ABSTRACT

Title of dissertation: BLOCK PRECONDITIONERS FOR THE NAVIER-STOKES EQUATIONS

Robert Shuttleworth
Doctor of Philosophy, 2007

Dissertation directed by: Professor Howard Elman
Computer Science Department

In recent years, considerable effort has been placed on developing efficient and robust solution algorithms for the incompressible Navier–Stokes equations based on preconditioned Krylov methods. These include physics-based methods, such as SIMPLE, and purely algebraic preconditioners based on the approximation of the Schur complement. All these techniques can be represented as approximate block factorization (ABF) type preconditioners. The goal is to decompose the application of the preconditioner into simplified sub-systems in which scalable multi-level type solvers can be applied. In this dissertation we develop a taxonomy of these ideas based on an adaptation of a generalized approximate factorization of the Navier-Stokes system first presented in [45]. This taxonomy illuminates the similarities and differences among these preconditioners and the central role played by efficient approximation of certain Schur complement operators. We then present a parallel computational study that examines the performance of these methods and compares them to an additive Schwarz domain decomposition (DD) algorithm. Results are presented for two and three-dimensional steady state problems for enclosed domains

and inflow/outflow systems on both structured and unstructured meshes. The numerical experiments are performed using MPSalsa, a stabilized finite element code. We have also tested the utility of these methods in a more realistic fluid setting by solving an optimization problem related to the shape and topology of a microfluidic mixing device. This flow is modeled by Induced Charged Electro-osmosis (ICEO) described in [54]. The numerical results are performed using Sundance, a tool for the development of finite-element solutions of partial differential equations.

Block Preconditioning the Navier-Stokes Equations

by

Robert Shuttleworth

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2007

Advisory Committee:
Professor Howard C. Elman, Chair/Advisor
Professor Bill Dorland
Professor Ramani Duraiswami
Professor Jian-Guo Liu
Professor James Baeder, Dean's Representative

Acknowledgments

I owe my gratitude to several people who have made this dissertation possible and because of whom my graduate experience has been one that I will cherish forever.

First and foremost I'd like to thank my advisor, Professor Howard Elman for giving me an invaluable opportunity to work on extremely challenging and interdisciplinary projects over the past few years. He has always been available for help, guidance, and advice. It has been a pleasure to work with and learn from such an extraordinary individual. I thank him for taking the time to train me to think critically, write succinctly, and to communicate effectively.

I would also like to thank my co-collaborators, Vicki Howle, John Shadid, and Ray Tuminaro, for their extraordinary help in keeping me sane by providing a summer outlet to the friendly confines of Sandia National Laboratory in Livermore, CA. Without their passion, extraordinary theoretical ideas and computational expertise, this thesis and my development as a computational scientist would have been a distant dream. I would also like to thank Professor Bruce Golden for the outlet his RIT has given me. Thanks are due to Professors Bill Dorland, Ramani Duraiswami, Jian-Guo Liu, and James Baeder for agreeing to serve on my thesis committee and for using their invaluable time reviewing this manuscript.

My colleagues at CSCAMM and in the Scientific Computing track of the AMSC program have enriched my graduate life in many ways and deserve a special mention. I would also like to thank the people who run the AMSC program, namely

C. David Levermore, James Yorke, Alverda McCoy, Patty Woodwell, Celeste Regalado, and Liz Wincek, for assisting me in navigating the necessary Graduate School hoops.

I want to thank Katie Meloney and Brian Alford for helping me decide to come to Maryland. Their friendship, support, and camaraderie over the past 5 years has been irreplaceable. A special thank you goes to Alyssa Finamore for all of the emotional support and guidance she has given me over the past few years. I will never forget our weekly Sunday brunches and all of our holiday drives back to Ohio. I would also like to thank Weigang Zhong, Danny Dunlavy, Darran Furnival, Chris Danforth, Aaron Lott, Carter Price, and Brandy Rapatski for their support, mentorship, computational assistance, and presentation suggestions.

My housemates at my place of residence have been an important factor in my success. I'd like to express my gratitude to my current and past housemates including Brian Alford, Dave Bourne, Avi Dalal, Nick Long, Miguel Pauletti, Dave Shoup, and Christian Zorn for their enduring friendship and support. I will never forget the times we have had together.

I owe my deepest thanks to my family - my mother, father, and sisters who have always stood by me and guided me through my career, and have pulled me through against impossible odds at times. Words cannot express the gratitude I owe them. I also want to thank my Uncle Ben and Uncle Ray who have always acted as an inspiration to me and motivated me to apply, attend, and finish graduate school.

I would like to acknowledge financial support from the Department of Energy, Mathematics Department, Computer Science Department, Center for Scien-

tific Computing and Applied Math Modeling, Sandia's ASCI program for all the projects discussed above. It is impossible to remember all, and I apologize to those I've inadvertently left out.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction and Background

1.1 Introduction

Current leading-edge engineering and scientific flow simulations often entail complex two and three-dimensional geometries with high resolution unstructured meshes to capture all the relevant length scales of interest. After suitable discretization and linearization of the governing partial differential equations, these simulations can produce large linear systems of equations with on the order 10^5 to 10^8 unknowns. This leads to a central challenge in computational science and engineering today which is efficiently and robustly solving large sparse linear systems that arise from linearization and discretization of the governing equations.

The two main techniques for solving large matrix problems are direct and iterative methods. Direct methods, based upon the factorization of the coefficient matrix into easily invertible matrices, are widely used in many industrial codes. These solvers can be very robust especially for two dimensional problems and are commonly used in structural analysis, computational fluid dynamics, and in the design of semiconductors. However, the number of operation counts and memory requirements needed by direct methods make them prohibitive for increasing problem size. Iterative methods require far less memory and fewer operations than direct methods. So for large three dimensional, multiphysics simulations, iterative

methods with preconditioning are the only option available to efficiently solve these problems.

Generally, iterative techniques do not produce an exact answer after a certain number of steps, but rather reduce the residual by a certain amount after each step. The iteration stops when the error is less than a user-supplied value. This is in contrast to direct methods, which in the absence of roundoff error produce an exact answer after a finite number of steps. Historically, iterative methods have been popular in the nuclear power and oil industries. The field of iterative methods comprises a large variety of techniques including basic techniques, such as Jacobi, Gauss-Seidel, SOR iterations, to Krylov subspace methods, and multilevel techniques. The focus of this work is to increase the reliability and performance of Krylov subspace methods by developing efficient and scalable preconditioning strategies for the incompressible Navier-Stokes equations.

Preconditioning refers to the process of transforming a linear system, $Ax = b$, into another $\hat{A}x = \hat{b}$, that has better properties with respect to iterative solution strategies. The matrix that transforms A to $\hat{A}$ is called a *preconditioner*. In other words, if Q is a matrix that approximates A, then

$$Q^{-1}Ax = Q^{-1}b \tag{1.1}$$

has the same solution as the original system, $Ax = b$, but should be easier to solve than $Ax = b$. In (1.1), the matrix Q acts as a left oriented preconditioner. We can also precondition on the right, by

$$AQ^{-1}y = b \tag{1.2}$$

where $x = Q^{-1}y$. In practice, a good preconditioner Q is chosen to be easy to construct and apply, while the preconditioned system should be easy to solve. In many cases, the preconditioning matrix is designed to improve the spectral properties of the original matrix.

The two major types of preconditioners are algebraic and physics-based. Algebraic preconditioners are designed to be used with any matrix and include incomplete factorization (ILU), sparse approximate inverses, and, to some extent, domain decomposition, and multilevel multigrid techniques. A further discussion of ILU methods and other algebraic preconditioners is found in Section 1.4.1. Physics-based preconditioners use the underlying physical problem as motivation for the derivation of the methodology. By using information about the underlying physical model one can develop more robust preconditioners. We briefly discuss such methods along with domain decomposition and multilevel multigrid techniques in Section 1.4.2. In Section 1.2, we give some background on our goal which is preconditioning the incompressible Navier-Stokes equations. In Section 1.3, we discuss two iterative solvers for solving sparse linear systems of equations, mainly CG and GMRES, and then tell why preconditioning is important to make these solvers function efficiently.

1.2 Background

The modeling of incompressible flows are useful for understanding diverse phenomena such as combustion, pollution, chemical reactions, and manufacturing processes. We consider solution methods for the incompressible Navier-Stokes equations

where the equations below represent conservation of momentum and mass, and the constitutive equation for the Newtonian stress tensor,

$$\begin{aligned} \textbf{Momentum:} \qquad \rho(\mathbf{u}\cdot\nabla)\mathbf{u} &= \nabla\cdot\mathbf{T}+\rho\mathbf{g} \\ \textbf{Mass:} \qquad \nabla\cdot\mathbf{u} &= 0 \\ \textbf{Stress Tensor:} \qquad \mathbf{T} &= -P\mathbf{I}+\mu(\nabla\mathbf{u}+\nabla\mathbf{u}^{\mathbf{T}}) \end{aligned} \tag{1.3}$$

in $\Omega \subset \mathbb{R}^d (d = 2 \text{ or } 3)$. Here the velocity, $\mathbf{u}$, satisfies suitable boundary conditions on $\partial\Omega$, P represents the hydrodynamic pressure, ρ the density, μ the dynamic viscosity, and $\mathbf{g}$ the body forces.

Our focus is on improving solution algorithms for the systems of equations that arise after discretization and linearization of the system (1.3) by physics based preconditioning. A nonlinear iteration based on an inexact Newton-Krylov method is used to solve this problem. If the nonlinear problem to be solved is written as $G(x) = 0$, where $G : \mathbb{R}^n \to \mathbb{R}^n$, then at the k^{th} step of Newton's method, the solution of the linear *Newton equation*

$$J(x_k)s_k = -g(x_k) \tag{1.4}$$

is required, where x_k is the current solution and $J(x_k)$ denotes the Jacobian matrix of G at x_k. Once the Newton update, s_k, is determined, the current approximation, x_k, is updated via

$$x_{k+1} = x_k + s_k.$$

Newton-Krylov methods [16] relax the requirement of computing an exact solution to (1.4) by using a Krylov subspace method, such as GMRES, to obtain an iterate

s_k that satisfies the *inexact Newton condition*,

$$\|g(x_k) + J(x_k)s_k\| \leq \eta_k \|g(x_k)\|, \tag{1.5}$$

where, $\eta_k \in [0, 1]$, is a tolerance. When $\eta_k = 0$, this is an exact Newton method. For a discussion of the merits of different choices of η_k, see [16]. In the computational results of Chapter 4, η_k is chosen to be a constant and our attention is focused on preconditioning methods for use with GMRES solving for the Newton update.

For the discrete Navier-Stokes equations, the Jacobian system at the kth step that arises from Newton's method is

$$\begin{pmatrix} F & B^T \\ \hat{B} & -C \end{pmatrix} \begin{pmatrix} \mathbf{\Delta u_k} \\ \Delta p_k \end{pmatrix} = \begin{pmatrix} \mathbf{g_u^k} \\ g_p^k \end{pmatrix}, \tag{1.6}$$

where F is a convection-diffusion-like operator, B^T is the gradient operator, $\hat{B}$ is the divergence operator that for some higher-order stabilized formulations can include a contribution from non-zero higher-order derivative operators in the stabilized formulation [10], and C is the operator that stabilizes the finite element discretization. The right hand side vector, $(\mathbf{g_u}, g_p)^T$, contains respectively the nonlinear residual for the momentum and continuity equations. This Newton procedure starts with some initial iterate $\mathbf{u}_0$ for the velocities, p_0 for the pressure; then updates for velocities and pressures are computed by solving the Newton equations (1.6). Problems with a saddle point structure of this type are also found in electrical networks, structural networks, optimal control problems, and computer graphics. This system of equations is indefinite and nonsymmetric, both qualities that make it a challenging and difficult problem to solve.

The cost for solving this system can be very high, and can be one of the most time consuming components for a given simulation. One way to reduce this computational time is by coupling iterative solvers with preconditioners to solve (1.6). With the rise of mixed finite element methods and constrained optimization problems numerous techniques have been developed to efficiently solve saddle point linear systems (1.6) and the Navier-Stokes equations (1.3), examples of which can be found in ([12, 17, 41]). Current solution techniques for the Navier-Stokes equations include fractional step methods, fully decoupled methods, and fully coupled methods. Fractional step methods, such as pressure projection or operator splitting ([12]), and fully decoupled techniques, such as SIMPLE (Semi-Implicit Method for Pressure Linked Equations) and SIMPLER ([41], [43]), do not preserve the coupling of physics, which can lead to slow convergence.

The research in this dissertation explores the pressure convection-diffusion methods which define solution techniques by coupling together the momentum and mass equations. This leads to algorithms with good convergence properties because the methods are insensitive to mesh size and CFL number ([17, 19, 21]). For stationary problems, there is a slight dependence on Reynolds number, whereas for transient problems there is no Reynolds number dependence.

1.3 Iterative solvers for linear systems

The Conjugate Gradient (CG) algorithm and the Generalised Minimum Residual (GMRES) method are two important iterative solvers for linear systems of the

form $Ax = b$, where A is a square n by n matrix, b is a column vector, and x is the sought solution. Detailed discussion on those methods along with other iterative methods can be found in [28, 46].

CG is one of the best known algorithms for solving large sparse linear Hermitian positive definite systems. It was developed in 1952 independently by Lanczos [36] and Hestenes and Stiefel [32]. Initially, this method was seen as an exact method because it converged to a solution in no more than n steps, where n is the problem size. In practice, CG converges to a solution in far fewer than n steps.

If x^* is the exact solution of $Ax = b$ and x_k is the k^{th} iterate of some iterative solution technique, then the error at step k is

$$e_k = x^* - x_k$$

with the residual at step k

$$r_k = b - Ax_k.$$

The CG algorithm minimises the A-norm of the error $\|e_k\|_A = \sqrt{e_k^H A e_k}$ for positive definite A over the affine space $x_0 + \mathcal{K}_k(A, r_0)$ where the k-dimensional Krylov subspace $\mathcal{K}_k(A, r_0)$ is given by

$$\mathcal{K}_k(A, r_0) = \text{span}\{r_0, Ar_0, \ldots, A^{k-1}r_0\}.$$

The CG algorithm is very attractive because the work required per iteration is very modest. It includes two inner products, three vector updates, and a sparse matrix-vector product. The preconditioned version of CG has the same costs, plus an additional (cheap) solve to apply the preconditioner. The error for this method can

be bounded by

$$\|e_k\|_A \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k \|e_0\|_A,$$

where $\kappa = \frac{\lambda_{max}}{\lambda_{min}}$ with λ_{max} the maximum eigenvalue and λ_{min} the minimum eigenvalue of the coefficient matrix [46]. The rate of convergence for the CG method depends on the distribution of the eigenvalues of A. The goal of preconditioning is to improve the convergence rate by reducing the condition number and/or number of distinct eigenvalue clusters of the system matrix. More detailed theory on the CG algorithm can be found in [28, 46].

When A is symmetric positive definite CG is an attractive solver because at each iteration it minimizes the A-norm of the error and the operations required are few and independent of the iteration. For nonsymmetric systems, there is not a method that has both properties [46]. The Generalised Minimum Residual (GMRES) method proposed in [47] by Saad and Schultz is our choice for solving nonsymmetric systems because it minimizes the Euclidean-norm of the residual r_k at each iteration. The main drawback to this method, when compared to CG, is that the work and storage per iteration grows linearly with the size of the problem. At the k^{th} iterate, GMRES chooses $x_k \in x_0 + \mathcal{K}_k(A, r_0)$ to minimize the least squares problem

$$\min_{x_k \in x_0 + \mathcal{K}_k(A,r_0)} \|b - Ax_k\|.$$

GMRES converges to the exact solution in at most n steps, where n is the size of the system. We can improve its performance by developing right-oriented preconditioners (1.2). We choose the right-oriented version so that the norm being

minimized does not depend on the preconditioner. For more details on GMRES we refer to [28, 46].

1.4 Preconditioning Krylov Subspace Methods

We further explore the concept of preconditioning by decomposing the coefficient matrix, A, into a splitting

$$A = Q - R$$

where Q is a splitting matrix and R is the error matrix. A splitting of this type gives rise to a stationary iterative method, which computes the approximation to the solution using the iteration

$$x_k = Q^{-1}(Rx_{k-1} + b).$$

This stationary iteration can be rewritten as a specific example of a preconditioned Krylov subspace method [17]. Furthermore, these splitting operations can be used in conjunction with a Krylov subspace method to accelerate convergence. If we decompose A as $A = D - L - U$, where D is the matrix containing the diagonal of A, L the matrix with the lower triangular factor, and U the matrix with the upper triangular factor. If we choose $Q = D$ and $R = L + U$, this iteration is known as Jacobi's method. If $Q = D - L$ and $R = U$, this is the Gauss-Seidel method.

1.4.1 Algebraic Preconditioners

Purely algebraic preconditioning strategies are derived directly from the coefficient matrix, A, and do not require any mesh or problem characteristics to be

used. These methods are more general-purpose than physics-based preconditioning strategies, but are not optimal for a specific problem instance. However, these methods tend not to be as effective as physics-based strategies. In this section, a few general preconditioners are discussed, including diagonal scaling, Gauss-Seidel, and incomplete factorization. The list discussed here is by no means exhaustive, but is presented because these techniques are the most straightforward to explain and are used in many industrial settings. Furthermore, the concepts behind these preconditioners are sometimes used as building blocks for physics-based preconditioning strategies, which we discuss in Section 1.4.2.

In Jacobi preconditioning, the preconditioner is chosen to be the diagonal of A. While this preconditioner choice is very cheap to construct, it normally only reduces the number of iterations by a small amount when compared to more sophisticated techniques. This method performs well when the matrix A is diagonally dominant, because the diagonal contains a lot of information about the matrix (and therefore its inverse) [1]. The Gauss-Seidel strategy is effective, but it is very dependent on the ordering of unknowns in the system. If the matrix is poorly ordered, then this strategy converges slowly. A further discussion of this issue can be found in [17].

Solving sparse systems with direct methods, such as Gaussian elimination, can cause a high amount of fill-in, therefore causing this method to be expensive in terms of storage and CPU time. The LU decomposition of a matrix can be used as an effective preconditioner by ignoring any fill-in that occurs within a certain

[1] As a historical note, Carl Jacobi found that using diagonal scaling reduced the computational time when he was determining the stability of the solar system in the early 1800s [4]

tolerance. Thus, this method is known as an incomplete LU (ILU) preconditioner [17]. One advantage is that these methods are chosen purely algebraically. For realistic problems the choice of the fill tolerance can be a hard quantity to determine [4]. These methods are not scalable and there is difficulty with implementing and effectively using this technique in parallel; however they are useful as smoothers in a multigrid iteration [14].

1.4.2 Physics-based Preconditioners

The methods discussed in Section 1.4.1 are developed from a purely algebraic point of view. Physics-based preconditioners use the underlying physical problem as motivation for the derivation of the methodology. By using information about the underlying physical model one can develop more robust preconditioners. This is especially useful in applications with PDEs, where one has good knowledge of the problem at hand, the domain of the problem, and the boundary conditions.

For the Navier-Stokes equations, the major bottleneck in terms of CPU time is the iterative solution of the linear systems that arise after discretizing and linearizing the underlying PDE equations. The main goal of this research is to study scalable preconditioning techniques for the incompressible Navier-Stokes equations and to explore their utility in several applied settings. In Chapter 3, we discuss SIMPLE, a purely physics-based preconditioning technique historically used as a solver for the Navier-Stokes equations. In the rest of this section we focus on domain decomposition and multilevel multigrid techniques, which combine properties

from both physics-based and purely algebraic preconditioning strategies.

The domain decomposition method is particularly useful in parallel computation because it allows the domain of the problem to be broken into disjoint (or slightly overlapping) subdomains, each of which can be solved on a corresponding processor. When the subproblems on the smaller domains are preconditioned with an ILU or sparse approximate inverse technique, domain decomposition parallelizes well, but the requirement that coarse grids be handled by direct methods causes it to scale poorly in parallel settings or as the problem size increases. A further discussion of these and other techniques can be found in [4].

Multigrid methods are the most effective methods for solving linear systems associated with discrete PDEs. Multigrid is based on the premise of resolving errors by using multiple problem resolutions in an iterative scheme. High oscillatory components of the error are mitigated by a smoothing procedure (such as Gauss-Seidel/Jacobi), whereas low energy components are eliminated using a lower resolution version of the discrete problem on a coarse grid. Interpolation and restriction operators are defined to move data, such as residuals, between meshes. Coarse grid operators can be defined using the same discretization technique as used for the fine grid operator. It is also possible to algebraically produce this operator by

$$A_{k+1} = R_k^T A_k P_k \tag{1.7}$$

where P_k interpolates a solution from one the fine operator to the coarse operator and R_k^T restricts the solution from the coarse operator to the fine operator [9, 59].

In geometric multigrid, a sequence of different resolution meshes are created.

Grid transfer operators are created to move data from one mesh to another, where a new discretization is constructed on all meshes. Algebraic multigrid (AMG) mitigates the dependence on the grid structure by requiring no mesh or geometric information, so instead of constructing a mesh, AMG develops coarse grid operators from the matrix data. This makes AMG attractive for problems posed on complex domains or unstructured meshes [59, 64].

These multilevel multigrid-like methods show promise, while combining stationary iterative methods, like Gauss-Seidel or ILU, as smoothers in a multigrid iteration. However, there are still many open questions about them. In particular, for algebraic multigrid, there are some issues with applying these techniques to indefinite systems, systems of PDE equations, and to three-dimensional problems [4].

The rest of this dissertation is organized as follows. In Chapter 2, we derive the Navier-Stokes equations from the principles of Conservation of Mass and Momentum and then discuss some aspects of linearizing and discretizing a mixed finite element formulation of the resultant partial differential equation. In Chapter 3, we describe and develop a taxonomy for preconditioning strategies specifically designed for the Navier-Stokes equations. These methods include pressure correction schemes, like SIMPLE, developed by [41, 42] and approximate Schur complement preconditioners designed by [19, 34, 53]. In Chapter 4, we describe numerical results posed on a variety of sample test problems. In Chapter 5, we describe how these methods can be applied to solving realistic shape and topology optimization of microfluidic problems. In Chapter 6, we discuss the high performance computing environment

used to implement and test the efficacy of these strategies.

Chapter 2

The Navier-Stokes Equations

In this chapter, we derive, then linearize and discretize the incompressible Navier-Stokes equations. In Section 2.1, we derive the Navier-Stokes equations using first principles, such as conservation of mass, momentum, and the transport theorem. In Section 2.2, we describe the finite element formulation and corresponding linearization of the Navier-Stokes equations that leads to the saddle point-like linear system that we derive our preconditioners from.

