2006

A Parallelized sharp-interface fixed grid method for moving boundary problems

Saikrishna V Marella
University of Iowa

A PARALLELIZED SHARP-INTERFACE FIXED GRID METHOD FOR MOVING BOUNDARY PROBLEMS

by

Saikrishna V Marella

An Abstract

Of a thesis submitted in partial fulfillment
of the requirements for the Doctor of
Philosophy degree in Mechanical Engineering
in the Graduate College of
The University of Iowa

July 2006

Thesis Supervisor: Associate Professor H.S.Udaykumar

ABSTRACT

The primary objective of this thesis is to develop a general computational framework to perform large scale moving boundary problems in fluid mechanics. The interactions of moving entities with fluid flow are common to numerous engineering and biomedical applications. The novel computational platform developed comprises of a) an efficient fluid flow solver b) an accurate and easily implemented unified formulation to capture the interactions of the moving bodies with the flow and c) parallel execution capability to enable large scale computations. The above features are formulated and implemented in a computer code, ELAFINT3D. The current thesis demonstrates the accuracy, efficiency and robustness of this framework. The performance of ELAFINT3D on distributed memory systems is also presented. Finally, this framework is employed to simulate a series of large scale, three-dimensional moving boundary problems involving complex interfacial motions and flow phenomena. These numerical experiments establish the strengths of the current tool.

Abstract Approved: _____
Thesis Supervisor

_____
Title and Department

_____
Date

A PARALLELIZED SHARP-INTERFACE FIXED GRID METHOD FOR MOVING BOUNDARY PROBLEMS

by

Saikrishna V. Marella

A thesis submitted in partial fulfillment
of the requirements for the Doctor of
Philosophy degree in Mechanical Engineering
in the Graduate College of
The University of Iowa

July 2006

Thesis Supervisor: Associate Professor H.S.Udaykumar

Graduate College
The University of Iowa
Iowa City, Iowa

CERTIFICATE OF APPROVAL

PH.D. THESIS

This is to certify that the Ph.D. thesis of

Saikrishna V. Marella

has been approved by the Examining Committee
for the thesis requirement for the Doctor of Philosophy
degree in Mechanical Engineering at the July 2006 graduation.

Thesis Committee: _____
H.S.Udaykumar, Thesis Supervisor

_____
V.C.Patel

_____
K.B.Chandran

_____
Jeffrey S. Marshall

_____
C.Beckermann

ABSTRACT

The primary objective of this thesis is to develop a general computational framework to perform large scale moving boundary problems in fluid mechanics. The interactions of moving entities with fluid flow are common to numerous engineering and biomedical applications. The novel computational platform developed comprises of a) an efficient fluid flow solver b) an accurate and easily implemented unified formulation to capture the interactions of the moving bodies with the flow and c) parallel execution capability to enable large scale computations. The above features are formulated and implemented in a computer code, ELAFINT3D. The current thesis demonstrates the accuracy, efficiency and robustness of this framework. The performance of ELAFINT3D on distributed memory systems is also presented. Finally, this framework is employed to simulate a series of large scale, three-dimensional moving boundary problems involving complex interfacial motions and flow phenomena. These numerical experiments establish the strengths of the current tool.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

Motivation and Objectives

The primary objective of this thesis is to develop a general computational framework to simulate large scale moving boundary problems involving fluid flow. The presence of interacting objects in fluid flow is a common feature to numerous engineering applications. In many cases, the interaction of these immersed entities with the flow is critical to the dynamics of the system. For instance, biomedical phenomena such as the dynamics of heart valves or the mechanics of GI tract, material processing applications such as interaction of solidifying metallic fronts with ceramic particles in metal-matrix composites, automotive/aerospace/marine applications involving design and analysis of vehicles and structures, all involve embedded objects and the flow evolving in tandem. The motivation of the current work is to develop an efficient computational tool to numerically simulate the flow-object interactions and thereby the evolution of the system.

The following aspects are to be considered while simulating these applications:

Dynamics of the embedded objects: The dynamics of an entity immersed in the flow is determined by its geometric definition, its kinematic constraints and its interactions with the flow. In general, the immersed objects have complicated shapes, be it an advancing dendritic front or the GI tract enacting peristaltic motions. These complicated geometries perform complex motions based on their physics. For instance, the trajectories of leaves falling from a tree, plaque growth in carotid artery, growth of the dendritic pattern, flexible membranes etc., are each driven by a different mechanism. Other interfacial mechanisms such as droplet breakup, evaporation, coalescence etc. that result in topological changes are prevalent in many applications. Dynamic interactions with other

immersed objects (e.g. collision of red blood cells or adhesion of platelets etc) require modeling in many applications.

Complex Flow Phenomena: Besides the interface dynamics, the fluid flow itself is complex in itself. Many of the real time flow applications involve three-dimensional unsteady flows. The flow features (gradients) and the flow paths can vary in the interval of observation. Depending on the dominant fluid forces, the flow can transition through different flow regimes. To efficiently capture these evolving flows to the required accuracy is a significant computational challenge.

In the current work, these challenges are addressed through a computational framework called ELAFINT3D (Eulerian Levelset based Algorithm For Interfacial Transport in 3D). This framework features: a) a Cartesian grid based solver for the fluid flow b) the Level-set algorithm to represent and evolve the embedded objects c) the Sharp-Interface Method to capture the interfacial phenomena d) Sparse matrix solvers to efficiently solve the discrete equations e) an Adaptive Local Mesh Refinement algorithm to obtain accurate solutions and d) MPI based parallel heuristics to simulate large scale problems in reasonable time.

The main contributions of this thesis are to conceive and formulate a numerical technique to capture "fluid flow – embedded object" interactions. This numerical technique is implemented in the computer code ELAFINT3D, and is used to solve a wide variety of moving boundary problems. The specific contributions of this thesis are a) to develop a Cartesian grid fluid flow solver with finite-difference schemes b) to develop a Sharp Interface Method to incorporate interfacial physics c) to boost the efficiency of the calculations by employing fast sparse matrix solvers and d) to enable high performance computing through parallel porting of the framework. These features along with Local Mesh Refinement are used to accurately solve a series of two and three dimensional

problems. The computational tool and the results from the numerical experiments are presented in the subsequent chapters of this thesis. The rest of this chapter is organized to review the state of the art in fluid-flow solvers for moving boundary problems. Also reviewed are the general practices in the high performance computing community with focus on CFD applications.

Methods for Moving Boundary Problems

Computational methods for flows in the presence of moving solid boundaries can be broadly classified into Lagrangian (Anderson et al. 1995; Glimm et al. 1998; Johnson et al. 2002; Monaghan and Kocharyan 1995; Tezduyar 2001) and Eulerian categories (Anderson et al. 1998; Benson 1995; Brackbill et al. 1992; Fedkiw et al. 1999; Glowinski et al. 2001; Osher and Fedkiw 2001; Sethian and Smereka 2003; Udaykumar et al. 2002b). In Lagrangian methods, calculations are performed using meshes or particles that follow the motion of the solids while in Eulerian methods the mesh remains fixed. A large volume of literature covers the solution of fluid-structure interaction problems with moving body-fitted meshes, using finite-volume (Glimm et al. 1988; Shyy et al. 1998) and finite-element (Tezduyar 2001) methods. Body-fitted moving mesh methods carry sharp interface definition and application of boundary conditions on the moving solid surface is straightforward. The main difficulty with moving body-fitted meshes is the complexity involved in following arbitrary boundary motions and topological changes while maintaining grid quality (Johnson and Tezduyar 1999). Thus, while for low and moderate deformations moving grid methods are a good choice, for truly large deformations it may be desirable to separate the boundary movement from the grid used to solve the flowfield.

Several strategies have been used in the past to handle the effects of moving immersed interfaces while computing the flowfield on a fixed mesh. Typically such fixed grid methods have relied on simple Cartesian meshes for solving the flow field (Peskin 1977; Udaykumar et al. 2001; Unverdi and Tryggvason 1992) although unstructured triangulated meshes have also been employed (Barth and Sethian 1998; Chessa et al. 2002; Glowinski et al. 1999). In developing Cartesian grid methods, special attention

needs to be paid to the imposition of boundary conditions at the immersed boundaries. In particular, since the immersed boundary can cut through the underlying Cartesian mesh in an arbitrary manner, the main challenge is to construct a boundary treatment which does not adversely impact the accuracy and conservation property of the underlying numerical solver. This is especially critical for viscous flows where adequate resolution of boundary layers which form on the immersed boundaries is required. Consequently, Cartesian grid methods have been used extensively for Euler flows (Almgren et al. 1997; Bayyuk et al. 1993; Pember et al. 1995; Quirk 1994) whereas methods for viscous flows are still being advanced.

Primary classifies of Cartesian Grid Method

Cartesian grid methods now come in several flavors, both in the diffuse and sharp interface categories. Unlike body-fitted curvilinear grid finite-volume methods or unstructured finite-volume and finite-element methods which are fairly standardized, the various Cartesian grid methods differ in implementation details and the choice of a specific implementation depends on the problem to be solved and the demands of accuracy, interface definition etc. Table 1.1 shows a comparison of the two categories. Typically, diffuse interface methods have been deemed to be simpler to implement and therefore have found wide usage (Anderson et al. 1998). In such methods the onus is on designing the source terms so that the formulation approaches the sharp interface limit for vanishing interface thicknesses. Although straightforward to formulate, sharp interface methods have been considered to be somewhat more difficult to implement and therefore their use has been restricted to some specific situations and to a few practitioners. However, it has been demonstrated with the current method (Marella et al., 2004; Liu et al. 2004; Yi and Udaykumar 2004) that with a proper framework, sharp interface

methods can be developed and implemented with considerable ease and applied to a variety of moving boundary problems.

Methods where the interface effects are transmitted through forcing functions

Perhaps the most commonly used fixed grid approach for moving boundary computations involving solid-fluid as well as fluid-fluid interfaces is the immersed boundary method due to Peskin (Peskin 1977). In the original immersed boundary method as well as later versions, the interaction of the boundary with the fluid is effected using smoothed delta-functions located at the boundaries. The effect of the boundaries, in particular boundary forces (such as elastic forces in structures, tethering forces and surface tensions) are transmitted to the momentum equations as source (or forcing) terms:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u}.\vec{\nabla}\vec{u} = -\frac{1}{\rho}\vec{\nabla}p + \upsilon\nabla^2\vec{u} + \vec{f} \qquad 1.1$$

here $\vec{f}$ is the forcing function that represents the effect of the immersed boundary. Typically a numerical δ-function with a support of a few mesh widths is used to convert singular surface forces (such as surface tension) into volume forces $\vec{f}$. The strategy of transmitting interface effects to the flow field through source terms has also been used to solve problems in multiphase flows (Marella and Udaykumar 2004; Tryggvason et al. 2003; Udaykumar et al. 1999) and solidification (Al-Rawahi and Tryggvason 2002). However, one shortcoming of these methods is that discontinuities at the immersed boundary are smeared across a few cell widths. It has been shown (Leveque and Li 1994) that such smearing can adversely impact the accuracy of solutions when the boundary motion is closely coupled with the evolution of surrounding fluid flow. There are also issues involved with stability and stiffness of the computations (Cheng and Peskin 1992; Stockie and Wetton 1999), particularly when the embedded objects deform

along with the flow. Improvements to the delta-function based immersed boundary methods have appeared in the literature in recent years(Lai and Peskin 2000; Lee and Leveque 2003; Roma et al. 1999). In the finite-element setting the fictitious domain method (Adalsteinsson and Sethian 1995; Glowinski et al. 1999; Patankar et al. 2000) and the immersed finite element method ((Wang and Liu 2004)) have followed this idea of transmitting the boundary forces to the fluid using an interaction source term. However, a considerable number of papers on immersed boundary methods in recent years have deviated from the use of delta-functions in transmitting boundary forces to the fluid. For example, in the finite difference/ finite volume methods presented in several recent papers (Balaras 2004; Fadlun et al. 2000; Kim et al. 2001; Tseng and Ferziger 2003) , the idea of using a forcing term in the momentum equation has been retained. These (non-smoothed) forces are placed at points that adjoin the immersed boundary (either inside or outside the solid object) in order to impose the appropriate velocity boundary conditions on the solid surface. Thus, unlike the original immersed boundary method of Peskin and its derivatives, these new immersed boundary methods are in fact sharp interface methods.

Methods where the interface effects are included in the
discrete spatial operators:

There are sharp interface methods, however, that do not use forcing terms but incorporate the presence of the embedded boundaries into the *discrete* form of the Navier-Stokes equations. Thus, in contrast with the above approach, the momentum equation is retained in the form:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u}.\vec{\nabla}\vec{u} = -\frac{1}{\rho}\vec{\nabla}p + \upsilon\nabla^2\vec{u} \qquad 1.2$$

During the discretization procedure however the spatial differential operators ($\vec{\nabla}, \nabla^2$) in the equation are constructed at points that adjoin the interface in such a way that the interfacial jump conditions are incorporated. Examples of this class are the immersed interface method (Lai and Li 2001; Leveque and Li 1995; Li and Lai 2001), the sharp interface method (Udaykumar et al. 2003; Udaykumar et al. 2002b; Udaykumar et al. 2001), the ghost fluid method (Fedkiw et al. 1999; Liu et al. 2000) and the XFEM method (Chessa and Belytschko 2003; Chessa et al. 2002; Dolbow et al. 2001; Sukumar et al. 2001). The immersed interface method (*abbr.* IIM) (Leveque and Li 1995) enables a sharp interface treatment by casting the governing equations in a coordinate system with axes oriented with the local normal and tangent to the interface. Problems involving embedded fluid-fluid interfaces with singular sources and jumps in material property across the interface have been solved using this approach (Leveque and Li 1995; Li and Lai 2001). The method seeks to preserve second-order accuracy at points adjacent to the interface as well as away from it. In Ghost Fluid Method (*abbr.* GFM) (Fedkiw et al. 1999)the governing equations are discretized along the Cartesian coordinate directions. Only first-order accuracy is demanded at interface-adjacent points. In both IIM and GFM, jumps and singular sources at the interface are incorporated into the discrete operators in the transport equations. In the sharp interface method (Udaykumar et al. 2002b; Ye et al. 1999) a finite volume technique is used to discretize the equations within the domains separated by the embedded boundary in such a way that information is not smeared at the immersed boundary. This requires reshaping of the control volumes through which the interface passes and the integration of the weak form of the governing equations over non-rectangular control volumes. Second-order accuracy is maintained at bulk as well as interface-adjacent grid points. Problems involving fluid-structure interactions (Udaykumar et al. 2002b; Udaykumar et al. 2001) and solidification (Udaykumar and Mao 2002; Udaykumar et al. 2002a; Udaykumar et al. 2003; Udaykumar et al. 1999) have been solved using this approach. In the finite element community, the XFEM

method (Sukumar et al. 2000) follows a similar strategy, in that the elements through which the boundary passes are enriched (i.e. these elements are subdivided and conform locally to the immersed boundary; stated differently degrees of freedom are added) to facilitate integration of the weak form of the governing equations.

The Current Method

The key issues in developing a numerical tool for moving boundary problems are accuracy, robustness, speed, ease of formulation and ease of implementation. Even though the sharp interface methods perform at par (if not better than) diffuse interface methods in the above aspects, they have been considered to be somewhat more difficult to implement. In contrary to the above conception, the current work demonstrates that with a proper framework, sharp interface methods can be developed and implemented with considerable ease and applied to a variety of moving boundary problems (Marella et al., 2004; Liu et al. 2004; Yi and Udaykumar 2004).

The current method improvises on the previous Sharp Interface Method of Udaykumar et al. (2001,2002b). Several key choices have been made in the current work to bypass the implementational difficulties apparent in the previous approach. The mainstays of the previous approach (FV+MT) were a finite-volume discretization (FV) of the governing equations and a Lagrangian marker tracking (MT) algorithm to represent and evolve the moving boundaries. The finite volume approach reshapes the interfacial control volumes (cut-cells) to align one of the faces with the interface. This enables the implementation of the interfacial conditions in a sharp manner. Even as FV+MT approach works effectively in 2D, the effort in formulating the cut-cell configurations is immense for 3D moving boundary problems. The current approach circumvents this issue by adopting finite difference (FD) schemes for the governing equations. The

operators/interfacial conditions are evaluated/applied in a sharp manner at the exact interfacial location. The exact interfacial location is calculated easily from the level-set field (LS). Unlike MT, the LS approach doesn't require surface meshing or re-meshing to represent/track the embedded object. Instead, an Eulerian level-set field is evolved in time to track the moving entities. By choosing FD+LS, the current method is easily extended to three-dimensional problems.

The current method differs from the other approaches in the following aspects of implementation:

a) The current approach, being a sharp-interface method, is different from the diffuse interface methods (e.g. immersed boundary method) which smear the interfacial effects over few cell widths.

b) Currently, the interfacial effects are accounted for, by carefully devising discrete spatial operators. By adopting this approach, there are no extra source terms in the interfacial cells to impose interfacial boundary condition unlike (Balaras 2004; Fadlun et al. 2000; Kim et al. 2001; Tseng and Ferziger 2003).

c) By employing finite-difference discretization rather than finite-volume discretization (Udaykumar et al.(2001,2000b)) eases the construction of discrete operators, more significantly in 3D. In addition to FD, choosing level-set approach for interface tracking is advancement over marker tracking approach employed previously.

d) Finally, with the FD+LS combination, the current method convolves the governing equations, the interface representation and the interfacial conditions into the discrete operators ($\vec{\nabla}_D, \nabla_D^2$). By doing so, a unified formulation that incorporates a variety of interface combinations (solid-fluid, fluid-fluid, solid-fluid-solid etc.) has been devised. The unified formulation is facilitated by switch functions which are functions of the level-set field and the type of interface. These

switch functions are the byproducts of the convolution of the governing equations of the problem. This aspect differentiates the current approach from the other methods (GFM, IIM etc) which are specific to a certain class of interfaces.

The above features make the current approach novel in its formulation and far more versatile in its applicability. The unified framework, which accounts for a variety of interfaces in a sharp manner, also shows that sharp interface methods can indeed be easily implemented by carefully formulation of the operators. The details of formulation and the implementation are presented in the later parts of this thesis.

Parallel Computing

Traditionally, the computing power has set the upper bound on the complexity of the problems that can be solved using computational techniques. A modest task such as calculating flow past a sphere (Re=300) with $2x10^6$ grid points takes about 2-3 weeks to obtain time-history of flow characteristics such as drag and lift coefficients on a HP-UX J2000 workstation. The solution turn-around time for many other complex physics-intensive problems such as heart valve simulation, a droplets splashing on a solid surfaces, interaction of growing solidification fronts with solid particles in metal matrix composites, peristaltic motion in GI tract will be enormous. Despite the advances in computer technologies in the recent past, there is a never ending need for computing resources in increasingly sophisticated computational fields such as CFD.

Even though, invention of faster computer chips is an option, many researchers, during the last decade or more, have viewed parallel computing or multiprocessing as an attractive and practical alternative. The strategy being to exploit the resources of

"numerous faster computers" instead of "a single fastest computer" to attempt solve the problems of interest. This idea has gained impetus with developments in computer architecture and standardization of software for parallel programming. A summary of the various system architectures and the software options is presented below.

The classic von Neumann machines which are Single Instruction Single Data (SISD) systems are traditional single processor machines. Basic extensions to these systems are the vector processors (e.g CRAY C90, NEC SX4 etc) which obtain parallelism by issuing vector instructions. For example, the addition of two vectors "a" and "b" of size 100 is a single instruction on a vector processor as opposed to 100 instructions on a classic von Neumann machine. These machines scale well only for small problems with regular structure. The Single Instruction Multiple Data (SIMD) systems (e.g CM-1 and CM-2 produced by Thinking Machines) are the successors of the vector machines. This class of systems has a single control unit and many subordinate Arithmetic Logic Units (ALUs). These systems traditionally use data parallel programming languages (e.g. HPF) to achieve parallelism. Data parallelism relies on distributing a certain data structure among all the processes and each process performs the same set of instructions on its portion of the data structure. Their operations are synchronous and they scale well for regular structures. However they do not scale well for irregular program structures and long conditional branches in the programs.

The most popular architectures are the general Multiple Instruction Multiple Data (MIMD) systems. In these systems, the each processor is a full-fledged CPU. There are two classes of MIMD systems: the shared-memory systems and the distributed memory systems. The shared-memory systems consist of a collection of processors and memory modules interconnected by a network. The shared memory system is the most appealing parallel architecture to the programmer, given an access to all the memory modules by all

the processors. However, scalability is an issue since the memory access speeds are non-uniform for each processor and is dependent on the hardware layout. The most popular programming paradigm for this class of machines is OpenMP. Even though the OpenMP is a well understood and effective programming library, the main drawback is in building scalable shared-memory MIMD systems with large number of processors (more than 128).

Distributed memory systems remain more popular architecture and they scale better compared to the shared memory systems. In distributed memory systems, each processor has its own private memory and the processors are connected in a network. The main drawback for these systems has been that they are difficult to program. Message passing has been the principal strategy to develop parallel algorithms for these architectures. Libraries such as Message Passing Interface (MPI) provide the required functions for effective communication across the processors. The scalability is however dependent on careful programming using the MPI directives. Issues of data communication speeds and relative speeds and locations of various processors are to be accounted for in the programming. With an increasing popularity of massively distributed network of workstation and Grid Computing, distributed MIMDs are the architectures for the future.

The most important issue in parallel CFD is to design numerical algorithms that efficiently exploit the capabilities of the parallel architecture. Especially, given that the flow configurations to be simulated are more often problem specific, to develop generic algorithms that are effective on architectures comprising of as many as several hundreds of processors is non-trivial. In the present case, the objective is to develop a parallel algorithm to be implemented on distributed memory systems such as Linux Clusters. This section details the general strategy for parallelization on distributed MIMD systems.

Domain decomposition is a more appropriate and efficient parallelization method for distributed MIMD architectures. The idea is to divide the computational domain into several subdomains and carry out the numerical solution on each subdomain separately. An information exchange within the set of subdomains is performed during the solving process to update each of the subdomains. This ensures that the solution procedure is carried out independently on each subdomain and in parallel on various processors.

The key factors in this procedure are to design a data decomposition strategy that ensures load balance among the processors and to develop a communication paradigm that minimizes the communication overhead during the synchronization phase. There are software packages such as JOSTLE, METIS that are available for domain decomposition. Given the nodal connectivities, the mesh partitioning programs such as METIS not only decompose the domain into well-balanced partitions but also minimize the number of edges being cut, thereby minimizing the communication costs. METIS also facilitates domain decomposition based on the nodal weights. For example, in complex flows involving immersed objects or multi-physics problems if the nodes are assigned weights based on the computational costs, METIS balances the computational load in each partition based on these nodal weights. The mesh partitioning software also has to account for disparity in the processor speeds across the network.

For the updating phase of the algorithm, the usual approach is to create a halo or ghost region around each domain partition. This halo region comprises of the inter-processor dependencies of the partition boundary mesh points. At the end of each solution process, the halo region of each subdomain is updated for the subsequent calculations. Updating step is essentially a synchronization process which is done by communicating the proper solution to the nodes in the halo region using message passing

such as MPI. The effectiveness of the communication paradigm is determined by the appropriate placement of message passing directives in the program.

NUMERICAL STRATEGY

The Current Computational Tool

This chapter presents a computational framework for simulating three-dimensional incompressible flows and heat and mass transport around complex stationary or moving solid boundaries. The key components of this framework are:

a). An accurate and efficient Sharp Interface Method to solve fluid flow around moving embedded objects.

b). A Local Mesh Refinement algorithm that adaptively resolves the computational space to accurately capture the flow physics.

The current chapter discusses the algorithmic and implementation specifics of each of the above components.

The Sharp Interface Method

A fixed grid sharp interface technique for solving flows with embedded moving boundaries requires a) an effective interface tracking method that allows for sharp interface identification, and b) an appropriate treatment of the interface-adjacent grid points to develop an accurate discretization procedure so that boundary conditions on the embedded boundaries are applied without smearing.

In the current method these two issues are addressed by using a) level-set methods to represent and track the moving interface and b) a finite-difference discretization of the governing equations with modifications of the stencils for the interfacial cells to incorporate the effect of the immersed boundary. The current method improves upon the sharp-interface method (Udaykumar et al. 2002b) which employs markers to track the interfaces and finite-volume discretization with interfacial cell reshaping to account for

the presence of immersed object. The current choice of finite-difference discretization in combination with level-set method for surface tracking circumvents the explicit surface tracking and cut-cell reshaping issues that arise in extending the sharp-interface method (Udaykumar et al, 2002b) to three dimensional problems.

The current method has been implemented in a computer code called ELAFINT3D (**E**ulerian **L**evel-set-based **A**lgorithm **F**or **IN**terfacial **T**ransport in **3D**). The salient features of the approach are:

a). A second-order accurate Cartesian grid based finite-difference scheme to discretize incompressible Navier-Stokes equations. The discretization depends essentially on convolving the differential operators with the distance function field. The result is an easily implemented algorithm where the discretization of the governing equations at all points (i.e. away from as well as adjoining the interface) can be presented in a unified format.

b). A mesh-based level set method to represent the immersed geometry and track the moving boundaries. The geometry is communicated to the flow solver solely through the distance function field.

c). A sharp-interface embedded boundary treatment using an appropriate modification of the stencil for the mesh points adjoining the boundaries. This approach does not smear discontinuities at the interface and does not require forcing terms to transmit boundary effects to the fluid.

d). A Krylov subspace based iterative method to efficiently solve the matrix system resultant from the discrete governing equations, especially the Pressure Poisson equation.

The rest of this chapter is designed to elucidate algorithmic and implementational details of the features in ELAFINT3D. The present framework, with certain adaptations, has

been applied to problems such as droplet-wall interaction (Liu et al. 2004) and solidification (Yi and Udaykumar 2004).

Equations to be solved

The flows of interest in the current work are viscous incompressible flows with density variations due to temperature or solutal inhomogeneities. The differential form of the governing equations for mass and momentum conservation (under Boussinesq approximation) are:

$$\vec{\nabla}.\vec{u} = 0 \tag{2.1}$$

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \vec{\nabla}\vec{u} = -\frac{1}{\rho_0}\vec{\nabla}p' + \frac{\mu}{\rho_0}\nabla^2\vec{u} + \frac{\rho'}{\rho_0}\vec{g} \tag{2.2}$$

$$\frac{D\rho'}{Dt} = -w\frac{\partial \rho_B}{\partial z} \tag{2.3}$$

In the above set of equations, the velocity ($\vec{u}$) and the pressure (p) are the primitive flow variables. The symbols ρ_0, μ and $\vec{g}$ correspond to the reference density, viscosity and acceleration due to gravity vector respectively. The unperturbed and perturbed components of density and pressure (ρ, p) are given by (ρ_B, p_B) and (ρ', p') respectively. The hydrostatic equilibrium ($dp_B/dz = -\rho_B g$) of the basic undisturbed density and pressure distributions (ρ_B, p_B) is already incorporated in Equation 2.2.

Based on the physics of the problem, characteristic velocity, length and density scales (U_0, D and ρ_0) are chosen to non-dimensionalize the governing equations. The governing equations transform to the following dimensionless form,

$$\vec{\nabla}.\vec{u} = 0 \tag{2.4}$$

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \vec{\nabla} \vec{u} = -\vec{\nabla} p + \frac{1}{\mathrm{Re}} \nabla^2 \vec{u} + \frac{\rho}{Fr^2} \vec{g} \qquad 2.5$$

$$\frac{\partial \rho}{\partial t} + \vec{u}.\vec{\nabla} \rho = w \qquad 2.6$$

In the above equation, $\mathrm{Re} = \rho_0 U_0 D/\mu$ and $Fr = U_0/ND$ represent the Reynolds number and internal Froude number respectively, where N is the buoyancy frequency defined as $N = \sqrt{(g/\rho_0)(d\rho_B/dz)}$. The characteristic scales will be defined (based on the problem physics) separately for each of the simulations in this work. The density variations in the domain are neglected in all cases except those involving stratified flows (Lin et al. 1992).

Flow Solver

A cell-centered collocated arrangement of the flow variables is used to discretize the governing equations. A two-step fractional step method (Ye et al. 1999; Zang et al. 1994) is used to advance the solution in time. The first step evaluates an intermediate velocity by solving an unsteady advection-diffusion equation.

$$\frac{\vec{u}^* - \vec{u}^n}{\Delta t} = -\vec{u} \cdot \vec{\nabla} \vec{u} + \frac{1}{\mathrm{Re}} \nabla^2 \vec{u} + \frac{\rho}{Fr^2} \vec{g} \qquad 2.7$$

where the intermediate velocity $\vec{u}^*$ is evaluated with central-difference discretization schemes for convection and diffusion terms. The convective terms are treated explicitly and discretized using a second-order accurate Adams-Bashforth method:

$$\vec{u} \cdot \vec{\nabla} \vec{u} = \frac{1}{2}\left(3\vec{u}^n \cdot \vec{\nabla} \vec{u}^n - \vec{u}^{n-1} \cdot \vec{\nabla} \vec{u}^{n-1}\right) \qquad 2.8$$

The diffusion terms are treated semi-implicitly using Crank-Nicholson scheme:

$$\frac{1}{\mathrm{Re}} \nabla^2 \vec{u} = \frac{1}{2\,\mathrm{Re}}\left(\nabla^2 \vec{u}^* + \nabla^2 \vec{u}^n\right) \qquad 2.9$$

The second fractional-step involves the correction of the intermediate velocity field $\vec{u}^*$ to enforce mass conservation:

$$\frac{\vec{u}^{n+1} - \vec{u}^*}{\Delta t} = -\vec{\nabla} p \qquad 2.10$$

where the pressure field p is evaluated to impose a divergence-free velocity field at time step *n+1*. This is done by taking the divergence of Eq. 2.10 to obtain a Poisson equation for pressure:

$$\nabla^2 p = \frac{\vec{\nabla}.\vec{u}^*}{\Delta t} \qquad 2.11$$

The final semi-discrete form of the equations including each of the above discretization schemes is as follows:

$$\frac{\vec{u}^*}{\Delta t} - \frac{1}{2\,\mathrm{Re}}\nabla^2\vec{u}^* = \frac{\vec{u}^n}{\Delta t} + \frac{\rho\vec{g}}{Fr^2} + \frac{1}{2\,\mathrm{Re}}\nabla^2\vec{u}^n - \frac{1}{2}\left(3\vec{u}^n.\vec{\nabla}\vec{u}^n - \vec{u}^{n-1}.\vec{\nabla}\vec{u}^{n-1}\right) \qquad 2.12$$

$$\nabla^2 p^{n+1} = \frac{\vec{\nabla}.\vec{u}^*}{\Delta t} \qquad 2.13$$

The intermediate velocity is then corrected to obtain the final divergence-free velocity field:

$$\vec{u}^{n+1} = \vec{u}^* - \Delta t\vec{\nabla} p \qquad 2.14$$

Implicit Interface Representation Using Levelsets

Embedded surfaces are represented implicitly on the mesh using a standard level-set approach (Osher and Sethian 1988; Sethian 2001; Sethian and Smereka 2003). In addition to the flow variables, the level-set method advects a scalar field ϕ_l, where subscript l denotes the l^{th} embedded interface. The value of ϕ_l at any point is the signed normal distance from the l^{th} interface with $\phi_l<0$ inside the immersed boundaries and $\phi_l>0$

outside. The interface location is implicitly embedded in the ϕ_l-field since the $\phi_l = 0$ contour represents the l^{th} immersed boundary.

