

Development of Parallel CFD solver for Three Dimensional Unstructured Grid

Vatsalya Sharma

Under guidance of

Dr.Raja Banarjee

A Thesis Submitted to

Indian Institute of Technology Hyderabad

In Partial Fulfillment of the Requirements for

The Degree of Master of Technology




Department of Mechanical and Aerospace Engineering

July 2014

Declaration

I declare that this written submission represents my ideas in my own words, and where ideas or words of others have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the Institute and can also evoke penal action from the sources that have thus not been properly cited, or from whom proper permission has not been taken when needed.



(Signature)

VATSALYA SHARMA

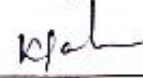
(Vatsalya Sharma)

NE12M1030

(Roll No.)

Approval Sheet

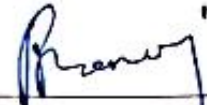
This Thesis entitled Development of Parallel CFD solver for Three Dimensional Unstructured Grid by Vatsalya Sharma is approved for the degree of Master of Technology from IIT Hyderabad



(Dr. Kirti Sahu) Examiner
Dept. of Chemical Engineering
IITH



(Prof. Vinayak Eswaran) Examiner
Dept. of Mechanical and Aerospace Engineering
IITH



(Dr. Raja Banarjee) Adviser
Dept. of Mechanical and Aerospace Engineering
IITH

Acknowledgements

The list of acknowledgements is long and never ending. Firstly, I would like to thank all my friends at SKF and COE-Roorkee, without whose support and encouragement I wouldn't have made it to this place itself. I would like to thank Ravitej for his invaluable contribution in terms of the hours spent together for understanding and implementing AMG. I would thank my SAE-BAJA team mates who helped me to understand my shortcomings, and made me a better person and engineer. I would like to thank Dr.Raja Banarjee for his consistent support and guidance. I would like to thank Prof Eswaran, for showing the right direction and for his moral support. Also I would like to acknowledge Dr. Amaresh Dalal, IIT-Guwahati for providing his code, that formed the basis of our thesis. I would like to acknowledge Dr.Naveen Sivadasan, Department of Computer Science and Engineering, IIT-Hyderabad for his support. I would also like to acknowledge my colleagues at IIT-Hyderabad for making my stay eventful.

Dedication

To my family

Abstract

The current work develops a general purpose Navier-Stokes semi implicit solver capable of handling three-dimensional unstructured grids. The flow needs to be laminar and incompressible. Species transport equation can also be solved using a segregated algorithm. Pressure Poisson equation that takes most of the solving time has been parallelized using CUDA programming language on GPU, with Algebraic Multigrid for orthogonal unstructured grids. Domain decomposition has been done using greedy colouring method. Single phase jets have been studied in presence of walls has been studied, which is of interest in Internal Combustion engines. Large Eddy simulation (LES) modeling has been employed for simulating turbulent flows using Static Smagrosnky model. Validations have been presented for turbulent round and plane jets. Laminar and turbulent coaxial jets for different velocity ratios for has been simulated and the effect of faster annular jet on the core of inner jet is analyzed and presented.

Contents

Declaration	ii
Approval Sheet	iii
Acknowledgements	iv
Abstract	vi
Nomenclature	ix
1 Introduction	1
1.1 Scope of current work	1
1.2 Literature Survey	2
1.3 Thesis Organisation	3
2 CGNS Format	4
2.1 Introduction to CGNS format	4
2.2 Unstructured Grid terminology in CGNS format	5
2.3 Processing connectivity information	7
2.4 Geometrical Properties	9
2.4.1 Cell Center	9
2.4.2 Surface Area	9
2.4.3 Volume	10
2.5 Example of Unstructured Domain	11
2.6 Closure	12
3 Governing Equations	13
3.1 Introduction	13
3.1.1 Finite Volume Method	13
3.2 Partial Differential Equations	13
3.3 Governing Equations: Integral Form	14
3.4 Discretization Procedure	14
3.4.1 Discretization of the Continuity Equation	14
3.4.2 Discretization of the General Convection-Diffusion Equation	15
3.5 Handling of terms at the Boundary	19
3.5.1 FaceCentroid Formulation	19
3.5.2 FaceNormal Formulation	21
3.5.3 Summary	23

3.6	Solution Algorithm	23
3.6.1	SemiImplicit Algorithm	25
3.7	Discretizing the Pressure Poisson Equation	26
3.8	Closure	27
4	Parallelization of Solver	28
4.1	Introduction	28
4.2	Algebraic Multigrid	28
4.3	GPU Architecture	29
4.4	CUDA Programming Model	30
4.4.1	Execution Model	30
4.4.2	Memory Hierarchy	31
4.4.3	Performance Optimizations	32
4.5	Multigrid Methods	32
4.5.1	Algebraic Multigrid	33
4.5.2	Multigrid Generation	34
4.5.3	Computing Matrices P and A_c	36
4.6	Improved Coarsening	36
4.7	Modifying A_c for Faster Smoothing in Coarser Grids	37
4.8	Parallelization of Gauss-Seidel Iterative Method	37
4.9	GPU Implementation	37
4.9.1	Vertex Coloring	38
4.9.2	Graph Representation	38
4.9.3	Multigrid Implementation	39
4.10	Classical Algebraic Multigrid	40
4.10.1	Selection of boundary nodes	40
4.10.2	Classical AMG and Non orthogonal unstructured grids	40
4.11	Closure	40
5	Laminar Validation cases	41
5.1	Introduction	41
5.2	Lid Driven Flow	41
5.3	Flow over Square Cylinder	43
5.4	Heat transfer through solid cube	45
5.5	Closure	46
6	Turbulence modeling using Large Eddy Simulation	47
6.1	Introduction	47
6.2	CFD Methods for turbulence	47
6.2.1	Direct Numerical Simulation	47
6.2.2	Large Eddy Simulation	48
6.3	Validation	50
6.3.1	Turbulent Jet	50

7	Numerical Simulation and Analysis of Coaxial Jets	54
7.1	Introduction	54
7.2	Jet emerging into quiescent fluid	54
7.2.1	Jet emerging from inlet at Reynold Number 10	55
7.2.2	Jet emerging from inlet at Reynold Number 100	58
7.3	Laminar Coaxial Jets	61
7.3.1	Effect of Injector inside the domain	65
7.4	Turbulent Coaxial Jets	69
A	The Coefficient Framework for the Unstructured Solver	72
A.1	For Diffusive Flux	72
A.1.1	Basic Expression For Diffusive Flux	72
A.1.2	Evaluating the Diffusion Coefficients	73
A.2	For Convective Flux	77
A.3	Final Form and Implementation	78
A.4	Pressure Equation	78
A.4.1	Modifications in the algorithm	79
	References	80

Chapter 1

Introduction

1.1 Scope of current work

To solve computational flows through complex domains for solving engineering problems, unstructured grids are used for grid generation because of the incentive of lesser computational cost for same, and their ease in adapting the complex domain, when compared to structured grids. Most of the engineering problems involve turbulent flow through complex flow domains. Interestingly, very less work is being done in development of unstructured solvers. There are mainly two type of grids defined by their connectivity:

- *Structured Grid*
- *Unstructured Grid*

Structured meshes generally tend to give better results, but take a lot of time of grid generation, and are difficult to implement over complex geometries. Unstructured grids are used for grid generation because of the incentive of lesser computational cost for same, and their ease in adapting the complex domain, when compared to structured grids.

Finite Volume method has been used for discretization of governing equations. In the Finite Volume method, the solution domain is subdivided into a number of finite volume cells defined by the coordinates of their vertices read from the CGNS grid. Collocated grid arrangement has been implemented where all the dependant variables are defined at the centroid of the individual cells. Primitive variables (like velocity and temperature) are being solved directly.

Most of the time for solving Navier Stokes is consumed in pressure poisson equation. In order to reduce the computational cost,parallization is implemented for the pressure poisson routine. Graphical Processing Unit (GPU) parallelization has been studied on unstructured grids, for which a novel colouring algorithm has been developed using the greedy colouring technique. Classical Algebraic Multigrid (AMG) has been used to further accelarate the solver. It was found that classical AMG was successful for orthogonal meshes, but for non orthogonal meshes, the AMG was not found to be effective as the diagonal dominance was not found in this case due to cross diffuson terms. The solver has been tested for few benchmark laminar cases for which results have been compared with literature and have been found to be in good agreement. Also speedup of these cases against the serial code is presented.

The scope of the solver is limited to GPU parallelized AMG solver for steady and unsteady flows, using semi implicit segregated solver for incompressible flow with species transport equation with turbulence modeling. Turbulence is ubiquitous in nature. Hence to study turbulence on unstructured grids in order to study and solve flow phenomena on complex problems, Large Eddy Simulation (LES) is being implemented in the solver. The LES model used for this purpose is Static Smagorinsky model.

1.2 Literature Survey

Unstructured grids are of great importance to reduce computational time for complex geometries. The governing equations were discretized using a collocated, cell-centered arrangement of velocity and pressure. Date [1] described a procedure for discretization of three-dimensional transport equations on unstructured meshes. The discretization was carried out by constructing a special line structure so that evaluations at the cell-centers can be carried out in a manner close to the structured grid. The present work relies heavily on the unstructured grid handling schemes proposed by Dalal et al [2].

Jets are of great practical relevance to both engineering applications and natural phenomena. Understanding the nature of Jets is crucial in many fields like mixing of fuels in an automobile or a jet engine, dispersion of pollutants in the atmosphere, cooling of gas turbine blades and exhaust gas cooling etc. In fluids, it is not a rare occurrence to see one fluid intruding into quiescent environment. Common examples are discharges from pipes into rivers or lakes and plumes exiting from industrial chimneys and in ventilator ducts. Spray injection in internal combustion engine can also be categorized into one of such applications. In every case, a fluid with some momentum and/or buoyancy exits from a relatively narrow orifice and intrudes into a larger body of fluid with same or different characteristics, such as different speed, temperature. Laminar and turbulent jets have been a subject of interest due to such wide applications. Such study of turbulent round jet of air discharging into quiescent air was studied experimentally was conducted by Panchapakesan and Lumley [3] for a variety of Reynolds number. Large eddy simulation has been a tool of preference for the study of jets. A novel algorithm to solve large eddy simulation on unstructured domains was proposed by Krishnan et al [4]. The algorithm was based on predictor corrector method, but was based on explicit time stepping of second order. Moreover, the algorithm proposed to take pressure at cell faces, to make pressure gradient term from the boundaries conservative in terms of their contribution to kinetic energy. As explicit solutions take a longer time and a smaller time step, the semi implicit algorithm was adapted for the same using second order implicit Crank Nicolson time stepping for our algorithm. The parallelization of Gauss-Seidel on unstructured grids using CUDA programming language for NVidia Graphical Processing Units (GPU) has been done by Jin [5]. Solving Navier-Stokes on GPU has been attempted multiple times. Corrigan et al [6] solved the Euler equation on GPU for compressible flows, and was able to achieve speedup of 9.5X. Incompressible Navier-Stokes was solved using finite element method by Asouti et al. [7] also solved unsteady Navier-Stokes using vertex-centered finite volumes for unstructured grids on GPUs, but the implementation was limited to two dimensions. Goddeke et al [8] has solved Navier-Stokes equation using finite element method in three dimensions. Shin and Vanka [9] [10] [11] have solved LES and DNS on GPU using finite difference method in three dimensions. Geometrical multigrid has been

used to further accelerate the solver. Greedy colouring was used in the paper for unstructured grids, that was used for domain decomposition. Classical Algebraic Multigrid (AMG) was described by Falgout [12], which was modified and used for implementation on GPU. Coaxial inlets have been widely used for Gas combustors. Moin et.al. [13] have carried out large eddy simulation of multiphase combustion for coaxial inlets. Swirl has been extensively studied for coaxial jets and simulations have also been carried out for the same by Pierce et. al [14]. Coaxial jets have been a subject of study for their application to airblast atomizers that has been described by Lefebvre [15]. Mixing regimes for various coaxial jet ratios has been studied by Rehab et. al [16], but the study is mostly focused on high velocity ratios and turbulent structures.

1.3 Thesis Organisation

- *Chapter 2* Deals about CGNS format and mesh handling capability of the solver.
- *Chapter 3* Discusses about discretization of governing equations for unstructured grid.
- *Chapter 4* Details the parallelization and its implementation on unstructured grids. Moreover, it discusses the implementation of Algebraic Multigrid (AMG) on GPU.
- *Chapter 5* Code validations and their results have been presented.
- *Chapter 6* Discusses about Turbulence, Large eddy simulation, its governing equation, implementation and validation.
- *Chapter 7* Simulation and Analysis of Coaxial jets has been presented for various velocity ratios.

Chapter 2

CGNS Format

2.1 Introduction to CGNS format

CGNS(CFD General Notation System) originated in 1994 as a joint effort between Boeing and NASA, and has since grown to include many other contributing organizations worldwide. It is an effort to standardize CFD input and output including grid(both structured and unstructured),flow solution, connectivity, boundary conditions, and auxiliary information within a single format. CGNS is also extensible,and allows for file-stamping and user-inserted-commenting. It employs ADF (Advanced Data Format), a system which reads binary files that are portable across computer platforms. CGNS also includes a second layer of software known as the mid-level library, API (Application Programming Interface), which eases the implementation of CGNS into existing codes.

A CGNS file is an entity that is organized (inside the file itself) into a set of nodes in a tree-like structure, in much the same way as directories are organized in the UNIX environment. The top-most node is referred as the root node. Each node below the root node is defined by both a name and a label, and may or may not contain information data. Each node can also be a parent to one or more child nodes. An example of the CGNS tree-like structure is shown in figure below.

A typical grid for a CFD problem may be divided into several zones (or blocks) each representing

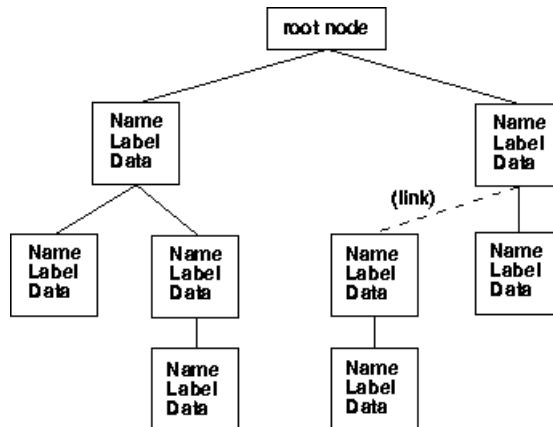


Figure 2.1: CGNS data structure

a section of the flow domain. In CGNS the CFD data is organized according to a topologically based hierarchical tree structure. In this topologically based graph, overall organization is by the zones; information pertaining to a particular zone, including its grid coordinates or flow solution, hangs of that zone. A simplified illustration of the database hierarchy is shown in Fig.2.2 for unstructured and in Fig.2.3 for structured grid.

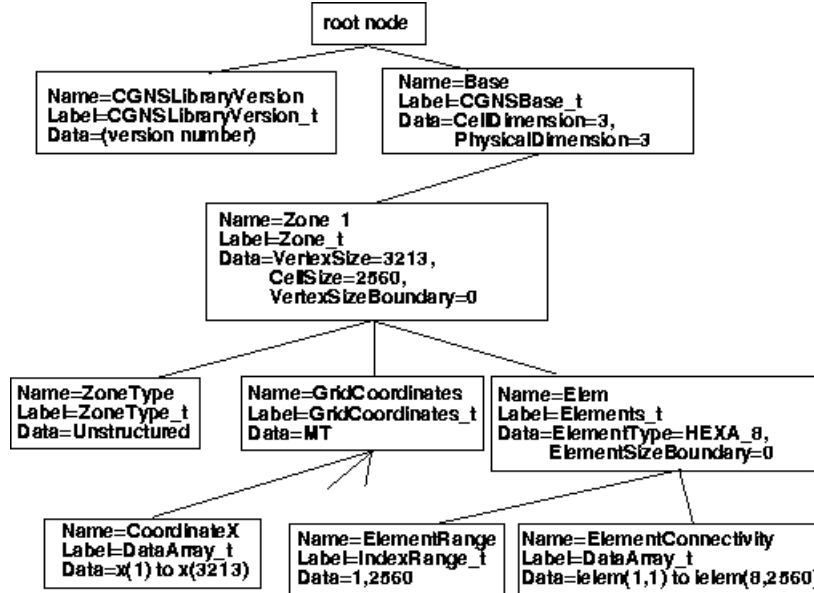


Figure 2.2: Database hierarchy for unstructured grids

Hanging of the root node of the database is a node containing global reference state information, such as free-stream conditions, global convergence criteria etc. Grid Coordinates of the mesh points, flow solution data arrays, boundary conditions and connectivity information etc. all hang from the individual zone nodes. The figure only shows the nodes that hang of the first zone; similar nodes will hang of each zone in the database.

In the present work CGNS version 2.4 file-format is used to store grids and flow solutions. Post processing is done using Tecplot and Paraview.

2.2 Unstructured Grid terminology in CGNS format

The major difference in the way structured and unstructured grids are recorded lies in the element definition. In a structured grid, the indices are recorded in (i,j,k) format (for 3D cases) and elements are indexed by the minimum of the connecting vertices. An example is shown for a 2D structured grid in Fig . Thus, in a structured grid, the elements can be recomputed using the computational coordinates. The element number $[i,j]$ will always be made up of nodes $[i,j]$, $[i+1,j]$, $[i+1,j+1]$ and $[i,j+1]$. Its 4 neighbouring elements will be $[i-1,j]$ to the left, $[i+1,j]$ to the right, $[i,j+1]$ at the top, and $[i,j-1]$ at the bottom. Clearly there is no need to store element connectivity information for structured grids. However, connectivity between zones would have to be stored as the latter do not have the same predictable pattern as elements within zones.

In an unstructured grid, the nodes are numbered using single index convention from 1 to N, where

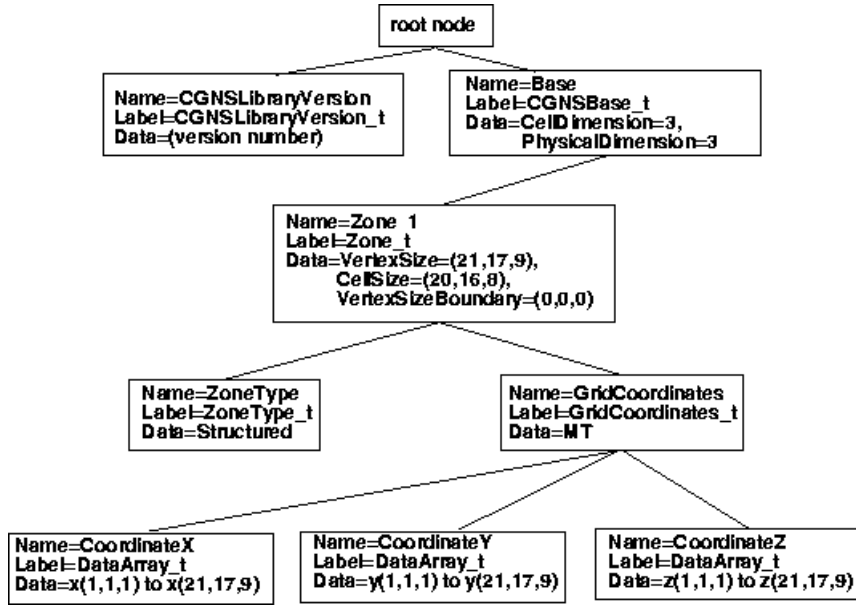


Figure 2.3: Database hierarchy for structured grids

N denotes the number of nodes in that zone. Further the node numbering follows no particular order. So element-connectivity information, i.e. , the information that tells us what are the node numbers that constitute a particular element, cannot be built easily. This additional information is generally added to the data file. The element information typically includes the element type or shape, and the list of nodes for each element numbers.

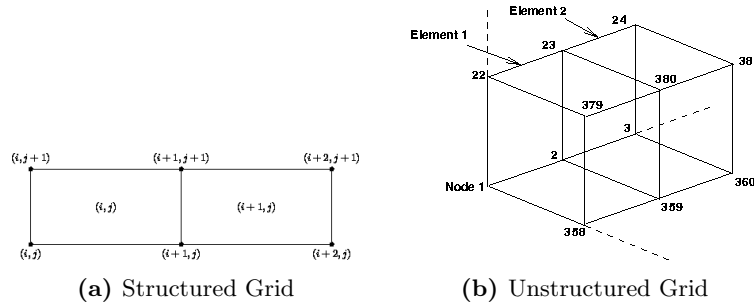


Figure 2.4: Structured and Unstructured Grids

The present solver handles 3-D unstructured meshes only. CGNS supports 4 types of linear element shapes for unstructured grids. These are Tetrahedrals (with 4 triangular faces), Pyramids (with one quadrilateral and 4 triangular faces), Prisms which are also called or Pentahedrals (with 2 triangular faces and 3 quadrilateral faces) and Hexahedrals (with 6 quadrilateral faces). The boundary patches are described using 2D elements which may be Triangles or Quadrilaterals depending on the type of face of the corresponding 3D cell that falls on the boundary patch. The ordering of the nodes within an element is important. Since the nodes in each element type could be ordered in multiple ways, it is necessary to define numbering conventions. The following insets describe the element numbering conventions used in CGNS for the different element types.

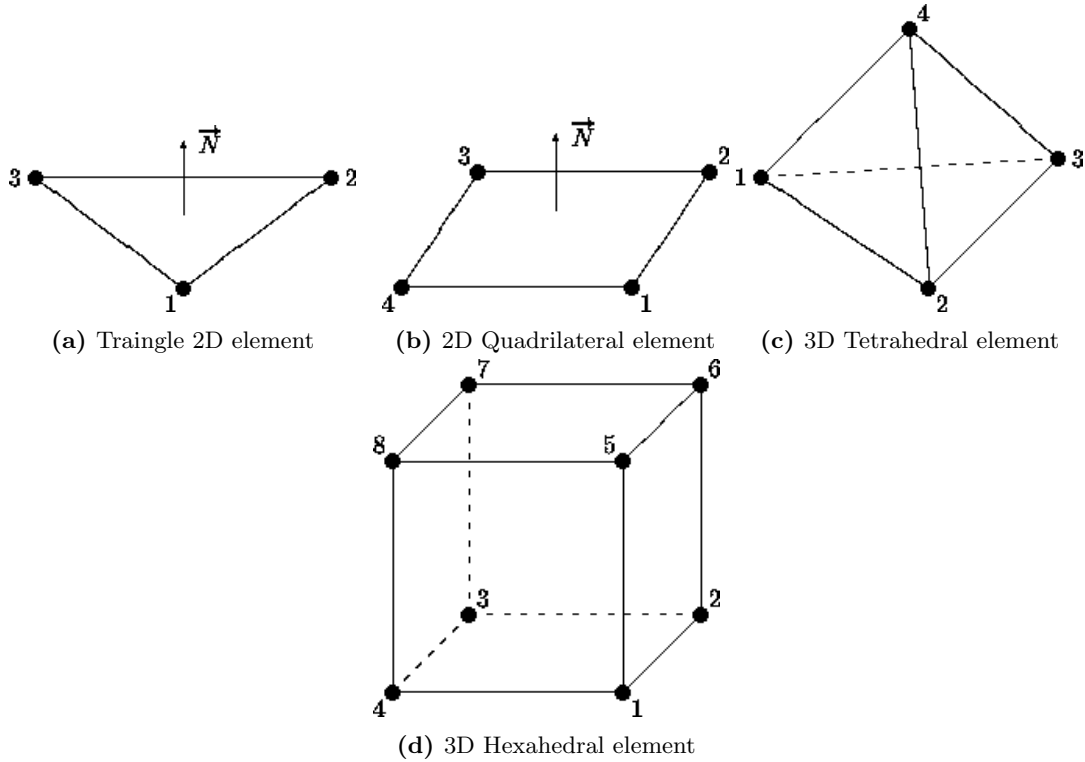


Figure 2.5: Unstructured Grid Elements and Node Numbering

2.3 Processing connectivity information

An unstructured grid may be divided into several zones. Each zone occupies a certain portion of the flow domain. The zone volume may be divided into three parts. The volume itself that is the interior of the zone, the zone faces that about a boundary of the flow domain, and the zone faces that about a neighbouring zone. They are classified as the interior section, boundary sections (which may be more than one if the zone coincides with two or more domain boundaries) and interfaces (which again may be more than one if there are several neighbouring zones) respectively. All local data pertaining to the mesh constructed within this zone are stored below the node in the datastructure corresponding to the zone (either in the zone datastructure itself or in one of its subsequent sub-nodes). A homogeneous grid is defined as one which is meshed by one and only one element type in all its zones. A mesh is called a hybrid grid and treated by the CGNS as such if either of the two situations described below hold true,

1. Each zone is composed of only one element type but element types are different for the different zones.
2. At least one zone exists which is made up of a mixture of two or more elements.

The information we can read directly from a grid converted to CGNS format are:-

- The total number of zones.
- Type of grid, hybrid or homogeneous.
- Number of interior elements in each zone.

- Number of sections in each zone-There will be one interior section and as many additional sections as there are boundary patches. Sections are not defined for interfaces.
- Start and end element numbers in each section of a zone. CGNS numbering system ensures that all elements numbering between the start and end element numbers of a section belong to that section.
- Element Connectivity information for each section of a zone.
- Number of interfaces for each zone
- The name label of the zone abutting a particular interface (this adjoining zone is called the DonorZone) of a zone.
- list of nodes that constitute an interface of the zone. This is called PointList.The zone in question in whose data-node PointList is being stored is called the Parent Zone corresponding list of nodes in the DonorZone that constitute the same interface.This is called PointListDonor.

The Element Connectivity information is arranged in the form of a 1-D array. There is a separate array for each section of each zone. Here ElType is an integer which characterizes the type (Hexa, Tetra etc.) of element. N1, N2 .. are the global node numbers of the nodes that comprise it. For each element, the node numbers in the Connectivity array come in the order defined by the CGNS element numbering conventions discussed in Section 2.2. For a homogeneous grid, element type is fixed for the section and hence ElType is not included in the element-connectivity information.

The following connectivity information are subsequently evaluated in the code :-

- Neighbour Element information that stores the neighbour element number for each face of each element. Default values are stored if the face is found to be at the boundary or interface.
- ParentElement information that has two memory locations for each boundary face element where we store the boundary section number and the number of the 3D parent cell whose face coincides with the boundary element.
- Interface Data which has four memory locations for each interface face-element. The first location stores Parent Element Number which gives the 3D interior element number of the current zone whose face coincides with this interface face. The second location stores Parent Face Number which gives the corresponding face number of this coinciding face. The third location stores Donor Element Number that gives the 3D interior element number of the adjoining donor zone whose face is coinciding with the same interface face. Finally, the fourth memory location stores Donor Face Number that gives the corresponding face number of this coinciding face.

Neighbour Element information is obtained by using the principle that if two elements share a face then they must have 3 (for triangular face) or 4 (for quadrilateral face) nodes in common.This is checked for in all the faces that constitute an element. When we get a matching element, its number is calculated from the position it occupies in the ElementConnectivity array. If no matches are found,then that face will lie on an interface or on a boundary section. If the face lies on a boundary section, its nodes must be present in the ElementConnectivity data of one of the boundary sections

of the zone. Otherwise, if it is at the interface, they must be present in a pointlist. Thus all the relevant information are extracted during the initialization process for a CFD simulation and stored in the DataStructure for future use.

2.4 Geometrical Properties

The following subsections describe how the different geometrical parameters are calculated.

2.4.1 Cell Center

CellCenter coordinates are evaluated from the mean of the coordinate values of that element's constituting nodes. Of course the number of nodes an element has will vary depending on its type. So we have,

$$X_c = \frac{\sum_{i=1}^n X_i}{n} \quad (2.1)$$

where X_c is the coordinate of cell centroid, X_i is the coordinate of nodes making the cell and n is the number of nodes the element has. For 2D boundary elements, the formula is the same as in 2.1; only these will be called FaceCenters.

2.4.2 Surface Area

Surface area vector of a triangular face of a 3D element is evaluated using the expression:

$$\mathbf{A} = 0.5(\mathbf{r}_{31} \times \mathbf{r}_{21}) \quad (2.2)$$

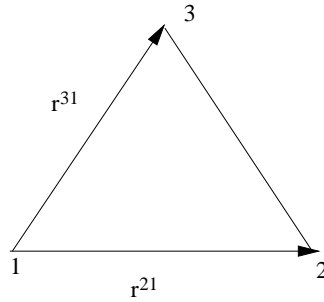


Figure 2.6: Surface Area evaluation for Triangular Face

The Surface Area vector of a Quadrilateral Face is evaluated using the expression, (refer Fig 2.7)

$$\mathbf{A} = 0.5(\mathbf{r}_{24} \times \mathbf{r}_{31}) \quad (2.3)$$

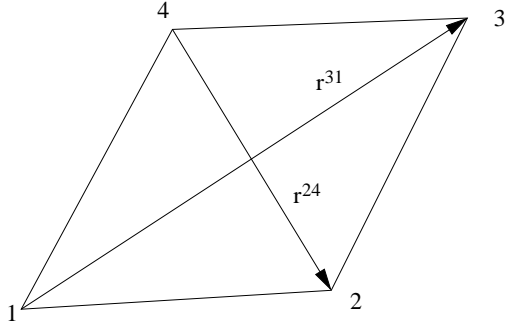


Figure 2.7: Surface Area evaluation for Quadrilateral Face

Care has been taken to ensure that the surface area vector is directed out of the 3D cell at all the faces. Along with surface area vectors, magnitude of the area $|\mathbf{A}|$ and unit surface normal n_f are also stored. Clearly:

$$n_f = \frac{\mathbf{A}}{|\mathbf{A}|} \quad (2.4)$$

2.4.3 Volume

For Hexahedral cells the expression for evaluating volume is given by,

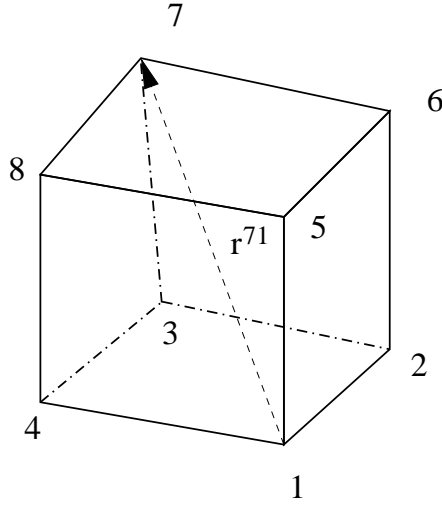


Figure 2.8: Volume evaluation for Hexahedral Cell

$$V = \frac{1}{3} |r_{71} \cdot (A_{1234} + A_{1265} + A_{1584})| \quad (2.5)$$

For a Tetrahedral cell, the expression for evaluating volume is,

$$V = \frac{1}{6} |(r_1 \times r_2) \cdot r_3| \quad (2.6)$$

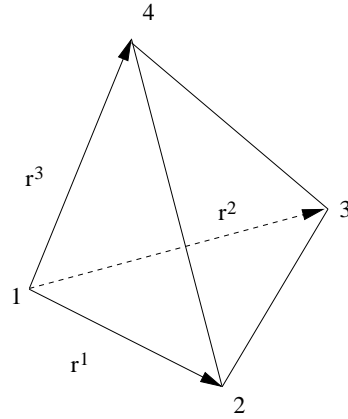


Figure 2.9: Volume evaluation for Tetrahedral Cell

2.5 Example of Unstructured Domain

Consider an unstructured zone in the shape of a cube, with each edge of the zone having three nodes. The resulting unstructured grid has a total of 27 nodes, as illustrated in the exploded figure below.

