Development of a parallel CFD solver with application to arterial flows.

Kapoor, Amarpal Singh

Swansea University
Prifysgol Abertawe

DOCTOR OF PHILOSOPHY

Development of a Parallel CFD solver with application to arterial flows

---

Amarpal Singh Kapoor
MRes., B.E

Thesis submitted to Swansea University in fulfilment of the requirements for the Degree of Doctor of Philosophy

December, 2014

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

Declaration and statements

DECLARATION

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed ..... (candidate)

Date .....23 - 4 - 2015.....

STATEMENT 1

This thesis is the result of my own investigations, except where otherwise stated. Where correction services have been used, the extent and nature of the correction is clearly marked in a footnote(s).
Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed ..... (candidate)

Date .....23 - 4 - 15.....

STATEMENT 2

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ..... (candidate)

Date .....23 - 4 - 15.....

Acknowledgements

My life during the tenure of this PhD has been an experience packed with lessons that will continue to guide me throughout. These lessons were made possible by all the wonderful people who witnessed me from an external frame of reference and provided me with insightful suggestions, motivation, feedback and courage. Reaching the end of my PhD, I would like to thank all of them and express my kindest gratitude, I hold towards them all.

I would like to thank my supervisors, Dr. Raoul van Loon and Prof. Perumal Nithiarasu, for their kind supervision and constant steering in the right direction(s). This research would not have been possible without their vision, support and guidance. The subject of research undertaken was so vast and the methodologies and tools developed have such a bright scope, that it is hoped that future collaborative work will result in more valuable and useful insights.

I would also like to express my gratitude towards my parents, who have untiringly motivated me during times of despair and supported me in every possible way. Thank you, for making me the package, that I have come to become. A special word goes out to my sister, who has accompanied me during this international endeavour and infused a vibrant and lively environment around. Thank you for walking along this journey that will be cherished with extreme fondness, upon reflecting back in a few years time. I would like to also thank all my friends and colleagues for their kind communications from far and close.

The Zienkiewicz scholarship from the college of Engineering, Swansea University, that realized this foreign endeavour for me, is heavily acknowledged. The computational resources that were made available for the purpose of this research, from Barcelona Supercomputing Center, Barcelona, Spain (HPC Europa 2) and

High Performance Computing Wales, Wales, UK are greatly acknowledged. The simulations of this research would not have been possible without the support from these supercomputing centres. Also, the numerous support tickets that were raised with these centres have contributed in solving problems on multiple occasions. I would also like to thank the quick PETSc team for all their technical support and knowledge shared in regard to the high performance computations of algebraic systems.

I would also like to thank Dr. Igor Sazonov, Swansea University, for providing patient specific meshes and the velocity profile generator.

I'm grateful to all my well wishers, from the entire team at PES Institute of Technology, Bangalore, India, for their teachings and blessings.

Finally, and as always,
To the creator of everything, for everything!

Summary

In this research, the finite element method (FEM) was used to solve the non-linear, incompressible, transient, three dimensional Navier-Stokes equations in their non-conservative form. Linear tetrahedron elements were employed with the elegant, equal order interpolation for both pressure and velocity. The characteristic based split scheme was formulated in a fully implicit manner to circumvent the time step restrictions of the classical explicit formulations. The monolithic (single step, fully coupled solution procedure for pressures and velocity) form of the CBS scheme was also derived and its suitability was positively demonstrated. Casting the CBS scheme in a monolithic framework, results in the generation of a pressure stabilization term in the mass conservation equation, thereby circumventing the LBB restriction by the elimination of the zero pressure block. An account of all the steps involved in discretizing the Navier-Stokes equations (both in split and monolithic frameworks) was presented in meticulous detail, which included the derivation of the convective and pressure stabilization terms, linearization of the non-linear terms and the consequent derivation of the highly efficient analytical jacobian matrix, along with the temporal and spatial discretizations of the corresponding terms.

The monolithic and the split version of the CBS scheme were integrated into a parallel, scalable and extensible Fortran90 software called IFENs. The development of IFENs started during the course of this research and all of its components have been designed and implemented by the author of this thesis. Multi processor parallelism was achieved using the Intel® implementation of the most widely used/preferred, Message Passing Interface (MPI) standard. The parallel support needed for the use of a variety of parallel, linear, iterative solvers belonging to the Krylov subspace family (e.g. GMRES and its variants, CG, BiCG, BiCG-stab, etc.), parallel non linear solvers belonging to the Newton-Krylov family (line search newton, trust region newton, nonlinear GMRES, etc.) and parallel preconditioners (incomplete LU, Additive Shwarz Method - ASM, algebraic multigrid, etc.), was provided by the incorporation of PETSc into IFENs. PETSc is a state of the art, non-trivial toolkit, which represents a collection of several parallel libraries useful in high performance scientific computing. Keeping in mind the specific requirements of IFENs, a custom mesh partitioner was implemented. It operated on meshes that were renumbered using bandwidth reducing algorithms like Revere Cuthill Mckee. The possibility of using established domain decomposition libraries like ParMETIS was explored and demonstrated to be counter productive for the demands of this research.

After the preliminary testing and validation of the procedures adopted before and during the execution of IFENs, large, high definition domains representative of human arteries (specifically, carotid bifurcations, found in the neck) were considered and the complete incompressible set of Navier-Stokes equations were solved for pressure and velocity fields. During the tenure of this research more than 1000 recorded parallel test cases were executed to test various components of IFENs, as well as various simulations representative of a wide variety of problems. IFENs can easily handle meshes with tens of millions of elements. The largest mesh used for the purpose of this research contained 14.58 million tetrahedrons and 2.489 million nodes, which on average required just 7 minutes per timestep, while executing the classical split framework of the CBS scheme. Results from the simulation of 9 carotid meshes, representative of 4 carotid geometries were presented and found to be in good agreement with the available ultrasound data. The flow fields were analysed and post processed using different techniques for each case. The haemodynamic wall parameters like time averaged wall shear stress and oscillatory shear index were calculated and mapped onto the corresponding boundary nodes. The region in the carotid bifurcation susceptible to the deposition of plaques and consequent stenosis were pointed out and other anomalies were highlighted.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

In the modern times, mathematical modellinghas come to become an integral part of various areas related to engineering, natural sciences and social sciences. These range from low fidelity models designed to only roughly capture the trends, to high fidelity models which mimic the underlying physics/mechanism within very close tolerances. Computational Fluid Dynamics (CFD) is one such discipline where numerical approximation methods like the Finite Element Method (FEM), Finite Volume Method (FVM), Finite Difference Method (FDM), Spectral Element Method (SEM) [7, 79, 42, 31]etc., are typically utilized to analyze and accurately solve complex problems of fluid flow, heat transfer and related phenomena using digital computers. Central to the mathematical description of fluid flow are the Navier-Stokes equations, which are presented in detail in Chapter 2. These are a set of conservation laws and are widely used for simulating fluid flow in a wide variety of applications.

1.2 Brief historical overview and applications

Just over a century ago, in 1910, Lewis Fry Richardson, presented one of the first studies on the use of finite differences for stress analysis of a masonry dam [130], at the Royal society of London. Less than two decades later, in 1928, the very first CFD applications of FDM began to appear. Some of these early contributions include the contributions from Courant, Friedrichs, and Lewy in 1928 [37]; Evans and Harlow in 1957 [102]; Lax and Wendroff in 1960 [89] and MacCormack in 1969 [93]. FEM applications to CFD appeared much later in 1965 with the works of Zienkiewicz and Cheung [159, 26], Oden in 1972 [116], Chung in 1978 [30], to name a few.

The emergence of digital computers in the early 1950s ushered the development of modern CFD. The advent of cheap and fast computational resources over the recent past has contributed to the success and popularity of CFD techniques for large, real world applications. Some common applications of CFD include flow simulations around aircrafts, spacecrafts, vehicles and ships; weather forecasting, reservoir modelling, electronic heat sink design, manufacturing process simulation, ventilation systems design, etc. The past few years have also witnessed a rise in the use of CFD techniques for biomedical applications. To date, several studies in the literature use Navier-Stokes equations, in some form, to predict and analyze complex flow mechanisms within the human body. Some common biomedical applications of CFD include vessel specific flow simulations (e.g. carotid [144, 55, 152], aorta [14, 85], cerebral arteries [148, 147] etc.), full arterial system simulations [107], fontan circulation [54, 20], cardiac and arterial remodeling [4], nasal airway simulations [138, 113, 100], lymph flow simulations [155], valve function [6, 48], etc.

1.3 Finite Element Method

In this research, the finite element method, which has received considerable attention, both in academia and industry, has been employed to solve the incompressible Navier Stokes equations. FEM is a common choice in the field of CAE (Computer Aided Engineering). FEM can easily accommodate complex geometries and unstructured meshes without the need for coordinate transformations, like that needed in FDM. Also, the Neumann boundary conditions can be conveniently and naturally imposed due to the occurrence of the first order derivatives after integration by parts (unlike FDM). As illustrated in [31], the finite volume method can be formulated both from FDM and FEM. For simple, one-dimensional problems, the finite difference/volume/element methods give identical algebraic equations. Making a choice between the use of different methods like FEM, FVM, FDM, etc. often depends on the nature of problem, available computational resources, geometry, mathematical and computational background; personal and experiential preferences, etc., and continues to remain a topic of debate.

The basic theme of the finite element method divides the domain of interest into a finite number of sub domains that have a finite number of connections with the neighbouring elements. This information is contained in what is called as the finite element mesh. The mesh contains information regarding nodes (non-physical points on the domain) and elements (region enclosed by a set of nodes - line in 1D, area in 2D and volume in 3D). As a result of splitting the domain, the original continuous problem with an infinite number of unknowns can be represented with a finite, solvable number of degrees of freedom. Typically, the variational or weak form of the governing equations is derived next using a suitable approach, for e.g. Galerkin Method [45], where the residual is weighted by the shape functions and integrated over the entire domain. This integral is set equal to zero to imply minimization of the errors. After assembling the contributions

from all the discrete elements that encompass the original domain, along with the simultaneous or post-assembly application of boundary conditions, an algebraic system representative of the weighted residual integral is ready to be solved. Sometimes, the arising matrices are lumped to get a matrix-free solution strategy. When a consistent matrix system is retained, suitable solvers (direct or iterative) must be used depending on the problem size.

When the Galerkin formulation is used for the solution of incompressible Navier Stokes equations, a couple of numerical instabilities arise. One results in oscillatory velocity field, while the other results in spurious pressure oscillations. The non self-adjoint [1] unsymmetric convective terms result in node to node oscillations in velocity, especially in convection dominated flows. The second instability is encountered when the same order of shape functions are used for both pressure and velocity. The equal order interpolations for both pressure and velocity results in the violation of the inf-sup or Ladyzhenskaya-Babuška-Breezi (LBB) restriction [9, 13], which is a condition for well posedness. From an engineering perspective, the LBB violation may be associated with the presence of zero diagonal terms, when visualising the discretized Navier-Stokes equations as an algebraic matrix system of the form $[A]\{x\} = \{b\}$, with,

$$A = \begin{bmatrix} C & G \\ D & 0 \end{bmatrix} \tag{1.1}$$

where, C, G and D are the discrete mass-convection-diffusion, pressure gradient and divergence sub-matrices/operators, respectively. The systems of the form represented in Equation (1.1) are called saddle point problems, which imply the presence of a zero block on the diagonal [154].

[1] self-adjoint: Consider an operator $\tilde{L} = P_0\frac{d^2u}{dx^2} + P_1\frac{du}{dx} + P_2u$. Its self adjoint operator is, $\tilde{L}^\dagger = \frac{d^2}{dx^2}(P_0u) - \frac{d}{dx}(P_1u) + P_2u = P_0\frac{d^2u}{dx^2} + (2P_0^{'} - P_1)\frac{du}{dx} + (P_0^{''} - P_1^{'} + P_2)$. For self adjoincy, $\tilde{L}^\dagger u = \tilde{L}u$. In this case, when, $P_0^{'} = P_1 (\implies P_0^{''} = P_1^{'}, \therefore P_0^{''} - P_1^{'} + P_2 = P_2)$, L will be self adjoint.

As illustrated in [162], the standard Galerkin spatial discretization has the effect of introducing negative diffusion by virtue of a central difference like approximation of the convective terms. Diffusion terms are known to introduce stability, hence with less overall diffusion in the standard Galerkin procedure, instabilities arise, especially when the Peclet number is high. In order to address the convective instability, several techniques have already been proposed, some of these were inspired by the Finite Difference community. Starting off with the steady-state assumption, one can list methods like, Petrov Galerkin method (PG) [65, 58, 162], streamline-upwind petrov Galerkin (SUPG) [71, 70, 83] (also referred to as streamline balancing diffusion), Galerkin least square approximation (GLS) [73, 136], and finite increment calculus method (FIC) [117, 118].

All the methods mentioned above result in approximations that are similar or comparable to that obtained with the Petrov-Galerkin method. Adopting the concept of one-sided finite differencing or up-winding, the PG stabilization was developed. With respect to the standard Galerkin procedure, the PG method employs different weighting functions, which have the effect of introducing additional numerical diffusion/damping, thereby addressing the issue of convective instability. When considering multiple spatial dimensions, the SUPG method turns out to be superior as it adds the balancing diffusion in the direction of the resultant velocity, i.e. in an anisotropic manner. This is also consistent with the idea that information propagates in the direction of velocity, which the Finite Difference community originally used while introducing the one-sided difference approximations.

For solving time dependent equations, the steady state strategies of the PG based methods have been accordingly adopted [156, 157, 35, 33, 23]. From the point of view of explict treatments, the PG methods are difficult to use as they result in non-symmetrical mass matrices, which are non-trivial to lump. Therefore, the use of methods like Taylor Galerkin (TG) [64, 43, 143] and more importantly, Char-

acteristic Galerkin (CG) [46, 123, 92] have been suggested by Zienkiewicz et al. [162]. Codina [34] compared these two methods along with some other methods to reveal the similarity between them. He concluded that just the operator acting on the test functions set each of the methods apart, consequently providing different stabilizing effects. In methods like TG and CG, the temporal discretization precedes spatial discretization. As a result of performing the temporal discretization in these methods, there is a natural introduction of the balancing diffusion like terms of the PG family, which provide convective stabilization.

The characteristic-type methods are based on the wave nature of the equations and convect the spatial coordinates along the characteristic. Doing so, the non-adjoint terms get eliminated (and therefore, Galerkin spatial approximations are optimal in the energy norm sense [162]), but a need to update the mesh arises. Since updating the mesh is computationally expensive and distorted elements might result in multi dimensions, a local Taylor expansion is generally used, which is described in Chapter 2. For solving the Navier Stokes equations, an extension of CG, called Characteristic Based Split (CBS) scheme is available. The splitting strategy used here follows from the finite difference work of Chorin [27, 28], for incompressible flows. In the search for a unified scheme for both compressible and incompressible flows, Zienkiewicz and Codina [160, 161] adapted Chorin's split into a finite element setting and introduced the CBS algorithm in 1995.

The split being referred to, in the CBS scheme relates to the treatment of the pressure terms in the momentum conservation equation and hence to the decoupling of pressure and velocity fields. Two different variants of the split are available in the CBS framework. In one, all the pressure gradient terms are removed (split A), while in the other, those pressure gradient terms corresponding to the start of step (time step n) are retained (split B). Although the second split appears to be more accurate, it is not the preferred choice, because it imposes restrictions on the nature of permissible interpolating functions for pressure and

velocity, in light of the LBB restriction. Therefore, using the split A of the CBS scheme, the LBB restriction, whose violation leads to oscillatory pressure fields, gets automatically addressed. Schemes that separate the pressure terms from the momentum equation during the solution phase, are referred to as *projection/split* schemes. Whereas, schemes that retain all pressure terms, as they appear in the Navier-Stokes equations, are referred to as *monolithic* schemes (although this term is more commonly used in the fluid structure interaction (FSI) community to indicate that the fluid and solid equations are solved together). In the monolithic framework, a range of different pressure stabilization techniques are available to satisfy the LBB restriction, when equal order interpolations for velocity and pressure are desired.

The pressure stabilizations in the monolithic framework are mostly themed around augmenting the mass conservation equation appropriately, in order to eliminate the zero pressure contribution. This results in the equations becoming nearly incompressible. A very intuitive pressure stabilization, inspired from the solid mechanics community would be to just add a pressure term, scaled by a penalty number in the mass conservation equation. This is however a crude solution and sensitive to the choice of the scaling factor chosen. The penalty method [8, 72], on the other hand circumvents the LBB restriction by eliminating the mass conservation equation altogether, with modification to the momentum balance equation and suitably imposing a constraint on the divergence of velocity, in the process. Another pressure stabilization introduces the second order, pressure laplacian in the mass conservation equation [119]. Another simple augmentation was suggested in [19, 50]. Here, a local averaging operator was used to construct a local matrix that was subtracted from the local mass matrix and assembled into the mass conservation equation. Chorin [29] introduced the artificial compressibility method for steady flows. Nithiarasu [112] extended the artificial compressibility method to the CBS scheme, where the transient density term in the mass

conservation equation was replaced by an equivalent pressure term (scaled suitably). The petrov-galerkin based methods have a pressure stabilized version called pressure stabilized petrov galerkin (PSPG) [145, 146], where the residual of the momentum conservation equation is used to populate the pressure block in the mass conservation equation. The community of iterative algebraic solvers view the problem of constructing the pressure stabilizer, from a preconditioning view point. This is a rather algebraic way of solving physical problems, but they have been successful in constructing block preconditioning strategies that suitably remove the zero block of Equation (1.1) and result in accurate, convergent systems. The purely algebraic considerations are often portrayed positively in such treatments, as they render the possibility of providing a black box like solution to potential users. Most of these preconditioners are based on constructing an approximation of the Schurcomplement [158] in an inexpensive manner. Such preconditioners include the least squares commutator (LSC), pressure convection diffusion (PCD), Augmented Lagrangian (AL) [15] etc, which have been described in [49, 82, 140, 15, 51] and the references within. In [154], another strategy called the SILU preconditioner is proposed, though for discretizations obeying the LBB condition. The SILU proposition is in regard to a particular arrangement of degrees of freedom, such that pivoting will not be necessary and the performance of SILU has been shown to be comparable to the block preconditioners mentioned before.

1.4 Aims and objectives

As briefly illustrated in the previous section, there are a number of possible schemes and their combinations to consider while numerically approximating the incompressible Navier-Stokes equations, each with its own merits and limitations. Following the past success and experience with the CBS scheme, it was chosen

to be used for this research. Also, a monolithic version of the CBS scheme was intended to be introduced with the aim of providing a single step solution. It was also intended to develop a computational framework in which these schemes could be efficiently and easily implemented for running large problems with millions of degrees of freedom. The major application of interest for this research was the flow within human arteries, especially the carotid bifurcations [150]. As a starting point, the fully implicit version of the CBS scheme would be considered and the framework essential for both split and monolithic versions will be developed simultaneously. This framework will be developed in a parallel, high performance computing environment. From a broader perspective, the monolithic framework not only circumvents the splitting errors but also provides the opportunity to easily make the code extensible for multi-physics applications like FSI.

1.5 Motivation - a biomedical view

As per the World Health Organization (WHO), cardiovascular diseases are the leading cause of deaths, globally [115]. As per the British Heart Foundation (BHF), coronary heart disease is the UK's single biggest killer. Various complex biochemical reactions (known and possibly unknown) result in conditions favourable for the deposition of plaque within the arteries that restrict blood flow. This condition is referred to as Atherosclerosis. A mathematical modelling approach that can truly represent and predict the flow inside arteries would be highly valuable in this regard. Such techniques can potentially enrich the medical decision-making process, which is mostly experiential in nature. Another important benefit of using such techniques is their inherently non-invasive nature. This implies that important and life saving predictions can be made without breaking the surface of the skin or inserting probes/sensors into the body. Furthermore, given the highly adaptive nature of the human body, every patient may be consid-

ered a unique system and use of such mathematical techniques have the potential of delivering non-generalized, highly patient-specific treatments.

Many computational studies can be found in literature, that attempt to simulate blood flow in human arteries like carotids. However, the mesh used in most of these studies is generally not a very good representation of the real anatomical geometry [152, 144, 85]. These models are mostly idealised and roughly/approximately represent the true shape. Since, slight geometric variations will result in considerable variation in the flow field and the combination of a couple of such variations can result in a completely new flow field, this is a serious problem. Also the meshes are generally not densely packed and do not contain refinement at the walls for capturing the high gradients that exist on these locations. Meshes with less than 100,000 tetrahedron elements and just a few thousand nodes have commonly been observed, to be used for carotid bifurcations. With such coarse meshes there is no point in seeking a high fidelity solution from the model. Also, many studies to date rely on commercial packages to solve the Navier-Stokes equations. This strategy comes with many constraints that are imposed by the software publisher, in the interest of selling the software to a wider user base. The closed source nature of these programs also imply that the fluid solvers get invisible to the users and it becomes impossible to make changes to the scheme to meet the requirements of the problem. In this research however, very accurate representation of the actual carotid anatomy is captured from the scans and structured boundary layers (upto 12) have been used with tens of millions of tetrahedrons in a single carotid mesh. A custom parallel solver called IFENs was also developed, which could easily accommodate these multi-million element meshes and give results in reasonable wall times.

1.5.1 Problem size

Ideally, such an approach would realize at the cost of being extremely multi physics. One would need a fluid solver, solid solver, fluid-solid coupling strategy, endothelial solver, several chemical kinetics kernels, remodelling strategies and an efficient implementation and parallelization of each of the components in an extensible, modular and strongly coupled computer code. To complete the perspective, it should be possible to run the entire multi physics engine on the entire human arterial system in all spatial dimensions and time, in realistic wall times [2]. The realization of such an ideal model will not only require collaboration between groups of mechanical engineers, chemists, computer scientists, doctors and managers, but also extensive real world validation using patient data. With all technology developed, validation against patient data alone involves months of applications for approvals from the local Research Ethics Committee and the R&D offices. This illustrates the mammoth problem size and gives a holistic view of the components that must be employed in making accurate predictions for systems, such as the human circulatory system.

Realistically, such a complete solution would not be generally possible due to constraints of available resources. However, various groups around the world model specific sub-components based on certain assumptions, that substitute for the absence of the highly comprehensive, high fidelity framework mentioned before.

1.6 Assumptions

The following assumptions have been made in this research:

- The fluid is incompressible and Newtonian.

[2]Execution time of a parallel code as measured on a wall clock (different from CPU time)

- The flow field is laminar.
- The walls of the arteries are rigid, i.e. they do not deflect or deform under the action of stresses generated by the pulsatile flow.
- CGS units are used throughout this thesis, unless stated otherwise.

1.7 A note on application of parallel computing to arterial flows

Being an active medical problem, blood flow simulations are increasingly being pursued by various computational groups around the world. As more realistic predictions are desired in 4D (space and time), the problem size inherently increases and therefore adoption of efficient methods like parallelization become mandatory, to get accurate, timely solutions. In as early as 1989, three dimensional simulations of the carotid bifurcations were being performed with the aid of parallel and vector processing techniques [131, 121].

In early 1990s, a standard for message passing between a grid of participating processors on distributed memory systems was conceived and put together. This standard came to be known as the Message Passing Interface (MPI) [56]. The MPI paradigm was realized to be a very natural and convenient parallel computing model for CFD applications and started being adopted in mid 1990s [1, 22, 105, 122]. Applications relating to arterial flows soon followed [151, 53, 39].

Another parallelization paradigm, called Open Multi-Processing (OpenMP) [38], was introduced in October 1997. This parallelization paradigm employs multi-threading, in which, a specified number of slave threads are forked by a master thread. With respect to MPI programs, OpenMP compatible program are relatively simpler for the end users to program. All interprocessor communications

are managed automatically under the OpenMP framework and therefore only trivial changes need to be made to parallelize serial codes. Although OpenMP has been utilized for blood flow applications [62, 106, 24], problems regarding portability, robustness and scalability exist.

The shortcomings of OpenMP get elegantly addressed by the MPI standard and has therefore became a popular choice for large problems. In the last decade, several groups have relied solely on MPI for parallelization [44, 17, 21, 32]. Hybrid MPI-openMP models have also become a popular choice for many bio-fluid simulations [84, 69, 18]. Such a hybridization seems simpler to implement for existing MPI codes, than for existing OpenMP codes.

Implicit and monolithic methods have recently gained popularity and are being adopted for arterial flows. Since large algebraic systems need to be usually solved, the parallelization process becomes more complicated. However, several libraries currently exist with varied capabilities to efficiently solve the algebraic systems arising in implicit discretizations. Some groups tend to program and parallelize algebraic solvers, but considering the libraries and toolkits that already provide accurate, efficient and scalable solver-preconditioner combinations, the recommended approach is to interface with existing tools. Many studies have successfully taken this path [88, 128, 96, 114] by employing libraries/toolkits like PETSc, Trilinos, Hypre, etc.

1.8 Thesis outline

A total of six chapters and 5 appendices constitute this thesis. A brief description about each of these chapters is presented in the following parts of this section. *This chapter* provides background and introduction to the matters of interest to this research. Starting with the chronological events that led to the modern CFD, a brief introduction of the available numerical schemes for dealing with incom-

pressible Navier-Stokes equations are presented. The main focus of this research and a motivation for considering bio-medical problem appears next, along with the grand nature of this problem. This chapter ends by providing an outline of this thesis.

Chapter 2 is dedicated to the mathematical formulations used in this research for solving the incompressible Navier- Stokes equations. It derives the Characteristic Based Split scheme in the split form and introduces the monolithic version of the CBS scheme. Since the fully implicit version of CBS is used, the inherent linearization required for the convective terms is presented next. This leads to the derivation of the jacobian matrix that is directly used in the Newton-Krylov solvers employed in this research. The aim of this chapter is to start with the continuous Navier-Stokes equations and derive their fully discrete forms, such that they can be directly coded.

Chapter 3 presents the computational framework in which the schemes derived in Chapter 2 were processed. The libraries used for realizing the parallel support needed were described along with 2 sample codes. The elegance and simplicity with which these samples present the parallel framework and execute typical tasks performed in a finite element application, justify their presence in the main body. Domain decomposition strategies used in this research appear next. A justification of a custom partitioner written for this research is also provided. The PETSc matrix preallocation concept is presented next. The Fortran90 code that was developed from scratch during this research is described in detail but is not exhaustive, in the interest of brevity. A description of mandatory supporting tasks that need to be performed, while running a typical simulation, with the code developed in this research is also provided for completeness. The aim of this chapter is to provide working knowledge of the parallel computing paradigm and to illustrate the ways in which MPI based parallelism was achieved in this research work. Considering the complexity of this step, majority of the time was

spent in writing and developing this framework.

Chapter 4 illustrates several tests that were performed to assess the correctness of both the scheme as well as its implementation within the code. These ranged from standard problems like lid driven cavity, backward facing step, flow past cylinder to certain other tests that were intended to evaluate the effects of mesh renumbering, parallelization, number of processors used, etc. The main aim of this chapter was to validate results of test problems against benchmark data.

Chapter 5 presents the results obtained when the Navier-Stokes equations are solved over complex biomedical domains, specifically the Carotid bifurcation. Four different meshes, acquired and reconstructed from real patients will be used. Typically each of these meshes had close to a million degrees of freedom to solve, for which parallelization was critical. Various Haemodynamic parameters were also calculated for each of the cases and the results were analysed. The main aim of this chapter was to illustrate the application of the code developed in this research to real world, bio-medical applications to get meaningful results.

Chapter 6 presents the conclusions that could be derived from the present research. It summarises the achievements of this research and lists the major milestones of this project. A list of possible future research directions along with the current limitations are also presented. A number of appendices are presented after this chapter.

Appendices A and B provide explanations of the sample MPI and PETSc codes, presented in Chapter 3. The function of various subroutines along with the idea behind the codes are explained. Appendix C includes all the spatial discretization steps that were omitted from Chapter 2. Appendix D explains the construction of a commonly used sparse matrix storage format, called the Compressed Sparse Row (CSR) format. Also, the matrix preallocation strategy is explained. CSR construction is described to present the possibility of using CSR data for preal-

locating matrices in PETSc. Finally, in Appendix E, the closed form expressions (for linear four noded tetrahedron elements are presented) that may be used for the efficient evaluation of the shape function integrals as well as the shape function derivatives, are presented.

Chapter 2

Governing Equations

2.1 Introduction

This chapter describes the formulations used to mathematically describe the flow of fluids. The goal of this chapter is to derive the fully discrete form of the Navier-Stokes equation. These fully discrete forms, resulting from the Finite Element Method, serve as a basis for the computer code that syntactically encodes the governing equations. The formulations will be presented in terms of the primitive variables (velocities and pressure). A monolithic as well as a split version will be presented. The convective and diffusive terms will be treated fully implicitly and hence a large and sparse system of linear equations needs to be solved, at least once, in every time step. However, a couple of Newton iterations will be typically required per time step. Also, the stabilization techniques used will be presented to complete the formulation.

2.2 Navier-Stokes equations

The combined efforts of *Claude Louis Marie Henri Navier, Euler, Cauchy, Poisson, Barre de Saint-Venant and George Gabriel Stokes* led to the derivation of the Navier-Stokes equations in a manner that is currently understood. The three dimensional, transient Navier-Stokes equations may be written in fully conservative standard form as shown in Eq. (2.1). For clarity the non-tensorial notation is adopted.

$$\frac{\partial \mathbf{\Phi}}{\partial t} + \frac{\partial \mathbf{F_i}}{\partial x_i} + \frac{\partial \mathbf{G_i}}{\partial x_i} + \mathbf{Q} = \mathbf{0} \tag{2.1}$$

where in general $\mathbf{\Phi}$ is a basic dependent, vector-valued variable, $\mathbf{F_i}$ is the convective flux, $\mathbf{G_i}$ is the diffusive flux and $\mathbf{Q}$ is the source vector.

In the expanded form,

$$\frac{\partial}{\partial t}\begin{Bmatrix} \rho \\ \rho u_1 \\ \rho u_2 \\ \rho u_3 \\ \rho E \end{Bmatrix} + \frac{\partial}{\partial x_i}\begin{Bmatrix} \rho u_i \\ \rho u_1 u_i + p\delta_{1i} \\ \rho u_2 u_i + p\delta_{2i} \\ \rho u_3 u_i + p\delta_{3i} \\ \rho H u_i \end{Bmatrix} + \frac{\partial}{\partial x_i}\begin{Bmatrix} 0 \\ -\tau_{1i} \\ -\tau_{2i} \\ -\tau_{3i} \\ -(\tau_{ij}u_j) - k\frac{\partial T}{\partial x_i} \end{Bmatrix} + \begin{Bmatrix} 0 \\ \rho g_1 \\ \rho g_2 \\ \rho g_3 \\ \rho g_i u_i - q_H \end{Bmatrix} = 0 \tag{2.2}$$

with

$$\tau_{ij} = \mu\left[\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) - \frac{2}{3}\delta_{ij}\frac{\partial u_k}{\partial x_k}\right] \tag{2.3}$$

where, ρ is the density, u_1, u_2 and u_3 represent the three velocity components, p is the pressure, E is the total energy per unit mass, δ represents kronecker delta, H is the enthalpy, τ represents the deviatoric stress, μ is the dynamic viscosity, k is the isotropic thermal conductivity, T is the absolute temperature, q_H represents the heat source terms per unit volume and g represents the body force.

Equation (2.2) represents the mass conservation, three momentum conservation and energy equations, in the same order (from top). Since the changes in density with time are negligible in blood flow applications, it is safe to assume the incompressibility condition. As a consequence of incompressibility, there are only four unknowns and hence we can ignore the energy equation. The mass and momentum conservation under the incompressibility condition, may be written as,

Mass Conservation

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{2.4}$$

Momentum Conservation

$$\frac{\partial u_j}{\partial t} + \frac{\partial}{\partial x_i}(u_j u_i) - \frac{1}{\rho}\frac{\partial \tau_{ji}}{\partial x_i} + \frac{1}{\rho}\frac{\partial p}{\partial x_i}\delta_{ji} = 0$$

The convective term is differentiated to get the non-conservative form (which is safe to use since we are not dealing with high-speed flows where shock capturing is important),

$$\frac{\partial u_j}{\partial t} + u_j\frac{\partial u_i}{\partial x_i} + u_i\frac{\partial u_j}{\partial x_i} - \frac{1}{\rho}\frac{\partial \tau_{ji}}{\partial x_i} + \frac{1}{\rho}\frac{\partial p}{\partial x_i}\delta_{ji} = 0$$

Since the divergence of velocity is equal to zero (from Eq. (2.4)),

$$\frac{\partial u_j}{\partial t} + u_i\frac{\partial u_j}{\partial x_i} - \frac{1}{\rho}\frac{\partial \tau_{ji}}{\partial x_i} + \frac{1}{\rho}\frac{\partial p}{\partial x_i}\delta_{ji} = 0$$

On removing the kronecker delta from the above equation, the derivative index

for p changes from i to j, as $\delta_{ij} = 1$, only if $i = j$,

$$\frac{\partial u_j}{\partial t} + u_i \frac{\partial u_j}{\partial x_i} - \frac{1}{\rho}\frac{\partial \tau_{ji}}{\partial x_i} + \frac{1}{\rho}\frac{\partial p}{\partial x_j} = 0 \tag{2.5}$$

Since Eq. (2.5) is not self-adjoint due to the convective term, this equation, as is, cannot be derived from any variational principle.

2.3 Characteristic Based Split (CBS) algorithm

The CBS algorithm is based on the split process initially proposed by Chorin [27, 28] in the Finite Difference context. In this research, the algorithm proposed by Zienkiewicz and Codina [160, 161], which is a rather general approach to numerically solving Navier-Stokes equations, is used. Starting off with the discretization in the time domain, the actual splitting followed by the spatial discretization, will be presented.

2.3.1 Time discretization

The momentum conservation equation will now be dicretized in time to get its semi-discrete form, using the Characteristic Galerkin scheme. In doing so, the convective stabilization will also be established.

To simplify the derivation, in the succeeding part of this section, the continuous form of the momentum conservation equation will be shown to be similar to the reduced, scalar form of Navier-Stokes equation. The characteristic Galerkin scheme will then be derived for the one dimensional, scalar Navier-Stokes equations and extended to the vector form of Navier-Stokes.

Rewriting the full Navier-Stokes equation, as in Eq.(2.1),

$$\frac{\partial \mathbf{\Phi}}{\partial t} + \frac{\partial \mathbf{F_i}}{\partial x_i} + \frac{\partial \mathbf{G_i}}{\partial x_i} + \mathbf{Q} = \mathbf{0}$$

If,

$$\mathbf{\Phi} = \phi \ ; \ \mathbf{F}_i = U_i\phi \ ; \ \mathbf{G}_i = -k\frac{\partial \phi}{\partial x_i} \ ; \ \mathbf{Q} = Q(x_i) \tag{2.6}$$

then,

$$\frac{\partial \phi}{\partial t} + \frac{\partial}{\partial x_i}(U_i\phi) - \frac{\partial}{\partial x_i}\left(k\frac{\partial \phi}{\partial x_i}\right) + Q = 0 \tag{2.7}$$

In one spatial dimension,

$$\frac{\partial \phi}{\partial t} + \frac{\partial}{\partial x}(U\phi) - \frac{\partial}{\partial x}\left(k\frac{\partial \phi}{\partial x}\right) + Q = 0$$

Like before, since divergence of velocity is zero, the non-conservation form yields,

$$\frac{\partial \phi}{\partial t} + U\frac{\partial \phi}{\partial x} - \frac{\partial}{\partial x}\left(k\frac{\partial \phi}{\partial x}\right) + Q = 0 \tag{2.8}$$

The continuous form of the momentum equation (Eq. (2.5)) is similar to the reduced NS equations (Eq. (2.8)), provided the pressure term in Eq. (2.5) is treated as a source term. Hence, the temporal discretization of Eq. (2.8) will be presented in the following section and extended to Eq.(2.5).

2.3.1.1 Characteristic Galerkin Scheme

Eq. (2.8), is indeed the convection-diffusion equation, which in this section will be discretized in the time domain. Characteristic Galerkin procedure involves a local Taylor expansion illustrated in the figure 2-1. We can write equation (2.8)

along the characteristic as,

$$\frac{\partial \phi}{\partial t}(x'(t),t) - \frac{\partial}{\partial x'}\left(k\frac{\partial \phi}{\partial x'}\right) + Q(x') = 0 \tag{2.9}$$

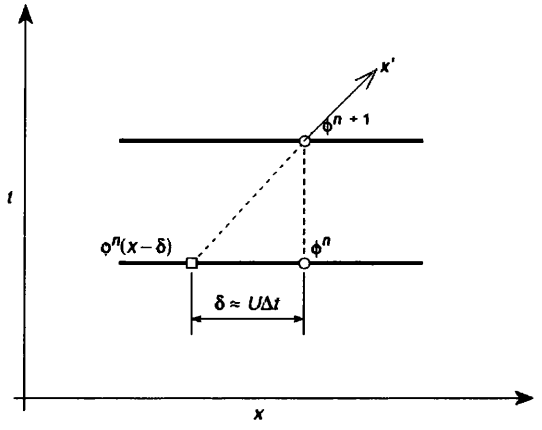


Figure 2-1: Visualizing the Characteristic-Galerkin procedure

In the moving coordinate x', the convective acceleration term disappears and the source and diffusion terms are averaged along the characteristic. In the absence of the convective term, Eq. (2.9) is fully self-adjoint and Galerkin spatial approximation is optimal. Time discretization of Eq. (2.9) along the characteristic gives (Fig. 2-1),

$$\frac{1}{\Delta t}\left(\phi^{n+1} - \phi^n|_{x-\delta}\right) \approx \theta\left[\frac{\partial}{\partial x}\left(k\frac{\partial \phi}{\partial x}\right) - Q\right]^{n+1} + (1-\theta)\left[\frac{\partial}{\partial x}\left(k\frac{\partial \phi}{\partial x}\right) - Q\right]^{n}\Bigg|_{x-\delta} \tag{2.10}$$

where θ is equal to zero for explicit forms, between zero and unity for semi-implicit and unity for fully implicit forms. Here, θ will be preserved until the end to maintain generality. The version of time discretization selected here, is first order

accurate.If the moving coordinates are retained, the mesh needs to be updated, which is a cumbersome process. Hence, the moving coordinate is eliminated by spatially Taylor expanding the three terms that need to be evaluated at $(x-\delta)$ in Eq. (2.10).

$$\begin{aligned}
\phi^n|_{x-\delta} &\approx \phi^n - \delta\frac{\partial\phi^n}{\partial x} + \frac{\delta^2}{2}\frac{\partial^2\phi^n}{\partial x^2} + O(\delta^3) \\
(1-\theta)\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)\Big|_{x-\delta} &\approx (1-\theta)\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)^n - \delta(1-\theta)\frac{\partial}{\partial x}\left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)^n\right] + O(\delta^2) \\
(1-\theta)Q|_{x-\delta} &\approx (1-\theta)(Q^n) - (1-\theta)(\delta)\frac{\partial Q^n}{\partial x} + O(\delta^2)
\end{aligned} \tag{2.11}$$

Substituting Eqs. (2.11) in Eq. (2.10),

$$\begin{aligned}
\frac{1}{\Delta t}\left(\phi^{n+1} - \left(\phi^n - \delta\frac{\partial\phi^n}{\partial x} + \frac{\delta^2}{2}\frac{\partial^2\phi^n}{\partial x^2}\right)\right) \approx\ & \theta\left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) - Q\right]^{n+1} + (1-\theta)\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)^n \\
& -\delta(1-\theta)\frac{\partial}{\partial x}\left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)^n\right] - (1-\theta)(Q^n) \\
& +(1-\theta)(\delta)\frac{\partial Q^n}{\partial x}
\end{aligned} \tag{2.12}$$

Rearranging,

$$\begin{aligned}
\frac{1}{\Delta t}\left(\phi^{n+1} - \phi^n\right) \approx\ & -\frac{\delta}{\Delta t}\frac{\partial\phi^n}{\partial x} + \frac{\delta^2}{2\Delta t}\frac{\partial^2\phi^n}{\partial x^2} \\
& +\theta\left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) - Q\right]^{n+1} + (1-\theta)\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)^n \\
& -\delta(1-\theta)\frac{\partial}{\partial x}\left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)^n\right] - (1-\theta)(Q^n) \\
& +(1-\theta)(\delta)\frac{\partial Q^n}{\partial x}
\end{aligned} \tag{2.13}$$

where, δ is the horizontal distance travelled along the x axis and is simply given by,

$$\delta = \bar{U}\Delta t \tag{2.14}$$

where, $\bar{U}$ is the average velocity along the characteristic and is chosen to be written as,

$$\bar{U} = \frac{U^{n+1} + U^n|_{x-\delta}}{2}$$

Again, Taylor expanding, to get rid of the moving coordinate, i.e.,

$$\bar{U} = \frac{U^{n+1} + \left(U^n - \Delta t U^n \frac{\partial U^n}{\partial x}\right)}{2} \tag{2.15}$$

Substituting (2.15) in (2.14),

$$\delta = \Delta t U^{n+\frac{1}{2}} - \frac{\Delta t^2}{2} U^n \frac{\partial U^n}{\partial x} \tag{2.16}$$

where, $U^{n+\frac{1}{2}} = \frac{U^{n+1}+U^n}{2}$

Substituting for δ from Eq. (2.16) into Eq. (2.13),

$$\begin{aligned}
\frac{\Delta\phi}{\Delta t} \approx & -\left[U^{n+\frac{1}{2}} - \frac{\Delta t}{2} U^n \frac{\partial U^n}{\partial x}\right] \frac{\partial \phi^n}{\partial x} \\
& + \frac{1}{2\Delta t}\left(\Delta t^2 U^{n+\frac{1}{2}} U^{n+\frac{1}{2}} + \frac{\Delta t^4}{4} U^n U^n \left(\frac{\partial U^n}{\partial x}\right)^2 - \frac{\Delta t^3}{2} U^{n+\frac{1}{2}} U^n \frac{\partial U^n}{\partial x}\right) \frac{\partial^2 \phi^n}{\partial x^2} \\
& + \theta\left[\frac{\partial}{\partial x}\left(k\frac{\partial \phi}{\partial x}\right) - Q\right]^{n+1} + (1-\theta)\frac{\partial}{\partial x}\left(k\frac{\partial \phi}{\partial x}\right)^n \\
& - \left(\Delta t U^{n+\frac{1}{2}} - \frac{\Delta t^2}{2} U^n \frac{\partial U^n}{\partial x}\right)(1-\theta)\frac{\partial}{\partial x}\left[\frac{\partial}{\partial x}\left(k\frac{\partial \phi}{\partial x}\right)^n\right] - (1-\theta)(Q^n) \\
& + (1-\theta)\left(\Delta t U^{n+\frac{1}{2}} - \frac{\Delta t^2}{2} U^n \frac{\partial U^n}{\partial x}\right) \frac{\partial Q^n}{\partial x}
\end{aligned}$$

Neglecting the higher order terms,

$$\begin{aligned}\frac{\Delta\phi}{\Delta t} \approx & -\left[U^{n+\frac{1}{2}} - \frac{\Delta t}{2}U^n\frac{\partial U^n}{\partial x}\right]\frac{\partial \phi^n}{\partial x} \\ & + \frac{1}{2\Delta t}\left(\Delta t^2 U^{n+\frac{1}{2}}U^{n+\frac{1}{2}}\right)\frac{\partial^2\phi^n}{\partial x^2} \\ & + \theta\left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) - Q\right]^{n+1} + (1-\theta)\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)^n \\ & - \left(\Delta t U^{n+\frac{1}{2}}\right)(1-\theta)\frac{\partial}{\partial x}\left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)^n\right] - (1-\theta)(Q^n) \\ & + (1-\theta)\left(\Delta t U^{n+\frac{1}{2}}\right)\frac{\partial Q^n}{\partial x} \end{aligned} \tag{2.17}$$

Simplifying and getting rid of the $U^{n+\frac{1}{2}}$ notation,

$$\begin{aligned}\frac{\Delta\phi}{\Delta t} \approx & -\left[\frac{U^{n+1}}{2} + \frac{U^n}{2} - \frac{\Delta t}{2}U^n\frac{\partial U^n}{\partial x}\right]\frac{\partial \phi^n}{\partial x} \\ & + \frac{1}{2}\left(\frac{\Delta t}{4}\left((U^{n+1})^2 + (U^n)^2 + 2U^{n+1}U^n\right)\right)\frac{\partial^2\phi^n}{\partial x^2} \\ & + \theta\left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) - Q\right]^{n+1} + (1-\theta)\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)^n \\ & - \frac{\Delta t}{2}\left(U^{n+1} + U^n\right)(1-\theta)\frac{\partial}{\partial x}\left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)^n\right] - (1-\theta)(Q^n) \\ & + \frac{\Delta t}{2}(1-\theta)\left(U^{n+1} + U^n\right)\frac{\partial Q^n}{\partial x} \end{aligned} \tag{2.18}$$

Fully Explicit case

In Eq. (2.18), put $\theta = 0$ and change all terms on the RHS that are to be evaluated at time level (n+1) to time level n,

$$\begin{aligned}\frac{\Delta\phi}{\Delta t} \approx & -\left[U^n - \frac{\Delta t}{2}U^n\frac{\partial U^n}{\partial x}\right]\frac{\partial\phi^n}{\partial x} + \frac{\Delta t}{8}\left(4(U^n)^2\right)\frac{\partial^2\phi^n}{\partial x^2} \\ & + \frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)^n - \Delta t U^n\frac{\partial}{\partial x}\left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)^n\right] - (Q^n) + \Delta t U^n\frac{\partial Q^n}{\partial x} \end{aligned} \tag{2.19}$$

Rearranging,

$$\frac{\Delta\phi}{\Delta t} \approx -\left[U\frac{\partial\phi}{\partial x} - \frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) + Q\right]^n + \frac{\Delta t}{2}\left[U\frac{\partial U}{\partial x}\frac{\partial\phi}{\partial x} + UU\frac{\partial^2\phi}{\partial x^2}\right]^n - \Delta t U^n \frac{\partial}{\partial x}\left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)^n\right] + \Delta t U^n \frac{\partial Q^n}{\partial x} \tag{2.20}$$

$$\boxed{\frac{\Delta\phi}{\Delta t} \approx -\left[U\frac{\partial\phi}{\partial x} - \frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) + Q\right]^n + \frac{\Delta t}{2}U^n\frac{\partial}{\partial x}\left[U\frac{\partial\phi}{\partial x} - 2\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) + 2Q\right]^n}$$

Semi Implicit case

In Eq. (2.18), put $\theta = \frac{1}{2}$ and using the substitution $\frac{()^{n+1}+()^n}{2} = ()^{n+\frac{1}{2}}$, evaluate all terms on the RHS at time level $\left(n+\frac{1}{2}\right)$,

$$\frac{\Delta\phi}{\Delta t} \approx -\left[U^{n+\frac{1}{2}} - \frac{\Delta t}{2}U^n\frac{\partial U^n}{\partial x}\right]\frac{\partial\phi^n}{\partial x} + \frac{\Delta t}{2}\left(U^{n+\frac{1}{2}}\right)^2\frac{\partial^2\phi^n}{\partial x^2} + \frac{1}{2}\left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) - Q\right]^{n+1} + \frac{1}{2}\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)^n - \frac{\Delta t}{2}U^{n+\frac{1}{2}}\frac{\partial}{\partial x}\left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)^n\right] - \frac{1}{2}(Q^n) + \frac{\Delta t}{2}U^{n+\frac{1}{2}}\frac{\partial Q^n}{\partial x} \tag{2.21}$$

writing, $\frac{1}{2}\left(\left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) - Q\right]^{n+1} + \left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) - Q\right]^n\right)$ as $\left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) - Q\right]^{n+\frac{1}{2}}$,

$$\frac{\Delta\phi}{\Delta t} \approx -\left[U^{n+\frac{1}{2}} - \frac{\Delta t}{2}U^n\frac{\partial U^n}{\partial x}\right]\frac{\partial\phi^n}{\partial x} + \frac{\Delta t}{2}\left(U^{n+\frac{1}{2}}\right)^2\frac{\partial^2\phi^n}{\partial x^2} + \left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) - Q\right]^{n+\frac{1}{2}} - \frac{\Delta t}{2}U^{n+\frac{1}{2}}\frac{\partial}{\partial x}\left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)^n\right] + \frac{\Delta t}{2}U^{n+\frac{1}{2}}\frac{\partial Q^n}{\partial x} \tag{2.22}$$

Rearranging,

$$\frac{\Delta\phi}{\Delta t} \approx -\left[U^{n+\frac{1}{2}}\frac{\partial\phi^n}{\partial x} - \left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) + Q\right]^{n+\frac{1}{2}}\right] + \frac{\Delta t}{2}\left[U^n\frac{\partial U^n}{\partial x}\frac{\partial\phi^n}{\partial x} + \left(U^{n+\frac{1}{2}}\right)^2\frac{\partial^2\phi^n}{\partial x^2}\right] - \frac{\Delta t}{2}U^{n+\frac{1}{2}}\frac{\partial}{\partial x}\left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right)^n\right] + \frac{\Delta t}{2}U^{n+\frac{1}{2}}\frac{\partial Q^n}{\partial x} \tag{2.23}$$

Writing all the left over terms at time level n as $\left(n+\frac{1}{2}\right)$,

$$\boxed{\frac{\Delta\phi}{\Delta t} \approx -\left[U\frac{\partial\phi}{\partial x} - \frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) + Q\right]^{n+\frac{1}{2}} + \frac{\Delta t}{2}U^{n+\frac{1}{2}}\frac{\partial}{\partial x}\left[U\frac{\partial\phi^n}{\partial x} - \frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) + Q\right]^{n+\frac{1}{2}}}$$

Fully Implicit case

In Eq. (2.18), put $\theta = 1$ and change all terms on the RHS that are to be evaluated at time level n to time level (n+1),

$$\begin{aligned}\frac{\Delta\phi}{\Delta t} \approx & -\left[U^{n+1} - \frac{\Delta t}{2}U^{n+1}\frac{\partial U^{n+1}}{\partial x}\right]\frac{\partial\phi^{n+1}}{\partial x} + \frac{\Delta t}{2}\left((U^{n+1})^2\right)\frac{\partial^2\phi^n}{\partial x^2} \\ & + \left[\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) - Q\right]^{n+1} + 0 \end{aligned} \tag{2.24}$$

Rearranging,

$$\boxed{\frac{\Delta\phi}{\Delta t} \approx -\left[U\frac{\partial\phi}{\partial x} - \frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) + Q\right]^{n+1} + \left[\frac{\Delta t}{2}U\frac{\partial}{\partial x}\left(U\frac{\partial\phi}{\partial x}\right)\right]^{n+1}}$$

In summary, the time discretization using the characteristic Galerkin scheme, results in the following semi-discrete equations. The second square-bracketed term in the RHS of the following 3 equations represents the convective stabilization.

Fully Explicit

$$\boxed{\frac{\Delta\phi}{\Delta t} \approx -\left[U\frac{\partial\phi}{\partial x} - \frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) + Q\right]^{n} + \frac{\Delta t}{2}U^{n}\frac{\partial}{\partial x}\left[U\frac{\partial\phi}{\partial x} - 2\frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) + 2Q\right]^{n}}$$

Semi Implicit

$$\boxed{\frac{\Delta\phi}{\Delta t} \approx -\left[U\frac{\partial\phi}{\partial x} - \frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) + Q\right]^{n+\frac{1}{2}} + \frac{\Delta t}{2}U^{n+\frac{1}{2}}\frac{\partial}{\partial x}\left[U\frac{\partial\phi^n}{\partial x} - \frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) + Q\right]^{n+\frac{1}{2}}}$$

Fully Implicit

$$\boxed{\frac{\Delta\phi}{\Delta t} \approx -\left[U\frac{\partial\phi}{\partial x} - \frac{\partial}{\partial x}\left(k\frac{\partial\phi}{\partial x}\right) + Q\right]^{n+1} + \left[\frac{\Delta t}{2}U\frac{\partial}{\partial x}\left(U\frac{\partial\phi}{\partial x}\right)\right]^{n+1}} \tag{2.25}$$

Since a fully implicit scheme is being formulated, Eq. (2.25) will be considered from this point forward.

2.3.1.2 Extension to multi dimensions (3D)

Eq. (2.25) represents the semi-discrete form of Eq. (2.8) and it may be written in three dimensional indicial notation as,

$$\frac{\Delta\phi}{\Delta t} \approx -\left[U_i\frac{\partial\phi}{\partial x_i} - \frac{\partial}{\partial x_j}\left(k\frac{\partial\phi}{\partial x_j}\right) + Q\right]^{n+1} + \left[\frac{\Delta t}{2}U_k\frac{\partial}{\partial x_k}\left(U_i\frac{\partial\phi}{\partial x_i}\right)\right]^{n+1} \tag{2.26}$$

2.3.1.3 Extension to the momentum equation

Eq. (2.26) represents the semi-discrete form of Eq. (2.8) in multi dimensions. Eq. (2.8) was similar in form to momentum equation represented by Eq. (2.5). Therefore, with a few modifications, Eq (2.26) can serve as the time discretization for the momentum equation.

First, replace ϕ by U_j and Q by $Q_j^{n+\theta_2}$, in Eq (2.26),

$$\frac{\Delta U_j}{\Delta t} \approx -\left[u_i\frac{\partial U_j}{\partial x_i} - \frac{\partial}{\partial x_j}\left(k\frac{\partial U_j}{\partial x_j}\right)\right]^{n+1} + Q_j^{n+\theta_2} + \left[\frac{\Delta t}{2}u_k\frac{\partial}{\partial x_k}\left(u_i\frac{\partial U_j}{\partial x_i}\right)\right]^{n+1} \tag{2.27}$$

where, $Q_j^{n+\theta_2}$ may be evaluated at $t = t+\theta_2\Delta t$. Setting $\theta_2 = 1$ and replacing Q_j^{n+1} by $\frac{\partial p}{\partial x_j}^{n+1}$, the final semi-discrete form of the momentum equation is obtained,

$$\frac{\Delta U_j}{\Delta t} \approx -\left[u_i\frac{\partial U_j}{\partial x_i} - \frac{\partial}{\partial x_j}\left(k\frac{\partial U_j}{\partial x_j}\right) + \frac{\partial p}{\partial x_j}\right]^{n+1} + \left[\frac{\Delta t}{2}u_k\frac{\partial}{\partial x_k}\left(u_i\frac{\partial U_j}{\partial x_i}\right)\right]^{n+1} \quad (2.28)$$

Replacing the diffusion term by deviatoric stress and noting that $U_i = \rho u_i$,

$$\frac{\Delta u_j}{\Delta t} \approx -\left[u_i\frac{\partial u_j}{\partial x_i} - \frac{1}{\rho}\frac{\partial \tau_{ji}}{\partial x_j} + \frac{1}{\rho}\frac{\partial p}{\partial x_j}\right]^{n+1} + \left[\frac{\Delta t}{2}\frac{u_k}{\rho}\frac{\partial}{\partial x_k}\left(u_i\frac{\partial u_j}{\partial x_i}\right)\right]^{n+1} \quad (2.29)$$

In summary,

Momentum Equation:

$$\frac{\partial u_j}{\partial t} + u_i\frac{\partial u_j}{\partial x_i} - \frac{1}{\rho}\frac{\partial \tau_{ji}}{\partial x_i} + \frac{1}{\rho}\frac{\partial p}{\partial x_j} = 0$$

Semi Discrete Momentum Equation:

$$\frac{u_j^{n+1} - u_j^n}{\Delta t} \approx -\left[u_i\frac{\partial u_j}{\partial x_i} - \frac{1}{\rho}\frac{\partial \tau_{ji}}{\partial x_j} + \frac{1}{\rho}\frac{\partial p}{\partial x_j}\right]^{n+1} + \left[\frac{\Delta t}{2}\frac{u_k}{\rho}\frac{\partial}{\partial x_k}\left(u_i\frac{\partial u_j}{\partial x_i}\right)\right]^{n+1}$$

2.3.2 The split

The core idea here is to separate the pressure terms from the semi-discrete momentum equation. Two version of this split are available in an explicit case. We could either remove the pressure gradient terms altogether (called split A) or retain the pressure gradient term corresponding to the beginning of the step, i.e. at time level n (called split B). However, split B has restrictions on the nature of interpolating functions that can be used for pressure and velocities. The restriction arises as a result of violating the Ladyženskaja-Babuška-Brezzi (LBB) condition. For stability, the mass conservation equation must have a small non-zero pressure contribution. Split A has the tendancy of augmenting the mass conservation

equation with a small pressure term and hence circumvents the LBB restriction. Split B on the other hand results in a zero pressure block, violating the LBB restriction and hence must be used with caution. In the fully implicit case however, there is only one pressure gradient term and there is just one possible split.