In case of moving interfaces, the motion of the boundary is tracked by advecting the level set using:

$$(\phi_l)_t + \vec{V}_l.\vec{\nabla}\phi_l = 0 \qquad 2.15$$

where $\vec{V}_l$ is the l^{th} level-set velocity field. A fourth-order ENO scheme in space and fourth-order Runge-Kutta integration in time are used for the evolution of the level-set field. Since $\vec{V}_l$ is prescribed by the physics only on the interface (i.e. on the zero-level-set), the value of velocity at the grid points that lie in the narrow band around the zero-level set needs to be obtained. This is done by extension of the interfacial velocity ((Sethian 2001)) away from the front using:

$$\psi_\tau + \vec{V}_{ext}.\vec{\nabla}\psi = 0 \qquad 2.16$$

where ψ is any quantity (such as interface velocity components $(\vec{V}_l)_x$ or $(\vec{V}_l)_y$) that needs to be extended away from the interface. A choice for the extension velocity is $\vec{V}_{ext} = sign(\phi_l)\nabla\phi_l / |\nabla\phi_l|$. This populates the narrow band around each interface in the time $\tau = O(\Delta x)$ with a level-set velocity that has been extended outward from the interface in a direction normal to it. A reinitialization procedure (Sussman and Fatemi 1999; Sussman et al. 1998) is carried out after level-set advection to return the ϕ_l-field to a signed distance function, i.e. to satisfy $|\vec{\nabla}\phi_l| = 1$. Suppose $(\phi_l)_0$ is the level-set field prior to re-initialization. The following equation is solved to steady state to re-initialize the level-set field.

$$(\phi_l)_\tau + \vec{w}\cdot\nabla\phi_l = sign(\phi_l) \qquad 2.17$$

$$\vec{w} = sign((\phi_l)_0)\frac{\nabla(\phi_l)_0}{|\nabla(\phi_l)_0|}, \text{ where } \quad sign((\phi_l)_0) = \frac{(\phi_l)_0}{\sqrt{(\phi_l)_0^{\,2} + (\Delta x)^2}} \qquad 2.18$$

with the initial condition $\phi_l(\vec{x},0) = (\phi_l)_0(\vec{x})$.

The calculation of normal and curvature of the interface from the level-set field is simple. The normal is given by:

$$\vec{n} = \vec{\nabla}\phi_l \Big/ \left|\vec{\nabla}\phi_l\right| \qquad 2.19$$

and curvature is obtained from:

$$\kappa = -\vec{\nabla}.\vec{n} \qquad 2.20$$

Discretization of operators

Classification of grid points

The grid points on the Cartesian mesh are classified as *bulk points* and *interfacial points*. The latter are points that lie immediately adjacent to the immersed interface. Figure(a) illustrates an immersed interface and the points that are classified as *interfacial points*, i.e. points that satisfy the condition $(\phi_l)_{i,j}(\phi_l)_{nb} \le 0$, where *nb* denotes an immediate neighbor along the coordinate directions. The discrete operators at the interfacial points are different from those that apply at the bulk points. The strategy adopted to deal with this situation for the differential operators in the governing equation are discussed below.

Discretization at bulk points

A standard 5-point central-difference stencil applies for a typical *bulk point* (i.e. a grid point that does not adjoin the embedded interface) in a two-dimensional Cartesian mesh. While only 2-dimensional situations are shown in the figures for ease of visualization, the discussion below carries over to 3-dimensions. The three-dimensional counterpart would involve a 7-point stencil and the discretization for the momentum equations is identical in all the three dimensions.

The second derivative w.r.t. x in the diffusion term is discretized as follows:

$$\frac{\partial^2 \psi}{\partial x^2} = \frac{\psi_{i+1,j} - \psi_{i,j}}{\Delta x^2} - \frac{\psi_{i,j} - \psi_{i-1,j}}{\Delta x^2} \qquad 2.21$$

while the convection term in the x-direction is obtained from:

$$\frac{\partial u \psi}{\partial x} = \frac{u_{i+1/2,j}\psi_{i+1/2,j} - u_{i-1/2,j}\psi_{i-1/2,j}}{\Delta x} \qquad 2.22$$

where u is the x-component of the velocity and

$$\psi_{i\pm1/2,j} = \frac{\psi_{i\pm1,j} + \psi_{i,j}}{2} \qquad 2.23$$

$u_{i\pm1/2,j}$ are the velocities on the cell faces. Similar considerations apply along the y- and z-directions.

Discretization at interfacial points

As pointed out before, the main challenge in sharp interface fixed-grid methods is to accurately impose interfacial conditions in the discrete system of equations. Moreover, a sharp-interface method demands one-sided discretization for all the partial derivatives to avoid smearing of the interfaces. For the case of immersed solid-fluid boundaries, as in fluid-structure interaction problems, the no-slip and no-penetration velocity boundary conditions are applied on the solid surfaces. For small Strouhal numbers the Neumann condition for pressure applies on such boundaries (Udaykumar et al. 2002a). These boundary conditions are then supplied to the governing equations through the discretization at the interfacial points.

$\partial^2\psi/\partial x^2$ with a Dirichlet boundary condition on the embedded boundary.

Second derivatives need to be computed in the diffusion terms in the momentum as well as scalar transport equations and in these cases typically a Dirichlet condition applies at the boundary. With particular reference to the point (i, j) in Figure(c), this point

lies in the fluid and thus the velocities, scalars (temperature and species concentration) and pressure are computed at this point. Note that for points that lie in the solid only the temperature and species fields are computed. In discretizing $\partial^2\psi/\partial x^2$ at point (i, j) in Figure(c), the neighbor point (i+1,j) lies across the interface in the solid and hence cannot be used in discretization. In order to include the interfacial values (i.e. apply interfacial conditions), it is necessary to find the location where the 0-levelset intersects the line joining the cell-centers (indicated by the square symbols in Figure(c)). In the following expressions for the coefficients frequent use will be made of the quantity $\chi = \Delta x_I / \Delta x$ (see Figure(c)), where Δx_I is the distance between the cell center and the intersection of cell centerlines with the interface (filled square); Δx is the nominal cell width. By noting that the intersection point has a zero level-set value, χ can be easily evaluated using the level-set information at (i,j) & (i+1,j):

$$\chi = \frac{\Delta x_I}{\Delta x} \cong \left| \frac{(\phi_l)_{I_x} - (\phi_l)_{i,j}}{(\phi_l)_{i+1,j} - (\phi_l)_{i,j}} \right| = \left| \frac{0 - (\phi_l)_{i,j}}{(\phi_l)_{i+1,j} - (\phi_l)_{i,j}} \right| \qquad 2.24$$

Note that in evaluating the quantity χ a linear profile is assumed for the distance function between adjacent grid points. Higher-order approximations of the geometry can be implemented as well (Chopp 2001; Strain 2001).

The second-derivative can be estimated to second-order accuracy using the form:

$$\frac{\partial^2\psi}{\partial x^2} = \alpha_I \psi_I + \alpha_{i,j}\psi_{i,j} + \alpha_{i-1,j}\psi_{i-1,j} + \alpha_{i-2,j}\psi_{i-2,j} \qquad 2.25$$

where ψ_I is the value on the interface at the location I_{+x}(Figure(c)).

Using Taylor series expansions for each of $\psi_I, \psi_{i-1,j}, \psi_{i-2,j}$ about point (i,j) and further demanding that $\partial^2\psi/\partial x^2$ be estimated to $O(\Delta x^2)$ yields the following expressions for the coefficients.

$$\alpha_I = 6/(\chi(\chi+1)(\chi+2)\Delta x^2) \qquad 2.26$$

$$\alpha_{i-1,j} = (4-2\chi)/((\chi+1)\Delta x^2) \quad 2.27$$

$$\alpha_{i-2,j} = (\chi-1)/((\chi+2)\Delta x^2) \quad 2.28$$

$$\alpha_{i,j} = -\alpha_I - \alpha_{i-1,j} - \alpha_{i-2,j} \quad 2.29$$

In practice the above implementation may present difficulties due to the singular behavior of α_I as $\chi \to 0$. Therefore for small values of χ (< 0.01), i.e. when the interface is very close to the mesh point, the value χ is replaced by max(χ,0.01). This involves a slight perturbation of the boundary within a grid cell and decreases the order of accuracy locally from second-order. However this situation arises at only a few mesh points and the global accuracy is not impacted. Note that a first-order approximation is given by:

$$\alpha_I = 2/(\chi(1+\chi)\Delta x^2) \quad 2.30$$

$$\alpha_{i-1,j} = 2/((\chi+1)\Delta x^2) \quad 2.31$$

$$\alpha_{i,j} = -\alpha_I - \alpha_{i-1,j} \quad 2.32$$

$$\frac{\partial^2 \psi}{\partial x^2} = \frac{2}{\chi(1+\chi)}\frac{(\psi_I - \psi_{i,j})}{\Delta x^2} - \frac{2}{(1+\chi)}\frac{(\psi_{i,j} - \psi_{i-1,j})}{\Delta x^2} \quad 2.33$$

The singularity with respect to χ remains in this case as well. However, positivity of the off-diagonal coefficients is maintained in the first-order case while the second-order form will lead to a negative coefficient $\alpha_{i-2,j}$. In practice this negatively impacts the convergence of the iterative solver used for solving the discrete system of equations; to maintain robustness a first-order treatment for the diffusion term is employed at the interfacial points in the present calculations. While this practice lowers the order of approximation in the lower-dimensional set of interfacial cells, global second-order accuracy is still maintained as shown in the results.

Convection and divergence terms ($\partial(u\psi)/\partial x$ and $\partial\psi/\partial x$)
with Dirichlet conditions on the boundary.

As for the second-derivative terms above, the discretization scheme for $\partial(u\psi)/\partial x$ consists of contributions from points in the same phase. The differential operator for the convection term is obtained in the following form:

$$\frac{\partial u\psi}{\partial x} = \lambda_I (u\psi)_I + \lambda_{i-1/2,j} (u\psi)_{i-1/2,j} + \lambda_{i-3/2,j} (u\psi)_{i-3/2,j} \qquad 2.34$$

Note that, as in (Udaykumar et al. 2002b), to avoid pressure-velocity decoupling in the current collocated variable arrangement cell face velocities are also stored along with cell center velocities(Zang et al. 1994). These cell-face velocities are used in evaluating the convective fluxes in Eq 2.34. By employing Taylor expansions for each of the $\psi_{i',j'}$ in Eq 2.34 about point (i,j,k) and demanding an O(Δx^2) scheme the following expressions are obtained for the constants:

$$\lambda_{i-1/2,j} = (2\chi - 3)/\big((2\chi + 1)\Delta x\big) \qquad 2.35$$

$$\lambda_{i-3/2,j} = (1 - 2\chi)/((3 + 2\chi\Delta x) \qquad 2.36$$

$$\lambda_I = 8/((4\chi^2 + 8\chi + 3)\Delta x) \qquad 2.37$$

The general form and the coefficients for a first-order scheme in this case are:

$$\frac{\partial u\psi}{\partial x} = \lambda_I (u\psi)_I + \lambda_{i-1/2,j} (u\psi)_{i-1/2,j} \qquad 2.38$$

$$\lambda_{i-1/2,j} = -2/((1 + 2\chi)\Delta x) \qquad 2.39$$

$$\lambda_I = 2/((1 + 2\chi)\Delta x) \qquad 2.40$$

The above general form can be rewritten as follows:

$$\frac{\partial u\psi}{\partial x} = \frac{2}{(1+2\chi)} \frac{\big((u\psi)_I - (u\psi)_{i,j}\big)}{\Delta x} \qquad 2.41$$

Since the convection terms are explicitly computed the second-order approximation can be employed except at those points where two opposing interfaces

approach to within a mesh point. In this exigency the first-order approximation needs to be adopted.

$\partial^2\psi/\partial x^2$ operator with a Neumann boundary condition on the embedded boundary.

This situation arises in the pressure Poisson equation for points that adjoin the embedded boundary. The stability and accuracy of the flow solver depends critically on the construction of this term. In fact, devising a discrete form for the Laplace operator with a Neumann condition on the immersed interface proved to be the key to the robustness of the overall flow solver.

From Figure(c) it is evident that when discretizing the above operator the boundary condition that will apply at point I_{+x} is $\partial\psi/\partial n = 0$. It is not immediately clear how such a Neumann condition can be incorporated into the discrete form of the above operator. Apart from being crucial to the robustness of the overall method, the treatment of the pressure boundary condition is a key distinguishing feature of the finite-difference approach adopted here as opposed to the finite-volume approach detailed in previous work (Udaykumar et al. 2002b). In the latter a weak form of the pressure Poisson equation was employed, i.e.

$$\oint \frac{\partial p}{\partial n} dS = \oint \frac{\vec{u}^* \cdot \vec{n}}{\Delta t} dS \qquad 2.42$$

The interfacial cells were reshaped into irregular shaped cells where one of the cell edges coincided with the interface. Due to the weak form above the Neumann boundary condition for pressure is easily incorporated by setting the interfacial contribution to zero (i.e. $\partial p/\partial n = 0$) implicitly in the discrete pressure Poisson equation. However, since the strong form is employed in the present finite-difference scheme on a Cartesian grid, implicit imposition of a Neumann boundary condition on the pressure is not straight forward. The section below presents two different (an implicit and an

explicit) formulations to impose the interfacial condition. The implicit approach is chosen over the explicit approach due its superior efficiency while using iterative solvers.

Explicit approach for $\partial^2\psi/\partial x^2$ operator with a Neumann boundary condition on the embedded boundary.

This explicit method is a robust, albeit a first-order implementation of the Neumann boundary condition. Recall the Laplace operator assembled for the Dirichlet boundary condition case (i.e. Eq 2.25)

$$\frac{\partial^2\psi}{\partial x^2} = \alpha_I\psi_I + \alpha_{i,j}\psi_{i,j} + \alpha_{i-1,j}\psi_{i-1,j} + \alpha_{i-2,j}\psi_{i-2,j} \qquad 2.43$$

The above operator is also applicable in the current scenario, except that the interfacial pressure ψ_I in this case is found using the Neumann condition as follows. Looking at Figure(d) the interface pressure can be estimated by extending a normal from point I and placing two points distant Δx apart along the normal. The locations of the points I_x, $I1_x$ and $I2_x$ on the probe are therefore,

$$\vec{x}_{I_x} = \vec{x}_{i,j} + \Delta x_I \vec{i} \qquad 2.44$$

$$\vec{x}_{I1_x} = \vec{x}_{I_x} + \vec{N}_{I_x}\Delta x \qquad 2.45$$

$$\vec{x}_{I2_x} = \vec{x}_{I_x} + 2\vec{N}_{I_x}\Delta x \qquad 2.46$$

Similarly:

$$\vec{x}_{I_y} = \vec{x}_{i,j} + \Delta y_I \vec{j} \qquad 2.47$$

$$\vec{x}_{I1_y} = \vec{x}_{I_y} + \vec{N}_{I_y}\Delta x \qquad 2.48$$

$$\vec{x}_{I2_y} = \vec{x}_{I_y} + 2\vec{N}_{I_y}\Delta x \qquad 2.49$$

Fitting a quadratic to the pressure field along the normal erected and demanding that $\partial\psi/\partial n = 0$ at point I, one obtains:

$$\psi_{I_{x/y}} = \frac{4}{3}\psi_{I1_{x/y}} - \frac{1}{3}\psi_{I2_{x/y}} \qquad 2.50$$

In the above expressions the normal vector appears in several places. The normal at any point is easily obtained by bilinear interpolation from the values at the grid points. Note that the values of ψ at the points $I1_{x/y}$ and $I2_{x/y}$ are required in the above equation. These are calculated by bilinear interpolation from the surrounding mesh points. One issue that arises while evaluating the $\psi_{I1x/y}$ and $\psi_{I2x/y}$ is that for certain interface orientations the bilinear interpolation may involve a point that lies in the solid. Thus, a single layer of ghost values of pressure are computed and stored at the points in the interfacial points in the solid. The ghost values of pressure are also obtained with the condition that the Neumann condition applies at the interface. Thus, a normal to the interface is erected from the ghost point as shown in Figure(d) where the locations of the points on the normal are:

$$\vec{x}_{G1} = \vec{x}_G + \vec{N}_G|\phi_G| \qquad 2.51$$

$$\vec{x}_{G2} = \vec{x}_{G1} + \vec{N}_G\Delta x \qquad 2.52$$

The normal at point G is obtained from the level-set field using Eq 2.19. Fitting a quadratic to the pressure field along the normal and demanding that $\partial\psi/\partial n = 0$ be satisfied at the point G1 (i.e. on the solid boundary) leads to:

$$\psi_G = \frac{\psi_{G1}d_2^2 - 2\psi_{G1}d_1d_2 + \psi_{G2}d_1^2}{d_2^2 - 2d_1d_2 + d_1^2} \qquad 2.53$$

where,

$$d_1 = |(\phi_l)_G| \text{ and } d_2 = |(\phi_l)_G| + \Delta x \qquad 2.54$$

The interfacial value of the pressure ψ_{G1} is obtained from Eq 2.53 above and the value at G2 is obtained by bilinear interpolation. Note that the values of the ghost

pressure and the interfacial pressure become inter-dependent through Eq 2.51 and Eq. 2.53 The determination of the interfacial pressure and ghost pressure are embedded within an iterative solver for the pressure. The $(k+1)^{th}$ iteration of the solver can be expressed in matrix form as follows:

$$A_P \psi^{k+1} = b_P + b_I^k \qquad 2.55$$

where, A_P is the coefficient matrix assembled from the Laplace operator for bulk and interfacial cells (Eq. 2.21 and Eq. 2.43), b_P is the source term for pressure (Eq. 2.13), ψ^{k+1} is the solution at $(k+1)^{th}$ iteration and $b_I^k (= \alpha_I \psi_I^k)$ is the interfacial term in Eq. 2.43 calculated from the solution at k^{th} iteration. The interfacial pressure and ghost pressure are embedded in b_I^k. During the iterative process this term is updated every iteration till convergence is attained. A converged solution also effectively imposes Neumann condition on the embedded interface.

This explicit scheme has been tested and applied to an entire range of Reynolds numbers and flow problems. However, there are certain drawbacks in the scheme that make it computationally unattractive. Firstly, the re-evaluation of the term b_I^k in Eq. 2.54 incurs extra computations during the iteration process. These additional calculations could be significant for large systems, especially in 3D. Secondly, altering the right hand side of a matrix system introduces new error in the residual, thereby, partly nullifying the reduction in residual due to the solver. This effect is especially significant in the initial stages of the iterative process. A comparison of convergence behavior of this approach versus an implicit approach is presented in the following section.

Implicit approach for $\partial^2\psi/\partial x^2$ operator with a Neumann boundary condition on the embedded boundary.

An implicit scheme is devised to incorporate the Neumann boundary condition into the discrete Laplace operator by expanding the stencil beyond the immediate neighbors. This procedure also uses Eq 2.24 as the starting point.

$$\frac{\partial^2 \psi}{\partial x^2} = \alpha_I \psi_I + \alpha_{i,j}\psi_{i,j} + \alpha_{i-1,j}\psi_{i-1,j} + \alpha_{i-2,j}\psi_{i-2,j} \qquad 2.56$$

In the above equation, the interface pressure, ψ_I is expressed in terms of other computational cells in the vicinity of *(i,j)*. The implicit formulation for the linear operator in Eq. 2.55 is illustrated for the point P in the configuration shown in Figure 2.3. Looking at Figure 2.3 the interface pressure ψ_I at I_x can be estimated by extending a normal from point I_x and placing two points $I1_x$ and $I2_x$ along the normal. Unlike, the explicit method, the chosen two points are the intersections of the normal at I_x with lines joining cell centers. The location of the points $I1_x$ and $I2_x$ in the Figure 2.3 can be calculated as the intersection of the normal from I_x with lines N-NE and N-NN respectively. Assuming a quadratic pressure profile between the points I_x, $I1_x$, $I2_x$ with zero normal gradient at the interface, the interfacial pressure can be expressed as,

$$\psi_{I_x} = \frac{\psi_{Ii_x} d_2^2 - \psi_{I2_x} d_1^2}{d_2^2 - d_1^2} \qquad 2.57$$

where, d_1 and d_2 are the distances of the points $I1_x$ and $I2_x$ from I_x respectively. Since points $I1_x$ and $I2_x$ lie on cell-center lines, their pressure values can be interpolated from the cell-centers straddling them. Let $\chi_1 = (N, I1_x)/(N, NE)$ and $\chi_2 = (N, I2_x)/(N, NN)$ (operator (A, B) representing the distance between points A and B) be the interpolation weights for the straddling points for $I1_x$ and $I2_x$. The pressure values at $I1_x$ and $I2_x$ can be expressed as:

$$\psi_{I1_x} = (1-\chi_1)\psi_N + \chi_1 \psi_{NE} \qquad 2.58$$

$$\psi_{I2_x} = (1-\chi_2)\psi_N + \chi_2 \psi_{NN} \qquad 2.59$$

Using Eq. 2.58 and Eq 2.59 and substituting Eq 2.57 in Eq. 2.56 Laplace operator can be rewritten as :

$$\frac{\partial^2 \psi}{\partial x^2} = \alpha_{i,j}\psi_{i,j} + \alpha_{i-1,j}\psi_{i-1,j} + \alpha_{i-2,j}\psi_{i-2,j} + \alpha_{i,j+1}\psi_{i,j+1} + \alpha_{i,j+2}\psi_{i,j+2} + \alpha_{i+1,j}\psi_{i+1,j} \qquad 2.60$$

The last three terms in the above expression (corresponding to N, NN and NE in Figure 2.3) are the consequence of the implicit imposition of the interfacial boundary condition. This operator when assembled for all the interfacial cells and when solved iteratively for the whole domain results in a converged solution that satisfies the interfacial boundary condition.

The above procedure is applicable to any interfacial orientation. The additional points included in the expanded stencil vary based on the orientation of the interfacial normal. This scheme has also been implemented for three-dimensional configurations. In 3D, the points $I1_x$ and $I2_x$ lie on planes connected by cell-centers in the vicinity of point (*i*,*j*). The 3D counterparts of Eq 2.58-2.59 involve bi-linear interpolation of the four points defining the intersecting planes (1-4 for $I1_x$ and 5-8 for $I2_x$).

$$\psi_{I1_x} = \beta_1\psi_1 + \beta_2\psi_2 + \beta_3\psi_3 + \beta_4\psi_4 \quad \sum_{i=1}^{4}\beta_i = 1 \qquad 2.61$$

$$\psi_{I2_x} = \beta_5\psi_5 + \beta_6\psi_6 + \beta_7\psi_7 + \beta_8\psi_8 \quad \sum_{i=5}^{8}\beta_i = 1 \qquad 2.62$$

There are several advantages of adopting the implicit approach over the explicit scheme described earlier. a) The implicit scheme generates a linear system with a fixed coefficient matrix and a constant right-hand side, thereby bypasses the issue of re-calculating the interfacial contribution during each step of the iterative process. b) In the explicit method, as the high frequency errors are being reduced by the smoother, the contribution due the interfacial term can introduce a significant addition to the net residual of the matrix system, thereby reducing the rate of convergence. However, in

using the implicit method, given the matrix system, the rate of reduction of the high frequency errors is nearly unaltered as the solution attains convergence. c) Even though the implicit scheme extends the stencil beyond the standard 5-point stencil (7-point in 3D) the coefficient matrix is still predominantly sparse, enabling the use of fast Krylov based solvers. Moreover, the computational gains due to solver efficiency justify the use of implicit formulation of Laplace operator near the interface. A comparison of the convergence histories of the implicit and explicit schemes shown in Figure 2.5 further strengthen this observation. The convergence histories shown in the Figure 2.5 are for a two-dimensional channel flow with imposed negative pressure gradient across the open ends of the channel. The channel walls are represented by solid interfaces with Neumann boundary condition on the pressure. The Neumann condition is incorporated in the solution of pressure Poisson equation using both the explicit and implicit schemes. The iterative solver used in this context is Algebraic Multigrid with four levels of multi-grid. The impact of using an explicit scheme is apparent from the convergence behavior of AMG. The slope of the convergence curve in semi-log plot is reduced drastically in case of an explicit scheme. The implicit scheme reduces the residuals to the order 10^{-10} while the explicit scheme nearly stagnates at 10^{-3}.

Moving boundaries

In Eulerian sharp interface methods, when the solid boundary moves across a grid point, the state of the point can change from liquid to solid or vice versa. Different approaches have been employed to handle this situation. In Ghost-Fluid type methods and immersed boundary methods (Fadlun et al. 2000; Kim et al. 2001; Liu et al. 2000) or fictitious domain methods (Glowinski et al. 2001) flowfields are computed within as well as outside the immersed solid object. Thus, when the boundary crosses over a grid point, changing the state from solid to fluid, the newly emerged fluid point simply takes on the

flowfield variables that were available at that point in the previous time step. In the sharp interface method (Udaykumar et al. 2002b) as well as immersed interface method (Leveque and Li 1994; Leveque and Li 1995) where the flow is computed separately in each subdomain (fluid and solid) separated by the interface and no ghost flowfield exists in the solid, a scheme must be devised to obtain the flowfield variables at the newly emerged fluid point. Note that the converse case, i.e. the emergence of a grid point that was in the fluid phase into the solid phase presents no issues since the flowfield is not computed in the solid phase.

A newly emerged fluid grid point is defined by the condition $(\phi_l)_{i,j}^{n+1}(\phi_l)_{i,j}^{n} < 0$. Since the point was previously in the solid phase ($(\phi_l)_{i,j}^{n} < 0$) it had no history in the fluid phase ($(\phi_l)_{i,j}^{n+1} > 0$), i.e. $\vec{u}^n$ (as also ξ^n) does not exist in the fluid phase for such a point. Therefore, these points are to be evolved to time level n+1 in a special fashion. Note that since the pressure Poisson equation does not have a time-dependent term the pressure in such a cell can be evaluated as usual once a $\vec{u}*$ value is available after solving the momentum equation. The method to obtain $\vec{u}^n$ (and ξ^n) for such points follows along the lines detailed in (Udaykumar et al. 2002b; Udaykumar et al. 2001) and is analogous to the approach taken in moving grid formulations when a fresh grid point is inserted following mesh refinement. The value at such points is obtained by interpolation from the known values in the surrounding cells and on the moving boundary (where the boundary conditions are specified). For the particular time step when a grid point changes from solid to fluid phase, the value of $\vec{u}^n$ there is found using a linear interpolation operator spanning points in the fluid and on the interface. The interpolation points that are picked depend on the orientation of the interface in the cell as illustrated in Figure 2.4. For the particular case in Figure 2.4, the value at the freshly cleared cell (i,j) is calculated as $\psi_{i,j}^{n} = (\chi_{-y}\psi_{i,j+1}^{n} + \psi_{I_{-y}}^{n})/(1+\chi_{-y})$, where χ_{-y} is the distance between the grid point (i,j) and the interfacial point I_{-y}. The interpolation points are chosen depending on the

direction of the normal vector at I_y ($\vec{n} = (n_x, n_y)$) and the ratio $n_y/n_{x,}$. For instance, in the above expression for the case in Figure 2.4, points I_y and (i,j+1) are chosen since ($n_y>0$ & $n_y>n_x$).Consistent with the difference scheme in the interface-adjacent grid points the treatment at the newly emerged fluid points is first-order accurate. Note that this procedure is equivalent, in analogy with purely Lagrangian (moving grid) methods, to interpolating the value of variables to a newly inserted point after mesh refinement from values at the old mesh points (i.e. before refinement). In diffuse interface Eulerian methods (where interfaces may be captured using VOF, level-set, phase field etc.), where the interfacial forces are spread over the mesh (Anderson et al. 2000; Brackbill et al. 1992) this issue of cross-over does not arise since there is no clear-cut interface location and all properties are taken to vary smoothly over a few mesh points.

Iterative Solvers for the Sparse Linear Systems

In any CFD tool, most of the computational time is spent in the solving the linear matrix systems resulting from the momentum equation and more importantly the pressure Poisson equation. Typically the discrete finite-difference operators lead to sparse linear systems for any multi-dimensional problem. A variety of iterative solvers such as Point-Jacobi, Multigrid methods such as AMG, Krylov subspace based methods such as Conjugate Gradients/ GMRES etc are available.

