The Solution of Large Viscoelastic Flow Problems Using Parallel Iterative Techniques

by

Anthony Edward Caola

B.S. in Chemical Engineering, Cum Laude, Washington University (1994)
M.S.C.E.P., Massachusetts Institute of Technology (1996)

Submitted to the Department of Chemical Engineering in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Chemical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September, 2000

© Massachusetts Institute of Technology, 2000. All Rights Reserved.

Author .

Department of Chemical Engineering
August 8, 2000

Certified by .

Robert A. Brown
Warren K. Lewis Professor of Chemical Engineering
Thesis Supervisor

Certified by .

Robert C. Armstrong
Chevron Professor of Chemical Engineering
Thesis Supervisor

Accepted by .

Robert E. Cohen
St. Laurent Professor of Chemical Engineering
Chairman, Committee for Graduate Students
The Solution of Large Viscoelastic Flow Problems Using Parallel Iterative Techniques

by

Anthony E. Caola

Submitted to the Department of Chemical Engineering on August 31, 2000 in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Abstract

Efficient parallel computation of complex viscoelastic flows is essential to bring modern computer power to bear on fluid calculations where complicated constitutive descriptions are required. We have developed an efficient, parallel time integration scheme for calculating viscoelastic flows. The method is a synthesis of an operator splitting and time integration method that decouples the calculation of the polymer portion of the stress by solution of a hyperbolic constitutive equation from the temporal updating of the velocity and pressure fields by solution of a generalized Stokes problem. Finite element discretizations using the DEVSS-G\textsuperscript{1} method are used; here a direct interpolation of the components of the velocity gradient is introduced.

The key to the parallelization is the incorporation of iterative matrix solution for each portion of the split calculation. The linear system that results from the generalized Stokes problem is asymmetric, indefinite, and block singular. At each timestep, this system is solved using an algorithm which combines a parallel preconditioner with a Krylov subspace method. The parallel preconditioner, called the Block Complement and Additive Levels Method (BCALM) preconditioner, is based on treating pressure unknowns separately from the variables velocities and gradients. A pressure preconditioner is constructed from a factor of the Schur complement of the pressures using a Jacobi iteration. The viscous operator is treated using the additive Schwarz method. The number of Krylov iterations required per implicit step with increasing numbers of processors is stabilized using a two-level additive Schwarz method. The coarse grid is generated using addition as a restriction operator and insertion for extension. The resulting iterative method is demonstrated to have high parallel efficiency, subject to effective domain decomposition. Robustness and good performance result from using geometric partitioning of the unknowns by the nested bisection routines supplied in CHACO\textsuperscript{2}. The software is imple-

mented using MPI and the PETSc toolkit\(^1\), so as to be readily portable to a variety of parallel computers.

This solver is developed in the context of the natural convection problem, where in the limit of high Grashof number, the linear system that arises during solution for steady state using Newton’s method is stiff, asymmetric and indefinite. With this model problem, the preconditioner’s robustness is tested and its efficiency is measured. The solver’s generalizability for viscous flow problems is also discussed, as the only assumption about problem physics is that the continuity equation exists.

For a given velocity field, the discretized equations from a typical differential constitutive model, e.g. the Oldroyd-B model, yields an asymmetric, nonsingular system of linear equations when nonlinearities are updated explicitly. Krylov iterations preconditioned with inexact solves in a two-level additive Schwarz framework are used to solve this system at each time step. Added efficiency is introduced by using a discontinuous Galerkin spatial discretization\(^2\), which allows element-by-element condensation of the linear systems and higher parallel efficiency.

The effectiveness of the algorithm is demonstrated by the calculation of transient, two-dimensional flow of an Oldroyd-B fluid past an isolated cylinder confined symmetrically between parallel plates and by calculation of the stability of the steady-state motion to small-amplitude, three-dimensional disturbances. The resulting method is far superior to the use of serial, frontal methods.

Thesis Supervisor: Robert A. Brown  
Title: Provost and Warren K. Lewis Professor of Chemical Engineering

Thesis Supervisor: Robert C. Armstrong  
Title: Chevron Professor of Chemical Engineering

---


Acknowledgments

More than I could have ever anticipated, my time at Tech has been a journey which has been rewarding in many ways. The breadth and depth of both professional and personal growth which I have experienced while here has been truly astonishing to me, particularly in reflection. The rewards have been many, and I thank all of my colleagues and friends who have been beside me on this path.

First, I would like to thank my advisors, Professor Robert Brown and Professor Robert Armstrong, for their input, support and guidance. Upon arriving at MIT, I had only a vague idea of the area in which I wanted to perform research, and I thank both of my advisors for leading me into an area which has been both very interesting and challenging. Both encouraged my independent thought, taught me self-criticality and provided motivation along the way. I feel particularly appreciative for their faith and patience with a project which, by it’s nature, only bore fruit towards the end of my stay here. I am pleased to have provided them with technology which I am confident will have a continuing impact after my departure. Thanks to the NASA Microgravity Science Program and the National Science Foundation for my funding.

Thanks, also, to my committee members, Professors Klavs Jensen, Gareth McKinley, and Einar Ronquist for input to my project, attendance at my meetings and review of the various papers which I have written during the course of this project. Interactions with my committee members helped reaffirm the value of my project in moments of doubt.

The rapidly developing field of parallel computing is still quite collaborative, so there are a number of colleagues who I must thank for their technical input and advice -- many of whom I have never met in person. The developers of the parallel library PETSc have
been very supportive in answering questions, responding to bug reports and providing feedback on my parallel programming technique. In particular, Barry Smith, Satish Balay and William Gropp in the MCS Division at Argonne National Laboratory have always been forthcoming with useful advice and criticism. The useful implementation of the gigabit ethernet drivers for our Pentium cluster would not have been possible without assistance from the authors of the drivers, including Keith Underwood at Clemson University, Eric Kasten at ARQ Consulting and Don Becker at NASA’s Goddard Space Flight Center. The LAM/MPI developers have also provided assistance in understanding how to best utilize their communication protocol. Finally, thanks to Dave Foss, system administrator of the SP2 at MIT, and Chris Hill, the system administrator for Xolas and Pleiades, for quick response to various administrative issues.

My officemates have always been a source of support and diversion during my stay at MIT. Without highly intelligent and fun co-workers such as Mark Smith, Garrett Sin, Yong Joo, Indranil Ghosh, Jason Suen, Howard Covert, Talid Sinno, Aleksei Lipchin, Tatsuuo Mori and Scott Phillips my relatively long stay at MIT would have seem much longer. I thank all of my co-workers for their support, conversation and friendship.

Thanks, too, to the departmental staff who have helped ease the administrative burden and made this a more pleasant place. This includes Arline Benford, Carol Phillips, Janet Fischer, Elaine Aufiero, Patsy Sampson, Emmi Snyder, Peter Maloof and Steve Wetzel.

Of course, thanks also are necessary for all of my friends who are outside my professional circle, but have always been a source of encouragement during my time at MIT. Thanks to my oldest friends, Mike Gill, James Corbett III and Quimby Kaizer for being supportive over these many years. I look forward to your ongoing companionship and
camaraderie. Thanks to Randy Myers his fraternalism and ability to keep me 'real' during the Ph.D. program -- and for my addiction to a weight lifting regimen. Thanks to Mike Mulqueen and Vivek Sujan for the numerous 'spots' and boxing lessons. Thanks to Shannon Mueller for her friendship and encouragement. Her kind words meant more to me than she will ever know and her company made my time in Boston very colorful. Thanks, too to Shahin Ali and John Tolsma for numerous outings and adventures.

Without question, the most loving support has come from my family. I give my love and thanks to my parents, Catherine and Leone Caola, my brother John Caola, and my sisters Mary Caola-Kullman and Kathleen Kerr. Love also to my grandparents, who did not live to see this works’ completion.

Thank you all for everything which you have done for and given to me.
Table of Contents

1 Introduction..................................................................................................................30

1.1 Prototype Viscoelastic Flow Models .....................................................................34

1.2 Mathematical Type of Partial Differential Equations.............................................37

1.3 The Need for A New Solver: Viscoelastic Flow Around an Array of Cylinders...44

1.4 Why Parallel Computing?....................................................................................48

1.5 Possible Solution Methods....................................................................................52

1.6 Basic Parallel Computing Considerations ............................................................60

1.7 Methodology and Outline ....................................................................................64

2 The finite element simulation of viscous flows.........................................................68

2.1 Introduction............................................................................................................68

2.2 Solving Linear Systems of Equations ....................................................................70

2.3 Formulations for the Governing Equations for Viscoelastic Flow .................108

2.4 Finite Element Formulation for DEVSS-G ..........................................................111

2.5 Finite Element Formulation for Natural Convection in a Cavity ....................119

2.6 Time Integration ..................................................................................................125

2.7 Studying the Stability of Viscoelastic Flows .......................................................129

2.8 Newton’s Method ...............................................................................................131

3 Parallel Processing for Finite Element Simulations .................................................134

3.1 Introduction............................................................................................................134

3.2 Parallel Computing Hardware .............................................................................137

3.3 Implementation of a Parallel Solver ....................................................................143
4 A Block Complement and Additive Levels Method (BCALM) Preconditioner for Iterative, Parallel Solution of Finite Element Discretization of Viscous Partial Differential Equations ...............................................................170

4.1 Introduction: Iterative Methods for Viscous Flow and the Natural Convection Problem .........................................................................................................................170

4.2 A Robust Preconditioner: The Block Complement and Additive Levels Method (BCALM) Preconditioner ..............................................................................................................173

4.3 Demonstrating the Preconditioner's Effectiveness and Efficiency ..........181

4.4 Conclusions .........................................................................................................................195

5 The Parallel Simulation of Transient Viscoelastic Flows .........................198

5.1 Introduction .............................................................................................................................198

5.2 A History of Solving Large Viscoelastic Flow Problems .........................211

5.3 The Parallel Solution of Transient Viscoelastic Flow Around a Periodic Array of Cylinders ..........................................................................................................................216

5.4 The parallelization of linear stability analyses .........................................................275

5.5 Conclusions .............................................................................................................................288

6 Discussion .................................................................................................................................292

Bibliography .................................................................................................................................306
List of Figures

Figure 1.1: The effect of deformation history on physical properties. Both strands are made from the same HDPE, but the opaque strand was conventionally extruded, while the other was extrusion drawn. [177] .................................................. 30

Figure 1.2: Flow past a linear, periodic array of cylinders. Geometry is specified by the radius, the spacing, and the channel half-height. [117] .............................................. 44

Figure 1.3: M1: The coarsest mesh considered by Liu, et. al. [117] ............................................. 45

Figure 1.4: In three dimensions, each additional triquadratic brick element introduces eight nodes, shown as the circles and the square on the example structured mesh, above. Trilinear brick elements introduce only one, the square.........46

Figure 1.5: M6: The finest mesh considered in Liu, et. al. [117] .................................................. 46

Figure 1.6: Overall levels of dissection in CFS. First, the physical geometry is dissected into the same number of sub-domains as processors. Then each sub-domain is broken into super-nodes, over which pivoting is done. Finally, the dissection is carried out to the level of individual elements to conserve memory. .......53

Figure 1.7: four-level incomplete nested dissection of a rectangle [128]................................. 55

Figure 1.8: The dissection of a processor's sub-domain into super-nodes and the sparsity of the resulting exterior and interior matrices [129] ................................................. 56

Figure 1.9: Elimination tree [128]. ............................................................................................. 56

Figure 1.10: Communication Bandwidth and Latency using MPI on the SP2 [69]................. 63

Figure 2.1: Pseudocode for pivoting LU-decomposition .......................................................... 71

Figure 2.2: Pseudocode for forward elimination ........................................................................ 71

Figure 2.3: Pseudocode for back substitution .......................................................................... 72
Figure 2.4: Gaussian elimination described as two tasks ........................................73

Figure 2.5: A task graph for Gaussian elimination with N=5 ..................................74

Figure 2.6: Frontal factorization of a band matrix in serial [119] ...............................75

Figure 2.7: Factorization of a domain using a multifrontal nested dissection algorithm. Each of the green areas may be solved in parallel. Parallelism then breaks down as processors must cooperate on the border regions ........................................76

Figure 2.8: Speedups for LU-factorization stage of CFS as a function of the number of unknowns for calculation of natural convection in an enclosed cavity. These speedups are based on computation rate, a metric which increases these values relative to what would be find using speedup based on time for computation [126]. ........................................................................................................78

Figure 2.9: Speedups for the forward elimination stage of CFS as a function of the number of unknowns for calculation of natural convection in an enclosed cavity. These speedups are based on computation rate, a metric which increases these values relative to what would be find using speedup based on time for computation [126]. Note that the addition of processors actually hurts the performance. ........................................................................................................79

Figure 2.10: Speedups for the forward elimination stage of CFS as a function of the number of unknowns for calculation of natural convection in an enclosed cavity. These speedups are based on computation rate, a metric which increases these values relative to what would be find using speedup based on time for computation [126]. Note that the addition of processors actually hurts the performance. ........................................................................................................80
Figure 2.11: Pseudocode for the method of Conjugate Gradients ........................................84

Figure 2.12: Pseudocode for GMRES. Note that this is guaranteed to converge in iterations, and that the update to the solution vector is obtained by a minimizer of the initial error minus .................................................................86

Figure 2.13: Pseudocode for the Bi-Conjugate Gradient (BCG) method .......................87

Figure 2.14: Pseudocode for bi-conjugate gradient stabilized (BiCGSTAB) ...............89

Figure 2.15: ILU(0) factorization of a five-point matrix, A. The sparsity pattern for the L and U are shown to be the same, in sum, as the original matrix. The sparsity pattern for LU is also shown. [154] .................................................................92

Figure 2.16: ILU(1) factorization of a five-point matrix, A. The sparsity pattern for the L and U are shown to be the same, in sum, as the sparsity pattern resulting from LU and ILU(0). The sparsity pattern for LU with one level of fill is also shown, which would determine the sparsity structure of ILU(2). [154] ......93

Figure 2.17: Pseudocode for ILU(0) ........................................................................94

Figure 2.18: Pseudocode for general ILU(p) ............................................................94

Figure 2.19: Schwarz’s original alternating method figure ............................................97

Figure 2.20: Coloring a rectangular domain with four colors .....................................100

Figure 2.21: The coarse grid used for the natural convection problem. Restriction is accomplished by insertion while projection is accomplished by bilinear interpolation. While theoretically sound, this method is difficult or impossible for an unstructured mesh and places demands on mesh generator. Complex data structures are generated for locating nodes on the fine grid in the coarse grid. Furthermore, for a fixed coarse grid this degrades as the fine mesh is decom-
posed. .................................................................104

Figure 2.22: Decomposition of the mesh for flow of fluid around a cylinder for four processors. One equation per variable is written per domain ..................105

Figure 2.23: Splitting of the original linear system into blocks which are summed to generate one equation per processor. For multivariate systems, one equation per variable type per processor is written. ................................................106

Figure 2.24: Schematic representation for natural convection in a fluid filled cavity..120

Figure 2.25: Solutions of the steady-state natural convection problem corresponding to Pr=0.015 and Gr=10,000. Clockwise from upper left, the solutions shown are for u velocity, v velocity, pressure, and temperature for a case of an aspect ratio of 1, a Prandtl number of 0.015, and a Grashof number of 10,000......121

Figure 2.26: Schematic for the splitting of a timestep [nDt,(n+1)Dt] using the theta method .................................................................127

Figure 2.27: Newton’s method used to converge the nonlinear equation set ..........131

Figure 2.28: Steady state solution of the nonlinear operator associated with a finite element discretization by Newton’s method ..............................................132

Figure 3.1: Example computation graph for a threaded code. The dashed arrows represent the fork of a child thread, while the left-to-right solid edges represent a join between parent and child. Vertical arrows represent serial dependencies. 145

Figure 3.2: A parallel scalar-vector product. Each processor holds two rows of the vectors and has the scalar. As a result each processor can perform this operation independently of the others.................................................................147

Figure 3.3: A parallel dot product. Each processor holds two rows of the vectors. Each
processor begins by generating the local dot product from the vector. This is followed by communication between the processors as the result is updated for the off-processor values. This is an example where Gustafson's law clearly applies. The interprocessor communication does not scale with problem size but with the number of processors. As a result each processor can perform this operation independently of the others...

Figure 3.4: A parallel matrix vector product. Each processor may begin working in parallel by multiplying its diagonal block by the local members of its right-hand-sides and storing that part of the solution in the right hand side. The off-diagonal blocks of the matrix, however, require communication either to obtain components of from to assign values to components of...

Figure 3.5: Parallel backwards elimination. This cannot be done with good efficiency as both processors 1 and 2 cannot proceed until processor 3 solves for its local components of. Then, only processor 2 can proceed. This dependency is one reason that parallel direct methods are not highly efficient...

Figure 3.6: Time (T) required for backwards elimination of CFS as a function of number of unknowns (N) for various numbers of processors. [126]

Figure 3.7: Organization of the PETSc library

Figure 3.8: Storage of a 6x6 matrix in compressed sparse row format (CSR)

Figure 3.9: Each processor represents its portion of a parallel matrix as two submatrices. One containing values along the diagonal, the other containing off-diagonal blocks.
Figure 3.10: Schematic for convective flow of a variable density fluid in a closed cavity..............................159

Figure 3.11: Sparsity structure of the Jacobian linear system resulting from the natural ordering and distribution onto 3 processors. Results are for the natural convection problem described in Section 2.5. Off-diagonal blocks are not minimized. ..........................................................160

Figure 3.12: Sparsity structure resulting from banded ordering. While this image is exaggerated, bandwidth minimization does not minimize the number of entries in the off-diagonal blocks. ..................................................161

Figure 3.13: Three possible dissection schemes for a 4x4 mesh......................162

Figure 3.14: First and second vibrational modes of a spring. The second vibrational mode corresponds to the decomposition obtained from spectral dissection. The positive regions would be assigned to one processor, the negative ones to the second. .................................................165

Figure 3.15: Decomposition (a) and subsequent Jacobian sparsity structure (b) resulting from recursive bisection of a cavity onto four processors using spectral nested dissection. .................................................................166

Figure 4.1: Plot showing sublinear relationship between iteration count and number of unknowns .................................................................185

Figure 4.2: Average Iteration Count (AIC) as a function of number of sub-domains for the various discretizations of the natural convection in a cavity problem. For the 16x16 discretization, a 4x4 coarse grid was chosen, while for both the 32x32 and 64x64 discretizations, a 8x8 coarse grid was selected. ..........187
Figure 5.1: Solution fields for Newtonian flow around a single cylinder in a channel with $H=2$. (a) Ten equally spaced contours are shown for the streamlines. Twenty equally spaced contours between the minimum and maximum values are shown for (b) the normal stress and (c) the shear stress. In each plot, the maximum and minimum locations are denoted by an X and a dot, respectively. From [116].

Figure 5.2: Axial velocity along the symmetry line for flow around a single cylinder in a channel with (a) upstream and (b) downstream of the cylinder. Results are for . From [116].

Figure 5.3: Comparisons of (a) axial normal stress and (b) average molecular extension for flow around a single cylinder ($H=2$) at $We=0.5$ for three viscoelastic models. From [116].

Figure 5.4: Flow geometry for the linear stability analysis presented by Öztekin et al. [73] for flow of an Oldroyd-B fluid around cylinder of radius $R$ confined in a concentric porous cylinder of radius $R_w$. A uniform streaming velocity $U$ is enforced at the surface of the porous cylinder. This figure is taken from reference [141].

Figure 5.5: The asymptotic behavior of the velocity gradient near the cylinder surface for flow around an isolated cylinder corresponding to a beta of 0.59 and a lamda of 0.793. The viscometric boundary layer extends approximately 1E-1 away from the cylinder surface for all Weissenberg numbers considered. [105]

Figure 5.6: The asymptotic behavior of the stress field near the cylinder surface for flow around an isolated cylinder corresponding to a beta of 0.59 and a lamda of
0.793. The viscometric boundary layer is closer to then cylinder than observed for the velocity gradient. Furthermore, as the Weissenberg number is increased, the magnitude of the stresses near the surface becomes both larger and closer to the surface. As a result, the viscometric boundary layer must either be completely missed by a mesh or a number of elements must be placed in the layer to resolve it. [105]

Figure 5.7: Schematic diagram for flow past a linear, periodic array of cylinders confined between parallel plates. The cylinders have radius , and the geometry is specified by the cylinder-to-cylinder spacing and the channel half-height . Computations were performed on the whole channel in the unit cell.