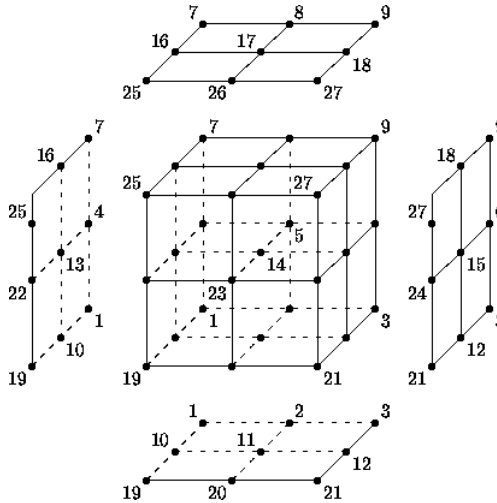


Figure 2.10: Exploded view of a single unstructured zone

This zone contains eight hexahedral cells, numbered 1 to 8, and the cell connectivity is:

Element Number	Element Connectivity
1	1, 2, 5, 4, 10, 11, 14, 13
2	2, 3, 6, 5, 11, 12, 15, 14
3	4, 5, 8, 7, 13, 14, 17, 16
4	5, 6, 9, 8, 14, 15, 18, 17
5	10, 11, 14, 13, 19, 20, 23, 22
6	11, 12, 15, 14, 20, 21, 24, 23
7	13, 14, 17, 16, 22, 23, 26, 25
8	14, 15, 18, 17, 23, 24, 27, 26

Table 2.1: Element connectivity table

In addition to the cells, the boundary faces could also be added to the element definition of this unstructured zone. There are 24 boundary faces in this zone, corresponding to element numbers 9 to 32. Each boundary face is of type QUAD4. The boundary faces will have only one neighbour which will be their parent cell.

2.6 Closure

In this chapter, the methods for handling unstructured data were thoroughly discussed. While the principle applied for finding neighbours, parents and other relevant connectivity information is not complex, it is somewhat time consuming if the number of cells in the domain is large. Hence in the code we have kept the option of writing the relevant connectivity information in files. So if a grid is generated for a particular problem, all the connectivity data are evaluated once only. In all subsequent runs, they are read directly from the file resulting in significant time savings.

Chapter 3

Governing Equations

3.1 Introduction

In this chapter we discuss the governing equations and their differential forms. As the solver is finite volume based, hence the discretization shall be carried out using the finite volume method.

3.1.1 Finite Volume Method

The Finite Volume method for solving the incompressible Navier Stokes equations has become very popular in recent years because of the following advantages:

1. It is easy to implement on non-orthogonal curvilinear grids.
2. The solution can be obtained in the actual physical domain without transforming the governing equations.
3. It is easy to implement the boundary conditions.

In the Finite Volume method, the solution domain is subdivided into a number of finite volume cells defined by the coordinates of their vertices read from the CGNS grid file. **Collocated grid arrangement** has been used where all the dependant variables are defined at the centroid of the individual cells. Primitive variables i.e. velocity and temperature are being solved directly. Thus the governing differential equations may be written for each of these small cells. While discretizing the equations there are two primary assumptions we make that during an integration over the cell volume, the value of the variable is constant and equal to its value at the cell center; while during a surface integral over a cell face, the value of the variable is constant throughout the surface and is equal to the face center value.

3.2 Partial Differential Equations

Navier Stoke equation has been discussed for incompressible, newtonian fluids for which the governing equations can be represented as following.

Continuity Equation

$$\frac{\partial(\rho u_i)}{\partial x_i} = 0 \quad (3.1)$$

Momentum Transport Equation

$$\frac{\partial(\rho u_i)}{\partial x_i} + \frac{\partial(\rho u_i u_j)}{\partial x_j} - \frac{\partial^2(\mu u_i)}{\partial x_j \partial x_j} = -\frac{\partial p}{\partial x_i} + S_{u_i} \quad (3.2)$$

where μ is the coefficient of dynamic viscosity and S_{u_i} is a source term which may or may not be present; p is the pressure and u_i represent the different velocity components.

Species Transport Equation:

$$\frac{\partial(\rho\phi)}{\partial t} + \frac{\partial(\rho\phi u_j)}{\partial x_j} - \frac{\partial}{\partial x_j} \left(\Gamma \frac{\partial\phi}{\partial x_j} \right) = S_\phi \quad (3.3)$$

where ϕ is the species in question, Γ is its diffusion coefficient and S_ϕ is the corresponding source term.

3.3 Governing Equations: Integral Form

The integral form of the governing differential equations is obtained when the differential forms are integrated over the entire flow control volume. Using the Gauss-Divergence theorem, the integral form of the governing equations are,

Continuity Equation

$$\int_S \rho \mathbf{u} \cdot d\mathbf{S} = 0 \quad (3.4)$$

Momentum Transport Equation

$$\frac{\partial}{\partial t} \int_V \rho u_i dV + \int_S (\rho u_j u_i - \mu \frac{\partial u_i}{\partial x_j}) d\mathbf{S}_j = - \int_S p d\mathbf{S}_i + \int_V S_{u_i} dV \quad (3.5)$$

Species Transport Equation

$$\frac{\partial}{\partial t} \int_V \rho \phi dV + \int_S (\rho u_j \phi - \Gamma \frac{\partial \phi}{\partial x_j}) d\mathbf{S}_j = \int_V S_\phi dV \quad (3.6)$$

3.4 Discretization Procedure

The integral form of the equations is then discretized using the following

3.4.1 Discretization of the Continuity Equation

The continuity equation is discretized in the following way:

$$\int_S \rho \mathbf{u} \cdot d\mathbf{S} \approx \sum_j \rho_j (\mathbf{u} \cdot \mathbf{S})_j = \sum_j \rho_j \mathbf{u}_j \cdot \mathbf{S}_j \quad (3.7)$$

where \mathbf{S}_j is the surface vector representing the area of the j^{th} cell face and \mathbf{u}_j is the velocity defined at the face center \mathbf{j} . Here ρ_j is the volume interpolated face centred value of ρ . If density is constant then this term may be taken out of the summation. Thus the discretized form of the continuity equation is:

$$\sum_j F_j = 0 \quad (3.8)$$

where the F_j is the outward mass flux through face j , defined by

$$F_j = \rho_j \mathbf{u}_j \cdot \mathbf{S}_j \quad (3.9)$$

3.4.2 Discretization of the General Convection-Diffusion Equation

Looking at Eq (3.5) and Eq. (3.6), it is clear that the only difference between these two equation is that in Eq (3.5) an additional source term due to pressure is present. So discretization of all other terms will be the same for both. Thus in this section, ϕ will denote both velocity components and species.

Rate of Change For the discretization of the temporal term it has been assumed that the value of the dependent variable at the centroid is the average over the entire control volume.

$$\frac{\partial}{\partial t} \int_V \rho \phi dV \approx \frac{(\rho \phi V)_P^{n+1} - (\rho \phi V)_P^n}{\Delta t} \approx V_P \frac{(\rho \phi)_P^{n+1} - (\rho \phi)_P^n}{\Delta t} \quad (3.10)$$

where V_P is the volume of the cell P and the suffix P denotes the centroidal value.

Convection Fluxes The approximation of the surface integral over convection flux of variable ϕ has been done in the following way:

$$\int_S \rho \mathbf{u} \phi \cdot d\mathbf{S} \approx \sum_j \rho_j \phi_j (\mathbf{u} \cdot \mathbf{S})_j = \sum_j F_j \phi_j \quad (3.11)$$

where ϕ_j is the value of ϕ at the center of the face j . To avoid unphysical oscillations that frequently occur for convection dominated flows, upwinding needs to be used to discretize the convective terms. In unstructured grids, higher order upwinding schemes like QUICK cannot be implemented. However, first order upwinding is not very accurate and introduces excessive numerical diffusion. Hence a linear combination of Central Difference Scheme(CDS) and Upwind Differencing Scheme(UDS) is used to discretize the convective term.

$$F_j \phi_j = (F_j \phi_j)^{UDS} + \gamma[(F_j \phi_j)^{CDS} - (F_j \phi_j)^{UDS}] \quad (3.12)$$

In Central Difference discretization a volume interpolation is used to evaluate the value of the variable ϕ at the face center. In upwind scheme, this value is taken to be equal to that at the upstream cell center. Thus Eq. (3.12) can be expanded as:

$$\begin{aligned} F_f \phi_f &= \phi_P[[F_f, 0]] - \phi_n[[-F_f, 0]] + \gamma\{F_f(\frac{V_n}{V_n + V_P}\phi_P + \frac{V_P}{V_n + V_P}\phi_n) \\ &- \phi_P[[F_f, 0]] + \phi_n[[-F_e, 0]]\} \end{aligned} \quad (3.13)$$

Here $[[p, q]]$ denotes the maximum of p and q , suffix f denotes the particular face of the cell in question and suffix n denotes values for the corresponding neighboring cell.

A fully implicit method for time-stepping has been used, where the upwind part of the above equations are incorporated in the coefficients of the unknown velocity during the pressure velocity iteration. The CDS terms are evaluated using previous iteration values and used as a source term on the right hand side of the same equation. This is called as “**deferred correction**” approach of Khosla and Rubin [?]. Multiplication of the explicit part by a factor γ ($0 \leq \gamma \leq 1$) allows the introduction of the numerical diffusion ($\gamma = 0$ means pure UDS, $\gamma = 1$ means pure CDS). The deferred correction approach enhances the diagonal dominance of the coefficient matrix, which adds to the stability of the solution algorithm.

Diffusion Fluxes The surface integral over diffusion flux of variable ϕ can be approximated as

$$\int_S \Gamma_\phi \nabla \phi \cdot \mathbf{dS} \approx \sum_f (\Gamma_\phi \nabla \phi \cdot \mathbf{S})_f = \sum_j -F_{d\phi f} \quad (3.14)$$

So for any face f , the diffusion flux is given by,

$$F_{d\phi f} = -\Gamma_f (\nabla \phi_f \cdot \mathbf{S}_f) \quad (3.15)$$

The face area vector is represented as \mathbf{S}_f . Its magnitude is given by A_f and its unit vector by $\hat{\mathbf{n}}_f$. Suppose that this face has the cell P_n as its neighbor. Let $\hat{\mathbf{n}}_{1f}$ be the unit vector along the line joining the center of the current cell P and the cell center of this neighbor P_n . Let r_{1f} be the magnitude of this distance. So,

$$\mathbf{r}_{1f} = \mathbf{x}_n - \mathbf{x}_P \quad (3.16)$$

$$r_{1f} = \|\mathbf{x}_n - \mathbf{x}_P\| \quad (3.17)$$

$$\hat{\mathbf{n}}_{1f} = \frac{\mathbf{r}_{1f}}{r_{1f}} \quad (3.18)$$

Next we define a new vector \mathbf{n}'_{2f} as (refer Fig 3.1),

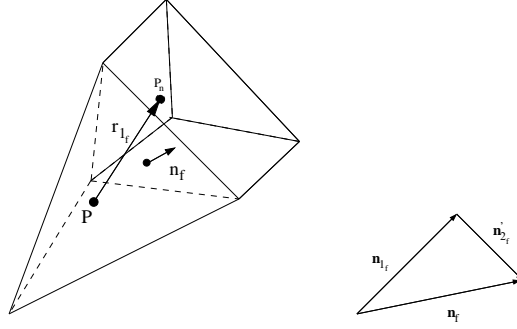


Figure 3.1

$$\hat{\mathbf{n}}_f = \hat{\mathbf{n}}_{1f} + \mathbf{n}'_{2f} \quad (3.19)$$

Clearly, one can calculate the vector \mathbf{n}'_{2f} at all those faces of a cell that do not lie at a boundary. Hence for any face that lies at the interior,

$$\begin{aligned} (\nabla\phi)_f \cdot \mathbf{S}_f &= A_f (\nabla\phi)_f \cdot \hat{\mathbf{n}}_f \\ &= A_f (\nabla\phi)_f \cdot (\hat{\mathbf{n}}_{1f} + \mathbf{n}'_{2f}) \end{aligned} \quad (3.20)$$

Since,

$$(\nabla\phi)_f \cdot (\hat{\mathbf{n}}_{1f}) = \frac{\phi_n - \phi_P}{r_{1f}} \quad (3.21)$$

Using (3.20) and (3.21) we get,

$$(\nabla\phi)_f \cdot \mathbf{S}_f = A_f \left[\frac{\phi_n - \phi_P}{r_{1f}} + (\nabla\phi)_f \cdot \mathbf{n}'_{2f} \right] \quad (3.22)$$

Now, setting

$$\beta_f = \frac{A_f}{r_{1f}} \quad (3.23)$$

$$\mathbf{n}_{2f} = A_f \mathbf{n}'_{2f} \quad (3.24)$$

we get

$$(\nabla\phi)_f \cdot \mathbf{S}_f = \beta_f(\phi_n - \phi_P) + (\nabla\phi)_f \cdot \mathbf{n}_{2f} \quad (3.25)$$

So for all interior cell faces the diffusion flux is discretized as,

$$F_{d\phi_f} = -\Gamma_f [\beta_f(\phi_n - \phi_P) + (\nabla\phi)_f \cdot \mathbf{n}_{2f}] \quad (3.26)$$

Discretizing $(\nabla\phi)_f \cdot \mathbf{n}_{2f}$:-

By the Gauss Divergence Theorem,

$$\int_V \frac{\partial\phi}{\partial x} dV = \int_S \phi \hat{i} \cdot d\mathbf{S} = \sum_f \phi_f S_{f_x} \quad (3.27)$$

$$\int_V \frac{\partial\phi}{\partial y} dV = \int_S \phi \hat{j} \cdot d\mathbf{S} = \sum_f \phi_f S_{f_y} \quad (3.28)$$

$$\int_V \frac{\partial\phi}{\partial z} dV = \int_S \phi \hat{k} \cdot d\mathbf{S} = \sum_f \phi_f S_{f_z} \quad (3.29)$$

Hence for a control volume cell,

$$\left(\frac{\partial\phi}{\partial x}\right)_P = \frac{1}{V_P} \sum_f \phi_f S_{f_x} \quad (3.30)$$

$$\left(\frac{\partial\phi}{\partial y}\right)_P = \frac{1}{V_P} \sum_f \phi_f S_{f_y} \quad (3.31)$$

$$\left(\frac{\partial\phi}{\partial z}\right)_P = \frac{1}{V_P} \sum_f \phi_f S_{f_z} \quad (3.32)$$

where values of ϕ at the faces is obtained using volume interpolation,

$$\phi_f = \frac{V_n \phi_P + V_P \phi_n}{V_n + V_P} \quad (3.33)$$

The Gradient of ϕ at the cell centers $(\nabla\phi)_P$ can be written as,

$$(\nabla\phi)_P = \left(\frac{\partial\phi}{\partial x}\right)_P \hat{i} + \left(\frac{\partial\phi}{\partial y}\right)_P \hat{j} + \left(\frac{\partial\phi}{\partial z}\right)_P \hat{k} \quad (3.34)$$

Using the known values of $(\nabla\phi)_P$, $(\nabla\phi)_f$ can be evaluated using volume interpolation,

$$(\nabla\phi)_f = \frac{V_n(\nabla\phi)_P + V_P(\nabla\phi)_n}{V_n + V_P} \quad (3.35)$$

Thus $(\nabla\phi)_f$ is evaluated at all interior cell faces. Finally,

$$(\nabla\phi)_f \cdot \mathbf{n}_{2f} = (\nabla\phi)_{fx}n_{2x} + (\nabla\phi)_{fy}n_{2y} + (\nabla\phi)_{fz}n_{2z} \quad (3.36)$$

Pressure Term: Pressure terms come as a source term in the momentum equation. The pressure term of Eq. (3.5) can be discretized as,

$$-\int_S p d\mathbf{S} \approx -\sum_f p_f S_{fi} \quad (3.37)$$

p_f is the pressure at the f^{th} face center and S_{fi} is the i^{th} direction component of the surface vector for face f . As usual, volume interpolation will be used to evaluate pressure at the face centers.

Other Source Terms: The source term is integrated over the cell volume as follows:

$$\int_V S_\phi dV \approx (S_\phi)_P V_P \quad (3.38)$$

In the momentum equations, buoyancy source terms may be present for a natural convection driven flow, while in species transport equations chemical reactions may act as a source term.

3.5 Handling of terms at the Boundary

The boundary sections are composed of two dimensional Triangular or Quadrilateral elements. We assume these to be zero volume cells with their centroids coinciding with the face centers. We implement face normal formulation for implementing boundary conditions on the boundary section. However the intuitive approach would be to use Face Centroid formulation, which has been explained in the following subsection.

3.5.1 FaceCentroid Formulation

In this method, the boundary conditions are used to find the values of ϕ at the face centers of the boundary cells. At the boundary face b , the vector \mathbf{r}_{1b} is defined as the vector joining the centroid of the parent cell to the boundary face center (refer Fig. 3.2).

The corresponding unit vector is $\hat{\mathbf{n}}_{1b}$ and magnitude is r_{1b} . So at the boundary face the vector \mathbf{n}'_{2b} is defined as,

$$\hat{\mathbf{n}}_b = \hat{\mathbf{n}}_{1b} + \mathbf{n}'_{2b} \quad (3.39)$$

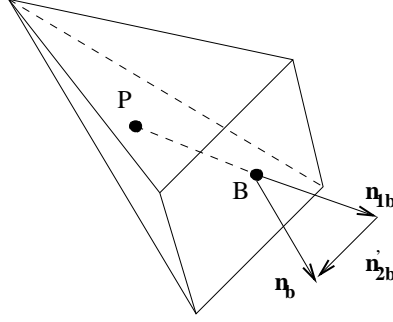


Figure 3.2: The Face Centroid Formulation

$$\beta_b = \frac{A_b}{r_{1_b}} \quad (3.40)$$

$$\mathbf{n}_{2_b} = A_b \mathbf{n}'_{2_b} \quad (3.41)$$

the Eq.(3.25) for diffusion flux is modified at the boundary faces to become

$$(\nabla\phi)_b \cdot \mathbf{S}_b = \beta_b(\phi_b - \phi_P) + (\nabla\phi)_b \cdot \mathbf{n}_{2_b} \quad (3.42)$$

Clearly both ϕ_b and $(\nabla\phi)_b$ needs to be estimated in order to obtain the diffusion fluxes at the boundary. Let us see how this is done for Dirichlet and Neumann boundary conditions.

Dirichlet Condition: In Dirichlet boundary condition ϕ_b is given. So this value is used directly as the value at the boundary face center and no further calculation is necessary. However estimating $(\nabla\phi)_b$ proves to be difficult. The simplest approximation is,

$$(\nabla\phi)_b \approx (\nabla\phi)_P \quad (3.43)$$

which makes

$$(\nabla\phi)_b \cdot \mathbf{S}_b \approx \beta_b(\phi_b - \phi_P) + (\nabla\phi)_P \cdot \mathbf{n}_{2_b} \quad (3.44)$$

Neumann Condition: Here outward flux normal to the boundary cell is specified as $\dot{\phi}_b$. So clearly,

$$(\nabla\phi)_b \cdot \mathbf{S}_b = A_b \dot{\phi}_b \quad (3.45)$$

and nothing needs to be done to evaluate diffusion fluxes. But we still need ϕ_b to evaluate convective fluxes at the boundary face. Using (3.42) we get,

$$A_b \dot{\phi}_b = \beta_b(\phi_b - \phi_P) + (\nabla\phi)_b \cdot \mathbf{n}_{2_b} \quad (3.46)$$

Again $(\nabla\phi)_b \cdot \mathbf{n}_{2_b}$ is difficult to evaluate and we use Eq.(3.43) to rewrite Eq.(3.46) as,

$$A_b \dot{\phi}_b \approx \beta_b(\phi_b - \phi_P) + (\nabla\phi)_P \cdot \mathbf{n}_{2_b} \quad (3.47)$$

Thus,

$$\phi_b = \phi_P + \frac{A_b \dot{\phi}_b}{\beta_b} - \frac{1}{\beta_b} (\nabla\phi)_P \cdot \mathbf{n}_{2_b} \quad (3.48)$$

However from Eqns.(3.30)— Eq.(3.34), it is clear that to evaluate $(\nabla\phi)_P$ we need to know ϕ_b beforehand. So Eq.(3.48) needs to be solved iteratively for ϕ_b , which makes the process of evaluating boundary conditions complex.

A test was done on the accuracy of Face centroid formulation by Dalal [2], which compared analytical results with numerical results and found that this formulation produced errors beyond acceptable levels. Many complex approaches were also tested in the same paper, which were scraped to favor a much simpler formulation as discussed in next subsection.

3.5.2 FaceNormal Formulation

In the face normal formulation, the boundary values ϕ_b are stored for the point where the perpendicular dropped from the centroid of the Parent Cell meets the boundary face. If we denote the centroid point as P, then in Fig 3.3 N is the point on the boundary face where the perpendicular from C meets the boundary face. We shall call this the Face Normal Point. The point B is the face center of the boundary face. Since \vec{PN} is normal to the Surface Area Vector \mathbf{S}_b , $\hat{\mathbf{n}}_b$ will be its unit vector. Also, $\hat{\mathbf{n}}_{1_b}$ will lie along \vec{PB} .

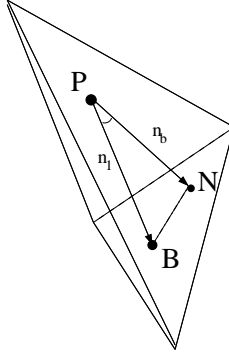


Figure 3.3: The Face Normal Formulation

Defining $\angle(BPN)$ as θ we have,

$$|\vec{PB}| = r_{1_b} \quad (3.49)$$

$$\cos \theta = \hat{\mathbf{n}}_{1_b} \cdot \hat{\mathbf{n}}_b \quad (3.50)$$

So in the right $\triangle PNB$ we have,

$$\begin{aligned} |\vec{PN}| &= |\vec{PB}| \cos \theta \\ &= r_{1_b}(\hat{\mathbf{n}}_{1_b} \cdot \hat{\mathbf{n}}_b) \end{aligned} \quad (3.51)$$

Then the diffusion flux at the boundary face may be written as,

$$\begin{aligned} (\nabla\phi)_b \cdot \mathbf{S}_b &= A_b(\nabla\phi)_b \cdot \hat{\mathbf{n}}_b \\ &= A_b \frac{(\phi_N - \phi_P)}{|\vec{PN}|} \\ &= A_b \frac{(\phi_N - \phi_P)}{r_{1_b}(\hat{\mathbf{n}}_{1_b} \cdot \hat{\mathbf{n}}_b)} \end{aligned} \quad (3.52)$$

Defining the value of β at the boundary faces as,

$$\beta_b = \frac{A_b}{r_{1_b}(\hat{\mathbf{n}}_{1_b} \cdot \hat{\mathbf{n}}_b)} \quad (3.53)$$

the final expression for diffusion flux at the boundaries become,

$$(\nabla\phi)_b \cdot \mathbf{S}_b = \beta_b(\phi_N - \phi_P) \quad (3.54)$$

The coordinate values of these Face Normal Points are also needed for the cases where the boundary condition is a function of space. If \mathbf{r}_P be the position vector of Cell Centroid P and \mathbf{r}_N be the position vector of the face normal point N, then,

$$\begin{aligned} \mathbf{r}_N &= \mathbf{r}_P + \mathbf{PN} \\ &= \mathbf{r}_P + (|PB| \cos \theta) \hat{\mathbf{n}}_b \\ &= \mathbf{r}_P + [r_{1_b}(\hat{\mathbf{n}}_{1_b} \cdot \hat{\mathbf{n}}_b)] \hat{\mathbf{n}}_b \\ &= \mathbf{r}_P + \frac{A_b}{\beta_b} \hat{\mathbf{n}}_b \end{aligned} \quad (3.55)$$

Hence coordinates of the face normal points are given by,

$$x_N = x_P + \frac{A_b}{\beta_b} \hat{\mathbf{n}}_{bx} \quad (3.56)$$

$$y_N = y_P + \frac{A_b}{\beta_b} \hat{\mathbf{n}}_{by} \quad (3.57)$$

$$z_N = z_P + \frac{A_b}{\beta_b} \hat{\mathbf{n}}_{bz} \quad (3.58)$$

Next we look at how the Face Normal formulation handles the boundary conditions.

Dirichlet Boundary Conditions: The boundary values of ϕ are known and are stored directly at the Face Normal Points. Subsequently, Eq.(3.54) is used to calculate the diffusion fluxes at the boundary faces.

Neumann Boundary Conditions: Here the flux at the boundary $\dot{\phi}_b$ is given. Hence using Eq.(3.54),

$$\begin{aligned} A_b \dot{\phi}_b &= (\nabla \phi)_b \cdot \mathbf{S}_b \\ &= \beta_b (\phi_N - \phi_P) \end{aligned} \quad (3.59)$$

So the values of ϕ at the Face Normal Points is given by,

$$\phi_N = \frac{A_b \dot{\phi}_b}{\beta_b} + \phi_P \quad (3.60)$$

Notice that unlike Eq.(3.48) of the FaceCentroid formulation, Eq.(3.60) does not need to be solved iteratively for ϕ at the boundary faces. This helps to decrease code runtime. It was found by Dalal [2] that the current formulation was not only simpler to implement, but also had a better numerical accuracy, as compared to the face centroid formulation.

3.5.3 Summary

The above discusses about handling of boundary condition and Face Normal based discretization .Face Normal Discretization has been adopted as the preferred discretization scheme in the Unstructured Solver due to its ease of implementation and its superior numerical accuracy. The subscript ‘‘b’’ will henceforth be used to denote variable values stored at the Face Normal points of the boundary elements.

3.6 Solution Algorithm

The final discretized form of the governing differential equations are:-

Continuity: The Continuity equation is discretized as follows,

$$\sum_f F_f^{n+1} = 0 \quad (3.61)$$

where

$$F_f = u_f^{n+1} S_{fx} + v_f^{n+1} S_{fy} + w_f^{n+1} S_{fz} \quad (3.62)$$

Momentum: The momentum equation is discretized as follows,

$$V_P \frac{(\rho_P u_P)^{n+1} - (\rho_P u_P)^n}{\Delta t} + \sum_f F_f^{n+1} u_f^{n+1} + \sum_f F_{duf}^{n+1} = - \sum_f p_f^{n+1} S_{fx} + (S_u)_P V_P \quad (3.63)$$

$$V_P \frac{(\rho_P v_P)^{n+1} - (\rho_P v_P)^n}{\Delta t} + \sum_f F_f^{n+1} v_f^{n+1} + \sum_f F_{dvf}^{n+1} = - \sum_f p_f^{n+1} S_{fy} + (S_v)_P V_P \quad (3.64)$$

$$V_P \frac{(\rho_P w_P)^{n+1} - (\rho_P w_P)^n}{\Delta t} + \sum_f F_f^{n+1} w_f^{n+1} + \sum_f F_{dwf}^{n+1} = - \sum_f p_f^{n+1} S_{fz} + (S_w)_P V_P \quad (3.65)$$

Scalar Equation: The scalar equations have the generic form,

$$\Delta V_P \frac{(\rho_P \phi_P)^{n+1} - (\rho_P \phi_P)^n}{\Delta t} + \sum_f F_f^{n+1} \phi_f^{n+1} + \sum F_{d\phi f}^{n+1} = S_\phi \quad (3.66)$$

where $(n + 1)$ denotes the unknown values of the current time step.

In the present study, the Navier Stokes and Energy equations have been solved using the finite volume method. We have used non-staggered (collocated) grid arrangement, where the dependent variables are calculated at the centroid of the finite volume. But this arrangement can produce non-physical oscillations in the pressure field, the so-called checker-board pressure distribution. When central differencing is used to represent both the pressure gradient term in the momentum equations and the cell-face velocity in the continuity equation, it then happens that the velocities depend on pressure at alternate nodes and not on adjacent ones and the pressure too depends on velocities at alternate nodes. This behaviour is called velocity-pressure decoupling (Patankar [17]).

To avoid this decoupling, the momentum interpolation method, first proposed by Rhie and Chow [18] has been used. In this approach, the cell-face velocity in the continuity equations are evaluated by linearly interpolating the so-called “mass” velocities computed without the pressure terms in the discretized equations while directly evaluating the pressure gradient using values at the adjacent cell centers. This results in a strong velocity-pressure coupling. The pressure gradient terms, appearing in the momentum equations, are still represented by the central difference approximation.