Among the above, AMG has been implemented and has been successfully used for a variety of moving boundary problems in ELAFINT3D framework. However, the following issues(explained best in the context of a channel flow) have prompted a review of the various solvers to identify the best solver for the current class of problems. Consider a 2D channel flow with a prescribed time-varying pressure gradient across the channel. In this scenario, the varying inlet pressure has to be communicated through out

the channel by solving the elliptic pressure Poisson equation. The newly solved pressure in general reflects the change in the inlet pressure. However, with AMG, registering of even small variations in the boundary conditions takes large number of iterations in comparison to fixed boundary condition. Moreover, even with the implementation of implicit scheme for the interfacial Laplace operator, the convergence of AMG could do better. In this view, a comparison of the point solver, line solver, AMG and BiCGSTAB*(l)* has been performed for a set of typical fluid flow problems. The details of the BiCGSTAB*(l)* solver used in this are presented in papers by Sleijpen and van der Vorst. Though not described in this thesis, the convergence behavior of GMRES has been found to be slower than BiCGSTAB*(l)*.

The cases chosen for this study are flow in a 2D channel and flow around a stationary cylinder. Figure 2.6 shows the convergence behavior for all the tested solvers for flow through a slanted channel with prescribed pressure gradients. As can be observed, the BiCGSTAB*(l)* converges faster than all the other solvers for both implicit and explicit schemes on the interface. Figure 2.6 compares the convergence behavior of all the solvers for flow around a cylinder. This case also illustrates the performance of the implicit scheme for all possible orientations of the interface.

Local Mesh Refinement

Many of the fluid flow problems entail fascinating physics in disparate length and time scales. In order to simulate these multi-scale problems, one requires appropriate resolution in spatial and temporal domains. As described previously, the ELAFINT3D framework relies on fixed Cartesian meshes to represent the flow field. Any embedded entity cuts across the fixed mesh as it interacts and traverses in the fluid medium. In the

context of the Cartesian grid methods, enhancing the spatial resolution in a certain region of the domain will result in increased resolution in other parts of the domain even if not required. This scheme of increasing the spatial resolution results in enormous computational cost and memory requirements for the simulation. This draw back of the fixed grid methods can be surpassed by adopting an adaptive mesh refinement strategy. The current ELAFINT3D framework features a quad/oct-tree based Local Mesh Refinement Algorithm to adaptively generate spatial resolution based on the flow features.

The critical features of the LMR algorithm are a) Refinement Criteria b) Mesh Data Structure and c) Discretization scheme for the governing equations. The above aspects as implemented in the current framework will be discussed.

Refinement Criteria

The selection of refinement criterion is mostly application specific. However, the problems of current interest being moving boundary problems, the criteria for refinement are broadly based on a) the proximity to any embedded entity and b) the gradients of solution variables(e.g vorticity). The refinement criteria used in the current work are: a) uniform refinement upto a certain distance from the interface and b) a solution based refinement based on the following equations

$$\frac{h|\nabla \phi|}{\max|\phi|} > \varepsilon \qquad 2.63$$

$$\max\left(\alpha_g \frac{\nabla\psi}{\psi_0}h, \alpha_c \frac{\nabla^2\psi}{\psi_0}h^2\right) > \varepsilon \qquad 2.64$$

where ε is the refinement criterion and $h \sim O(\Delta x)$

Data Structures

The LMR algorithm relies on quad-tree (oct-tree in 3D) data structures to store the refined meshes. This strategy provides an easy way to perform coarsening and refinement operations and also enables easy access to the immediate neighbors. In the present framework, the information pertaining to each cell is stored in a used defined data structure. This data structure comprises of the location of the cell, its position in the tree hierarchy and the physical variables (e.g velocity, pressure etc) associated with it. Based on the inputs, the flow domain is initially represented by a base mesh. The base mesh is typically coarse in comparison to the eventually refined mesh. Each cell of the base mesh is at the top of a tree. Based on the refinement criteria, each of the base meshes is either refined or not refined. If a certain cell (say A) is decided to be refined, the child cells (A1-A4) are created for this parent cell (A). Each of these child cells is assigned indices that indicate their position in the tree hierarchy of the cell A. Also pointers to the child and parent cells are assigned to each of the cells. Using logical operations, the neighbors of the child cells can be determined and linked using pointers. To avoid implentational complexities, especially in discretization schemes, the refined grid is smoothed to assure that no two neighboring cells differ by more than one level.

Discretization

The finite-difference schemes for the governing equations introduced earlier are based on meshes with uniform refinement. The challenge in devising a discretization scheme is that of mass conservation across the mesh-interfaces. In the current implementation of LMR, a mixed finite-difference and finite-volume scheme is used for discretization. The computational domain can be classified into three distinct categories. The "normal cells" which are the bulk cells with away from the refinement boundaries, the "ref-boundary cells" which are adjacent to the boundary between different levels of refinement and the "interfacial cells" which are right next to the interface. For the "normal" cells, the finite-difference and finite-volume schemes result in the same discrete form. For the "interfacial cells", the finite-difference schemes introduced in previous sections are applicable as the region up to a certain distance from the interface is refined to the same level. For the "ref-boundary cells", special attention is required in devising discretization schemes that ensure flux conservation across the mesh interfaces. A finite-volume method is implemented in the current framework for the "ref-boundary cells".

As described in earlier, a smoothing of the refined meshes in performed to ensure that any pair of neighboring cells differ by utmost one level of refinement. Based on the smoothing, there are two possible configurations at the mesh-interfaces. The neighboring cell is a) one level higher or b) one level lower than the current cell.

Neighbor cell higher than the current cell

The discretization of the diffusion operation is shown in Eq. (2.62). f represents the flux across the side. The 'north' cell 'N' is a divided cell while the current cell 'P' is

undivided.

$$\nabla^2\phi = \frac{(f_e - f_w)}{\Delta x} + \frac{(f_n - f_s)}{\Delta y} \qquad 2.65$$
$$where \;\; f_n = f_{n1} + f_{n2}$$

The flux from each of the north faces into the current cell is given by Eqn(2.63). $\phi_{P1'}$ and $\phi_{P2'}$ are ghost cells, which are interpolated from the surrounding cells by bi-linear weighted averaging. Finally, the stencil for $\nabla^2\phi_P$ includes all the surrounding neighboring cells as shown in Eqn(2.64).

$$f_{n1} = \frac{\phi_{N2} - \phi_{P2'}}{0.5\Delta y}(0.5\Delta x)$$
$$where \;\; \phi_{P2'} = f(\phi_P, \phi_E, \phi_{NE}, \phi_{N_0}) \qquad 2.66$$
$$\phi_{N_0} = f(\phi_{N1}, \phi_{N2}, \phi_{N3}, \phi_{N4})$$
$$Similarly \;\; f_{n2}$$

$$\nabla^2\phi_P = f(\phi_P, \phi_E, \phi_W, \phi_S, \phi_{NW}, \phi_{N1}, \phi_{N2}, \phi_{N3}, \phi_{N4}) \qquad 2.67$$

Neighbor level lower than current cell

This is the case where the neighboring cell is a level lower than the current cell under consideration. The discretization of the diffusion operator is shown in Eqn(2.65). This time the cells to the east and south, represented by ‘E’ and ‘S’ respectively are a level lower than the current cell ‘P’. The flux from the east and south faces are given by equation(2.65).

$$f_e = \frac{\phi_{E'} - \phi_P}{0.5\Delta y}(0.5\Delta x)$$
$$f_s = \frac{\phi_P - \phi_{S'}}{0.5\Delta y}(0.5\Delta x) \qquad 2.68$$

where $\phi_{E'}$ *and* $\phi_{S'}$ are ghost points obtained by interpolating from surrounding cells as shown in equation 58(b).

$$\phi_{E'} = f(\phi_{P_0}, \phi_E, \phi_{SE}, \phi_S)$$
$$\phi_{P_0} = f(\phi_P, \phi_N, \phi_W, \phi_{NW}) \qquad 2.69$$

Finally the diffusion operator is obtained as a function of all the surrounding cells as

$$\nabla^2 \phi_P = f(\phi_P, \phi_E, \phi_W, \phi_N, \phi_S, \phi_{NW}, \phi_{SW}) \qquad 2.70$$

Results and Discussion

Finite-difference and Finite-volume approximations

The accuracy of the current finite-difference and level-set combination (FD+LS) is estimated and compared to that of a finite-volume method that used marker tracking (FV+MT)(Udaykumar et al. 2002b). The main differences between the present FD+LS method and the FV+MT method lie in the manner in which the sharp-interface treatments are constructed in the interface-adjacent grid points. In the FV+MT method a cut-cell technique was used to reshape the control volumes at such points and the weak form of

the governing equations was discretized. The convective and diffusive fluxes at control volume faces were obtained from a nominally second-order accurate linear-quadratic shape-function that incorporated points additional to the usual 5-point (in 2D) stencil in the discretization at interface-adjacent points. The present FD+LS method operates with the strong form of the governing equations and the diffusion term is approximated to first-order accuracy at the interfacial points. However, the present FD+LS scheme is far easier to implement and makes extension to 3-dimensions straightforward.

The solutions to scalar diffusion and scalar convection-diffusion problems are chosen as a basis for comparison in this section. The computational setup for all of these cases is a circular cylinder of radius 0.25 placed at the center of a 1 x 1 domain as shown in Figure (a). For the diffusion problem, Dirichlet boundary conditions are applied at the cylinder surface (ξ=0),the left and bottom boundaries of the domain (ξ=1) and the right and top boundaries(ξ=0). The system is allowed to reach steady state and the L_2-norm of error in temperature over the whole domain is calculated. Both FD+LS and FV+MT methods are used to solve this problem. Variation of the L_2 norm with grid spacing using the two methods is compared in Figure (b) by taking the solution on the finest mesh (200x200) to be the exact solution. The figure shows that the present FD+LS solution compares well with the previous FV+MT method in terms of the order of accuracy. Both methods provide globally second-order accurate solutions although the errors in the finite difference case are somewhat higher in magnitude than the FV+MT case.

The scalar convection-diffusion equation is solved next in the same setup as for the above case but a uniform flow (u = 1.0, v= 1.0) is imposed everywhere in the domain. As in the previous case, both FD+LS and FV+MT methods are used to solve the convection-diffusion equation. The variation of L_2 error norms with grid spacing for the solution at steady state for each of these methods is plotted in Figure (c). The slopes of

the lines indicate that both methods yield second-order accurate results. These accuracy studies indicate that the present FD+LS method compares well with the finite-volume sharp interface method while carrying the advantage of simplicity of construction.

Benchmarking the Flow Solver

Two-dimensional flows

Case I. Flow past a stationary circular cylinder This study validates the current FD+LS method for flow around immersed stationary boundaries over a range of Reynolds numbers by simulating steady and unsteady flows past a circular cylinder immersed in an unbounded flow. These 2D cases are used to directly compare the current FD+LS technique with experimental data as well as benchmarked numerical results, including the previous FV+MT technique (Udaykumar et al. 2002b). The flow around a cylinder for Reynolds number (Re) = 40, 80, and 300 are simulated and the results are compared to various numerical and experimental results. A large domain size of 30 x 30 with computational mesh of 452x452 has been used to minimize the boundary effects and to resolve the boundary layer on the embedded solid. The grid is fine and uniform in the vicinity of the immersed boundary and is stretched linearly away from it. The boundary conditions on the top, bottom and the left boundaries correspond to potential flow past a cylinder and the right boundary is specified as an outlet.

For Re = 40 the flow develops to a steady-state with a steady recirculation zone behind the cylinder as shown in Figure (a). The length of the recirculation and the drag coefficient, $C_D = F_D/(1/2)\rho U_o^2 D$ (F_D being the drag force), are computed for comparison with benchmark results. Table 2.1 shows that the results from the current FD+LS method compare well with other experimental and numerical solutions. At Re = 80, the wake of the cylinder develops into the classic Karman vortex street as shown in Figure (b). The

steady vortex shedding is illustrated from the plot of drag and lift coefficients shown in Figure (c). The mean drag coefficient computed is 1.37 which is in agreement with the previous FV+MT results. The Strouhal number ($St = fD/U_o$, where f is the shedding frequency) for the vortex shedding is obtained to be 0.15. Figure (a) and 2.9(b) show the streamlines and pressure contours at an instant of time for Re = 300. The variation of drag and lift coefficients are shown in Figure (c). The values of mean drag coefficient C_D=1.28 and St = 0.21 agree well with the experimental value of (Wieselsberger 1922) as well as previous FV+MT results. Thus, for flows with embedded boundaries the current FD+LS method yields results that are indistinguishable from the FV+MT calculations.

Case II. Flow past an oscillating cylinder To validate the capability of handling flows past moving boundaries the flow past a transversely oscillating cylinder and the classical "lock-on" phenomenon for vortex shedding is examined. Flow past a cylinder at Re = 200 undergoing sinusoidal transverse oscillation is simulated for a range of oscillation frequencies and the results are compared to the experimental results (Koopmann 1967).

The cylinder center (x_o,y_o) is located at (8, 10) relative to the left bottom corner in a 20x20 domain with 400x400 mesh points. A uniform flow U_o is prescribed on the left, top and bottom boundaries and the right boundary has an outlet boundary condition. A sinusoidal motion of the form $x_c(t) = x_o$, $y_c(t) = y_o + A\sin(2\pi f_f t)$ is imposed on the cylinder, where t is the non-dimensional time, A is the amplitude of oscillation, f_f is the frequency of the imposed oscillation.

The unsteady flow past a stationary cylinder at Re = 200 is used as a starting solution before proceeding to oscillate the cylinder. A vortex-shedding frequency of f_o = 0.198 and drag coefficient of C_D = 1.37 are calculated for this flow. With the solution

from the above simulation as an initial condition, flow past an oscillating cylinder for a range of oscillation amplitudes and frequencies has been simulated. The sets of parameters chosen to study the lock-on phenomenon are: a) A= 0.1, f_f = {0.15, 0.17,0.19,0.2,0.21,0.23,0.25} and b) A = 0.2, f_f = {0.15, 0.17,0.19,0.21}. Each of the simulations was performed for about 200 non-dimensional time units to ensure that the flow has attained a quasi-steady state. Shown in Figure (a) are the vortex shedding patterns behind a cylinder oscillating with amplitude of 0.1 and frequency of 0.21. The non-dimensional shedding frequency (Strouhal number) is calculated by evaluating the periodicity of the velocity fluctuations in the near wake region. The variation of velocity at a probe point in the wake for the above mentioned case is shown in Figure (b). The deduced shedding frequency from this plot is 0.21, which implies that the vortices shed by the cylinder are "locked-on" to the cylinder oscillation. Based on experiments, Koopmann (1967) obtained a curve in amplitude vs frequency parameter space that demarcates the lock-on region from non-lock-on regions. The results from the present study are benchmarked by verifying the lock-on prediction against Koopmann's experimental curve. The current results are compared with the experimental curve in Figure (c) and show good agreement. The current simulations predict lock-on at frequencies close to the natural frequency. The points where lock-on was observed (filled-squares) fall within the experimental lock-on regime (region between the solid lines in Figure 2.10(c)) thereby validating the current technique.

Three - dimensional flows

Case I. Flow around a stationary sphere Simulation of flow around a stationary sphere is used to validate the numerical method for problems involving three-dimensional fixed immersed boundaries. Benchmark numerical solutions and experimental data are used to validate the solutions. There are distinct regimes in the flow around a sphere that

can be identified based on the Reynolds number of the imposed flow(Johnson and Patel 1999). These include the transition from steady axisymmetric to non-axisymmetric flow (at Re ≅ 210), followed by transition from steady to unsteady flow (at Re≅ 270). Laminar flows up to Re = 300 are simulated and compared to benchmark data. The transition points as well as quantitative measures such as recirculation length, drag coefficient and Strouhal number (for unsteady flows) are used to validate the current method. A 15x15x15 domain with the sphere centered at (x=5, y=7.5, z=7.5) is used for the simulations in the current study. A computational grid consisting of 130 x 110 x 110 mesh points with fine and uniform grid in the vicinity of the sphere and linearly stretched mesh away from the sphere is used. Computational resources limit the domain and mesh size in 3D transient calculations. The domain and mesh size used are deemed sufficient for the present study based on the agreement between the present results and benchmarks. The boundary conditions on all the boundaries correspond to potential flow past a sphere.

Steady Axisymmetric Flow: The flow around a sphere is steady and separated for Reynolds numbers in the range 20-210 (Johnson and Patel 1999).The flows for Re starting from 50 are simulated. Figure (a-d) illustrate the streamlines, through the symmetry plane (z =7.5) plane for Reynolds numbers of 50,100,150 and 210 respectively. As expected, the flow has a steady, axisymmetric wake in all these cases. For these Reynolds numbers, the quantitative data, *viz.* length of the recirculation bubble and the drag coefficient tabulated in Table 2.2 compare well with the published data in the literature.

Steady non-axisymmetric flow: It has been experimentally established (Magarvey and Bishop 1961) that a non-axisymmetric steady flow regime prevails in the range 210<Re< 270. Numerical results (Johnson and Patel 1999) indicate the value for onset of the asymmetry to be close to Re= 215. The onset of this flow regime has been verified by

simulating flows for Re = 210 and Re = 215. The plots in Figure (d-e) show the streamlines and the pressure contours on z = 7.5 plane for a Re = 210 and Re = 215 respectively. At Re=215 the flow field indicates an incipient axial asymmetry which is highlighted by the circled region in Figure (e). This establishes the transition to asymmetric flow in this Re interval.

Unsteady flow: The plots in Figure 2.12(a-c) show the flow features on the z = 7.5 plane at Re = 270. As can be observed from the vorticity contours in this graph, the flow is still steady in this case. Since the transition to unsteady flow is known to occur in 270<Re<300 range (Magarvey and Bishop 1961; Tomboulides et al. 1993), flows are computed within this range by varying Re in intervals of 10. A steady solution from Re = 270 is employed as an initial condition for all the test simulations. The vorticity contours on various planes shown, for Re=280, in Figure (d-f) reflect the unsteadiness in the flow at that Reynolds number. It can also be observed that the vorticity patterns are similar to those at Re=300(shown in Figure) where the flow is truly unsteady.

Calculations for Re = 300 were carried out to observe the well-established vortex shedding phenomenon and also to obtain quantitative data for comparison with benchmarks. The vorticity contours on the x-y and x-z planes are shown in Figure . Several techniques can be used for identifying and extracting vortical structures in 3D. The Δ-method proposed by (Chong et al. 1990) has been used for visualizing the vortical structures in the present investigation. The above method defines the vortex core as a region with closed or spiral local streamline pattern. Such a region is identified by complex eigenvalues of velocity gradient tensor ∇u. The iso-surfaces of complex eigenvalues of ∇u are plotted in Figure . The structure of the vortices shed from the sphere resemble the numerical results in (Johnson and Patel 1999). For quantitative comparison, the time variation of lift and drag coefficients are plotted in Figure . The

calculated average drag coefficient of 0.621 as well as the Strouhal number of 0.133 compare well with results of (Johnson and Patel 1999) (Table 2.2).

Case II. Flow around a moving sphere Flow around a sphere undergoing stream-wise oscillations in a stably stratified fluid has been studied to validate the current numerical method for three-dimensional flows involving moving embedded objects. The computational setup is the same as in the stationary sphere cases presented in the previous section. The sphere undergoes forced oscillation in the stream-wise direction, $U(t) = U_1 \cos f_f t$, where U_1 and f_f are the amplitude and frequency of oscillation. The governing equations for the flow are the same as discussed in §2.2. The fluid is stably stratified with an undisturbed linear density gradient in the z-direction, $\rho_B(z) = \rho_{H/2} + (\Delta\rho/2)(1 - 2z/H)$, where $\rho_{H/2}$ is the density at the mid-depth and $\Delta\rho$ is the density difference between the bottom(z=0) and top (z=H) planes of the domain. The terms involving density variations assume significance due to stratification. The density field is computed using incompressibility condition. The presence of the oscillating sphere introduces additional non-dimensional quantities, *viz.,* the Keulegan-Carpenter number ($KC = U_1/f_f D$), which characterizes the non-dimensional amplitude of the oscillations, and normalized forcing frequency ($S_f = f_f D/(2\pi U_0)$).

Stratified flows display a range of interesting phenomena usually not found in the homogeneous flow situations. For flow around a sphere, (Lin et al. 1992) have shown experimentally that for sufficiently small Fr or strong stratification, the incoming flow prefers to go around the sphere rather than over it. For Fr < 0.2, the flow is quasi-two-dimensional. The stratification suppresses the formation of hairpin vortices at higher Reynolds numbers. For sufficiently large Re, vortex shedding in the lee of the sphere resembles the classic Karman vortex street observed for uniform flow past a circular cylinder and the flow becomes nearly two-dimensional in -0.5<z/D<0.5. The wide range

of experimental results reported by (Lin et al. 1994; Lin et al. 1992) for these parameter values serve for validation. The parameter values chosen for the current study are Re = 190 and Fr = 0.07. (Lin et al. 1994; Lin et al. 1992) created a flow regime diagram in KC-S_f parameter space. The test points chosen for the current study are labeled A(KC=0.03, S_f =0.35) and B(KC=0.2, S_f =0.35). Experimental results indicate that test points A and B fall into the natural vortex shedding and lock-on alternate single vortex regimes respectively. The simulations of flow around an oscillating sphere are performed to test of the flow behavior predicted agrees with the experimental flow regime diagram.

The flow around a stationary sphere at Re = 190 and Fr = 0.07 is first simulated and the solution is used as an initial condition for the moving sphere cases. The vortex structures in the lee of the sphere resemble the classical Karman vortex street observed in the case of a circular cylinder, as seen from the vorticity contours on the central z = 7.5 plane shown in Figure 2.16(a). The contours of the z-component of velocity and the density on the z = 7.5 plane are plotted in Figure 2.16(b) & (c). The small variations in the plane in these plots illustrates the essentially quasi-two dimensional nature of this flow. A Strouhal number of 0.2 calculated from the periodicity of the velocity for a probe point in the lee of the sphere (Figure) matches with experimental value reported in the literature (Lin et al. 1992).

The quasi-steady state solution of the above simulation is used as the starting condition for the oscillating sphere cases. For the first test point A, the amplitude of oscillation (KC = 0.03) is small, so that the far wake shedding behavior is similar to that of the stationary case. The Strouhal number, calculated to be 0.2, is unchanged from the natural shedding frequency of the stationary sphere. The flow being similar to the stationary case, this flow regime is classified as being in the “natural vortex shedding” regime.

For test point B, at a higher KC of 0.2, z-vorticity contours are plotted in Figure . However, the vortical structures are better visualized using the Δ-method (as described in the previously) and plotted in Figure 2.1. The cylindrical vortices apparent in the x-y view shown in Figure 2.1(b) illustrate that the vortex shedding is self similar in z-direction. This demonstrates the quasi-two-dimensionality imposed due to stratification. The x-z view of vortex shedding shown in Figure 2.1(c) supports the above observation. The Strouhal number calculated from the probe velocity variation with time (Figure) is 0.35 which indicates that the shedding under these conditions locks on with the oscillation frequency of the sphere. Figure 2.1 also illustrates that the vortices are being shed alternately from either side of the sphere. Hence this flow regime is named "lock-on alternate single vortex" regime(Lin et al. 1994). The flow structures viewed from the various perspectives are visually identical to those obtained in the experiments of(Lin et al. 1994). In summary, flow around an oscillating sphere in the presence of density stratification has been simulated for selected set of parameters KC & S_f. The flow regimes, vortical patterns and shedding frequencies predicted by the current technique for these test points are in agreement with the experimental results reported by Lin & coworkers and thereby provide validation for the present methodology for three-dimensional flows involving moving immersed objects.

Conclusions

A numerical technique has been developed to solve incompressible fluid flows with immersed bodies in two and three dimensions. The salient features of the technique are a globally second-order accurate finite-difference discretization, a special treatment for the points lying adjacent to the immersed interface to capture the interface features in a sharp fashion and a level-set interface representation. The method does not employ

forcing functions in the governing equations. Instead the discretization is performed in such a way that the interface boundary conditions are incorporated into the discrete system of equations at the interface-adjacent grid points. In contrast to a previous finite-volume sharp interface method the discretization procedure is quite simple and entails only a modest modification of a simple Cartesian grid flow solver. The simplicity is facilitated by the level-set description of the interface. An algebraic multigrid method has been adapted to include moving immersed solid-fluid and fluid-fluid interfaces in order to accelerate the solution of the discrete pressure Poisson equation.

Several computational tests have been carried out to benchmark the method. The results from the accuracy studies, in which the present method is compared to a finite-volume method, show that the order of accuracy is not compromised by the present finite-difference method. Validation for two dimensional flows around stationary and oscillating cylinders has been performed. The current results match well with the benchmark results. Flow around a stationary sphere has been simulated for a range of Reynolds numbers to validate the three-dimensional capability. Flow around an oscillating sphere has been computed as a demonstration of the ability to handle 3-dimensional moving boundaries. The results in the 3-dimensional cases compare well with the established experimental and numerical results.

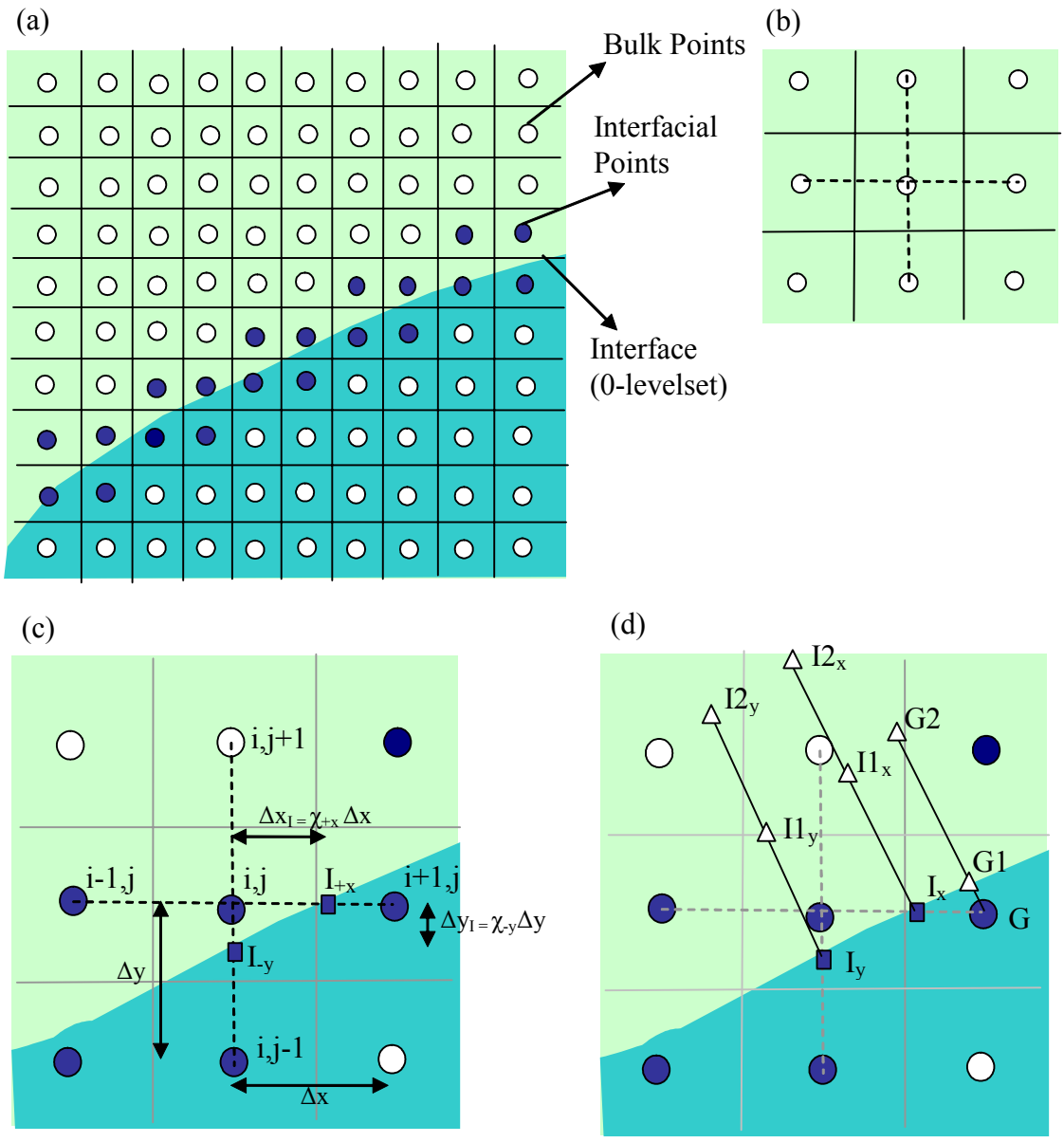


Figure 2.1 (a) Definition of the bulk (clear circles) and interfacial (filled circles) points. The interface is given by the 0-levelset. (b) Standard 5-point bulk point stencil in 2-dimensions. (c) The configuration of a typical interfacial point. (d) System for evaluating the Neumann boundary condition on the interface and evaluation of ghost pressures.

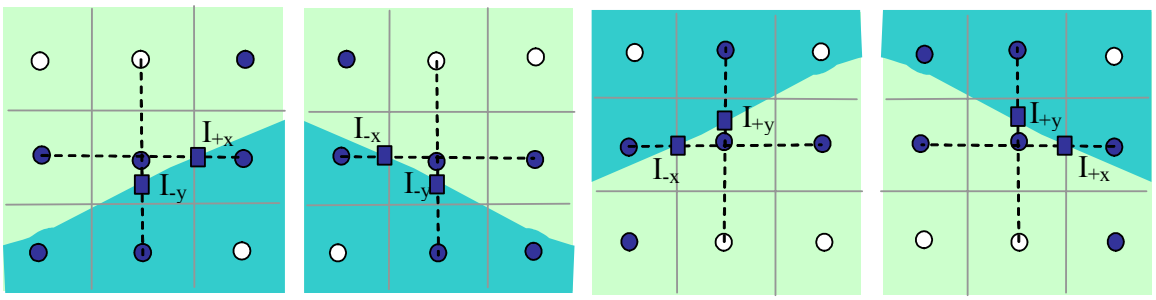


Figure 2.2 Some of the possible interfacial point situations in the 2-dimensional case.