2.1 Derivation of the Incompressible Navier-Stokes Equations

2.1.1 Notation

Fluid flow in a region $\Omega \in \mathbb{R}^2$ (or $\mathbb{R}^3$), over a time interval $[0, t]$ is described with the following notation:

- $p \in \mathbb{R}$ denotes the pressure
- $\nu \in \mathbb{R}$ denotes the viscosity
- $\mathbf{u} \in \mathbb{R}^N$ denotes the velocity
- $\rho \in \mathbb{R}$ denotes the density
- Ω denotes the physical region on which the model applies

- $\mathbf{f}$ represents the body forces

The incompressible Navier-Stokes equation are

$$\alpha \mathbf{u}_t - \nu \nabla^2 \mathbf{u} + (\mathbf{u} \cdot \text{grad})\mathbf{u} + \text{grad } p = \mathbf{f} \qquad \text{(Momentum equation)} \tag{2.1}$$

$$-\text{div } \mathbf{u} = 0 \qquad \text{(Continuity equation).} \tag{2.2}$$

The parameter $\alpha = 0$ corresponds to the steady state problem and $\alpha = 1$ the transient case. The incompressible Navier-Stokes equations are derived using the laws of conservation of mass and momentum. For this derivation, we assume that the system is closed, so no fluids flow across the boundaries of Ω.

2.1.2 Conservation of Mass

We begin by stating the transport theorem, which states how to calculate the time integral of a domain changing over time [29].

Transport Theorem: For a differentiable function, $\mathbf{f}$,

$$\frac{d}{dt}\int_{\Omega_t} \mathbf{f}(x,t)dx = \int_{\Omega_t} \left(\frac{\partial}{\partial t}\mathbf{f} + \text{div}(\mathbf{fu})\right)(x,t)dx. \tag{2.3}$$

The mass of a fluid at time, t, is the integral over the density of the fluid. Since this is a closed system, the amount of fluid at time $t = 0$ must equal the amount of fluid at any other time. Therefore,

$$\int_{\Omega_0} \rho(x,0)dx = \int_{\Omega_t} \rho(x,t)dx.$$

Using (2.3), we have

$$\int_{\Omega_t} \left(\frac{\partial}{\partial t}\rho + \text{div}(\rho\mathbf{u})\right)(x,t)dx = 0. \tag{2.4}$$

Equation (2.4) holds for any region (Ω_t), so the integral can be dropped, leaving

$$\frac{\partial}{\partial t}\rho + \text{div}(\rho \mathbf{u}) = 0$$

as the local form of the Conservation of Mass (COM). For an incompressible fluid, ρ is constant so the local COM simplifies to

$$\text{div } \mathbf{u} = 0$$

which is the continuity equation (2.2).

2.1.3 Conservation of Momentum

The momentum of a body is mass times velocity. Since a fluid's velocity changes with respect to its position, the momentum of a fluid is represented by the integral over the domain,

$$m(t) = \int_{\Omega_t} \rho(x,t)\mathbf{u}(x,t)dx. \tag{2.5}$$

Recall Newton's second law, which states:

$$\frac{d}{dt}m(t) = \sum \text{forces.}$$

For our problem, we have two types of forces, those that arise on the surface of the region and those found on the inside of the region. The interior forces are represented by $\int_{\Omega_t} \rho(x,t)g(x,t)dx$ (where g is the force density). The surface forces are represented by $\int_{\Omega_t} \sigma(x,t)\mathbf{n}ds$, where σ is the stress tensor. The interior stress relies on two quantities, the pressure and internal friction. So, the stress tensor, σ, is

$$\sigma(x,t) = -p(x,t)I + \lambda(\text{div } \mathbf{u})I + \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$$

where λ and μ are two quantities that characterize the viscosity of the fluid. Then,

$$\frac{d}{dt}m(t) = \frac{d}{dt}\int_{\Omega_t} \rho(x,t)\mathbf{u}(x,t)dx = \int_{\Omega_t} \rho(x,t)g(x,t)dx + \int_{\partial\Omega_t} \sigma(x,t)\mathbf{n}\ ds$$

Substitution of $\sigma(x,t)$ into (2.5) gives

$$\frac{d}{dt}m(t) = \int_{\Omega_t} \rho(x,t)g(x,t)dx + \int_{\partial\Omega_t} -p(x,t)I + \lambda(\text{div } \mathbf{u})I + \mu\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)\mathbf{n}ds \tag{2.6}$$

Applying the transport and divergence theorem term by term to (2.6) gives

$$\frac{\partial}{\partial t}\rho\mathbf{u} + (\mathbf{u}\cdot\text{grad }(\rho\mathbf{u})) + \text{div }\mathbf{u} + \text{grad } p = (\mu+\lambda)\text{ grad}(\text{div }\mathbf{u}) + \mu\triangle\mathbf{u} + \rho g. \tag{2.7}$$

Since ρ is constant for an incompressible fluid, (2.7) simplifies to

$$\frac{\partial}{\partial t}\mathbf{u} + (\mathbf{u}\cdot\text{grad})\ (\rho\mathbf{u}) + \frac{1}{\rho}\text{grad } p = \frac{\mu}{\rho}\triangle\mathbf{u} + g.$$

Then, let the viscosity be $\nu = \frac{\mu}{\rho}$ and lump ρ into the pressure term, to get

$$\frac{\partial}{\partial t}\mathbf{u} + (\mathbf{u}\cdot\text{grad})\mathbf{u} + \text{grad } p - \nu\triangle\mathbf{u} = g.$$

Reordering terms gives

$$\mathbf{u}_t - \nu\nabla^2\mathbf{u} + (\mathbf{u}\cdot\text{grad})\mathbf{u} + \text{grad } p = g,$$

which is (2.1). A further discussion of this derivation can be found in [29].

2.2 Finite Element Discretization

We describe how to discretize (using a mixed finite element approach) the incompressible Navier-Stokes equations (2.1)-(2.2). We focus our attention on the

spatial discretization of these equations. For the temporal component one can use a variety of time-stepping strategies, including the Crank-Nicholson or Backward Euler method [11]. In discretizing the Navier-Stokes equations, finite element spaces and appropriate bases for these spaces are defined, followed by a discussion of how to linearize the nonlinear convection term. In Section 2.2.4, we discuss a few details on stabilized finite element methods for the Navier-Stokes equations.

2.2.1 Weak Formulation

To define a weak formulation of the Navier-Stokes equations, we define a set of solution and test spaces for the velocity field, namely

$$\mathbf{H}_E^1 = \{\mathbf{u} \in \mathcal{H}^1(\Omega) | \mathbf{u} = \mathbf{w} \text{ on } \partial\Omega_D\}$$

$$\mathbf{H}_{E_0}^1 = \{\mathbf{v} \in \mathcal{H}^1(\Omega) | \mathbf{v} = \mathbf{0} \text{ on } \partial\Omega_D\}$$

where $\mathcal{H}^1(\Omega)$ is the Sobolev space of functions defined on Ω whose first derivatives exist and are square integrable in Ω. An analogous space for the pressure field is the space of functions that are square-integrable, denoted as $L_2(\Omega)$. One can choose $L_2(\Omega)$ as the test space for the pressure, p, since derivatives of the pressure field do not appear in the weak formulation. The weak formulation of (2.1)-(2.2) is:

Find $\mathbf{u} \in \mathbf{H}_E^1$ and $p \in L_2(\Omega)$ such that

$$\nu \int_\Omega \nabla\mathbf{u}\nabla\mathbf{v} + \int_\Omega (\mathbf{u} \cdot \nabla\mathbf{u}) \cdot \mathbf{v} - \int_\Omega p(\nabla \cdot \mathbf{v}) = \int_\Omega \mathbf{f} \cdot \mathbf{v} \quad \forall \mathbf{v} \in \mathbf{H}_{E_0}^1$$

$$\int_\Omega q(\nabla \cdot \mathbf{u}) = 0 \quad \forall q \in L_2(\Omega).$$

The key to an effective weak formulation is ensuring that the inf-sup condition,

$$\inf_{q\neq\text{constant}}\sup_{v\neq 0}\frac{|q,\nabla\cdot v|}{||v||_1||q||}\geq 0 \tag{2.8}$$

where $||\mathbf{v}||$ is a norm for the functions in $\mathbf{H}^1_E$ and $||q||$ is a quotient space norm is satisfied. Further discussions of how this condition applied to this problem can be found in [17, 26].

2.2.2 Treating the Nonlinear Term

Solving the Navier-Stokes equations is difficult due to the nonlinear term found in the convection term of the momentum equation, $\mathbf{u}\cdot\nabla\mathbf{u}$. For treating the nonlinearity, we use Newton's method or Picard's method. We begin with how Newton's method applies to this problem in Section 2.2.2.1, followed by Picard's method in the following subsection.

2.2.2.1 Newton's Method

If we begin with an initial guess, denoted $(\mathbf{u_0}, p_0)$, the nonlinear residual of the k^{th} iterate $(\mathbf{u_k}, p_k)$ produced by an iteration associated with the weak formulation is

For any $\mathbf{v}\in\mathbf{H}^1_{E_0}(\Omega)$ and $q\in L_2(\Omega)$,

$$R_k=\int_\Omega \mathbf{f}\cdot\mathbf{v}-\int_\Omega(\mathbf{u_k}\nabla\mathbf{u_k})\cdot\mathbf{u}_k-\nu\int_\Omega\nabla\mathbf{u_k}\nabla\mathbf{v}+\int_\Omega p_k(\nabla\cdot\mathbf{v}) \tag{2.9}$$

$$r_k=-\int_\Omega q(\nabla\cdot\mathbf{u_k}). \tag{2.10}$$

Now, denote the nonlinear updates for the velocity and pressure by $\mathbf{u_k} = \mathbf{u_k} + \delta\mathbf{u_k}$ and $p_k = p_k + \delta p_k$, respectively. Since $\delta\mathbf{u_k} \in \mathbf{H}^1_{E_0}$ and $\delta p_k \in L_2(\Omega)$, it follows that

$$\int_\Omega (\delta\mathbf{u_k}+\mathbf{u_k})\cdot\nabla(\delta\mathbf{u_k}+\mathbf{u_k})\cdot\mathbf{v} - \int_\Omega (\mathbf{u_k}\cdot\nabla\mathbf{u_k})\cdot\mathbf{v} + \nu\int_\Omega \nabla\delta\mathbf{u_k}\nabla\mathbf{v} - \int_\Omega \delta p_k(\nabla\cdot\mathbf{v}) = R_k(\mathbf{v})$$

$$\int_\Omega q(\nabla\cdot\delta\mathbf{u_k}) = r_k(q). \tag{2.11}$$

If the quadratic term in (2.11) is dropped and the remaining expressions are expanded, then a linear subproblem is determined, namely

For all $\mathbf{v} \in \mathbf{H}^1_{E_0}$ and $q \in L_2(\Omega)$, find $\delta\mathbf{u}_k \in \mathbf{H}^1_{E_0}$ and $\delta p_k \in L_2(\Omega)$ such that

$$\int_\Omega (\delta\mathbf{u_k}\cdot\nabla\mathbf{u_k})\cdot\mathbf{v} + \int_\Omega (\mathbf{u_k}\cdot\nabla\delta\mathbf{u_k})\cdot\mathbf{v} + \nu\int_\Omega \nabla\delta\mathbf{u_k}\nabla\mathbf{v} - \int_\Omega \delta p_k(\nabla\cdot\mathbf{v}) = R_k(\mathbf{v}) \tag{2.12}$$

$$\int_\Omega q(\nabla\cdot\delta\mathbf{u_k}) = r_k(q) \tag{2.13}$$

holds. Solving (2.13), generates the Newton correction. The next iterate in the sequence is defined by $\mathbf{u_{k+1}} = \mathbf{u_k} + \delta\mathbf{u_k}$ and $p_{k+1} = p_k + \delta p_k$.

2.2.2.2 Picard's Method

Another means of linearizing the convective term is by generating an iterate for the nonlinear problem using Picard's method. If we begin with (2.11), then this method is derived by dropping both the nonlinear term, $\int_\Omega(\delta\mathbf{u_k}\cdot\nabla\delta\mathbf{u_k})\cdot\mathbf{v}$, and the linear term, $\int_\Omega(\delta\mathbf{u_k}\cdot\nabla\mathbf{u_k})\cdot\mathbf{v}$ (a zeroth order operator in $\delta\mathbf{u}_k$). Therefore, the following linear problem is formulated

For all $\mathbf{v} \in \mathbf{H^1_{E_0}}$ and $q \in L_2(\Omega)$, find $\delta\mathbf{u_k} \in \mathbf{H^1_{E_0}}$ and $p_k \in L_2(\Omega)$ such that

$$\int_\Omega (\mathbf{u_k} \cdot \nabla \delta\mathbf{u_k}) \cdot \mathbf{v} + \nu \int_\Omega \nabla \delta\mathbf{u_k} \nabla \mathbf{v} + \int_\Omega \delta p_k (\nabla \cdot \mathbf{v}) = R_k(\mathbf{v})$$

$$\int_\Omega q(\nabla \cdot \delta\mathbf{u_k}) = r_k(q) \tag{2.14}$$

holds. Solving (2.14), generates the Picard correction. The next iterate in the sequence is defined by $\mathbf{u_{k+1}} = \mathbf{u_k} + \delta\mathbf{u_k}$ and $p_{k+1} = p_k + \delta p_k$.

2.2.2.3 Differences between Newton's and Picard's Methods

These two strategies for treating the nonlinear convection term each have advantages and disadvantages. Newton's method provides quadratic convergence, whereas the Picard iteration is linearly convergent. One disadvantage of Newton's method is that its radius of convergence is typically proportional to the Reynolds number [17]. Therefore for increasing Reynolds number better and better initial guesses are needed to ensure this technique converges to a solution. On the other hand, the Picard iteration has a much larger radius of convergence than Newton's method [33].

2.2.3 Mixed Finite Element Approximation

A discrete weak formulation is defined using finite-dimensional spaces $\mathbf{X}_0^h \subset \mathbf{H}^1_{E_0}(\Omega)$ and $M^h \subset L_2(\Omega)$. Since these approximations are made independently, this discretization is known as a mixed finite element approximation. More specifically, given a velocity solution space, $\mathbf{X}_E^h$, the discrete version of the weak formulation is

Find $\mathbf{u}_h \in \mathbf{X}_E^h$ and $p_h \in M^h$ such that

$$\nu \int_\Omega \nabla \mathbf{u}_h \nabla \mathbf{v}_h + \int_\Omega (\mathbf{u}_h \cdot \nabla \mathbf{v}_h) \cdot \mathbf{v}_h - \int_\Omega p_h (\nabla \cdot \mathbf{v}_h) = \int_\Omega \mathbf{f} \cdot \mathbf{v}_h \qquad \forall \mathbf{v}_h \in \mathbf{X}_0^h$$

$$\int_\Omega q_h (\nabla \cdot \mathbf{u}_h) = 0 \qquad \forall q_h \in M^h$$

If we linearize the problem using Newton's method, this gives the following discrete weak version of (2.12):

Find corrections, $\delta \mathbf{u}_h \in \mathbf{X_0^h}$ and $\delta p_h \in M^h$ for all $\mathbf{v_h} \in \mathbf{X}_0^h$ and $q_h \in M^h$ such that

$$\int_\Omega (\delta \mathbf{u_h} \cdot \nabla \mathbf{u_h}) \cdot \mathbf{v_h} + \int_\Omega (\mathbf{u_h} \cdot \nabla \delta \mathbf{u_h}) \cdot \mathbf{v_h} + \nu \int_\Omega \nabla \delta \mathbf{u_h} \nabla \mathbf{v_h} - \int_\Omega \delta p_h (\nabla \cdot \mathbf{v_h}) = R_k(\mathbf{v_h})$$

$$\int_\Omega q_h (\nabla \cdot \delta \mathbf{u}_h) = r_k(q_h)$$

holds. A set of vector basis functions, $\{\phi_j\}$ is introduced to define the corresponding linear subproblem, namely

$$\mathbf{u_h} = \sum_{j=1}^{N_d} \mathbf{u}_j \phi_j + \sum_{j=N_d+1}^{N_d+N_D} \mathbf{u}_j \phi_j, \qquad \delta \mathbf{u}_h = \sum_{j=1}^{N_d} \triangle \mathbf{u}_j \phi_j \tag{2.15}$$

with $\sum_{j=1}^{N_d} \mathbf{u}_j \phi_j \in \mathbf{X}_0^h$, where the coefficients of $\mathbf{u}_j$ are fixed so that the second sum represents the boundary conditions on the domain, $\partial \Omega_D$. By introducing a set of scalar valued pressure basis functions, $\{\psi_k\}$, a similar discrete sum can be defined:

$$p_h = \sum_{i=1}^{m} \mathbf{p}_i \psi_i, \qquad \delta p_h = \sum_{i=1}^{m} \triangle \mathbf{p}_i \psi_i \tag{2.16}$$

Substituting these basis functions into the linearized formulation, a system of linear equations can be determined:

$$\begin{bmatrix} \nu \mathbf{A} + \mathbf{W} + \mathbf{N} & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} \triangle \mathbf{u} \\ \triangle \mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{g} \end{bmatrix}. \tag{2.17}$$

The matrices are defined as

$$\mathbf{A} = [\mathbf{a}_{ij}], = \mathbf{a_{ij}} = \int_\Omega \nabla\phi_i \nabla\phi j$$

$$B = [b_{mj}], b_{kj} = -\int_\Omega \psi_k \nabla \cdot \phi_j$$

$$\mathbf{N} = [\mathbf{n}_{ij}], = \mathbf{n_{ij}} = \int_\Omega (\mathbf{u}_h \cdot \nabla\phi_j) \cdot \phi_i$$

$$\mathbf{W} = [\mathbf{w}_{ij}], = \mathbf{w_{ij}} = \int_\Omega (\phi_j \cdot \nabla\mathbf{u}_h) \cdot \phi_i$$

for i and $j = 1, \ldots, n_d$. The matrix $\mathbf{A}$ is the vector-Laplacian matrix and $\mathbf{B}$ the divergence matrix. In addition, $\mathbf{W}$ is the vector-convection matrix, and $\mathbf{N}$ is the Newton derivative matrix. The latter three matrices depend on the current value of the velocity, $\mathbf{u}_h$, whereas the vector-Laplacian and divergence matrix do not depend on this quantity. The entries of the right hand vectors $\mathbf{f}$ and $\mathbf{g}$ are

$$\mathbf{f} = [f_i], \quad f_i = \int_\Omega \mathbf{f} \cdot \phi_i - \int_\Omega \mathbf{u}_h \cdot \nabla\mathbf{u}_h \cdot \phi_i - \nu \int_\Omega \nabla\mathbf{u}_h \nabla\phi_i + \int_\Omega p_h(\nabla \cdot \phi_i)$$

$$\mathbf{g} = [g_k], \quad g_m = \int_\Omega \psi_m(\nabla \cdot \mathbf{u}_h).$$

When (2.17) is derived using Newton's method, this saddle point system is called the discrete Newton problem. If the Newton derivative matrix, $\mathbf{N}$, is not present (i.e. for Picard iteration), then this is a discrete Oseen problem. We denote the $(1,1)$ block of the saddle point matrix F, so that we can develop preconditioners for either Newton's method or the Oseen subproblem in a uniform manner. In the case of Newton's method, F is of the form, $F = \nu\mathbf{A} + \mathbf{W} + \mathbf{N}$, whereas the Oseen operator, $F = \nu\mathbf{A} + \mathbf{W}$. Therefore, the resulting subproblem is of the form

$$\mathcal{S} = \begin{bmatrix} F & B^T \\ B & 0 \end{bmatrix}.$$

For a further description of the stability of this method along with issues that arise regarding uniqueness of a solution, the reader is encouraged to look at [8, 17, 26].

2.2.4 Stabilized Finite Element Discretization

In this section, we include some details on stabilized finite element methods for the Navier-Stokes equations. This also includes details of the stabilization used in the finite element discretization used in our results in Chapter 4. For stable finite elements, the pressure and velocity approximations must satisfy the discrete version of the inf-sup condition defined in (2.8),

$$\min_{q_h \neq \text{constant}} \max_{v \neq 0} \frac{|q_h, \nabla \cdot v_h|}{||v_h||_1 ||q||} \geq 0. \tag{2.18}$$

Some examples of stable finite element pairs are $Q2 - Q1$, $Q2 - P0$, or $P2 - P1$, where Q refers to a quadrilateral element and P a triangular element [17]. The value following the letter Q or P refers to the order of the approximation. So, $P2 - P1$ represents piecewise quadratic approximations posed on triangles for the velocity approximation and piecewise linear approximations for the pressure approximation.

Many seemingly "natural" elements violate (2.18), so the associated discretizations are not stable. This is true, for example, for equal order velocity and pressure elements defined on a common grid. The premise behind stabilization is to relax the incompressibility constraint in a special way to allow the use of approximations that do not satisfy the inf-sup condition.

Consistently stabilized finite element formulations enable the use of a wider range of velocity-pressure pairs. Equal order pairs provide more uniform data struc-

tures and discrete algebraic systems that are easier to solve using iterative methods than non equal order pairs. For these reasons, stabilized finite element methods are commonly used for discretizing both steady-state and time-dependent Navier-Stokes problems.

We focus our attention on consistently stabilized methods which stabilize the finite element formulation by weighting the residuals of the underlying differential equations. More specifically, we discuss the weak formulation for a Q1-Q1 steamwise upwinded Petrov-Galerkin method, which is a specific case of the stabilized Galerkin Least Squares finite element method [56]. The weak formulation for our consistently stabilized Galerkin-least squares mixed method is:

Find corrections, $\delta\mathbf{u}_h \in \mathbf{X_0^h}$ and $\delta p_h \in M^h$ $\forall \mathbf{v}_h \in \mathbf{H}^1_{E_0}$ and $\forall q_h \in L_2(\Omega)$ such that

$$\int_\Omega (\delta\mathbf{u_h}\cdot\nabla\mathbf{u_h})\cdot\mathbf{v_h} + \int_\Omega (\mathbf{u_h}\cdot\nabla\delta\mathbf{u_h})\cdot\mathbf{v_h} + \nu\int_\Omega \nabla\delta\mathbf{u_h}\nabla\mathbf{v_h} - \int_\Omega \delta p_h(\nabla\cdot\mathbf{v_h}) - \int_\Omega q_h(\nabla\cdot\delta\mathbf{u}_h)$$
$$= \int_\Omega \mathbf{f}\cdot\mathbf{v}_h + \sum_{K\in\mathcal{T}_h}\int_\Omega \tau_K(-\nabla^2(\mathbf{u}_h+\delta\mathbf{u}_h) + (\mathbf{u}_h+\delta\mathbf{u}_h)\cdot\nabla(\mathbf{u}_h+\delta\mathbf{u}_h) + \nabla p_h - \mathbf{f})$$
$$\cdot(-\nabla^2\mathbf{v}_h + (\mathbf{u}_h+\delta\mathbf{u}_h)\cdot\nabla\mathbf{v}_h + \nabla q_h)_K \tag{2.19}$$

holds. The stabilization parameter, τ_K, is proportional to δh^2 where h a measure of the element size and $\delta > 0$ is proportional to the viscosity. Note that the first four integrals are similiar to the stable formulation above. The term with τ_K represents the least-squares stabilization term that is added to make the mixed Galerkin formulation stable for equal order elements [7, 56]. The constant τ_K determines the weight of the stabilization terms in the weak formulation.