Step 1

Remove all pressure terms from Eq. (2.29) and introduce the intermediate fields (velocity) represented by superscript *

$$\frac{\Delta u_j^\star}{\Delta t} + \frac{\partial}{\partial x_i}(u_i^\star u_j^\star) - \frac{1}{\rho}\frac{\partial \tau_{ji}{}^\star}{\partial x_i} - \frac{\Delta t}{2}\frac{u_k^\star}{\rho}\frac{\partial}{\partial x_k}\left(u_i^\star \frac{\partial u_j^\star}{\partial x_i}\right) \approx 0 \tag{2.31}$$

where, $\Delta u_j^\star = u_j^\star - u_j^n$

Step 3

While deriving, it is convenient to present the step 3 before step 2. However, while coding, the correct order is used. In this step, the pressure term is recovered by subtracting Eq. (2.29) from Eq. (2.31).

$$\frac{\Delta u_j - \Delta u_j^\star}{\Delta t} + \frac{1}{\rho}\frac{\partial p}{\partial x_j} \approx 0$$

$$\implies u_j^{n+1} \approx u_j^\star - \frac{\Delta t}{\rho}\frac{\partial p}{\partial x_j} \tag{2.32}$$

Step 2

From mass conservation and gas law,

$$\frac{\Delta \rho}{\Delta t} = \frac{1}{c^2}\frac{\Delta p}{\Delta t} = -\rho \frac{\partial u_i^{n+\theta}}{\partial x_i} \tag{2.33}$$

Equation (2.32) may be written as,

$$u_i^{n+1} \approx u_i^\star - \frac{\Delta t}{\rho}\frac{\partial p}{\partial x_i} \tag{2.34}$$

Assuming $\theta = 1$ and substituting equation (2.34) in (2.33),

$$\frac{1}{c^2}\frac{\Delta p}{\Delta t} = -\rho\frac{\partial u_i^\star}{\partial x_i} + \Delta t\frac{\partial^2 p}{\partial x_i^2} \tag{2.35}$$

2.3.3 Spatial discretization

The three steps of the CBS scheme are now ready to be discretized spatially. The standard Galerkin procedure, which is fully justified to be used with the characteristic Galerkin time discretization, is employed. A very detailed account of the spatial discretization, considering 4 noded, linear tetrahedron finite elements is given in Appendix C. In appendix C, the final expressions that are actually coded are derived and presented meticulously. Here just a few details are presented.

Spatial discretization using the standard Galerkin method, results in the following forms for steps 1 through 3,

$$\int_\Omega \hat{N}^T \left(\frac{\Delta u_j^\star}{\Delta t} + u_i^\star \frac{\partial u_j^\star}{\partial x_i} - \frac{1}{\rho}\frac{\partial {\tau_{ji}}^\star}{\partial x_i} - \frac{\Delta t}{2}\frac{u_k^\star}{\rho}\frac{\partial}{\partial x_k}\left(u_i^\star \frac{\partial u_j^\star}{\partial x_i} \right) \right) d\Omega \approx 0 \tag{2.36}$$

$$\int_\Omega \hat{N}^T \left(-\rho\frac{\partial u_i^\star}{\partial x_i} + \Delta t\frac{\partial^2 p}{\partial x_i^2} \right) d\Omega \approx 0 \tag{2.37}$$

$$\int_\Omega \hat{N}^T \left(u_j^{n+1} \right) d\Omega \approx \int_\Omega \hat{N}^T \left(u_j^\star - \frac{\Delta t}{\rho}\frac{\partial p}{\partial x_j} \right) d\Omega \tag{2.38}$$

Just the non-linear convective acceleration term will be spatially discretized in the next section (2.3.3.1). As indicated before, the remaining terms are discretized in appendix C.

2.3.3.1 Linearization

Since a fully implicit treatment is sought for in this research, the non-linear convective acceleration and convective stabilization terms are linearized, deriving the Jacobian matrix in the process, which in turn is utilized by the Non-linear solvers in PETSc to realize the Krylov-Newton-like solver(s). Considering a non-linear system of the form,

$$f(\phi) = 0 \tag{2.39}$$

the update of a Newton iteration is given by,

$$\phi^{nn+1} = \phi^{nn} - \frac{f(\phi)}{f'(\phi)} \tag{2.40}$$

where, the subscript nn represents the newton iteration counter.

The SNES (Scalable Nonlinear Equation Solver) objects in PETSc approximately solve,

$$J\Delta\phi = -f(\phi) \tag{2.41}$$

where, $J = f'(\phi)$, $\Delta\phi = \phi^{nn+1} - \phi^{nn}$ and the end of step update being,

$$\phi_{nn+1} = \phi_{nn} + \Delta\phi$$

Equation (2.41) is solved iteratively until ϕ_{nn+1} is equivalent to ϕ_{nn} within a suitable tolerance, at which point the Newton solver is considered to have converged. In the light of Eq. (2.41), which is completely linear, the non-linear solver may

be viewed as a set of linear solves. Since the non-linear solution is centred around solving a set of linear systems formulated in terms of change in unknowns rather than the unknown variables themselves, in order to build a system in terms of consistent unknowns, the linear terms need to be, so called linearized.

Next, only the non-linear convective acceleration term (Term 10) of Eq. (2.36) will be linearized and spatially discretized. The same treatment may be extended to the rest of the terms with suitable modifications. Noting that at Newton convergence, $u^{n+1} \approx u^{nn+1}$, the convective term may be written in non-conservative form as,

$$\text{Term } 10 = \int_\Omega \hat{N}^T \left(u_i^\star \frac{\partial u_j^\star}{\partial x_i} \right)^{nn+1} d\Omega \tag{2.42}$$

Approximating terms at Newton iteration $nn+1$ as (and dropping the superscript * for convenience),

$$u^{nn+1} \approx u^{nn} + \delta u$$

we may rewrite Eq. (2.42) as,

$$\begin{aligned} \text{Term } 10 &= \int_\Omega \hat{N}^T \left((u_i^{nn} + \delta u_i) \frac{\partial}{\partial x_i} (u_j^{nn} + \delta u_j) \right) d\Omega \\ &= \int_\Omega \hat{N}^T \left(u_i^{nn} \frac{\partial u_j^{nn}}{\partial x_i} + u_i^{nn} \frac{\partial}{\partial x_i}(\delta u_j) + \delta u_i \frac{\partial u_j^{nn}}{\partial x_i} + \cancel{\delta u_i \frac{\partial}{\partial x_i}(\delta u_j)} \right) d\Omega \end{aligned} \quad \approx 0$$

For 4 noded, linear tetrahedron elements, we may approximate u_i^{nn} and $\frac{\partial u_j}{\partial x_i}^{nn}$ as,

$$u_i^{nn} = N_1 \tilde{u}_{i_1}^{nn} + N_2 \tilde{u}_{i_2}^{nn} + N_3 \tilde{u}_{i_3}^{nn} + N_4 \tilde{u}_{i_4}^{nn} = \mathbf{N} \tilde{\mathbf{u}}_i^{nn} \tag{2.43}$$

$$\frac{\partial u_j}{\partial x_i}^{nn} = \frac{\partial N_1}{\partial x_i}^{nn} \tilde{u}_{j_1} + \frac{\partial N_2}{\partial x_i}^{nn} \tilde{u}_{j_2} + \frac{\partial N_3}{\partial x_i}^{nn} \tilde{u}_{j_3} + \frac{\partial N_4}{\partial x_i}^{nn} \tilde{u}_{j_4} = \frac{\partial \mathbf{N}}{\partial x_i} \tilde{\mathbf{u}}_\mathbf{j}^{\mathbf{nn}} \tag{2.44}$$

and therefore $Term10$ takes the following form,

$$Term10 = \int_{\Omega} \hat{\mathbf{N}}^{\mathbf{T}} \left[(\hat{\mathbf{N}}\tilde{\mathbf{u}}_i^{nn}) \left(\frac{\partial \hat{\mathbf{N}}}{\partial x_i} \tilde{\mathbf{u}}_j^{nn} \right) + (\hat{\mathbf{N}}\tilde{\mathbf{u}}_i)^{nn} \left(\frac{\partial \hat{\mathbf{N}}}{\partial x_i} \delta\tilde{\mathbf{u}}_j \right) + (\hat{\mathbf{N}}\delta\tilde{\mathbf{u}}_i) \left(\frac{\partial \hat{\mathbf{N}}}{\partial x_i} \tilde{\mathbf{u}}_j^{nn} \right) \right] d\Omega \tag{2.45}$$

Finally, term10 takes the following fully discrete form (the superscript ~ is dropped for convenience),

$$Term10^{1} = \frac{V}{20}[\mathbf{M}]\{\mathbf{u}_i\}^{nn}\{\mathbf{D_i}\}\{\mathbf{u}_j\}^{nn} + \frac{V}{20}[\mathbf{M}]\{\mathbf{u}_i\}^{nn}\{\mathbf{D_i}\}\{\delta\mathbf{u}_j\} + \frac{V}{20}\{\mathbf{D_i}\}\{\mathbf{u}_j^{nn}\}[\mathbf{M}]\{\delta\mathbf{u}_i\} \tag{2.46}$$

where, $[\mathbf{M}] = \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix}$ and $\mathbf{D_i} = \left\{ \frac{\partial N_1}{\partial x_i} \quad \frac{\partial N_2}{\partial x_i} \quad \frac{\partial N_3}{\partial x_i} \quad \frac{\partial N_4}{\partial x_i} \right\}$

Following a similar treatment for the rest of the terms, the 3 steps of the CBS scheme my be written in the fully discrete form as,

---

step 1

$$\begin{aligned} &\frac{V}{20(\Delta t)}[M]\{\Delta\hat{u}_j^{\star}\} + \frac{V}{20}[M]\{\hat{u}_i\}^{nn}\{D_i\}\{\delta\hat{u}_j\} + \frac{V}{20}\{D_i\}\{\hat{u}_j^{nn}\}[M]\{\delta\hat{u}_i\} + \\ &\frac{\mu V}{\rho}[H_{ii}]\{\delta\hat{u}_j\} + \frac{\Delta t}{2\rho}\frac{V}{20}\{A\hat{U}X\}_k^{nn}\{\hat{u}_i\}^{nn}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\delta\hat{u}_j\} + \\ &\frac{\Delta t}{2\rho}\frac{V}{20}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\hat{u}_j\}^{nn}\{A\hat{U}X\}_k^{nn}\{\delta\hat{u}_i\} + \frac{\Delta t}{2\rho}\frac{V}{20}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\hat{u}_j\}^{nn}\{A\hat{U}X\}_i^{nn}\{\delta\hat{u}_k\} = \\ &-\frac{V}{20}[M]\{\hat{u}_i\}^{nn}\{D_i\}\{\hat{u}_j\}^{nn} - \frac{\mu V}{\rho}[H_{ii}]\{\hat{u}_j\}^{nn} - \frac{\Delta t}{2\rho}\frac{V}{20}\{A\hat{U}X\}_k^{nn}\{\hat{u}_i\}^{nn}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\hat{u}_j\}^{nn} \end{aligned} \tag{2.47}$$

[1] variables enclosed within [.], represent matrices, while those within {.}, represent vectors.

step 2

$$\frac{V}{20\Delta tc^2}[M]\{\Delta\hat{p}\} + \frac{V\rho}{4}[ND_i]\{\hat{u}_i^*\} + V\Delta t[H_{ii}]\{\delta\hat{p}\}^{nn} = -V\Delta t[H_{ii}]\{\hat{p}\}^{nn} \quad (2.48)$$

step 3

$$\frac{V}{20}[M]\{\delta\hat{u}_j\} - \frac{V}{20}[M]\{\hat{u}_j^*\} - \frac{\Delta t}{\rho}\frac{V}{4}[D_jN]\{\delta\hat{p}\} \approx -\frac{V}{20}[M]\{\hat{u}_j\}^{nn} + \frac{\Delta t}{\rho}\frac{V}{4}[D_jN]\{\hat{p}^{nn}\} \quad (2.49)$$

---

where $AUX = \begin{Bmatrix} 2u_{k_1} + u_{k_2} + u_{k_3} + u_{k_4} \\ u_{k_1} + 2u_{k_2} + u_{k_3} + u_{k_4} \\ u_{k_1} + u_{k_2} + 2u_{k_3} + u_{k_4} \\ u_{k_1} + u_{k_2} + u_{k_3} + 2u_{k_4} \end{Bmatrix}^{nn^T}$

The index-free, fully expanded form of the above equation will be presented in the next section (2.4) for the monolithic case. By removing the pressure terms from the monolithic case, the fully expanded form of step1 can be obtained. For steps 2 and 3, a similar expansion procedure may be used. It must be noted that the system matrix arising from Equation (2.47), is actually the jacobian matrix that can be directly used by the nonlinear solver.

2.4 Monolithic CBS scheme

The three steps of the actual characteristic based split scheme may be recombined suitably to establish its monolithic counterpart. In monolithizing the split CBS scheme, a pressure stabilization term is automatically established. This pressure stabilization term augments the mass-conservation equation and thereby prevents the zero diagonal pressure block in the system matrix. By substituting step3 into steps 1 and 2, the momentum and mass conservations equations, respectively,

may be recovered. The final form is presented below

X momentum equation,

$$\frac{V}{20}\{D_1\}\{\hat{u}_1^{nn}\}[M]\{\delta\hat{u}_1\} + \frac{V}{20}\{D_2\}\{\hat{u}_1^{nn}\}[M]\{\delta\hat{u}_2\} + \frac{V}{20}\{D_3\}\{\hat{u}_1^{nn}\}[M]\{\delta\hat{u}_3\} +$$
$$\left(\frac{V}{20\Delta t}[M] + \frac{V}{20}[M]\left(\{\hat{u}_1\}^{nn}\{D_1\} + \{\hat{u}_2\}^{nn}\{D_2\} + \{\hat{u}_3\}^{nn}\{D_3\}\right)\right)\{\delta\hat{u}_1\} +$$
$$\frac{\mu V}{\rho}[H_{11} + H_{22} + H_{33}]\{\delta\hat{u}_1\} + \frac{V}{4\rho}[ND_1]\{\delta\hat{p}\} \approx -\frac{V}{20\Delta t}[M]\{\hat{u}_1\}^{nn} - \frac{V}{4\rho}[ND_1]\{\hat{p}^{nn}\} -$$
$$\frac{V}{20}[M]\left(\{\hat{u}_1\}^{nn}\{D_1\} + \{\hat{u}_2\}^{nn}\{D_2\} + \{\hat{u}_3\}^{nn}\{D_3\}\right)\{\hat{u}_1\}^{nn} -$$
$$\frac{\mu V}{\rho}[H_{11} + H_{22} + H_{33}]\{\hat{u}_1\}^{nn} + \frac{V}{20(\Delta t)}[M]\{\hat{u}_1^n\} \tag{2.50}$$

i.e.

$$\alpha_1\{\delta\hat{u}_1\} + \alpha_2\{\delta\hat{u}_2\} + \alpha_3\{\delta\hat{u}_3\} + \beta\{\delta\hat{p}\} = \gamma \tag{2.51}$$

Y momentum equation,

$$\frac{V}{20}\{D_1\}\{\hat{u}_2^{nn}\}[M]\{\delta\hat{u}_1\} + \frac{V}{20}\{D_2\}\{\hat{u}_2^{nn}\}[M]\{\delta\hat{u}_2\} + \frac{V}{20}\{D_3\}\{\hat{u}_2^{nn}\}[M]\{\delta\hat{u}_3\} +$$
$$\left(\frac{V}{20\Delta t}[M] + \frac{V}{20}[M]\left(\{\hat{u}_1\}^{nn}\{D_1\} + \{\hat{u}_2\}^{nn}\{D_2\} + \{\hat{u}_3\}^{nn}\{D_3\}\right)\right)\{\delta\hat{u}_2\} +$$
$$\frac{\mu V}{\rho}[H_{11} + H_{22} + H_{33}]\{\delta\hat{u}_2\} + \frac{V}{4\rho}[ND_2]\{\delta\hat{p}\} \approx -\frac{V}{20\Delta t}[M]\{\hat{u}_2\}^{nn} - \frac{V}{4\rho}[ND_2]\{\hat{p}^{nn}\} -$$
$$\frac{V}{20}[M]\left(\{\hat{u}_1\}^{nn}\{D_1\} + \{\hat{u}_2\}^{nn}\{D_2\} + \{\hat{u}_3\}^{nn}\{D_3\}\right)\{\hat{u}_2\}^{nn} -$$
$$\frac{\mu V}{\rho}[H_{11} + H_{22} + H_{33}]\{\hat{u}_2\}^{nn} + \frac{V}{20(\Delta t)}[M]\{\hat{u}_2^n\} \tag{2.52}$$

i.e.

$$\delta_1\{\delta\hat{u}_1\} + \delta_2\{\delta\hat{u}_2\} + \delta_3\{\delta\hat{u}_3\} + \epsilon\{\delta\hat{p}\} = \zeta \tag{2.53}$$

Z momentum equation,

$$\frac{V}{20}\{D_1\}\{\hat{u}_3^{nn}\}[M]\{\delta\hat{u}_1\} + \frac{V}{20}\{D_2\}\{\hat{u}_3^{nn}\}[M]\{\delta\hat{u}_2\} + \frac{V}{20}\{D_3\}\{\hat{u}_3^{nn}\}[M]\{\delta\hat{u}_3\} +$$

$$\left(\frac{V}{20\Delta t}[M] + \frac{V}{20}[M]\left(\{\hat{u}_1\}^{nn}\{D_1\} + \{\hat{u}_2\}^{nn}\{D_2\} + \{\hat{u}_3\}^{nn}\{D_3\}\right)\right)\{\delta\hat{u}_3\} +$$
$$\frac{\mu V}{\rho}[H_{11} + H_{22} + H_{33}]\{\delta\hat{u}_3\} + \frac{V}{4\rho}[ND_3]\{\delta\hat{p}\} \approx -\frac{V}{20\Delta t}[M]\{\hat{u}_3\}^{nn} - \frac{V}{4\rho}[ND_3]\{\hat{p}^{nn}\} -$$
$$\frac{V}{20}[M]\left(\{\hat{u}_1\}^{nn}\{D_1\} + \{\hat{u}_2\}^{nn}\{D_2\} + \{\hat{u}_3\}^{nn}\{D_3\}\right)\{\hat{u}_3\}^{nn} -$$
$$\frac{\mu V}{\rho}[H_{11} + H_{22} + H_{33}]\{\hat{u}_3\}^{nn} + \frac{V}{20(\Delta t)}[M]\{\hat{u}_3^n\} \tag{2.54}$$

i.e.

$$\eta_1\{\delta\hat{u}_1\} + \eta_2\{\delta\hat{u}_2\} + \eta_3\{\delta\hat{u}_3\} + \theta\{\delta\hat{p}\} = \iota \tag{2.55}$$

Mass conservation (augmented automatically),

$$\frac{V\rho}{4}[ND_1]\{\delta\hat{u}_1\} + \frac{V\rho}{4}[ND_2]\{\delta\hat{u}_2\} + \frac{V\rho}{4}[ND_3]\{\delta\hat{u}_3\} + V\Delta t[H_{11} + H_{22} + H_{33}]\{\delta\hat{p}\} -$$
$$\frac{5\Delta t V}{4}\left([ND_1][M^{-1}][D_1N] + [ND_2][M^{-1}][D_2N] + [ND_3][M^{-1}][D_3N]\right)\{\delta\hat{p}\} \approx$$
$$\frac{5\Delta t V}{4}\left([ND_1][M^{-1}][D_1N] + [ND_2][M^{-1}][D_2N] + [ND_3][M^{-1}][D_3N]\right)\{\hat{p}\}^{nn} -$$
$$V\Delta t[H_{11} + H_{22} + H_{33}]\{\hat{p}\}^{nn} - \frac{V\rho}{4}[ND_1]\{\hat{u}_1\}^{nn} - \frac{V\rho}{4}[ND_2]\{\hat{u}_2\}^{nn} - \frac{V\rho}{4}[ND_3]\{\hat{u}_3\}^{nn} \tag{2.56}$$

i.e.

$$\kappa\{\hat{u}_1\}^{n+1} + \lambda\{\hat{u}_2\}^{n+1} + \mu\{\hat{u}_3\}^{n+1} + \nu\{\hat{p}\}^{n+1} = \xi \tag{2.57}$$

In block matrix form, a single element system may be represented as,

$$\begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \beta \\ \delta_1 & \delta_2 & \delta_3 & \epsilon \\ \eta_1 & \eta_2 & \eta_3 & \theta \\ \kappa & \lambda & \mu & \nu \end{bmatrix} \begin{Bmatrix} \delta\hat{u}_1 \\ \delta\hat{u}_2 \\ \delta\hat{u}_3 \\ \delta\hat{p} \end{Bmatrix} = \begin{Bmatrix} \gamma \\ \zeta \\ \iota \\ \xi \end{Bmatrix} \tag{2.58}$$

For linear tetrahedron elements, every block-matrix entry is a sub-matrix of size (4×4) and every vector entry is a vector of size (4×1). The degrees of freedom are trivially arranged one after the other, on a nodal basis.

2.5 Summary

Starting off with the Navier-Stokes equations, this chapter derived their fully discrete forms both in monolithic and split frameworks, as obtained from the characteristic based split algorithm. The time discretization was presented first for a simplified form of Navier-Stokes equations using the characteristic Galerkin scheme in explicit, semi-implicit and fully implicit forms. The convective stabilization term was derived in the process. The semi-discrete form was then extended to three dimensions and consequently to the momentum equation. Next, the splitting of pressure from the momentum equation as prescribed by the CBS scheme was performed in a 3 step procedure, which includes solving for intermediate velocity fields by removing pressure terms from the momentum conservation equation, pressure-poisson solve and correcting the intermediate velocity fields (in the same order). Each of the terms in each of the three steps were then considered separately and discretized spatially using the standard Galerkin procedure. The linearization procedure used for the non-linear convective and convective stabilization terms was presented next. As a result an analytical jacobian matrix was derived, which could be readily used in the Newton-like methods to solve the resulting non-linear algebraic set of equations. A monolithic version of the CBS scheme was presented next, which results in an additional term in the mass conservation equation. This additional term acts as a pressure stabilization and renders the possibility of using equal order interpolations for pressure and velocity. Some details of the derivation have been moved to Appendix C.

Chapter 3

Computational Framework

3.1 Introduction

This chapter deals with the computer implementation of the discretized Navier Stokes equations derived in chapter 2. A pre-processing-enabled Fortran90 code was written from scratch and parallelized using the MPI standard. The parallel support needed for the use of pre-programmed linear solvers (e.g. Conjugate Gradient, Biconjugate gradient, Generalized Minimal Residual Method, etc.) was realized by interfacing Intel® Math Kernel Library (IMKL) [2] and Portable Extensible Toolkit for Scientific Computation (PETSc) [11] with the code developed for this research. The structure of the code, libraries used and the additional steps that inherently need to be performed in a parallel framework will be described in this chapter. A sample parallel code each for MPI and PETSc are also presented in this chapter. These are specifically written to be included here as they elegantly demonstrate the usage of MPI and PETSc to establish the foundation for the development of a parallel FE application in a succinct and encouraging manner.

In the most simplest terms, the Fortran90 code houses a nesting of a loop over

elements followed by an iterative linear/non-linear solve, within a loop over time, all in a parallel framework. The loop over elements constructs the element level matrices and vectors and assembles them suitably into the global matrix and vector. The global matrix and vector constitute the left hand side (LHS) matrix and right hand side (RHS) vector of a linear system, which the iterative solver attempts to solve using iterative methods belonging to the Krylov family [77], along with a suitable preconditioner. The loop over time helps solve transient problems.

To conveniently test various schemes and strategies in the same code, the fortran preprocessor, "fpp" was also used. Conditional compilation without the overhead of invoking traditional "if statement" in a nested-loop environment was an important motivation for the incorporation of a compiler preprocessor.

IMKL was only used in the very early versions of the code to solve linear systems using the preconditioned GMRES [133] method. Although, the IMKL implementation was parallel, it was limited to threading using OpenMP, i.e. it was able to only utilize multiple cores available in a single processor, but couldn't parallelize across several processors, like that supported in an MPI environment. Since the primary problems of interest for this research were expected to involve the use of high definition meshes with millions of elements, multi processor parallelism was essential and consequently the use of IMKL was discontinued.

3.2 Parallelization

The primary motivators for code parallelization were speed and memory. Since this research was expected to use large patient specific meshes with millions of degrees of freedom, the run time and memory requirements would make the computations infeasible with traditional serial computers.

Parallelization is a vast area. Not only can one find several different types of parallel computers but also several types of classifications for parallel computers. Some common classifications are based on instruction and data stream, structure of computers, memory access and grain size. For a thorough introduction to parallel computers, the reader is directed to standard texts on this subject [87, 90].

The Single Program Multiple Data (SPMD) programming style is used for the code developed in this research. SPMD is a common style for message-passing programming on distributed memory computer architectures. In SPMD, all the participating processors receive the same copy of the executable, but operate on a different data set. Despite receiving the same code, individual processors can perform dissimilar tasks, if necessary. This capability requires generic task allocation based on the processor ranks/IDs.

For conventional serial programs, the coding process may be perceived as writing along with a simultaneous mental simulation of the effects of the statements being written. The coder constantly thinks from the processor's viewpoint and writes code that performs the actions dictated by the algorithm. This is eventually achieved by a sequence of syntactical characters that encompass the body of the code. Every line in the code executes one after the other, in an orderly fashion. However, when writing parallel MPI codes, the situation is more complicated. Depending on the per-processor-load, every processor invariably executes a different line of the code and completes its task in a certain wall-time, dissimilar to other processors. Some sort of synchronization, either implicit or explicit is essential for the problems of interest for this research. Since all the processors get a copy of the same code, data must be suitably arranged/distributed and must be accessed with generic variable names and indices. The sample MPI code enclosed in section 3.2.1.1 will attempt to illustrate this.

3.2.1 MPI

MPI is not a library/implementation/language. It is a set of *specifications* that prospective message-passing library interface developers may adhere to. The collaborative efforts of 40 American and European organizations resulted in the MPI standard. MPI started off in 1992 as being a conglomeration of the attractive features of a number of existing message passing systems. Over the years, the MPI standard incorporated new types of functionalities and is being widely used. A number of MPI implementations are currently available (e.g. MPICH, winmpich, HP MPI, IBM's MPI, Intel MPI, etc.). This research work uses the implementation from Intel® Corporation. The latest version is the MPI-3.0, which is a major update to the MPI standard.

Right from the start, the developers of the MPI standard aimed at allowing easy integration with programs written in Fortran and C, thus making the use of MPI in this research work, not very atypical. There are around 370 subroutines in a typical MPI implementation, although the exact numbers are specific to the implementation in question. Generally, an MPI code uses/needs far less than the total number of subroutines available. Simply stated, these MPI subroutines give a coder the ability to simultaneously employ a group of communicating processors to perform a collection of tasks.

3.2.1.1 Sample MPI code

A simple MPI code is presented in this section with a level of complexity to illustrate basic inter-processor communications in an MPI environment. This sample code is presented with the aim of illustrating the basic framework that may be used in a parallel Finite Element application.

```
program thesis_sample_mpi
implicit none
include 'mpif.h'

integer,parameter  :: charlen=20,msglen=11
integer            :: myrank,numProcs,ierr,unitno,tag,i
integer            :: status(MPI_STATUS_SIZE),errorcode
integer            :: source,dest
character(charlen) :: myrank_char,my_character,i_char
character(charlen) :: base_name,proc_specific_name,ext
character(LEN=1)   :: result_basket(msglen),msg

call MPI_INIT(ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD, numProcs, ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD, myrank, ierr)
if (myrank.eq.0)then
    write(*,*)'Starting program'
endif !myrank
if (numProcs.ne.msglen+1)then
    write(*,*)'Incorrect number of processors used!'
    write(*,*)'Rerun with 12 processors'
    call MPI_ABORT(MPI_COMM_WORLD,errorcode,ierr)
endif !numProcs
write(myrank_char,'(I)')myrank+1
myrank_char = adjustl(myrank_char)
! Every processor reads a suitable data file.
dest = numProcs - 1
if (myrank.ne.dest)then
   base_name = 'sample_mpi'
   ext = '.dat'
   proc_specific_name = trim(base_name)//'_'// &
                        trim(myrank_char)//trim(ext)
   unitno = 10 + myrank + (1*numProcs)
   open(unitno,file=proc_specific_name,status='old')
   read(unitno,*)my_character
endif !myrank
if (myrank.eq.0)then
    result_basket(:) = 'X'
    write(*,*)'result_basket=',result_basket
endif !myrank
call MPI_BCAST(result_basket,msglen,MPI_CHARACTER,0, &
                                    MPI_COMM_WORLD,ierr)
! Prepare the message to send
```

```
msg = trim(my_character)
if (myrank.eq.dest)then
    do i = 1,numProcs-1
       ! Receive message
       call MPI_RECV(result_basket(i), 1, MPI_CHARACTER, &
                     i-1, 1, MPI_COMM_WORLD, status, ierr)
       write(i_char,'(I)')i
       i_char = adjustl(i_char)
       write(*,*)'[P12]','ReceivedMessageFrom[P',trim(i_char),']'
       ! Track progress
       write(*,*)'result_basket=',result_basket
    enddo !i
endif !myrank
if (myrank.ne.dest)then
    ! Send message
    call MPI_SEND(trim(msg), 1, MPI_CHARACTER, dest, 1, &
                                        MPI_COMM_WORLD, ierr)
    write(*,*)'[P',trim(myrank_char),'] sent message:', &
                                         trim(msg),' to [P12]'
endif
! Synchronization
call MPI_BARRIER(MPI_COMM_WORLD,ierr)
if (myrank.eq.0)then
    write(*,*)'Ending program'
endif !myrank
call MPI_FINALIZE(ierr)
end program thesis_sample_mpi
```

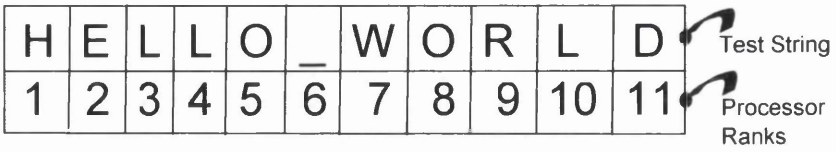


Figure 3-1: Character-Processor mapping for the MPI sample program

The sample code enclosed above makes use of 9 MPI subroutines and is expected to run in a parallel MPI environment on 12 processors. This code simply generates a string that reads *HELLO_WORLD*. However, the string is assembled non-trivially in parallel to illustrate the participation of various processors. Figure 3-1 presents the processor-alphabet mapping. The use of 12 processors for

such a simple task is unnecessary and is intended here for purely illustrational purposes. Processors 1-11 read a processor-specific file. Each of these files contain an alphabet of the *HELLO_WORLD* string. After every processor has its string (i.e. message to communicate), the message is sent to the 12^{th} processor which acts as a container for the final string being constructed.

3.2.1.2 Sample code output

The output enclosed below shows how the assembly process progresses to completion.

```
Starting program
result_basket=XXXXXXXXXXX
[P1] sent message:H to [P12]
[P2] sent message:E to [P12]
[P3] sent message:L to [P12]
[P4] sent message:L to [P12]
[P5] sent message:O to [P12]
[P6] sent message:_ to [P12]
[P7] sent message:W to [P12]
[P8] sent message:O to [P12]
[P9] sent message:R to [P12]
[P10] sent message:L to [P12]
[P11] sent message:D to [P12]
[P12]received message from [P1]
result_basket=HXXXXXXXXXX
[P12]received message from [P2]
result_basket=HEXXXXXXXXX
[P12]received message from [P3]
result_basket=HELXXXXXXXX
[P12]received message from [P4]
result_basket=HELLXXXXXXX
[P12]received message from [P5]
result_basket=HELLOXXXXXX
[P12]received message from [P6]
result_basket=HELLO_XXXXX
[P12]received message from [P7]
result_basket=HELLO_WXXXX
[P12]received message from [P8]
result_basket=HELLO_WOXXX
[P12]received message from [P9]
result_basket=HELLO_WORXX
[P12]received message from [P10]
result_basket=HELLO_WORLX
[P12]received message from [P11]
result_basket=HELLO_WORLD
Ending program
```

A detailed explanation of the sample program is provided in Appendix A.

Although a message passing library is sufficient to parallelize a code, it is not sufficient to solve the linear systems arising as a result of the discretization of Navier-Stokes equations. Several libraries specialize in providing parallel iterative solvers and pre-conditioners, but two of the most powerful ones are PETSc (Argonne National Laboratory, IL, USA) and Trilinos (Sandia National Laboratory, USA). It is often worth the effort to spend time in learning to use these non-trivial libraries that to program the solvers and pre-conditioners (especially in parallel).

3.2.2 PETSc

PETSc is an open source suite of data structures and routines for parallel solution of large-scale scientific applications modeled by partial differential equations. It supports MPI, shared memory pthreads, and GPUs[1] through CUDA[2] or OpenCL[3], as well as hybrid MPI-shared memory pthreads or MPI-GPU parallelism. PETSc provides interfaces for programs written in Fortran, C, C++ and python.

PETSc is currently run by a group of around 12 very enthusiastic and motivated scientists. Even with such a limited number of active developers a new version/update is rolled out several times in a year. The current version of PETSc is 3.5, released in June 2014. It is worth noting the excellent support provided by this dedicated team. Almost all support queries are dealt with, in the same hour at no charge! Also, extensive documentation is available, both in the form of a user manual and online manual pages. Using PETSc is far more complicated and involved when compared to other conventional libraries. It by no means is a plug

[1]Graphics Processing Unit
[2]Compute Unified Device Architecture
[3]Open Computing Language

and play library from the user's perspective. Although the number of subroutines is ballooning, it is in the range of a few thousand. Having access to quick and direct support under these conditions is highly valuable.

The PETSc team employ and encourage the "use as much as you like" ideology. This lets users control the extent of PETSc's involvement in their application programs. In this work, PETSc is purely used to solve the linear and non-linear systems arising from the discretized Navier-Stokes equations. Although, the usage seems minimal, the consequences of using PETSc penetrate several layers upwards into the code and considerably change its overall structure. This makes PETSc ideally suited to be employed in the early phases of application program development. Although possible, incorporating PETSc in an existing code is rather cumbersome.

3.2.2.1 Building blocks of PETSc

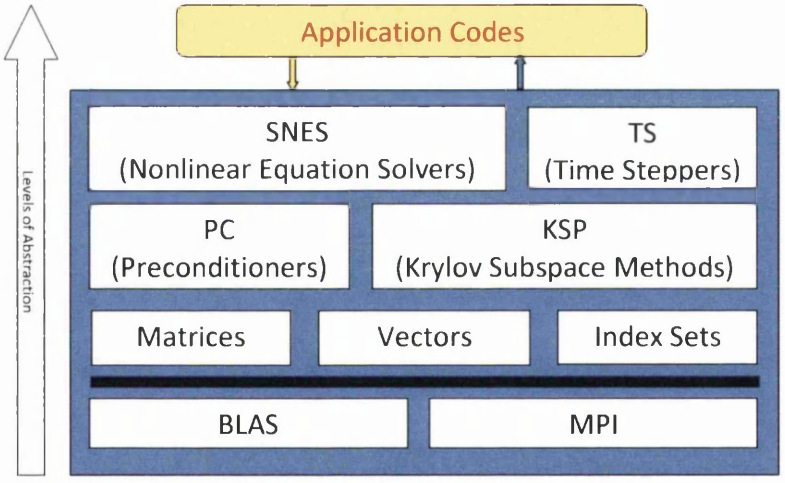


Figure 3-2: *Hierarchical organisation of PETSc libraries*

Figure 3-2 shows the libraries present in PETSc. Each library consists of an abstract interface, which is a set of calling sequences with a specific set of arguments.

A brief description of all the libraries is presented below.

Vector, denoted by Vec, is one of the simplest objects in PETSc. As the name suggests, it is used to store the solutions and RHS (Right Hand Side) of linear systems. These vectors may be sequential or parallel. Although specific subroutines are provided to create both sequential (VecCreateSeq(args)) and parallel (VecCreateMPI(args)) vectors, it is considered a good practise to use the generic vector generation subroutine - VecCreate(args). Depending on the number of processors employed, the VecCreate subroutine can automatically generate the required type of vector. This simple subroutine selection criterion may also be extended to other objects in PETSc. Another benefit of using generic object generation subroutines is the ability to explicitly control their behaviour by using suitable options in the PETSc options database. Since the options database file is external to the source code, different settings may be tested without having to recompile the entire code. Just for the Vec object alone around 250 subroutines are available. This gives an idea about the scale and extent of PETSc.

Index Set, denoted by IS, is a set of indexing integers used to define scatters, gathers and similar operations on vectors and matrices. Scatters and gathers refer to operations where a specific subset of a vector is either selected for insertion or to update/add to a subset of another vector. Although, ISs are useful for problems involving unstructured meshes, they haven't been currently employed in the code developed in this research. PETSc provides around 170 IS related subroutines.

Matrix, denoted by Mat, provides a variety of matrix implementations to cater for a wide range of applications. Sequential and parallel versions of both dense and sparse matrices are provided. The default matrix representation within PETSc is the AIJ format (Yale Sparse Matrix format or Compressed Sparse Row (CSR) format). The Mat objects are used to store the Jacobian and system matrices arising while solving non-linear and linear systems. In order to efficiently use

this object for systems with large number of overall degrees of freedom, memory preallocation is of paramount importance. The matrix assembly performance can be increased by more than a factor of 50 if correct preallocation data is specified. The preallocation data essentially consists of the number of non-zero entries occurring in the matrix, both in the diagonal and the off-diagonal blocks, for each row of the matrix owned by every processor. PETSc currently provides around 460 Mat specific subroutines.

KSP, represents the scalable-linear-equation-solvers component available in PETSc to access parallel and sequential, direct and iterative solvers for non-singular systems of the form $[A]\{x\} = \{b\}$. Some of the methods available under KSP are Richardson, Chebyshev, Conjugate Gradient, BiConjugate Gradient, Generalized Minimal Residual, Generalized Conjugate Residual, BiConjugate Gradient Stabilized, Conjugate Gradient Squared, Transpose-Free Quasi-Minimal Residual, Conjugate Residual and Least Squares. The elegance of this library lies in the fact that the same code, without recompiling, can be used to test each of the methods listed above, with just a change in the options database file. Standard convergence monitoring is provided for all methods. However, if there is a need for a special convergence monitoring test to be included, PETSc provides for subroutines which can invoke a user-defined monitoring routine and hence alter the behaviour of the method as per the new test. There are currently 252 KSP subroutines in the PETSc toolkit.

PC, provides access to a variety of preconditioners, which are typically used to accelerate the convergence rate of iterative methods. All KSP implementations available in PETSc default to left preconditioning. Using suitable options in the options database file of PETSc, right preconditioning may be activated for some methods. Preconditioners like Jacobi, block Jacobi, SOR (Successive Over Relaxation Gauss Seidel), Incomplete LU, Incomplete Cholesky, Additive Schwarz and Algebraic Multigrid are available in PETSc. Some preconditioners are difficult

to use when compared to others. There are around 270 PC subroutines currently available. One can also use matrix element based preconditioners in the LLNL package hypre.

SNES, stands for Scalable Nonlinear Equation Solvers and provides access to various non-linear solvers within PETSc. These include Newton like methods which internally employ the Krylov solvers described earlier. There are around 300 SNES subroutines currently available. SNES may be used to create a generic framework to solve both linear and non-linear equations, which is helpful for developing general, multi purpose applications. Various methods like the line search and trust region Newton methods, non-linear Richardson, non-linear conjugate gradient, non-linear GMRES, etc are available within SNES. Typically the SNES solvers are capable of calculating the jacobian matrix using finite differences, but for large problems it is faster and efficient to provide a subroutine to evaluate the jacobian. One also needs to provide a subroutine to calculate the non-linear function. PETSc can invoke external user defined subroutines for the jacobian and function evaluations, as necessary during the solution process.

TS, provides access to frameworks for solving ODEs and DAEs that arise by virtue of discretization of the time dependent partial differential equations. Implict, Explicit and mixed implicit and explicit methods are available currently. Provisions for pseudo time stepping has also been implemented for steady state problems. Since, the primary motivation behind employing PETSc in this research was to gain access to parallel solvers and pre-conditioners, the TS library was not used.

3.2.2.2 Sample PETSc code

The sample code enclosed below, iteratively calculates the solution to a dense $4 \star 4$ linear system in parallel on 2 processors, using GMRES. Although simple,

this sample PETSc code is an attempt to provide a taster of PETSc. Some MPI subroutines will be used in the process, though it is not mandatory. The following linear system is solved in the sample PETSc code,

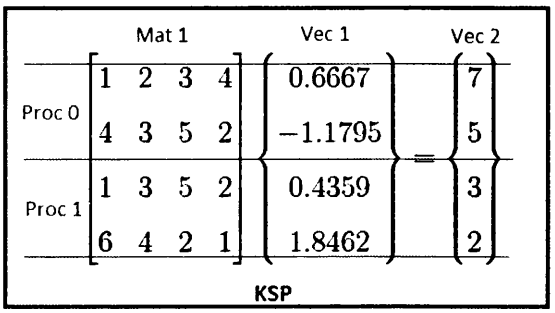


Figure 3-3: Sample linear system: Elements of KSP along with parallel object partitioning

```
program thesis_sample_petsc
implicit none
! PETSc specific include files
#include <finclude/petscsys.h>
#include <finclude/petscviewer.h>
#include <finclude/petscvec.h>
#include <finclude/petscvec.h90>
#include <finclude/petscmat.h>
#include <finclude/petscksp.h>
! PETSc data types
Mat             LHS
Vec             RHS,x
KSP             ksp
PetscErrorCode  ierr
PetscMPIInt     myrank,numProcs
! Fortran data types
integer,parameter :: nnodes=4
integer           :: i,partition_info(2,2),def1,idxm(4)
double precision  :: values(4)
! PETSc Initialization - Auto initialization of MPI
call PetscInitialize(PETSC_NULL_CHARACTER,ierr)
call MPI_COMM_RANK(PETSC_COMM_WORLD,myrank,ierr)
call MPI_COMM_SIZE(PETSC_COMM_WORLD,numProcs,ierr)
! Set matrix partition
partition_info(1,:) = (/1,2/);partition_info(2,:) = (/3,4/)
def1 = partition_info(myrank+1,2)-partition_info(myrank+1,1)+1
! Generate parallel matrix
```

```
28 call MatCreate(PETSC_COMM_WORLD,LHS,ierr)
29 call MatSetSizes(LHS,def1,def1,nnodes,nnodes,ierr)
30 call MatSetUp(LHS,ierr)
31 ! Generate parallel right hand side and solution vector
32 call VecCreate(PETSC_COMM_WORLD,RHS,ierr)
33 call VecSetSizes(RHS,def1,nnodes,ierr)
34 call VecSetUp(RHS,ierr)
35 call VecDuplicate(RHS,x,ierr)
36 ! Set dummy values in parallel matrix and vector
37 idxm = (/1,2,3,4/) ; idxm = idxm - 1
38 if (myrank.eq.0)then
39     values = (/1,2,3,4/)
40     call MatSetValues(LHS,1,0,4,idxm,values,INSERT_VALUES,ierr)
41     values = (/4,3,5,2/)
42     call MatSetValues(LHS,1,1,4,idxm,values,INSERT_VALUES,ierr)
43     values = (/7,5,3,2/)
44     call VecSetValues(RHS,2,idxm(1:2),values(1:2),&
45                                 INSERT_VALUES,ierr)
46 else
47     values = (/1,3,5,2/)
48     call MatSetValues(LHS,1,2,4,idxm,values,INSERT_VALUES,ierr)
49     values = (/6,4,2,1/)
50     call MatSetValues(LHS,1,3,4,idxm,values,INSERT_VALUES,ierr)
51     values = (/7,5,3,2/)
52     call VecSetValues(RHS,2,idxm(3:4),values(3:4),&
53                                 INSERT_VALUES,ierr)
54 endif !myrank
55 ! Parallel object assembly
56 call MatAssemblyBegin(LHS,MAT_FINAL_ASSEMBLY,ierr)
57 call MatAssemblyEnd(LHS,MAT_FINAL_ASSEMBLY,ierr)
58 call VecAssemblyBegin(RHS,ierr)
59 call VecAssemblyEnd(RHS,ierr)
60 ! Call krylov solver
61 call KSPCreate(PETSC_COMM_WORLD,ksp,ierr)
62 call KSPSetOperators(ksp,LHS,LHS,SAME_NONZERO_PATTERN,ierr)
63 call KSPSolve(ksp,RHS,x,ierr)
64 ! Display solution vector
65 call VecView(x,PETSC_VIEWER_STDOUT_WORLD,ierr)
66 ! Terminate Petsc
67 call PetscFinalize(ierr)
68 end program
```

Even for such a simple program around 20 PETSc subroutines had to be used, most of which are mandatory to use and many more become both mandatory and necessary when writing an entire finite element application. A brief explanation of the entire sample code is provided in Appendix B.

3.2.3 Domain decomposition

The basic paradigm in most parallel computing frameworks is that a large problem can be divided into smaller, simultaneously solvable sub problems. The process of suitably breaking down the problem into smaller parts or partitions is referred to as domain decomposition or partitioning. An application which performs this task is called a partitioner. The domain, in this case is represented by the mesh. The partitioner typically reads this mesh and splits it into the desired number of partitions, each representing a sub domain for the participating processors to assume ownership of, and process. It must also be mentioned that the use of a SPMD framework usually necessitates the partitioning of the global domain.

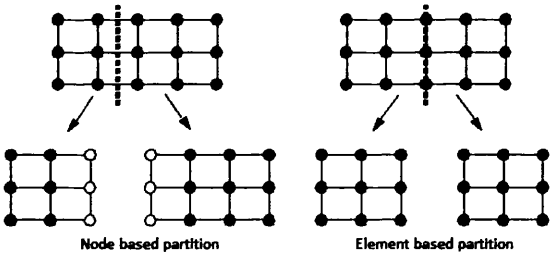


Figure 3-4: Types of mesh partitioning

Two different types of partitioning are possible. These are node based (edge cut) and element based (vertex cut). As illustrated in Figure 3-4, node based partitioning cuts the mesh across element faces, whereas the element based partitioning cuts the mesh along element faces. The element based partitioning is adopted in this research. This type of partitioning is also in line with the finite

element method, where the element level matrices are constructed in a loop and assembled into the global matrices. With the interface nodes [4] being duplicated across communicating processors, the finite element assembly operation may be viewed as the incorporation of contributions from various super elements. All elements contained in a certain processor/partition/patch are collectively referred to as a super element.

The initial choice for partitioning meshes was ParMETIS [81]. It is an MPI based parallel library written in C, with various implementations of partitioning algorithms for unstructured meshes. ParMETIS was the successor of the popular METIS library [80], which implemented multilevel partitioning and fill-reducing ordering algorithms.

For use in parallel frameworks, partitioners must typically ensure:

1. Load balancing: The sub domains generated by a partitioner must roughly contain equal number of elements. This ensures that all processors get evenly loaded, ensuring that they can complete their respective tasks in approximately the same length of time. An important implication of satisfying this condition is that the processors will have to spend little time while waiting for the slowest processor to complete.

2. Minimising communications: Processors communicate only when they do not have the required information. This typically occurs at the interface, along which the mesh will be cut. A partitioning scheme that can reduce the number of these interface nodes, therefore directly addresses the problem of reducing communication across participating processors.

Although, the above conditions were elegantly addressed by ParMETIS, another problem was encountered. Since, the code developed in this research used PETSc,

[4]Interface nodes: Nodes along which the mesh is cut in an element based strategy. Consequently, these nodes are duplicated across adjacent mesh partitions.

another type of partitioning - matrix partitioning, had to be addressed. In the PETSc framework, the matrix arising from the linear system is partitioned across participating processors in a row-wise manner. Although, users can specify the ownership limits for every processor, they must be contiguous. This might get addressed in future releases, but remains a constraint in the PETSc version 3.4.0 employed in this research. The contiguity constraint implies that the ParMETIS partitions could not be directly used within the PETSc framework, in an efficient manner. The inefficiency creeps from the fact that the majority of the element level matrix entries generated in a certain processor might get assembled within global matrix entries that belong to other processor(s), i.e. many off-processor assembly operations might occur. Such off-processor assembly operations add to the communication overhead. It is therefore necessary to also ensure that the partitioner decomposes the domain by minimising the number of off-processor assembly operations. This problem is illustrated in the following section.

3.2.3.1 Illustration of inconsistency between ParMETIS mesh partitions and PETSc matrix partitions

For the purpose of this illustration, a tetrahedral mesh representative of a cuboid, is considered. The mesh is assumed to be partitioned into 16 sub domains and has refinement towards one of the long faces. Assuming the standard approach, if one was to mesh with Gmsh and partition straight with ParMETIS, then the resulting inconsistency with respect to the PETSc matrix partitioning is illustrated in Figure 3-5. If we consider patch number 16 of the ParMETIS partition (right), and compare the corresponding region in the figure on the left showing the PETSc partition, it is obvious that the processor 16 will generate element level matrices that will need to be assembled in processors 2,3,8 and 9. It turns out that all assembly operations from processor 16, will be in other processors leading to communications overhead. The same is the case with almost all the remaining

processors.

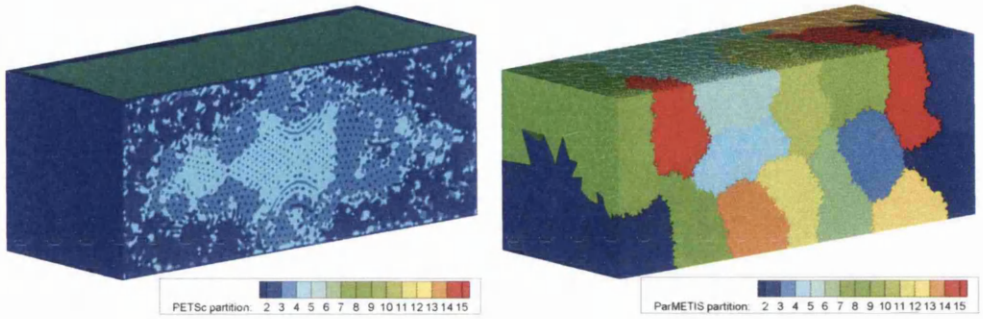


Figure 3-5: Left: PETSc matrix partition projected on the mesh. Right: ParMETIS partition

In Figure 3-6 various slices are extracted along the cuboid to reveal the presence of such inconsistency not just on the surface, but even on the inside of the cuboid.

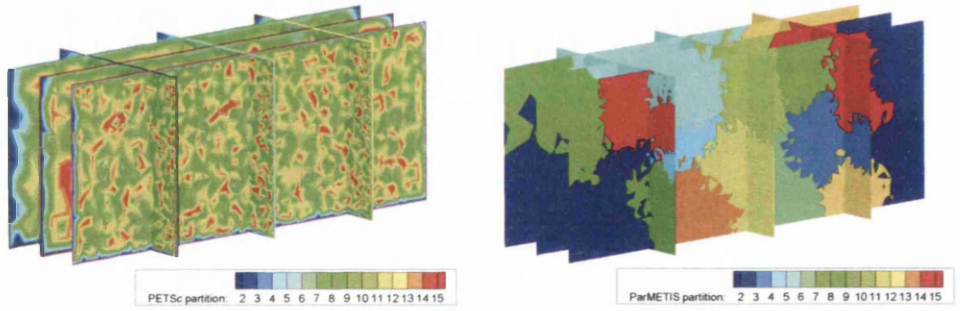


Figure 3-6: Slice extraction from various sections of the domain in Figure 3-5

The quality of the initial mesh may be improved by renumbering the mesh, so that the node numbers of interconnected nodes are close to each other. In other words, the random distribution of node numbers occurring in the first plot of Figure 3-6 can be corrected. This is also traditionally referred to as reducing the bandwidth of the matrix, which may be visualized by plotting the sparsity structure of the matrix. Techniques like Cuthill Mckee, Reverse Cuthill Mckee [91], Minimum degree permutation [5], etc. are available to reduce the matrix bandwidth. Figure 3-7 presents the results of partitioning, when the Reverse

Cuthill Mckee (RCM) renumbering scheme is adopted. Although, RCM helps cluster the nodes, the problem of inconsistency between the two partitions is not addressed.

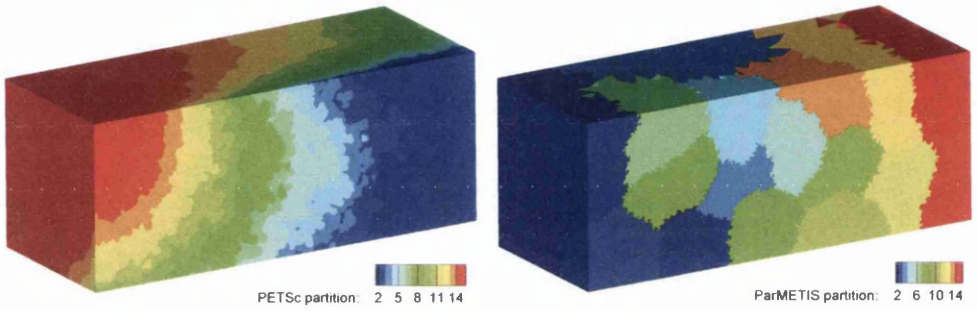


Figure 3-7: Comparison of PETSc and ParMETIS partition with RCM preprocessing

The benefit of using RCM renumbering becomes apparent by plotting the sparsity structure of the finite element system matrix. The sparsity structure pre and post renumbering are presented in Figure 3-8. The matrix bandwidth after RCM renumbering gets reduced by 92%. Such an elegantly renumbered mesh will inherently result in a reduction in the MPI communications by virtue of clustered entries along the diagonal. Also, diagonal dominance leads to better rates of convergence [67]. However, the inconsistency between the mesh and matrix partitioning still exists.

Another strategy was implemented with the aim of making the mesh and matrix partitions coherent. For this purpose, an MPI Fortran renumberer was written and was intended to operate on the mesh after partitioning it. A global list of interface nodes on all processors was first constructed and assigned to the first patch in the mesh, after renumbering them. With the interface nodes being dealt with, all the nodes that remained in the remaining patches would be under the exclusive ownership of the patch/processor in which they appear. This implies that a top-down approach could easily be employed to renumber the nodes such

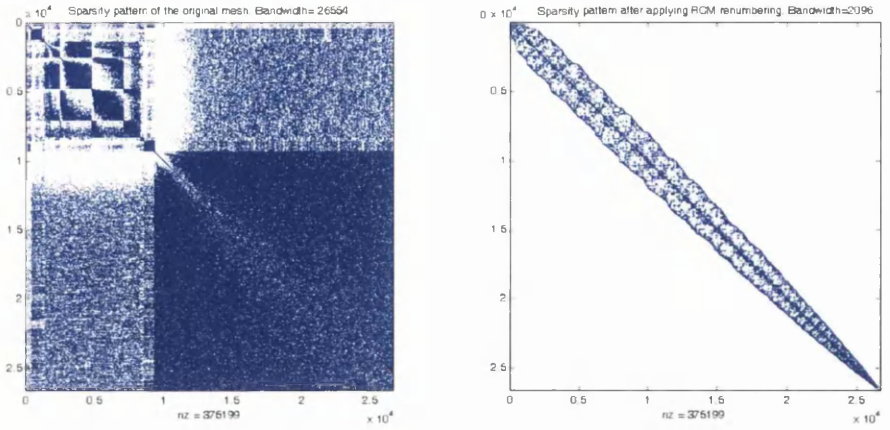


Figure 3-8: Sparsity pattern of the finite element system matrix. Left: Before RCM renumbering. Right: After RCM renumbering

that the consistency between mesh and matrix partitions could be achieved.

