega\end{aligned}} \tag{3.33}$$

where, $\int_{CV}$ represents the integral over a CV.

The discretized equation for u-momentum equation can be expressed as

$$a_{i,J}\,u_{i,J} \;=\; \sum_{ab} a_{nb}u_{nb} + (p_{I-1,J} - p_{I,J})\;A_{i,I} + b_{i,J} \tag{3.34}$$

Equation (3.34) has been obtained from the approximation of integrals in equation (3.33) over a CV defined around $u_{(i,J)}$. The details of the intermediate steps between these two equations can be found in (Versteeg and Malalasekera, 1995). The definition of the terms involved in equation(3.34) are as follows,

(a) $A_{i,J}$ is area of the (east or west) face of the u-CV.

(b) $\sum_{nb} a_{nb}u_{nb}$ is the summation over the coefficients of neighbouring nodes. The neighbours for the node (i, J) are $(i-1, J)$, $(i+1, J)$, $(i, J+1)$ and $(i, J-1)$.

(c) $b_{i,J}$ is the momentum source term.

(d) $a_{i,J}$ is the value of u-coefficient at $(i,\ J)$.

(e) $p_{I-1,J}$ and $p_{I,J}$ are the value of pressure gradient terms.

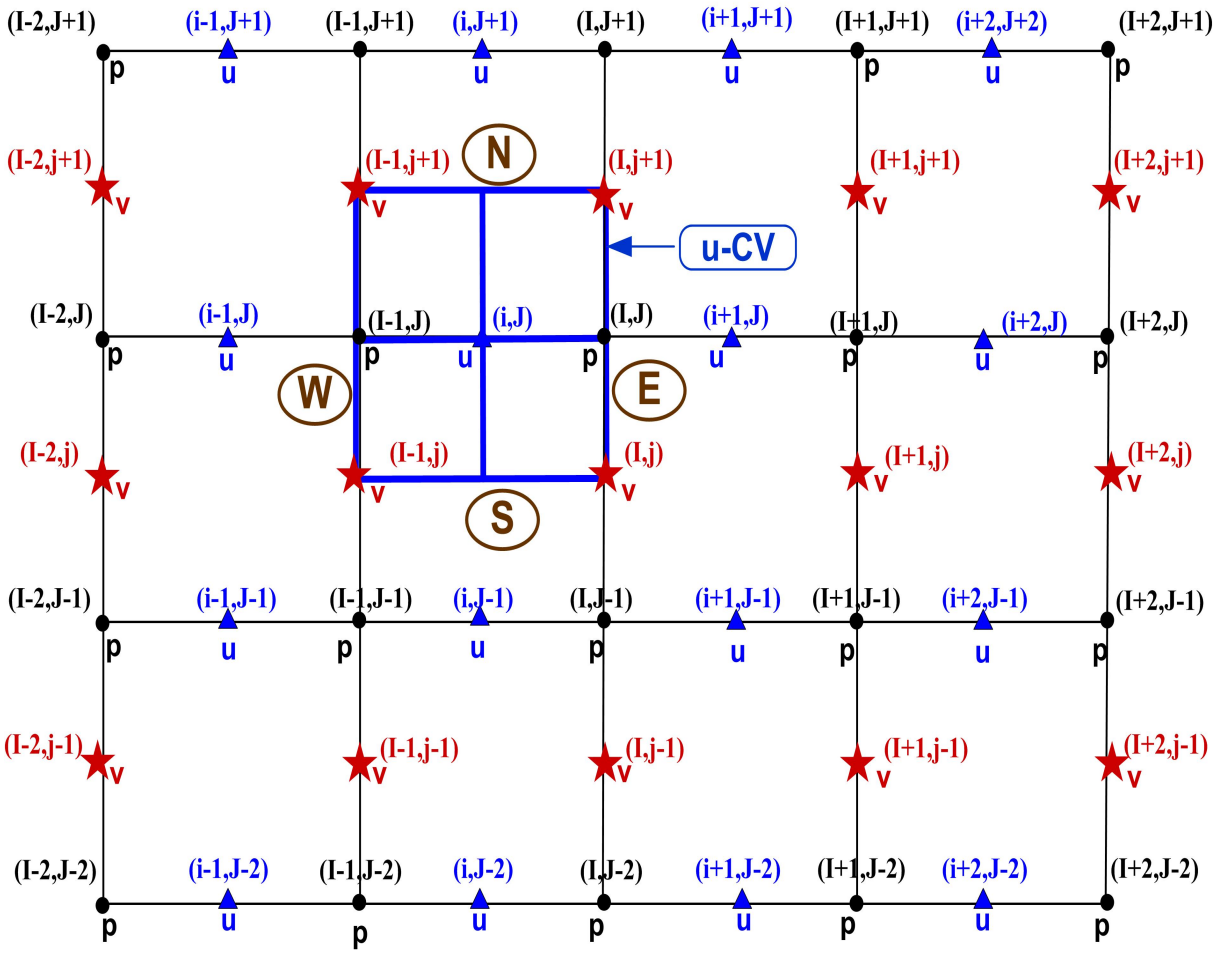


Figure 3.14: Close up of u-control volume: E,W,N,S represent east,west, north and south faces respectively.

To understand the meaning of the different terms involved in this equation, a close up of the u-Control Volume (u-CV) is shown in Fig.(3.14). The location of the neighbours as defined in item (b) above is represented in Fig.(3.14). The four faces of the CV are marked by E,W,N and S whose meanings are '*east*', '*west*', '*north*' and '*south*' respectively.

The pressure gradient in the u-CV may be calculated by taking a linear interpolation of the pressure nodes located in the CV. Differencing methods such as *Upwind* or

QUICK can be employed to calculate the values of coefficients $a_{i,J}$ and a_{nb} (Versteeg and Malalasekera, 1995). In a similar way, the discretized equation for the v-momentum equation can be expressed as

$$a_{(I,j)}v_{(I,j)} = \sum_{ab} a_{nb}u_{nb} + (p_{I,J-1} - p_{I,J})\ A_{I,j} + b_{I,j} \tag{3.35}$$

One of the conditions of pressure-velocity coupling is that the velocities obtained from equations (3.34) and (3.35) should satisfy the continuity equation (2.1) which can be discretized as

$$\left[(uA)_{i+\frac{1}{2},j} - (uA)_{i-\frac{1}{2},j}\right] - \left[(vA)_{i+\frac{1}{2},j} - (vA)_{i-\frac{1}{2},j}\right] = 0 \tag{3.36}$$

During simulation, at each time step the velocity obtained from the momentum equation should satisfy this equations.

3.6.1 Pressure Correction Method

The discretized momentum equations (3.34) and (3.35) cannot be solved directly due to the non-linearity and coupling of pressure terms. The non-linearity comes from the coefficient terms $a_{i,J}$, $a_{I,j}$ or a_{nb} which contain the velocity terms. Thus, in order to solve them, an iterative scheme is required. The relaxation-based iterative method techniques used for inviscid flows are not useful for solving equations governing viscous flows because the mathematical behaviour (the PDEs have mixed elliptic-parabolic behaviour) of these equations are different (Anderson, 1995). Another technique which is widely used for these equations is known as the *pressure correction method* developed by Patankar and Spalding (Patankar, 1980). This method is an iterative scheme where

an initial guess of pressure is used to start the iterations for solving the momentum equations.

The discrete form of the momentum equations (3.34) and (3.35) is written for every node in the appropriate staggered grid (u or v-momentum grid) implicitly which as a result generates systems of linear algebraic equations. These linear systems are solved by taking the pressure values from the previous time step. However, the velocity obtained from the solution of the linear systems does not satisfy the continuity equation (3.32). To rectify this problem, the velocities are corrected by introducing a correction term and this requires a correction term for the pressure variable. Due to the correction in pressure, these methods are called *pressure correction* methods (Patankar and Spalding, 1972).

The problems associated with the non-linearity in the momentum equations and pressure-velocity coupling can be resolved by using the SIMPLE algorithm(as discussed in section 2.4.3) which is based on a pressure correction technique.

3.7 The SIMPLE Algorithm

In this algorithm (Versteeg and Malalasekera, 1995), a pressure field, say p^*, is guessed and substituted into the momentum equations (3.34) and (3.35). The new equations can be written as

$$a_{(i,J)}u^*_{(i,J)} = \sum_{ab} a_{nb}u^*_{nb} + \left(p^*_{I-1,\,J} - p^*_{I,\,J}\right)\ A_{i,\,J} + b_{i,\,J} \tag{3.37}$$

$$a_{(I,j)}v^*_{(i,j)} = \sum_{ab} a_{nb}v^*_{nb} + \left(p^*_{I,\,J-1} - p^*_{I,\,J}\right)\ A_{I,\,j} + b_{I,\,j} \tag{3.38}$$

In these equations, the subscript "nb" denotes the neighbouring cells and $\sum a_{nb}u^*_{nb}$ has the same meaning as described in equation (3.34). Solution of these equations provides the velocities $u*$ and $v*$. Since the velocities obtained from these solutions do not satisfy the continuity equation, a correction term in the pressure and velocity fields is introduced. The updated values for pressure and velocities are defined as follows:

$$p = p^* + \acute{p} \tag{3.39a}$$

$$u = u^* + \acute{u} \tag{3.39b}$$

$$v = v^* + \acute{v} \tag{3.39c}$$

where, $\acute{u}$, $\acute{v}$ are the corrections to the velocities and $\acute{p}$ denotes the correction in the pressure term. The refined value of pressure (p^*) is obtained by solving the pressure equation. This value is then substituted back into the discretized momentum equations. Next, by manipulating the original momentum equation the new equation, i.e the equations of correct velocities can be written as (for more detail see (Patankar, 1980)),

$$u = u^* + d_u(\Delta\acute{p}_u) \tag{3.40}$$

$$v = v^* + d_v(\Delta\acute{p}_v) \tag{3.41}$$

where,

- $\Delta\acute{p}_u$: is the difference in pressure correction at the two faces (east and west face)

of u-CV as shown in Fig.(3.14).

- Similarly, $\Delta\acute{p}_v$ is the difference of the pressure correction at the north and south faces of the v-CV.
- d_u, d_v are the coefficients which comes from equations (3.34) and (3.35) and are defined as (at a particular node (i, J));

$$(d_u)_{i,J} = \frac{A_{i,J}}{a_{i,J}}, \qquad (d_v)_{I,j} = \frac{A_{I,j}}{a_{I,j}}$$

It may be observed that in the equations (3.40) and (3.41), the summation terms $\sum_{nb}$ are missing. Actually, the omission of $\sum_{nb}$ is the main approximation in the SIMPLE algorithm (Versteeg and Malalasekera, 1995) while estimating the correct velocities from equations (3.34) and (3.35).

Now, since equations (3.40) and (3.41) provide the corrected velocities, they should satisfy the discrete continuity equation (3.32) which is one of the constraints of the problem under consideration. The discrete form of equation (3.32) can be expressed as

$$[(\rho\, u\, A)_{i+1,\,J} - (\rho\, u\, A)_{i,\,J}] + [(\rho\, v\, A)_{I,\,j+1} - (\rho\, v\, A)_{I,\,j}] = 0 \tag{3.42}$$

Substitution of the corrected velocities in the equation (3.42) yields a pressure correction equation,

$$a_{I,\,J}\,\acute{p}_{I,J} = a_{I+1,\,J}\;\acute{p}_{I+1,J} + a_{I-1,\,J}\;\acute{p}_{I-1,J} + a_{I,\,J+1}\;\acute{p}_{I,J+1} + a_{I,\,J-1}\;\acute{p}_{I,J-1} \tag{3.43}$$

where,

$$a_{I,\,J} = a_{I+1,\,J} + a_{I-1,\,J} + a_{I,\,J+1} + a_{I,\,J-1}$$

$$a_{I+1,J} = (\rho\, d\, A)_{i+1,J} \;\; ; \quad a_{I-1,J} = (\rho\, d\, A)_{i,J}$$

$$a_{I,J+1} = (\rho\, d\, A)_{I,j+1} \;\; ; \quad a_{i,J-1} = (\rho\, d\, A)_{I,j}$$

$$\acute{b}_{I,J} = (\rho\, u^*\, A)_{i,J} - (\rho\, u^*\, A)_{i+1,J} + (\rho\, v^*\, A)_{I,j} - (\rho\, v^*\, A)_{I,j+1}$$

Equation (3.43) generates a system of equations whose solution provides the pressure correction values, which on substituting in equation (3.39) yields the correct values of the pressure and velocities. At each time iteration, the estimated values u^*, v^* and p^* are replaced by these corrected values. The whole SIMPLE algorithm may be summarized in the flow chart depicted in Fig.(3.15) which was adapted from (Versteeg and Malalasekera, 1995).

3.8 Chapter Conclusion

The concepts behind the solution of the multifluid model have been developed in this chapter. Implementation of both steps of VOF(PLIC) methods have been derived. An extension to the full normalized domain of an analytic relation for the reconstruction step has been established. Details of the different steps of the Lagrangian advection methods have been provided. The main features of these topics can be summarized as:

(a) The VOF method uses a step function and provides the value of the volume fraction of fluids in each CV,

(b) An analytical relation provides a way to calculate the exact value line constant and

(c) The Lagrange advection method in conjunction with an operator-splitting scheme

provides better flux update procedure.

The domain discretization and equation discretization schemes are explained in detail and the main conclusion here can be summarized as follows:

(1) the staggered grid is more suitable for storing variables as it avoids a non-physical situation,

(2) pressure-correction methods can deal with the non-linearity of the momentum equations and

(3) the SIMPLE algorithm resolves the problem of pressure-velocity coupling in the equations.

This chapter has developed the basic computational tools for the computer simulation of a multiphase flow model as depicted in Fig.(2.4). The SIMPLE algorithm, used for solving the momentum equations, generates large sparse linear system of equations. These systems are solved by iterative methods such as K-S methods as observed in the literature survey. The convergence rate of these methods can be accelerated by preconditioning techniques. Moreover, large sparse matrices for linear systems call for a special data structure. All these topics are the subject of next chapter.

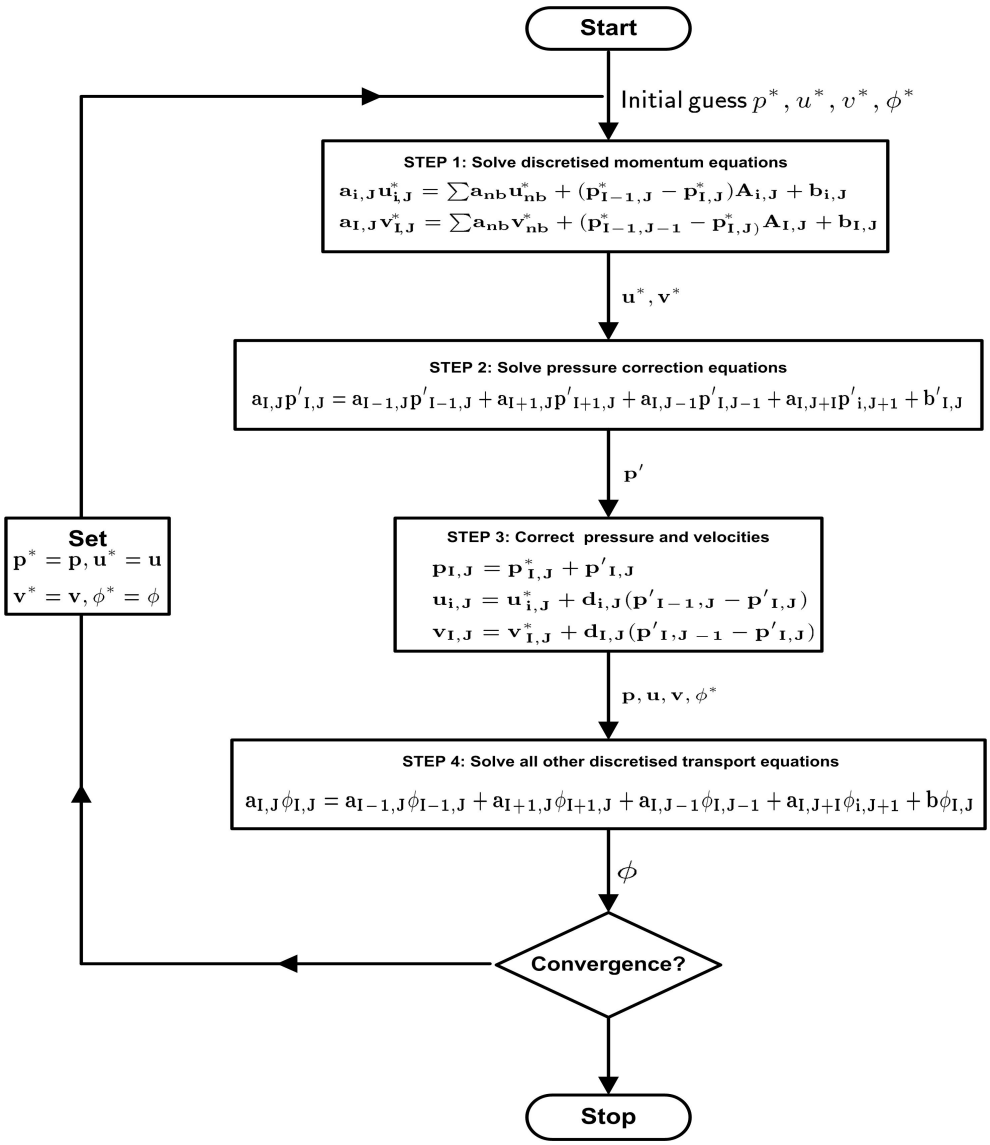


Figure 3.15: Flow chart of SIMPLE algorithm

Chapter 4

Solving Linear Systems

4.1 Introduction

The previous chapter provided details of the building blocks of the computer simulation of the multiphase flows. Whole simulation procedure can be viewed as two sub procedures: the implementation of the VOF method and the solution of the Navier-Stokes equations. The latter step involves the solution of linear systems of equations as required in the SIMPLE Algorithm. In this work, a sequential code for a Navier-Stokes solver has been put to use.

This chapter focuses on methods for solving large sparse linear systems of equations. In chapter 2, it was mentioned that the discretization of PDEs results in large sparse systems of equations $\mathbf{Ax} = \mathbf{b}$, where $\mathbf{A} \in \mathbb{R}^{n \times n}$ is an $n \times n$ matrix and $\mathbf{x}, \mathbf{b} \in \mathbb{R}^n$ are n-dimensional column vectors. The given code adopted the *index* format for storing the sparse matrices as indicated in section 2.4.4.1 and the non-zero entries of the matrix have been stored in a two dimensional array having five columns. The given code also has a library for basic matrix and vector operations in the *index* format.

Storage analysis of different sparse matrix formats is needed in order to know their memory requirements for storing the large sparse matrices. The linear system is solved by non-stationary methods which are treated with preconditioning techniques to accelerate their convergence rate. In the whole simulation of multiphase flow problems, solving the linear system has been found to consume most of the computational time. Therefore, one of the objectives of the present research is to reduce this computational time. In order to achieve this, at least two tasks are required; one of them is the development of parallel algorithms for the different parts of the solvers. The other task is to choose a specific data structure so that the computational complexity of major steps such as matrix-vector products can be reduced. In context of these issues, the present chapter covers the following topics:

- storage analysis of different sparse matrix formats,
- the reasons for choosing the diagonal format in linear systems,
- an implementation of the matrix-vector product in diagonal format necessary by the linear solvers,
- a brief overview of the stationary and non-stationary (i.e. Krylov Subspace) solvers,
- some implementation issues of the preconditioners for solvers,
- a novel algorithm for the ILUT preconditioner in the diagonal format
- some algorithmic details of the forward and backward substitution needed for this preconditioner
- a note on the computational complexity of the novel Algorithm,
- a justification of the needs of parallelization of the Algorithms for solvers and preconditioners,

- the implementation details of suitable parallel Algorithms,
- the details of the parallel Bi-CGSTAB Algorithm and
- the integration of parallel solvers with the available Navier-Stokes solver used.

After reading this chapter, it is intended that the reader will have an idea of the conclusive themes of the whole project.

4.2 Matrix Format

As mentioned in Chapter 2, the four main data structures for sparse matrices are *Compressed Sparse Coloum(CSC)*, *Compressed Sparse Row(CSR)*, *index format* and *diagonal format*. In this section we shall analyse the storage requirements of these formats. First of all, starting with CSR format, in this format all the non-zero entries

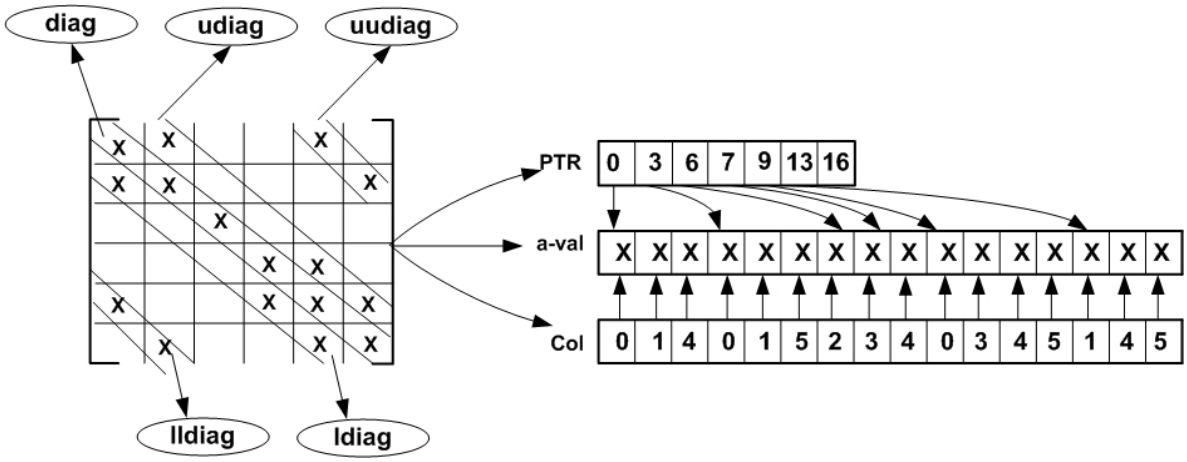


Figure 4.1: Representation of CSR format

are saved in a 1-D array of double precision say '**a_val**'. The column locations of these values are stored in another 1-D (integer) array called e.g., '**Col**'. To find the position of a new row in the array '**a_val**', the pointers to the starting point of new

row are stored in another 1-D integer array '**PTR**'. A typographical representation of CSR format of a penta-diagonal matrix is illustrated in Fig.(4.1). Similarly, in the CSC format there are three arrays namely:

- '**a_val**': nonzero elements of the matrix,
- '**row**' : an integer array of row locations of the entries and
- '**PTC**': integer array of pointers to the starting point of a new column.

In the index format, there are three arrays of the same size; two integer arrays and one double precision array, for example, '**i_index**', '**j_index**','**a_val**' respectively. These array corresponding to the matrix in Fig.(4.1) are shown in Fig.(4.2). There

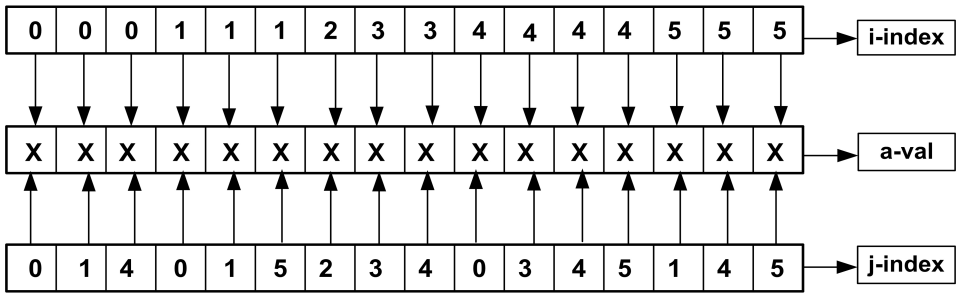


Figure 4.2: Representation of index format

are five 1-D arrays of double precision in the diagonal format each of which may be of maximum length 'n' where 'n' is the dimension of the matrix. These arrays have been named here as '**lldiag**', '**ldiag**', '**diag**', '**udiag**' and '**uudiag**' which are shown in Fig.(4.1). The maximum number non-zero entries in a penta-diagonal matrix can be $5n$. Therefore, the arrays '**a_val**', '**i_index**' and '**j_index**' will have maximum $5n$ values and the arrays '**PTR**' or '**PTC**' will contain $n + 1$ entries. To estimate the storage requirements, one needs to calculate the size of each array, defined above, in bytes. An integer variable takes 4 bytes while double precision numbers occupies

8 bytes in memory. Based on these details, the size of different arrays in bytes have been calculated and are listed in table (4.1). From table (4.1), it is clear that diagonal

Array name	**CSR/CSC**	**Index Format**	**Diagonal Format**
PTR/PTC	4n +1	×	×
col/row	20n	×	×
a_val	40n	40n	×
i_index	×	20n	×
j_index	×	20n	×
5 *diag's*	×	×	40n
Total	**64n+1**	**80n**	**40n**

Table 4.1: Storage requirement of different arrays in bytes

format occupies less space for penta-diagonal matrices in comparison to other sparse matrix formats. Once the matrix format is decided, one needs to choose an efficient iterative method and develop the algorithm in that format. The next two Sections are devoted on the details of this topic. In particular, the algorithmic to details of non-stationary iterative methods in diagonal format are provided in section 4.4.

4.3 Stationary Iterative Methods

In the stationary iterative method, from an initial solution $\mathbf{x_0}$ (guessed), the iteration scheme proceeds accordings to (Saad, 1996),

$$\mathbf{x}_{k+1} = \mathbf{H}\,\mathbf{x}_k + \mathbf{f} \tag{4.1}$$

where $\mathbf{H}$ is an iteration matrix obtained from the parts of the matrix $\mathbf{A}$ and $\mathbf{f}$ is a vector also obtained from part of $\mathbf{A}$ multiplication with vector $\mathbf{b}$. The matrix $\mathbf{A}$ is decomposed as $\mathbf{A} = \mathbf{D} + \mathbf{L} + \mathbf{U}$, where $\mathbf{D}$ is diagonal, $\mathbf{L}$ is lower and $\mathbf{U}$ is the upper

part of matrix $\mathbf{A}$. Examples of basic stationary iterative methods include (Golub and Van Loan, 1996),

Jacobi Method: In this method, $\mathbf{H_{JA}} = \mathbf{D^{-1}(L + U)}$ and $\mathbf{f} = \mathbf{D^{-1}b}$

Gauss-Seidel Method: Here, $\mathbf{H_{GS}} = \mathbf{(D - L)^{-1}U}$ and $\mathbf{f} = \mathbf{(D - l)^{-1}b}$

The necessary condition for convergence of stationary iterative methods is that the spectral radius of the iteration matrix $\rho(\mathbf{H}) < \mathbf{1}$ in magnitude (Saad, 1996). In other words, the magnitude of the maximum eigenvalue of the iteration matrix $\mathbf{H}$ should be less than 1. It has been observed that the matrices arising from the present problems are unsymmetrical and ill-conditioned, and the magnitude of their maximum eigenvalue is always more than 1. Therefore, the stationary methods are not suitable for solving the linear systems having these matrices. An alternative choice for the solution of ill-conditioned sparse linear systems is non-stationary methods which are described in the next section.