A set of vector basis functions, $\{\phi_j\}$ for the velocity and $\{\psi_j\}$ for the pressure as defined in (2.15) and (2.16), respectively, are needed to determine the system of linear equations. Substituting these basis functions into the linearized formulation, determines the following system of equations:

$$\begin{bmatrix} \nu\mathbf{A} + \mathbf{W} + \mathbf{N} & B^T \\ B - \tilde{S} & -\tilde{C} \end{bmatrix} \begin{bmatrix} \triangle\mathbf{u} \\ \triangle\mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \tilde{\mathbf{g}} \end{bmatrix}. \tag{2.20}$$

The matrices A, W, N, B^T, B, and f are defined as above. The stabilization matrices, $\tilde{S}$, $\tilde{C}$, and $\tilde{g}$ are defined as:

$$\tilde{S} = [\tilde{s}_{mj}], \quad s_{mj} = -\sum_{K\in\mathcal{T}_h} \tau_K \int_\Omega (\triangle\phi_k \nabla\psi_j)_K$$

$$\tilde{C} = [c_{ij}], \quad c_{ij} = \sum_{K\in\mathcal{T}_h} \tau_K \int_\Omega (\nabla\psi_j \cdot \nabla\psi_i)_K$$

$$\tilde{g} = [g_k], \quad g_m = \sum_{K\in\mathcal{T}_h} \tau_K \int_\Omega (\mathbf{f} \cdot \nabla\phi_i)_K.$$

When the $\sim$ matrices are not present, one recovers a similiar matrix to that of the unstabilized method. The presence of the $\tilde{C}$ matrix in the $(2,2)$ entry of the system relaxes the incompressiblity constraint and enables the use of equal order velocity-pressure pairs.

Chapter 3

Navier-Stokes Preconditioners - Taxonomy and Classification

Factorization Background

3.1 Taxonomy of Approximate Block Factorization Preconditioners

We focus on solution algorithms for the algebraic system of equations that results from linearization and discretization of the incompressible Navier-Stokes equations. The coefficient matrices have the general form

$$A = \begin{pmatrix} F & B^T \\ \hat{B} & -C \end{pmatrix} \tag{3.1}$$

where F is a convection-diffusion-like operator, B^T is the gradient operator, $\hat{B}$ is the divergence operator that for some higher-order stabilized formulations can include a contribution from non-zero higher-order derivative operators in the stabilized formulation [10], and C is the operator that stabilizes the finite element discretization as described in Section 2.2.4. The strategies we employ for solving (3.1) are derived from the LDU block factorization of this coefficient matrix,

$$A = \begin{pmatrix} I & 0 \\ \hat{B}F^{-1} & I \end{pmatrix} \begin{pmatrix} F & 0 \\ 0 & -S \end{pmatrix} \begin{pmatrix} I & F^{-1}B^T \\ 0 & I \end{pmatrix}, \tag{3.2}$$

where

$$S = C + \hat{B}F^{-1}B^T \tag{3.3}$$

is the Schur complement (of F in A). They require methods for approximating the action of the inverse of the factors of (3.2), which, in particular, requires approximation to the actions of F^{-1} and S^{-1}. For large-scale computations, use of the exact Schur complement is not feasible. Therefore, effective approximate block factorization (ABF) preconditioners are often based on a careful consideration of the spectral properties of the component block operators and the approximate Schur complement operators. There has been a great deal of recent work on ABF methods (e.g. [5, 6, 13, 17, 34]). These techniques take a purely linear algebraic view of preconditioning. Through these decompositions a simplified system of block component equations is developed that encodes a specific "physics-based" decomposition. Alternatively, one could start with "physics-based" iterative solution methods for the Navier-Stokes equations (e.g. [41, 42]) and develop preconditioners based on these techniques as described in [35]. In both these cases, the system has been transformed by the factorization to component systems that are essentially convection-diffusion and Poisson type operators. The result is a system to which multi-level methods, and in our particular case, algebraic multi-level methods (AMG), can be applied successfully for parallel unstructured mesh simulations.

We adopt a nomenclature for projection type methods based on algebraic splittings developed by Quarteroni, Saleri, and Veneziani [45] for algebraic splittings of A for projection type methods. Let H_1 represent an approximation to F^{-1} in the Schur complement (3.3) and let H_2 be an approximation to F^{-1} in the upper triangular

block of the factorization (3.2). This results in the following decomposition:

$$A_s = \begin{bmatrix} I & 0 \\ \hat{B}F^{-1} & I \end{bmatrix} \begin{bmatrix} F & 0 \\ 0 & -(C + \hat{B}H_1B^T) \end{bmatrix} \begin{bmatrix} I & H_2B^T \\ 0 & I \end{bmatrix} = \begin{bmatrix} F & FH_2B^T \\ \hat{B} & -(C + \hat{B}(H_1 - H_2)B^T) \end{bmatrix}. \tag{3.4}$$

The error matrix $E_s = A - A_s$ is

$$E_s = \begin{bmatrix} 0 & (I - FH_2)B^T \\ 0 & -\hat{B}(H_2 - H_1)B^T \end{bmatrix}.$$

This decomposition is used in [45] to illuminate the structure of several projection techniques for solving the time-dependent Navier-Stokes equations. By examining the error, we can determine which equation (momentum or continuity) in the original problem is perturbed by the approximations H_1 or H_2 in the above decomposition. For example, if $H_1 = F^{-1}$ and $H_1 \neq H_2$, then the operators applied to the pressure in both the momentum equation and continuity equation are perturbed, whereas operators applied to the velocity are not perturbed. On the other hand, if $H_2 = F^{-1}$ and $H_1 \neq H_2$, then the $(1, 2)$ block of the error matrix is zero. So, the momentum equation is unperturbed, thus giving a "momentum preserving strategy," whereas a perturbation of the incompressibility constraint occurs [45]. If $H_1 = H_2 \neq F^{-1}$, then the scheme is "mass preserving" because the $(2, 2)$ block of the error matrix is zero, so the continuity equation is not modified. Finally, if $H_1 \neq H_2 \neq F^{-1}$, then both the momentum and continuity equations are modified.

The above factorization can be generalized to incorporate "classical" methods used for these problems, such as SIMPLE, SIMPLEC, SIMPLER, [42, 43], as well as newer approximate commutator methods devised to generate good approximatations

to the Schur complement [34, 53]. Let us modify (3.4) using some approximation H_1 in place of F^{-1} in the lower triangular block. In addition, let $\hat{S}$ represent an approximation of the Schur complement. Then

$$\tilde{A} = \begin{bmatrix} I & 0 \\ \hat{B}H_1 & I \end{bmatrix} \begin{bmatrix} F & 0 \\ 0 & -\hat{S} \end{bmatrix} \begin{bmatrix} I & H_2B^T \\ 0 & I \end{bmatrix} = \begin{bmatrix} F & FH_2B^T \\ \hat{B}H_1F & \hat{B}H_1FH_2B^T - \hat{S} \end{bmatrix}. \quad (3.5)$$

The error, denoted $\tilde{E} = A - \tilde{A}$, is

$$\tilde{E} = \begin{bmatrix} 0 & B^T - FH_2B^T \\ \hat{B} - \hat{B}H_1F & \hat{S} - (C + \hat{B}H_1FH_2B^T) \end{bmatrix}.$$

By examining the error matrix, we can determine which equation (momentum or continuity) in the original problem is perturbed by the approximations H_1, H_2, or $\hat{S}$.

Techniques explored in this chapter can be classified into two categories: those whose factorization groups the lower triangular and the diagonal components as $[(LD)U]$, and those that group the diagonal and lower triangular components as $[L(DU)]$. Methods with the $(LD)U$ grouping have the factorization

$$\tilde{A}_{(LD)U} = \begin{bmatrix} F & 0 \\ \hat{B}H_1F & -\hat{S} \end{bmatrix} \begin{bmatrix} I & H_2B^T \\ 0 & I \end{bmatrix}. \quad (3.6)$$

Methods with the $L(DU)$ grouping have the factorization

$$\tilde{A}_{L(DU)} = \begin{bmatrix} I & 0 \\ \hat{B}H_1 & I \end{bmatrix} \begin{bmatrix} F & FH_2B^T \\ 0 & -\hat{S} \end{bmatrix}. \quad (3.7)$$

Some of the techniques considered do not use the complete factorization (3.6) or (3.7), but rather use only triangular components of the factorization. SIMPLE uses

the block $(LD)U$ grouping. The approximate commutator methods are derived from the block $L(DU)$ grouping and use just the diagonal and upper triangular (DU) components in the method. Finally, these classifications are further refined by specifying strategies for approximating the Schur complement.

Next we introduce the pressure correction method as a stationary iteration that begins with an initial iterate for the velocity, $\mathbf{u}_n$, and pressure, p_n, and calculates the next velocity, $\mathbf{u}_{n+1}$ and pressure, p_{n+1} iterates using a systematic procedure. Then, these methods are put into the LDU grouping in (3.5). For our computational results in Chapters 5 and 6, both the pressure correction and approximate commutator methods are implemented as preconditioners for a Krylov subspace method. Therefore, applying the action of the inverse of $\tilde{A}$ is required at each iteration.

3.1.1 Pressure Correction

The pressure correction family of Navier-Stokes preconditioners is derived from the divergence free constraint with decoupling of the incompressible Navier-Stokes equations. In the following sections, three pressure correction methods are derived, SIMPLE, SIMPLEC, and SIMPLER (Semi-Implicit Method for Pressure Linked Equations Revised) [42, 43, 44, 63].

3.1.1.1 The SIMPLE Preconditioner

The SIMPLE-like algorithm described here begins by solving a variant of the momentum equation for an intermediate velocity using a previously generated pres-

sure; then the continuity equation is solved using the intermediate velocity to calculate the pressure update. This value is used to update the velocity component. The SIMPLE algorithm is as follows:

1. Solve: $F\mathbf{u}_{n+\frac{1}{2}} = f - B^T p_n$ for the velocity, $\mathbf{u}$.

2. Solve: $-(C + \hat{B}\text{diag}(F)^{-1}B^T)\delta p = \hat{B}\mathbf{u}_{n+\frac{1}{2}} + Cp_n$ for δp.

3. Calculate the velocity correction: $\delta\mathbf{u} = \mathbf{u}_{n+1} - \mathbf{u}_{n+\frac{1}{2}} = (-\text{diag}(F)^{-1}B^T)\delta p$.

4. Update the pressure: $p_{n+1} = p_n + \alpha\delta p$

5. Update the velocity: $\mathbf{u}_{n+1} = \mathbf{u}_{n+\frac{1}{2}} + \delta\mathbf{u}$

The quantity α is a parameter in $(0, 1]$ that damps the pressure update [43].

An alternative derivation of SIMPLE is obtained using the LDU framework described above. The block lower triangular factor (L) and the block diagonal (D) are grouped together. In terms of the taxonomy described above, this corresponds to the choices $H_1 = F^{-1}$, $H_2 =(\text{diag}(F))^{-1}$, and $\hat{S} = C + \hat{B}(\text{diag}(F))^{-1}B^T$ in (3.6). The decomposition is

$$\begin{bmatrix} F & B^T \\ \hat{B} & -C \end{bmatrix} \approx \begin{bmatrix} I & 0 \\ \hat{B}F^{-1} & I \end{bmatrix} \begin{bmatrix} F & 0 \\ 0 & \hat{S} \end{bmatrix} \begin{bmatrix} I & (\text{diag}(F))^{-1}B^T \\ 0 & \alpha I \end{bmatrix}$$
$$= \begin{bmatrix} F & 0 \\ \hat{B} & -\hat{S} \end{bmatrix} \begin{bmatrix} I & (\text{diag}(F))^{-1}B^T \\ 0 & \alpha I \end{bmatrix} = \tilde{A}_{SIMPLE}.$$

Thus, one iteration of SIMPLE corresponds to

$$\begin{bmatrix} \mathbf{u}_{n+1} \\ p_{n+1} \end{bmatrix} = \begin{bmatrix} \mathbf{u}_n \\ p_n \end{bmatrix} + \tilde{A}^{-1}_{SIMPLE} \left(\begin{bmatrix} \mathbf{f} \\ 0 \end{bmatrix} - A \begin{bmatrix} \mathbf{u}_n \\ p_n \end{bmatrix} \right)$$

where A is defined in (3.1).

The error for this method (when $\alpha = 1$) is

$$E_{SIMPLE} = A - \tilde{A}_{SIMPLE} = \begin{bmatrix} 0 & B^T - F(\text{diag}(F))^{-1}B^T \\ 0 & 0 \end{bmatrix}.$$

SIMPLE does not affect the terms that operate on the velocity, but it perturbs the pressure operator in the momentum equation. This results in a method that is "mass preserving." When $\text{diag}(F)^{-1}$ is a good approximation to F^{-1}, then E_{SIMPLE} is close to a zero matrix, so this method generates a very close approximation to the original Jacobian system. From our computational experiments in MpSalsa, we have found that the diagonal approximation can yield poor results because the diagonal approximation does not capture enough information about the convection operator.

3.1.1.2 The SIMPLEC Preconditioner

The SIMPLEC [63] algorithm is a variant of SIMPLE [42]. It replaces the diagonal approximation of the inverse of F with the diagonal matrix whose entries contain the absolute value of the row sums of F. The matrix structure is the same $(LD)U$ as that of SIMPLE. The symbol $\sum(|F|)$ denotes a matrix whose entries are equal to the absolute value of the row sum of F. With the choices $H_1 = F^{-1}$, $H_2 = (\sum |F|)^{-1}$, and $\hat{S} = C + \hat{B}(\sum |F|)^{-1}B^T$ the SIMPLEC method can be expressed in

terms of the block factorization (3.6). The decomposition is

$$\begin{bmatrix} F & B^T \\ \hat{B} & -C \end{bmatrix} \approx \begin{bmatrix} F & 0 \\ \hat{B} & -\hat{S} \end{bmatrix} \begin{bmatrix} I & (\sum(|F|)^{-1}B^T \\ 0 & \alpha I \end{bmatrix} = \tilde{A}_{SIMPLEC}$$

where $\hat{S} = C + \hat{B}(\sum |F|)^{-1}B^T$ and α is a parameter in $(0, 1]$ that damps the pressure update. The error, for this method is

$$E_{SIMPLEC} = A - \tilde{A}_{SIMPLEC} = \begin{bmatrix} 0 & B^T - F(\sum |F|)^{-1}B^T \\ 0 & -\hat{B}(\sum |F|)^{-1}B^T + \hat{B}F^{-1}B^T \end{bmatrix}.$$

This method perturbs the pressure operator in both the momentum and continuity equations. The choice of the absolute value of the row sum tends to provide a better approximation to the matrix F, therefore reducing the error associated with this method [43]. We have found that this choice works reasonably well and is easy to construct. Further variations of this class of methods can be determined by choosing different approximations to F^{-1}, such as sparse approximate inverses. For our computational results, we use the absolute value of the row sum variant.

3.1.1.3 The SIMPLER Preconditioner

The SIMPLER algorithm is very similar to SIMPLE, except that it first determines $\hat{p}_{n+1}$ using $\mathbf{u}_n$, then it calculates an intermediate velocity value, $\mathbf{u}_{n+\frac{1}{2}}$. This intermediate velocity is projected to enforce the continuity equation, which determines $\mathbf{u}_{n+1}$. The steps required are as follows:

1. Solve: $(C + \hat{B}\text{diag}(F)^{-1}B^T)\hat{p}_{n+1} = -\hat{B}\text{diag}(F)^{-1}(f + F\mathbf{u}_n - B^T p_n)$ for the pressure, $\hat{p}_{n+1}$.

2. Solve: $F\mathbf{u}_{n+\frac{1}{2}} = f - B^T(\hat{p}_{n+1} - p_n)$ for the velocity, $\mathbf{u}$.

3. Project $\mathbf{u}_{n+\frac{1}{2}}$ to obtain $\mathbf{u}_{n+1}$ by: $[I+(\text{diag}(F)^{-1})^{-1}\hat{B}(C+B\text{diag}(F)^{-1}B^T)^{-1}B^T)]\mathbf{u}_{n+\frac{1}{2}}$

4. Update the pressure: $p_{n+1} = \alpha\hat{p}_{n+1}$

Once again, α is a parameter in $(0, 1]$ that damps the pressure update. SIMPLER can also be expressed using the LDU framework. The block diagonal (D) and the block upper triangular (U) factors are grouped together and an additonal matrix, P, a projection matrix for the velocity projection in step 3, is added to the factorization. Then the corresponding preconditioner is of the form: $P^{-1}(DU)^{-1}L^{-1}$.

In terms of the taxonomy, this corresponds to the choices of $H_1 = \text{diag }(F)^{-1}$, $H_2 = F^{-1}$, and $\hat{S} = C + \hat{B}(\text{diag}(F))^{-1}B^T$ in (3.7). Then

$$\begin{aligned}\begin{bmatrix} F & B^T \\ \hat{B} & -C \end{bmatrix} &\approx \begin{bmatrix} I & 0 \\ \hat{B}F^{-1} & I \end{bmatrix}\begin{bmatrix} F & B^T \\ 0 & S \end{bmatrix} \\ &\approx \begin{bmatrix} I & 0 \\ \hat{B}(\text{diag}(F))^{-1} & I \end{bmatrix}\begin{bmatrix} F & B^T \\ 0 & -\hat{S} \end{bmatrix}\end{aligned}$$

where $\hat{S} = C + \hat{B}(\text{diag}(F))^{-1}B^T$. Now, the projection matrix is added to give the SIMPLER algorithm in matrix form. This results in

$$\tilde{A}_{SIMPLER} = \begin{bmatrix} I + (\text{diag}(F))^{-1}\hat{B}\hat{S}^{-1}B^T & 0 \\ 0 & \alpha I \end{bmatrix}\begin{bmatrix} I & 0 \\ \hat{B}(\text{diag}(F))^{-1} & I \end{bmatrix}\begin{bmatrix} F & B^T \\ 0 & -\hat{S} \end{bmatrix} \tag{3.8}$$

[43]. Thus, one iteration of SIMPLER corresponds to

$$\begin{bmatrix} \mathbf{u}_{n+1} \\ p_{n+1} \end{bmatrix} = \begin{bmatrix} \mathbf{u}_n \\ p_n \end{bmatrix} + \tilde{A}^{-1}_{SIMPLER}\left(\begin{bmatrix} \mathbf{f} \\ 0 \end{bmatrix} - A\begin{bmatrix} \mathbf{u}_n \\ p_n \end{bmatrix}\right)$$

where A is defined in and $\tilde{A}_{SIMPLER}$ is defined in (3.8). The use of the projection matrix, which has subsidiary solves that must be performed to very high accuracy, greatly degrades the performance of this method when compared to SIMPLE. However, the projection matrix is needed to enforce the continuity equation, and therefore produce a solution that is divergence free [43]. This method perturbs the pressure operator in both the momentum and continuity equations.

3.1.1.4 Remarks on Pressure Correction Methods

In this section, the pressure correction methods (SIMPLE/SIMPLEC) that begin with the underlying factorization, $(LD)U$ and use approximations to the components of the factors to define the preconditioner have been given. SIMPLER is based on the decomposition $L(DU)$ with approximations to $P^{-1}(DU)^{-1}L^{-1}$ as the preconditioner, where P is the projection operator defined in step 3 of the SIMPLER method. These methods are useful for steady state flow problems. However, these methods tend to converge slowly and require the user to input a relaxation parameter to improve convergence.

3.1.2 Approximate Commutator Methods

The pressure convection-diffusion preconditioners, groups together the diagonal and upper triangular factors and omit the lower triangular factor. Let $H_1 =$

$H_2 = F^{-1}$, then the block factorization of the coefficient matrix is

$$\begin{pmatrix} F & B^T \\ \hat{B} & -C \end{pmatrix} = \begin{pmatrix} I & 0 \\ \hat{B}H_1 & I \end{pmatrix} \begin{pmatrix} F & FH_2B^T \\ 0 & -S \end{pmatrix} = \begin{pmatrix} I & 0 \\ \hat{B}F^{-1} & I \end{pmatrix} \begin{pmatrix} F & B^T \\ 0 & -S \end{pmatrix}. \tag{3.9}$$

where the diagonal (D) and upper triangular (U) factors are grouped together. For our computations, we only use the upper triangular factor, and replace the Schur complement S by some approximation $\hat{S}$ (to be specified later). The efficacy of this strategy can be seen by analyzing the following generalized eigenvalue problem:

$$\begin{pmatrix} F & B^T \\ \hat{B} & -C \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \lambda \begin{pmatrix} F & B^T \\ 0 & \hat{S} \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix}.$$

If $\hat{S}$ is the Schur complement, then all the eigenvalues of the preconditioned matrix are identically one. Further, this operator contains Jordan blocks of dimension at most 2, and consequently at most two iterations of a preconditioned GMRES iteration would be needed to solve the system [40].

We motivate the Approximate Commutator Methods by examining the computational issues associated with applying this preconditioner $\mathcal{Q}$ in a Krylov subspace iteration. At each step, the application of $\mathcal{Q}^{-1}$ to a vector is needed. By expressing this operation in factored form,

$$\begin{pmatrix} F & B^T \\ 0 & -S \end{pmatrix}^{-1} = \begin{pmatrix} F^{-1} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} I & -B^T \\ 0 & I \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & -S^{-1} \end{pmatrix}$$

two potentially difficult operations can be seen: S^{-1} must be applied to a vector in the discrete pressure space, and F^{-1} must be applied to a vector in the discrete velocity space. The application of F^{-1} can be performed relatively cheaply using

an iterative technique, such as multigrid. However applying S^{-1} to a vector is too expensive. An effective preconditioner can be built by replacing this operation with an inexpensive approximation. We discuss three preconditioning strategies, the pressure convection-diffusion (P-CD), the Least Squares Commutator (LSC), and the approximate scaled commutator (ASC).