Figure 5.8: Oscillations in the adaptive viscosity near the cylinder surface. Similar oscillations were also reported in the stress field. These oscillations result in a steady-state solution which is inappropriate for use in linear stability analysis, and are likely due to the inability of the mesh used to resolve the viscometric boundary layer in the stresses [174].

Figure 5.9: Contours of for flow of an Oldroyd-B fluid around an isolated cylinder. Results were calculated with mesh B for and . Taken from [167].

Figure 5.10: The -component (filled symbols) and the -component (open symbols) of the polymer stress at an angle of 0.034 radians away from the center-plane in the cylinder wake for values of the Weissenberg number of 0.667 (m), 0.75 (r) and 0.833 (d). Taken from [167].

Figure 5.11: The normalized drag force on a cylinder for five mesh sizes. Increasing mesh
refinement results in method failure at successively lower Deborah numbers, yet the method is O(h) accurate. Taken from [53].

Figure 5.12: The highly-resolved mesh used to resolve the stress gradients in the viscometric boundary layer. This mesh has 23,598 elements and 187,816 nodes. Discretization of viscoelastic flow with DEVSS/DG results in 751,110 total unknowns with 281,850 velocity, pressure or gradients unknowns and 469,260 stress unknowns. The detail shows the smallest elements, which are O(1E-3).

Figure 5.13: Contours of trace around the cylinder for flow around a linear array of cylinders (L=18, H=2, R=1). Results are for Oldroyd-B at De=1.0 on a coarse mesh used by Joo [105]. At regions in front of and in the wake of the cylinder, the configuration tensor has lost positive definiteness.

Figure 5.14: Detail of contours of trace in wake of the cylinder for flow around a linear array of cylinders (L=18, H=2, R=1). Results are for Oldroyd-B at De=1.0 on a coarse mesh used by Joo [105]. The negative regions in the wake correspond to regions where and become negative.

Figure 5.15: Detail of contours of trace in front of the cylinder for flow around a linear array of cylinders (L=18, H=2, R=1). Results are for Oldroyd-B at De=1.0 on a coarse mesh used by Joo [105]. The negative regions in the wake correspond to regions where becomes negative.

Figure 5.16: Contours of trace around the cylinder for flow around a linear array of cylinders (L=18, H=2, R=1). Results are for Oldroyd-B at De=1.0 on the present mesh. The configuration tensor is positive definite throughout the flow field.
due to mesh refinement .................................................................................................................. 232

Figure 5.17: Contours of around the cylinder for flow around a linear array of cylinders
(L=18, H=2, R=1). Results are for Oldroyd-B at De=0.5 and De=1.0 on the present mesh. The development of both the birefringent strand in the wake of and the large stress gradients in front of the cylinder may be seen with increasing Deborah number. .................................................................................................................. 233

Figure 5.18: Contours of around the cylinder for flow around a linear array of cylinders
(L=18, H=2, R=1). Results are for Oldroyd-B at De=0.5 and De=1.0 on the present mesh. The development of both the birefringent strand in the wake of and the large stress gradients in front of the cylinder may be seen with increasing Deborah number. .................................................................................................................. 234

Figure 5.19: Contours of pressure around the cylinder for flow around a linear array of cylinders (L=18, H=2, R=1). Results are for Oldroyd-B at De=0.5 and De=1.0 on the present mesh.................................................................................................................. 235

Figure 5.20: The asymptotic behavior of the velocity and velocity gradient in the space between the cylinder and the top of the channel. The flow is near-Newtonian, with linear velocity profile near the cylinder and constant velocity |

|gradient.................................................................................................................. 236

Figure 5.21: The asymptotic behavior of the stress in the space between the cylinder and the top of the channel. The development of a region of increased stress due to the convection of stretched polymers from the front of the cylinder is seen to develop with increasing Deborah number. .................................................................................................................. 237

Figure 5.22: Detail of the asymptotic behavior of the stress in the space between the cyl-
inder and the top of the channel. The region shown is represents where the stresses have reached 98% of their maximum value. Due to secondary elongational effects, this region does not decrease monotonically.  

Figure 5.23: Stress profiles along the centerline in the wake of the cylinder for flow around a linear array of cylinders (L=18, H=2, R=1). Results are for Oldroyd-B at various Deborah numbers on the present mesh. Stress gradients become increasingly steep and the birefringent strand propagates further into the wake for increasing Deborah number.  

Figure 5.24: Velocity profiles along the centerline in the wake of the cylinder for flow around a linear array of cylinders (L=18, H=2, R=1). Results are for Oldroyd-B at various Deborah numbers on the present mesh. Stress gradients become increasingly steep and the birefringent strand propagates further into the wake for increasing Deborah number.  

Figure 5.25: The normalized magnitude of the stresses across the wake of the cylinder at the location of the stress maximum. As the Deborah number increases, the gradients in the stresses become much sharper and the dimension of the wake becomes smaller. For a Deborah number of 1.1, the structure was not resolved.  

Figure 5.26: Results obtained from Dou and Phan-Thien [54] for flow around a cylinder in a channel using DAVSS-Omega. For high Deborah number, the first normal stress difference, , becomes negative due to the stresses becoming aphysical in the wake of the cylinder.  

Figure 5.27: Comparison of the azimuthal velocity dependency of Öztekin et al.’s [141]
similarity solution to the current calculations for \( L = 18 \) and \( De = 0.0 \). The region where Öztekin’s solution is valid near the cylinder decreases in size with increasing \( De \).

Figure 5.28: Comparison of the azimuthal velocity dependency of Öztekin et al.’s [141] similarity solution to the current calculations for \( L = 18 \) and \( De = 0.5 \). The region where Öztekin’s solution is valid near the cylinder decreases in size with increasing \( De \).

Figure 5.29: Comparison of the azimuthal velocity dependency of Öztekin et al.’s [141] similarity solution to the current calculations for \( L = 18 \) and \( De = 0.75 \). The region where Öztekin’s solution is valid near the cylinder decreases in size with increasing \( De \).

Figure 5.30: Comparison of the radial velocity dependency of Öztekin et al.’s [141] similarity solution to the current calculations for \( L = 18 \) and \( De = 0.0 \).

Figure 5.31: Comparison of the radial velocity dependency of Öztekin et al.’s [141] similarity solution to the current calculations for \( L = 18 \) and \( De = 0.0 \).

Figure 5.32: Mesh SM1. This mesh has 1,930 elements and 15,240 nodes. Discretization of viscoelastic flow with DEVSS/DG results in 60,900 total unknowns with 22,990 velocity, pressure or gradients unknowns and 37,910 stress unknowns.

Figure 5.33: Mesh SM2. This mesh has 3,582 elements and 42,774 nodes. Discretization of viscoelastic flow with DEVSS/DG results in 113,464 total unknowns with 42,774 velocity, pressure or gradients unknowns and 70,690 stress unknowns.
Figure 5.34: Mesh SM3. This mesh has 7,122 elements and 56,556 nodes. Discretization of viscoelastic flow with DEVSS/DG results in 226,130 total unknowns with 85,090 velocity, pressure or gradients unknowns and 141,040 stress unknowns. ..............................................................254

Figure 5.35: Mesh SM4. This mesh has 13,273 elements and 105,532 nodes. Discretization of viscoelastic flow with DEVSS/DG results in 422,052 total unknowns with 158,642 velocity, pressure or gradients unknowns and 263,410 stress unknowns. ..............................................................254

Figure 5.36: A comparison of the 2-norm of the deviation of the velocity solution from the parabolic solution that would result in an unobstructed channel. The solution obtained from the serial, frontal code previously used in the group is compared to the solution obtained from the parallel iterative method for the case of a Deborah number of 0.5 and 18. ..................................................256

Figure 5.37: The solution obtained from the serial, frontal code previously used in the group is compared to the solution obtained from the parallel iterative method for the case of a Deborah number of 0.5 and 18. ..................................................257

Figure 5.38: The total number of iterations required for BiCGStab to converge the Stokes-like subproblem in RK4/DG during the first timestep. The iteration count is stable for a range of Deborah numbers much larger than the RK4/DG time-stepping scheme. ..............................................................260

Figure 5.39: The number of iterations required for BiCGStab to converge the Stokes-like subproblem in RK4/DG from an initial guess of zero during the first timestep. The iteration count is stable and for the largest meshes the addition of proces-
sors actually improves convergence. ...........................................261

Figure 5.40: The number of iterations required for BiCGStab to converge the Stokes-like subproblem in RK4/DG using the previous timestep as an initial guess during the second timestep. The iteration count is stable and for the largest meshes the addition of processors actually improves convergence. .................262

Figure 5.41: Detail: The number of iterations required for BiCGStab to converge the Stokes-like subproblem in RK4/DG using the previous timestep as an initial guess during the second timestep for the largest mesh considered in this Section. The iteration count clearly improves convergence in this case. ........264

Figure 5.42: The total number of iterations required for GMRES to converge the Stokes-like subproblem during the first timestep for a Deborah number of 0.5 and for a 16 processor run as a function of drop tolerance. Beyond a critical drop tolerance, the number of iterations required stabilizes. .............................265

Figure 5.43: The number of nonzeros in each sub-block ILU factorization of the viscous operator of the Stokes-like subproblem for a Deborah number of 0.5 and for a 16 processor run. Beyond the same critical drop tolerance where the iteration count stabilizes, the number of nonzeros in the factorization is also flat. For mesh SM1, the “A” subblock corresponding to the viscous operator has 1100901 nonzeros. For SM2, it has 2077361, for SM3 it has 4184115, and for SM4 it has 7858793. .................................................................................266

Figure 5.44: Speedup for the initial factorization of the Stokes-like linear system at a Deborah number of 0.5 using a drop tolerance for each sub-problem of 1E-6. Superlinear speedups are possible at this step due to the scaling of the
subproblem factorization dimension..................................................268

Figure 5.45: Speedup for the subsequent solution of the Stokes-like linear system at a Deborah number of 0.5 using the factorization obtained with drop tolerance for each sub-problem of 1E-6. As there is no “unfair” advantage, the method’s efficiency for a fixed problem size will decrease with the addition of processors..................................................269

Figure 5.46: Speedup for the explicit discontinuous Galerkin update at a Deborah number of 0.5. As the DG update may be accomplished in parallel, the deterioration with the addition of processors is minimal.................................270

Figure 5.47: Speedup through 32 processors for the initial factorization of the Stokes-like linear system at De = 0.5 using a drop tolerance for each sub-problem of 1E-6. Because very little communication is required for this factorization, the efficiencies remain high..................................................271

Figure 5.48: Speedup through 32 processors for the subsequent solution of the Stokes-like linear system at De = 0.5 using the factorization obtained with drop tolerance for each sub-problem of 1E-6. As communication is required in this step, for small problems running multiple processors per box rapidly degrades performance..................................................272

Figure 5.49: Speedup on up to 32 processors for the explicit discontinuous Galerkin update at De = 0.5 using the factorization obtained with drop tolerance for each sub-problem of 1E-6. As the DG update may be accomplished in parallel, the deterioration with the addition of processors is minimal.................................273

Figure 5.50: Speedup for the initial factorization of the Stokes-like linear system for both
the transient code and linear stability code based on the RK4/DG timestepping
method. Runs were performed at $\text{De} = 0.5$ using a drop tolerance for each sub-
problem of $1\text{E}-6$. In both cases, superlinear speedups are possible at this step
due to the scaling of the subproblem factorization dimension. The two cases
for each mesh are virtually indistinguishable due to the similarity of the im-
plementation...........................................................................................................279

Figure 5.51: Speedup for the subsequent solution of the Stokes-like linear system for both
the transient code and linear stability code based on the RK4/DG timestepping
method. Runs were performed at $\text{De} = 0.5$ using the factorization obtained
with a drop tolerance for each sub-problem of $1\text{E}-6$. Because the two prob-
lems are so closely related, it is impossible to distinguish between the tran-
sient runs and the linear stability runs. ...............................................................280

Figure 5.52: Speedup for the explicit discontinuous Galerkin update of the stress field for
both the transient code and linear stability code based on the RK4/DG timestepping method. Runs were performed at $\text{De} = 0.5$ using the factoriza-
tion obtained with drop tolerance for each sub-problem of $1\text{E}-6$. As the DG
update may be accomplished in parallel, the deterioration with the addition of
processors is minimal for both types of runs. Since this step has high efficien-
cy for all the number of processors and unknowns considered, the lines are
very close to each other. A slight improvement is seen with increasing num-
bers of unknown, the speedup from 14.2 to 15.9................................................281

Figure 5.53: The speedups observed for the theta-method SUPG linear stability code.

Again, high parallel efficiency is observed for BCALM-preconditioned
BiCGStab, used for the Stokes solve. A lower efficiency is observed for the stress solve due to its small size. ......................................................... 284

Figure 5.54: The mesh used to study the flow of an Oldroyd-B fluid around a closely-spaced array of cylinders. This mesh has 2470 elements and 10,009 nodes. Discretization of viscoelastic flow with DEVSS/SUPG results in 142,134 total unknowns with 111,354 velocity, pressure or gradient unknowns and 30,780 stress unknowns. ......................................................... 285

Figure 5.55: The effect of cylinder spacing on critical De for flow of an Oldroyd-B fluid past a closely spaced array of cylinders. The result for L=3.5 was not obtainable by Smith [167] due to the number of unknowns that arise. Data for L < 3.5 taken from [167]. ......................................................... 287
List of Tables

Table 1.1: Storage requirements for CFS based on various levels of dissection. All storage requirements are given in double precision words (i.e. 1 integer = 1/2 double precision real) [126]. ................................................................. 58

Table 1.2: Lines of Code [70] .................................................................................... 61

Table 1.3: Comparison of SISAL and SR Timings on 4 Processors (in seconds). [70]. 61

Table 1.4: Comparison of CM-Fortran (Implicit) and CMMD (Explicit) Timings on the CM-5 (in seconds) [73] .................................................................................. 62

Table 2.1: Number of iterations for BiCGSTAB convergence for single-level BCALM decomposed into 16 domains for a 32x32 discretization of the natural convection in a cavity problem................................................................. 101

Table 3.1: Typical processor speeds and communication bandwidths measured using the LINPACK DGEFA 1000x1000 benchmark [52] and the MPBench Bandwidth of Send benchmarks [135]. ................................................................. 139

Table 4.1: AIC per Newton’s iteration for solution of the Jacobian linear system using unpreconditioned Bi-CGStab Krylov algorithm for several finite element mesh resolutions and Grashof numbers (Gr), Where Neq is the number of linear equations. Data from [34]. ................................................................. 183

Table 4.2: Average iteration count per Newton’s iteration for various Grashof numbers and fixed problem conditions. ................................................................. 184

Table 4.3: Efficiency calculated from problem speedup for various problem sizes with Gr=10,000 with a 8x8 coarse grid and drop tolerance of for the solution on both the coarse grid and the sub-blocks. ................................................................. 189

28
Table 4.4: Gustav efficiency calculated for Gr=10,000 with a 8x8 coarse grid and drop tolerance of 1E-5 for both the solution on both the coarse grid and the sub-blocks...

190

Table 4.5: Comparison of the CFS sparse direct solver to BCALM preconditioned BiCGSTAB. ................................................................. 192

Table 4.6: Efficiencies for four processor runs on the IBM SP2. Efficiency is based on speedup instead of operation count. For contrast, when based on computation rate as in [126], the efficiency is 83% for the problem with 13,764 unknowns... 194

Table 5.1: Comparison of the dimensionless drag per unit length obtained by various authors for the Oldroyd-B fluid and the h/R=2.0 geometry. Note that the results from Dou et al. [54] represent an extrapolation to a mesh size of zero. The results are from Dou et al. [54], Sun et al. [174], and Liu et al. [117]. ......................... 245

Table 5.2: The form and skin contributions to the dimensionless drag per unit length for the Oldroyd-B fluid and the h/R=2.0 geometry. The polymer presence clearly influences the solvent behavior. ................................................................. 245

Table 5.3: The time required for 100 timesteps for each of the meshes. The serial, frontal method compared to an eight-processor implementation of the parallel iterative method. For small problems, the methods are comparable, but as the problem size and the bandwidth increases, the parallel iterative version becomes much faster than the frontal method. The solution of SM4 was not possible with the frontal method. ................................................................. 258
Chapter 1

Introduction
The macroscopic properties of polymer materials can be greatly influenced by their deformation histories. The mechanical properties of plastic materials are strongly influenced by the polymer morphology imparted during processing. Molecular alignment caused by flow and by the thermal history of the material can cause mechanical defects and residual stresses in the material. These properties include many of importance to the polymer processing industry, such as strength, translucence, birefringence, surface finish, and dimensional stability. For example, the two bands shown in Figure 1.1 are both made from the same kind of polymer, HDPE, but the opaque, bottom band was conventionally extruded, while the clear, top one was conventionally drawn. The microscopic arrangements of the

Figure 1.1: The effect of deformation history on physical properties. Both strands are made from the same HDPE, but the opaque strand was conventionally extruded, while the other was extrusion drawn. [177]
polymer chains, set by the extrusion process, affected the optical properties of the final product. A good introduction to the effect of polymer process design on polymer morphology may be found in [10].

Quantitative modelling of viscoelastic flows and polymer processing can lead to better understanding of polymer shaping operations, leading to polymers with improved mechanical and physical properties. This fact has been recognized for years, yet attempts to base plastic part design on a sound theoretical foundation have been thwarted by complexity of the modelling and simulation problems. Challenges range from characterizing the non-Newtonian nature of the polymer melt, through the issues of discretizing the mathematical problem in a temporally and spatially stable manner, to solving the large systems of equations that are generated by the numerical method.

The first issue requires the development of viscoelastic constitutive equations that describe the complex rheology of polymer solutions and melts. Much progress has been made on this topic in the last decade starting from a basic kinetic theory description of polymer solutions and melts [22,23]. In their simplest form, these models are described by nonlinear hyperbolic differential equations for the polymer contribution to the stress; see Section 2.3 for detailed description. More complex models based on phase-space kinetic theory [139] and stochastic simulation methods [105, 173] are under development and will need to be accommodated in any robust numerical method. We have this context in mind, as will be illustrated in Section .

There has also been considerable effort aimed at developing stable and accurate numerical methods for discretizing the coupled set of conservation laws for momentum and mass and differential constitutive models. The issue that these methods address is that of the equation type, as will be described in detail in Section 1.2. Partial differential equations can be elliptic, hyperbolic or parabolic, depending on the method of information
transport. Elliptic equations transport information in all directions instantaneously, while hyperbolic equations transport information along streamlines. Parabolic equations have both types of character, transporting some information along streamlines and other information instantaneously. The type of equation determines choice of weighted residual method used to solve the system [103]. The development of stable and accurate numerical methods from the viscous formulation [150] to the sophisticated Discrete Elastic Viscous Split Stress - Gradient method [86] used for the viscoelastic simulations in this thesis, is presented in Section 2.3, with the fundamental issue in these formulations being maintaining the elliptic character of the velocity equations. These formulations result in a set of differential algebraic equations which are either fully simulated in both space and time, as presented in Section 2.6, or they may be restricted to steady-state, as presented in Section 2.8. As shown in these Sections, either case leads to the solution of these problems requires the inversion of a large, linear system. As will be illustrated in Section 1.3, the inversion of this linear system represents a significant challenge, as this will eventually impede progress towards three-dimensional time dependent simulations.