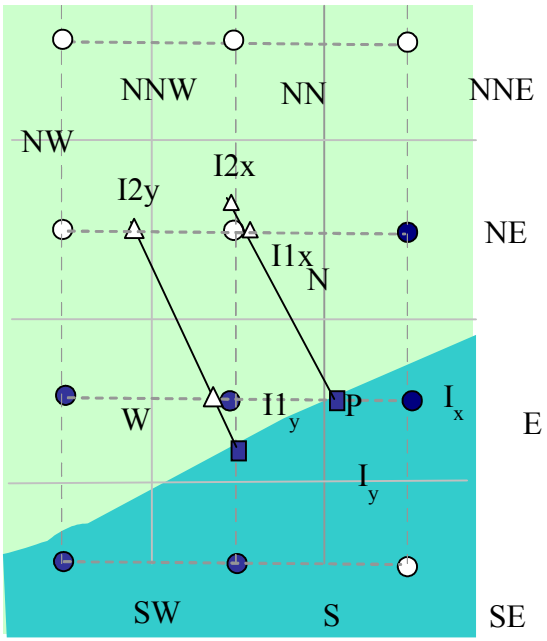


Figure 2.3 Implicit method for Neumann boundary condition on the interface.

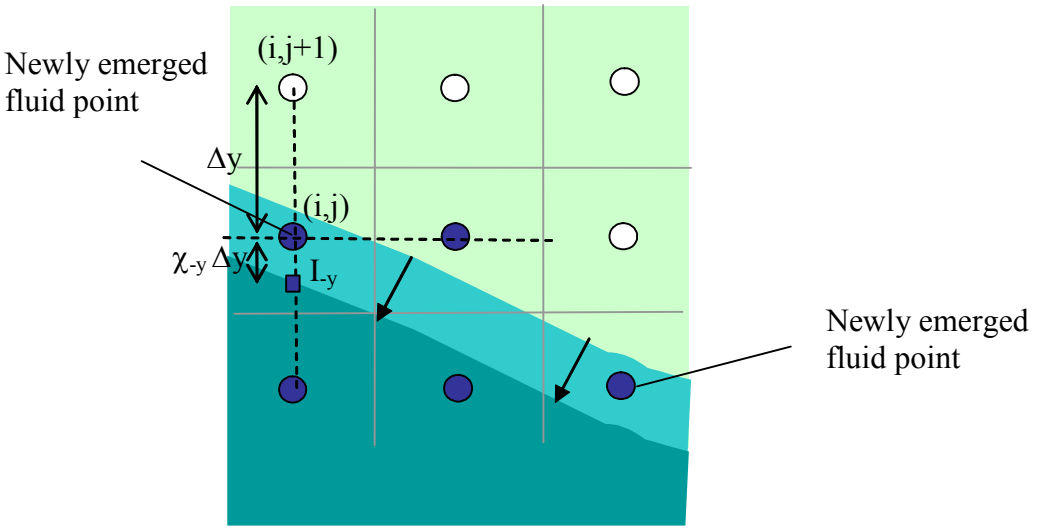


Figure 2.4 Illustration of the emergence of points from the solid to fluid phase when the sharp interface moves through the mesh

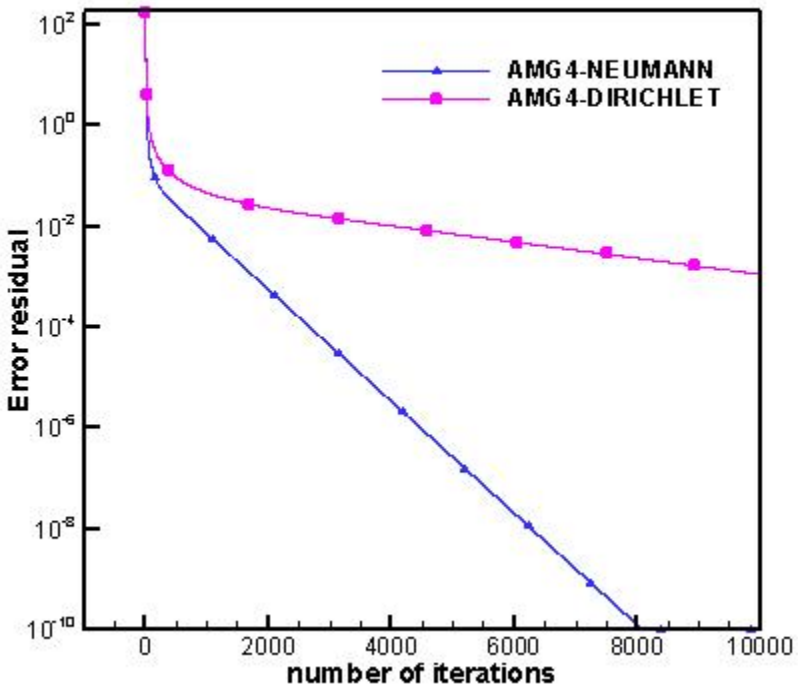


Figure 2.5 Comparison of implicit and explicit schemes for Neumann boundary condition implementation

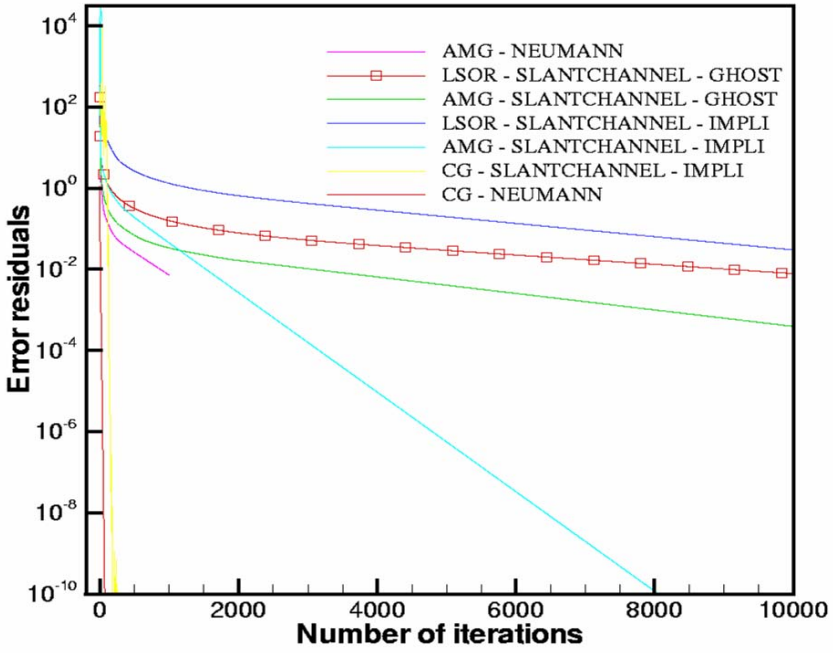


Figure 2.6 Comparison of various solvers for a channel flow.

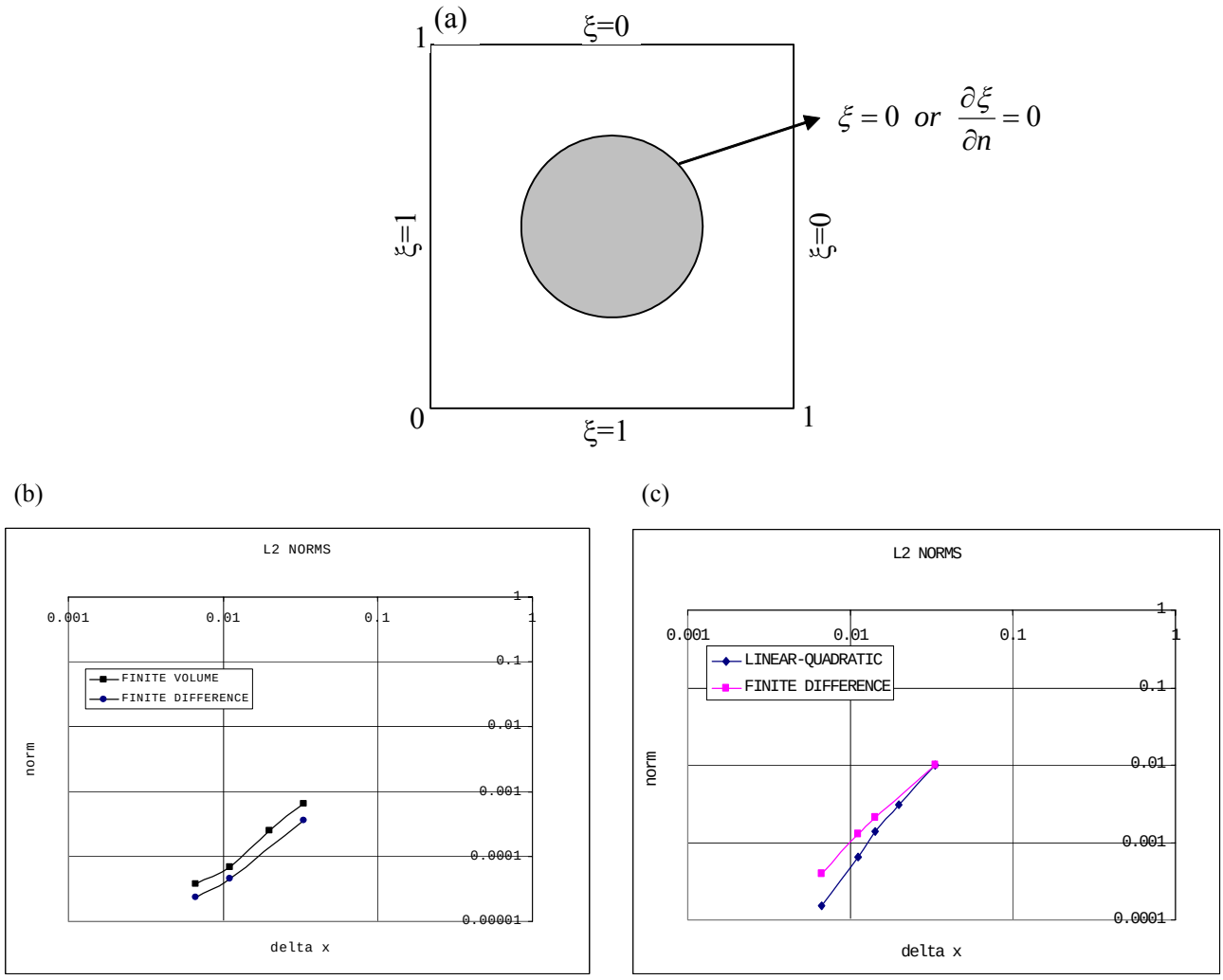


Figure 2.7 Comparison of error norms for finite-difference and finite-volume formulations.(a) Computational Setup (b) Diffusion problem (c) Convection-Diffusion problem.

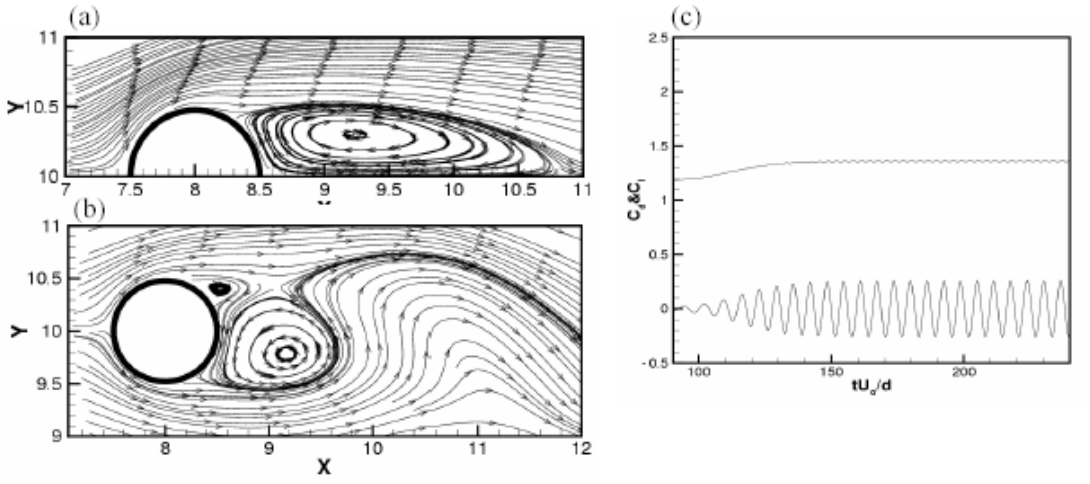


Figure 2.8(a) Axisymmetric wake behind the cylinder at Re = 40 (b) Streamlines past the cylinder at Re= 80 (c) Variation of Lift and Drag coefficients at Re = 80.

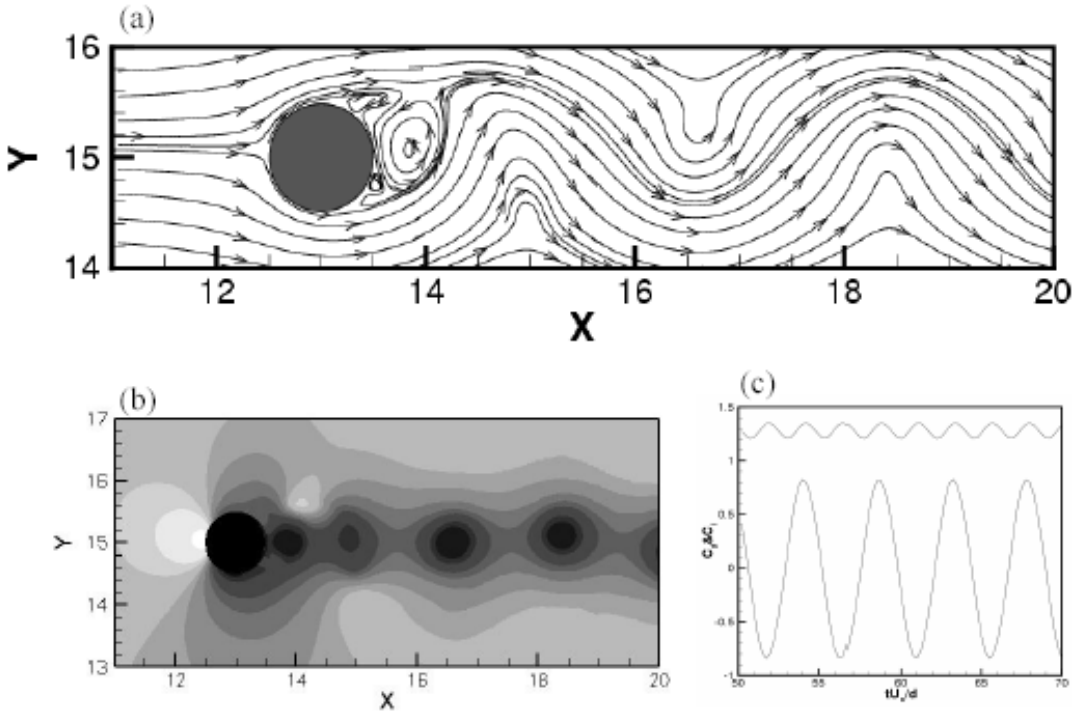


Figure 2.9 Unsteady flow around a circular cylinder at Re = 300. (a)Streamlines (b) Pressure contours and (c) Time history of lift and drag cofficients

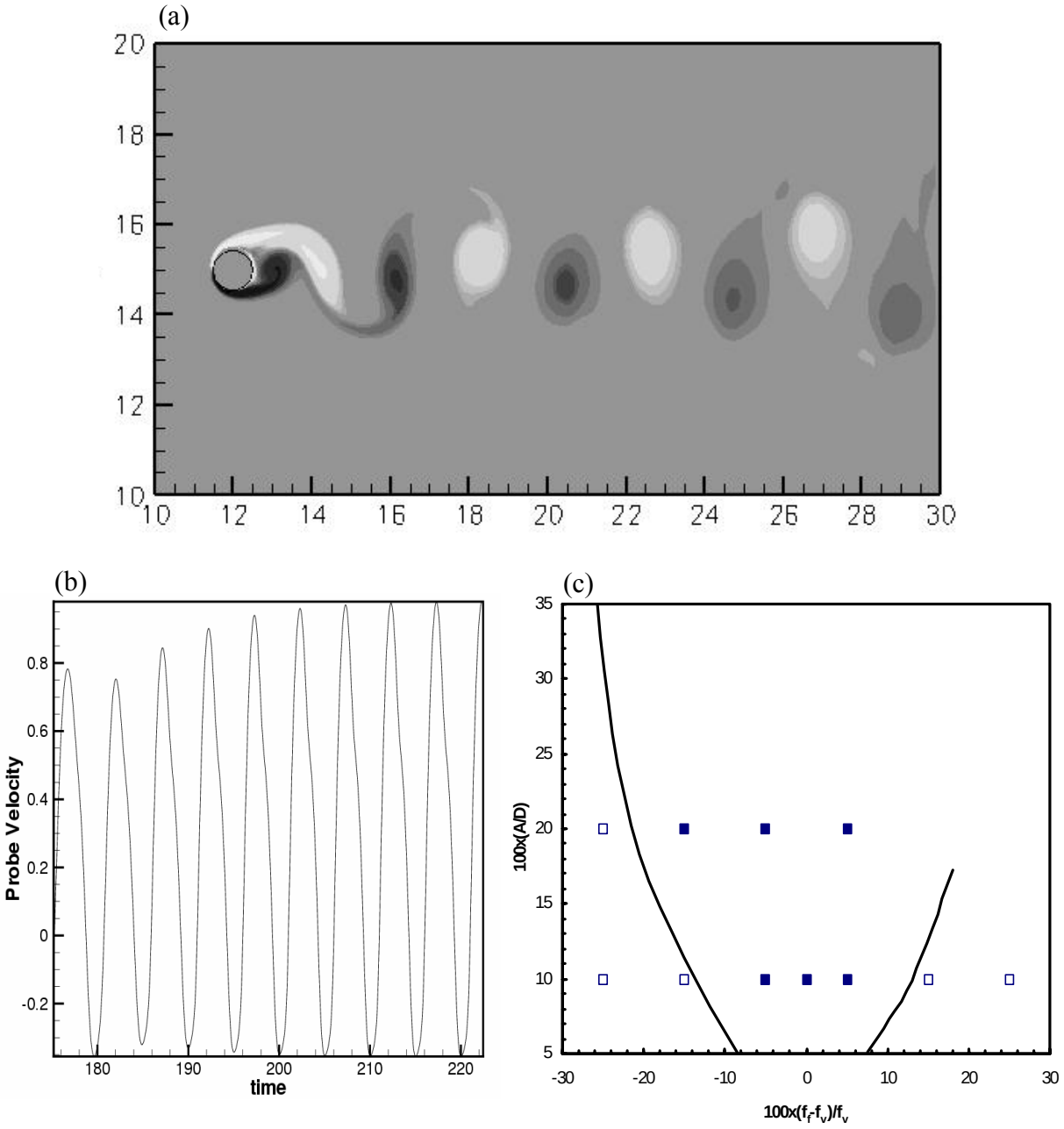


Figure 2.10 Flow past an oscillating cylinder at Re =200 (a) Spanwise velocity contours (b) Fluctuations in spanwise velocity component at a point in the wake region.(c) Comparison of the present results with the Koopmann curve for lock-on region. The closed squares indicate lock-on frequencies and open square indicate no lock-on.

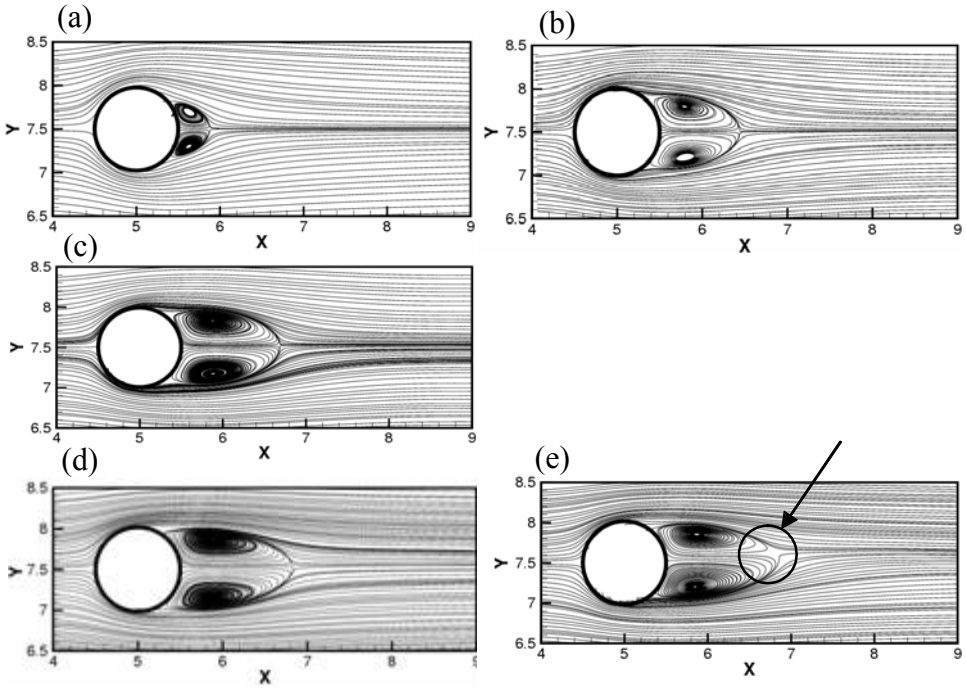


Figure 2.11 The axisymmetric streamlines past the sphere. (a) Re = 50, (b) Re = 100, (c) Re = 150, (d) Re=225, u,v vectors on the x-y plane, (e) u,w vectors on x-z plane.

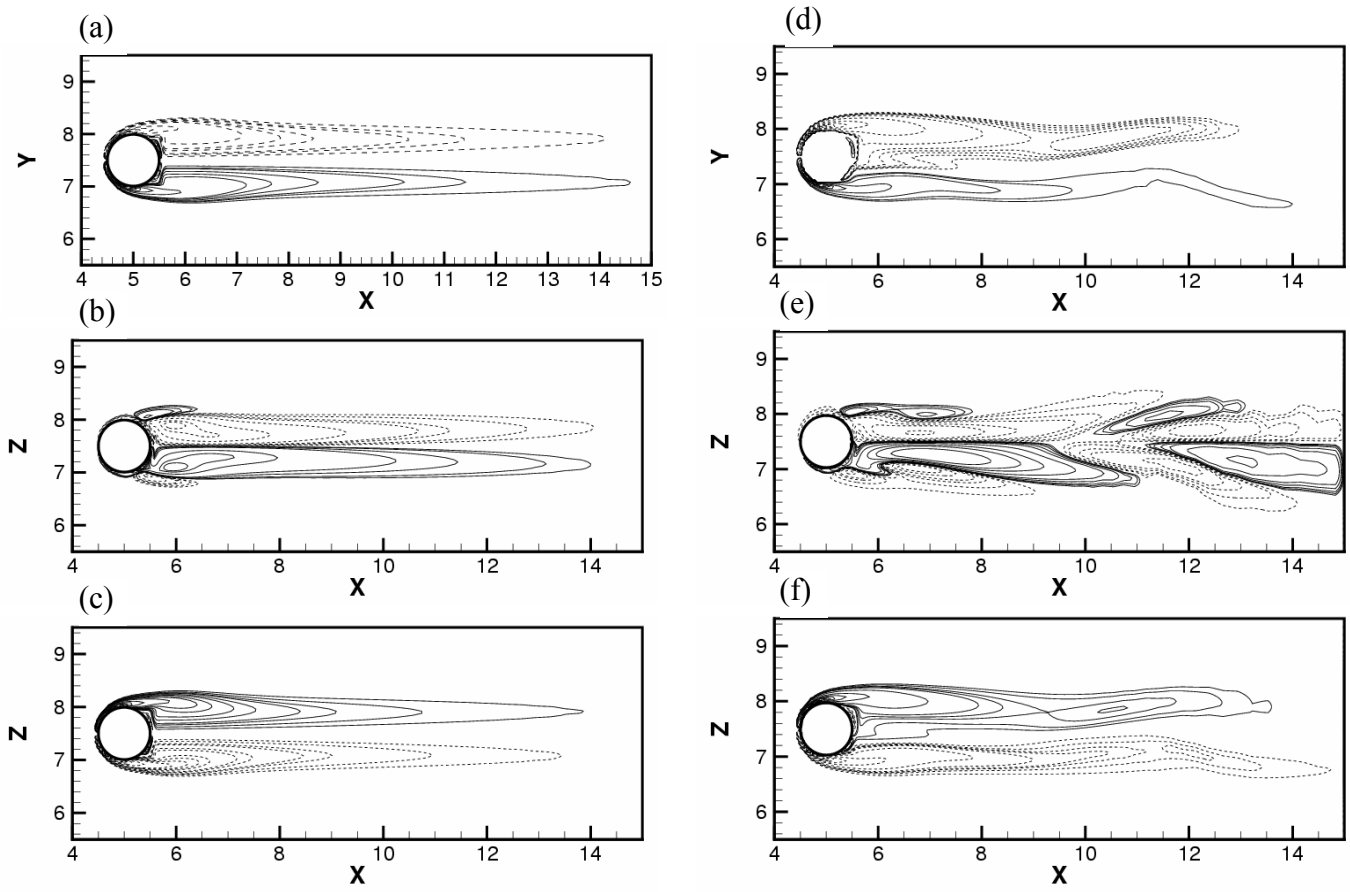


Figure 2.12 Vorticity contours for Re = 270 [a) ω_z on x-y plane b) ω_x on x-z plane c) ω_y on x-z plane] and Re = 280 [d) ω_z on x-y plane e) ω_x on x-z plane f) ω_y on x-z plane]

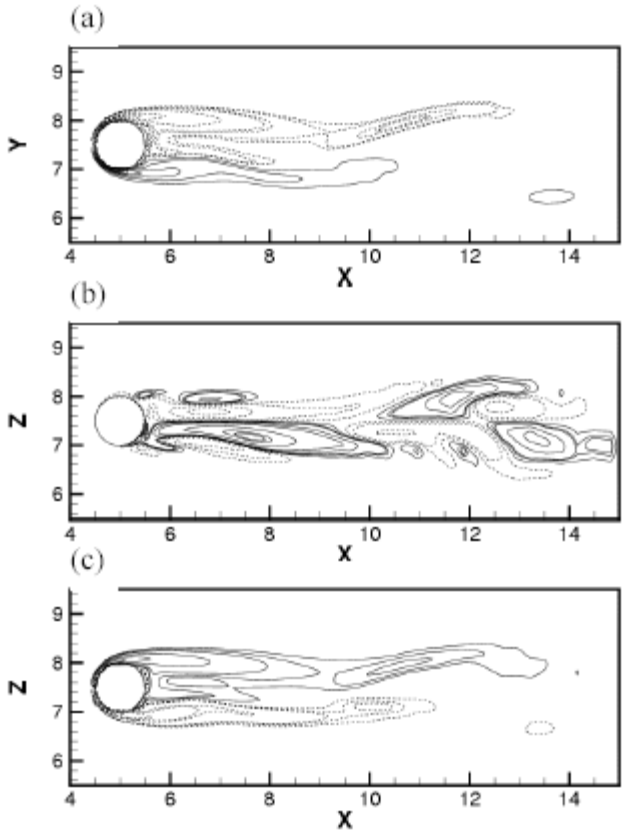
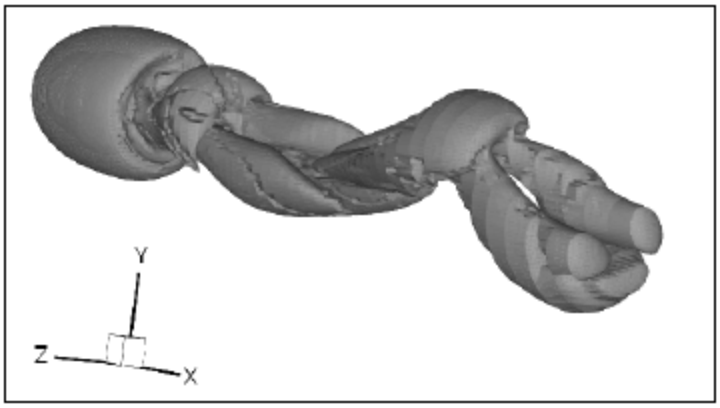


Figure 2.13 Vorticity contours for Re = 300 [a) ω_z on x-y plane b) ω_x on x-z plane c) ω_y on x-z plane.