4.4 Non-stationary Iterative Methods

These methods are based on projection methods in which the approximate solution $\tilde{\mathbf{x}}$ is extracted from subspace of $\mathbb{R}^n$ with some constraints. The definition of the projection method can be written as (van der Vorst, 2003), find $\tilde{\mathbf{x}}$

$$\tilde{\mathbf{x}} \in \mathbf{x_0} + \mathbf{K} \quad \text{such that } \mathbf{b} - \mathbf{A}\tilde{\mathbf{x}} \perp \mathbf{L} \tag{4.2}$$

where $\mathbf{K}$ is the search space, $\mathbf{L}$ is the space of constraints and $\mathbf{x_0}$ is the initial guess of the solution. The search space is constructed by the residual vector $\mathbf{r_0} = \mathbf{b} - \mathbf{Ax_0}$

and is defined as (Saad, 1996; van der Vorst, 2003)

$$\mathbf{K_m}(\mathbf{A}, \mathbf{r_0}) = \text{span}\left\{\mathbf{r_0}, \mathbf{Ar_0}, \mathbf{A^2 r_0}, \cdots, \mathbf{A^{m-1} r_0}\right\} \tag{4.3}$$

where $\mathbf{m}$ is the dimension of the space and span means the set of all possible linear combinations of vectors. The space $\mathbf{K_m}(\mathbf{A}, \mathbf{r_0})$ (hereafter will be denoted by $\mathbf{K_m}$)

Choose initial $\mathbf{x_0}$ and calculate $\mathbf{r_0} = \mathbf{b} - \mathbf{Ax_0}$;
Choose $\tilde{\mathbf{r}} = \mathbf{r^0}$, k=1. ;
Calculate $\rho^1 = < \tilde{\mathbf{r}}, \mathbf{r_0} >$;
while *($\rho^k > tolerance$)* **do**
  if *($k == 1$)* **then**
    $\mathbf{p^k} = \mathbf{r^k}$
  else
    $\beta^k = \left(\frac{\rho^k}{\rho^{k-1}}\right) \times \left(\frac{\alpha_k}{\omega_k}\right)$
  endif
  Solve $\tilde{\mathbf{p}}^k$ from $\mathbf{M}\tilde{\mathbf{p}}^k = p^k$ //Apply preconditioner ;
  $\mathbf{w}^k = \mathbf{A}\tilde{\mathbf{p}}^k$;
  $\alpha^k = \frac{\rho^k}{<\tilde{\mathbf{r}}^{\mathbf{T}}, \mathbf{w}>}$;
  $\mathbf{s}^k = \mathbf{r}^k - \alpha_k \tilde{p}^k$;
  Solve $\tilde{\mathbf{s}}_k$ from $\mathbf{M}\tilde{\mathbf{s}}_k = \mathbf{s}_k$ //Apply preconditioner ;
  $\mathbf{t}^k = \mathbf{A}\tilde{\mathbf{s}^k}$;
  $\omega^k = \frac{<\mathbf{t}^k, \mathbf{s}^k>}{<\mathbf{t}^k, \mathbf{t}^k>}$;
  $\mathbf{x}^k = \mathbf{x}^{k-1} + \alpha_k \tilde{\mathbf{p}}^k + \omega^k \tilde{\mathbf{s}}^k$ //update solution vector ;
  $\rho^k = < \tilde{\mathbf{r}}^k, \mathbf{r}^k >$;
  k = k+1 ;
endw

Algorithm 4.1: Preconditioned Bi-Conjugate Gradient Stabilised (BiCGSTAB) algorithm(Pommerell, 1999).

is a subspace of $\mathbb{R}^n$ and is called the *Krylov Subspace* and the methods based on this search space are called *Krylov Subspace (K-S) Methods* (Saad, 1989, 1996). Based on the choice of constraints space $\mathbf{L}$, these K-S methods can be divided (van der Vorst, 2003; Sun et al., 2009) into four categories:

The Ritz-Galerkin Approach: $\mathbf{L} = \mathbf{K}_m \implies \tilde{\mathbf{x}} \in \mathbf{x_0}+\mathbf{K}_m$ such that $\mathbf{b} - \mathbf{A}\tilde{\mathbf{x}} \perp \mathbf{K}_m$

The Minimal Residual Approach: Find $\tilde{\mathbf{x}}$ such that $\|\mathbf{b} - \mathbf{A}\tilde{\mathbf{x}}\|_2$ is minimal over $\mathbf{K}_m$

The Petrov-Galerkin Approach Choose $\mathbf{L} \neq \mathbf{K}_m \implies \tilde{\mathbf{x}} \in \mathbf{x_0}+\mathbf{K}_m$ such that $\mathbf{b} - \mathbf{A}\tilde{\mathbf{x}} \perp \mathbf{L}$

The Minimum Error Approach: Search $\tilde{\mathbf{x}} \in \mathbf{A^T K}_m(\mathbf{A^T r_0})$ such that $\|\tilde{\mathbf{x}} - \mathbf{x}\|_2$ is minimal.

The first approach leads to the Conjugate Gradient Method (CGM). The Generalized Minimal RESidual (GMRES) belongs to the minimal residual approach but for unsymmetric systems this approach leads to long recurrence relations for approximate solutions due to renormalization, therefore, another search space $\mathbf{L}$ is chosen to relieve this relation and falls in the Galerkin category (van der Vorst, 2003).

The Bi-Conjugate Gradient Stabilised (Bi-CGSTAB) is a hybrid approach and gives smooth convergence for unsymmetrical matrices (Pommerell, 1999). Nordsveen and Moe (Nordsveen and Moe, 1999) applied Bi-CGSTAB to the solution of 2D transient two-phase flows and documented a considerable decrease in the computational time in comparison to methods such as Gauss-Seidel method. It is this convergence behaviour of the Bi-CGSTAB method which motivates author to employ it for solving linear systems of equations arising from the present problem. An algorithm for Bi-CGSTAB is provided in Algorithm 4.1.

4.5 Sparse Matrix-Vector Product

From storage analysis of different sparse matrix formats, it has been observed that the diagonal format occupies less space to store the matrix. Therefore, we adopted

this format in the present work. It may be observed from algorithm 4.1 that the Bi-CGSTAB method requires two matrix vector products. Since the product of a matrix and a vector consumes most of the computational time (Pichel et al., 2009; Shahnaz et al., 2006), a new algorithm has been developed in the diagonal format with the intention to reduce this time. For penta-diagonal matrices, the offset distance of lower/upper diagonal, viz., '**ldiag**' or '**udiag**' from the main diagonal '**diag**' has been denoted by 'offset1'. Similarly, the offset distance from the main diagonal to **uudiag/lldiag** has been denoted by 'offset2'. These notations are elucidated in Fig. (4.3) and, by using them a new algorithm for sparse matrix-vector product has been designed which is delineated in Algorithm 4.2. As it can be seen from this algorithm

1 **for** $(i = 0, i < n, i++)$ **do**
2 **if** $((\mathrm{i} - \mathrm{offset1}) < 0)$ **then**
3 $\mathbf{ax}_i = (\mathbf{diag})_\mathbf{i} \times \mathbf{x}_i + (\mathbf{udiag})_\mathbf{i} \times \mathbf{x}_{(i+\mathbf{offset1})} + (\mathbf{uudiag})_\mathbf{i} \times \mathbf{x}_{(i+\mathbf{offset2})}$
4 **endif**
5 **if** $((\mathrm{i} - \mathrm{offset1}) >= 0$ && $(\mathrm{i} - \mathrm{offset2}) < 0)$ **then**
6 $\mathbf{ax}_i = (\mathbf{ldiag})_\mathbf{i} \times \mathbf{x}_{(i-\mathrm{offset1})} + (\mathbf{diag})_\mathbf{i} \times \mathbf{x}_i + (\mathbf{udiag})_\mathbf{i} \times$
  $\mathbf{x}_{(i+\mathrm{offset1})}(\mathbf{uudiag})_\mathbf{i} \times \mathbf{x}_{(i+\mathrm{offset2})}$
7 **endif**
8 **if** $((\mathrm{i} - \mathrm{offset2}) >= 0$ && $(\mathrm{i} - \mathrm{offset2}) < n)$ **then**
9 $\mathbf{ax}_i = (\mathbf{lldiag})_\mathbf{i} \times \mathbf{x}_{(i-\mathrm{offset2})} + (\mathbf{ldiag})_\mathbf{i} \times \mathbf{x}_{(i-\mathrm{offset1})} + (\mathbf{diag})_\mathbf{i} \times \mathbf{x}_i +$
  $(\mathbf{udiag})_\mathbf{i} * \mathbf{x}_{(i+\mathrm{offset1})} + (\mathbf{uudiag})_\mathbf{i} \times \mathbf{x}_{(i+\mathrm{offset2})}$
10 **endif**
11 **if** $((\mathrm{i} + \mathrm{offset2}) >= 0$ && $(\mathrm{i} + \mathrm{offset2}) < n)$ **then**
12 $\mathbf{ax}_i = (\mathbf{lldiag})_\mathbf{i} \times \mathbf{x}_{(i-\mathrm{offset2})} + (\mathbf{ldiag})_\mathbf{i} \times \mathbf{x}_{(i-\mathrm{offset1})} + (\mathbf{diag})_\mathbf{i} \times \mathbf{x}_i +$
  $(\mathbf{udiag})_\mathbf{i} \times \mathbf{x}_{(i+\mathrm{offset1})}$
13 **endif**
14 **if** $((\mathrm{i} + \mathrm{offset1}) >= n)$ **then**
15 $\mathbf{ax}_i = (\mathbf{lldiag})_\mathbf{i} \times \mathbf{x}_{(i-\mathrm{offset2})} + (\mathbf{ldiag})_\mathbf{i} \times \mathbf{x}_{(i-\mathrm{offset1})} + (\mathbf{diag})_\mathbf{i} \times \mathbf{x}_i$
16 **endif**
17 **endfor**

Algorithm 4.2: Matrix-vector multiplication in diagonal format.

that only one '*for*' loop is required for computing the product while in any other matrix format two such loops are required (Shahnaz et al., 2006; Straubhaar, 2008; Williams et al., 2009). Using '*if*' conditions based on the offset distances of the diagonals, one '*for*' loop has been eliminated. Elimination of a loop from the Algorithm reduces its computational complexity. Consequently Algorithm 4.2 saves memory as well as reduces the computational time.

4.6 Implementation of Preconditioners

The second important step in Algorithm 4.1 is the employment of a preconditioner as shown in lines 10 and 14. Before discussing the implementation details of preconditioners, it is necessary to a closer look at the reasons why they are necessary.

4.6.1 The Need for a Preconditioner

As mentioned above in Chapter 3, to simulate multifluid flow, the VOF method has been implemented. This method treats the mixture of two fluids as one fluid which is determined by the interface. During the advection, the mesh is kept fixed and the interface is reconstructed from the values of the colour function and its gradients in a grid cell as explained in Section 3.3. These values contribute to the calculation of the coefficient matrix entries. Since the fluid moves at each time step, it has been observed that the magnitudes of the matrix entries change because of the changes in the interface position. Due to this change, the condition number (the ratio of the highest to the lowest eigenvalues as defined in Section 2.4.4.2) of the matrix may vary.

A matrix with a high condition number makes Krylov Subspace solvers converge slowly and, so, in order to increase the convergence rate, preconditioning techniques are

applied (Saad, 1992; Sun et al., 2009). As mentioned in Chapter 2, ILU, SSOR and DS are the most appropriate preconditioners for the solvers. In general, the performance of the ILUT preconditioner has been found to be better than other versions of the ILU preconditioner (Saad, 1996). A discussion on the implementation details of the ILUT preconditioner is provided in the next subsection.

4.6.2 Implementation Procedure

In ILU factorization, the original matrix $\mathbf{A}$ is decomposed into two matrices, viz., $\mathbf{L}$ and $\mathbf{U}$. Its Algorithm in dense format is given in Algorithm 4.3. In this Algorithm, there are three nested '*for*' loops required (steps 1,2 and 9). These nested loops

```
1  for (i = 0 to n) do
2  |  for (j = i + 1 to n) do
3  |  |  U_{j,i} = A_{j,i} ;
4  |  |  if (U_{j,i} == 0.0) then
5  |  |  |  L_{j,i} = 0.0 ;
6  |  |  else
7  |  |  |  L_{j,i} = U_{j,i} / U_{i,i} ;
8  |  |  endif
9  |  |  for (k = i + 1 to n) do
10 |  |  |  if (U_{j,k} ≠ 0.0) then
11 |  |  |  |  U_{j,k} -= L_{j,i} × U_{i,k}
12 |  |  |  endif
13 |  |  endfor
14 |  endfor
15 |  L_{i,i} = 1.0 ;
16 endfor
```

Algorithm 4.3: Dense ILUT algorithm (Saad, 1996).

generate the data dependency of the matrix elements of $\mathbf{L}$ and $\mathbf{U}$. This dependency implies that for calculating the elements of the $(i+1)^{th}$, $(i+2)^{th}$ rows, the elements from the i^{th} or previous row are required. This data dependency is a hindrance to

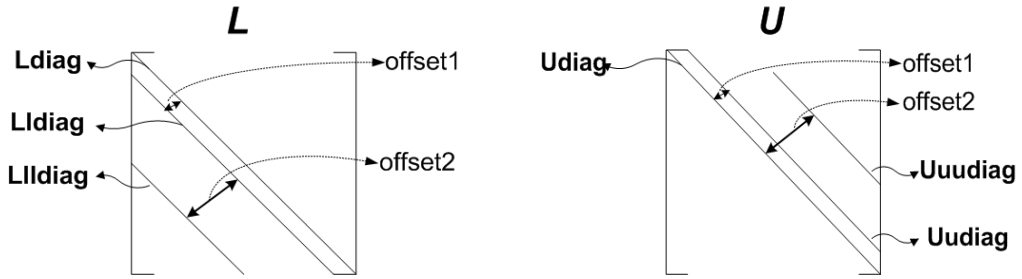
parallelising the Algorithm (Basermann, 2000). In the parallel version, the matrix is divided into different parts which are available on different processors of the parallel computer (Li, 2005). Therefore, the elements of previous rows may not be available on the same processor and those elements have to be brought from other processors.

4.6.3 The Sparse Version of the ILUT Preconditioner

In this subsection we will discuss a new sparse version of the ILUT algorithm. A novel approach to this Algorithm for penta-diagonal matrices is proposed in this thesis which is summarised in Algorithm 4.4. The literature has been examined and no paper for ILUT algorithm for exact penta-diagonal matrices which replaces the three loops with one loop has been found. The main feature of this Algorithm is that it requires only one '*for*' loop in comparison to three loops in the dense ILU Algorithm and hence it reduces the data dependency. There are five conditions in this Algorithm which are related to offset distances of the matrix diagonals. Using these conditions, two nested '*for*' loops of the Algorithm 4.3 have been incorporated by setting the values of $j =$ offset1 and $j =$ offset2.

It can be observed in Algorithm 4.4 that only one '*for*' loop is utilized from line 1 to line 72. It should be noted that line number 36 is continued again in the next page thus line numbers 37 and 38 are repeated; the reader can read the Algorthim from line 1 to 36 then in the next page continue from line 38 to 72. In this algorithm the arrays '**Ldiag**', '**Lldiag**' and '**Llldiag**' represent the diagonals of matrix **L**. Similarly, the arrays with prefix **U** denote the diagonals of matrix **U**. A graphical representation of the arrays of the **U** and **L** matrices is illustrated in Fig.(4.3). The diagonals of the matrix **L** are shown in Fig. 4.3(a). In this figure, offset1 is the distance of the main diagonal from the lower diagonal and offset2 is the distance from the diagonal below

than the lower diagonal. Similarly the three diagonals of the matrix **U** are depicted in Fig. 4.3(b). Employment of the ILU or the SSOR preconditioner transforms the original linear system in to a preconditioned system,



(a) Penta-diagonal L matrix (b) Penta-diagonal U matrix

Figure 4.3: Representation of penta-diagonal **L** and **U** matrices

$$\mathbf{LU}\tilde{\mathbf{p}} = \mathbf{b} \tag{4.4}$$

This system can be decomposed into two systems

$$\mathbf{Lz} = \mathbf{b} \tag{4.5a}$$

$$\mathbf{U}\tilde{\mathbf{p}} = \mathbf{z} \tag{4.5b}$$

These two systems are solved by *forward* and *backward* substitution procedures. Here $\tilde{p}$ is solution vector obtained from preconditioned system. New Algorithms have been developed for these two procedures in the diagonal format. The algorithm for forward substitution is presented in Algorithm 4.5. In this algorithm, only one '*for*' loop is required which contains three '*if*' conditions based on the diagonal 'offset' values. The 0^{th} element is calculated before the '*for*' loop because the loop has to be started from $i = 1$. Otherwise, it will try to access the memory location z_{0-1} at line 5 which is out of the allocated memory segment. A similar algorithm for backward substitution in diagonal format has also been developed.

```
1  for (0 to n − 1) do copy U = A
   //copy arrays diag, ldiag etc. to Udiag Uldiag ;
2  for (i = 0; i < n; i++) do
3      if (i − offset1) < 0 then                          /* first condition */
4          [j = i + offset1] ;
5          if ((Uldiag)_j ≠ 0.0) then (Lldiag)_j = (Uldaig)_j/(Udiag)_i ;
6          if (Udiag ≠ 0.0) then (Udiag)_j − = (Lldiag)_j × (Uudiag)_i ;
7          if ((Uudiag)_j ≠ 0.0) then
8              if (j − i > 1) then
9                  (Uudiag)_j = (Uudiag)_j;
10             else
11                 (Uudiag)_j − = (Lldiag)_j × (Uuudiag)_i
12             endif
13         endif
14         [j = i + offset2] ;
15         if ((Ulldiag)_j ≠ 0.0) then (Llldiag)_j = (Ulldiag)_j/(Udiag)_j ;
16         if ((Uldiag)_j ≠ 0.0) then
17             if ((j − i) > 2) then
18                 (Uldiag)_j = (Uldiag)_j ;
19             else
20                 (Uldiag)_j − = (Llldiag)_j × (Uudiag)_i ;
21             endif
22         endif
23         if ((Udiag)_j ≠ 0.0) then
24             (Udiag)_j − = (Llldiag)_j × (Uuudiag)_i ;
25             (Uudiag)_j = (Uudiag)_j;
26             (Uuudiag)_j = (Uuudiag)_j;
27         endif
28     endif
29     if ( (i − offset1) >= 0 && (i − offset2) < 0) then      /* second condition */
30         [j = i + offset1] ;
31             ... ;
32             ... ;
33         [j = i + offset2] ;
34             ... ;
35             ... ;
36     endif
37 endfor
38 Note: The previous line is not the end of the 'for' loop and line 36 is continued at line 38 in the next page.
```

```
37 for ( Same loop condition from line 2 of previous page ) do
38 |  Continuation from line 36 of previous page                    if
   |  ( (i − offset2) >= 0 && (i + offset2) < n) then     /* third condition */
39 |  |  [ j = i + offset1 ] ;
40 |  |  if ((Uldiag)_j ≠ 0.0) then (Lldiag)_j = (Uldiag)_j/(Udiag)_i;
41 |  |  if ((Udiag)_j ≠ 0.0) then
   |  |  (Udiag)_j = (Udiag)_j − (Llldiag)_j × (Uudiag)_i ;
42 |  |  if ((Uudiag)_j ≠ 0.0) then
43 |  |  |  if ((j − i) > 1) then
44 |  |  |  |  ((Uudiag)_j = (Uudiag)_j ;
45 |  |  |  else
46 |  |  |  |  (Uudiag)_j − = (Lldiag)_j × (Uuudiag)i ;
47 |  |  |  endif
48 |  |  endif
49 |  |  [ j = i + offset2 ] ;
50 |  |  if ((Ulldiag)_j) ≠ 0.0 then (Llldiag)_j = (Ulldiag)_j/(Udiag)_j ;
51 |  |  if ((Uldiag)_j ≠ 0.0) then
52 |  |  |  if ((j − i) > 2) then
53 |  |  |  |  (Uldiag)_j = (Uldiag)_j
54 |  |  |  else
55 |  |  |  |  (Uldiag)_j − = (Llldiag)_j × (Uudiag)_i ;
56 |  |  |  endif
57 |  |  endif
58 |  |  if ((Udiag)_j ≠ 0.0) then  (Udiag)_j − = (Llldiag)_j × (Uuudiag)_i
59 |  endif
60 |  if ( (i + offset2) >= n && (i + offset1) < n) then  /* fourth condition */
61 |  |  [ j = i + offset1 ]  ;
62 |  |  |  |  |  |  |  |  |  |  ... ;
63 |  |  [ j = i + offset2 ]  |  |  |  |  |  |  |  |  |  |  ... ;
64 |  endif
65 |  if ( (i + offset1) >= n then                          /* fifth condition */
66 |  |  [ j = i + offset1 ]  ;
67 |  |  if (j <= n − 1) then
68 |  |  |  if ((Uldiag)_j ≠ 0.0) then  (Lldiag)_j = (Uldiag)_j/(Udiag)_j ;
69 |  |  |  if (Udiag ≠ 0.0) then  (Udiag)_j − = (Lldiag)_j × (Uudiag)_i ;
70 |  |  endif
71 |  endif
72 endfor
```

Algorithm 4.4: ILUT Algorithm in diagonal format

```
1  z_0 = r_0/(Ldiag)_0;
2  for (i = 1; i < n; i++) do
3      if ((i − offset1) < 0) then
4          if ((offset1 − i) <= 0) then
5              z_i = r_i − (Lldiag)_i × z_(i−offset1) / (Ldiag)_i
6          else
7              z_i = r_i / (Ldiag)_i
8          endif
9      endif
10     if ((i − offset1) >= 0 && (i − offset2) < 0) then
11         z_i = r_i − (Lldiag)_i × z_(i−offset1) / (Ldiag)_i
12     endif
13     if ((i − offset2) >= 0) then
14         z_i = r_i − ((Llldiag)_i × z_(i−offset2) + (Lldiag)_i × z_(i−offset1)) / (Ldiag)_i
15     endif
16 endfor
```

Algorithm 4.5: Forward substitution algorithm in diagonal format

4.7 Computational Complexity

In Algorithm 4.3, there are three '*for*' loops. Its computational complexity can be calculated by observing the number of counts in each loop. The outer loop runs from 0 to $n-1$ and other two loops run from $i+1$ to $n-1$. Therefore, the total number of counts can be expressed by the formula

$$\sum_{i=0}^{n}(n-i)^2 \tag{4.6}$$

which has complexity of order $\mathcal{O}(n^3 - n^2)$. In Algorithm 4.4, there is only one '*for*' loop so its complexity can be given as $\mathcal{O}(n)$. The matrix-vector product in Algorithm

4.2, requires only one '*for*' loop. Therefore, its computational complexity can also be given as n. On the other hand the computational complexities of the matrix-vector products in other sparse formats (Shahnaz et al., 2006; Straubhaar, 2008) are $5n$.

4.8 The Need for Parallelization

We have seen that the diagonal format saves space and reduces the computational complexity in comparison to other sparse matrix formats. However, in the entire simulation, solving large systems of equations takes a large percentage of the total computational time, when run on a single computer. Details of the average time taken by the main parts of the simulation of the bubble rising problem are provided in Table 4.2. The simulation has been run for bubble radius 0.003 m with $dt = 1.0 \times 10^{-5}$ seconds and grid size 128×128.

Steps	Time(sec)
Coefficients	355.19
Solving **U** & **V** system	1192.49
Solving **P** system	32498.30
Continuity residual	34.80
Writing data	22.27
VOF	460.37
Total Simulation Time	35606

Table 4.2: Time taken by different steps for the simulation of bubble rise problem

In this table, *coefficients* means the time required for calculating the coefficient of the $\mathbf{U}, \mathbf{V}$ velocities and pressure systems of equations. Solving the (**U** & **V** and **P**) systems indicate the time taken by the Bi-CGSTAB solver for solving these linear systems. The time required for calculating the continuity residual and writing the data file are represented by the continuity residual and writing date respectively. VOF

represents the time taken by the VOF method. Total simulation time is the total computational time required for the simulation (it is not the sum of the above times).

The alternate approach to reduce the computational time is the development of parallel algorithms for linear solvers. The parallel versions of the Bi-CGSTAB method with the ILU and SSOR preconditioners in diagonal format have been developed in this work using the MPI. The next section provides the details of the steps involved in parallel algorithms.

4.9 Implementation of Parallel Algorithms

In the development of a parallel version of an algorithm, the first step is to find out the components which can be parallelized. In the case of preconditioned Krylov Subspace

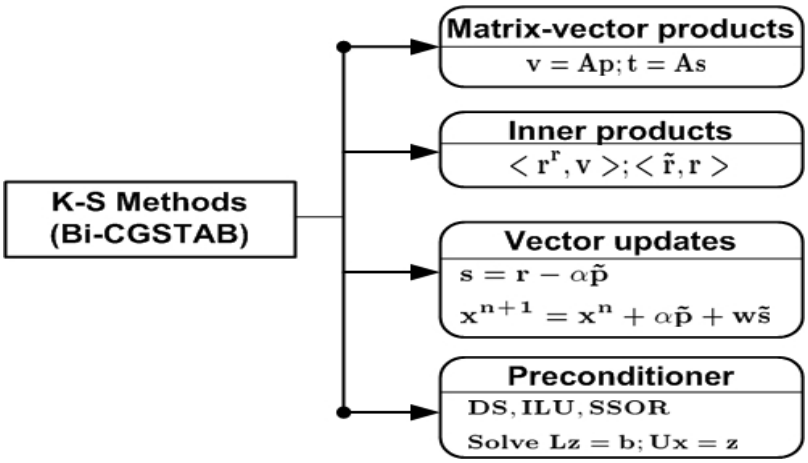


Figure 4.4: Main computational steps for K-S methods corresponding to Algorithm 4.1

methods namely Bi-CGSTAB, the main computational steps as can be seen in Fig.4.4.

- two matrix-vector products,
- the inner products,
- vector updates and

- the generation of the preconditioned linear system

These steps corresponding to Algorithm 4.1 are elucidated in Fig.4.4. The second step is to choose the parallel paradigm as discussed in Chapter 2. This section provides the implementation details of these topics.

4.9.1 Parallelism in Krylov Subspace Methods

The main computational steps in K-S methods involve operations on matrices and vectors as depicted in Fig.(4.4). The important tasks in the parallelization of these

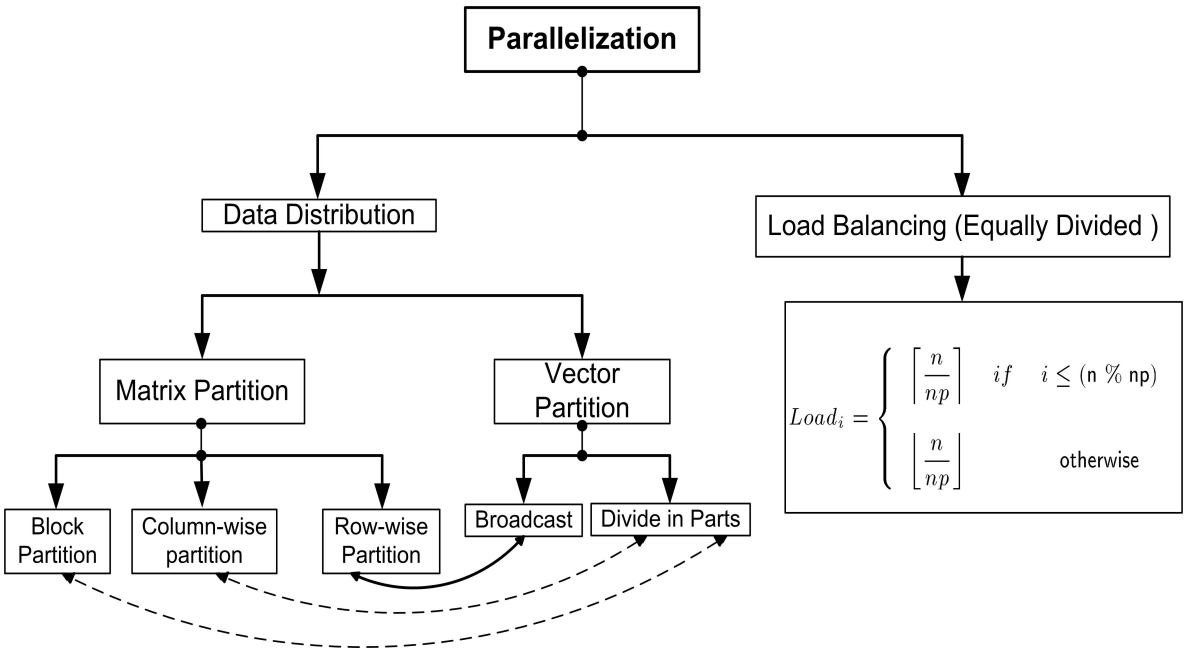


Figure 4.5: Parallelization steps for Krylov Subspace methods

methods are data distribution and load balancing on different nodes of the parallel computer as shown in Fig.(4.5). This figure shows that one of the features of this task is to divide vectors into parts or broadcast the vectors to all processors depending

upon the matrix partition. The matrix or vector partition is carried out using a load balancing scheme described in Section(4.9.3) and summarised below.