The segregated solvers are used only if the scalars are not coupled with momentum transport equation and a steady state solution is needed. For all other cases, coupled solvers are used. In coupled solvers, the velocity values at each time step are used to find the values of the scalar at that time step. In segregated solver, first the velocity values are completely converged to steady state using the false transient method; and only then these steady state velocity values are used to separately converge the scalars to steady state. This results in significant time savings in the case when the problem is steady and the scalars and velocity equations are not coupled.

3.6.1 SemiImplicit Algorithm

In the Semi-Implicit algorithm the Navier Stokes and Scalar equations are solved using flux value of the previous time step. This eliminates the need for flux convergence loop, decreasing the amount of time taken to acheive convergence. One expects Fully Implicit scheme to be more accurate, but both are 1st order accurate in time. Hence Semi-Implicit algorithm gives acceptable results even for unsteady problems. Initial Conditions for velocity and pressure are prescribed at all points in the domain and boundary conditions are defined at the start of the problem,

Step 0: Fix all parameters and initialize all variables to their respective initial conditions.

Step 1: Evaluate the so called “mass velocities”, \mathbf{u}^* , by solving the “mass velocity equation” which is basically the Navier Stokes equation but without the pressure term. These are given by,

$$V_P \frac{(\rho_P u_P)^* - (\rho_P u_P)^n}{\Delta t} + \sum_f F_f^n u_f^* + \sum_f F_{duf}^* = (S_u)_P^n V_P \quad (3.67)$$

$$V_P \frac{(\rho_P v_P)^* - (\rho_P v_P)^n}{\Delta t} + \sum_f F_f^n v_f^* + \sum_f F_{dvf}^* = (S_v)_P^n V_P \quad (3.68)$$

$$V_P \frac{(\rho_P w_P)^* - (\rho_P w_P)^n}{\Delta t} + \sum_f F_f^n w_f^* + \sum_f F_{dwf}^* = (S_w)_P^n V_P \quad (3.69)$$

Usually the source terms are lagged to the values of the previous time step. Note the flux values of the previous time step are being used here.

Step 2: Calculate the “mass” flux at each face of the control volume using the newly evaluated mass velocity values.

$$F_f^* = \mathbf{u}_f^* \cdot \mathbf{S}_f \quad (3.70)$$

$$\mathbf{u}_f^* = \frac{V_P \mathbf{u}_n^* + V_n \mathbf{u}_P^*}{V_P + V_n} \quad (3.71)$$

Step 3: Evaluate the value of pressure at $(n+1)^{th}$ time step using the pressure Poisson equation,

$$\sum_f (\nabla p_f^{n+1}) \cdot \mathbf{S}_f = \frac{\rho}{\Delta t} \sum_f F_f^* \quad (3.72)$$

Step 4: Calculate the volume flux of $(n+1)^{th}$ time step using the expression,

$$F_f^{n+1} = F_f^* - \frac{\Delta t}{\rho} (\nabla p_f^{n+1}) \cdot \mathbf{S}_f \quad (3.73)$$

Note that the volume flux will satisfy continuity.

Step 5: Solve the complete Navier Stokes equation given by ,

$$V_P \frac{(\rho_P u_P)^{n+1} - (\rho_P u_P)^n}{\Delta t} + \sum_f F_f^n u_f^{n+1} + \sum_f F_{duf}^{n+1} = - \sum_f p_f^{n+1} S_{fx} + (S_u)_P V_P \quad (3.74)$$

$$V_P \frac{(\rho_P v_P)^{n+1} - (\rho_P v_P)^n}{\Delta t} + \sum_f F_f^n v_f^{n+1} + \sum_f F_{dvf}^{n+1} = - \sum_f p_f^{n+1} S_{fy} + (S_v)_P V_P \quad (3.75)$$

$$V_P \frac{(\rho_P w_P)^{n+1} - (\rho_P w_P)^n}{\Delta t} + \sum_f F_f^n w_f^{n+1} + \sum_f F_{dwf}^{n+1} = - \sum_f p_f^{n+1} S_{fz} + (S_w)_P V_P \quad (3.76)$$

to get converged values of u^{n+1}, v^{n+1} and w^{n+1} . Here the fluxes are taken from previous time step values. The source terms are usually lagged to the previous time step values.

Step 6: Solve for the scalars using the expression,

$$\Delta V_P \frac{(\rho_P \phi_P)^{n+1} - (\rho_P \phi_P)^n}{\Delta t} + \sum_f F_f^n \phi_f^{n+1} + \sum F_{d\phi f}^{n+1} = S_\phi \quad (3.77)$$

to get ϕ^{n+1} . This step is omitted for the Segregated scheme.

Step 7: If this is a steady state problem, check if the velocities and scalars have converged to the required level of accuracy. If not converged then set $\mathbf{u}^{n+1} \rightarrow \mathbf{u}^n$, $F_f^{n+1} \rightarrow F_f^n$, $\phi_{n+1} \rightarrow \phi_n$, $t \rightarrow t + \Delta t$ and go to **Step 1**. If it is an unsteady problem then continue for as many time steps as needed.

3.7 Discretizing the Pressure Poisson Equation

The pressure Poisson equation is given by,

$$\sum_f (\nabla p_f) \cdot \mathbf{S}_f = \frac{\rho}{\Delta t} \sum_f F_f^* \quad (3.78)$$

Its discretization will be identical to that of $\sum_f (\nabla \phi)_f \cdot \mathbf{S}_f$ when we replace p with ϕ . So following Eq.(3.25), at the interior faces we have,

$$(\nabla p)_f \cdot \mathbf{S}_f = \beta_f (p_n - p_P) + (\nabla p)_f \cdot \mathbf{n}_{2f} \quad (3.79)$$

While Eq. (3.54) gives the discretization at the boundary faces,

$$(\nabla p)_b \cdot \mathbf{S}_b = \beta_b(p_b - p_P) \quad (3.80)$$

where p_b is the value of pressure at the Face Normal Point.

3.8 Closure

The present chapter has presented the discretization procedures and solution algorithms implemented in the unstructured code in some detail. Method for handling boundary values was discussed. Next the code is validated using benchmark problems. This constitutes the subject matter of the next chapter.

Chapter 4

Parallelization of Solver

4.1 Introduction

Complex engineering problems are computationally expensive. The expense comes in form of solving computational domain and mesh generation. In order to be successfully able to solve such problems we need to reduce the cost. The use of unstructured grids has already reduced the mesh generation cost, but this leads to additional increase in solution time. The pressure poisson equation takes about 60% of the total solution time. Hence in order to reduce the overall computational time, this equation needs to be parallelized. The parallelization done here is the Graphical Processing Unit (GPU) parallelization using Compute Unified Device Architecture (CUDA) programming language on NVidia GPUs. Moreover classical Algebraic Multigrid (AMG) has also been explored and an algorithm with AMG+GPU has been implemented, for overall acceleration of the solver. However, the limits of AMG have been encountered while testing for non-orthogonal grids. In this chapter, we discuss about the concept, implementation of parallelization implemented in the solver. This work was done in collaboration with Ravitej K. and Dr. Naveen Sivadasan of Department of Computer Science and Engineering, Indian Institute of Technology, IIT-Hyderabad. In the thesis by Ravitej [19], one can find in depth details of the work.

4.2 Algebraic Multigrid

As mentioned earlier, Pressure Poisson equation consumes most of the time during solving for flow field. The equation can be represented as:

$$\sum_f (\nabla p_f^*) \cdot \mathbf{S}_f = \frac{\rho}{\Delta t} \sum_f F_f^* \quad (4.1)$$

whose discretization has been described in 3.7. But to apply AMG, the equation must be expressed in the form :

$$Au = f \quad (4.2)$$

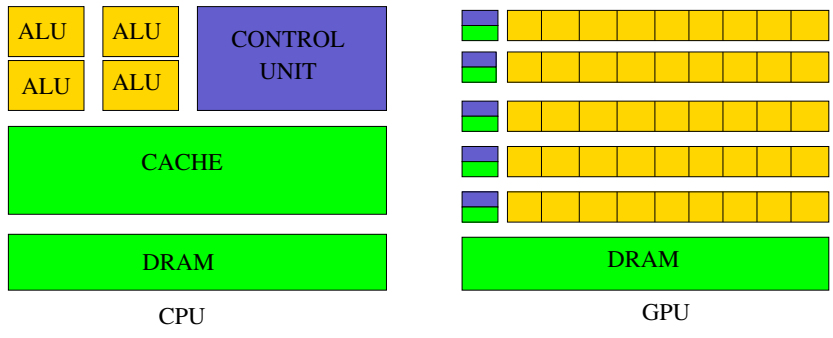


Figure 4.1: Comparison between CPU and GPU architectures [20]

Please refer Appendix A for details of discretization. This represents a system of sparse matrices, where A is $n \times n$ matrix with real entries a_{ij} , u and f are vectors. The numerical methods that exist to solve set of linear equations can be categorized into direct solvers like Gauss Elimination, LU factorization etc. and iterative solvers like Gauss Seidel, Jacobi, Conjugate Gradient etc. Though direct solvers give exact solution, they are inefficient for solving large set of sparse linear equations. Hence, iterative methods which give reasonably accurate approximate solution are preferred over direct solvers. Gauss Seidel is an efficient and most commonly used iterative solver. It starts with an initial guess and produces series of improving results till convergence. Multigrid (MG) method is an efficient and scalable approach that accelerates the convergence of the iterative solvers. In multigrid method, the problem is solved on coarser representation and the solution is interpolated back to the finer representation to get a better approximation faster. This is recursively applied which creates an hierarchy of coarser grids. Algebraic multigrid is a multigrid technique which derives the hierarchy of grids from the information available in the set of linear equations.

4.3 GPU Architecture

Graphics Processing Units (GPUs) which were primarily designed for accelerating video or graphics rendering, had all its functionalities hardwired. Over the last few years, GPUs became programmable and are being used to solve general purpose programs, also called as General Purpose Computation on GPUs (GPGPU).

GPUs are specially designed hardware devices to cater the needs of highly parallel and compute intensive applications. CPU and GPU are designed using two completely different philosophies. Figure 4.1 compares and contrasts the CPU, GPU architectures.

CPU aims at minimizing the latency where as GPU tries to hide the latency. CPU has large cache memory and does sophisticated things like out of order instruction execution, branch prediction etc. It is for this reason, more transistors are dedicated to control unit than arithmetic logic units (ALUs). CPU is well suited for sequential/serial code execution. On the flip side, GPU has relatively smaller cache and more transistors are dedicated to ALUs than the control unit. GPU can execute large number of threads in parallel and is well suited for compute intensive tasks. To execute large number of threads in parallel, GPU uses an architecture called SIMT (Single Instruction Multiple Threads). It is closely related to SIMD (Single Instruction Multiple data) where different processing elements execute same instruction but on different data items. In SIMD all the threads follow same execution

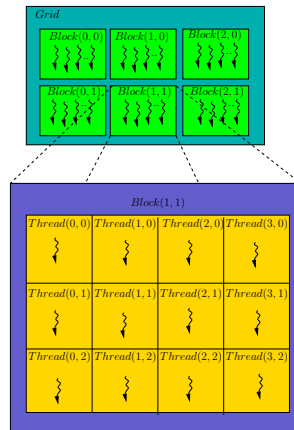


Figure 4.2: Thread Organization in CUDA

path where as SIMT facilitates threads to take different execution paths. A typical GPU contains ALU, Control Unit, cache memory and DRAM. GPU cores are organized as an array of Streaming Multiprocessors (SMs). Each SM contains number of Streaming Processors (SPs or simply GPU cores), instruction cache and control unit. In GPU computing model, the terms *host*, *device* are used to refer to CPU and GPU respectively. Each SM creates, manages and executes threads in group (typically of size 32) called warps. Warp Scheduler, which is also a part of SM schedules these warps for execution.

4.4 CUDA Programming Model

Compute Unified Device Architecture (CUDA) is an interface that enables programmer to utilize the massive parallel computing capability provided by the GPU for general purpose computing. CUDA also provides developers a set of libraries and extensions to standard programming languages like C, C++ etc. A CUDA source file will be a mixture of host code - which runs on the CPU and device code - which runs on the GPU. The CUDA compiler segregates the code into host and device code.

4.4.1 Execution Model

Using CUDA, the compute intensive and data parallel parts of an application are parallelized by launching large number of concurrent threads on GPU. Each thread executes same instruction but on different data. For this purpose, users define kernel which contains the code to be executed by each thread. Kernel configuration specifies the number, organization of the threads and can be determined either at compile time or run time. In CUDA, threads are organized in two level hierarchy namely blocks and grids as shown in Fig 4.2.

At the first level, the threads are grouped into thread blocks. The block size is a multiple of warp size and is decided by the programmer. Block size of 128 or 256 is most frequently used and often provide optimal performance. Each thread block can run independent of other and hence can be scheduled across any SM. The thread blocks are further grouped into grids. The size of grid is determined by the size of data that the application is dealing with. CUDA allows user to

organize blocks and grids in one, two or three dimensions thereby allowing easy mapping of threads to multi-dimensional data structures. Each block within the grid is uniquely identified using the built-in *blockIdx* variable, from the kernel. Similarly, each thread in a thread block is uniquely identified using the built-in *threadIdx* variable. Both blockID and threadID are built in structures that contains three components to store index in each dimension. The thread and block size are stored in built-in variables *blockDim* & *gridDim*. CUDA maps each software thread block to a hardware SM. Multiple blocks can be mapped to same SM and are executed in time sharing fashion.

Threads within a block can communicate with each other using shared memory and can synchronize using *_syncthread()* method. However, threads across the blocks can't synchronize with each other and can communicate only using global memory. Any data that kernel operates on should reside in device global memory. CUDA API provides three functions for this purpose: (a) *cudaMalloc* allocates memory on the device (b) *cudaMemcpy* transfers the data from host to device and vice-versa and (c) *cudaFree* is used to free the memory on the device.

4.4.2 Memory Hierarchy

The GPU memory is organized as three level hierarchy as shown in Fig 4.3

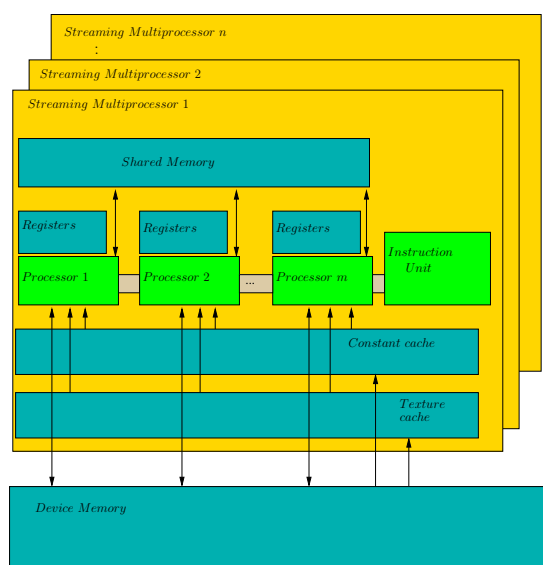


Figure 4.3: CUDA Memory Hierarchy

- **Device Memory** : It is the the largest memory in the hierarchy and also the one with highest latency. Device Memory is to GPU what DRAM is to a CPU. Device memory is logically further divided into global memory, local memory and constant, texture memory. All threads can access global memory and is the only part of device memory which CPU can read as well as write. The local memory which is private to each thread also resides on device memory. Constant memory is used to store read only data such as constant tables etc. and is cached into constant cache. Texture memory which is cached into texture cache is optimized for 2D spatial locality and is preferred over global memory when there is no access pattern to do memory coalescing. GPU can only read from constant and texture memory where as CPU can only write to them.

- **Shared Memory** : It is per SM memory that resides on-chip and is shared by all the threads within a thread block. Access to shared memory is very fast when compared to that of global memory. Any data that is shared or reused by the threads within a block can be transferred to shared memory for improved performance. Shared memory can also be used to share data among threads of same thread block.
- **Registers** : Each thread has its own set of registers. Accessing data in the registers is extremely fast and the CUDA compiler automatically tries to place the frequently accessed variables by the thread into registers.

4.4.3 Performance Optimizations

In addition to effective parallelization of the code, it is crucial to optimize the implementation with respect to the underlying GPU architecture to extract maximum performance [21]. The optimizations include maximizing SM utilization, memory and instruction throughput [22]. Increasing occupancy, coalesced memory access and avoiding warp divergence greatly increase the performance of the applications. Occupancy is defined as the ratio of number of active warps to maximum number of warps supported by SM. Access to global memory data requires hundreds of clock cycles and the warp scheduler switches between warps to hide this latency. Increasing thread pool size i.e., occupancy of SM helps in hiding the latency and also maximizes SM utilization. The hardware also checks if all the threads in a warp are accessing collocated global memory locations. In such scenario, all the accesses can be consolidated and is known as *Coalesced memory access*. Scattered memory access by threads in a warp will result in unnecessary data transfer from global memory to cache. Hence, storing the data accessed by thread warp in collocated global memory locations results in increased memory throughput. Conditional statements in the kernel may cause threads of same warp to follow different execution paths, called as Warp Divergence. Warp divergence causes delay in execution of entire warp and can be avoided by re-ordering the data so that all the threads in warp take same branch.

4.5 Multigrid Methods

Multigrid (MG) method offers an efficient way of solving large system of linear equations especially those from finite volume, finite difference and finite element discretizations of governing PDEs. Multigrid methods are known to scale linearly with respect to number of unknowns i.e., for a given level of convergence multigrid methods provide a solution in $O(n)$ time where n is the number of unknowns [23]. Instead of working on a single mesh, multigrid method works on hierarchy of meshes, which are carefully constructed in such a way that the low frequency error in finer mesh turns out to be high frequency level in the coarse mesh, which can again be effectively smoothed using an iterative method. Multigrid method is a recursive error correcting method and has following steps:

Smoothing: Reduce high frequency error component using iterative methods like Jacobi or Gauss Seidel

Restriction: Transfer the residual from finer mesh to coarser mesh

Prolongation: Transfer the error correction calculated on coarser mesh to finer mesh

Defining MG components include constructing hierarchy of grids, defining inter-grid transfer operations i.e., restriction and prolongation. Two different multigrid approaches exist namely, Geometric multigrid (GMG) and Algebraic multigrid (AMG) [24]. Geometric multigrid, uses the geometry of the problem (grid) to define various multigrid components. On the other hand, algebraic multigrid uses only the information present in the set of linear equations obtained by discretizing the governing PDEs to define various multigrid components.

Though GMG is more natural or intuitive, its applicability is restricted due to requirement of explicit knowledge about problem geometry. Also, the coarsening becomes very complicated/impossible for complex and concave grids. AMG is preferred over GMG due to following advantages:

- It is purely a matrix based approach and doesn't use any geometric information
- No special handling is required for concave grids during coarsening
- AMG can be used as a black-box to solve problems without any geometric background provided the underlying matrices has certain properties [?].

4.5.1 Algebraic Multigrid

In AMG, it is often very helpful to visualize the $n \times n$ matrix A as a graph G on the vertex set $\{1, \dots, n\}$. Each variable corresponds to a vertex in G and each non-zero matrix entry a_{ij} in the matrix A (which is assumed to be symmetric positive definite) corresponds to a directed edge between vertices i and j . In the rest of the paper, the terms grid, mesh, graph and mesh graph are used interchangeably. So, are the terms nodes, points and vertices. If there is a directed edge from vertex u to vertex v then we say that u depends on v and that v influences u . AMG works on the heuristic that the smooth error varies slowly in the direction of relatively large negative coefficients of the matrix A [12].

Definition 1 (Strength of Connection, [12]) *Given a threshold $0 < \theta \leq 1$, the variable i strongly depends on variable j if*

$$-a_{ij} \geq \theta \max_{k \neq i} \{-a_{ik}\}$$

Strength of connection is always measured relative to the largest off diagonal entry. Off diagonal entries which do not satisfy above condition are considered *weak* connections. The matrix obtained by deleting *weak* connections in A is called *Strength Matrix* A_s . We note that strength of connection need not be symmetric i.e., a variable i can strongly depend on j but not vice-versa.

Each level of AMG uses a prolongation matrix P , and the corresponding restriction matrix P^T which is the transpose of P . These matrices are defined based on corresponding strength matrix and is discussed in Section 4.5.3. The coarser system will have lesser number of variables, say $n_c < n$ where n is number of variable in the finer system. Hence P is an $n \times n_c$ matrix. Let $Au = f$ be the equations governing the finer system. Main steps in a two level AMG (which can be extended to multi-level) can be summarized as:

Compute estimate u^* for u in $Au = f$;

Compute the residual $r = f - Au^* = Ae$;

Solve for e_c in the coarser system $A_c \cdot e_c = P^T \cdot r$, where $A_c = P^T A P$;

Correct $u^* \leftarrow u^* + P \cdot e_c$.

Couple of smoothing steps are executed while computing the initial estimate for u^* and after obtaining the correction from the coarser system.

4.5.2 Multigrid Generation

In classical AMG, hierarchy of grids are created from the initial grid by applying a coarsening algorithm recursively. Coarsening algorithm partitions the points into two disjoint sets. One is set of C -points i.e., points that are part of coarse grid as well and the other is F -points i.e., points that are not part of the coarse grid. To compute C , the coarsening algorithm [24] considers the strength matrix A_s and the corresponding mesh graph G_s . Each vertex u is assigned a weight which is the total number of vertices that depend on u . The algorithm proceeds iteratively and at each step, a vertex u with highest weight is chosen as a C point and all vertices depending on u are marked as F points. The weights are updated for vertices that are connected by outgoing edges from the new set of C and F vertices. Weights of all points that influence the new C point is decremented by one. For each new F point u , weights of all points that influence u is incremented by one. Figure 5.4 illustrates the coarsening process.

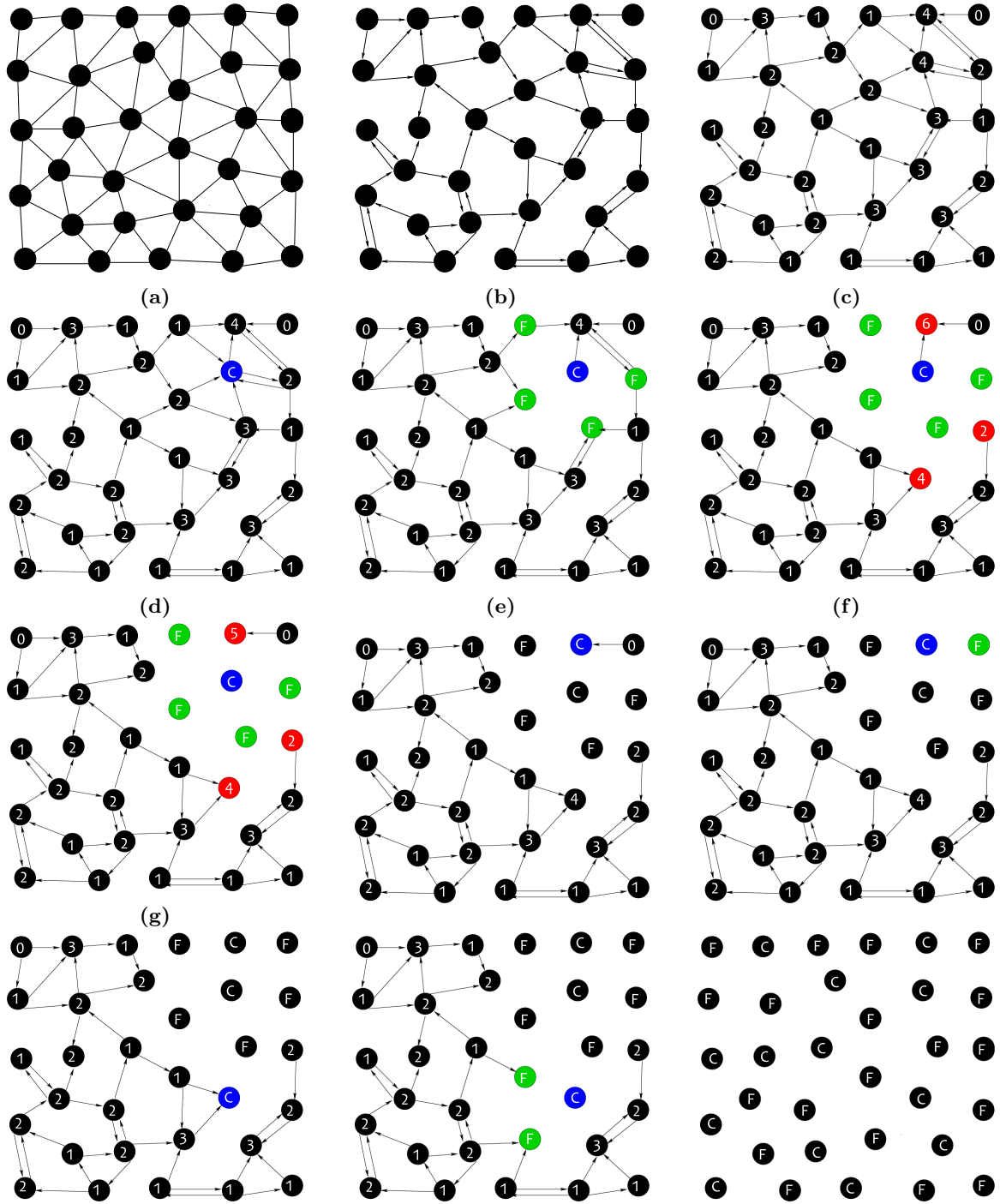


Figure 4.4: Illustration of the coarsening algorithm. (a) The graph corresponding to the A matrix. (b) The graph after deleting weak connections. (c) Nodes of the graph are assigned a weight equal to the number of nodes that depend on it. (d) A point with maximal weight is chosen as a C -point. (e) The neighbors of the new C -point are marked as F -points. (f) For each new F -point, increment the weights of nodes that influence it to make them more likely to be C -points. (g) For new C -point, decrement the weights of nodes that influence it. The algorithm continues in this way until all points are either C or F points.

4.5.3 Computing Matrices P and A_c

Given the C/F splitting of points, the goal is to define P and thereby compute A_c . Let n_c denote the size of C and let n denote the size of $C \cup F$. We follow the approach in [24] to define the $n \times n_c$ matrix P . Let $u_1, u_2, \dots, u_{|C|}$ be an ordering of the vertex set C . Let C_i denote subset of C that strongly influence vertex i . For each $i \in C \cup F$ and each $j \in \{1, \dots, |C|\}$, the entry w_{ij} of P is define as :

$$w_{ij} = \begin{cases} 1 & \text{if } i \in C \text{ and } i = u_j; \\ a_{ij} / \sum_{k \in C_i} a_{ik} & \text{if } i \notin C \text{ and } u_j \in C_i; \\ 0 & \text{otherwise.} \end{cases}$$

The coarser system A_c is obtained using the Galerkin operator

$$A_c = P^T A P.$$

In the following, we discuss the specific algorithmic improvements that we incorporate for faster GPU implementation.

4.6 Improved Coarsening

The accuracy of the solver also depends on the quality of the coarsening. Each level in the multigrid should retain adequate number of boundary nodes and the coarsening algorithm as such will not ensure this. To overcome this, we modify the coarsening (Algorithm 1) and at each stage of coarsening, the boundary nodes are assigned a weight which is α times the number of points that depend on it, for a predefined $\alpha > 1$. As shown in the experiments, by this coarsening, more number of boundary nodes become part of highly coarser grids and thereby improving the coarsening quality.

Algorithm 1 Improved Coarsening

Require: Graph representation of matrix A

- 1: Delete weak connections in the graph
 - 2: For each non-boundary point u , assign a weight equal to the number of points that depend on u .
 - 3: For each boundary point v , assign a weight equal to α times the number of points that depend on v .
 - 4: Choose a point p with maximum weight as C point.
 - 5: Mark points depending on p as F points.
 - 6: For each new F point u , increment weights of all points that influence u by one.
 - 7: Decrement the weights of all points that influence p by one.
 - 8: Repeat steps (4) to (7) till all the points are marked as C or F .
-

4.7 Modifying A_c for Faster Smoothing in Coarser Grids

We incorporate the following transformation to matrix A_c in our coarsening procedure for improving the performance of GPU implementation. As we go down the AMG hierarchy, the number of neighbors for each node in the coarser graphs increases rapidly making the coarse systems more denser. Large degrees result in fetching more data from global memory during smoothing operation in GPU and thereby degrading the GPU performance on coarser systems. To overcome this, our coarsening procedure modifies matrix A_c in such a way that the neighbors with insignificant influence in the corresponding graph is ignored. Entries a_{ij} in the i th row of A_c are modified as follows. Let δ denote the average value of off-diagonal entries in row i (they are negative valued in A_c and positive valued in the graph). Let J' denote the subset of columns such that for each $j \in J'$, $a_{ij} \leq \beta \cdot \delta$, where β is a user defined constant. Let $|J'| = n'$ and let $\epsilon = \sum_{j \notin J'} a_{ij} / n'$. Modified a_{ij} is given by

$$a_{ij} \leftarrow a_{ij} + \epsilon \text{ if } j \in J', \text{ and } a_{ij} = 0 \text{ otherwise.}$$

By the above modification, we ignore all the neighbors whose influence is less than β times the average influence, and their total influence is distributed among the remaining neighbors of i . Though this might slightly slow down the convergence, it is compensated by the reduced smoothing time in coarser grids.

4.8 Parallelization of Gauss-Seidel Iterative Method

To smooth high frequency error component at each level in the multigrid, iterative solvers like Jacobi or Gauss-Seidel can be employed. Both these methods assume an initial guess and visit nodes in an arbitrary order to update the value at the node. However, they differ in the values of neighboring nodes that are used during updating. Jacobi method is preferred if vector or parallel processor is available at disposal due to its ease of parallelization. However, Gauss-Seidel method has faster convergence than Jacobi methods and hence is used in this work. Gauss-Seidel is inherently sequential as we can't update all the inter-dependent nodes simultaneously. Graph vertex coloring in the corresponding mesh graph is used to obtain independent sets corresponding to the color classes (Fig 4.5). All points in one color class can be updated in parallel [5, 25]. We discuss the details in the next chapter.