A new numerical methodology is needed to simulate the transient flow of viscoelastic materials, especially with the long-term goals of providing industry with methods for characterizing arbitrary three dimensional geometries. This is accomplished, here, through the application of state-of-the-art parallel linear solution techniques coupled with time accurate methods for discretizing the viscoelastic equations. Furthermore, the numerical methods for solving large linear systems which arise in viscoelastic flow modelling are of general interest to the scientific community, as similar large, linear problems arise in problems ranging from Newtonian fluid mechanics to electro-magnetics, quantum mechanical scattering problems, and large economic models.
The remainder of this chapter describes why an explicitly-coded parallel iterative technique is necessary for the solution of large viscoelastic flow problems, and outlines the strategy used in developing the work put forth in this thesis. The fundamental equations of viscoelastic fluid flow are described in Section 1.1. The concept of mathematical type is introduced in Section 1.2, as this motivates the splitting of the equations presented in Section 1.1 in the formulations presented in this thesis. The structure of the equations presented in Section 1.1 for viscoelastic flow are then compared to those for the Stokes' flow problem and the natural convection in a cavity problem in Section . Typical discretization sizes for three-dimensional flows are described in Section 1.3, based on viscoelastic flow around a periodic array of cylinders. This estimate establishes the need for a more sophisticated solver methodology. The motivation for using parallel computing follows in Section 1.4. Some potential solution methods are described in Section 1.5. A state-of-the-art parallel direct solver, concurrent factorization and storage [126], is presented, and challenges to its continued use are discussed. An alternative to a parallel direct method such as concurrent factorization and storage, parallel iterative methods are then introduced. The two dominant programming paradigms for writing parallel code, implicit and explicit, are reviewed in Section 1.6, and the early choice of using an explicit communication method, MPI is defended. Finally, an outline of the following chapters is described in Section 1.7.
1.1 Prototype Viscoelastic Flow Models

A prototypical viscoelastic flow problem is described by the conservation equations for mass and momentum combined with a constitutive differential equation for the stress [22,23]. The fluid is assumed to be incompressible, so that the conservation of mass is written as

\[ \nabla \cdot \mathbf{v} = 0 \]  \hspace{1cm} (1.1)

where \( \mathbf{v} \) is the velocity vector. The conservation of momentum equation for the fluid is

\[ \nabla p + \nabla \cdot \mathbf{\tau} = 0 \]  \hspace{1cm} (1.2)

where \( p \) is the pressure and \( \mathbf{\tau} \) is the deviatoric stress [22]. The most general form for the stress tensor in simple shear flow is:

\[ \pi \equiv p\delta + \mathbf{\tau} \]  \hspace{1cm} (1.3)

where \( \pi \) is the total stress tensor and the deviatoric stress \( \mathbf{\tau} \) is assumed to be symmetric:

\[ \pi = \begin{bmatrix} p + \tau_{xx} & \tau_{yx} & 0 \\ \tau_{xy} & p + \tau_{yy} & 0 \\ 0 & 0 & p + \tau_{zz} \end{bmatrix} \]  \hspace{1cm} (1.4)

The deviatoric stress tensor is split into the sum of two contributions from the polymer, \( \tau_p = -(1 - \beta)\dot{\gamma} \), and from the Newtonian solvent, \( \tau_s = -\beta\dot{\gamma} \) as:

\[ \mathbf{\tau} = \tau_p + \tau_s \]  \hspace{1cm} (1.5)

Here, \( \dot{\gamma} \) is the dimensionless rate-of-strain tensor and \( \beta \) is defined as the ratio between the Newtonian solvent viscosity, \( \eta_s \), and the total zero-shear-rate viscosity for the fluid, \( \eta_0 \) which is the sum of the solvent \( \eta_s \) and polymer \( \eta_p \) viscosities. A constitutive equation is required for the stress, and may be expressed generally as
\[ \tau + h \left( \frac{\partial \mathbf{x}}{\partial t} \right) = -\dot{\gamma} - \text{De} \beta \left( \frac{\partial \mathbf{y}}{\partial t} \right) \]  \hspace{1cm} (1.6) 

where \( \beta \) is a parameter characterizing solvent viscosity and \( \dot{\gamma} \) is the rate-of-strain tensor, which is defined by:

\[ \dot{\gamma} = \nabla \nu + \nabla \nu^T \]  \hspace{1cm} (1.7)

and the convected time derivative of the stress tensor, \( \tau_{(1)} \), is [22]:

\[ \tau_{(1)} = \frac{\partial \tau}{\partial t} + \nu \cdot \nabla \tau - (\tau \cdot \nabla \nu)^T - (\tau \cdot \nabla \nu) \]  \hspace{1cm} (1.8)

Several different viscoelastic constitutive equations are recovered from (1.6) by changing the values of the functions \( h \) and \( f \). For example, the Oldroyd-B model is expressed by setting \( f=1 \) and \( h=\text{De} \). After substitution, this results in the following form for the viscoelastic flow equations:

\[ -\beta \nabla^2 \nu + \nabla \cdot \tau_p + \nabla p = 0 \]  \hspace{1cm} (1.9)

\[ \nabla \cdot \nu = 0 \]  \hspace{1cm} (1.10)

\[ \tau_p + \text{De} \tau_{p_{,ij}} = -(1 - \beta) \dot{\gamma} \]  \hspace{1cm} (1.11)

where Eq. (1.9) is the statement of conservation of momentum, Eq. (1.10) is the conservation of mass, and Eq. (1.11) is the Oldroyd-B constitutive equation. If additionally \( \beta = 0 \), the upper-convected Maxwell (UCM) model is recovered.

For the purpose of this thesis, viscoelastic models based on an Oldroyd-B fluid are utilized. However, research continues to develop increasingly complex models of constitutive equations. Evolutionary models, such as devised by Ghosh [82], seek to vary the internal configuration variables in the constitutive expression. This model is very nonlinear and requires the calculation of the largest eigenvalue of the rate of strain tensor at every point in the flow. Coupled multimodes that consider the constitutive equation as
having more than one relaxation time associated with it is also an area of active research [139]. Due to the evolutionary nature of this field, it is important that the numerical schemes developed in this thesis are not specific to a particular formulation or discretization. The manner in which these flow equations are abstracted is described in Section .
1.2 Mathematical Type of Partial Differential Equations

Partial differential equations are characterized by their mathematical type, which is important as the type determines the mechanism by which information is transmitted throughout the domain. Because of this, the mathematical type of the equations must be considered for the implementation of boundary conditions to make the PDEs well-posed [37,102].

The classification of PDEs is most readily explained by considering a single second-order PDE as in [93,176]. The general constant-coefficient second-order non-homogeneous partial differential equation for the scalar $u(x, y)$ may be written:

$$Au_{xx} + Bu_{xy} + Cu_{yy} + Du_x + Eu_y + Fu = G$$

(1.12)

where $A$, $B$, $C$, $D$, $E$, $F$ and $G$ are constants. The classification of Eq. 1.12 depends on the sign of the discriminant: $B^2 - 4AC$, where the terminology elliptic, hyperbolic, or parabolic is chosen to reflect an analogy between the form of the discriminant that classifies PDEs and the form of the discriminant that classifies conic Sections from the general second-order algebraic equation:

$$Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$$

(1.13)

By defining an arbitrary curve $\Gamma$ which is specified by $\xi(x, y) = 0$, the mathematical type of the equation may be determined by the specification of $u$ and its normal derivative and checking of this boundary data guarantees a unique solution [102,176]. A new coordinate system is defined, $\eta(x, y) = 0$ where $\xi$ and $\eta$ are not tangent. By expanding $u$ in a Taylor series around $\Gamma$ and changing the independent variables to $(\xi, \eta)$, an equation for the normal derivative of $u$, $u_{\xi\xi}$, is obtained [37,102,176]:

$$\left(2A\left(\frac{\xi}{\xi_y}\right)^2 + B\left(\frac{\xi}{\xi_y}\right) + 2C\right)u_{\xi\xi} + f(u, \text{ other derivatives}) = 0$$

(1.14)

The coefficients of the highest order terms alone, therefore, determine the scientific type of
the Cauchy problem and determine the mathematical type [37,93,102]. There are three possibilities.

If \( B^2 - 4AC < 0 \), then the equation is elliptic and no real curves exist where the Cauchy data are insufficient to specify \( u(x, y) \). The problem is well-posed, so the speed of propagation of information is infinite throughout the domain, and the Cauchy data are used as boundary conditions. This is similar to the transmission of information spatially in a CSTR.

If \( B^2 - 4AC > 0 \), then the equation is hyperbolic. In this case, there are two real curves, or characteristics, where the Cauchy data do not specify \( u(x, y) \), so the problem is ill-posed. In this case, information is transferred forward in time along the characteristic curve in wave-like form. Initial conditions are specified at the inflow boundary and are convected through the domain to an outflow, as in a plug flow reactor.

Finally, if \( B^2 - 4AC = 0 \), the equation is parabolic. In this case, there is exactly one characteristic, and the Cauchy problem is not solvable. Information propagates as in the hyperbolic case in the direction of the real characteristic but is shared instantaneously in the other direction. This is similar to the startup of a CSTR, where information is shared instantaneously in space but is transported in time.

Analysis of type can be generalized to a system of \( n \)th order PDEs by rewriting the equations as a set of first order ODEs [37]. For example, consider a system of PDEs which may be written:

\[
A \frac{\partial u}{\partial x} + B \frac{\partial u}{\partial y} = C
\]  

(1.15)

The type for this system of equations is determined by the characteristic slopes, \( \alpha = \partial x / \partial y \), found by evaluating:
\[ \det \left( A - \frac{\partial x}{\partial y} B \right) = 0 \]  

(1.16)

In this case, real characteristics are associated with hyperbolicity while imaginary characteristics are associated with ellipticity.

The mathematical type of the viscoelastic flow equations change type depending on the values of the parameters in the equations. The equation of motion is elliptic in the inertialess limit studied in this thesis, however it can change type from elliptic to hyperbolic if inertia is included [106]. When coupled with the continuity equation, an elliptic saddle point problem is formed [36]. This type of problem leads to constraints on the numerical method used to solve the system of equations [36]. The addition of the constitutive equation further complicates analysis as this equation, alone, is hyperbolic.

The mathematical type of the governing equations for an Oldroyd-B fluid, described in Section 1.1, was studied in [176]. For a two-dimensional flow, the characteristics, \( \alpha \), are given by the equation

\[ (1 + \alpha^2)^2(-v_x + \alpha v_y)^3 \]  

(1.17)

This equation has three real roots corresponding to the three hyperbolic component equations of the stress tensor and four imaginary roots which are associated with the elliptic momentum and continuity equations. What this analysis suggests is that the momentum and continuity equations remain essentially an elliptic saddle point problem, while the constitutive equations retain their hyperbolic character. This suggests a crucial splitting which is exploited in this thesis. The momentum and continuity equations can be solved using a Stokes solver for nonhomogeneous data from the polymer stresses. The constitutive equations can be solved by methods appropriate for hyperbolic equations as if the velocity field is a given.
The time-splitting methods presented in Section 2.6 and developed in [167,176] exploit this splitting, such that it is important to develop a robust Stokes' solver for the momentum and continuity equations, while appropriate hyperbolic methods are necessary for treatment of the constitutive equation. In the next Section, the similarity between the governing equations for an Oldroyd-B fluid, the Stokes'-flow problem, and another test problem, that of natural convection in a fluid-filled cavity is detailed. Viscous Flow Problem Structure

The governing equations of the flow of an Oldroyd-B fluid, Eqs. (1.9)-(1.11), must be solved numerically for the dependent variables, the velocity, stress and pressure, via their discretization using a numerical method. The finite element method (FEM) has proven to be the method of choice for numerical solution of viscoelastic flow problems. The complex geometries, free surfaces and complex physics which are important in polymer processing are all easily incorporated into FEM models. Furthermore, the FEM is the best method for approximating arbitrarily shaped domains, and it has large flexibility for resolving boundary layers in the flow using local mesh refinement.

In Section 1.2, the topic of mathematical type for these equations was considered [176]. The momentum and continuity equations together form an elliptical saddle-point problem for the velocity and pressure field:

\[-\beta \nabla^2 \mathbf{v} + \nabla \cdot \mathbf{\tau}_p + \nabla p = 0 \tag{1.18}\]

\[\nabla \cdot \mathbf{v} = 0 \tag{1.19}\]

while the continuity equation is hyperbolic for the components of the stress tensor:

\[\tau_p + \text{De} \tau_{p,ij} = -(1 - \beta) \dot{\gamma} \tag{1.20}\]

For this reason, most finite element formulations are based on artificially splitting the momentum and continuity equations from the constitutive equation. The formulations of
most mixed finite element methods for viscoelastic flows are based on separate finite
element approximations for the velocity and pressure fields, associated with the momentum
and continuity equations, and for the extra elastic stress, governed by the constitutive
equation. Several mixed finite element methods have been developed in recent years for
the solution of these problems. These methods include the viscous formulation [150], the
Elastic Viscous Split Stress method [150], the Elastic Viscous Split Stress-Gradient
method [176], and the Discrete Elastic-Viscous Split-Stress-Gradient method [176]. The
development of these methods is detailed in Section 2.3. The treatment of the elliptic saddl
point problem for the Discrete Elastic-Viscous Split Stress-Gradient formulation for an
Oldroyd-B fluid using mixed finite elements is discussed in Section 2.4.2. Methods for
treating the hyperbolic constitutive equation is discussed in detail in Section 2.4.3.

The timestepping methods described in detail in Section 2.6 exploit these different
equation types and split the two equation types from each other in the timestepping
schemes. The result, as will be shown in Sections 2.6.1 and 2.6.2 for both fourth order
Runge-Kutta and the theta method, is that two problems need to be solved: one for the
velocities, pressures, and velocity gradients, and another for the stresses. Under both of
these discretizations, as will be demonstrated in Section 5.3, the Stokes-like problem
resulting from the continuity equation and momentum equation pair is by far the most
time consuming part of the algorithm. The reason for this is the treatment of the stress
unknowns using methods appropriate for hyperbolic equations as detailed in Section 2.4.3.
These methods either cause the stress equations to have far fewer unknowns, as will be
shown for the SUPG discretization, or to have a far simpler problem structure, as will be
shown for discontinuous Galerkin. A discussion of these effects may be found in Section
2.4.3.
The inertialess Newtonian flow problem, or Stokes’ problem, is relevant to the current work as the $De = 0$ limit of the viscoelastic problem. The equations of motion are

$$-\mu \nabla^2 v + \nabla p = F \quad (1.21)$$

$$\nabla \cdot v = 0 \quad (1.22)$$

where $\mu$ is the viscosity, $v$ is the velocity, $p$ is the thermodynamic pressure, and $F$ is the body force, usually gravity. The flow is assumed to satisfy the boundary condition $v = 0$ on the boundary of the domain. Comparison of Eqs. (1.21) and (1.22) to the momentum equation and constitutive equation for an Oldroyd-B fluid in Eqs. (1.18) and (1.19) show why the latter equation pair is often referred to as a modified Stokes’ problem.

The Stokes’ flow problem has been extensively studied [36], and it is work with this problem which lead to the mixed finite element scheme used for the momentum-continuity equation pair as presented in 2.4.2. Another problem similar to both the Stokes’ flow problem and the modified-Stokes’ like momentum and continuity pair in the equations governing viscoelastic flow is the problem of buoyancy-driven natural convection in a fluid filled cavity. This problem is discussed in detail in Section 2.5, involving a fluid with variable density enclosed in a differentially heated cavity, illustrated in Figure 2.24. Flow results in the cavity and is governed by the following modified Navier-Stokes’ equations:

$$\rho v \cdot \nabla v - \mu \nabla^2 v = \rho e_y T - \nabla P \quad (1.23)$$

$$v \cdot \nabla T - \alpha \nabla^2 T = 0 \quad (1.24)$$

$$\nabla \cdot v = 0 \quad (1.25)$$

Here, $\alpha$ is the thermal diffusivity, $v$ is the velocity vector, $P$ is the pressure field, $e_y$ is the unit vector in the direction of gravity, $\rho$ is the density, and $\mu$ is the viscosity. These equations and their discretization is presented fully in Section 2.5, and they, too may be
understood as a modified Stokes' problem. While an additional energy equation exists, Eq. (1.24), and some hyperbolic character exists due to convection occurring in the domain, these equations are strongly elliptic in character. Furthermore, this problem is interesting in the current work because it allows a point of comparison for the methods developed in this thesis to a prior state-of-the-art solution method, Concurrent Factorization and Storage [126,127,128,129] as will be described in detail in Sections 1.7 and 2.2.

In all three cases, the governing equations describe an elliptic operator with some forcing functions coupled with a continuity equation. For this reason, the study of the Stokes' like natural convection problem provides insight into potential solution methods for the Stokes-like momentum equation and continuity equation pair in the viscoelastic flow problem.
1.3 The Need for A New Solver: Viscoelastic Flow Around an Array of Cylinders

Before examining solution specifics, we would like to estimate the size of problems which investigators will be interested in solving. As described in [117] and the references therein, one viscoelastic flow problem of recent interest is that of flow past an array of cylinders, illustrated in Figure 1.2.

This is a fundamental problem in viscoelastic fluid flow, involving extensional polymer rheology [31,125]. Furthermore, this flow has been found to have a three dimensional instability in experiment [31,125]. These factors make this problem a good test problem for constitutive models [116]. A dimensionless number used to characterize this system is the Weissenberg number, $\text{We} \equiv \lambda \langle v \rangle / R_c$, where $\lambda$ is the relaxation time of the fluid, $\lambda \langle v \rangle$ is the average flow rate across the channel, and $R_c$ is the radius of the cylinders. McKinley et al., [125], observed that, for the flow of a viscoelastic fluid around a cylinder in a rectangular channel, with a ratio of channel height to cylinder diameter between 2.0 and 5.9 and beyond a critical $\text{We}$, a flow instability was observed, characterized as a transition from two dimensional steady to three dimensional steady, time periodic flow. This instability is also reported by [117]. We consider the size of the problem which would be required to model this instability.

![Figure 1.2: Flow past a linear, periodic array of cylinders. Geometry is specified by the radius $R_c$, the spacing $L_c$, and the channel half-height $H_c$. [117]](image-url)
Consider the finite element solution of two-dimensional viscoelastic flow around a periodic array of cylinders as described in [117]. The flow of an Oldroyd-B fluid is assumed, and the discretization of this problem in two dimensions is covered extensively in Section 2.4. We examine the case where \( H_c = 2R_c \), \( L_c = 6R_c \) in Figure 1.2, and assume that we are interested in modeling cylinders of length \( 10R_c \) out of the page.

The coarsest and finest meshes which this two-dimensional problem was solved over in [117] are pictured in Figures 1.3 and 1.5. The coarsest mesh, M1, has 888 elements while the finest mesh, M6, has 4,238 elements. Assuming that the side length of the elements along the channel wall in the two-dimensional meshes, \( \frac{R_c}{4} \), can be used to determine mesh size in the direction of the cylinder's length, we find that we need 40 elements in the direction of the cylinder length. Clearly, this is a conservative estimate, since refinement in the third dimension would be necessary. The three dimensional discretization of mesh M1 would have \( 888 \times 40 = 35,520 \) elements while the discretization based on mesh M6 would have \( 4238 \times 40 = 169,520 \) elements. The \textit{inf-sup} condition [36], discussed in Section 2.3, requires that for a convergent numerical method, the finite elements used for the velocity field be \textit{higher order} than those used for the pressure field. This means that in our current discretization, bilinear basis functions are used for the pressure while biquadratic basis functions are used for the velocities. Neglecting boundary

![Figure 1.3: M1: The coarsest mesh considered by Liu, et. al. [117]](image)
nodes, each bilinear element introduces one node while biquadratic elements introduce eight nodes, as illustrated in Figure 1.4. Since there are 15 variables represented by bilinear elements (velocity gradients, stresses, and pressure) and three represented by biquadratic elements (velocities), we see that the three-dimensional discretization based on M1 would have 1,385,280 unknowns, while that based on M6 would have 6,611,280

![Figure 1.4: In three dimensions, each additional triquadratic brick element introduces eight nodes, shown as the circles and the square on the example structured mesh, above. Trilinear brick elements introduce only one, the square.](image)

![Figure 1.5: M6: The finest mesh considered in Liu, et. al. [117]](image)
unknowns. Clearly, this is a very conservative estimate, both because boundary nodes exist and refinement would likely be necessary in the direction of cylinder's height.

If we attempt to solve this system using banded LU decomposition, the storage required will be the product of the total number of unknowns and the bandwidth of Jacobian system. While bandwidth is a function of the problem geometry, we can develop a crude estimate by assuming that all of the elements are in a cube of size $N = \sqrt[3]{T}$ on a side, where $T$ is the number of elements in the three-dimensional discretization. For the three-dimensional M1 mesh, we find $N_{M1} \approx 33$ and $N_{M6} \approx 56$. The bandwidth for the finite element discretization of a three dimensional cube is $O(N^2)$. Based on this approximation, we find that the storage required for the LU decomposition of the three dimensional mesh based on the coarse mesh M1 is approximately 1.4 Giga-words and that based on the fine mesh M6 would be approximately 19.3 Tera-words.

Beyond the massive storage requirement for solution of this system using conventional technology, the time required to accomplish solution of this system renders it intractable. For a banded LU decomposition, the operation count is $O(b^2 Neq)$ [83]. If we consider running this on a state-of-the-art Hewlett-Packard workstation, which would yield around 50 MFLOPS of computation speed [52], the discretization based on the coarse mesh M1 would require 19 days for computation while the discretization based on the finer mesh, M6, would require over 2 years! For three dimensional simulations to be tractable, new solver technology, such as put forth in this thesis, is necessary.
1.4 Why Parallel Computing?
Many rationales can be used to conclude that research into parallel computing is necessary for high-performance computations. First, we argue that the development of parallel computation is inevitable to expand the limits of feasible computation. Then, the reduced computational cost associated with doing parallel computation is examined. Finally, despite years of work on methods for automating parallelization [70,73,143], many time-consuming models in scientific computing remain serial codes. There is a need for development activity for parallel computing to reach its potential, and this is discussed in the context of some of the challenges which are unique to parallel computer algorithms.