The partitioning of the matrix and vectors may be carried out by dividing them into parts and distributing the parts using a load balancing scheme as indicated in Fig.(4.5) which contains the formula for static load balancing. In this figure, the notations $\lceil\ \rceil$ and $\lfloor\ \rfloor$ represent the *ceiling* and *floor*[1] functions respectively. The variables n and np are the order of the matrix and the number of available processors respectively and $Load_i$ is the amount of the data to be held by the i^{th} processor. It is observed from Fig.(4.5) that in this load balancing scheme, two functions, viz., *ceiling* and *floor* are called during the load balance calculation. As pointed out in Section 2.5.2, these

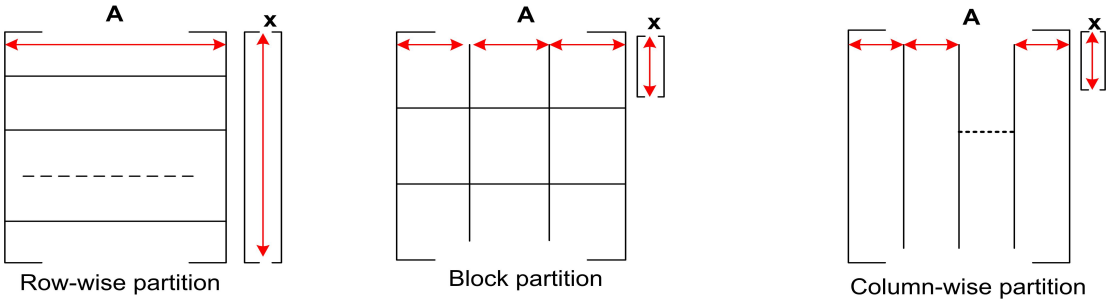


Figure 4.6: Different ways of matrix partition

functions divide the total number of processors into two parts called '*lowered numbered half*' and '*upper numbered half*' (Jordan and Alaghband, 2003). Hence, every processor in a part has equal load of the data, however a processor from another part may have different amounts of the data. A similar strategy can be adopted for inner product operations.

[1] $\lceil x \rceil = \min\{n \in \mathbb{Z} \mid n \geq x\}$ and $\lfloor x \rfloor = \max\{n \in \mathbb{Z} \mid n \leq x\}$ where $\mathbb{Z}$ is set of all integers. For example, $\lceil 2.4 \rceil = 3$ and $\lfloor 2.9 \rfloor = 2$.

4.9.2 Data Distribution for Parallel the Krylov-Subspace Method

Generally, matrix partitioning is carried out in three different ways as depicted in Fig. (4.6) (Karniadakis and Kirby, 2003; Quinn, 2004):

- row-wise partition,
- block partition and
- column-wise partition.

For row-wise matrix partition, the whole vector is required for matrix-vector products, as can be seen in Fig.(4.6). Therefore, there is no need to divide the vectors required for the matrix-vector products (Shang, 2009). Because of this feature we used this partitioning scheme to decompose the matrix $\mathbf{A}$ in this work. Thus, the matrix is distributed across np processors as the of rows. Further, the vectors required for matrix-vector multiplication are replicated on all the processors. In the master-slave paradigm

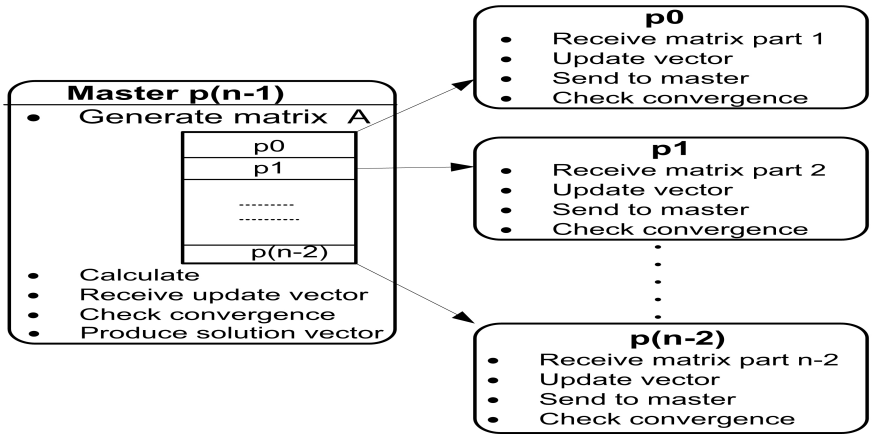


Figure 4.7: Representation of master slave paradigm

(Mezher and Philippe, 2002), the initialization of data (i.e. matrix $\mathbf{A}$, vector $\mathbf{b}$) takes place on the master processor. The initialized data is then distributed to all processors using an efficient load balancing scheme described below. The master processor also

executes all other required I/O operations. All the processors, including the master, execute the algorithm on their own part of the data. If any need arises, the slave processors can also communicate to each other, for example, in case of parallelization of the ILU and ICH preconditioners, the i^{th} processor sends some data to the $(i+1)^{th}$ processor.

4.9.3 A Novel Load Balancing Scheme

In this subsection, a new load balancing scheme will be discussed. The load balancing

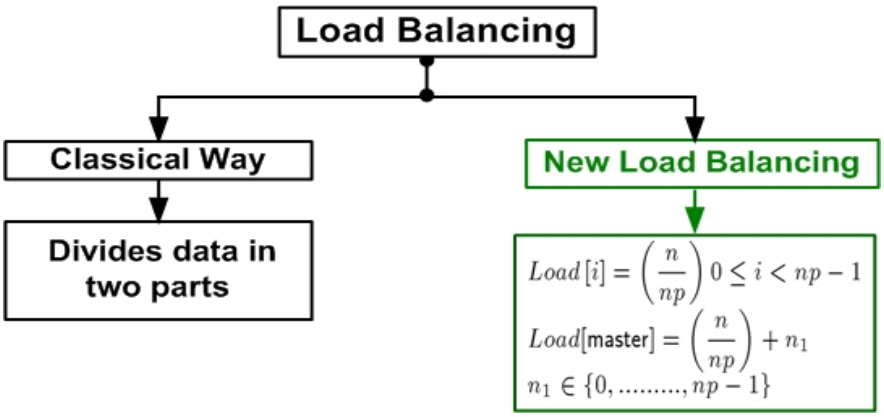


Figure 4.8: The novel load balancing scheme

scheme, as depicted in Fig.(4.5), has been implemented in a different way in this work, so as to facilitate data distribution. Fig.(4.8) delineates the new load balancing scheme. In this scheme, the slave processors are indexed from 0 to $(np-2)$ while the master processor has the value $(np-1)$, np being the total number of available processors. Using this new scheme, the master processor divides the whole data n by np, i.e. dsize$(= n/np$: an integer division), and then sends the dsize amount of the data to the slave processors. The remaining amount of data (dsize $+ n_1$) is stored by the master processor. Functions *ceiling* and *floor* are not called in this scheme. Consequently,

the new scheme reduces extra calculations during data distribution. Furthermore, there is equal data distribution among all the slave processors, in particular, each slave processor contains the dsize amount of data (either rows of a matrix or the elements of a vector). During the parallelization of ILUT every processor exchanges data with every other one. This scheme allows easier data calculation to be sent from one processor to the next.

Moreover, this scheme incorporates only a little extra load on the master processor. As shown in Fig.(4.8), the extra load on the master processor is n_1, where n_1 can be either the number of extra rows of a matrix or the number of extra elements of a vector whose partition is carried out. The maximum value of n_1 is equal to $(np-1)$. Therefore, when the matrix size is quite large, for example $\mathcal{O}(10^6)$, the value of n_1 is negligible. As $(np-1)$ is the last number in the index, the master processor does not send data to any other processor. Hence, the extra load on the master processor does not affect the communication process.

4.9.4 Parallelization of the Sparse Matrix-Vector Product

The above load balancing scheme has been implemented for data distribution i.e. for partitioning the matrix. After data distribution, each processor receives its appropriate part of the matrix. Algorithm 4.2 for the matrix-vector product is implemented on all processors and parts of the product vector **Ax** are calculated in parallel. According to the new load balancing scheme, the master processor calculates the last part of the vector. All the slave processors send their part of the vector **Ax** to the master processor. These parts of the vector are placed in the whole vector held by the master processor.

4.9.5 Parallelization of the ILUT algorithm in diagonal format

We have seen earlier in this chapter that the ILUT algorithm in diagonal format saves memory and reduces the computational complexity. Now we will discuss how the parallel version of Algorithm 4.4 can be developed using the new load balancing scheme defined above. Algorithm 4.6 provides an overview of the main parallelization steps for the master ILUT functions.

```
1  Calculate data size m_dsize.      //using new load balancing scheme ;
2  for (m_dsize to n − 1) do
3  |  copy U = A                     //copy arrays diag, ldiag etc. to Udiag Uldiag ;
4  endfor
5  for (0 to offset2 − 1) do
6  |  receive data from processor (np − 2) ;
7  endfor
8  for (m_dsize to n − 1) do
9  |  implement all five conditions of Algorithm 4.4
10 endfor
```

Algorithm 4.6: Steps for master ILUT function

As a part of the discussion on this algorithm, a few remarks have been provided.

Remarks for Algorithm 4.6

1. At step number 6, there are 7 arrays which require the elements from the previous processor. These arrays are '**L_lldiag**', '**L_ldiag**', '**U_lldiag**', '**U_ldiag**', '**U_udiag**' and '**U_uudiag**'. Therefore, a total of 7×offset2 elements are needed to be sent by the previous processor. All these elements are stored in a buffer array say '**buf**' in a particular order (e.g., the elements from array '**L_lldiag**' are stored at the first offset2 location followed by the element from the array '**L_ldiag**' and so on) and received by calling the *MPI_ receive* function only once. After receiving array '**buf**', the elements of particular arrays (like '**L_lldiag**'or

'**L_ldiag**') are extracted using the same order (as they were stored).

2. Note that the arrays '**U_lldiag**' and '**U_ldiag**' (which are not shown in Fig. 4.3(b)) contain the elements of their original matrix arrays, viz., '**lldiag**' and '**ldiag**' as they are copied from the original matrix at step 3.

3 Since the master processor resides at the end of the processor index, it does not have to send the array elements to the next processor.

4 After receiving elements from processor $(np-2)$, the master processor implements all the five conditions of the Algorithm and generates the elements of the last part of the matrices **L** and **U**.

```
1  Calculate data size dsize.      //using new load balancing scheme ;
2  for (0 to dsize − 1) do
3  |  copy sU = A                  //copy arrays diag, ldiag etc. to sUdiag sUldiag ;
4  endfor
5  if ((myid > 0) && (myid < np − 1)) then
6  |  for (0 to offset2) do
7  |  |  receive data from previous processor ;
8  |  endfor
9  endif
10 for (0 to dsize) do
11 |  implement all five conditions of Algorithm 4.4
12 endfor
13 for (0 to offset2) do
14 |  (buf)_i            = (sLlldiag)_{i+dsize}        //fill the array buf ;
15 |  (buf)_{i+offset2}   = (sLldiag)_{i+dsize} ;
16 |  (buf)_{i+2×offset2} = (sUlldiag)_{i+dsize} ;
17 |  (buf)_{i+3×offset2} = (sUldiag)_{i+dsize} ;
18 |  (buf)_{i+4×offset2} = (sUdiag)_{i+dsize} ;
19 |  (buf)_{i+5×offset2} = (sUudiag)_{i+dsize} ;
20 |  (buf)_{i+6×offset2} = (sUuudiag)_{i+dsize} ;
21 endfor
22 Send buf to next processor
```

Algorithm 4.7: Steps for slave ILUT function

Next, we will discuss the slave part of the ILUT function. As mentioned before, the slave processors have their indices numbered from 0 to $np-2$ and the index $np-1$ is reserved for the master processor. Each processor gets its index as one of the function parameters 'myid'. The main steps of the slave ILUT function are presented in Algorithm 4.7.

Remarks for Algorithm 4.7

(1) A question in Algorithm 4.7 arises regarding the need of sending the array '**buf**' to the next processor. To answer this question, one has to look back to Algorithm 4.4. In each condition (except for condition 5) of this Algorithm, the value of $j = \mathsf{i} + \mathsf{offset2}$. Therefore, for a particular value i, the array elements up to the location $(\mathsf{i} + \mathsf{offset2})$ are calculated. Moreover, for calculating these elements, new values of elements of other arrays are required. To clarify this, observe line 15 of Algorithm 4.4. At this step, the value of **Ulldiag** at location $(\mathsf{i} + \mathsf{offset2})$ is needed for calculating the value of **Llldiag** at the same location. This value has not yet been calculated on this processor but was calculated on the previous processor.

(2) For the 0^{th} slave processor these values are available from the original matrix values as they have been copied already at line 3. This processor then calculates the elements up to the location $(\mathsf{dsize}-1)+\mathsf{offset2}$. Thus, the 0^{th} processor calculates the offset2 extra elements of all arrays. At the 1^{st} processor– which has elements of the location from dsize to $(2\times\mathsf{dsize}-1)$, the extra offset2 elements calculated by 0^{th} processor are required.

(3) Similarly, the extra offset2 elements calculated by the first processor are required by the next (2^{nd}) processor and so on.

(4) Since the 0^{th} processor lies at the first location in the processor index, it does

not need to receive the elements from any previous processor. This condition is assured by the statement at line 5 of this Algorithm.

4.9.6 Parallelization of forward substitution in diagonal Format

The details of the steps involved in the parallelization of Algorithm 4.5 are presented in this subsection. Algorithm 4.8 provides the main steps of the slave's forward

```
1  if ((myid > 0) && (myid < np − 1)) then
2  |  Receive data from previous processor ;
3  endif
4  for (0 to dsize) do
5  |  i1 = i + myid × dsize ;
6  |  i2 = i + offset2 ;
7  |  if ( (i1 − offset1) < 0) then
8  |  |  if ((offset1 − 1) < 0) then
9  |  |  |  (sz)_i2 = ((sr)_i − (sLldiag)_i × (sz)_i2−offset1) / (sLdiag)_i
10 |  |  else
11 |  |  |  (sz)_i2 = (sr)_i / (sLdiag)_i
12 |  |  endif
13 |  endif
14 |  if ( (i1 − offset1) >= 0) && (i1 − offset2) < 0) then
15 |  |  (sz)_i2 = ((sr)_i − (sLldiag)_i × (sz)_i2−offset1) / (sLdiag)_i
16 |  endif
17 |  if ((i1 − offset2) >= 0) then
18 |  |  (sz)_i2 = ((sr)_i − (sLlldiag)_i × (sz)_i2−offset2 − (sLldiag)_i × (sz)_i2−offset1) / (sLdiag)_i
19 |  endif
20 endfor
21 for (j = 0 to offset2) do                    /* fill buffer sbufz */
22 |  (sbufz)_j = (sz)_j+dsize
23 endfor
24 Send sbufz to next processor
```

Algorithm 4.8: Steps for slave forward substitution function

substitution function.

Remarks for Algorithm 4.8

(1) The algorithm for the master part is similar to this Algorithm except the steps from lines 19-22. This is because the master does not need to send elements to the next processor.

(2) The processor at 0^{th} position does not receive data from any other processor as indicated in lines 1-3.

(3) Each processor calculates the offset2 extra elements which are required by the next processor.

(4) The values of the vector **sz** (or vector **z** for the master processor) are required to calculate the preconditioned solution vector **r** as explained in equations 4.5b and 4.5a.

4.10 Parallelization of Bi-CGSTAB

The complete parallelization of the preconditioned Bi-CGSTAB method has been developed using the parallel Algorithms already developed in this chapter for the matrix-vector product, the ILUT preconditioner and forward and backward substitution. The scalar variables α, β and ω are calculated by the master processor and then broadcast to all processors. Parallel implementation of the preconditioners has been performed by the master and all slave processors.

The main steps accomplished by the master processor are listed in Algorithm 4.9. As can be seen in this algorithm, the preconditioner is implemented by calling its

master function. For the case of the ILUT preconditioner, there is some communication required as mentioned in the remarks of Algorithms 4.6 and 4.7. But in the case of the Symmetric Successive Over Relaxation (SSOR) and Diagonal Scaling preconditioners, no communication is required among processors because there is not data dependency involved. At line number 35 of this Algorithm, the master processor calculates the norm of the residual vector $\mathbf{r}$, viz., $||\mathbf{r}||$ and broadcasts it to all processors. It is this norm which is used to check the convergence criteria by all the processors. The convergence condition is satisfied by all the processors by utilizing a *while* loop.

The Bi-CGSTAB method steps carried out by the slave processors are presented in Algorithm 4.10. All the scalar variables, viz. α, β and ω are received from the master processor. The requisite parts of the inner-products are calculated and then sent to the master processor. Similarly, the parts of the vectors (such as the residual vector $\mathbf{r}$, the solution vector $\mathbf{x}$ etc) are sent to the master after updating. The norm $||\mathbf{r}||$ is received from the master which is used for checking the convergence condition. It can be noticed here that no communication between the slave processors is required. The slave processors only communicate with the master processor.

After convergence, the master processor holds the full solution vector $\mathbf{x}$ which can be the solution of the velocity ($\mathbf{u}$ or $\mathbf{v}$ velocity) system. This solution vector is then assigned to the appropriate velocity vector. After copying the solution vectors, the master processors broadcast it to all the processors because it is also required by the slave processors to copy into the appropriate vectors. Furthermore, the velocity solution is required to calculate the coefficients of the pressure systems as indicated in Fig.(4.9).

1 Calculate data size, set $k = 1$;
2 Calculate matrix-vector product $\mathbf{Ax}$ // master part ;
3 Receive parts of matrix-vector product from other processors. ;
4 Accommodate parts of these product in the whole vector $\mathbf{ax}$;
5 Calculate residual vector $\mathbf{r} = \mathsf{b} - \mathbf{ax}$ // master part ;
6 Receive and accommodate the residual vector. ;
7 Calculate ρ_0 and $||\mathbf{b}||$ and broad cast them to all processors ;
8 **while** (convergence) **do**
9 | **if** $(k == 1)$ **then**
10 | | $\mathbf{p}^k = \mathbf{r}^k$ // master part ;
11 | **else**
12 | | $\beta = \frac{\alpha^k}{\omega^k} * \frac{\rho^k}{\rho^{k-1}}$;
13 | | $\mathbf{p}^k = \mathbf{r}^k + \beta(\mathbf{p}^{k-1} - \omega^k \times \mathbf{w}^k)$;
14 | | Broadcast β to all processors ;
15 | **endif**
16 | Solve $\mathbf{M}\tilde{\mathbf{p}}^k = \mathbf{p}^k$ // master part;
17 | Receive parts of $\tilde{\mathbf{p}}$ from other processors and accommodate them into whole vector ;
18 | Broadcast $\tilde{\mathbf{p}}$ to all processors ;
19 | Calculate matrix-vector product $\mathbf{w}^k = \mathbf{A}\tilde{\mathbf{p}}$ // master part ;
20 | Calculate inner product $< \mathbf{r}^T, \mathbf{w} >$ // master part;
21 | Receive parts of inner-product and add all of them. ;
22 | Calculate α and broadcast to all processors. ;
23 | Calculate vector $\mathbf{s}^k = \mathbf{r}^k - \alpha^k \mathbf{w}^k$ // master part;
24 | Receive part of vector $\mathbf{s}$ and put them into whole vector. ;
25 | Solve preconditioer $\mathbf{M}\tilde{\mathbf{s}} = \mathbf{s}$ // master part;
26 | Calculate matrix-vector product $\mathbf{t}^k = \mathbf{A}\tilde{\mathbf{s}}^k$ // master part;
27 | Receive parts of vector $\mathbf{t}^k$, place them in to whole vector and broadcast the whole vector. ;
28 | Calculate inner products $< (\mathbf{t}^T)^k, \mathbf{s}^k >$ and $< (\mathbf{t}^T)^k, \mathbf{t}^k >$ // master part;
29 | Receive the parts of these inner-products and add them. ;
30 | Calculate $\omega = \frac{<(\mathbf{t}^T)^k, \mathbf{s}^k>}{<(\mathbf{t}^T)^k, \mathbf{t}^k>}$ and broadcast to all processors. ;
31 | Update the residual vector and $\mathbf{r}^k = \mathbf{s}^k - \omega * \mathbf{t}^k$ // master part;
32 | Receive parts the vector $\mathbf{r}$ and place them into whole vector. ;
33 | Update the solution vector $\mathbf{x}^k = \mathbf{x}^{k-1} + \alpha\tilde{\mathbf{p}} + \omega\tilde{\mathbf{s}}$ // master part;
34 | Receive parts the vector $\mathbf{x}$ and place them into whole vector. ;
35 | Calculate $||\mathbf{r}^k||$ and broadcast. ;
36 | $k = k + 1$;
37 **endw**

Algorithm 4.9: Outline of master part of parallel Bi-CGSTAB algorithm

1 Calculate data size, set $k = 1$;
2 Calculate matrix-vector product $\mathbf{Ax}$ // slave part ;
3 Send part of matrix-vector product to master. ;
4 Calculate residual vector $\mathbf{r} = \mathbf{b} - \mathbf{ax}$ // slave part ;
5 Receive ρ_0 and $||\mathbf{b}||$ from master ;
6 **while** (convergence) **do**
7 **if** $(k == 1)$ **then**
8 $\mathbf{p}^k = \mathbf{r}^k$ // slave part ;
9 **else**
10 Receive β from master. ;
11 $\mathbf{p}^k = \mathbf{r}^k + \beta(\mathbf{p}^{k-1} - \omega^k * \mathbf{w}^k)$;
12 **endif**
13 Solve $\mathbf{M}\tilde{\mathbf{p}}^k = \mathbf{p}^k$ // slave part;
14 Send part of $\tilde{\mathbf{p}}$ to master ;
15 Receive whole vector $\tilde{\mathbf{p}}$ from master ;
16 Calculate matrix-vector product $\mathbf{w}^k = \mathbf{A}\tilde{\mathbf{p}}$ // slave part ;
17 Calculate inner product $< \mathbf{r}^T, \mathbf{w} >$ // slave part;
18 Send this part of inner-product to master. ;
19 Receive α from master. ;
20 Calculate vector $\mathbf{s}^k = \mathbf{r}^k - \alpha^k \mathbf{w}^k$ // slave part;
21 Send part of vector $\mathbf{s}$ to master. ;
22 Solve preconditioer $\mathbf{M}\tilde{\mathbf{s}} = \mathbf{s}$ // slave part;
23 Calculate matrix-vector product $\mathbf{t}^k = \mathbf{A}\tilde{\mathbf{s}}^k$ // slave part;
24 Send part of vector $\mathbf{t}^k$, to master. ;
25 Calculate inner products $< (\mathbf{t}^T)^k, \mathbf{s}^k >$ and $< (\mathbf{t}^T)^k, \mathbf{t}^k >$ // slave part;
26 Send the parts of these inner-products to master. ;
27 Receive ω from master. ;
28 Update the residual vector and $\mathbf{r}^k = \mathbf{s}^k - \omega * \mathbf{t}^k$ // slave part;
29 Send part the vector $\mathbf{r}$ to master ;
30 Update the solution vector $\mathbf{x}^k = \mathbf{x}^{k-1} + \alpha\tilde{\mathbf{p}} + \omega\tilde{\mathbf{s}}$ // slave part;
31 Send part the vector $\mathbf{x}$ to master. ;
32 Receive $||\mathbf{r}^k||$ from master. ;
33 $k = k + 1$;
34 **endw**

Algorithm 4.10: Outline of slave part of parallel Bi-CGSTAB algorithm

4.11 Integration of Parallel Solvers into N-S Solver

As mentioned in the chapter introduction, the sequential N-S solver code was made available for this research.

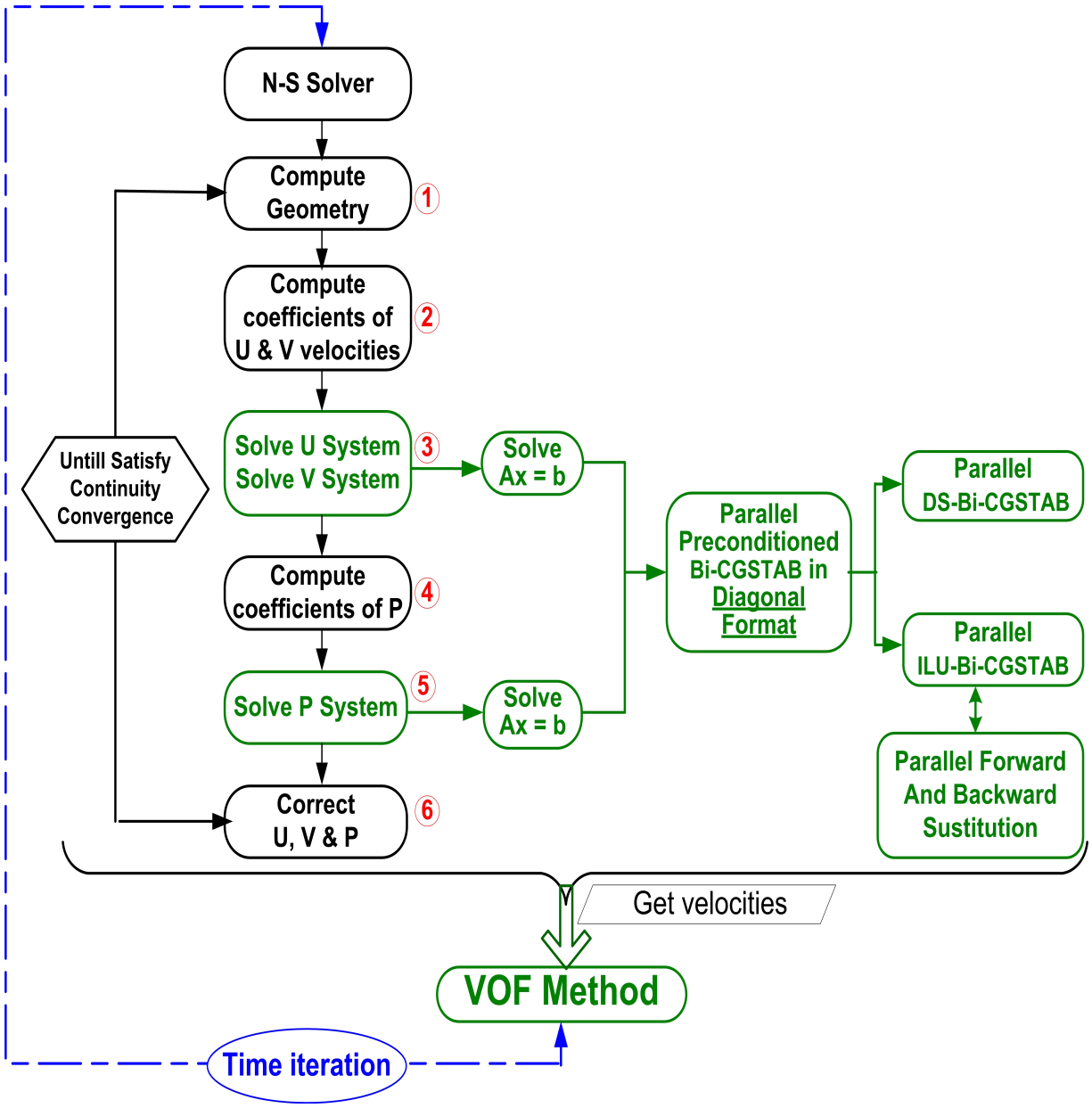


Figure 4.9: Flow chart of whole code

The original code had adopted the index format for storing sparse matrices. In the present work, the diagonal format has been chosen and all the algorithms have been developed in this format. The outline of the whole code for one time iteration is shown in the flow chart depicted in Fig.(4.9). As shown in this figure, steps (3) and (5) require solving the linear system of equations. One of the main objectives of the present work is to demonstrate the power of preconditioners and reduce the computational time, therefore these steps have been parallelized. The original code was developed in C++, therefore, all the functions have been developed in object oriented way using MPI routines. The driver function of the code, i.e., the 'main' function calls only two functions. The first function initializes the necessary parameters which include:

- initialization of geometry,
- reading the parameter from the input file,
- calling the solvers

In other words, the first function corresponds to step (1) of the flowchart. The second function executes all the steps from (2) to (6). The parallel version of the 'main' function contains these two functions with additional parameters processor identification number 'myid' and the value of total number of processors 'np'. The master and slave versions of the functions are called based on the value of the variable 'myid'. Each processor generates part of the matrix separately at each time step, hence avoiding distribution of the matrix by the master processor. After matrix generation the parallel version of the BiCGSTAB method (master or slave) is called.

4.12 Chapter Conclusions

Details of the novel algorithms developed for the linear solvers have been provided in this chapter. A global picture of the whole computer simulation of multiphase flow problem has been created. The first half of the chapter (i.e. up to Section 4.6) focuses on the development of the solvers and preconditioners in the diagonal format. The second part gives details of the development of parallel versions of these algorithms. The last section throws light on the implementation of the whole code.

The main conclusive points from both parts have been summarized here. The first part deals with the solvers and preconditioners in diagonal format. Their main conclusions can be summarised as follows:

(1) the diagonal format occupies less memory for storing penta-diagonal matrices,

(2) solving the linear system consumes most of the computational time of the simulation.