4.9 GPU Implementation

We use CUDA programming model for our implementation. Implementing graphs algorithms on GPU is challenging due to irregular data access pattern associated with graphs. Using appropriate data structures and data organization/arrangement that maximizes coalesced memory access is the key for effective GPU implementation. A total of seven GPU kernels are used in our implementation: One kernel to perform smoothing, two kernels each for restriction and prolongation operations. A kernel for array reduction is used to get root mean square error for convergence testing. The different algorithmic techniques and data structures used for GPU implementation of the solver are discussed in the following.

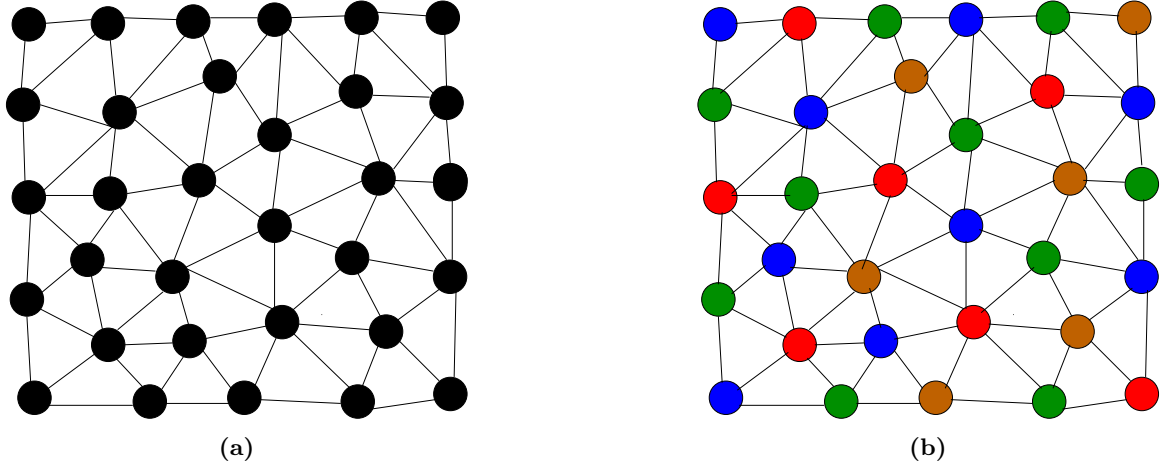


Figure 4.5: Multi Colored Gauss Seidel Smoother (a) Graph corresponding to the matrix A (b)The graph is colored to get independent sets of nodes

4.9.1 Vertex Coloring

To parallelize Gauss-Seidel iterative smoothing, we use standard vertex coloring technique to get independent sets of nodes in the graph. As no two adjacent nodes have same color, each color class forms an independent set. The minimum number of colors required to color a graph G is called its chromatic number denoted by $\chi(G)$. As Gauss-Seidel method allows us to update nodes in any arbitrary order, we update them in the order of color class i.e., update nodes in one color class after the other. Within a color class, all the nodes can be updated in parallel as they form an independent set. Though an easy $\Delta + 1$ coloring is possible for any graph, where Δ is maximum degree, a $\chi(G)$ coloring is known to be NP-hard in general. We use the standard greedy coloring algorithm employed in [5], which gives a 6 coloring for planar graphs. Let the vertices of the graph be ordered as u_1, u_2, \dots, u_n , in such a way that u_i is a minimum degree vertex in the graph induced by vertices $\{u_1, u_2, \dots, u_i\}$. Now color each vertex with a free color in the order u_1, u_2, \dots, u_n .

4.9.2 Graph Representation

The memory representation of graph used for GPU processing has significant impact on the performance. Graph data includes (a) Data corresponding to each vertex - degree, value at vertex etc. (b) Edge information - indices of neighboring vertices and their corresponding scale factors etc. Vertex data is re-ordered according to the color of vertices i.e., data of all the vertices having same color will be co-located. An array of pointers is maintained to store the starting index of each color class. The implementation processes the vertices of each color in sequence and for each color, creates as many threads as the number of vertices in the color class. Re-ordering the data according to color results in coalesced memory access [5] as show in Fig 4.6 (adapted from [5]).

To store edge data, we use the semi-compact column major matrix representation as in [5] which requires $O(\Delta \cdot |V|)$ space for a graph with maximum degree Δ , which is generally small for many practical problems. Each column stores the adjacency information of a single vertex. The edge data accessed by threads will be collocated and hence results in a coalesced access as shown in Fig 4.7 (adapted from [5]).

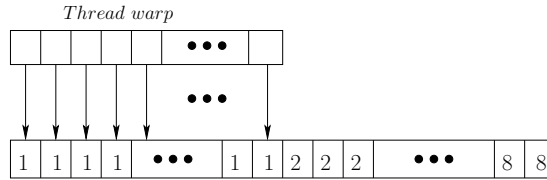


Figure 4.6: Re-ordered for coalesced memory access

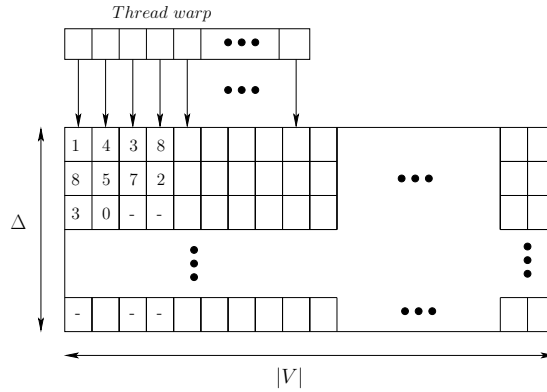


Figure 4.7: Coalesced memory access in column major adjacency

4.9.3 Multigrid Implementation

Hierarchy of grids created during pre-processing phase are stored on device memory. The inter-grid transfer operators which include prolongation and restriction matrices (stored in column major matrix representation) are also stored as part of grid. Following GPU kernels are used for implementing different steps in the multigrid method.

- Smoothing The kernel takes the starting and ending index of each color class, creates as many thread as the number of vertices in the color class and updates the value at each vertex.
- Restriction Two kernels are used for implementing restriction operation. One of the kernels creates as many threads as the number of vertices in the finer mesh and calculates residual at each vertex. The other kernel creates as many threads as the number of vertices in the coarser mesh and updates residual at each vertex using restriction matrix.
- Prolongation Two kernels are used for implementing prolongation operation as well. One of them creates as many threads as the number of vertices in the coarser mesh and calculates error correction at each vertex. The other kernel creates as many threads as the number of vertices in the finer mesh and updates the value at each vertex using prolongation matrix.

We use V -cycle multigrid, which is made up of a down cycle and up cycle. Down cycle is a sequence of smoothing and restriction operations performed alternately starting from finest grid till we reach coarsest grid. Up cycle is a combination of prolongation and smoothing operations performed alternately starting with the coarsest grid till we reach finest grid. The multigrid V -cycle is repeated till the desired convergence is reached.

4.10 Classical Algebraic Multigrid

Classical AMG has been implemented to accelerate the performance of the solver. AMG implementation led to considerable speedups in most of the problems. But some discrepancies were found while doing experiments with the code.

4.10.1 Selection of boundary nodes

In the jet problem, the inlet boundary patch is quite small, as compared to that of the domain size. When the AMG is performed, with equal preference to interior and boundary cells, with number of levels more than 2, it was found that insufficient number of inlet nodes were taken for simulation, hence giving false results, or no results at all. One of the possible ways was to increase the number of inlet boundary cells, but that would also lead to increase in preprocessing and computational time. In order to avoid this from happening the priority for selecting the boundary nodes was increased in each level. This resulted in selection of more number of boundary nodes, and better results.

4.10.2 Classical AMG and Non orthogonal unstructured grids

According to Classical AMG theory as referred in 4.5.1, the $Au = f$ is the format of the equation to be solved. As we have also seen in the same section, the matrix A must be diagonally dominant, that is one of the pre requisites of AMG to be successfully applied. Also the discretization of pressure poisson leads to introduction of cross diffusion terms. The detailed discretization has been given in Appendix A. Upon seeing the discretization on non orthogonal grids, it is clear that the final equation involves the dependence of cell center on its neighbours and its neighbours of neighbours, increasing the degree of connectivity for each cell. Experimentally, it was found that the matrix A was no longer diagonally dominant, and hence the solver diverged, with any given grid of mesh skewness 0.85. This problem can be eliminated using a modified algorithm for AMG as given in Lonsdale et al [26].

Hence the current solver is capable of solving orthogonal unstructured grids on GPU accelerated AMG, but for non orthogonal grids, pressure poisson has been accelerated as given in section 4.8. This has been considered as the future scope for the work.

4.11 Closure

In this section, we discussed the basics of Algebraic Multigrid (AMG) to solve a set of linear equations. We also looked into modified algorithm for better coarsening through giving high priority to boundary nodes. Implementation of AMG and Gauss seidel on GPU through greedy colouring was also discussed along with experimental findings and shortcomings of the algorithm.

Chapter 5

Laminar Validation cases

5.1 Introduction

The code has been validated for 3D laminar benchmark problems. The problems have been also tested with and without AMG and GPU for speedup and accuracy, and have also been compared with the benchmark problems.

The problems that have been considered are:

1. 3D Lid driven cavity.
2. Flow over Square Cylinder at low Reynold numbers.
3. Heat transfer through solid cube.

The results have been presented and discussed in the following sections of this chapter.

5.2 Lid Driven Flow

The lid-driven cavity problem has long been used a test or validation case for new codes or new solution methods. The problem consists of Dirichlet boundary condition at all sides with fluid contained in a cubical domain, with all stationary sides and one moving side (with velocity tangent to the side). Pressure has uniform neumann boundary condition on all the sides. The domain has been shown schematically in figure 5.1. This problem is a nice one for testing for several reasons. First, as mentioned above, there is a great deal of literature to compare with. Second, the (laminar) solution is steady. Third, the boundary conditions are simple and compatible with most numerical approaches.

The domain size for our problem $1 \times 1 \times 1$, with different number of unstructured hexahedral cells. The figures below represent the results obtained by coarsest configuration on 0.1 million cells for Reynolds Number 100 and 1000. Results have been compared for X and Y component of velocity with vertical and horizontal centerline which have been presented in figure 5.2. For same number of cells and same flow conditions, results have also been computed for unstructured tetrahedral cells. The results obtained have been compared against Ku. et. al. [27].

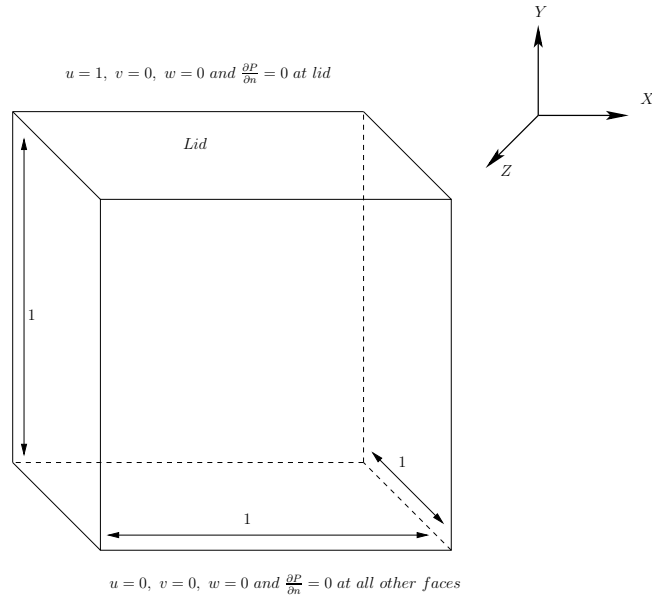


Figure 5.1: Domain for Lid Driven Cavity

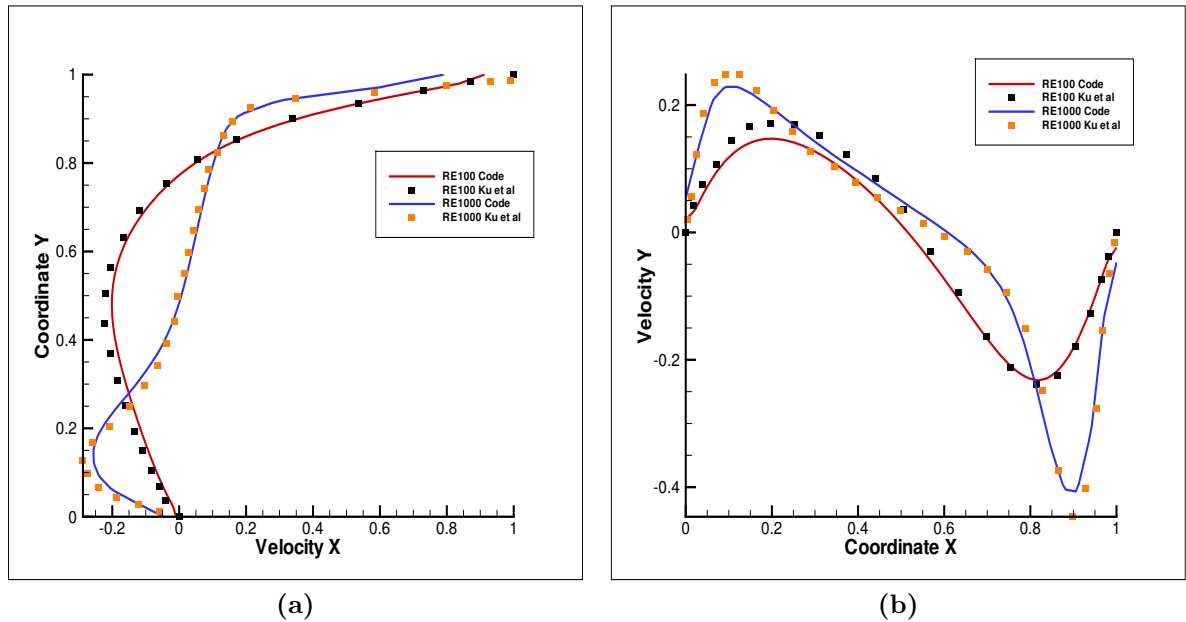


Figure 5.2: Comparison of (a) the horizontal velocity component along the vertical centerline and (b) the vertical velocity component at the horizontal centerline of cubical lid driven cavity at $Re = 100$ and 1000 with the results of Ku et al. [27]

Figure shows the results to be in good agreement with the standard literature. Validations were performed for both serial and parallel code.

Table 5.1 shows comparison of multigrid speedup obtained by GPU and GPU Multigrid over serial code for different mesh sizes in first timestep of solving Pressure poisson.

Table 5.1: Pressure Poisson Solve Time Comparison for Flow Over Lid Driven Cavity Problem

Grid Size	serial	GPU without MG	GPU with MG
1060000	3 hr 8 min 54 sec	21.09 sec	9.69 sec
1580000	5 hr 23 min 33 sec	42.89 sec	18.84 sec
2100000	11 hr 30 min 50 sec	1 min 16 sec	33.09 sec
2620000	20 hr 8 min 37 sec	2 min 4 sec	1 min 19 sec

5.3 Flow over Square Cylinder

In this section we analyze flow over square cylinder, the domain for which has been sketched in fig. aa. The flow simulated is essentially 3D, but to test the code and boundaries, as well as the stability of code for unsteady problem and for concave mesh, this problem was chosen. The results have been compared in fig 5.3 with work of breuer et. al. [28] for a range of Reynold number. The reynold number for this problem is defined as : $RE = \frac{UD}{\nu}$, where U is the velocity through the inlet, D is the side of the square, and ν is the kinematic viscosity of the fluid.

When the flow is set upon a square cylinder, at very low reynold number ($RE=0.7$) which is <1 , we see that creep flow develops as viscous forces dominate leading to no separation of flow, but as we increase the reynold number beyond 1, it leads to formation of wake beyond the cylinder where a recirculation zone is formed leading to a low pressure zone just behind the cylinder. With increasing Re, the flow separates first at the trailing edges of the cylinder and a closed steady recirculation region consisting of two symmetric vortices is observed behind the body. The size of the recirculation region increases with increase in RE.

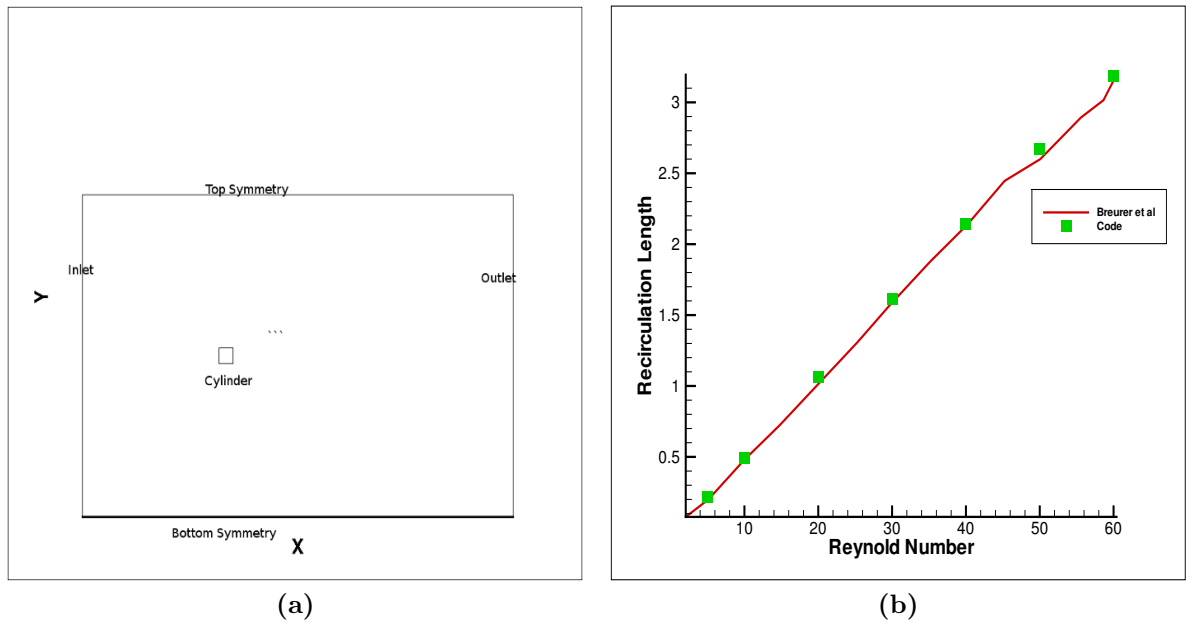


Figure 5.3: Comparison of (a) The domain for flow over square cylinder and (b) Validation of recirculation length vs reynold number with results of Breuer et al. [28]

Different meshes were used to test the solver and mesh indepedece and speedup obtained from

mulgrid for the problem. As the problem is two dimensional in current regime(ref), and the solver can handel three dimensional problems, the grid is only a single unit in the Z direction, i.e only a single cell width is given in Z direction, and symmetry boundary condition is applied on the faces (front and back) normal to Z direction. The boundary conditions imposed are shown as below:

- **Inlet:** $u = 1; v = w = 0; \frac{\partial P}{\partial n} = 0$
- **Symmetry front and back :** $\frac{\partial u}{\partial n} = \frac{\partial w}{\partial z} = \frac{\partial P}{\partial z} = 0, w = 0$
- **Symmetry top and bottom :** $\frac{\partial u}{\partial n} = \frac{\partial w}{\partial n} = \frac{\partial P}{\partial z} = 0, v = 0$
- **Outlet :** $\frac{\partial u}{\partial n} = \frac{\partial v}{\partial n} = \frac{\partial w}{\partial n} = 0, P = 0$
- **Cylinder :** $u = v = w = 0; \frac{\partial P}{\partial n} = 0$ where “n” is the direction normal to the plane of boundary.

The flow has the property to be steady below Reynold number 54. The recirculation zone starts forming at around RE 1, and as we keep on increasing the Reynold number, When this critical Reynolds number Re_{crit} is exceeded, the well-known von Karman vortex street with periodic vortex shedding from the cylinder can be detected in the wake.As we go beyond RE, the wake becomes unstable and starts shedding vortices that travel through the domain and exit through outlet boundary, without inducing any numerical instability in the flow.

Simulation has not been performed beyond RE 60, as the intention is to test for laminar flow, steady and unsteady (vortex street) and to not go in transition and turbulent regime. Moreover this regime gives us a 2 dimensional flow. The streamlines plots for the same have been presented in the figures below:

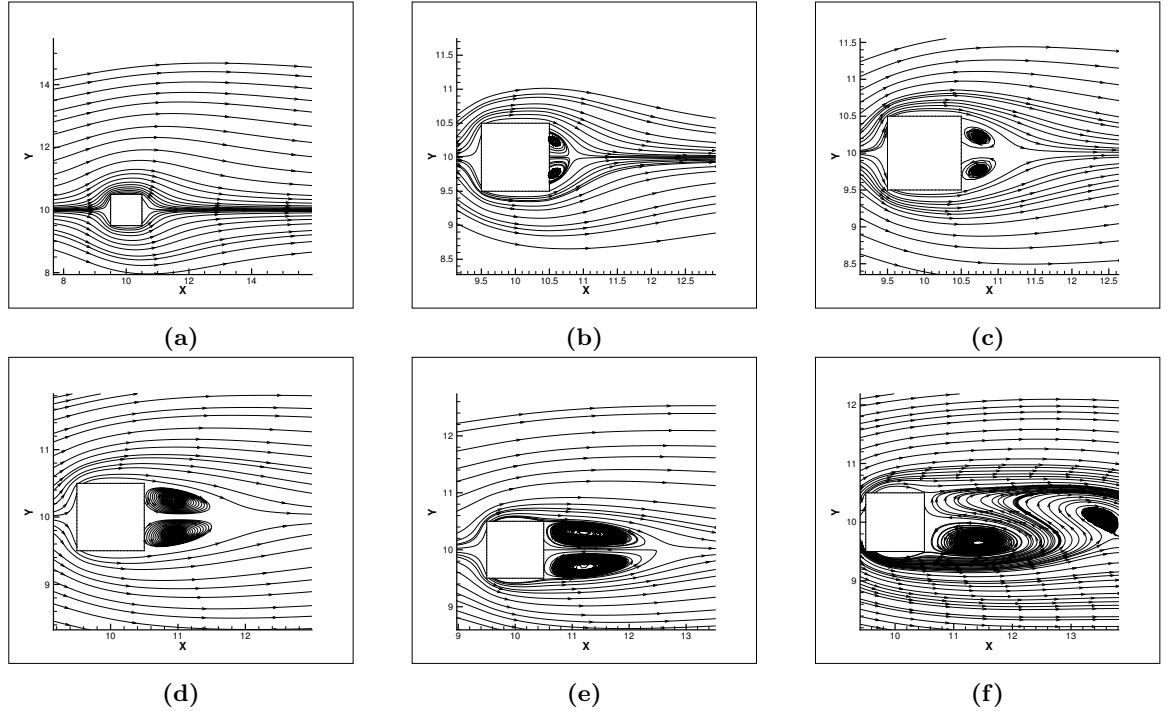


Figure 5.4: Streamlines at different Reynold Number. (a) RE0.7, (b) RE5, (c) RE10, (d) RE20, (e) RE60 vortex shedding.

The speedup obtained through GPU and GPU AMG implementation is given as in Table 5.2

Table 5.2: Pressure Poisson Solve Time Comparison for Flow Past Square Cylinder Problem

Grid Size	CPU without MG	GPU without MG	GPU with MG
203748	2 hr 2 min 25 sec	41.35 sec	6.61 sec
302310	3 hr 4 min 40 sec	57.09 sec	9.19 sec
403510	7 hr 13 min 14 sec	2 min 1 sec	15.47 sec
1713160	26 hr 19 min 48 sec	13 min 53 sec	1 min 33 sec

5.4 Heat transfer through solid cube

A solid cube represents a case of pure conduction. The temperature equation can be substituted in the scalar equation for pure conduction and forced convection, where the temperature field does not affect the flow field. To validate the scalar implementation as described in ref chap, the scalar variable ϕ was considered as Temperature. As it is a solid, there would be no convection. Same mesh used for Lid driven cavity is used for the purpose. The material was assumed to be Copper, which has a density as $\rho = 8933kg/cm^3$, thermal sensible heat transfer coefficient as $C_p = 385$ and thermal conductivity $K = 401$. The diffusion coefficient γ can be defined for heat transfer problem as $\gamma = \frac{K}{C_p}$. The problem is solved for following boundary conditions:

- **Top, Bottom, Right, Left, Rear Wall:** $u = v = w = 0; \frac{\partial P}{\partial n} = 0, \phi = 300K$
- **Bottom Wall :** $u = v = w = 0; \frac{\partial P}{\partial n} = 0, \phi = 600K$ where “n” is the direction normal to the plane of boundary.

The analytical solution for a 2D conduction problem can be given as:

$$T = T_1 + (T_2 - T_1) * \left(\frac{2}{\pi}\right) * \left(\sum_{n=1}^{\infty} \frac{(-1)^{n+1} + 1}{n} \sin \frac{n\pi x}{W} \frac{\sinh \frac{n\pi y}{W}}{\sinh \frac{n\pi H}{W}}\right) \quad (5.1)$$

details on which can be found in works by Holman [29]. The results have been compared at Z= 0.5 plane of the geometry for X center line.

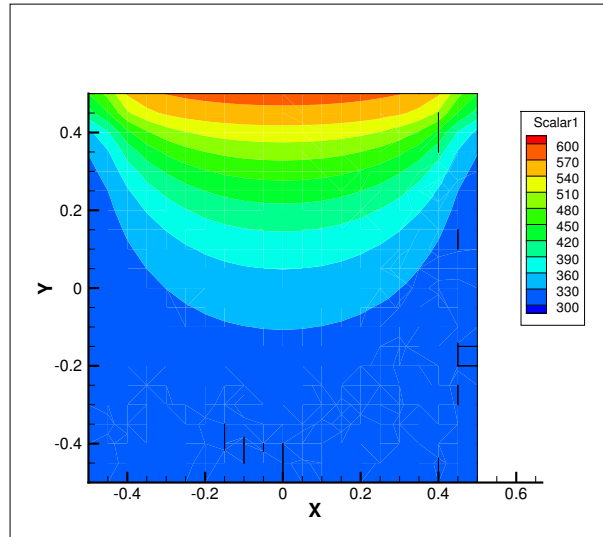


Figure 5.5: Contours of Scalar at Z= 0.5 midplane

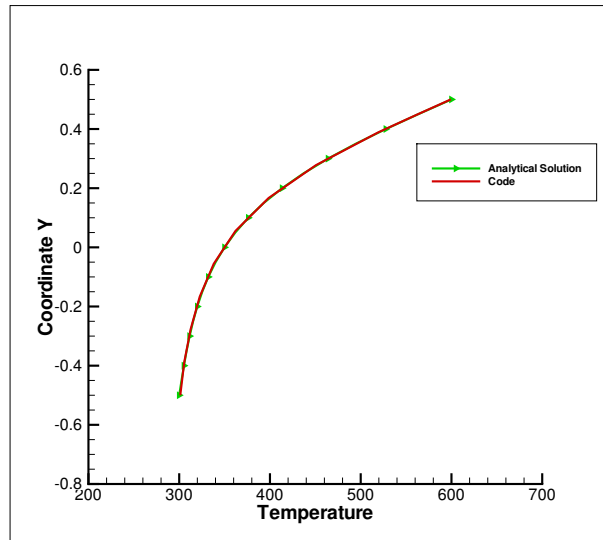


Figure 5.6: Comparison of analytical and numerical solution

5.5 Closure

In this chapter, we have seen the validation of the code against problems listed in section 5 and the speedup obtained when compared with serial code. The serial simulations were carried out on Intel Xeon E5-2600 2.60 GHz processor. Parallel simulations which includes non-multigrid and multigrid GPU implementations of AMG solver are run on NVIDIA Kepler K20Xm GPU with CUDA driver version 5.5. The GPU has 2668 cores, 6GB device memory. We can conclude that the solver is fairly accurate for the type of problems we aim to solve.

Chapter 6

Turbulence modeling using Large Eddy Simulation

6.1 Introduction

Turbulence is ubiquitous phenomenon. Fluid flows generally encountered in engineering applications are turbulent in nature which is characterized by

1. Random and Chaotic nature - temporally and spatially.
2. Diffusivity - intense mixing due to the fluctuating quantities.
3. Large Reynolds number phenomenon - occurs at high Re number regime.
4. Three-Dimensionality 1 - fluctuations are three dimensional even though the mean flow field is two dimensional.
5. Rotationality - always involves vortices.
6. Dissipative - more energy consuming compared to laminar flow.
7. Continuum - smallest possible eddies in the flow field are larger than the molecular mean free path

Whenever a moving fluid enters a quiescent body of the same fluid, a velocity shear is created between the entering and ambient fluids, causing turbulence and mixing. There are many computational strategies to solve the turbulent fluid flow problems which are described in the following section.

6.2 CFD Methods for turbulence

6.2.1 Direct Numerical Simulation

The most accurate of solutions of turbulent flows is Direct Numerical Simulation (DNS) which solves the complete Navier-Stokes equations directly without any modelling assumptions. A good DNS has to resolve the complete spectrum of length scales which is very broad in a turbulent fluid flow. A

very fine grid has to be taken in order to resolve the smallest possible length scale, defined by the Kolmogorov length scale, which in turn increases the number of grid points needed. In DNS, the number of grid points are related to Reynold number as

$$N \propto Re^{\frac{3}{4}} \quad (6.1)$$

As we can see the computational cost for DNS is very large, and hence is avoided in industries. DNS is generally used for academic purpose. It is also regarded as numerical experiment as the results obtained are highly accurate.

6.2.2 Large Eddy Simulation

One of the methods to simulate turbulent flows is Large eddy simulation (LES). Turbulence consists of a continuous spectrum of scales ranging from the largest scale to the smallest scale. The transfer of energy from one scale to other is called energy cascading.