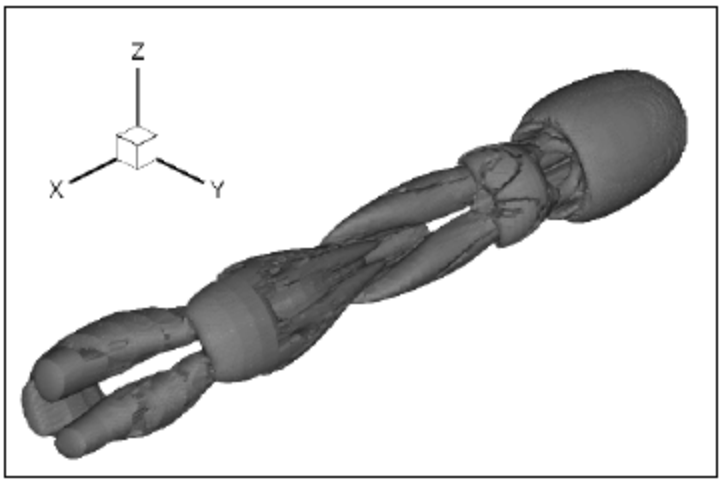


Figure 2.14 Vortical structures for flow at Re = 300. Oblique views

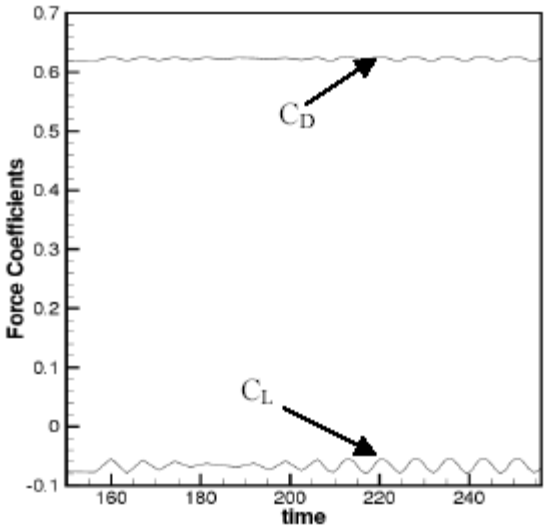


Figure 2.15 Time variation of lift and drag coefficients at Re = 300.

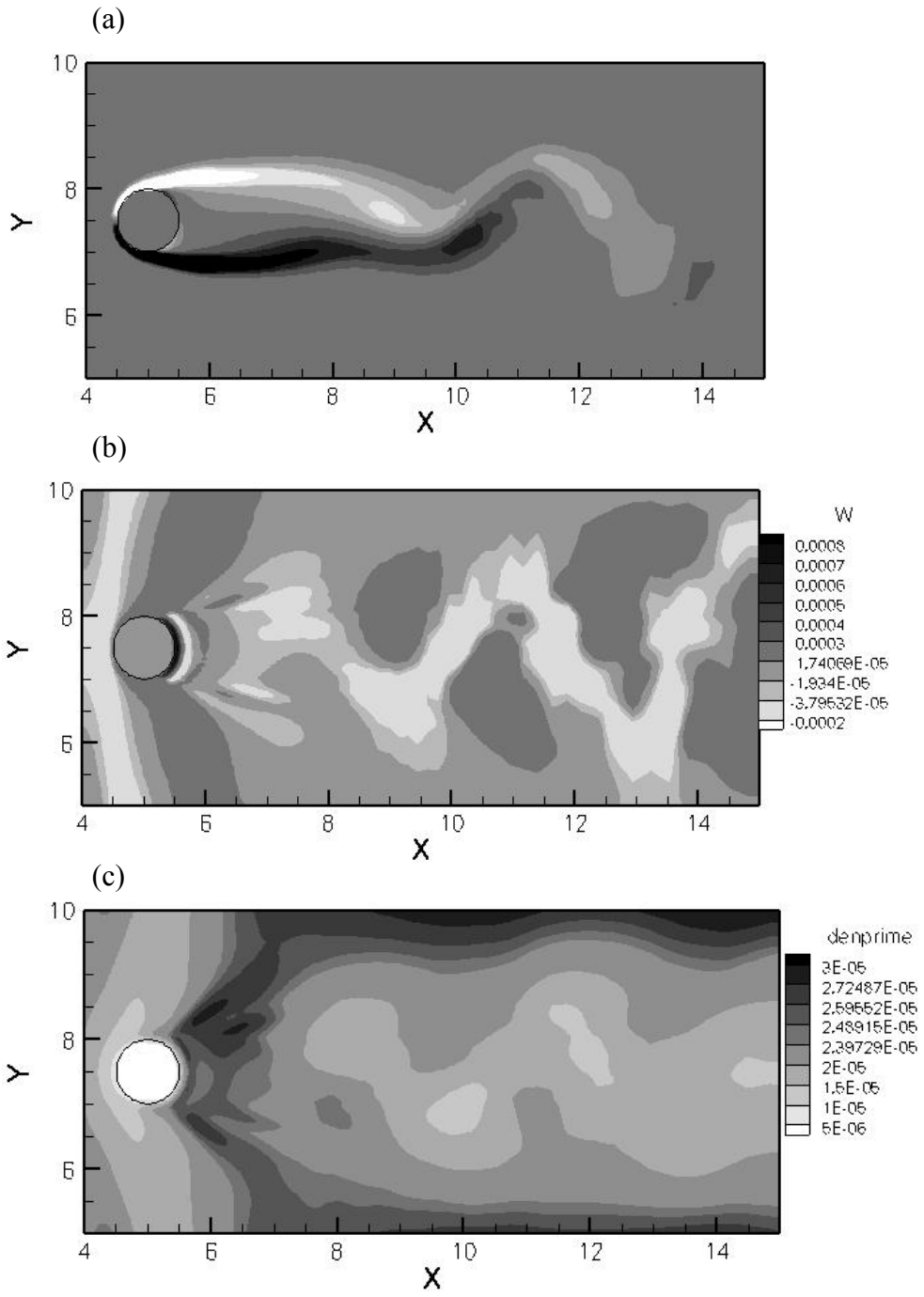


Figure 2.16 Flow characteristics for Re = 190 and Fi = 0.07 KC = 0.0 S_f = 0.0 [a) vorticity, ω_z on x-y plane b) w velocity countours on x-y plane c) density contours on x-y plane].

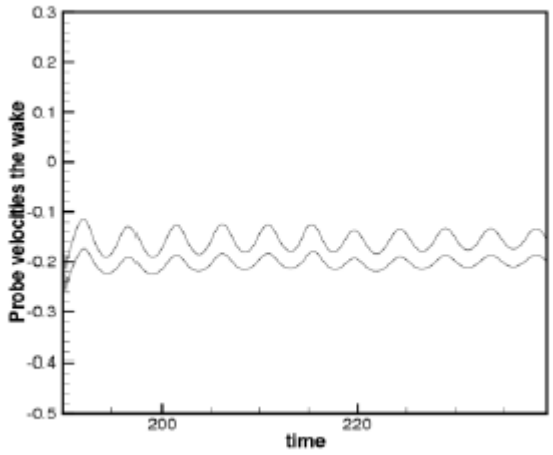


Figure 2.17 Probe velocity profile for points in the wake of the sphere [Re = 190 and Fi = 0.07 KC = 0.0 S_f = 0.0]

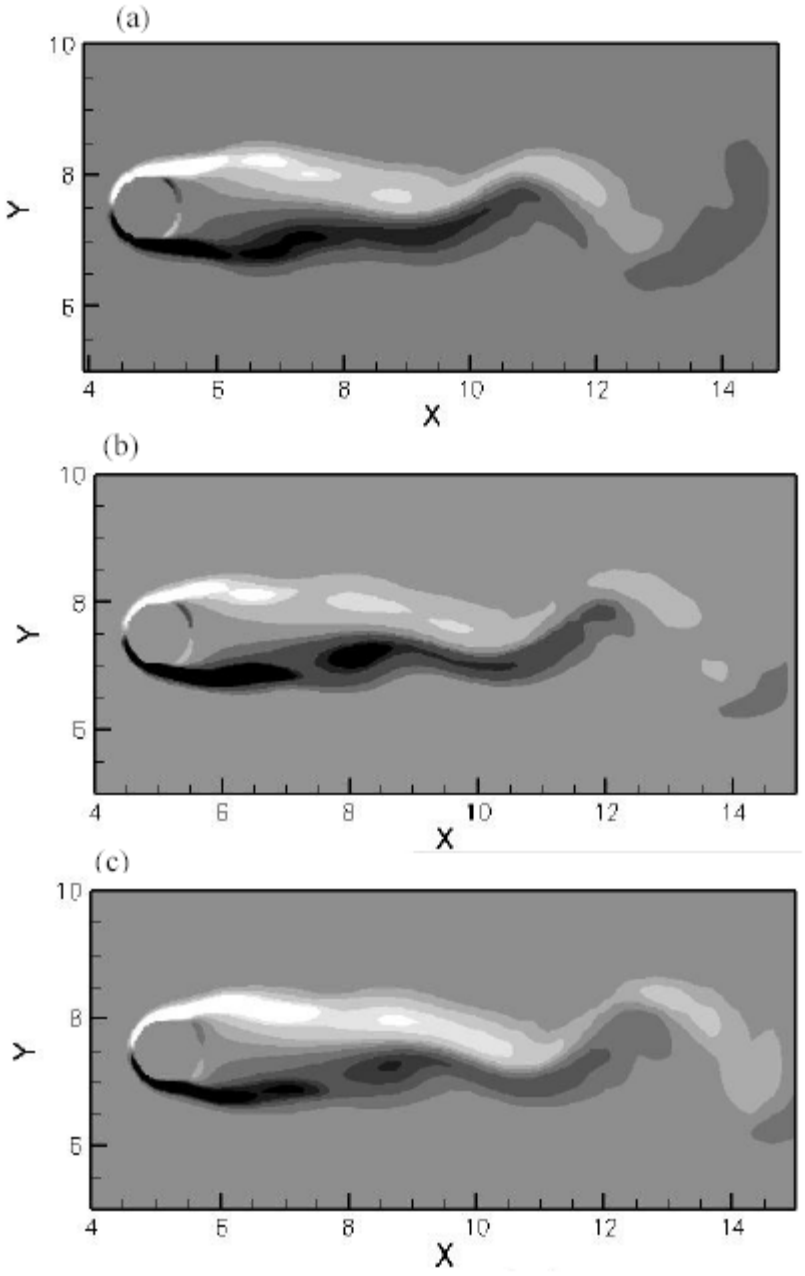


Figure 2.18 Vorticity contours on z = 7.5 plane at different stages in the oscillation cycle.[Re = 190 and Fi = 0.07 KC = 0.2 S_f = 0.35]

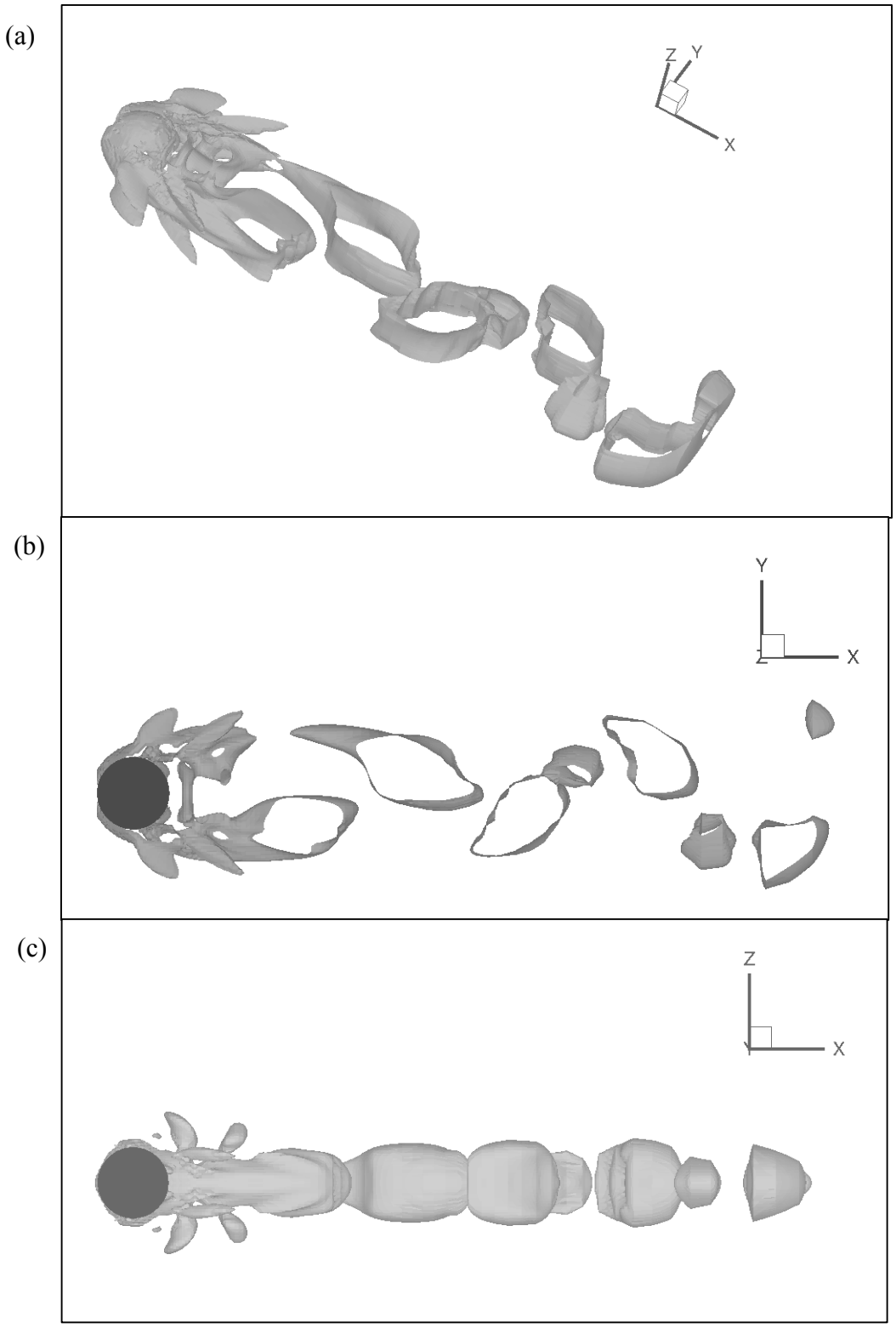


Figure 2.19 Vortical structures for the flow around oscillating sphere flow at Re = 190 Fi = 0.07 KC = 0.2 Sf = 0.35 (a) Oblique view (b) x-y view (c) x-z view

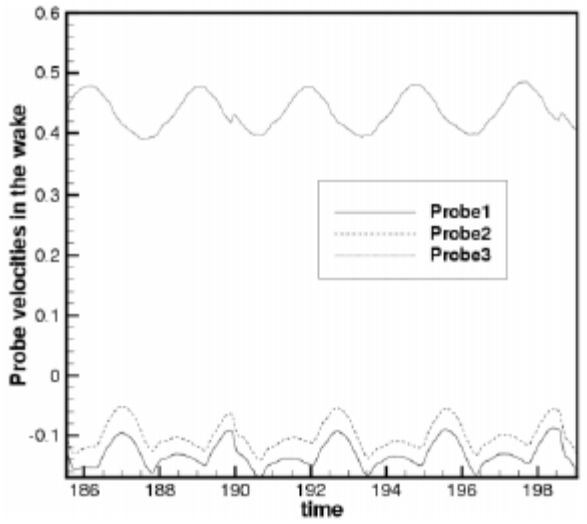


Figure 2.20 Probe velocity profile for points in the wake of the sphere [Re = 190 and Fi = 0.07 KC = 0.2 S_f = 0.35]

Re →	40		80		300	
Study ↓	C_D	L/D	C_D	St	C_D	St
Tritton[7]	1.48	-	1.29	-	-	-
Fornberg[8]	1.50	2.24	-	-	-	-
Mittal and Balachandar[9]	-	-	-	-	1.37	0.21
Williamson[10]	-	-	-	0.15	-	0.20
Finite volume[3]	1.52	2.27	1.37	0.15	1.38	0.21
Current	1.52	2.30	1.36	0.15	1.28	0.22

Table 2.1Comparison with benchmark data for flow around cylinder

Re →	50		100		150		215		300	
Study ↓	C_D	L/D	C_D	L/D	C_D	L/D	C_D	L/D	C_D	St
Mittal[13]	1.57	0.44	1.09	0.87	-	-				
Clift et al.[14]	1.57	-	1.09	-	0.89	-	0.74			
Johnson & Patel[15]	1.57	0.40	1.08	0.86	0.90	1.20			0.629	0.137
Current	1.56	0.39	1.06	0.88	0.85	1.19	0.70	1.31	0.621	0.133

Table 2.2 Comparison of results with benchmark data for flow around a sphere

PARALLELIZATION ASPECTS OF ELAFINT3D

Objectives of Parallel Implementation

As mentioned at the outset, one of the objectives of the current thesis is to develop a parallel paradigm to perform large scale computations. In Chapter 1, among other things, a brief analysis of the available parallel architectures and the popularly used parallel programming softwares has been presented. In Chapter 2, ELAFINT3D has been established as an efficient algorithm to solve the moving boundary problems. The focus in this part of the thesis is to put forth a parallelization approach for the ELAFINT3D framework. The general objectives of the parallelization process are:

a). Firstly, to develop a computer code that seamlessly executes in a serial mode on single-processor machines as well as in a parallel mode on a network of processors. The aim is to achieve this goal with minimal alterations to the structure of ELAFINT3D program, and

b). Secondly, to establish an efficient framework that maximizes the utility of the available computational resources on all the processors.

The specific aims in this study are the parallelization of the following key components of the ELAFINT3D code:

a) Flow solver. This is the critical and the most computationally taxing portion of the calculations. Withstanding the issues of parallel I/O, an efficient parallel flow solver relies on well-balanced domain sub-divisions, a clever communication protocol that synchronizes the execution of the program on all the processors and a parallel matrix system solver.

b) Local Mesh Refinement. LMR is at the crux of the ELAFINT3D framework. LMR enables to adaptively create the required spatial resolution to capture the

important flow features. An efficient parallel LMR presents a tremendous advantage for the current tool. However, some of phases of LMR being intrinsically serial in nature, seek special attention during parallelization.

c) Other Algorithms: An efficient and synchronous parallel execution of the program requires parallelization of all the other models/algorithms employed. Even though the flow solver and the LMR are the most computationally intensive components, other modules such as GENLS (Object Rendering Algorithm) and Lagrangian Particle Tracking Algorithm present significant challenges for parallel porting.

Chapter 3 lays out a parallelization algorithm, the strategies adopted for the various components of the computational tool presented in Chapter 2. The logical aspects of the strategies employed and the parallel performance aspects are presented. The performance statistics of each of the components and their collective behavior is assessed. A series of 2D and 3D simulations of several moving boundary problems are performed to validate as well as to demonstrate the capabilities of the computational tool.

Parallel Architecture

The parallel algorithms developed as part of the current work are designed to be executed on distributed memory systems. The synchronization among the processors relies on data exchanges initiated by calls to MPI subroutines. The performance analysis and the case simulations presented in this thesis are preformed on a 12 node Linux Cluster running Sun Grid Engine. Each of the nodes is a dual 3.06GHz Intel 32-bit processor chip with 4GB RAM and 36GB hard drive. The interconnect on this architecture is Myrinet and Gigabit Ethernet.

Parallel Flow Solver

The key components of any computational tool are the Input phase, Calculation Phase and the Output phase. Depending on the size of the problem and the frequency of I/O operations, the Input and Output phases can be significant. The calculation phase is, in most cases, the computationally intensive and the most time consuming part of the simulations. In the discussion below, the Input and Output phases are analyzed together followed by the Calculation phase, which involves lot more details.

Input and Output Phases

Input operations are the essential supplements to the calculation phase of the program. These operations can be significant depending on the size of the inputs and the frequency at which they are provided to the program. The Input phase is typically, but not always, a one-time procedure per execution in non-interactive computational tools. However, the operations in Output Phase are mostly performed at regular intervals during the execution. The size of the data written is proportional to the size of the problem being solved and hence is significantly large. Despite being an important task, there is yet no standard consensus on the implementation of parallel I/O techniques. A common practice in parallel CFD has been to use serial I/O phase in conjunction with parallelize compute phase. In the current implementation, the Input phase is performed in a serial fashion and the data output is performed individually on each processor and hence in parallel. For the Input operations, one of the processors, say processor P0, reads in all the inputs and broadcasts them to all the other processors using MPI calls. All the processors except P0 are idle while input data is being read in by P0. If the input data is significantly large, this will lead to a certain initial inefficiency. The Input Phase is followed by the Calculation

Phase. The details of this Phase are discussed in the subsequent sections. The results from the Calculation Phase are outputted individually by each processor. The typical operations in the Output phase such as writing ASCII files with flow variables or generating the restart files could be intensive for problems with large meshes. Writing the data separately on each processor rather than generating a single data file for all the processors is less intensive on the memory and saves significant communication overhead. The data format is designed such that the data files from each of the processors can simply be concatenated to obtain the data set for the whole domain. The concatenation operation is performed outside of the ELAFINT3D using simple shell scripts. The current strategy ensures parallel execution of the intensive Output phase and compromises on the parallel execution of the Input phase

Calculation Phase

Upon completion of the input protocol in a serial fashion, the computational phase is executed in the parallel mode. The natural choice for distributed memory systems is to adopt a Single Program Multiple Data (SPMD) paradigm for parallelization. This strategy relies on each of the processors executing the same program on a certain subset of the whole domain. Therefore, a domain decomposition algorithm that creates balanced partitions becomes crucial for an efficient parallel program. The other crucial portion is the communication scheme. In the current setup, METIS, a graph partitioning software is used to create load-balanced partitions. METIS takes in the nodal connectivities as the input information and creates partitions that are either optimally load-balanced or with minimal edge-cuts depending on the requirement. The nodes in each of the created partitions are assigned to respective processors. These nodes are called “host nodes” and their corresponding processor is called “host processor”. Each processor stores the data

and computes the solution only for its "host nodes". This ideally reduces the computational requirement by N times and speeds up the computation by N times if executed on an N processor machine. However, the "host nodes" adjacent to the partition boundaries have dependencies on the nodes from neighboring processors. These inter-processor dependencies are called "ghost nodes" and the region that encompasses them is called the "ghost region". To ensure the correctness and accuracy of the solution, the local data lists on each processor are modified to include "ghost nodes" in addition to the "host nodes". The solution for the "ghost nodes" on each processor is updated during the solve phase by communicating with the corresponding "host processor". The communication overhead due to the solution exchange and the storage overhead due to the inclusion of "ghost region" reduce the scalability of the parallel algorithm from the ideal value of N. In case of Local Mesh Refinement, the base mesh is refined or coarsened adaptively in the course of the simulation. To ensure proper load balance through out, domain partitions are altered adaptively to account for the newly created meshes. The following presents a step-wise description of the parallel algorithm:

1. Read in the Inputs on P0 and broadcast to other processors.
2. Generate the base mesh and use METIS to create domain partitions
3. Assign the partitions to the respective processors
4. Define a ghost region on each processor to account for dependencies on neighboring processors
5. Exchange the information in ghost region.
6. Refine or Coarsen the mesh as required
7. Repartition the domain to ensure load balance
8. Execute the serial ELAFINT3D on each processor
9. Go to step 5 and repeat till the end of simulation.

In summary, the key operations in the above procedure are domain decomposition, repartition in case of LMR, definition of the "ghost region" and communication of the "ghost region" within the iterative solver. The p-METIS program from METIS package is used in the current code for domain decomposition and adaptive repartition. This feature has been shown to be efficient and scalable for a wide variety of problems. The adaptive repartition algorithm assumes the existent partitions as the initial condition in creating the new load-balanced partitions. The cost of repartition is minimal since it is performed each time LMR is executed. A parallel version of BiCGSTAB(l) solver from PeTSc is used to solve the matrix system. An overview of the working of METIS and details of the communication protocols are presented below.

Domain Decomposition using METIS

METIS is a openly available software consisting of a library of programs used to partition large irregular graphs, partition large finite-element meshes and to produce fill reducing orderings for sparse matrices. The key objective of METIS is to solve the k-way partitioning problem defined as follows: Given a graph G=(V,E) with |V| = n, partition V into k subsets, V_1, V_2, V_3,…… V_k such that $V_i \cap V_j = \varnothing$ for $i \neq j$, $|V_i| = n/k$, and $\cup_i V_i = V$, and the number of edges of E whose incident vertices belong to different subsets is minimized. This strategy can be extended to graphs with weights associated to vertices and edges, in which case, k disjoint sets with equal net vertex-weights are generated. In this context, the vertex weight represents the computational task and the edge weight signifies the amount of data communication. This weighted approach is relevant to problems with immersed interfaces where the convergence rate and hence the computational load is non-uniform among the nodal points. An efficient graph partitioning tool like METIS, creates partitions with a) equal total weights, hence equal computation load and b) minimal edge-cuts meaning optimal communication overhead.

METIS adopts the multilevel graph partitioning approach(Figure 3.1) that reduces the size of the graph by collapsing its vertices and edges (coarsening phase), partitions the smaller graph (partitioning phase) and then uncoarsens it to construct a partition for the original graph (uncoarsening phase). Specifically, METIS solves the k-way partitioning using recursive bisection method. An initial 2-way partition is followed by further bisecting each of the parts. By performing log k phases of bisection creates k partitions of the original graph G. The details of the algorithms used in these three phases are detailed in the publications by METIS developers, Karypis and coworkers. A synopsis of the key concepts in each of the phases is presented here to provide an understanding of how the high-quality partitions are created.

1. Coarsening Phase: The graph G_0 is transformed into a series of smaller graphs $G_1,G_2,\ldots..G_m$ such that $|V_0|>|V_1|>|V_2|>\ldots..>|V_m|$. The strategy in creating coarser graphs is to combine a set of vertices of G_i to form a single vertex at the next coarse level G_{i+1}.If V_i^{ν} are the set of nodes of G_i combined to form a vertex ν (referred to as multimode) of G_{i+1} then the weight of vertex ν is set equal to the sum of weights of vertices in V_i^{ν} . The connectivity information on G_i is preserved by setting the edges of ν as the union of edges of V_i^{ν} . This enables one to create partitions with the same edge-cut on both coarser and finer levels. Figure 3.2 shows some of the methods of updating the vertex and edge weights in coarsening phase. There are various heuristics proposed to determine the set of multinodes in graph coarsening. Four such methods are described in detail in Karypis, 1998.
2. Partitioning Phase: In the partitioning phase of the multilevel algorithm, a bisection P_m of the coarse graph G_m that features minimal edge-cut and equal total vertex-weights is generated. This task becomes relatively easy, since the coarsening phase sets the vertex and edge weights on the coarser graph to reflect the weights of the original graph. Several algorithms viz. spectral bisection, geometric bisection or

combinatorial methods can be used to create the partitions P_m. The reader is referred to the corresponding literature for details on these methods. partition

3. Uncoarsening Phase: The partition P_m of G_m is projected back to G_0 by going through intermediate partitions P_{m-1}, $P_{m-2,}$…..,P_1,P_0. Assembling the partition P_i from P_{i+1} is done by the assigning the set of vertices V_i^v collapsed to form $v \in G_{i+1}$ to the partition P_i. ($P_i[u] = P_{i+1}[v]$, $\forall\ u \in V_i^v$). Even though P_{i+1} minimizes the edge-cuts on G_{i+1}, the projected partition P_i may not be the optimal partition for G_i given more no. of degrees of freedom. Hence a partition refinement algorithm is performed to fine-tune the partition on the i^{th} level before projecting it on to the next finer graph G_{i-1}. Again, the reader is referred to the articles by Karypis et al. for further details on the refinement algorithms.

Definition of Ghost Region

Though the data structures used in parallel implementation are similar to their serial counterparts, only a list of data local to a processor is generated due to domain partitioning. To ensure exactly the same solution from both serial and parallel executions, a communicating list consisting of all the inter-processor dependencies or the "ghost nodes" is defined on each processor. This list is referred to as "ghost region" in this article. As mentioned before, the definition of "ghost region" is critical to the performance of the parallel algorithm. The definition of "ghost region" in the present study is as follows: If p and q are processors hosting adjoining partitions and the solution at a point p_1 on processor p depends on the value at point q_1 on processor q, then, q_1 belongs to "ghost region" of the processor p. The mesh point q_1 is solved for on its host-processor q and the solution is communicated to processor p and thereby updated on the processor p.

Even though inclusion of "ghost region" is not the only method of achieving the same solution as the serial code, it is preferred over other methods because it also serves the purpose of minimizing the alterations to the existing serial code. This feature is illustrated in the following example from Figure 3.3(a). The discretization of Laplacian operator at point P involves its neighbor E as in the following equation.

$$\nabla^2\phi = \frac{\phi_E - 2\phi_E + \phi_W}{\Delta x^2} + \frac{\phi_N - 2\phi_E + \phi_S}{\Delta y^2} \qquad 3.1$$

By virtue of partitioning, P and E belong to processors p and q respectively. If processor p were to store data lists for only the "host nodes", point P would require special treatment while evaluating the Laplacian operator. However, with the concept of "ghost region", point E is also included in the data list on processor p but the value ϕ_E is obtained by communicating with the processor q. This preserves the structure of the stencil at point P.

In the above illustration, the evaluation of the Laplacian operator requires information about one computational point from the adjacent partition and hence it would suffice to include one layer of grid points from the partition q in the "ghost region" of processor p. The extreme possibility one could encounter in ELAFINT3D is during the flux evaluation for ENO scheme in the level-set method, where information for up to four neighbors in each direction is required. This prompts us to set the width of the "ghost region" to four mesh points around each partition. The extent of ghost region close to the partition boundary is clear from the close up view shown in Figure 3.3(b).

In case of LMR, the definition of "ghost region" is more complicated. Unlike the case of uniform grid, the Laplacian operator has dependencies on many more computational points. For example in Figure 3.3(c), the Laplacian operator at point P is dependent on the point E_1 which is at the far top corner of the neighboring base mesh.

This happens especially for the points on the mesh interfaces or "ref-boundary cells" as discussed in Chapter 2. By virtue of the finite-volume discretization at these points, the extent of the computational stencil is expanded beyond the 5-point stencil. Hence for the case of LMR, if one of the child cells of a certain base cell belongs to the "ghost region", all the other child cells belong to that base cell also belong to the "ghost region". Even though this strategy can be costly for very high levels of refinement and also in 3D, this significantly eases the logical aspects of the definition of "ghost region". Also, it will be apparent later in this chapter, including all the cells in the tree structure of a base cell will help devise efficient strategy for parallel LMR.