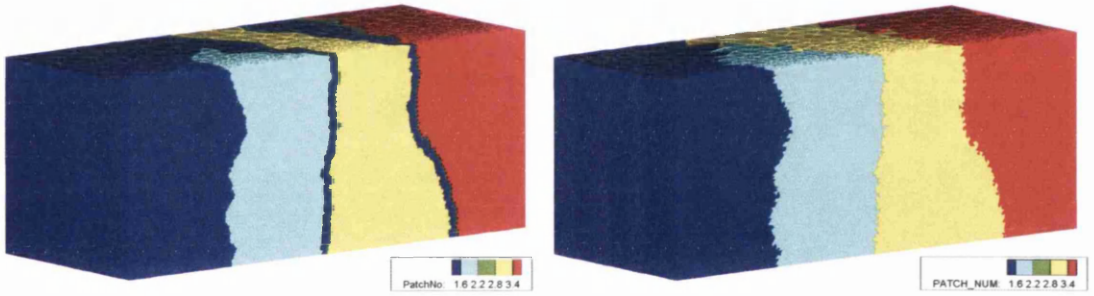


Figure 3-9: Partition renumbering scheme on 4 patches. (Left) PETSc partition (Right) ParMETIS partition

In other words, the range of nodes contained in a mesh-patch that gets assigned to a specific processor would be the same as the range of matrix rows owned by that processor. Assuming four partitions, the partition contours before and after partition-renumbering are presented in Figure 3-9. The box type sparsity pattern arising out of this approach is presented in Figure 3-10. Although, this approach (renumbering the mesh patches) appears to generate consistent partitions, the first processor is heavily loaded, since all interface nodes appear in it. This leads

to poor load balancing.

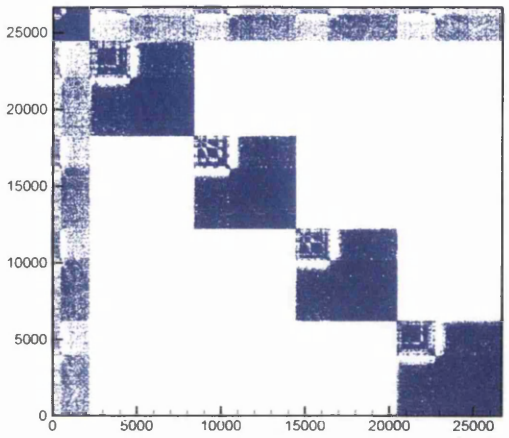


Figure 3-10: Sparsity pattern of the partitioned renumbered finite element system matrix

To circumvent the overloading of the first processor, it is possible to distribute the interface nodes equally amongst all processors. However, this will lead to a slightly more cumbersome renumbering strategy. Also, the matrix bandwidth is relatively larger in the partition-renumbering case when compared against RCM renumbering, leading to performance penalties.

3.2.3.2 Current working strategy

In order to amalgamate the benefits of RCM renumbering along with a partitioning scheme that permits coherence between mesh and matrix partitions, a custom partitioner was implemented serially. This partitioner first applies RCM renumbering on the un-partitioned mesh and then partitions the nodal block in a mesh, like that done in PETSc. As a result of partitioning the nodal block in this manner, mesh patches consistent with the matrix partitions are obtained. Next, the elements are partitioned. This is not trivially done, because the element ownership depends on the nodal ownership. The following rules apply when

partitioning the elements:

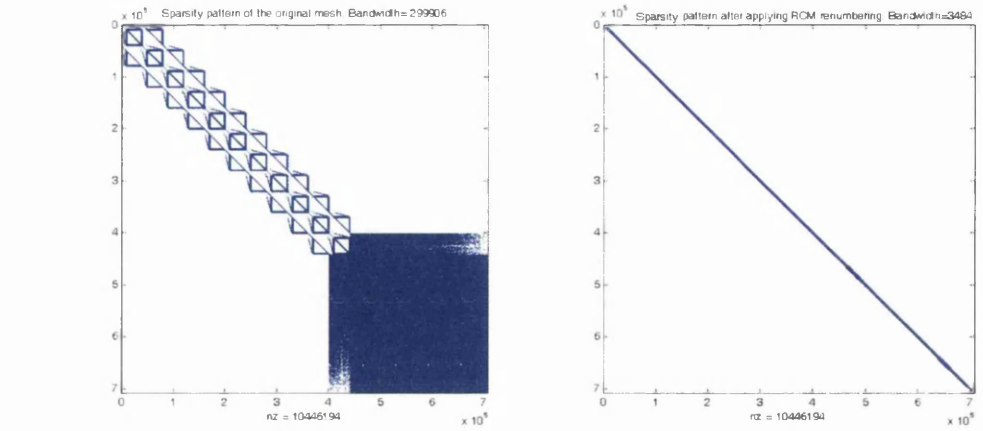
1. Pure ownership: An element with all of its nodes in a certain partition, belongs to that partition.

2. Biased shared ownership: An element with nodes belonging to multiple patches will get assigned to the patch in which majority of its nodes lie.

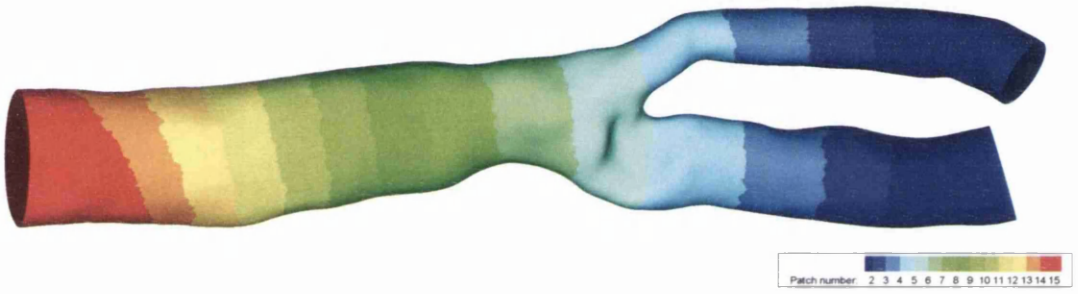
3. Unbiased shared ownership: An element whose nodes are equally shared across multiple patches gets assigned to either of the patches.

4. Poor ownership: An element whose every node is in a different processor, currently terminates the program, as this indicates excessive partitioning of the mesh concerned. If the partitioner runs before RCM renumbering, then most of the elements are expected to fall in this category and hence this check currently terminates the program.

While partitioning the elements in accordance with the above rules, a simultaneous record of nodes that need to be duplicated in various patches is made. This step becomes essential when dealing with elements whose nodes are under shared ownership (for completeness of information while constructing element level matrices). Finally, the boundary block is partitioned based on the element partitioning data, since every boundary element will be independently associated with an inside element. This completes the mesh partitioning process. The partitioned mesh is then written to processor specific files with the header (containing local mesh size), local connectivity information, local nodal coordinate information along with the duplicated nodes and the local boundary information, in the same order. For convenience, the node and element numbers appearing in the mesh patches are global, i.e. fully consistent with the un-partitioned mesh. The partitioning results for a Carotid mesh containing 16 partitions is presented in Figure 3-11.



(a)



(b)

Figure 3-11: Renumbering a carotid mesh. (a) Sparsity pattern before and after renumbering (b) Patch contours

3.2.4 PETSc matrix preallocation

Preallocating memory to the parallel matrices within PETSc is an efficiency measure that can increase the performance by a factor of more than 50 [41]. In the absence of preallocation data, PETSc defaults to dynamically allocating memory. This means that every time a new non-zero entry is encountered, all the entries are copied from the old location to a new, larger location. This becomes very expensive when dealing with large systems. In the early stages of code development, the CSR (Compressed Sparse Row) sparse matrix representation of the system matrix was generated to aid in preallocation (see appendix D). As per the

standard CSR representation, the difference of consecutive entries of the CSR-row array gives the exact number of non-zero matrix entries in every row of the matrix. Splitting this data further into the exact number of non-zeros appearing in the diagonal and off-diagonal blocks of every row in the matrix, results in an accurate preallocation. Cases where exact preallocation data is unavailable, inaccurate overestimation is cheaper than underestimation, the former involving significantly smaller or no data movement. The calculation of preallocation data for a small pseudo-sparse system is presented in Appendix D.

3.3 Code overview

A brief description of the main code developed in this research (IFENs) will be provided in this section. As mentioned before, this code is written in Fortran90 and parallelized using the MPI standard and PETSc. There are several ways of using PETSc in a Fortran code. Here one of the recommended approaches, called "Classic Fortran 90 style" is used. In this approach, various PETSc header files are included at the top of the program using the fortran preprocessor. Apart from the above mentioned use of the preprocessor (automatically invoked using the F90 file extension rather than the f90 extension), it was used to conditionally omit or include various sections of the code before compiling. This is beneficial from the point of view of making the code multi purpose, without having to invoke traditional if statements several times in a loop. Also, the code can be elegantly made portable to take care of platform specific source text.

Figure 3-12 gives an outline of the main code developed in this research. In the remaining part of this section some details of the code are provided, which by no means is exhaustive. The code starts off by defining variables that control the actions of the preprocessor. These variables are defined by what are called as 'preprocessor directives', that must begin with # in the first column of the

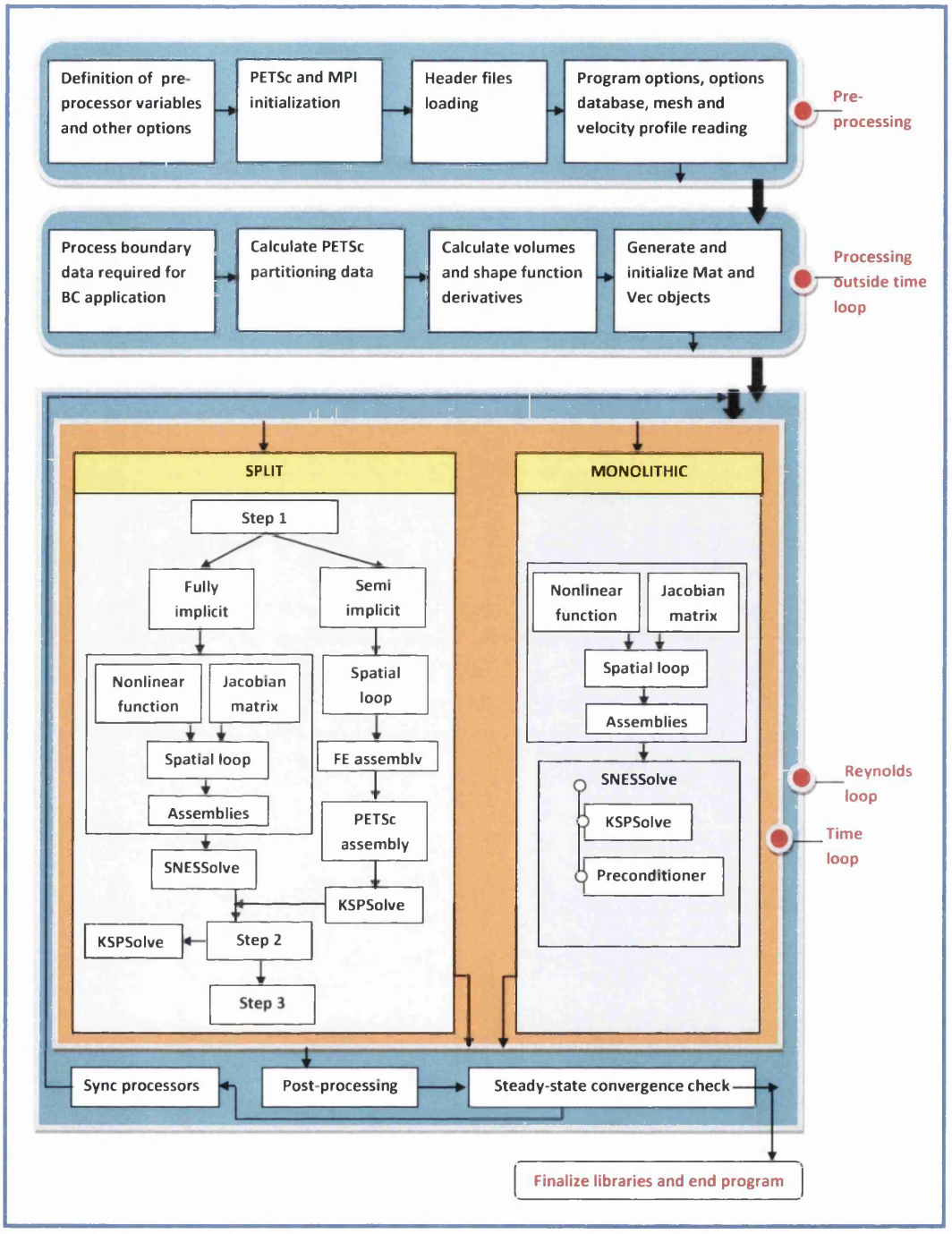


Figure 3-12: Overview of the parallel multi purpose Navier-Stokes Solver

desired source line. After the initial definition, the conditional directives are used (similar to typical conditional statements) to control visibility of source lines to the compiler. The Fortran variables defined by the preprocessor are now ready to be used in any source location beyond the initial definition. The variables that need to be visible globally in all subroutines are defined next within Fortran module(s). The actual program starts now with the typical 'program'keyword. PETSc specific header files are loaded next, which in turn loads the MPI header files by default. Hence, there is no strict need to separately initialize MPI. Although MPI may be explicitly initialized, if desired. The standard PETSc data types like Mat, Vec, KSP, PC, SNES, etc. are defined next, followed by the definition of standard Fortran data types. The validity of the values set in each of the preprocessor specific variables are tested next and a record of the current options are made in appropriate log files. If any of the checks fail, the code aborts via *MPI_ABORT*. Since in a supercomputing environment, running simulations is not interactive, any data that needs to be entered through the keyboard, is placed in the right order in a file called the options file, which will be accessed by the code, every time it expects to read data from the keyboard. This way an interactive code can be dealt, without problems in a job scheduler driven program execution environment. All the options are read in the rank 0 processor only, which broadcasts the data to the remaining processors. If vectors whose sizes are not known a priori, need to be transmitted, then care must be taken to first broadcast their size, allocate variables based on this size (in the receiving processors) and finally broadcast the vector entries. This will require the use of processor specific task allocation (based on processor ranks). Depending on the number of processors requested, the sequential or the partitioned mesh will be read next. The format of the partitioned files depends on the partitioner used, for which checks have been included. Each processor reads a specific mesh file assigned to it, there by reading the mesh in parallel. In cases where velocity profiles are being imposed, these are read from separate files (depending on the number

of boundaries). If the velocity profiles file are meant to impose transient dirichlet boundary conditions, then the file format changes significantly and therefore the code flow is adjusted to invoke the appropriate file readers. Currently, the velocity profiles are read sequentially and the required data is broadcast to the remaining processors. This completes the preprocessing stage.

The processing stage starts off in preparation for the application of boundary conditions. Every boundary is assigned a specific flag in the mesh. The options file mentioned before activates the boundaries on which to apply the required boundary conditions. A list of nodes occurring on each of the active boundaries is made, so that these can be prevented from being included in the parallel matrix, under the elimination approach of handling the dirichlet boundary conditions. Since there are multiple degrees of freedom (DOF) per node, the nodes list generated here must be suitably augmented. Two different arrangement of DOF are currently permitted in the system matrix. Either all DOF corresponding to a node appear together [u1,v1,w1,p1,....,un,vn,wn,pn] or a certain DOF-type for all nodes appear together [u1,...un,v1,...vn,w1,...,wn,p1,...pn]. Once a list of all dirichlet nodes is made, corresponding lists of boundary flags and values are also made. These arrays/lists are useful for incorporating the contributions from non-zero dirichlet boundary conditions in the RHS vector. Since this list generation process is a one time affair and is also very efficient by virtue of associative lists, which helps in direct/search-free indexing, this step is performed sequentially and the lists are broadcast to the remaining processors. In general, light weight and serially efficient operations that are **performed outside the time loop** may be run sequentially, if the parallelization is non-trivial with the data structures being used. Provisions for the inclusion of node-specific-values to successfully impose the steady velocity profiles is included at this point. However, for transient velocity profiles, the values need to imposed inside the time loop.

The contiguous matrix partitioning is calculated next, based on the size of the

augmented-reduced [5] system. Special provisions are set in place to repeat this process when the code is set to run in the split mode, because the step 2 of the CBS scheme involves a pressure-Poisson solve, which would require just one DOF per node unlike the CBS step 1 (which requires three). From the point of view of calculating the contributions that need to be assembled into the RHS vector, another array of contributing elements is generated. For these elements, the exact nodes that need to be considered is also pre computed and stored. In the split framework, the corresponding arrays for the pressure system are also generated and stored. Next, the volumes and shape function derivatives with respect to all spatial dimensions are calculated for all the elements in the mesh (the expressions for which, are presented in Appendix E). The quality of the mesh is also evaluated at this point, by calculating the smallest edge length.

The matrix preallocation data for all the matrices involved, is calculated next and fed into PETSc via suitable routines. The parallel matrices and vectors are generated and their sizes (global and local) are set suitably in all processors. The initial conditions are set next, via data entered in the options file. For restarting a simulation from an intermediate calculation, there are arrangements in place to read in the last known solution as initial conditions and restart the simulation from the point of exit. This is a useful feature for carrying long simulations that cannot complete within the specified wall-time limits of the job scheduler or in cases where a job gets killed after running for substantial lengths of time. When using the monolithic framework, various other data structures are generated to calculate and hold the pressure stabilization matrices.

After all the data structures are ready to aid the actual computations, the loop

[5] Augmented: The mesh assumes one DOF per nodes in the numbering of nodes. This data needs to be suitably modified depending on the number of DOF per node.
Reduced: After augmenting the node numbering, care must be taken to remove the dirichlet boundary nodes, by generating suitable mapping between the node numbers in the full and reduced systems.

over time begins. This loop over time is nested within a Reynolds loop[6]. Reynolds loop may be activated when a better initial guess is required for the Newton solvers. Setting the maximum number of Reynolds steps to one, the typical time loop environment can be activated.

In the time loop, the parallel matrices and vectors are initialized. Before the spatial loop starts all the data that would be needed in the construction of the elemental matrices and vectors is off-loaded from the parallel PETSc Vec objects into standard Fortran arrays. Depending on the type of solvers selected, in the very first time step, the required linear (KSP) and non-linear (SNES) objects are created. This is a one time operation, as these objects can be reused during consequent time and/or Reynolds steps. Subroutines that are specifically written for the generation of non-linear function and the Jacobian matrix, are made aware to PETSc at this point using the SNESSetFunction and SNESSetJacobian subroutines. In the non-linear mode, the spatial loops are hidden inside the subroutines specified by the SNESSetFunction and SNESSetJacobian subroutines. The SNESSet functions mentioned here are limited in terms of the number of arguments they allow. Therefore, data other than that allowed by these arguments will need to be made visible via other means. If the extra data needed is a PETSc data type, then a user defined context is provided as an optional argument to the concerned subroutines. Analogous to the concept of structs in C, the various parallel objects needed must be packaged into this single argument. If the additional data needed is a Fortran data type, then suitable Fortran modules must be employed to make these variables visible in the user defined routines.

The subroutines that evaluate the non-linear function and the Jacobian matrix are constructed based on equations C.27 through C.29 in the split framework and equations 2.50 through 2.57 in the monolithic framework. In these subroutines

[6]Analogous to the concept of load steps in solid mechanics, the reynolds loop gradually approaches the actual velocities in desired number of velocity steps

there are 2 notions of old solution. One being the previous time step solution and other being the previous Newton iteration solution. Given the old solution, these subroutines construct 16×16 elemental matrices and 16×1 elemental vectors (monolithic) or 12×12 elemental matrices and 12×1 elemental vectors (split). These elemental quantities are assembled into the global, parallel matrices and vectors by assigning the global indices in which they must be added. In PETSc, when Mat or Vec entries need to be selectively suppressed (while encountering dirichlet nodes), a negative index needs to be assigned. Although, this is the default behaviour for matrices, the vector objects can exhibit this behaviour by turning on the $VEC\_IGNORE\_NEGATIVE\_INDICES$ option using the VecSetOption sub routine. All the elemental matrix-vector and vector-vector multiplications are done without using the intrinsic matmul function in Fortran. For matrix-vector operations involving the parallel PETSc objects, subroutine like MatAXPY and VecAXPY may be used, where, AXPY denotes the standard $A * X + Y$ operation of several standard linear algebra libraries. In the PETSc framework, apart from the finite element assembly, all parallel objects need to be assembled to ensure that they contain valid entries in them (data may be cached). Setting appropriate values of the pre processor directive results in appropriate pressure stabilization matrices being constructed and assembled, in the monolithic framework. Finally, the function to invoke the actual non-linear solver (SNESSolve) is called. This subroutine automatically controls the invocation of the subroutines for the construction of the function and the jacobian matrix, during the solution process until convergence is achieved. Crucial information regarding the solution process may be obtained using appropriate flags in the options-database file.

While using PETSc, various options can be hard coded into the program. For flexibility, a file called the options-database file may be used to override these options. Depending on the object whose options need to be overridden, it is

therefore a good practice to include the *SetFromOptions sub routine for the concerned objects in the source, which activates the overriding procedures. The name of the object needs to be prepended to the SetFromOptions string to get the actual subroutine to invoke. These options can override the hard coded options by the use of *SetFromOptions sub routine. The options-database file is made visible to PETSc by providing its path and name while initializing PETSc. For transient simulations, the steady state convergence checks are performed next, the post-processing routines to invoke file writes are called before executing the next time step. ASCII Tecplot and binary ParaView writers have been programmed to visualize the results.

However, in the split framework the pressure-poisson solve (step 2) as well as the velocity correction (step 3) need to be performed before moving to the next time step. Provisions are made to be able to solve the intermediate velocity field of step 1 via a semi-implicit scheme (convective terms in the RHS and diffusion terms in the matrix). Equation specific coding is also performed to be able to run the Laplace, Stokes and Burgers equations in addition to the Navier-Stokes equations. The current version of the code is packaged into a code-folding enabled (when viewed in Vim [95]), single file, with the main program and all sub routines in a bit more than 10000 lines. Despite the massive line count for a single file, the code folding technique makes it possible to see the overall hierarchical structure and specific sections may be opened, while others are hidden.

3.4 Operation sequence

Typical steps that are generally performed while setting up simulations are presented in this section. Depending on the simulation being run, extra data might be needed, which is suitably read using the problem specific components in the code.

3.4.1 Meshing

Construction of the computational domain, i.e. the finite element meshing is the first step. For simple/idealized geometries, the 3D meshing package, Gmsh was utilized. The mesh format of Gmsh is converted into a format compatible with the IFENs solver, using a translator routine, specifically written for this purpose. Once the mesh is translated, the current working strategy requires it to be renumbered to reduce the bandwidth of the resulting system matrix, as mentioned before. The mesh is then partitioned using a custom serial routine specifically written for this purpose.

For patient-specific meshes additional steps need to be performed. A good representation of the vessel geometries is required to accurately predict the flow within them. Non-invasive data acquisition methods like computed tomography (CT) and magnetic resonance imaging (MRI) are usually utilized to get the initial patient data, which is basically an image set. The next step is to reconstruct the geometries. This involves extracting the surface/wall of the geometry from the image, i.e. to separate the vessel from the rest of the image, commonly referred to as segmentation. The implicit deformable model (IDM), based on the geometric potential force (GPF) field, was used to generate the meshes used in this research. An overview of the available segmentation methods along with their limitations and the IDM-GPF method have been presented in [134]. This reference also describes the surface and volume meshing strategy along with the mesh cosmetics/smoothing procedures. The entire sequence of operations is summarized in Figure 3-13. Once the patient specific mesh is ready, like for the idealized geometries the renumbering and partitioning programs need to operate on the mesh.

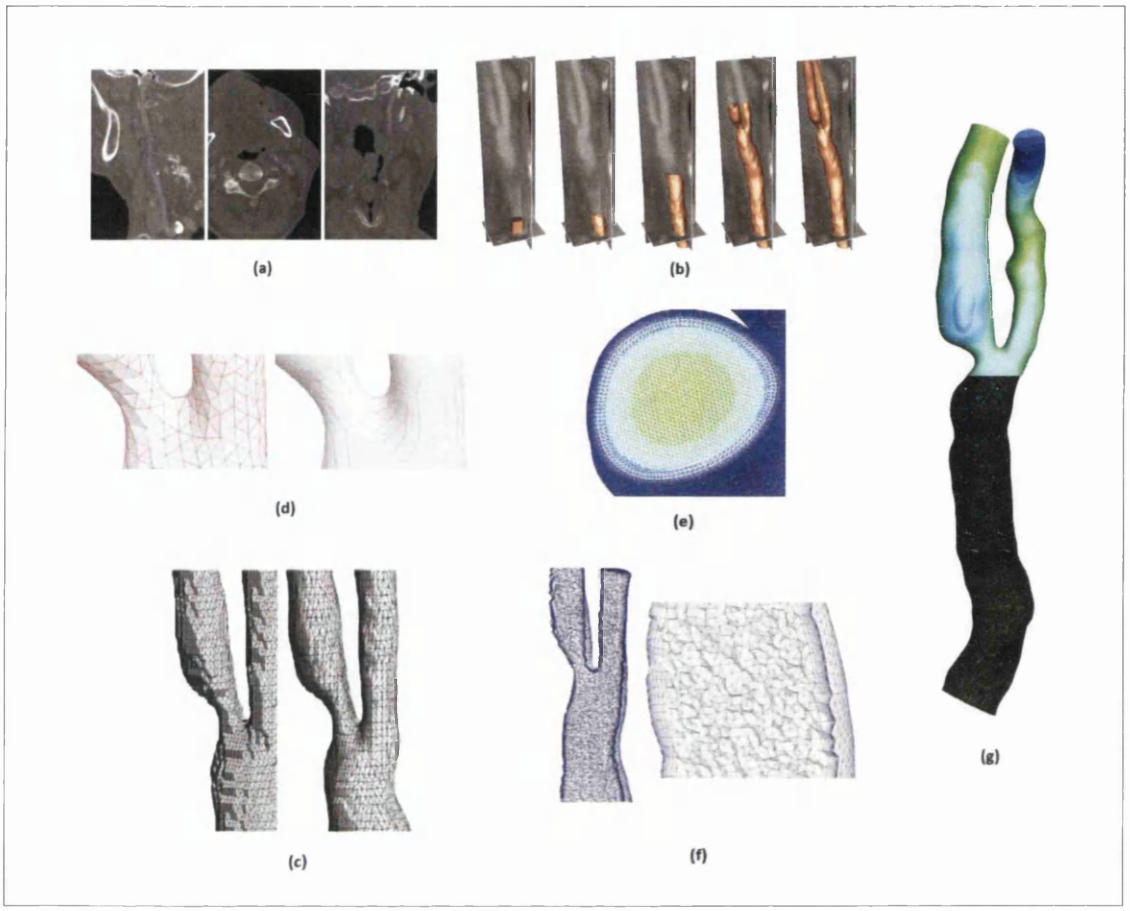


Figure 3-13: Patient specific meshing [134]: (a) CT scan of a carotid artery (b) Segmentation using IDM-GPF method (c) Surface meshing (d) Surface mesh refinement (e) Generation of boundary layers near the walls (f) Volume meshing (g) Final mesh, with mesh visible in the lower half

3.4.2 Generation of boundary condition data

This step is directly related to imposing dirichlet conditions at the boundaries. These boundary conditions may be constant, function of time and/or space. For constant boundary conditions, using a suitable flag-value mapping in the program options file will be sufficient. For conditions that change in space, for e.g. imposing a fully developed poiseuille profile at the inlet of a cylindrical pipe, a MATLAB script is used to generate the relevant values and the X,Y and Z components of the velocity profile are written to a data file. This file is read through

the right selection of preprocessor directives and using the boundary flag data, the required profile is imposed. However, for cases where the solution variable is a function of both space and time, e.g. imposing a womersley profile on non-circular boundary, the work flow involves additional steps, which are described in detail in subsections 5.1 through 5.5 of [134]. The velocity profile typically imposed for the carotid geometries considered in this research are shown in Figure 3-14. The harmonics used in the construction of this velocity profile are presented in Table 3.1

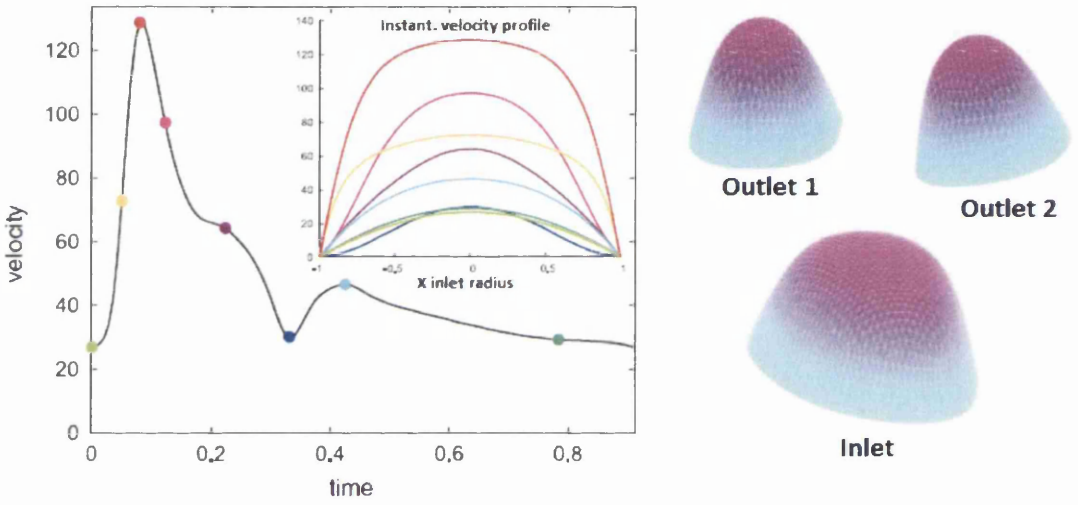


Figure 3-14: Typical velocity profiles for carotids [134]: (a) 2D inlet velocity profiles as a function of space and time (b) 3D peak velocity profiles at the boundaries

3.4.3 Job setup

The Navier-Stokes solver of this research was developed and executed on the supercomputing facilities of HPC Wales. The Load Sharing Facility (LSF) system is used there to schedule and execute workloads over the HPC environment. The workload consists of so called jobs which are generated and submitted by users to run their simulations. This job submission script is used by LSF scheduler to

Harmonic	Frequency (Hz)	Amplitude	Phase (rad)
0	0.0000000e+000	4.6926373e+001	0.0000000e+000
1	1.0875476e+000	2.1524133e+001	-1.1757456e+000
2	2.1750952e+000	1.7614591e+001	-1.6399569e+000
3	3.2626427e+000	1.2147710e+001	-2.4126155e+000
4	4.3501903e+000	6.7678515e+000	-2.5923204e+000
5	5.4377379e+000	9.0132960e+000	-2.8654415e+000
6	6.5252855e+000	8.0155307e+000	2.5792239e+000
7	7.6128331e+000	4.4200926e+000	2.0271286e+000
8	8.7003806e+000	3.5711419e+000	1.9278901e+000
9	9.7879282e+000	3.4320565e+000	1.3878100e+000
10	1.0875476e+001	2.3272178e+000	7.7936298e-001
11	1.1963023e+001	1.4287817e+000	6.2628156e-001
12	1.3050571e+001	1.5817702e+000	3.5279629e-001
13	1.4138119e+001	1.2964663e+000	-3.6075375e-001
14	1.5225666e+001	7.1691449e-001	-7.5219771e-001
15	1.6313214e+001	6.4928471e-001	-8.7984983e-001
16	1.7400761e+001	5.5890994e-001	-1.4506194e+000
17	1.8488309e+001	3.5656742e-001	-1.8778747e+000
18	1.9575856e+001	2.7656662e-001	-2.0518077e+000
19	2.0663404e+001	2.5371520e-001	-2.4611692e+000
20	2.1750952e+001	1.8540847e-001	-2.9991421e+000
21	2.2838499e+001	1.1846889e-001	3.0061654e+000
22	2.3926047e+001	1.0273053e-001	2.7403593e+000
23	2.5013594e+001	7.7012238e-002	2.2301464e+000
24	2.6101142e+001	4.9128254e-002	1.9239422e+000

Table 3.1: Harmonics used for the construction of the velocity profile in 3-14

access all the mandatory information required for it to schedule and execute jobs. This includes information about the number of processors to use, maximum wall time, queue name and execution mode. These are specified as directives to the job scheduler in the header section of the script. The paths and names of the executable along with all the input and output files are also included here. Any modules that may be required by the executable may be loaded here.

The job submission script also makes reference to another file, called the program options file. In the program options file, all the queries from the read(*,*) state-

ments of the source code are addressed (to substitute for the data entered by users at runtime in an interactive environment). Another file specific to PETSc, called, the options database file must be appropriately updated to suit the requirement of the simulation being run. Once the source code has been compiled with suitable flags and libraries, the job submission script is ready to be submitted. Depending on the work load, the program will be executed. Various commands are available to check the status of a job, post submission. The code itself writes a log file which is updated intermittently during the execution phase of the program.

3.4.4 Post-processing

Currently, the output from all processors is assembled together in a single file. Depending on the nature of simulations, a single solution block or multiple time blocks are appended to the same file. The results were analysed and required quantities were calculated using custom programs and subroutines. These were written in either Fortran or MATLAB, depending on the problem size. For visualizing the results, both Tecplot [75] and ParaView [66] were employed. Currently ASCII tecplot and/or binary vtk [76] files are generated. These steps generally complete the process of running a simulation.

3.5 PETSc specific details

For the fully implicit solution of the incompressible Navier-Stokes equations, the so called, Newton-Krylov framework was used both in the split and monolithic versions of the CBS implementation. With the non-linear function, as well as the jacobian matrix being derived, PETSc invokes its krylov solvers in every Newton iteration to find a better approximation of the solution. Variants of the Newton method, like line search and trust region are generally used. In this research,

the initial success with Newton line search resulted in that being chosen as the default non-linear solver. The restarted GMRES as well as the Loose GMRES - LGMRES [10] have been usually used in this research. The ASM preconditioner [47] was found to perform well with the chosen combination of solvers.

3.6 Summary

This chapter presented the computational framework in which the discretized Navier-Stokes equations were solved. Starting with a brief introduction to parallel computing, MPI and PETSc toolkit were introduced along with two basis examples. The domain decomposition process suitable in a matrix environment was then described along with the transition from the use of ParMETIS to a custom partitioner. A brief note on matrix preallocation was provided next, followed by an example in Appendix D. A description of the IFENs solver, developed in this research was also included. This helped highlight some of the important steps. Finally, the operation sequence involved in the execution of a typical simulation was described. This included meshing, generation of boundary condition data, parallel environment job setup and post-processing of the results.

Chapter 4

Benchmarking

4.1 Introduction

Chapter 3 described the framework used for computing solutions to the dicretized form of Navier-Stokes equations derived in Chapter 2. The current chapter aims to verify the overall correctness of the scheme and the code developed in this research. Various aspects like mesh renumbering, iterative solver framework, parallelization, number of processors, scheme and code, etc. may be separately subjected to validation. Simulations set up to check these aspects, as well as the results are presented in the following sections.

4.2 Mesh renumbering

As already illustrated in chapter 3, mesh renumbering is an efficiency measure, which improves convergence, while using iterative solvers. The solutions obtained by using a mesh generated with gmsh [60] were compared with those obtained by using the same mesh renumbered with RCM[1]. This test was performed serially

[1]Reverse Cuthill Mckee

(nprocs=1) using the non-monolithic formulation. The convection and diffusion terms were retained in the LHS matrix, hence employing the SNES objects to realize the Newton-Krylov solvers.

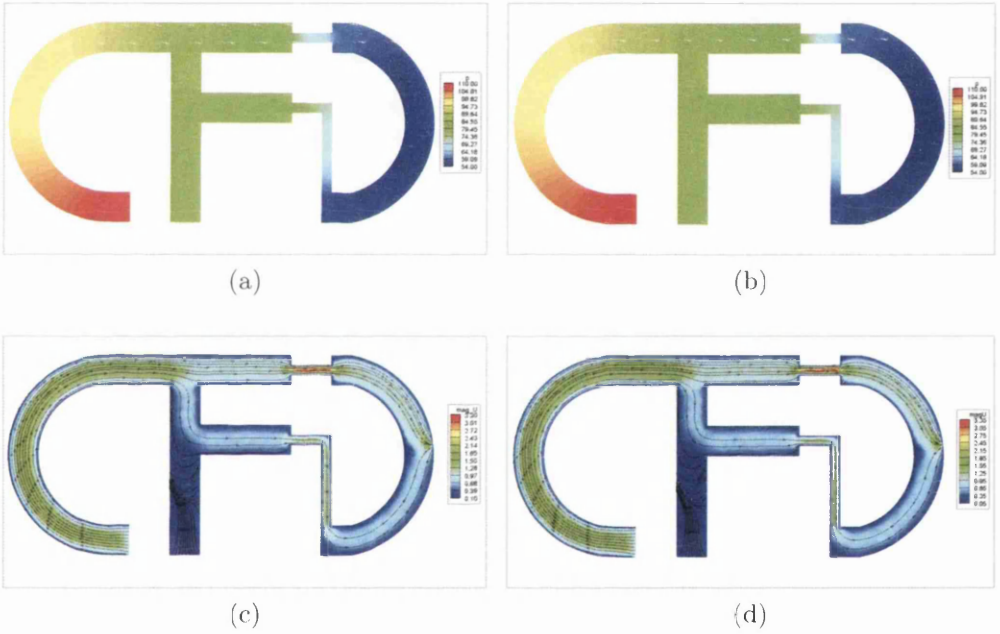
Figure 4-1: Mesh used for checking renumbering

The mesh used for checking the correctness of mesh renumbering is shown in Figure 4-1. It represents a 3D, internal-flow domain cast in the shape of characters C,F and D, connected in a way to result in some back flow. A uniform unit flow field was imposed along the X axis at the base of the character, C and a dirichlet boundary condition was imposed for pressure on just one node at approximately the mid height of character, D. A no slip condition was imposed on all boundaries. This domain had no outlet. The dirichlet boundary condition on pressure in the character D attracts the flow, by virtue of being visible as a low resistance point to the fluid.

Figure 4-2 presents the steady state pressure contours, with the streamtraces superimposed. The solutions before and after renumbering are identical. To make 2D comparisons, a slice was first extracted at mid thickness, i.e. $Z = 0.25cm$.

Velocities and pressures were then extracted along the X axis at a height of $2.815cm$. This comparison for velocity and pressure is presented in Figures 4-3a and 4-3b respectively. The extracted data was found to be in good agreement, reflecting the correctness of the renumbering algorithm. Also, both the cases converged to a steady state in 37 time steps.

The renumbering procedure was subjected to a couple of similar tests and the results with and without renumbering was observed to be identical in all the cases (results not presented).



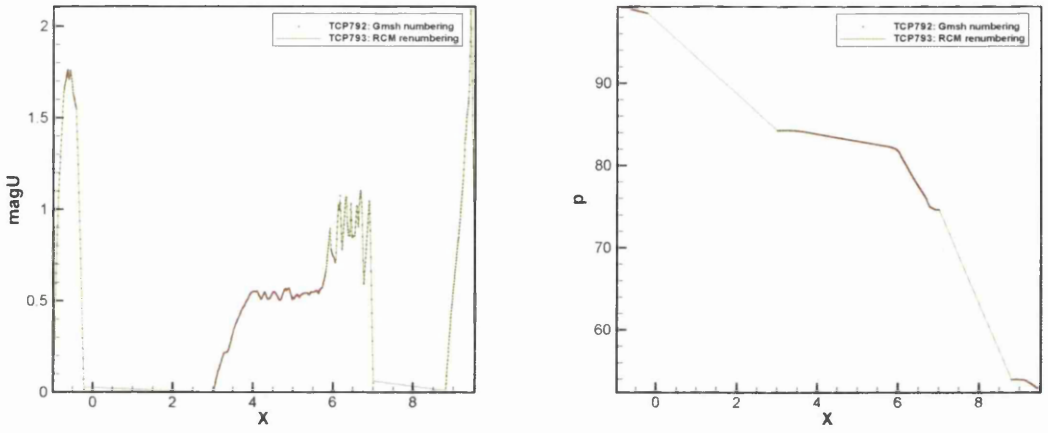
(a) (b)

(c) (d)

Figure 4-2: Comparison of pressures and velocity magnitude contours and streamtraces at steady state. (a,c) Gmsh numbering (b,d) RCM renumbering

4.3 Iterative linear solvers

All problems considered in this research (linear and non-linear) made use of iterative solvers, which in most cases was GMRES and its variants. Although, the iterative solver(s) were not coded, it is a good practise to check for any bugs in the

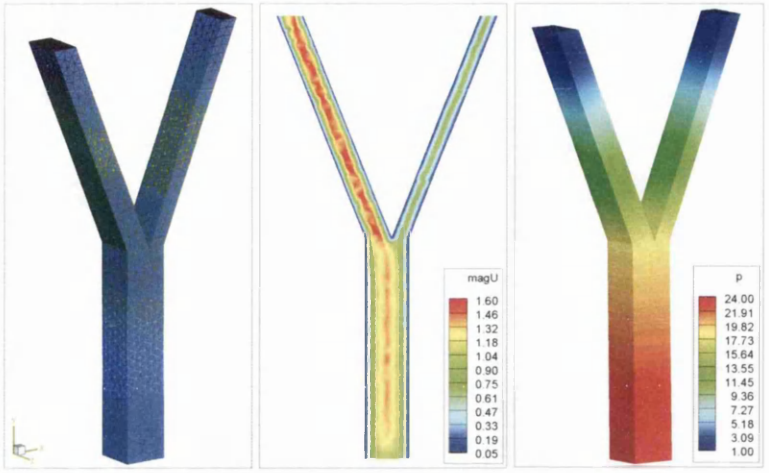


(a) (b)

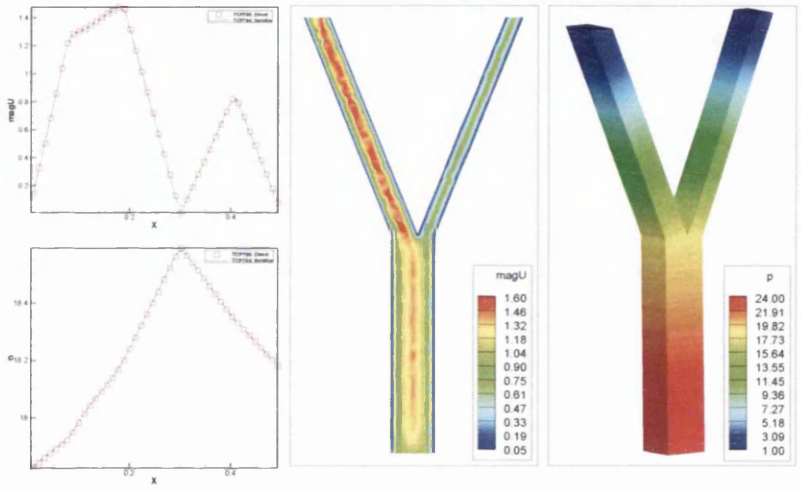
Figure 4-3: Checking renumbering (a) Velocity magnitudes (b) Pressure

library itself, PETSc in this case. Direct linear solvers (also available in PETSc) served as benchmarks. The solvers were validated in serial because PETSc by default doesn't provide for a parallel direct solver. Although, this is possible by interfacing with suitable external libraries (e.g. SuperLU, MUMPS, e.t.c). For the purpose of this test, the Newton-Krylov and pure Krylov solvers use GMRES (iterative) to test both the direct and iterative solution procedures. It might seem counter intuitive, but the key to realizing a direct solver in the PETSc framework is to use LU as a preconditioner with an iterative solver. When a direct preconditioner is used with an iterative solver, convergence is achieved in one iteration.

A simple bifurcation with a rectangular cross-section was used to test the iterative solvers. A $0.5cm \times 0.5cm$ parent branch bifurcates into 2 daughter branches of cross-sections $0.5cm \times 0.3cm$ and $0.5cm \times 0.2cm$. The overall length of the domain is $5cm$. A uniform vertical velocity of $1cm/s$ was imposed at the inlet ($Y = 0cm$), a zero dirichlet pressure condition was imposed on both the exits ($Y = 5cm$) and a no slip condition on all walls.



(a)



(b)

Figure 4-4: Comparison of direct and iterative solutions at steady state. (a) **Left** - Mesh; **Centre** - Velocity magnitudes in the XY plane at $Z = 0.25cm$ with the iterative solver; **Right** - Wall pressure obtained with the iterative solver. (b) **Left** - Comparison of velocity and pressure at the bifurcation; **Center** - Velocity magnitudes at corresponding locations with a direct solver; **Right** - Pressure contours on the walls as obtained with the direct solver.

A coarse mesh consisting of only 9218 tetrahedrons and 2299 nodes was used for this test. Figure 4-4 presents the mesh, pressure and velocity contours (with both iterative and direct solvers) and a comparison of pressures and velocities ex-

tracted along a horizontal line at the bifurcation (in the XY plane at $Z = 0.25cm$ and at a height of $Y = 2.5cm$). The direct and iterative solutions were found to be in good agreement over all time steps. Both simulations converged to a steady state in 22 time steps. While the direct solver required a wall time of 112.53 s, the iterative solver required 110.69 s. This test seems to represent a break-even point in terms of the computational load, since the execution times of direct and iterative solvers was almost equal. For smaller systems, direct solvers are known to outperform iterative solvers. As the size of the system increases, the iterative solvers outperform direct solvers (both in terms of speed and memory), by a large margin. The direct solution procedures become prohibitively expensive for moderately large systems and one easily gets into situations where direct solvers will no longer be able to give a solution.

Generally, in the world of iterative solvers, a large number of combinations of iterative linear and non-linear solvers; and preconditioners are available. For every iterative linear solver, one can use a combination of different non-linear solvers and preconditioners, resulting in a very huge number of possible settings, when tested methodically. The same is valid, exclusively for the remaining 2 components. Also, each of the 3 components mentioned above, invariably have one or more parameters to adjust, resulting in an even larger number of possible combinations. A detailed study of these combinations is not performed, but since the PETSc framework is being used, as required, a particular combination can easily be realized (in most cases, without even recompiling the code!). A rigorous analysis of this kind has been presented in the work of George et al [59].

The main aim here is to only check for consistency between solutions generated from direct and iterative procedures. The physical interpretation of the solution is not important in this context. Since a coarse mesh is used, the solutions indeed are not very smooth. In the succeeding sections of this chapter, the actual solution

will become important.

4.4 Parallelization

Since a parallel code was developed in this research it was necessary to validate the parallelization. The solutions obtained by running the code with just one processor are representative of executing a purely serial code, and hence these were used as benchmark data to validate the parallelization (both MPI and PETSc components). In testing for parallelization, the domain decomposition algorithm gets automatically tested.

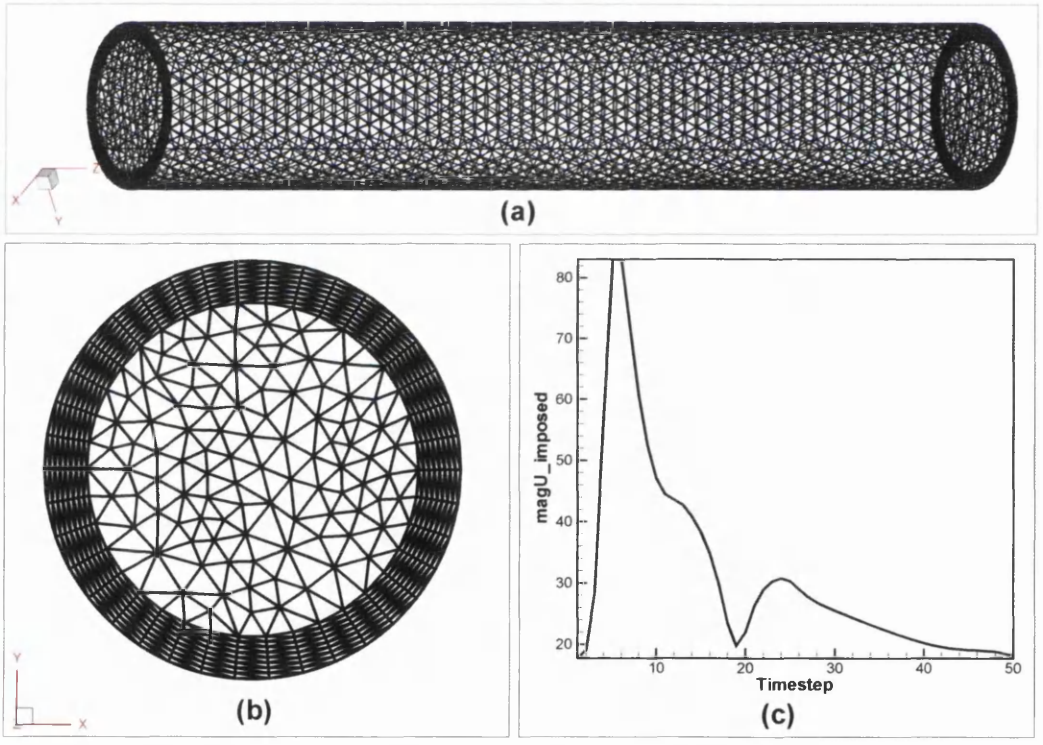
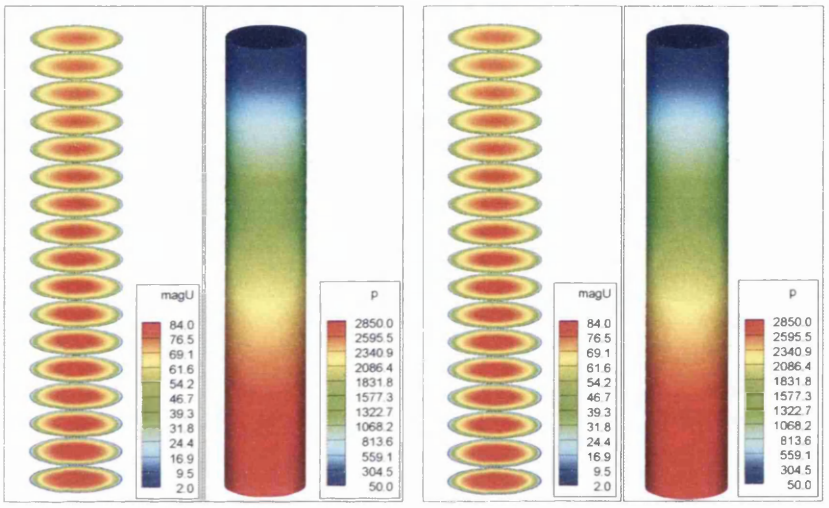


Figure 4-5: Inputs for validating parallelization. (a) Surface mesh (b) Inlet section depicting structured refinement at the walls (c) Transient velocity profile imposed at the inlet plane

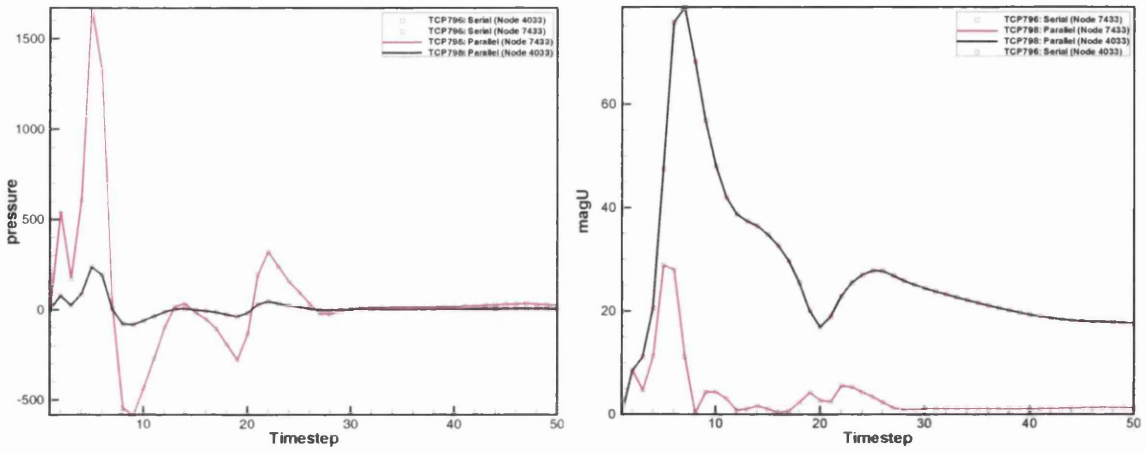
Flow through a pipe of prismatic cross-section was considered to validate the parallelization. A medium quality mesh with 8 boundary layers as shown in Figure 4-5 was used. It consisted of 169666 tetrahedrons and 30088 nodes. A transient velocity profile was imposed at the inlet plane with a peak velocity of $83.051 cm/s$. A total of 50 time steps were run in this test, with a constant time step of $0.01839 s$.



(a) (b)

Figure 4-6: Comparison of solutions obtained in serial and parallel. (a) Serial (b) Parallel

The pressures and velocities at peak flow, obtained from the serial and parallel runs are presented in Figure (4-6). The primitive variables are also tracked at 2 nodes throughout the transient cycle. Node 4033, which is approximately at the centre of the pipe, and node 7433, which is located proximal to the center of the exit plane, are tracked in time (Figure 4-7). All the solutions obtained from serial and parallel runs are found to be in good agreement with each other over the entire transient cycle.



(a) (b)

Figure 4-7: Transient solution tracking at nodes 4033 (located at approximately the central exit region) and 7433 (located in the mid length region, close to wall).

4.5 Number of processors

While solving linear systems in parallel, there is a possibility of the solution changing slightly as a function of the number of processors used. The PETSc documentation states that this is because of the use of MPI_ALLREDUCE(args) command for computing the inner products and norms. Depending on the size of local data, the values will get computed and arrive at a given processor in a certain order, which might change every time the system is solved under similar conditions. Owing to the non-associativity of floating point arithmetic, the computed quantity may be slightly different. These errors gradually build up over time. Also, the algorithm for most preconditioners, with the exception of Jacobi, is different for different number of processors, which will consequently result in greater differences in the solution computed [41]. Under these conditions it becomes necessary to assess the effect of the number of processors used.

For the purpose of testing the effect of number of processors on the solution, the test case of transient flow through a pipe similar to the one used for validating the parallelization is used. The simulation is repeated for the number of processors

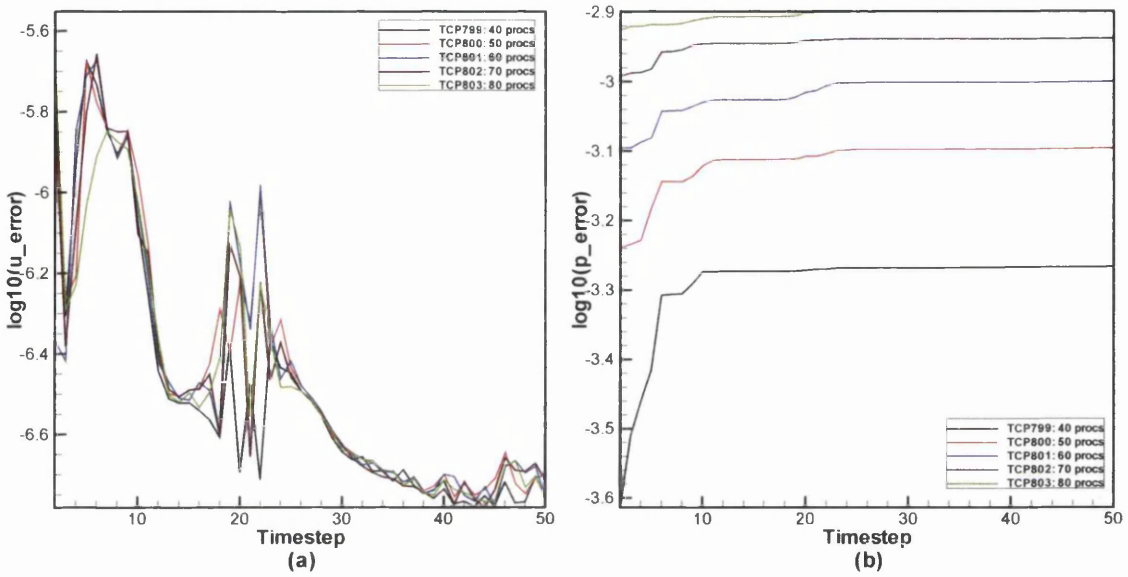


Figure 4-8: Change in solution as a function of number of processors. (a) Velocity error measure (b) Pressure error measure

ranging from 36 to 80. For a thorough comparison of the solution, the following error measure is calculated for both velocity magnitude and pressure. These are plotted in Figure 4-8.

$$\text{error} = log_{10}\left(\frac{\sqrt{\sum_{i=1}^{n}\left(u_{ref_i} - u_i\right)^2}}{n}\right)$$

where, u_{ref_i} is the nodal solution obtained using 1 processor, u_i is the nodal solution obtained in parallel and n is the number of nodes in the mesh. The change in solution variables was found to be within acceptable limits. The error in pressure shows a convergent trend, over the processor space that was possible to be used for this problem. The errors in velocity on the other hand behave even better and are orders of magnitude smaller than those encountered in pressure.