An implication of Kolmogorov's theory of self-similarity is that the large eddies of the flow are dependent on the geometry while the smaller scales more universal. This feature allows one to explicitly solve for the large eddies in a calculation and implicitly account for the small eddies by using a sub-grid-scale model (SGS model). Hence complete Navier stokes is solved for energy containing large eddies, whereas eddies in sub inertial region are modelled using some mathematical model. Mathematically, one may think of separating the velocity field into a resolved and sub-grid part. The resolved part of the field represent the "large" eddies, while the subgrid part of the velocity represent the "small scales" whose effect on the resolved field is included through the subgrid-scale model. Formally, one may think of filtering as the convolution of a function with a filtering kernel G :

$$\bar{u}_i(\vec{x}) = \int G(\vec{x} - \vec{\xi}) u(\vec{\xi}) d\vec{\xi}, \quad (6.2)$$

resulting in $u_i = \bar{u}_i + u'_i$

, where \bar{u}_i is the resolvable scale part and u'_i is the subgrid-scale part. However, most practical (and commercial) implementations of LES use the grid itself as the filter (the box filter) and perform no explicit filtering

Navier stokes equation is solved for region up to sub grid region decided using low pass filtering. For sub inertial region, terms are modelled. They appear like viscous terms, hence are called sub grid stress modelling. Static Smagorski Model of LES has been implemented in the solver for simulation of turbulent flows.

The filtered equations are developed from the incompressible Navier-Stokes equations of motion:

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{1}{\rho} \frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j} \left(\nu \frac{\partial u_i}{\partial x_j} \right). \quad (6.3)$$

Substituting in the decomposition $u_i = \bar{u}_i + u'_i$ and $p = \bar{p} + p'$ and then filtering the resulting equation gives the equations of motion for the resolved field:

$$\frac{\partial \bar{u}_i}{\partial t} + \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} = -\frac{1}{\rho} \frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j} \left(\nu \frac{\partial \bar{u}_i}{\partial x_j} \right) + \frac{1}{\rho} \frac{\partial \tau_{ij}}{\partial x_j}. \quad (6.4)$$

We have assumed that the filtering operation and the differentiation operation commute, which is not generally the case. It is thought that the errors associated with this assumption are usually small, though filters that commute with differentiation have been developed ("ref?"). The extra term $\frac{\partial \tau_{ij}}{\partial x_j}$ arises from the non-linear advection terms, due to the fact that

$$\overline{u_j \frac{\partial u_i}{\partial x_j}} \neq \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} \quad (6.5)$$

and hence

$$\tau_{ij} = \bar{u}_i \bar{u}_j - \overline{u_i u_j} \quad (6.6)$$

Similar equations can be derived for the subgrid-scale field (i.e. the residual field). Subgrid-scale turbulence models usually employ the Boussinesq hypothesis, and seek to calculate (the deviatoric part of) the SGS stress

Static Smagorinski Model

Static Smagorinski Model could be summarized as:

$$\tau_{ij} - \frac{1}{3} \tau_{kk} \delta_{ij} = -2(C_s \Delta)^2 S_{ij} |\bar{S}| \quad (6.7)$$

Where the filter width is usually taken to be

$$\Delta = (\text{Volume})^{\frac{1}{3}} \quad (6.8)$$

In Static Smagorinsky model, eddy viscosity is modelled by:

$$\nu_{sgs} = (C_s \Delta)^2 |\bar{S}| \quad (6.9)$$

S is symmetric tensor and can be represented as:

$$\bar{S}_{ij} = \frac{1}{2} \left(\frac{\partial \bar{u}_i}{\partial x_j} + \frac{\partial \bar{u}_j}{\partial x_i} \right) \quad (6.10)$$

and

$$|\bar{S}| = \sqrt{2 S_{ij} S_{ij}} \quad (6.11)$$

Hence the effective viscosity is calculated:

$$\nu_{eff} = \nu_{molecular} + \nu_{sgs} \quad (6.12)$$

Hence the complete Navier Stokes equation is given as:

$$\frac{\partial \bar{u}_i}{\partial t} + \bar{u}_j \frac{\partial \bar{u}_i}{\partial x_j} = -\frac{1}{\rho} \frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j} \left([\nu + \nu_t] \frac{\partial \bar{u}_i}{\partial x_j} \right), \quad (6.13)$$

where we have used the incompressibility constraint to simplify the equation and the pressure is now modified to include the trace term $\tau_{kk} \delta_{ij}/3$. For C_s , value around 0.1 is found to be good for various engineering flow problems. For the scalar equation, the Static Smagorinski model can be

given as:

$$\gamma_{sgs} = \frac{(C_s \Delta)^2 |\bar{S}|}{S_c} \quad (6.14)$$

hence

$$\gamma_{eff} = \gamma + \gamma_{sgs} \quad (6.15)$$

where S_c is the Schmidt number of the flow.

6.3 Validation

The turbulent flow has been simulated for a Reynolds Number of 11000 and has been validated with the results of Panchapakesan and Lumley [3]. Simulations have been carried out with Static Smagorisky model for modeling turbulence. Two inlets were simulated for the flow, a square inlet and a circular inlet.

6.3.1 Turbulent Jet

The domain that has been used for the simulation along with the boundary conditions has been shown below.

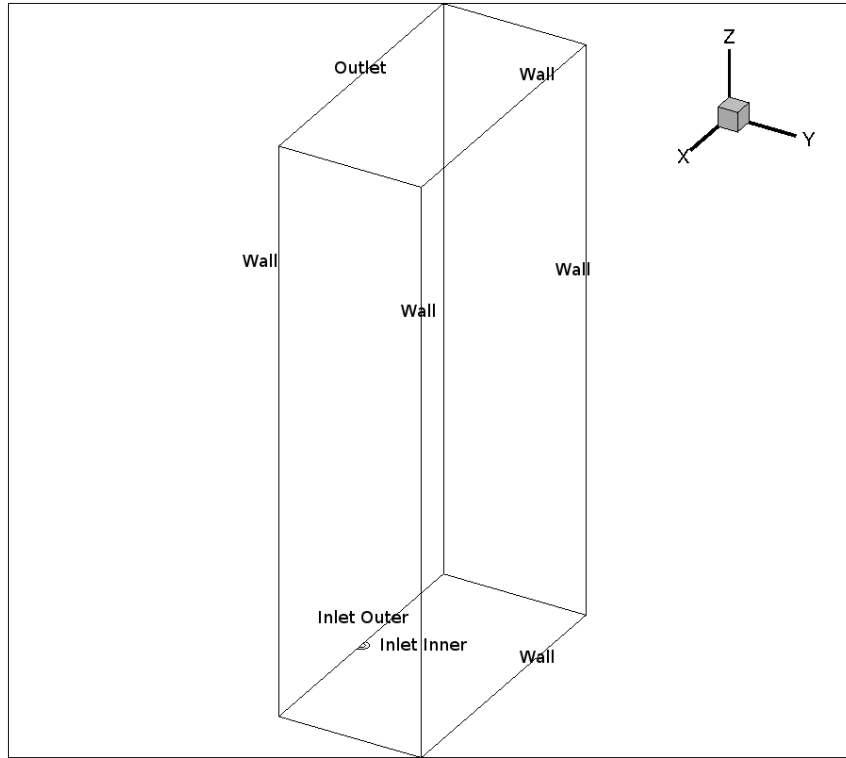


Figure 6.1: 3 Dimensional Domain for simulation of Turbulent Jet

Boundary conditions

- **Inlet Inner:** $w = 1; u = v = 0; \frac{\partial P}{\partial n} = 0 \phi = 1$

- **Inlet outer :** $w = u = v = 0; \frac{\partial P}{\partial n} = 0 \phi = 0$
- **Outlet :** $\frac{\partial u}{\partial n} = \frac{\partial v}{\partial n} = \frac{\partial w}{\partial n} = 0 P = 0 \frac{\partial \phi}{\partial n} = 0$
- **Bottom Symmetry :** $\frac{\partial u}{\partial n} = \frac{\partial w}{\partial n} = v = 0 P = 0 \frac{\partial \phi}{\partial n} = 0$
- **Walls :** $u = v = w = 0; \frac{\partial P}{\partial n} = 0 \frac{\partial \phi}{\partial n} = 0$ where 'n' is the direction normal to the plane of boundary.

The size of the domain is $40D \times 20D \times 80D$, with the inner inlet of D and outer inlet of $2D$. Only half of the domain has been simulated to lower the computational cost of the simulation. The number of cells for which the simulation was carried out was 2.3 million unstructured hexahedral cells.

The structures obtained are shown in the figures below for flow time of 12 seconds:

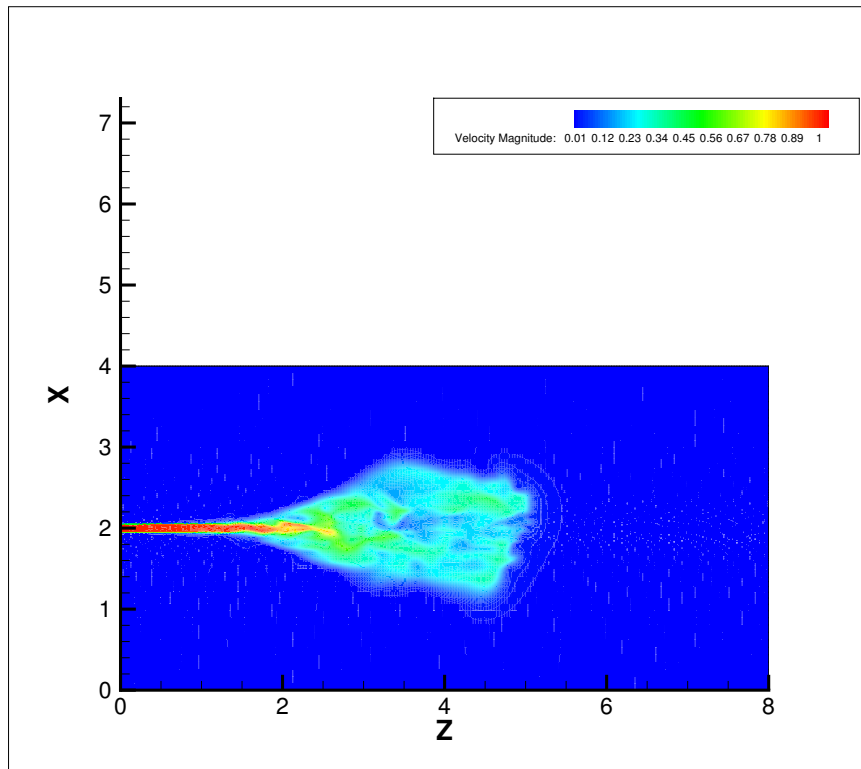


Figure 6.2: Velocity contours for round Turbulent Jet at $Y = 0$ plane

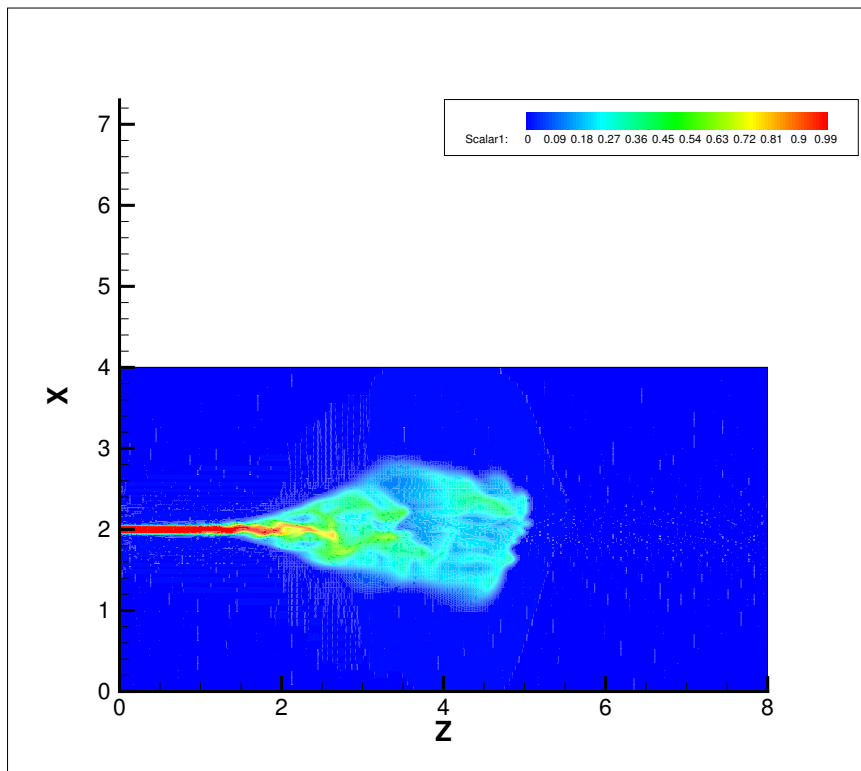


Figure 6.3: Scalar contours for round Turbulent Jet at $Y = 0$ plane

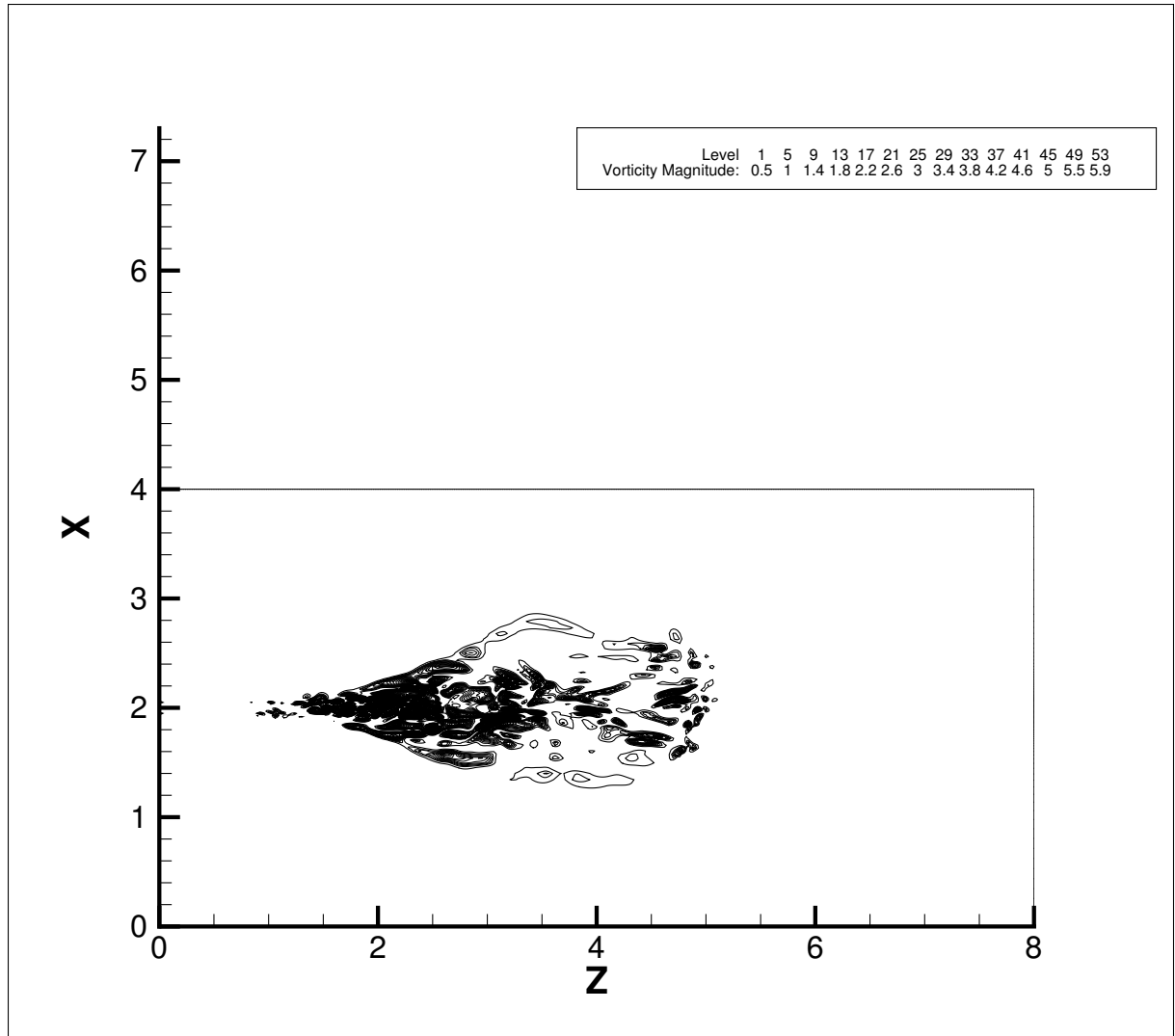


Figure 6.4: Vorticity contours for round Turbulent Jet at $Y = 0$ plane

Chapter 7

Numerical Simulation and Analysis of Coaxial Jets

7.1 Introduction

Jets are of great practical relevance to both engineering applications and natural phenomena. Understanding the nature of Jets is crucial in many fields like mixing of fuels in an automobile or a jet engines, dispersion of pollutants in the atmosphere, cooling of gas turbine blades and exhaust gas cooling etc. This use of coaxial jets is wide spread in context of air blast atomizer. The coaxial jets geometry, operating with a large outer (annular) to inner (central) momentum ratio is used for its ability to destabilize, fragment and mix the central stream in the outer, rapid stream at higher speeds. Jet emerging into quiescent fluid in presence of walls from a single inlet at Reynold number 10 and 100 have been discussed. In present work, the properties of single phase laminar coaxial jets for different velocity ratio $V_r = \frac{U_{outer}}{U_{inner}}$ for RE 10 has been discussed, mainly focusing on changes in the structure of inner jet, due to fast moving outer jet.

7.2 Jet emerging into quiescent fluid

Free shear layers are unaffected by walls and develop and spread in open ambient fluid. They possess velocity gradients, created by some upstream mechanism, which try to smooth out viscous diffusion in presence of convective deceleration. Study of Round jet issuing into quiescent air is a classical free shear problem. A jet emerging into still and ambient and identical fluid from its inlet is considered for simulation in presence of walls, which are sufficiently away. It is investigated whether the free behaviour of the flow is retained in such a case. The domain simulated for our problems is given in figure 7.1.

The diameter of inlet of domain is D , and the domain size is $5D$ radially, and $50D$ in axial direction. The flow has been defined using the inlet Reynold number, defined as $RE = \frac{UD}{\nu}$ where U is the inlet velocity, D is inlet diameter of the inner inlet, and ν is the kinematic viscosity of the fluid. The boundary conditions for the problem have been discussed as below.

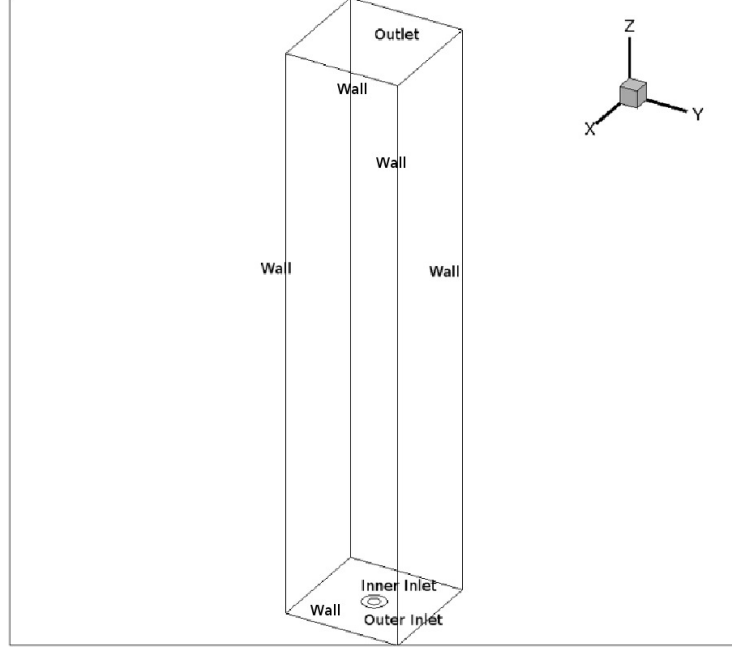


Figure 7.1: Domain for Free shear jet

- **Inlet Inner:** $w = 1; u = v = 0; \frac{\partial P}{\partial n} = 0 \phi = 1$
- **Inlet outer :** $w = u = v = 0; \frac{\partial P}{\partial n} = 0 \phi = 0$
- **Outlet :** $\frac{\partial u}{\partial n} = \frac{\partial v}{\partial n} = \frac{\partial w}{\partial n} = 0 P = 0 \frac{\partial \phi}{\partial n} = 0$
- **Walls :** $u = v = w = 0; \frac{\partial P}{\partial n} = 0 \frac{\partial \phi}{\partial n} = 0$ where 'n' is the direction normal to the plane of boundary.

Mesh of 0.5 million unstructured hexahedral elements have been used for both the cases. If the jet emerges with sufficient momentum, it remains narrow and grows slowly, with radial changes being much larger than axial.

The simulations have been carried out for inlet Reynold numbers of 10 and 100, for Navier Stoke and species transpor equations using semi implicit segregated algorithm as described in section 3.6.1.

7.2.1 Jet emerging from inlet at Reynold Number 10

Fluid enters the domain through innner inlet and outer inlet remains a part of the wall. Following results were obtained for single jet entering in quincient fluid. This case is the control jet with $V_r = 0$.

The axial profile has been validated against the analytical solution given by Schlichting [30] as:

$$U(r, z) = \frac{3K}{8\pi\nu(z - z_0)} \left(1 + \frac{3K}{64\pi(\nu)^2} \left(\frac{r}{z - z_0}\right)^2\right)^{-2} \quad (7.1)$$

where z_0 is the virtual origin of the jet, r denotes the radial distance from the jet centerline and K is the momentum supplied by point source at $z=z_0$. We find that the results are in good agreement with the numerical simulation. We can see from the figure that the fluid diffuses near the inlet itself

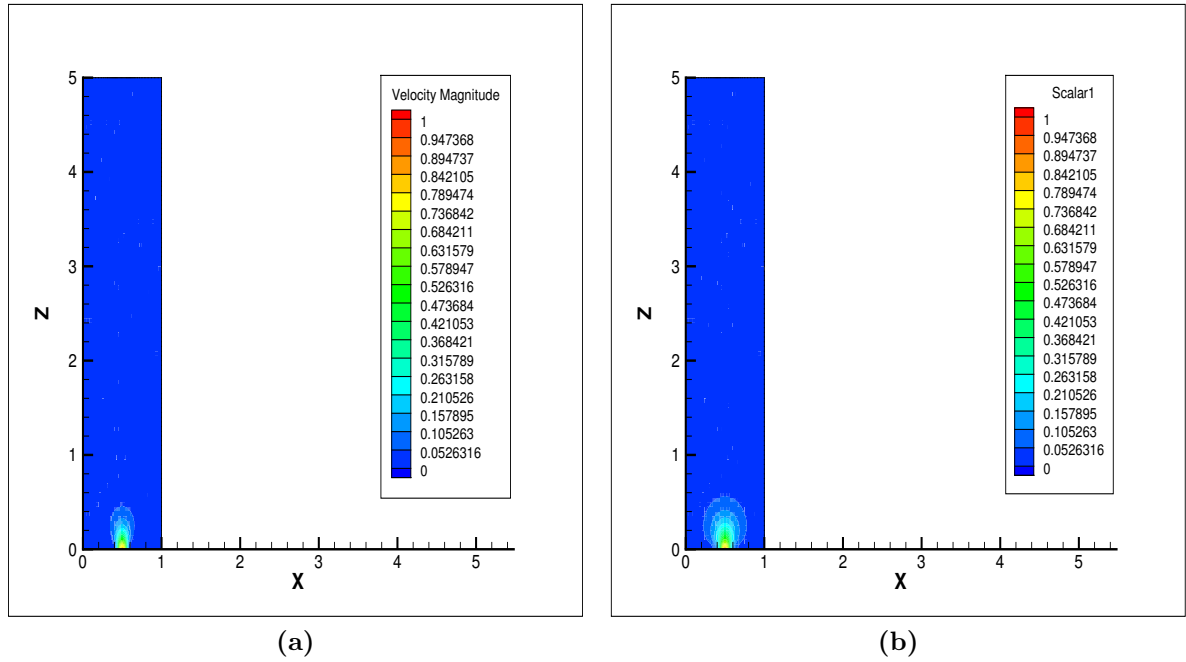


Figure 7.2: Contour at $Y = 5D$ midplane for (a)Velocity (b)Passive Scalar solution at RE 10

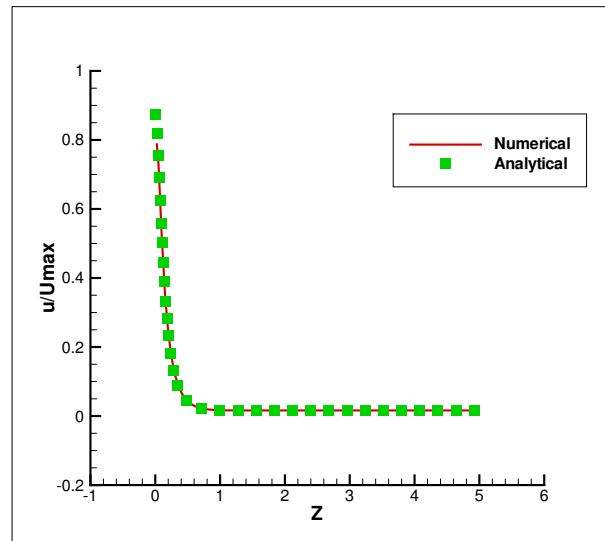


Figure 7.3: Validation against analytical solution for RE = 10

due to low Reynold number and therefore due to low momentum. As the flow simulated is at very low RE and diffuses near the inlet. As the Reynold number is very small, the fluid diffuses and loses its momentum far before it can reach the outlet. The same case was simulated on Fluent commercial solver with same boundary conditions. The comparisons at near wake and far wake have been given in figure 7.4. Figure 7.5, the velocity and the scalar profiles have been normalized with respect to the local maximum values and then plotted. The Schmidt number for the flow can be defined as $Sc = \frac{\mu}{\gamma}$, where γ is the diffusivity of scalar and μ is the dynamic viscosity of the fluid. Schmidt number is taken to be 1, but still the diffusion boundary layer and the momentum boundary layer

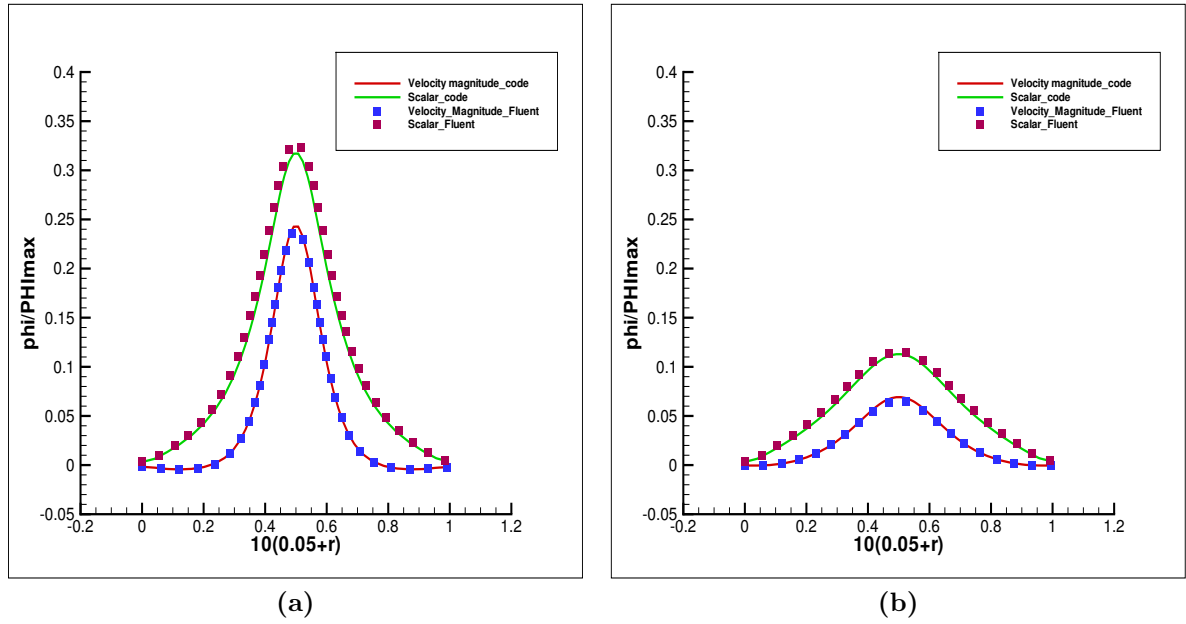


Figure 7.4: Comparison at $Y = 5D$ midplane for (a) NearWake at axial distance of $2D$ (b) FarWake at axial distance of $4D$

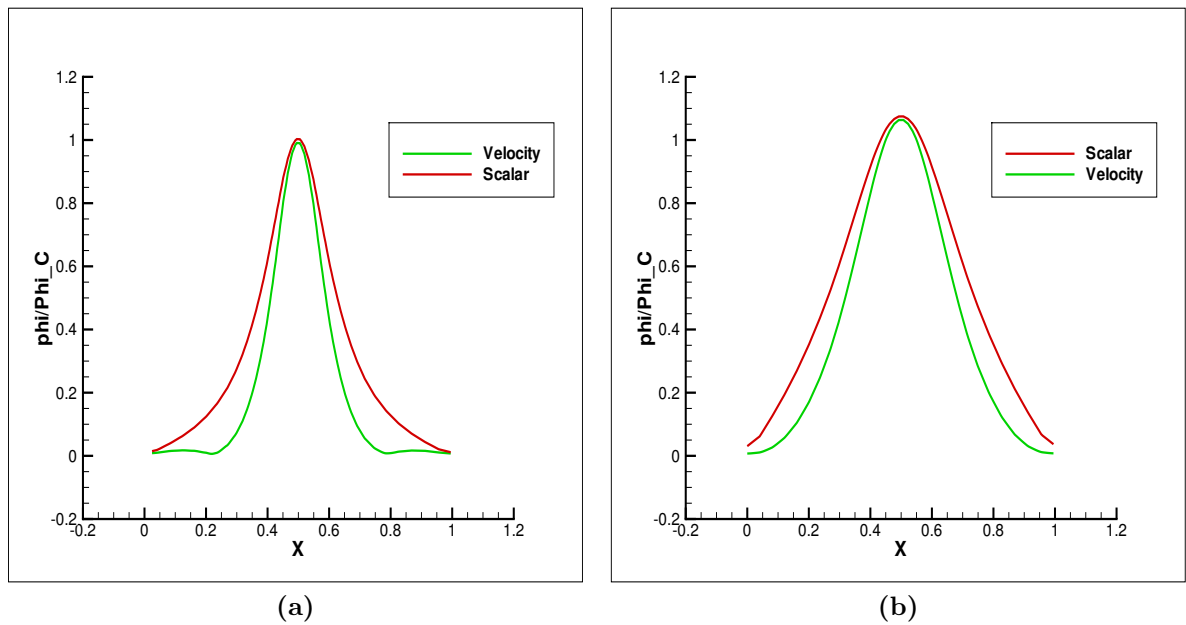


Figure 7.5: Comparison at $Y = 5D$ midplane for normalized with respect to local maxima for velocity and scalar profiles (a) NearWake at axial distance of $2D$ (b) FarWake at axial distance of $4D$

do not coincide with each other.