A peripheral aspect for better parallel performance is the indexing of the "host nodes" and the "ghost nodes". It has been shown that by numbering the "host nodes" followed by the "ghost nodes" will produce significant gains in the context of parallel iterative solvers, especially for matrix-vector multiplications. The numbering scheme determines the order in which the elements of the matrix are loaded into the computer memory. Storing all the "host nodes" sequentially ensures data proximity in assembling the global matrix. This strategy reduces the memory latency in the solver phase.

Communication of Ghost Region using MPI

As mentioned before, Message Passing Interface (MPI) is used for data communication in parallel code. The system-independent feature of MPI makes it the right software for distributed systems architecture where different workstations potentially possess varied configurations. The MPI features that are widely used in the code are a) MPI_(I)SEND, MPI_(I)RECV for (non-blocking)blocking point-point communications during domain decomposition and ghost region exchange b) MPI_BCAST for collective communication such as broadcasting the inputs to all the

processors, c) MPI_GATHER and MPI_REDUCE to perform collective operation such as for residual evaluation and output operation, d) MPI_BARRIER,MPI_WAIT for synchronization of the code at various junctures, e) MPI_WTIME to calculate the computation and communication times of the code etc. There are several MPI data types that have been defined to facilitate in data communication.

The exchange of the "Ghost Region" is at the crux of the parallel code performance. This operation can be done in either a synchronous or asynchronous fashion using MPI. In the synchronous communication, the computations are halted till all the processors complete their communication calls. This is suited to problems in which the load is perfectly divided among a series of processors with uniform processor and data communication speeds. Although it strategy ensures the same solution as a serial execution, synchronous communication protocol leads to under utilization of the computer resources in not so perfect systems.

The asynchronous mode of communication is better suited for the general case, with disparities in the load-balancing, processor and network performances. This is a non-blocking strategy which allows an overlap in the communication calls without having to wait for each processor to finish its communication related tasks. This scheme allows for the processors to perform useful computations upon performing their communication calls. However, this imposes an onus of managing buffers to hold messages until the receiving process is ready. Hence in this mode of communication the asynchronous calls are interspersed with logically placed barrier calls to ensure the correctness of the solution.

The key to an efficient scalable solver will be communication overhead incurred during the solution of the discrete system. Since each processor assembles the matrix and

the vectors for its "host nodes", the data exchange due to the "ghost nodes" during the matrix-vector operations will determine the communication overhead and hence the solver efficiency. In the current study, parallel version of BiCGSTAB is used from PeTSC package. The numbering scheme adopted for "host" and "ghost" nodes enables for efficient use of Petsc solvers.

Parallel Local Mesh Refinement

As detailed in Chapter 2, LMR is used in the current framework to adaptively create spatial resolution as required by the simulations. This strategy enables an efficient use of meshes to resolve the required flow features. Since a parallel paradigm is adopted to perform large scale calculations, it is imperative to create an efficient parallel algorithm for LMR. The sequence of operations for a serial LMR implementation are :

1. Based on the refinement criteria, refine or coarsen the existent mesh structure
2. Assign pointers for parent/child cells and also the neighbors
3. Smooth the refined mesh to ensure utmost a single level difference between any pair of neighbors.
4. Assign variable values to the newly created meshes by interpolating from the older mesh.

In the above algorithm, the Steps 1,2 and 4 are independent on each processor. Given an initial mesh structure, the refinement criteria can be evaluated at each "host cell" and the cell can be refined or coarsened accordingly. However, for the "ghost cells" the refinement criteria have to be evaluated on their "host processor" and be communicated to the "ghost processor". This operation can bear a significant communication overhead depending on the number of "ghost cells" on each processor. Due to the high processor speeds, it is sometimes efficient to perform computations rather than exchange

information if all the data is available. One of the strategies here is to evaluate the refinement criteria for the "ghost cells" along with the "host cells" and refine or coarsen them as required. In the current context this strategy is adopted to avoid the communication overhead. Assigning the pointers for each of the newly created meshes doesn't require any communication. Also the 4th step which involves interpolation of the values from the older meshes to the newly created meshes is only computational work. Only the 3rd Step which involves smoothing of the refined meshes involves communication among the processors. In serial algorithm, the smoothing of the refined meshes is a global operation. This operation is performed starting with the first base mesh and is recursively performed for all the cells in the tree structure. Once the whole tree has been traversed, the operation proceeds to the next base mesh. However, in parallel, executing the smoothing operation on each processor independently could create different meshes in the "ghost region". Some of the strategies to perform this mesh smoothing are

1. Strategy I
    a. Choose a base cell A from the global mesh.
    b. Traverse through the tree structure of A on its "host processor". Choose a cell B in the tree.
    c. Consider the neighbor (say C) B.
    d. If (levelB – levelC) > 1. Refine C. Smooth the cell C
    e. If (levelC – levelB) > 1 Refine B. Smooth the cell B
    f. If neither of the above conditions is satisfied go to Step c till all the neighbors are exhausted.
    g. Go to the next cell in tree structure of A and repeat Steps (c-f)
    h. Go to the next base cell and repeat Steps (a-g)

    The above strategy operates on only one cell at a time. All the processors are idle except the "host processor" of the cell under consideration. Although the

resultant smoothed mesh is unique and is identical to the mesh created by a serial execution, this approach is completely sequential.

2. Strategy II

   In this approach, the "host cells" on each processor are considered for smoothing. Any cell refined by virtue of smoothing is communicated to all its "ghost processors".

   a. Consider a "host cell" A on processor p.
   b. Perform smoothing operation for the A on its "host processor".
   c. If the cell A or one of its neighbors B is refined due to smoothing, communicate the change in level to their "ghost processors" and refine the corresponding cells on the "ghost processors".
   d. The updated refinement initiates a smoothing operation on the "ghost processor" as applicable.

   By adopting the above strategy recursively for all the "host cells", and performing smoothing operation when a cell in the "ghost region" is updated will create a smoothed refined mesh. However, the number of recursions on each processor is dependent on its "host cells" and also the timing of the update of "ghost cells". Even though this strategy leads to a unique refined mesh as in the previous case, the number of recursions for higher levels of refinement, especially in 3D, will be humungous.

3. Strategy III

   A third strategy which is effective and also scalable is adopted in the current thesis. The main drawback of Strategy II was to update the "ghost cells" as soon as they are refined. This update in many cases, also triggers the smoothing operation in the vicinity of the "ghost cell" leading to an increase in the number of recursions. So a different approach is adopted where, the aim

is not to create the same smoothed mesh as in the serial algorithm, but to create a smoothed yet marginally different mesh.

a. Consider a cell A on processor p. cell A can be a "host" or a "ghost" cell.
b. Perform smoothing operation for the cell A.
c. Perform the above steps recursively on all the cells on processor p.
d. After all the cells are exhausted, match the "ghost region" on the processor p with the other processors. If there is a mismatch of refinement of any particular cell, correct it by refining both of the cells to the same level.
e. Step d ensures that the smoothing operation will create consistent refinement in all the ghost regions.

Even though the current strategy will not create an unique smoothed mesh, the scalable nature of the approach makes it attractive. Moreover, the smoothing operation of LMR is performed in the view of discretization schemes for the governing equations. Since, the initial refinement based on the solution criteria has been performed exactly, the refinement alterations due to smoothing operation will only enhance but not deteriorate the refinement in the domain. Also the current approach can be performed independently on each domain and only has to synchronized for the cells in the ghost region.

Parallel Sparse-Linear Solver

The key component of the flow solver is the iterative solver for the sparse linear system created from the discrete set of governing equations. As described in Chapter 2, BiCGSTAB*(l)* is used in the current framework to solve the matrix system. The parallel version of BiCGSTAB*(l)* from the PeTSc package is used in the current framework. A block Jacobi preconditioner is used for the parallel version while an incomplete LU decomposition is used for the serial version of ELAFINT3D.

Other Parallel Algorithms and Miscellaneous Issues

The main components of ELAFINT3D in view of parallelization are the flow solver and LMR. The strategies described above have been found to be very efficient in terms of scalability and implementation. The parallel performance of the strategies is presented later in this chapter. Apart from the above components, other models and algorithms implemented in the framework pose significant challenges to parallel porting. Some of such issues are

1. GENLS – an object rendering algorithm:

   The level-set algorithm is an important component of the current computational tool. The embedded objects, which are Lagrangian entities, are represented on the Eulerian mesh using the level-set method. The interaction of the flow solver with the immersed objects is also through the level-set field. However, generating a level-set field for arbitrary shapes is not straightforward. GENLS algorithm in the current tool generates a level-set field for any three-dimensional object represented by a surface mesh. The GENLS algorithm comprises of the following key steps:

   a. Identify the all the fluid mesh points that straddle the embedded object. For these points, calculate the shortest signed distance from the object by drawing a normal on to the embedded surface.
   b. Once this layer of mesh points adjacent to the object are identified and their signed distance function is calculated, use fast-marching method to calculate the signed-distance of all the other points in the domain.

   The above approach is a key component of ELAFINT3D, since the level-set field can be explicitly defined for very few shapes. Using GENLS any surface mesh can be imported into the fluid domain in the form of a level-set field. During parallel execution, it is imperative that the above algorithm be implemented in

parallel. The first step of identifying the fluid mesh points ("host cells") straddling the object can be performed individually on each processor. However, executing the fast-marching algorithm in parallel requires communication between the processors. The parallel fast-marching strategy developed by Hermann has been implemented in the current tool. The speedup realized in the implementation of GENLS in the current code was comparable to that reported by Hermann. Strategy to create the partitions on the base mesh

Since the applications of interest are mostly moving boundary problems, LMR is invoked for regions near the immersed object. However, in the parallel implementation, the global base mesh is partitioned even before defining level-set field. Hence the initial partitions are oblivious to the refinement to occur near the interfaces. Depending on the location of the interface, the initial partitions could create a huge load imbalance during the initial LMR execution leading to large computational times. A strategy to bypass the above conundrum is to weigh the cells near the interface higher than the cells away from the interface. In case of standard geometries, it is easy to determine the proximity to the object, but for arbitrary objects, created using GENLS, even an approximate assessment seem to have improved the performance significantly.

2. Strategy to create the partitions with LMR

   As described earlier, ParMetis is used in the current code for domain decomposition. One of the advantages of using ParMetis is that, it has been designed to create partitions for multi-physics problems. In the current framework, the computational load on all the mesh points is not uniform. For example, a base mesh cell has to perform far less computations as compared to a cell at 5^{th} level of refinement. The "ref-boundary cells" incur extra computation during coefficient assembly and also in the solver due to their extended stencil. Similarly, the refined cells have to perform extra logical operations when

performing interpolations or identifying the neighbors in LMR algorithm. The non-uniformity in the computational loads of refined and unrefined mesh points prompts for strategies to assess the differences and balance these loads. ParMetis allows one to weigh each of the vertices according to their computational task, so as to create partitions of equal computational task. This facility is exploited in the current thesis, to weigh the cells based on their level of refinement. The weight of a cell at n^{th} level is assessed to be 2^{n-1}. So a cell at level 1 will have a weight of 1. This strategy ensures equal distribution of refined cells across the processors. In fact, the above strategy proved to be the key to attain scalability for the parallel LMR algorithm.

Performance analysis

The most commonly used measures of performance of parallel programs are speedup and efficiency. Speedup is defined as the ratio of runtime of the serial code to that of the parallel code. Unlike serial program, the parallel runtime depends on the no. of processors (p) in addition to the problem size. The speedup is defined as

$$S(n, p) = \frac{T_{serial}(n)}{T_{parallel}(n, p)} \qquad 3.2$$

T_{serial} is defined as the execution time for unified ELAFINT3D running in serial mode on a single processor of the parallel system. If the speedup equals p, the program is said to attain linear speedup. Such an occurrence is unusual in these simulations because of the communication overhead. The parallel execution time can be decomposed as follows

$$T_{serial}(n, p) = T_{calc}(n, p) + T_{comm}(n, p) + T_{startup}(n, p) \quad 3.3$$

The startup time includes the CPU time used up for I/O operations and for domain decomposition. The communication time accounts for the exchange of "ghost region" information across various processors. The calculation time is the time taken for the floating point operations involved in the execution of the serial ELAFINT3D on each processor on its local data set. In addition to the overall speedup as defined in Eq. 3.2, the solver speedup is measured as the ratio of CPU time for the solve phase in serial simulation to the corresponding runtime (includes T_{calc} and T_{comm} but not $T_{startup}$) in parallel execution mode. Efficiency is the measure of process utilization in a parallel program, relative to a serial program. It is defined as

$$E(n, p) = \frac{S(n, p)}{p} = \frac{T_{serial}(n)}{pT_{parallel}(n, p)} \quad 3.4$$

If E(n,p) = 1 the program is said to be exhibiting linear speedup, if E(n,p) < 1/p the program is exhibiting slowdown.

The performance of the parallel ELAFINT3D has been evaluated for both two dimensional and three dimensional problems. The key parallel algorithms implemented in the current setting are a) parallel flow solver b) parallel LMR c) parallel GENLS. The speedup for each of these components individually is calculated based on the CPU time.

The timing data for two dimensional problems are from flow around a stationary cylinder. The domain size of 10x10 with a cylinder of diameter of 1 located at (5, 5) is used in these calculations. Table 3.1 tabulates the CPU time for a 500x500 mesh without

LMR. The number of processors is increased from 1 to 24. It is apparent that the speedup in both the solver time and total calculation time is linearly proportional to the computational load on each processor. Table 3.2 shows the scaling of the data output phase for a mesh size of 1000x1000. The linear speedup with increasing processor count shows the gains of executing output operations individually on each processor. Table 3.3 shows the effect of increasing computational load by fixing the number of processors. Despite the discrepancies at low computational loads, the CPU time scales proportional to the load for mesh sizes 250x250 to 1000x1000. The computational time increases by 14 times with an increase of 15 times in the number of computational points.

The timing data from a LMR case is tabulated in Table 3.4. With a base mesh of 200x200 and 5 levels of refinement, the number of processors is varied from 1 to 12 to obtain a nearly linear speedup. It is to be noted that the number of mesh cells on each of the processors is higher due to an increased extent of ghost region. The LMR time is therefore proportional to the number of cells including the ghost cells. The solver time however, is proportional to the number of processors with a deviation due to increased dependencies on the ghost region. This deviation increases with an increase in the number of levels. Table 3.5 shows the CPU times for a base mesh of 200x200 on 12 processors. The refinement levels are varied from 1 to 5. The variation of LMR time is non-linear with the increasing number of levels. This is apparent from the fact that the size of tree and hence the tree searches increase proportional to 4^n where n is the number of refinement levels. The CPU time for the solver is still proportional to the increase in the computational load.

The timing data for the three dimensional calculations are based on flow around a stationary sphere in a 10x10x10 domain. A sphere of diameter 1 is located at the center of the domain with an oncoming flow in x-direction. A set of timing data has been tabulated

in Table 3.6 for a 100x100x100 mesh without LMR. In this case the number of processors are increased from 1 to 24. The solvers and the total code show a speedup of 22.18 and 21.15 on a 24 processor run. This data confirms that speedup can be attained for both two and three dimensional problems.

The other component of the parallel framework is parallel GENLS. In GENLS, the interface distance is exactly calculated on each interfacial cell. The distance front is advanced in both + and – directions starting with the closest point. Table 3.7 shows the timing data for 1-8 processors for the case of a sphere. The timing data shows that with increasing number of processors, the code scales well with the processor count. In this case, the interface is located in a symmetric manner in the domain thereby uniformly distributing GENLS load to all the processors.

Unified ELAFINT3D

One of the specific goals of the current work is to develop a framework to perform simulations in either serial or parallel modes as deemed fit by the user based on the problem size. This goal is achieved by including CPP (C PreProcessor) #ifdef statements in the ELAFINT3D code as a means to switch between the single processor version or the parallel execution mode. This avoids maintaining different versions for varying modes of operation.

Results and Discussion

The ELAFINT3D is currently being used to simulate flows involving stationary and moving embedded interfaces in both two and three dimensions. Hence it is desirable to ensure that the unified parallel version of ELAFINT3D possesses similar capabilities.

In this view, the following test cases have been coined for to demonstrate the parallel execution capability and performance assessment.

(a) Flow around a stationary circular cylinder at Re = 300 (2D stationary boundary)

(b) Flow around a transversely oscillating cylinder at Re = 200 (2D moving boundary)

(c) Flow around a stationary sphere at Re = 300 (3D stationary boundary)

(d) Simulation of interacting spheres settling under gravity (3D moving boundary)

(e) Simulation of the flutter of a falling oblate ellipsoid (3D moving boundary)

The results from these simulations are presented in the following sections along with numerical data from performance analysis.

Flow around a stationary cylinder at Re = 300

The computational setup for the simulations of flow around stationary cylinder is shown in Figure 3.4(a). The cylinder of diameter 1.0 is placed at (12,15) in a 30x30 domain. The left domain boundary is an inlet with a uniform velocity of 1.0 and the right boundary acts as an outlet. Symmetry boundary condition is imposed on the top and bottom boundaries. A base mesh density of 150x150 is used with 5 levels of refinement for LMR. The maximum number of mesh points in the course of simulation is 44444. The current simulation is carried out on 6 processors. The domain partition is illustrated in Figure 3.4(b). Figures 3.4(d) display the mesh refinement in the vicinity of the cylinder and the wake region respectively. The time history of drag and lift coefficients is recorded for validation purposes. The mean drag value in this case is 1.38. The Strouhal number calculated from the oscillation frequency of the lift coefficient is 0.217. These values match well with the literature. The Karman vortices being shed from the cylinder are displayed in Figure 3.4(c). This simulation illustrates the capability to simulate two-dimensional flows around stationary objects employing the parallel version of

ELAFINT3D. The computational time for simulating 50,000 timesteps or 100 non-dimensional time units is approximately 8 hours.

Flow around a transversely oscillating cylinder

The flow around a circular cylinder, oscillating transversely with amplitude of 0.25D and frequency of 0.2 at Re = 200 has been simulated to validate serial version of ELAFINT3D. For the above combination of amplitude and frequency, the simulations and Koopmann's experiments predicted lock-on phenomena. The aim of the current simulation is to validate the parallel algorithm for 2D moving boundary problems by reproducing the lock-on behavior. The computational setup for this case is similar to that of the stationary cylinder. The base mesh employed in this case is 150x150. The calculations are performed on 4 processors with 5 levels of refinement. Figure 3.5(a) illustrates the vortex shedding from the oscillating cylinder. The time-history of lift coefficient on the cylinder is used to calculate a Strouhal number of 0.2 in this case. The wall clock time for simulating 50000 timesteps of this case is 15 hours.

Flow around a stationary sphere at Re= 300

One of the canonical simulations used to validate three-dimensional flow capability of ELAFINT3D was flow around sphere. The flow around sphere at a Reynolds number of 300 generates unsteady vortex shedding. The classical hair pin vortex structures are evident for this Reynolds number. The current study aims to validate the parallel algorithm for three-dimensional flows by reproducing the above result. The computational setup consists of a sphere of diameter 1.0 located at (6, 7.5, 7.5) in a domain of 30x15x15. The base mesh is 84x42x42 with 5 levels of refinement for LMR. The number of mesh points at the beginning of the simulation after refinement are

318589 and at the end of the simulation with the wake being refined are 753800. A slice of the center plane shown in Figure 3.6(a) illustrates that the refinement has occurred only as required in the wake of the sphere. The mesh away from the sphere and away from the wake has not been refined because of low velocity gradients. The hairpin vortices being shed from the sphere are illustrated in Figure 3.7(a-c) by the iso-surfaces of λ_2. A mean drag coefficient of 0.658 has been calculated in this case. The Strouhal number is 0.136. These values compare well with the study of Johnson et al, 1999. The observations from this numerical experiment illustrate the efficiency of LMR in capturing three-dimensional flows around stationary embedded objects. Also, the execution on 12 processors has made this simulation possible in 5 days for 12000 timesteps and non-dimensional time of 56 units.

Interaction of two spheres settling under gravity

The purpose of this current study is to simulate the interaction of two spheres settling under gravity. The computational setup comprises of two spheres of radius 1.0 initially located at (4,5,44) and (6,5,48) falling freely under gravity in a domain of 10x10x50. The initial base mesh is 25x25x125 with 4 levels of refinement. The non-dimensional parameters for the simulation are Froude number(Fr) = 1.0, density ratio = 1.468, moment of inertia = 2.46. From their numerical simulations, (Johnson et. al., 1996) have observed that the spheres drift, kiss and then tumble as they settle under gravity. The objective of the current study is to predict the time to collision in order to validate the current tool. The velocity profiles for the vertical velocity of the spheres and their vertical positions are show in Figure 3.8(a-b). The contours of the vertical velocity components are shown in Figure 3.9(a-b). A close up of the y-component of vorticity as the spheres approach to collide is shown in Figure 3.10(c, d). The time to collision as predicted by the current calculations is 38 non-dimensional units which is close to 40

units as predicted by Johnson et. al.(1996). The current simulation is performed on 6 processors with a maximum of 705691 mesh points at the end of simulation. The mesh refinement is illustrated in the Figure 3.10(a). The iso-contours of λ_2 reveal the vortical structures in the wake of the interacting spheres.

Flutter of an settling ellipsoid

The focus of the current numerical experiment is the study the settling of an oblate ellipsoid settling under gravity. Numerous studies, both experimental and numerical, have been done in regard to settling discs, cards, ellipsoids etc (Field et al.1997, Fonesca et al.,2005). It has been observed that the fall of these objects occurs in either settling or fluttering or tumbling regime. In the current study, the combination of parameters (Fr = 0.6 & Re = 450) chosen results in a fluttering motion as the ellipsoid settles under gravity. The computational setup consists of an oblate ellipsoid with an aspect ratio of 7.5 located at (15,25,140) in a (50,50,150) domain. The initial orientation of the normal of the ellipsoid is at 63.4^o with the vertical axis. A base mesh of 63x63x175 is used with 3 levels of refinement are used in this simulation. The computations are performed on 10 processors with about 1.5 million grid points at the end of the simulation. The refinement pattern is shown in the Figure 3.11(b). The vortical structures generated by the fluttering motion of the ellipsoid are illustrated by the λ_2 iso-surfaces. The graphs shown in Figure 3.13(a-c) show the x and z- components of the velocity and also the y-component of the angular velocity of the ellipsoid. From these plots the oscillatory behavior of the motion is apparent. This application illustrates the ability of the current tool to simulate flows involving arbitrary geometries undergoing complex motions. The execution of the parallel version of ELAFINT3D in conjunction with LMR enables one to accurately capture intricate flow features of complex three-dimensional applications.

Conclusions

A unified algorithm that can be compiled to be executed either in serial mode on a sequential machine or in a parallel mode on a series of workstations has been developed. The current framework features automatic domain decomposition method and efficient communication heuristic that results in a scalable parallel code. A parallel performance strategy has been presented. The capabilities of the current technique have been demonstrated for two and three dimensional flow situations. The timing data indicates significant speed up for both 2D and 3D cases.

N	NCELLS	Ratio	T_{total}	T_{solver}	Speed up	
					Total	Solver
1	252004	1	27.189	15.403	-	-
2	126003	2	14.051	7.938	1.93	1.81
4	64480	3.9	7.083	3.980	3.84	3.88
8	31636	7.96	3.694	2.036	7.36	7.56
12	21520	11.7	2.528	1.415	10.76	10.88
24	10747	23.4	1.312	0.709	20.73	21.72

Table 3.1 CPU times for flow around a cylinder with increasing number of CPUs

N	NCELLS	Ratio	T_{output}	Speedup
1	252004	1	5.89	15.403
2	126003	2	3.06	1.92
4	64480	3.9	1.55	3.80
8	31636	7.96	0.83	7.10
12	21520	11.7	0.56	10.52
24	10747	23.4	0.27	21.81

Table 3.2 CPU times for data output in case of flow around a cylinder.

Mesh	NCELLS	RATIO	T_{total}	T_{solver}	Speed up	
					Total	Solver
50 x 50	219	387.11	0.080	0.056	122.25	100.7
100 x 100	876	96.77	0.12	0.081	81.51	69.64
250 x 250	5339	15.88	0.59	0.390	16.58	14.46
500 x 500	21520	3.94	2.528	1.415	3.87	3.99
1000x1000	84776	1.0	9.78	5.641	1	1

Table 3.3 CPU times for flow around a cylinder with varying mesh densities.

N	NCELLS	NCELLS (GHOSTS)	Ratio	T_{LMR}	T_{solver}	Speed up	
						LMR	Solver
1	67389	67389	1	4.32	6.35	-	-
2	32184	33908	1.05	2.33	3.33	1.85	1.91
4	16173	18124	1.12	1.22	1.68	3.55	3.78
8	8828	11081	1.35	0.74	0.86	5.82	7.43
12	5465	7682	1.40	0.50	0.59	8.58	10.76

Table 3.4 Performance analysis of LMR with varying no. of processors.

NLEV	NCELLS	NCELLS (GHOSTS)	NCELLS (RATIO)	T_{LMR}	TIME (RATIO)
1	3332	3482	1.0	0.14	1.0
2	3687	3945	1.13	0.16	1.15
3	4238	4673	1.34	0.21	1.5
4	4976	6120	1.76	0.31	2.21
5	5465	7682	2.21	0.50	3.57

Table 3.5 Performance analysis of LMR with increasing refinement levels.

N	T_{total}	T_{solver}	Speed up	
			Total	Solver
1	89.64	63.62	-	-
2	41.18	33.14	1.90	1.92
4	23.47	17.01	3.82	3.74
8	12.05	8.35	7.44	7.62
12	8.09	5.80	11.08	10.97
24	4.16	2.87	21.56	22.18

Table 3.6 Timing data for flow around a sphere with a mesh size of 100x100x100.

N	T_{GENLS}	SpeedUp
1	5.66	1
2	2.90	1.95
4	1.44	3.92
8	0.72	7.86

Table 3.7 Parallel performance for rendering a sphere using GENLS.

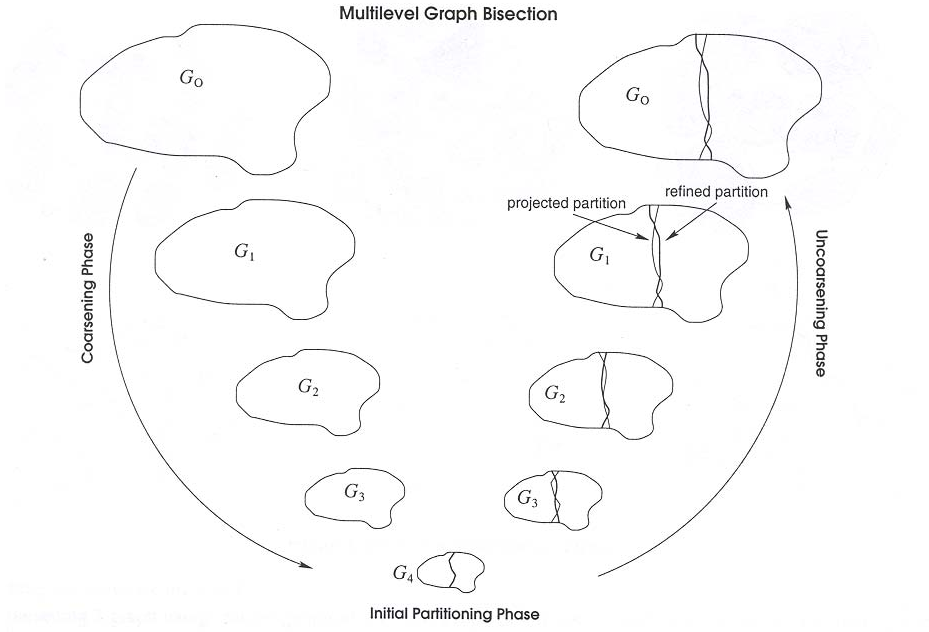


Figure 3.1 Illustration of the three phases of multilevel graph.