3.1.2.1 The Pressure Convection-Diffusion Preconditioner

Pressure convection-diffusion preconditioners take a fundamentally different approach to approximate the inverse Schur complement. The basic idea hints on the notion of an approximate commutator. To understand this, consider a discrete version of the convection-diffusion operator

$$(\nu\nabla^2 + (\mathbf{w} \cdot \text{grad})). \tag{3.10}$$

where $\mathbf{w}$ is a constant vector. When w is an approximation to the velocity obtained from the previous nonlinear step, (3.10) can be viewed as an Oseen linearization of the nonlinear term in (2.1). Suppose that there is an analogous operator defined on the pressure space,

$$(\nu\nabla^2 + (\mathbf{w} \cdot \text{grad}))_p.$$

Consider the commutator of these operators with the gradient:

$$\epsilon = (\nu\nabla^2 + (\mathbf{w} \cdot \text{grad}))\nabla - \nabla(\nu\nabla^2 + \ (\mathbf{w} \cdot \text{grad}))_p. \tag{3.11}$$

Supposing that ϵ is small, multiplication on both sides of (3.11) by the divergence operator gives

$$\nabla^2(\nu\nabla^2 + (\mathbf{w}\cdot\text{grad}))_p^{-1} \approx \nabla\cdot(\nu\nabla^2 + \ (\mathbf{w}\cdot\text{grad}))^{-1}\nabla \tag{3.12}$$

In discrete form, using finite elements this usually takes the form

$$(Q_p^{-1}A_p)(Q_p^{-1}F_p)^{-1} \approx (Q_p^{-1}B)(Q_v^{-1}F)^{-1}(Q_v^{-1}B^T)$$

$$A_pF_p^{-1} \approx Q_p^{-1}(BF^{-1}B^T)$$

where here F represents a discrete convection-diffusion operator on the velocity space, F_p is the discrete convection-diffusion operator on the pressure space, A_p is a discrete Laplacian operator, Q_v the velocity mass matrix, and Q_p is the lumped pressure mass matrix. This suggests the approximation for the Schur complement

$$S \approx \hat{S} = A_pF_p^{-1}Q_p \tag{3.13}$$

for a stable finite element discretization when $C = 0$. In the case of pressure stabilized finite element discretizations, the same type of approximation is required [17]:

$$S = C + \hat{B}F^{-1}B^T \approx A_pF_p^{-1}Q_p. \tag{3.14}$$

Applying the action of the inverse of $A_pF_p^{-1}Q_p$ to a vector requires solving a system of equations with a discrete Laplacian operator, then multiplication by the matrix F_p, and solving a system of equations with the pressure mass matrix. The convection-diffusion-like system, F, and the Laplace system, A_p, can be handled using multigrid with little deterioration of effectiveness.

In terms of the taxonomy, the pressure convection-diffusion method is generated by grouping together the upper triangular and diagonal factors, choosing $H_2 = F^{-1}$ and $\hat{S}$ as in (3.14). In matrix form this is

$$\tilde{A}_{PCD} = \begin{bmatrix} F & FH_2B^T \\ 0 & -\hat{S} \end{bmatrix} = \begin{bmatrix} F & B^T \\ 0 & -A_pF_p^{-1}Q_p. \end{bmatrix}.$$

The error matrix is

$$E_{PCD} = A - \tilde{A}_{PCD} = \begin{bmatrix} 0 & 0 \\ 0 & -A_pF_p^{-1}Q_p + C - \hat{B}F^{-1}B^T \end{bmatrix},$$

which shows that the momentum equation is unperturbed and only the pressure operator in the continuity equation is perturbed by this method, thus giving a "momentum preserving" strategy.

Considerable empirical evidence for two and three-dimensional problems indicates that this preconditioning strategy is effective, leading to convergence rates that are independent of mesh size and mildly dependent on Reynolds numbers for steady flow problems [18, 23, 34, 53]. A proof that convergence rates are independent of the mesh is given in [38]. One drawback is the requirement that the matrix F_p be constructed. There might be situations where a developer of a solver does not have access to the code that would be needed to construct F_p. This issue is addressed in the next section.

3.1.2.2 The Least Squares Commutator Preconditioner

The Least Squares Commutator (LSC) method automatically generates an F_p matrix by solving the normal equations associated with a certain least squares problem derived from the commutator [19]. This approach leads to the following definition of F_p:

$$F_p = Q_p(\hat{B}Q_v^{-1}B^T)^{-1}(\hat{B}Q_v^{-1}FQ_v^{-1}B^T). \tag{3.15}$$

Substitution of the operator into (3.14) generates an approximation to the Schur complement for div-stable finite element discretizations (i.e. $C = 0$):

$$\hat{B}F^{-1}B^T \approx (\hat{B}Q_v^{-1}B^T)^{-1}(\hat{B}Q_v^{-1}FQ_v^{-1}B^T)^{-1}(BQ_v^{-1}B^T). \tag{3.16}$$

For stabilized finite element discretizations, this can be modified to

$$\hat{B}F^{-1}B^T \approx (\hat{B}Q_v^{-1}B^T + \gamma C)^{-1}(\hat{B}Q_v^{-1}FQ_v^{-1}B^T)(\hat{B}Q_v^{-1}B^T + \gamma C)^{-1} + \alpha D^{-1} \tag{3.17}$$

where α, and β are scaling factors, and D is the diagonal of $(\hat{B}\text{diag}(F)^{-1}B^T + C)$ [20]. For a further discussion of the merits of this method including heuristics for generating α and β, see [20].

In terms of the taxonomy, the LSC is generated by grouping together the upper triangular and diagonal factors, choosing $H_2 = F^{-1}$ and $\hat{S}$ as in (3.17). In matrix form this is

$$\begin{aligned} A_{LSC} = = & \begin{bmatrix} F & FH_2B^T \\ 0 & -\hat{S} \end{bmatrix} \\ = & \begin{bmatrix} F & B^T \\ 0 & (\hat{B}Q_v^{-1}B^T + \gamma C)(\hat{B}Q_v^{-1}FQ_v^{-1}B^T)^{-1}(\hat{B}Q_v^{-1}B^T + \gamma C) + \alpha D \end{bmatrix}. \end{aligned}$$

The error matrix is

$$E_{LSC} = A - \tilde{A}_{LSC}$$
$$= \begin{bmatrix} 0 & 0 \\ 0 & (\hat{B}Q_v^{-1}B^T + \gamma C)(\hat{B}Q_v^{-1}FQ_v^{-1}B^T)^{-1}(\hat{B}Q_v^{-1}B^T + \gamma C) + \alpha D - C - \hat{B}F^{-1}B^T \end{bmatrix},$$

so that the momentum equation is again unperturbed. Empirical evidence indicates that this strategy is effective, leading to convergence rates that are mildly dependent on Reynolds numbers for steady flow problems.

3.1.2.3 The Approximate SIMPLE Commutator Preconditioner

In this section, we define an alternative strategy that uses the same factors as SIMPLE, together with the commutator used to derive the P-CD and LSC factorizations. This results in a "mass preserving" strategy. In terms of the taxonomy, this method is generated by grouping together the lower triangular and diagonal factors, choosing $H_1 = F^{-1}$ and $\hat{S} = C + \hat{B}\text{diag}(F)^{-1}B^T F_p^{-1}$. Insertion of the choices into (3.7) leads to

$$A_{ASC} = \begin{bmatrix} F & B^T \\ \hat{B} & -C \end{bmatrix} = \begin{bmatrix} F & 0 \\ \hat{B}H_1F & -\hat{S} \end{bmatrix} \begin{bmatrix} I & H_2B^T \\ 0 & I \end{bmatrix}$$
$$= \begin{bmatrix} F & 0 \\ \hat{B} & -(C + \hat{B}\text{diag}(F)^{-1}B^T F_p^{-1}) \end{bmatrix} \begin{bmatrix} I & H_2B^T \\ 0 & I \end{bmatrix}.$$

We can approximate the $H_2 B^T$ term in the upper triangular factor by $\text{diag}(F)^{-1} B^T F_p^{-1}$. In matrix form this becomes

$$A_{ASC} = \begin{bmatrix} F & B^T \\ \hat{B} & -C \end{bmatrix} = \begin{bmatrix} F & 0 \\ \hat{B} & -(C + \hat{B}\text{diag}(F)^{-1} B^T F_p^{-1}) \end{bmatrix} \begin{bmatrix} I & \text{diag}(F)^{-1} B^T F_p^{-1} \\ 0 & I \end{bmatrix}.$$

The error matrix is

$$\begin{aligned} E_{ASC} &= A - \tilde{A}_{ASC} \\ &= \begin{bmatrix} 0 & B^T - F\text{diag}(F)^{-1} B^T F_p^{-1} \\ 0 & 0 \end{bmatrix}. \end{aligned}$$

Here the continuity equation is unperturbed. This method performs well when the error in the $(1, 2)$ block is small. More details on the method with a further discussion of how this method compares to SIMPLE can be found in [19].

Chapter 4

Numerical Results in MpSalsa

4.1 Introduction

For our computational study, we have focused our efforts on steady solutions of two benchmark problems, the lid driven cavity problem and flow over an obstruction, each posed in both two and three spatial dimensions. We have tested the methods discussed above using MPSalsa [49], a code that models chemically reactive, incompressible fluids, developed at Sandia National Laboratory. The discretization of the Navier-Stokes equations provided by MPSalsa is a pressure stabilized, streamline upwinded Petrov Galerkin finite element scheme [57] with Q_1-Q_1 elements. In Section 4.2, we describe these benchmark problems. In Section 4.4, we discuss the CPU timings and iteration counts for these benchmark problems. In Section 4.5, we discuss a few of the optimization steps we took to get efficient parallel results.

4.2 Benchmark Problems

4.2.1 Driven Cavity Problem

For the two-dimensional driven cavity, we consider a square region with unit length sides. Velocities are zero on all edges except the top (the lid), which has a driving horizontal velocity of one. For the three-dimensional driven cavity, the

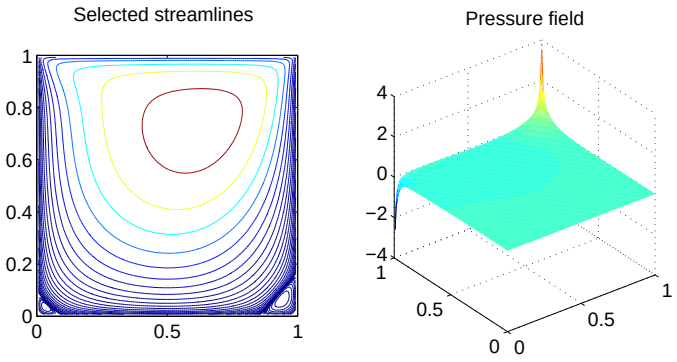

Figure 4.1: Sample velocity field and pressure field from a 2D lid driven cavity. $h = 1/128$, $Re = 100$.

domain is a cube with unit length sides. Velocities are zero on all faces of the cube, except the top (lid), which has a driving velocity of one. Each of these problems is then discretized on a uniform mesh of width h. In two dimensions, we have approximately $3/h^2$ unknowns, i.e. $1/h^2$ pressure and $2/h^2$ velocity unknowns. In three dimensions, we have approximately $4/h^3$ unknowns.

The lid driven cavity is a well-known benchmark for fluids problems because it contains many features of harder flows, such as recirculations. The lid driven cavity poses challenges to both linear and nonlinear solvers and exhibits unsteady solutions and multiple solutions at high Reynolds numbers. In two dimensions, unsteady solutions appear around Reynolds number 8000 [25]. In three dimensions, unsteady solutions appear around Reynolds number 100 [52]. Figure 4.1 shows the velocity field and pressure field for an example solution to a two-dimensional lid driven cavity problem with $h = 1/128$.

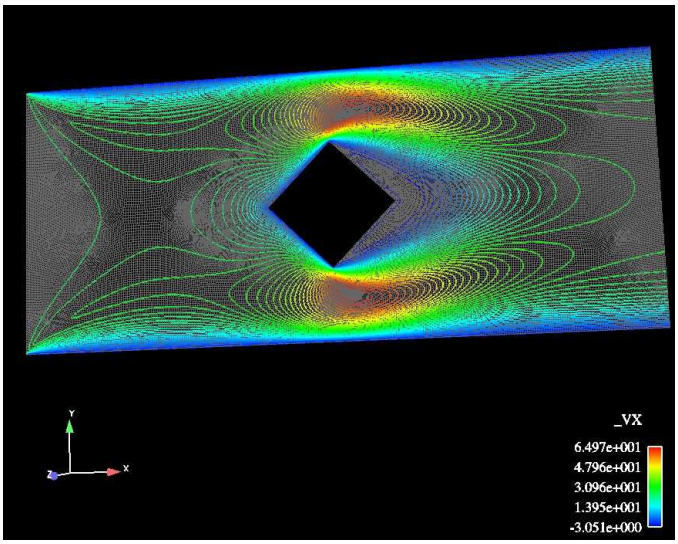

Figure 4.2: Sample velocity field from a 2D flow over a diamond obstruction. 62K unknowns, $Re = 25$.

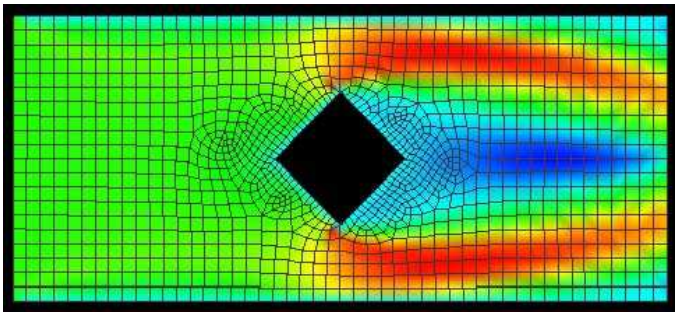

Figure 4.3: Sample velocity field and unstructured mesh from a 2D flow over a diamond obstruction.

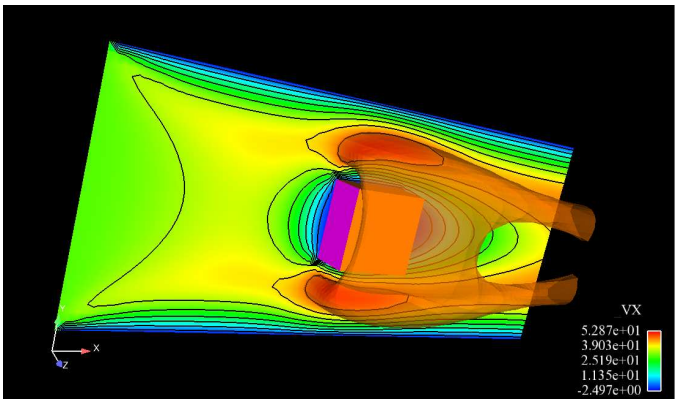

Figure 4.4: Sample contour plot and isosurface from a 3D flow over a cube obstruction, $Re = 50$.

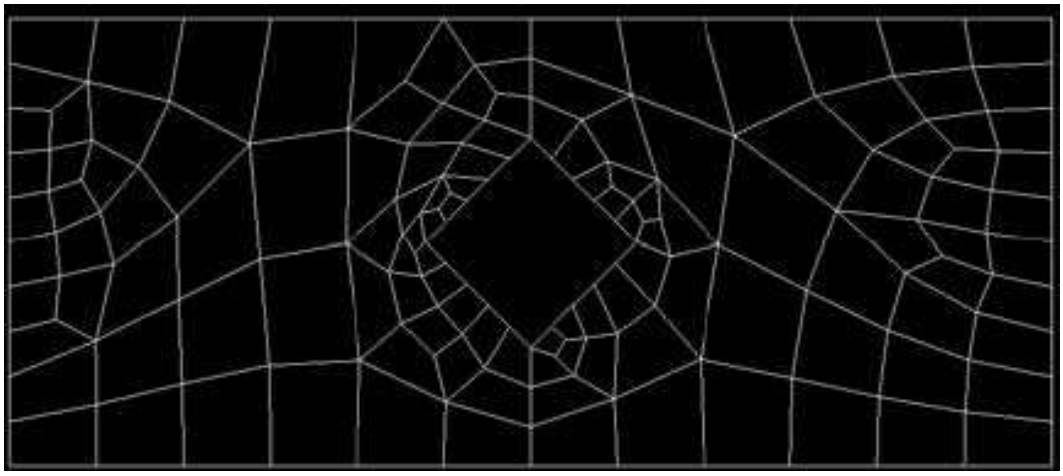

Figure 4.5: Sample unstructured mesh from a 2D flow over a diamond obstruction. 62K unknowns, $Re = 25$.

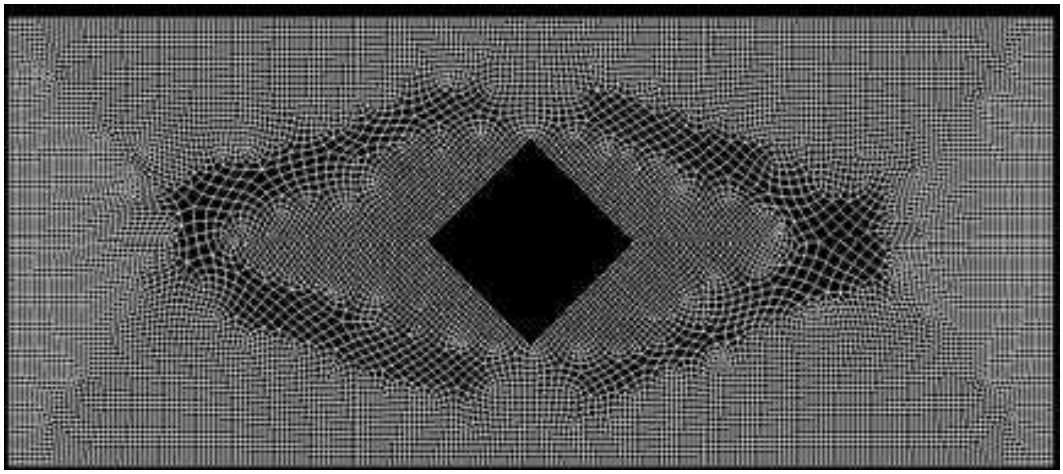

Figure 4.6: Sample refined unstructured from a 2D flow over a diamond obstruction. 1M unknowns, $Re = 25$.

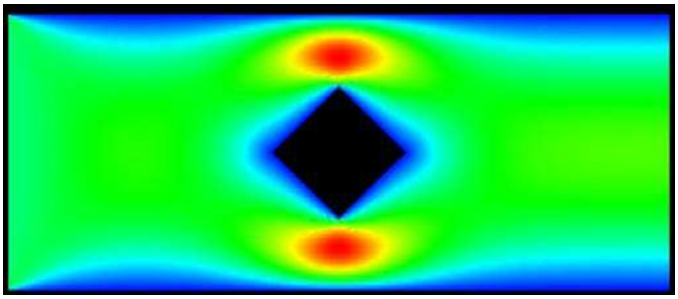

Figure 4.7: Sample velocity streamlines from a 2D flow over a diamond obstruction. 62K unknowns, $Re = 25$.

4.2.2 Flow over an Obstruction

For the two-dimensional flow over a diamond obstruction, we consider a rectangular region with width of unit length and a channel length of seven units, where the fluid flows in one side of a channel, then around the obstruction and out the other end of the channel. Velocities are zero along the top and bottom of the channel and along the obstruction. The flow is set with a parabolic inflow condition, i.e. $\mathbf{u}_x = 1 - y^2, \mathbf{u}_y = 0$ and a natural outflow condition, i.e. $\frac{\partial u_x}{\partial x} = p$ and $\frac{\partial u_y}{\partial x} = 0$.

For the three-dimensional flow over a cube, we consider a rectangular region with a width of one and a half units, a height of three units, and a channel length of five units. The fluid flows in one side of the channel, then around the cube, and out the other end of the channel. Velocities are zero along the top and bottom of the channel, and along the obstruction. The flow is set with a parabolic inflow condition similiar to the two-dimensional case and with a natural outflow condition.

The flow over an obstruction also poses many difficulties for both linear and nonlinear solvers. This problem contains an unstructured mesh with inflow and outflow conditions which generates a different flow than the enclosed flow of a driven cavity. In two dimensions, unsteady solutions appear around Reynolds number 50 [24]. Figure 4.2 and Figure 4.3 shows the velocity field and unstructured mesh for an example solution to a two-dimensional flow over a diamond obstruction for Re 25. Figure 4.4 shows the velocity field and mesh for an example solution to a three-dimensional flow over a cube obstruction for Re 50.

4.3 Implementation Environment

We have tested the methods discussed above using MPSalsa [49], a code that models chemically reactive, incompressible fluids, developed at Sandia National Laboratory. The discretization of the Navier-Stokes equations provided by MPSalsa is a pressure stabilized, streamline upwinded Petrov Galerkin finite element scheme [57] with Q_1-Q_1 elements. One advantage of equal order interpolants is that the velocity and pressure degrees of freedom are defined at the same grid points, so the same interpolants for both velocity and pressure are used.

4.3.1 Problem and Preconditioner Structure

The nonlinear system is solved by Newton's method where the structure of a two-dimensional steady version of F is a 2×2 block matrix consisting of a discrete version of the operator

$$\begin{pmatrix} -\nu\Delta + u^{(n-1)} \cdot \nabla + (u_1^{(n-1)})_x & (u_1^{(n-1)})_y \\ (u_2^{(n-1)})_x & -\nu\Delta + u^{(n-1)} \cdot \nabla + (u_2^{(n-1)})_y \end{pmatrix}. \tag{4.1}$$

For the pressure convection-diffusion preconditioning strategy, we need to specify the operators F_p, A_p, and Q_p. These operators, are generated using the application code, MPSalsa. For the A_p operator required by this strategy, we choose it by taking $1/\nu$ times the symmetric part of F_p. This generates a Laplacian type operator suitable for the use in this preconditioning strategy. For Q_p, we use a lumped version of the pressure mass matrix. For problems with inflow boundary conditions, we specify Dirichlet boundary conditions on the inflow boundary for all of the preconditioning

operators [17]. For singular operators found in problems with enclosed flow, the hydrostatic pressure makes B^T and the Jacobian system rank-deficient by one. Since we are given a Jacobian matrix from MPSalsa that is "pinned," i.e. a row and column that is causing the rank deficiency is removed, we pin all of the operators in the preconditioner (F_p, A_p, Q_p) as the Jacobian matrix is pinned. The other methods (i.e. SIMPLE, LSC) in this study were built as described in Section 3.1.

One aspect of the block preconditioners discussed here is that they require two subsidiary scalar computations, solutions for the Schur complement approximation and convection-diffusion-like subproblem. Both of these computations are amenable to multigrid methods. We employ smoothed aggregation algebraic multigrid (AMG) for these computations because AMG does not require mesh or geometric information, and thus is attractive for problems posed on complex domains or unstructured meshes. More details on AMG can be found in [61, 64].

4.3.2 Operations Required

Once all of the matrices and matrix-vector products are defined, we can use Trilinos [31], a software environment developed at Sandia National Laboratories to develop parallel solution algorithms using a collection of object-oriented software packages, to solve the incompressible Navier–Stokes equations. We use our block preconditioner with specific choices of linear solvers for the Jacobian system, the convection–diffusion, and Schur complement approximation subproblems.