(3) a short description of the iterative methods has been provided,

(4) the Bi-CGSTAB method requires four inner-products and two matrix-vector products,

(5) these products are developed in diagonal format reducing the computational complexity of the solver,

(6) the need for precondtioners has been highlighted ,

(7) a novel algorithm for ILUT in diagonal format has been shown to have reduced computational complexity in comparison to dense format and

(8) the backward and forward substitution procedures have been developed in the diagonal format

Details of parallel versions of the novel ILUT algorithm and Bi-CGSTAB methods have been provided in the second half. The important points of this part can be summarized as follows,

(1) the main computational steps of Krylov Subspace methods have been described.

(2) the master-slave paradigm has been used to design the parallel Algorithms,

(3) a new load balancing scheme has been shown to provide easier data distribution,

(4) using this scheme, the parallelization of novel ILUT Algorithms has been developed,

(5) similarly the parallel version of backward and forward substitutions in the diagonal format required for preconditioner has been developed using a new load balancing scheme and

(6) using these parallel Algorithms, the parallel version of Bi-CGSTAB method has been developed.

(7) the parallel Bi-CGSTAB method has been integrated into the Navier-Stokes solver.

Now all the parallel computational tools (C++ code with MPI) for simulating the multiphase flow phenomena have been developed. The developed code will be employed to simulate the following problems:

(1) lid driven cavity flow,

(2) a single air bubble rising,

(3) dam breaking problems and

(4) the shape advection test for validating the VOF code.

The next chapter will provide the results obtained from the simulation of these problems which also include the performance of the preconditioners applied to BiCGSTAB. Analysis and discussions of these results are also provided in that chapter.

Chapter 5

Results and Discussions

5.1 Introduction

As mentioned in chapter 1, the model implemented in this research includes two main parts:

1. the Navier-Stokes(N-S) solution methods which includes the solution of linear systems of equations and
2. the methods for multifluid modelling.

The VOF (PLIC) method has been employed, in this work, for the second part. An analytic relation has been implemented for the interface reconstruction step and a Lagrangian advection method has been used to advect the interface. This method has been validated in the first stage with pure advection problems.

The N-S code as the PDE solver in conjunction with VOF code has been validated by simulating benchmark problems involving one or more fluids. A number of standard benchmark problems are mentioned in chapter 2. The solution of linear systems of

equations is obtained by employing the preconditioned BiCGSTAB method. The performances of the preconditioners are measured by comparing the number of iterations taken by the preconditioned solvers.

This chapter presents results of validations of the methods and speedup from parallel implementation. The key points can be summarized as follows:

- the validation of the VOF method for benchmark problems,translation of a square box and rotation of a solid disk.
- the introduction of a new benchmark problem for the VOF method.
- the validation of N-S code for the single and two-fluid flow problems by simulating the lid driven cavity flow, dam breaking and rising bubble problems.
- the advantage of choosing the diagonal format for sparse matrices, involved in the linear system solution of N-S code.
- the performance measurement of the preconditioners used in the linear solvers.
- the speedup data for parallel linear solvers.

5.2 Validation of the VOF Method

This chapter begins with validation of the VOF method. The most common benchmark problems for this purpose are the translation and rotation of the geometrical shapes as documented in (Hirt and Nichols, 1981; Harvie and Fletcher, 2000, 2001; Rudman, 1997; Nobari et al., 2009; Zalesak, 1979). The main purpose of these is to check the accuracy of the reconstruction and advection schemes applied for the method. One of the simplest but most important problems in this category is the translation of a square box in a computational domain with uniform velocity field.

5.2.1 Translation of a Square Box

In this test, a square box filled with fluid is translated with a uniform constant velocity field in the computational domain. The accuracy of the advection algorithm and the reconstruction method affects the general accuracy of the interface capturing method. This problem poses a significant challenge for the VOF method because it contains a discontinuity in the interface direction at the corners of the square. In order to compare the accuracy of the developed VOF method, we have chosen the parameters as given in (Harvie and Fletcher, 2001; Nobari et al., 2009).

The computational domain is taken as a 2D cartesian region of dimensions $1m \times 1m$, where m stands for meter. The domain is divided into 100 cells in the x direction as well as in the y direction. At location $(0.15m, 0.15m)$, a square block of fluid of the dimensions $0.1m \times 0.1m$ is placed. Then a uniform velocity field $(1, 1)m/s$ (s is second) is applied to move the block. The simulation has been carried out for 0.7 seconds and the position of the box is noted at intervals of 0.1 seconds. Fig.(5.1) shows the position of the fluid block computed using the VOF-Analytic method and compares the same using with other schemes. The graphs of the fluid positions obtained from other schemes have been taken from reference (Nobari et al., 2009). The value of computational time step in the other method has taken 1×10^{-3} seconds while in our case its value is 7×10^{-3} seconds. This higher value of δt has been chosen after verifying the numerical errors for lower values. Therefore it shows that the developed VOF-Analytic method performs better even for higher δt. It can be seen from Fig.(5.1), that in the case of the Hirt-Nicolas algorithm, the calculated VOF contours show significant numerical diffusion in the direction normal to the fluid velocity. The shape of the box has flattened tangentially at end of the calculation with this scheme. In the

case of the DDR-Pukett and DDR-Young schemes, developed by Harvie and Fletcher (Harvie and Fletcher, 2001), the shape of the box has not flattened tangentially but its corners are rounded as the advection progresses. The box shape looks better in the case of the DDAR scheme developed by Nobari and co-workers (Nobari et al., 2009) although there is numerical diffusion in the tangential direction. There is significant improvement in the numerical diffusion as well as in the corners of the box in the case of the VOF-Analytic method employed in this work.

In order to judge the accuracy of the solver, an error function is defined based on the concept that the total amount of the fluid (VOF) must be conserved during numerical computations. This error function can be expressed by as in (Rudman, 1997),

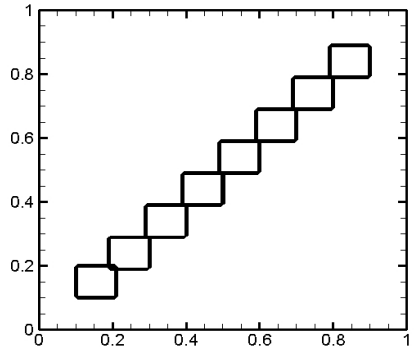
$$error = \frac{\sum_{i,j} |C^n_{i,j} - C^e_{i,j}|}{\sum_{i,j} C^e_{i,j}} \tag{5.1}$$

where $C^n_{i,j}$ is the volume of fraction (VOF) calculated after n time steps and $C^e_{i,j}$ is the exact value of VOF calculated using the exact values of the colour function at that position. Based on this function the errors obtained from different schemes (Nobari et al., 2009; Rudman, 1997) are shown in Table (5.1).

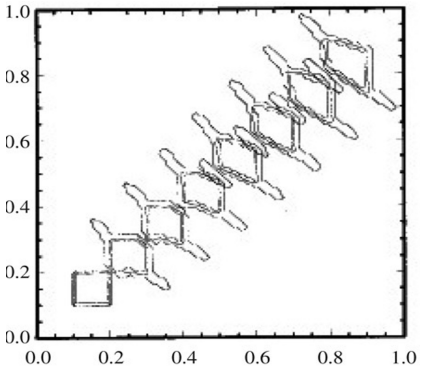
H-N	**DDR/Pukett**	**DDR/Young**	**DDAR**	**VOF-Analytic**
0.539	0.216	0.19	0.126	0.0668

Table 5.1: Errors calculated using equation (5.1) for translation of a box.

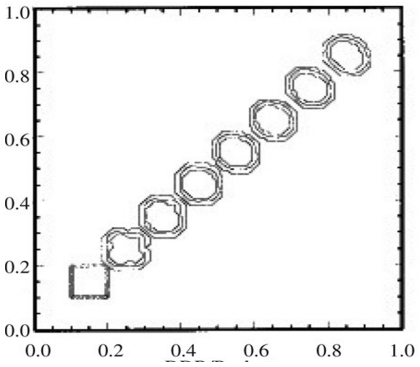
From this table, it is clear that the VOF-Analytic method, employed in this work, is more accurate than the other schemes to conserve the total volume of fluid.



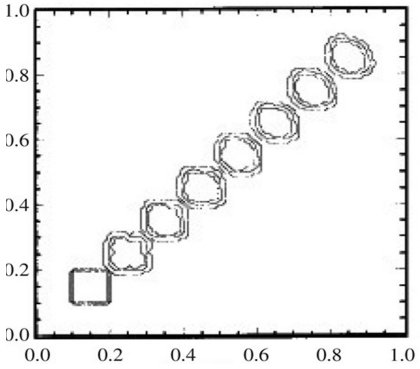
(a) Exact Solution



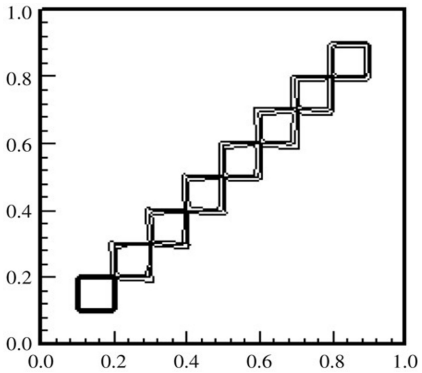
(b) Hirt-Nicholas



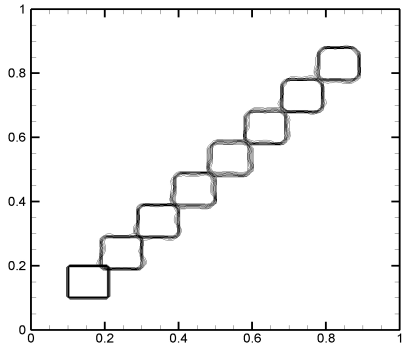
(c) DDR-Pukett



(d) DDR-Young



(e) DDAR



(f) VOF-Analytic

Figure 5.1: Translation of a square box using different algorithms.

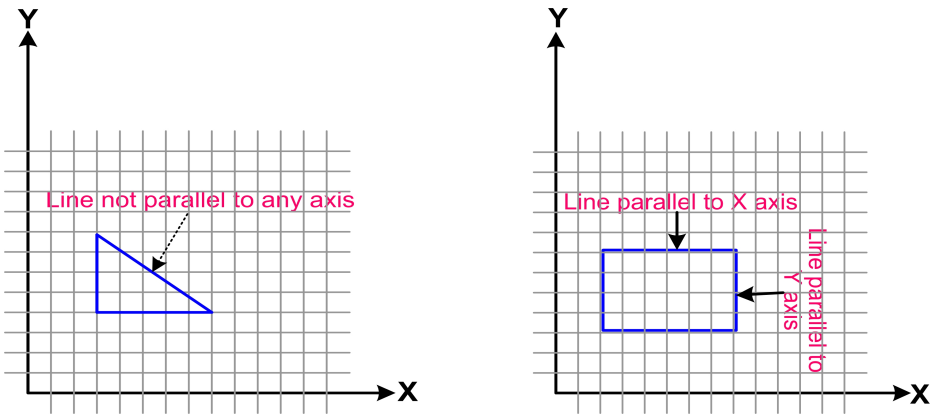
5.2.2 Translation of Triangular Shape

In this work, we have also considered one more example for testing the accuracy of the VOF advection scheme. In this test, a triangular block of fluid is translated using a uniform velocity field similar to the translation of a square box.

Reason for choosing this example

The reconstruction step of the VOF-PLIC method uses a different approximation whether the line which approximates the interface, is parallel to the coordinate axes or not. This situation can be viewed clearly by observing any of the figures in section (3.3). In the example of square box translation, the interface line is parallel to either of the axes. In this example, the volume fraction of the fluid can be calculated as the area of a square. Because in this case no triangle is present, there is no need to calculate the area of the triangle which is quite complex as discussed in section (3.3). But, in the case of a triangular shape, one of the lines is not parallel to any axes and creates triangles whose area are required to calculate the volume of fluid. These two cases are clearly shown in Fig.(5.2). Therefore, this example contains very sharp corners similar to the square box as well as an interface not parallel to either of the axes. Hence it also serves as a good benchmark problem. A similar computational domain to the one in the case of square box translation has been taken for this example. The triangular block of fluid is kept at the same location as for the square box, i.e. at $(0.15\,m, 0.15\,m)$ and translated by applying the velocity field $(1.0, 1.0)$ meter/second toward the right-up corner. The simulation is run for 0.7 seconds and the position of the block is noted at intervals of 0.1 second.

The VOF contours for this example at different time steps are depicted in Fig.(5.3). Exact solutions for those time steps are also presented in this figure. As can be seen



(a) Triangular shape (b) Rectangular shape.

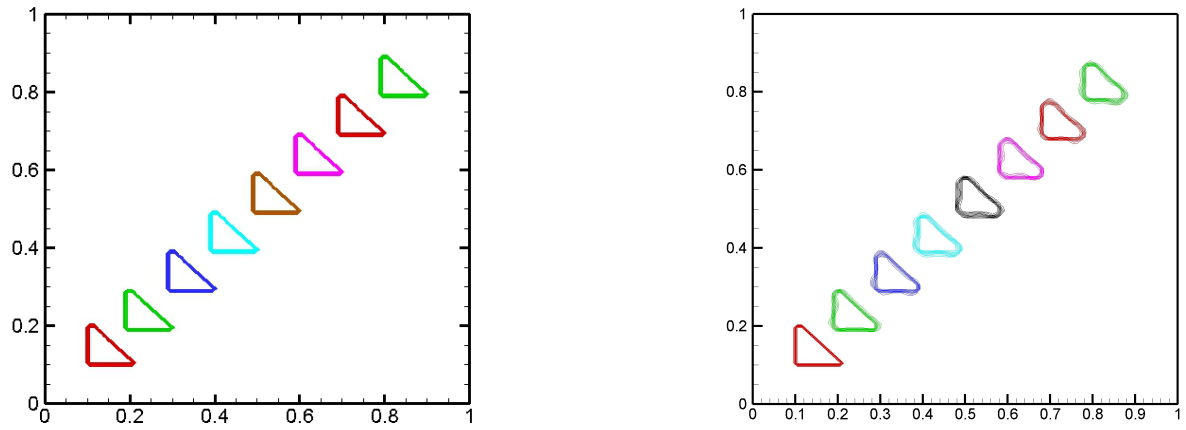
Figure 5.2: Difference between triangular and rectangular shapes for validation of the VOF method.

from this figure, the shape is not flattened and numerical diffusion is also very small. The time step for this simulation was taken as 7×10^{-3} seconds which is the same as in the case of a square box. The error is calculated using the same error function as defined in equation (5.1). The calculated error is compared with the same in the case of square box which is provided Table (5.2).

Shape	Grid	δt	error
Square	100×100	7×10^{-3}	0.0668
Triangular	100×100	7×10^{-3}	0.1467

Table 5.2: Errors calculated using equation (5.1) for translation of a triangular and square box.

The data of this table shows that the error in the case of triangular box is more than the same for the square box. The reason for this big error is the same as was explained above. The error can be reduced by decrementing in the time step and grid refinement as shown in Table(5.3). The data from this table shows that by halving the grid size (and reducing the δt) the error in the case of square box is reduced by about 50%. Similarly, a significant reduction in the numerical error in the case of triangular



(a) Exact Solution (b) Calculated value using VOF-Analytic method.

Figure 5.3: Translation of a triangle block of fluid.

box has been observed.

Shape	Grid	δt	error
Square	200×200	2×10^{-3}	0.0386
Triangular	200×200	2×10^{-3}	0.0922

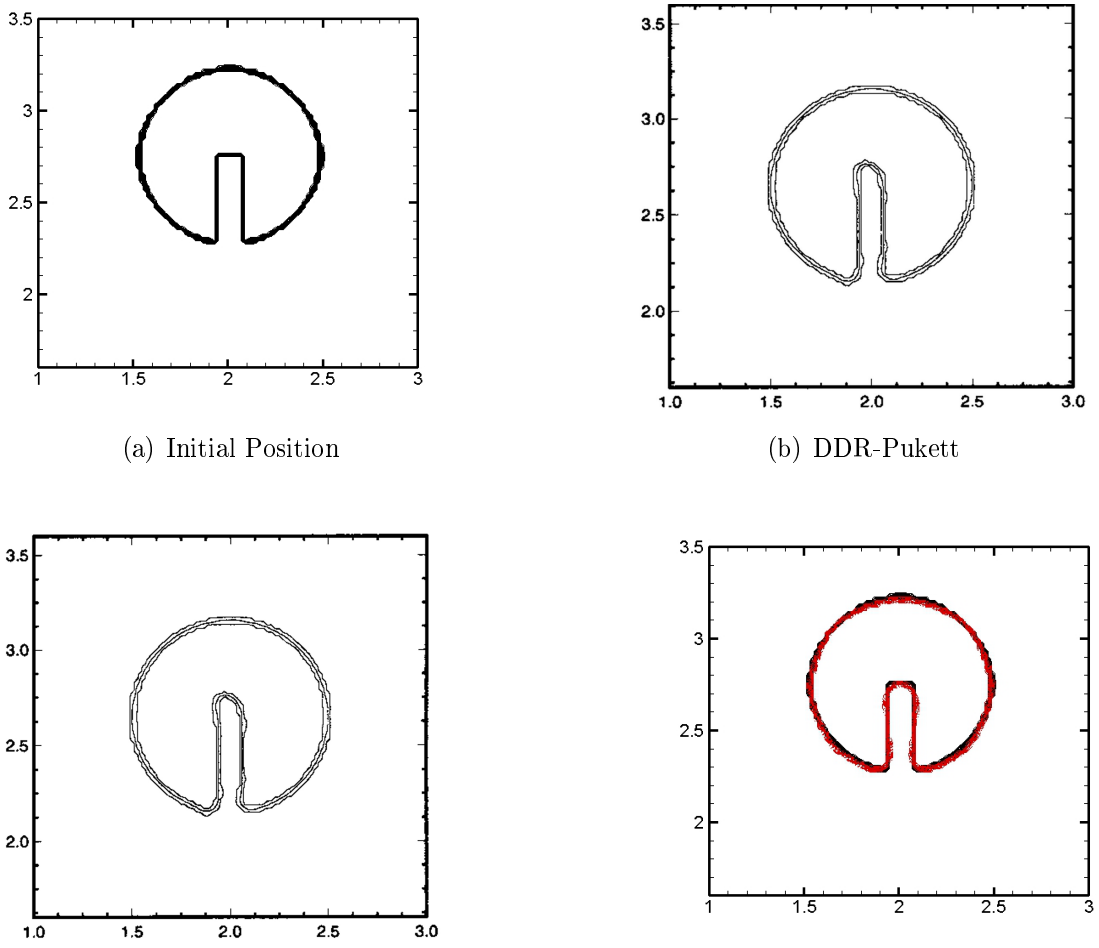
Table 5.3: Errors calculated for translation of a triangular and square box with a refined grid.

5.2.3 Solid Disk Rotation

Another important benchmark problem to test the VOF method is the Zalesak slotted disk rotation (Zalesak, 1979). In this test, a slotted disk is rotated through one complete rotation within the computational domain using a uniform vorticity velocity field. In this case, the advection of flow is rotational and satisfies continuity. The slot is inserted to include sharp interface corners avoiding a purely circular shape which is the simple to model. The accuracy of the solver is assessed by comparing the initial and final position of the disk. For comparison of the VOF-Analytic method with other methods, the same

parameters for the Zalesak disk have been taken as given in references (Rudman, 1997; Harvie and Fletcher, 2001).

The dimensions of the computational domain is $4 \times 4\, m^2$. It is divided into 200×200 equally sized cells and a disk of diameter $1.0\, m$ is placed with centre at $(2.0\, m, 2.75\, m)$. The disk is slotted with a rectangle of width $0.12\, m$. An angular velocity of magnitude 1 revolution/second has been applied and simulation was run for 2425 time steps to complete one revolution.



(a) Initial Position

(b) DDR-Pukett

(c) DDR-Young

(d) VOF-Analytic

Figure 5.4: Zalesak test problem for solid disk rotation: (a) shows the initial position of the disk. Other figures represent position of the disk after one complete revolution.

The chosen time step corresponds to Courant number of 0.25 which is taken from the above references. The errors for different algorithms (calculated by the error function which is the same as in equation (5.1)) are provided in the reference (Rudman, 1997). A comparison of these errors including the error obtained from the present methods is provided in Table (5.4). The data in this table shows that the VOF-Analytic method provides better performance in comparison to any other method except the Young's scheme. Fig (5.4) represents the position of the disk after completing one revolution. From this figure, it is clear that the VOF-Analytic has slightly better accuracy than the DDR-Young and DDR-Pukett algorithms. Comparison of this figure with the figure presented in the above references (Rudman, 1997) shows that the VOF-Analytic method has a similar level of accuracy as that of Young's algorithm (shown in the last column of Table (5.4)). The less error produced by this algorithm may be due to the

H-N	SLIC	FCT-VOF	DDR/Young	DDR/Pukett	VOF-Analytic	Young
9.62×10^{-2}	8.38×10^{-2}	3.29×10^{-2}	1.56×10^{-2}	1.50×10^{-2}	1.41×10^{-2}	1.09×10^{-2}

Table 5.4: Errors (taken from references (Nobari et al., 2009; Rudman, 1997)) calculated using equation (5.1) for rotation of the slotted disk.

method used for the estimation of the normal vector which involves the calculation of the angle the interrface makes with x-axis as explained in reference (Rudman, 1997).

5.3 Validation of the N-S Solver

In section 5.2, a velocity field that satisfies continuity was imposed. In a fluid flow problem, this field must be approximated by the solution of the N-S equations. Solving the PDEs numerically is the kernel of the multiphase flow modelling. As noted in chapter 3 in this study, the N-S equations are solved by employing the FVM. The

numerical schemes used in this section are assessed by implementing on single fluid and multifluid benchmark problems.

5.3.1 Lid Driven Cavity Flow

One of the most studied problems for measuring the accuracy of the N-S solver is the flow in a lid driven cavity as explained in Section (2.4.5). The geometry (or experimental set up) used for this problem is illustrated in Fig.(5.5). In a discussion on

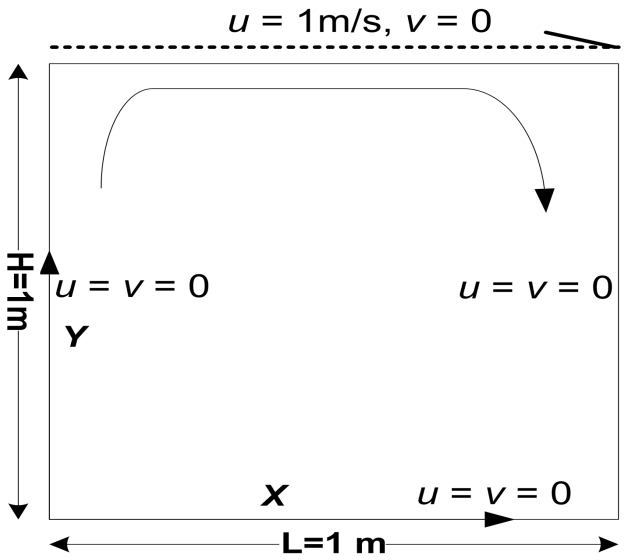


Figure 5.5: Lid driven cavity geometry.

driven cavity flow documented by Erturk (Erturk, 2008), it was stated that at high Reynolds number ($Re = H|u|/\nu$, where ν is the kinematic viscosity and H is characteristic length), there is a chance of spurious solutions. Hence, the numerical solutions need to be validated at these Reynolds numbers and the solution must satisfy the continuity equation. For validating the solver, authors calculate the velocity (horizontal or vertical) profiles along the cavity centrelines (vertical or horizontal).

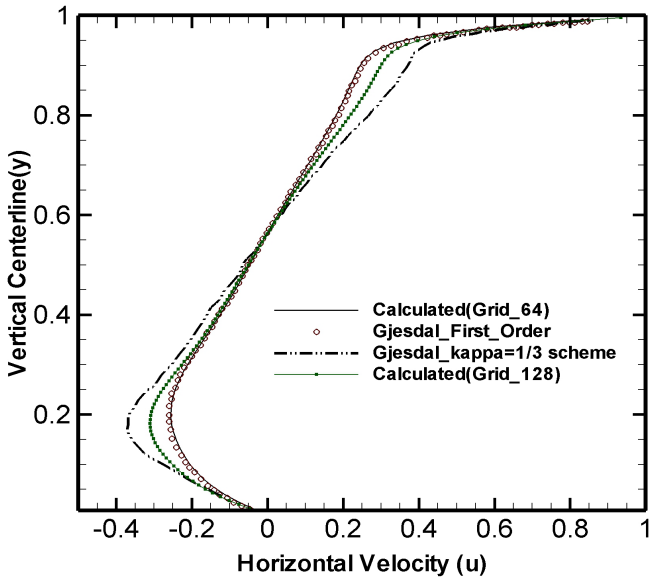
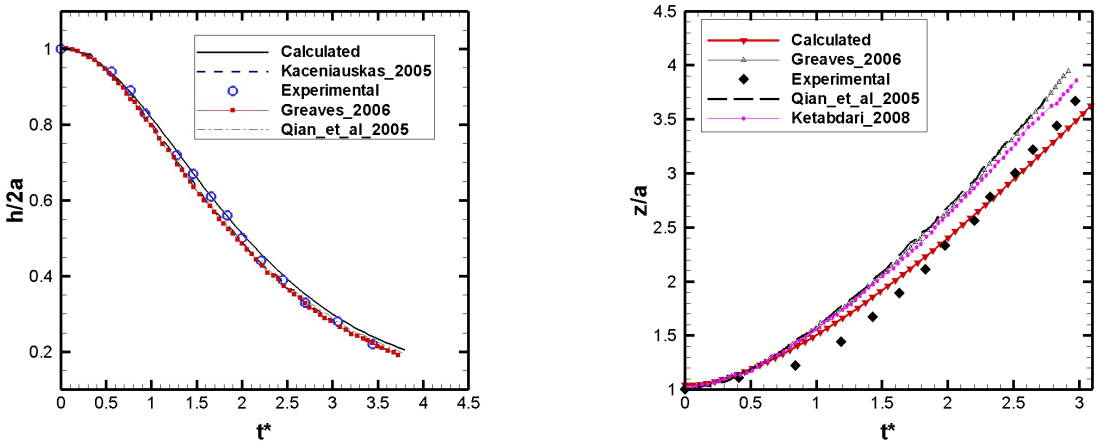


Figure 5.6: u-velocity profile for Re =1000.

In this work, the u-velocity profile has been calculated and compared this with those given in (Gjesdal and Lossius, 1997) which provides the results for a first order scheme for the discretization of the convection terms. Since, in this study, a first order scheme has been employed, I chose this reference for comparison purpose. Fig. (5.6) illustrates the comparison of u- velocity for $Re = 1000$. In (Gjesdal and Lossius, 1997), the grid size (for the first order scheme) was 64×64 which is adopted in this work for comparison. It is clear from this figure that the velocity profile is in good agreement. This figure also represents the velocity profile obtained by a third order scheme marked by "Gjesdal_kappa =1/3". In this work, the u-velocity has been calculated for a more refined grid (128×128) and plotted in this figure. It can been seen that by refining the grid, the velocity profile (using a first order scheme) approaches the profile obtained by the third order scheme.