Parallelism makes available more computational performance than is available in a single processor at a given time. Furthermore, in designing a single chip, physical limits are being approached which will limit their performance. Consider the speed of light as a limitation to the speed of computers. Assume that we have a 1 Tflop uniprocessor computer with 1 TB of memory. Because the fastest data speed is the speed of light $c = 3 \times 10^8 \text{ m/sec}$ and must travel between the CPU and the memory $10^{12}$ times per second, the maximum distance that we could place memory from the CPU is $c/10^{12}$ or 0.3 mm! Furthermore, if the memory were conventionally built, as a planar grid of bits, one word would have to occupy an area of about 9 square Angstroms, the size of a small atom. Though observed that the basic speed of microprocessors has gone up by a factor of 2 every 18 months, the so-called Moore’s law [133], this speedup has come about partially due to the incorporation of parallel techniques into microprocessors.

For example, the Intel Pentium series of microprocessors use internal parallelism [84]. Through the use of multiple functional units, the CPU is broken up into sections which work concurrently. This is combined with data pipelining, where the functional units are further broken into segments, each of which is responsible for a partial decoding or execu-
tion of the process. The Pentium CPU is also branch-predictive between the multiple functional units. Based on the instruction sequence received, the CPU attempts to extract parallelism from the assembly level code. Sometimes, this will work successfully, resulting in speedup. At other time, this will not work, and inefficiency is introduced. In all of these cases, the parallelism occurs without the user's interference, with the CPU attempting to extract parallelism from the instruction set without intervention [84].

Economically, parallel computing also makes sense. Both Mehrabi [126] and Fischer [64] discuss the economics of parallel computing in terms of computational cost. Consider that computers have two kinds of costs associated with them: direct costs ($C_{direct}$) which include the costs of purchasing and maintaining the computer and indirect costs ($C_{indirect}$) which are the costs associated with waiting for a solution to the problem, proportional to the computation time ($t_{compute}$) required on the computer.

Consider two computers using state-of-the-art processors:

- **Computer-1**: A single processor with direct costs ($C_{direct,1}$) and indirect costs ($C_{indirect,1}$) associated with the computation time, $t_{compute,1}$

- **Computer-2**: A computer composed of $P$-processors found in Computer-1, with direct costs $C_{direct,2} \geq P \times C_{direct,1}$ and indirect costs $C_{indirect,2}$ associated with $t_{compute,2} \geq t_{compute,1}$

If one is most concerned with minimizing the direct costs of solving a problem, the problem should be solved on computer-1, the low-cost single processor machine. However, this assumes that both computers can solve the problem of interest. In reality, computer-2 probably has $P$ times as much memory as computer-1 -- and can probably solve problems which computer-1 cannot.
If one wants to minimize indirect costs, however, computer-2 is the reasonable choice. Consider replacing computer-1 with one which has the same \( t_{\text{compute}} \) as computer-2, thereby making their indirect costs the same. The direct costs associated with this single processor would be particularly high because improving the technology behind the processors of computer-2 is very difficult. Parallel computing is attractive, then, because it minimizes direct costs compared with single processor machines with similar indirect costs.

Parallelism can be viewed as inevitable in many respects, and the roots of parallel computing can be traced back about twenty years to computers such as the CDC660 and IBM360/91 [165]. Yet progress in this area has proceeded at a very slow pace in comparison with the development of computation in other areas; it is still used primarily in the niche of scientific computing. This is probably partially due to human nature: like the document which you are reading now, human consciousness tends to be a serial endeavor.

Other, more tangible considerations exist. There has not been a generally accepted model for what a parallel computer should look like, with theory following after technology: software could not be developed which could span all of the various models. Today, most researchers accept the paradigm that most scientific parallel computing will be done on distributed memory computers where each processor has its own memory which must be communicated with by other processors over a network. Work on shared memory computers continues, where all processors share a common memory storage, but at this time no-one knows how to scale this concept to large computers.

Further slowing the move to parallel computing, writing computer programs for parallel computations is difficult. Consider, as an example, an algorithm which spends some fraction, \( f < 1 \), doing parallelizable work. This algorithm then spends the fraction,
\( s = 1 - f \), of its time doing serial work. The time that the algorithm takes on a parallel computer with \( P \) processors is then

\[
t_p = \frac{t_1 \times f}{P} + t_1 \times s
\]

(1.26)

where \( t_1 \) is the amount of time that it took on one processor. If we define the speedup, \( S \), as the ratio of the time that the algorithm takes on one processor to the time it takes on \( P \) processors and define the efficiency, \( \varepsilon \) as the speedup divided by \( P \), we find the following relationships:

\[
S \equiv \frac{t_1}{t_p} = \frac{1}{\frac{f}{P} + s}
\]

(1.27)

\[
\varepsilon \equiv \frac{1}{f + s \times P}
\]

(1.28)

In other words, the speedup is bounded by \( 1/s \) and

\[
\lim_{P \to \infty} \varepsilon = 0!
\]

(1.29)

This result is known as Amdahl's law [4], and underscores that it is imperative that serial bottlenecks be avoided in parallel code. Disregarding the issues of controlling communication between processors, this example underscores how intrinsically tricky parallel programming can be. Fortunately, for many problems \( f \) increases with problem size, as first observed by Gustafson [87].
1.5 Possible Solution Methods

The application of the timesteping schemes presented in Section 2.6 to the viscoelastic flow problem leads to the formation of large linear systems which must be solved at each timestep for the discrete unknowns. As illustrated in Section 1.3, conventional direct solution methods cannot be used for the problem sizes of interest. As a result, we consider the application of parallel solution methods. There have been attempts to parallelize direct solution methods, and one of the most robust methods, concurrent factorization and storage (CFS) [126,127,128,129] has been shown to be faster than commercial packages. This is presented here, and an alternative to parallel direct methods, parallel iterative methods, are introduced.

1.5.1 CFS: A Parallel Direct Method

The concurrent factorization and storage (CFS) scheme is a parallel sparse direct solver developed at in with the expectation of seeing use as a general solver for continued crystal growth and viscoelastic investigations [126]. While banded algorithms for LU decomposition are used extensively on single-processor machines due to ease of implementation, they cannot be effectively implemented on parallel computers because of their non-hierarchical nature. Instead, CFS is an algorithm which utilizes domain decomposition via nested dissection to divide the geometrical domain recursively into sub-domains and separators, extending the method of Lucas, et. al [119] to include applicability to asymmetric matrices [128]. The scheme is described in [126], [128] and [129].

The original problem domain is dissected into three primary levels, as depicted in Figure 1.6, with nested dissection occurring at each level. First, incomplete nested dissection is performed on the original domain to divide the problem into the same number of pieces as the number of processors. Notice that this ties us to using \(2^n\) processors, consistent with the then-popular hypercube architecture. The pieces which result from this dissection are
Figure 1.6: Overall levels of dissection in CFS. First, the physical geometry is dissected into the same number of sub-domains as processors. Then each sub-domain is broken into super-nodes, over which pivoting is done. Finally, the dissection is carried out to the level of individual elements to conserve memory.
then numbered using the reflected grey code, in order to minimize communication in a hypercube architecture [40]. This process is illustrated for a rectangular domain in Figure 1.7. The resulting Jacobian has the sparsity illustrated in equation (1.30)

$$ J = \begin{bmatrix} J^{(ii)} & J^{(ie)} \\ J^{(ei)} & J^{(ee)} \end{bmatrix} $$

(1.30)

where $J^{(ii)}$ represents the interactions within each sub-domain, $J^{(ie)}$ and $J^{(ei)}$ represent the couplings with the domain's exterior, and $J^{(ee)}$ corresponds with interactions between two or more sub-domains on the separators. The $J^{(ii)}$, $J^{(ie)}$, and $J^{(ei)}$ are all internal to a particular domain, and are the components of that domain's interior matrix. The $J^{(ee)}$ are partitioned column-wise between adjacent processors to minimize memory requirements. The sparsity of each of the $J$ components is further exploited by nested dissection of the elements of each domain into super-nodes. This process is illustrated for a processor's sub-domain in Figure 1.8, and the sparsity of the processor's exterior and interior matrices is also presented. Row pivoting is allowed within these super-nodes. Finally, each of these super-nodes is then dissected to lower the memory requirements of the algorithm.

After formulation, factorization takes place. This sequence proceeds from the leaves of the elimination tree to the root, illustrated in Figure 1.9 for the rectangular domain in Figure 1.7. First, each domain or super-node has its interior factorized, and then the exterior is formed. When forming the exterior matrix, the processors may have to communicate with each other if an update column is assigned to different processor. This was shown to be an efficient fan-in procedure [7,128]. As the factorization continues, however, it becomes less efficient as coarse parallelism, where several super-nodes are factorized simultaneously, degrades to fine parallelism, where several processors operate on a single super-node. The
Figure 1.7: four-level incomplete nested dissection of a rectangle [128].
Figure 1.8: The dissection of a processor's sub-domain into super-nodes and the sparsity of the resulting exterior and interior matrices [129].

Figure 1.9: Elimination tree [128].
factorization is followed by forward and backward elimination, operations which are inherently serial.

Implementation of CFS on more complicated and general problems presented several challenges. First of all, issues of topology are less important to the scientific user today than when CFS was originally developed. The hypercube-based interconnection network, for which the CFS algorithm was optimized, has largely been abandoned. The High-Performance Switch topology of the IBM/SP2 was designed to keep the bandwidth between any communicating pair of nodes constant: Bandwidth available to a node is \((lk)/h\) where \(l\) is the link bandwidth, \(k\) is the number of links per node, and \(h\) is the average number of hops. On the SP2, both \(h\) and \(k\) scale logarithmically with the number of nodes [1]. Other modern computer switches randomize network routing so that users have no control over the communication network. This has the result that CFS would have to be re-written to accommodate arbitrary numbers of processors laid out in an arbitrary topology.

The question of optimal storage technique should also be addressed. For this example, we consider the natural convection problem, introduced in Section and discussed in detail in Sections 2.5 and Chapter 4. The storage for a nested dissection scheme is estimated to be \(O(n^3\log 2(2))\) for an \(n \times n\) two dimensional grid discretized with finite elements [79,80]. This ideal, however, is not realized in practice due to the increased complexity introduced by increasing levels of dissection. Consider the various storage requirements for a \(32 \times 32\) discretization shown in Table 1.1. While storage for double-precision numbers decreases as the number of super-nodes increases, both integer storage requirements and computation times rise. For meshing and decomposing an arbitrary domain, CFS would have to optimize memory usage, while assuring that the super-nodes are large
<table>
<thead>
<tr>
<th>Supernodes</th>
<th>Integer Storage</th>
<th>Double Precision Storage</th>
<th>Total Requirement</th>
<th>Required Computation Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4x4</td>
<td>15,901</td>
<td>14,576,569</td>
<td>14,592,470</td>
<td>98</td>
</tr>
<tr>
<td>8x8</td>
<td>23,682</td>
<td>5,219,385</td>
<td>5,243,067</td>
<td>25</td>
</tr>
<tr>
<td>16x16</td>
<td>1,469,550</td>
<td>2,922,777</td>
<td>3,216,327</td>
<td>51</td>
</tr>
<tr>
<td>32x32</td>
<td>2,128,630</td>
<td>2,509,169</td>
<td>4,637,799</td>
<td>703</td>
</tr>
</tbody>
</table>

Table 1.1: Storage requirements for CFS based on various levels of dissection. All storage requirements are given in double precision words (i.e. 1 integer = 1/2 double precision real) [126].

enough for partial pivoting to ensure numerical stability, currently an empirical operation. Furthermore, the algorithm would have to balance memory usage with efficient solution, because the computation times are dramatically affected by the level of dissection.

1.5.2 Iterative Solution

While the solution provided by direct methods such as CFS will be exact and terminate in a finite number of operations, the required memory usage is high and the required operations do not lend themselves to ready parallelization. This thesis considers using iterative methods to achieve solution of our linear system. Iterative methods are explained in detail in Section 2.2.2, but broadly, iterative schemes generate successive refinements \( x_i \) to an initial approximation to the solution of a linear system, \( x_0 \), of the form:

\[
x_i = x_0 + \sum \delta_i
\]

In a successful iterative procedure, the iterates, \( x_i \), converge to the actual solution of the problem. In this thesis, we consider Krylov iterative methods, a class of methods which has been shown to be applicable to a variety of complicated problems [154]. These methods are described in detail in Section 2.2.2, but generally, the iterates \( x_i \) are generated from an affine subspace \( x_i = x_0 + \kappa_i \) of dimension \( i \). Here, \( \kappa_i \) is the Krylov subspace.
generated from the system matrix $A$:

$$\kappa_i = \text{span}\{x_0, Ax_0, A^2x_0, \ldots, A^{i-1}x_0\}$$  \hspace{1cm} (1.32)

from which iterates are selected using a minimization criteria, presented in detail in Section 2.2.2. Further details are given in Section 2.2.2 in the next chapter, which discusses the selection of iterative method for solving linear systems. For now, we wish to disregard these details and focus, instead, on the fundamental characteristics of these methods.

For implementation of a Krylov method, the only required non-scalar operations are matrix-vector products and dot products -- which, as will be shown in chapter 3, are both operations which are generally efficient in parallel [154]. Furthermore, the memory requirement of these methods is far lower than required for direct methods, as they are suitable for use with compressed matrix formats that require only the nonzeros of the linear systems. This is presented in Section 3.3.2.

However, straightforward application of these methods to the viscoelastic flow problem fails [35]. This is because the convergence of an interactive scheme is dictated by the condition number of the linear system, and, as shown in Sections and 2.4, the linear system which arises in discretization of the viscoelastic flow problem is both singular and asymmetric. Because of this, we do not wish to apply a Krylov method to the original linear system, $Jx = b$. Instead, we consider the preconditioned linear system

$$\bar{J}^{-1}Jx = \bar{J}^{-1}b.$$  \hspace{1cm} (1.33)

where $\bar{J}^{-1}$ is some approximation to the inverse of $J$ [154]. Typically, parallel implementation of the preconditioning step is generally the most challenging step in an iterative method [130]. The topic of preconditioning is considered in Section 2.2.3. Currently, there is no general, parallelizable preconditioning scheme for ill-conditioned matrices.
1.6 Basic Parallel Computing Considerations

The size of the discretizations for large viscoelastic flow problems clearly motivates parallel computing. Without getting into the details of modern parallel computing, as described in chapter 3, here we consider the early selection of an explicit programming model rather than an implicit model.

Basically, two kinds of programming paradigms are available to the modern scientific user of parallel computers. These are *implicit languages*, where the programmer writes a sequential model and the compiler must locate, distribute, and schedule all of the parallelism, and *explicit languages*, where the programmer is responsible for all of the parallelism, and the compiler must simply implement what is specified. Both of these programming paradigms are implementable in the multiple data framework which current commercial parallel machines utilize.

For the purposes of this discussion, we will include parallelizing compilers of all kinds as implicit languages. Even though functional languages and data-parallel languages are not strictly implicit both because they rely on the programmer to specify some of the data distribution and communication and because the compiler may not perform dependence analysis or reordering transformations, the parallel abstraction is clearly at a higher level than for the message passing languages, which are explicit. Languages of the implicit type typically produce code which is of the single instruction multiple data (SIMD) model: Each of the processors performs the same instructions on its own set of data. Sample implicit languages include the functional language SISAL [70] along with the data-parallel languages CM-Fortran [157], and High Performance Fortran (HPF) [49]. We will consider explicit languages to be those which require *message passing* for communication. With explicit languages, communication is specified with extensions to the underlying
Table 1.2: Lines of Code [70]

language which allow nodes in the parallel computer to post *sends* and *receives*. All communication must be specified by the programmer with these languages. Examples of these types of languages include SR [70], CMMD [181], PVM [75], and MPI [184], with the latter two being the most common.

The big advantage of implicit languages are that the codes are typically much shorter than for explicit languages. Freeh [70] compared the state-of-the-art implicit language SISAL to the explicit language SR on a shared memory Silicon Graphics Iris 4D multiprocessor. He observed that codes written in SISAL are typically 2 to 3 times smaller than those written in SR, as can be seen in Table 1.2. Table 1.3 presents the execution time required for the performance of various routines in SISAL, SR, and C using message passing libraries, as reported in [70]. Notice that implicit programming tends to do very well at those routines which rely heavily on basic matrix routines such as matrix multiplication and Jacobi iteration. However, the program in SISAL failed to outperform the SR code in

<table>
<thead>
<tr>
<th></th>
<th>SISAL (Implicit)</th>
<th>SR (Explicit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix Multiplication</td>
<td>19</td>
<td>51</td>
</tr>
<tr>
<td>Jacobi Iteration</td>
<td>58-62</td>
<td>136</td>
</tr>
<tr>
<td>Adaptive Quadrature</td>
<td>36</td>
<td>62-99</td>
</tr>
<tr>
<td>LU Decomposition</td>
<td>49-76</td>
<td>105</td>
</tr>
<tr>
<td>Mandelbrot</td>
<td>31</td>
<td>90</td>
</tr>
</tbody>
</table>

Table 1.3: Comparison of SISAL and SR Timings on 4 Processors (in seconds). [70]
<table>
<thead>
<tr>
<th></th>
<th>CM-Fortran</th>
<th>CMMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaussian Elimination</td>
<td>645</td>
<td>427</td>
</tr>
<tr>
<td>(2047)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Odd-Even Sort</td>
<td>31.1</td>
<td>69.5</td>
</tr>
<tr>
<td>(32,000)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1.4: Comparison of CM-Fortran (Implicit) and CMMD (Explicit) Timings on the CM-5 (in seconds) [73]

a complex algorithm such as LU-decomposition. Furthermore, the SR routines were consistently unable to come close to matching the speed of the optimized message passing hand coded into the C-code. This implied that SR introduced a lot of overhead through passing messages without programmer interaction. Similar timing results were obtained by [73] on the CM-5, comparing the implicit, CM-Fortran language to the explicit, CMMD message passing library. He obtained the results summarized in Table 1.4. Again, the use of implicit languages slowed the performance of complicated algorithms. Padua and Peterson [143] reported similar, disappointing results for the performance of KAP, a commercial parallelizing compiler. Although implicit languages seem promising because of the decreased complexity of their code, explicit languages seem necessary for high performance at this time.

An explicit, message-passing paradigm was selected, layering communication calls with Fortran and C computer code. The key concern then is portability: It was desired that the codes written during this investigation would be able to run efficiently on the parallel computers which are in use when at its conclusion. For portability, PVM and MPI were the obvious choices. PVM, however, exhibits poor performance [62]. MPI, on the other hand has been embraced by vendors, and offers better performance. In fact, Franke, et. al. [69] observed on the IBM/SP2 that MPI-F, the IBM implementation of MPI for the SP2, performed nearly as well as the proprietary message passing library, MPL, in terms of
both maximizing bandwidth and minimizing communication latency, as illustrated in Figure 1.10.

MPI is a standardized communication protocol used on massively parallel machines with distributed-memory and shared-memory architectures. It is implemented as a set of library calls that can be called from different programming languages. Currently, it is the most popular message passing parallel computing tool, and is very flexible [147].

Because it is an efficient, explicit, portable library, MPI was chosen as the best language for parallel computing at this time. Five years after selecting it using this criteria, MPI is still unsurpassed in terms of combined performance and portability.
1.7 Methodology and Outline

As the work presented in this thesis will show, viscoelastic flow problems can be efficiently solved with iterative techniques. Because of the large storage savings which associated with iterative methods, similar computational times favor iterative methods. Though robustness is a concern with iterative methods, leading into this work, many investigators were having good results with preconditioned conjugate gradient methods. Work by Natarajan [138], compared the performance of direct elimination methods to a preconditioned conjugate-gradient iterative method on the dense, complex-symmetric, indefinite linear systems which arise in acoustics boundary element computations. Seventeen iterative methods were compared in [114], applied to the solution of the nonlinear three-dimensional ground-water flow equation, and concluded that the preconditioned conjugate-gradient method performed best. Christara [43] applied parallel Schur complement preconditioned conjugate-gradient methods to the finite element discretization of linear second order partial differential equations, and observed parallel efficiencies of about 90% on a 32 node iPSC/860 hypercube.

While ILU factorizations have been shown to be successful preconditioners for a variety of problems, this method cannot be efficiently implemented in parallel. In this work, a parallelizable preconditioner was sought for the fully-implicit formulations of a general viscous flow problem, natural convection in a fluid filled cavity. This general solver was then ported into a viscoelastic code which utilizes an operator splitting method, the \textit{theta} method.