For solving the system with coefficient matrix F we use GMRES precondi-

tioned with four levels of algebraic multigrid, and for the pressure Poisson problem, we use the conjugate gradient (CG) preconditioned with four levels of algebraic multigrid. For the convection-diffusion problem, a block Gauss Seidel (GS) smoother is used and for the pressure Poisson problem, a multilevel smoother polynomial is used for the smoothing operations [2]. The block GS smoother is a domain-based Gauss Seidel smoother where the diagonal blocks of the matrix (the velocity components) correspond to subdomains, and a traditional point GS sweep occurs in the smoothing step. The local Gauss-Seidel procedure includes a communication step (which updates ghost values around each subdomain's internal boundary) followed by a traditional Gauss-Seidel sweep within the subdomain. For the coarsest level in the multigrid scheme, a direct LU solve was employed. We used the smoothed aggregation multigrid solvers available in Trilinos. To solve the linear problem associated with each Newton iteration, we use GMRESR, a variation on GMRES proposed by van der Vorst and Vuik [62] allowing the preconditioner to vary at each iteration. GMRESR is required because we use a preconditioned Krylov subspace method to generate approximate solutions in the subsidiary computations (pressure Poisson and convection-diffusion-like) of the preconditioner, so the preconditioner is not a fixed linear operator.

In our experiments, we compare methods from pressure correction (SIMPLEC) and approximate commutator (PC-D) with a one-level Schwarz domain decomposition preconditioner [50]. This preconditioner does not vary from iteration to iteration (as the block preconditioners do), so GMRES can be used as the outer solver. Domain decomposition methods are based upon computing approximate solutions

on subdomains. Robustness can be improved by increasing the coupling between processors, thus expanding the original subdomains to include unknowns outside of the processor's assigned nodes. Again, the original Jacobian system matrix is partitioned into subdomains using CHACO, whereas AztecOO is used to implement the one-level Schwarz method and automatically construct the overlapping submatrices. Instead of solving the submatrix systems exactly we use an incomplete factorization technique on each subdomain (processor). For our experiments, we used an ILU with a fill-in of 1.0 and a drop tolerance of 0.0. Therefore, the ILU factors have the same number of nonzeros as the original matrix with no entries dropped. A 2-level or 3-level Schwarz scheme might perform better. However, there are some issues with directly applying a coarsening scheme to the entire Jacobian-system due to the indefinite nature of the system [50].

In order to minimize the CPU time and thus reduce the number of outer iterations, we have found that for the SIMPLEC preconditioner, we could not perform the Schur complement approximation solve and the solve with F as loosely as we did with the pressure convection-diffusion preconditioner. For SIMPLEC, we fix a tolerance of 10^{-5} for the solve with coefficient matrix F in (4.3) and the solve with the Schur complement approximation.

For the pressure convection-diffusion and SIMPLEC preconditioners, we use a Krylov subspace size of 300 and a maximum number of iterations of 900. For the 2D domain decomposition preconditioner, we use a Krylov subspace of 600 and a maximum number of iterations of 1800. For the 3D domain decomposition preconditioner, we use a Krylov subspace of 400 and a maximum number of iterations of

1200. All of these values are chosen to limit the number of restarts needed for the solver, while balancing the memory on the compute node. The results were obtained in parallel on Sandia's Institutional Computing Cluster (ICC). Each of this cluster's compute nodes are dual Intel 3.6 GHz Xenon processors with 2GB of RAM.

4.4 Numerical Results

We terminate the nonlinear iteration when the relative error in the residual is 10^{-4}, i.e.

$$\left\| \begin{pmatrix} \mathbf{f} - (F(\mathbf{u})\mathbf{u} + B^T p) \\ g - (\hat{B}\mathbf{u} - Cp) \end{pmatrix} \right\| \leq 10^{-4} \left\| \begin{pmatrix} \mathbf{f} \\ g \end{pmatrix} \right\|. \tag{4.2}$$

The tolerance η_k for (1.5), the solve with the Jacobian system, is fixed at 10^{-5} with zero initial guess. For all of the problems with the pressure convection-diffusion preconditioner, we employ inexact solves on the subsidiary pressure Poisson type and convection-diffusion subproblems. For solving the system with coefficient matrix A_p, we use six iterations of algebraic multigrid preconditioned CG and for the convection-diffusion-like subproblem, with coefficient matrix F, we fix a tolerance of 10^{-2}, i.e. this iteration is terminated when

$$\|(\mathbf{y} - F\mathbf{u})\| \leq 10^{-2}\|\mathbf{y}\|. \tag{4.3}$$

We compare this method to a one-level overlapping Schwarz domain decomposition preconditioner [51] that uses GMRES to solve the Jacobian system at each step using the same tolerances for the Jacobian system and nonlinear iteration.

4.4.1 Lid Driven Cavity Problem

We first compare the performance of the pressure convection-diffusion preconditioner to the domain decomposition preconditioner on the lid driven cavity problem generated by MPSalsa. In the first column of Table 4.1, we list the Reynolds number followed by three mesh sizes in column two. In columns three, four, and five, we list the total CPU time and the average number of outer linear iterations per Newton step for the pressure convection-diffusion, domain decomposition, and SIMPLEC preconditioners, respectively. For the pressure convection-diffusion preconditioner, we notice iteration counts that are largely independent of mesh size for a given Reynolds number. As the mesh is refined, we do notice an increase in the computational time for a given Reynolds number. This is mostly due to the increasing cost of the coarsest level solve in the multilevel method, which relies on a sparse direct solver. One can control this cost by adding additional levels to the multilevel method or by changing the coarse direct solve to an incomplete LU factorization or iterative solve. The domain decomposition preconditioner does not display mesh independent convergence behavior as the mesh is refined. However, there is much less computational effort involved in one iteration of preconditioning with domain decomposition than in one iteration of preconditioning with pressure convection-diffusion. For the fine meshes, the CPU time for the pressure convection-diffusion preconditioner is four times smaller than for domain decomposition. The SIMPLEC method does not display mesh independent convergence behavior, but it provides solutions in fewer iterations and in less CPU time for finer meshes than the domain

decomposition preconditioner. For large Re, SIMPLEC is sensitive to the damping parameter (described in Section 3.1.1.2) on the pressure update. For the results below, the damping factor was 0.01; for larger values of α the method stagnated. We found SIMPLE to be less effective than SIMPLEC and do not report results for SIMPLE.

For the 3D driven cavity problems in Table 4.3, we find that the pressure convection-diffusion method is faster on larger meshes than the one-level domain decomposition method. The pressure convection-diffusion method again displays iteration counts that are largely independent of the mesh size, and it displays a slight dependence on the Reynolds number. The SIMPLEC method produces iteration counts that are less dependent on the Reynolds number than domain decomposition, but it is competitive and in many cases faster than domain decomposition in terms of CPU time.

The timings for the pressure convection-diffusion (and SIMPLEC) solvers are functions of the costs of the component operations that define them. In particular, as Reynolds number increases, the convection-diffusion-like solve is becoming more expensive, i.e., more steps and therefore more CPU time is needed to reach the stopping tolerance (4.3). In addition, the coarse grid solve in the multigrid iterations, which is a direct LU factorization, increases as the mesh is refined. Solving nonsymmetic problems with algebraic multigrid is an active research topic [4]; if a more effective scalable solver did exist for this subproblem, then the CPU timings would be considerably lower and more scalable.

In the first column of Table 4.2, we list the Reynolds number followed by two

Re Number	Mesh Size	Pressure C-D		SIMPLEC		DD One-level		Procs
		iters	time	iters	time	iters	time	
$Re = 10$	64×64	19.4	17.2	41.8	32.9	79.4	19.4	1
	128×128	21.2	28.4	66.0	78.9	220.6	79.8	4
	256×256	23.0	69.3	104.3	229.2	467.2	619.4	16
	512×512	23.2	257.2	164.0	619.4	1356.8	2901.9	64
$Re = 100$	64×64	35.0	28.7	52.0	50.8	86.5	26.4	1
	128×128	35.9	59.5	71.8	87.9	300.3	130.2	4
	256×256	41.3	102.1	109.8	410.5	528.8	593.1	16
	512×512	41.0	345.7	169.4	941.2	NC	NC	64
$Re = 500$	64×64	73.0	200.5	73.9	206.7	89.7	44.4	1
	128×128	79.1	385.6	107.5	401.2	334.9	215.9	4
	256×256	84.3	607.4	177.6	1600.6	896.1	1592.5	16
	512×512	90.2	1811.1	204.3	4109.2	NC	NC	64
$Re = 1000$	64×64	NC	NC	NC	NC	NC	NC	1
	128×128	126.4	570.9	142.0	1220.4	352.5	275.8	4
	256×256	126.6	1207.6	251.6	3494.2	839.5	2009.6	16
	512×512	143.2	2563.2	401.2	7598.2	NC	NC	64

Table 4.1: Comparison of the iteration counts and CPU time for the pressure convection-diffusion, SIMPLEC, and domain decomposition preconditioners for the 2D lid driven cavity problem. NC stands for no covergence.

mesh sizes in column two. In columns three, four, five, and six, we list the average number of outer linear iterations per Newton step for the pressure convection-diffusion, least squares commutator, SIMPLEC, and SIMPLE preconditioners, respectively. In this table we see that the PC-D and LSC perform similiarly to one other for low Re, but as the Re number increases the LSC iteration counts increase greatly. This is likely due to the preconditioning operator not properly accounting for the stabilization in the finite element discretization. A similiar trend is for SIMPLE and SIMPLEC. For Re 500, SIMPLE does not perform as well as SIMPLEC. As the problem becomes more convective, the diagonal of F does not contain enough information about the physical nature of the problem. Therefore, the diagonal approximation in SIMPLE begins to break down. For the rest of our benchmark problems, we just compare PC-D and SIMPLEC to DD.

4.4.2 Flow over a Diamond Obstruction

The pressure convection-diffusion preconditioner, SIMPLEC, and the domain decomposition preconditioner are compared for the diamond obstruction problem. Many of the trends are similiar to the results from the driven cavity problem, mainly iteration counts that are largely independent of mesh size for a given Reynolds number and an increase in the computational time as the mesh size is refined. The domain decomposition preconditioner does not display mesh independent convergence behavior as the mesh is refined. For Re 10 and Re 25, the pressure convection-diffusion preconditioner was faster in all cases. For Re 40, it was faster for all

Re Number	Mesh Size	Pressure C-D	LSC	SIMPLEC	SIMPLE	Procs
$Re = 10$	64×64	19.4	21.8	41.8	42.3	1
	128×128	21.2	22.6	66.0	66.8	4
$Re = 100$	64×64	35.0	39.2	52.0	57.0	1
	128×128	35.9	40.5	71.8	83.0	4
$Re = 500$	64×64	73.0	115.9	73.9	201.8	1
	128×128	79.1	104.1	107.5	242.7	4

Table 4.2: Comparison of the iteration counts for the pressure convection-diffusion, LSC, SIMPLEC, and SIMPLE preconditioners for the 2D lid driven cavity problem. NC stands for no covergence.

meshes except for the small problems with 62,000 unknowns run on one processor. Note that the GMRES solver preconditioned with domain decomposition stagnated before a solution was found for the problems with 4 million unknowns. The pressure convection-diffusion preconditioner converged without difficulty on this problem. On modest sized problems (those with more than 256K unknowns) where both methods converged, the pressure convection-diffusion preconditioner ranged from 4 to 15 times faster than domain decomposition.

In Table 4.5, we compare the impact of inexact solves of the subsidiary systems required for the pressure convection-diffusion preconditioner. In particular, we look at the "exact" pressure convection-diffusion preconditioner, where we solved the subsidiary systems to a tolerance of 10^{-5}. The exact pc-d preconditioner shows

Re Number	Mesh Size	Pressure C-D		SIMPLEC		One-level DD		Procs
		iters	time	iters	time	iters	time	
$Re = 10$	$32 \times 32 \times 32$	28.0	803.2	30.5	1205.6	67.0	634.6	1
	$64 \times 64 \times 64$	28.4	865.2	50.8	2034.1	159.8	1507.5	8
	$128 \times 128 \times 128$	31.1	1249.0	280.8	12490.5	356.2	4529.3	64
$Re = 50$	$32 \times 32 \times 32$	40.2	946.9	33.3	1302.6	62.2	615.5	1
	$64 \times 64 \times 64$	47.8	1061.6	52.5	2457.6	162.6	1533.2	8
	$128 \times 128 \times 128$	50.1	2101.2	291.2	14987.2	385.5	6460.9	64
$Re = 100$	$32 \times 32 \times 32$	56.0	1232.7	40.8	1884.4	61.7	730.7	1
	$64 \times 64 \times 64$	62.1	1697.8	61.6	3184.4	168.5	2131.6	8
	$128 \times 128 \times 128$	64.2	3019.2	299.1	17184.2	404.6	6953.9	64

Table 4.3: Comparison of the iteration counts and CPU time for the pressure convection-diffusion, SIMPLEC, and domain decomposition preconditioners for the 3D lid driven cavity problem.

iteration counts that are mesh independent and reduce as the mesh is refined, but with increasing CPU cost. However, the exact method is still considerably faster than domain decomposition for this problem. For a user of these methods, we recommend the inexact variant because the iteration counts are nearly independent and require less CPU time.

4.5 Code Optimization

Generating the results found in the above tables took countless trials and considerable optimization steps. Due to space limitations, we comment on only two steps we took to optimize the solver and the CPU times in a high performance computing environment. These include making the algebraic multigrid (AMG) more efficient by implementing a more efficient way of accessing data and the addition of extra memory which helps better control the CPU cost of the coarse grid solve in the convection-diffusion solver.

4.5.1 CSR Matrix Optimization

The sparse Jacobian matrix generated by MpSalsa is initially stored in a sparse matrix format known as variable-block row (VBR). Due to efficiency reasons, we convert these VBR matrices and store them in compressed storage row (CSR) sparse matrix format [15]. This matrix representation format allows more optimized matrix vector products than VBR. Moreover, the CSR format is the most general sparse matrix storage format because it makes no assumptions about the structure of the

Re Number	Unknowns	PC-D		SIMPLEC		DD One-level		Procs
		iters	time	iters	time	iters	time	
$Re = 10$	62K	21.7	138.8	52.8	502.2	110.8	186.6	1
	256K	22.6	192.7	83.6	1203.9	282.6	1054.9	4
	1M	25.6	252.3	130.8	1845.3	890.2	6187.4	16
	4M	29.7	397.5	212.6	5834.6	NC	NC	64
$Re = 25$	62K	34.9	248.0	66.5	760.5	101.7	198.8	1
	256K	40.4	384.6	104.7	1920.3	273.8	1118.6	4
	1M	43.6	445.9	160.8	2985.2	864.5	6226.0	16
	4M	49.1	736.6	402.1	8241.3	NC	NC	64
$Re = 40$	62K	64.6	565.8	74.8	1278.7	70.4	267.2	1
	256K	68.9	975.2	113.6	2718.9	203.9	1269.3	4
	1M	72.7	1039.2	260.9	7535.0	770.0	6933.5	16
	4M	78.3	1528.6	410.1	11992.2	NC	NC	64

Table 4.4: Comparison of the iteration counts and CPU time for the pressure convection-diffusion, SIMPLEC and domain decomposition preconditioners for the 2D flow over a diamond obstruction. NC stands for no convergence.

Re Number	Unknowns	Inexact PC-D		Exact PC-D		DD One-level		Procs
		iters	time	iters	time	iters	time	
$Re = 10$	62K	21.7	138.8	18.7	194.8	110.8	186.6	1
	256K	22.6	192.7	16.8	294.0	282.6	1054.9	4
	1M	25.6	252.3	16.1	406.4	890.2	6187.4	16
	4M	29.7	397.5	14.8	655.8	NC	NC	64
$Re = 25$	62K	34.9	248.0	32.8	695.2	101.7	198.8	1
	256K	40.4	384.6	31.6	621.4	273.8	1118.6	4
	1M	43.6	445.9	28.6	778.8	864.5	6226.0	16
	4M	49.1	736.6	25.3	1312.8	NC	NC	64
$Re = 40$	62K	64.6	565.8	44.4	781.3	70.4	267.2	1
	256K	68.9	975.2	39.2	1116.7	203.9	1269.3	4
	1M	72.7	1039.2	38.7	1352.7	770.0	6933.5	16
	4M	78.3	1528.6	35.2	2280.3	NC	NC	64

Table 4.5: Comparison of the iteration counts and CPU time for the inexact pressure convection-diffusion, exact pressure convection-diffusion and domain decomposition preconditioners for the 2D flow over a diamond obstruction. NC stands for no convergence.

Re Number	Unknowns	PC-D		SIMPLEC		DD One-level		Procs
		iters	time	iters	time	iters	time	
$Re = 10$	270K	20.7	997.7	45.2	1897.1	67.2	859.8	1
	2.1M	21.7	1507.5	79.3	4593.2	151.2	2004.0	8
	16.8M	24.7	1997.7	118.7	19907.1	667.2	20908.0	64
$Re = 50$	270K	35.9	1209.7	49.2	2109.2	69.4	889.2	1
	2.1M	38.7	1797.7	84.9	3201.3	132.4	2676.1	8
	16.8M	44.7	2397.7	140.2	28156.1	637.2	18646.0	64

Table 4.6: Comparison of the iteration counts and CPU time for the pressure convection-diffusion and domain decomposition preconditioners for the flow over a 3D cube. NC stands for no convergence.

matrix. It is prudent in its memory usage because it does not store any unnecessary elements and stores the nonzero entries in contiguous memory locations.

Once the matrix is stored in this manner, we can more efficiently access the data, thus reducing the number of times of accessing the disk when applying the algebraic multilevel multigrid preconditioner. The CSR format resulted in matrix vector products (and smoothing operations) that were approximately 20% more efficient than those implemented in the VBR format. This savings was realized in the smoothing operations and by eliminating the getrow function call, which forced the AMG to create a local copy of the matrix and required many read/write calls to disk to build this local matrix copy. Since the matrix was formatted as a CSR matrix, we could eliminate this procedure by getting the memory address of the first stored matrix entry and then passing through the data in a sequential way since the rows of the matrix are stored in contiguous memory blocks. The following listing shows a sample profile for the code before the change to CSR:

```
percent     time    function call
35.72      79.88    Epetra_CrsMatrix::Multiply
27.87      62.32    Epetra_ML_getrow
16.48      36.86    ML_Smoother_BlockGS
 2.95       6.60    Epetra_CrsGraph::ExtractMyRowView
 2.92       6.53    CSR_matvec
 1.77       3.95    Epetra_BlockMap::GlobalToLocalSetup
 1.62       3.63    ML_get_matrix_row
```

```
1.15        2.57     ML_Cheby
```

In the above listing, the first column represents the percentage of time in that function call, the second column the time in that function, and the last column names the function being profiled. After eliminating the getrow call (which cost 27%), the profile reduced to:

```
 percent     time    function call
  22.12   37.86    ML_Smoother_BlockGS
  14.88   25.33    epetra_dcrsmv_
  14.17   24.29    dgemv_
  9.72    13.88    Epetra_CrsMatrix::GeneralMV
  3.70     3.53    CSR_matvec
  2.32     2.95    Epetra_BlockMap::GlobalToLocalSetup
  1.95     2.77    ML_Cheby
```

Note that the change to CSR format also allowed us use better optimized BLAS2 matrix vector product routines (epetra_dcrsmv_) versus the original Epetra matrix vector products (Epetra_CrsMatrix::Multiply) we were using before [3]. In Figure 4.8, we include a bar chart of the timings of the various components (prolongation, restriction, etc) of the multigrid iteration. For this particular example, one can see that the application of the BlockGS smoother to the finest level of the convection-diffusion solve takes the most computational effort, followed by the application of the MLS smoothing polynomial on the finest level of the A_p solve, and finally the direct LU decomposition in the coarse level solve takes the third largest amount

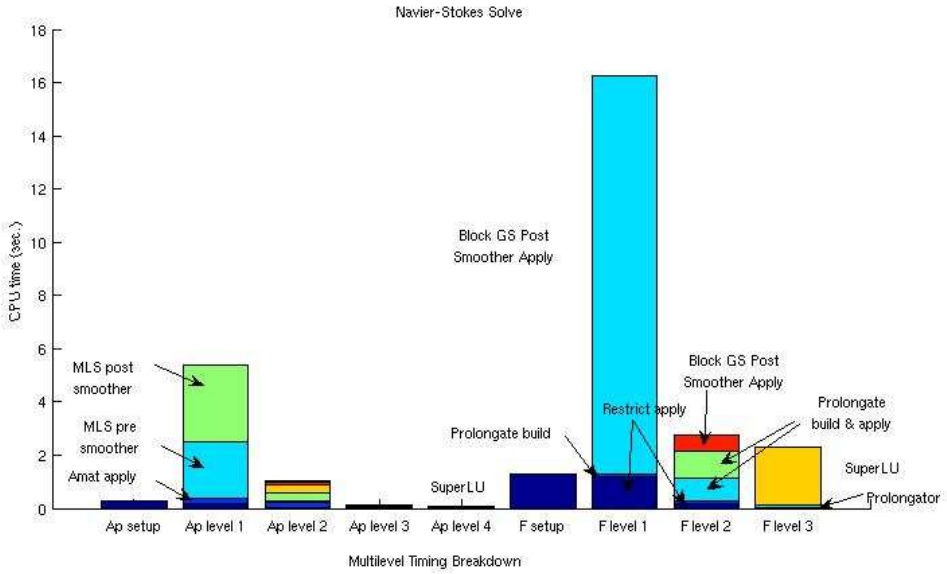

Figure 4.8: Sample velocity field from 2D flow over a diamond obstruction. 62K unknowns, $Re = 25$.

of time. For larger problems, we have seen the cost of the coarse level direct solve increase. We comment more on this point in the next subsection.

4.5.2 Effects of Additional Memory/Cost of LU solve

The cost of the preconditioning operator is a function of the subsidiary computations that define it, such as matrix vector products or algebraic multigrid. For a given simulation, any inefficiency in these subsidiary computations has a great effect on the entire CPU time. We found that it is crucial to keep a close watch on the coarsest level direct solve for the convection-diffusion solver.

We realized great improvements in the CPU time when a mechanism was added to the computing cluster that allowed the simulations to increase memory usage from 1 GB of RAM to 2 GB per node. This reduced memory swapping and

disk input/output[1]. Most importantly this change allowed us to add more levels to the multilevel multigrid iteration, thus reducing the solve time for the convection-diffusion matrix. In Table 4.7, we show the problem size in column 1, followed by the number of processors in column 2, then the number of levels used in the multilevel method for the F solve in column 3, followed by an approximate number of iterations of the F solver in column 4, and finally the cost of the LU time in column 5. As the mesh is refined, one notices an increase not only in the number of iterations, but in the CPU time of the LU solve.

Unknowns	Nprocs	Number of levels	F its	LU time
256K	4	3	7	0.02
1M	16	3	7	0.01
4M	64	3	15	1
256K	4	4	8	0.001
1M	16	4	8	0.004
4M	64	4	11	0.03

Table 4.7: Comparison of CPU times and iterations for three and four levels of Re 25 on the diamond obstruction problem.