The inherent design of this test imposed a limit on the number of processors being used to 80. On one hand, since the reference solution was provided by the single processor serial run, the mesh had to be small enough to complete in

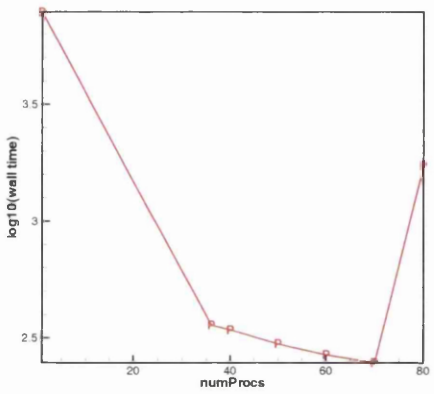


Figure 4-9: Illustration of the effect of over partitioning the mesh, on wall time

reasonable time. Since the mesh was small, excessive partitioning resulted in a critical point, beyond which the MPI communications become more expensive than the actual calculation itself. In this case this critical point was in the range of 70-80 processors. With 80 processors, the time to complete the simulation was unreasonably high, as shown in Figure 4-9. Considering that all computations were timed and only a limited number of CPU hours were available to use, this mesh was not partitioned beyond 80 parts. Close to the range of 70-80 processors the number of shared matrix entries far exceeds the number of local entries leading to an increased MPI communications load. For a well arranged banded matrix, the critical point mentioned before is reached when the size of matrix partition tends towards the bandwidth of the matrix.

4.6 Schemes and code

To validate the code and the mathematical schemes encoded within, several standard benchmark problems were executed. These were single lid driven cavity, backward facing step, flow past a cylinder and flow through a pipe. All problems were three dimensional and steady, with the exception of flow past a cylinder. Flow past cylinder is also the only external flow case, while the remaining

benchmarks represent internal flow domains. The lid driven cavity is completely enclosed, i.e. no flow occurs across the cavity boundaries. For the case of a backward facing step and flow through a pipe, fluid enters and leaves the domain, like in carotid bifurcations.

Figure 4-10: 3D Lid driven cavity mesh

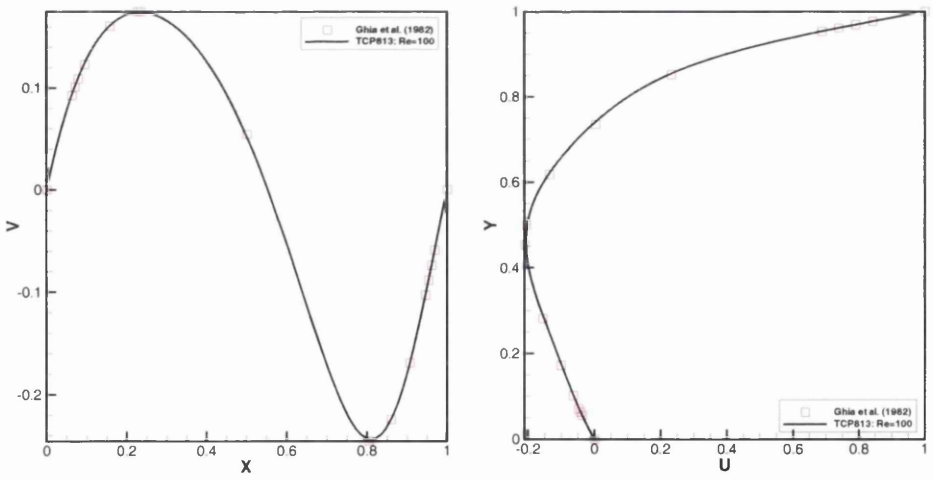
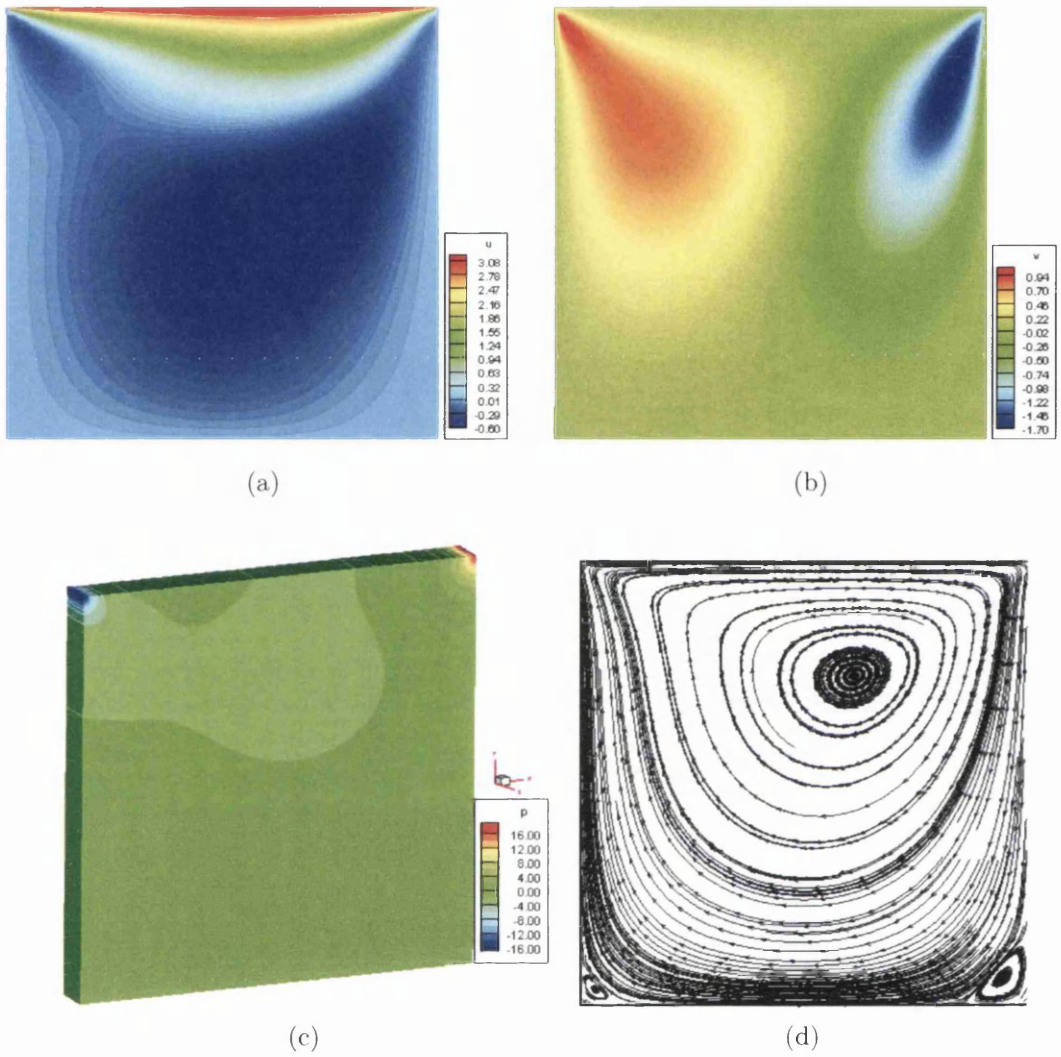


Figure 4-11: 3D Lid driven cavity at $Re = 100$



(a) (b)

(c) (d)

Figure 4-12: Lid driven cavity at Re = 100: (a) Horizontal velocity contours (b) Vertical velocity contours (c) Pressure contours (d) Streamtrace plot of the velocity field

4.6.1 Single lid driven cavity

This is one of the most widely used benchmarks for testing new schemes and codes. A cuboid of size 1.0 cm × 1.0 cm × 0.1 cm, constitutes the computational domain in this test case. The mesh used is presented in Figure 4-10. It contains 52999 nodes, 293784 tetrahedrons and was partitioned into 36 sub domains. The top face of the cuboid is called the lid. A positive, non-zero horizontal velocity

component is imposed on the lid while a no slip condition is imposed on the three remaining rectangular lids. A zero dirichlet pressure is imposed on a single node, in the nort-east quadrant of the cuboid (imposing zero pressure on any of the bottom corners, prevents the formation of a recirculation zone at that corner. This is because such a point represents the least pressure with respect to its surroundings, hence attracting the fluid towards it). The moving lid drives the flow within the cavity, in the clockwise direction, forming a primary vortex occupying most of the cavity. At the bottom corners two recirculation zones also appear as the flow separates and shears the trapped fluid pockets. Such flow features are often encountered in complicated patient specific geometries and therefore the lid driven cavity is a representative test case. Results from Ghia et. al. [61] are commonly used as reference data for this problem. For quantitative comparisons, results along the vertical and horizontal, geometric centres were compared against the reference solutions. These are presented in Figures 4-11 and 4-13 for Reynolds numbers of 100 and 400 respectively. The results were found to be in good agreement with the reference solution. The plots of Figure 4-12 present the horizontal and vertical velocity contours; pressure contours and the streamtrace plots for a Reynolds number of 100. Figure 4-14 presents a similar set of plots for a Reynolds number of 400.

4.6.2 Backward facing step

This is an important benchmark as it contains a sudden change in cross sectional area at the step, in combination with channel flow features away from the step. Since the patient specific meshes presented in chapter 5, represent very complex tubular geometries, this benchmark is also representative of flow fields expected in carotid geometries. The mesh used, along with the horizontal velocity and pressure contours are presented in Figure 4-15.

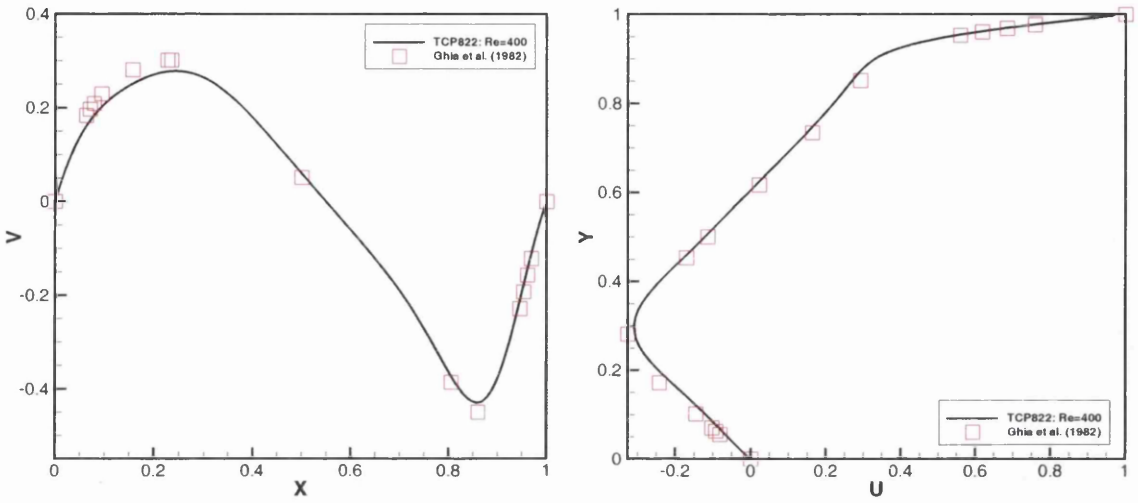
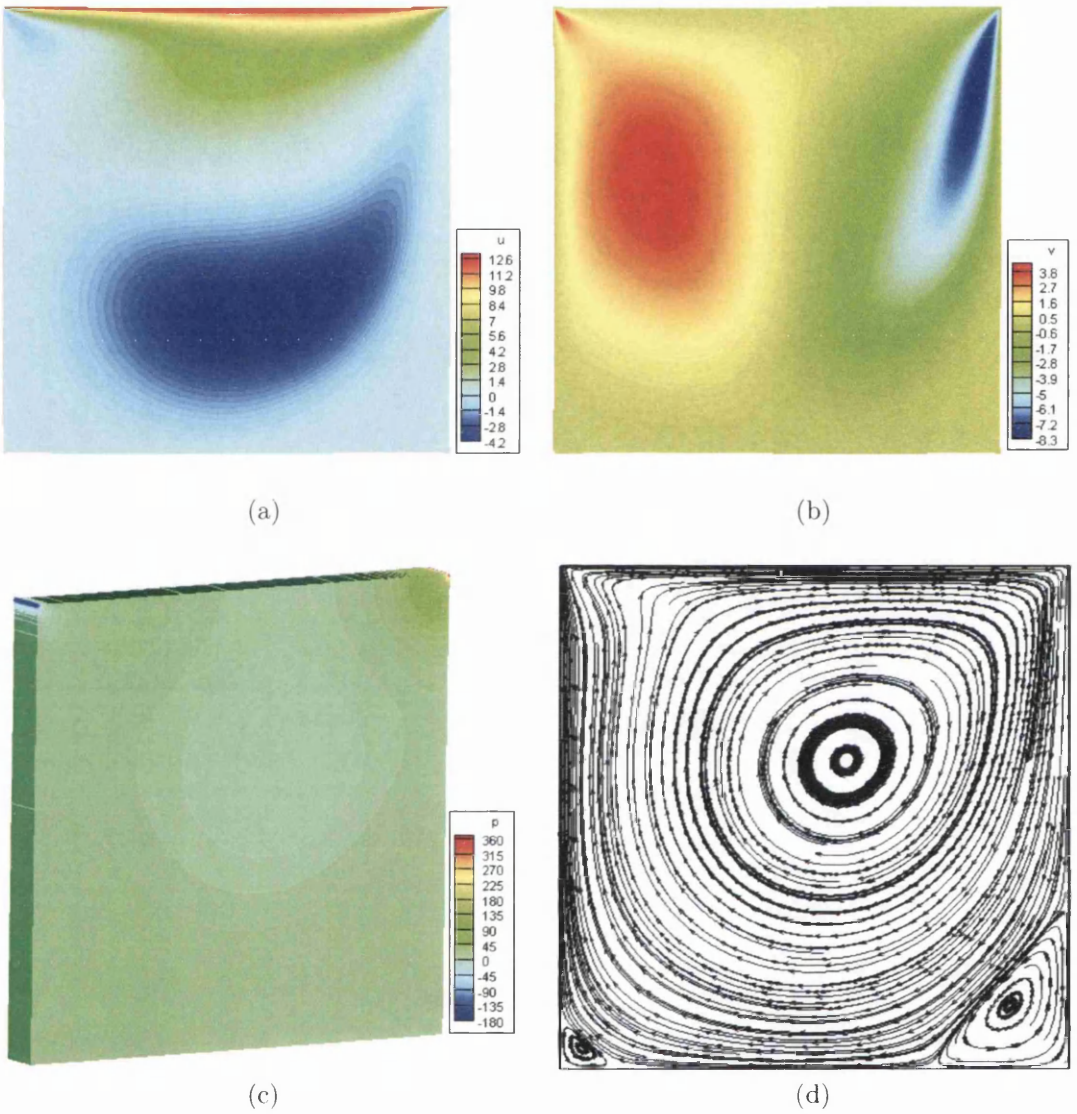


Figure 4-13: 3D Lid driven cavity at $Re = 400$

The backward facing geometry is formed by subtracting a cuboid of size 4L × 1L × 1L from a bigger cuboid of size 40L × 3L × 1L, such that the left bottom vertices of both the cuboids coincide. Here, L is the step height and is chosen to be 0.1 cm. The mesh used contains 52934 nodes, 287631 tetrahedronsand was partitioned into 36 sub-domains.

For validation, experimental results from Denham and Patrick [40] were used. The experimental velocity profile imposed at the inlet is not exactly parabolic. For obtaining the numerical results, the experimental values were suitably interpolated. To reduce the errors from interpolation the mesh was refined at the inlet. However, due to the unstructured nature of the mesh, slight differences exist along the thickness and hence the velocity profile imposed at the inlet is slightly different from the experimental profile. The numerical horizontal velocity component at 6 different sections along the length of the domain (in the mid Z plane)were compared with corresponding values from [40] and found to be in good agreement with each other. This comparison is presented in Figure 4-16.



(a) (b)

(c) (d)

Figure 4-14: Lid driven cavity at Re=400: (a) Horizontal velocity contours (b) Vertical velocity contours (c) Pressure contours (d) Streamtrace plot of the velocity field

4.6.3 Flow past a cylinder

This is an external flow, transient benchmark and provides an opportunity to test the evolution of solution in time. The region around the cylinder is represented by a rectangular domain, whose length and width are $25D$ and $10D$ respectively,

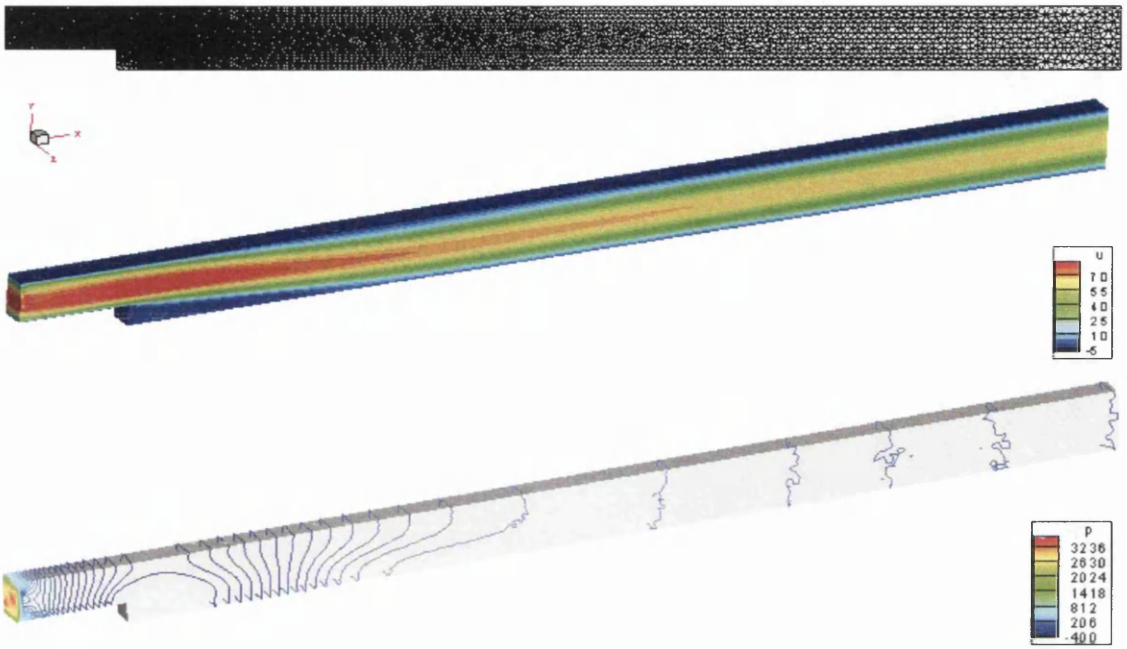


Figure 4-15: Backward facing step at $Re = 229$. Top: Mesh used. Centre: Horizontal velocity contours. Bottom: Pressure contour lines

where, D = 1.0 cm, is the diameter of the cylinder. In order to capture the vortex shedding downstream of the cylinder, the mesh along the centreline region is refined (Figure 4-17). The mesh contained 17382 nodes, 69948 tetrahedrons and was partitioned into 32 sub-domains.

The horizontal and vertical velocity contours; pressure contours, as well as the streamtrace in the vicinity of the cylinder are presented in Figures 4-18,4-19,4-20 and 4-21a. The vertical velocity is tracked through the entire time period (250 s) and plotted in Figure 4-21b. The Strouhal number for this case was found to be 0.14, which is lesser than the expected value of 0.165. The underestimation of the Strouhal number occurs because the fluid is not constrained in the vertical direction at the horizontal walls. As a result, the vortex shedding results in some mass loss at the walls, especially in the regions close to the cylinder. This overall has an effect of reducing the primary velocity component (horizontal) that is experienced by the cylinder. As a result the frequency of vortex shedding will be

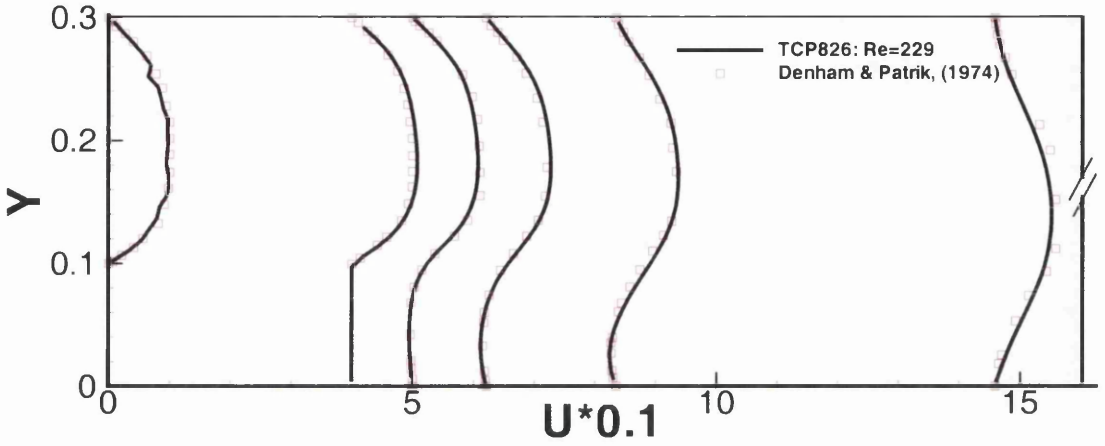


Figure 4-16: Validating the velocity distributions at various sections (up to a length of $14.586L$) for a backward facing step at $Re = 229$

slightly smaller, which consequently explains the underestimation of the Strouhal number.

4.6.4 Flow through a prismatic pipe

A steady flow through a circular pipe of uniform cross-section is considered here. A uniform velocity profile is imposed at the inlet and the length of the pipe is sufficient for the flow to transition into a fully developed flow. A zero dirichlet boundary condition is imposed on the entire exit plane. A no slip condition is imposed on the walls. A summary of the problem definition along with the quality of the mesh used for this test is presented in Figure 4-22.

The developed region spanning $6 \leq x \leq 10$, along the pipe length is considered as the test section for validating against the analytical pressure drop, where the flow is fully developed. In this case, the entrance length for laminar flow may be evaluated from the following relation,

$$\frac{l_e}{D} = 0.06Re \tag{4.1}$$

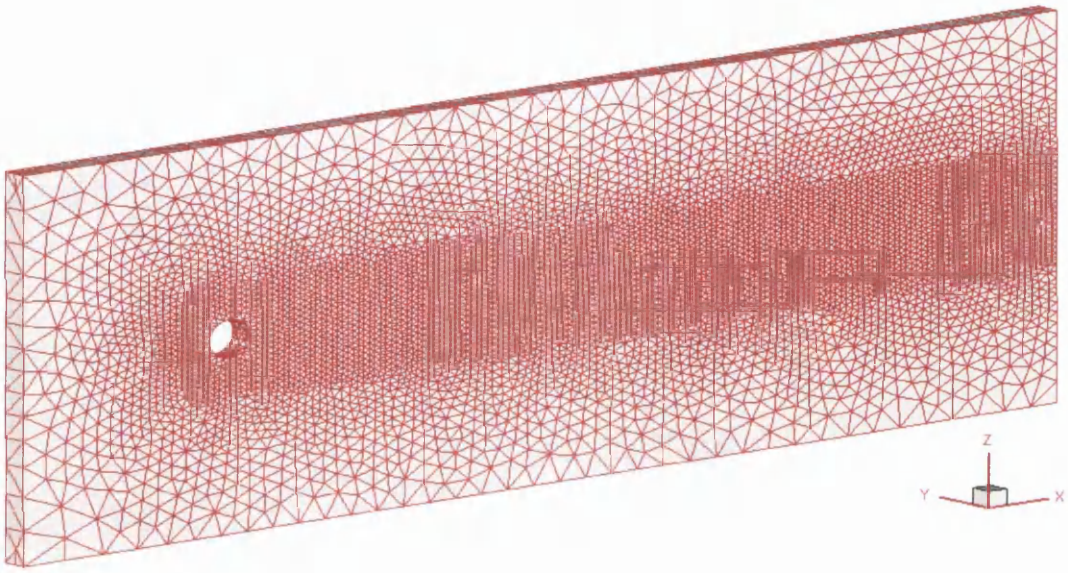


Figure 4-17: Mesh used for flow past a cylinder

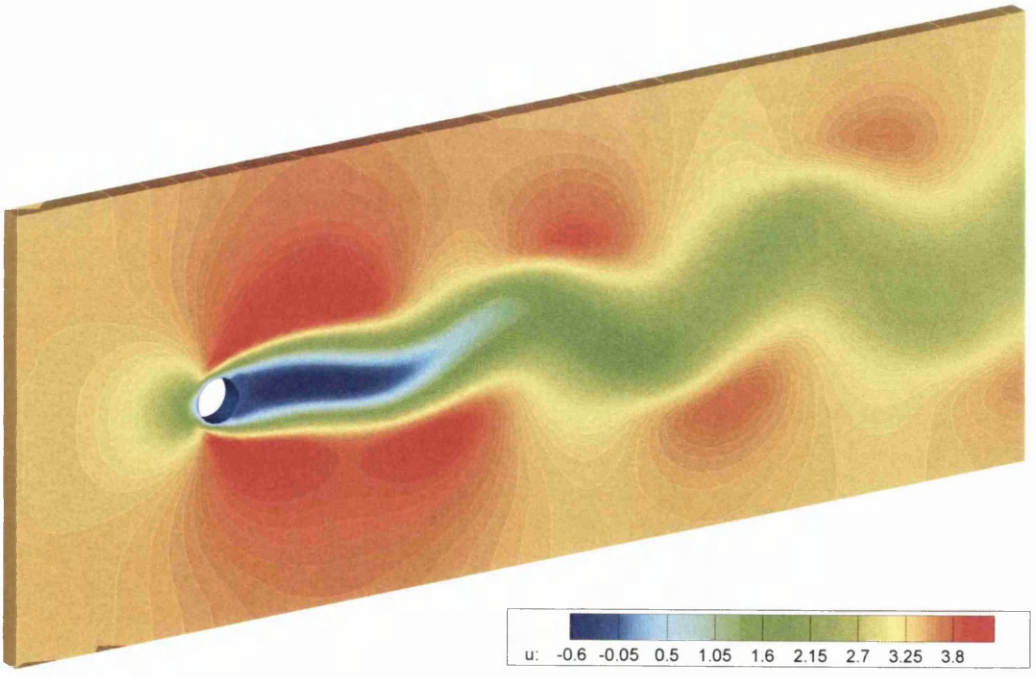


Figure 4-18: Flow past a cylinder: Horizontal velocity contours

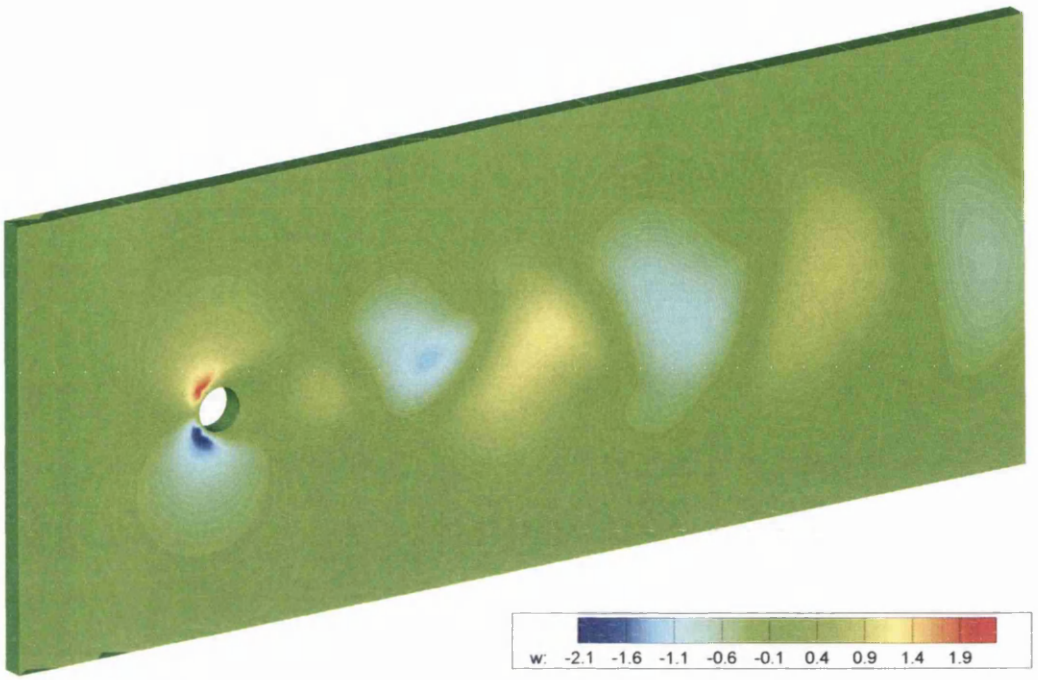


Figure 4-19: Flow past a cylinder: Vertical velocity contours

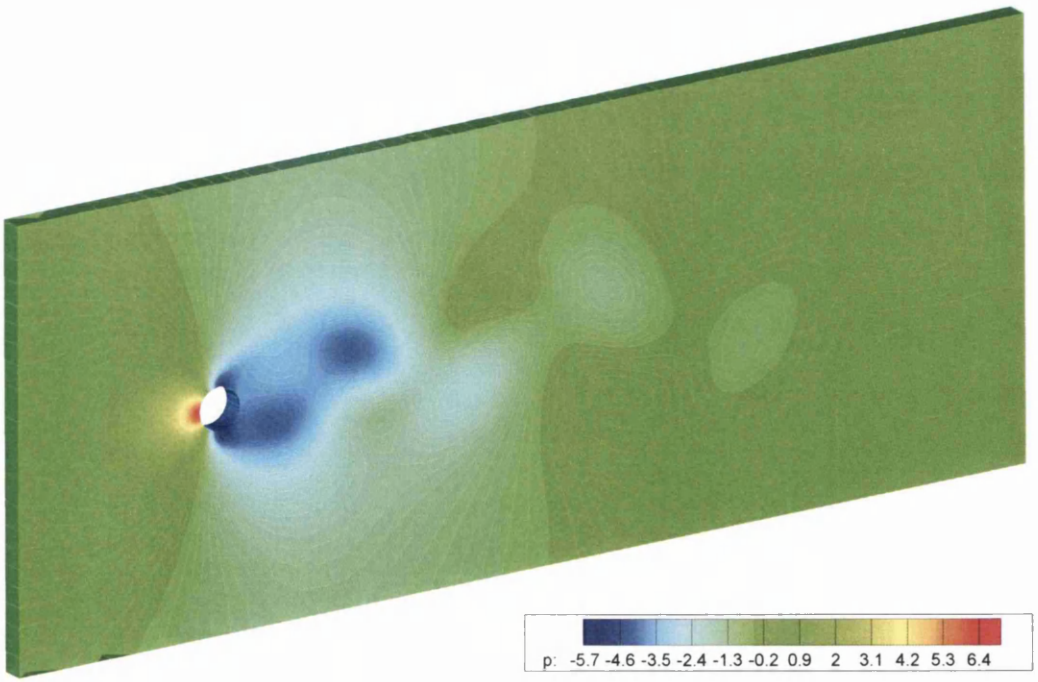
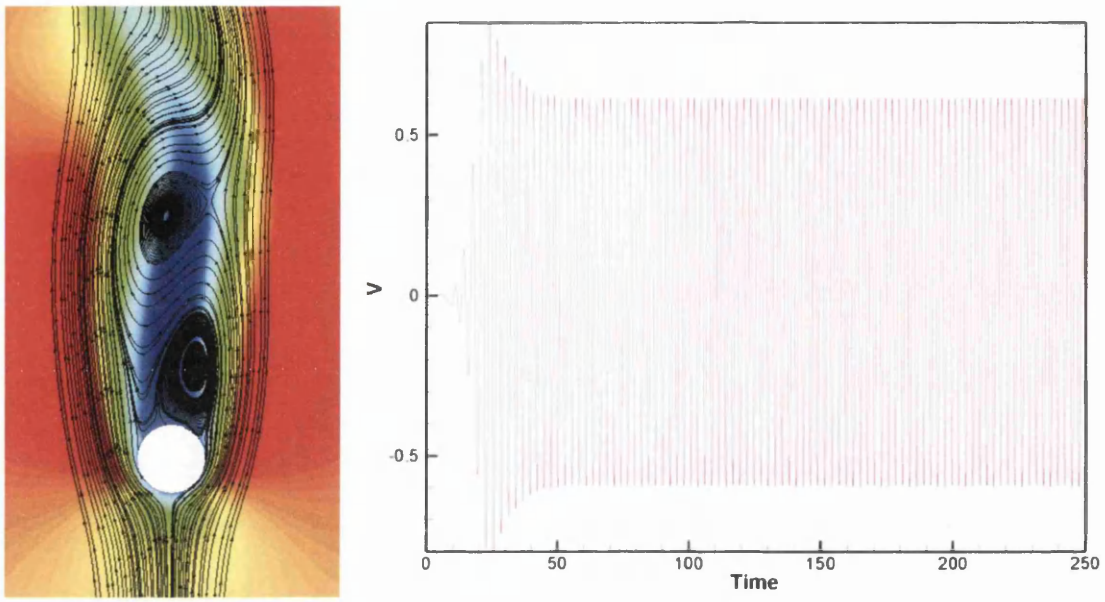


Figure 4-20: Flow past a cylinder: Pressure contours



(a) (b)

Figure 4-21: Flow past a cylinder: (a) Streamlines imposed on horizontal velocity contour in the vicinity of the cylinder (b) Time history of vertical velocity at a central exit node

where, l_e is the entrance length, D is the diameter and Re is the reynolds number. Since $D = 1$ cmand $Re = 100$, the entrance length would be $6cm$.

The pressure and sectional velocity contours are presented in Figure 4-23. The velocity profiles at various sections along the length are presented in Figure 4-24. The numerical velocity profile at the exit was found to be in very good agreement with the fully developed, analytical velocity profile. The pressure distribution along the test length is presented in Figure 4-25. The predicted pressure drop was in close agreement with the analytical pressure drop, given by the Hagen-Poiseuille equation,

$$\Delta P = \frac{8\mu LQ}{\pi r^4} \tag{4.2}$$

where, ΔP is the pressure drop, L is the pipe length, μ id the dynamic viscosity,

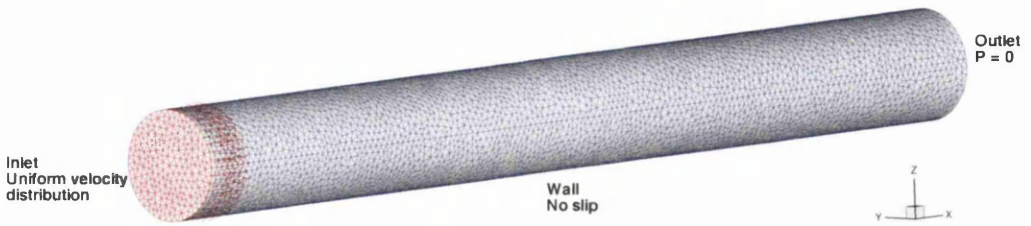


Figure 4-22: Flow through a prismatic pipe: Problem definition and computational domain

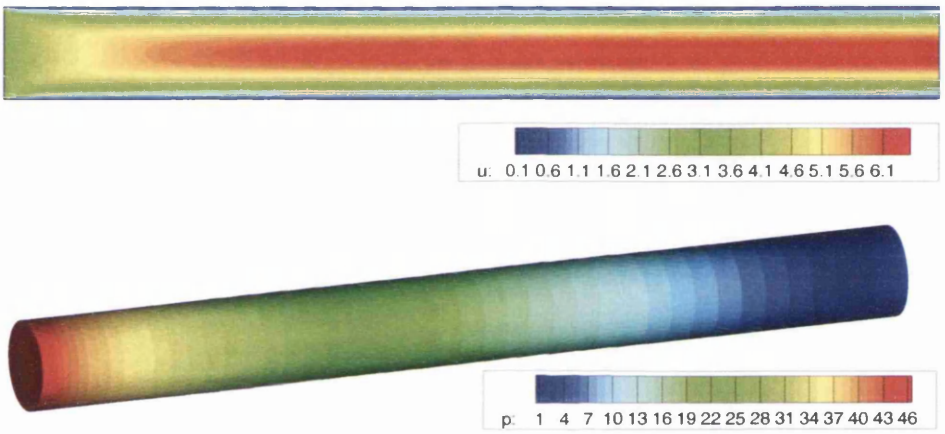


Figure 4-23: Flow through a prismatic pipe: Pressure and sectional velocity contours

Q is the volumetric flow rate and r is the radius.

4.6.5 Scalability

Ideally, in a parallel computing environment, the wall time requirements must get halved as the number of processors used are doubled. This property represents how well a parallel code scales in performance, by virtue of parallelization under similar test conditions. A scalability study therefore measures the quality of parallelization. A common metric used in scalability studies, is called speedup,

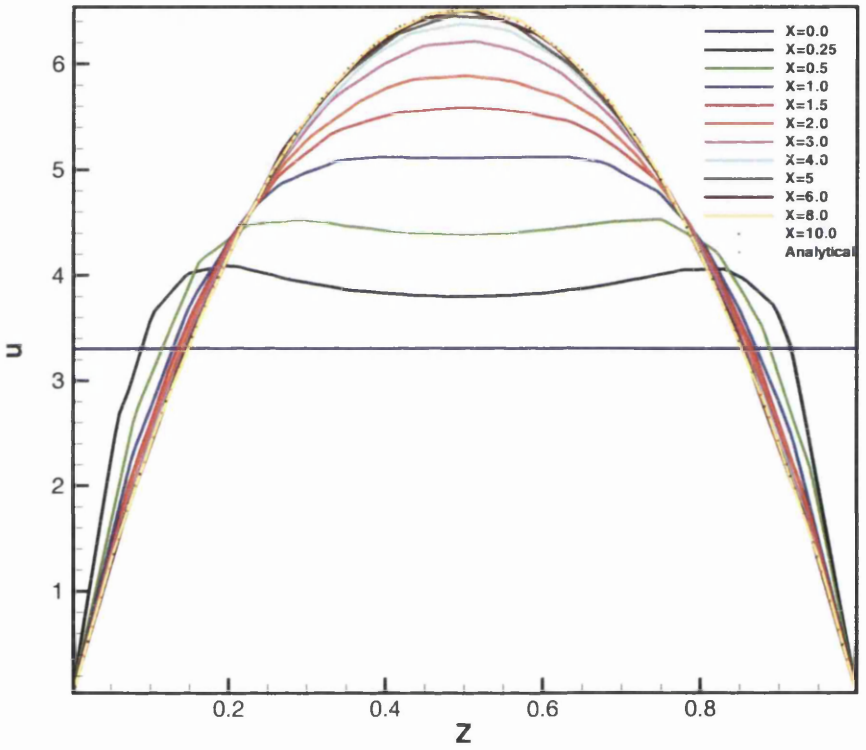


Figure 4-24: Flow through a prismatic pipe: Velocity profiles at various sections along the pipe length

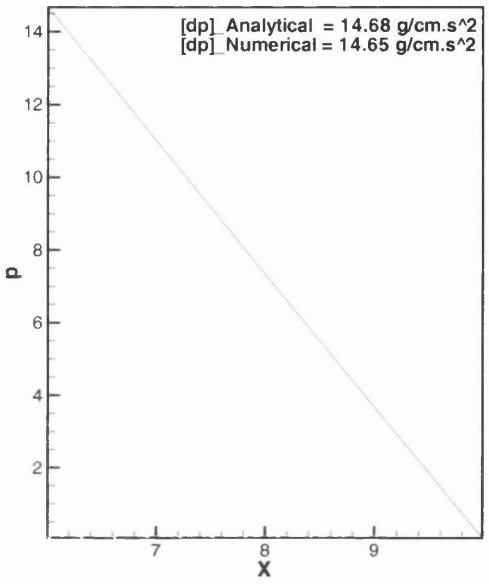


Figure 4-25: Flow through a prismatic pipe: Pressure distribution along the pipe length in the test section of the pipe

S. In this context, speedup may be defined as,

$$\mathbf{S} = \frac{T_s}{T_p}$$

where, T_s denotes the wall time with 1 processor (serial) and T_p represents the wall time in parallel. Ideal speedup occurs, when,

$$\mathbf{S} = p$$

where, p represents the number of processors.

Generally, ideal speedup is easily achieved in situations where there is no or negligible interprocessor communication. These are some times refereed to as *embarrassingly parallel* problems. Computer graphics rendering is one such example. Such problems are usually rare. A parallel code renders itself to be potentially able to scale linearly/ideally, when there are no/negligible serial components. In

cases, where serial components are prevalent in a parallel application, there is a limit on speedup imposed by the serial components. Irrespective of the quality of parallelization, the quickest such a program will complete will always be greater than the time required to perform the serial operation(s). Amdahl's law [132] summarises this in an equation form as,

$$\mathbf{S(p)} = \frac{1}{B + \frac{1}{p}(1 - B)} \tag{4.3}$$

where, $B \in [0, 1]$, represents the serial fraction of the program.

Also, matrix-free schemes (e.g. explicit) render themselves to scale linearly or better (moderately super-linear). Super linear speedup is said to occur when the speedup exceeds the number of processors employed ($S > p$). Although rare, these speedups are possible. If a problem gets so small after partitioning that it can completely fit into the cache memory, which is the fastest computer memory, then superlinear speedup occurs, as the data access is tremendously efficient. Techniques like backtracking [141], searching large data sets [129] and neural network based optimizations [104] lend themselves to superlinear speedup. However, superlinear speedups must be handled with caution. If not caused by the situations listed above, it most likely indicates a very inefficient serial implementation [16, 139].

In order to demonstrate the parallel performance of the code developed in this research, two scalability studies are presented in this section - low range (LR) scalability and high range (HR) scalability. The LR[2] tests were carried out with 1, 2, 4, 8, 16 and 32 processors, with a carotid mesh that contained 323856 tetrahedrons and 57234 nodes. The HR[3] tests were executed on 36, 72 and 144 processors, with a carotid mesh containing 4014037 tetrahedrons and 695605 nodes. The reasons for splitting this test into 2 ranges are similar to those mentioned

[2]Job codes - 1:TCP885; 2:TCP884; 4:TCP883; 8:TCP882; 16:TCP881; 32:TCP880
[3]Job codes - 36:TCP780; 72:TCP778; 144:TCP892

in section 4.5, i.e., the mesh must be small enough to be able to complete in reasonable times, but big enough to prevent excessive communication overhead due to over partitioning.

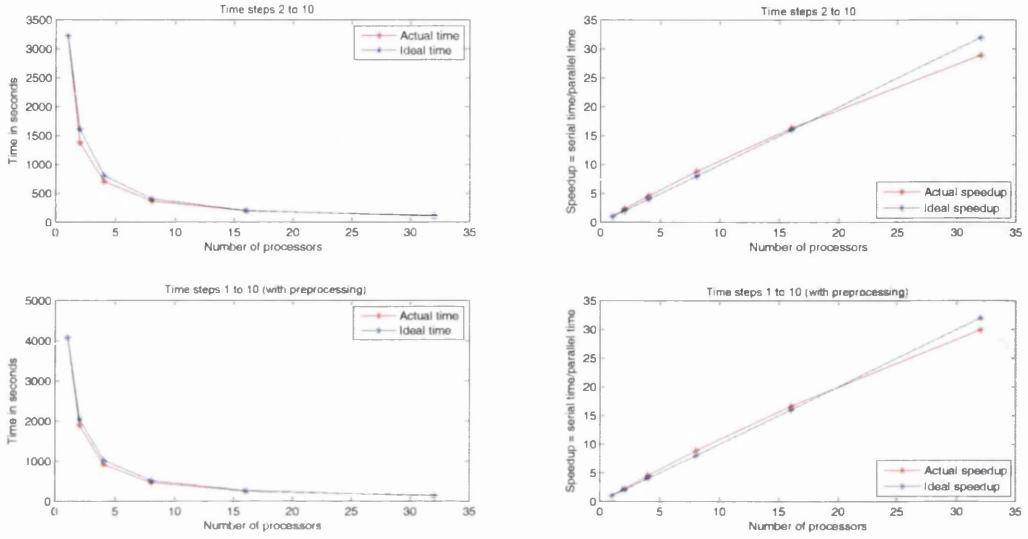


Figure 4-26: Low range scalability results: Wall time and speedup comparisons

The scalability tests were carried out on the facilities of HPC Wales, Bangor. A total of 10 time steps were run in each case and the I/O from the code was limited to just the essential data. Within every time step, several Newton-Krylov and pure-krylov iterations are performed. The iteration data is presented in Tables 4.1 and 4.2 . Two different time samples were extracted in the LR study. Since some operations related to applying boundary conditions were performed serially outside the time loop, the first set of time measurements didn't include this phase of the code. Also, the first time step is relatively expensive as the matrix preallocation is performed, this too was excluded from the first measurement set. These results are presented on the top half of 4-26. In the second set of measurements, all preprocessing operations, along with the first timestep are included, to assess the overall performance. These results are presented in the bottom half of Figure 4-26. In both types of sampling of the low range study, the scalability

nprocs	T1		T2		T3		T4		T5	
	NKi	Ki	NKi	Ki	NKi	Ki	NKi	Ki	NKi	Ki
1	7,34	1210	5,47	1484	4,38	862	3,31	675	3,31	760
2	7,34	1238	5,47	1355	4,38	848	3,31	674	3,31	721
4	7,34	1276	5,48	1310	4,41	839	3,31	680	3,31	737
8	7,34	1392	5,48	1330	4,41	858	3,31	677	3,32	718
16	7,34	1590	5,49	1433	4,41	1001	3,31	683	3,33	866
32	7,34	1132	5,52	1413	4,46	885	3,37	695	3,36	1024

Table 4.1: Iteration history (T:Time step; NKi:Newton Krylov-iterations. Format=(i,j) i=newton iterations and j = cumulative krylov iterations in i newton iterations; Ki:Krylov iterations)

nprocs	T6		T7		T8		T9		T10	
	NKi	Ki	NKi	Ki	NKi	Ki	NKi	Ki	NKi	Ki
1	3,32	0801	3,33	0660	3,33	0682	4,48	714	4,51	892
2	3,32	780	3,33	662	3,33	686	4,48	715	4,51	899
4	3,32	894	3,33	656	3,35	660	4,48	655	4,51	775
8	3,33	866	3,35	694	3,35	741	4,49	687	4,52	927
16	3,35	752	3,36	862	3,37	905	4,53	965	4,56	769
32	3,38	764	3,39	748	3,41	887	4,57	932	4,61	1014

Table 4.2: Iteration history with 1 to 32 processors: Time steps 6 to 10

was found to be near linear. This is also indicative of the overall good quality, appropriate serial fraction selection and serial efficient routines, in the code. With 32 partitions of the medium sized mesh, the local matrix patches would approach the matrix bandwidth, resulting in increased MPI communication. As a result, sub-optimal speedup is expected around the range of 32 processors, for the mesh employed. However, the average parallelization efficiency for the LR tests, considering both types of time sampling was found to be 105.13%.

For the HR scalability test, a complete simulation ranging the entire cardiac cycle was executed. No special arrangements, like limiting the amount of I/O or selective time sampling were employed. As a result, the true scalability or working scalability that would be experienced while running real problems with the code, will get demonstrated. The wall time and speedup results for the HR scalability

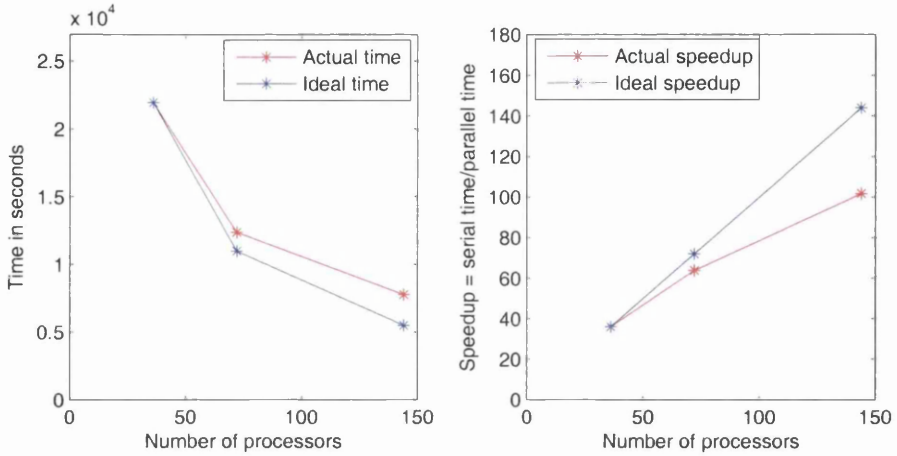


Figure 4-27: High range scalability results: Wall time and speedup comparisons

test are presented in Figure 4-27. The HR scalability was not performed on less than 36 processors, as a result there was insufficient data for the speedup calculation (i.e. serial time, T_s). In light of the corresponding linear speedup in the low range tests, it was assumed that the serial time for the high range case, could be extrapolated form the time required for the simulation to run with 36 processors. The average parallelization efficiency for the HR tests was found to be 86.43%, while the overall parallelization efficiency for both the low and high range scalability tests was found to be 96%.

The localised sub-linear speedup may be improved by pursuing the following directions:

- The PETSc toolkit can be complied in two modes - debug and non-debug. For performance measurements, the non-debug mode is recommended. In the debug mode, large number of additional checks are performed, which can negatively reflect upon the code performance. For the version of PETSc currently being used on HPC Wales clusters, only the debug version was found to run successfully. Therefore, there is scope for further improvement by just using a different compiled version of PETSc, when it becomes

available in due course.

- Since just iterative solvers are employed for solving the algebraic systems resulting from the discretization, the overall scalability is a function of not just the quality of parallelization but also of the convergence of the solvers employed, which in turn depends on the quality of the preconditioners used. Therefore, unless scalability studies are neutral to solver-preconditioner choices, the speedup figures for codes involving iterative solvers must be looked at with this paradigm in mind. Although cumbersome, measuring the scalability for all possible combinations of solvers and preconditioners will help visualize the true scaling exhibited by such codes.

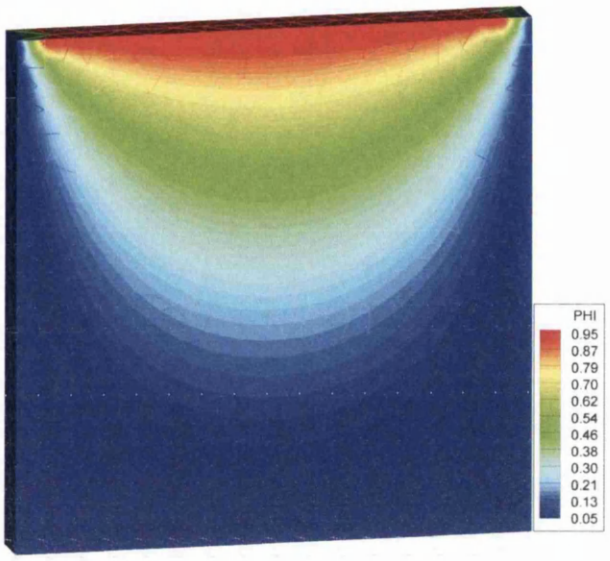
- By further optimising the domain decomposition strategy employed, it might be possible to prevent localized sub-linear behaviour in the excessive partitioning limits (which are mesh dependent).

4.7 Monolithic treatment for pressure

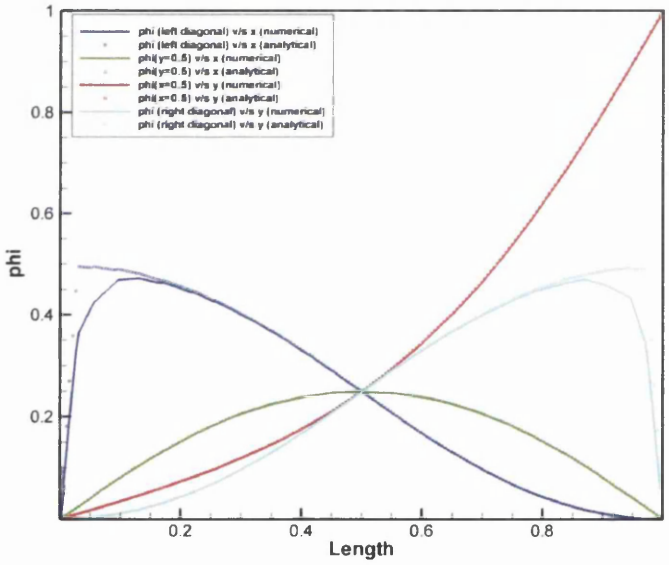
In this section, unlike the previous cases which used the CBS scheme in its classical - split version, the monolithic scheme proposed in section 2.4 will be used. The process of monolithising the CBS scheme, generates pressure stabilization terms in the mass conservation equation thereby circumventing the LBB restriction while using equal order interpolations for both velocity and pressure.

Since this is a single equation framework, the usual (split) code flow changes and therefore the single equation framework is tested first via the solution of a simple Laplace equation on a cuboidal domain[4]. This test is not strictly necessary but is merely used to test the single equation components of the code. Dirichlet boundary conditions are imposed on each of the four rectangular lids. A unit dirichlet

[4]TCP192



(a)



(b)

Figure 4-28: Testing the single equation framework of IFENS by solving the poisson equation on a 3D cuboidal domain: (a) Contour plot with the mesh superimposed (b) Comparison of the numerical and analytical solutions along several polylines

boundary condition is imposed on the north lid. The east, west and south facing lids have a zero dirichlet boundary condition imposed. The numerical solution is presented in Figure 4-28a. A very coarse, nearly-uniform mesh (superimposed on

the solution contours), with just 6210 tetrahedron elements and 2192 nodes was used for this test. A comparison with the analytical solution, which is given by,

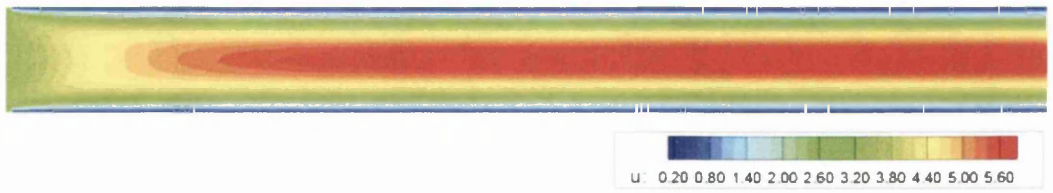
$$\phi = \sum_{n=1}^{\infty} \left(\frac{2(1-(-1)^n)}{n\pi} sin\left(\frac{n\pi x}{L}\right) \frac{sinh\left(\frac{n\pi y}{L}\right)}{sinh\left(\frac{n\pi W}{L}\right)} \right) \tag{4.4}$$

was performed by extracting data along multiple lines from a 2D slice (z = zmax = 0.1 units). A good agreement was observed between the numerical and analytical results (Figure 4-28b), except at the regions close to the left and right edges of the north lid. At the north corners, the coarseness of the mesh results in poor application of dirichlet boundary conditions and poor capturing of the high gradients that exist in this neighbourhood of the domain. Also, the average value of phi in the domain centre along the entire thickness was found to be 0.2487 units, which is expected to converge to the expected value of 0.25 units, with mesh refinement.

4.7.1 Flow through pipe: Monolithic CBS framework for solving NS equations[5]

The problem being considered here is identical to the one considered in Section 4.6.4. The performance of the monolithic scheme is assessed here. Figure 4-29 presents the sectional horizontal velocity contours and wall pressure contours. The overall solution was found to be over damped. Even though a parabolic profile is developed at the exit and mass conservation is achieved, the maximum horizontal velocity at the exit is 5.95 cm/s, as opposed to the expected value of 6.602 cm/s. The same was the case with the pressure drop in the fully developed section. The numerical pressure drop was found to be 13.5 dynes/cm^2, while the analytical value is 14.68 dynes/cm^2. These represent an average error of

[5]TCP915



(a)

(b)

Figure 4-29: Testing the monolithic CBS scheme implemented within IFENS for the problem of flow through a prismatic pipe at Re = 100: (a) Sectional velocity magnitude contours (b) Wall pressure contours

8.95%. A similar damped behaviour was observed for transient simulations like flow past cylinder. However, the current results from the monolithic framework seem promising and will be pursued in the future.