As described in Section 1.6, current implicit methods are not inherently faster than explicit, message passing methods, but can be competitive for regular matrix operations. For this reason, the message passing library MPI was used along with a matrix toolkit for basic operations, the Parallel Extensible Tools for Scientific Computing (PETSc [12]).
Since selecting this toolkit early in this project’s life, several high performance applications have been developed using PETSc, including a Gordon Bell award winner [5,107]. Load balancing was accomplished using the spectral dissection tool [90], CHACO by Bruce Hendrickson and Robert Leland [91]. Load balancing using spectral dissection is reviewed in Section 3.3.3.

As a basis for forming this method, a two-dimensional unstructured mesh was assumed. Furthermore, the general framework of an asymmetric viscous flow problem was utilized, leading to a portable solver which is readily adaptable to legacy unstructured mesh codes which have been developed by many researchers over several years.

The thesis is organized in the following manner. An introduction to many fundamental concepts used throughout this thesis is provided in Chapter 2. The modelling of the viscoelastic flow of an Oldroyd-B fluid is presented, encompassing both equation formulation and finite element discretization. The governing equations for the problem of natural convection in a fluid-filled cavity is also presented along with finite element discretization. Both direct and iterative methods are discussed along with the topic of preconditioning. Spatial problem discretization at steady-state is first discussed. Then, two time dependent formulations are presented. Finally, two methods for studying the stability of viscoelastic flow are presented. The first one is a Runge-Kutta/Discontinuous Galerkin 2D linear stability method developed in a straightforward manner from the time dependent formulation [175]. The other is a 3D linear stability method based on a time integration technique known as the theta method [167]. Solution of the discretized equations at steady state using Newton’s method is also addressed.

An overview of parallel processors in presented in Chapter 3. Available computer architectures and their evolution is briefly discussed. The potential for parallel processing
and its incentives are analyzed. Factors affecting implementation of a parallel method for solving partial differential equations are discussed.

The parallel iterative method devised here for viscous flow systems is described fully in Chapter 4. This algorithm is a robust, efficient method for solution of the partial differential equations resulting from viscous flow problems. Because viscous problem physics are exploited in the developed solver, a test problem, buoyancy-driven natural convection in a fluid-filled cavity, is presented and the solver is developed in this framework. This allows comparison with the work done previously by Mehrabi [126]. The linear system arising at each Newton's iteration for the finite element discretization of this system is solved using a preconditioned Krylov method, the bi-conjugate gradient stabilized (BiCGSTAB) linear iterative method [185]. The incorporated novel block preconditioning technique is based on treating pressure unknowns separately from the temperature and velocity unknowns. A pressure preconditioner is constructed from a factor of the Schur complement of the pressures using Jacobi iteration. The viscous operator is treated using the additive Schwarz method [166]. The number of Krylov iterations required per Newton's method step with increasing numbers of processors is stabilized using a two-level additive Schwarz method. The coarse grid for this step is generated by formulating the problem on a conforming coarse grid. The resulting iterative method is demonstrated to have high parallel efficiency, subject to effective domain decomposition. Robustness and good performance result from using geometric partitioning of the unknowns by the nested sectioning routines supplied in CHACO [91]. The software is implemented using MPI [184] and the PETSc toolkit [12], so as to be readily portable to a variety of parallel computers. The method is shown to be superior to a current state-of-the-art direct method, CFS [126].
The results from this implementation for the viscoelastic flow problem are then presented in Chapter 5. The solution of an extremely large problem is motivated from a review of prior attempts, and the time dependent simulation of flow around a periodic linear array of cylinders is considered. The results obtained in this section are unique due to the high resolution of the mesh considered. The mesh utilized is much finer near the cylinder surface than has been used previously [54,167,174]. This allows the capturing of previously unresolved, detailed physics near the elongational region in the wake of the cylinder. This allows the capture of the purely extensional flow predicted in the wake of the cylinder by [141]. Furthermore, boundary layers near the cylinder surface which was previously averaged into the coarse mesh is observed. The performance of the algorithm is analyzed in detail in terms of both computation speed and parallel efficiency. Finally, the performance of two linear stability codes based on the time dependent code are analyzed.

The results of this thesis are summarized in Chapter 6. There are many possible directions for future research, and these, too, are discussed here.
Chapter 2

The finite element simulation of viscous flows

2.1 Introduction
As discussed in Chapter 1, many properties of engineering interest in a polymer product result from molecular orientations in the polymer which are imparted during processing. The effect of molecular orientation is also observable in the melt properties, causing a marked deviation from Newtonian flow. This deviation is represented in experiment through the observation of polymer solutions. For example, the terminal velocity of a sphere falling under the influence of gravity in a tube filled with a solution of polyisobutylene dissolved in a mixture of tetradecane and polybutene (Boger fluid), deviates twenty percent from the terminal velocity observed in Newtonian liquids [6]. From a processing standpoint, a more relevant example which demonstrates the non-Newtonian behavior of these fluids is die swell. In the limit of inertialess flow in the absence of gravity, a Newtonian fluid passing through a die will expand to 13% larger than the diameter of the die. A polymeric fluid, by contrast, can swell to a diameter more than twice the diameter of the die [113]. This phenomena must be considered in process design, particularly fiber spinning.

While the origin of viscoelasticity is molecular, the modelling of the phenomena is based on continuum mechanics. The spatial length scales of interest are much larger than the molecular length scales. As a result, balance equations can be written in differential form for mass, momentum and energy. The subsequent discretization of these equations results large, linear systems which must be inverted. It is the necessary inversion of these linear systems that motivates the study of parallel methods for solving them.
An overview of methods for solving these linear systems and the manner in which they arise are the topics of this Chapter. While the final goal of this thesis was the application of the developed parallel iterative solver to viscoelastic flow modelling, the solver was first developed for the natural convection problem. First, potential solution methods for these linear systems are explored in Section 2.2. This includes a discussion of both iterative and direct methods, along an introduction to preconditioning. The viscoelastic flow problem is then introduced, with a discussion of how the equations governing viscoelastic flow are formulated for stability in Section 2.3. The spatial discretization of these equations using finite elements is covered in Section 2.4. The natural convection problem is then introduced along with this problem's spatial discretization using finite element in Section 2.5. Temporal discretization techniques are then explored for the viscoelastic flow problem in Section 2.6, giving the necessary background for solving transient flow problems. Because of these method’s relation to stability analysis, methods for exploring the stability of the model equations are described in Section 2.7. In some cases, such as for the natural convection problem considered in this thesis, a steady-state solution suffices. For this reason, Newton’s method is reviewed in Section 2.8.
2.2 Solving Linear Systems of Equations

Most broadly, the algorithms for solving linear systems may be classified as either direct or iterative methods. With direct methods, such as LU-decomposition, the linear system is solved exactly to machine precision after a finite number of operations. By contrast, iterative methods such as Jacobi rely on beginning with an approximation to the solution which is subsequently refined through a number repeated steps. These iterations correct the approximate solution successively to improve it. Each class has a number of advantages and disadvantages.

2.2.1 Direct Methods

Most generally, direct methods are preferable due to their robustness. Their limitations lie in the fact that they become too costly for large problems and that they do not parallelize very efficiently.

The most well-known direct method is probably Gaussian elimination with pivoting. The idea behind Gaussian elimination is that a factorization for the matrix is found such that it may be written as the product of an upper triangular and a lower triangular factor:

\[ A = LU \]  \hspace{1cm} (2.1)

The linear system may be written as:

\[ LUx = b \]  \hspace{1cm} (2.2)

The solution is then accomplished in two steps. First, forward elimination occurs to generate an intermediate result:

\[ Ls = b \]  \hspace{1cm} (2.3)

This is followed by backwards substitution to generate the actual problem solution:

\[ Ux = s \]  \hspace{1cm} (2.4)
\[
\begin{align*}
\textbf{Algorithm 1: Pseudocode for Pivoting LU-decomposition} \\
\text{k} &= 1; k \leq N; \text{k++} \\
&\quad \text{find } p \text{ so } |a_{pk}| = \max\{a_{kk}, \ldots, a_{sk}\} \\
&\quad \text{pivot}(k) = p \\
&\quad \text{interchange } a_{\text{pivot}(k),k} \text{ and } a_{kk} \\
&\quad t = \frac{1}{a_{kk}} \\
&\quad i = k + 1; i \leq N; i++ \\
&\quad a_{ik} = a_{ik} \cdot t \\
&\quad j = k + 1; j \leq N; j++ \\
&\quad \text{interchange } a_{\text{pivot}(k),j} \text{ and } a_{kj} \\
&\quad i = k + 1; i \leq N; i++ \\
&\quad a_{ij} = a_{ij} \cdot a_{kj}
\end{align*}
\]

\[
\begin{align*}
\textbf{Algorithm 2: Pseudocode for Forward Elimination} \\
k &= 1; k \leq N - 1; k++ \\
&\quad \text{interchange } b_{\text{pivot}(k)} \text{ and } b_{k} \\
&\quad i = k + 1; i \leq N; i++ \\
&\quad b_{i} = b_{i} - a_{ik} \cdot b_{k}
\end{align*}
\]

This is combined with column pivoting to lend the scheme numerical stability on finite-precision computers. The algorithms for these steps are shown in Figures 2.1-2.3.

The factorization of sparse positive definite systems have been extensively analyzed using elimination trees and graph approaches, but these systems do not require pivoting,
\[
\begin{align*}
k &= N; k > 0; k-- \\
b_k &= \frac{b_k}{a_{kk}} \\
i &= 1; i \leq k - 1; i++ \\
b_i &= b_i - a_{ik} \cdot b_k
\end{align*}
\]

Figure 2.3: Pseudocode for back substitution which is generally necessary for numerical stability [7,76,78]. Many of the concepts still generalize to algorithms for LU-factorization with pivoting.

As was discussed in Section 3.3, parallelization of iterative methods when used in conjunction with a parallel preconditioner is straightforward because the basic operations parallelize well. This contrasts to parallelizing LU-factorization and the subsequent forward elimination and backward substitution operations.

One method of parallelizing Gaussian elimination, as developed in [122] using a graph approach, involves representing Gaussian elimination as a task list, illustrated in Figure 2.4. This work developed the factorization into independent tasks which could then be divided among individual processors. Examining this task list, the precedence ordering between the two tasks is clear. Because the pivot column should be prepared before the other columns are modified, we see that in this list:

- \( T_{k+1} \) should precede \( T_{kj} \)

Similarly, the update of column \( j \) at step \( k \) should be complete before it may be updated at step \( k + 1 \). This tells us that:

- \( T_{kj} \) should precede \( T_{k+1,j} \)
Two types of tasks:

1- $T_{kk}$
   - Locate the pivot
   - Move the pivot onto the diagonal in the current row
   - Scale all rows below the current row

2 - $T_{kj}$
   - Modify the columns to the right of and below the pivot

Gaussian elimination:

$k = 1; k \leq N - 1; k++$

$$T_{kk}$$

$$j = k + 1; j \leq N; j++$$

$$T_{kj}$$

Figure 2.4: Gaussian elimination described as two tasks

The resulting task graph for this method is shown in Figure 2.5 [118], depicting the interconnection of the two types of tasks for a 5x5 linear system. One common drawback of parallel direct methods can be seen in this task list -- as factorization proceeds, fine grained parallelism, where multiple tasks may be done in parallel, degrades to coarse grain parallelism, where tasks must cooperate. Eventually processors are left idle.

In practice, Gaussian elimination is rarely used for large problems. Instead, frontal or banded formulations, as illustrated in Figure 2.6, are used in single processor environments because of their simplicity and robustness [119]. Multifrontal methods with domain decomposition attempt to fit frontal methods into a hierarchical framework in an attempt to make them better suited for parallelism [19,57,126].

In serial frontal implementations, matrix formulation and LU-decomposition are done at the same time with a window holding the active equations moving through the matrix as shown in Figure 2.6. Once an equation is fully assembled, it is factored. This results in
Figure 2.5: A task graph for Gaussian elimination with N=5
both a decrease in bandwidth and requirements for in-core memory -- necessary due to the overall large storage requirement for these methods. The entire bandwidth must fit in core memory, while the entire band of the matrix is assumed to have entries.

In parallel multifrontal implementations, each processor is assigned a part of the physical domain, similar to as was described in Chapter 3 for the method put forth in this thesis. Each processor then goes through it's assigned subdomain to formulate and eliminate the equations internal to its subdomain. These steps may be performed in parallel, and correspond to parts of the rectangular domain in Figure 2.7 shaded green. The processors then cooperate to solve the equations for the borders of the subdomains, colored yellow, resulting in coarse-grained parallelism. This process continues until the final border region, colored red, is treated. At this point parallelism has completely broken down and all the processors must communicate to generate the final factorization. The breakdown of parallelism which occurs in this system is inherent in parallel factorizations. For example, consider the performance of a state of the art direct solver, Concurrent Factorization and Storage (CFS) [126] for the case of natural convection in a cavity discussed in Chapter 4. In Figure 2.8, the speedup for this frontal, direct method is illustrated. While the initial
Figure 2.7: Factorization of a domain using a multifrontal nested dissection algorithm. Each of the green areas may be solved in parallel. Parallelism then breaks down as processors must cooperate on the border regions.
steps of this factorization occur in parallel, the eventual degradation to coarse-grained parallelism results in low parallel efficiencies for the factorization stage associated with these methods [126].

While some of the factorization may be accomplished in parallel, the situation is considerably worse for the operations of backwards substitution and forward elimination. These operations are inherently sequential and do not exhibit any parallel efficiency. The speedup for these operations is shown in Figures 2.9 and 2.10 for CFS [126], and as can be seen, the addition of processors either has no effect or a negative effect on method performance. At first glance, this may seem to be an insignificant penalty as most of the method's time is spent in factorization. However not all problems are nonlinear in nature. In fact, the Stokes' solves involved in the time dependent formulations put forth later in this Chapter are linear in nature and only require factorization at the first timestep. Subsequent steps would not parallelize.

2.2.2 Iterative Methods

The preconditioner put forth in Chapter 4 is not specific to a particular iterative method. For the purpose of the runs in Chapter 4, the Krylov iterative method BiCGSTAB [185] was selected and used, but there are other methods which may have been considered. Here the question of why this method was selected along with other possible choices are presented.

The term "iterative method" refers to a range of techniques that attempt to obtain successively more accurate solutions to the linear system by developing approximations to the solution at each step. Broadly speaking, there are two classes of iterative methods. Stationary methods are older methods which are generally easily understood and implemented, but are not generally very effective. Nonstationary methods are based on the idea of sequences of orthogonal vectors.
Figure 2.8: Speedups for LU-factorization stage of CFS as a function of the number of unknowns for calculation of natural convection in an enclosed cavity. These speedups are based on computation rate, a metric which increases these values relative to what would be find using speedup based on time for computation [126].
Figure 2.9: Speedups for the forward elimination stage of CFS as a function of the number of unknowns for calculation of natural convection in an enclosed cavity. These speedups are based on computation rate, a metric which increases these values relative to what would be found using speedup based on time for computation [126]. Note that the addition of processors actually hurts the performance.
Figure 2.10: Speedups for the forward elimination stage of CFS as a function of the number of unknowns for calculation of natural convection in an enclosed cavity. These speedups are based on computation rate, a metric which increases these values relative to what would be find using speedup based on time for computation [126]. Note that the addition of processors actually hurts the performance.
Stationary methods, including the common Jacobi, Gauss-Seidel and successive over-relaxation (SOR), are methods which can be written:

$$x^k = Bx^{k-1} + c$$  \hspace{1cm} (2.5)

Here, neither $B$ nor $c$ depend on the iteration count.

The most basic method is Jacobi, where each of the equations is considered independently. This scheme may be written in matrix form as:

$$x^k = D^{-1}(L + U)x^{k-1} + D^{-1}b$$  \hspace{1cm} (2.6)

In this manner each unknown is solved for at the current iteration using the other unknowns evaluated at the previous iteration. Gauss-Seidel, by contrast uses the updated values as soon as they are available. These sequential displacements cause the method to be inherently serial and also generate a dependency in the convergence of the method on the order in which the unknowns are considered. This scheme may be written:

$$x^k = (D - L)^{-1}(Ux^{k-1} + b)$$  \hspace{1cm} (2.7)

SOR may be understood as an Gauss-Seidel but with extrapolation from the previous iterate using a weighted average. This may be written succinctly in matrix form:

$$x^k = (D - \omega L)^{-1}(\omega U + (1 - \omega)D)x^{k-1} + \omega(D - \omega L)^{-1}b$$  \hspace{1cm} (2.8)

If $\omega = 1$ this method reduces to Gauss-Seidel. While it has been shown that SOR only converges for $\omega \in (0, 2)$, it is not generally possible to compute an optimal value of $\omega$ in advance, the choice of which significantly impacts convergence rate. While non-stationary methods are easy to understand and code, they are not generally applicable to the complicated linear systems of interest. Their convergence for indefinite asymmetric systems is not assured in a finite number of steps for sophisticated systems.
Non-stationary methods, by contrast, involve information that changes at each step in the iteration. The constants involved are computed from inner products involving either residuals or other vectors that arise in the course of the iterative method. One popular class of non-stationary methods are Krylov subspace methods or projection methods [51,71,83,144].

With these methods, the linear system, $A$, is written in two pieces:

$$ A = P - Q $$

(2.9)

Here $P$ is the preconditioner, an easily invertible approximation to the linear system $A$, discussed in detail in Section 2.2.3, and $Q$ is the remainder. Iterates are then formed as:

$$ Px^{k+1} = Qx^k + b $$

(2.10)

which may be written:

$$ x^{k+1} = P^{-1}(Qx^k + b) $$

(2.11)

$$ x^{k+1} = x^k + P^{-1}(b - Ax^k) $$

(2.12)

The residual is:

$$ r^0 = b - Ax^0 $$

(2.13)

So the iterates may be written:

$$ x^{k+1} = x^0 + \beta^0(P^{-1}r^0) + \beta^1(P^{-1}A)P^{-1}r^0 + \ldots + \beta^k(P^{-1}A)^{k-1}P^{-1}r^0 $$

(2.14)

Because the $\beta^k$ are constants which are determined by the particular method, the iterates are equal to a linear combination of:

$$ P^{-1}r^0, (P^{-1}A)P^{-1}r^0, \ldots, (P^{-1}A)^{k-1}P^{-1}r^0 $$

(2.15)

plus the initial guess. The iterates are in the $k$-dimensional subspace spanned by these
vectors. The \( k^{th} \) dimensional Krylov subspace corresponding to a matrix \( B \) and a vector \( f \) is written:

\[
\kappa^k(B; f) = \text{span}\{f, Bf, B^2f, \ldots, B^{k-1}f\}
\]  

(2.16)

For unpreconditioned Krylov methods, \( P = I \) and the corresponding Krylov subspace is \( \kappa^k(A; r^0) \). The iterates are then generally of the form:

\[
x^{k+1} = x^0 + d
\]

(2.17)

where

\[
d \in \kappa^k(P^{-1}A, P^{-1}r^0)
\]  

(2.18)

Since there is flexibility in choosing the \( \beta^k \) coefficients, a wide variety of Krylov subspace iterations are possible.

Probably the most commonly known Krylov subspace method is the method of conjugate gradients (CG). This method is effective for symmetric positive definite systems and is shown in Figure 2.11 [14]. This was originally proposed as a direct method [92], but was not considered useful until it was discovered useful as an iterative method. In exact arithmetic, the method converges in the same number of iterations as the dimension of the linear system. In the unpreconditioned method, the \( k \)th iterate is selected from \( \kappa^k(A, r^0) \) such that:

\[
(x^k - x)^T A (x^k - x)
\]

(2.19)

is minimized. This is only guaranteed for symmetric positive definite \( A \). The residuals are \( A \)-orthogonal. The method is attractive because for symmetric \( A \) an orthogonal basis for \( \kappa^k(A, r^0) \) can be constructed from only a three term recurrence. Two two-term recurrences are used in practice, with one updating the residuals using a search direction vector and the other updating the search direction with a newly computer residual.
\[ r_0 = b - Ax_0 \]

\[ p_0 = r_0 \]

Until converged...

\[ j++ \]

\[ \alpha_j = \frac{(r_j, r_j)}{(Ap_j, p_j)} \]

\[ x_{j+1} = x_j + \alpha_j p_j \]

\[ r_{j+1} = r_j - \alpha_j Ap_j \]

\[ \beta_j = \frac{(r_{j+1}, r_{j+1})}{(r_j, r_j)} \]

\[ p_{j+1} = r_{j+1} + \beta_j p_j \]

Figure 2.11: Pseudocode for the method of Conjugate Gradients
Because the linear systems to be studied are not generally symmetric, an orthogonal basis for \( \kappa^t(A, r^0) \) cannot be as easily constructed and the choice of iterative method becomes much more limited. One method shown to be successful for a variety of asymmetric systems and used successfully for the viscoelastic flow problem in Chapter 5 is the generalized minimal residual method, GMRES [156]. This method is shown in Figure 2.12. All previously computed vectors in the orthogonal sequence are retained, and the orthogonal basis for \( \kappa^t(A, r^0) \) is formed explicitly using Arnoldi orthogonalization, a modification of Gram-Schmidt. Like any orthogonalizing Krylov-subspace method, GMRES will converge in no more iterations than the dimension of the linear system. However, in order to reduce the necessary storage and to prevent the operation count from scaling with the number of iterations, the method is commonly restarted periodically. This introduces a the danger that the method might stall. If \( A \) is real and nearly positive definite then metrics exist to select the restart parameter [156] but in general the parameter must be found through experiment.