5.3.2 The Dam Breaking Problem

Another benchmark problem for testing the N-S solvers is the dam breaking problem which simulates collapse of a water column. This test problem is used as a test case for free surfaces to assess the numerical stability of multiphase flow models. It has been studied by many researchers such as (Greaves, 2006; Ketabdari et al., 2008; Qian et al., 2003). Experimental data have been provided by Martin and Moyce (Martin and Moyce, 1952b). In this test problem, a square tank containing a column of water,



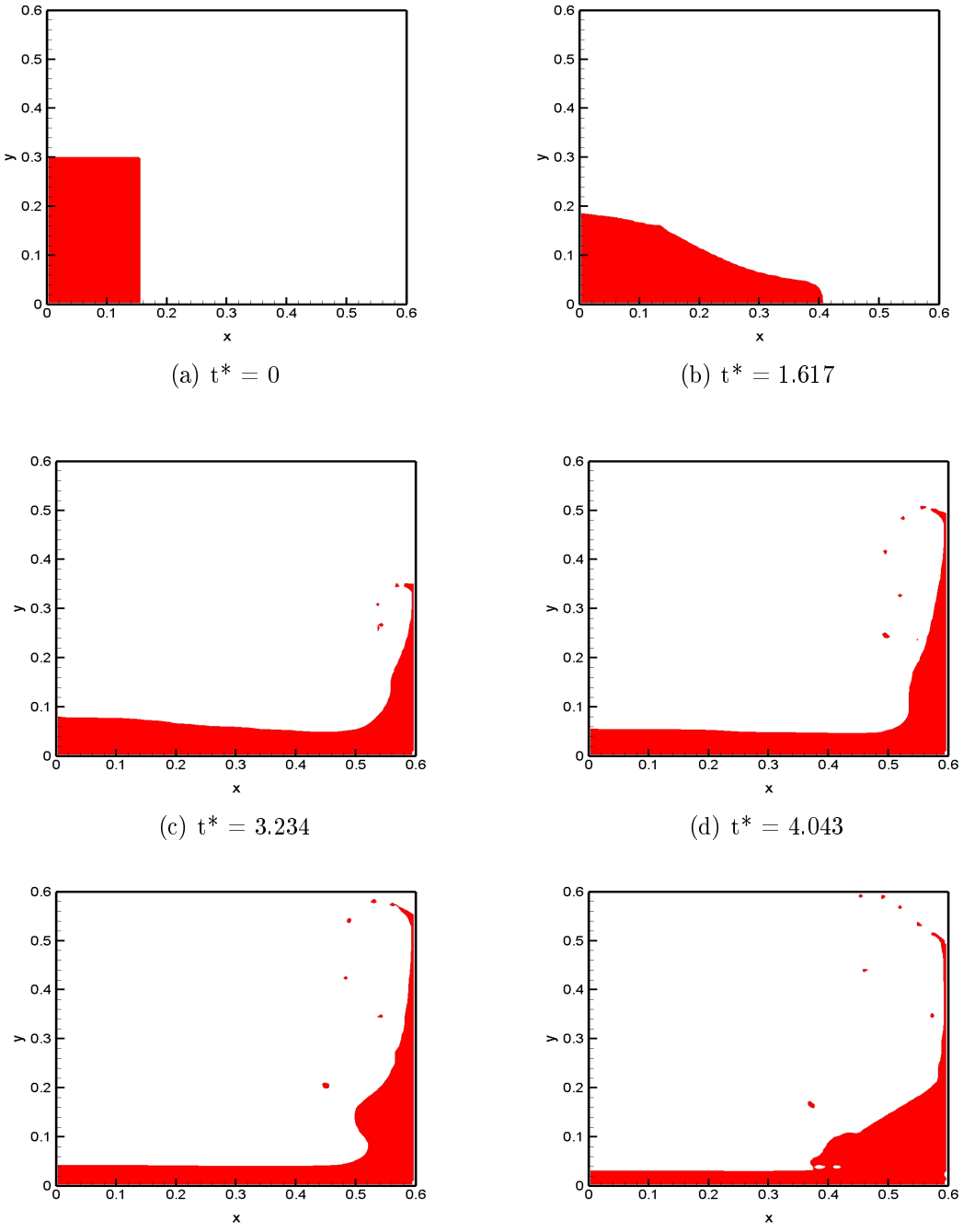
(a) Height of water column versus time

(b) Position of leading edge versus time

Figure 5.7: Height of collapsing water column and position of leading edge versus dimension less time t^*. Where $t^* = t\sqrt{g/a}$ and t is the simulation time.

is held at position $t = 0.0$ second. The motion of the water column, after removing the restraint instantaneously, is simulated. During the simulation, the water elevation at a point in the computational domain (referred as the height of water column) and the distance of the travelling water from the left vertical boundary (referred as position of leading edge) is monitored. Parameters used for this simulation are given in Table (5.5). The dimension of the computational domain was chosen $0.6\,m \times 0.6\,m$ and domain was

divided by 100×100 uniform cells.



(a) t* = 0 (b) t* = 1.617

(c) t* = 3.234 (d) t* = 4.043

(e) t* = 4.85 (f) t* = 5.98

Figure 5.8: Time history of the interface position of collapsing water column

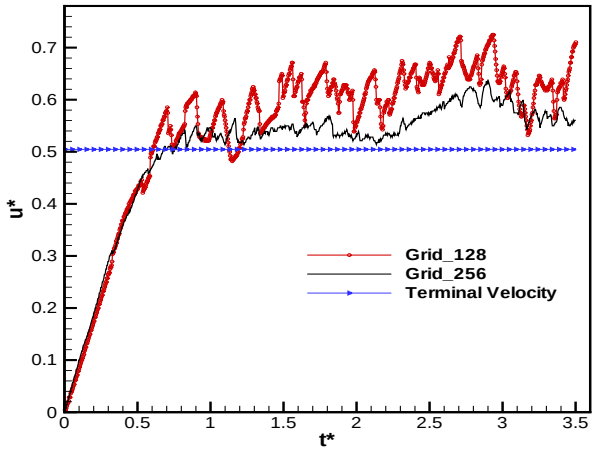
The accuracy of the model can be gauged by observing the height of the collapsing water column and the position of the leading edge of the water front with respect to time. The non-dimensional height of the water column at the left wall with respect to the non-dimensional time is shown in Fig.(5.7(a)). This figure shows that the calculated height is in very good agreement with experimental data as well the numerical data. Fig. (5.7(b)) represents the non-dimensional position of the leading edge against the non-dimensional time which also agrees with the experimental and numerical data. The time histories of the interface position of the collapsing water column are plotted in Fig.(5.8).

Parameter	Value
Height of water column (L)	$0.3\,m$
Width of water column (a)	$0.15\,m$
Viscosity of water (μ_f)	$1 \times 10^{-3}\ kg/ms$
Viscosity of air (μ_g)	$1.7 \times 10^{-5}\ kg/ms$
Density of water (ρ_f)	$1000\ kg/m^3$
Density of air (ρ_g)	$1\ kg/m^3$
Gravity acceleration (g)	$9.81\ m/s$

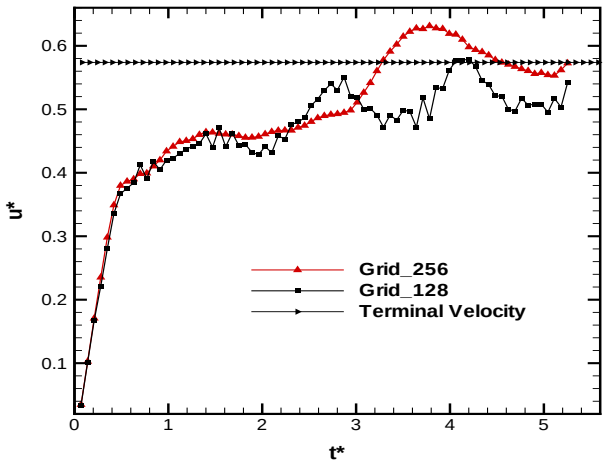
Table 5.5: Parameter values used for the simulation of dam breaking problem.

5.3.3 The Bubble Rising Problem

The computation of the position of a free rising bubble in a viscous liquid also serves as a benchmark problem for multiphase flow simulation. This problem has been used to validate the numerical methods for interfacial flows (Chen et al., 1999; Raymond and Rosant, 2000). The study of the fundamentals of bubble rise has wide applications in the industry such as the rise of steam bubbles in boilers tubes, gas bubbles in oil wells (Chen et al., 1999) and bubbly flow in nuclear power safety (Cerne et al., 1998).



(a) Bubble rising velocity verses time for D=0.008 m.



(b) Bubble rising velocity verses time for D=0.006 m.

Figure 5.9: Bubble rising velocity verses time.

In this problem, the shape of the bubble and its rising velocity are observed. These properties depend on different flow parameters. The parameter values used in this study are given in Table(5.6). As it rises, the bubble shape can deform to ellipsoidal, skirted and toroidal shapes (Clift et al., 1978). These shapes and the rising velocities

of the bubbles can be characterised by some non-dimensional parameters such as (Hua and Lou, 2007; Bhaga and Weber, 1981):

Parameter	Value
Surface tension of water (σ)	$0.073\,kg/s^2$
Diameter of bubble (D)	$0.006 - 0.008\,m$
Viscosity of water (μ_f)	$1 \times 10^{-3}\ kg/ms$
Viscosity of air (μ_g)	$1.7 \times 10^{-5}\ kg/ms$
Density of water (ρ_f)	$1000\ kg/m^3$
Density of air (ρ_g)	$1\ kg/m^3$
Gravity acceleration (g)	$9.81\ m/s^2$

Table 5.6: Parameter values used for the simulation of dam breaking problem.

$$Morton\,Number \quad Mo = \frac{g\mu_f^4}{\rho_f \sigma^3} \tag{5.2a}$$

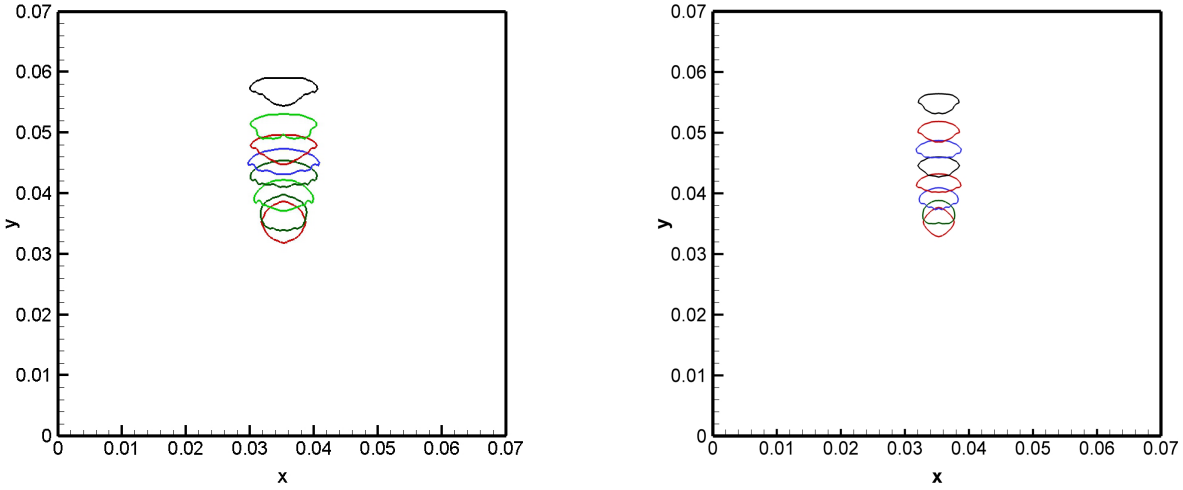
$$Reynolds\,Number \quad Re = \frac{\rho_f D\,u}{\mu_f} \tag{5.2b}$$

$$Eotvos\,(or\,Bond)\,Number \quad Eo = \frac{\rho_f g D^2}{\sigma} \tag{5.2c}$$

where u is the bubble rising velocity. The computational domain is taken as $0.07\,m \times 0.07\,m$. The time step is selected according to grid size. Fig.(5.9) illustrates the air bubble rising velocities for two grid sizes and two bubbles of different diameters. The rising velocities for a bubble of diameter $0.008\,m$ for grid sizes 128×128 and 256×256 are shown in Fig.(5.9(a)). Similar velocities for bubble of diameter $0.006\,m$ are depicted in Fig.(5.9(b)). The value of the time step in this case is taken as $\delta t = 1 \times 10^{-5}$ seconds. In this figure, u^* and t^* are the dimensionless velocity and time respectively which are defined as (Hua and Lou, 2007),

$$u^* = \frac{u}{\sqrt{gD}} \qquad t^* = t\sqrt{\frac{g}{D}} \tag{5.3}$$

It can be seen from Fig.(5.9) that in the case of a coarse grid (i.e. grid size 128×128), there are significant fluctuations in the bubble rising velocity. Similar fluctuations have been reported in the reference (Hua and Lou, 2007).



(a) Grid size = 128×128 and D = 0.008m (b) Grid size = 128×128 and D = 0.006m

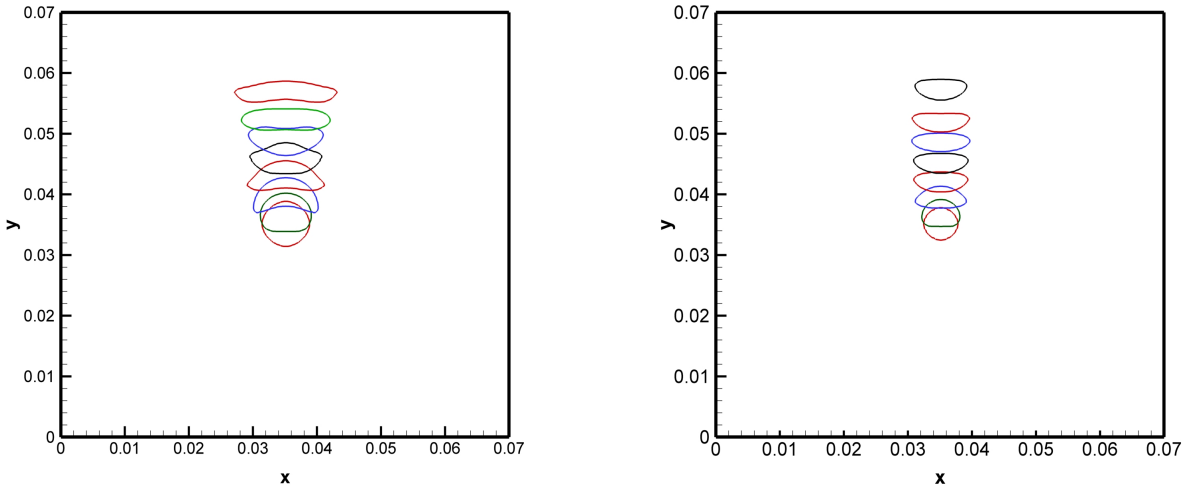
Figure 5.10: Instantaneous shapes of air bubble verses time for grid size 128×128

Positions and instantaneous shapes of air bubbles of diameters $0.008\,m$ and $0.006\,m$ using grid size 128×128 at different time are represented in Fig.(5.10). The similar positions and instantaneous shapes of the air bubble for grid size 256×256 are illustrated in Fig.(5.11). The values of the non-dimensional parameters (as defined in equations 5.2a - 5.2c) for these two air bubbles are provided in Table(5.7). It has been

Bubble Diameter(D)	Eo	Re	Mo
6 mm	4.83	1439	2.6×10^{-11}
8 mm	8.60	2237	2.5×10^{-11}

Table 5.7: Values of non-dimensional parameters.

reported in (Chen et al., 1999) that as the values of Eo and Re increase the shape of



(a) Grid size = 256×256 and $D = 0.008$m (b) Grid size = 256×256 and $D = 0.006$m

Figure 5.11: Instantaneous shapes of air bubble verses time for grid size 256×256

bubble changes to toroidal. Similar results have been obtained here and this fact can be observed in figures 5.10(a) and 5.11(a).

In the previous sections (5.1 to 5.3), the results from the validation of the solvers have been presented. The another part of this research is the adoption of the diagonal format for storing the sparse matrices. The next Section provides the comparison of simulation times taken in the case of the index format and the diagonal format.

5.4 Advantages of Diagonal Storage Format

We have seen in section (4.2), that the diagonal format for storing matrices requires less memory in comparison to any other formats. Another advantage of this format is a reduction in computational time. A simulation for the bubble rising problem using the index format and diagonal format has been carried out. Total simulation time for this

simulation has been noted down and significant differences have been observed. The times taken by simulation for 2000 iterations using grid size 256×256 are provided in Table (5.8). Table data shows that by adopting the diagonal format, the linear solvers

Format	Grid	No. Iterations	Time
Index	256×256	2000	2135 min
Diagonal	256×256	2000	224 min

Table 5.8: Simulation time required for the bubble rising problem using two formats.

take much less time. The time reduction in computing the matrix vector product affects the total simulation time. Similarly, the simulation for the dam breaking problem has also been carried out using the index format and the diagonal format. Time taken by this simulation is shown in Table(5.9).

Format	Grid	No. Iterations	Time
Index	100×100	38120	1743 min
Diagonal	100×100	38120	573 min

Table 5.9: Simulation time required for the bubble rising problem using two formats.

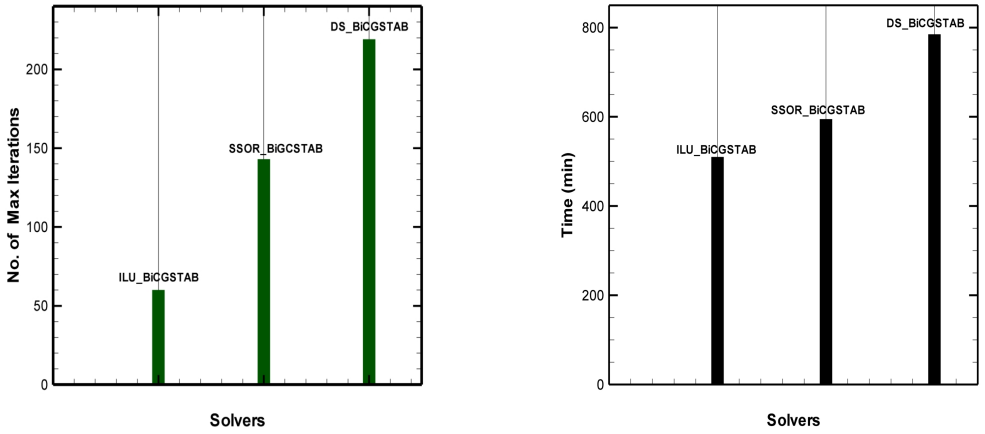
From this table it can be stated that the algorithm developed in the diagonal format is three times faster than that developed in index format for the bubble rising problem. The reduction in the computation time is due the reduced time complexity of the matrix-vector multiplication and less memory requirement which may induce a cache hit (i.e. the required data is available in the CPU cache memory).

5.5 Performance of Preconditioners

As mentioned in chapter 2, the preconditioners accelerate the convergence rate of the Krylov Subspace methods. During simulation, the matrix is being generated at each

time step. Hence the matrix entries changes at each time step which changes the matrix properties. In this research, the Bi-CGSTAB method has been employed because it has been found, in literature, to provide smooth convergence (see Section 2.4.4.2) for non-symmetric matrices. In this section, the effects of the preconditioners applied to the Bi-CGSTAB method implemented on different problem are discussed. Lets start with the effect on the dam breaking problems.

5.5.1 Effect of Different Preconditioners on The Dam Breaking Problem



(a) Overall maximum iterations taken by different solvers for the dam breaking simulation using grid 100×100.

(b) Time taken by different solvers for the dam breaking simulation using grid 100×100.

Figure 5.12: Time taken by different solvers for the dam breaking simulation using grid 100×100.

This problem is ideal to test the performance of preconditioners. As seen before, in this case, the water column collapses and moves in the positive x direction . Since the position of the water, i.e., the values of the volume of fluid in the grid cells changes, the matrix entries generated at each time steps also changes. The changes in the value

of entries may change the condition number of the matrix and subsequently the number of iterations taken by the solver. The number of iterations needed by the solver using different preconditioners and their variations have been recorded. Fig.(5.12(a)) represents the maximum number of iterations taken by the preconditioned BiCGSTAB solvers to run the simulation for the dam breaking problem. This figure shows that the ILU preconditioner takes fewer iterations in comparison to SSOR and DS (Diagonal Scaling). The time taken by the solver using these preconditioners are shown in Fig.(5.12(b)). This figure demonstrates that although the ILU preconditioner takes significantly fewer iterations in comparison to SSOR, the difference in the time taken by ILU BiCGSTAB and SSOR BiCGSTAB is not significant. This result indicates that the computational cost of ILU is more than SSOR. Furthermore, the DS preconditioner takes more computational time in comparison to both ILU and SSOR preconditioners.

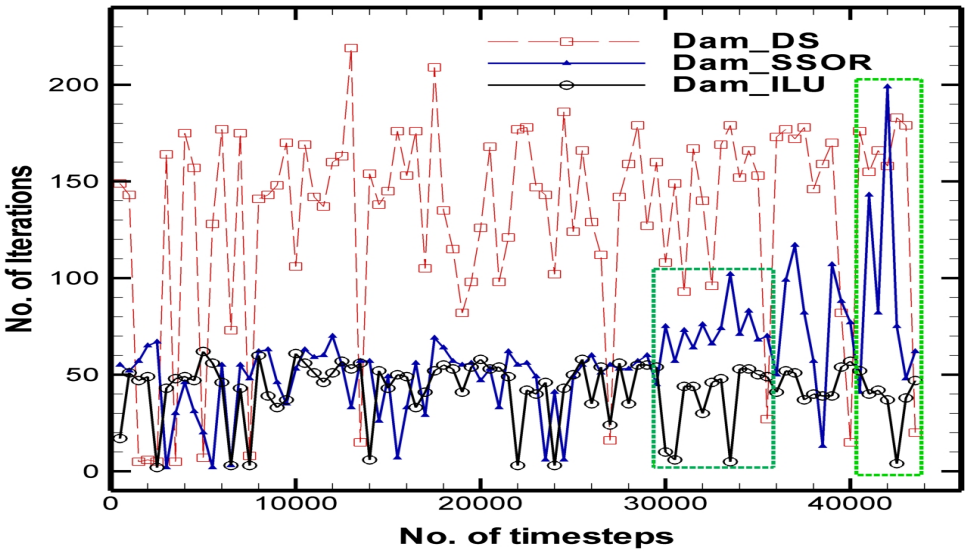
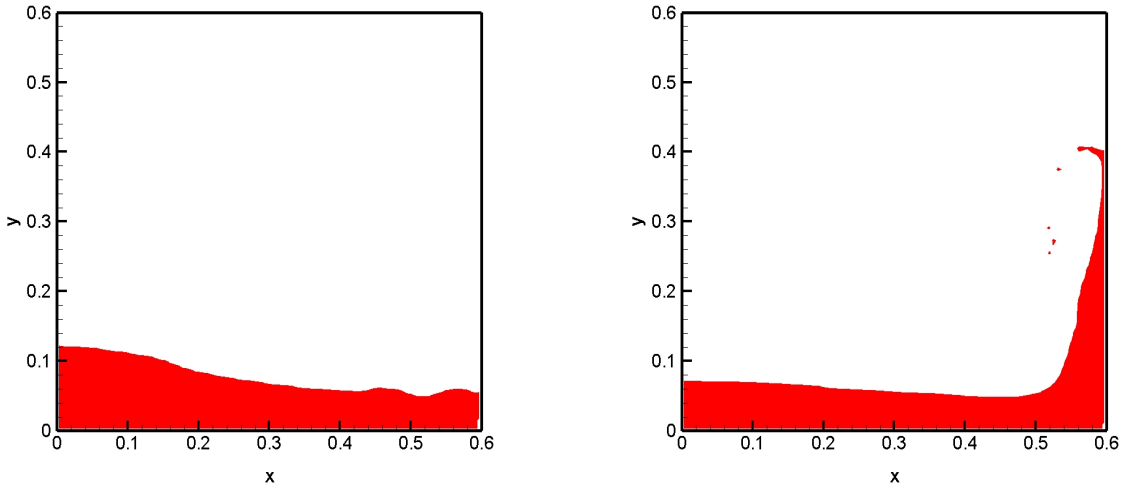


Figure 5.13: Variation in the number of iterations taken by preconditioned BiCGSTAB solvers for Dam breaking problem using grid 100×100.

The variation of the iterations at different time steps during this simulation is depicted

in Fig.(5.13). It can been seen from Fig.(5.13) that there are variations in the number of iterations at two locations which are covered by rectangles. These two locations are related to time $t = 0.3$ seconds and $t = 0.43$ seconds. At the first location, the water has just touched the right wall and started climbing up. The second location is where water has started moving away from the wall. A graphical representation of these two situations is shown in Fig. (5.14).



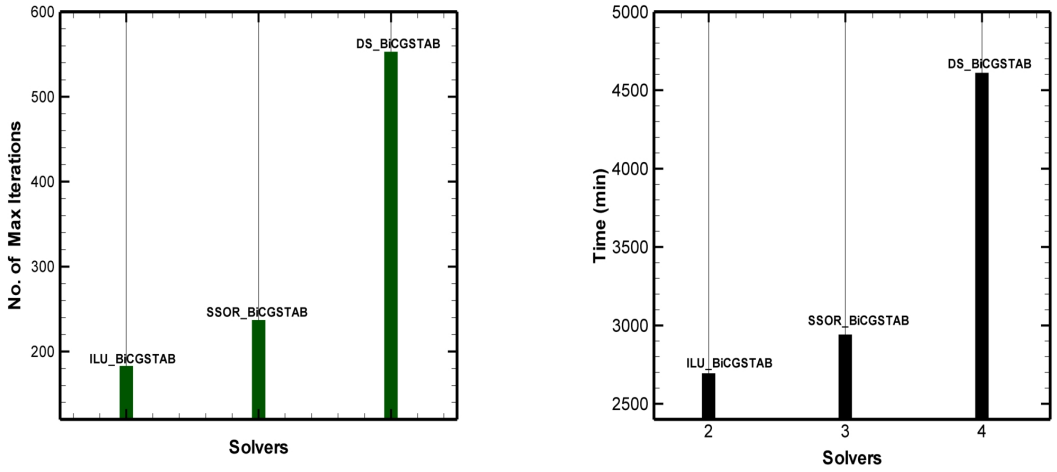
(a) Water touches the right wall after 0.3 seconds (t* = 2.426)

(b) Water starts moving away from the right wall after 0.43 seconds (t* = 3.477)

Figure 5.14: Locations of water at which variation in the number of iteration occurs for grid 128×128.

An investigation of the performance of preconditioners has also been done for a fine grid, namely, for grid size 256×256. The variations in iterations taken by different preconditioners for this case are represented in Fig.(5.16).

The maximum numbers of iterations and the total simulation time taken by the preconditioners are shown in Fig.(5.15). Similar to the case of grid size 100×100, in this case also the ILU takes fewer iterations to SSOR. But here, the time taken by ILU is less than that for the SSOR. Further there is a big difference in the times taken by



(a) Overall maximum iterations taken by different solvers for the dam breaking simulation using grid 256×256.

(b) Time taken by different solvers for the dam breaking simulation using grid 256×256.

Figure 5.15: Time taken by different solvers for the dam breaking simulation using grid 256.

DS and ILU preconditioners which was not observed in the previous case.

In Fig.(5.16), the jumps (or variations) in the number of iterations are marked by symbols L1, L2 and L3. The positions of the front of the water column corresponding to these three locations are shown in Fig.(5.17). The first location L1 is at an early stage when the water column starts collapsing. When the height of the water column decreases by about half then there is a jump in the number of iterations graph and this location has been marked by L2. The third location, L3, is when water touches the right hand wall. This analysis indicates that the grid size also affects the performance of the preconditioners. One more point that can be observed in the figure is that, at a certain time step, the number of iterations taken by DS preconditioners suddenly jumps and generates a spike. But at the same time step, there is no such jumps are observed by ILU and SSOR preconditioners.

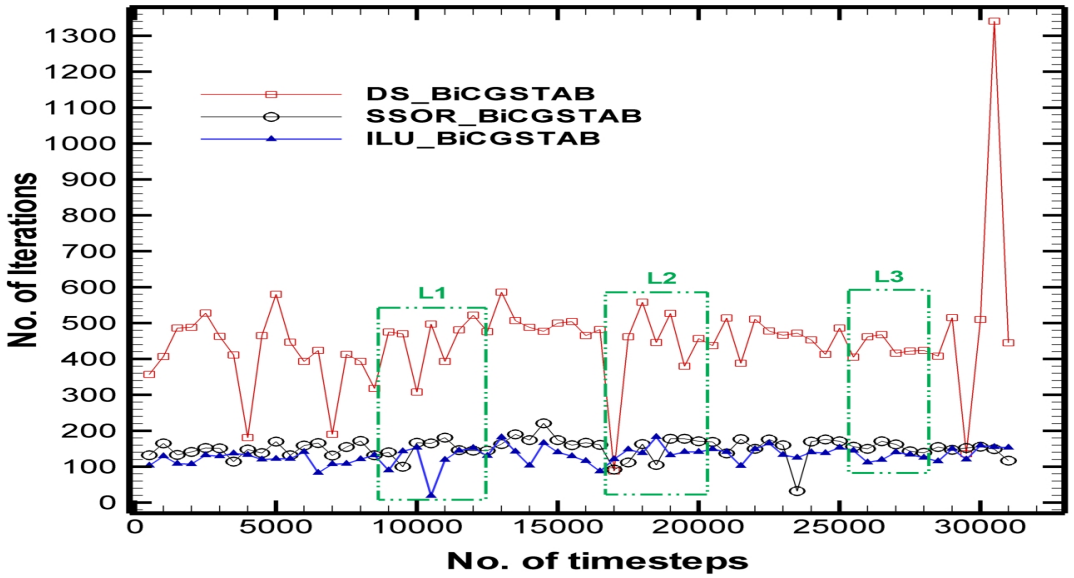


Figure 5.16: Variation in the number of iterations taken by preconditioned BiCGSTAB solvers for Dam breaking problem using grid 256×256.