We can curb the rise in the CPU time by adding additional levels to the AMG scheme. This is because the size of the coarsest level has drastically reduced with

[1]By invoking a "top" command from a UNIX shell we could see that all of the memory on a compute node was being used in a particular simulation.

the additional level. In Table 4.8, we detail this trend. In this table, we vary the size of the problem and then show the number of unknowns across the columns in each level of the AMG scheme. For example, in a three level AMG scheme on the 64K unknown problem, the finest level has 41,536 unknowns, the second level 2,750, while the coarsest level has 200 unknowns. This 3-level scheme is fine for smaller problems, but in larger problems, say with 4M unknowns, this coarse level matrix is 12,440 by 12,440. Solving a system with this matrix using a direct LU solver is too cumbersome (especially since the coarse level matrix is nearly fully dense). So, adding an additional level (which was not effective before the additional memory was added) reduces the size of the coarse grid operator to 932 by 932. This is much more manageable for a sparse direct solver and is crucial to control the CPU time for this solve. Prior to having 2 GB of memory per compute node, a fourth level did not help the CPU times because the additional matrix operations required by the additional level were too expensive.

Unknowns	L1	L2	L3	L4
64K	41,536	2,750	200	19
256K	164,608	10,586	766	68
1M	655,360	43,691	3,088	274
4M	2,615,296	170,708	12,440	932

Table 4.8: Comparison of the size of the various levels in the AMG solver.

In Table 4.9, we show results for the lid driven cavity problem for 1 GB of

memory per compute node. Contrasting these results with those found in Table 4.1 (which use 2 GB of memory), we notice a substantial decrease in the CPU time by adding the additional memory. We also notice a similiar result for the flow over a diamond problem in Table 4.10 when compared with Table 4.4.

Re Number	Mesh Size	Pressure C-D		SIMPLEC		DD One-level		Procs
		iters	time	iters	time	iters	time	
$Re = 10$	64×64	19.4	18.2	42.0	38.8	79.4	19.4	1
	128×128	21.2	28.4	66.0	98.9	220.6	91.4	4
	256×256	23.0	69.3	104.3	249.2	1018.6	596.6	16
$Re = 100$	64×64	34.2	38.7	52.0	82.7	86.5	26.4	1
	128×128	35.9	69.5	69.8	152.7	300.3	150.2	4
	256×256	40.3	152.1	109.8	365.9	1603.9	1326.6	16
$Re = 500$	64×64	73.0	290.5	74.9	206.7	89.7	44.4	1
	128×128	78.1	428.0	111.4	401.2	334.9	258.9	4
	256×256	83.3	767.4	187.6	1650.1	5433.1	4543.9	16
$Re = 1000$	64×64	NC	NC	NC	NC	NC	NC	1
	128×128	120.4	770.9	152.0	1520.4	352.5	325.5	4
	256×256	124.6	3207.6	261.6	4494.2	4412.7	7497.9	16

Table 4.9: Comparison of the iteration counts and CPU time for the pressure convection-diffusion, SIMPLEC, and domain decomposition preconditioners for the 2D lid driven cavity problem with 1GB of memory per compute node. NC stands for no convergence.

Re Number	Unknowns	PC-D		SIMPLEC		DD One-level		Procs
		iters	time	iters	time	iters	time	
$Re = 10$	62K	20.5	138.8	54.6	502.2	110.8	186.6	1
	256K	22.5	266.2	86.6	1203.9	284.6	1657.4	4
	1M	22.9	501.0	153.0	3513.3	1329.0	7825.5	16
	4M	29.4	1841.7	392.6	48891.7	NC	NC	64
$Re = 25$	62K	32.9	248.0	66.5	760.5	101.7	198.8	1
	256K	35.9	480.6	104.7	1920.3	273.8	1583.1	4
	1M	38.3	956.9	207.3	6259.4	1104.8	7631.5	16
	4M	42.0	4189.8	506.8	78326.5	NC	NC	64
$Re = 40$	62K	54.6	565.8	74.8	1278.7	70.4	267.2	1
	256K	70.1	1280.9	113.6	2718.9	203.9	1420.7	4
	1M	65.4	2011.7	260.9	10356.4	997.1	8188.2	16
	4M	79.8	9387.9	660.6	140412.5	NC	NC	64

Table 4.10: Comparison of the iteration counts and CPU time for the pressure convection-diffusion, SIMPLEC, and domain decomposition preconditioners for the 2D flow over a diamond obstruction with 1GB of memory per compute node. NC stands for no convergence.

Chapter 5

Applications to Shape and Topology Microfluidic Applications

The increased ability to manufacture devices at small length scales has created a growing interest in construction of miniature devices for use in biomedical screening and chemical analysis. At the heart of these devices are flow problems that have a length scale between 10-100 μm, with a low fluid volume, so the Reynolds number is small. This results in laminar flow of the type commonly found in modeling blood samples, bacterial cell suspensions, or protein/antibody solutions. Microfluidics, the methods for controlling and manipulating fluids with length scales less than 1 mm, is a key ingredient in this process [55]. However, robust techniques for pumping and mixing in microfluidic devices are in short supply. Although mixing is one of the most time-consuming steps in biological agent detection, research and development of microfluidic mixing systems is relatively new. In this chapter, we explore the use of techniques for modeling microfluidic mixing systems, and we show that the numerical solution algorithms discussed in Chapter 3 for the discrete Navier-Stokes equations can be used to solve the resulting algebraic systems.

We divide the types of mixing into two broad classes, passive (pressure/capillary) and active (electric/magnetic) mixing. Passive mixing, which occurs when liquids are forced through tortuous paths (baffles, turns, etc), continually dilutes the sample as long as the process continues. Such pressure-driven flows are commonly used

in microfluidic devices and can be very effective when the channel dimensions are not too small ($> 10\mu m$). However, these methods scale poorly with miniaturization and do not offer local control of flow direction [37].

Active mixing does not suffer from these problems because an independent source of motion is used to mix liquids which are otherwise stationary. Strategies for active mixing include production of recirculating flows by ultrasonic means (these tend to be larger, bulky systems) or by oscillatory electro-osmosis [54]. Both of these methods mix dyes in a few seconds, but require specific properties of the reagents. From our prospective, the latter approach, which is denoted Induced Charge Electro-osmosis (ICEO) provides the best option because it can mix dyes in a few seconds, while producing time dependent 2D/3D flows [54].

In the ICEO model, the flow is pumped by electrokinetic means, where an obstruction in a microchannel is charged, generating a varying "double layer" of ions on the walls of the channel. When an electric field is applied to this channel, the double layer moves to the opposite polarity, thus creating motion of fluid near the obstruction [54]. This motion is transferred to the bulk of the fluid, creating mixing and movement of fluid in the channel. ICEO provides a bounty of desirable effects that allow the flow to be tailored to the volume. These include the ability to control the position, shape, and potential of the flow "inducing surfaces" in microchannels. In addition, the flows are confined because they can recirculate within a prescribed volume so the mixing operation does not disperse the sample beyond the intended volume.

ICEO results when an electrical conductor (polarizable material) is placed

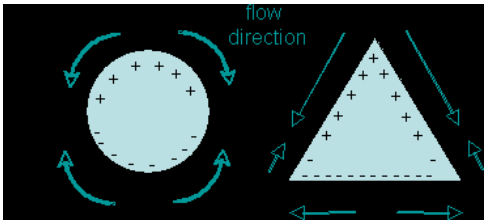

Figure 5.1: Double layer flows around a circular and triangular conductor.

in a liquid (electrolyte) in the presence of an electric field. Consider a cylindrical conductor immersed in a liquid with an electric field present, as shown in Figure 5.1 (left). The conductor is free floating and free of current so it becomes polarized, thus making the field within it zero. The charge on the surface of the conductor attracts counter ions in the surrounding liquid so an electrical double layer is formed adjacent to the conductor surface. The applied field acts on this ionic charge layer, which has been created by the field. The mobile ions move in response to the electric field, and the ions drag the surrounding fluid with them by viscous forces. The resultant "slip" velocity at the surface of the conductor is proportional to the product of the electric field squared and the characteristic length of the conductor [54].

The ICEO flow pattern depends on the shape of the conductor(s). A symmetric shape typically results in symmetric recirculating flows surrounding the conductor. In the case of the cylindrical conductor shown in Figure 5.1, the flow will be composed of four symmetric vortices. There may be many of these conductors resulting in a periodic flow pattern. An asymmetric shape, such as the triangle shown in Figure 5.1 (right), creates non-symmetric flows because it follows the non-symmetric shape of the post.

The two types of ICEO are fixed-charge and fixed-potential. Fixed-charge

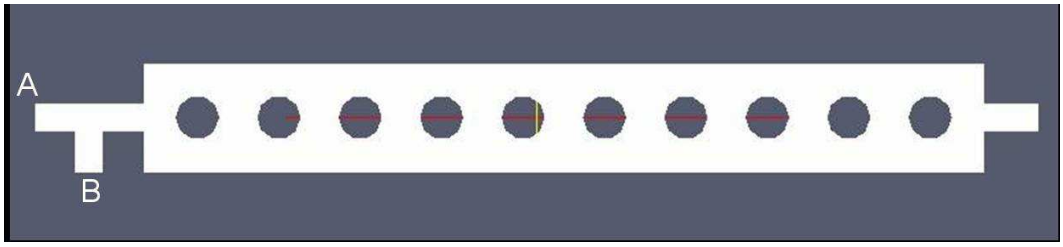

Figure 5.2: Sample domain for the multiple cylinder problem.

ICEO results when the conductor is isolated such that its potential floats, and takes on a value dependent on its location in the field. Fixed-potential ICEO results when the conductor is energized to a prescribed potential. It provides another method for creating non-symmetric flows and it can produce much faster flows than fixed-charge ICEO flows. In Section 5.1 we discuss how we model this scenario.

We would like to design an ICEO driven microfluidic mixing device for combining a sample fluid with a reagent. This device could be useful as part of a miniaturized biological detector. However, the shape and topology of the ICEO conducting region is an open question. Our goal is to investigate this question by running an optimization loop, where a sequence of fluid problems like (3.1) must be solved, to minimize the mixing of two fluids by manipulating the shape and topology of the charged region. Therefore, scalable solvers are needed to effectively solve this optimization problem. Our initial configuration is a rectangular region with 10 circular posts equally spaced in the middle of the rectangle, two inlets to the left of the rectangular region, and one outlet to the right of the rectangular region. An image of this domain can be found in Figure 5.2.

5.1 Model Description

A finite element model was used to calculate the electric field, the ICEO flow, and the mass transport in a multi-species liquid. The low-frequency AC field is assumed to be in a liquid with neutral charge. Under these conditions the electric field is governed by Laplace's equation,

$$\nabla\phi = 0 \tag{5.1}$$

where ϕ is the electric potential. The electric field is obtained from the potential as $E = \nabla\phi$. The boundary conditions for (5.1) are either Neumann conditions (zero normal gradient of the potential) at the channel boundaries and Dirichlet conditions (specified potential) at the electrodes. Note that the insulating boundary condition applied on the surfaces of the posts is the same as that for the remainder of the channel walls. The metallized posts are assumed to be completely shielded from the field by the double layer.

The incompressible form of the Navier-Stokes Equations is used to calculate momentum transport

$$-\nu\nabla^2\mathbf{u} + (\mathbf{u}\cdot\text{grad})\mathbf{u} + \text{grad } p = \mathbf{f} \tag{5.2}$$

$$-\text{div } \mathbf{u} = 0 \tag{5.3}$$

as was defined in Chapter 2. No-slip velocity boundary conditions were used on all channel surfaces except the metallized post surfaces, where a slip velocity boundary condition was used. This is given by

$$v = \frac{\epsilon\delta E_t}{\mu} \tag{5.4}$$

where ϵ is the fluid permittivity, δ is the potential drop across the electrical double layer, E_t is the tangential electric field obtained from solving (5.1), and μ is the fluid viscosity [54].

Once the velocities are obtained from the Navier-Stokes equations, the mass fraction of the solute is obtained by solving the advection-diffusion equation for mass transport,

$$\mathbf{u} \cdot \nabla m = D\nabla^2 m \tag{5.5}$$

where m is the mass fraction of solute and D is the diffusivity. We chose the diffusivity coefficient, D $= 1.8 \times 10^{-9} \frac{cm^2}{s}$, which represents $\sim$ 3 mm particles in an aqueous solution. This value was chosen because particles create the largest challenge to mixing and are a good test case for any mixing device. The small value of D results in a large ($\sim 10^5$) Peclet number, $Pe = uL/D$, where L is the characteristic length scale of the device, indicating that much of the mass transport needed for mixing occurs by advection. For the boundary conditions in this equation, we use Neumann zero flux conditions on the solid surfaces and Dirichlet conditions of 1 on one inflow boundary (denoted A in Figure 5.2) and 0 on the other inflow boundary (denoted B in Figure 5.2) . The mass transport equation in (5.5) is a useful formula to model mixing because it is a mass transfer process that occurs through a combination of convection and diffusion. The fluids of interest here are liquids, where diffusive mass transport is very slow over distances typical of microchannels. Thus, convective transport is needed to stretch and fold the liquids, i.e. to increase interfacial area between the two liquid volumes and to reduce the distances over

which diffusion must occur.

A mixing metric was defined to quantify the extent of mixing based on the calculated results,

$$M = \frac{\int (m - \bar{m})^2 dV}{V} \tag{5.6}$$

where $\bar{m}$ is the average concentration of solute in the liquid mixture and the integral was carried out over the volume, V, of the mixing domain. This metric varies from some initial value that depends on the degree of segregation at the beginning of the mixing process and after the loading process, to zero as perfect mixing is approached.

Our goal is to determine an optimal mixing strategy for this microfluidic problem by varying the shape and orientation of the electrically charged posts. This requires us to solve a series of problems (5.1),(5.2), (5.3), and (5.5) at each step of the optimization loop where we want to

$$\text{minimize} \qquad M \qquad \text{subject to} \tag{5.7}$$

$$d_i \geq 0 \tag{5.8}$$

where d_i is a 38 component design variable related to the shape and orientation of the charged posts and the objective function, M, is the mixing metric defined in (5.6). A further description of the design variables is found in Section 5.4.

5.2 Implementation and Testing Environment

We have modeled the ICEO mixing process using Sundance, a finite element code developed at Sandia National Laboratory [39]. To minimize the objective function we use APPSPACK which is an Asynchronous Parallel Pattern Search code

also developed at Sandia National Laboratory. Both Sundance and APPSPACK are further described in Section 6.3.2.

At each step of the optimization loop we need to perform a series of computations. First we generate a mesh to correspond to the new choice of design variables, then use that mesh to solve a series of problems to model the ICEO flow and the mixing process. We generate the mesh using the software package CUBIT, which is developed at Sandia National Laboratory [1]. The meshes we develop in the course of the optimization loop are triangular elements with an extra level of refinement around the conducting surfaces. This is done to resolve the relevant physics found in the boundary layer. Then we model the ICEO flow, by solving a potential equation,(5.1), which we use to implement a slip velocity boundary condition, (5.4), on the Navier-Stokes equations (5.2)-(5.3). The calculated velocity value from the solution of the Navier-Stokes equations is used in the mass-transport equation, (5.5). The mass fraction, calculated from the mass transport equation, is used to calculate the mixing metric, (5.6), which is the value we want to minimize. These are the major calculations required at each step of the optimization algorithm. In the remainder of this section we describe the discretization details for each equation and our solver choice for each of the discrete systems of equations. We solve the problems using the techniques and software described in the previous chapters, including AMG methods to solve the potential and advection-diffusion equations.

- We discretize the *potential equation* using piecewise quadratic, $P2$, finite elements interpolated with 2^{nd} order Gaussian quadrature. For solving the linear

system resulting from the discretization of the potential equation we use CG preconditioned with two levels of algebraic multigrid. The smoother for this problem is an incomplete LU factorization. We terminate this iteration when the residual is reduced by a factor of 10^{-10}.

- We discretize the *incompressible Navier Stokes equations* using Taylor-Hood $P2-P1$ finite elements with 4^{th} order Gaussian quadrature. This is a stable choice of finite element pairs, so the stabilization matrix, C, is zero. The nonlinear system is solved by Picard's method where the structure of a two-dimensional steady version of F is a 2×2 block matrix consisting of a discrete version of the operator

$$\begin{pmatrix} -\nu\Delta + u^{(n-1)}\cdot\nabla & 0 \\ 0 & -\nu\Delta + u^{(n-1)}\cdot\nabla \end{pmatrix} \tag{5.9}$$

where $u^{(n-1)}$ is a velocity value from a previous iteration. We terminate the nonlinear iteration when the relative error in the residual is 10^{-4}, i.e.

$$\left\| \begin{pmatrix} \mathbf{f} - (F(\mathbf{u})\mathbf{u} + B^T p) \\ g - B\mathbf{u} \end{pmatrix} \right\| \leq 10^{-4} \left\| \begin{pmatrix} \mathbf{f} \\ g \end{pmatrix} \right\|. \tag{5.10}$$

The tolerance η_k for (1.5), the solve with the Oseen system, is fixed at 10^{-5} with zero initial guess. We solve the resulting linear system using GMRES preconditioned with the pressure convection-diffusion preconditioner. We described this method in Chapter 3 and have found this method to work well when tested on some sample fluids problems in Chapter 4. This method is scalable and mesh independent and we wish to show its applicability in a more

applied setting. The F_p, A_p, and Q_p operators required by this strategy are all generated by the application code, Sundance. For the pressure convection-diffusion preconditioner, we solve the subsidiary pressure Poisson type and convection-diffusion subproblems to a tolerance of 10^{-5}, i.e. this iteration is terminated when

$$\|(\mathbf{y} - F\mathbf{u})\| \leq 10^{-5}\|\mathbf{y}\|. \tag{5.11}$$

For solving the system with coefficient matrix F we use GMRES preconditioned with four levels of algebraic multigrid, and for the pressure Poisson problem with coefficient matrix (A_p), we use conjugate gradient (CG) preconditioned with four levels of algebraic multigrid. For the convection-diffusion problem, a block Gauss Seidel (GS) smoother is used and for the pressure Poisson problem, a traditional point GS smoother is used for the smoothing operations. The block GS smoother is a domain-based Gauss Seidel smoother where the diagonal blocks of the matrix (the velocity components) correspond to subdomains, and a traditional point GS sweep occurs in the smoothing step. The local Gauss-Seidel procedure includes a communication step (which updates ghost values around each subdomain's internal boundary) followed by a traditional Gauss-Seidel sweep within the subdomain. For the coarsest level in the multigrid scheme, a direct LU solve was employed. We used the smoothed aggregation multigrid solvers available in Trilinos. To solve the linear problem associated with each Picard iteration, we use GMRES.

For the pressure convection-diffusion solver we use a Krylov subspace size

of 300 and a maximum number of iterations of 600. All of these values are chosen to limit the number of restarts needed for the solver, while balancing the memory on the compute node.

- We discretize the *mass transport equation* (5.5), using $P2$ finite elements with 4^{th} order Gaussian quadrature. For solving (5.5), we use GMRES preconditioned with three levels of smoothed aggregation multigrid. The smoother at the finest two levels is an incomplete LU factorization. On the coarsest level, we use a direct LU solve. We terminate this iteration when the residual is reduced by a factor of 10^{-5}.

- To minimize the objective function found in (5.7), we use APPSPACK, which is an Asynchronous Parallel Pattern Search code developed at Sandia National Laboratory [27]. This code minimizes the objective function by asynchronous parallel generating set search, which handles bound and linear constraints by choosing search directions that conform to a nearby boundary. In generating set search, the next point in the optimization is determined solely by the value of the function on a set of points around the current point. These search points are generated from a fixed set of directions, called the generating set. The evaluation of the function on the search points, or search step, lends itself naturally to a parallel implementation. This code is suited for problems with a small number of design variables (i.e., $n \leq 100$), but expensive objective function evaluations. So this code is well suited for solving this problem.

In Section 5.3.1, we discuss some examples where we tested a microfluidic

flow problem with one charged circular post in the center of the domain. Section 5.3.2 describes an example where we solved the flow problem on multiple cylinders without the mixing coefficients. Finally, Section 5.4 gives more insight into the optimization process and objective function, and shows a few sample meshes and numerical results that were created during the optimization loop. The results were obtained in parallel on Sandia's Institutional Computing Cluster (ICC) using 8 to 100 processors per run. Each of this cluster's compute nodes are dual Intel 3.6 GHz Xenon processors with 2GB of RAM.

5.3 Numerical Results for Microfluidic Cylinders

5.3.1 One Cylinder

In Table 5.1, we display a set of iteration counts for the linearized Navier-Stokes solves for the microfluidic problem with one charged cylinder. These numbers are the average iteration counts for solving the Oseen equations during the course of the Picard iterations. In Figure 5.3 we show an image of an example mesh and in Figure 5.4 we show an image of the refined mesh with $320K$ unknowns. We also show images of a variety of flow field in Figures 5.5, 5.6, and 5.7. In Table 5.1, we list example iteration counts for two meshes. Both of these display iteration counts that are independent of the mesh size for a given Reynolds number. We also see very slight dependence on Reynolds number.

Re Number	Mesh Size	Pressure C-D
$Re = 1$	140K	52.1
	320K	51.2
$Re = 20$	140K	57.2
	320K	56.3

Table 5.1: Average number of linear iterations per nonlinear step for the pressure convection-diffusion preconditioner for the one cylinder microfluidic problem.

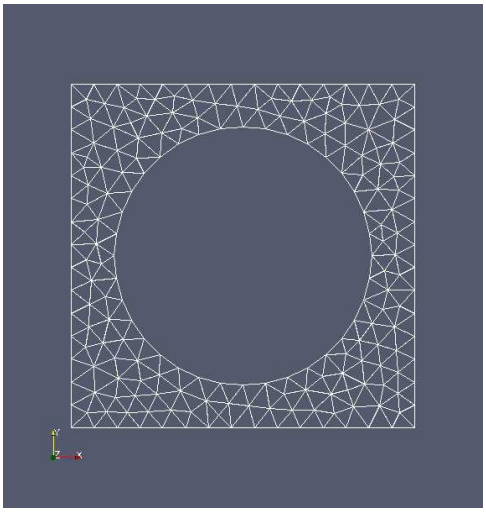

Figure 5.3: Sample coarse mesh for the multiple cylinder domain.

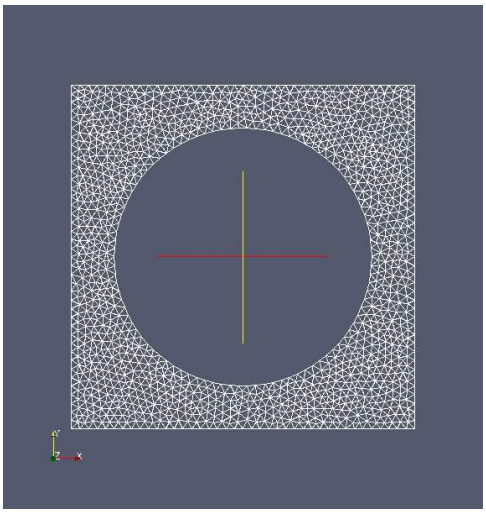

Figure 5.4: Sample refined mesh for one charged cylinder.