4.8 Summary

In this chapter, various validation scenarios were presented from the point of view of checking the overall correctness of the solution procedure used in this research. Various aspects like mesh renumbering, iterative linear solvers, parallelization and effect of number of processors were considered and verified to be correct by running simulations tailored to test specific aspects. In order to check the correctness of the schemes and the parallel software that was written from scratch to encode these schemes (IFENs), the four common benchmarks, namely lid driven cavity, backward facing step, flow past cylinder and flow through a prismatic tube, were executed and benchmarked. Two scalability studies were also presented and the

overall parallelization efficiency for the low and high range scalability tests was found to be 96 %. Finally, the single equatio framework of IFENs was tested via solution of the Laplace equation. Flow through arteries being the primary application of this research, the test case of flow through a prismatic tube was repeated for validating the CBS scheme in its monolithic form.

Chapter 5

Patient Specific Geometries

5.1 Introduction

With the fluid solver/software developed in this research being validated in Chapter 4, this chapter utilizes the software to solve Navier-Stokes equations on high definition computational domains, representative of carotid bifurcations within real patients. These meshes are three dimensional and are made up of tens of millions of linear tetrahedral elements, typically with structured refinement near the walls to accurately capture the steep velocity gradients experienced in these regions.

5.2 Carotid anatomy

A reference to the carotid typically includes 3 vessels - common carotid artery (CCA - parent vessel), internal carotid artery (ICA - branch 1) and external carotid artery (ECA - branch 2) which together constitute a bifurcation. These carotid bifurcations occur in pairs on either side of the neck. Their typical appearance and placement within the human arterial tree is shown in the MR-

Angiograph of Figure 5-1 [94]. The left and the right CCA differ in their points of origin. While the left CCA originates directly from the Aorta, the right CCA emanates after an extra level of branching. The ICAs supply oxygenated blood to the brain while the ECAs supply blood to the face, scalp, skull, and meninges, hence carotids play an especially important role. Severe blockage or narrowing of the carotid artery (stenosis) may therefore lead to stroke. The medical literature suggests that regions such as entrances of branching arteries, like carotid bifurcations, and bends, tend to develop conditions favourable for atherosclerosis [103, 63, 109, 52, 110, 57]. Another region of interest in the carotid arteries is called the carotid sinus or bulb. It is usually located in the ICA, near the bifurcation region. The carotid sinus contains baroreceptors that modulate blood pressure. The complex heamodynamic patterns found in these regions seem to promote the deposition of platelet thrombi and biochemical reactions in the inner lining of the vessels. The propensity of carotid bifurcations to develop stenosis, as well as their clinical importance led them to be chosen as the zone of study for this research.

5.3 Boundary conditions

As illustrated in Figure 5-1, carotid bifurcations are not terminal vessels. They occur in the midst of a complex, branching vessel network. The ICAs form a bridge between the CCA and the *circle of willis*. The *circle of willis* is a complex branching network that supplies blood to the brain, as a result there will be a back pressure experienced within the ICA, towards the bifurcation. The ECA on the other hand, trifurcates and bifurcates away from the carotid bifurcation, resulting in a cumulative back pressure in the ECA close to the bifurcation. Since the CCA is directly connected to the Aorta, which is relatively larger in diameter, flow disturbances exist in the inlet section of the carotid bifurcation, due

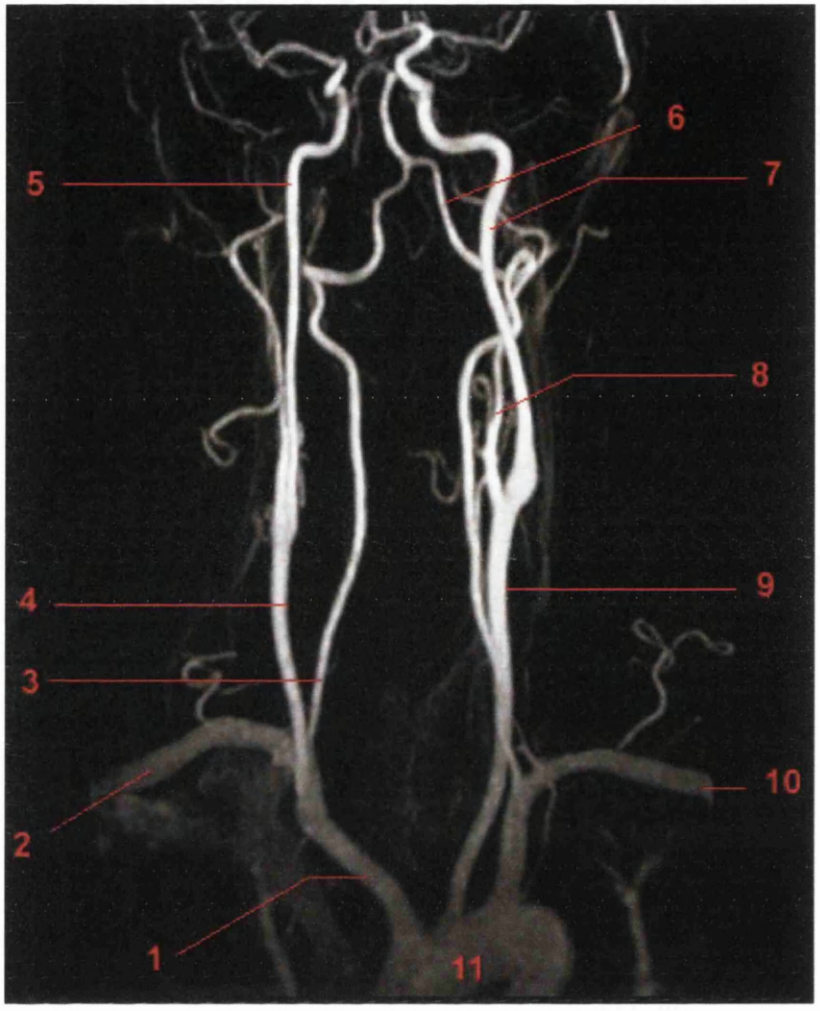


Figure 5-1: Location of carotid bifurcation in the neck [94]. Legend: (1) Brachiocephalic trunk (4) Right Common carotid artery (5) Right internal carotid artery (7) Left internal carotid artery (8) Left external carotid artery (9) Left common carotid artery (11)Aorta

to the sudden change in cross-sectional area. This makes the process of imposing boundary conditions highly non-trivial in computational studies like these. Ideally, pressure and/or velocity measurements at the extremities of the bifurcations under consideration, will be valuable. However, such detailed readings often involve complicated procedures (e.g. 4D flow MR imaging) and use of invasive

devices (like catheters [1]). Also, the heart itself pumps blood by a wringing motion that is often likened in the literature to the *wringing of a wet towel* [127, 99]. A pair of crescent shaped vortices are formed in the left atrium, which make their way into the Aorta and the pulmonary artery [108, 135]. This suggests there would be some vorticity effects retained in the blood entering the CCA, on either sides, which must be accounted for in the construction of the inlet velocity profile of the CCA. No velocity or pressure boundary data was available for the carotid geometries being used in this research. The way forward under these conditions is to assert a paradigm which demonstrates the potential of predicting physiologically correct results, when the correct boundary conditions will become available. Until then, surrogate data serve to substitute for the real boundary conditions. This may result in the solution lying in a space away from the physiological range, but this is merely an artefact of inputting erroneous data into the model.

Under these conditions, the velocity profile generator referenced in 3.4.2 was employed to generate a physiologically realistic, pulsatile velocity profiles across the boundary faces (Womersley profiles). Some studies, for e.g. [17] impose a Womersley profile on both exits. However, from the point of view of imposing a flow split across the two branches, just imposing one velocity profile is sufficient. Owing to the property of mass conservation, the flow at the other exit gets adjusted automatically. Doing so, the system is less constrained by not imposing the spatial distributions in velocity at the other exit. This lets the system naturally develop a velocity gradient along the face of the free outlet.

In split schemes, like the CBS, it becomes mandatory to also impose a dirichlet pressure boundary condition in the pressure poisson solve of step 2. Some studies, for e.g. [142], use equal pressures at both the exits, which seems unrealistic considering the unequal back pressures from the downstream beds of ECA

[1]Catheters are medical devices that can be inserted in the body to treat diseases, perform surgical procedures or take measurements (e.g. pressure)

and ICA. Balossino et al. [12] also found that the imposition of equal dirichlet pressure at the exits was erroneous when a stenosis occurs in one of the carotid branches. The same behaviour was reported in abdominal aortic bifurcations [153]. Therefore, the free outlet (where there is no velocity profile imposed) was chosen to impose a dirichlet pressure boundary condition (on the entire outlet face). Hyun et al. [74] used a similar combination of pressure and velocity dirichlet boundary conditions, with zero pressure outlet condition for the ECA. In this research, several simulations [2] confirmed that the flow field and pressure difference (between ICA and ECA) remained the same irrespective of the point of application of the dirichlet pressure boundary condition. It was observed that imposing a zero pressure on the ICA outlet resulted in a negative pressure in the ECA branch, while imposing zero pressure on the ECA outlet resulted in a completely positive pressure field. However, the pressure difference at the exits of ICA nd ECA remained approximately the same, irrespective of the location of zero pressure imposition. Imposing zero pressure at a fixed distance from the inlet, in the CCA, also resulted in the same pressure drop between the ICA and ECA exits.

5.4 Mesh convergence

Since the flow field is highly dependent on the geometry of the flow domain, it was important to use meshes that were close representations of the true geometries. In this research, high quality meshes containing up to 14 million tetrahedron elements were used. The large number of mesh elements made it possible to render smooth representations of the non-uniform cross sections found in carotid bifurcations. The trimmed carotid bifurcation geometry presented in Figure 5-2 was considered to run a series of flow simulations (Geometry 1). Another impor-

[2]TCP768:p=0 on ICA exit; TCP771:p=0 on ECA exit; TCP770:P=0 at a fixed height from the inlet in the CCA

tant mesh feature, especially with linear elements, is the steep velocity gradient capturing capability. The no slip condition imposed on the walls, results in the development of high shear stresses at the walls (WSS). These stresses get pronounced at the bifurcation region, where the fast moving fluid mass gets skewed towards the walls due to the flow division. In order to capture these gradients accurately, the regions close to the wall along the entire volume of the mesh were refined with smaller elements. These elements were added in a structured manner via finite number of layers, called boundary layers. Starting off with a purely unstructured mesh, i.e. a mesh with no boundary layers, convergence oriented simulations were carried out for meshes with up to 12 boundary layers. The boundary layers become clearly visible at the inflow/outflow boundaries. Figure 5-3 presents the top/plan view of the ECA exit to illustrate the boundary layers. The velocity magnitudes and pressures in the entire domain were used to assess convergence.

A flow split of 40:60 (ECA% : ICA%) was imposed on the internal and external branches of the carotid bifurcation. This imposition was realised by applying suitably constructed velocity profiles at the exits. A number of factors contribute to the true flow split in the carotid bifurcations. These include heart beat period (systole or diastole) [101], neck position and movement, stenosis [12], bifurcation angle [111], overall carotid geometry, to name a few. Marshall et al. [97] also found that around systole, the sum of ICA and ECA outflow was significantly less than the CCA inflow. Although no conclusive evidence was presented in [97] to explain the ratio of outflow to inflow being significantly less than 1, during systole, it might be possible that distensible vessel walls lead to such apparent mass loss. However, these effects disappear when the flow rates are time averaged over the cardiac cycle. In light of ambiguity arising from the range and types of flow splits available in literature, this research employs a constant flow split.

Although, it was previously mentioned that one exit velocity profile was sufficient

Figure 5-2: Geometry 1: Truncated carotid mesh used for assessing mesh convergence

and realistic to impose, for backward compatibility here the velocity profiles are imposed on all exits and a zero pressure was imposed on the ECA exit. A no slip condition exists on all walls. A peak velocity magnitude of 54.9876 cm/s was imposed at the CCA which corresponds to a mean flow rate of 1.45516 cm^3/s. The harmonics used in the construction of the velocity profile are presented in Table 5.1. These resulted from the fast Fourier transform of a measured aortic waveform [107, 17]. The period of the resulting transient wave was 0.61144 s, which corresponds to a heart rate of 98.128 bpm (beats per minute). Flow rate as a function of time for one cardiac cycle, is presented in Figure 5-4. All the velocity profiles are in phase, because disturbances propagate instantaneously in incompressible flows.

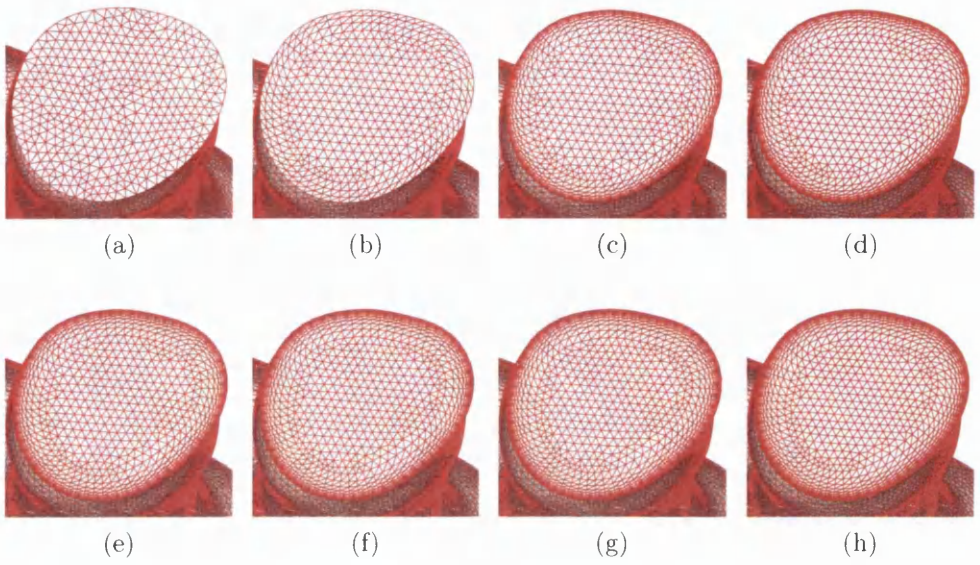


Figure 5-3: Mesh boundary layers, as visible in the plan view of the ECA: (a) No boundary layers (b) 3 boundary layers (c) 7 boundary layers (d) 8 boundary layers (e) 9 boundary layers (f) 10 boundary layers (g) 11 boundary layers (i) 12 boundary layers

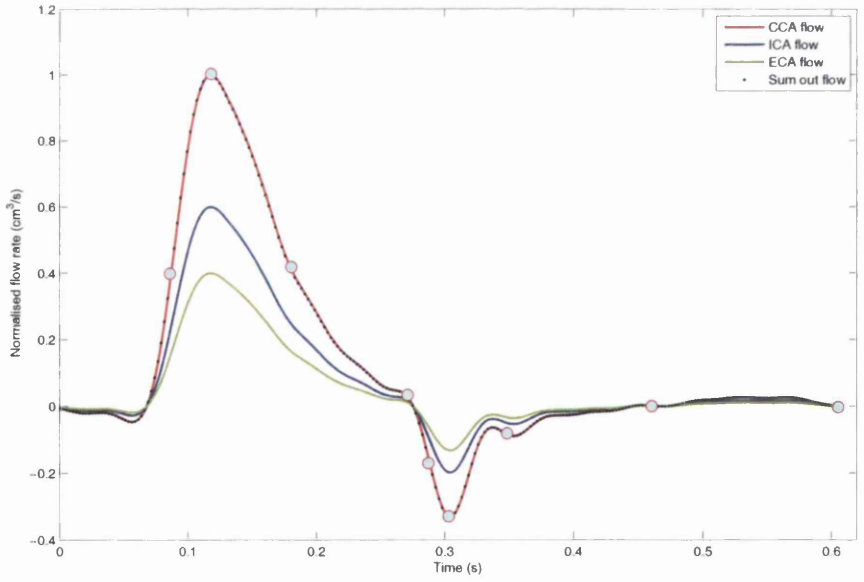


Figure 5-4: Cardiac cycle: velocity profile as a function of time (the filled circles show the time points of interest for the plots of Figure 5-7)

Harmonic	Frequency (Hz)	Amplitude	Phase (rad)
1	0	126.88125	0.00000
2	1.63548	219.04800	-1.59841
3	3.27097	156.06450	3.04933
4	4.90645	80.25450	1.81371
5	6.54193	57.64425	1.22483
6	8.17742	56.19338	-0.107859
7	9.81290	25.00121	-1.55024
8	11.44840	17.10540	-1.40587
9	13.08390	24.24746	-2.93862
10	14.71940	8.16308	1.45781
11	16.35480	10.90725	2.74547
12	17.99030	12.69634	0.490823
13	19.62580	2.756033	-2.48452
14	21.26130	5.983050	-0.254379
15	22.89680	5.026950	-2.84223
16	24.53230	2.558115	0.376408

Table 5.1: Harmonics used for the construction of the velocity profile used for assessing convergence

Figures 5-5 and 5-6 present the results from meshes representing the domain shown in Figure 5-2, but containing different number of boundary layers. The peak and peak time averaged values were used in the construction of these plots. Both velocity magnitudes and pressure converged really well. This becomes evident from the slope of the plot in the bottom figures of 5-5 and 5-6, which is almost horizontal for the points concerning 11 and 12 boundary layers. With respect to the 12 boundary layer case, the peak and time averaged errors for the mesh with 11 boundary layers was found to be 0.07% and 0.03% for velocity magnitudes and 0.04% and 0.06% for pressure, respectively. This demonstration of the invariance of solution variables with mesh refinement is expected and physically realistic.

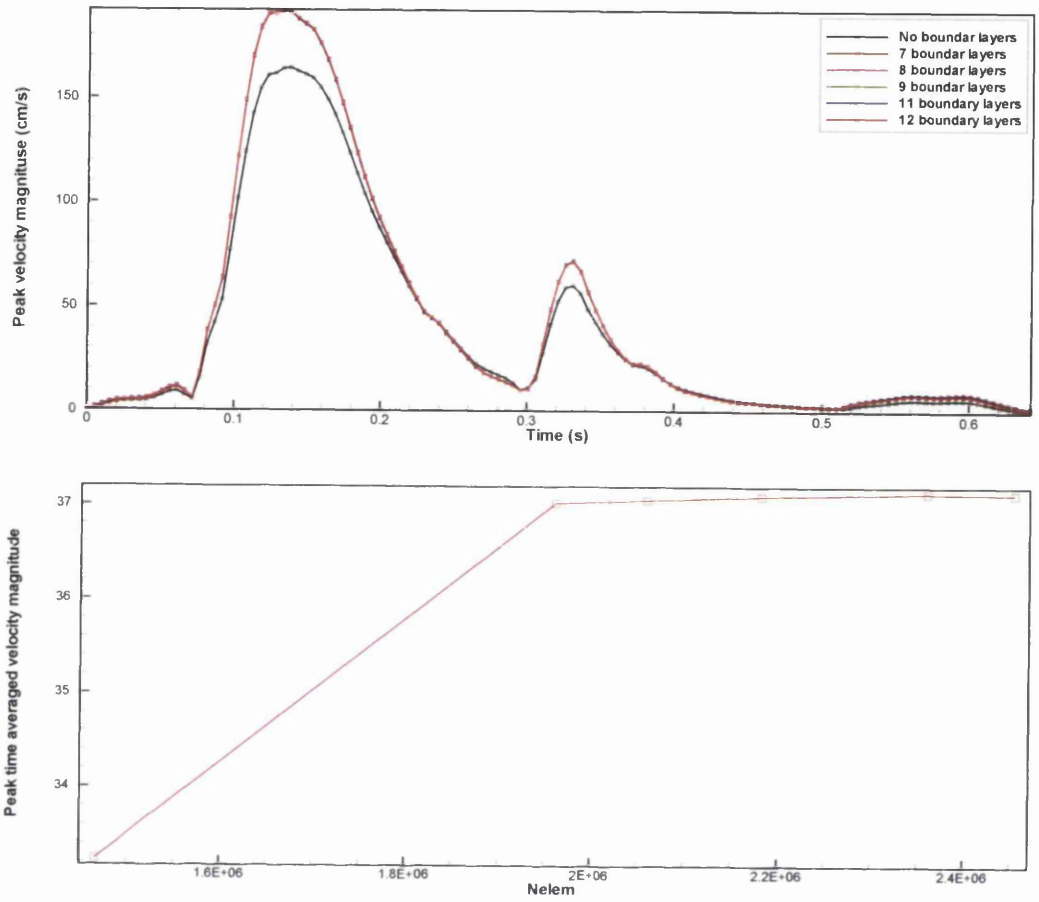


Figure 5-5: Convergence of velocity magnitudes as a function of boundary layers for the geometry of Figure 5-2

5.4.0.1 Flow field - Geometry 1

The visualization of flow field becomes challenging with large data sets like these. Data storage, even in binary/unformatted files, results in file sizes of several gigabytes. Since a no slip condition is imposed on all the walls, all flow data becomes visible only after extracting 2D slices from the 3D domain. Depending on the location of the slices, some flow features might not get captured. Also, since the simulations are transient, certain flow features may be visible in the extracted slices only during certain sensitive instants in time. To be able to select suitable slicing locations at the correct instants of time is laborious and

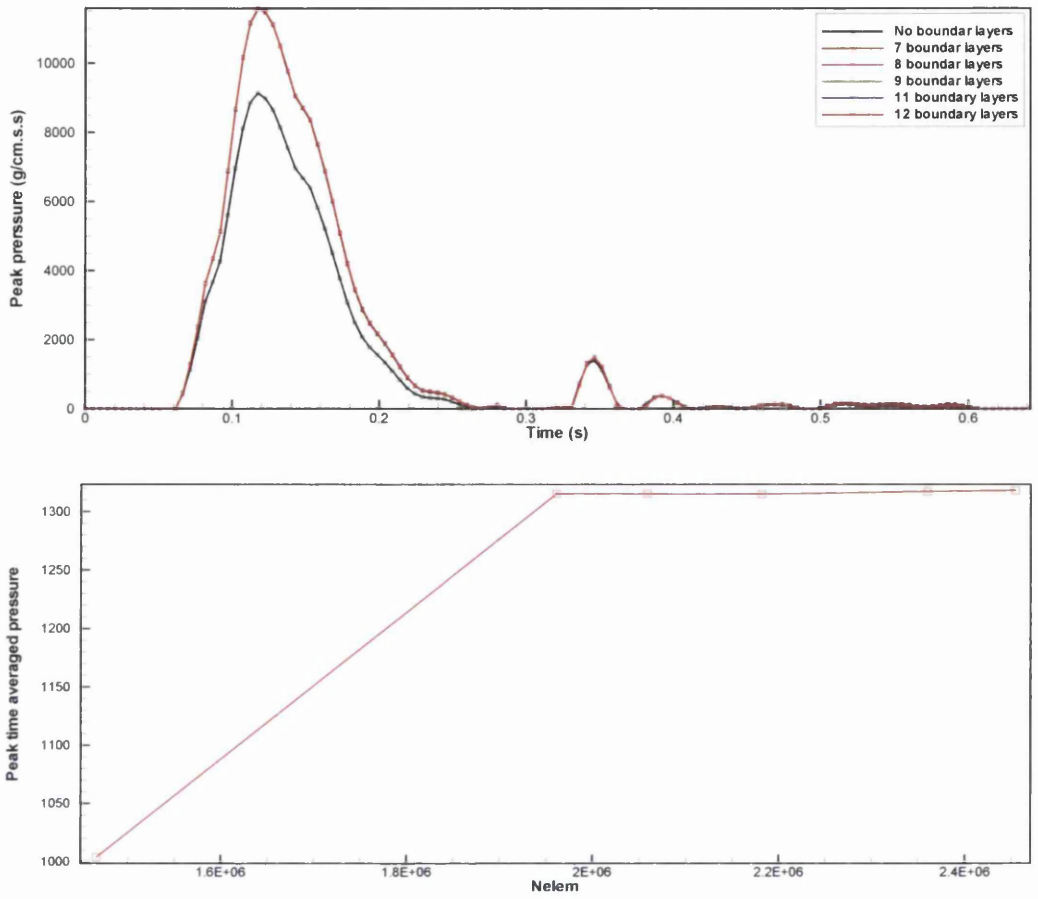


Figure 5-6: Convergence of pressure as a function of boundary layers for the geometry of Figure 5-2

time consuming. Like in this case, the use of different number of boundary layers for the same domain, exacerbates the visualization process.

Figure 5-7 presents the velocity and pressure contours, extracted at different instants in time. A streamtrace plot is also superimposed on the contours of velocity magnitudes to show the trajectories of fluid particles. These results were generated from the finest mesh, that contained 12 boundary layers. The primary direction of flow within carotid bifurcations is from the CCA to the ICA/ECA. Just at the start of the cardiac cycle the flow direction is briefly reversed. At the start of systole, as heart pumps out the fluid mass, all the flow within the

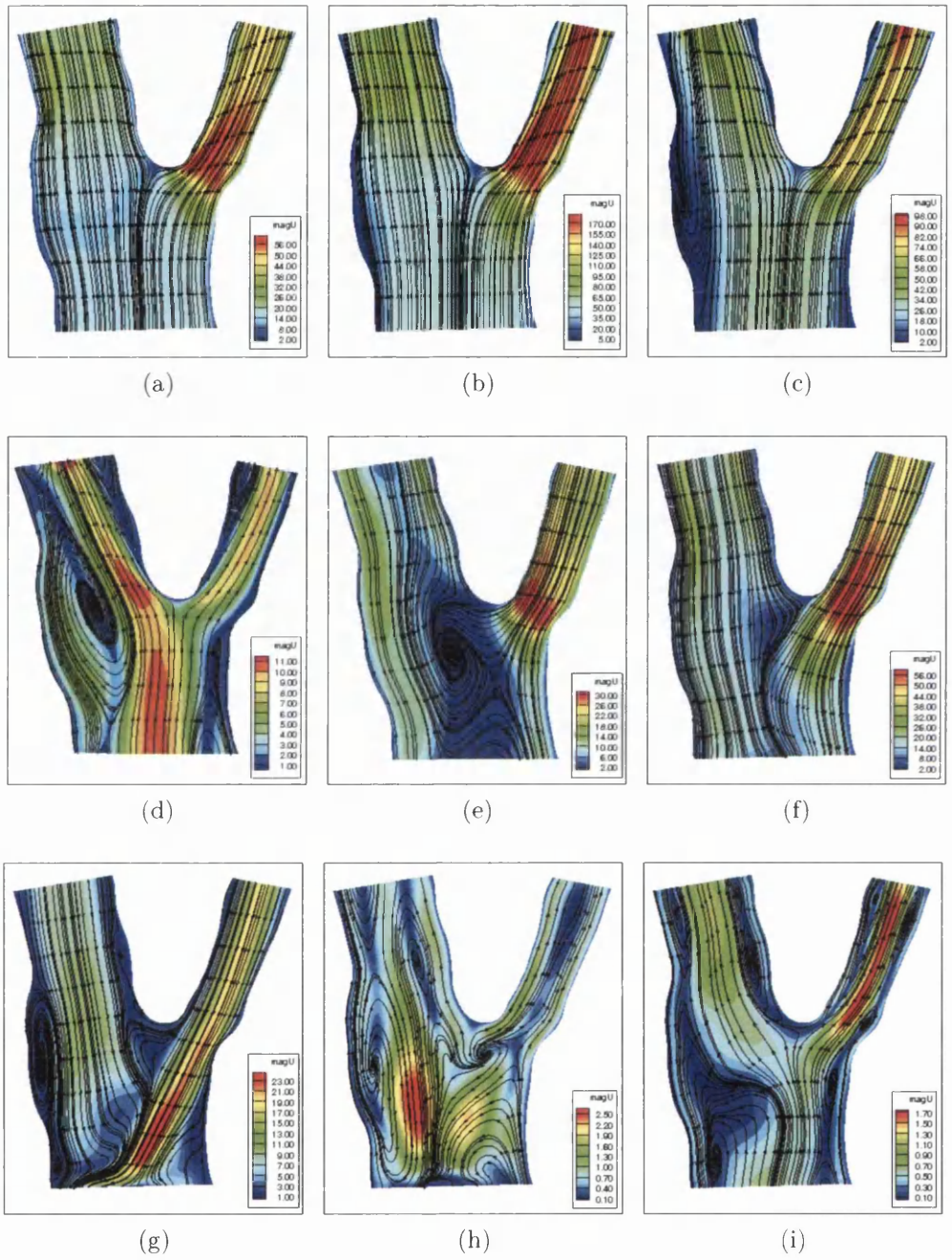
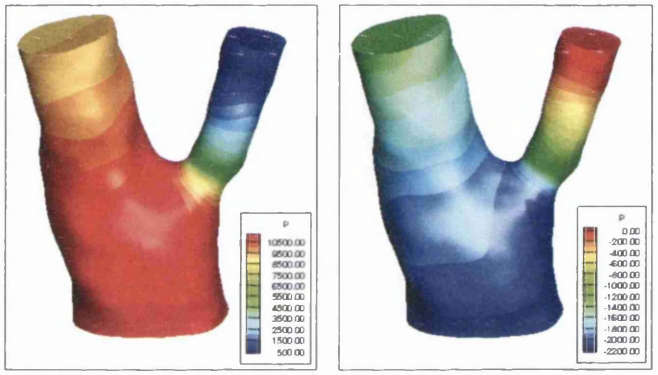


Figure 5-7: Comparison of velocity fields as a function of time, in an approximate central, 2D axial slice: (a) Mid acceleration (t=0.0882 s), (b) Peak flow (t=0.1215 s), (c) Mid deceleration (t=0.1882 s), (d) End deceleration (t=0.277 s), (e) Mid acceleration during flow reversal (t=0.293 s), (f) Peak reversal (t=0.3039 s), (g) Remnant flow start (t=0.347 s), (h) Mid remnant flow (t=0.459 s) and (i) Remnant flow end(t=0.6078 s)

carotid bifurcation is directed outwards (Figure 5-7a). The fluid accelerates as it flows through the ECA, to maintain the specified flow rate under reduced flow area. High pressures are developed at the CCA inlet relative to the exits. The pressure drop between the CCA and ECA (pressure constrained) is greater than that between CCA and ICA. Since the cross-sectional area of the ECA is smaller, it exerts greater resistance to blood flow and a larger pressure drop occurs at this region. This pressure drop increases at peak flow (Figure 5-8a). A peak velocity of 190.505 cm/s occurs in the ECA during peak flow.



(a) (b)

Figure 5-8: Instantaneous pressure fields: (a) Peak forward flow (b) Peak reversed flow

In the mid deceleration phase, a recirculation zone appears on the left side upstream of the bifurcation, owing to the geometry driven flow separation. The next time zone of interest is the flow reversal phase, when the boundary conditions drive the flow in the opposite direction for a short period. Around the end of the deceleration phase, the recirculation zone significantly extends into the inside of the domain and the peak velocity develops in the ICA. As the flow reversal takes place, the recirculation zone occurs just below the bifurcation point, as the fluid travelling towards the inlet from the 2 branches shears a pocket of fluid held around the stagnation point [3] at the bifurcation. However, at peak flow reversal,

[3]Stagnation point: The valley formed between the ECA and the ICA, where the fluid is brought to rest.

the streams travelling from the two exits coherently flow outwards, without any flow separation. During this phase, i.e. diastole, as the flow enters the remnant phase of very low flow, the high frequency flow reversals, induce a very complicated flow field of small magnitudes (Figure 5-7g). The time until next diastole witnesses the generation of multiple vortices close to the walls of the entire domain. These results suggest that flow disturbances occur at periods other than peak flow, especially when there is flow reversal.

Figure 5-9 presents the wall shear stress and the oscillatory shear index. At the bifurcation, where the majority of the fluid mass gets skewed towards the inner wall of the ECA (Figure 5-10), large velocity gradients occur with respect to the no slip walls, resulting in the largest wall shear stresses in this region. In this case, the peak time averaged wall shear stress was found to be 1736.07 dynes/cm^2. The time averaged wall shear stress (WSS_Tavg) is defined as,

$$\text{WSS\_Tavg} = \frac{1}{N}\sum_{t=1}^{t=N} ||\mathbf{t_s}|| \tag{5.1}$$

where, N is the number of time steps and $\mathbf{t_s}$ is the surface traction vector, defined as,

$$\mathbf{t_s} = \mathbf{t} - (\mathbf{t}\cdot\mathbf{n})\mathbf{n} \tag{5.2}$$

where, $\mathbf{n}$ is the normal and $\mathbf{t}$ is the traction vector defined as,

$$\mathbf{t} = \boldsymbol{\sigma}\cdot\mathbf{n} \tag{5.3}$$

where, $\boldsymbol{\sigma}$ is the cauchy stress tensor, defined as,

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{11} & \tau_{12} & \tau_{13} \\ \tau_{21} & \sigma_{22} & \tau_{23} \\ \tau_{31} & \tau_{32} & \sigma_{33} \end{bmatrix} \tag{5.4}$$

While high values of WSS have the potential of causing rupture in the vessels, regions with low and oscillating WSS seems to promote the development of atherosclerosis [25, 86].

The oscillatory shear index (OSI) is a dimensionless quantity lying between 0 and 0.5 [86]. OSI is a measure of the flow directionality over time. When the flow mostly remains unidirectional, i.e. when there are no cyclic variations in the WSS vector, its value tends to zero. Flow disturbances that result in secondary flows and flow reversals make the OSI tend towards 0.5. A value of 0.5 indicates 180° deflection of the WSS direction. OSI is defined as,

$$\mathrm{OSI} = \frac{1}{2}\left(1 - \frac{\tau_{mean}}{\tau_{abs}}\right) \tag{5.5}$$

where, τ_{mean} is the mean wall shear stress, defined as,

$$\tau_{mean} = || \sum_{t=1}^{t=N} \mathbf{t_s} || \tag{5.6}$$

and τ_{abs} = WSS_Tavg.

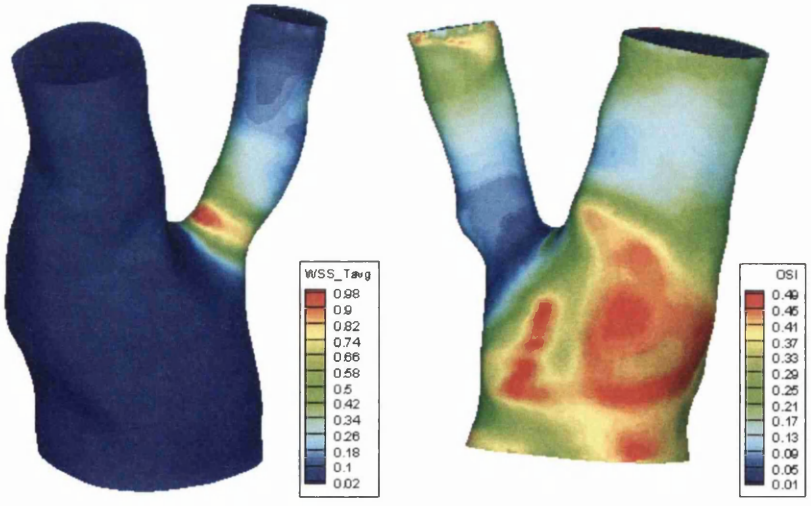


Figure 5-9: (L) Normalised time averaged wall shear stress (R) Oscillatory shear index

OSI predictions of Figure 5-9 are consistent with the velocity fields of Figure 5-7. In the time instants selected, the flow remained mainly unidirectional in the ECA entrance, therefore, the OSI in this region is expected to remain low. Just before the bifurcation, in the CCA, the flow remains disturbed with travelling vortices for majority of the cardiac cycle, making it the region of highest OSI. It can also be observed that regions of high WSS_Tavg experience the lowest OSI and vice versa. In the remaining sections of this chapter, 3 more carotid bifurcations will be analysed.

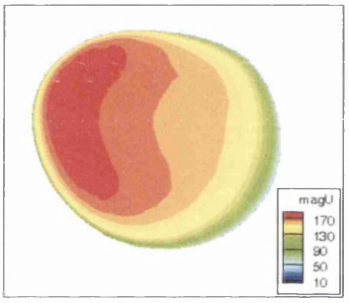


Figure 5-10: Illustration of flow skewing at the ECA close to the bifurcation

5.5 Geometry 2

In this section, unlike the geometry considered for convergence testing, a more complete carotid bifurcation geometry is considered (Figure 5-11), though it still is a truncation of the carotids in situ. A slight narrowing of the CCA occurs before the bifurcation point and a sharp bend feature was observed in the ECA downstream of the bifurcation. The carotid bulb appears normal in the anterior view, but is constricted when viewed from the rear. The mesh used to represent this geometry consisted 6,934,199 tetrahedron elements and 1,185,109 nodes. Following the convergence results, meshes with 10 boundary layers will be considered from this point onwards, unless otherwise stated. This provides an even balance

between accurate solution capturing and moderate simulation costs. There is also some experiential bias in selecting 10 boundary layers.

The velocity profiles used for this geometry are slightly different and adopted from the work of Holdsworth et al. [68], who characterised the blood velocity waveforms in carotid arteries of 17 normal volunteers, analysing 3560 cardiac cycles. The temporal flow variations and the resulting harmonics are presented in Figure 3-14 and Table 3.1, respectively. An important difference between this velocity profile with respect to the one used before, is the absence of flow reversal. The current waveform is also slightly longer, with a period of 0.9195 s, which corresponds to a heart rate of 65.2 bpm. As explained in Section 5.3, just 1 velocity profile is imposed here at the exit, which in this case is the exit plane of the ICA. The zero dirichlet imposition on pressure will therefore apply on the ECA exit plane. A total of 240 time steps were used to span the total period. The total CPU-time consumed for this simulation [4] was 2,487,944.25 seconds on 72 processors, which corresponds to a wall-time [5] of an impressive 9.59 hours. This involves solving a non-linear, sparse system of size 3,555,327 $\times$ 3,555,327, a linear sparse system of size 1,185,109 $\times$ 1,185,109 and a correction of 3,555,327 degrees of freedom, in every time step, along with all the pre/post processing and parallelization tasks.

5.5.1 Flow field - Geometry 2

Figure 5-13 presents the contours of velocity magnitude, plotted at several sections perpendicular to the length axis, at 7 different instants of important flow transitions of the cardiac cycle. The single visible colour bar, based on peak flow, is applicable to all time instants. As the flow approaches the bifurcation during mid acceleration it deflects from the inner walls of both the ICA and ECA to-

[4] TCP877

[5] wall-time = $\frac{\text{CPU-time}}{\text{nprocs}}$

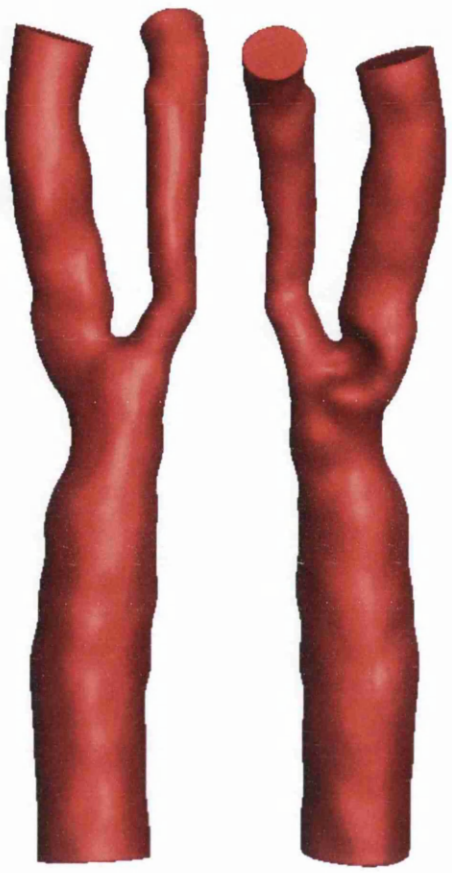


Figure 5-11: Mesh 2: (L) Anterior (R) Posterior

wards the respective exits. This effect can be clearly seen in an animation across time, where the fast moving chunk of fluid appears to be swaying from the inner to the outer walls of the bifurcation, while travelling towards the exit. At peak flow, the fluid accelerates through the ECA forming a zone of the fastest moving fluid at the point where the ECA bends sharply. At peak flow, the velocity in the ECA is 278.2 cm/s, while that in the ICA is around 185.79 cm/s. These values are in very good agreement with the available ultrasound measurements, for this very same geometry, in-vivo. The ultrasound readings predicted a peak systolic velocity of 281 cm/s and 181 cm/s in the ECA and ICA, respectively (Figure 5-12). This represents a mean error of 1.8% between the numerical and measured values. However, the location of the peak values within the ICA and

ECA was not available and hence could not be compared with the results of this simulation.

For this geometry, the peak-inlet Reynolds number (based on the average inlet velocity magnitudes at peak flow) and average-inlet Reynolds number (based on the time averaged inlet velocity magnitudes) were found to be 938 and 253, respectively. Since the critical Reynolds numbers for this geometry (and in general, for patient specific geometries) is difficult to experimentally establish, one cannot make flow regime predictions based on the Reynolds number.

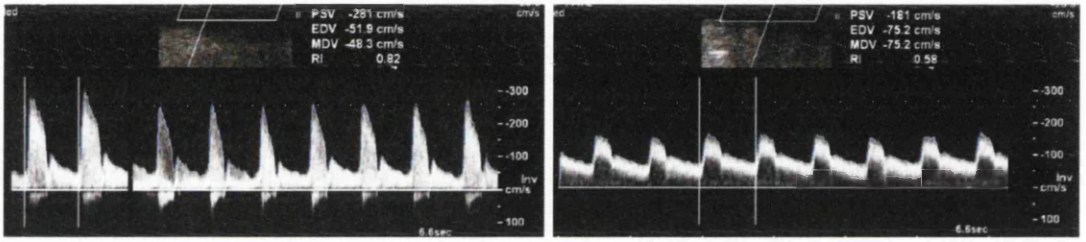


Figure 5-12: Ultrasound measurements for geometry 2

As the flow progresses into the mid deceleration phase, flow separation begins to occur in the ICA, which becomes visible as patches of very slow moving concentric and/or crescent shaped regions in the flow field (visible in slices of the deceleration phase and beyond). Figure 5-14 presents the secondary field, which is plotted by eliminating the axial velocity component. Downstream of the bifurcation, both in the ECA and ICA, small vortices are generated close to the wall, which decay towards the exit. The orientation of particle traces throughout the length of the ICA and ECA suggests that the fluid swirls in making its way to the exit.

Throughout the cardiac cycle, the peak domain velocity is maintained in the ECA. Flow in the CCA is initially skewed towards the ECA, before hitting the narrowing in the CCA upstream of the bifurcation. However, from the mid deceleration phase until the end of cycle the flow is skewed towards the ICA. A series of streamtrace plots (coloured based on peak velocity magnitudes) are presented in

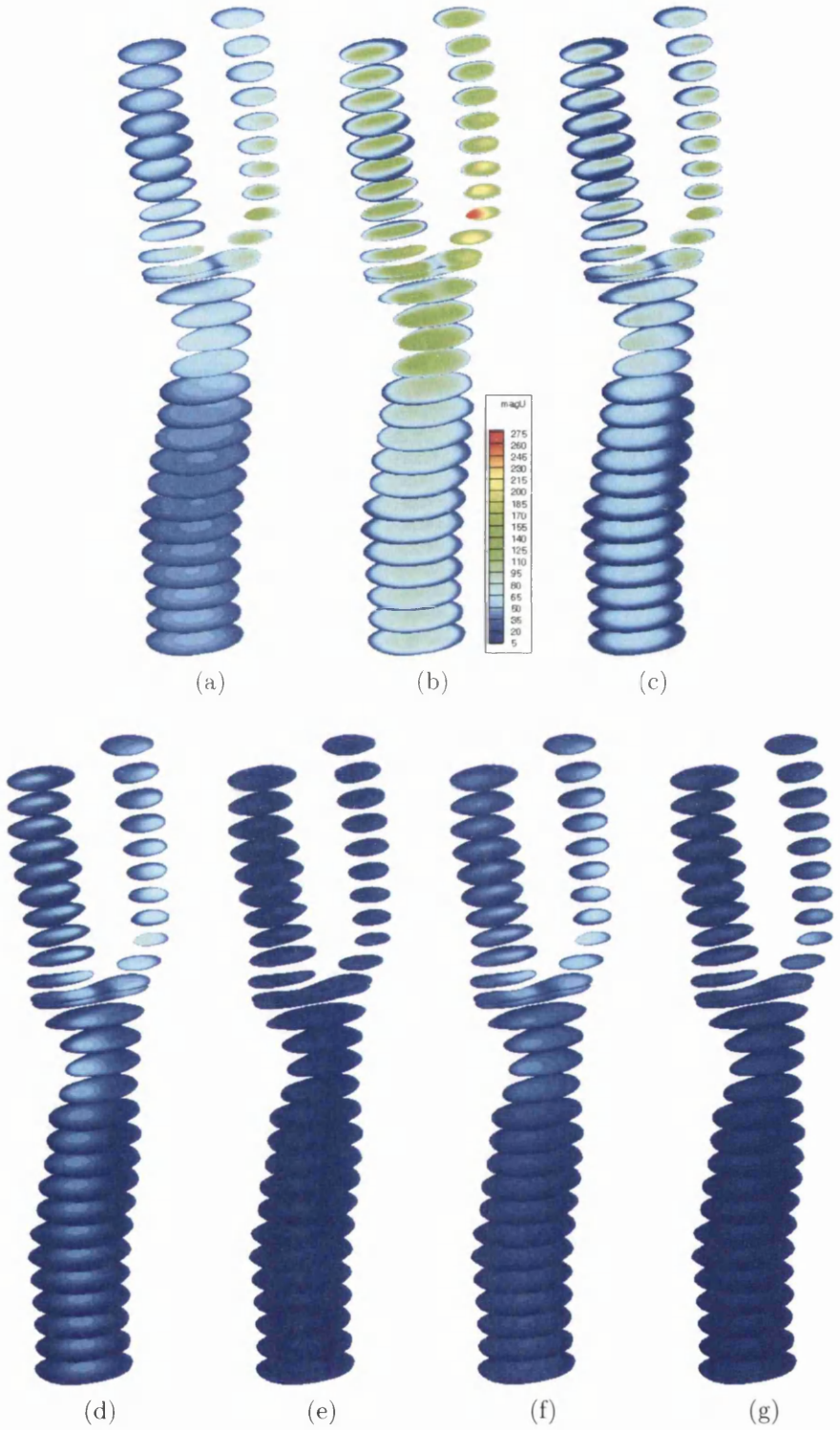


Figure 5-13: Sectional velocity profiles for geometry 2 at various time instants: (a) Mid acceleration (b) Peak flow (c) Mid deceleration (d) Minimum remnant flow (e) Peak remnant flow (f) Mid remnant flow

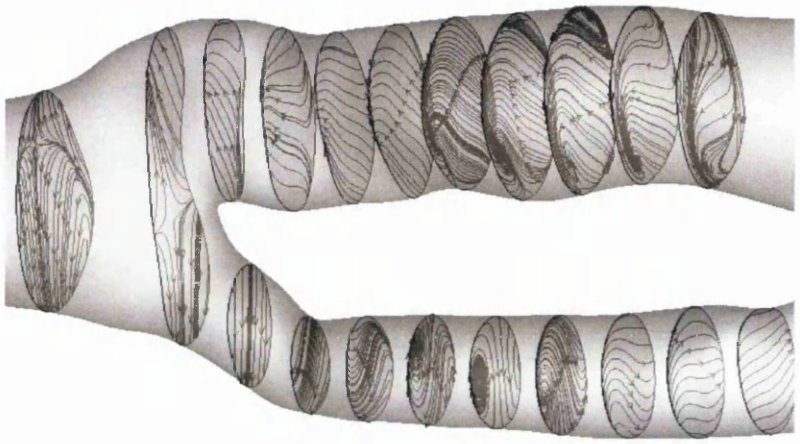


Figure 5-14: Streamtraces of secondary flow revealing the flow disturbances.

Figure 5-15: Streamtrace plots coloured by velocity magnitudes: (a) Peak flow (b) Lowest flow (c) Peak remnant flow

Figure 5-15 to reveal the flow disturbances that arise during the cardiac cycle. At peak flow, the fast moving particles travel in a smooth, undisturbed manner. At the sinus bulb region, slow moving vortices are generated and directed diagonally

opposite towards the inner wall of the ICA, as the fluid flows from the bifurcation point to the exit. A similar effect can be observed on the opposite side, in the ECA. Like with the first geometry, flow disturbances of small velocity magnitudes arise in the remnant flow phase of the cardiac cycle. Disturbed flow is maintained in all the branches of the carotid bifurcation. Downstream of the bifurcation, a big recirculation zone is formed in the ICA, while helical flow patterns are observed along the entire length of the ECA. During peak flow in the remnant phase, the flow regains a much ordered state. Based on the results of geometry 1, it appears that the flow field in geometry 2 would have been more disturbed, if a flow reversal were to be imposed at the boundaries.

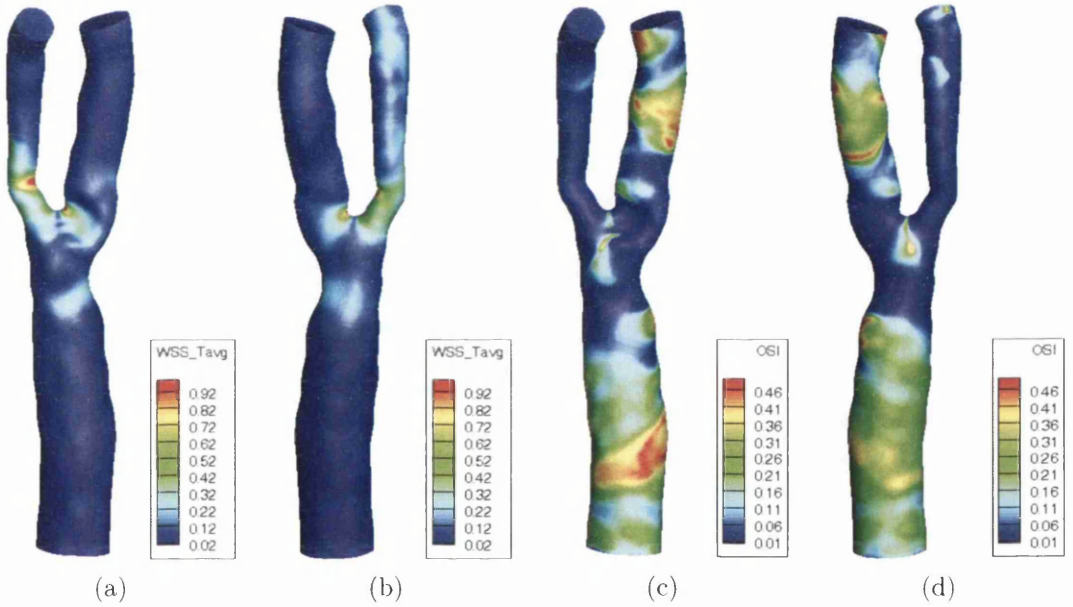


Figure 5-16: Maps of time averaged normalised WSS and OSI: (a,c) Posterior (b,d) Anterior

The time averaged WSS contours, along with the OSI map is presented in Figure 5-16. The largest wall shear stresses are localised at one location each, in the ECA and ICA. Owing to the division of flow and the presence of a stagnation point, high WSS values are expected at the bifurcation. However, the sharp bend feature in the ECA, downstream of the bifurcation and the unusually constricted

ICA, in the sinus bulb region seem to be the reason behind the sharp localization of stresses in the 2 branches. However, on the other side of the sinus bulb, which is normally shaped (without constriction), low time averaged WSS values are observed, which is in line with the findings of papathanasopoulou et al. [120]. The maximum time averaged WSS experienced in this geometry is 2017.51 dynes/cm^2. This value is higher than those quoted in the literature. The WSS predictions of CFD studies are generally higher than the in-vivo measurements. This may partly be due to the poor resolution of the scans [120], which is compensated in the CFD world by the use of boundary layers. The OSI maps suggest that away from the bifurcation, most of the regions in the CCA and ICA experience oscillatory stresses. In the bifurcation zone, a pair of longitudinal bands of moderately high OSI is observed, making them favourable sites for atherosclerotic plaque deposition. In the WSS plots of Figure 5-16, these regions also have valleys of low shear stresses appearing between peaks of high wall shear stresses, making them suitable for plaque deposition.

5.6 Geometry 3

The carotid geometry considered next (Figure 5-17) is relatively complicated in shape and truncated farther away from the bifurcation zone. The ICA and ECA are unusually curved in a semi circular manner as they make their way upwards of the neck. The common trend of the ECA being smaller in cross-sectional area than the ICA, is not valid for this geometry. The opposite seems to be the case. In the ICA a number of constrictions and expansions occur in tandem, with the smallest constriction occurring at approximately the mid length of the ICA. Upstream of the bifurcation, in the CCA, a stenosis occurs, resulting in a sudden reduction in the cross-sectional area. This geometry, by virtue of its atypical configuration, exemplifies the importance of patient specific analyses.

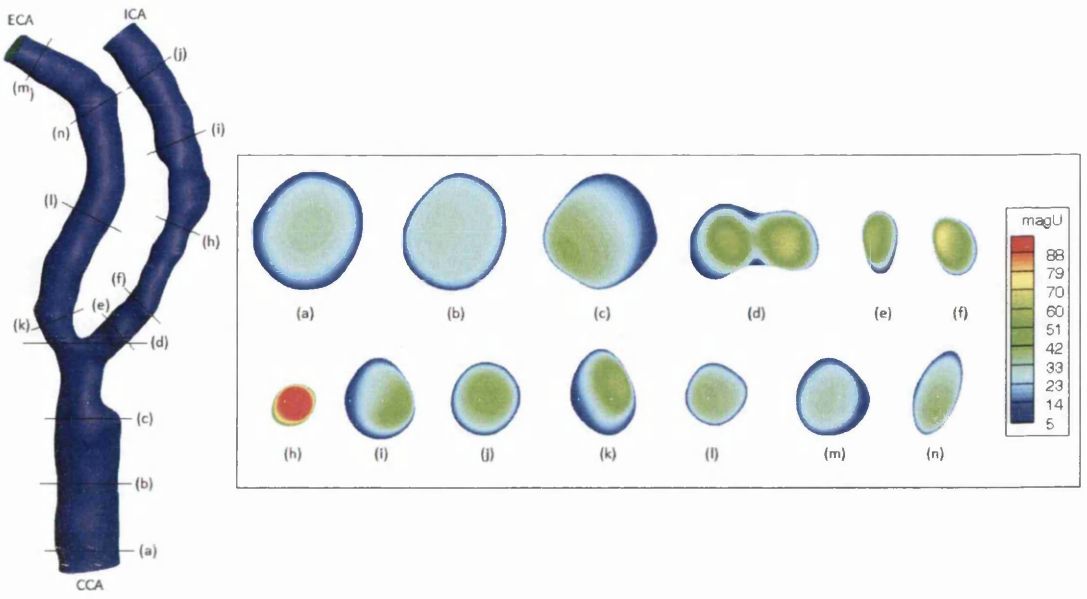


Figure 5-17: Sectional velocity profiles for geometry 3 at mid acceleration

Boundary conditions similar to the ones used for geometry 2, were employed. The harmonics of Table 5.1 were used in the construction of velocity profiles. A transient dirichlet velocity profile was imposed at the inlet and the exit (ECA). Consequently, the zero dirichlet pressure boundary condition was imposed on the ECA exit plane. A mesh with 10 boundary layers, 13,124,417 elements and 2,239,739 nodes, was used to represent geometry 3. The total CPU time consumed with 72 processors was 6,424,563 seconds, which corresponds to a wall time of 24.78 hours.