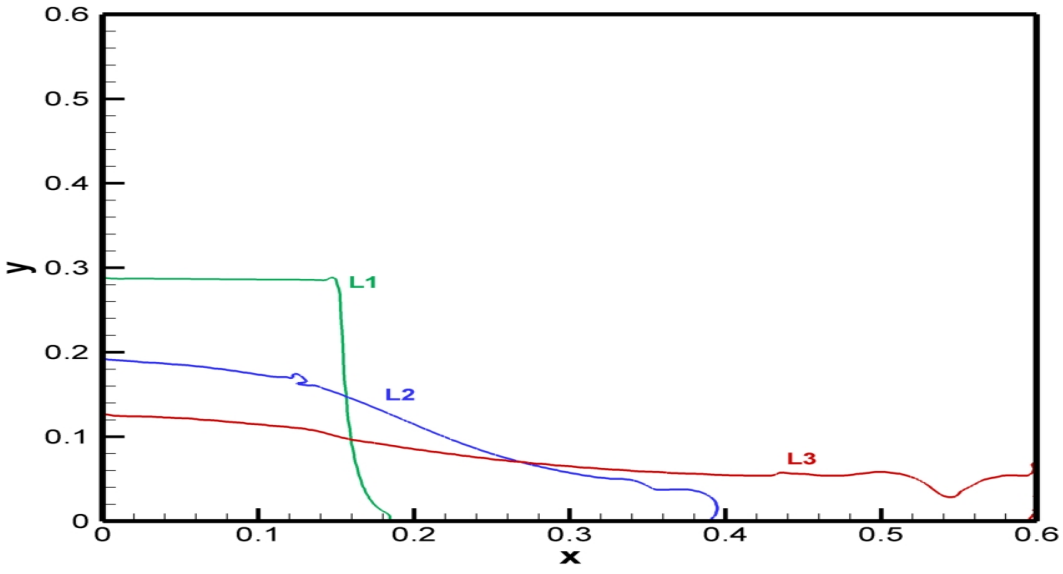


Figure 5.17: The positions of front of water column corresponding to Fig(5.16).

5.5.2 Effect of Different Preconditioners on The Rising Bubble Problem

Next, the performance of the preconditioners on the simulation of a rising air bubble has been investigated.

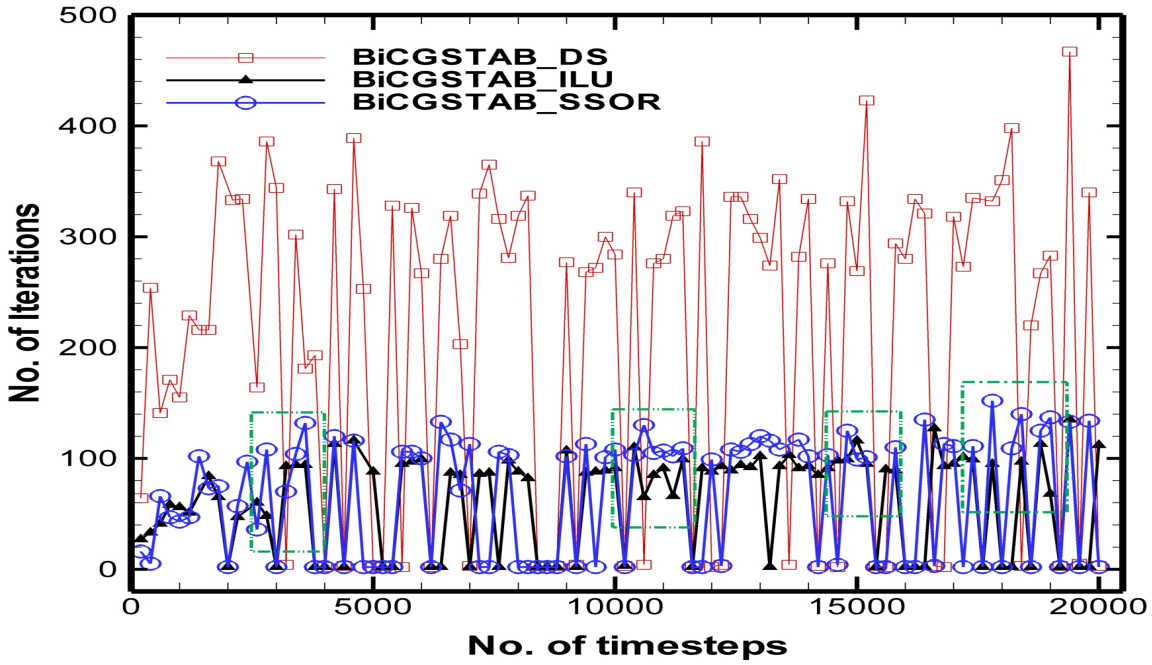


Figure 5.18: Variation in the number of iterations taken by preconditioned BiCGSTAB solvers for rising bubble problem using grid 128×128.

As shown before, in this problem, the air bubble moves in a viscous liquid in the positive y direction and during the simulation the bubble shape changes. Due to the changes in the shape of bubble, the surface tension near to boundary changes which creates a pressure difference in those area. This pressure difference changes the amount of VOF in the neighbouring grid cells near to the boundary of its surface which in turn makes sharp variation in the matrix coefficients on the pressure system. The changes

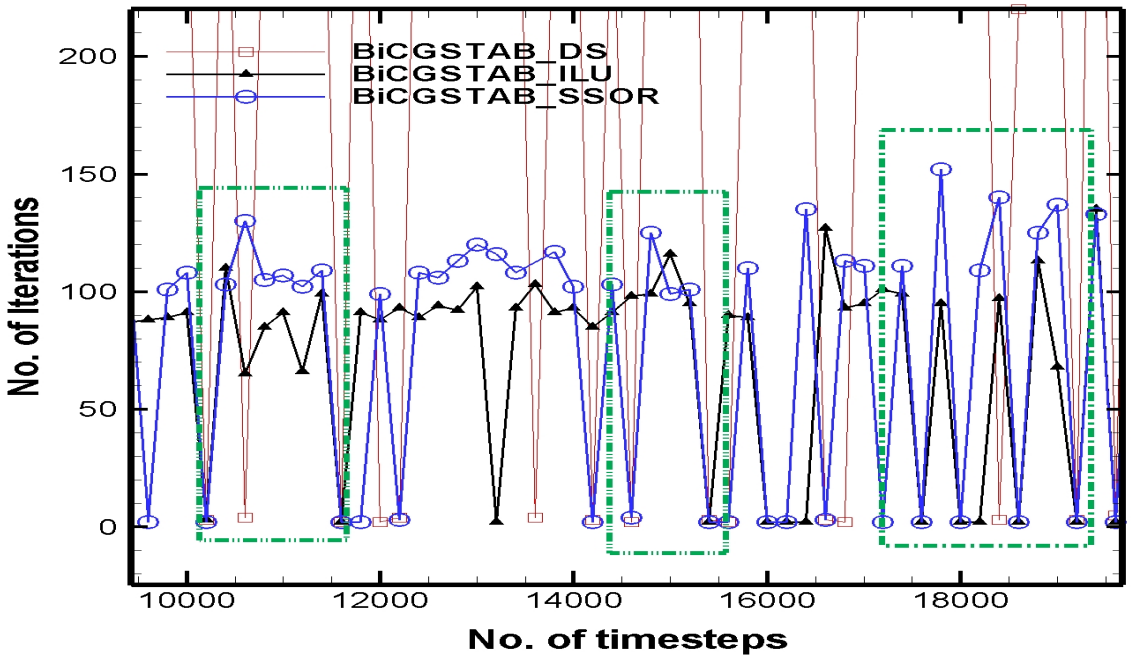


Figure 5.19: The zoomed view of Fig.(5.18)

in the matrix coefficient alter its condition number, consequently, solver takes less or more iterations for convergence at different time steps. The iterations at certain time steps and the maximum iterations during the entire simulation have been recorded for a bubble of diameter 0.006m using grid 128×128. The time history of the number of iterations taken by the preconditioned BiCGSTAB method is depicted in Fig.(5.18).

As can be seen in this figure, there are many locations where the variations in the number of iterations occurs. Four such locations are highlighted by green rectangles. The corresponding positions and shapes of the air bubble are shown in Fig.(5.20). It is clear from this figure that at all these locations, there are some changes in the air bubble shape, which in turn make changes in the values of matrix entries. These changes in the values of matrix entries alter its condition numbers. Fig.(5.19) shows the zoomed

view of Fig.(5.18). In this figure, three locations, where jumps in the iteration have observed, have been highlighted. The different shapes of the bubble corresponding to the first rectangle of Fig.(5.19) have been illustrated in Fig.(5.21). This figure shows that there are many sharp changes in the bubble shape which is due to the variation in the surface tension. This variation creates pressure difference at these locations responsible for the sharp change in the matrix coefficients.

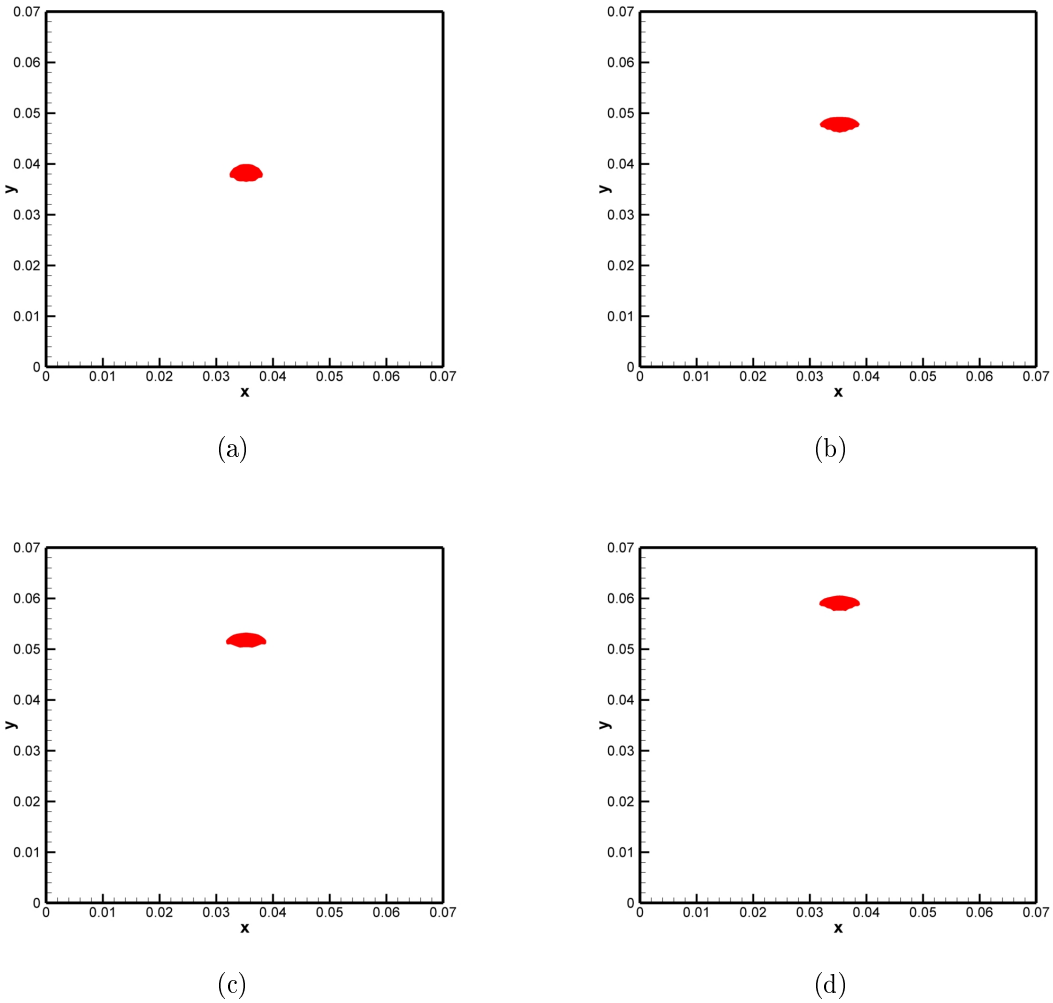


Figure 5.20: Locations and instantaneous shapes of the air bubble rising in the viscous liquid correspond to the areas surrounded by green rectangles in Fig. (5.18) .

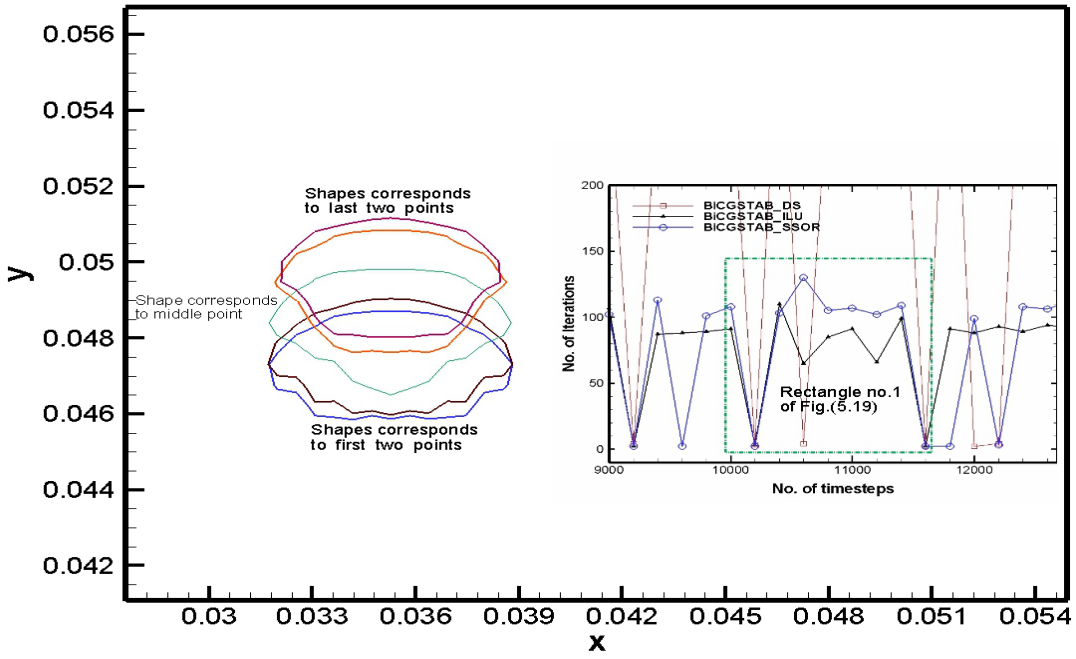
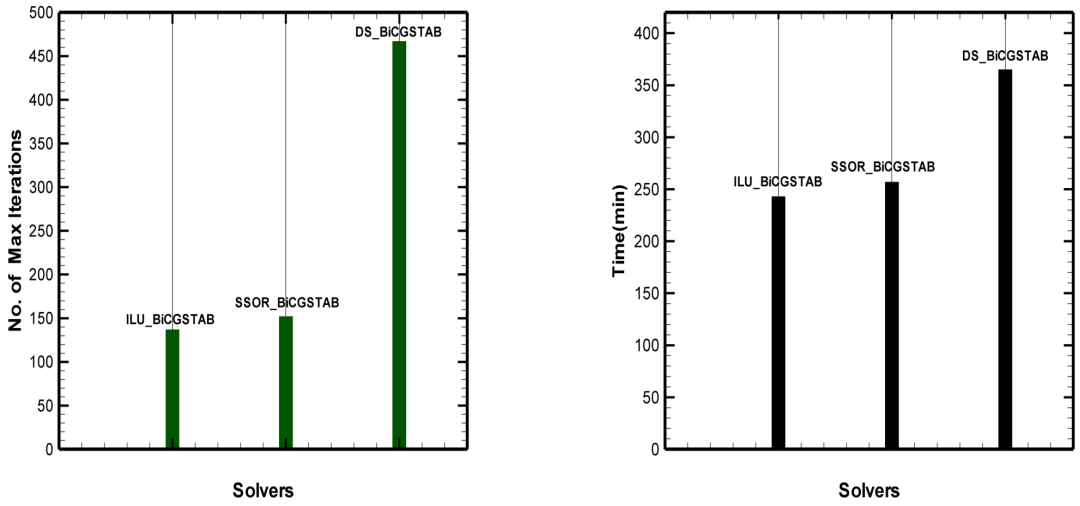


Figure 5.21: Locations and instantaneous shapes of the air bubble rising correspond to the first rectangles in Fig. (5.19)

The maximum number of iterations taken by the simulation for different preconditioners are represented in Fig.(5.22(a)). The times taken by the BiCGSTAB solver with these preconditioners have also been recorded and are shown in Fig.(5.22(b)). It is clear from Fig.(5.22) that for this problem, both the ILU and SSOR preconditioners perform similarly. The difference in the number of iterations and times taken by ILU_BiCGSTAB and SSOR_BiCGSTAB is small.

In order to observe the effect of preconditioners on large size matrices, the simulation for a rising bubble has been run for grid size 256×256. The variations in the iterations at certain time steps are shown in Fig.(5.23). This figure shows that there are mainly three locations where the changes in the numbers of iterations occurs. These locations



(a) Overall maximum iterations taken by different solvers for the rising bubble simulation.

(b) Time taken by different solvers for the rising air bubble simulation.

Figure 5.22: Time taken by different solvers for the rising air bubble simulation using grid 128×128.

are marked by L1, L2 and L3 in the figure. At location L1, it is observed that the SSOR preconditioner takes too many iterations for a particular time step (step number 3800) and generates a spike there. The instantaneous shapes of the bubbles at these locations are illustrated in the Fig.(5.24).

Since at these locations, the shapes of the bubble change, the matrix entries change there also and hence the solver takes a different number of iterations (fewer or more) for convergence. The maximum number of iterations taken by these preconditioners with the BiCGSTAB method is shown in (5.25). In this case the performance of the ILU preconditioner is found to be best in comparison with the other two preconditioners. The total time for the entire simulation taken by the ILU preconditioner is less than the same for both SSOR and DS preconditioners. Furthermore, there is a significant difference between times taken by the ILU and the SSOR preconditioners

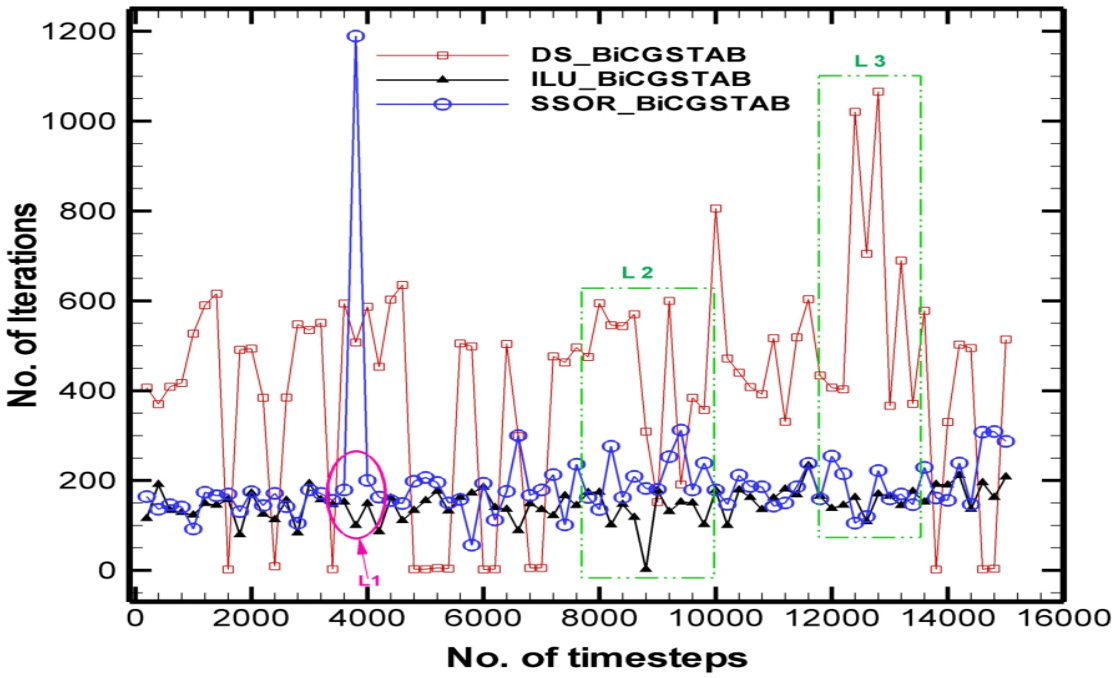


Figure 5.23: Variation in the number of iterations taken by preconditioned BiCGSTAB solvers for rising bubble problem using grid 256×256.

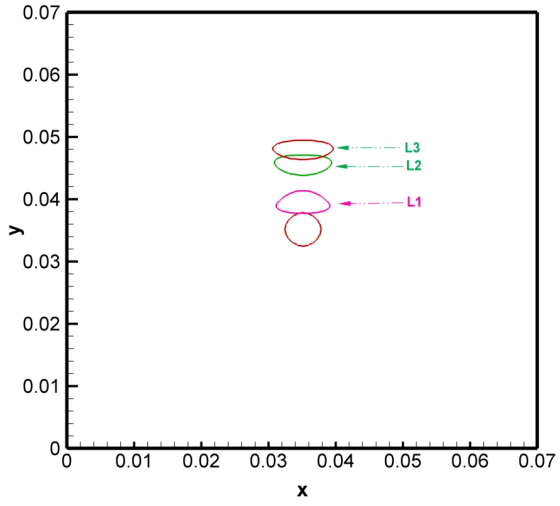
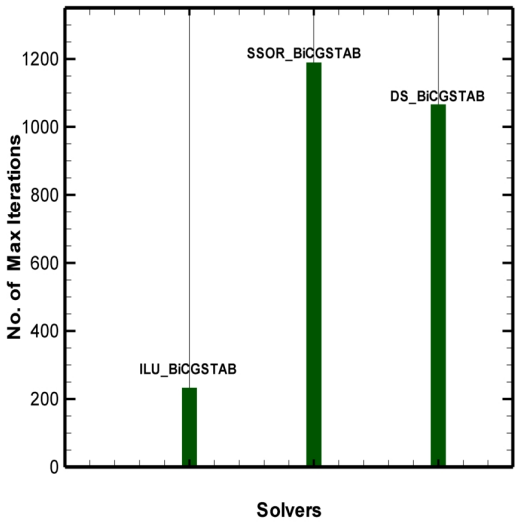
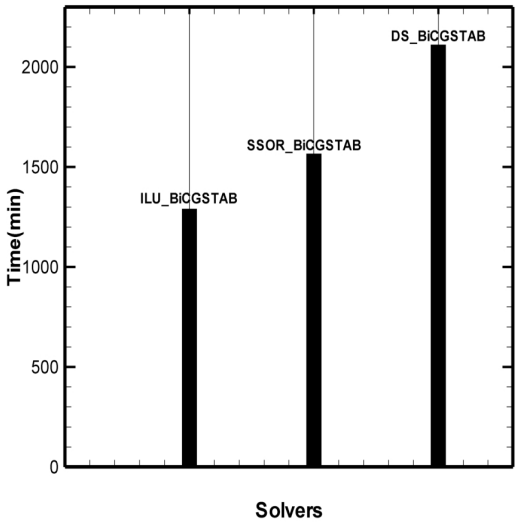


Figure 5.24: Instantaneous shapes of air bubble at locations corresponding to Fig.(5.23).



(a) Overall maximum iterations taken by different solvers for the rising bubble simulation .



(b) Time taken by different solvers for the rising air bubble simulation.

Figure 5.25: Time taken by different solvers for the rising air bubble simulation using grid 256×256.

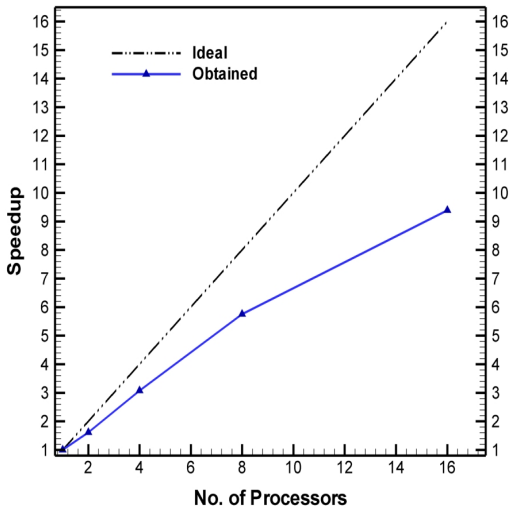
5.6 Results from Parallel Algorithms

The parallel preconditioned BiCGSTAB method developed in this work has been applied to the dam breaking and the bubble rising problems. In order to measure the performance of the parallel algorithms, the speedup factors have been calculated using equation (2.4). The computation time taken by the master processor to solve the linear system of equations generated by the pressure system (as indicated by step no. 5 in Fig.(4.9)) has been recorded. From Table (4.2), it has been noticed that solving the pressure equations requires significantly more time than solving the **U** or **V** velocity equations. Hence the time required for the pressure equations has been recorded to calculate speedup.

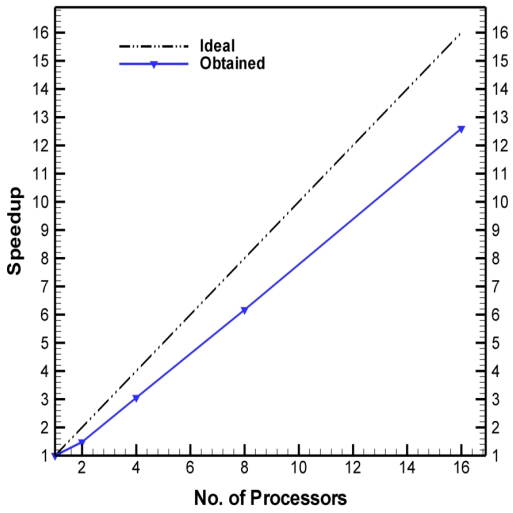
5.6.1 Parallellization of the Preconditioner Algorithms

The speedup obtained from the preconditioners has been calculated separately using the computational and communication times. Fig.(5.26) illustrates and compares the speedup factors for three preconditioners. The generation of the preconditioned system with the ILU preconditioner requires communication between the processors as noted in algorithms (4.6) and (4.7). In the case of SSOR and DS preconditioners, no such communication is required for generating the preconditioned system.

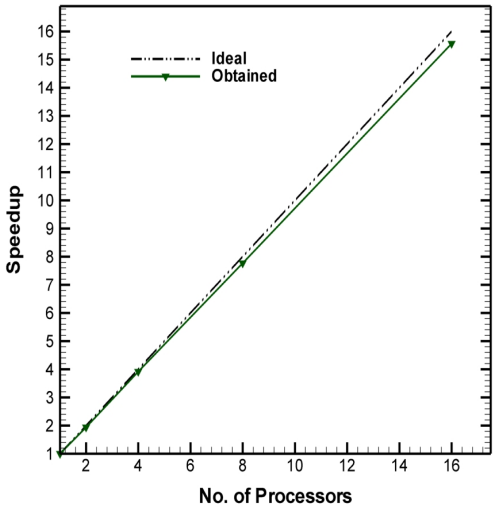
The ILU and SSOR preconditioners decompose the original system of equations into two systems as discussed in the section (4.6.3). These systems are solved by the backward and forward substitutions algorithms (Saad, 1996). Parallel implementation of the backward and forward algorithms requires communication between processors as observed in Algorithm (4.8). Thus, the parallel implementation of the ILU preconditioner needs more communication than the same for SSOR.



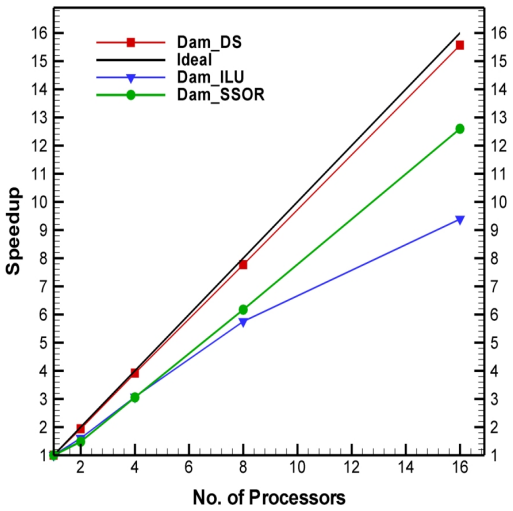
(a) Speedup for ILU preconditioner



(b) Speedup for SSOR preconditioner



(c) Speedup for DS preconditioner



(d) Comparison of speedup factors for preconditioners

Figure 5.26: Speedup factors for the preconditioners calculated using the computation time plus communication time required for the dam breaking problem.