Because of the long recurrences introduced with GMRES, another approach has been used, replacing the orthogonal sequence of residuals by two mutually orthogonal sequences. This no longer provides a minimization, so few theoretical results are known about the convergence of these methods. One of the earliest methods of this type is the bi-conjugate gradient method (BiCG), shown in Figure 2.13. The search directions then are drawn from two Krylov subspaces, \( \kappa^t(A, r^0) \) and \( \kappa^t(A^T, r^0) \). For symmetric positive definite systems, the method performs approximately as well as CG, but it requires twice the number of operations per iteration. For nonsymmetric systems BiCG has been shown to perform as well as GMRES when there is a significant reduction in the norm of the residual[72]. Convergence behavior can be very irregular, however. Additionally, when the reduction in error slows, BiCG is prone to breakdown. A modification to this method, bi-
\begin{align*}
    r_0 &= b - Ax_0 \\
    \beta &= \|r_0\|_2 \\
    v_0 &= \frac{r_0}{\beta} \\

    \text{Define } \tilde{H} \in \mathbb{R}^{n+1 \times m}, \quad \tilde{H} = 0 \\

    \text{Until converged sufficiently, } j = 1, m \\
    w_j &= Av_j \\
    i &= 1, j \\
    h_{ij} &= (w_j, v_i) \\
    w_j &= w_j - h_{ij}v_i \\
    h_{j+1,j} &= \|w_j\|_2 \\

    \text{Check above for convergence} \\
    v_{j+1} &= \frac{w_j}{h_{j+1,j}}
\end{align*}

Figure 2.12: Pseudocode for GMRES. Note that this is guaranteed to converge in $m$ iterations, and that the update to the solution vector is obtained by a minimizer of the initial error minus $\tilde{H}$. 
\[
\begin{align*}
r_0 &= b - Ax_0 \\
\text{Choose } r'_0 \text{ such that } & \quad (r_0, r'_0) \neq 0 \\
p_0 &= r_0, p'_0 = r'_0 \\
\text{Until converged} \ldots \\
\begin{align*}
j & \quad j++ \\
\alpha_j &= \frac{(r_j, r'_j)}{(Ap_j, p'_j)} \\
x_{j+1} &= x_j + \alpha_j p_j \\
r_{j+1} &= r_j - \alpha_j Ap_j \\
r'_{j+1} &= r'_j - \alpha_j A^T p'_j \\
\beta_j &= \frac{(r_{j+1}, r'_{j+1})}{(r_j, r'_j)} \\
p_{j+1} &= r_{j+1} + \beta_j p_j \\
p'_{j+1} &= r'_{j+1} + \beta_j p'_j
\end{align*}
\end{align*}
\]
conjugate gradient stabilized (BiCGSTAB), illustrated in Figure 2.14, was developed to try to avoid the irregular convergence pattern observed with BiCG [185]. Instead of drawing from $\kappa^t(A^T, r^0)$, the second Krylov subspace from which directions are drawn is $\kappa^t(Q(A), r^0)$, where $Q$ is a polynomial describing a steepest descent update. As a result, BiCGSTAB may be interpreted as the product of BiCG and GMRES(1). A residual vector is locally minimized which leads to a smoother convergence behavior. The danger is that if the GMRES(1) step stagnates, the Krylov subspace is not expanded and the method will break down. This represents a possibility for stagnation in addition to the underlying BiCG method.

In the current work, both BiCGSTAB and GMRES(m) were applied. In the case of natural convection in a fluid-filled cavity, GMRES failed to converge so BiCGSTAB was preferred. Results for viscoelastic flow relative to these two method's performance are presented in Chapter 5.

### 2.2.3 Preconditioning

The rate at which these methods converge depends greatly on the spectrum of the linear system. This is why typically a second matrix, $P$, is used which transforms the original linear system into one with a more favorable spectrum. This second matrix is the preconditioner. A good preconditioner has the property that it improves the convergence of the iterative method sufficiently to overcome the cost of constructing and applying the method. In fact, many iterative methods may fail for complex asymmetric indefinite linear systems without an effective preconditioner.

The preconditioning matrix, $P$, should be similar to $A$, such that the new linear system $P^{-1}A = \hat{A}$ is better conditioned than the original linear system. Solution for the system:
\[ r_0 = b - Ax_0 \]

Choose \( r_0 \) arbitrary

\[ p_0 = r_0 \]

Until converged...

\[ j++ \]

\[ \alpha_j = \frac{(r_j, r_j)}{(Ap_j, r_j)} \]

\[ s_j = r_j - \alpha_j Ap_j \]

\[ \omega_j = \frac{(As_j, s_j)}{(As_j, As_j)} \]

\[ x_{j+1} = x_j + \alpha_j p_j + \omega_j s_j \]

\[ r_{j+1} = s_j - \omega_j As_j \]

\[ \beta_j = \frac{(r_{j+1}, r_0)}{(r_j, r_0)} \times \frac{\alpha_j}{\omega_j} \]

\[ p_{j+1} = r_{j+1} + \beta_j p_j - \beta_j \omega_j Ap_j \]

Figure 2.14: Pseudocode for bi-conjugate gradient stabilized (BiCGSTAB)
\[ \hat{A}x = \hat{b} \]

is attempted where \( \hat{b} = P^{-1}b \).

For the iterative methods considered, the preconditioning matrix never need be explicitly generated. Instead, the effect of its application must be readily computable. The preconditioned linear system must be solved at each iteration, so it must be simple to invert. This leads to a couple of goals for the preconditioner:

- It must be inexpensive to apply
- It must be parallelizable

As we will see, in general, this represents a trade-off. In building a preconditioner, there are two directions to consider following:

- Find a matrix \( P \) that approximates \( A \) in some manner
- Approximately invert \( A \) in some manner

The BCALM preconditioner uses a combination of these methods, as presented in Section 4.2.

**Basic Preconditioners**

The most basic preconditioner is Jacobi. This is simply the diagonal of the matrix used as a point preconditioner. Block jacobi methods can then be derived by partitioning the unknowns in some manner and solving for these blocks of unknowns simultaneously. As in the discussion of stationary iterative methods above, SOR preconditioners can be generated by extrapolating the corrections with a relaxation factor. However, these basic meth-
ods do not tend to yield good results. By far, the most effective preconditioners are the class of incomplete factorizations.

**Incomplete Factorizations**

A factorization is incomplete if, during the course of factorization, certain fill elements are ignored. There are two common criteria for determining if a fill element should be dropped. One may opt to restrict new entries by location or magnitude.

When determining fill by location, there are many possibilities for selecting what locations to keep. Most generally, a set of ordered pairs defines the allowed fill-in [131]. An ILU(k) scheme results, where k is the levels of fill allowed, when this definition is based on the sparsity structure of the original linear system. When k=0, the sparsity structure of the original linear system is used to determine the nonzero locations in the factored matrix. Increasing k results in increasing levels of fill, as will be discussed below.

When no fill-in is used, an ILU(0) is obtained. The fill pattern of the preconditioner is exactly that of the original linear system, as illustrated for a five-point matrix in Figure 2.15 [154]. The algorithm for ILU(0) is shown in Figure 2.17. It is not generally possible to produce $A$ with this $L$ and $U$, so consideration is given to adding more levels of fill to the factorization in order to improve the rate of convergence associated with this preconditioner.

The sparsity structure associated with an ILU(k) with $k \neq 0$ has the fill pattern that would result from ILU(0) applied $k + 1$ times to the original linear system. Another way of considering this is to think of the sparsity pattern for ILU(k) as having the sparsity structure which would result from performing ILU(k-1) and multiplying out the LU. This is illustrated for the case of one level of fill in Figure 2.16 and is obtained in general using the algorithm illustrated in Figure 2.18. While this generally represents an improvement
Figure 2.15: ILU(0) factorization of a five-point matrix, \( A \). The sparsity pattern for the \( L \) and \( U \) are shown to be the same, in sum, as the original matrix. The sparsity pattern for \( LU \) is also shown. [154]
Figure 2.16: ILU(1) factorization of a five-point matrix, A. The sparsity pattern for the L and U are shown to be the same, in sum, as the sparsity pattern resulting from LU and ILU(0). The sparsity pattern for LU with one level of fill is also shown, which would determine the sparsity structure of ILU(2). [154]
\[
i = 2; i \leq N; i++
\]
\[
k = 1; k \leq i - 1; k++
\]
\[
\text{if } a_{ik} \neq 0
\]
\[
a_{ik} = \frac{a_{ik}}{a_{kk}}
\]
\[
j = k + 1; j \leq N; j++
\]
\[
\text{if } a_{ij} \neq 0
\]
\[
a_{ij} = a_{ij} - a_{ik}a_{kj}
\]

Figure 2.17: Pseudocode for ILU(0)

relative to ILU(0) \([18, 46, 164]\), there are still a number of problems with this scheme. While inexpensive, it is not generally possible to predict the amount of fill-in associated with an arbitrary level of fill. Furthermore, ILU(k) is blind to the magnitude of matrix

\[
\text{define } level_{ij} = 0 \text{ for all nonzero locations, otherwise } \infty
\]
\[
i = 2; i \leq N; i++
\]
\[
k = 1; k \leq i - 1; k++
\]
\[
\text{if } level_{ik} \leq p
\]
\[
a_{ik} = \frac{a_{ik}}{a_{kk}}
\]
\[
a_{i,(all)} = a_{i,(all)} - a_{ik}a_{i,(all)}
\]
\[
level_{ij} = \min(level_{ij}, level_{ik} + level_{kj} + 1) \text{ for all nonzeros}
\]
\[
j = k + 1; j \leq N; j++
\]
\[
\text{if } level_{ij} \text{ in row } i \text{ greater than } p, \text{ replace by zero}
\]

Figure 2.18: Pseudocode for general ILU(p)
entries and is very poor for the problems covered in this thesis, in the sense that \( LU - A \) is large.

Alternative ILU methods are based on dropping elements during Gaussian elimination according to their magnitude rather than their locations, resulting in ILU-dt schemes [3,136,188]. The sparsity pattern of the ILU is determined dynamically in this manner. Generically, this could be obtained from a sparse direct solver and simply adding lines that ignore “small” elements. However, one must be careful when generating ILU-dt because factors are being dropped so that ordering impacts the effectiveness of ILU-dt. In general, a nested-dissection ordering has been shown to be near-optimal [38,46,164], and this is the ordering which has been used for the work in this thesis. Another complication is that it is possible to for a zero pivot to occur due to dropping factors that may have previously filled the diagonal at a particular entry. This introduces the need for a pivoting ILU-dt scheme. This helps to remedy the following three possible breakdown scenarios:

- ILU-dt completes normally but has computed an unstable factorization

- ILU-dt encounters a zero pivot

- ILU-dt encounters overflow or underflow due to entry growth in the factor

A scheme for ILU-dt developed by Saad [153,154] was used in the current work. For normalized matrices, in general, a small drop tolerance \( O(1 \times 10^{-5}) \) is effective along with a large pivot tolerance \( O(1 \times 10^{-1}) \). This algorithm is the one which was extensively modified and wrapperized use in the PETSc libraries by the author [12]. The idea of drop-tolerance is coupled with the idea of restricting by fill in this implementation. While widely applicable, the ILU scheme present the same problems for parallelism as the direct methods discussed in Section 2.2.1.
Domain Decomposition
While effective, the serial nature of ILU preconditioners makes them inappropriate for parallel methods. This motivates domain decomposition as a technique for preconditioning. In general, the advantages of domain decomposition methods include[166]:

- Ease of parallelization

- Good performance

- Simple to apply to complex geometries

- Good convergence properties

First, it should be noted that the term domain decomposition means different things in different areas involved in solving PDEs.

Most broadly in parallel computing, domain decomposition refers to the process of distributing data from a computer model in discrete form to the processors of a parallel computer. For the BCALM preconditioner, this is accomplished using the domain’s geometry as described in Section 3.3.3.

When working with asymptotic analysis, domain decomposition refers to splitting the domain into pieces which can be modelled with different equations which are then matched at the boundary. Here, domain decomposition refers to the determination of which PDEs to solve.

In the context of preconditioners, domain decomposition refers to the process of splitting a large linear system into smaller problems which can be solved to generate a preconditioner (solver) for the system of equations on the entire domain.
All three may appear in a particular code, and, as was seen in Section 4.2, the first and third senses of the term are coupled in the implementation of the BCALM preconditioner. So in some sense, the terms multiple means are closely related.

The simplest and oldest domain decomposition method is due to Schwarz in 1870. This was not originally intended as a numerical method, but may be used to solve elliptic boundary value problems on domains that may be expressed as two subdomains as illustrated in Figure 2.19. Consider solving:

$$\nabla^2 x = f$$

(2.21)

on this domain, $\Omega = \Omega_1 \cap \Omega_2$ subject to the boundary condition $x|_{\partial \Omega} = g$ where $\partial \Omega$ denotes the boundary. Internal to $\Omega$, there are artificial boundaries $\Gamma$, which lie on the boundary of the domains $\Omega_i$. As an iterative method, alternating Schwarz starts with an initial guess $x_0^1$. Iteratively for $n = 1, 2, \ldots$ a boundary value problem,

![Figure 2.19: Schwarz's original alternating method figure](image)

97
\[ \nabla^2 x_i^n = f \text{ in } \Omega_i \]  
(2.22)

\[ x_i^n = g \text{ on } d\Omega_i \setminus \Gamma_i \]  
(2.23)

\[ x_i^n = x_i^{n-1} \text{ on } \Gamma_i \]  
(2.24)

is solved for \( x_i^n \). This is followed by a boundary value problem on the second domain,

\[ \nabla^2 x_2^n = f \text{ in } \Omega_2 \]  
(2.25)

\[ x_2^n = g \text{ on } d\Omega_2 \setminus \Gamma_2 \]  
(2.26)

\[ x_2^n = x_1^{n-1} \text{ on } \Gamma_2. \]  
(2.27)

At each half-step, therefore, an elliptic boundary problem is solved on the subdomain \( \Omega_i \),
taking the boundary values on the true boundary \( d\Omega_i \setminus \Gamma_i \) and using the previous approxi-
mate solution on the interior boundary \( \Gamma_i \).

This is the basic concept behind the Schwarz methods. The domain is broken into a
number of pieces on which the problem is solved either independently or interdepen-
dently.

Schwarz domain-decomposed preconditioning algorithms begin by splitting our over-
all domain, \( \Omega \), into a number of sub-domains such that

\[ \Omega = \sum_{i=1}^{P} \Omega_i \]  
(2.28)

where \( P \) corresponds to the total number of subdomains in our decomposition. For notational convenience, we write a restriction operator \( R_i \) which restricts our operator to a partic-
ular subdomain \( i \). The transpose of this operator, \( R_i^T \), is a prolongation operator which
projects the solution of the operator on a particular domain to conform to the overall
domain. That is, \( R_i^T \) locates the local solution properly in the overall domain. If discretiza-
tion of our overall operator can be represented as a linear system \( A \), then the linear system
restricted to one domain can be written:

\[ A_i = R_i A R_i^T \]  \hspace{1cm} (2.29)

The linear problem then solved on each subdomain may be written in terms of these operators:

\[ R_i A R_i^T \cdot R_i x = R_i b \]  \hspace{1cm} (2.30)

Two possibilities exist for how the solutions from the individual subdomains may interact. As in the case of the two-domain example for Schwarz's original problem in Figure 2.19, each processor may use the solution computed for the other domains as soon as it is available. Writing a projection operator

\[ P_i = R_i^T A^{-1} R_i \]  \hspace{1cm} (2.31)

this operator has the effect of restricting the residual to one subdomain, solving the problem on the subdomain, and then extending the correction back to the entire domain. Computing the new values for \( x \) then becomes a serial procedure:

\[ x^{n+\frac{1}{p}} = x^n + P_i(f - Ax^n) \]  \hspace{1cm} (2.32)

\[ x^{n+\frac{2}{p}} = x^{n+\frac{1}{p}} + P_i(f - Ax^{n+\frac{1}{p}}) \]  \hspace{1cm} (2.33)

\[ \ldots \]  \hspace{1cm} (2.34)

\[ x^{n+1} = x^{n+\frac{p-1}{p}} + P_p(f - Ax^{n+\frac{p-1}{p}}) \]  \hspace{1cm} (2.35)

Substituting and grouping terms, this scheme can then be written concisely:
\[ x^{n+1} = x^n + \left[ I - \prod_{i=1}^{P} (I - P_i) \right] (f - Ax^n) \]  

This method is referred to as multiplicative Schwarz because the corrections are multiplied together in this expression of the method.

While the method is inherently serial, multicoloring may be used to extract parallelism from this scheme. As illustrated in Figure 2.20, depending on the decomposition, there should be many subdomains that do not share common points. The solution of these subdomains could be updated simultaneously in parallel. In general, the larger the number of colors, the more convergence deteriorates. If the subdomains are rectangular, then the number of colors required in two dimensions is usually only four, but a few extra are usually used because coloring with the minimum number is difficult. Furthermore, scheduling the domains becomes a more complicated task.

To obtain even more parallelism, consider only updating the solution after each of the subdomains has been gone through. In this case, updates are written:

Figure 2.20: Coloring a rectangular domain with four colors

100
\[ x^{n+1} = x^n + \left( \sum_{i=1}^{P} P_i (f - Ax^n) \right) \] (2.37)

Because the components of the preconditioner are added together, this scheme is known as additive Schwarz. There is no guarantee that this simple iteration even converges. In general, it will not. However, as we are simply seeking a preconditioner for our iterative method, this is often successful, and this scheme is much simpler to implement.

For example, in the case of natural convection in a fluid filled cavity, both additive Schwarz and multiplicative Schwarz were tested early on as a preconditioner for the block matrix \( A \). The number of iterations required to converge a linear system with 13,764 unknowns are reported in Table 2.1. Indeed, multiplicative Schwarz performs better than additive Schwarz. However the difference in the number of iterations required is small, so additive Schwarz was selected for the current implementation of the BCALM preconditioner.

One key property of elliptic operators is that the solution at any point in the domain is influenced by all the boundary values [166]. An iterative method based strictly on alternating Schwarz cannot converge until data from all the boundary affects all of the interior

<table>
<thead>
<tr>
<th>Grashof Number</th>
<th>Additive Schwarz</th>
<th>Multiplicative Schwarz</th>
</tr>
</thead>
<tbody>
<tr>
<td>1E+00</td>
<td>126</td>
<td>144</td>
</tr>
<tr>
<td>1E+02</td>
<td>187</td>
<td>144</td>
</tr>
<tr>
<td>1E+04</td>
<td>139</td>
<td>114</td>
</tr>
</tbody>
</table>

Table 2.1: Number of iterations for BiCGSTAB convergence for single-level BCALM decomposed into 16 domains for a 32x32 discretization of the natural convection in a cavity problem
points. Because of this, we consider adding overlap to our subdomains such that iterations affect more of the domain with each iteration. Furthermore, increasing the overlap makes the problem per-subdomain more resemble the direct solution. In the limit where overlap involves the entire domain, the exact problem is obtained. Perhaps surprising is how quickly the method improves with overlap. Generally an overlap which is on the order of the mesh size is able to dramatically affect the number of iterations required [32,111,166]. This can be understood by considering the Green's function representation of the solution of an elliptic differential equation [166]. Here the solution \( u(x) \) is written:

\[
    u(x) = \int G(x, x') f(x') dx'
\]

(2.38)

For many elliptic PDEs, while \( G \) is non-zero everywhere inside the domain, it rapidly drops off away from \( x' \). A small overlap then can capture this effect.