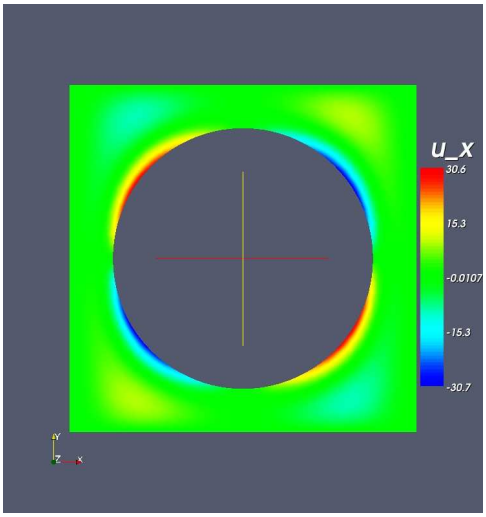

Figure 5.5: Sample u_x velocity field for one charged cylinder.

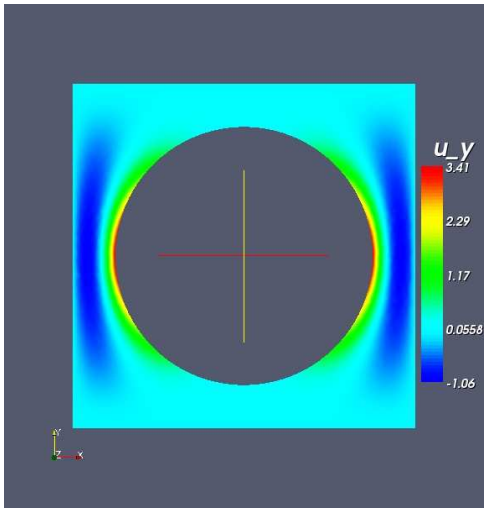

Figure 5.6: Sample u_y velocity field for one charged cylinder.

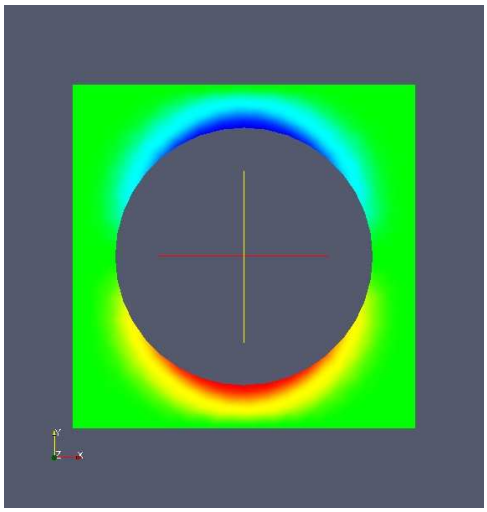

Figure 5.7: Sample higher intensity u_x velocity field for one charged cylinder.

5.3.2 Multiple Cylinders

In Figure 5.8 we show an image of the triangular mesh used to discretize this problem. We also show a few images of a variety of flow fields in Figures 5.9 and 5.10. Note that the flow field is asymmetric around each post, but symmetric with respect to the origin. We see enlarged recirculation zones around the inflow and outflow, which is a result of the boundary effects not be totally resolved. In this table, we have example iteration counts for two meshes. Both of these display iteration counts that are independent of the mesh size for a given Reynolds number. These iteration counts are the average number of linear iterations per nonlinear iteration. We also see very slight dependence on Reynolds number.

Re Number	Mesh Size	Pressure C-D
1	62K	64.0
	256K	62.1
20	62K	68.0
	256K	67.9
50	62K	77.0
	256K	74.9

Table 5.2: Average number of linear iterations per nonlinear step for the pressure convection-diffusion preconditioner for the multiple cylinder microfluidic problem.

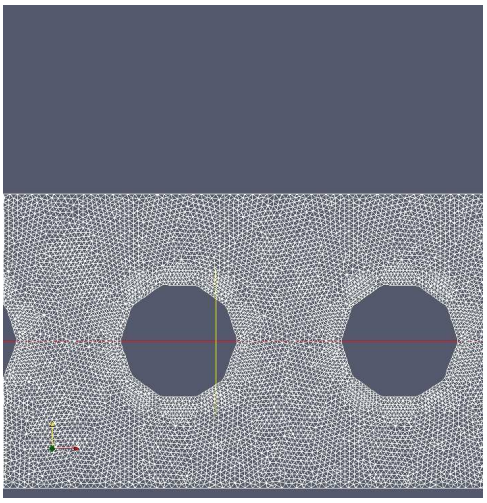

Figure 5.8: Sample mesh for the multiple cylinder domain.

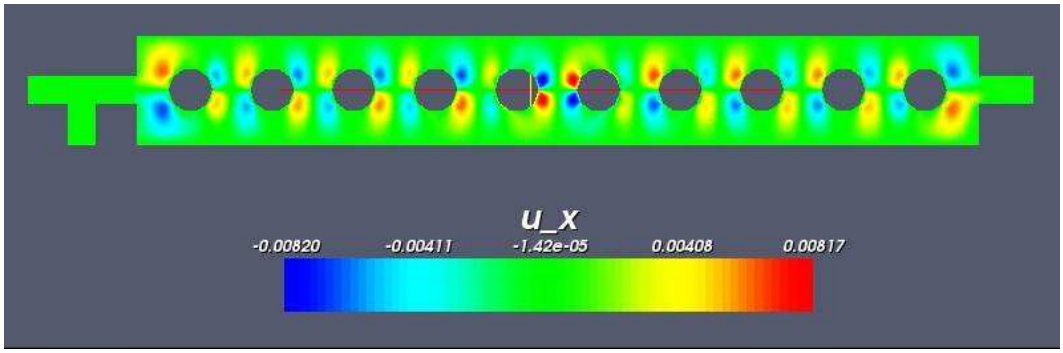

Figure 5.9: Sample velocity field for the multiple cylinder domain.

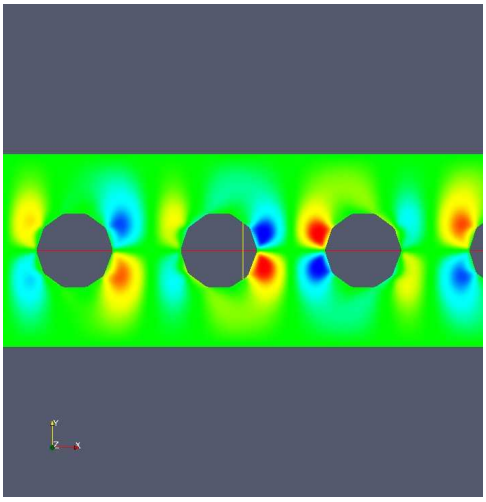

Figure 5.10: Sample velocity field for a zoomed in region of the multiple cylinder domain.

5.4 Numerical Results for Shape Optimization

Our goal is to optimize the shape of the microfluidic mixing device to maximize the amount of mixing being done in the channel. To do so, we use the objective function found in (5.7) constrained to 38 design variables. We parameterize each post as a set of piecewise line segments that connect 10 points. Each of these points is defined using a distance and angle with respect to the origin of our system. This results in 20 design variables. Each of the other 9 posts is offset by a distance from the central post and rotated by an angle, giving 18 more variables. We use the same finite element formulation and solvers as was used in the one and multiple post cases.

5.4.1 Simulation and Numerical Results

In Section 5.2, we described the steps necessary to solve our optimization problem. In this section, we show a variety of flow fields obtained from various steps of the optimization loop and include the value of the mixing metric to show the quality of mixing for that particular mesh. We follow these images with a table listing the iteration counts for solving the ICEO flow problem and the CPU time required to evaluate the mixing metric, i.e. to generate the mesh, then solve (5.1),(5.2)-(5.3), and (5.5). The solver for the Navier-Stokes component of the ICEO flow was GMRES preconditioned with the pressure convection-diffusion preconditioner. This method generated scalable results for the results in Chapter 4 and we see similiar trends when applying this technology to this problem.

For the original configuration of the posts found in Figure 5.9, the initial value of the mixing metric is 0.0287106. We improved on this value by manipulating the posts. In the following figures, we show the flow field at various points of the optimization loop and list the value of the mixing metric in the caption of each figure. Figure 5.11 shows one of the preliminary configurations chosen by the optimization algorithm. Notice that the posts are dimpled. In Figure 5.12 we show a flow field where the fifth and sixth posts have been stretched apart; this resulted in an increase in the mixing metric from the initial value. Due to this increase, the pattern search algorithm tended to stay away from configurations of this type. Figures 5.13, 5.14, and 5.15 show a few sample flow fields where the mixing function value is decreasing, but the obstructions are not aligned for optimal mixing. Figures 5.16 and 5.17 show two configurations for a low mixing metric (values of 0.000811796 and 0.00092394). The value of the mixing metric in these two examples is two orders of magnitude lower than the value of the original mixing metric. We consider this to be an adequate reduction in the cost function. It is interesting to note that the final configurations retained a strong memory to the initial configuration. This suggests that this initial configuration (symmetric circles) leads to a local minimum. In further studies, we hope to change the initial post configuration and see what change (if any) this has on the final post configuration.

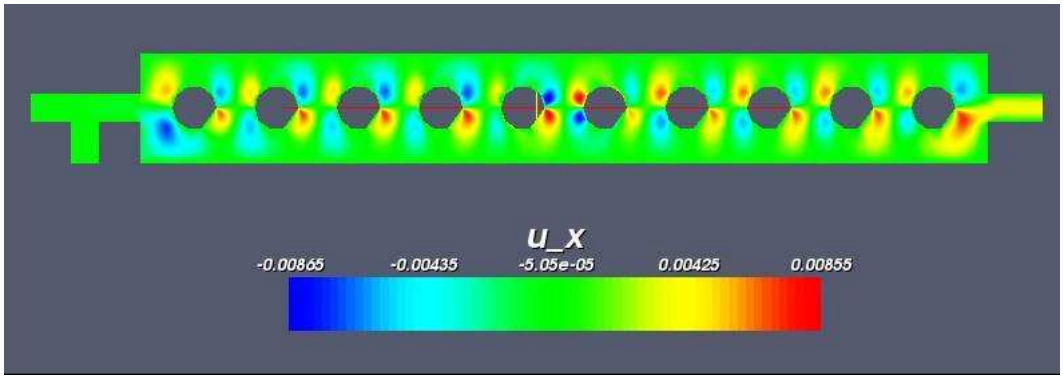

Figure 5.11: Mixing Value: 0.0233216

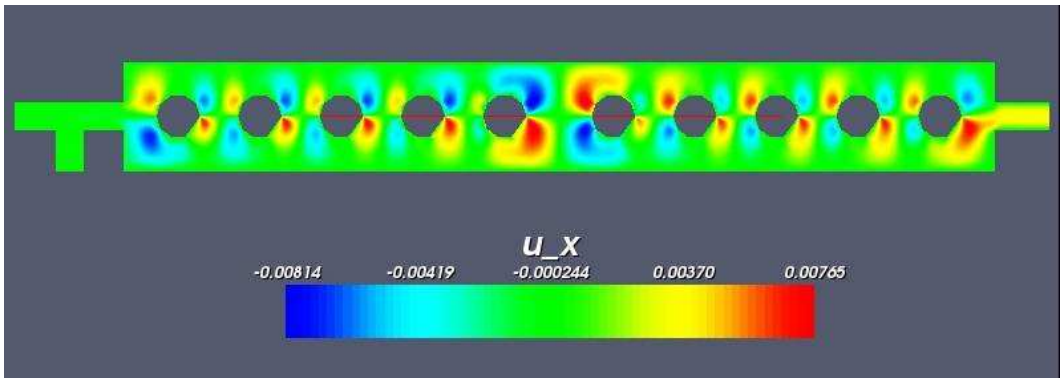

Figure 5.12: Mixing Value: 0.032451

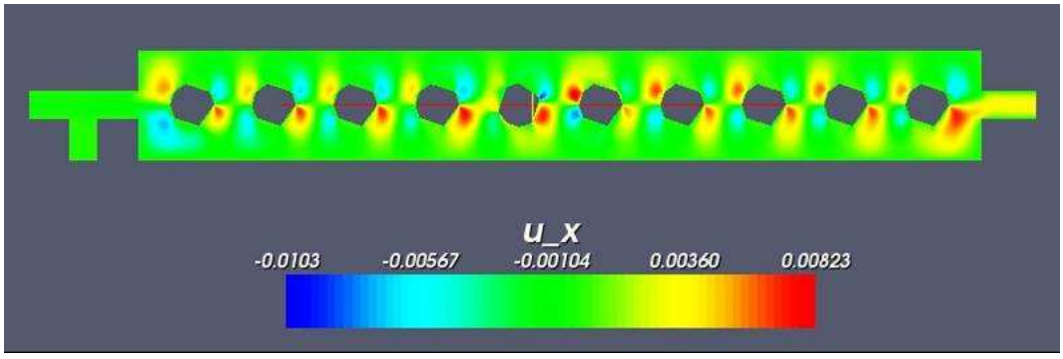

Figure 5.13: Mixing Value: 0.0249871

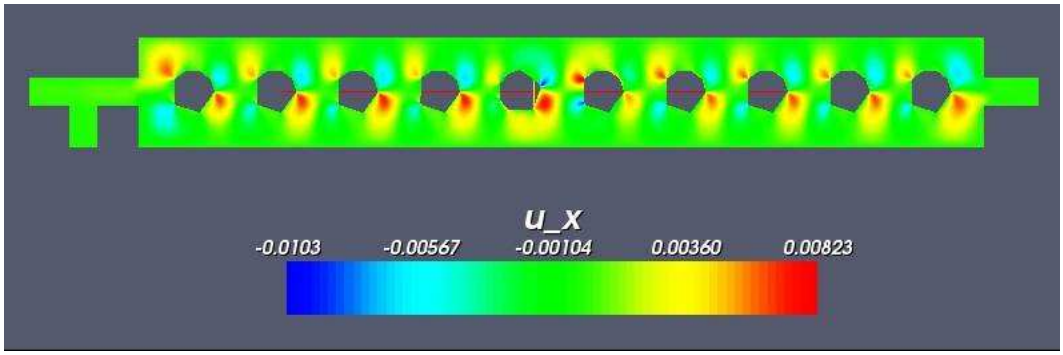

Figure 5.14: Mixing Value: 0.018406

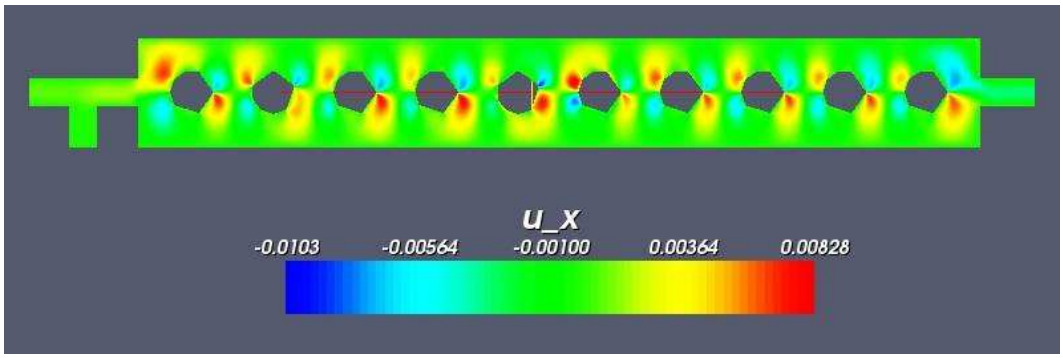

Figure 5.15: Mixing Value: 0.00127773

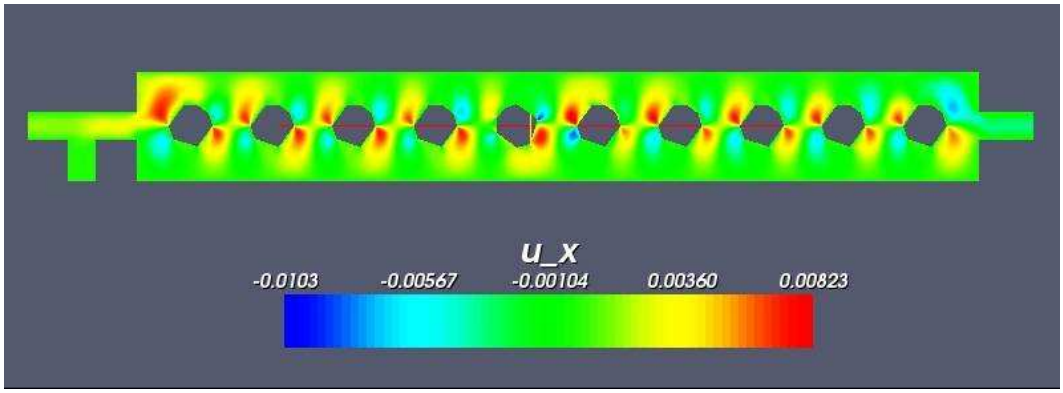

Figure 5.16: Mixing Value: 0.000811796

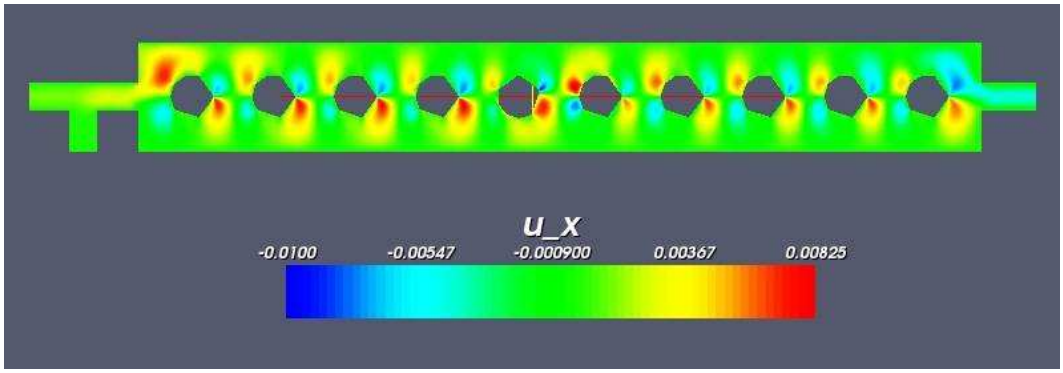

Figure 5.17: Mixing Value: 0.000923394

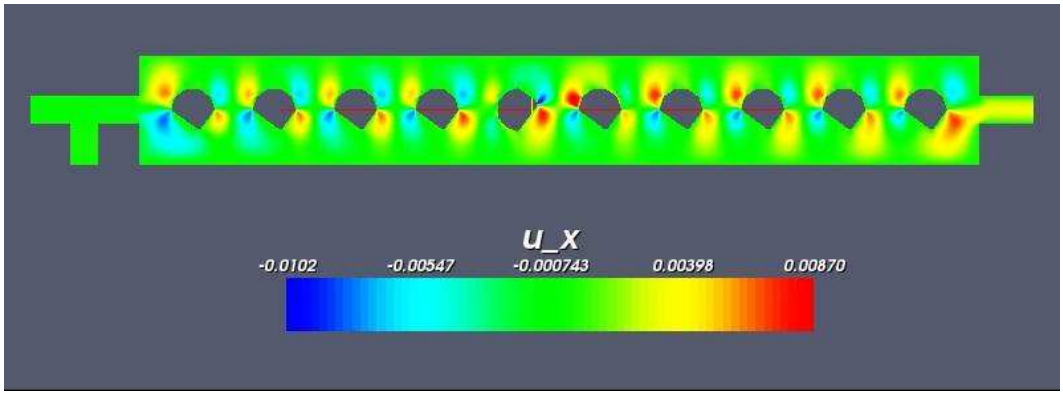

Figure 5.18: Mixing Value: 0.0331203

Figure Number	Iteration Counts	CPU Time (sec)
5.8	64.0	21765.1
5.11	62.1	20831.1
5.12	67.1	21874.1
5.13	66.1	20923.9
5.14	68.2	20643.1
5.15	69.2	20173.8
5.16	60.4	20515.5
5.17	67.3	20488.9
5.18	66.3	20898.2

Table 5.3: Average number of iteration counts per nonlinear step for the pressure convection-diffusion preconditioner for the optimization of a multiple cylinder microfluidic problem.

In Table 5.3, we list the figure number in column one, followed by the average number of outer linear iterations per Picard step for the pressure convection-diffusion preconditioner in column two, and then list the CPU time in seconds in column 3. The iteration counts for this problem are all in the range of 80 to 90 iterations per nonlinear step. This suggests that changes in the obstruction have little effect on the solver for the discrete Navier-Stokes linear systems of equations. The CPU

times are also very consistent from one type of configuration to another. In the future, we plan to experiment with the effect of changing boundary conditions on the preconditioning operator. This change may decrease the number of linear iterations needed to converge to a solution, thus decreasing the overall CPU time for this simulation.

Chapter 6

Implementation and Testing Environment

We have tested the utility of the preconditioning methods described in Chapter 3 in a realistic industrial fluids code. This required the creation of a parallel, object-oriented software package, called Meros. This chapter summarizes the construction and use of this package. Section 6.1 provides an overview of Trilinos, which is the software framework where Meros is embedded. An example problem with Meros is discussed in Section 6.2. Section 6.3 provides an overview of the two application codes we have linked Meros to in order to test the preconditioning strategies. Finally, we conclude with Section 6.4, which lists some of the verification and validation steps that were done to make sure Meros generates accurate solutions and is functioning properly.

6.1 Trilinos Framework

Our implementation of preconditioned Krylov subspace solution algorithms uses *Trilinos* [31], a software environment developed at Sandia National Laboratories to develop parallel solution algorithms using a collection of object-oriented software packages for large-scale, parallel multiphysics simulations. This project is designed to facilitate the design, development and support of mathematical software libraries. One advantage of using Trilinos is its capability to seamlessly use other component

packages for core operations. We use the following components of Trilinos:

1. *Epetra* - This package provides the fundamental construction routines and operations needed for serial and parallel linear algebra libraries. It is one of the base packages in Trilinos. Many Trilinos solver packages use Epetra objects for basic linear algebra computations. Epetra also facilitates matrix construction on parallel distributed machines. Each processor constructs the subset of matrix rows assigned to it via the static domain decomposition partitioning generated by stand-alone library,i.e. CHACO [30], and a local matrix-vector product is defined. Epetra handles all of the details of performing distributed parallel matrix operations (e.g. local indices versus global indices, communication for matrix-vector products, etc.). Once the matrices F, B, $\hat{B}$, and C are defined, a global matrix-vector product for (1.6) is defined using the matrix-vector products for the individual systems. Construction of the preconditioner follows in a similiar fashion.