The DS preconditioner requires no communication for parallel implementation. It

can be seen from Fig.(5.26) that the speedup for ILU is less that that for SSOR. This is due to the fact that there is more communication in the ILU.

The speedup obtained from three preconditioners applied to the bubble rising problem are calculated in the same way as for the dam breaking problem. A comparison of the these speedups is depicted in Fig.(5.27). This figure indicates that in the case of bubble rising simulation, the performances of the ILU and SSOR preconditioners are almost the same. Further, the speedup generated by the DS preconditioner is better than for its ILU and SSOR counterparts because of no communication involved in its parallel version.

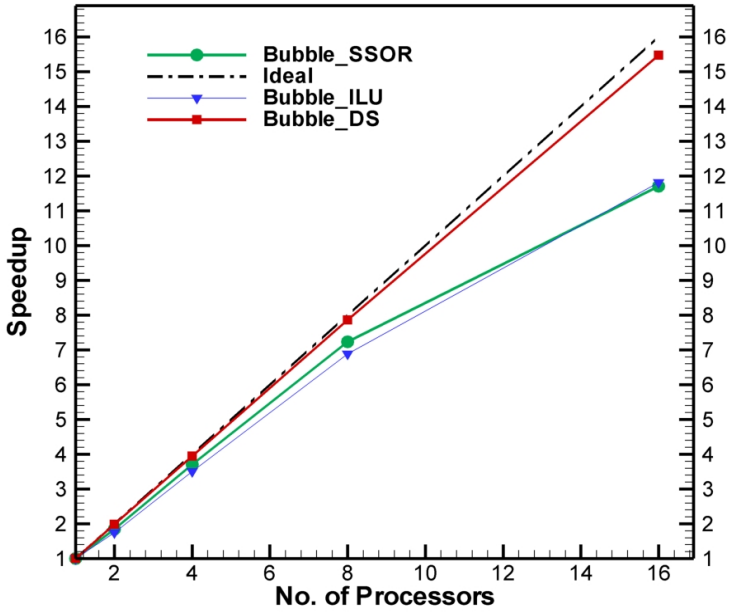


Figure 5.27: The speedup factor calculated for the preconditioned BiCGSTAB method employed to bubble rising problem.

5.6.2 Parallel Preconditioned BiCGSTAB

The speedups obtained from the parallel implementation of the BiCGSTAB method with the three preconditioners employed for the dam breaking problem are depicted in Fig.(5.28). It was observed here that the wall clock time remains the same when running the simulation on more than 2 processors. This is due to the communication overhead required for the parallel BiCGSTAB algorithm. In algorithm (4.1), there are two matrix-vector products, four inner products and preconditioners are called twice. All these steps increase communication overhead. Thus, as the number of processors increases, the significant amount of communication increases. These speedups are calculated using the computational time taken by the method. As shown in Fig.(5.28), the speedup for the ILU_BiCGSTAB and the SSOR_BiCGSTAB are almost the same. But the speedup for the DS_BiCGSTAB is less than both of these.

Fig.(5.29) shows the speedup obtained from the parallel preconditioned BiCGSTAB method applied to the bubble rising problem. In this case the SSOR (in conjunction with the BiCGSTAB method) provides better speedup than the DS and ILU preconditioners. The performance of the DS preconditioner is again poor as in the case of the dam breaking problem.

5.6.3 Multiple-core Vs. Single Core Processors

In this research, the parallel algorithms have been implemented on a Linux cluster. This cluster has 56 nodes, each having 2 quad core processors. Thus it is assumed that each node can furnish the computational power equivalent to 8 processors. In order to examine the power of a node, the parallel algorithms have been implemented on one node with many (maximum 8) processors to avail of the facility of multiple-core architectures

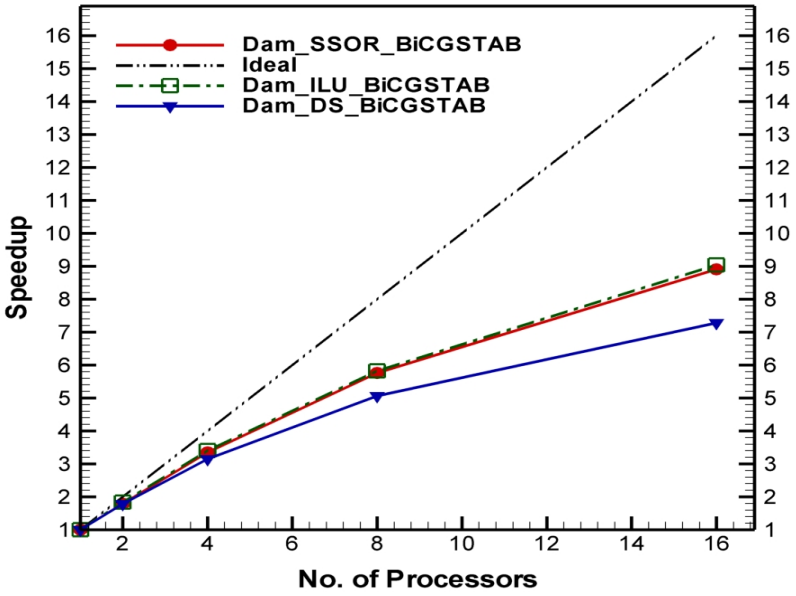


Figure 5.28: The speedup factor calculated for the preconditioned BiCGSTAB method employed to dam breaking problem.

and on n nodes using one processor on each node. In both cases, the times have been recorded and compared. The computational and communication times required using one node (with many processors) and many nodes for the ILU_BiCGSTAB method applied to the dam breaking problem are shown in Fig.(5.30).

In this figure, the legend "Time 1_node" indicates the time taken by the ILU_BiCGSTAB method when run on one node using more than one processor (i.e. using multiple-core). While the legend "Time n_node" means the same time when the program is run on n nodes (2,4 or 8) using one processor per node (i.e. using single core). It can be observed here that the times taken by 2 processors in both cases are almost the same except a little difference in the communication times as shown in Fig.(5.30(b)). From Fig(5.30(a)), one can see that the computation time, required for the ILU_BiCGSTAB,

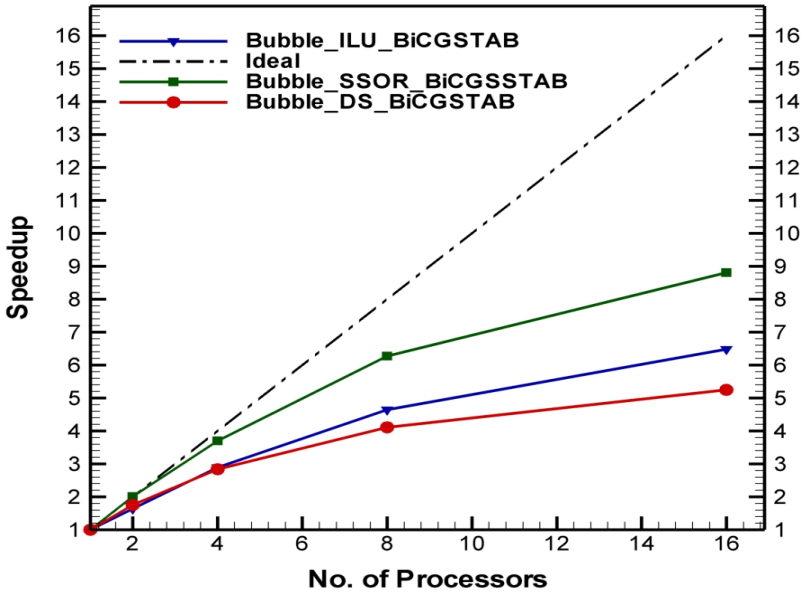
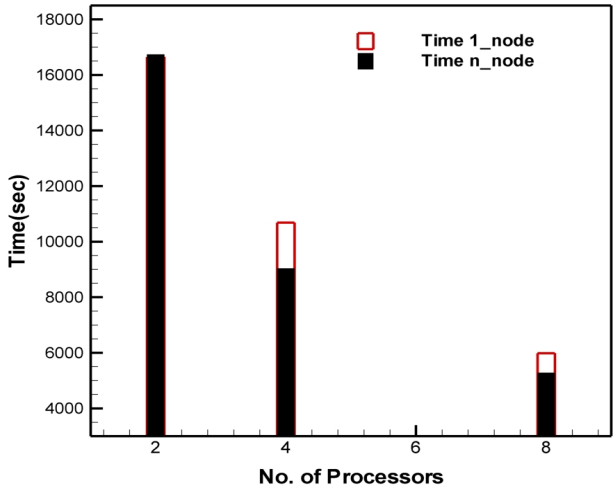
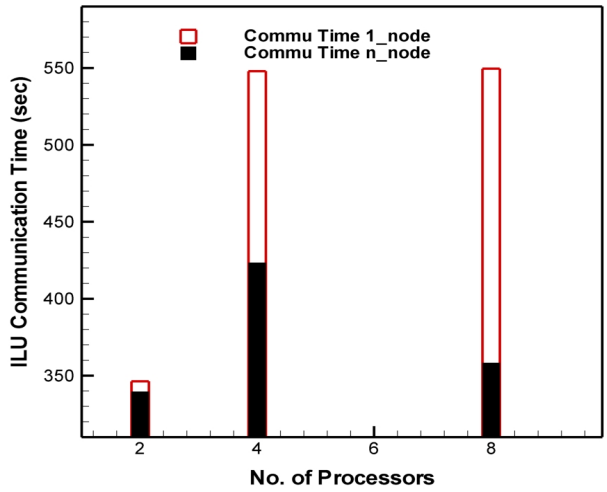


Figure 5.29: The speedup factor calculated for the preconditioned BiCGSTAB method employed to bubble rising problem.

when the program is run using multiple-core (viz., 4 and 8 processors) is more than the single core. Similarly, the communication times required for the ILU preconditioner using multiple-cores is more than that for a single core. Moreover, in this case, there is no reduction in the time is observed when the program is run on 4 processors and 8 processors. The SSOR_BiCGSTAB algorithm has also been implemented on one node (using many processors) and many nodes. The computation time taken by the method and communication time taken by the SSOR preconditioner are depicted in Fig. (5.31). In this case also, the times taken by 4 and 8 processors, when the program is run on one node, are more than the times taken by the same processors on many nodes. A difference in the communication time can be observed from Fig. (5.31(b)), when the program is run using 4 and 8 processors on the same node.

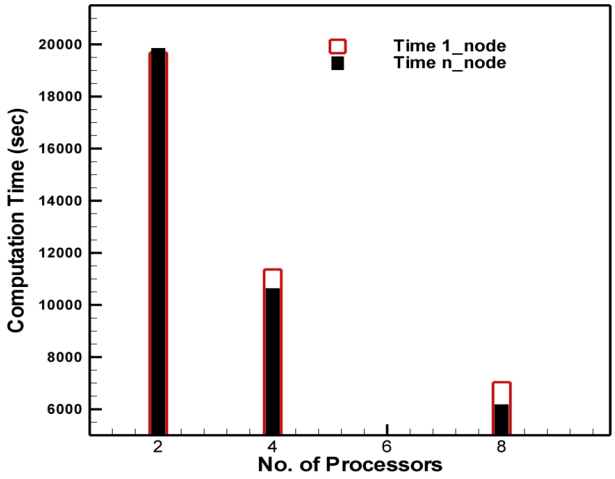


(a) Computation times for ILU_BiCGSTAB on 1 node (using multi-core) and many nodes.

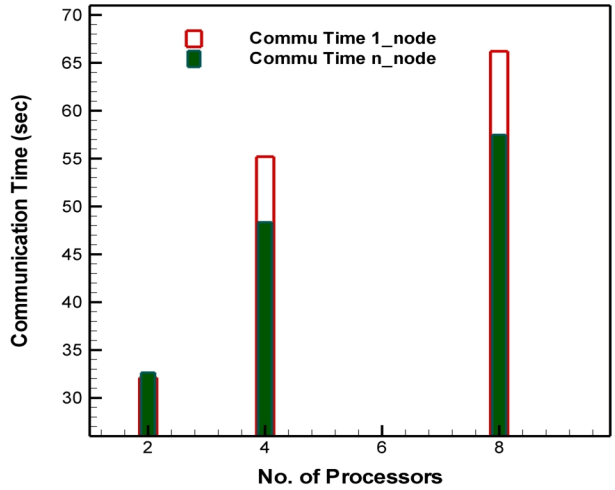


(b) Communication times for ILU on 1 node (using multi-core) and many nodes.

Figure 5.30: Comparison of times for ILU using multi-core and single core processors



(a) Computation times for SSOR_BiCGSTAB on 1 node (using multi-core) and many nodes



(b) Communication times for SSOR on 1 node (using multi-core) and many nodes.

Figure 5.31: Comparison of times for SSOR using multi-core and single core processors

5.7 Chapter Conclusions

This chapter focuses on the results obtained from the methods and algorithms developed in chapter 3 and chapter 4. The VOF method developed using the analytic relation has been validated for two benchmark problems, namely, the translation of a square box and the rotation of a solid disk. A new benchmark problem is presented here. The main findings from this validation are as follows:

(1) The VOF-analytic method provides the best performance for the case of the translation of a square box,

(2) For the solid disk rotations test, this method provides a smaller error compared to the literature except for Young's method (Rudman, 1997).

(3) The translation of a triangular box provides a new benchmark test.

The VOF code was integrated with the N-S code and validated for the following benchmark problems: the lid driven cavity flow, the breaking of a dam and the rising of an air bubble in a viscous liquid. The main results from this validation can be summarised as follows;

(a) The data for the u-velocity for the lid driven cavity flow agree well with the literature data available for a first order scheme.

(b) The results obtained from the dam breaking simulation are in very good agreement with experimental data as well as numerical data from the literature.

(c) The rising bubble simulation has been carried out for grid sizes 128×128 and 256×256 and the terminal velocities and instantaneous bubble shapes are represented.

The advantages of using the diagonal storage format have been discussed by comparing the simulation times taken in this format and using the index format. The performances of the preconditioners have been measured by comparing the number of iterations taken by them. The performance measurements have been done for the bubble rising and dam breaking problems. The key observation from this comparison are;

- for the dam breaking problem, the variations in the number of iterations to converge have been found when the water touches the right hand wall,
- the ILU preconditioner takes fewer iterations in comparison to SSOR and DS when applied with the BiCGSTAB method.
- the size of grid affects the performance of the preconditioners for the dam breaking problem,
- the number of iterations and the time taken by both the ILU and the SSOR preconditioner are almost the same for the bubble rising problem. However, these two factors have differences when grid sizes increase.

The parallel algorithms developed for the BiCGSTAB method, the ILU and the SSOR preconditioners have been applied to the dam breaking and bubble rising problems. The computation times taken by the master processor, for solving the pressure system of equations, are recorded. The times taken by different numbers of processors are compared and the speedup data are generated. A comparison of the computation and communication times taken by using single core and multi-core processors has been carried out.

Chapter 6

Conclusions and Future Research

The computer simulation of problems from engineering and science applications requires efficient numerical schemes and robust algorithms to reduce the computational time as well as memory requirements. One of the most important parts in the simulation of the fluid flows is that of solving linear systems of equations. Another critical part is the modelling of the multifluid flows which requires an efficient method to capture the interface between two fluids. In this research, both of these issues have been dealt with by developing algorithms for the solvers and implementing a modified method for the interface capturing.

6.1 Mulifluid Flows

In chapter 3, a modified version of the VOF method has been documented. The reconstruction step of this method was devised using an analytic formula. The details of the derivation of this formula have been provided in this work. Further, the relation of this analytic formula with the geometric formula has been established. In the second

step of the VOF method, viz., the transportation step, the mathematical derivation of the Lagrange advection scheme is explained. The whole transportation algorithm has been summarized in a flow chart; also the geometrical representations of the flux calculations have been illustrated. This chapter provides a short description of the domain discretization and the equation discretization approach adopted in this research. The domain discretization has been carried out using the staggered grid arrangement whose concept has been explained diagrammatically. For equation discretization, the SIMPLE algorithm has been implemented and its description has been provided in the form of a flow chart.

6.1.1 Main Conclusions from Multifluid Methods

The developed VOF_Analyic code has been validated quantitatively to measure its accuracy. The results obtained were found to be in very good agreement with the experimental as well as numerical data available in previous literature. The inverse relation of the analytic formula facilitates accurate calculation of the interface position and simplifies the extension of the reconstruction step into 3D. Hence it is concluded here that the developed VOF method performs better than other versions of this method, and it can be extended to 3D modelling problems more easily. The Lagrange advection scheme, using the alternate spatial direction for time integration, has been found to be in almost perfect agreement with the developed VOF_Analytic method for the movement of the interface without wrinkling or smearing.

Code for the VOF_Analytic method has been integrated with the N-S code and applied to the simulation of two-phase flow problems, namely, dam breaking and the rising of an air bubble in a viscous liquid. Additionally, this code has been implemented

for the lid driven cavity flow problem. The results obtained from the dam breaking problem were matched very well with the experimental as well numerical data from previous literature. Some small spots have been observed in the graphical representation of this problem. This is due to numerical diffusion which may be caused due to the method used for normal estimation – the nine point finite difference scheme– or the first order upwind difference scheme utilized here. The results obtained from the simulation of the lid driven cavity flow were found to be in good agreement with the literature data for the first order scheme. For the simulation of the air bubble rising in a viscous liquid, the instantaneous shape of the bubble and the terminal velocity have been documented.

6.2 Algorithms for Linear Solvers

Chapter 4 is devoted to the optimization of algorithms for the linear solvers and preconditioners. One of the issues in this topic is the memory requirement for storing the sparse matrices. An investigation of the storage requirement for different sparse matrix format has been carried out. Short descriptions of stationary and non-stationary iterative solvers have been provided. An algorithm for sparse matrix-vector product in the diagonal format has been developed. The need for preconditioners applied to the Krylov Subspace methods is discussed and the algorithm for the ILUT preconditioner in the dense format is explained. A novel algorithm for the ILUT in the diagonal format has been developed. The computational complexities of this algorithm in the diagonal format and dense formats have been analyzed.

The times required for the different parts of the simulation, for rising bubble problems, were recorded and the need for the parallelization of algorithms has been detailed.

The parallelism involved in the Krylov Subspace methods has been highlighted. Much focus has been devoted to the data distribution and load balancing schemes. The parallel algorithms for the BiCGSTAB method, the ILUT preconditioer and the forward substitution in the diagonal format have been developed and documented. Integration of the N-S code and the developed parallel algorithms has been explained with the help of a flow chart.

6.2.1 Key Observations and Conclusions from the Solution to the Linear Solvers

It has been found that the diagonal format for storing the structured matrices brings down the memory requirement in comparison to other sparse matrix formats applicable to unstructured matrices. Furthermore, a significant reduction in the time taken by the BiCGSTAB method developed in the diagonal format has been observed in comparison to the same for the index format. Therefore, it is concluded here that the diagonal format brings down both the memory requirement as well as the computational time of the algorithms developed in this format. The most computationally expensive task in the BiCGSTAB method is the matrix-vector multiplication. Hence a large reduction in the overall computational time can be obtained from developing sparse matrix-vector multiplication in the diagonal format. The computational complexity of the ILUT preconditioer has been reduced from $\mathcal{O}(n^3 - n^2)$ to $\mathcal{O}(n)$.

It has been noted that, in general, the ILUT preconditioner requires fewer iterations when applied with the BiCGSTAB method, for multiphase fluid flow problems, in comparison to the SSOR and the DS preconditioners. However, in the case of rising bubble simulation for grid size 128×128, the number of iterations taken by the ILUT

was found to be almost the same as that of taken by SSOR. In the case of the simulation of the bubble rising problem, the SSOR preconditioner generates a spike at a certain time step.

Although the number of iterations taken by the ILUT preconditioner is smaller in most cases (e.g., the dam breaking problem with grid size 100×100) the times taken by the ILU_BiCGSTAB and SSOR_BiCGSTAB methods are almost same. For large size matrices (such as in the case of grid size 256×256), some difference in this time has been observed indicating that the computational cost of the ILUT preconditioner is higher than that of the SSOR.

The effect of the preconditioners, on the dam breaking and rising bubble problems, has been monitored by investigating the variations in the iterations taken. Such variations within a certain range of time steps have been observed and their relation with the physical situation during the simulation have been established. It has been found that a large variation in the number of iterations occurs when the shape of the bubble or the position of the leading water front (in the case of dam breaking) changes in a certain range of time steps. These variations take place due to a change in the condition number of the matrix generated at those time steps. Therefore, it may be concluded here that changes in the position of the water front or in the bubble shape brings about changes in the matrix condition number.

Speedup data from the parallel preconditioned BiCGSTAB method has been generated. The parallel algorithms have been implemented for the dam breaking and rising bubble problems. The speedup generated from the ILUT and SSOR preconditioners developed in the diagonal storage format have shown good scaling, which concludes that the parallel algorithms developed using the new load balancing scheme for ILUT,

in the diagonal format, reduced the overall communication overhead. A reasonable speedup has been obtained from the parallel BiCGSTAB method using the computational time taken by the master node. However, the communication overhead increases with an increase in the number of processors. It has also been observed here that using more than one processors in single node of multicore architecture performed poorly perhaps because of sharing of the same cache and data transfer bus.

6.3 Overall Conclusion and Further Steps

The VOF_Analytic method was found to be more accurate in capturing the sharp interface between two fluids than other methods available in literature. The simulation in this research has been carried out for 2D problems; however, in order to investigate real life problems from science and engineering applications, simulation in 3D is required and in such simulations, the computational load would increase substantially. The memory requirements and computational costs of the VOF_Analytic method would also increase correspondingly. Therefore, a parallel version of the VOF_Analytic method should be developed further to deal with these issues.

There are many commercial software packages available for the simulation of multiphase fluid flows. The most widely used packages in industry are FLUENT (Ansys, 2006) and Fine/Marine from NUMECA (Numeca, 2008). Such softwares use an algebraic approach for the reconstruction of the interface. Since the geometric approach with analytic relation has been found to be more accurate, the developed C++ code for the VOF_Analytic method can be integrated into these software packages to facilitate the interface capturing step. It has been observed from the literature (Ghia et al., 1982) that the multigrid techniques are more appropriate to solve the linear system

of equations obtained from multiphase flow problems. The matrices generated for the multigrid techniques and for the 3D equivalent of these problems for these problems, using FVM, have generally more than five diagonals (seven or nine). Therefore, the development of a new algorithm for the linear solvers and the preconditioners in the diagonal format for these matrices would be another step for future research.

The parallel algorithms developed for the ILU and the SSOR preconditioners require less communication overhead as can be seen from the speedup factors (calculated using computation and communication times).

The numerical simulation of the multiphase fluid flows can be affected by the application of the appropriate preconditioners for the Krylov Subspace methods. It was found that the ILU preconditioner accelerates the convergence rate by more than the SSOR does. However, the computational cost of the ILU is more than that of the SSOR and in some cases the computational time taken by the preconditioned BiCGSTAB method using these two preconditioners was found to be the same.

The final conclusion is regarding the multiple-core architecture; it has been observed that performance of the parallel algorithms (which require many communication steps) using single core is better than that of using multiple-cores on the same node. Some possible solutions for the communication overhead have been noted here, and discussed in the next subsection. These possible ways are projected as one of the future research directions of this work.

6.3.1 Possible Solutions for Communication Bottleneck

It has been observed that in the BiCGSTAB method, the communication of the data is a big hindrance in achieving speedup. Furthermore, increments in the computational

times have been observed when programs were run on one node using several processors on the multiple-core architecture. One of the possible solutions for these issues is the utilization of a new generation multiple-core processor having dynamic scalability such as the Nehalem processor from Intel (Nehalem, 2009).

The other possible solution for these problems could be to use the General Purpose Graphics Processing Unit (GPGPU) (Brandvik and Pullan, 2008; Dongarra et al., 2008; Suda et al., 2009). In such a system the graphics card is used for general purpose computing in a similar way to the CPU. The GPU consists of one or more multiprocessors (MP) each of which has Stream Processors (SPs), register memory, shared memory and constant cache (Suda et al., 2009). Thus each MP can be viewed as a multicore CPU with extra memories. Furthermore, it can be linked up with the CPU using the special interface such as PCI express to transfer the data from GPU memory to CPU memory. This data transfer is carried out by writing code explicitly using special programming languages provided by GPU vendors.

Some researchers have explored the power of GPU for matrix operations, flow solvers and other applications. Dongarra and co-workers (Dongarra et al., 2008) implemented an algorithm for matrix operations such as LU decomposition and Cholesky factorization on the GPU and compare the computational and communication times. Implementation of Euler solvers has been carried out by Brandvik and Pullan (Brandvik and Pullan, 2008) on the GPU and a CPU with single core. They observed that the GPU is twenty-nine times faster for 2D problems and sixteen times faster for 3D problems. Further, they conclude that GPU is well suited for CFD problems. This is perhaps a way forward towards the solution of such problems as detailed above.

Bibliography

Abdolmaleki, K., Thiagarajan, K., and Morris-Thomas, M. (2004). Simulation of the Dam Break Problem and Impact Flows using a Navier-Stokes Solver. Technical report, 15th Australasian Fluid Mechanics Conference, The University of Sydney, Australia.

Aliabadi, S. and Shujaee, K. (2001). Free-surface Flow Simulations Using Parallel Finite Element Method. *SIMULATION*, 76(5):257–262.

Anderson, J. D. (1995). *Computational Fluid Dynamics (The Basics with Application)* . McGraw-Hill Book Co. Singapore.

Ansorge, R. (2003). *Mathematical Models of Fluid Dynamics: Modelling, Theory, Basic Numerical Fact- An Introduction*. WILEY-VCH GmbH & Co. KGaA, Weinheim, Germany.

Ansys (2006). *Fluent 6.3 User's Guide*. http://www.ansys.com/products/fluid-dynamics/fluent/.

Arabshahi, M. and Dehghan, M. (2006). Preconditioning Techniques for Solving Large Sparse Linear Systems Arising from the Discretization of the Elliptic Partial Differential Equations. *Applied Mathematics and Computation*, 188:1371–1388.

Ashgriz, N. and Poo, J. (1991). FLAIR: Flux Line Segment Model for Advection and Interface. *Journal of Computational Physics*, 93:449–468.

Aulisa, E., Manservisi, S., Scardovelli, R., and Zaleski, S. (2007). Interface reconstruction with least-squares fit and split advection in three-dimensional Cartesian geometry. *Journal of Computational Physics*, 225(2):2301–2319.

Axelsson, O. (1996). *Iterative Solution Methods*. Cambridge University Press Cambridge, UK.

Basermann, A. (2000). Parallel Block ILUT/ILDLT Preconditioning for Sparse Eigenproblems and Sparse Linear Systems. *Numerical Linear Algebra with Application*, 7:635–648.

Beaumont, O., Legrand, A., Rastello, F., and Robert, Y. (2001). Static LU Decomposition on Heterogeneous Platforms. *International Journal of High Performance Computation Applications*, 15(3):310–323.

Behra, S. and Mittal, S. (2009). Parallel finite element computation of incompressible flows. *Parallel Computing*, 35:195–212.

Beltrán, M. and Guzmán, A. (2009). How to balance the load on heterogeneous clusters. *International Journal of High Performance Computing Applications*, 23(1):99–118.

Benzi, M. (2002). Preconditioning Techniques for Large Linear Systems: A Survey. *Journal of Computational Physics*, 182:418–477.

Benzi, M., Golub, G. H., and Liesen, J. (2005). Numerical solution of saddle point problems. *Acta Numerica*, 14:1–137.

Bergamaschi, L. and Àngeles, M. (2005). Parallel Acceleration of Krylov Solvers by Factorized Approximate Inverse Preconditioners. *Lecture notes in Computer Science*, 3402:623–636.

Bhaga, D. and Weber, M. (1981). Bubbles in viscous liquids: shapes, wakes and velocities. *International Journal of Fluid Mechanics*, 105:61–85.

Birken, P., Tebbens, J., Meister, A., and Tuma, M. (2008). Preconditioner Updates Applied to CFD Model Problems. *Applied Numerical Mathematics*, 58:1628–1641.

Brandvik, T. and Pullan, G. (2008). Acceleration of a 3D Euler Solver Using Commodity Graphics Hardware. In *Proceedings 48th Aerospace Sciences Meeting and Exhibit, AIAA Press*, page 607.

Bruneau, C. and Saad, M. (2001). The 2D Lid-Driven Cavity Problem Revisited. *Computers and Fluids*, 35:326–348.

Buttari, A., Eijkhout, V., Langou, J., and Filippone, S. (2007). Performance Optimization and Modelling of Blocked Sparse Kernels. *International Journal of High Performance Computing Applications*, 21(4):99–118.