5.6.1 Flow field - Geometry 3

Figures 5-17 through 5-19 presents the velocity magnitudes at different sections along the length of the carotid bifurcation. The precise locations are presented in Figure 5-17 (left). These sections were chosen based on the geometrical changes occurring at these locations. The size and angular orientation of these slices is unchanged and represents the in-situ configurations. During mid acceleration,

the imposed Womersley profile is maintained only for a short distance from the inlet, after which the geometrical variations induce skewing effects. These skewing effects (towards the ECA), become more pronounced at the stenotic section C. Downstream of the bifurcation, the flow skews towards the inner walls of the ICA and ECA. A peak velocity of 97.32 cm/s occurs at the region of highest contraction. Unlike before, where the peak domain velocities always occurred very close to the bifurcation, the shape of geometry 3 pushed the peak closer to the exit plane. The secondary or in-plane mid acceleration flow field (Figure 5-20a) reveals the in plane vortices of upto 55 cm/s occurring in the ICA close to the bifurcation. Even higher magnitudes of secondary flow exists in the ECA, close to the exit, since the ECA aligns itself almost perpendicular to the CCA (such that the primary flow tends to occur in the secondary direction).

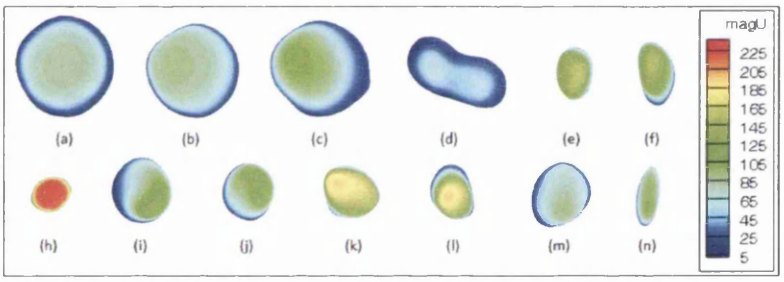


Figure 5-18: Sectional velocity profiles for geometry 3 at peak flow

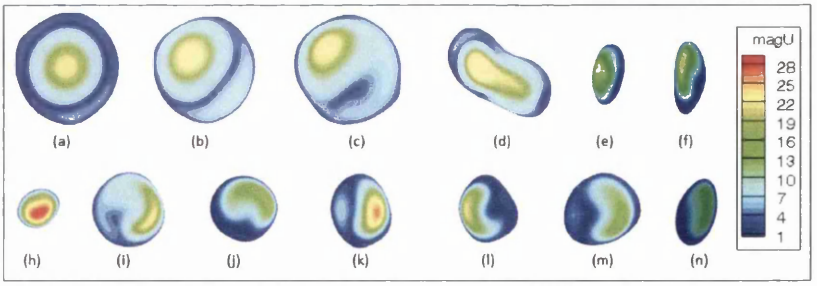


Figure 5-19: Sectional velocity profiles for geometry 3 during the stage of lowest flow

At peak flow, the velocity profiles (both primary (Figure 5-18) and secondary

(Figure 5-20b)) seem to inherit the spatial distributions from the mid-acceleration phase, but the actual magnitudes are relatively higher. The peak velocity occurs in section h with the velocity magnitude reaching 237.02 cm/s. The appearance of high velocity fluid mass very close to the boundaries suggests that high shear stresses will occur in this region. The peak secondary flow on the other hand, appears in the ICA but close to the bifurcation. Secondary flow of upto 100 cm/s was observed. Ultrasound measurements, proximal to the bifurcation in the ICA, were available for this geometry. The peak recorded velocity was 169 cm/s. As per the numerical predictions for this geometry at peak flow, the maximum velocity magnitude occurs in/around section h, but this section is far away from the bifurcation (not proximal, like in the ultrasound). Sections e and f qualify better, to be at close proximity to the bifurcation. A series of 5 sections (perpendicular to the length axis (Z)) were considered in the ICA, beginning from the start of the ICA and ending approximately at section f. The average of peak velocity magnitudes was found to be 168 cm/s. From slice f to slice h, the velocity magnitude increases. Further towards the exit, the velocity magnitude decreases and maintains a near constant peak value, owing the similar flow areas in that region.

As the fluid decelerates into the mid deceleration phase, flow separation occurs in the ECA, as can be observed from the plots of Figure 5-21. The in-plane streamtraces suggest that fluid particles spiral into a stationary point (Figure 5-21a). This is merely an artefact of plotting a 2D streamtrace for a 3D flow field. The volume lines [6] of Figure 5-20a clarify this behaviour. Since, flow near the inner walls of the ECA is swirling as it makes its way towards the exit, the sectional streamtraces appear to lock to a point. A recirculation zone also appears close to the outer wall of the ECA as the fluid changes direction in traversing twin pseudo semi circular paths, while travelling towards the exit.

[6]Volume line: A type of 3D streamline which is not confined to remain on a surface and may travel through 3D volume data.

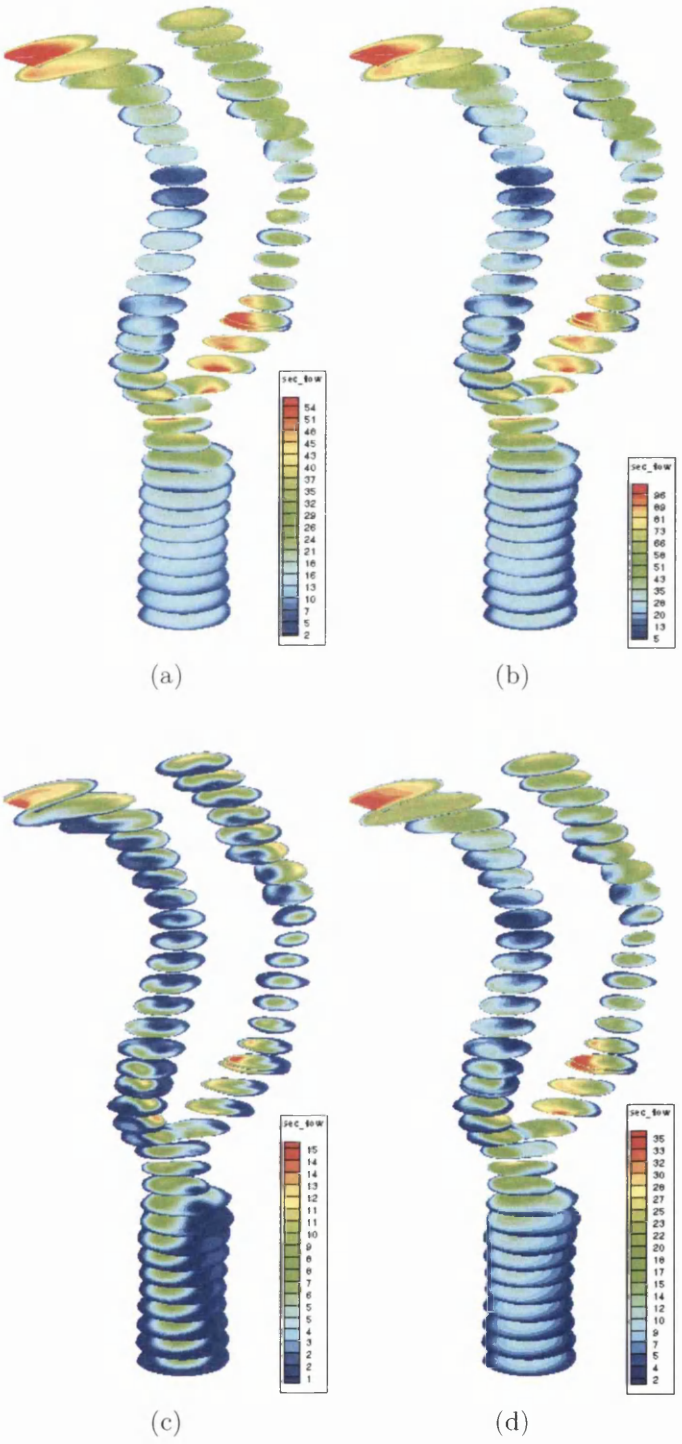


Figure 5-20: Sectional velocity profiles of the secondary flow field: (a) Mid acceleration (b) Peak flow (c) Least flow (d) Peak remnant flow

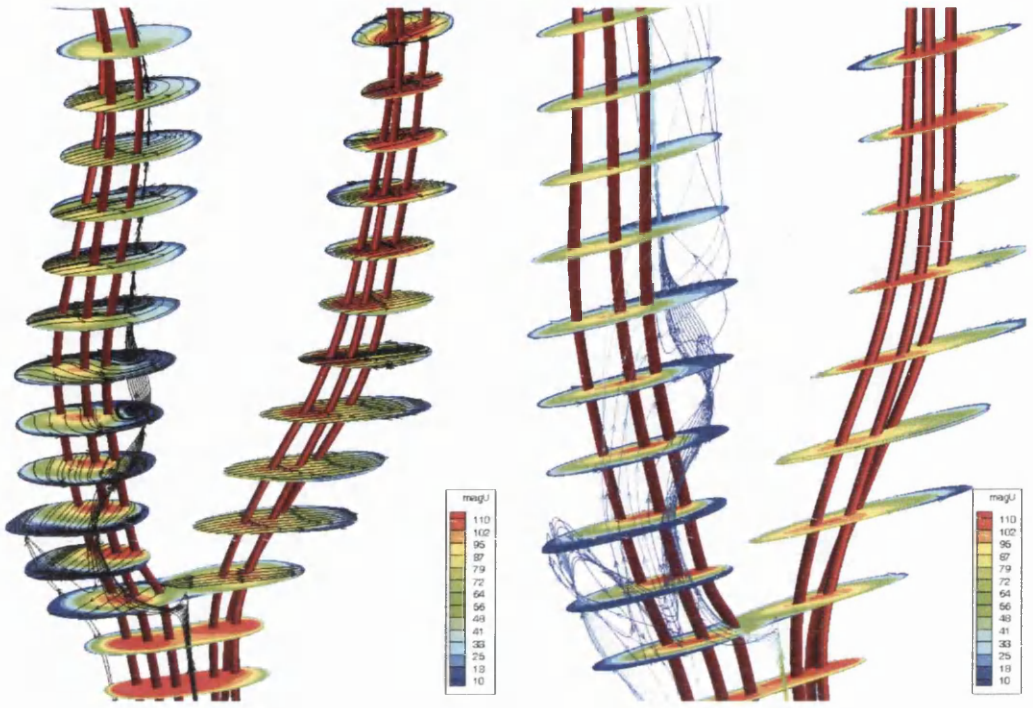


Figure 5-21: Bifurcation region of geometry 3: sectional velocity profiles with streamtraces (surface lines, volume lines and volume rods) superimposed, during mid deceleration

In the time instant of least flow, flow separation occurs in almost the entire length of the carotid (more pronounced towards the exits). These can be identified from the crescent shaped, relatively fast moving patches in the sectional plots of Figure 5-19. The sectional velocity profiles in the low flow time instants tend to show marked variations when compared to the high flow instants, when the bulk of the fluid traverses coherently through the carotid bifurcation. The sectional plots concerning the lowest flow phase (Figure 5-20c) show the presence of in-plane vortices reaching magnitudes of up to 16 cm/s. The peak secondary flow appears in the ICA in regions close to the bifurcation and in a couple of slices in between the mid length and the exit plane.

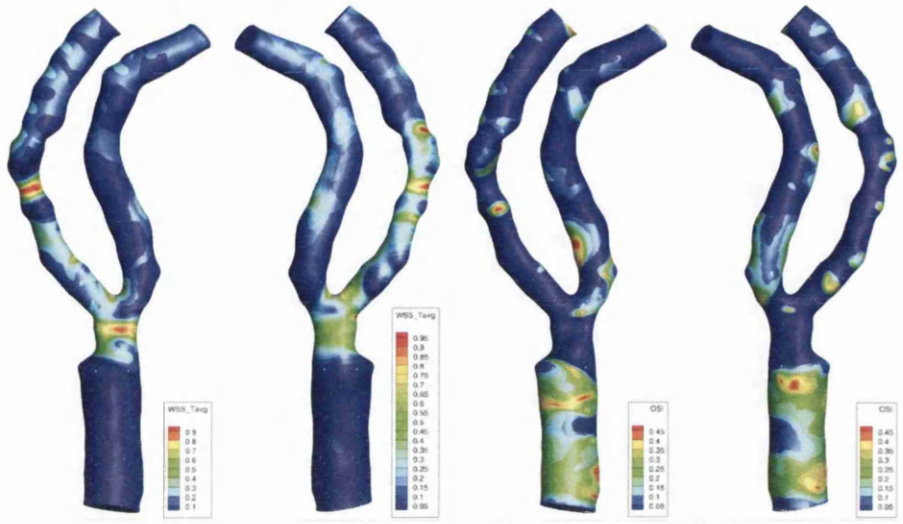
As the flow progresses in to the peak remnant phase, the primary flow field becomes more ordered. Regions of flow separation are observed in multiple slices

of both the ICA and ECA, primarily where the geometrical variations result in changes in flow direction (curvature effects). The secondary flow field (Figure 5-20d) at peak remnant flow, preserves the spatial distributions from the time instant of least flow.

Figure 5-22 presents the time averaged WSS and OSI. Bands of highest wall shear stresses occur in the CCA just before the bifurcation and in the ICA at approximately mid lengths, where the flow direction changes significantly. In the sectional plots for the primary flow field, the highest velocities were recorded in this region during all time instants. The entire lumen in these regions was flooded with fast moving fluid mass thereby resulting in large velocity gradients at the walls. Two very localised spots of high time averaged WSS also appear very close to the stagnation point, on either side. These are the points where the CCA jet impacts the bifurcation and eventually deflects into the two outlet branches. The peak time averaged WSS for geometry 3 was found to be 1421.86 dynes/cm^2. Most of the CCA and ECA experience low shear stresses at the walls during the entire cardiac cycle. As expected, high OSI appears in the CCA in almost the entire length before the stenosis. The ICA also experiences localised zones of high OSI, typically in regions where local expansion occurs. Plotting sectional streamtraces in these expanded locations close to the walls reveals the presence small recirculation zones, thereby resulting in OSI spikes in these regions. A streak of high OSI also appears in the stagnation region of the bifurcation, which is not visible in the views presented in Figures 5-22c and 5-22d.

5.7 Geometry 4

The carotid bifurcation considered in this section represents severe stenosis in the ICA at the bifurcation region (Figure 5-23a). The mesh used to represent this geometry also happened to be the largest mesh used in this research. It



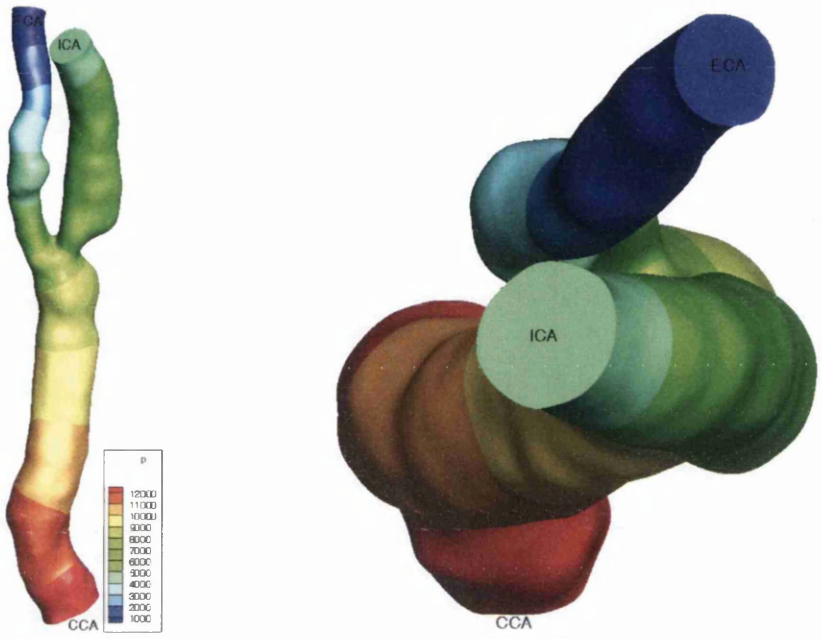
(a) (b) (c) (d)

Figure 5-22: Time average WSS (normalized) and OSI maps for geometry 3: (a,c) Anterior (b,d) Posterior

contained 10 boundary layers, 14,583,254 tetrahedron elements and 2,489,345 nodes. Unlike the previous geometries, the CCA is bent in the inlet region. Since this bend occurs very close to the inlet the skewness effects are neutralised as the fluid travels towards the bifurcation. Also, the ECA and ICA turn in opposing directions towards the exit planes (Figure 5-23b). In the ECA, towards the mid length, a short region of relatively large cross-sectional area appears. The boundary conditions of geometry 3 were reused for this case. The total CPU time consumed on 72 processors was 7,294,710 seconds, which corresponds to a wall time of 28.14 hours.

5.7.1 Flow field - Geometry 4

The plots in Figures 5-25 and 5-27 present the sectional velocity profiles and streamtraces at different time instants. As the boundary conditions effect the flow inside the carotid bifurcation, the fluid particles travel along simple, well defined paths towards the exit. Half way through the mid acceleration phase, peak



(a) (b)

Figure 5-23: Geometry 4 (with pressure contours during mid acceleration phase): (a) Front view (b) Top view

velocities begin to appear mostly in the peripheral locations of the ECA. A peak velocity of 64.23 cm/s was recorded in the mid acceleration phase. As the fluid accelerates further, a recirculation zone begins to appear in the sinus region of the ICA, close to the bifurcation. This recirculation zone launches fluid particles towards the inner walls of the ICA via a slow moving helical jet, clearly visible in Figure 5-25b. The bend feature present in the CCA inlet results in a pronounced swirling effect as the velocity increases. As a result, the streamtraces of relatively slow moving particles tends to skew further, resulting in such particles to traverse slightly longer paths towards the bifurcation. At peak flow, maximum velocity magnitudes of 183.8 cm/s was observed in the ECA, close to the exit in a region of localized narrowing (this is the first observable narrowing in the ECA when viewed from the exit plane). Unlike for geometries 2 and 3, no accompanying ultrasound data was available for this geometry (4).

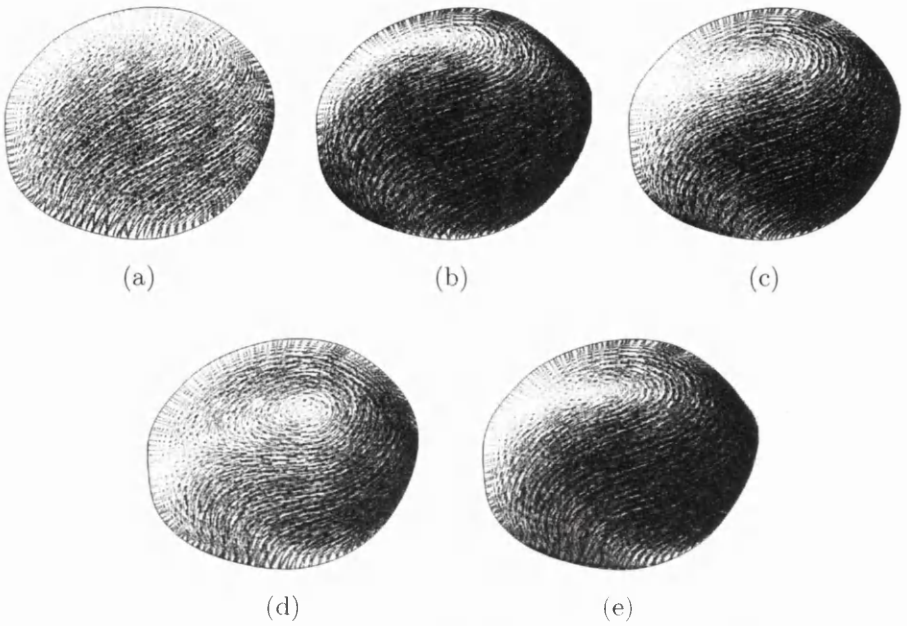


Figure 5-24: Vector plots of secondary field for Geometry 4 in the ECA (mid length section): (a) Mid acceleration (b) Peak flow (c) Mid deceleration (d) Least flow (e) Peak remnant flow

The recirculation zone in the sinus bulb of the ICA grows in size just past the peak flow, before contracting in size. This is because a slight lag occurs between the application of boundary conditions and their effects becoming visible within the domain. The sectional velocity profiles of the mid deceleration phase are remarkably different from those in the mid acceleration phase, especially in the ICA. Although, the peak velocity is maintained at nearly the same value of around 61 cm/s. In the slices close to the bifurcation, in the ICA, flow separation of more than 50% occurs. This can be verified by the presence of relatively large, blue (slow moving) sectional patches appearing adjacent to relatively smaller, green (fast moving) patches, in the corresponding regions (Figure 5-25c). The occurrence of the stenosis at the ICA entrance seems to result in such extreme flow separation. Since the velocity magnitudes in the CCA are smaller during mid deceleration (than during mid acceleration), the fluid has less momentum as it squeezes through the stenosis and is therefore less capable of driving the

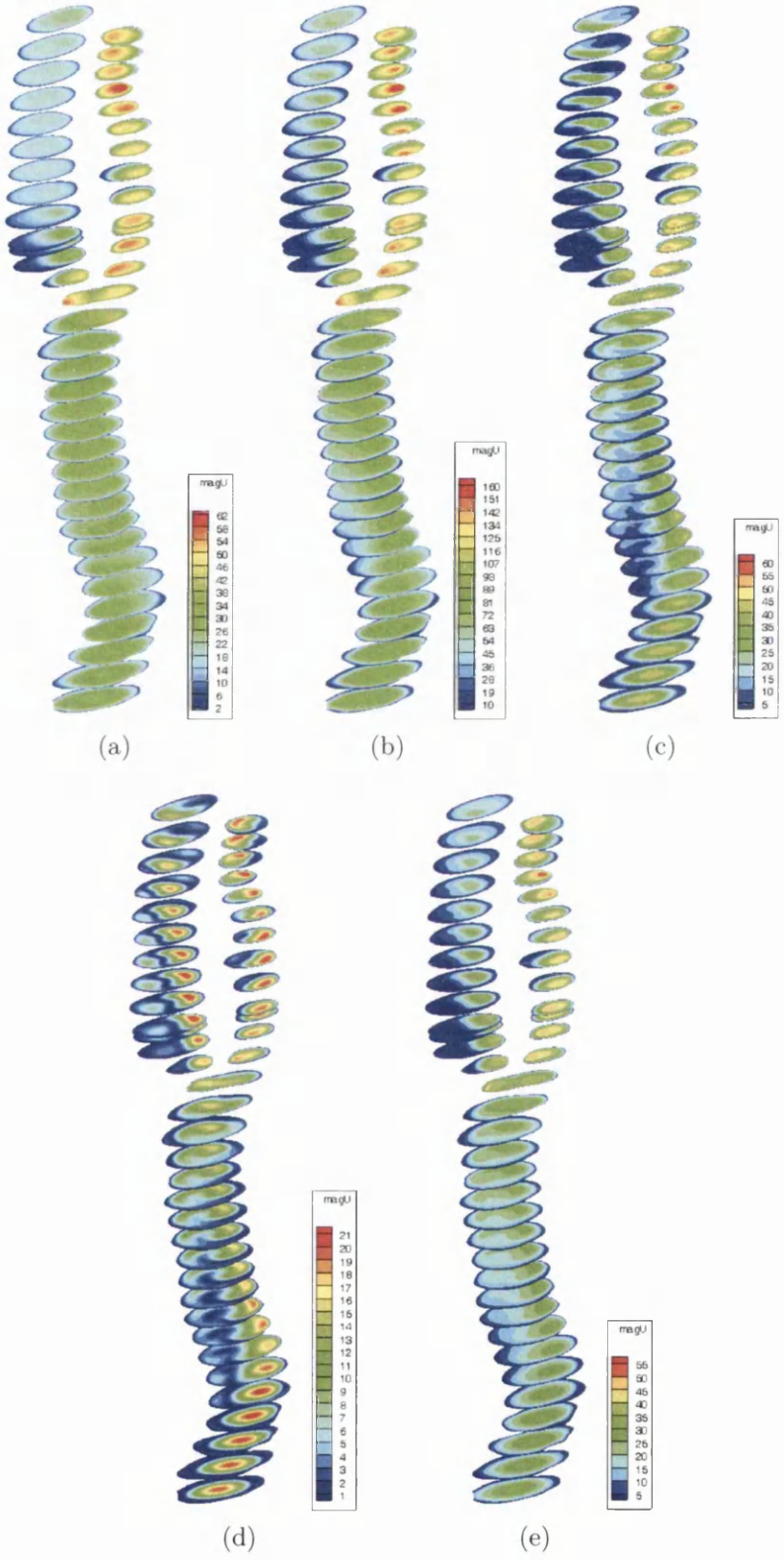


Figure 5-25: Sectional velocity profiles for primary flow field of geometry 4: (a) Mid acceleration (b) Peak flow (c) Mid deceleration (d) Least flow (e) Peak remnant flow

recirculating mass (in the axial direction) that resides in the sinus bulb.

The time instant of least flow seems to give every slice (of Figure 5-25d) an opportunity to accommodate fast moving regions in the range of peak domain velocities (21.96 cm/s). Unlike other time instants, since localization of the highest velocity magnitudes doesn't occur (from the point of view of length-wise distributions), the brightest regions of every section, visibly trace the locus of fluid paths that will be traversed by the bulk of the fluid. Given the geometry, it is somewhat intuitive to see how the fluid bounces off from one wall to another in making its way through the carotid bifurcation. The velocity distributions in the ICA, at peak flow show an interesting characteristic. The recirculation zone extends to almost $3/4^{th}$ the length of the ICA. It not only grows in size but begins to recirculate faster. The shearing effect from the fast moving fluid towards the inner ICA wall contributes towards stirring up the recirculation zone with more energy in the low flow phase. The recirculation in the anti-clockwise direction (Figure 5-26) confirms this. The blue patch that separates the fast moving patches in the sections of the ICA represents the centre of the three dimensional recirculation zone. Beyond the major recirculation zone of ICA during low flow, a second recirculation zone begins to appear in the slices closer to the ICA exit. Some intermediate sections have flow separations occurring on 2 major locations resulting in 3 coplanar and incoherent fluid masses, resulting in complex flow fields. Obviously these regions of recirculation represent regions of low wall shear stress and possible sites for the development of plaque. However, since there was already a stenosis at the ICA entrance, one needs to retrospectively analyse this situation. Under the premise that low WSS results in stenosis, geometry 4 represents an advanced (time wise) case and the converse of the premise also seems true. As the flow progresses into the peak remnant phase, the peak velocities localize towards the ECA exit and the recirculation zones in the ICA bulb shrink considerably in size and velocity magnitudes.

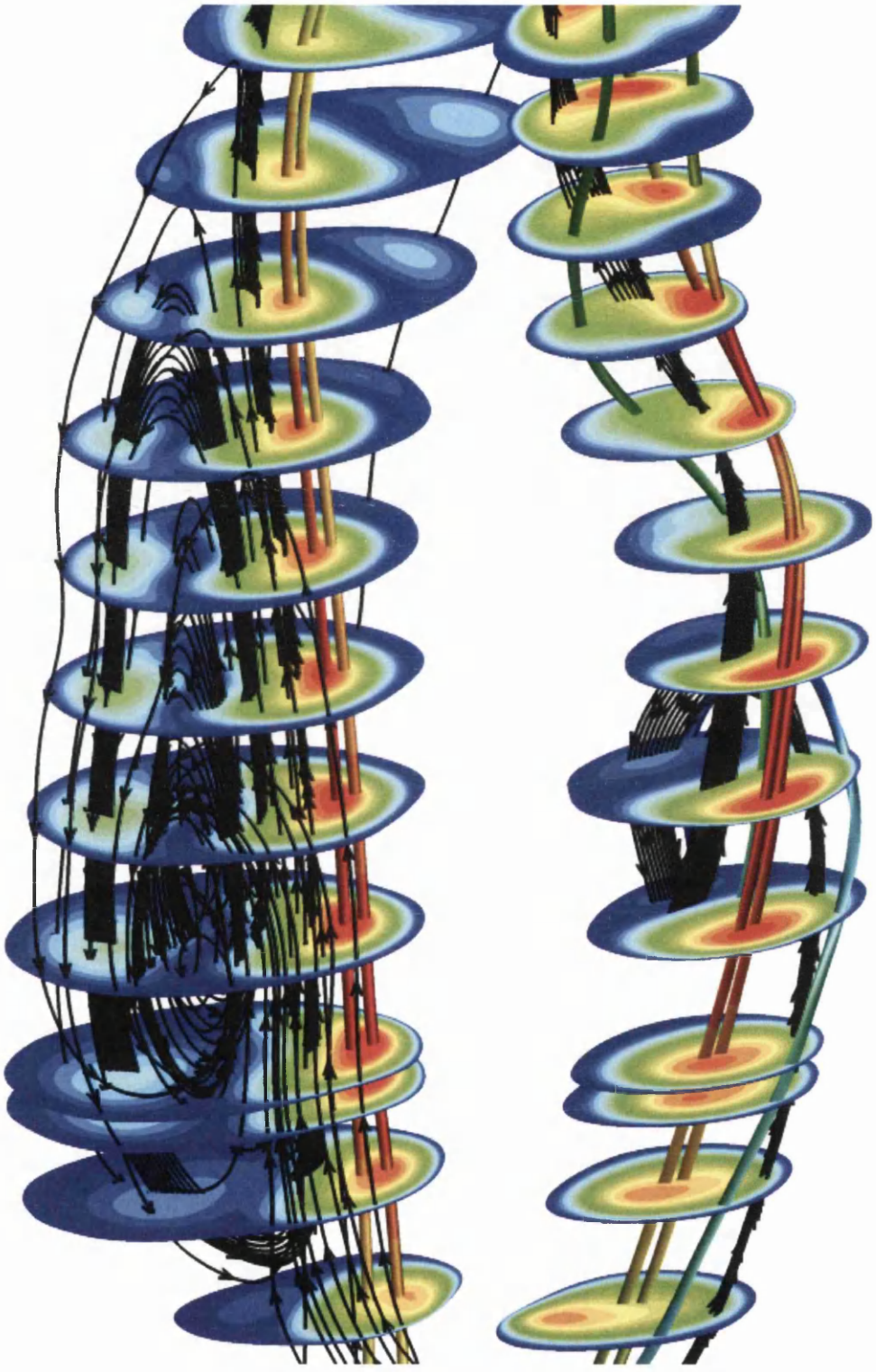


Figure 5-26: Geometry 4: Evolution of recirculation zone during the phase of least flow (Left branch is the ICA)

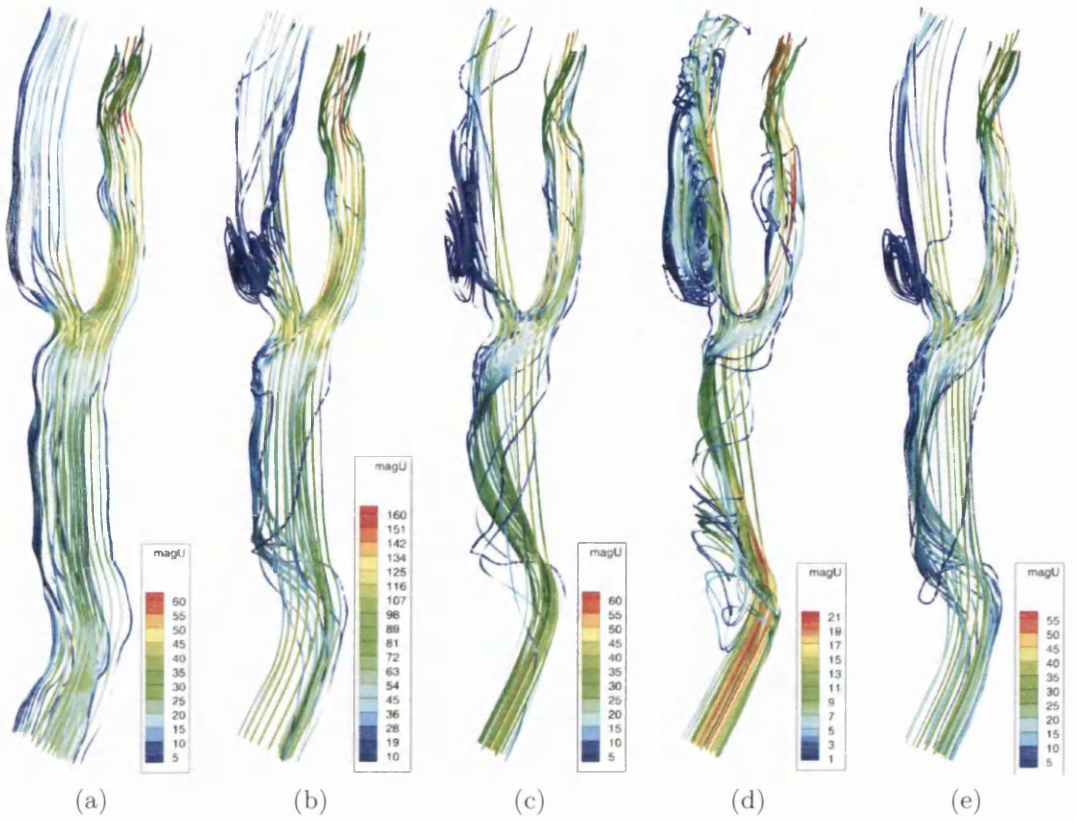
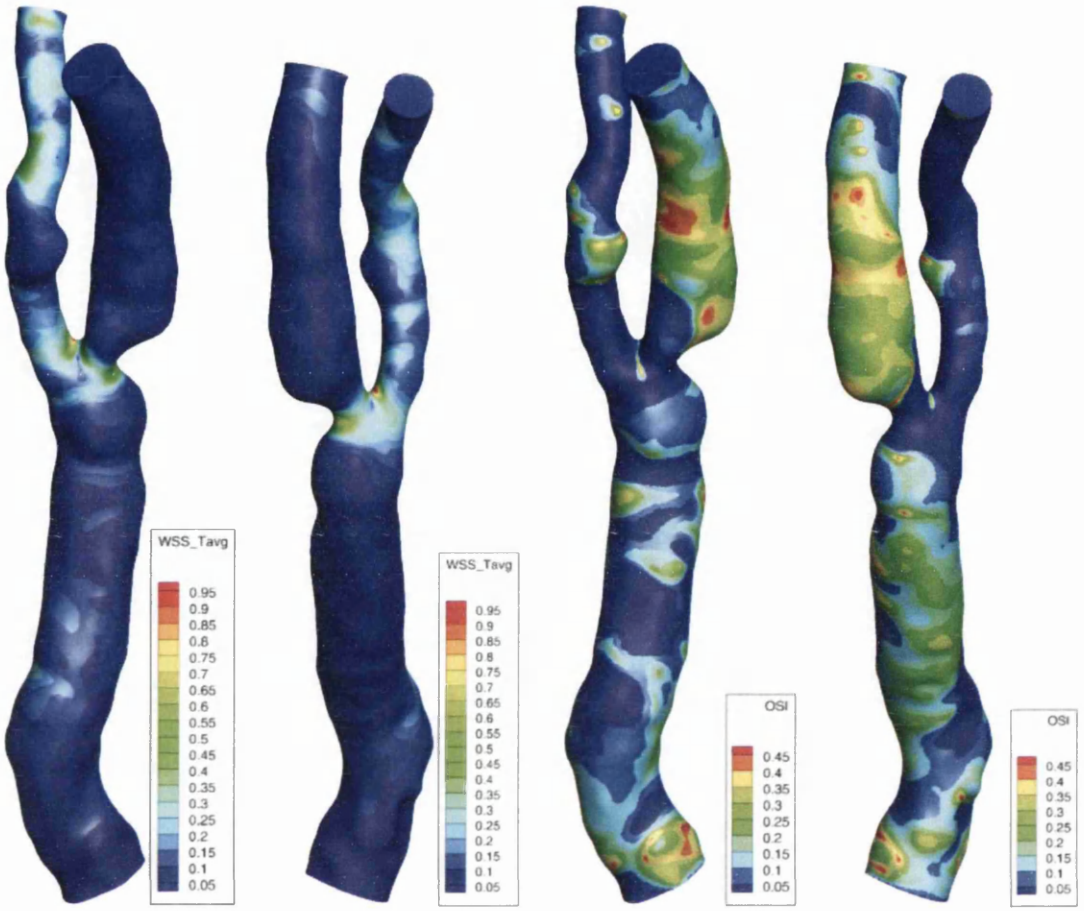


Figure 5-27: Streamtraces for geometry 4: (a) Mid acceleration (b) Peak flow (c) Mid deceleration (d) Least flow (e) Peak remnant flow

Figure 5-28 presents the normalised, time averaged wall shear stresses and oscillatory shear indices for geometry 4. Unlike other cases, the peak time averaged wall shear stress is heavily localized in a single, tiny region appearing at the bifurcation, on the inner wall of the ECA. The peak value being 1495.63 dynes/cm^2. The outer-bend region in the CCA, bifurcation zone and almost the entire ECA experience moderate levels of wall shear stresses. Most of the CCA and ICA have experience minimal shear stresses at the walls. The least stresses are experienced in the outer walls of the ICA, close to the sinus bulb region where extreme recirculation and flow separation occurs. This is supported by the high OSI values appearing on most of the CCA and ICA. The ECA on the other hand just ex-



(a) (b) (c) (d)

Figure 5-28: Time average WSS (normalized) and OSI maps for geometry 4: (a,c) Anterior (b,d) Posterior

periences high OSI in the nodule [7] appearing at mid length, which can now be predicted to contain at least one recirculation zone. Like before, high OSI spike occurs in the stagnation region of the bifurcation.

5.8 Summary

This chapter introduced the human carotid bifurcation, their functions, position in the human arterial network and the reason behind their selection for this re-

[7]Nodule: region of sudden expansion, relative to its surroundings

search. A note on the importance of boundary conditions was presented next. Concentrating on specific regions of the vasculature often presents problems imposing boundary conditions on the extremities of the domain of interest. This fact was illustrated from the point of view of carotid bifurcations and some of the possible boundary conditions commonly used were presented. A full arterial system modelling with the inclusion of the heart, provides an idealistic setting to relieve the burden of applying realistic boundary conditions in studies like these. Such systems, however are often low fidelity (1D/0D) and therefore their predictions are usually inaccurate. As mentioned before, measuring the boundary conditions is an invasive option. A mesh convergence study from the point of view of boundary layers in the wall regions was carried and the convergence of pressure and velocity up to a solution was demonstrated for up to 12 boundary layers. A total of 6 meshes were considered for the mesh/boundary-layer convergence study. In the remaining sections of this chapter, 4 different carotid geometries (1 mesh per geometry) were considered and their flow fields were presented and analysed. Haemodynamic parameters like time averaged wall shear stress and oscillatory shear index were calculated for all geometries considered.

Geometrically, every carotid bifurcation considered was unique. Geometry 1 was a heavily truncated geometry with inclusions of CCA, ECA and ICA sections very close to the bifurcation. It demonstrated all the features of a typical carotid bifurcation with mild stenosis at the ECA entrance. Relative to Geometry 1, the second geometry was truncated far away from the bifurcation and represents the length usually used in carotid bifurcation studies. The ECA represented a sharp bend feature, while the ICA had an occlusion close to the bifurcation which is visible in the posterior view only. Geometry 3 was unusually curved at the exits of the ECA and ICA and had inconsistent ECA-ICA sizes. The ICA had a series of nodules along the entire length and the CCA had an unusual stenosis, a short distance away from the bifurcation (towards the CCA inlet). The mesh

used to represent Geometry 4 was one of the largest with around 14.58 million tetrahedron elements and 2.5 million nodes. Among the geometries considered, it had the most severe stenosis, occurring at the bifurcation region of the ICA. A nodule occurred in the mid-length section of the ECA and the ECA and ICA exits point in different directions. Also the CCA was bent at the exit, making this, the geometry with most peripheral bends.

At peak flow, the peak velocities are always observed to occur beyond the bifurcation. The localization was found to be mostly within the ECA, except for the third carotid geometry, where the peak velocity occurred in the ICA. The ECA and ICA of the third geometry were unusual in their relative sizes, which explains the occurrence of the peak in the ICA instead of the ECA. The second carotid geometry recorded the highest peak velocity among all the cases considered, the value being around 275 cm/s. The sharp bend in the ECA, seems to be the reason behind this. Consequently, the highest time averaged wall shear stresses also occur for this geometry (since the sharp bend results in skewing of the sectional velocity magnitude at this region, very close to the inner wall, which gets captured by the fine boundary layered mesh of this region). Ultrasound data (velocity magnitude) was available for the second (ECA and ICA) and the third (ICA only) carotid geometries considered. The mean error between the numerical results and the ultrasound data, was found to be 1.41 %.

The medical data didn't contain information regarding the precise locations, where the peak velocity magnitudes occurred. However, as described below, in each case the locations of peak velocities could be correlated to the changes in geometry at those locations:

1. In the first geometry, the mild stenosis at the ECA and the fairly constant cross-sectional area along the entire length results in the peak velocity magnitude occurring along the entire ECA segment.

2. As mentioned before, in the second geometry the sharp bend causes localization of high velocity magnitude in the ECA.

3. In the third carotid geometry, the section h, where the maximum velocities occur, is a section of the smallest cross-sectional area in the entire section (this zone may be likened to a converging-diverging nozzle) and therefore the fluid accelerates to maintain the required flow rate, in travelling through this constriction.

4. In the fourth carotid geometry, the fluid accelerates as it enters the ECA, but begins to decelerate thereafter because of the presence of a nodule (local expansion). Past the nodule, the ECA geometry begins to contract to normal state and therefore the fluid accelerates to conserve mass. As the fluid undergoes cycles of acceleration and deceleration, the smallest ECA section close to exit records the maximum velocity magnitude at peak flow. This suggests that the ECA constriction is more severe downstream of the nodule.

The recirculation zones occurring in various geometries were also presented through the streamtrace plots (both 2D and 3D), in this chapter. The sinus bulb region of the ICA, was observed to be the common area for the flow separation to occur, in all geometries. The flow separation and the consequent recirculation zones shrink at peak flow and evidently show up during low flow and flow reversals. In the fourth carotid geometry, a recirculation zone was also observed in the ECA. Generally, recirculation zone(s) appear in regions where a localised contraction or expansion exist. In order for these to show up, one must be careful in selecting the seed (start) points for the generation of streamtraces. When flow reversal is imposed, as in the case of the first carotid geometry, a recirculation zone appears right below the bifurcation point. This is due to the shearing effect from the fluid reversing from the ICA and ECA, on the pocket of fluid trapped below the

bifurcation.

The presence of vortices in the flow field may not be detected, unless the vortex is of a relative large velocity magnitude. The small magnitude vortices that usually occur are hidden under the influence of the dominant velocity components and in-plane vector plots of secondary field reveal their presence (Figure 5-24). In-plane, 2D streamlines may also be used to reveal the secret life of such vortices. Plotting the contours of velocity magnitudes without the primary velocity component (Figure 5-20) also helps in visualizing the vortices in a slightly different way. Contrary to the contour plots of primary velocity field, where the sectional slices indicate the velocity magnitudes approximately normal to the slices, the secondary contours represent in-plane components and are therefore (somewhat) representative of the in-plane vortices.

Maps of time averaged wall shear stress, as well as the oscillatory index were plotted for every geometry considered in this chapter. The highest WSS_Tavg appeared in regions where fast moving fluid mass was skewed toward the walls. The peaks appeared at the bifurcation region, mostly. In geometry 2, the peak also appeared in the sharp bend region. While, for geometry 3, multiple bands of high shear stress occurs in the ICA at its mid section. These peaks were found to occur in the regions of localized contractions of the ICA. The lowest wall shear stresses occur in regions of the CCA and ICA. The opposite is true, just for the third carotid geometry. The OSI values seem inversely proportional to the time averaged WSS. Therefore, most of the carotid surface is flooded with moderately high values of OSI throughout (again, with an exception for carotid geometry 3).

It would also be useful to see, how the WSS and OSI predictions change in the absence of the abnormalities that are present in the carotid geometries considered. Unless, a longitudinal study is undertaken the remodeling of arteries make it difficult to predict the configuration of these geometries under normal conditions.

Chapter 6

Conclusions and Future Research

The global aim of this research was to provide solutions to cardiovascular problems from the points of view of mathematics and computer-modelling. Since the dynamics of self-regulating living systems are complex and tightly coupled, a mathematical description of all the components of such system was difficult to incorporate in a single comprehensive model. Making suitable and realistic assumptions, a framework that simulates blood flow within human carotid arteries was developed, benchmarked and utilised to solve systems with millions of degrees of freedom. The characteristic based split scheme was utilised to solve the non-linear, transient and incompressible Navier-Stokes equations in their non-conservative form. An attempt was also made in the direction of developing a monolithic version of the characteristic based split scheme, with the goal of making this framework extensible to multi-physics problems, like FSI. The monolithic framework of the CBS scheme generates a new stabilization term in the mass conservation equation, that stabilizes the pressure field. This framework has also been fully developed and has currently progressed to the performance tuning phase, which is critical for solving large systems.

The entire framework was cast in a fully parallel, high performance computing

environment. Multi processor parallelism was established using the Message Passing Interface standard, which is currently used by all major supercomputers to communicate messages across processors. PETSc which is a non-trivial, state of the art toolkit (collection of libraries) for solving algebraic systems in parallel, was successfully incorporated and utilized. Both MPI and PETSc were coherently utilised in a Fortran90 software called **IFENs**, which is an acronym for **I**mplicit **F**inite **E**lement **N**avier-Stokes **S**olver. IFENs was developed from scratch during this research. IFENs was an extension of another software developed during this research, called SIFENs, i.e. serial IFENs. SIFENs started off as a purely serial convection-diffusion equation solver. As SIFENs matured, multithreaded parallelism was incorporated by the use of Intel Math Kernel Library (MKL). Gradually, the sub components of SIFENs paved the way for the development of the fully parallel Navier Stokes Solver - IFENs. Currently IFENs has been tested on up to 256 processors. IFENs executes at appreciable speeds. A 240 step cardiac cycle, executes on a computational domain with 14.583 million tetrahedron elements in 15.6 hours with just 72 processors. This is definitely comparable and perhaps better than explicit codes, which are known for their speed.

IFENs was utilized in this research for solving flow problems over biomedical domains, but can be utilized as a general purpose tool for a variety of flow problems. IFENs is fully implicit and capable of constructing the jacobian matrix while using the Newton like methods to solve the Navier-Stokes equations with the non-linear, convective terms in the left hand matrix. This renders the software to be highly efficient in solving non-linear equations. IFENs is capable of solving the Navier-Stokes equations both in split and monolithic frameworks. Different types of pressure stabilizations are also available to use. By far, more than 1000 recorded test cases for different problems have been executed using IFENs, on the facilities of HPC Wales, which provided a generous CPU time of 150000 hours.

IFENs, with its current capabilities has been successfully demonstrated to solve

and provide realistic solutions to the problem of flow within human arteries. However, in such mathematical models, the physics of the problem is highly dependent on the imposed boundary conditions. These boundary conditions are often unavailable or difficult to acquire from the human body. Even when such measurements are available, their correctness is often under question as the tolerance bands are really large. The pressure and velocity boundary conditions that were employed for the carotid arteries in this research yielded results consistent with the evidences in literature. Regions of possible plaque deposition have been identified in the geometries studied. However, validation with real patient data seems like a mandatory step to be absolutely confident about the predictions made. This must definitely be a future research direction. In the absence of medical data, computational studies like these will not be able to make predictions with high levels of confidence, which is essential in introducing this technology as a usable tool to doctors and clinicians.

Since compliance of arteries play a damping role under pulsatile flow, incorporating the solid dynamics models to be able to simulate the interaction between blood and the arterial walls would be more complete, for which IFENs already serves as a starting point. Although, IFENs is currently closed source, undistributed and solely developed by the author of this thesis.

Unlike conventional flow problems, arterial flow problems have an additional and active feedback loop, whereby the arterial walls adapt to the changing flow by sensing various parameters in the inner lining of the vessels called the endothelial cells. These parameters range from flow quantities (like the wall shear stress and oscillatory shear index, that were used in Chapter 5) to concentration of various biochemical species in the blood stream (like Adenosin Diphosphate (ADP), Adenosine Triphosphate (ATP), etc.) [78, 137]. This mono layer of endothelial cells also acts as a selective barrier and controls the passage of materials to and from the endothelium [36]. The endothelial cells respond to these stimuli via

eNOS (endothelial Nitric Oxide synthase) synthesis and calcium ion (Ca^{2+}) signalling [126, 124, 125, 149]. Under normal conditions atheroprotective reactions are elicited within the endothelial cells. When the endothelium ceases to perform its intended function and enters a non-adaptive state, a state called endothelial dysfunction is said to be attained. Endothelial function and bio-availability of Nitric Oxide (NO) affect myocardial function, systemic and pulmonary hemodynamics, and coronary and renal circulation [98]. As a matter of fact, the 1998 Nobel prize in Physiology/Medicine was awarded to Robert Furchgott, Louis Ignarro and Ferid Murad for their independent discoveries that "a short-lived gas, nitric oxide, NO, was endogenously produced and acted as a signaling molecule between cells" [3]. The incorporation of endothelial dysfunction and signalling models to the fluid simulation will therefore be valuable in better understanding the causal mechanisms, of which only the effects are getting better understood lately.

Following this direction of research, an implicit, parallel mass transport model was developed using SIFENs and the results for the concentration of biochemical species were benchmarked for idealised geometries. Also, the eNOS and calcium ion signalling models presented in [124] were implemented and tested for idealised geometries. The unison of the Navier Stokes solver - IFENs, the implicit mass transport solver (for ATP/ADP) and the eNOS and Ca^{2+} simulator has resulted in the addition of a one sided feedback mechanism to the problem of arterial flow. It remains to run this collection of software for patient specific meshes and to complete the feedback loop, which can be realised efficiently with the incorporation of a solid mechanics framework for the remodelling of the walls.

Appendix A

Sample MPI code explained

The MPI-based code enclosed in section 3.2.1.1 is considered here in this appendix. This code is analogous to a MWE [1], intended for illustrating the framework of message passing rather than a bug, that MWEs are generally associated with. This code aims to provide an easy to understand and intuitive base on which to build the actual parallel Finite Element application.

The code along with the 9 MPI subroutines used are briefly explained below:

1. **Header**

   The header, here makes reference to the first 3 lines in the code. As with non-MPI codes, the *program* keyword informs the compiler about the start of the program and *implicit none* results in no assumptions being made by the compiler about the data-type of variables based on the starting character of their names. This is standard for Fortran90 programs. Line 3 however is MPI related. It results in the inclusion of the header files necessary to interface with the actual MPI implementation itself.

[1] short for Minimal Working Example, in the field of computing

2. **Standard Datatypes**

   Line 4 - Line 12. This is standard in Fortran90 and needs no additional explanation. It must be noted that use of *status(MPI_STATUS_SIZE)*, which would result in a syntax error in standard Fortran90, is valid when the MPI header file has been included.

3. **Basic mandatory calls**

   Line 13 - Line 15. These subroutines need to be invoked in every MPI program.

   MPI_INIT(1 arg)

   This call initializes the MPI derived library. In Fortran90 there is just 1 argument to this call. This argument serves as an error-checking integer. This is a strictly mandatory call and the actual program must preferably start after invocation of this subroutine.

   MPI_COMM_SIZE(3 args)

   The abstract space formed by participating processors in an MPI environment is called a communicator. It is possible to have more than 1 communicator in an MPI code. The code developed in this research uses just 1 communicator. The default communicator is named *MPI_COMM_WORLD*. The *MPI_COMM_SIZE* subroutine finds out the total number of participating processors in a particular communicator. In this case, this number is stored in the variable named *numProcs*

   MPI_COMM_RANK(3 args)

   When having multiple processors to deal with, it becomes mandatory to be able to index them and send/receive specific instructions/data. This is made possible by processor ranks. Ranks may be treated equivalent to names which aid in processor identification. When *MPI_COMM_RANK* is invoked, the calling processor gets to know its rank in the communicator. In this case, this number is stored in a variable named *myrank*.

4. **Selective Printing**

   Line 16 - Line 18. To monitor the progress, it is often desirable to print suitable messages at selected points in the code. Since in an MPI environment all statements in the source code are visible to all processors, it is often tidy to print the progress monitoring statements in 1 processor only. Now that we know the ranks of all processors, we can print conditionally based on their ranks. In this case, the string *Starting program* gets printed by rank0 processor only.

5. **Check 1**

   Line 19 - Line 23. Since this is a very specific program, it needs to be run on 12 processors only. In these lines, every processor checks if the correct number of processors are being used. If not, *MPI_ABORT* gets invoked, which is equivalent to the standard *stop* statement in Fortran90.

   It might have been possible to perform this check in 1 processor only and the invocation of *MPI_ABORT* in just 1 processor would have resulted in all other processors to abort. However, in the meanwhile, other processors might have moved on to execute the remaining statements. In this code, it is not strictly necessary to worry about synchronization. However, in production codes, careful thought must be given to such situations. Either the underlying design, must be neutral to the need for synchronization or suitable means must be in place to prevent asynchronous behaviour.

6. **Processor Rank Strings**

   Line 24 - Line 25. On occasions, where messages are printed by all processors, it is often important to know where the messages came from. To avoid confusion, it is considered a good practise to print the name/rank of the processor before every message is printed. To do this a write statement is used. Instead of conventionally printing to file, it is possible to print also to

variables. Line 24 accomplishes this. Since processor ranks are 0 based by default, for convenience the ranks are offset by 1, to get a 1 based indexing. In this case, the variable *myrank_char* is used as an identifier at the start of every printed message.

7. **Processor Specific File Read**

   Line 27 - Line 36. Every processor (except the last processor, i.e. rank = numProcs) is assigned to read a specific file. The names of each of these files are similar and appended by the processor ranks to make the identification obvious. Using the rank data from MPI_COMM_RANK subroutine, every processor generates the names of the files to be opened by them and reads its contents.

8. **Conditional Task Allocation**

   Line 37 - Line 40. The result of this code is stored in a string called *result_basket*, which is initialized to a series of Xs prior to the commencement of MPI communications, in rank 0 processor only.

9. **MPI_BCAST(6 args)**

   Line 41 - Line 42. The MPI_BCAST subroutine is used to broadcast messages from a specified processor to all other processors in the communicator. It may be used to communicate both standard and user defined data types. The arguments include the variable to be broadcast, length, data type, rank of broadcast root, communicator and error status, in the order of appearance.

10. **Communicate Messages**

    Line 44 - Line 63. The messages are prepared, sent and received in these lines.

    MPI_RECV(8 args)

    As the name suggests, the MPI_RECV subroutine is used to receive mes-

sages from other processors. This must typically be accompanied by a corresponding MPI_SEND subroutine. There are 8 arguments for the MPI_RECV subroutine. These are variable in which to receive, length of data, data type, rank of the processor from which to receive, tag, communicator, status, error status, in the order of appearance. This subroutine is invoked in just the last processor, in which the result string is generated. The MPI_RECV subroutine is of the blocking type, i.e. it returns only after the message has been completely received.

MPI_SEND(7 args)

As the name suggests, this MPI subroutine is used to send messages to other processors. This is typically associated with a corresponding MPI_RECV subroutine. There are 7 arguments for this subroutine. These are variable to send, length of data, data type, rank of the destination processor, tag, communicator, error status, in the order of appearance. Due to the nature of communications here, the MPI_SEND subroutine is invoked by all processors except the last one.

11. **Synchronization**

Line 65. The MPI_BARRIER call may be used to synchronize all processors. Since every processor typically finishes its tasks in dissimilar times, depending on the nature of the global problem, the results may be intolerant to the absence of synchronization. Although the use of the MPI_BARRIER call here is not mandatory, it is used to just illustrate its usage. This MPI subroutine prevents code execution beyond the occurrence of MPI_BARRIER call, in all processors. Only after all processors enter this call, will the code execute beyond this point.

12. **MPI_FINALIZE**

Line 69. Like MPI_INIT initializes the MPI library, the MPI_FINALIZE library terminates the MPI library. It is a mandatory and correct way of

ending an MPI program. This must be invoked at the very end, but before the Fortran *end program line.*

Appendix B

Sample PETSc code explained

The PETSc code enclosed in Section 3.2.2.2 is considered here and a brief description of the code will be provided with the aim of highlighting the PETSc components in the code. The 4×4 linear matrix system of Figure **??** is solved with MPI parallelism, using the KSP objects of PETSc. A block wise description of the code will be presented below:

1. **Header**

   Lines 4 through 9. These are the standard header files that must be used in every application intending to use PETSc components. There is a header file specific to every object (e.g. Mat, Vec, KSP, SNES, etc.) that must be loaded for these data types to be recognised by the compiler. Also, these header lines begin with an include statement, preceded by a #, i.e. these are preprocessor specific statements that are processed before the compilation begins. It is therefore mandatory that the file name with which these statements occur, have the F90 extension.