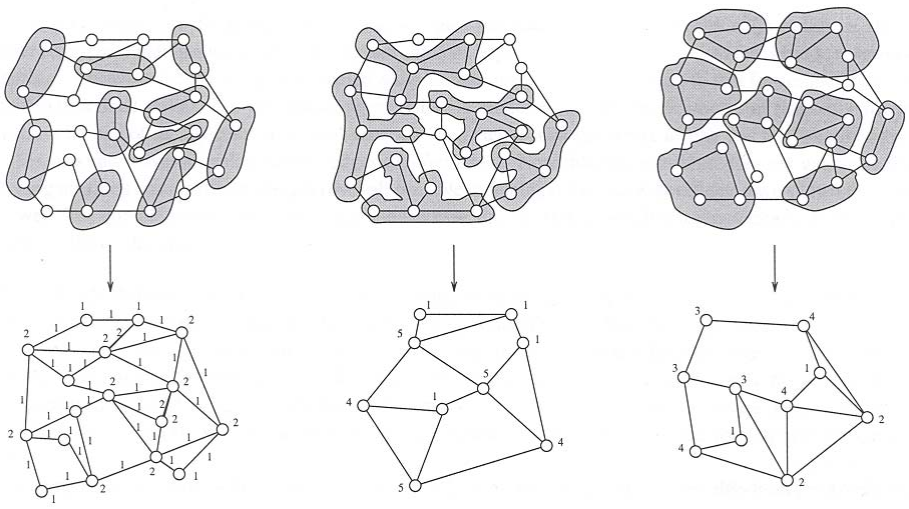


Figure 3.2 Different ways to coarsen a graph

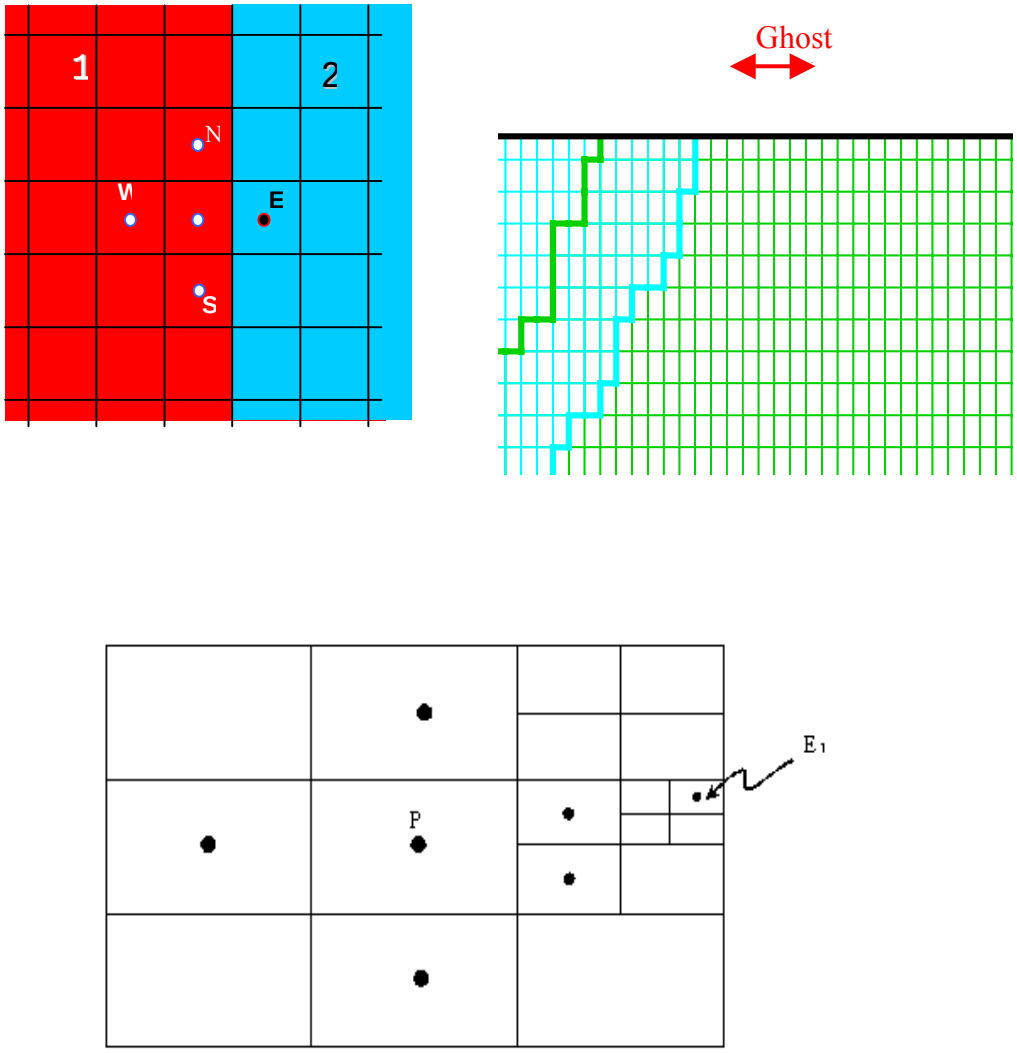


Figure 3.3 Illustration of Ghost Region. (a) Discretization of a mesh point next to partition boundary (b) Extent of Ghost Region near partition boundary (c) Discretization of mesh point at the mesh interface

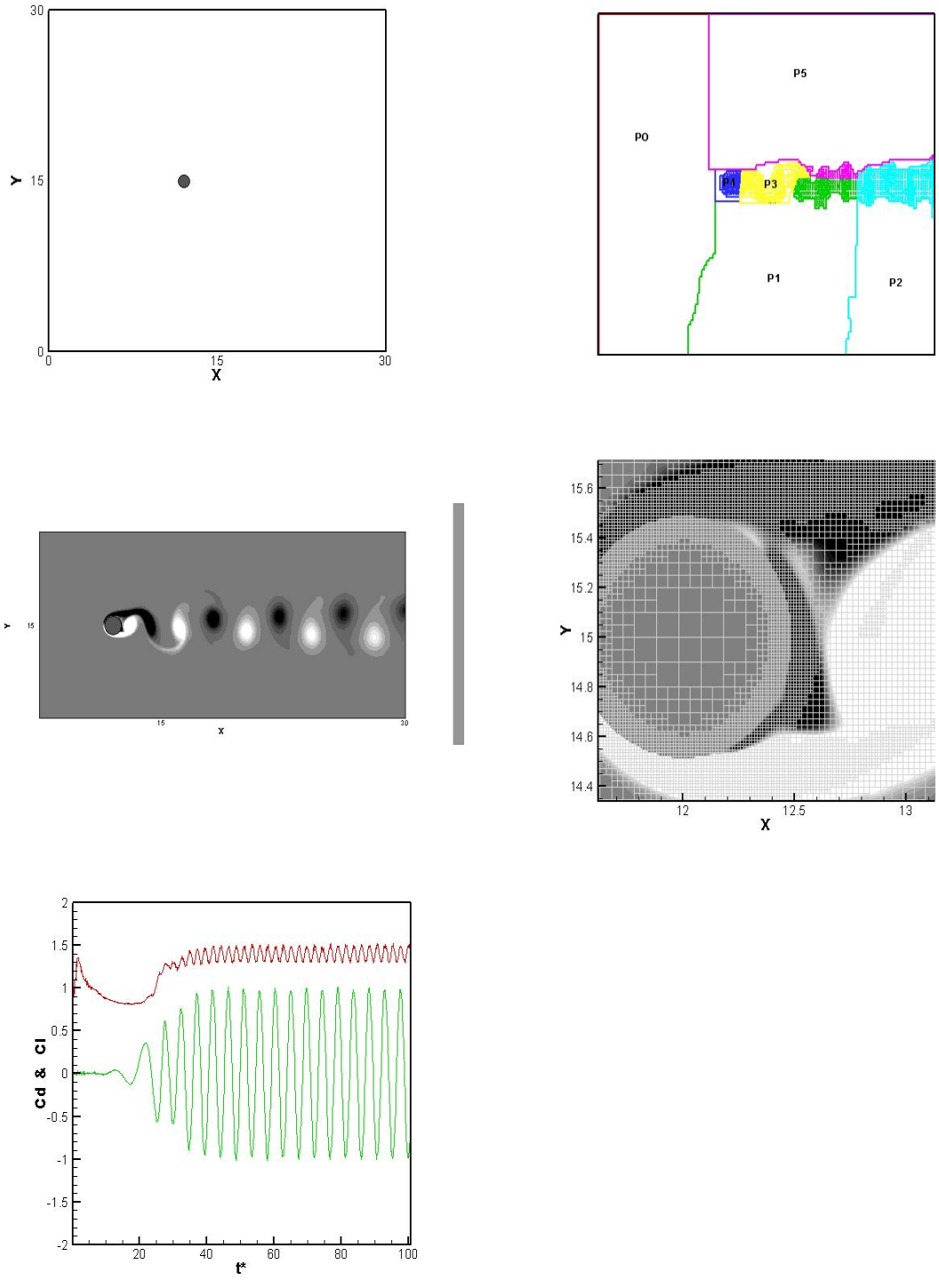


Figure 3.4 Flow around stationary cylinder at Re = 300. a) Computational Setup b) Domain decomposition c) Unsteady vortex shedding d) Close up view of the cylinder with mesh refinement. e) Time history of drag and lift coefficients.

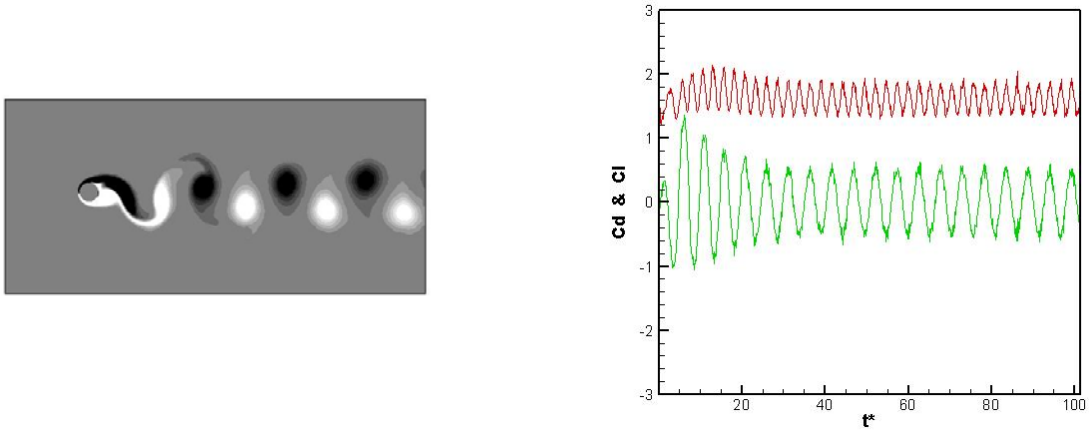


Figure 3.5 Flow around an transversely oscillating cylinder a) Vortex shedding from the oscillating cylinder. b) The time history of drag and lift coefficients.

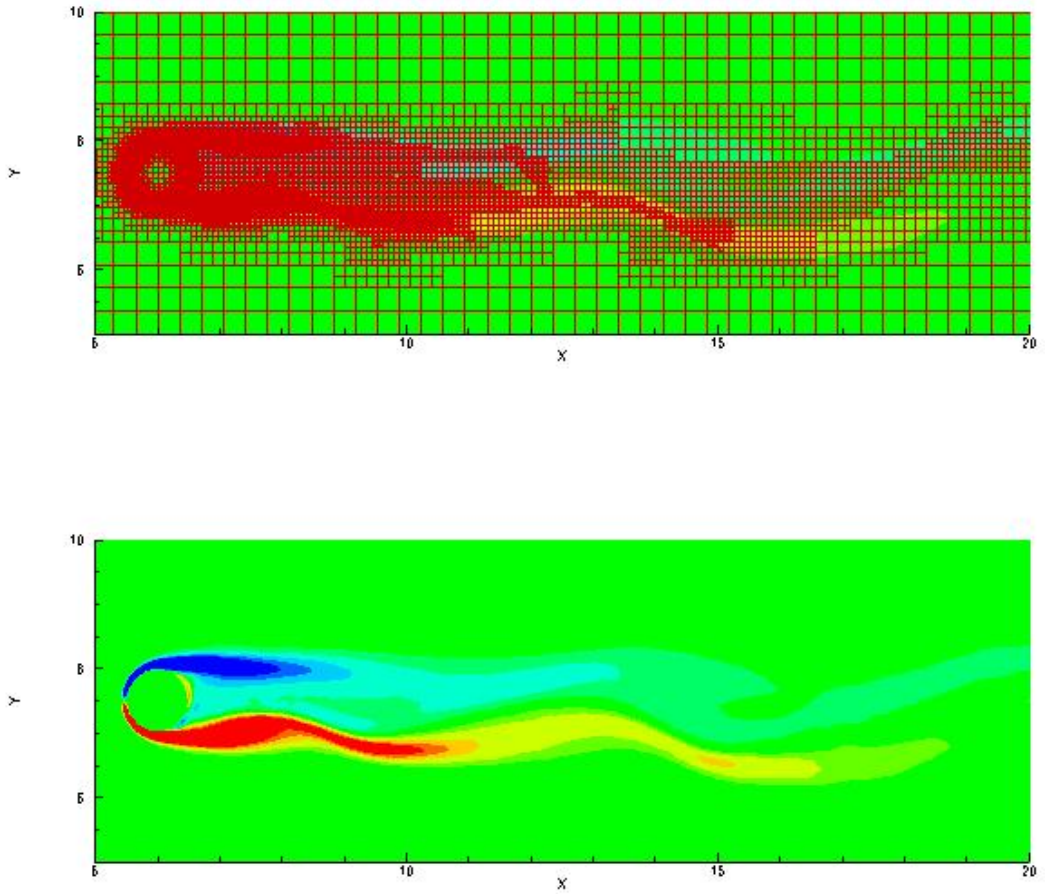


Figure 3.6 Flow features on z= 7.5 plane.for flow around a sphere. a) The adaptively refined mesh in the wake of a sphere b) Contours of z-component of vorticity

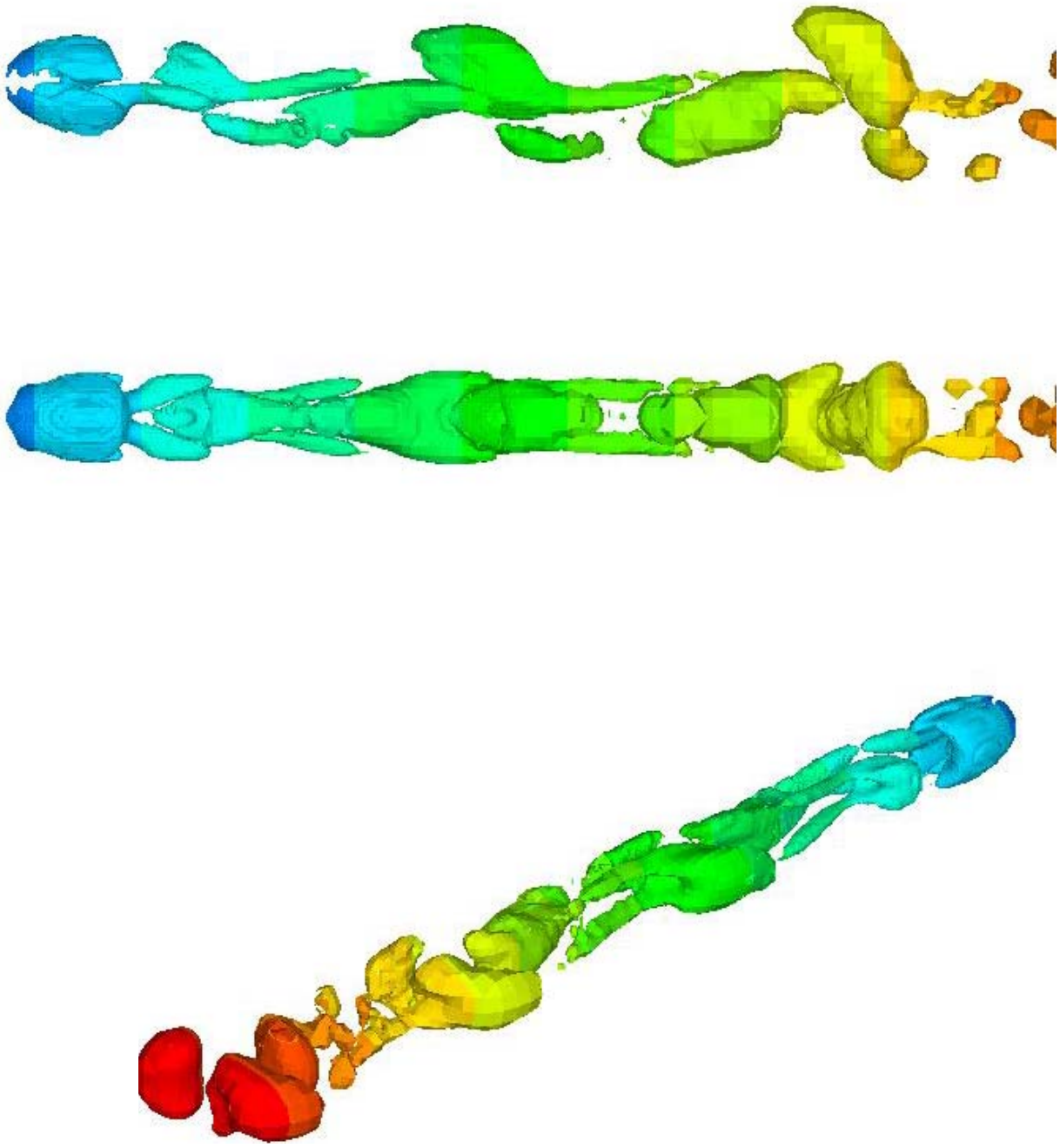


Figure 3.7 Vortical Hairpin structures in the wake of a sphere a) x-y view b_ x-z view and c) isometric view. vorticity

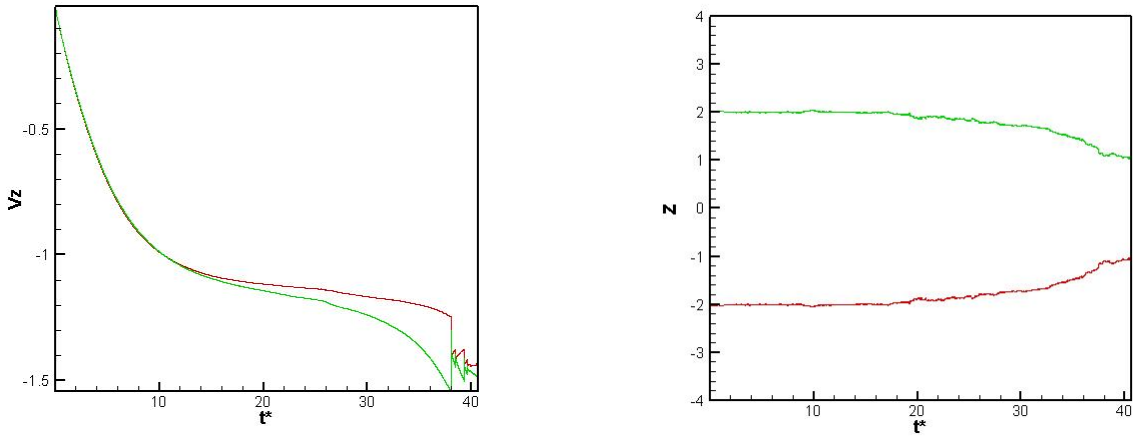


Figure 3.8 The vertical velocity profile and the positions of the spheres. a) Vertical velocity profile with collision. b) The vertical position of the spheres in time.

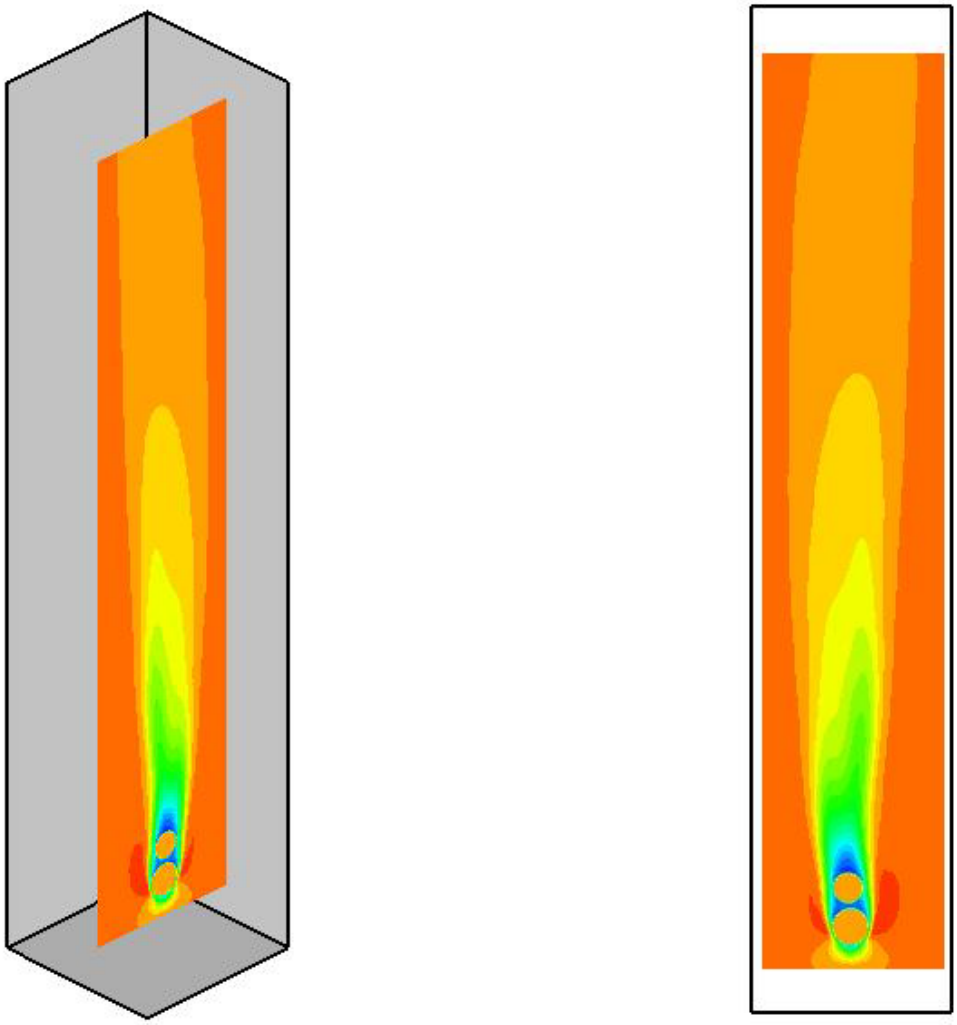


Figure 3.9 Flow around two interacting sphere on y= 5.at the instant of collision a) The vertical velocity contours on the plane in isometric view b) The xz view of y= 5.0 plane depicting the velocity contours.

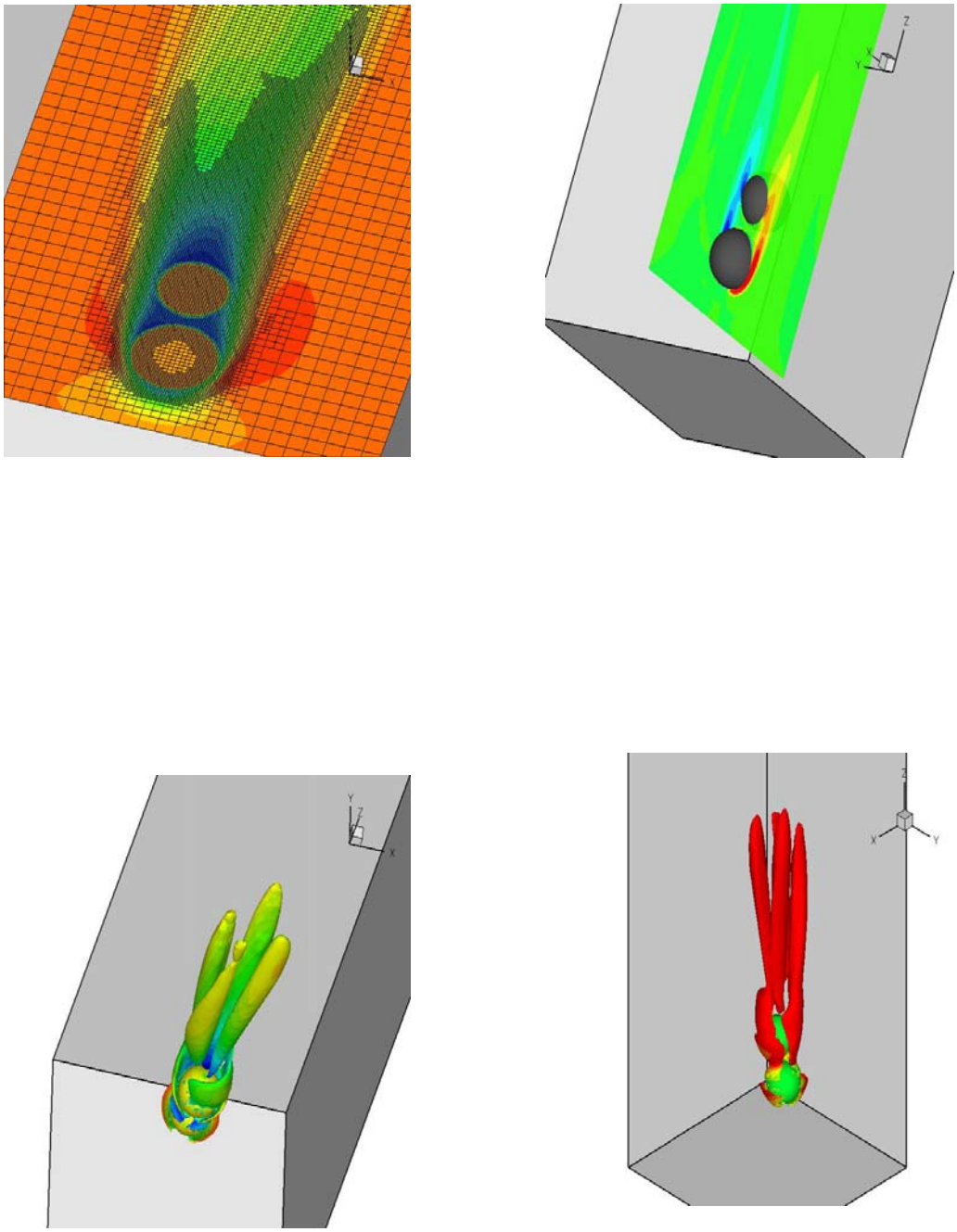


Figure 3.10 Flow around two interacting sphere at the instant of collision a) The mesh refinement in the wake of the spheres. b) The position of the two spheres relative to each other and also relative to y = 0.5 plane. c) The iso-contours of λ_2 indicating the vortical structures. d) An isometric view of the vortical structures.

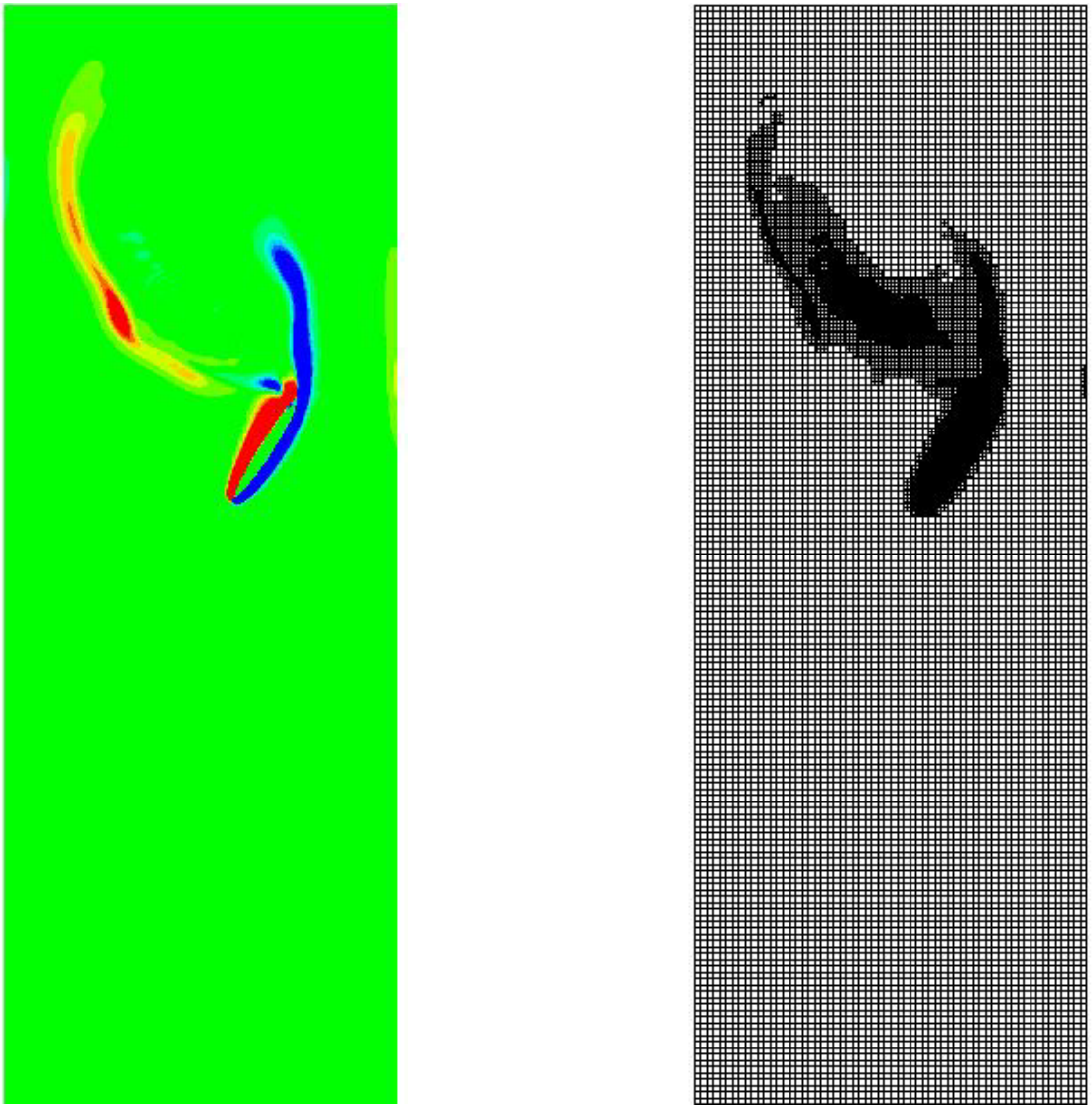


Figure 3.11 Flow around an oblate ellipsoid fluttering under gravity plotted on y = 25 plane. a) The contours of y-component of vorticity b) The adaptive mesh created in regions of high vorticity.

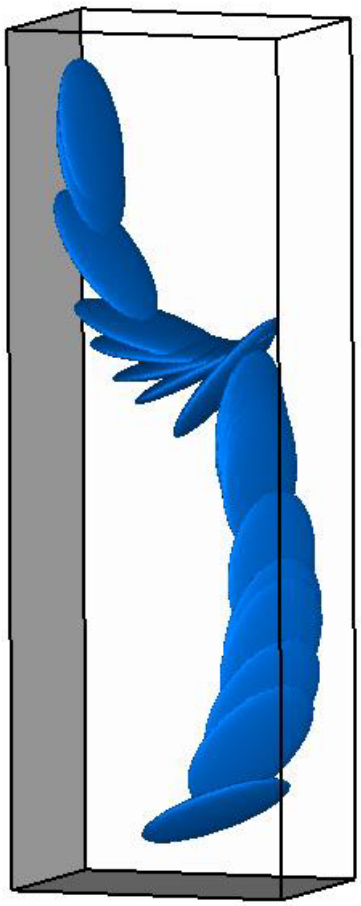


Figure 3.12 The trajectory of the oblate settling under gravity.