2. *AztecOO* - This package is a massively parallel iterative solver library for sparse linear systems. AztecOO is a collection of C++ classes that support the construction and use of objects for solving linear systems of equations of the form $Ax = b$ via preconditioned Krylov methods, as provided in Aztec. AztecOO also provides mechanisms for using Ifpack, ML [58] and AztecOO itself as preconditioners. All of the Krylov methods (i.e. those for solving (1.6), for the F, and Schur complement approximation subsystems) are supplied by AztecOO [60].

3. *ML* - This is a multilevel algebraic multgrid preconditioning package. We use this package with AztecOO to solve the F and Schur complement approximation subsystems. ML is designed to solve large sparse linear systems of equations arising primarily from elliptic PDE discretizations. ML based smoothed aggregation preconditioners have been used on thousands of processors for a variety of problems, including the incompressible Navier-Stokes equations with heat and mass transfer, linear and nonlinear elasticity equations, the Maxwell equations, semiconductor equations, and more.

4. *NOX* - This is a package for solving nonlinear systems of equations. We use NOX for the inexact nonlinear Newton solver.

The actual preconditioning strategies are found in Meros, which we discuss next.

6.2 Meros: Software for Block Preconditioning the Navier-Stokes Equations

Meros is a segregated preconditioning package within Trilinos. Meros provides scalable block preconditioning for problems, such as Navier-Stokes problems, that couple simultaneous solution variables . Both the pressure convection-diffusion and variants of the SIMPLE preconditioner detailed in this study are implemented in this package. Meros uses the Epetra package for basic linear algebra functions.

The block preconditioners can be used to solve block linear systems with co-

efficient matrices of the form

$$\tilde{A} = \begin{bmatrix} F & B^T \\ \hat{B} & -C \end{bmatrix} \tag{6.1}$$

where $\tilde{A}$ is a user supplied matrix of size $n \times n$ that arises from linearization and discretization of the incompressible Navier-Stokes equations. We denote F the convection-diffusion-like operator of dimension $v \times v$, B^T the pressure gradient of dimension $v \times p$, $\hat{B}$ the divergence operator of dimension $p \times v$, and C is a stabilization matrix of dimension $p \times p$. Depending on the discretization C might be the zero matrix [17]. Meros is intended to be used on large block sparse linear systems arising from partial differential equation (PDE) discretizations. Meros is designed for linear systems with components that are amenable to solution by multigrid methods (e.g. elliptic PDEs). The motivations for block preconditioning, include

- The desire for the scalability and mesh-independence of multigrid
- Difficulties of applying multigrid to the whole system
- Efficiences achieved by segregating blocks and applying multigrid separately to subproblems

The released version of Meros 1.0 includes the following classes of methods:

- Approximate Commutator Methods
    1. Pressure Convection-Diffusion (PC-D) methods
    2. Least Squares Commutator (LSC) methods
- Pressure-Projection Methods

1. SIMPLE - Semi Implicit Methods for Pressure Linked Equations
2. SIMPLEC - Semi Implicit Methods for Pressure Linked Equations Constrained
3. SIMPLER - Semi Implicit Method for Pressure Linked Equations Revised

Meros has been designed to balance usability with efficiency. Here is an example of the code required to build the block preconditioner:

```
// Build an Fp block preconditioner with Meros
  //
  // | inv(F) 0 | | I  -Bt | | I        |
  // | 0      I | |     I  | |   -inv(X)|
  //
  // where inv(X) = Fp inv(Ap)
  // We'll do this in 4 steps:
  // 1) Build a solver for inv(F)
  // 2) Build a SchurFactory that can make an inv(X) approximation
  // 3) Build a block preconditioner factory with the F solver and
  //    Schur factory
  // 4) Make the preconditioner and get a TSFLinearOperator
  //    representing the prec.


  // 1) Build inv(F) so that it corresponds to using GMRES with ML.
  // Set up an F solver using parameter lists
```

```
string FSolverFile = ‘‘FParams.xml’’;
string path = ‘‘../../example/’’;
ParameterXMLFileReader FReader(path + FSolverFile);
ParameterList FSolverParams = FReader.getParameters();
LinearSolver<double> FSolver =
      LinearSolverBuilder::createSolver(FSolverParams);

// 2) Build a SchurFactory that can make an inv(X) approximation
// First set up an Ap solver using parameter lists

string ApSolverFile = ‘‘ApParams.xml’’;
ParameterXMLFileReader ApReader(path + ApSolverFile);
ParameterList ApSolverParams = ApReader.getParameters();
LinearSolver<double> ApSolver =
      LinearSolverBuilder::createSolver(ApSolverParams);

RefCountPtr<FpSchurFactory> fpschur = rcp(new FpSchurFactory(ApSolver));

// 3) Build a block preconditioner with the F solver and Schur factory

RefCountPtr<PressConvDiffPreconditionerFactory> precfac =
  rcp(new PressConvDiffPreconditionerFactory(FSolver, fpschur));
```

```
// Build an FpOperatorSource to give us access to the saddle

// operator and the Ap and Fp operators.

RefCountPtr<FpOperatorSource> fpOpSrc =

  rcp(new FpOperatorSource(blockOp, Ap, Fp));


// 4) Get the preconditioner.

Preconditioner<double> Prec = precfac->createPreconditioner(fpOpSrc);

LinearOperator<double> saddlePrec = Prec.right();


LinearOperator<double> PA = saddlePrec*blockOp;

// Set up a solver (outer solver)

ParameterList myParams;

myParams.set(``Max Iterations'', 100);

myParams.set(``Restart'', 100);

myParams.set(``Tolerance'', 10e-6);

myParams.set(``Verbosity'', 4);


LinearSolver<double> solver = new GMRESSolver<double>(myParams);

SolverState<double> state = solver.solve(PA,

                                         PA*rhsblockvec,

                                         solnblockvec);
```

6.3 Application Codes

6.3.1 Testing using MpSalsa

We have tested the preconditioning methods discussed in Chapter 3 using MPSalsa [49], a code developed at Sandia National Laboratory, that models chemically reactive, incompressible fluids. The discretization of the Navier-Stokes equations provided by MPSalsa is a pressure stabilized, streamwise upwinded Petrov-Galerkin least squares finite element scheme [57] with Q_1-Q_1 elements. One advantage of equal order interpolants is that the velocity and pressure degrees of freedom are defined at the same grid points, so the same interpolants for both velocity and pressure are used.

6.3.2 Testing using Sundance

We have also tested these methods on solving an optimization problem related to the shape and topology of a microfluidic mixing device. To model this problem, which we describe in Chapter 6, we use Sundance [39] for the finite element discretization and APPSPACK [27] for the optimization. Sundance is a new tool for development of finite-element solutions of partial differential equations developed at Sandia National Laboratory. It is built using an engine for automatic differentiation of symbolic objects, which allows the user to enable differentiable simulations for use in optimization problems. The motivation behind the development of Sundance is the belief that a user should be able to code a finite element problem using the same level of abstraction as one would use to describe the problem in a classroom

setting or in a book. Sundance provides a set of high-level components with which the user can setup, describe, and solve a problem without worrying about book-keeping details. This approach allows a high degree of flexibility in the formulation, discretization, and solution of a problem [39].

For the optimization loop, where we want to determine the optimal mixing strategy for a microfluidic device by manipulating the shape of the obstruction we use APPSPACK, which is an Asynchronous Parallel Pattern Search code developed at Sandia National Laboratory. APPSPACK is a parallel, derivative-free optimization software package for solving nonlinear unconstrained, bound-constrained, and linearly-constrained optimization problems, with possibly noisy and expensive objective functions. To find a solution of this optimization problem, APPSPACK implements asynchronous parallel generating set search, which handles bound and linear constraints by choosing search directions that conform to the nearby boundary. In generating set search, the next point in the optimization is determined solely by the value of the function on a set of points around the current point. These search points are generated from a fixed set of directions, called the generating set. The basic optimization problem is of the form

$$\begin{aligned} \min \quad & f(x) \\ \text{subject to} \quad & c_L \leq A_I x \leq c_u \\ & A_E x = b \\ & l \leq x \leq u \end{aligned}$$

where $f(x)$ is the objective function, the inequality constraints are denoted by the

matrix A_I and the upper and lower bounds by c_L and c_U respectively. The equality constraints are denoted by the matrix A_E and the right hand side, B . Finally, l and u denote the lower and upper bounds [27].

APPSPACK is written in C++ and uses MPI for parallelism. Our approach for using APPSPACK to solve optimization problems is that only function values are required for the optimization, so it can be applied easily. We have a small number of design variables (i.e., $n \leq 100$), but expensive objective function evaluations. Parallelism is achieved by assigning the individual function evaluations to different processors. The asynchronism enables better load balancing.

6.4 Verification and Validation

We have performed many steps to ensure that Meros is implemented correctly and that the solvers are functioning in the correct manner. Software Verification and Validation (V&V) is the process of ensuring that software being developed or changed will satisfy functional and other requirements (validation) and each step in the process of building the software yields the right products (verification). This is important because it indicates whether or not the codes are solving the governing equations correctly [48].

We designed Meros to fit into Sandia's Trilinos framework [31]. We initially tested Meros on matrices dumped directly from the Matlab package, IFISS [22]. This software generates linear systems arising from finite element discretizations of PDEs that govern diffusion, convection-diffusion, Stokes flow and Navier-Stokes

flow problems. With this package, we compared iteration counts, residuals at each iteration of the solver, and plots of the velocity streamlines. Once we received agreement on these fronts, we considered the code verified.

To test Meros inside of MpSalsa, we benchmarked our solver to the already verified domain decomposition solver in MpSalsa. This allowed us to check the maximum, minimum, and average of the velocity at each nonlinear iteration. It also allowed us to compare the linear system solutions from the Meros solver with the domain decomposition solver. We were able to benchmark the LSC and PCD solvers by making sure their analytical properties, such as mesh independence and slight Re number dependence held.

Chapter 7

Conclusions

In this dissertation, we have shown how some preconditioning strategies for the linearized incompressible Navier-Stokes equations can be used effectively in high-performance computing environments to solve a variety of flow problems in two and three dimensions. We have described a taxonomy for preconditioning techniques which includes traditional methods of pressure projection and pressure correction type along with newer approximate commutator methods derived from an approximation of the Schur complement. This taxonomy is based upon a block factorization of the Jacobian matrix in the Newton nonlinear iteration where methods are determined by making choices on the grouping of the block upper, lower, and diagonal factors along with approximations to the action of the inverse of certain operators and the Schur complement. All the methods require solutions of discrete scalar systems of convection diffusion and pressure Poisson-type that are significantly easier to solve than the entire coupled system.

In experiments with these methods using benchmark problems from MPSalsa we have demonstrated that the pressure convection-diffusion method gives superior iteration counts and CPU times for 2D and 3D problems with the one-level additive Schwarz domain decomposition method. For the approximate commutator methods we have demonstrated asymptotic convergence behavior that is essentially

mesh independent in 2D and 3D for problems generated by an application code, MPSalsa, over a range of Reynolds numbers and problems discretized on structured and unstructured meshes with inflow and outflow conditions. For the steady-state problems explored, the iteration counts show only a slight degradation for increasing Reynolds number.

We have also demonstrated the effectiveness of the pressure convection-diffusion method in solving an optimization problem to determine the optimal conducting region of an ICEO driven microfluidic mixing device. The optimization was driven by APPSPACK where the 2D problems were generated by an application code, Sundance. For the approximate commutator methods we have demonstrated asymptotic convergence behavior that is essentially mesh independent in 2D. In the course of the optimization, we were also able to reduce the mixing metric by two orders of magnitude by manipulating the shape of the obstructions. We hope to explore this further by varying the objective function in the optimization algorithm. We also hope to improve the Navier-Stokes solver by exploring the effects of boundary conditions on the preconditioning operator.

Bibliography

[1] *Cubit geometry and mesh generation toolkit*, 2006. Sandia National Laboratory, http://www.cubit.sandia.gov/index.html.

[2] M. Adams, M. Brezina, J. Hu, and R. Tuminaro, *Parallel multigrid smoothing: Polynomial versus Gauss-Seidel*, Journal of Computational Physics, 188 (2003), pp. 593–610.

[3] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK User's Guide*, SIAM, Philadelphia, 1997.

[4] M. Benzi, *Preconditioning techniques for large linear systems: A survey*, Journal of Computational Physics, 182 (2002), pp. 418–477.

[5] M. Benzi and G. H. Golub, *A preconditioner for generalized saddle point problems*, SIAM Journal on Matrix Analysis and its Applications, 26 (2004), pp. 20–41.

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

[7] P. Bochev, M. Gunzberger, and R. Lehoucq, *On stabilized finite element methods for the stokes problem in the small time-step limit*, International Journal for Numerical Methods in Fluids, Volume, 53 (2007), pp. 573–597.

[8] D. Braess, *Finite Elements: Theory, Fast Solvers, and Applications in Solid Mechanics*, Cambridge University Press, 2001.

[9] W. L. Briggs, V. E. Henson, and S. McCormick, *A Multigrid Tutorial, Second Edition*, SIAM, Philadelphia, 2000.

[10] A. N. Brooks and T. Hughes, *Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations*, Computational Methods in Applied Mechanics and Engineering, 32 (1982), pp. 199–259.

[11] R. Burden and J. D. Faires, *Numerical Analysis*, Brooks Cole, 2004.

[12] A. Chorin, *A numerical method for solving incompressible viscous problems*, Journal of Computational Physics, 2, pp. 12–26.

[13] E. Chow and Y. Saad, *Approximate inverse techniques for block-partitioned matrices*, SIAM Journal on Scientific Computing, 18 (1997), pp. 1657–1675.

[14] J. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[15] J. Dongarra, *Lapack working note 50: Distributed sparse data structures for linear algebra operations*, tech. report, Computer Science Department, University of Tennessee, Tech. Rep. CS 92-169.

[16] S. C. Eisenstat and H. F. Walker, *Choosing the forcing terms in an inexact Newton method*, SIAM Journal on Scientific Computing, 17 (1996), pp. 16–32.

[17] H. Elman, D. Silvester, and A. Wathen, *Finite Elements and Fast Iterative Solvers*, Oxford University Press, Oxford, UK, 2005.

[18] H. C. Elman, *Preconditioning for the steady-state Navier–Stokes equations with low viscosity*, SIAM Journal on Scientific Computing, 20 (1999), pp. 1299–1316.

[19] H. C. Elman, V. E. Howle, J. Shadid, R. Shuttleworth, and R. Tuminaro, *Block preconditioners based on approximate commutators*, SIAM Journal on Scientific Computing, 27 (2005), pp. 1651–1668.

[20] H. C. Elman, V. E. Howle, J. Shadid, D. Silvester, and R. Tuminaro, *Least squares preconditioners for stabilized discretizations of the Navier-Stokes equations*, tech. report, University of Maryland, College Park, 2006.

[21] H. C. Elman, V. E. Howle, J. Shadid, and R. Tuminaro, *A parallel block multi-level preconditioner for the 3D incompressible Navier-Stokes equations*, Journal of Computational Physics, 180 (2003).

[22] H. C. Elman, A. Ramage, and D. J. Silvester, *IFISS: A matlab toolbox for modelling incompressible flow*, ACM Transactions on Mathematical Software. To appear (Accepted 2007).

[23] H. C. Elman, D. J. Silvester, and A. J. Wathen, *Performance and analysis of saddle point preconditioners for the discrete steady-state Navier–Stokes equations*, Numerische Mathematik, 90 (2002), pp. 665–688.

[24] B. Fornberg, *Computing incompressible flows past blunt bodies–a historical overview*, Numerical Methods for Fluid Dynamics, IV (1993).

[25] A. Fortin, M. Jardark, J. Gervais, and R. Pierre, *Localization of Hopf bifurcations in fluid flow problems*, International Journal for Numerical Methods in Fluids, 24 (1997), pp. 1185–1210.

[26] V. Girault and P. Raviart, *Finite Element Methods for Navier-Stokes Equations*, Springer, Berlin, 1986.

[27] G. A. Gray and T. G. Kolda, *Algorithm 8xx: APPSPACK 4.0: Asynchronous parallel pattern search for derivative-free optimization*, ACM Transactions on Mathematical Software. To appear (Accepted Nov. 2005).

[28] A. Greenbaum, *Iterative Methods for Solving Linear Systems*, SIAM, Philadelphia, 1997.

[29] M. Griebel, T. Dornseifer, and T. Neunhoeffer, *Numerical Simulation in Fluid Dynamics, a Practical Introduction*, SIAM, Philadelphia, 1998.

[30] B. Hendrickson and R. Leland, *A users guide to Chaco, version 1.0.*, Tech. Report SAND93-2339, Sandia National Laboratories, 1993.

[31] M. A. Heroux, *Trilinos/Petra: linear algebra services package*, Tech. Report SAND2001-1494W, Sandia National Laboratories, 2001.

[32] M. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, Journal of Res. National Bureau of Standards, 49 (1952).

[33] O. Karakashian, *On a Galerkin-Lagrange multiplier method for the stationary Navier-Stokes equations*, SIAM Journal on Numerical Analysis, 19 (January 1982).

[34] D. Kay, D. Loghin, and A. J. Wathen, *A preconditioner for the steady-state Navier–Stokes equations*, SIAM Journal on Scientific Computing, 24 (2002), pp. 237–256.

[35] D. A. Knoll and D. E. Keyes, *Jacobian-free Newton–Krylov methods: a survey of approaches and applications*, Journal of Computational Physics, 193 (2004), pp. 357–397.

[36] C. Lanczos, *Solutions of systems of linear equations by minimized iterations*, Journal of Res. National Bureau of Standards, 49 (1952).

[37] J. Levitan, S. Devasenathipathy, V. Studer, Y. Ben, T. Thorsen, T. Squires, and M. Bazant, *Experimental observation of induced-charge electro-osmosis around a metal wire in a microchannel*, Colloids and Surfaces, 267 (2005), pp. 122–132.

[38] D. Loghin, A. Wathen, and H. Elman, *Preconditioning techniques for Newton's method for the incompressible Navier-Stokes equations*, BIT, 43 (2003), pp. 961–974.

[39] K. Long, *Sundance 2.0*, tech. report, Sandia National Laboratories, SAND2004-4793.

[40] M. F. Murphy, G. H. Golub, and A. J. Wathen, *A note on preconditioning for indefinite linear systems*, SIAM Journal on Scientific Computing, 21 (2000), pp. 1969–1972.

[41] S. V. Patankar, *Numerical heat transfer and fluid flow*, Hemisphere Publishing Corporation, New York, 1980.

[42] S. V. Patankar and D. A., *A calculation procedure for heat, mass and momentum transfer in three dimensional parabolic flows*, International Journal on Heat and Mass Transfer, 15 (1972), pp. 1787–1806.

[43] M. Pernice and M. D. Tocci, *A multigrid- preconditioned Newton–Krylov method for the incompressible Navier–Stokes equations*, SIAM Journal on Scientific Computing, 123 (2001), pp. 398–418.

[44] J. B. Perot, *An analysis of the fractional step method*, Journal of Computational Physics, 108 (1993), pp. 51–58.

[45] A. Quarteroni, F. Saleri, and A. Veneziani, *Factorization methods for the numerical approximation of Navier–Stokes equations*, Computational Methods in Applied Mechanical Engineering, 188 (2000), pp. 505–526.

[46] Y. Saad, *Iterative Methods for Sparse Linear Systems*, PWS Publishing, Boston, 1996.

[47] Y. Saad and M. Schultz, *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems.*, SIAM Journal on Scientific Statistical Computing, 7 (1986), pp. 856–869.

[48] K. Salari and P. Knupp, *Code verification by the method of manufactured solutions*, tech. report, Sandia National Laboratory, 2000.

[49] J. Shadid, A. Salinger, R. Schmidt, T. Smith, S. Hutchinson, G. Hennigan, K. Devine, and H. Moffat., *MPSalsa version 1.5: A finite element computer program for reacting flow problems*, tech. report, Sandia National Laboratories, 1998.

[50] J. Shadid, R. Tuminaro, K. Devine, G. Hennigan, and P. Lin, *Performance of fully-coupled domain decomposition preconditioners for finite element transport/reaction simulations*, Journal of Computational Physics, 205 (2005), pp. 24–47.

[51] ——, *Performance of fully-coupled domain decomposition preconditioners for finite element transport/reaction simulations*, Journal of Computational Physics, 205 (2005), pp. 24–47.

[52] P. N. Shankar and M. D. Deshpande, *Fluid mechanics in the driven cavity*, Annual Review of Fluid Mechanics, 32 (2000), pp. 93–136.

[53] D. Silvester, H. Elman, D. Kay, and A. Wathen, *Efficient preconditioning of the linearized Navier–Stokes equations for incompressible flow*, Journal on Computational and Applied Mathematics, 128 (2001), pp. 261–279.

[54] T. Squires and M. Bazant, *Induced Charge Electro-osmosis*, Journal of Fluid Mechanics, 509 (2004), pp. 217–252.

[55] H. Stone, A. Strock, A. Ajdari, T. Squires, and M. Bazant, *Engineering flows in small devices: microfluidics towards a lab on a chip*, Annual Review of Fluid Mechanics, 36 (January 2004).

[56] T. Tezduyar, *Stabilized finite element formulations for incompressible flow computations*, Advances in Applied Mechanics, 28 (1991).

[57] T. E. Tezduyar, *Stabilized finite element formulations for incompressible flow computations*, Advances in Applied Mechanics, 28 (1991), pp. 1–44.

[58] C. Tong, R. Tuminaro, K. Devine, J. Shadid, and D. Day, *On a multilevel preconditioning module for unstructured mesh Krylov solvers: Two-level Schwarz*, Comm. Num. Meth. Eng., 18 (2002), pp. 363–389.

[59] U. Trottenberg, C. Oosterlee, and A. Schüller, *Multigrid*, Academic Press, London, UK, 2001.

[60] R. Tuminaro, M. Heroux, S. Hutchinson, and J. Shadid, *Official Aztec user's guide: Version 2.1*, Tech. Report Sand99-8801J, Sandia National Laboratories, Albuquerque NM, 87185, Nov 1999.

[61] R. Tuminaro and C. Tong, *Parallel smoothed aggregation multigrid: Aggregation strategies on massively parallel machines*, in SuperComputing 2000 Proceedings, J. Donnelley, ed., 2000.

[62] H. A. van der Vorst and C. Vuik, *GMRESR: a family of nested GMRES methods*, Numerical Linear Algebra with Applications, 1 (1994), pp. 369–386.

[63] J. van Doormaal and G. D. Raithby, *Enhancements of the SIMPLE method for predicting incompressible fluid flows*, Numerical Methods in Heat Transfer, 7 (1984).

[64] P. Vanek, M. Brezina, and J. Mandel, *Convergence of algebraic multigrid based on smoothed aggregation*, Numerische Mathematik, 88 (2001), pp. 559–579.