2. **PETSc data type**

   Lines 11 through 15. Here all the variables that use PETSc data types are defined.

3. **Fortran data types**

   Lines 17 through 19. The standard Fortran data types are associated with the required variables here.

4. **Initialization routines**

   Lines 21 through 23. Like in an MPI program, the scope for MPI parallelism starts with an **MPI_INIT** call, in PETSc the equivalent is **PetscInitialize**. The first argument for **PetscInitialize** could either be a dummy argument, called **PETSC_NULL_CHARACTER**, or it could also be the name of the options database file. This options file may be used to set different solvers, preconditioners, matrix formats, etc. The **MPI_COMM_RANK** and **MPI_COMM_SIZE** subroutines are already described in Appendix A. The only difference exists in the name of the communicator, which instead of being **MPI_COMM_WORLD** is called **PETSC_COMM_WORLD**.

5. **Matrix partitioning**

   Lines 25 through 26. Since the problem is so small, the matrix partitioning information is manually set and selected in each processor based on the processor rank.

6. **Parallel Matrix Generation**

   Line 28 through 30. Objects are created and destroyed, as and when necessary. The names of the subroutine are quite descriptive and therefore intuitive. The MatCreate subroutine generates a parallel matrix in the communicator specified. Also, a name is assigned to this parallel matrix (LHS). The sizes are set using MatSetSizes subroutine. One needs to provide the global size (total number of rows and columns) and the local size (number of rows and columns owned by the processor that is calling this subroutine). The call to MatSetUp, sets up the internal matrix data structures that will be used as the entries will be assembled into the matrix. The

matrix being constructed here will serve as the main system matrix.

7. **Parallel Vector Generation**

   Line 32 through 35. Like for matrices, a similar procedure is adopted to define, size and setup a parallel vector. The solution vector is generated here. In addition, the right hand side vector is simply constructed by using the VecDuplicate subroutine.

8. **Population of the parallel matrix and vector**

   Line 37 through 54. The global indices (of the parallel matrix) in which to insert the elemental matrices, are generated first (0 based indexing). In this simple case, the values are added row wise, one after the other. The same procedure is adopted for vectors too. This simple method is chosen for illustrational purposes only. Efficient value insertions must occur for real problems. When this step completes, the finite element assembly would have completed automatically.

9. **Parallel object assembly**

   Line 56 through 59. Like the finite element assembly, one needs to perform also the parallel object assembly in PETSc, to ensure that every processor has updated entries, ready to be used for the computations of the next step.

10. **Solve the linear system**

    Line 61 though 63. Before the linear system can be solved, a KSP object must be created. This object must then be made aware of the matrix, matrix preconditioner, right hand vector and the solution vector, before it can solve the parallel system iteratively.

11. **Display** Line 65. To print solutions contained in parallel objects to files or screens, object based *View commands are available.

12. **Termination** Line 67 though 68. Like an MPI program terminates with

a finalize statement, PETSc programs terminate with call to PetscFinalize. The scope of the calling Fortran program is then brought to an end using the usual end program statement.

Appendix C

Detailed formulation of Navier-Stokes Equations

C.1 Spatial discrertization of step 1

$$\int_\Omega \hat{N}^T \frac{\Delta u_j^\star}{\Delta t} d\Omega + \int_\Omega \hat{N}^T \frac{\partial}{\partial x_i}(u_i u_j)^{n+1} d\Omega - \frac{1}{\rho}\int_\Omega \hat{N}^T \frac{\partial \tau_{ji}^{n+1}}{\partial x_i} d\Omega$$
$$- \int_\Omega \hat{N}^T \frac{\Delta t}{2} \frac{u_k^{n+1}}{\rho} \frac{\partial}{\partial x_k}\left(\frac{\partial}{\partial x_i}(u_i u_j)^{n+1}\right) d\Omega \approx 0$$
i.e. $Term1 + Term10 + Term3 + Term11 \approx 0$

Term 1

$$\begin{aligned} Term1 &= \int_\Omega \hat{N}^T \frac{\Delta u_j^\star}{\Delta t} d\Omega \\ &= \frac{1}{\Delta t}\int_\Omega \hat{N}^T \hat{N} \Delta \hat{u}_j^\star d\Omega \end{aligned}$$

$$= \frac{V}{20(\Delta t)} \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix} \begin{Bmatrix} \Delta u^{\star}_{j_1} \\ \Delta u^{\star}_{j_2} \\ \Delta u^{\star}_{j_3} \\ \Delta u^{\star}_{j_4} \end{Bmatrix} \tag{C.1}$$

$$Term1 = \frac{V}{20(\Delta t)}[M]\{\Delta \hat{u}^{\star}_j\} \tag{C.2}$$

Term 10

$$\begin{aligned} Term10 &= \int_{\Omega} \hat{N}^T \frac{\partial}{\partial x_i}(u_i u_j)^{n+1} d\Omega \\ &= \int_{\Omega} \hat{N}^T \left(u_j \frac{\partial u_i}{\partial x_i} + u_i \frac{\partial u_j}{\partial x_i} \right)^{n+1} d\Omega \\ &= \int_{\Omega} \hat{N}^T \left(u_i^{n+1} \frac{\partial u_j^{n+1}}{\partial x_i} \right) d\Omega \end{aligned}$$

(The divergence of velocity is 0 for incompressible flows)

The notion of the Newton iteration will be introduced here for the non-linear term, which will be solved using a Newton_Krylov type solver in PETSc. The superscripts nn and nn+1 represent the Newton iterations. The terms at time level $n+1$ are actually equivalent to terms at the $(nn+1)^{th}$ Newton iteration, when the Newton method has converged. The $Term10$ can therefore be written as,

$$Term10 = \int_{\Omega} \hat{N}^T \left(u_i^{nn+1} \frac{\partial u_j^{nn+1}}{\partial x_i} \right) d\Omega$$

Noting that,

$u^{n+1} \approx u^{nn+1}$, at Newton convergence, and

$u^{nn+1} \approx u^{nn} + \delta u,$

$$
\begin{aligned}
Term10 &= \int_\Omega \hat{N}^T \left((u_i^{nn} + \delta u_i) \frac{\partial}{\partial x_i} (u_j^{nn} + \delta u_j) \right) d\Omega \\
&= \int_\Omega \hat{N}^T \left(u_i^{nn} \frac{\partial u_j^{nn}}{\partial x_i} + u_i^{nn} \frac{\partial}{\partial x_i}(\delta u_j) + \delta u_i \frac{\partial u_j^{nn}}{\partial x_i} + \cancelto{0}{\delta u_i \frac{\partial}{\partial x_i}(\delta u_j)} \right) d\Omega \\
&= Term10a + Term10b + Term10c
\end{aligned}
$$

Term 10a

$$
\begin{aligned}
Term10a &= \int_\Omega \hat{N}^T u_i^{nn} \frac{\partial u_j^{nn}}{\partial x_i} d\Omega \\
&= \int_\Omega \hat{N}^T (\hat{N} \hat{u}_i^{nn}) \left(\frac{\partial \hat{N}}{\partial x_i} \hat{u}_j^{nn} \right) d\Omega \\
&= \int_\Omega \begin{Bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \end{Bmatrix} \begin{Bmatrix} N_1 & N_2 & N_3 & N_4 \end{Bmatrix} \begin{Bmatrix} u_{i_1} \\ u_{i_2} \\ u_{i_3} \\ u_{i_4} \end{Bmatrix} \frac{\partial}{\partial x_i} \begin{Bmatrix} N_1 & N_2 & N_3 & N_4 \end{Bmatrix} \begin{Bmatrix} u_{j_1} \\ u_{j_2} \\ u_{j_3} \\ u_{j_4} \end{Bmatrix} d\Omega \\
&= \frac{V}{20} \begin{bmatrix} 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 1 & 1 & 2 \end{bmatrix} \begin{Bmatrix} u_{i_1} \\ u_{i_2} \\ u_{i_3} \\ u_{i_4} \end{Bmatrix}^{nn} \frac{\partial}{\partial x_i} \begin{Bmatrix} N_1 & N_2 & N_3 & N_4 \end{Bmatrix} \begin{Bmatrix} u_{j_1} \\ u_{j_2} \\ u_{j_3} \\ u_{j_4} \end{Bmatrix}^{nn} \\
&= \frac{V}{20} [M] \{\hat{u}_i\}^{nn} \{D_i\} \{\hat{u}_j\}^{nn} \qquad \text{(C.3)}
\end{aligned}
$$

Term 10b

$$
\begin{aligned}
Term10b &= \int_\Omega \hat{N}^T u_i^{nn} \frac{\partial}{\partial x_i} (\delta u_j) d\Omega \\
&= \int_\Omega \hat{N}^T (\hat{N} \hat{u}_i)^{nn} \left(\frac{\partial \hat{N}}{\partial x_i} \delta \hat{u}_j \right) d\Omega
\end{aligned}
$$

$$= \frac{V}{20}[M]\{\hat{u}_i\}^{nn}\{D_i\}\{\delta\hat{u}_j\} \tag{C.4}$$

Term 10c

$$\begin{aligned} Term10c &= \int_\Omega \hat{N}^T \delta u_i \frac{\partial u_j^{nn}}{\partial x_i} d\Omega \\ &= \int_\Omega \hat{N}^T (\hat{N}\delta\hat{u}_i)\left(\frac{\partial \hat{N}}{\partial x_i}\hat{u}_j^{nn}\right) d\Omega \\ &= \frac{V}{20}[M]\{\delta\hat{u}_i\}\{D_i\}\{\hat{u}_j^{nn}\} \\ &= \frac{V}{20}\{D_i\}\{\hat{u}_j^{nn}\}[M]\{\delta\hat{u}_i\} \end{aligned} \tag{C.5}$$

Term 3

$$\begin{aligned} Term3 &= -\frac{1}{\rho}\int_\Omega \hat{N}^T \frac{\partial \tau_{ji}^{n+1}}{\partial x_i} d\Omega \\ &= -\frac{\mu}{\rho}\int_\Omega \hat{N}^T \frac{\partial}{\partial x_i}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3}\delta_{ij}\frac{\partial u_k}{\partial x_k}\right)^{n+1} d\Omega \\ &= -\frac{\mu}{\rho}\int_\Omega \hat{N}^T \frac{\partial}{\partial x_i}\left(\frac{\partial u_i}{\partial x_j}\right)^{n+1} d\Omega - \frac{\mu}{\rho}\int_\Omega \hat{N}^T \frac{\partial}{\partial x_i}\left(\frac{\partial u_j}{\partial x_i}\right)^{n+1} d\Omega \end{aligned} \tag{C.6}$$

(the third term on the RHS reduces to zero from the mass conservation law for incompressible flows)

Applying the mass conservation law after interchanging the derivatives in the first term results in,

$$Term3 = -\frac{\mu}{\rho}\int_\Omega \hat{N}^T \frac{\partial}{\partial x_j}\left(\frac{\partial u_i}{\partial x_i}\right) d\Omega - \frac{\mu}{\rho}\int_\Omega \hat{N}^T \frac{\partial}{\partial x_i}\left(\frac{\partial u_j}{\partial x_i}\right) d\Omega$$

$$= -\frac{\mu}{\rho}\int_\Omega \hat{N}^T \frac{\partial}{\partial x_i}\left(\frac{\partial u_j}{\partial x_i}\right) d\Omega \tag{C.7}$$

Again noting that, $u^{n+1} \approx u^{nn+1}$ & $u^{nn+1} \approx u^{nn} + \delta u$,

$$\begin{aligned} Term3 &= -\frac{\mu}{\rho}\int_\Omega \hat{N}^T \frac{\partial}{\partial x_i}\left(\frac{\partial u_j}{\partial x_i}\right)^{nn+1} d\Omega \\ &= -\frac{\mu}{\rho}\int_\Omega \hat{N}^T \frac{\partial}{\partial x_i}\left(\frac{\partial}{\partial x_i}(u_j^{nn} + \delta u_j)\right) d\Omega \\ &= -\frac{\mu}{\rho}\int_\Omega \hat{N}^T \frac{\partial}{\partial x_i}\left(\frac{\partial u_j^{nn}}{\partial x_i}\right) d\Omega - \frac{\mu}{\rho}\int_\Omega \hat{N}^T \frac{\partial}{\partial x_i}\left(\frac{\partial(\delta u_j)}{\partial x_i}\right) d\Omega \end{aligned} \tag{C.8}$$

Integrating by parts and neglecting the boundary integrals,

$$\begin{aligned} Term3 &= \frac{\mu}{\rho}\left[\int_\Omega \frac{\partial \hat{N}^T}{\partial x_i}\frac{\partial u_j^{nn}}{\partial x_i} d\Omega + \int_\Omega \frac{\partial \hat{N}^T}{\partial x_i}\frac{\partial(\delta u_j)}{\partial x_i} d\Omega\right] \\ &= Term3_1 + Term3_2 \end{aligned} \tag{C.9}$$

***Term* 3_1**

$$\begin{aligned} Term3_1 = &= \frac{\mu}{\rho}\int_\Omega \frac{\partial \hat{N}^T}{\partial x_i}\frac{\partial u_j^{nn}}{\partial x_i} d\Omega \\ &= \frac{\mu}{\rho}\int_\Omega \frac{\partial \hat{N}^T}{\partial x_i}\frac{\partial \hat{N}}{\partial x_i} d\Omega \hat{u}_j^{nn} \\ &= \frac{\mu}{\rho} V \frac{\partial \hat{N}^T}{\partial x_i}\frac{\partial \hat{N}}{\partial x_i}\hat{u}_j^{nn} \end{aligned}$$

$$= \frac{\mu}{\rho} V \begin{bmatrix} \left(\frac{\partial N_1}{\partial x_i}\right)^2 & \frac{\partial N_1}{\partial x_i}\frac{\partial N_2}{\partial x_i} & \frac{\partial N_1}{\partial x_i}\frac{\partial N_3}{\partial x_i} & \frac{\partial N_1}{\partial x_i}\frac{\partial N_4}{\partial x_i} \\ \frac{\partial N_2}{\partial x_i}\frac{\partial N_1}{\partial x_i} & \left(\frac{\partial N_2}{\partial x_i}\right)^2 & \frac{\partial N_2}{\partial x_i}\frac{\partial N_3}{\partial x_i} & \frac{\partial N_2}{\partial x_i}\frac{\partial N_4}{\partial x_i} \\ \frac{\partial N_3}{\partial x_i}\frac{\partial N_1}{\partial x_i} & \frac{\partial N_3}{\partial x_i}\frac{\partial N_2}{\partial x_i} & \left(\frac{\partial N_3}{\partial x_i}\right)^2 & \frac{\partial N_3}{\partial x_i}\frac{\partial N_4}{\partial x_i} \\ \frac{\partial N_4}{\partial x_i}\frac{\partial N_1}{\partial x_i} & \frac{\partial N_4}{\partial x_i}\frac{\partial N_2}{\partial x_i} & \frac{\partial N_4}{\partial x_i}\frac{\partial N_3}{\partial x_i} & \left(\frac{\partial N_4}{\partial x_i}\right)^2 \end{bmatrix} \begin{Bmatrix} u_{j_1} \\ u_{j_2} \\ u_{j_3} \\ u_{j_4} \end{Bmatrix}^{nn}$$

$$Term3_1 = \frac{\mu V}{\rho}[H_{ii}]\{\hat{u}_j\}^{nn} \tag{C.10}$$

Term3$_2$:

$$\begin{aligned} Term3_2 &= \frac{\mu}{\rho}\int_\Omega \frac{\partial \hat{N}^T}{\partial x_i}\frac{\partial \delta u_j}{\partial x_i}d\Omega \\ &= \frac{\mu V}{\rho}[H_{ii}]\{\delta\hat{u}_j\} \end{aligned} \tag{C.11}$$

Term 11

$$Term11 = -\frac{\Delta t}{2\rho}\int_\Omega \hat{N}^T u_k^{n+1}\frac{\partial}{\partial x_k}\left(\frac{\partial}{\partial x_i}(u_i u_j)\right)^{n+1} d\Omega$$

The incompressiblity condition leads to,

$$Term11 = -\frac{\Delta t}{2\rho}\int_\Omega \hat{N}^T u_k^{n+1}\frac{\partial}{\partial x_k}\left(u_i\frac{\partial u_j}{\partial x_i}\right)^{n+1} d\Omega$$

Integrating by parts,

$$Term11 = \frac{\Delta t}{2\rho}\int_\Omega \frac{\partial}{\partial x_k}(\hat{N}^T u_k^{n+1})\left(u_i\frac{\partial u_j}{\partial x_i}\right)^{n+1} d\Omega - \frac{\Delta t}{2\rho}\int_\Gamma \hat{N}^T u_k^{n+1}\left(u_i\frac{\partial u_j}{\partial x_i}\right)^{n+1} d\Gamma$$

Neglecting the boundary integral and using the incompressiblity condition again,

$$Term11 = \frac{\Delta t}{2\rho}\int_\Omega u_k^{n+1}\frac{\partial \hat{N}^T}{\partial x_k}\left(u_i\frac{\partial u_j}{\partial x_i}\right)^{n+1} d\Omega$$

Noting that $u^{n+1} \approx u^{nn+1}$ and $u^{nn+1} \approx u^{nn} + \delta u$,

$$\begin{aligned}
Term11 &= \frac{\Delta t}{2\rho}\int_\Omega (u_k^{nn}+\delta u_k)\frac{\partial \hat{N}^T}{\partial x_k}(u_i^{nn}+\delta u_i)\frac{\partial}{\partial x_i}\left(u_j^{nn}+\delta u_j\right)d\Omega \\
&= \frac{\Delta t}{2\rho}\int_\Omega (u_k^{nn}+\delta u_k)\frac{\partial \hat{N}^T}{\partial x_k}\left(u_i^{nn}\frac{\partial u_j^{nn}}{\partial x_i}+u_i^{nn}\frac{\partial \delta u_j}{\partial x_i}+\delta u_i\frac{\partial u_j^{nn}}{\partial x_i}+\cancelto{0}{\delta u_i\frac{\partial \delta u_j}{\partial x_i}}\right)d\Omega \\
&= \frac{\Delta t}{2\rho}\int_\Omega \left(u_k^{nn}\frac{\partial \hat{N}^T}{\partial x_k}u_i^{nn}\frac{\partial u_j^{nn}}{\partial x_i}+u_k^{nn}\frac{\partial \hat{N}^T}{\partial x_k}u_i^{nn}\frac{\partial \delta u_j}{\partial x_i}+u_k^{nn}\frac{\partial \hat{N}^T}{\partial x_k}\delta u_i\frac{\partial u_j^{nn}}{\partial x_i}+\right. \\
&\qquad \left.\delta u_k\frac{\partial \hat{N}^T}{\partial x_k}u_i^{nn}\frac{\partial u_j^{nn}}{\partial x_i}+\cancelto{0}{\delta u_k\frac{\partial \hat{N}^T}{\partial x_k}u_i^{nn}\frac{\partial \delta u_j}{\partial x_i}}+\cancelto{0}{\delta u_k\frac{\partial \hat{N}^T}{\partial x_k}\delta u_i\frac{\partial u_j^{nn}}{\partial x_i}}\right)d\Omega \\
&= Term11a + Term11b + Term11c + Term11d
\end{aligned} \tag{C.12}$$

Term 11a

$$\begin{aligned}
Term11a &= \frac{\Delta t}{2\rho}\int_\Omega u_k^{nn}\frac{\partial \hat{N}^T}{\partial x_k}u_i^{nn}\frac{\partial u_j^{nn}}{\partial x_i}d\Omega \\
&= \frac{\Delta t}{2\rho}\int_\Omega u_k^{nn}u_i^{nn}\frac{\partial \hat{N}^T}{\partial x_k}\frac{\partial u_j^{nn}}{\partial x_i}d\Omega
\end{aligned}$$

Approximating as usual,

$$\begin{aligned}
Term11a &= \frac{\Delta t}{2\rho}\int_\Omega (\hat{N}\hat{u}_k^{nn})(\hat{N}\hat{u}_i^{nn})\frac{\partial \hat{N}^T}{\partial x_k}\left(\frac{\partial \hat{N}}{\partial x_i}\hat{u}_j^{nn}\right)d\Omega \\
&= \frac{\Delta t}{2\rho}\int_\Omega \left\{N_1 \quad N_2 \quad N_3 \quad N_4\right\}\begin{Bmatrix} u_{k_1} \\ u_{k_2} \\ u_{k_3} \\ u_{k_4}\end{Bmatrix}^{nn}\left\{N_1 \quad N_2 \quad N_3 \quad N_4\right\}\begin{Bmatrix} u_{i_1} \\ u_{i_2} \\ u_{i_3} \\ u_{i_4}\end{Bmatrix}^{nn}\frac{\partial \hat{N}^T}{\partial x_k}\left(\frac{\partial \hat{N}}{\partial x_i}\hat{u}_j^{nn}\right)d\Omega
\end{aligned}$$

$$
\begin{aligned}
&= \frac{\Delta t}{2\rho}\int_\Omega (N_1 u_{k_1} + N_2 u_{k_2} + N_3 u_{k_3} + N_4 u_{k_4})^{nn} \begin{Bmatrix} N_1 & N_2 & N_3 & N_4 \end{Bmatrix} \begin{Bmatrix} u_{i_1} \\ u_{i_2} \\ u_{i_3} \\ u_{i_4} \end{Bmatrix}^{nn} \\
&\quad \frac{\partial \hat{N}^T}{\partial x_k}\left(\frac{\partial \hat{N}}{\partial x_i}\hat{u}_j^{nn}\right) d\Omega \\
&= \frac{\Delta t}{2\rho}\int_\Omega \begin{Bmatrix} N_1^2 u_{k_1} + N_1 N_2 u_{k_2} + N_1 N_3 u_{k_3} + N_1 N_4 u_{k_4} \\ N_1 N_2 u_{k_1} + N_2^2 u_{k_2} + N_3 N_2 u_{k_3} + N_4 N_2 u_{k_4} \\ N_1 N_3 u_{k_1} + N_2 N_3 u_{k_2} + N_3^2 u_{k_3} + N_4 N_3 u_{k_4} \\ N_1 N_4 u_{k_1} + N_2 N_4 u_{k_2} + N_3 N_4 u_{k_3} + N_4^2 u_{k_4} \end{Bmatrix}^{nn^T} \begin{Bmatrix} u_{i_1} \\ u_{i_2} \\ u_{i_3} \\ u_{i_4} \end{Bmatrix}^{nn} \frac{\partial \hat{N}^T}{\partial x_k}\left(\frac{\partial \hat{N}}{\partial x_i}\hat{u}_j^{nn}\right) d\Omega \\
&= \frac{\Delta t}{2\rho}\frac{V}{20} \begin{Bmatrix} 2u_{k_1} + u_{k_2} + u_{k_3} + u_{k_4} \\ u_{k_1} + 2u_{k_2} + u_{k_3} + u_{k_4} \\ u_{k_1} + u_{k_2} + 2u_{k_3} + u_{k_4} \\ u_{k_1} + u_{k_2} + u_{k_3} + 2u_{k_4} \end{Bmatrix}^{nn^T} \begin{Bmatrix} u_{i_1} \\ u_{i_2} \\ u_{i_3} \\ u_{i_4} \end{Bmatrix}^{nn} \frac{\partial \hat{N}^T}{\partial x_k}\left(\frac{\partial \hat{N}}{\partial x_i}\hat{u}_j^{nn}\right) \\
&= \frac{\Delta t}{2\rho}\frac{V}{20}\{A\hat{U}X\}_k^{nn}\{\hat{u}_i\}^{nn}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\hat{u}_j\}^{nn}
\end{aligned}
\tag{C.13}
$$

where $AUX = \begin{Bmatrix} 2u_{k_1} + u_{k_2} + u_{k_3} + u_{k_4} \\ u_{k_1} + 2u_{k_2} + u_{k_3} + u_{k_4} \\ u_{k_1} + u_{k_2} + 2u_{k_3} + u_{k_4} \\ u_{k_1} + u_{k_2} + u_{k_3} + 2u_{k_4} \end{Bmatrix}^{nn^T}$ is an auxiliary vector, which is introduced in the interest of space.

Term 11b

$$
Term11b = \frac{\Delta t}{2\rho}\int_\Omega u_k^{nn}\frac{\partial \hat{N}^T}{\partial x_k}u_i^{nn}\frac{\partial \delta u_j}{\partial x_i}d\Omega
$$

$$
\begin{aligned}
&= \frac{\Delta t}{2\rho}\frac{V}{20}\begin{Bmatrix} 2u_{k_1}+u_{k_2}+u_{k_3}+u_{k_4} \\ u_{k_1}+2u_{k_2}+u_{k_3}+u_{k_4} \\ u_{k_1}+u_{k_2}+2u_{k_3}+u_{k_4} \\ u_{k_1}+u_{k_2}+u_{k_3}+2u_{k_4} \end{Bmatrix}^{nn^T} \hat{u}_i^{nn}\frac{\partial \hat{N}^T}{\partial x_k}\frac{\partial \hat{N}}{\partial x_i}\delta\hat{u}_j \\
&= \frac{\Delta t}{2\rho}\frac{V}{20}\{A\hat{U}X\}_k^{nn}\{\hat{u}_i\}^{nn}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\delta\hat{u}_j\}
\end{aligned}
\tag{C.14}
$$

Term 11c

$$
\begin{aligned}
Term11c &= \frac{\Delta t}{2\rho}\int_\Omega u_k^{nn}\frac{\partial \hat{N}^T}{\partial x_k}\delta u_i\frac{\partial u_j^{nn}}{\partial x_i}d\Omega \\
&= \frac{\Delta t}{2\rho}\int_\Omega u_k^{nn}\delta u_i\frac{\partial \hat{N}^T}{\partial x_k}\frac{\partial u_j^{nn}}{\partial x_i}d\Omega \\
&= \frac{\Delta t}{2\rho}\frac{V}{20}\begin{Bmatrix} 2u_{k_1}+u_{k_2}+u_{k_3}+u_{k_4} \\ u_{k_1}+2u_{k_2}+u_{k_3}+u_{k_4} \\ u_{k_1}+u_{k_2}+2u_{k_3}+u_{k_4} \\ u_{k_1}+u_{k_2}+u_{k_3}+2u_{k_4} \end{Bmatrix}^{nn^T} \delta\hat{u}_i\frac{\partial \hat{N}^T}{\partial x_k}\frac{\partial \hat{N}}{\partial x_i}\hat{u}_j^{nn} \\
&= \frac{\Delta t}{2\rho}\frac{V}{20}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\hat{u}_j\}^{nn}\{A\hat{U}X\}_k^{nn}\{\delta\hat{u}_i\}
\end{aligned}
\tag{C.15}
$$

Term 11d

$$
\begin{aligned}
Term11d &= \frac{\Delta t}{2\rho}\int_\Omega \delta u_k\frac{\partial \hat{N}^T}{\partial x_k}u_i^{nn}\frac{\partial u_j^{nn}}{\partial x_i}d\Omega \\
&= \frac{\Delta t}{2\rho}\int_\Omega u_i^{nn}\delta u_k\frac{\partial \hat{N}^T}{\partial x_k}\frac{\partial u_j^{nn}}{\partial x_i}d\Omega
\end{aligned}
$$

$$= \frac{\Delta t}{2\rho}\frac{V}{20}\frac{\partial \hat{N}^T}{\partial x_k}\frac{\partial \hat{N}}{\partial x_i}\hat{u}_j^{nn}\begin{Bmatrix} 2u_{i_1}+u_{i_2}+u_{i_3}+u_{i_4} \\ u_{i_1}+2u_{i_2}+u_{i_3}+u_{i_4} \\ u_{i_1}+u_{i_2}+2u_{i_3}+u_{i_4} \\ u_{i_1}+u_{i_2}+u_{i_3}+2u_{i_4} \end{Bmatrix}^{nn^T} \delta\hat{u}_k$$

$$= \frac{\Delta t}{2\rho}\frac{V}{20}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\hat{u}_j\}^{nn}\{A\hat{U}X\}_i^{nn}\{\delta\hat{u}_k\} \tag{C.16}$$

Step 1 in fully discrete form

$$\begin{aligned} Term1 + Term10 + Term3 + Term11 &= 0 \\ \Longrightarrow\ Term1 + Term10a + Term10b + Term10c + Term3_1 & \\ +Term3_2 + Term11a + Term11b + Term11c + Term11d &= 0 \end{aligned}$$

i.e.

$$\begin{aligned} &\frac{V}{20(\Delta t)}[M]\{\Delta\hat{u}_j^\star\} + \frac{V}{20}[M]\{\hat{u}_i\}^{nn}\{D_i\}\{\hat{u}_j\}^{nn} + \frac{V}{20}[M]\{\hat{u}_i\}^{nn}\{D_i\}\{\delta\hat{u}_j\} + \\ &\frac{V}{20}\{D_i\}\{\hat{u}_j^{nn}\}[M]\{\delta\hat{u}_i\} + \frac{\mu V}{\rho}[H_{ii}]\{\hat{u}_j\}^{nn} + \frac{\mu V}{\rho}[H_{ii}]\{\delta\hat{u}_j\} + \\ &\frac{\Delta t}{2\rho}\frac{V}{20}\{A\hat{U}X\}_k^{nn}\{\hat{u}_i\}^{nn}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\hat{u}_j\}^{nn} + \frac{\Delta t}{2\rho}\frac{V}{20}\{A\hat{U}X\}_k^{nn}\{\hat{u}_i\}^{nn}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\delta\hat{u}_j\} + \\ &\frac{\Delta t}{2\rho}\frac{V}{20}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\hat{u}_j\}^{nn}\{A\hat{U}X\}_k^{nn}\{\delta\hat{u}_i\} + \frac{\Delta t}{2\rho}\frac{V}{20}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\hat{u}_j\}^{nn}\{A\hat{U}X\}_i^{nn}\{\delta\hat{u}_k\} \approx 0 \end{aligned} \tag{C.17}$$

$$\frac{V}{20(\Delta t)}[M]\{\Delta\hat{u}_j^\star\} + \frac{V}{20}[M]\{\hat{u}_i\}^{nn}\{D_i\}\{\delta\hat{u}_j\} + \frac{V}{20}\{D_i\}\{\hat{u}_j^{nn}\}[M]\{\delta\hat{u}_i\} +$$

$$\frac{\mu V}{\rho}[H_{ii}]\{\delta\hat{u}_j\} + \frac{\Delta t}{2\rho}\frac{V}{20}\{A\hat{U}X\}_k^{nn}\{\hat{u}_i\}^{nn}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\delta\hat{u}_j\} +$$
$$\frac{\Delta t}{2\rho}\frac{V}{20}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\hat{u}_j\}^{nn}\{A\hat{U}X\}_k^{nn}\{\delta\hat{u}_i\} + \frac{\Delta t}{2\rho}\frac{V}{20}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\hat{u}_j\}^{nn}\{A\hat{U}X\}_i^{nn}\{\delta\hat{u}_k\} =$$
$$-\frac{V}{20}[M]\{\hat{u}_i\}^{nn}\{D_i\}\{\hat{u}_j\}^{nn} - \frac{\mu V}{\rho}[H_{ii}]\{\hat{u}_j\}^{nn} - \frac{\Delta t}{2\rho}\frac{V}{20}\{A\hat{U}X\}_k^{nn}\{\hat{u}_i\}^{nn}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\hat{u}_j\}^{nn} \tag{C.18}$$

C.2 Spatial discrertization of step 2

$$\begin{aligned}\frac{1}{\Delta t c^2}\int_\Omega \hat{N}^T\Delta p d\Omega &= -\rho\int_\Omega \hat{N}^T\frac{\partial u_i^\star}{\partial x_i}d\Omega + \Delta t\int_\Omega \hat{N}^T\frac{\partial^2 p}{\partial x_i^2}d\Omega \\ Termm &= Term5 + Term6\end{aligned} \tag{C.19}$$

Term m

$$\begin{aligned}Termm &= \frac{1}{\Delta t c^2}\int_\Omega \hat{N}^T\Delta p d\Omega \\ &= \frac{1}{\Delta t c^2}\int_\Omega \hat{N}^T\hat{N}d\Omega\Delta\hat{p} \\ &= \frac{V}{20\Delta t c^2}\begin{bmatrix}2&1&1&1\\1&2&1&1\\1&1&2&1\\1&1&1&2\end{bmatrix}\begin{Bmatrix}\Delta p_1\\\Delta p_2\\\Delta p_3\\\Delta p_4\end{Bmatrix} \\ Termm &= \frac{V}{20\Delta t c^2}[M]\{\Delta\hat{p}\}\end{aligned} \tag{C.20}$$

Term 5

$$
\begin{aligned}
Term5 &= -\rho \int_\Omega \hat{N}^T \frac{\partial u_i^*}{\partial x_i} d\Omega \\
&= -\rho \int_\Omega \hat{N}^T \frac{\partial \hat{N}}{\partial x_i} d\Omega \hat{u_i^*} \\
&= -\frac{V\rho}{4} \begin{bmatrix} \frac{\partial N_1}{\partial x_i} & \frac{\partial N_2}{\partial x_i} & \frac{\partial N_3}{\partial x_i} & \frac{\partial N_4}{\partial x_i} \\ \frac{\partial N_1}{\partial x_i} & \frac{\partial N_2}{\partial x_i} & \frac{\partial N_3}{\partial x_i} & \frac{\partial N_4}{\partial x_i} \\ \frac{\partial N_1}{\partial x_i} & \frac{\partial N_2}{\partial x_i} & \frac{\partial N_3}{\partial x_i} & \frac{\partial N_4}{\partial x_i} \\ \frac{\partial N_1}{\partial x_i} & \frac{\partial N_2}{\partial x_i} & \frac{\partial N_3}{\partial x_i} & \frac{\partial N_4}{\partial x_i} \end{bmatrix} \begin{Bmatrix} u_{i_1}^* \\ u_{i_2}^* \\ u_{i_3}^* \\ u_{i_4}^* \end{Bmatrix} \\
Term5 &= -\frac{V\rho}{4} [ND_i]\{\hat{u}_i^*\}
\end{aligned}
\tag{C.21}
$$

Term 6

$$
\begin{aligned}
Term6 &= \Delta t \int_\Omega \hat{N}^T \frac{\partial^2 p^{nn+1}}{\partial x_i^2} d\Omega \\
&= \Delta t \int_\Omega \hat{N}^T \frac{\partial^2 p^{nn}}{\partial x_i^2} d\Omega + \Delta t \int_\Omega \hat{N}^T \frac{\partial^2 (\delta p)}{\partial x_i^2} d\Omega
\end{aligned}
\tag{C.22}
$$

Integrating by parts and neglecting the boundary integrals,

$$
\begin{aligned}
Term6 &= -\Delta t \int_\Omega \frac{\partial \hat{N}^T}{\partial x_i} \frac{\partial p^{nn}}{\partial x_i} d\Omega - \Delta t \int_\Omega \frac{\partial \hat{N}^T}{\partial x_i} \frac{\partial (\delta p)}{\partial x_i} d\Omega \\
&= Term6_1 + Term6_2
\end{aligned}
$$

Term 6_1

$$
\begin{aligned}
Term6_1 &= -\Delta t \int_\Omega \frac{\partial \hat{N}^T}{\partial x_i} \frac{\partial p^{nn}}{\partial x_i} d\Omega \\
&= -\Delta t \int_\Omega \frac{\partial \hat{N}^T}{\partial x_i} \frac{\partial \hat{N}}{\partial x_i} d\Omega \hat{p}^{nn}
\end{aligned}
$$

$$
\begin{aligned}
&= -V\Delta t \frac{\partial \hat{N}^T}{\partial x_i}\frac{\partial \hat{N}}{\partial x_i}\hat{p}^n n \\
&= -V\Delta t \begin{Bmatrix} \frac{\partial N_1}{\partial x_i} \\ \frac{\partial N_2}{\partial x_i} \\ \frac{\partial N_3}{\partial x_i} \\ \frac{\partial N_4}{\partial x_i} \end{Bmatrix} \begin{Bmatrix} \frac{\partial N_1}{\partial x_i} & \frac{\partial N_2}{\partial x_i} & \frac{\partial N_3}{\partial x_i} & \frac{\partial N_4}{\partial x_i} \end{Bmatrix} \begin{Bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{Bmatrix}^{nn} \\
&= -V\Delta t \begin{bmatrix} \frac{\partial N_1^2}{\partial x_i^2} & \frac{\partial N_1}{\partial x_i}\frac{\partial N_2}{\partial x_i} & \frac{\partial N_1}{\partial x_i}\frac{\partial N_3}{\partial x_i} & \frac{\partial N_1}{\partial x_i}\frac{\partial N_4}{\partial x_i} \\ \frac{\partial N_2}{\partial x_i}\frac{\partial N_1}{\partial x_i} & \frac{\partial N_2^2}{\partial x_i^2} & \frac{\partial N_2}{\partial x_i}\frac{\partial N_3}{\partial x_i} & \frac{\partial N_2}{\partial x_i}\frac{\partial N_4}{\partial x_i} \\ \frac{\partial N_3}{\partial x_i}\frac{\partial N_1}{\partial x_i} & \frac{\partial N_3}{\partial x_i}\frac{\partial N_2}{\partial x_i} & \frac{\partial N_3^2}{\partial x_i^2} & \frac{\partial N_3}{\partial x_i}\frac{\partial N_4}{\partial x_i} \\ \frac{\partial N_4}{\partial x_i}\frac{\partial N_1}{\partial x_i} & \frac{\partial N_4}{\partial x_i}\frac{\partial N_2}{\partial x_i} & \frac{\partial N_4}{\partial x_i}\frac{\partial N_3}{\partial x_i} & \frac{\partial N_4^2}{\partial x_i^2} \end{bmatrix} \begin{Bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{Bmatrix}^{nn} \\
&= -V\Delta t[H_{ii}]\{\hat{p}\}^{nn} \qquad \text{(C.23)}
\end{aligned}
$$

Term 6_2

$$
\begin{aligned}
Term6_2 &= -\Delta t \int_\Omega \frac{\partial \hat{N}^T}{\partial x_i}\frac{\partial (\delta p)^{nn}}{\partial x_i} d\Omega \\
&= -V\Delta t[H_{ii}]\{\delta\hat{p}\}^{nn} \qquad \text{(C.24)}
\end{aligned}
$$

Step 2 in fully discrete form

$$
\begin{aligned}
Termm &= Term5 + Term6 \\
Termm &= Term5 + Term6_1 + Term6_2
\end{aligned}
$$

i.e.

$$\frac{V}{20\Delta tc^2}[M]\{\Delta\hat{p}\} = -\frac{V\rho}{4}[ND_i]\{\hat{u}_i^*\} - V\Delta t[H_{ii}]\{\hat{p}\}^{nn} - V\Delta t[H_{ii}]\{\delta\hat{p}\}^{nn}$$

$$\text{i.e, } \frac{V}{20\Delta tc^2}[M]\{\Delta\hat{p}\} + \frac{V\rho}{4}[ND_i]\{\hat{u}_i^*\} + V\Delta t[H_{ii}]\{\delta\hat{p}\}^{nn} = -V\Delta t[H_{ii}]\{\hat{p}\}^{nn}$$

C.3 Spatial discrertization of step 3

$$\begin{aligned}\int_\Omega \hat{N}^T u_j^{n+1} d\Omega - \int_\Omega \hat{N}^T u_j^* d\Omega + \frac{\Delta t}{\rho}\int_\Omega \hat{N}^T \frac{\partial p}{\partial x_j} d\Omega &= 0 \\ Term7 + Term8 + Term9 &= 0\end{aligned}$$

Term 7

$$\begin{aligned}Term7 &= \int_\Omega \hat{N}^T u_j^{n+1} d\Omega \\ &= \int_\Omega \hat{N}^T u_j^{nn+1} d\Omega \\ &= \int_\Omega \hat{N}^T u_j^{nn} d\Omega + \int_\Omega \hat{N}^T \delta u_j d\Omega \\ &= \frac{V}{20}[M]\{\hat{u}_j\}^{nn} + \frac{V}{20}[M]\{\delta\hat{u}_j\} \qquad \text{(C.25)}\end{aligned}$$

Term 8

$$\begin{aligned}Term8 &= -\int_\Omega \hat{N}^T u_j^* d\Omega \\ &= -\frac{V}{20}[M]\{\hat{u}_j^*\} \qquad \text{(C.26)}\end{aligned}$$

Term 9

$$
\begin{aligned}
Term9 &= \frac{\Delta t}{\rho}\int_\Omega \hat{N}^T \frac{\partial p^{n+1}}{\partial x_j} d\Omega \\
&= \frac{\Delta t}{\rho}\int_\Omega \hat{N}^T \frac{\partial p^{nn+1}}{\partial x_j} d\Omega \\
&= \frac{\Delta t}{\rho}\int_\Omega \hat{N}^T \frac{\partial p^{nn}}{\partial x_j} d\Omega + \frac{\Delta t}{\rho}\int_\Omega \hat{N}^T \frac{\partial \delta p}{\partial x_j} d\Omega \\
&= Term9_1 + Term9_2
\end{aligned}
$$

***Term* 9_1**

$$
\begin{aligned}
Term9_1 &= \frac{\Delta t}{\rho}\int_\Omega \hat{N}^T \frac{\partial p^{nn}}{\partial x_j} d\Omega \\
&= -\frac{\Delta t}{\rho}\int_\Omega \frac{\partial \hat{N}^T}{\partial x_j} \hat{N} d\Omega \hat{p}^{nn} \\
&= -\frac{\Delta t}{\rho}\frac{V}{4}[D_j N]\{\hat{p}^{nn}\}
\end{aligned}
$$

***Term* 9_2**

$$
\begin{aligned}
Term9_2 &= \frac{\Delta t}{\rho}\int_\Omega \hat{N}^T \frac{\partial \delta p}{\partial x_j} d\Omega \\
&= -\frac{\Delta t}{\rho}\frac{V}{4}[D_j N]\{\delta \hat{p}\}
\end{aligned}
$$

Step 3 in fully discrete form

$$
\begin{aligned}
Term7 + Term8 + Term9 &= 0 \\
Term7 + Term8 + Term9_1 + Term9_2 &= 0
\end{aligned}
$$

i.e.

$$\frac{V}{20}[M]\{\hat{u}_j\}^{nn} + \frac{V}{20}[M]\{\delta\hat{u}_j\} - \frac{V}{20}[M]\{\hat{u}_j^*\} - \frac{\Delta t}{\rho}\frac{V}{4}[D_j N]\{\hat{p}^{nn}\} - \frac{\Delta t}{\rho}\frac{V}{4}[D_j N]\{\delta\hat{p}\} \approx 0$$

$$\frac{V}{20}[M]\{\delta\hat{u}_j\} - \frac{V}{20}[M]\{\hat{u}_j^*\} - \frac{\Delta t}{\rho}\frac{V}{4}[D_j N]\{\delta\hat{p}\} \approx -\frac{V}{20}[M]\{\hat{u}_j\}^{nn} + \frac{\Delta t}{\rho}\frac{V}{4}[D_j N]\{\hat{p}^{nn}\}$$

In summary the fully discrete form of steps 1 through 3 are as follows,

step 1

$$\frac{V}{20(\Delta t)}[M]\{\Delta\hat{u}_j^*\} + \frac{V}{20}[M]\{\hat{u}_i\}^{nn}\{D_i\}\{\delta\hat{u}_j\} + \frac{V}{20}\{D_i\}\{\hat{u}_j^{nn}\}[M]\{\delta\hat{u}_i\} +$$
$$\frac{\mu V}{\rho}[H_{ii}]\{\delta\hat{u}_j\} + \frac{\Delta t}{2\rho}\frac{V}{20}\{A\hat{U}X\}_k^{nn}\{\hat{u}_i\}^{nn}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\delta\hat{u}_j\} +$$
$$\frac{\Delta t}{2\rho}\frac{V}{20}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\hat{u}_j\}^{nn}\{A\hat{U}X\}_k^{nn}\{\delta\hat{u}_i\} + \frac{\Delta t}{2\rho}\frac{V}{20}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\hat{u}_j\}^{nn}\{A\hat{U}X\}_i^{nn}\{\delta\hat{u}_k\} =$$
$$-\frac{V}{20}[M]\{\hat{u}_i\}^{nn}\{D_i\}\{\hat{u}_j\}^{nn} - \frac{\mu V}{\rho}[H_{ii}]\{\hat{u}_j\}^{nn} - \frac{\Delta t}{2\rho}\frac{V}{20}\{A\hat{U}X\}_k^{nn}\{\hat{u}_i\}^{nn}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\hat{u}_j\}^{nn} \quad \text{(C.27)}$$

step 2

$$\frac{V}{20\Delta t c^2}[M]\{\Delta\hat{p}\} + \frac{V\rho}{4}[ND_i]\{\hat{u}_i^*\} + V\Delta t[H_{ii}]\{\delta\hat{p}\}^{nn} = -V\Delta t[H_{ii}]\{\hat{p}\}^{nn} \quad \text{(C.28)}$$

step 3

$$\frac{V}{20}[M]\{\delta\hat{u}_j\} - \frac{V}{20}[M]\{\hat{u}_j^*\} - \frac{\Delta t}{\rho}\frac{V}{4}[D_jN]\{\delta\hat{p}\} \approx -\frac{V}{20}[M]\{\hat{u}_j\}^{nn} + \frac{\Delta t}{\rho}\frac{V}{4}[D_jN]\{\hat{p}^{nn}\}$$

C.3.1 Monolithising step

Writing the step 3 equation in terms of 'i' index, to make the substitution from step 3 into step 2 possible.

Step 3 in terms of i

$$\frac{V}{20}[M]\{\delta\hat{u}_i\} - \frac{V}{20}[M]\{\hat{u}_i^*\} - \frac{\Delta t}{\rho}\frac{V}{4}[D_iN]\{\delta\hat{p}\} \approx -\frac{V}{20}[M]\{\hat{u}_i\}^{nn} + \frac{\Delta t}{\rho}\frac{V}{4}[D_iN]\{\hat{p}^{nn}\}$$

$$\{\hat{u}_i^*\} = \{\delta u_i\} - \frac{\Delta t}{\rho}\frac{V}{4}\frac{20}{V}[M^{-1}][D_iN]\{\delta\hat{p}\} + \{\hat{u}_i\}^{nn} - \frac{\Delta t}{\rho}\frac{V}{4}\frac{20}{V}[M^{-1}][D_iN]\{\hat{p}^{nn}\} \quad \text{(C.29)}$$

Substitute (C.29) in (C.27) and Eq. (C.29) in Eq. (C.28),

$$\frac{V}{20\Delta t}[M]\{\delta\hat{u}_j\} + \frac{V}{4\rho}[ND_j]\{\delta\hat{p}\} + \frac{V}{20\Delta t}[M]\{\hat{u}_j\}^{nn} + \frac{V}{4\rho}[ND_j]\{\hat{p}^{nn}\} +$$
$$\frac{V}{20}[M]\{\hat{u}_i\}^{nn}\{D_i\}\{\delta\hat{u}_j\} + \frac{V}{20}\{D_i\}\{\hat{u}_j^{nn}\}[M]\{\delta\hat{u}_i\} + \frac{\mu V}{\rho}[H_{ii}]\{\delta\hat{u}_j\} +$$
$$\frac{\Delta t}{2\rho}\frac{V}{20}\{A\hat{U}X\}_k^{nn}\{\hat{u}_i\}^{nn}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\delta\hat{u}_j\} + \frac{\Delta t}{2\rho}\frac{V}{20}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\hat{u}_j\}^{nn}\{A\hat{U}X\}_k^{nn}\{\delta\hat{u}_i\} +$$
$$\frac{\Delta t}{2\rho}\frac{V}{20}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\hat{u}_j\}^{nn}\{A\hat{U}X\}_i^{nn}\{\delta\hat{u}_k\} = -\frac{V}{20}[M]\{\hat{u}_i\}^{nn}\{D_i\}\{\hat{u}_j\}^{nn} - \frac{\mu V}{\rho}[H_{ii}]\{\hat{u}_j\}^{nn} +$$

$$\frac{V}{20(\Delta t)}[M]\{\hat{u}_j^n\} - \frac{\Delta t}{2\rho}\frac{V}{20}\{A\hat{U}X\}_k^{nn}\{\hat{u}_i\}^{nn}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\hat{u}_j\}^{nn} \tag{C.30}$$

$$\frac{V}{20\Delta t c^2}[M]\{\Delta\hat{p}\}^{n+1} + \frac{V\rho}{4}[ND_i]\left(\{\delta u_i\} - \frac{5\Delta t}{\rho}[M^{-1}][D_iN]\{\delta\hat{p}\} + \{\hat{u}_i\}^{nn}\right) -$$
$$\frac{V\rho}{4}[ND_i]\left(\frac{5\Delta t}{\rho}[M^{-1}][D_iN]\{\hat{p}^{nn}\}\right) + V\Delta t[H_{ii}]\{\delta\hat{p}\}^{nn} = -V\Delta t[H_{ii}]\{\hat{p}\}^{nn} \tag{C.31}$$

Rearranging,

$$\left(\frac{V}{20}\{D_i\}\{\hat{u}_j^{nn}\}[M] + \frac{\Delta t}{2\rho}\frac{V}{20}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\hat{u}_j\}^{nn}\{A\hat{U}X\}_k^{nn}\right)\{\delta\hat{u}_i\}$$
$$\left(\frac{V}{20\Delta t}[M] + \frac{V}{20}[M]\{\hat{u}_i\}^{nn}\{D_i\} + \frac{\mu V}{\rho}[H_{ii}] + \frac{\Delta t}{2\rho}\frac{V}{20}\{A\hat{U}X\}_k^{nn}\{\hat{u}_i\}^{nn}\{\hat{D}_k^T\}\{\hat{D}_i\}\right)\{\delta\hat{u}_j\} +$$
$$\left(\frac{\Delta t}{2\rho}\frac{V}{20}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\hat{u}_j\}^{nn}\{A\hat{U}X\}_i^{nn}\right)\{\delta\hat{u}_k\} + \left(\frac{V}{4\rho}[ND_j]\right)\{\delta\hat{p}\} \approx$$
$$-\frac{V}{20\Delta t}[M]\{\hat{u}_j\}^{nn} - \frac{V}{4\rho}[ND_j]\{\hat{p}^{nn}\} - \frac{V}{20}[M]\{\hat{u}_i\}^{nn}\{D_i\}\{\hat{u}_j\}^{nn} -$$
$$\frac{\mu V}{\rho}[H_{ii}]\{\hat{u}_j\}^{nn} + \frac{V}{20(\Delta t)}[M]\{\hat{u}_j^n\} - \frac{\Delta t}{2\rho}\frac{V}{20}\{A\hat{U}X\}_k^{nn}\{\hat{u}_i\}^{nn}\{\hat{D}_k^T\}\{\hat{D}_i\}\{\hat{u}_j\}^{nn} \tag{C.32}$$

$$\cancel{\frac{V}{20\Delta t c^2}[M]\{\Delta\hat{p}\}} + \frac{V\rho}{4}[ND_i]\{\delta\hat{u}_i\} + \left(-\frac{5\Delta t V}{4}[ND_i][M^{-1}][D_iN] + V\Delta t[H_{ii}]\right)\{\delta\hat{p}\} \approx$$
$$\frac{V\rho}{4}[ND_i]\left(\frac{5\Delta t}{\rho}[M^{-1}][D_iN]\{\hat{p}^{nn}\}\right) - V\Delta t[H_{ii}]\{\hat{p}\}^{nn} - \frac{V\rho}{4}[ND_i]\{\hat{u}_i\}^{nn} \tag{C.33}$$

Just the δ terms are retained in the LHS to form the Jacobain matrix and the terms at Newton iteration nn are moved to the RHS.

C.4 Monolithic equations without indices

Equations (C.32) and (C.33) represent the fully discrete form of incompressible Navier-Stokes equations, in monolithic form and will be expanded in this section to get the index-free form, that makes the coding process easier.

j = 1, i = 1 to 3 & k = 1 to 3 in (C.32) leads to,

$$\begin{aligned}
&\frac{V}{20}\{D_1\}\{\hat{u}_1^{nn}\}[M]\{\delta\hat{u}_1\} + \frac{V}{20}\{D_2\}\{\hat{u}_1^{nn}\}[M]\{\delta\hat{u}_2\} + \frac{V}{20}\{D_3\}\{\hat{u}_1^{nn}\}[M]\{\delta\hat{u}_3\} + \\
&\left(\frac{V}{20\Delta t}[M] + \frac{V}{20}[M]\left(\{\hat{u}_1\}^{nn}\{D_1\} + \{\hat{u}_2\}^{nn}\{D_2\} + \{\hat{u}_3\}^{nn}\{D_3\}\right)\right)\{\delta\hat{u}_1\} + \\
&\frac{\mu V}{\rho}[H_{11} + H_{22} + H_{33}]\{\delta\hat{u}_1\} + \frac{V}{4\rho}[ND_1]\{\delta\hat{p}\} \approx -\frac{V}{20\Delta t}[M]\{\hat{u}_1\}^{nn} - \frac{V}{4\rho}[ND_1]\{\hat{p}^{nn}\} - \\
&\frac{V}{20}[M]\left(\{\hat{u}_1\}^{nn}\{D_1\} + \{\hat{u}_2\}^{nn}\{D_2\} + \{\hat{u}_3\}^{nn}\{D_3\}\right)\{\hat{u}_1\}^{nn} - \\
&\frac{\mu V}{\rho}[H_{11} + H_{22} + H_{33}]\{\hat{u}_1\}^{nn} + \frac{V}{20(\Delta t)}[M]\{\hat{u}_1^n\}
\end{aligned} \tag{C.34}$$

i.e.

$$\alpha_1\{\delta\hat{u}_1\} + \alpha_2\{\delta\hat{u}_2\} + \alpha_3\{\delta\hat{u}_3\} + \beta\{\delta\hat{p}\} = \gamma \tag{C.35}$$

j = 2, i = 1 to 3 & k = 1 to 3 in (C.32) leads to,

$$\begin{aligned}
&\frac{V}{20}\{D_1\}\{\hat{u}_2^{nn}\}[M]\{\delta\hat{u}_1\} + \frac{V}{20}\{D_2\}\{\hat{u}_2^{nn}\}[M]\{\delta\hat{u}_2\} + \frac{V}{20}\{D_3\}\{\hat{u}_2^{nn}\}[M]\{\delta\hat{u}_3\} + \\
&\left(\frac{V}{20\Delta t}[M] + \frac{V}{20}[M]\left(\{\hat{u}_1\}^{nn}\{D_1\} + \{\hat{u}_2\}^{nn}\{D_2\} + \{\hat{u}_3\}^{nn}\{D_3\}\right)\right)\{\delta\hat{u}_2\} + \\
&\frac{\mu V}{\rho}[H_{11} + H_{22} + H_{33}]\{\delta\hat{u}_2\} + \frac{V}{4\rho}[ND_2]\{\delta\hat{p}\} \approx -\frac{V}{20\Delta t}[M]\{\hat{u}_2\}^{nn} - \frac{V}{4\rho}[ND_2]\{\hat{p}^{nn}\} -
\end{aligned}$$

$$\frac{V}{20}[M]\left(\{\hat{u}_1\}^{nn}\{D_1\}+\{\hat{u}_2\}^{nn}\{D_2\}+\{\hat{u}_3\}^{nn}\{D_3\}\right)\{\hat{u}_2\}^{nn}-$$
$$\frac{\mu V}{\rho}[H_{11}+H_{22}+H_{33}]\{\hat{u}_2\}^{nn}+\frac{V}{20(\Delta t)}[M]\{\hat{u}_2^n\} \tag{C.36}$$

i.e.

$$\delta_1\{\delta\hat{u}_1\}+\delta_2\{\delta\hat{u}_2\}+\delta_3\{\delta\hat{u}_3\}+\epsilon\{\delta\hat{p}\}=\zeta \tag{C.37}$$

j = 3, i = 1 to 3 & k = 1 to 3 in (C.32) leads to,

$$\frac{V}{20}\{D_1\}\{\hat{u}_3^{nn}\}[M]\{\delta\hat{u}_1\}+\frac{V}{20}\{D_2\}\{\hat{u}_3^{nn}\}[M]\{\delta\hat{u}_2\}+\frac{V}{20}\{D_3\}\{\hat{u}_3^{nn}\}[M]\{\delta\hat{u}_3\}+$$
$$\left(\frac{V}{20\Delta t}[M]+\frac{V}{20}[M]\left(\{\hat{u}_1\}^{nn}\{D_1\}+\{\hat{u}_2\}^{nn}\{D_2\}+\{\hat{u}_3\}^{nn}\{D_3\}\right)\right)\{\delta\hat{u}_3\}+$$
$$\frac{\mu V}{\rho}[H_{11}+H_{22}+H_{33}]\{\delta\hat{u}_3\}+\frac{V}{4\rho}[ND_3]\{\delta\hat{p}\}\approx-\frac{V}{20\Delta t}[M]\{\hat{u}_3\}^{nn}-\frac{V}{4\rho}[ND_3]\{\hat{p}^{nn}\}-$$
$$\frac{V}{20}[M]\left(\{\hat{u}_1\}^{nn}\{D_1\}+\{\hat{u}_2\}^{nn}\{D_2\}+\{\hat{u}_3\}^{nn}\{D_3\}\right)\{\hat{u}_3\}^{nn}-$$
$$\frac{\mu V}{\rho}[H_{11}+H_{22}+H_{33}]\{\hat{u}_3\}^{nn}+\frac{V}{20(\Delta t)}[M]\{\hat{u}_3^n\} \tag{C.38}$$

i.e.

$$\eta_1\{\delta\hat{u}_1\}+\eta_2\{\delta\hat{u}_2\}+\eta_3\{\delta\hat{u}_3\}+\theta\{\delta\hat{p}\}=\iota \tag{C.39}$$

no j, i = 1 to 3 in (C.33) leads to,

$$\frac{V\rho}{4}[ND_1]\{\delta\hat{u}_1\}+\frac{V\rho}{4}[ND_2]\{\delta\hat{u}_2\}+\frac{V\rho}{4}[ND_3]\{\delta\hat{u}_3\}+V\Delta t[H_{11}+H_{22}+H_{33}]\{\delta\hat{p}\}-$$
$$\frac{5\Delta tV}{4}\left([ND_1][M^{-1}][D_1N]+[ND_2][M^{-1}][D_2N]+[ND_3][M^{-1}][D_3N]\right)\{\delta\hat{p}\}\approx$$
$$\frac{5\Delta tV}{4}\left([ND_1][M^{-1}][D_1N]+[ND_2][M^{-1}][D_2N]+[ND_3][M^{-1}][D_3N]\right)\{\hat{p}\}^{nn}-$$
$$V\Delta t[H_{11}+H_{22}+H_{33}]\{\hat{p}\}^{nn}-\frac{V\rho}{4}[ND_1]\{\hat{u}_1\}^{nn}-\frac{V\rho}{4}[ND_2]\{\hat{u}_2\}^{nn}-\frac{V\rho}{4}[ND_3]\{\hat{u}_3\}^{nn} \tag{C.40}$$

i.e.

$$\kappa\{\hat{u}_1\}^{n+1} + \lambda\{\hat{u}_2\}^{n+1} + \mu\{\hat{u}_3\}^{n+1} + \nu\{\hat{p}\}^{n+1} = \xi \tag{C.41}$$

In matrix form,

$$\begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \beta \\ \delta_1 & \delta_2 & \delta_3 & \epsilon \\ \eta_1 & \eta_2 & \eta_3 & \theta \\ \kappa & \lambda & \mu & \nu \end{bmatrix} \begin{Bmatrix} \delta\hat{u}_1 \\ \delta\hat{u}_2 \\ \delta\hat{u}_3 \\ \delta\hat{p} \end{Bmatrix} = \begin{Bmatrix} \gamma \\ \zeta \\ \iota \\ \xi \end{Bmatrix} \tag{C.42}$$

For linear tetrahedron elements, every matrix entry is a matrix of size (4*4) and every vector entry is a vector of size (4*1).