On plotting the pressure gradient of the flow along the center line of $Y = 5D$ centre plane in figure, we find that the pressure gradient is not zero at the inlet, and hence the difference. Developing jet is displacing a stationary fluid, hence work has to be done as therefore this will act as a sink

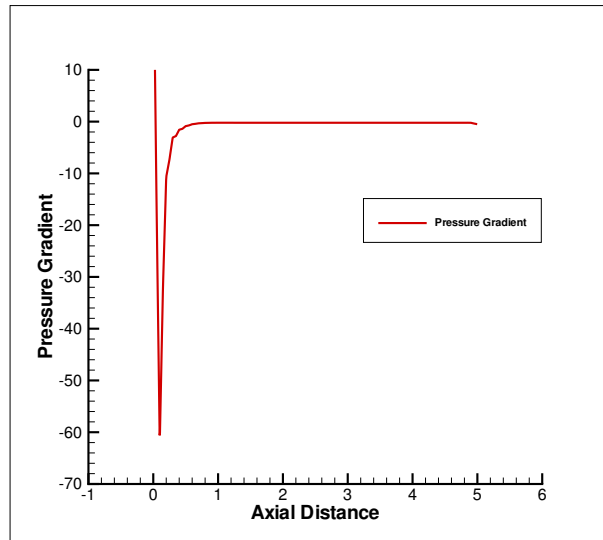


Figure 7.6: Pressure Gradient variation along the axial length of the domain

term in the momentum equation. As we see that the slope is positive, i.e. the jet is working against an adverse pressure gradient and therefore energy is being spent to overcome this adverse pressure gradient.

7.2.2 Jet emerging from inlet at Reynold Number 100

Now the simulations were carried out for inlet $RE = 100$. The velocity and scalar contours have been shown in figure 7.7. The validation was carried out for axial velocity distance using the analytical

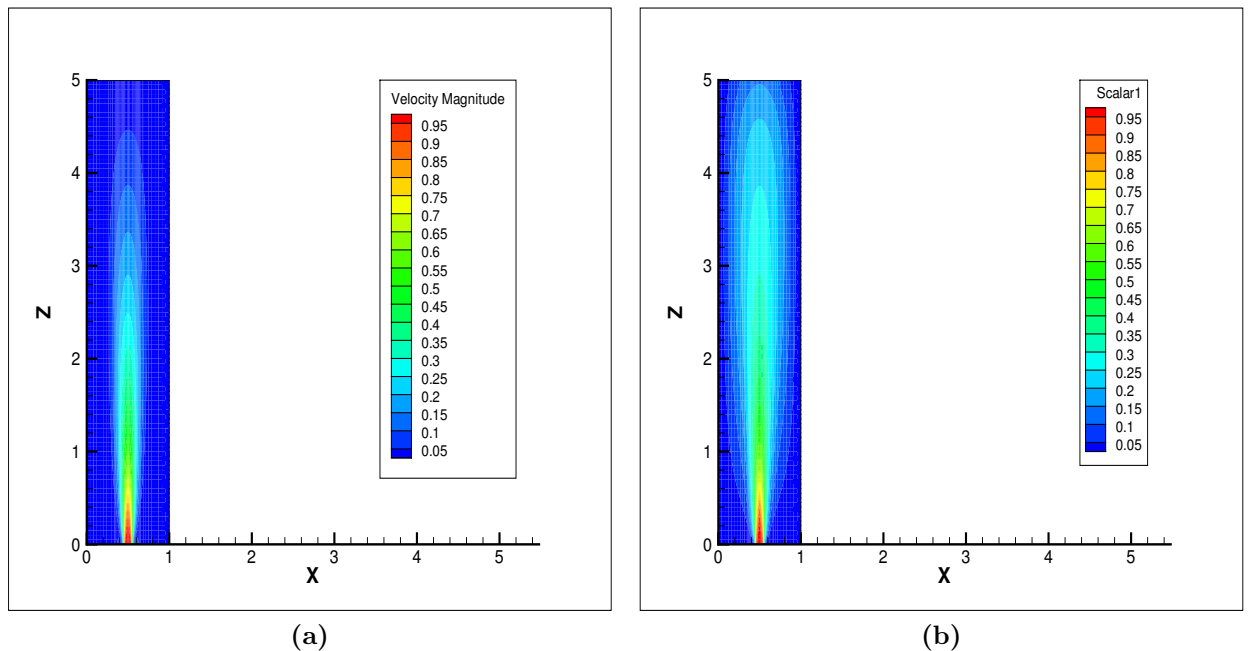


Figure 7.7: Contours of (a) Velocity Magnitude (b) Scalar for $RE = 100$

solution given by equation 7.2.1.

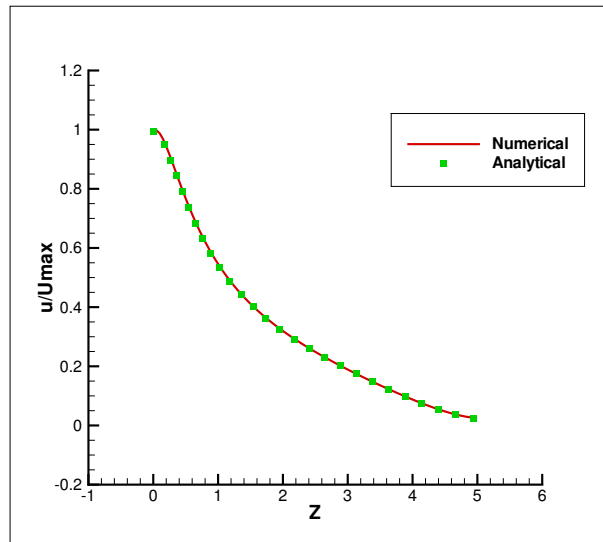


Figure 7.8: Validation against analytical solution for $RE = 100$

As we can see from the figure 7.7, the flow is bounded by the walls, which constrains the fluid flow. The velocity and scalar profiles have been plotted and compared with Fluent for different axial distances, as given in figure 7.9 and figure 7.11 .

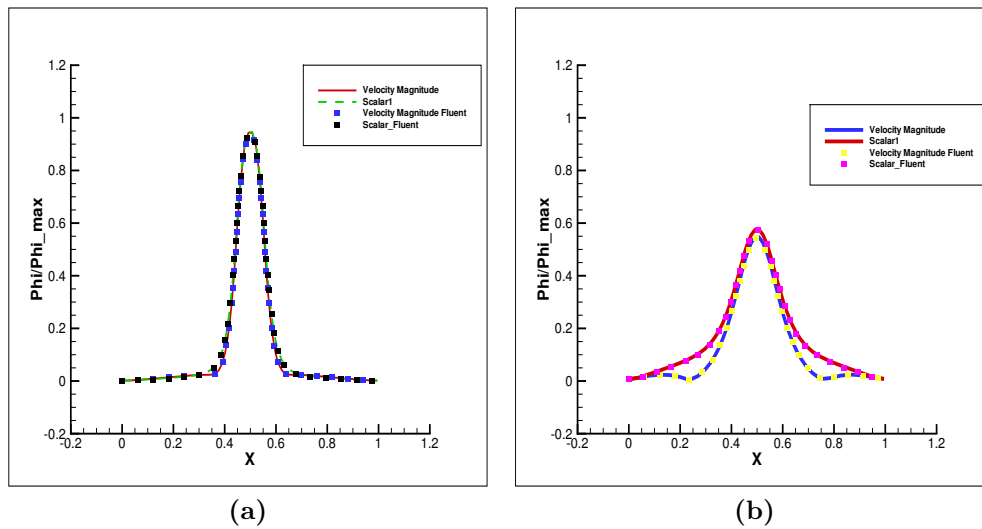


Figure 7.9: Comparison at $Y = 5D$ midplane for (a) NearWake at axial distance of $2D$ (b) NearWake at axial distance of $10D$

The variation of pressure gradient with axial distance has been shown in figure 7.13. Due to presence of pressure gradient throughout the flow, the momentum and diffusion layers donot coincide with each other. The speedup obtained through GPU for first time step for $Re 10$ is shown in table 7.1.

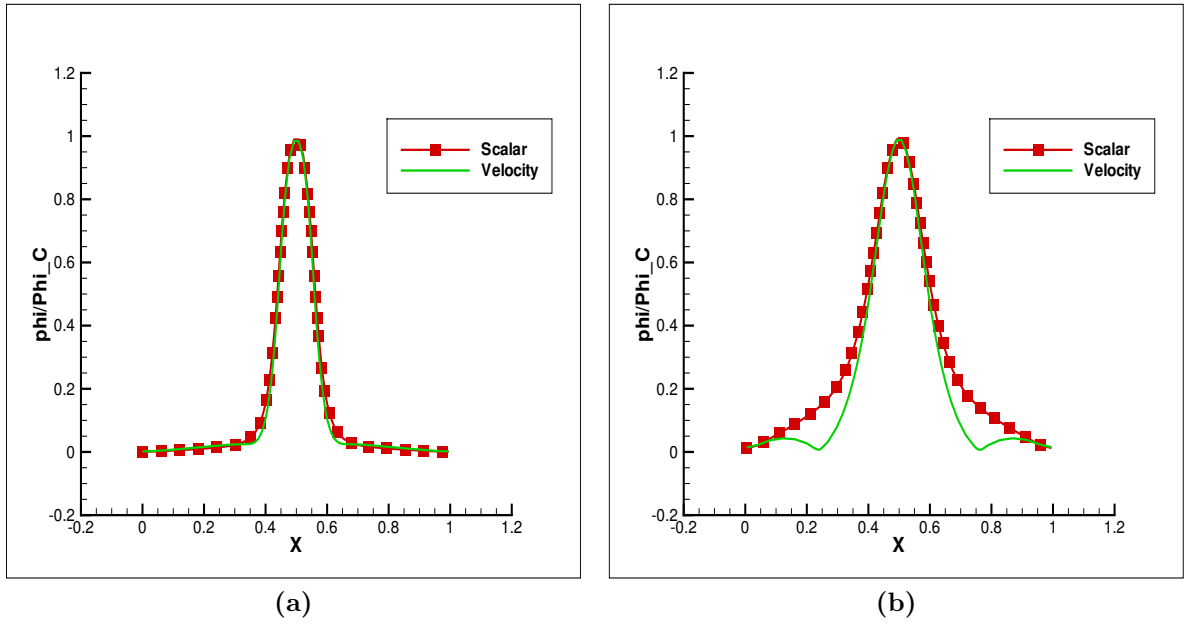


Figure 7.10: Comparison at $Y = 5D$ midplane for normalized with respect to local maxima for velocity and scalar profiles (a)NearWake at axial distance of $2D$ (b)FarWake at axial distance of $4D$

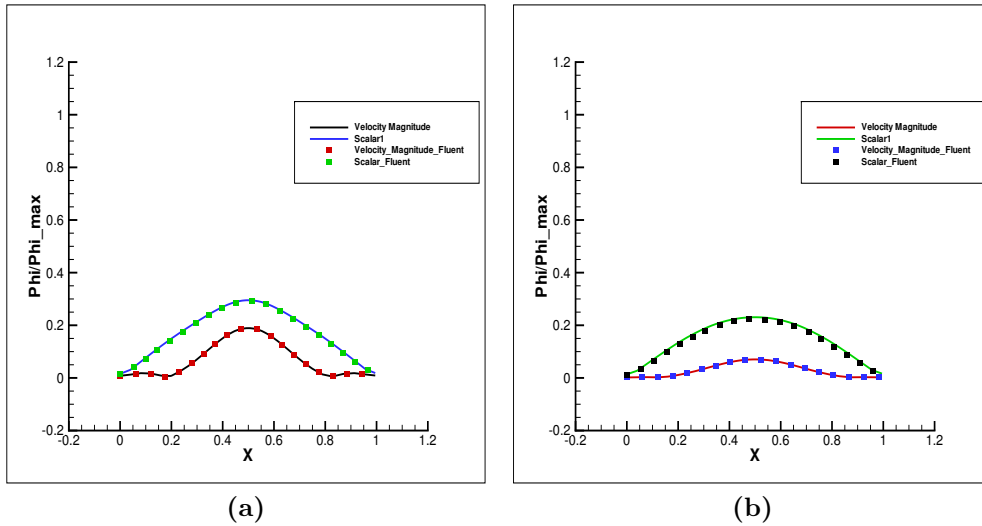


Figure 7.11: Comparison at $Y = 5D$ midplane for (a) FarWake at axial distance of $30D$ (b) NearWake at axial distance of $42D$

Table 7.1: Pressure Poisson Solve Time Comparison for Jet

Grid Size	Serial	GPU without MG	GPU with MG
203748	2 hr 2 min 25 sec	41.35 sec	6.61 sec
302310	3 hr 4 min 40 sec	57.09 sec	9.19 sec
403510	7 hr 13 min 14 sec	2 min 1 sec	15.47 sec
1713160	26 hr 19 min 48 sec	13 min 53 sec	1 min 33 sec

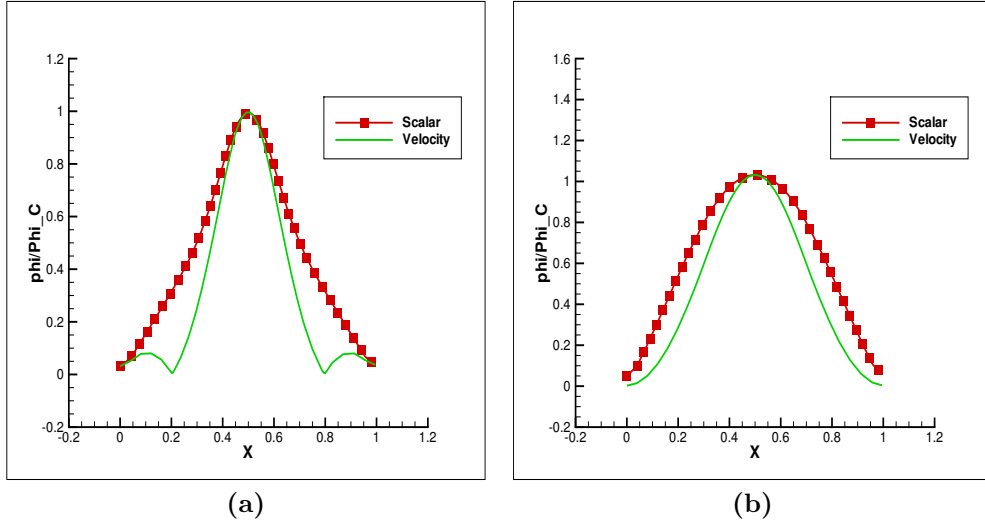


Figure 7.12: Comparison at $Y = 5D$ midplane for normalized with respect to local maxima for velocity and scalar profiles (a)NearWake at axial distance of $30D$ (b)FarWake at axial distance of $42D$

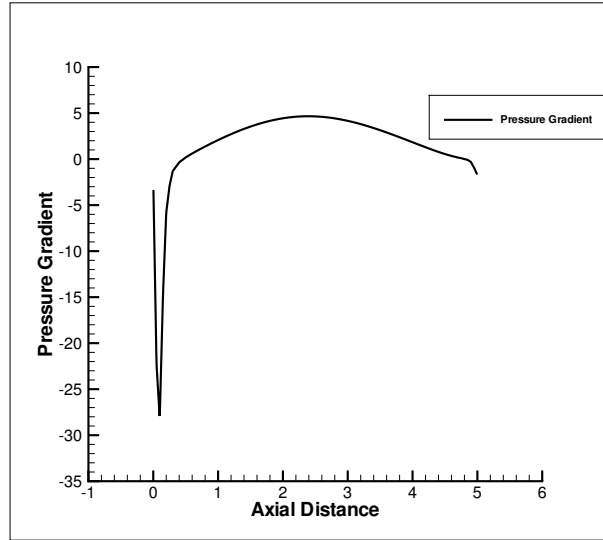


Figure 7.13: Pressure Gradient variation along the axial length of the domain

7.3 Laminar Coaxial Jets

We shall study the effect of different velocity ratio (V_r) for the control problem, as given in section 7.2 for Reynold Number 10. Same geometry and boundary conditions have been used, but for one notable exception that the outer inlet shall now be prescribed velocity. The results of the laminar coaxial flow control jet have been described in section 7.2.1. The boundary conditions for the coaxial cases is given as following :

- **Inlet Inner:** $w = 1; u = v = 0; \frac{\partial P}{\partial n} = 0; \phi = 1$
- **Inlet outer:** $w = V_r; u = v = 0; \frac{\partial P}{\partial n} = 0; \phi = 0$
- **Outlet :** $\frac{\partial u}{\partial n} = \frac{\partial v}{\partial n} = \frac{\partial w}{\partial n} = 0; P = 0; \frac{\partial \phi}{\partial n}$

- **Walls :** $u = v = w = 0$; $\frac{\partial P}{\partial n} = 0$; $\frac{\partial \phi}{\partial n}$ where 'n' is the direction normal to the plane of boundary.

The contours obtained for the steady state velocity and scalar are shown as below:

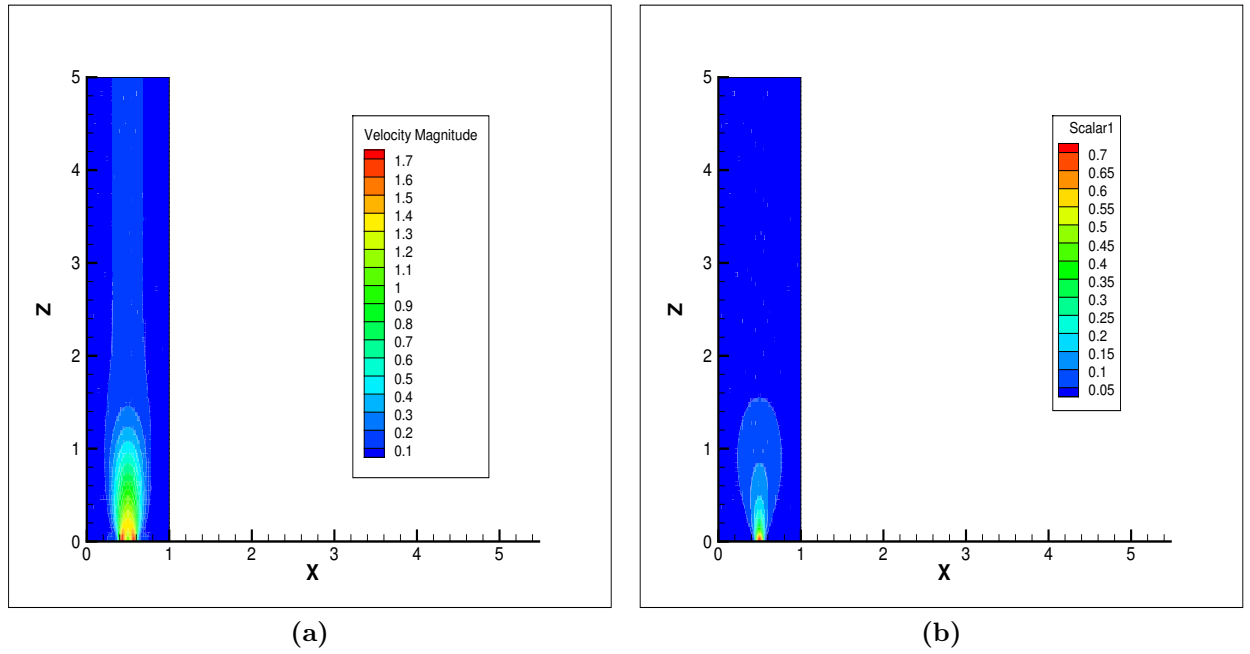


Figure 7.14: Contours of (a) Velocity Magnitude (b) Scalar for $V_r = 2$ at $Re = 10$

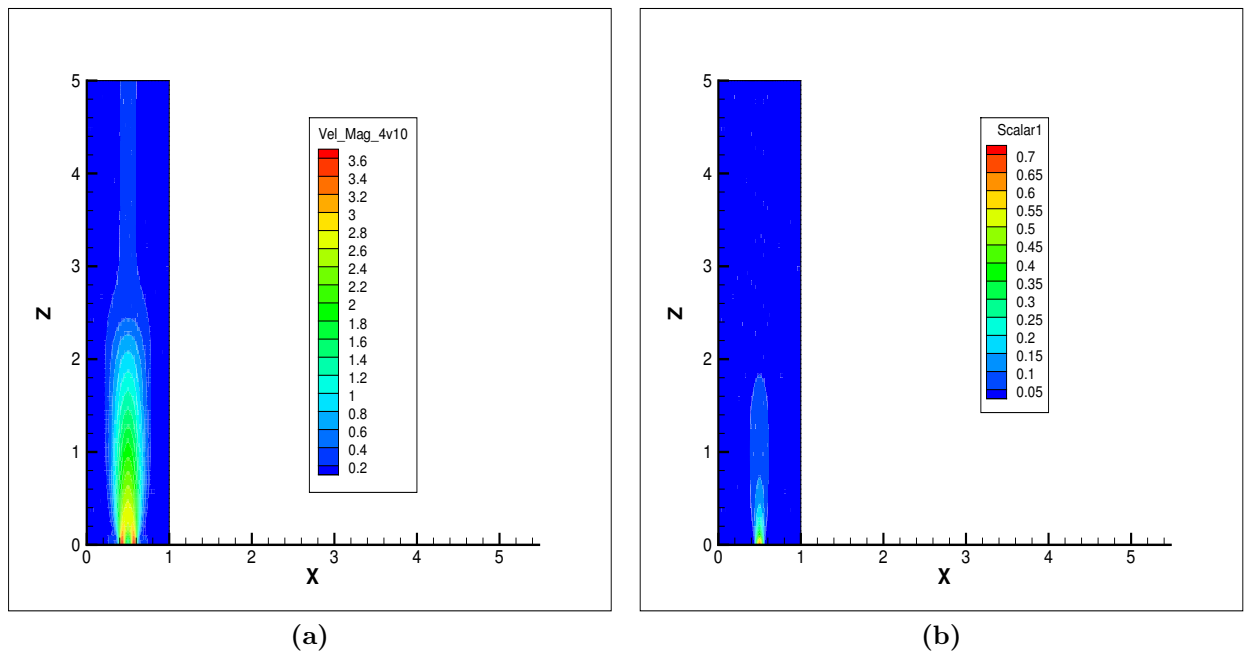


Figure 7.15: Contours of (a) Velocity Magnitude (b) Scalar for $V_r = 4$ at $Re = 10$

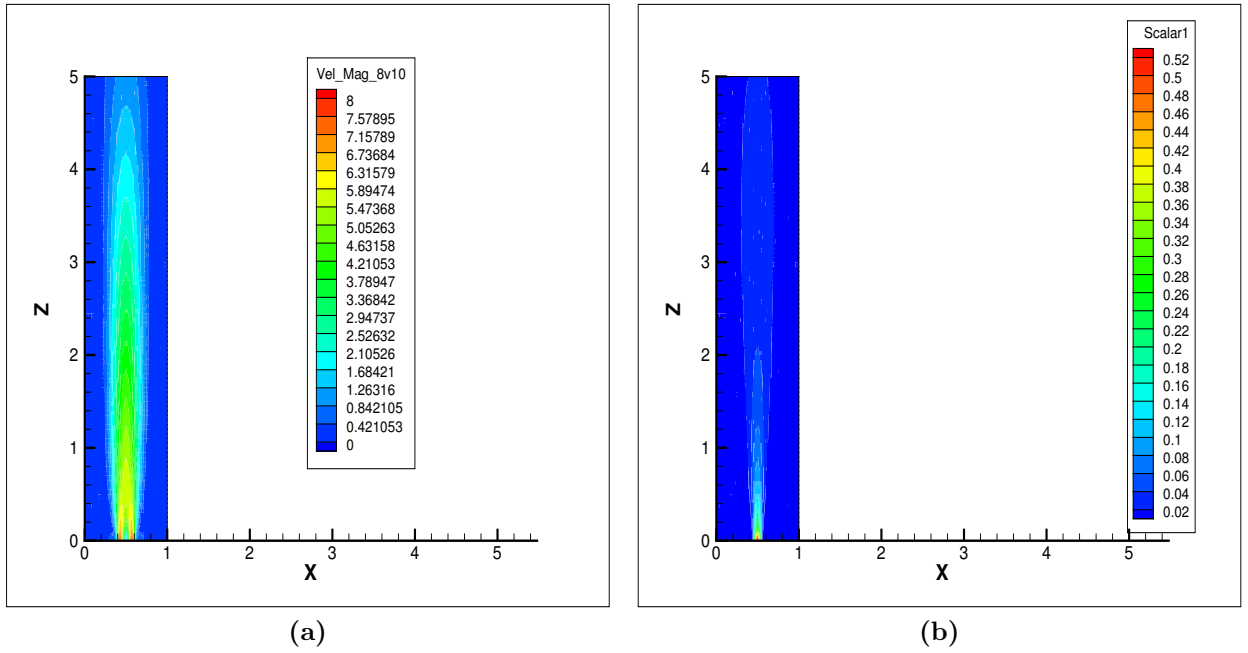


Figure 7.16: Contours of (a) Velocity Magnitude (b) Scalar for $V_r = 8$ at $Re = 10$

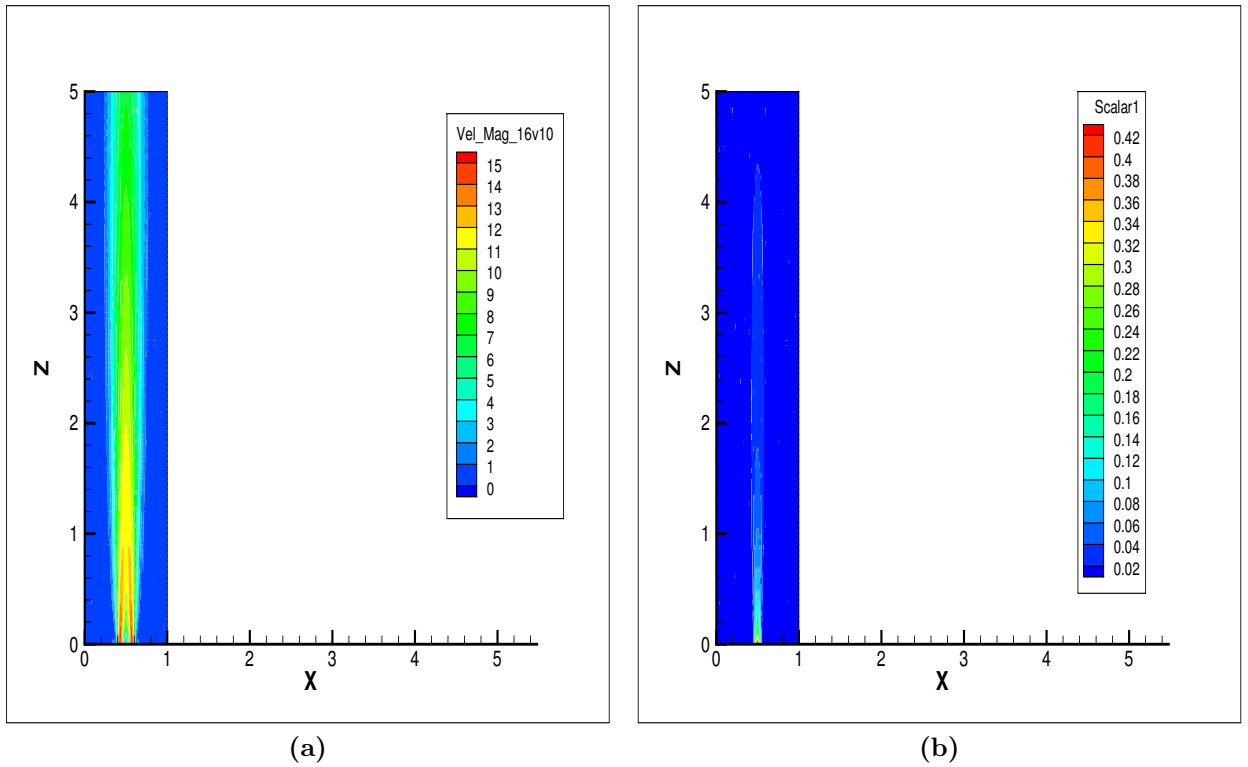


Figure 7.17: Contours of (a) Velocity Magnitude (b) Scalar for $V_r = 16$ at $Re = 10$

At large axial distances from the end of the inner potential core, the two jets merge in a single jet carrying the sum of the momenta injected in each jet, and this distance also depends upon the

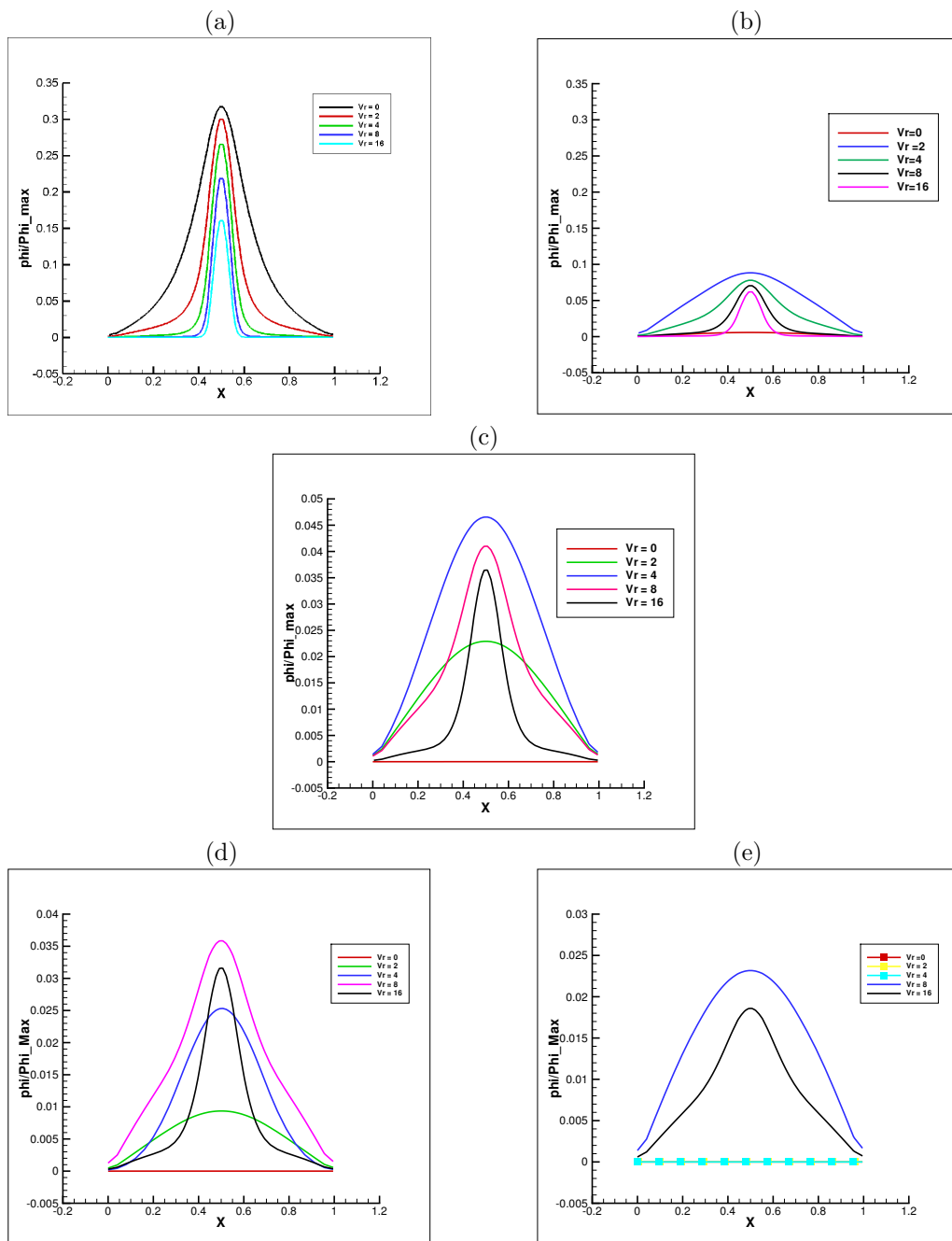


Figure 7.18: Variation of Scalar boundary layer thickness for different Velocity Ratio V_r at axial distance of (a) $Z=2D$ (b) $Z=10D$ (c) $Z=20D$ (d) $Z=30D$ (e) $Z=48D$

outer jet. The outer jet shears the inner jet, peeling and stretching the core of the jet, thus leading to decrease in boundary layer thickness but increases in its length. We can see that as the Reynolds Number of the annular jet increases, the momentum of the inner jet decreases and so does the boundary layer thickness. The scalar region becomes thinner with increase in Reynolds number. Before diffusing in the domain, the width of boundary layer increases like a plume as represented in the contours. The scalar magnitude for all the velocity ratio has been plotted for near wake for a distance of 2D and 10D and in the far wake at a distance of 20D, 30D and 48D as in figure 7.18 (a), (b), (c), (d) and (e) respectively.