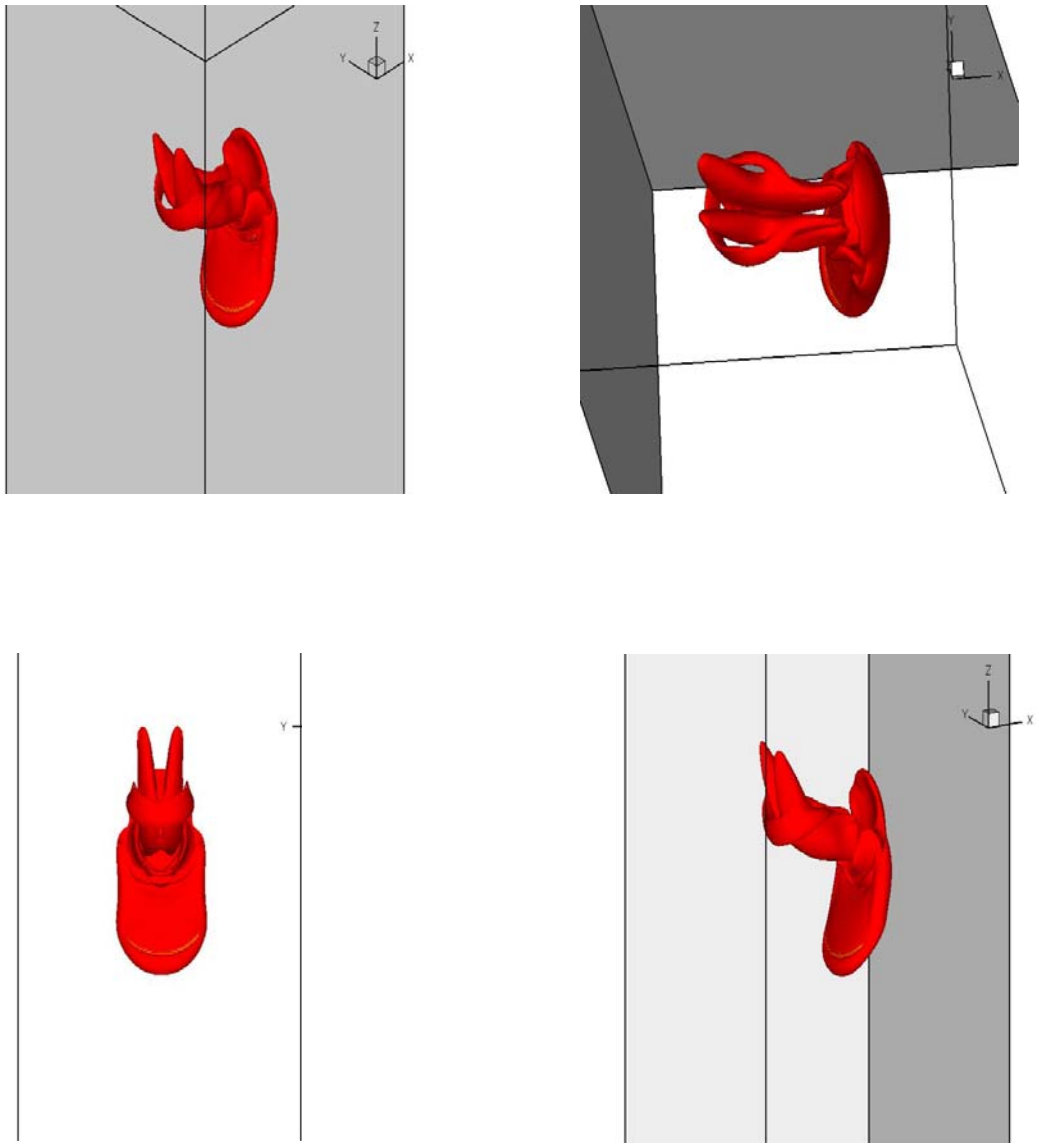


Figure 3.13 Different views of the vortical structures emanating from a settling ellipsoid.

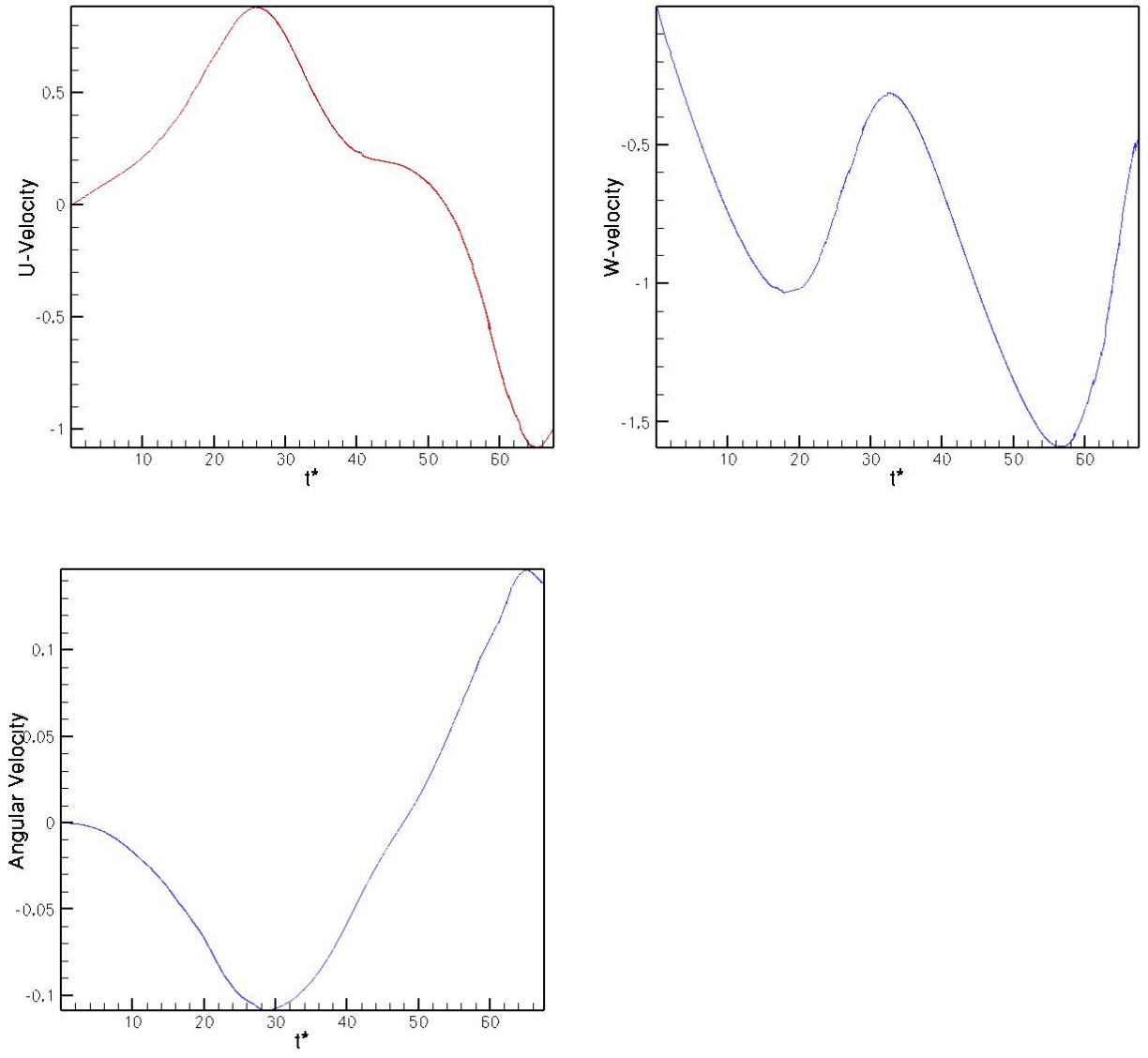


Figure 3.14 The velocity components of the ellipsoid as is moves with fluid forces a) The vertical fall velocity, b) The transverse velocity developed mainly due to the initial orientation and inertia, and c) The angular velocity of the ellipsoid as it rotates due to the fluid dynamic moments

SUMMARY AND FUTURE WORK

Summary

The objective of this thesis was to develop a framework capable of solving a wide variety of moving boundary problems in fluid mechanics. The applications of interest are biomedical applications such as dynamics of heart valves, peristaltic motion in GI tract, or material processing applications such as interaction of an growing dendrite with its own melt and other embedded ceramic particles or general fluid-structure interaction applications such as falling of leaves from trees or flow around bluff bodies.

The numerical tool that can solve these problems has to be capable of capturing the interface-flow interactions and also solving the complex flow phenomena. In the current thesis, these features are facilitated by developing the ELAFINT3D framework. This framework entails a) a Cartesian grid based fluid flow solver b) a Level-set algorithm for interface tracking c) Sharp-Interface Method to capture the interfacial dynamics d) LMR to improve the accuracy of the solution as required e) Sparse matrix solvers to boost the efficiency of the method and finally f) MPI based parallel algorithm to enable the solution of large scale problems.

The main contributions of the current thesis are a) to conceive and formulate the sharp interface method to capture the fluid-object interactions b) to implement the numerical technique in a computer code ELAFINT3D c) to increase the efficiency of the tool by implementing sparse matrix solvers, d) to develop parallel heuristics of ELAFINT3D, thereby enabling large scale computations. By developing the above components and employing LMR, the current ELAFINT3D tool is capable of solving large scale three-dimensional moving boundary problems.

This general purpose tool has been employed to solve a variety of problems. A set of canonical problems such as flows around cylinders and spheres have been simulated for validation purposes. The scaling analysis of the parallel code is performed. This analysis has shown the individual components, (namely solver, LMR, GENLS) and all of them in conjunction perform well on a distributed memory architecture.

More complicated scenarios such as settling of two spheres and fluttering motion of an ellipsoid have been simulated to demonstrate the strengths of the tool. These simulations have resulted in accurate results with reasonable computational expense.

Future Work

The numerical experiments performed in the current thesis have demonstrated the capability of the framework to solve large scale, three-dimensional moving boundary problems. The various components incorporated in the ELAFINT3D framework such as Sharp Interface Method, LMR, GENLS etc., have been thoroughly analyzed individually and in combination with other components of the framework.

However, in an integrated framework such as ELAFINT3D, the performance is dependent on the efficiency of the individual components as well as their interactions. Also, the applications of interest are large scale multi-physics problems in general. Many a times, the efficiency of a particular approach/implementation could be more suitable to a certain class of applications than others. Especially in parallel implementation of ELAFINT3D, the optimization of load among the processors for various phases of the

calculations is particularly difficult. However, load imbalance in any stage of the simulation results in an inefficient overall calculation. The current parallel implementation seeks improvements in this regard. The following list elucidates the specific issues that could have an immediate effect on the parallel performance of ELAFINT3D:

a) With LMR in place, during domain decomposition phase of the base mesh, the regions of refinement are not known apriori. However, the initial performance of LMR depends critically on the distribution of refined meshes across the processors. A strategy has to be devised to automate the assignment of weights to the vertices based on their proximity to the embedded objects.

b) The definition of "ghost region" for parallel LMR, is an over estimate of the actual requirement. The current scheme either includes the whole tree structure of a given base mesh. This extra storage required could be critical in cases of higher levels of refinements and in 3D

c) The size of the mapping data structures in the current implantation are of the global mesh size. These arrays are stored on each processor. The memory requirement to store these arrays can be unreasonable for large three dimensional problems.

d) Another key component of ELAFINT3D, the Lagrangian Particle Tracking has not be parallelized in the current thesis. The multi weight options in ParMetis can be exploited to create well balanced loads for both fluid and particle meshes. But this is another situation that requires optimization due to the multi-physics involved in the application itself.

In conclusion, an efficient and general purpose tool capable of tackling a variety of moving boundary problems has been developed. The numerical techniques employed in this tool are easily to formulate and implement. The accuracy and efficiency of the

current tool has been presented through a series of benchmark simulations. A high performance computing capability has been incorporated into ELAFINT3D framework. Several large scale three dimensional applications have been simulated with this framework. The performance of the tool for these problems provides great promise. By fine-tuning some of the implementation aspects, this tool can be used to solve many complex multi-scale problems.

REFERENCES

Adalsteinsson, D., and Sethian, J. A. (1995). "A Fast Level Set Method for Propagating Interfaces." *Journal of Computational Physics*, 118(2), 269-277.

Almgren, A. S., Bell, J. B., Colella, P., and Marthaler, T. (1997). "A Cartesian grid projection method for the incompressible Euler equations in complex geometries." *Siam Journal on Scientific Computing*, 18(5), 1289-1309.

Al-Rawahi, N., and Tryggvason, G. (2002). "Numerical simulation of dendritic solidification with convection: Two-dimensional geometry." *Journal of Computational Physics*, 180(2), 471-496.

Anderson, C. E., Hohler, V., Walker, J. D., and Stilp, A. J. (1995). "Time-Resolved Penetration of Long Rods into Steel Targets." *International Journal of Impact Engineering*, 16(1), 1-18.

Anderson, D. M., McFadden, G. B., and Wheeler, A. A. (1998). "Diffuse-interface methods in fluid mechanics." *Annual Review of Fluid Mechanics*, 30, 139-165.

Anderson, D. M., McFadden, G. B., and Wheeler, A. A. (2000). "A phase-field model of solidification with convection." *Physica D*, 135(1-2), 175-194.

Balaras, E. (2004). "Modeling complex boundaries using an external force field on fixed Cartesian grids in large-eddy simulations." *Computers & Fluids*, 33(3), 375-404.

Barth, T. J., and Sethian, J. A. (1998). "Numerical schemes for the Hamilton-Jacobi and level set equations on triangulated domains." *Journal of Computational Physics*, 145(1), 1-40.

Bayyuk, S. A., Powell, K. G., and (1993)., B. v. L. "A simulation technique for 2-D unsteady inviscid flows around arbitrarily moving and deforming bodies of arbitrary geometry."

Benson, D. J. (1995). "A Multimaterial Eulerian Formulation for the Efficient Solution of Impact and Penetration Problems." *Computational Mechanics*, 15(6), 558-571.

Brackbill, J. U., Kothe, D. B., and Zemach, C. (1992). "A Continuum Method for Modeling Surface-Tension." *Journal of Computational Physics*, 100(2), 335-354.

Brezina, M., Cleary, A. J., Falgout, R. D., Henson, V. E., Jones, J. E., Manteuffel, T. A., McCormick, S. F., and Ruge, J. W. (2001). "Algebraic multigrid based on element interpolation (AMGE)." *Siam Journal on Scientific Computing*, 22(5), 1570-1592.

Cheng, T., and Peskin, C. S. (1992). "Stability and Instability in the Computation of Flows with Moving Immersed Boundaries - a Comparison of 3 Methods." *Siam Journal on Scientific and Statistical Computing*, 13(6), 1361-1376.

Chessa, J., and Belytschko, T. (2003). "An extended finite element method for two-phase fluids." *Journal of Applied Mechanics-Transactions of the Asme*, 70(1), 10-17.

Chessa, J., Smolinski, P., and Belytschko, T. (2002). "The extended finite element method (XFEM) for solidification problems." *International Journal for Numerical Methods in Engineering*, 53(8), 1959-1977.

Chong, M. S., Perry, A. E., and Cantwell, B. J. (1990). "A general classification of three-dimensional flow fields." *Physics of Fluids*, 2(A), 765.

Chopp, D. L. (2001). "Some improvements of the fast marching method." *Siam Journal on Scientific Computing*, 23(1), 230-244.

Cleary, A. J., Falgout, R. D., Henson, V. E., Jones, J. E., Manteuffel, T. A., McCormick, S. F., Miranda, G. N., and Ruge, J. W. (2000). "Robustness and scalability of algebraic multigrid." *Siam Journal on Scientific Computing*, 21(5), 1886-1908.

Dolbow, J., Moes, N., and Belytschko, T. (2001). "An extended finite element method for modeling crack growth with frictional contact." *Computer Methods in Applied Mechanics and Engineering*, 190(51-52), 6825-6846.

Fadlun, E. A., Verzicco, R., Orlandi, P., and Mohd-Yusof, J. (2000). "Combined immersed-boundary finite-difference methods for three-dimensional complex flow simulations." Journal of Computational Physics, 161(1), 35-60.

Fedkiw, R., and Liu, X. D. (1998). *The Ghost fluid method for viscous flows, progress in numerical solutions of partial differential equations*, Arachon, France.

Fedkiw, R. P., Aslam, T., Merriman, B., and Osher, S. (1999). "A non-oscillatory Eulerian approach to interfaces in multimaterial flows (the ghost fluid method)." *Journal of Computational Physics*, 152(2), 457-492.

Field, S.B. M. Klaus, M. G. Moore, and Franco Nori (1997). " Chaotic Dynamics of Falling Disks", v 388, n 6639, p 252-4.

Fonseca, F. , Herrmann, H.J. (2005), " Simulation of the sedimentation of a falling oblate ellipsoid" , Physica A, v 345, n 3-4, p 341-55.

Glimm, J., Graham, M. J., Grove, J., Li, X. L., Smith, T. M., Tan, D., Tangerman, F., and Zhang, Q. (1998). "Front tracking in two and three dimensions." *Computers & Mathematics with Applications*, 35(7), 1-11.

Glimm, J., Grove, J., Lindquist, B., Mcbryan, O. A., and Tryggvason, G. (1988). "The Bifurcation of Tracked Scalar Waves." *Siam Journal on Scientific and Statistical Computing*, 9(1), 61-79.

Glowinski, R., Pan, T. W., Hesla, T. I., Joseph, D. D., and Periaux, J. (1999). "A distributed Lagrange multiplier/fictitious domain method for flows around moving rigid bodies: Application to particulate flow." *International Journal for Numerical Methods in Fluids*, 30(8), 1043-1066.

Glowinski, R., Pan, T. W., Hesla, T. I., Joseph, D. D., and Periaux, J. (2001). "A fictitious domain approach to the direct numerical simulation of incompressible viscous flow past moving rigid bodies: Application to particulate flow." *Journal of Computational Physics*, 169(2), 363-426.

Griebel, M., Neunhoeffer, T., and Regler, H. (1998). "Algebraic multigrid methods for the solution of the Navier-Stokes equations in complicated geometries." *International Journal for Numerical Methods in Fluids*, 26(3), 281-301.

Haase, G., Kuhn, M., and Reitzinger, S. (2002). "Parallel algebraic multigrid methods on distributed memory computers." *Siam Journal on Scientific Computing*, 24(2), 410-427.

Henson, V. E., and Yang, U. M. (2002). "BoomerAMG: A parallel algebraic multigrid solver and preconditioner." *Applied Numerical Mathematics*, 41(1), 155-177.

Jayaraman, V., Udaykumar, H. S., and Shyy, W. S. (1997). "Adaptive unstructured grid for three-dimensional interface representation." *Numerical Heat Transfer Part B-Fundamentals*, 32(3), 247-265.

Johnson A. A., Tezduyar, T.E., (1996). "Simulation of multiple spheres falling in a liquid-filled tube".Computer Methods in Applied Mechanics and Engineering, v 134, n 3-4, 1, p 351-73

Johnson, A. A., and Tezduyar, T. E. (1999). "Advanced mesh generation and update methods for 3D flow simulations." *Computational Mechanics*, 23(2), 130-143.

Johnson, G. R., Beissel, S. R., and Stryk, R. A. (2002). "An improved generalized particle algorithm that includes boundaries and interfaces." *International Journal for Numerical Methods in Engineering*, 53(4), 875-904.

Johnson, T. A., and Patel, V. C. (1999). "Flow past a sphere up to a Reynolds number of 300." *Journal of Fluid Mechanics*, 378, 19-70.

Kang, M., and Fedkiw, R. P. (2002). "A boundary condition capturing method for multiphase incompressible flow." *J. Sci. Comput.*, 15, 323-360.

Kim, J., Kim, D., and Choi, H. (2001). "An immersed-boundary finite-volume method for simulations of flow in complex geometries." *Journal of Computational Physics*, 171(1), 132-150.

Koopmann, G. H. (1967). "The vortex wakes of vibrating cylinders at low Reynolds numbers." *J.Fluid Mech.*, 28, 501-.

Lai, M. C., and Li, Z. L. (2001). "A remark on jump conditions for the three-dimensional Navier-Stokes equations involving an immersed moving membrane." *Applied Mathematics Letters*, 14(2), 149-154.

Lai, M. C., and Peskin, C. S. (2000). "An immersed boundary method with formal second-order accuracy and reduced numerical viscosity." *Journal of Computational Physics*, 160(2), 705-719.

Lee, L., and Leveque, R. J. (2003). "An immersed interface method for incompressible Navier-Stokes equations." *Siam Journal on Scientific Computing*, 25(3), 832-856.

Leveque, R. J., and Li, Z. L. (1994). "The Immersed Interface Method for Elliptic-Equations with Discontinuous Coefficients and Singular Sources." *Siam Journal on Numerical Analysis*, 31(4), 1019-1044.

Leveque, R. J., and Li, Z. L. (1995). "The Immersed Interface Method for Elliptic-Equations with Discontinuous Coefficients and Singular Sources (Vol 31, Pg 1019, 1994)." *Siam Journal on Numerical Analysis*, 32(5), 1704-1704.

Li, Z. L., and Lai, M. C. (2001). "The immersed interface method for the Navier-Stokes equations with singular forces." *Journal of Computational Physics*, 171(2), 822-842.

Lin, Q., Boyer, D. L., and Fernando, H. J. S. (1994). "The Vortex Shedding of a Streamwise-Oscillating Sphere Translating through a Linearly Stratified Fluid." *Physics of Fluids*, 6(1), 239-252.

Lin, Q., Lindberg, W. R., Boyer, D. L., and Fernando, H. J. S. (1992). "Stratified Flow Past a Sphere." *Journal of Fluid Mechanics*, 240, 315-354.

Liu, X. D., Fedkiw, R. P., and Kang, M. J. (2000). "A boundary condition capturing method for Poisson's equation on irregular domains." *Journal of Computational Physics*, 160(1), 151-178.

Magarvey, R. H., and Bishop, R. L. (1961). "Transition ranges for three-dimensional wakes." *Can.J.Phys.*, 39, 1418-1422.

Marella, S. V., and Udaykumar, H. S. (2004). "Computational analysis of the deformability of leukocytes modeled with viscous and elastic structural components." *Physics of Fluids*, 16(2), 244-264.

Monaghan, J. J., and Kocharyan, A. (1995). "Sph Simulation of Multiphase Flow." *Computer Physics Communications*, 87(1-2), 225-235.

Osher, S., and Fedkiw, R. P. (2001). "Level set methods: An overview and some recent results." *Journal of Computational Physics*, 169(2), 463-502.

Osher, S., and Sethian, J. A. (1988). "Fronts Propagating with Curvature-Dependent Speed - Algorithms Based on Hamilton-Jacobi Formulations." *Journal of Computational Physics*, 79(1), 12-49.

Patankar, N. A., Singh, P., Joseph, D. D., Glowinski, R., and Pan, T. W. (2000). "A new formulation of the distributed Lagrange multiplier/fictitious domain method for particulate flows." *International Journal of Multiphase Flow*, 26(9), 1509-1524.

Pember, R. B., Bell, J. B., Colella, P., Crutchfield, W. Y., and Welcome, M. L. (1995). "An Adaptive Cartesian Grid Method for Unsteady Compressible Flow in Irregular Regions." *Journal of Computational Physics*, 120(2), 278-304.

Peskin, C. S. (1977). "Numerical analysis of blood flow in the heart." *J. Comp. Phys*, 25(220-243).

Popinet, S. (2003). "Gerris: a tree-based adaptive solver for the incompressible Euler equations in complex geometries." *Journal of Computational Physics*, 190(2), 572-600.

Quirk, J. J. (1994). "An Alternative to Unstructured Grids for Computing Gas-Dynamic Flows around Arbitrarily Complex 2-Dimensional Bodies." *Computers & Fluids*, 23(1), 125-142.

Roma, A. M., Peskin, C. S., and Berger, M. J. (1999). "An adaptive version of the immersed boundary method." *Journal of Computational Physics*, 153(2), 509-534.

Sethian, J. A. (1999). "Fast marching methods." *Siam Review*, 41(2), 199-235.

Sethian, J. A. (2001). "Evolution, implementation, and application of level set and fast marching methods for advancing fronts." *Journal of Computational Physics*, 169(2), 503-555.

Sethian, J. A., and Smereka, P. (2003). "Level set methods for fluid interfaces." *Annual Review of Fluid Mechanics*, 35, 341-372.

Shyy, W., Pal, S., Udaykumar, H. S., and Choi, D. (1998). "Structured moving grid and geometric conservation laws for fluid flow computation." *Numerical Heat Transfer Part a-Applications*, 34(4), 369-397.

Stockie, J. M., and Wetton, B. R. (1999). "Analysis of stiffness in the immersed boundary method and implications for time-stepping schemes." *Journal of Computational Physics*, 154(1), 41-64.

Strain, J. (1999). "Fast tree-based redistancing for level set computations." *Journal of Computational Physics*, 152(2), 664-686.

Strain, J. (2001). "A fast semi-Lagrangian contouring method for moving interfaces." *Journal of Computational Physics*, 170(1), 373-394.

Sukumar, N., Chopp, D. L., Moes, N., and Belytschko, T. (2001). "Modeling holes and inclusions by level sets in the extended finite-element method." *Computer Methods in Applied Mechanics and Engineering*, 190(46-47), 6183-6200.

Sukumar, N., Moes, N., Moran, B., and Belytschko, T. (2000). "Extended finite element method for three-dimensional crack modelling." *International Journal for Numerical Methods in Engineering*, 48(11), 1549-1570.

Sussman, M., and Fatemi, E. (1999). "An efficient, interface-preserving level set redistancing algorithm and its application to interfacial incompressible fluid flow." *Siam Journal on Scientific Computing*, 20(4), 1165-1191.

Sussman, M., Fatemi, E., Smereka, P., and Osher, S. (1998). "An improved level set method for incompressible two-phase flows." *Computers & Fluids*, 27(5-6), 663-680.

Tezduyar, T. E. (2001). "Finite element methods for flow problems with moving boundaries and interfaces." Archives of Computational Methods in Engineering, 8(2), 83-130.

Tomboulides, A. G., Orszag, S. A., and Karniadakis, G. E. "Direct and large-eddy simulation of axisymmetric wakes." *AIAA Paper*.

Tryggvason, G., Bunner, B., Esmaeeli, A., and Al-Rawahi, N. (2003). "Computations of multiphase flows." *Advances in Applied Mechanics, Vol 39*, 39, 81-120.

Tseng, Y. H., and Ferziger, J. H. (2003). "A ghost-cell immersed boundary method for flow in complex geometry." *Journal of Computational Physics*, 192(2), 593-623.

Udaykumar, H. S., and Mao, L. (2002). "Sharp-interface simulation of dendritic solidification of solutions." *International Journal of Heat and Mass Transfer*, 45(24), 4793-4808.

Udaykumar, H. S., Mao, L., and Mittal, R. (2002a). "A finite-volume sharp interface scheme for dendritic growth simulations: Comparison with microscopic solvability theory." *Numerical Heat Transfer Part B-Fundamentals*, 42(5), 389-409.

Udaykumar, H. S., Marella, S., and Krishnan, S. (2003). "Sharp-interface simulation of dendritic growth with convection: benchmarks." *International Journal of Heat and Mass Transfer*, 46(14), 2615-2627.

Udaykumar, H. S., Mittal, R., and Rampunggoon, P. (2002b). "Interface tracking finite volume method for complex solid-fluid interactions on fixed meshes." *Communications in Numerical Methods in Engineering*, 18(2), 89-97.

Udaykumar, H. S., Mittal, R., Rampunggoon, P., and Khanna, A. (2001). "A sharp interface cartesian grid method for simulating flows with complex moving boundaries." *Journal of Computational Physics*, 174(1), 345-380.

Udaykumar, H. S., Mittal, R., and Shyy, W. (1999). "Computation of solid-liquid phase fronts in the sharp interface limit on fixed grids." *J. Comput. Phys.*, 153(2), 535-574.

Unverdi, S. O., and Tryggvason, G. (1992). "A Front-Tracking Method for Viscous, Incompressible, Multi-Fluid Flows." *Journal of Computational Physics*, 100(1), 25-37.

Wagner, C. (1998). "Introduction to Algebraic Multigrid."

Wang, X. D., and Liu, W. K. (2004). "Extended immersed boundary method using FEM and RKPM." *Computer Methods in Applied Mechanics and Engineering*, 193(12-14), 1305-1321.

Webster, R. (1994). "An Algebraic Multigrid Solver for Navier-Stokes Problems." *International Journal for Numerical Methods in Fluids*, 18(8), 761-780.

Webster, R. (1996). "An algebraic multigrid solver for Navier-Stokes problems in the discrete second-order approximation." *International Journal for Numerical Methods in Fluids*, 22(11), 1103-1123.

Wieselsberger, C. (1922). "New data on laws of fluid resistance." *NACA TN*, 84.

Ye, T., Mittal, R., Udaykumar, H. S., and Shyy, W. (1999). "An accurate Cartesian grid method for viscous incompressible flows with complex immersed boundaries." *Journal of Computational Physics*, 156(2), 209-240.

Yiu, K. F. C., Greaves, D. M., Cruz, S., Saalehi, A., and Borthwick, A. G. L. (1996). "Quadtree grid generation: Information handling, boundary fitting and CFD applications." *Computers & Fluids*, 25(8), 759-769.

Zang, Y., Street, R. L., and Koseff, J. R. (1994). "A Non-Staggered Grid, Fractional Step Method for Time-Dependent Incompressible Navier-Stokes Equations in Curvilinear Coordinates." *Journal of Computational Physics*, 114(1), 18-33.