C.5 Non-linear context

A typical non-linear system may be represented as,

$$f_n(\phi) = 0 \tag{C.43}$$

and the update of the Newton iteration is,

$$\phi_{n+1} = \phi_n - \frac{f(\phi)}{f'(\phi)} \tag{C.44}$$

The SNES (Scalable Nonlinear Equation Solver) solvers in PETSc approximately solve,

$$f'(\phi)\Delta\phi = -f(\phi) \tag{C.45}$$

In general, the jacobian matrix, may be written as,

$$[J] = f'(\phi) = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \frac{\partial f_1}{\partial u_2} & \frac{\partial f_1}{\partial u_3} & \frac{\partial f_1}{\partial p} \\ \frac{\partial f_2}{\partial u_1} & \frac{\partial f_2}{\partial u_2} & \frac{\partial f_2}{\partial u_3} & \frac{\partial f_2}{\partial p} \\ \frac{\partial f_3}{\partial u_1} & \frac{\partial f_3}{\partial u_2} & \frac{\partial f_3}{\partial u_3} & \frac{\partial f_3}{\partial p} \\ \frac{\partial f_4}{\partial u_1} & \frac{\partial f_4}{\partial u_2} & \frac{\partial f_4}{\partial u_3} & \frac{\partial f_4}{\partial p} \end{bmatrix} \tag{C.46}$$

Since the system in Eq. (C.42) has been formulated in terms of δu, the system matrix itself serves as the Jacobian matrix, making the code highly efficient. Therefore,

$$f'(x) = [J] = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & \beta \\ \delta_1 & \delta_2 & \delta_3 & \epsilon \\ \eta_1 & \eta_2 & \eta_3 & \theta \\ \kappa & \lambda & \mu & \nu \end{bmatrix} \tag{C.47}$$

It should be noted here, that the RHS in Eq. (C.42) is equivalent to the matrix vector product of the system matrix and the solution vector at a guess, had the system be formulated in terms of u instead of δu.

C.6 Definitions

$$[ND_i] = \begin{Bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{Bmatrix} \frac{\partial}{\partial x_i} \begin{Bmatrix} N_1 & N_2 & N_3 & N_4 \end{Bmatrix} = \begin{bmatrix} \frac{\partial N_1}{\partial x_i} & \frac{\partial N_2}{\partial x_i} & \frac{\partial N_3}{\partial x_i} & \frac{\partial N_4}{\partial x_i} \\ \frac{\partial N_1}{\partial x_i} & \frac{\partial N_2}{\partial x_i} & \frac{\partial N_3}{\partial x_i} & \frac{\partial N_4}{\partial x_i} \\ \frac{\partial N_1}{\partial x_i} & \frac{\partial N_2}{\partial x_i} & \frac{\partial N_3}{\partial x_i} & \frac{\partial N_4}{\partial x_i} \\ \frac{\partial N_1}{\partial x_i} & \frac{\partial N_2}{\partial x_i} & \frac{\partial N_3}{\partial x_i} & \frac{\partial N_4}{\partial x_i} \end{bmatrix} \tag{C.48}$$

$$[D_i N] = \frac{\partial}{\partial x_i} \begin{Bmatrix} N_1 \\ N_2 \\ N_3 \\ N_4 \end{Bmatrix} \begin{Bmatrix} 1 & 1 & 1 & 1 \end{Bmatrix} = \begin{bmatrix} \frac{\partial N_1}{\partial x_i} & \frac{\partial N_1}{\partial x_i} & \frac{\partial N_1}{\partial x_i} & \frac{\partial N_1}{\partial x_i} \\ \frac{\partial N_2}{\partial x_i} & \frac{\partial N_2}{\partial x_i} & \frac{\partial N_2}{\partial x_i} & \frac{\partial N_2}{\partial x_i} \\ \frac{\partial N_3}{\partial x_i} & \frac{\partial N_3}{\partial x_i} & \frac{\partial N_3}{\partial x_i} & \frac{\partial N_3}{\partial x_i} \\ \frac{\partial N_4}{\partial x_i} & \frac{\partial N_4}{\partial x_i} & \frac{\partial N_4}{\partial x_i} & \frac{\partial N_4}{\partial x_i} \end{bmatrix} \tag{C.49}$$

Appendix D

CSR matrix and preallocation example

In this appendix, a small pseudo sparse matrix system is considered. The CSR sparse matrix representation will be described, followed by an accurate preallocation calculation. The following matrix system is considered in this appendix. Its rows are assumed to be contiguously partitioned across 3 processors.

$$\begin{array}{l} Proc0 \\ \\ \\ Proc1 \\ \\ Proc2 \end{array} \left[\begin{array}{cc:cc:c} 1 & 2 & 0 & 0 & 0 \\ 0 & 3 & 4 & 5 & 0 \\ \hdashline 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 8 & 9 \\ \hdashline 0 & 10 & 0 & 0 & 11 \end{array}\right] \approx \left[\begin{array}{c:c:c} A & B & C \\ \hdashline D & E & F \\ \hdashline G & H & I \end{array}\right] \tag{D.1}$$

D.1 CSR representation

CSR is an acronym for the Compressed Sparse Row storage representation. It is almost identical to the yale format. Within the PETSc framework, the CSR representation is called the AIJ format. This format is used to efficiently store

matrices when the number of non zero entries (NNZ) are significantly smaller than the number of zero entries, i.e. when the matrix is sparse. Memory saving occurs by virtue of CSR storage, if,

$$NNZ < \frac{m(n-1)-1}{2}$$

where m and n are the number of rows and columns, respectively.

The CSR format stores a sparse matrix as a collection three 1D arrays - CSRval, CSRcol and CSRrow. The CSRval array stores all the non-zero entries, the CSRcol array stores the column index of the non-zero entries and the CSRrow array stores the indices at which every row starts. The CSRval array is simply written by making a list of all non-zero entries occurring in the matrix in a row wise manner from left to right. The number of entries in CSRval array is therefore equal to NNZ. As the CSRval array is being generated, a simultaneous record of the corresponding column indices generates the CSRcol array. Consequently, the length of CSRval array is also equal to NNZ. The CSRrow array is relatively complicated to construct. Depending on whether a 1 based or 0 based indexing is being used, the first entry of the CSRrow array will be 1 or 0, respectively. 1 based indexing is used here. For the second entry, the number of non-zeros occurring in row 1 are counted and this number is incremented with the previous entry (i.e. 1), to get the second entry of the CSRrow array. The same procedure is repeated for the remaining rows. The number of entries in the CSRrow array is one more than the number of rows. The data contained in the last entry of the CSRrow array is one more than the total number of non-zeros in the original sparse matrix.

The CSR representation of the matrix appearing in Eq. D.1 is,

$$\text{CSRval} = \begin{Bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 8 & 9 & 10 & 11 \end{Bmatrix}$$

$$\text{CSRcol} = \left\{1 \;\; 2 \;\; 2 \;\; 3 \;\; 4 \;\; 3 \;\; 4 \;\; 5 \;\; 2 \;\; 5\right\}$$
$$\text{CSRrow} = \left\{1 \;\; 3 \;\; 6 \;\; 7 \;\; 9 \;\; 11\right\}$$

From the preallocation point of view, the CSRrow data may be used to calculate the exact number of non-zeros in every row of the sparse matrix. This can be simply done by taking the difference of consecutive entries. From the matrix partitioning information, this data can be made patch specific. As described in Section D.2, the NNZ data for diagonal and off-diagonal blocks may be generated by also taking into consideration the column numbers of the non-zero entries from the CSRcol array.

D.2 Preallocation

In Eq. D.1, the solid lines represent the matrix partitioning across the participating processors and the dotted lines separate the diagonal (D) blocks from the off-diagonal (OD) ones. This distinction will usually be required to be made during the preallocation of memory for matrices in PETSc, as each processor stores the diagonal and off-diagonal blocks as separate serial matrices. The RHS of Eq. D.1, represents the block matrix form of the matrix on the LHS. The sub matrices [**A**], [**B**] and [**C**] are owned by Proc0 (first processor), [**D**], [**E**] and [**F**] are owned by Proc1 (second processor) and [**G**], [**H**] and [**I**] are owned by Proc2 (third processor). For Proc0, [**A**] is the diagonal block and [**B**] and [**C**] are the off-diagonal blocks. For Proc1, [**E**] is the diagonal block and [**D**] and [**F**] are the off-diagonal blocks. Similarly, for Proc2, [**I**] is the diagonal block and [**G**] and [**H**] are the off-diagonal blocks. By counting the number of non-zeros occurring in every row of both the diagonal and off-diagonal blocks of all processors, the required preallocation data is constructed.

Proc rank	NNZ_D	NNZ_{OD}
P0	(2,1)	(0,2)
P1	(1,1)	(0,1)
P2	(1)	(1)

Table D.1: Matrix preallocation data

As an example, the second processor (Proc 1) is considered. Since just 1 entry occurs in the first (6) and second (8) row of sub matrix **[E]**, $NNZ_D = [1, 1]$. In sub matrices **[D]** and **[F]**, put together, no non-zero entries appear in the first row and just 1 entry (9) occurs in the second row, therefore, $NNZ_{OD} = [0, 1]$. In summary, the complete preallocation data is presented in Table D.1.

Appendix E

Closed form expressions

In this appendix, the closed form expressions for the integral of shape functions and the shape function derivatives with respect to all spatial dimensions will be enclosed for the case of linear, 4 noded, tetrahedral elements, that were employed in construction of Finite element meshes used in this research.

E.1 Integral of shape functions

The following closed form expression may be used for evaluating the typical integrals. This prevents the need to perform numerical integration, thereby making the computations less expensive.

$$\int_{\Omega} N_1^i N_2^j N_3^k N_4^l \; d\Omega = \frac{i!j!k!l!}{(i+j+k+l+3)!} 6V \tag{E.1}$$

E.2 Shape function derivatives

In cartesian coordinates, the shape functions for linear tetrahedral elements are given by,

$$
\begin{aligned}
N_1 &= +\frac{1}{|A|}\begin{vmatrix} x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \end{vmatrix} - \frac{1}{|A|}\begin{vmatrix} 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \\ 1 & y_4 & z_4 \end{vmatrix} x + \frac{1}{|A|}\begin{vmatrix} 1 & x_2 & z_2 \\ 1 & x_3 & z_3 \\ 1 & x_4 & z_4 \end{vmatrix} y - \frac{1}{|A|}\begin{vmatrix} 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \\ 1 & x_4 & y_4 \end{vmatrix} z \\
N_2 &= -\frac{1}{|A|}\begin{vmatrix} x_1 & y_1 & z_1 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \end{vmatrix} + \frac{1}{|A|}\begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_3 & z_3 \\ 1 & y_4 & z_4 \end{vmatrix} x - \frac{1}{|A|}\begin{vmatrix} 1 & x_1 & z_1 \\ 1 & x_3 & z_3 \\ 1 & x_4 & z_4 \end{vmatrix} y + \frac{1}{|A|}\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_3 & y_3 \\ 1 & x_4 & y_4 \end{vmatrix} z \\
N_3 &= +\frac{1}{|A|}\begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_4 & y_4 & z_4 \end{vmatrix} - \frac{1}{|A|}\begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_4 & z_4 \end{vmatrix} x + \frac{1}{|A|}\begin{vmatrix} 1 & x_1 & z_1 \\ 1 & x_2 & z_2 \\ 1 & x_4 & z_4 \end{vmatrix} y - \frac{1}{|A|}\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_4 & y_4 \end{vmatrix} z \\
N_4 &= -\frac{1}{|A|}\begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix} + \frac{1}{|A|}\begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} x - \frac{1}{|A|}\begin{vmatrix} 1 & x_1 & z_1 \\ 1 & x_2 & z_2 \\ 1 & x_3 & z_3 \end{vmatrix} y + \frac{1}{|A|}\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix} z
\end{aligned}
\tag{E.2}
$$

where

$$
|A| = \begin{vmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 \end{vmatrix} \tag{E.3}
$$

Applying the following row operations, we can simplify the calculation,

$$
\begin{aligned}
R_1' &= R_1 - R_2 \\
R_2' &= R_2 - R_3
\end{aligned}
$$

$$R_3' = R_3 - R_4$$

Therefore,

$$\begin{aligned}
|A| &= \begin{vmatrix} 1 & x_1 & y_1 & z_1 \\ 1 & x_2 & y_2 & z_2 \\ 1 & x_3 & y_3 & z_3 \\ 1 & x_4 & y_4 & z_4 \end{vmatrix} \\
&= \begin{vmatrix} 0 & (x_1 - x_2) & (y_1 - y_2) & (z_1 - z_2) \\ 0 & (x_2 - x_3) & (y_2 - y_3) & (z_2 - z_3) \\ 0 & (x_3 - x_4) & (y_3 - y_4) & (z_3 - z_4) \\ 1 & x_4 & y_4 & z_4 \end{vmatrix} \\
&= 1 \begin{vmatrix} (x_1 - x_2) & (y_1 - y_2) & (z_1 - z_2) \\ (x_2 - x_3) & (y_2 - y_3) & (z_2 - z_3) \\ (x_3 - x_4) & (y_3 - y_4) & (z_3 - z_4) \end{vmatrix}
\end{aligned}$$

$$\begin{aligned}
|A| = {} & (z_1 - z_2)(x_2 y_3 - x_3 y_2) + (z_1 - z_3)(x_4 y_2 - x_2 y_4) + (z_1 - z_4)(x_3 y_4 - x_4 y_3) \\
+ {} & (z_3 - z_2)(x_4 y_1 - x_1 y_4) + (z_2 - z_4)(x_3 y_1 - x_1 y_3) + (z_3 - z_4)(x_1 y_2 - x_2 y_1)
\end{aligned} \tag{E.4}$$

Next, the shape function derivatives are enclosed,

$$\frac{\partial N_1}{\partial x} = -\frac{1}{|A|} \begin{vmatrix} 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \\ 1 & y_4 & z_4 \end{vmatrix} = \frac{1}{|A|}[-(y_3 z_4 - y_4 z_3) + (y_2 z_4 - y_4 z_2) - (y_2 z_3 - y_3 z_2)]$$

$$\frac{\partial N_2}{\partial x} = +\frac{1}{|A|}\begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_3 & z_3 \\ 1 & y_4 & z_4 \end{vmatrix} = -\frac{1}{|A|}[-(y_4z_1 - y_1z_4) + (y_3z_1 - y_1z_3) - (y_3z_4 - y_4z_3)]$$

$$\frac{\partial N_3}{\partial x} = -\frac{1}{|A|}\begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_4 & z_4 \end{vmatrix} = \frac{1}{|A|}[-(y_1z_2 - y_2z_1) + (y_4z_2 - y_2z_4) - (y_4z_1 - y_1z_4)]$$

$$\frac{\partial N_4}{\partial x} = +\frac{1}{|A|}\begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} = -\frac{1}{|A|}[-(y_2z_3 - y_3z_2) + (y_1z_3 - y_3z_1) - (y_1z_2 - y_2z_1)]$$

$$\frac{\partial N_1}{\partial y} = +\frac{1}{|A|}\begin{vmatrix} 1 & x_2 & z_2 \\ 1 & x_3 & z_3 \\ 1 & x_4 & z_4 \end{vmatrix} = \frac{1}{|A|}[x_2(z_3 - z_4) + x_3(z_4 - z_2) + x_4(z_2 - z_3)]$$

$$\frac{\partial N_2}{\partial y} = -\frac{1}{|A|}\begin{vmatrix} 1 & x_1 & z_1 \\ 1 & x_3 & z_3 \\ 1 & x_4 & z_4 \end{vmatrix} = -\frac{1}{|A|}[x_3(z_4 - z_1) + x_4(z_1 - z_3) + x_1(z_3 - z_4)]$$

$$\frac{\partial N_3}{\partial y} = +\frac{1}{|A|}\begin{vmatrix} 1 & x_1 & z_1 \\ 1 & x_2 & z_2 \\ 1 & x_4 & z_4 \end{vmatrix} = \frac{1}{|A|}[x_4(z_1 - z_2) + x_1(z_2 - z_4) + x_2(z_4 - z_1)]$$

$$\frac{\partial N_4}{\partial y} = -\frac{1}{|A|}\begin{vmatrix} 1 & x_1 & z_1 \\ 1 & x_2 & z_2 \\ 1 & x_3 & z_3 \end{vmatrix} = -\frac{1}{|A|}[x_1(z_2 - z_3) + x_2(z_3 - z_1) + x_3(z_1 - z_2)]$$

$$\frac{\partial N_1}{\partial z} = -\frac{1}{|A|}\begin{vmatrix} 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \\ 1 & x_4 & y_4 \end{vmatrix} = \frac{1}{|A|}[x_2(y_4 - y_3) + x_3(y_2 - y_4) + x_4(y_3 - y_2)]$$

$$\frac{\partial N_2}{\partial z} = +\frac{1}{|A|}\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_3 & y_3 \\ 1 & x_4 & y_4 \end{vmatrix} = -\frac{1}{|A|}[x_3(y_1 - y_4) + x_4(y_3 - y_1) + x_1(y_4 - y_3)]$$

$$\frac{\partial N_3}{\partial z} = -\frac{1}{|A|}\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_4 & y_4 \end{vmatrix} = \frac{1}{|A|}[x_4(y_2 - y_1) + x_1(y_4 - y_2) + x_2(y_1 - y_4)]$$

$$\frac{\partial N_4}{\partial z} = +\frac{1}{|A|}\begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix} = -\frac{1}{|A|}[x_1(y_3 - y_2) + x_2(y_1 - y_3) + x_3(y_2 - y_1)]$$

Bibliography

[1] *MPI implementation of CFD program PHOENICS.* Oct 1994.

[2] Intel math kernel library reference manual. Technical Report 630813-052US, Intel Corporation, 2012. URL `https://software.intel.com/en-us/articles/intel-math-kernel-library-documentation`.

[3] The nobel prize in phsiology or medicine 1998: Award ceremony speech, 2014. URL `http://www.nobelprize.org/nobel_prizes/medicine/laureates/1998/presentation-speech.html`.

[4] C. Alberto Figueroa, Seungik Baek, Charles A. Taylor, and Jay D. Humphrey. A computational framework for fluid–solid-growth modeling in cardiovascular simulations. *Computer methods in applied mechanics and engineering*, 198(45):3583–3602, 2009.

[5] Patrick R. Amestoy, Timothy A. Davis, and Iain S. Duff. Algorithm 837: Amd, an approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw.*, 30(3):381–388, September 2004. ISSN 0098-3500. doi: 10.1145/1024074.1024081. URL `http://doi.acm.org/10.1145/1024074.1024081`.

[6] Matteo Astorino, Jean-Frédéric Gerbeau, Olivier Pantz, and Karim-Frédéric Traore. Fluid–structure interaction and multi-body contact: application to aortic valves. *Computer Methods in Applied Mechanics and Engineering*, 198(45):3603–3612, 2009.

[7] A.K. Aziz, Baltimore County. Division of Mathematics University of Maryland, and United States. Office of Naval Research. *The Mathematical Foundations of the Finite Element Method with Applications to Partial Differential Equations.* Academic Press Rapid Manuscript Reproduction. Academic Press, 1972. URL `http://books.google.co.uk/books?id=7PZQAAAAMAAJ`.

[8] Ivo Babuška. The finite element method with penalty. *Mathematics of computation*, 27(122):221–228, 1973.

[9] Ivo Babuška and Theofanis Strouboulis. *The finite element method and its reliability*. Oxford university press, 2001.

[10] Allison H. Baker, Elizabeth R. Jessup, and Thomas Manteuffel. A technique for accelerating the convergence of restarted gmres. *SIAM Journal on Matrix Analysis and Applications*, 26(4):962–984, 2005.

[11] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.5, Argonne National Laboratory, 2014. URL `http://www.mcs.anl.gov/petsc`.

[12] R. Balossino, G. Pennati, F. Migliavacca, L. Formaggia, A. Veneziani, M. Tuveri, and G. Dubini. Computational models to predict stenosis growth in carotid arteries: Which is the role of boundary conditions? *Computer methods in biomechanics and biomedical engineering*, 12(1):113–123, 2009.

[13] Klaus-Jürgen Bathe. The inf–sup condition and its evaluation for mixed finite element methods. *Computers & structures*, 79(2):243–252, 2001.

[14] Y. Bazilevs, J.R. Gohean, T.J.R. Hughes, R.D. Moser, and Y. Zhang. Patient-specific isogeometric fluid–structure interaction analysis of thoracic aortic blood flow due to implantation of the jarvik 2000 left ventricular assist device. *Computer Methods in Applied Mechanics and Engineering*, 198 (45):3534–3550, 2009.

[15] Michele Benzi and Maxim A. Olshanskii. An augmented lagrangian-based approach to the oseen problem. *SIAM Journal on Scientific Computing*, 28 (6):2095–2113, 2006.

[16] Sonja Berner. Parallel methods for verified global optimization practice and theory. *Journal of Global Optimization*, 9(1):1–22, 1996.

[17] R.L.T. Bevan, P. Nithiarasu, R. Van Loon, I. Sazonov, H. Luckraz, and A. Garnham. Application of a locally conservative galerkin (lcg) method for modelling blood flow through a patient-specific carotid bifurcation. *International Journal for Numerical Methods in Fluids*, 64(10-12):1274–1295, 2010.

[18] Rupak Biswas, Dochan Kwak, Cetin Kiris, and Scott Lawrence. Impact of the columbia supercomputer on nasa space and exploration missions. In *Space Mission Challenges for Information Technology, 2006. SMC-IT 2006. Second IEEE International Conference on*, pages 8–pp. IEEE, 2006.

[19] Pavel B. Bochev, Clark R. Dohrmann, and Max D. Gunzburger. Stabilization of low-order mixed finite elements for the stokes equations. *SIAM Journal on Numerical Analysis*, 44(1):82–101, 2006.

[20] S.S. Bossers, M. Cibis, F.J. Gijsen, M. Schokking, J.L. Strengers, R.F. Verhaart, A. Moelker, J.J. Wentsel, and W.A. Helbing. Computational fluid dynamics in fontan patients to evaluate power loss during simulated exercise. *Heart*, 100:696–701, 2014.

[21] Lorenzo Botti, Marina Piccinelli, Bogdan Ene-Iordache, Andrea Remuzzi, and Luca Antiga. An adaptive mesh refinement solver for large-scale simulation of biological flows. *International Journal for Numerical Methods in Biomedical Engineering*, 26(1):86–100, 2010.

[22] Xiao-Chuan Cai, David E Keyes, and Venkatasubramanian Venkatakrishnan. Newton-krylov-schwarz: An implicit solver for cfd. Technical report, DTIC Document, 1995.

[23] James A. Cardle. A modification of the petrov–galerkin method for the transient convection–diffusion equation. *International journal for numerical methods in engineering*, 38(2):171–181, 1995.

[24] Juan R Cebral, Christopher M Putman, Marcus T Alley, Thomas Hope, Roland Bammer, and Fernando Calamante. Hemodynamics in normal cerebral arteries: qualitative comparison of 4d phase-contrast magnetic resonance and image-based computational fluid dynamics. *Journal of engineering mathematics*, 64(4):367–378, 2009.

[25] Yiannis S. Chatzizisis, Michael Jonas, Ahmet U. Coskun, Roy Beigel, Benjamin V. Stone, Charles Maynard, Ross G. Gerrity, William Daley, Campbell Rogers, Elazer R. Edelman, et al. Prediction of the localization of high-risk coronary atherosclerotic plaques on the basis of low endothelial shear stress an intravascular ultrasound and histopathology natural history study. *Circulation*, 117(8):993–1002, 2008.

[26] YK Cheung and OC Zinkiewicz. Plates and tanks on elastic foundation-sâĂŤan application of finite element method. *International Journal of Solids and structures*, 1(4):451–461, 1965.

[27] A. J. Chorin. Numerical solution of navier-stokes equations. *Math. comput.*, 22:745–762, 1968.

[28] A. J. Chorin. On the convergence of discrete approximation to the navier-stokes equations. *Math. comput.*, 23:341–353, 1969.

[29] Alexandre Joel Chorin. A numerical method for solving incompressible viscous flow problems. *Journal of computational physics*, 2(1):12–26, 1967.

[30] TJ Chung. Finite element analysis in fluid dynamics. *NASA STI/Recon Technical Report A*, 78:44102, 1978.

[31] T.J. Chung. *Computational Fluid Dynamics*. Cambridge University Press, 2002. ISBN 9780521594165. URL `http://books.google.co.uk/books?id=Un1vG371Yq4C`.

[32] Jonathan R Clausen, Daniel A Reasor, and Cyrus K Aidun. Parallel performance of a lattice-boltzmann/finite element cellular blood flow solver on the ibm blue gene/p architecture. *Computer physics communications*, 181 (6):1013–1020, 2010.

[33] Bernardo Cockburn and Chi-Wang Shu. The local discontinuous galerkin method for time-dependent convection-diffusion systems. *SIAM Journal on Numerical Analysis*, 35(6):2440–2463, 1998.

[34] Ramon Codina. Comparison of some finite element methods for solving the diffusion-convection-reaction equation. *Computer Methods in Applied Mechanics and Engineering*, 156(1):185–210, 1998.

[35] Ramon Codina, Eugenio Oñate, and Miguel Cervera. The intrinsic time for the streamline upwind/petrov-galerkin formulation using quadratic elements. *Computer Methods in Applied Mechanics and Engineering*, 94(2): 239–262, 1992.

[36] A. Comerford, M.J. Plank, and T. David. Endothelial nitric oxide synthase and calcium production in arterial geometries: an integrated fluid mechanics/cell model. *Journal of biomechanical engineering*, 130(1):011010, 2008.

[37] R Courant. Über partielle differenzengleichungen. In *Atti Congr. Int. Mat. Bologna*, volume 3, pages 83–89. 1928.

[38] Leonardo Dagum and Ramesh Menon. Openmp: an industry standard api for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.

[39] Enzo A Dari, Mariano I Cantero, and Raúl A Feijóo. Computational arterial flow modeling using a parallel navier-stokes solver. In *European Congress on Computational Methods in Applied Science and Engineering, Barcelona, Spain*, 2000.

[40] M. K. Denham and M. A. Patrik. Laminar flow over a downstream-facing in a two-dimensional flow channel. *Transactions of the Institution of Chemical Engineers*, 52:361–367, 1974.

[41] PETSc developers. Petsc documentation: Faq, 2014. URL http://www.mcs.anl.gov/petsc/documentation/faq.html.

[42] E. Dick. Introduction to finite volume methods in computational fluid dynamics. In JohnF. Wendt, editor, *Computational Fluid Dynamics*, pages 275–301. Springer Berlin Heidelberg, 2009. ISBN 978-3-540-85055-7. doi: 10.1007/978-3-540-85056-4_11. URL http://dx.doi.org/10.1007/978-3-540-85056-4_11.

[43] Jean Donea. A taylor–galerkin method for convective transport problems. *International Journal for Numerical Methods in Engineering*, 20(1):101–119, 1984.

[44] Suchuan Dong, Joseph Insley, Nicholas T Karonis, Michael E Papka, Justin Binns, and George Karniadakis. Simulating and visualizing the human arterial system on the teragrid. *Future Generation Computer Systems*, 22 (8):1011–1017, 2006.

[45] J. Douglas, Jr. and T. Dupont. Galerkin methods for parabolic equations. *SIAM Journal on Numerical Analysis*, 7(4):575–626, 1970. doi: 10.1137/0707048. URL http://dx.doi.org/10.1137/0707048.

[46] Jim Douglas, Jr and Thomas F. Russell. Numerical methods for convection-dominated diffusion problems based on combining the method of characteristics with finite element or finite difference procedures. *SIAM Journal on Numerical Analysis*, 19(5):871–885, 1982.

[47] Maksymilian Dryja and Olof Widlund. *An additive variant of the Schwarz alternating method for the case of many subregions.* Ultracomputer Research Laboratory, Univ., Courant Inst. of Mathematical Sciences, Division of Computer Science, 1987.

[48] K. Dumont, J. Vierendeels, R. Kaminsky, G. Nooten van, P. Verdonck, and D. Bluestein. Comparison of the hemodynamic and thrombogenic performance of two bileaflet mechanical heart valves using a cfd/fsi model. *Journal of Biomechanical Engineering*, 129(4):558–565, 2007.

[49] Howard Elman, Victoria E. Howle, John Shadid, Robert Shuttleworth, and Ray Tuminaro. Block preconditioners based on approximate commutators. *SIAM Journal on Scientific Computing*, 27(5):1651–1668, 2006.

[50] Howard Elman, Victoria E. Howle, John Shadid, David Silvester, and Ray Tuminaro. Least squares preconditioners for stabilized discretizations of the navier-stokes equations. *SIAM Journal on Scientific Computing*, 30(1): 290–311, 2007.

[51] Howard Elman, David Silvester, and Andy Wathen. *Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics.* Oxford University Press, 2014.

[52] V. Filardi. Carotid artery stenosis near a bifurcation investigated by fluid dynamic analyses. *The neuroradiology journal*, 26(4):439–453, 2013.

[53] Paul F Fischer and Henry M Tufo. High-performance spectral element algorithms and implementations. *Parallel Computational Fluid Dynamics. Towards Teraflops, Optimization and Novel Formulations*, pages 17–26, 1999.

[54] F. Fontan and E. Baudet. Surgical repair of trisucpid atresia. *Thorax*, 26 (3):240–248, 1971.

[55] Luca Formaggia, Jean-Frédéric Gerbeau, Fabio Nobile, and Alfio Quarteroni. On the coupling of 3d and 1d navier–stokes equations for flow problems in compliant vessels. *Computer Methods in Applied Mechanics and Engineering*, 191(6):561–582, 2001.

[56] Message P Forum. Mpi: A message-passing interface standard. Technical report, Knoxville, TN, USA, 1994.

[57] Morton H. Friedman, Owen J. Deters, Frank F. Mark, C. Brent Bargeron, and Grover M. Hutchins. Arterial geometry affects hemodynamics: a potential risk factor for atherosclerosis. *Atherosclerosis*, 46(2):225–231, 1983.

[58] Augusto Cesar Galeão and Eduardo Gomes Dutra do Carmo. A consistent approximate upwind petrov-galerkin method for convection-dominated problems. *Computer Methods in Applied Mechanics and Engineering*, 68(1): 83–95, 1988.

[59] Thomas George, Anshul Gupta, and Vivek Sarin. An empirical analysis of iterative solver performance for spd systems. *IBM TJ Watson Research Center, Tech. Rep. RC24737*, 2009.

[60] Christophe Geuzaine and Jean-Francois Remacle. Gmsh reference manual. Technical Report 2.8, 2014. URL `http://geuz.org/gmsh/doc/texinfo/gmsh.pdf`.

[61] U. Ghia, K. N. Ghia, and C. T. Shin. High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method. *Journal of Computational Physics*, 48:387–411, 1982.

[62] Trushar Gohil, Robert HP McGregor, Dominik Szczerba, Kathrin Burckhardt, Krishnamurthy Muralidhar, and Gábor Székely. Simulation of oscillatory flow in an aortic bifurcation using fvm and fem: A comparative

study of implementation strategies. *International Journal for Numerical Methods in Fluids*, 66(8):1037–1067, 2011.

[63] H.L. Goldsmith and T. Karino. Mechanically induced thromboemboli in quantitative cardiovascular studies - clinical and research applications of engineering principles. *Hwang NHC, Gross DR, Patel DJ (eds)*, pages 289–351, 1978.

[64] D.M. Hawken, H.R. Tamaddon-Jahromi, P. Townsend, and M.F. Webster. A taylor–galerkin-based algorithm for viscous incompressible flow. *International Journal for Numerical Methods in Fluids*, 10(3):327–351, 1990.

[65] J.C. Heinrich, P.S. Huyakorn, O.C. Zienkiewicz, and A.R. Mitchell. An 'upwind' finite element scheme for two-dimensional convective transport equation. *International Journal for Numerical Methods in Engineering*, 11 (1):131–143, 1977.

[66] Amy Henderson, Jim Ahrens, and Charles Law. The paraview guide. version 4.2. Technical report, Kitware Inc.

[67] Joe Hoffman and Steven Frankel. *Numerical Methods for Engineers and Scientists.* McGraw-Hill, New York, 2001.

[68] D.W. Holdsworth, C.J.D. Norley, R. Frayne, D.A. Steinman, and B.K. Rutt. Characterization of common carotid artery blood-flow waveforms in normal human subjects. *Physiological measurement*, 20(3):219, 1999.

[69] G Horzeaux, R Aubry, M Vázquez, and H Calmet. Large-scale cfd in cerebral hemodynamics: Characterizing arterial flow. In *1st International Conference on Computational & Mathematical Biomedical Engineering (CMBE 2009).*

[70] Thomas J.R. Hughes and A. Brooks. A theoretical framework for petrov-galerkin methods with discontinuous weighting functions: Application to the streamline-upwind procedure. *Finite elements in fluids*, 4:47–65, 1982.

[71] Thomas J.R. Hughes and Alec Brooks. A multidimensional upwind scheme with no crosswind diffusion. *Finite element methods for convection dominated flows, AMD*, 34:19–35, 1979.

[72] Thomas J.R. Hughes, Wing Kam Liu, and Alec Brooks. Finite element analysis of incompressible viscous flows by the penalty function formulation. *Journal of Computational Physics*, 30(1):1–60, 1979.

[73] Thomas J.R. Hughes, Leopoldo P. Franca, and Gregory M. Hulbert. A new finite element formulation for computational fluid dynamics: Viii. the

galerkin/least-squares method for advective-diffusive equations. *Computer Methods in Applied Mechanics and Engineering*, 73(2):173–189, 1989.

[74] Sinjae Hyun, Clement Kleinstreuer, and Joseph P. Archie Jr. Computational analysis of effects of external carotid artery flow and occlusion on adverse carotid bifurcation hemodynamics. *Journal of vascular surgery*, 37 (6):1248–1254, 2003.

[75] Amtec Engineering Inc. Tecplot user's manual. version 10. Technical report, 2003. URL `http://www.tecplot.com/`.

[76] Kitware Inc. Vtk file formats for vtk version 4.2 - an excerpt from the vtk user's guide, 2014. URL `http://www.vtk.org/VTK/img/file-formats.pdf`.

[77] I. C. F. Ipsen and C. D. Meyer. The idea behind krylov methods. In *American Mathematical Monthly.*

[78] Karin John and Abdul I. Barakat. Modulation of atp/adp concentration at the endothelial surface by shear stress: effect of flow-induced atp release. *Annals of biomedical engineering*, 29(9):740–751, 2001.

[79] G. Karniadakis and S. Sherwin. *Spectral/hp Element Methods for Computational Fluid Dynamics: Second Edition.* Numerical Mathematics and Scientific Computation. OUP Oxford, 2013. ISBN 9780199671366. URL `http://books.google.co.uk/books?id=7lepAgAAQBAJ`.

[80] G. Karypis and V. Kumar. Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, version 4.0. Technical report, University of Minnesota, 1998.

[81] George Karypis and Kirk Schloegel. Parallel graph partitioning and sparse matrix ordering library. Technical report, University of Minnesota, 2013.

[82] David Kay, Daniel Loghin, and Andrew Wathen. A preconditioner for the steady-state navier–stokes equations. *SIAM Journal on Scientific Computing*, 24(1):237–256, 2002.

[83] D.W. Kelly, S. Nakazawa, O.C. Zienkiewicz, and J.C. Heinrich. A note on upwinding and anisotropic balancing dissipation in finite element approximations to convective diffusion problems. *International journal for numerical methods in engineering*, 15(11):1705–1711, 1980.

[84] Chang Sung Kim, Cetin Kiris, Dochan Kwak, and Tim David. Numerical models of human circulatory system under altered gravity: brain circulation. *Paper AIAA*, 1092, 2004.

[85] Hyun Jin Kim, C. A. Figueroa, T.J.R. Hughes, K.E. Jansen, and C.A. Taylor. Augmented lagrangian method for constraining the shape of velocity profiles at outlet boundaries for three-dimensional finite element simulations of blood flow. *Computer Methods in Applied Mechanics and Engineering*, 198(45):3551–3566, 2009.

[86] David N. Ku, Don P. Giddens, Christopher K. Zarins, and Seymour Glagov. Pulsatile flow and atherosclerosis in the human carotid bifurcation. positive correlation between plaque location and low oscillating shear stress. *Arteriosclerosis, Thrombosis, and Vascular Biology*, 5(3):293–302, 1985.

[87] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to parallel computing: design and analysis of algorithms.* Benjamin/Cummings Publishing Company Redwood City, CA, 1994.

[88] Adamos Kyriakou, Esra Neufeld, Dominik Szczerba, Wolfgang Kainz, Roger Luechinger, Sebastian Kozerke, Robert McGregor, and Niels Kuster. Patient-specific simulations and measurements of the magneto-hemodynamic effect in human primary vessels. *Physiological measurement*, 33(2):117, 2012.

[89] Peter Lax and Burton Wendroff. Systems of conservation laws. *Communications on Pure and Applied mathematics*, 13(2):217–237, 1960.

[90] F. Thomson Leighton. *Introduction to parallel algorithms and architectures.* Morgan Kaufmann San Francisco, 1992.

[91] Wai-Hung Liu and Andrew H. Sherman. Comparative analysis of the cuthill-mckee and the reverse cuthill-mckee ordering algorithms for sparse matrices. *SIAM Journal on Numerical Analysis*, 13(2):198–213, 1976.

[92] Rainald Löhner, K. Morgan, and Olgierd C. Zienkiewicz. The solution of non-linear hyperbolic equation systems by the finite element method. *International Journal for Numerical Methods in Fluids*, 4(11):1043–1063, 1984.

[93] Robert W MacCormack. Numerical solution of the interaction of a shock wave with a laminar boundary layer. In *Proceedings of the second international conference on numerical methods in fluid dynamics*, pages 151–163. Springer, 1971.

[94] Andre Macdonald, Anne G. Osborn, and Jeff Ross. Diagnostic and surgical imaging anatomy: Brain, head and neck, spine, 2006.

[95] User mailing list and newsgroup. Vim documentation: vim faq, 2014. URL `http://vimdoc.sourceforge.net/htmldoc/vimfaq.html`.

[96] Emilie Marchandise, Paolo Crosetto, Christophe Geuzaine, Jean-François Remacle, and Emilie Sauvage. Quality open source mesh generation for cardiovascular flow simulations. In *Modeling of Physiological Flows*, pages 395–414. Springer, 2012.

[97] Ian Marshall, Panorea Papathanasopoulou, and Karolina Wartolowska. Carotid flow rates and flow division at the bifurcation in healthy volunteers. *Physiological measurement*, 25(3):691, 2004.

[98] Catherine N. Marti, Mihai Gheorghiade, Andreas P. Kalogeropoulos, Vasiliki V. Georgiopoulou, Arshed A. Quyyumi, and Javed Butler. Endothelial dysfunction, arterial stiffness, and heart failure. *Journal of the American College of Cardiology*, 60(16):1455–1469, 2012.

[99] C. Matter, E. Nagel, M. Stuber, P. Boesiger, and O.M. Hess. Assessment of systolic and diastolic lv function by mr myocardial tagging. *Basic research in cardiology*, 91(1):23–28, 1996.

[100] Mihai Mihaescu, Shanmugam Murugappan, Maninder Kalra, Sid Khosla, and Ephraim Gutmark. Large eddy simulation and reynolds-averaged navier–stokes modeling of flow in a realistic pharyngeal airway model: An investigation of obstructive sleep apnea. *Journal of biomechanics*, 41(10): 2279–2288, 2008.

[101] Jaques S. Milner, Jennifer A. Moore, Brian K. Rutt, and David A. Steinman. Hemodynamics of human carotid artery bifurcations: computational studies with models reconstructed from magnetic resonance imaging of normal subjects. *Journal of Vascular Surgery*, 28(1):143–156, 1998.

[102] JJ Monaghan. Particle methods for hydrodynamics. *Computer Physics Reports*, 3(2):71–124, 1985.

[103] MlNEO Motomiya and TAKESHI Karino. Flow patterns in the human carotid artery bifurcation. *Stroke*, 15(1):50–56, 1984.

[104] Heinz Mühlenbein, Martina Gorges-Schleuter, and Ottmar Krämer. Evolution algorithms in combinatorial optimization. *Parallel Computing*, 7(1): 65–85, 1988.

[105] Kazuhiro Muramatsu, Shun Doi, Takumi Washio, and Toshiyuki Nakata. Cenju-3 parallel computer and its application to cfd. In *Parallel Architectures, Algorithms and Networks, 1994.(ISPAN), International Symposium on*, pages 318–325. IEEE, 1994.

[106] Fernando Mut, Susan Wright, Christopher Putman, Giorgio Ascoli, and Juan Cebral. Image-based modeling of the hemodynamics in cerebral arterial trees. In *SPIE Medical Imaging*, pages 72620I–72620I. International Society for Optics and Photonics, 2009.

[107] J.P. Mynard and P. Nithiarasu. A 1d arterial blood flow model incorporating ventricular pressure, aortic valve and regional coronary flow using the locally conservative galerkin (lcg) method. *Communications in numerical methods in engineering*, 24:367–417, 2008. doi: 10.1002/cnm.1117.

[108] Jagat Narula, Mani A. Vannan, and Anthony N. DeMaria. Of that waltz in my heart. *Journal of the American College of Cardiology*, 49(8):917–920, 2007.

[109] R.M. Nerem. Arterial fluid dynamics and interactions with the vessel walls. *Structure and Function of the Circulation*, 2:719–835, 1981.

[110] R.M. Nerem and J.F. Cornhill. The role of fluid mechanics in atherogenesis. *Journal of biomechanical engineering*, 102(3):181–189, 1980.

[111] Kien T. Nguyen, Christopher D. Clark, Thomas J. Chancellor, and Dimitrios V. Papavassiliou. Carotid geometry effects on blood flow and on risk for vascular disease. *Journal of biomechanics*, 41(1):11–19, 2008.

[112] P. Nithiarasu. An efficient artificial compressibility (ac) scheme based on the characteristic based split (cbs) method for incompressible flows. *International Journal for Numerical Methods in Engineering*, 56(13):1815–1845, 2003.

[113] P. Nithiarasu and C.-B. Liu. An artificial compressibility based characteristic based split (cbs) scheme for steady and unsteady turbulent incompressible flows. *Computer methods in applied mechanics and engineering*, 195 (23):2961–2982, 2006.

[114] Fabio Nobile, Matteo Pozzoli, and Christian Vergara. Time accurate partitioned algorithms for the solution of fluid–structure interaction problems in haemodynamics. *Computers & Fluids*, 86:470–482, 2013.

[115] Noncommunicable Diseases and Mental Health. Global status report on noncommunicable diseases. Technical Report 9789241564229, World Health Organization, 2010. URL `http://www.who.int/nmh/publications/ncd_report_full_en.pdf`.

[116] John Tinsley Oden and Noboru Kikuchi. Theory of variational inequalities with applications to problems of flow through porous media. *International Journal of Engineering Science*, 18(10):1173–1284, 1980.

[117] Eugenio Oñate. Derivation of stabilized equations for numerical solution of advective-diffusive transport and fluid flow problems. *Computer Methods in Applied Mechanics and Engineering*, 151(1):233–265, 1998.

[118] Eugenio Onate. A stabilized finite element method for incompressible viscous flows using a finite increment calculus formulation. *Computer Methods in Applied Mechanics and Engineering*, 182(3):355–370, 2000.

[119] Eugenio Oñate, Sergio R. Idelsohn, and Carlos A. Felippa. Consistent pressure laplacian stabilization for incompressible continua via higher-order finite calculus. *International Journal for Numerical Methods in Engineering*, 87(1-5):171–195, 2011.

[120] Panorea Papathanasopoulou, Shunzhi Zhao, Uwe Köhler, Malcolm B. Robertson, Quan Long, Peter Hoskins, X. Yun Xu, and Ian Marshall. Mri measurement of time-resolved wall shear stress vectors in a carotid bifurcation model, and comparison with cfd predictions. *Journal of Magnetic Resonance Imaging*, 17(2):153–162, 2003.

[121] K Perktold and R Peter. Numerical 3d-simulation of pulsatile wall shear stress in an arterial t-bifurcation model. *Journal of biomedical engineering*, 12(1):2–12, 1990.

[122] Ingemar Persson. Performance analysis of a cfd-code on the ibm-sp2. Technical report, Citeseer, 1995.

[123] Olivier Pironneau. On the transport-diffusion algorithm and its applications to the navier-stokes equations. *Numerische Mathematik*, 38(3):309–332, 1982.

[124] Michael J. Plank, David J.N. Wall, and Tim David. Atherosclerosis and calcium signalling in endothelial cells. *Progress in biophysics and molecular biology*, 91(3):287–313, 2006.

[125] M.J. Plank, A. Comerford, T. David, and D.J.N Wall. Concentration of blood-borne agonists at the endothelium. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science*, 462(2066):671–688, 2006.

[126] M.J Plank, D.J.N Wall, and T. David. The role of endothelial calcium and nitric oxide in the localisation of atherosclerosis. *Mathematical biosciences*, 207(1):26–39, 2007.

[127] P. Hendrik Pretorius, Michael A. King, Benjamin M.W. Tsui, Karen J. LaCroix, and Weishi Xia. A mathematical model of motion of the heart for use in generating source and attenuation maps for simulating emission imaging. *Medical Physics*, 26(11):2323–2332, 1999.

[128] Abtin Rahimian, Ilya Lashuk, Shravan Veerapaneni, Aparna Chandramowlishwaran, Dhairya Malhotra, Logan Moon, Rahul Sampath, Aashay Shringarpure, Jeffrey Vetter, Richard Vuduc, et al. Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE Computer Society, 2010.

[129] V. Nageshwara Rao and Vipin Kumar. Superlinear speedup in parallel state-space search. In *Foundations of Software Technology and Theoretical Computer Science*, pages 161–174. Springer, 1988.

[130] L.F. Richardson. The approximate arithematical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam. *Phil. Trans. R. Soc. London Ser. A*, 210:307–357, 1910.

[131] CCM Rindt, AA Van Steenhoven, JD Janssen, RS Reneman, and A Segal. A numerical analysis of steady flow in a three-dimensional model of the carotid artery bifurcation. *Journal of biomechanics*, 23(5):461–473, 1990.

[132] David P. Rodgers. Improvements in multiprocessor system design. In *ACM SIGARCH Computer Architecture News*, volume 13, pages 225–231. IEEE Computer Society Press, 1985.

[133] Youcef Saad and Martin H. Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7(3):856–869, July 1986. ISSN 0196-5204. doi: 10.1137/0907058. URL `http://dx.doi.org/10.1137/0907058`.

[134] Igor Sazonov, Si Yong Yeo, Rhodri L. T. Bevan, Xianghua Xie, Raoul van Loon, and Perumal Nithiarasu. Modelling pipeline for subject-specific arterial blood flow - a review. *International Journal for Numerical Methods in Biomedical Engineering*, 27(12):1868–1910, 2011. ISSN 2040-7947. doi: 10.1002/cnm.1446.

[135] Partho P. Sengupta, Jagat Narula, and Y. Chandrashekhar. The dynamic vortex of a beating heart: Wring out the old and ring in the new!âĹŮ. *Journal of the American College of Cardiology*, 64(16):1722–1724, 2014.

[136] Farzin Shakib and Thomas J.R. Hughes. A new finite element formulation for computational fluid dynamics: Ix. fourier analysis of space-time galerkin/least-squares algorithms. *Computer Methods in Applied Mechanics and Engineering*, 87(1):35–58, 1991.

[137] JIAN Shen, FRANCIS W. Luscinskas, ANDREW Connolly, C. Forbes Dewey Jr, and M.A. Gimbrone Jr. Fluid shear stress modulates cytosolic free calcium in vascular endothelial cells. *Am J Physiol*, 262(2 Pt 1): C384–90, 1992.

[138] Tony W.H. Sheu, S.K. Wang, and S.F. Tsai. Finite element analysis of particle motion in steady inspiratory airflow. *Computer methods in applied mechanics and engineering*, 191(25):2681–2698, 2002.

[139] Yuan Shi. Program scalability analysis. In *International Conference on Distributed and Parallel Processing, Geogetown University, Washington DC*, 1997.

[140] David Silvester, Howard Elman, David Kay, and Andrew Wathen. Efficient preconditioning of the linearized navier–stokes equations for incompressible flow. *Journal of Computational and Applied Mathematics*, 128(1):261–279, 2001.

[141] Ewald Speckenmeyer, Burkhard Monien, and Oliver Vornberger. Superlinear speedup for parallel backtracking. In *Supercomputing*, pages 985–993. Springer, 1988.

[142] J.S. Stroud, S.A. Berger, and D. Saloner. Numerical analysis of flow through a severely stenotic carotid artery bifurcation. *Journal of biomechanical engineering*, 124(1):9–20, 2002.

[143] Kumar K. Tamma and Raju R. Namburu. A new finite element based lax-wendroff/taylor-galerkin methodology for computational dynamics. *Computer methods in applied mechanics and engineering*, 71(2):137–150, 1988.

[144] Charles A. Taylor, Thomas J.R. Hughes, and Christopher K. Zarins. Finite element modeling of blood flow in arteries. *Computer methods in applied mechanics and engineering*, 158(1):155–196, 1998.

[145] Tayfun E. Tezduyar. Stabilized finite element formulations for incompressible flow computations. *Advances in applied mechanics*, 28:1–44, 1991.

[146] Tayfun E. Tezduyar, Sanjay Mittal, S.E. Ray, and R. Shih. Incompressible flow computations with stabilized bilinear and linear equal-order-interpolation velocity-pressure elements. *Computer Methods in Applied Mechanics and Engineering*, 95(2):221–242, 1992.

[147] Tayfun E. Tezduyar, Matthew Schwaab, and Sunil Sathe. Sequentially-coupled arterial fluid–structure interaction (scafsi) technique. *Computer Methods in Applied Mechanics and Engineering*, 198(45):3524–3533, 2009.

[148] Ryo Torii, Marie Oshima, Toshio Kobayashi, Kiyoshi Takagi, and Tayfun E. Tezduyar. Fluid–structure interaction modeling of blood flow and cerebral aneurysm: significance of artery and aneurysm shapes. *Computer Methods in Applied Mechanics and Engineering*, 198(45):3613–3621, 2009.

[149] Q-K Tran and H. Watanabe. Calcium signalling in the endothelium. In *The Vascular Endothelium I*, pages 145–187. Springer, 2006.

[150] Sheffield Teaching Hospitals NHS Foundation Trust. Carotid artery disease, 2014. URL `http://www.sth.nhs.uk/clientfiles/File/pd4727_CarotidArteryDisease.pdf`.

[151] Henry M Tufo and Paul F Fischer. Terascale spectral element algorithms and implementations. In *Proceedings of the 1999 ACM/IEEE conference on Supercomputing*, page 68. ACM, 1999.

[152] S.A. Urquiza, P.J. Blanco, M.J. Vénere, and R.A. Feijóo. Multidimensional modelling for the carotid artery blood flow. *Computer Methods in Applied Mechanics and Engineering*, 195(33):4002–4017, 2006.

[153] Irene E. Vignon-Clementel, C. Alberto Figueroa, Kenneth E. Jansen, and Charles A. Taylor. Outflow boundary conditions for three-dimensional finite element modeling of blood flow and pressure in arteries. *Computer methods in applied mechanics and engineering*, 195(29):3776–3796, 2006.

[154] C. Vuik, G. Segal, et al. A comparison of preconditioners for incompressible navier–stokes solvers. *International Journal for Numerical methods in fluids*, 57(12):1731–1751, 2008.

[155] T. John Wilson, Wei Wang, H. Augustus Hellerstedt, C. David Zawieja, and E. James Moore. Confocal image-based computational modeling of nitric oxide transport in a rat mesenteric lymphatic vessel. *Journal of biomechanical engineering*, 135. doi: 10.1115/1.4023986.

[156] C.-C. Yu and Juan Carlos Heinrich. Petrov-galerkin methods for the time-dependent convective transport equation. *International journal for numerical methods in engineering*, 23(5):883–901, 1986.

[157] C.-C. Yu and Juan Carlos Heinrich. Petrov-galerkin method for multidimensional, time-dependent, convective-diffusion equations. *International Journal for numerical methods in engineering*, 24(11):2201–2215, 1987.

[158] Fuzhen Zhang. *The Schur complement and its applications*, volume 4. Springer, 2006.

[159] O. C. Zienkiewicz and Y. K. Cheung. The finite element method for analysis of elastic isotropic and orthotropic slabs. In *ICE Proceedings*, volume 28, pages 471–488. Thomas Telford, 1964.

[160] O. C. Zienkiewicz and R. Codina. Search for a general fluid mechanics algorithm. *Frontiers of Computational Fluid Dynamics*, pages 101–113, 1995.

[161] O. C. Zienkiewicz and R. Codina. A general algorithm for compressible and incompressible flow, part i. the split characteristic based scheme. *Int. J. Num. Meth. Fluids*, 20:869–885, 1995.

[162] O.C. Zienkiewicz and R.L. Taylor. *The finite element method. Volume 3. Fluid dynamics.* Butterworth-Heinemann, 2000.