Still, as a large number of subdomains is introduced, the ability of the individual domains to resolve the long-range dependencies in the domain is reduced. These overlapping methods capture the high frequency solution, each domain solving for the local unknowns. The drawback to this method is that low frequency error still exists, as the single-level Schwarz methods do not resolve the long-range elliptic effects in the operator. To compensate for this, a coarse space is added that captures the long-range effects of the operator. The linear system for the coarse space is then denoted \( A_0 \) and, like the other subdomains, is associated with restriction and projection operators \( R_0 \) and \( P_0 \). Understood in this manner, incorporation into multiplicative Schwarz is straightforward:

\[
    x^{n+1} = x^n + \left[ I - \prod_{i=0}^{p} (I - P_i) \right] (f - Ax^n)
\]

(2.39)

Similarly, additive Schwarz is written simply:
\[ x^{n+1} = x^n + \left( \sum_{i=0}^{P} P_i(f - Ax^n) \right) \]  \hspace{1cm} (2.40)

The coarse space 'subdomain' is treated like any of the others in the method in these implementations.

The choice of coarse space for the operator can be made in a number of ways. For example, because of the natural convection problem's regular geometry, we choose to generate a nested coarse grid and then both formulate the equations over and impose the boundary conditions on this coarse grid. This coarse grid scheme is illustrated in Figure 2.21. The restriction operator for the coarse grid, then, becomes the method by which values are moved from the fine grid to the coarse grid. For a Galerkin coarse grid of this type, this is achieved by simple insertion. The corresponding prolongation is achieved by bilinear interpolation, avoiding costly computation during implementation of this preconditioner. While theoretically sound [166], there are a number of drawbacks to this approach. First of all, this scheme is either difficult or impossible for an unstructured mesh. For a structured mesh, this scheme puts demands on the choice of mesh generator as the coarse mesh must nodally conform to the fine mesh. Furthermore, complicated mapping structures arise in applying this type of coarse space as projecting from the coarse space to the fine space requires that each fine grid point be located in the coarse grid both by location and with bilinear weights. Finally, as the problem is broken into larger numbers of subdomains the performance of the coarse grid decreases. This leads to additional complications as more and more processors are used to solve a large problem.

Another choice of coarse space which has only recently been applied to finite element formulations and parallel computing is the idea of a matrix lumping coarse space [101]. Consider, for example, solving for a single component in the mesh for flow around a cyl-
Figure 2.21: The coarse grid used for the natural convection problem. Restriction is accomplished by insertion while projection is accomplished by bilinear interpolation. While theoretically sound, this method is difficult or impossible for an unstructured mesh and places demands on mesh generator. Complex data structures are generated for locating nodes on the fine grid in the coarse grid. Furthermore, for a fixed coarse grid this degrades as the fine mesh is decomposed.
Figure 2.22: Decomposition of the mesh for flow of fluid around a cylinder for four processors. One equation per variable is written per domain

In the diagram, illustrated in Figure 2.22, decomposed into four subdomains. The idea is that for each subdomain we write one equation. For the overall linear system, illustrated in Figure 2.23, this equation writes the single domain in terms of the contributions from the others. This entry is obtained by lumping together or adding all of the contributions in each block of the linear system. The (Red,Blue) coarse grid matrix entry may be written:

\[ A_0(R, B) = \sum_{i \in R} \sum_{j \in B} A(i, j) \]  

At first glance this may seem a very crude approximation, but it has a number of advantages over Galerkin coarse grids. First of all, it makes no demands on the mesh structure, so it is easily implemented for either structured or unstructured meshes. The bookkeeping requirements are greatly simplified as the only information required to generate the coarse problem is decomposition information which has already been computed. Finally, as the number of subdomains in the decomposition increases, the performance of this coarse space increases. In the limit where there is one node per domain the original problem is regenerated. This is the coarse grid applied to the viscoelastic flow problem in Chapter 5 with very impressive results. In practice, one equation per processor is too crude
Figure 2.23: Splitting of the original linear system into blocks which are summed to generate one equation per processor. For multivariate systems, one equation per variable type per processor is written.

and mixes too many variable types, so one equation per processor per variable type is utilized. For additional details on multilevel additive Schwarz, see [22] and the references therein.

The final piece that needs to be considered in multilevel Schwarz methods is the choice of solver used to invert the subdomain problems. Because none of these methods are iterated to convergence, use of an exact solver, while certainly a possible choice, is probably overkill. Instead, these methods are effectively coupled with ILU-dt type approximate solvers saving both computation cost and memory. The effect of this is studied in Chapter 5.

At this point it is helpful to restate the BCALM preconditioner for viscous flow systems. First, we re-write our viscous flow equations such that the zero block on the diagonal, typically only the conservation of mass, corresponds to the block $D$ in the linear system:

$$
\begin{bmatrix}
A & B \\
C & D
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix} =
\begin{bmatrix}
b_1 \\
b_2
\end{bmatrix}
$$

(2.42)
This structure is then exploited to form a preconditioner for the entire linear system. During each iteration of the preconditioned Krylov method, two inversions of the preconditioning linear system are required. Each of these preconditioning linear system solves, $\tilde{J}u = v$, are implemented in the following manner:

1 - Solve $\tilde{S}u_2 = v_2$ where $\tilde{S} = diag(D - CB)$

2 - Solve $Ar_1 = s_1 - Br_2$ using two-level additive Schwarz and inexact subdomain solves

3 - $r = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}$

Application of the BCALM preconditioner coupled with a Krylov iterative method to the equations for viscoelastic flow is now explored.
2.3 Formulations for the Governing Equations for Viscoelastic Flow

Before considering the finite element formulation of the governing equations, it is important to recognize that there are several criteria necessary to ensure that the resulting numerical method converges as the mesh is refined [176].

First, compatibility refers to the issue of assuring that the numerical method is well posed as the mesh size goes to zero.

Next, if the exact solution is substituted into the discretization and the exact solution is regenerated, then the method is classically consistent. If the solution recovered is only accurate to a particular order, than the order of accuracy retained is the order of consistency of the method.

Finally, there are two issues of stability to consider. First, the physical stability of the equations themselves will ensure that the solution does not go to infinity as time goes to infinity. Numerical stability assures that the magnitude of random disturbances to the steady state will decay.

When it was discovered that the numerical methods for solving the viscoelastic flow equations failed above a critical Deborah number which often decreased with increasing mesh refinement, the methods were seeming not convergent in terms of these criteria. This led researchers to consider the mathematical equation type of the governing equations [106] whereupon it was discovered that the equations’ type could change.

2.3.1 Viscous formulation

The governing equations for viscoelastic flow for an Oldroyd-B fluid are written:

\[-\beta \nabla^2 \mathbf{v} + \nabla \cdot \mathbf{\tau}_p + \nabla p = 0\]  \hspace{1cm} (2.43)

\[\nabla \cdot \mathbf{v} = 0\]  \hspace{1cm} (2.44)

\[\mathbf{\tau}_p + \text{De} \mathbf{\tau}_{p,ij} = -(1 - \beta) \dot{\gamma}\]  \hspace{1cm} (2.45)
where the deviatoric stress tensor is split into the sum of two contributions: one from the polymer, \( \tau_p = -(1 - \beta)\dot{\gamma} \), and another from the Newtonian solvent, \( \tau_s = -\beta\dot{\gamma} \). \( \beta \) is the ratio between the Newtonian solvent viscosity and the total zero-shear-rate for the fluid, calculated as the sum of both the solvent and the polymer contributions.

This viscous form of the equations [150] guarantees the ellipticity of the momentum equation since the solvent viscosity multiplies the Laplacian. However, it is readily observed that as \( \beta \to 0 \) the equation is singular. Furthermore, if \( \beta \ll 1 \) the ellipticity of the numerical approximation may be lost as it is swamped by the other terms in the equation. This has led to the search for alternative formulations.

### 2.3.2 Alternate Formulations

The Elastic-Viscous Split Stress (EVSS) formulation, proposed in [150], seeks to remove \( \beta \) from the elliptic term in the momentum equation. The polymer stress contribution to the stress is split into two pieces by introducing the elastic stress tensor \( \Sigma \).

\[
\tau_p = -(1 - \beta)\dot{\gamma} + \Sigma
\]  
(2.46)

Substituting Eq. (2.46) into the viscous formulation for an Oldroyd-B fluid yields

\[
-\nabla^2 v + \nabla \cdot \Sigma + \nabla \tau_p = 0
\]  
(2.47)

\[
\nabla \cdot v = 0
\]  
(2.48)

\[
\Sigma + De[\Sigma_{(1)} - (1 - \beta)\dot{\gamma}_{(1)}] = -(1 - \beta)\dot{\gamma}
\]  
(2.49)

The Laplacian operator in Eq. (2.47) is now not scaled by \( \beta \), ensuring that this equation is explicitly elliptic in \( v \). The momentum and continuity equations represent an elliptic saddle point problem while the constitutive equation remains a hyperbolic equation in terms of the elastic stress tensor, Eq. (2.46).
Because the $\gamma_{(1)}$ term contains second order derivatives of velocity, the integration of this term by parts requires a boundary condition on the entire domain. This is not possible to specify for a hyperbolic equation such as Eq. (2.49), so a variation on EVSS, the Elastic Viscous Split Stress - Gradient (EVSS-G) method was developed. In the EVSS-G formulation, the velocity gradient in $\gamma_{(1)}$ is taken as another unknown, $G$, and used for all appearances of the velocity gradient in Eqs. (2.47) through (2.49). A constraint equation is added:

$$ G - \nabla v = 0 \tag{2.50} $$

The EVSS and EVSS-G formulations require that the stress be written in terms of $\Sigma$. This can be difficult or impossible for complicated constitutive equations such as FENE-P. This motivated the Discrete Elastic Viscous Split Stress - Gradient (DEVSS-G) formulation. Here, the substitution for the elastic stress tensor $\Sigma$ is made in only the momentum equation. With this substitution, the DEVSS-G equations for an Oldroyd-B fluid are recovered:

$$ -\nabla^2 v + (1 - \beta) \nabla \cdot [G + G^T] + \nabla \cdot \tau_p + \nabla p = 0 \tag{2.51} $$

$$ \nabla \cdot v = 0 \tag{2.52} $$

$$ \tau_p + \text{De} \left[ \frac{\partial \tau_p}{\partial t} + v \cdot \nabla (\tau_p - G^T \cdot \tau_p - \tau_p \cdot G) \right] = (1 - \beta) [G + G^T] \tag{2.53} $$

For the calculations in this thesis, DEVSS-G is utilized.
2.4 Finite Element Formulation for DEVSS-G

Analytic solutions to the DEVSS-G formulation for viscoelastic flow cannot be found analytically. Furthermore, if for a specific problem a solution was found, it is likely that a small change in boundary conditions or problem type would render the solution unusable.

For this reason it is necessary to use numerical methods to solve for the flow kinematics and stress fields. One widely used option is the finite element method (FEM) which is a widely used solution technique for solving problems that arise in computational physics, including fluid dynamics. FEM is a member of the class of weighted residual methods, such that the solution of the governing equations is accomplished in a weak or integrated sense over the domain.

2.4.1 Basic Concepts

It is instructional to consider the application of the finite element method to a simple problem. Much of the material for this Section is taken from [29,103,167]. Here we consider application to a simple diffusion equation in 1-D.

The boundary value problem which must be solved for the unknown \( u(x) \) over the domain \( 0 \leq x \leq 1 \) is:

\[
-\frac{d^2 u}{dx^2} = f.
\]  

(2.54)

This equation is combined with two boundary conditions:

\[
u(0) = 0 \]  

(2.55)

\[
\frac{du}{dx} \bigg|_{x=1} = 0 \]  

(2.56)

This specifies the problem.

By defining two functionals:
\[
\begin{align*}
a(w, v) &\equiv \int_0^1 \left(\frac{dw}{dx}\right) \left(\frac{dv}{dx}\right) dx \\
(w, v) &\equiv \int_0^1 w v dx
\end{align*}
\] (2.57)

and

the model problem may be restated. Now the goal is to find a function \( u \in V \) such that

\[
a(u, v) = (f, v) \quad \forall v \in V
\] (2.59)

Where \( V \) is an arbitrary function space. It makes sense to select \( V \) as being continuous and bounded, such that:

\[
V = \{ v : v \in C^0 \text{ on } [0, 1], v(0) = 0 \}
\] (2.60)

Solving Eq. (2.59) generates the same solution as solving Eqs. (2.54)-(2.56). This statement of the problem is called the weak formulation of the problem whereas the problem as it appears in Eqs. (2.54)-(2.56) is known as the strong form of this boundary value problem.

The weak form is constructed by multiplying the original differential equation by a test function, in this case \( v \), and then integrating over the domain. Neumann boundary conditions are enforced naturally through integration by parts which causes additional terms in our weak form. By contrast Dirichlet boundary conditions are essential boundary conditions which we select our function space to satisfy. It is important to note that our problem statement can take the form of a minimization. In this case it is desired to find \( u^h \in V \) such that:

\[
\frac{1}{2} a(u, u^h) - (f, u^h) \leq \frac{1}{2} a(u, v) - (f, v) \forall v \in V.
\] (2.61)
In practice, the infinite dimensional space \( V \) is replaced by a finite dimensional subspace \( V^h \). This is the Ritz-Galerkin approximation which yields the spatial discretization. Now our approximate variational problem becomes find:

\[
u^h \in V^h \tag{2.62}\]
such that

\[
a(u^h, v) = (f, v) \quad \forall v \in V^h \tag{2.63}\]

A linearly independent set of basis functions, \( \{\phi_i\} \), that span the space \( V_h \) are defined. These functions \( V_h \) can be written:

\[
v(x) = \sum_i c_i \phi_i(x) \tag{2.64}\]

This means that the solution of our representation of the problem, Eq. (2.64), may be written as a linear system:

\[
A \cdot u^h = f \tag{2.65}\]

Entries in the linear system are \( A_{ij} = a(\phi_i, \phi_j) \), \( u_h \) is the vector of coefficients in \( u^h(x) = \sum_i u^h_i \phi_i(x) \) and \( f_i = (f, \phi_i) \).

The finite element method exhibits the ability to treat consistently a variety of boundary conditions and complex domains. It is the linear systems that arise from finite element discretizations that are of interest in this thesis, and further details of finite element background as applied to the viscoelastic flow problem may be found in [44,158,167,176].

For the viscoelastic flow problem, the computational domain is divided into elements on which the equations of motion are discretized using classical finite element methods for the velocities, pressures, stresses, and gradients. The DEVSS-G FEM method is based on
discretizing the equations of motion using methods appropriate for an elliptic saddle-point problem while treating the hyperbolic constitutive equation separately.

2.4.2 Momentum Equation Treatment

Application of the finite element method to DEVSS-G formulation yields a problem in four variables, \{\nu^h, p^h, \tau^h, G^h\}. Because a saddle point of the Lagrange multiplier problem corresponds to the solution of the Stokes-like momentum equation, a solution to the problem exists only if the approximate solutions \(\nu^h \in V^h\) and \(p^h \in S^h\) are selected from appropriate finite dimensional spaces such that a consistency condition is satisfied. This condition is related to the choice of these spaces. This condition, the Ladyzhenskaya-Babuska-Brezzi or inf-sup condition is satisfied if biquadratic basis functions are used for velocity:

\[
\phi(x) \in V^0_h
\]  
(2.66)

and bilinear basis functions are used for pressure [36]:

\[
\psi(x) \in S^0_h
\]  
(2.67)

Therefore, bilinear basis functions are used for all of the variables except the velocities. The mesh discretization, \(T_h = \{t_i, q_i\}\) is defined for the two-dimensional domain \(\Omega\) and consists of a set of conforming, non-overlapping subdomains which are either triangles \(t_i\) or quadrilaterals \(q_i\). The function spaces \(V^0_h\) and \(S^0_h\) are therefore defined as in [174] as:

\[
V^0_h = \{v \in C^0: |v|_{t_i} \in P_2(t_i) \forall t_i \in T_h, v |_{q_i} \in Q_2(q_i) \forall q_i \in T_h\}
\]  
(2.68)

\[
S^0_h = \{v \in C^0: |v|_{t_i} \in P_1(t_i) \forall t_i \in T_h, v |_{q_i} \in Q_1(q_i) \forall q_i \in T_h\}
\]  
(2.69)

The field variables are expressed in terms of these basis functions:
\[
\begin{bmatrix}
u(x, y) \\
v(x, y)
\end{bmatrix} = \sum_{j=1}^{N} \phi_j \begin{bmatrix}
u_j \\
v_j
\end{bmatrix}
\]

(2.70)

\[
\begin{bmatrix}
p(x, y) \\
\tau_{xx}(x, y) \\
\tau_{yy}(x, y) \\
\tau_{xy}(x, y) \\
\tau_{zz}(x, y) \\
G_{xx}(x, y) \\
G_{yy}(x, y) \\
G_{xy}(x, y) \\
G_{yz}(x, y)
\end{bmatrix} = \sum_{k=1}^{N} \psi_k \begin{bmatrix}
p_k \\
\tau_{xx} \\
\tau_{yy} \\
\tau_{xy} \\
\tau_{zz} \\
G_{xx} \\
G_{yy} \\
G_{xy} \\
G_{yz}
\end{bmatrix}
\]

(2.71)

The global position vector \((x, y)\) is mapped isoparametrically onto a local coordinate system \((\xi, \eta)\) using the same interpolating functions.

For an Oldroyd-B fluid, the weak form of the continuity equation corresponding to Eq. (2.48) is written:

\[
\int_{\Omega} (\nabla \cdot \nu^h) \psi^p d\Omega = 0
\]

(2.72)

The constraint, Eq. (2.50), is expressed:

\[
\int_{\Omega} (G^h - \nabla \nu^h) : e_m e_n \psi^G d\Omega = 0, m, n = 1, 2
\]

(2.73)

These are exactly the same as would be recovered for the EVSS-G formulation. The momentum and constitutive equations. Modifications in the formulation for DEVSS-G make the momentum and constitutive equations take on unique forms. The momentum equation, Eq. (2.47), may be expressed in weak form as:

\[
-\int_{\Gamma} \phi^r n \cdot (\nabla \nu^h) \cdot e_4 d\Gamma
\]
\[ + \int_{\Omega} [\nabla \phi \cdot \nabla v^h \cdot e_k + \phi (\nabla v^h)^T \cdot \nabla e_k] d\Omega \]

\[ -(1 - \beta) \int_{\Omega} \phi^v \nabla \cdot (G^h + (G^h)^T) \cdot e_k d\Omega \quad (2.74) \]

\[ - \int_{\Gamma} \phi^v n \cdot (\Sigma^h + p^h I) \cdot e_k d\Gamma \]

\[ + \int_{\Omega} [\nabla \phi^v (\Sigma^h + p^h I) \cdot e_k] d\Omega = 0 \]

\[ k = 1, 2 \]

In the weak formulation for DEVSS-G, the elastic stress is defined by
\[ \tau_p = -(1 - \beta) \gamma + \Sigma \]. This substitution is not made in the constitutive equation, Eq. (2.49), so that in expanded form it is written:

\[ \left( \psi^v, \left[ \tau_p^h + De \left( \frac{\partial \tau_p^h}{\partial t} + v^h \cdot \nabla \tau_p^h - (G^h)^T \cdot \tau_p^h - \tau_p^h \cdot G^h \right) \right] \right) \]

\[ -De(1 - \beta)(\psi^v, [G^h + (G^h)^T]) = 0, \forall \psi^v \in E^h \quad (2.75) \]

Neither the momentum equation nor the constitutive equation are highly modified, illustrating how easily this method is implemented for various constitutive equations.

**2.4.3 Hyperbolic Equations**

While Galerkin’s method is appropriate for treating the momentum equation, the advective nature of a hyperbolic equation means that there are further considerations. With hyperbolic equations there is a characteristic direction associated with the operator. Consider the case of steady convection of a temperature field:

\[ \nabla \cdot \nabla T = 0 \quad (2.76) \]
The temperature field is convected along streamlines of velocity in the domain. Since the
differential constitutive equations are hyperbolic in terms of the stress tensor. Because the
modelling of viscoelastic flow means solving equations with hyperbolic character, a stable
and convergent method for these equations is essential.

The weighted residual equations associated with hyperbolic discretization are not sub-
ject to variation theory such that the weak form is equivalent to a minimization problem.
This means that the Galerkin finite element method is not optimal for solving these equa-
tions. The order of convergence for the Galerkin finite element discretization of hyper-
bolic equations using $k^{th}$ order basis functions is $O(h^{k-1})$ whereas it is $O(h^k)$ for purely
elliptic equations[169]. Furthermore, it is well known that the Galerkin method gives spu-
rious oscillations in the solution profiles for highly convective systems. While there are a
number of methods to add elliptic character to the constitutive equation, two formulations
are considered here. The Streamline-Upwind Petrov-Galerkin (SUPG) method uses
upwind-biased weighting functions to attempt to capture the convection. With Discontinu-
ous Galerkin (DG), the basis functions are not held continuous at element boundaries,
enabling sharp gradients to be accurately represented.