Bycul, R., Jordan, A., and Cichomski, M. (2002). A New Version of Conjugate Gradient Method Parallel Implementation. *Proceeding of the International Conference on Parallel Computing in Electrical Engineering(PARELE,02).*

Castillo, Z., Xie, X., Sorensen, D., Embree, M., and Pasquali, M. (2009). Parallel Solution of Large-Scale Free Surface Viscoelastic Flows via Sparse Approximate Inverse Preconditioning. *Journal of Non-Newtonian Fluid Mechanics*, 157:44–54.

Cerne, G., Petelin, S., and Tiselj, L. (1998). Shape and Stability of the Bubble Moving Through the Viscid Incompressible Fluid. *Nuclear Energy in Central Europe*, 46:211–218.

Chen, G. and Kahrif, C. (1999). Two-dimensional Navier-Stokes Simulation of Breaking Waves. *Physics of Fluids*, 11(1):121–133.

Chen, L., Garimella, S., Reizes, J., and Leonardi, E. (1999). The Development of a Bubble Rising in a Viscous Liquid. *Journal of Fluid Mechanics*, 387:61–96.

Chow, E. and Saad, Y. (1997). Experimental Study of ILU Preconditioners for Indefinite Matrices. *Journal of Computational and Applied Mathematics*, 86:387–414.

Christopher, E. (2005). *Fundamentals of Multiphase Flow.* Cambridge University Press, U.K.

Clift, R., Grace, J., and Weber, M. (1978). *Bubbles, Drops and Particles.* Academic Press, U.K.

Cowles, G. W. (2008). Parallelization of the FVCOM Coastal Ocean Model. *International Journal of High Performance Computing Applications*, 22(2):177–193.

Dag, H. (2007). An Approximate Inverse Preconditioner and its Implementation for Conjugate Gradient Method. *Parallel Computing*, 33:83–91.

Dalcin, L., Paz, R., Storti, M., and D'Elia, J. (2008). MPI for Python: Performance improvments and MPI-2 extensions. *Journal of Parallel and Distributed Computing*, 68:655–662.

Delaure, Y. (2001). *A Hydrodynamics Investigation of Oscillating water Column Wave Energy Converters.* PhD thesis, University College Cork,Cork Ireland.

Delaure, Y. and Lewis, A. (2003). 3D Hydrodynamics Modelling of Fixed Oscillating Water Column Wave Power Plant by a Boundary Element Method. *Ocean Engineering*, 30(3):309–330.

Denis, G., Ali, N., Scardovelli, R., and Zaleski, S. (1999). Volume-of-fluid Interface Tracking with Smoothed Surface Stress Methods for Three Dimensional Flows. *Journal of Computational Physics*, 152:423–456.

Devals, C., Heniche, M., Beratrand, F., Tanguy, P., and Hayes, R. (2007). A Two-Phase Flow Interface-capturing Finite Element Method. *International Journal for Numerical Methods in Fluids*, 53:735–751.

Diosady, L. and Darmofal, D. (2009). Preconditioning methods for discontinuous Galerkin solutions of the Navier-Stokes equations . *Journal of Computational Physics*, 228:3917–3935.

Dongarra, J., Moore, S., Peterson, G., Tomov, S., Allred, J., Natoli, V., and Richie, D. (2008). Exploring new architectures in accelerating CFD for Air Force applications. In *Proceedings of HPCMP Users Group Conference 2008, July 14-17*.

Duff, I. and van der Vorst, H. (1998). Preconditioning and Parallel Preconditioning. Technical report, Ruthrford Appleton Laboratory, Oxon, OX11 0QX U.K.

Duncan, R. (1990). A Survey of Parallel Computer Architectures. *Computer*, 23(2):5–16.

Elman, H., Howle, V., Shadid, J., Shuttleworth, R., and Tuminaro, R. (2008). A taxonomy and comparison of parallel block multi-level preconditioners for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 227:1790–1808.

Elman, H. C. (1992). Iterative Methods for Linear Systems. In *Advances in Numerical Analysis,vol 3 Proceedings of the Summer School in Numerical Analysis*, volume 20, pages 69–118. Clarendon Press.

Ernst, T. (2009). q-Bernoulli and q-Euler Polynomials, an Umbral Approach II. Technical report, Department of Mathematics Uppsala University, Sweden. http://www.math.uu.se/research/pub/Ernst19.pdf.

Erturk, E. (2008). Discussions on Driven Cavity Flow. *International Journal for Numerical Methods in Fluids*, 60(3):275–294.

Erturk, E., Corke, T., and Gokcol, C. (2001). Numerical Solutions of 2D Steady Incompressible Driven Cavity Flow at High Reynolds Numbers. *International Journal for Numerical Methods in Fluids*, 48:747–774.

Ferziger, J. and Peric, M. (1999). *Computational Methods for Fluid Dynamics*. Springer-Verlag Berlin Heidelberg, Germany.

Fischer, E. (1983). *Intermediate Real Analysis*. Springer-Verlag, New York, USA.

Fletcher, C. A. (1991). *Computational Techniques for Fluid Dynamics*. Springer-Verlag, Berlin-Heidelberg, Germany.

Fletcher, R. (1976). Conjugate Gradient Method for Indefinite Systems. *Lecture notes in Mathematics*, 506:73–89.

Galdi, G. (2000). An Introduction to the Navier-Stokes Initial-Boundary Value Problem . In *Fundamental Directions in Mathematical Fluid Mechanics*, pages 1–98, Switzerland. Verlag.

Ghia, U., Ghia, K., and Shin, C. (1982). High-Resolutions for Incompressible Flow using the Navier-Stokes Equations and a Multigrid Method. *Journal of Computational Physics*, 48:387–411.

Gibou, F., Chen, L., Nguyen, D., and Banerjee, S. (2007). A Level Set Based Sharp Interface Method for the Multiphase Incompressible Navier-Stokes Equations with Phase Change. *Journal of Computational Physics*, 222:536–555.

Giraud, L., Haidar, A., and Watson, L. (2008). Parallel scalability study of hybrid preconditioners in three dimensions. *Parallel Computing*, 34:363–379.

Gjesdal, T. and Lossius, M. E. H. (1997). Comparison of Pressure Correction Smoothers for Multigrid Solution of Incompressible Flow. *International Journal for Numerical Methods in Fluids*, 25(4):393–405.

Golub, G. H. and Van Loan, C. F. (1996). *Matrix Computations*. The Johns Hopkins University Press London.

Greaves, D. (2004). A Quadtree Adaptive Method for Simulating Fluid Flows with Moving Interfaces. *Journal of Computational Physics*, 194:35–56.

Greaves, D. (2006). Simulation of Viscous Water Column Collapse Using Adapting Hierarchical Grids. *International Journal for Numerical Methods in Fluids*, 50:693–711.

Grille, A., Yang, B., Khomami, B., and Shaqfeh, E. (1999). Modeling of Viscoelastic Lid Driven Cavity Flow using Finite Element Simulations. *Journal of Non-Newtonian Fluid Mechanics*, 88:99–131.

Gu, T., Zuo, X., Liu, X., and Li, P. (2009). An Improved Parallel Hybrid Bi-conjugate Gradient Method Suitable for Distributed Parallel Computing. *Journal of Computational and Applied Mathematics*, 226(1):55 – 65.

Harvie, D. and Fletcher, D. (2000). A New Volume of Fluid Advection Algorithm: The Stream Scheme. *Journal of Computational Physics*, 162:1–32.

Harvie, D. and Fletcher, D. (2001). A New Volume of Fluid Advection Algorithm: The Defined Donating Region Scheme. *International Journal for Numerical Methods in Fluids*, 35:151–172.

Hecquet, D., Ruskin, H. J., and Crane, M. (2007). Optimisation and parallelisation strategies for Monte Carlo simulation of HIV infection. *Computers in Biology and Medicine*, 37(5):691–699.

Henon, P., Ramet, P., and Roman, J. (2008). On finding approximate supernodes for an efficient block-ILU(k) factorization. *Parallel Computing*, 34:345–362.

Henrie, M. and Stanley, G. (1997). *Two Phase Flow in Pipeline Simulation Slack Line Condition in Trans-Alaska Crude Oil Pipeline*. http://www.psig.org/papers/1990/9708.pdf (accessed on May 2009).

Hieu, P., Katsutoshi, T., and Ca, V. (2004). Numerical Simulation of Breaking Waves using a Two-Phase Flow Model. *Applied Mathematical Modelling*, 28:983–1005.

Hirt, C. and Nichols, B. D. (1981). Method for the Dynamics of Free Boundaries. *Journal of Computational Physics*, 39:201.

Hogg, P., Gu, X., and Emerson, D. (2006). An Implicit Algorithm for Capturing Sharp Fluid Interfaces in the Volume of Fluid Advection Method. Technical report, Council for the Central Laboratory of the Research Council(CCLRC) Daresbury Laboratory, Warrington, UK.

Horn, R. A. and Johnson, C. R. (1985). *Matrix Analysis*. Cambridge University Press.

Hua, J. and Lou, J. (2007). Numerical Simulation of Bubble Rising in Viscous Liquid. *Journal of Computational Physics*, 222:769–795.

Hua, J., Stene, J., and Lin, P. (2008). Numerical Simulation of Bubble Rising in Viscous Liquid Using a Front Tracking Method. *Journal of Computational Physics*, 227:3358–3382.

Hyman, M. J. (1984). Numerical Methods for Tracking Interfaces. *Physica*, 12D:396–407.

Itoh, S. and Namekawa, Y. (2003). An Improvement in DS-Bi-CGSTAB(l) and Its Application for Linear Systems in Lattice QCD . *Journal of Computational and Applied Mathematics*, 159:65–75.

Jennings, A. and McKeown, J. (1992). *Matrix Computations*. 2nd Ed. J. Wiley and Sons, New York.

Jordan, H. F. and Alaghband, G. (2003). *Fundamentals of Parallel Processing*. Prentice Hall Pearson Education Inc. Upper Saddle River, NJ.

Kaceniauskas, A. (2005). Dam Break Flow Simulation by the Pseudo-Concentration Method. *MECHANIKA*, 51(1):39–43.

Kahou, G., Grigori, L., and Sosonkina, M. (2008). A partitioning algorithm for block-diagonal matrices with overlap. *Parallel Computing*, 34(10):332 – 344.

Karniadakis, G. M. and Kirby, R. (2003). *Parallel Scientific Computing in C++ and MPI*. Cambridge University Press, New York, NY, USA.

Karypis, G. and Kumar, V. (1997). Parallel threshold-based ILU factorization. In *Supercomputing '97: Proceedings of the 1997 ACM/IEEE conference on Supercomputing (CDROM)*, pages 1–24.

Ketabdari, M., Nobari, M., and Larmaei, M. (2008). Simulation of Waves Group Propagation and Breaking in Coastal zone using a Navier-Stokes Solver with an Improved VOF Free Surface Treatment. *Applied Ocean Research*, 30:130–143.

Kim, M. and Lee, W. (2003). A New VOF-based Numerical Scheme for the Simulation of Fluid Flow with Free Surface. Part I: New free surface-tracking algorithm and its verification. *International Journal for Numerical Methods in Fluids*, 42:765–790.

Kumar, B., Delaure, Y., and Crane, M. (2008). VOF(PLIC) method for capturing sharp fluid interfaces. *Irish Society for Scientific and Engineering Computations and Irish Mechanics Society:Joint Symposium on Advances in Mechanics, University College Dublin, Ireland*, pages 15–16.

Kumar, B.V.R. and Kumar, B. (2005a). On the Development of Parallel Sparse CGM Solver for CFD Computation on ANU-cluster. *Applied Mathematics and Computations*, 160(3):739–761.

Kumar, B.V.R. and Kumar, B. (2005b). Parallel Preconditioners for Heat Transfer Applications on ANU-cluster. *Applied Mathematics and Computations*, 163(3):1243–1263.

Kumar, B.V.R., Kumar, B., Shalini, and Mehra, M. (2005). Non-stationary Iterative Solver on a PC Cluster. *Advances in Engineering Software*, 36:393–400.

Kumar, B.V.R., Shalini, and Mehra, M. (2004). Krylov Subspace Solvers in Parallel Numerical Computations of Partial Differential Eqautions Modelling Heat Transfer Applications. *Numerical Heat Transfer: Part A*, 45:479–503.

Lawson, N., Rudman, M., Guerra, A., and Liow, J. (1999). Experimental and numerical comparison of the break-up of large bubble. *Experiments in Fluids*, 26:524–534.

Le, T. T. and Rejeb, J. (2006). A detailed MPI communication model for distributed systems. *Future Genration Computer Systems*, 22:269–278.

Leonard, B. (1991). The Ultimate Conservative Difference Scheme Applied to Unsteady One-dimensional Advection. *Computer Methods in Applied Mechanics and Engineering*, 88:17–74.

Li, K. (2005). Fast and Scalable Parallel Matrix Computations on Distributed Memory Systems. In *19th IEEE International Parallel and Distributed Processing Symposium*.

Liovic, P., Liow, J., and Rudman, M. (2001). A Volume of Fluid (VOF) Method for the Simulation of Metallurgical Flows. *The Iron and Steel Institute of Japan(ISIJ) International*, 41:225–233.

Liu, C., Woo, C., and D.Y.C., L. (2005). Performance Analysis of a Linux PC Cluster Using a Direct Numerical Simulation of Fluid Turbulence . *International Journal of High Performance Computing Applications*, 19(4):365–374.

Liu, P. and Li, K. (2002). Programming the Bi-CGSTAB matrix Solver for HPC and Benchmarking IBM SP3 and Alpha ES40. *Proceeding of the International Parallel and Distributed Processing Symposium*.

Loth, E. (2000). Numerical approaches for motion of dispersed particles, droplets and bubble. *Progress in Energy and Combustion*, 26:161–223.

Malas, T. and Gürel, L. (2007). Incomplete LU Preconditioning with the Multilevel Fast Multipole Algorithm for Electromagnetic Scattering. *SIAM Journal on Scientific Computing*, 29(4):1476–1494.

Martin, J. and Moyce, W. (1952a). An Experimental Study of the Collapse of Fluid Columns on a Rigid Horizontal Plane. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 244(882):312–324.

Martin, J. and Moyce, W. (1952b). An Experimental Study of the Collapse of Fluid Columns on a Rigid Horizontal Plane, In a Medium of Lower, But Comparable, Density. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 244(882):325–334.

Martinez-Bazaz, C., Montanes, J., and Lasheras, J. (1999a). On the Breakup of an Air Bubble Injected into a Fully Developed Turbulent Flow. Part 2.Size PDF of the Resulting Daughter Bubbles. *Journal of Fluid Mechanis*, 401:183–207.

Martinez-Bazaz, C., Montanes, J., and Lasheras, J. (1999b). On the Breakup of an Air Bubble Injected into a Fully Developed Turbulent Flow.Part 1. Breakup Frequency. *Journal of Fluid Mechanis*, 401:157–182.

Mavriplis, D. J., Aftosmis, M. J., and Berger, M. (2007). High Resolution Aerospace Applications Using the NASA Columbia Supercomputer. *International Journal of High Performance Computing Applications*, 21(1):106–126.

Mayer, A. and Lenhard, R. (1998). Special Issue on Multiphase Flow and Chemical Transport. *Advances in Water Resources*, 21:75–76.

Mellor-Crummey, J. and Gravin, J. (2004). Optimizing Sparse MatrixŰVector Product Computations Using Unroll and Jam. *International Journal of High Performance Computing Applications*, 18(2):225–236.

Melville, W. (1996). The Role Of Surface-wave Breaking in Air-Sea Interaction. *Annual Review of Fluid Mechanics*, 28:279–321.

Mezher, D. and Philippe, B. (2002). Parallel Computation of pseudospectra of large sparse matices. *Parallel Computing*, 28:199–221.

Morris, J. (2000). Simulating Surface Tension with Smooth Particle Hydrodynamics. *International Journal of Numerical Methods in Fluids*, 33:333–353.

Moses, B. (2007). *Simulation of the Multiphase Fluid Flow Using a Spatial Filtering Process*. PhD thesis, Stanford University.

Navti, S., Lewis, R., and Taylor, C. (1998). Numerical Simulation of Viscous Free Surface Flow. *International Journal for Numerical Methods in Fluids*, 8(4):445–464.

Nehalem (2009). *First the Tick, Now the Tock:Next Generation Intel Microarchitecture (Nehalem)*. http://www.intel.com/technology/architecture-silicon/next-gen/.

Nguyen, V., Peraire, J., Khoo, B. C., and Persson, P. (2009). A discontinuous Galerkin front tracking method for two-phase flows with surface tension. *Computers and Fluids*.

Nobari, M., Ketabdari, M. J., and Moradi, M. (2009). A Modified Volume of Fluid Advection Method for Uniform Cartesian Grids. *Applied Mathematical Modelling*, 33:2298–2310.

Nordsveen, M. and Moe, R. (1999). Preconditioned Krylov subspace methods used in solving two-dimensional transient two-phase flows. *International Journal for Numerical Methods in Fluids*, 31(7):1141–1156.

Numeca (2008). *Numeca Fine/Marine*. http://www.numeca.com.

Osei-Kuffuor, D. and Saad, Y. (2009). Preconditioning Helmholtz linear systems. *Applied Numerical Mathematics (To appear)*, pages 1–19.

Ouarraui, C. and Kaeli, D. (2004). Developing Object-oriented Parallel Iterative Methods. *High Performance Computing and Networking*, 1(1/2/3):85–90.

Patankar, S. (1980). *Numerical Heat Transfer and Fluid Flow*. Hemisphere Publishing Corporation, Taylor and Francis Group, New York.

Patankar, S. and Spalding, D. (1972). A Calculation Procedure of Heat, Mass and Momentum Transfer in Three-dimensional Parabolic Flows. *International Journal of Heat and Mass Transfer*, 15:1787–1806.

Peiró, J. and Sherwin, S. (2005). Finite Difference, Finite Element and Finite Volume Methods for Partial Differential Equaiotns. *Handbook of Materials Modelling*, I:1–32.

Peric, M., Kessler, R., and Scheuerer, G. (1988). Comparison of Finite-Volume Numerical Methods with Staggered and Collocated Grids. *Computers and Fluids*, 16(4):389–403.

Petersen, D. E., Li, S., Stokbro, K., Sorensen, H. H. B., Hansen, P. C., Skelboe, S., and Darve, E. (2009). A hybrid method for the parallel computation of Green's functions. *Journal of Computational Physics*, 228(14):5020–5039.

Pichel, J., Heras, D., Cabalerio, J., Garcìa-Loureiro, A., and Rivera, F. (2009). Increasing The Locality of Iterative Methods and Its Application to the Simulation of Semiconductor Devices. *International Journal of High Performance Computing Applications*, 00(0):1–18.

Pommerell, C. (1999). *Solution of Large Unsymmetric Systems of Linear Equations*. Hartung-Gorre Verlag Konstanz, Zurich.

Qian, L., Causon, D. M., Ingram, D. M., and Mingham, C. (2003). Cartesian Cut Cell Two-Fluid Solver for Hydraulic Flow Problems. *Journal of Hydraulic Engineering*, 129(9):688–696.

Quan, S., Lou, J., and Schmidt, D. P. (2009). Modeling merging and breakup in the moving mesh interface tracking method for multiphase flow simulations. *Journal of Computational Physics*, 228(7):2660–2675.

Quan, S. and Schmidt, D. P. (2007). A Moving Mesh Interface Tracking Method for (3D) Incompressible Two-Phase Flows. *Journal of Computational Physics*, 221(2):761–780.

Quinn, M. (2004). *Parallel Programming in C with MPI and open MP*. Dubuque, Iowa McGraw-Hill.

Rahman, M., N., M., and Kawasaki, K. (2006). Numerical modelling of dynamics responses and mooring forces of submerged floating breakwater. *Costal Engineering*, 55:799–815.

Raymond, F. and Rosant, J. (2000). A Numerical and Exeperimental Study of the Terminal Velocity and Shape of Bubble in Viscous Liquids. *Chemical Enmgineering Sciences*, 55:943–955.

Rider, W. and Kothe, D. (1998). Reconstructing Volume Tracking. *Journal of Computational Physics*, 141:112–152.

Ruben, S. and Zaleski, S. (2003). Interface Reconstruction with Least-square Fit and Split Eulerian-Lagrangian Advection. *International Journal for Numerical Methods in Fluids*, 41:251–274.

Rudman, M. (1997). Volume-Tracking Methods for Interfacial Flow Calculations. *International Journal for Numerical Methods in Fluids*, 24:671–691.

Saad, Y. (1989). Krylov Subspace Methods on Super Computers. *SIAM Journal of Scientific and Statistical Computing*, 10:1200–1232.

Saad, Y. (1992). Preconditioning Techniques for Nonsymmetric Indefinite Linear Systems. Technical report, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign.

Saad, Y. (1996). *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company; International Thomson Publishing Inc, Bostan.

Saad, Y. (1997). ILUT: A Dual Threshold Incomplete LU Factorization. *Numerical Linear Algebra with Applications*, 1(4):387–402.

Saad, Y. and Schultz, M. H. (1986). GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. *SIAM Journal on Scientific and Statistical Computing*, 7:856–869.

Saad, Y. and van der Vorst, H. (2000). Iterative Solution of Linear Systems in the 20th Centuary. *Journal of Computational and Applied Mathematics*, 123:1–33.

Scardovelli, R. and Zaleski, S. (2000). Analytical Relations Connecting Linear Interfaces and Volume Fractions in Rectangular Grids. *Journal of Computational Physics*, 164(6):228–237.

Scardovelli, R. and Zaleski, S. (2003). Interface Reconstruction with Least-square fit and Split Eulerian-Lagrangian advection. *International Journal of Numerical Methods in Fluids*, 41:251–274.

Secher, B., Belliard, M., and Christophe, C. (2009). Numerical Platon: A unified linear equation solver interface by CEA for solving open foe scientific applications. *Numerical Engineering and Design*, 239:87–95.

Shahnaz, R., Usman, A., and Chughati, I. (2006). Implementation and Evaluation of Parallel sparse Matrix-Vector Products on Distributed Memory Parallel Computers,Barcelona, Spain . In *IEEE International Conference on Cluster Computing (CLUSTER) Barcelona, Spain* , pages 1–6.

Shang, Y. (2009). A Distributed Memory Parallel Gauss-Seidel Algorithm for Linear Algebraic Systems. *Computers and Mathematics with Applications*, 57(8):1369–1376.

Shankar, P. and Deshpande, M. (2000). Fluid Mechanics in the Driven Cavity. *Annual Rev. Fluid Mechanics*, 32:921–937.

Sheen, S. and Wu, J. (1998). Preconditioning Techniques for the Bi-CGSTAB Algorithm used in Convection-Diffusion Problems. *Numerical Heat Transfer*, 34(6):241–256.

Shen, C., Zhang, J., and Wang, K. (2003). Parallel Multilevel Block ILU Preconditioning Techniques for Large Sparse Linear Systems. In *Proceedings of the International Parallel and Distributed Processing Symposium*.

Singhal, A. K., Athavale, M. M., Li, H., and Jiang, Y. (2002). Mathematical Basis and Validation of the Full Cavitation Model. *Journal of Fluids Engineering*, 124:617–624.

Skuratovsky, I. and Levy, A. (2004). Finite Volume Approach for Solving Multiphase Flows in Vertical Pnueumatic Dryers. *International Journal of Heat and Fluid Flow*, 14(8):980–1001.

Sleijpen, G. and Fokkema, D. (1993). Bi-CGSTAB(l) for Linear Equations Involving Unsymmetric Matrices with Complex Spectrum. *Electronic Transactions on Numerical Analysis*, 1:11–32.

Straubhaar, J. (2008). Parallel Preconditioners for the Conjugate Gradient Algorithm using Gram-Schmidt and least squares methods. *Parallel Computing*, 34(10):551 – 569.

Suda, R., Aoki, T., Hirasawa, S., Akira, N., Honda, H., and Matsuoka, S. (2009). Aspects of GPU for General Purpose High Performance Computing. In *Proceedings of the 2009 Asia and South Pacific Design Automation Conference*, pages 216–223.

Sun, J., Cao, J., and Yang, C. (2009). Parallel Preconditioners for Large Scale Partial Difference Equation Systems. *Journal of Computational and Applied Mathematics*, 226:125–135.

Sussman, M. and Smereka, P. (1994). A Level Set Approach for Computing Solution to Incompressible Two-Phase Flow. *Journal of Computational Physics*, 114:146–159.

Tao, T. (2005). Moving Mesh Methods for Computational Fluid Dynamics. In *Recent Advances in Adaptive Computations Contemporary Mathematics*, pages 1–31. American Mathematical Society.

Tao, W., Cai, D., Yan, X., Ichi, N., and Lembege, B. (2008). Scalability analysis of parallel Particle-In-Cell codes on computational grids. *Computer Physics Communications*, 179:855–864.

Topsakal, E., Kindt, R., Sertel, K., and Volakis, J. (2001). Evaluation of the Bi-CGSTAB(l) Algorithm for the Finite-element/Boundary-integral Method. *IEEE Antennas and Propagation Magazine*, 43(6):124–131.

Tornberg, A. (2000). *Interface Tracking Methods with Applications to Multiphase Flows*. PhD thesis, Royal Institute of Technology, Stokeholm.

Tryggvason, G., Bunner, B., Esmaeeli, A., Juric, D., and Al-Rawahi, N. (2001). A Front-tracking Method for the Computation of Multiphase Flow. *Journal of Computational Physics*, 169:708–759.

Ubbink, O. (1997). *Numerical prediction of two fluid systems with sharp interfaces*. PhD thesis, University of London.

Ubbink, O. and Issa, R. (1999). A Method for Capturing Sharp Fluid Interfaces on Arbitrary Meshes. *Journal of Computational Physics*, 153:26–50.

Unverdi, S. O. and Tryggvason, G. (1992). A Front-tracking Method for Viscous, Incompressible, Multi-fluid Flows. *Journal of Computational Physics*, 100:25–37.

van der Vorst, H. (2002). Efficient and reliable iterative methods for linear systems. *Journal of Computational and Applied Mathematics*, 149:251–265.

van der Vorst, H. A. (2003). *Iterative Krylov Methods for Large Linear Systems*. Cambridge University Press, Cambridge, UK.

Versteeg, H. K. and Malalasekera, W. (1995). *An Introduction to Computational Fluid Dynamics:(The Finite Volume Approach)*. Pearson Prentice Hall, London.

Vuik, C., Van, R. R. P. N., and Wesseling, P. (1998). Parallelism in ILU-preconditioned GMRES. *Parallel Computing*, 24(14):1927–1946.

Wang, W., Kosakowski, G., and Kolditz, O. (2009). A Parallel Finite Element Scheme for Thermo-hydro-mechanical (THM) Coupled Problems in Porous Media. *Computers and Geosciences*, In press(1).

Welch, W. (1995). Local Simulation of Two-Phase Flows Including Interface Tracking with Mass Transfer. *Journal of Computational Physics*, 121(1):142–154.

Williams, S., Oliker, L., Vuduc, R., Shalf, J., and Yelick, K. (2009). Optimization of Sparse Matrix-vector Multiplication on Emerging Multicore Plateforms. *Parallel Computing*, 35:178–194.

Wu, C., Lai, L., Yang, C., and Chiu, P. (2009). Using hybrid MPI and Open MP Programming to optimize Communications in Parallel loop Self-scheduling Schemes for Multicore PC Clusters. *The Journal of Supercomputing (online)*.

Wu, J. and Shao, Y. (2004). Simulation of Lid-Driven Cavity Flows by Parallel Lattice Boltzmann Method using Multi-Relaxation-Time Scheme. *International Journal for Numerical Methods in Fluids*, 46:921–937.

Yu, B., Ozoe, H., and Tao, W. (2005). A Collocated Finite Volume Method for Incompressible Flow on Unstructured Meshes. *Progress in Computational Fluid Dynamics*, 5:181–189.

Zalesak, S. (1979). Fully Multidimensional Flux-Corrected Transport Algorithms for Fluids. *Journal of Computational Physics*, 31:335–362.

Zhao, Q., Armfield, S., and Tanimoto, K. (2004). Numerical simulation of breaking waves by a multi-scale turbulence model. *Coastal Engineering*, 51(1):53–80.

Zhi, Z. and hai, G. (2007). A Second Order Volume of Fluid(VOF) Scheme for Numerical simulation of 2-D Breaking Waves. *Journal of Marine science and Application,*, 6(2):1–5.

Zienkiewicz, O. C. and Morgan, K. (1983). *Finite Elements and Approximation*. Wiley-Interscience, New York.

Zucchini, A. (2000). A parallel Preconditioned Conjugate Gradient Solution Method for Finite Element Problems with Coarse-fine Mesh Formulation. *Computers and Structures*, 78:781–787.