As we can see from the figure 7.18 (a), the boundary layer thickness is maximum for least value of V_r . But as we move from near wake towards far wake, the boundary layer becomes zero for velocity ratio in increasing order. For example by $Z = 10D$, the boundary layer thickness becomes zero for $V_r = 0$, where as the stretching continues for other velocity ratios. At $Z = 20D$, the plume like structure of scalar magnitude exists for $V_r = 4$, due to which the scalar magnitude has decreased for the same, but the boundary layer thickness is at its maximum, before diffusing into the quiescent fluid. As we go further downstream, we see the jets keep diffusing with increasing velocity ratio. At just near the outlet, the plume like structure is formed for $V_r = 8$, due to which its boundary layer thickness is at maximum, which shall decrease and then diffuse in quiescent fluid. The trend is assumed to be continued in a larger domain for $V_r = 8$ and 16 and scalar diffusion occurs first for $V_r = 8$ and then for $V_r = 16$.

7.3.1 Effect of Injector inside the domain

To study the effect of surrounding wall near the inlet on the flow an injector inside the computational domain, as shown in figure 7.19. The injector was placed at 5D axial distance from the previous location of the inlet. Same boundary conditions as given in section 7.2.1 were used for this problem.

The velocity and scalar contours have been shown for RE 10 and RE 100 in figures 7.20 and 7.21 respectively.

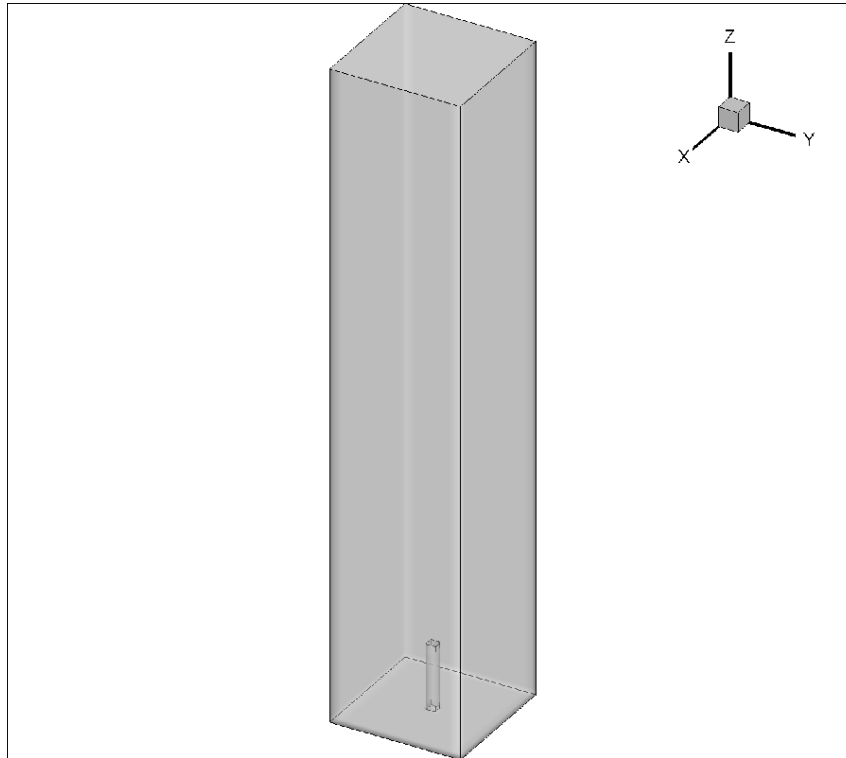


Figure 7.19: 3 Dimensional Domain with injector inlet at an axial distance of 5D from previous inlet

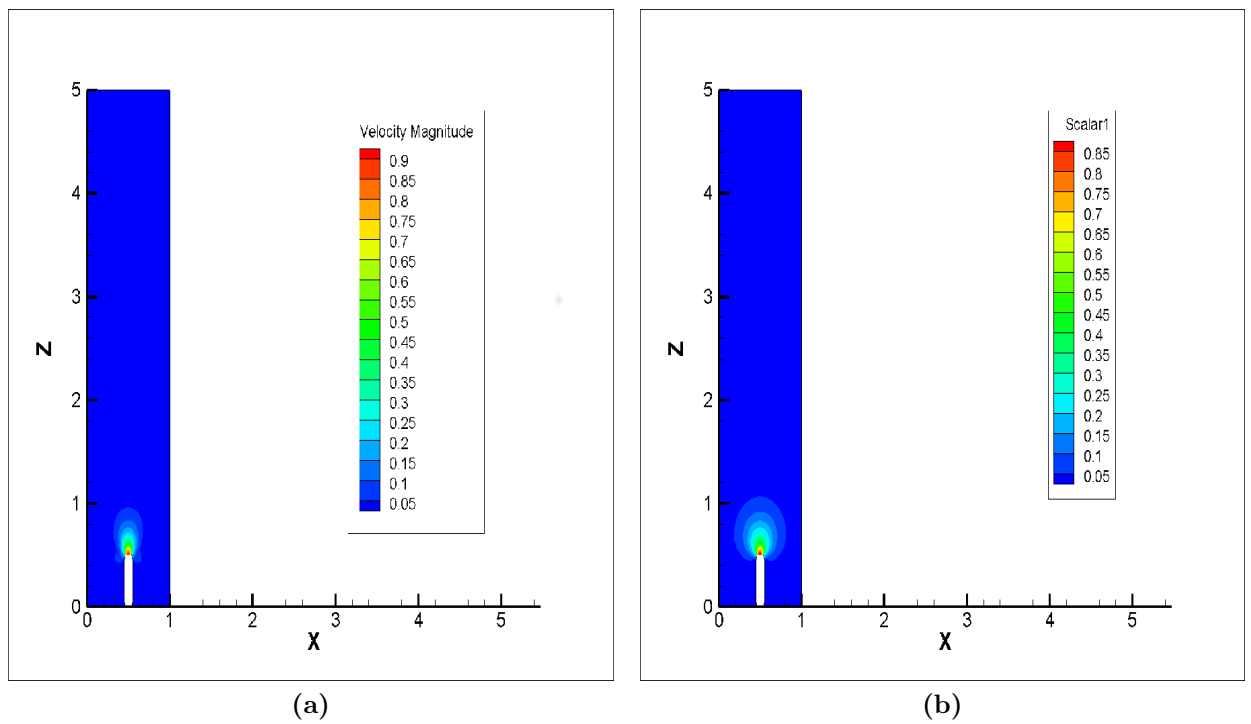


Figure 7.20: Contours of (a) Velocity Magnitude (b) Scalar for $Re = 10$

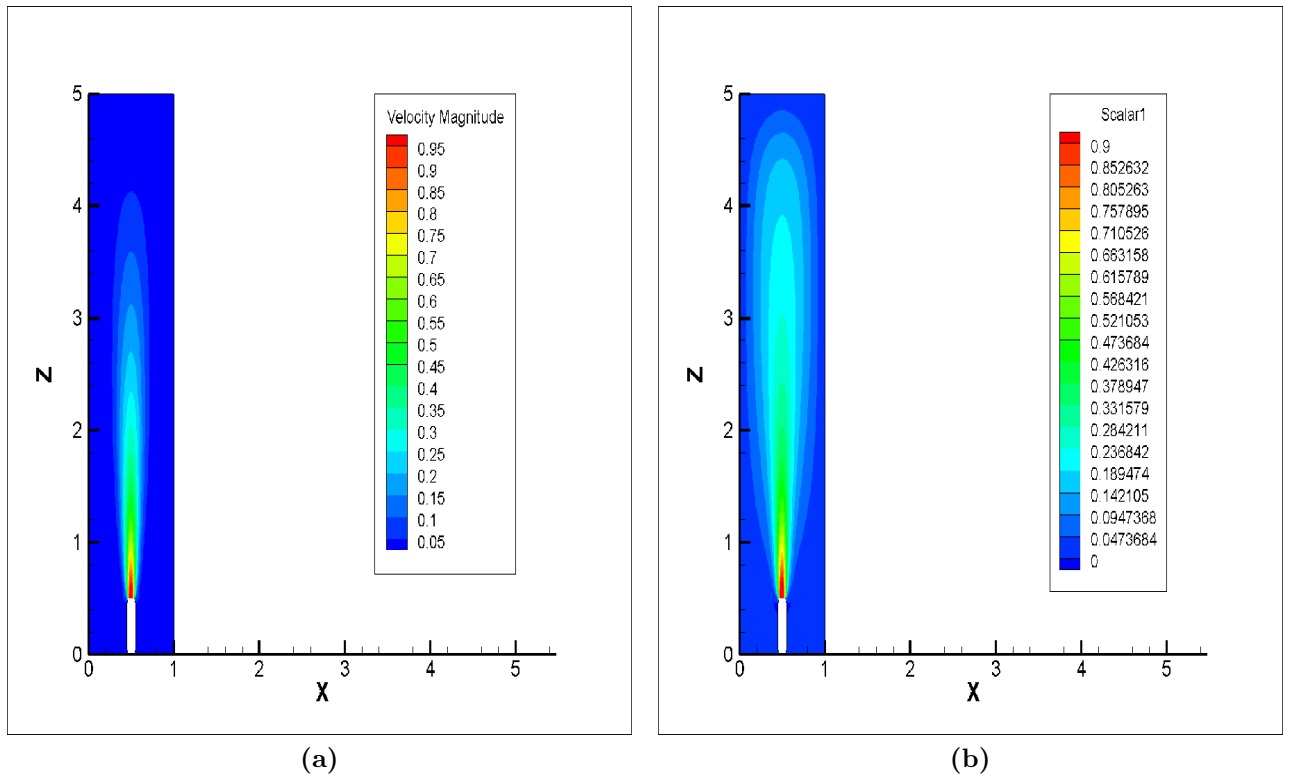


Figure 7.21: Contours of (a) Velocity Magnitude (b) Scalar for at $Re = 100$ with Injector in domain

The velocity and scalar have been compared in the figures given below:

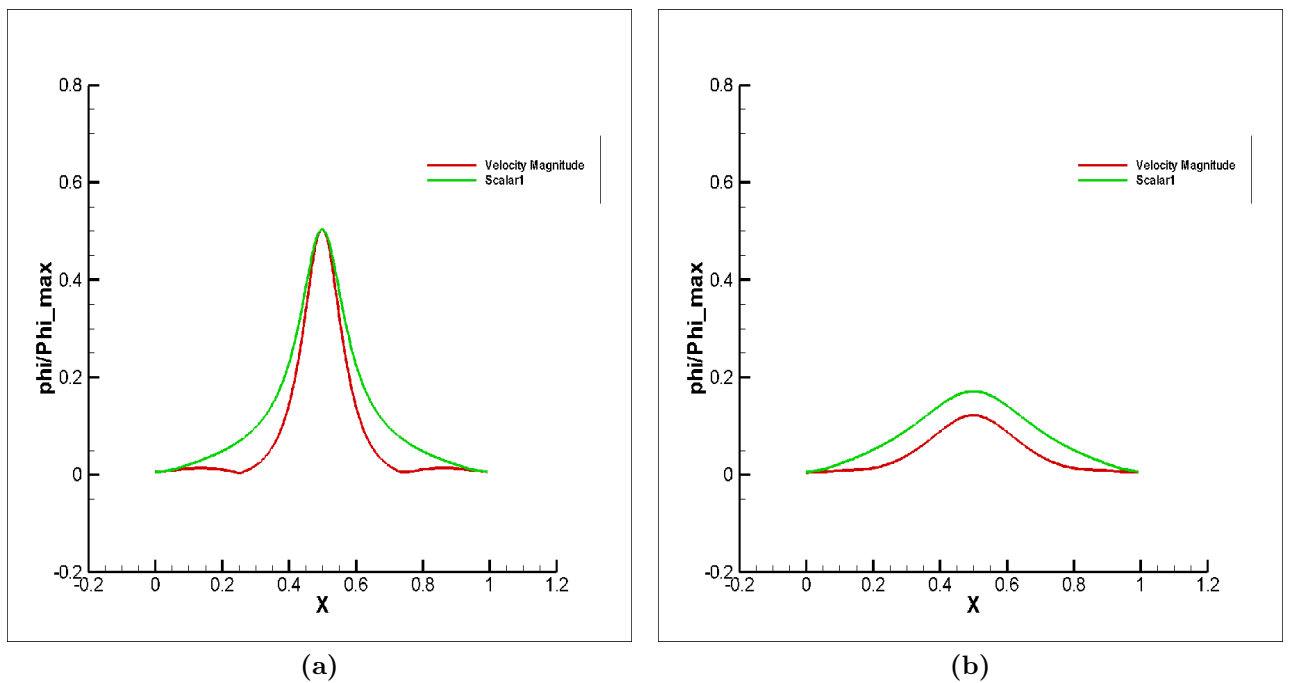


Figure 7.22: Comparison of Velocity and Scalar on $Y = 5D$ midplane at (a) $Z = 2D$ (b) $Z = 4D$ $Re = 10$

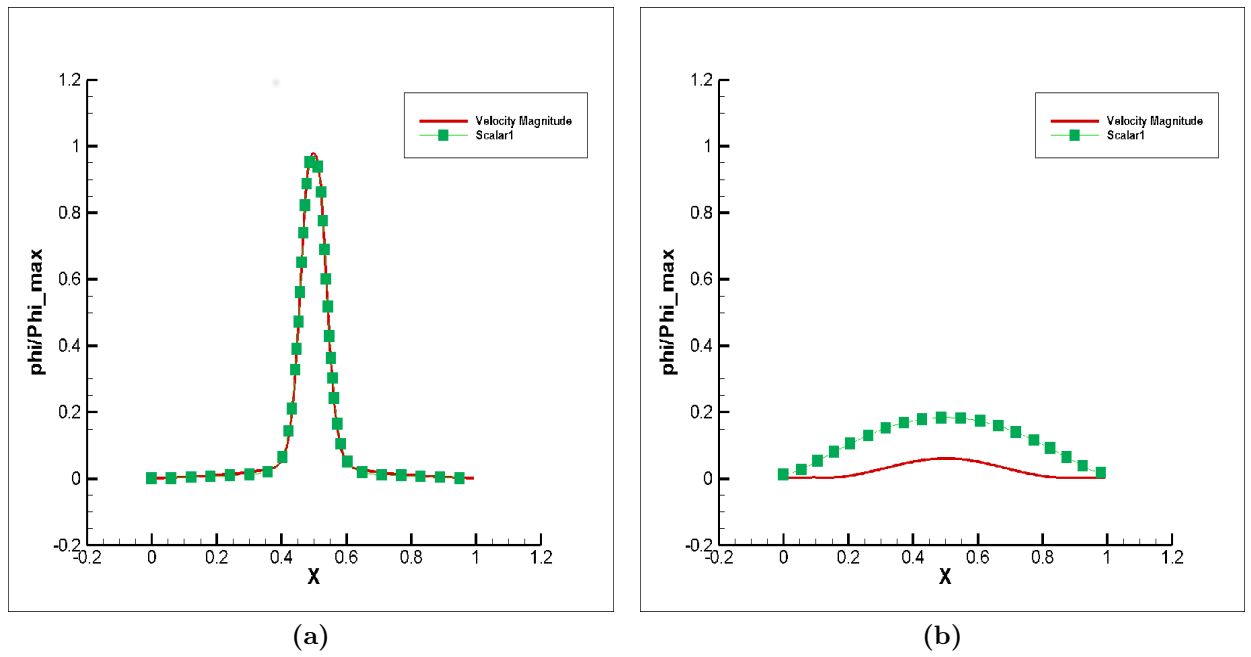


Figure 7.23: Comparison of Velocity and Scalar on $Y = 5D$ midplane at (a) $Z = 2D$ (b) $Z = 40D$ $Re = 100$

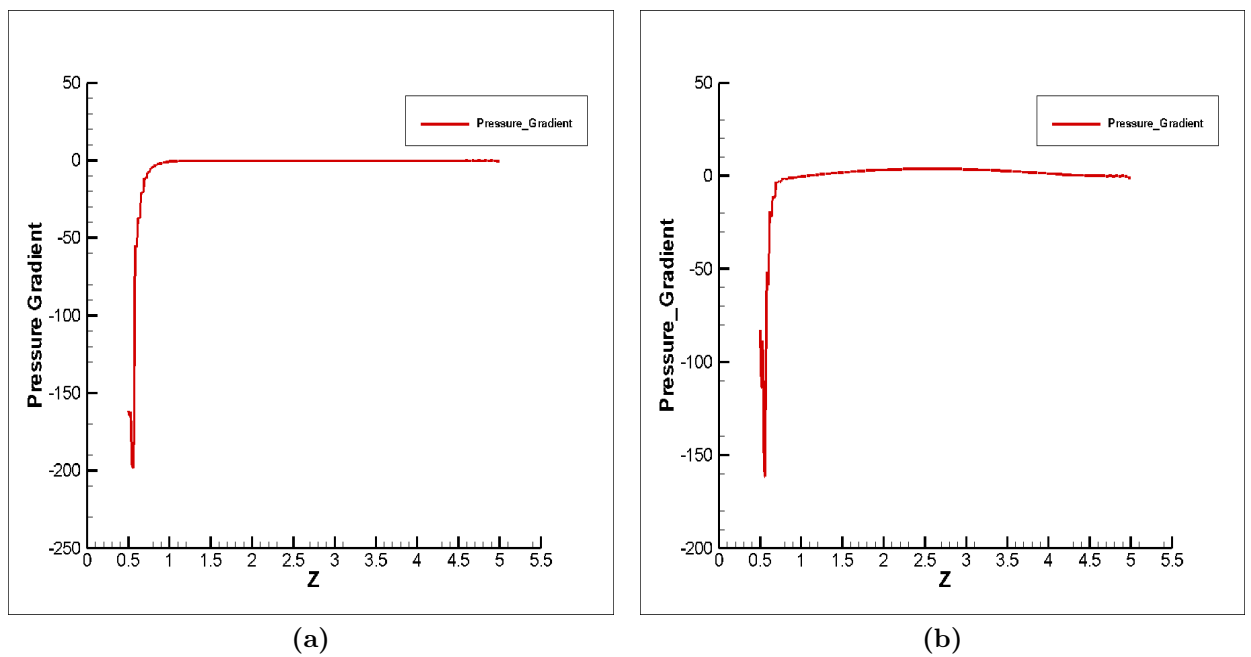


Figure 7.24: Pressure Gradient of (a) $Re = 10$ (b) $Re = 100$ along axial direction

As we can see from the figures, due to existence of pressure gradient at the inlet, the diffusion layer and the velocity boundary layer don't coincide, and hence we obtain results similar to previous case.

7.4 Turbulent Coaxial Jets

Turbulent jet was simulated with coaxial inlets, with Velocity ratio $V_r = 2$ and diameter ratio also of 2. The Reynolds number of the core jet is taken to be 11000. The scalar is also injected from the same inlet, with boundary conditions as given below:

- **Inlet Inner:** $w = 1; u = v = 0; \frac{\partial P}{\partial n} = 0 \phi = 1$
- **Inlet outer :** $w = V_r; u = v = 0; \frac{\partial P}{\partial n} = 0 \phi = 0$
- **Outlet :** $\frac{\partial u}{\partial n} = \frac{\partial v}{\partial n} = \frac{\partial w}{\partial n} = 0 P = 0 \frac{\partial \phi}{\partial n} = 0$
- **Bottom Symmetry :** $\frac{\partial u}{\partial n} = \frac{\partial w}{\partial n} = v = 0 P = 0 \frac{\partial \phi}{\partial n} = 0$
- **Walls :** $u = v = w = 0; \frac{\partial P}{\partial n} = 0 \frac{\partial \phi}{\partial n} = 0$ where 'n' is the direction normal to the plane of boundary.

The domain used for the simulation is given as following:

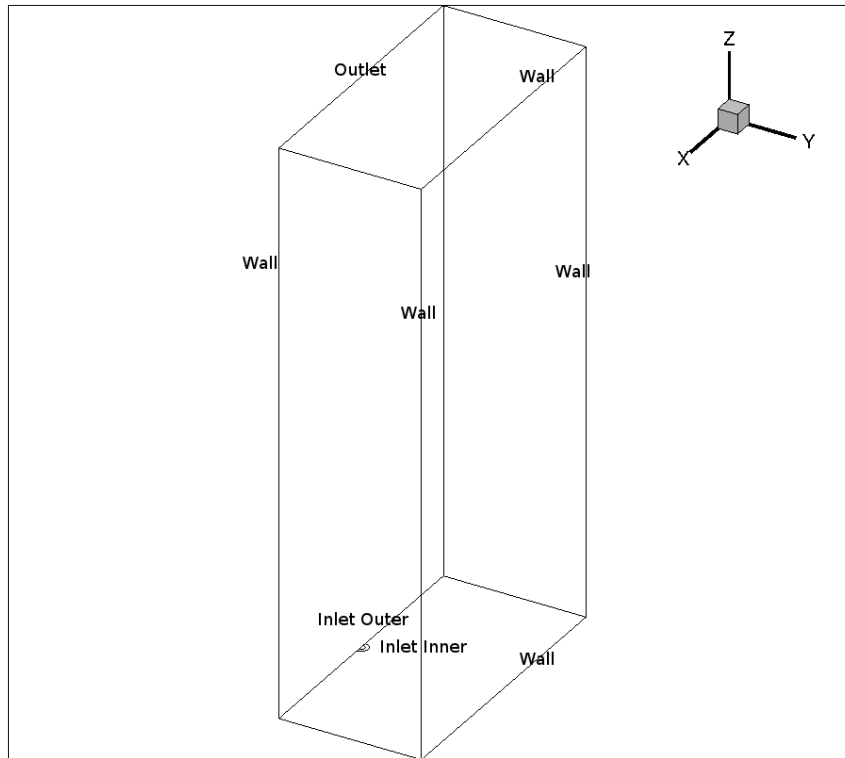


Figure 7.25: 3 Dimensional Domain for simulation of Turbulent coaxial Jet

The size of the domain is $40D \times 20D \times 80D$, with the inner inlet of D and outer inlet of $2D$. Only half of the domain has been simulated to lower the computational cost of the simulation. The number of cells for which the simulation was carried out was 3.3 million unstructured hexahedral cells. Following results have been obtained for flow time of 4 seconds.

We see that the jet is being stretched by the fast moving outer jet, and the instabilities have started to set in. Vortices are being generated and dissipated due to the turbulent nature of the flow.

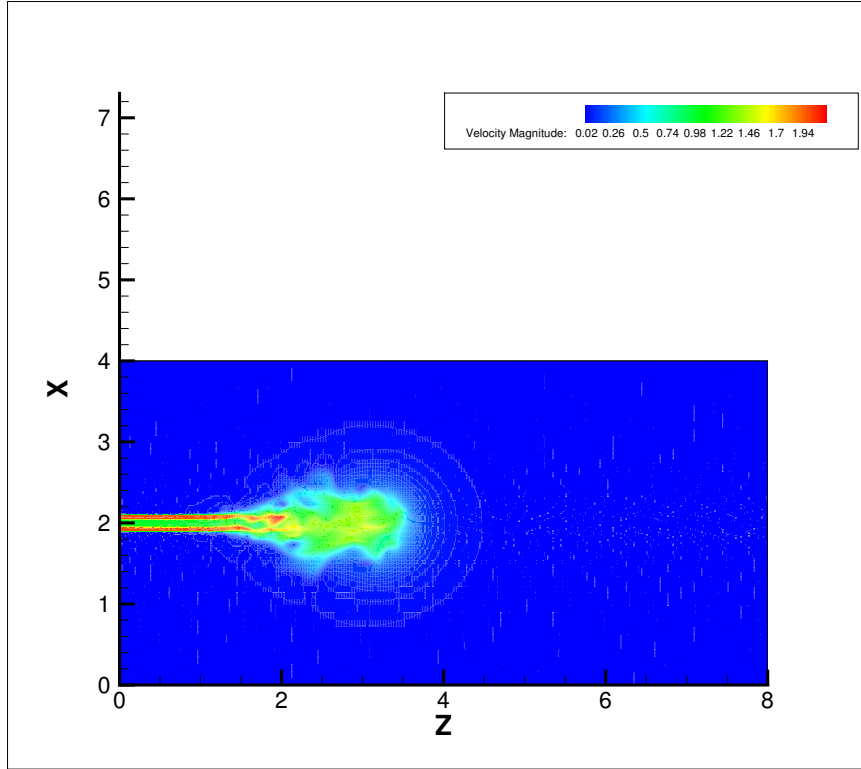


Figure 7.26: Velocity contours for Coaxial jet at $Y = 0$ plane

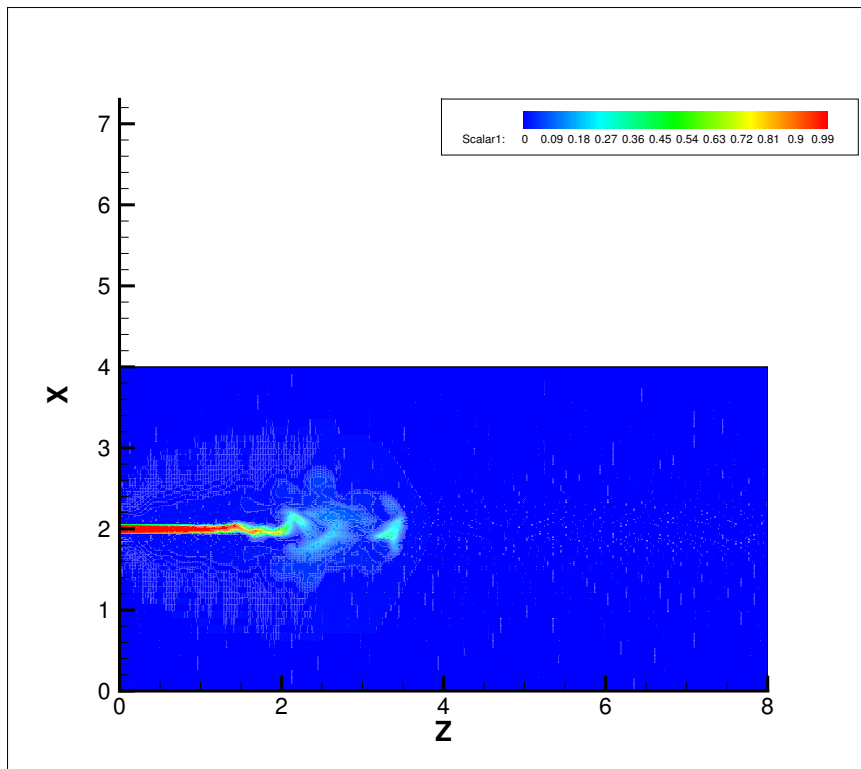


Figure 7.27: Scalar contours from inner inlet at $Y = 0$ plane

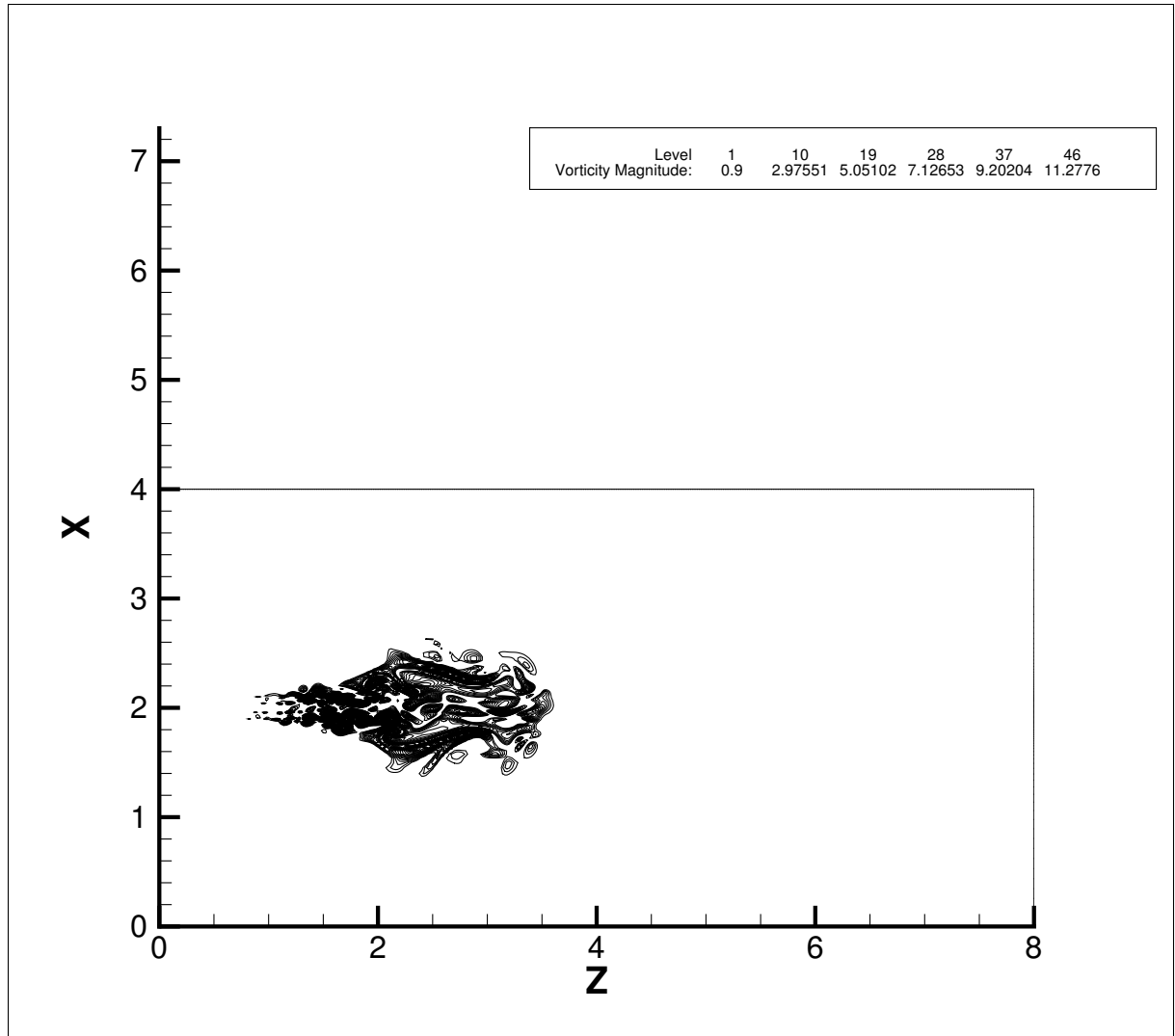


Figure 7.28: vorticity Magnitude contours and line contours at $Y = 0$ plane

Appendix A

The Coefficient Framework for the Unstructured Solver

A.1 For Diffusive Flux

A.1.1 Basic Expression For Diffusive Flux

The Basic expression for Diffusive Flux at a cell P surrounded by Neighbor Cells (denoted by subscript “n”) and having interior faces (denoted by subscript “f”) and faces on the boundary (denoted by subscript “b”) is

$$F_{diff} = \sum_f F_{d\phi_f} + \sum_b F_{d\phi_b} \quad (\text{A.1})$$

where

$$F_{d\phi_f} = -\Gamma_f(\nabla\phi_f \cdot \mathbf{S}_f) \quad (\text{A.2})$$

$$F_{d\phi_b} = -\Gamma_b(\nabla\phi_b \cdot \mathbf{S}_b) \quad (\text{A.3})$$

Where Γ_b and Γ_f is the volume interpolate value of the diffusion coefficient at that boundary and interior face respectively. The Face area vector is represented as \mathbf{S}_f . Its magnitude is given be A_f . Similarly for boundary face we have S_b and A_b .

Now, if $\hat{\mathbf{n}}_f$ be the unit vector normal to Face “f” and $\hat{\mathbf{n}}_1$ be the unit vector joining the cell center of P with the neighboring cell center of P_n , r_{1f} being the corresponding magnitude, then we can define a new unit vector, \mathbf{n}'_{2f} as,

$$\hat{\mathbf{n}}_f = \hat{\mathbf{n}}_1 + \mathbf{n}'_{2f} \quad (\text{A.4})$$

Also,

$$(\nabla\phi)_f \cdot \hat{\mathbf{n}}_1 = \frac{\phi_n - \phi_P}{r_{1f}} \quad (\text{A.5})$$

where the subscript “n” denotes values at the neighboring cell.

But

$$\begin{aligned} (\nabla\phi)_f \cdot \mathbf{S}_f &= A_f (\nabla\phi)_f \cdot \hat{\mathbf{n}}_f \\ &= A_f (\nabla\phi)_f \cdot (\hat{\mathbf{n}}_1 + \mathbf{n}'_{2f}) \end{aligned}$$

So using Eq.(A.4) and Eq.(A.5) above we get,

$$(\nabla\phi)_f \cdot \mathbf{S}_f = A_f \left[\frac{\phi_n - \phi_P}{r_{1f}} + (\nabla\phi)_f \cdot \mathbf{n}'_{2f} \right] \quad (\text{A.6})$$

Now, setting

$$\beta_f = \frac{A_f}{r_{1f}} \quad (\text{A.7})$$

$$\mathbf{n}_{2f} = A_f \mathbf{n}'_{2f} \quad (\text{A.8})$$

we get

$$(\nabla\phi)_f \cdot \mathbf{S}_f = \beta_f (\phi_n - \phi_P) + (\nabla\phi)_f \cdot \mathbf{n}_{2f} \quad (\text{A.9})$$

For a boundary face, Face Normal Formulation (Section 3.5.2) is being used. In this discretization scheme the boundary values are stored at the point where the perpendicular from parent cell centroid meets the boundary face. This eliminates the need for a cross diffusion term. So we get,

$$(\nabla\phi)_b \cdot \mathbf{S}_b = \beta_b (\phi_b - \phi_P) \quad (\text{A.10})$$

where,

$$\beta_b = \frac{A_b}{r_{1b} (\hat{\mathbf{n}}_1 \cdot \hat{\mathbf{n}}_b)} \quad (\text{A.11})$$

A.1.2 Evaluating the Diffusion Coefficients

In this section we shall evaluate $\sum_f F_{d\phi_f}$ and $\sum_b F_{d\phi_b}$, and flesh out coefficients that can be stored a priori. Since the faces are interior ones, we can write for each face “f”

$$\vartheta_n = \frac{V_n}{V_n + V_P} \quad (\text{A.12})$$

$$\vartheta_P = \frac{V_P}{V_n + V_P} \quad (\text{A.13})$$

$$(\nabla\phi)_f = \vartheta_n(\nabla\phi)_P + \vartheta_P(\nabla\phi)_n \quad (\text{A.14})$$

We have,

$$(\nabla\phi)_P = \frac{1}{V_P} \left\{ \begin{array}{l} \sum_f \phi_f S_{fx} + \sum_b \phi_b S_{bx} \\ \sum_f \phi_f S_{fy} + \sum_b \phi_b S_{by} \\ \sum_f \phi_f S_{fz} + \sum_b \phi_b S_{bz} \end{array} \right\} \quad (\text{A.15})$$

$$(\nabla\phi)_n = \frac{1}{V_n} \left\{ \begin{array}{l} \sum_{f_n} \phi_f S_{fx} + \sum_{b_n} \phi_{b_n} S_{b_n x} \\ \sum_{f_n} \phi_f S_{fy} + \sum_{b_n} \phi_{b_n} S_{b_n y} \\ \sum_{f_n} \phi_f S_{fz} + \sum_{b_n} \phi_{b_n} S_{b_n z} \end{array} \right\} \quad (\text{A.16})$$

But,

$$\phi_f = \vartheta_n \phi_P + \vartheta_P \phi_n \quad (\text{A.17})$$

So,

$$\frac{1}{V_P} \sum_f \phi_f S_{fx} = \frac{1}{V_P} \phi_P \sum_f (\vartheta_n S_{fx}) + \frac{1}{V_P} \sum_f (\vartheta_P S_{fx} \phi_n) \quad (\text{A.18})$$

$$\frac{1}{V_P} \sum_f \phi_f S_{fy} = \frac{1}{V_P} \phi_P \sum_f (\vartheta_n S_{fy}) + \frac{1}{V_P} \sum_f (\vartheta_P S_{fy} \phi_n) \quad (\text{A.19})$$

$$\frac{1}{V_P} \sum_f \phi_f S_{fz} = \frac{1}{V_P} \phi_P \sum_f (\vartheta_n S_{fz}) + \frac{1}{V_P} \sum_f (\vartheta_P S_{fz} \phi_n) \quad (\text{A.20})$$

Hence setting

$$B_{xP} = \frac{1}{V_P} \sum_f \vartheta_n S_{fx} \quad (\text{A.21})$$

$$B_{yP} = \frac{1}{V_P} \sum_f \vartheta_n S_{fy} \quad (\text{A.22})$$

$$B_{zP} = \frac{1}{V_P} \sum_f \vartheta_n S_{fz} \quad (\text{A.23})$$

for each cell and setting

$$D_{xP_f} = \frac{1}{V_P} \vartheta_P S_{fx} \quad (\text{A.24})$$

$$D_{yP_f} = \frac{1}{V_P} \vartheta_P S_{fy} \quad (\text{A.25})$$

$$D_{zP_f} = \frac{1}{V_P} \vartheta_P S_{fz} \quad (\text{A.26})$$

for each face of a cell we can express Eq. (A.15) as,

$$(\nabla\phi)_P = \left\{ \begin{array}{l} B_{xP}\phi_P + \sum_f D_{xP_f}\phi_n + \sum_b \frac{S_{bx}}{V_P}\phi_b \\ B_{yP}\phi_P + \sum_f D_{yP_f}\phi_n + \sum_b \frac{S_{by}}{V_P}\phi_b \\ B_{zP}\phi_P + \sum_f D_{zP_f}\phi_n + \sum_b \frac{S_{bz}}{V_P}\phi_b \end{array} \right\} \quad (\text{A.27})$$

Similarly the gradient of Neighbor Cells can be expressed as

$$(\nabla\phi)_n = \left\{ \begin{array}{l} B_{xn}\phi_n + \sum_{f_n} D_{xn_{f_n}}\phi_{n_n} + \sum_{b_n} \frac{S_{bnx}}{V_n}\phi_{b_n} \\ B_{yn}\phi_n + \sum_{f_n} D_{yn_{f_n}}\phi_{n_n} + \sum_{b_n} \frac{S_{bny}}{V_n}\phi_{b_n} \\ B_{zn}\phi_n + \sum_{f_n} D_{zn_{f_n}}\phi_{n_n} + \sum_{b_n} \frac{S_{bnz}}{V_n}\phi_{b_n} \end{array} \right\} \quad (\text{A.28})$$

So since

$$(\nabla\phi)_f \cdot \mathbf{n}_{2_f} = (\vartheta_n(\nabla\phi)_P + \vartheta_P(\nabla\phi)_n) \cdot \mathbf{n}_{2_f} \quad (\text{A.29})$$

We can put Eq. (A.27) and Eq. (A.28) above to give

$$\begin{aligned} (\nabla\phi)_f \cdot \mathbf{n}_{2_f} &= \phi_P \vartheta_n (B_{xP}n_{2_{fx}} + B_{yP}n_{2_{fy}} + B_{zP}n_{2_{fz}}) \\ &\quad + \phi_n \vartheta_P (B_{xn}n_{2_{fx}} + B_{yn}n_{2_{fy}} + B_{zn}n_{2_{fz}}) \\ &\quad + \vartheta_n n_{2_{fx}} \sum_f D_{xP_f}\phi_n + \vartheta_n n_{2_{fy}} \sum_f D_{yP_f}\phi_n \\ &\quad + \vartheta_n n_{2_{fz}} \sum_f D_{zP_f}\phi_n + \vartheta_P n_{2_{fx}} \sum_{f_n} D_{xn_{f_n}}\phi_{n_n} \\ &\quad + \vartheta_P n_{2_{fy}} \sum_{f_n} D_{yn_{f_n}}\phi_{n_n} + \vartheta_P n_{2_{fz}} \sum_{f_n} D_{zn_{f_n}}\phi_{n_n} \\ &\quad + \vartheta_n \sum_b \frac{S_{bx}}{V_P}\phi_b n_{2_{fx}} + \vartheta_n \sum_b \frac{S_{by}}{V_P}\phi_b n_{2_{fy}} \\ &\quad + \vartheta_n \sum_b \frac{S_{bz}}{V_P}\phi_b n_{2_{fz}} + \vartheta_P \sum_{b_n} \frac{S_{bnx}}{V_n}\phi_{b_n} n_{2_{fx}} \\ &\quad + \vartheta_P \sum_{b_n} \frac{S_{bny}}{V_n}\phi_{b_n} n_{2_{fy}} + \vartheta_P \sum_{b_n} \frac{S_{bnz}}{V_n}\phi_{b_n} n_{2_{fz}} \end{aligned} \quad (\text{A.30})$$

where x_{n_n} represents the value of x at the cell adjacent to the neighbor cell i.e. “neighbor-of-neighbor” cell. From Eq. (A.9) it is clear that

$$\sum_f (\nabla\phi)_f \cdot \mathbf{S}_f = \sum_f [\beta_f(\phi_{n_f} - \phi_P)] + \sum_f (\nabla\phi)_f \cdot \mathbf{n}_{2f} \quad (\text{A.31})$$

Putting Eq. (A.30) into the above equation and rearranging the terms carefully we get,

$$\begin{aligned} \sum_f (\nabla\phi)_f \cdot \mathbf{S}_f &= \phi_P \sum_f G_{C_f} + \sum_f G_n \phi_n + \sum_{b_P} G_b \phi_b \\ &+ \sum_f \left[\sum_{f_n} G_{n_n} \phi_{n_n} + \sum_{b_n} G_{b_n} \phi_{b_n} \right] \end{aligned} \quad (\text{A.32})$$

where the coefficients are,

$$G_{C_f} = -\beta_f + [\vartheta_n(B_{xP}n_{2fx} + B_{yP}n_{2fy} + B_{zP}n_{2fz})] \quad (\text{A.33})$$

$$\begin{aligned} G_n &= \beta_f + \vartheta_P(B_{xn}n_{2fx} + B_{yn}n_{2fy} + B_{zn}n_{2fz}) + D_{xP_f} \sum_f (\vartheta_{n_f}n_{2fx}) \\ &+ D_{yP_f} \sum_f (\vartheta_{n_f}n_{2fy}) + D_{zP_f} \sum_f (\vartheta_{n_f}n_{2fz}) \end{aligned} \quad (\text{A.34})$$

$$G_b = \frac{S_{b_x}}{V_P} \sum_f \vartheta_n n_{2fx} + \frac{S_{b_y}}{V_P} \sum_f \vartheta_n n_{2fy} + \frac{S_{b_z}}{V_P} \sum_f \vartheta_n n_{2fz} \quad (\text{A.35})$$

$$G_{n_n} = D_{xn_{f_n}} \vartheta_P n_{2fx} + D_{yn_{f_n}} \vartheta_P n_{2fy} + D_{zn_{f_n}} \vartheta_P n_{2fz} \quad (\text{A.36})$$

$$G_{b_n} = \frac{S_{b_{nx}}}{V_n} \vartheta_P n_{2fx} + \frac{S_{b_{ny}}}{V_n} \vartheta_P n_{2fy} + \frac{S_{b_{nz}}}{V_n} \vartheta_P n_{2fz} \quad (\text{A.37})$$

According to the Normal Point Formulation,

$$\sum_b (\nabla\phi)_b \cdot \mathbf{S}_b = \sum_b \phi_b \beta_b - \phi_P \sum_b \beta_b \quad (\text{A.38})$$

So,

$$F_{diff} = -\frac{1}{\rho} \left[\sum_f \Gamma_f (\nabla\phi)_f \cdot \mathbf{S}_f + \sum_b \Gamma_b (\nabla\phi)_b \cdot \mathbf{S}_b \right]$$

Thus, using Eq. (A.32) and Eq. (A.38) above, the final coefficient based formulation of the diffusion term becomes,

$$F_{diff} = -\frac{1}{\rho} \left[\phi_P \left(\sum_f \Gamma_f G_{C_f} + \sum_b \Gamma_b G_{C_b} \right) + \sum_f \Gamma_f G_n \phi_n + \sum_{b_P} \Gamma_b G_b \phi_b + \sum_f \Gamma_f \left(\sum_{f_n} G_{n_n} \phi_{n_n} + \sum_{b_n} G_{b_n} \phi_{b_n} \right) \right] \quad (\text{A.39})$$

where the coefficients are redefined as,

$$G_{C_f} = -\beta_f + [\vartheta_n (B_{xP} n_{2_{fx}} + B_{yP} n_{2_{fy}} + B_{zP} n_{2_{fz}})] \quad (\text{A.40})$$

$$G_{C_b} = -\beta_b \quad (\text{A.41})$$

$$G_n = \beta_f + \vartheta_P (B_{xn} n_{2_{fx}} + B_{yn} n_{2_{fy}} + B_{zn} n_{2_{fz}}) + D_{xP_f} \sum_f (\vartheta_{n_f} n_{2_{fx}}) + D_{yP_f} \sum_f (\vartheta_{n_f} n_{2_{fy}}) + D_{zP_f} \sum_f (\vartheta_{n_f} n_{2_{fz}}) \quad (\text{A.42})$$

$$G_b = \beta_b + \frac{S_{b_x}}{V_P} \sum_f \vartheta_n n_{2_{fx}} + \frac{S_{b_y}}{V_P} \sum_f \vartheta_n n_{2_{fy}} + \frac{S_{b_z}}{V_P} \sum_f \vartheta_n n_{2_{fz}} \quad (\text{A.43})$$

$$G_{n_n} = D_{xn_{fn}} \vartheta_P n_{2_{fx}} + D_{yn_{fn}} \vartheta_P n_{2_{fy}} + D_{zn_{fn}} \vartheta_P n_{2_{fz}} \quad (\text{A.44})$$

$$G_{b_n} = \frac{S_{b_{nx}}}{V_n} \vartheta_P n_{2_{fx}} + \frac{S_{b_{ny}}}{V_n} \vartheta_P n_{2_{fy}} + \frac{S_{b_{nz}}}{V_n} \vartheta_P n_{2_{fz}} \quad (\text{A.45})$$

If Γ is not a function of space, then it can come out of the summations of Eq.(A.39)

A.2 For Convective Flux

We know that (Eq.(3.12)),

$$F_f \phi^{l+1}_f = [F_f \phi^{l+1}_f]_{1_{st\ upwind}} + \gamma \left[(F_f \phi_f)^l_{CD} - (F_f \phi_f)^l_{1_{st\ upwind}} \right] = \phi^{l+1}_P [[F_f, 0]] - \phi^{l+1}_n [[-F_f, 0]] + \gamma [F_f (\vartheta_n \phi^l_P + \vartheta_P \phi^l_n)] - \gamma \phi^l_P [[F_f, 0]] + \gamma \phi^l_n [[-F_f, 0]] \quad (\text{A.46})$$

where “ $(l+1)$ ” denotes values in the current iteration, “ l ” denotes values at the previous iteration and γ is the ratio of central difference to first upwind scheme specified by the user. The convective term is given by,

$$F_{conv} = \sum_f F_f \phi^{l+1}_f + \sum_b F_b \phi^{l+1}_b \quad (\text{A.47})$$

So using Eq.(A.46) in the above equation and rearranging the terms we get,

$$F_{conv} = H_1 \phi^{l+1}_P + H_3 \phi^l_P - \sum_f H_2 \phi^{l+1}_n + \sum_f H_4 \phi^l_n + \sum_b F_b \phi_b \quad (\text{A.48})$$

where,

$$H_1 = \sum_f |F_f, 0| \quad (\text{A.49})$$

$$H_2 = |-F_f, 0| \quad (\text{A.50})$$

$$H_3 = \sum_f (\gamma \vartheta_n F_f) - \sum_f \gamma |F_f, 0| \quad (\text{A.51})$$

$$H_4 = \gamma \vartheta_P F_f + \gamma |-F_f, 0| \quad (\text{A.52})$$

A.3 Final Form and Implementation

The Convective Diffusion equation is of the form,

$$\frac{V_P}{\Delta t} (\phi^{l+1}_P - \phi^n_P) + \sum_f F_f \phi^{l+1}_f + \frac{1}{\rho_P} F_{diff}^{l+1} = S^l \quad (\text{A.53})$$

Using Eq.(A.39) and Eq.(A.48) above we get the final expression as,

$$\begin{aligned} \phi^{l+1}_P \left[\frac{V_P}{\Delta t} + H_1 - \frac{1}{\rho_P} \left(\sum_f \Gamma_f G_{C_f} + \sum_b \Gamma_b G_{C_b} \right) \right] + \sum_f \phi^{l+1}_n \left[-H_2 - \frac{\Gamma_f}{\rho_P} (G_n) \right] \\ + \sum_b \phi_b \left[F_b - \frac{\Gamma_b}{\rho_P} (G_b) \right] + \sum_f \left[-\frac{\Gamma_f}{\rho_P} \left(\sum_{n_n} G_{n_n} \phi^{l+1}_{n_n} + \sum_{b_n} G_{b_n} \phi^{l+1}_{b_n} \right) \right] \\ = S^l - H_3 \phi^l_P - \sum_f H_4 \phi^l_n + \frac{V_P}{\Delta t} \phi^n_P \quad (\text{A.54}) \end{aligned}$$

A.4 Pressure Equation

The pressure poisson equation is given by,

$$\sum_f (\nabla p_f^*) \cdot \mathbf{S}_f = \frac{\rho}{\Delta t} \sum_f F_f^* \quad (\text{A.55})$$

Clearly, the left hand side of the equation is identical to the diffusion term and hence will have identical coefficients. So in the coefficient framework, the pressure equation may be written as,

$$p_P \left(\sum_f G_{C_f} + \sum_b G_{C_b} \right) + \sum_f (p_n G_n) + \sum_b (p_{b_P} G_b) + \sum_f \left[\left(\sum_{f_n} G_{n_n} p_{n_n} + \sum_{b_n} G_{b_n} p_{b_n} \right) \right] = \frac{\rho}{\Delta t} \sum_f F_f^* \quad (\text{A.56})$$

A.4.1 Modifications in the algorithm

Modifications made in the solution algorithm mainly pertain to when the coefficients are to be evaluated. The diffusion related coefficients (denoted here by the symbol “G”) depend on geometry information only and are calculated just once . In contrast the convection related coefficients (denoted here by the symbol “H”) depend on flux values and hence need to be evaluated once per time step for the Semi-Implicit Scheme.

References

- [1] A. Date. Solution of transport equations on unstructured meshes with cell-centered colocated variables. Part I: Discretization. *International journal of heat and mass transfer* 48, (2005) 1117–1127.
- [2] A. Dalal, V. Eswaran, and G. Biswas. A finite-volume method for Navier-Stokes equations on unstructured meshes. *Numerical Heat Transfer, Part B: Fundamentals* 54, (2008) 238–259.
- [3] N. Panchapakesan and J. Lumley. Turbulence measurements in axisymmetric jets of air and helium. Part 1. Air jet. *Journal of Fluid Mechanics* 246, (1993) 197–223.
- [4] K. Mahesh, G. Constantinescu, and P. Moin. A numerical method for large-eddy simulation in complex geometries. *Journal of Computational Physics* 197, (2004) 215–240.
- [5] J. Sebastian, N. Sivadasan, and R. Banerjee. GPU Accelerated Three Dimensional Unstructured Geometric Multigrid Solver. *Accepted in IEEE HPCS 2014* .
- [6] A. Corrigan, F. F. Camelli, R. Löhner, and J. Wallin. Running unstructured grid-based CFD solvers on modern graphics hardware. *International Journal for Numerical Methods in Fluids* 66, (2011) 221–229.
- [7] V. Asouti, X. Trompoukis, I. Kampolis, and K. Giannakoglou. Unsteady CFD computations using vertex-centered finite volumes for unstructured grids on Graphics Processing Units. *International Journal for Numerical Methods in Fluids* 67, (2011) 232–246.
- [8] D. Goddeke, S. H. Buijssen, H. Wobker, and S. Turek. GPU acceleration of an unmodified parallel finite element Navier-Stokes solver. In *High Performance Computing & Simulation, 2009. HPCS'09. International Conference on*. IEEE, 2009 12–21.
- [9] S. P. Vanka, A. F. Shinn, and K. C. Sahu. Computational fluid dynamics using graphics processing units: challenges and opportunities. In *ASME 2011 International Mechanical Engineering Congress and Exposition*. American Society of Mechanical Engineers, 2011 429–437.
- [10] A. F. Shinn and S. P. Vanka. Large eddy simulations of film-cooling flows with a micro-ramp vortex generator. *Journal of Turbomachinery* 135, (2013) 011,004.
- [11] A. Shinn, S. Vanka, and W. Hwu. Direct numerical simulation of turbulent flow in a square duct using a graphics processing unit (GPU). In *40th AIAA Fluid Dynamics Conference*. 2010 .

- [12] R. D. Falgout. An introduction to algebraic multigrid. *Computing in Science and Engineering* 8, (2006) 24–33.
- [13] P. Moin and S. V. Apte. Large-eddy simulation of realistic gas turbine combustors. *AIAA journal* 44, (2006) 698–708.
- [14] C. D. Pierce and P. Moin. Large eddy simulation of a confined coaxial jet with swirl and heat release. *AIAA paper* 2892.
- [15] A. Lefebvre. Atomization and sprays, volume 1040. CRC press, 1988.
- [16] H. Rehab, E. Villiermaux, and E. Hopfinger. Flow regimes of large-velocity-ratio coaxial jets. *Journal of Fluid Mechanics* 345, (1997) 357–381.
- [17] S. Patankar. Numerical heat transfer and fluid flow. CRC Press, 1980.
- [18] C. Rhie and W. Chow. Numerical study of the turbulent flow past an airfoil with trailing edge separation. *AIAA journal* 21, (1983) 1525–1532.
- [19] R. Kamakolanun. Parallel AMG Solver for Three Dimensional Unstructured Grids Using GPU. Master’s thesis, Indian Institute of Technology Hyderabad, Yeddumailaram, Medak 2014.
- [20] C. NVidia. C best practices guide. *NVIDIA, Santa Clara, CA* .
- [21] A. R. Brodtkorb, T. R. Hagen, and M. L. SæTra. GPU programming strategies and trends in GPU computing. *Journal of Parallel and Distributed Computing* .
- [22] CUDA C Best Practices Guide. Nvidia Corporation, May 2011.
- [23] W. Briggs, V. Henson, and S. McCormick. A Multigrid Tutorial. Miscellaneous Bks. Society for Industrial and Applied Mathematics, 2000.
- [24] K. Stüben. Algebraic multigrid (AMG): an introduction with applications. GMD-Forschungszentrum Informationstechnik, 1999.
- [25] H.-C. Hege and H. Stüben. Vectorization and parallelization of irregular problems via graph coloring. In Proceedings of the 5th international conference on Supercomputing. ACM, 1991 47–56.
- [26] R. Lonsdale. An algebraic multigrid solver for the NavierStokes equations on unstructured meshes. *International Journal of Numerical Methods for Heat & Fluid Flow* 3, (1993) 3–14.
- [27] H. C. Ku, R. S. Hirsh, and T. D. Taylor. A pseudospectral method for solution of the three-dimensional incompressible Navier-Stokes equations. *Journal of Computational Physics* 70, (1987) 439–462.
- [28] M. Breuer, J. Bernsdorf, T. Zeiser, and F. Durst. Accurate computations of the laminar flow past a square cylinder based on two different methods: lattice-Boltzmann and finite-volume. *International Journal of Heat and Fluid Flow* 21, (2000) 186–196.
- [29] J. Holman. Heat transfer, 1986. *Mc Gran–Hill Book Company, Soythern Methodist University*, .

[30] H. Schlichting, K. Gersten, and K. Gersten. *Boundary-layer theory*. Springer, 2000.