**SUPG Discretization**
A higher order accuracy than can be achieved with Galerkin finite elements may be
accomplished using the SUPG method. This method was introduced and shown conver-
gent in [104] to $O(h^{k-\frac{1}{2}})$ accuracy for linear first-order hyperbolic systems as long as the
dependent variable is square integrable. This is superior to both artificial diffusion (AD)
schemes or simple streamline-upwinding, both of which exhibit $O(h)$ convergence.

With this technique, the weighting functions are taken to be different from the basis
functions. The weighting functions are selected to weight upstream nodes more heavily
than downstream. One form is a modification of the standard Galerkin weighting function:
\[ \psi^h = \psi^h + h \frac{\nabla \psi^h}{|\nabla \psi^h|} \cdot \nabla \psi^h \]  

(2.77)

Normalization assures that the gradient is \( O(1) \) while the "upwinding" term takes into account diffusion along streamlines.

**DG Discretization.**

An improvement to the convergence observed with SUPG can be found with DG. The DG method uses piecewise finite elements which do not share nodes. The approximate solution belongs to the finite element space \( V_k \), which is defined:

\[ V_k = \{ v \in L_1(0,1) : v|_{E_j} \in Q^k(E_j), j=1,N \} \]  

(2.78)

Here, \( E_j \) is a finite element and \( Q^k(E_j) \) is the polynomial space in \( E_j \) of degree \( k \). Because the values at the nodes are not uniquely defined, it is necessary to separate the upstream from the downstream boundaries of each element.

The primary advantage of these methods is that convection dominated equations can be represented without spurious oscillations in the solution, even in the presence of a discontinuity or steep gradient. The robustness is a result of the discontinuous approximation functions which allow the solution to jump in profile across the element boundaries. The cost of this is that for implicit formulations, the inflow boundary must be tracked. However, in the current work DG is used with an explicit solver allowing this update to occur naturally and in parallel.
2.5 Finite Element Formulation for Natural Convection in a Cavity

The two-dimensional natural convection problem refers to the experiment where a fluid, whose density is a function of temperature, is contained between two plates which are held at different temperature. This geometry is shown in Figure 2.24. Because gravity points downward, the lateral temperature gradient drives flow in the cavity due to the imposed temperature gradient. The importance of buoyant forces relative to viscous forces is measured by the Grashof number, which measures the imposed temperature gradient for fixed system geometry and properties.

A representative solution for this problem is illustrated in Figure 2.25. As may be seen, the steady solution consists of a single cell which is symmetric about the center of the cavity. Flow in this case is strong due to the high Grashof number of 10,000. The temperature field deviates slightly from linear because of the low Prandtl number of 0.015. The largest deviation in the temperature field occurs at the center of the cavity where the contours reflect the effects of the flow.

This flow is described at steady-state by the incompressible Navier-Stokes equations with the Boussinesq approximation for the density:

\[
\rho \mathbf{v} \cdot \nabla \mathbf{v} - \mu \nabla^2 \mathbf{v} = \rho \mathbf{e}_y T - \nabla P \tag{2.79}
\]

\[
\nabla \cdot \mathbf{v} = 0 \tag{2.80}
\]

where \( \mathbf{v} \) is the velocity vector, \( P \) is the pressure field, \( \mathbf{e}_y \) is the unit vector in the direction of gravity, \( \rho \) is the density, and \( \mu \) is the viscosity. The energy equation is:

\[
\mathbf{v} \cdot \nabla T - \alpha \nabla^2 T = 0 \tag{2.81}
\]

where \( \alpha \) is the thermal diffusivity. This is defined as:
Figure 2.24: Schematic representation for natural convection in a fluid filled cavity.
Figure 2.25: Solutions of the steady-state natural convection problem corresponding to Pr=0.015 and Gr=10,000. Clockwise from upper left, the solutions shown are for u velocity, v velocity, pressure, and temperature for a case of an aspect ratio of 1, a Prandtl number of 0.015, and a Grashof number of 10,000.
\[ \alpha = \frac{\rho}{kC_p} \]  

where \( k \) is the thermal conductivity and \( C_p \) is the heat capacity.

The governing field equations, Eqs. (2.80)-(2.82), are cast in dimensionless form using scales for velocity and pressure shown appropriate for flows with intense convection and for large Grashof numbers [152]. In this case, the dominant balance is between the inertia and buoyancy forces. Dimensionless variables are denoted with a prime, and defined as:

\[ x' = \frac{x}{H} \]  

(2.83)

\[ v' = \frac{H}{\nu \sqrt{Gr}} v \]  

(2.84)

\[ P' = \frac{H^2}{\rho \nu^2 \sqrt{Gr}} P \]  

(2.85)

\[ T' = A \frac{T - T_c}{T_h - T_c} \]  

(2.86)

Several dimensionless groups arise in the problem statement. The aspect ratio of the cavity, \( A = \frac{L}{H} \), scales the length to the height of the cavity. The Grashof number, \( Gr = \beta g \Delta T H^3 / (\nu^2) \) measures the buoyancy forces in the system relative to the viscous forces, where \( \beta \) is the coefficient of thermal expansion, \( g \) is the acceleration of gravity, \( \nu \) is the kinematic viscosity of the fluid, and \( \Delta T \) is the temperature difference between \( x = 0 \) and \( x = A \). The Prandtl number, \( Pr = \nu / \alpha \), measures the momentum diffusivity relative to the thermal diffusivity of the liquid. The following dimensionless equations result by using the definitions given in Eqs. (2.83)-(2.86), dropping the prime symbols for simplicity:

\[ \sqrt{Gr} \nu \cdot \nabla \nu - \nabla^2 \nu = \sqrt{Gr} e_x \nabla T - \nabla P \]  

(2.87)
\sqrt{\text{Gr}} \ v \cdot \nabla T - \text{Pr}^{-1} \nabla^2 T = 0 \quad (2.88)

\nabla \cdot v = 0 \quad (2.89)

The Dirichlet boundary conditions are illustrated in Figure 2.24 and are specified on the velocity and temperature fields:

\begin{align*}
u &= v = 0 \text{ on } x = 0, x = A, y = 0, y = 1 \quad (2.90) \\
T &= 0 \text{ on } x = 0 \quad (2.91) \\
T &= A \text{ on } x = A \quad (2.92) \\
T &= x \text{ on } y = 0 \text{ and } y = 1 \quad (2.93)
\end{align*}

The Navier-Stokes equations require a mixed finite element formulation for a convergent numerical scheme, and the formulation used for this thesis is based on that by Mehrabi, [36]. As for the viscoelastic formulation presented in Section 2.4, biquadratic basis functions, \( \phi_j \), are selected for the velocity while lower order bilinear ones, \( \psi_j \), are chosen for the pressure. This satisfies the \textit{LBB} compatibility condition [36]. As is standard for natural convection problems, a biquadratic approximation for the temperature is selected [126]. The field variables are expressed in terms of these basis functions as:

\begin{align*}
\begin{bmatrix}
u(x, y) \\
v(x, y) \\
T(x, y)
\end{bmatrix}
&= \sum_{j=1}^{N} \begin{bmatrix} u \\
v \\
T_j
\end{bmatrix}
\quad (2.94) \\
\rho(x, y) &= \sum_{k=1}^{N} \psi_k p_k
\quad (2.95)
\end{align*}

where the index \( j = \{ 1, 2, \ldots, N \} \) spans the nodes of the biquadratic elements while the index \( k = \{ 1, 2, \ldots, M \} \) spans the nodes of the bilinear elements. Choosing bilinear weighting functions for the continuity equation and biquadratic ones for the momentum and energy equations yields the weak forms of Eqs. (2.87)-(2.89):
\begin{equation}
\sqrt{Gr} \int_{\Omega} \phi \cdot \nabla v - \nabla^2 v d\Omega = \int_{\Omega} \nabla \phi \cdot \nabla d\Omega - \int_{\Omega} \phi v d\Omega + \sqrt{Gr} e_y \int_{\Omega} \phi T d\Omega
\end{equation}

\begin{equation}
\sqrt{Gr} \int_{\Omega} \phi v \cdot \nabla T d\Omega - Pr^{-1} \int_{\Omega} \nabla \phi \cdot \nabla T = 0
\end{equation}

\begin{equation}
\int_{\Omega} (\psi \nabla \cdot v) = 0
\end{equation}

In Eqs. (2.96)-(2.98), \( \phi \) are the biquadratic weighting functions and \( \psi \) are the bilinear weighting functions, and the weighting functions are zero on the domain boundaries because of the essential boundary conditions which are imposed there.

The position of the mesh nodal points are approximated with biquadratic interpolants. Substitution of the approximate forms of the field variables into Eqs. (2.96)-(2.98) results in a set of nonlinear algebraic equations in the unknowns, \( \{ u_p, v_p, T_p, P_j \} \), and this system is solved by Newton's method, as described by Figure 2.27 in Section 2.8. The elements of the Jacobian and the right hand side vector are formed at each Newton step using Gaussian quadrature.
2.6 Time Integration

Having discussed the manner in which the equations for viscoelastic flow are treated spatially, temporal discretization is now considered. A mathematical model for a physical process, such as presented in sections 2.4 and 2.5 for viscoelastic flow and natural convection, is a set of partial differential equations expressed in the general form of:

\[ \frac{\partial \mathbf{u}}{\partial t} = \mathbf{F}(\mathbf{u}(\mathbf{x}, t)) \]  \hspace{1cm} (2.99)

where \( \mathbf{u}(\mathbf{x}, t) \) is the vector of field variables, \( \mathbf{x} \) is the position vector, \( t \) is the time, and \( \mathbf{F} \) is a nonlinear differential operator. This equation is defined on a domain \( \Omega \) for some time with initial conditions at \( t = 0 \) and boundary conditions on the boundaries of the domain. Solution of Eq. (2.99) requires a full simulation in space and time.

In this Section two semi-implicit time integration techniques are presented: the theta method and fourth order Runge-Kutta. As will be seen the choice of timestepping scheme effects the choice of spatial discretization and therefore the parallel implementation.

The spatial discretization of our viscoelastic flow problem using the techniques discussed in Section 2.4 yield a linear set of differential algebraic equations which may be expressed:

\[ \mathbf{M} \cdot \frac{d\mathbf{u}}{dt} = \mathbf{A}(\mathbf{u}) \]  \hspace{1cm} (2.100)

Here, \( \mathbf{u} \) is the vector of unknowns, \( \mathbf{u} = [\tau_p, v, p, G]^T \), and for inertialess flow \( \mathbf{M} \) has the entries \( \mathbf{M} \equiv \text{diag}[De, 0, 0, 0]^T \), as the time dependence is restricted to the stresses. This is combined with a set of consistent initial conditions for the modified Stokes problem using an arbitrary polymer stress. The implementation of parallel timestepping schemes presented in Chapter 5 are based on models by Smith [175], so much of this Section has been taken from this source.
2.6.1 θ-method Timestepping

The first method considered here is the operator splitting scheme known as the θ-method. Under this method, the momentum and continuity equations are decoupled yielding more manageable equation sets that reduce the cost of time integration [159,160,176]. Here, we consider the θ-method as applied to the full set of nonlinear equations describing viscoelastic flows as was done in these prior work. The θ-method operator-splitting has also been applied to the linearized equations for the perturbations of the unknowns to two- and three- dimensional disturbances [175].

Each θ-method time step consists of three substeps as illustrated in Figure with the overall operator broken into two operators: \( A_1 \) and \( A_2 \) where the overall operator is recovered as the sum, \( A = A_1 + A_2 \)

\[
M \cdot \frac{u^{n+\theta} - u^n}{\theta \Delta t} = A_1(u^{n+\theta}) + A_2(u^n)
\]

\[
M \cdot \frac{u^{n+1-\theta} - u^{n+\theta}}{(1 - 2\theta)\Delta t} = A_1(u^{n+\theta}) + A_2(u^{n+1-\theta})
\]

\[
M \cdot \frac{u^{n+1} - u^{n+1-\theta}}{\theta \Delta t} = A_1(u^{n+1}) + A_2(u^{n+1-\theta})
\]

The two operators are then:

\[
A_1 \equiv \begin{bmatrix}
-\omega \tau_p \\
-\nabla^2 \nu + \nabla \cdot \tau_p + \nabla p + (1 - \beta) \nabla \cdot (G + G^T) \\
\nabla \cdot \nu \\
G - \nabla \nu
\end{bmatrix}
\]

\[
A_2 \equiv \begin{bmatrix}
-(1 - \omega) \tau_p D e(\nu \cdot \nabla \tau_p - \{ \tau_p \cdot G \} - \tau_p \cdot G^T) \\
-(1 - \beta)(G + G^T)
\end{bmatrix}
\]

126
\[ n\Delta t \quad (n+\theta)\Delta t \quad (n+1-\theta)\Delta t \quad (n+1)\Delta t \]

Figure 2.26: Schematic for the splitting of a timestep \([n\Delta t,(n+1)\Delta t]\) using the theta method.

The splitting parameter is chosen \(\omega \equiv (1 - 2\theta)/(1 - \theta)\) as published in [176]. The resulting \(\theta\)-method scheme is first-order accurate in time.

2.6.2 Fourth-Order Runge-Kutta Timestepping

An alternative time integration approach is based on the forth-order Runge-Kutta method. Using this approach, the kinematic variables and the pressure are separated from the equations for the stress, resulting in an ordinary differential equation for the stress. The kinematic variables and the pressure are solved from a modified Stokes' problem. These time integrators have been applied to piecewise linear DG spatial discretizations of the linear hyperbolic test problems, [45].

The algebraic, kinematic variables are first separated from the differential expression for the stress. This results in a set of ODEs. Rewriting Eq. (2.100):

\[
M \cdot \frac{d\tau_p}{dt} = \frac{1}{De} f(\tau_p, v, G) \quad (2.103)
\]

\[
g(\tau_p, v, p, G) = 0 \quad (2.104)
\]

in Eq. (2.103), \(f\) is the discretization of the constitutive equation, and \(g\) is a Stokes-like problem for the kinematic variables resulting from the discretization of Eqs. (2.72)-(2.74). Equation (2.104) is invertible for the kinematic unknowns, so this may be written:
\[
\begin{bmatrix}
\nu \\
p \\
G
\end{bmatrix} = g(\tau_p)
\] (2.105)

Equation (2.105) is then substituted into Eq. (2.103). This allows us to write an function in terms of the differential variable alone, as is necessary for the application of Runge-Kutta. We define a new function, \( f \):

\[
f(\tau_p) \equiv M^{-1}f(\tau_p, g(\tau_p))
\] (2.106)

Fourth order Runge-Kutta may then be written:

\[
\begin{align*}
k_1 &= \Delta t f(\tau_p^n) \\
k_2 &= \Delta t f(\tau_p^n + \frac{k_1}{2}) \\
k_3 &= \Delta t f(\tau_p^n + \frac{k_2}{2}) \\
k_4 &= \Delta t f(\tau_p^n + k_3)
\end{align*}
\] (2.107)

\[
\tau_p^{n+1} = \tau_p^n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\]

Each step in Runge-Kutta then requires the implicit solution of a Stokes-like problem for the kinematic variables. The constitutive equation is then evaluated explicitly for the stresses.

Both of these timestepping schemes are implemented in parallel in Chapter 5, and details about the combinations utilized are explained in more detail there. It should be noted that the Courant number for the fourth order Runge-Kutta timestepping scheme is slightly than it is for the \( \theta \) method -- 0.3 versus 0.4. As a result, larger timestep may be taken with the \( \theta \) method, which also requires two fewer Stokes solves per timestep. However, fourth order Runge-Kutta is fourth order accurate so it may be preferable if high accuracy is desired.
2.7 Studying the Stability of Viscoelastic Flows

Another problem of interest in our research group is the prediction of the onset of flow instability via the analysis of the steady-state solution to two- and three-dimensional perturbations. This may be done with DEVSS-G along with whichever discretization scheme is selected for the constitutive equation for the stress and with whatever temporal discretization is selected.

To study the response of the system to two dimensional perturbations, the solution is written as the sum of the steady state solution plus a transient, perturbation:

\[ u = u_{ss} + u_p \]  \hspace{1cm} (2.108)

When this is substituted into the equations governing viscoelastic flow, a new linear system is recovered in terms of the perturbed quantities with the steady state values as data.

\[
\begin{bmatrix}
De \frac{d \tau_p}{dt} \\
0 \\
0 \\
0
\end{bmatrix} = \begin{bmatrix}
\tau_p + De \tau_{p,1} + (1 - \beta)(G + G^T) \\
\nabla^2 \nu - \nabla \cdot \tau_p - \nabla p - (1 - \beta) \nabla \cdot (G + G^T) \\
\nabla \cdot \nu \\
G - \nabla \nu
\end{bmatrix}
\]

(2.109)

\[ \tau_{p,1} = \nu_{ss} \cdot \nabla \tau_p + \nu \cdot \nabla \tau_{p,ss} - \tau_{p,ss} \cdot G - \tau_p \cdot G_{ss} - (\tau_{p,ss} \cdot G) - (\tau_p \cdot G_{ss})^T \]  \hspace{1cm} (2.110)

By studying the behavior of the perturbed variables the stability of the flow to the perturbation is learned.

Furthermore, by writing the disturbance as in Eq. (2.111) it is possible to study the two dimensional base case response to a disturbance in the third dimension. The perturbed quantities are represented by finite elements just as the original base case, so application of either timestepping scheme may be applied. These calculations, too have been parallelized and results are summarized in Chapter 5.
\[
\begin{bmatrix}

v(x_1, x_2, x_3, t) \\
p(x_1, x_2, x_3, t) \\
G(x_1, x_2, x_3, t) \\
\tau_p(x_1, x_2, x_3, t)
\end{bmatrix}
= \begin{bmatrix}

v''(x_1, x_2) \\
p''(x_1, x_2) \\
G''(x_1, x_2) \\
\tau_p''(x_1, x_2)
\end{bmatrix}
+ \varepsilon \text{Re}
\begin{bmatrix}

\varphi(x_1, x_2, t) \\
\dot{\varphi}(x_1, x_2, t) \\
\dot{G}(x_1, x_2, t) \\
\dot{\tau}_p(x_1, x_2, t)
\end{bmatrix} e^{-i\kappa s}
\]

(2.111)
2.8 Newton’s Method

While methods for the full simulation in space and time for the set of partial differential equations resulting from a mathematical model for a physical process have been presented in Section 2.6, sometimes only the steady-state solutions are required. In that case the time derivatives are zero and a spatial simulation for \( F(u(x)) = 0 \) suffices. In this thesis, this is the case for the natural convection in a cavity flow considered in Section 2.5.

A steady state solution procedure is illustrated in Figure 2.28 using a finite element/Newton algorithm. As illustrated in this Figure and described in Sections 2.4-2.5, partial differential equations are discretized and boundary conditions are applied with finite elements. For steady-state problems, this results in a nonlinear system of algebraic equations, written:

\[
F(u(x)) = 0
\]  
(2.112)

where \( F \) is the now discretized nonlinear differential operator. Newton’s method is used to solve this nonlinear set through iteration as illustrated in Figure 2.27. In this Figure, \( F_k \)

\[
k = 1, k \leq k_{\text{max}}, k++
\]

- solve \( F_k \delta u = -F_k \)
- \( u^{k+1} = u^k + \delta u^k \)
- check for convergence

Figure 2.27: Newton’s method used to converge the nonlinear equation set
Figure 2.28: Steady state solution of the nonlinear operator associated with a finite element discretization by Newton's method
is the Jacobian matrix at step $k$ with the entries:

$$\{ F^k_u \} = \left( \frac{\partial F}{\partial u} \right)^k$$

(2.113)

The correction vector to the solution at step $k$ is $\delta u^k$. Newton's method may be derived from a Taylor series expansion of the residual, keeping only the linear terms:

$$F^{k+1} \equiv F^k + F^k_u \delta u^k = 0$$

(2.114)

If the initial guess $u^0$ is close to the actual solution, this method converges quadratically.

A linear system of equations needs to be solved at each Newton iteration step.
Chapter 3

Parallel Processing for Finite Element Simulations

3.1 Introduction
The need for a new numerical solver technology was presented in Section 1.3 based on the intractability of three-dimensional problems due to memory and speed limitations with current methods. In Section 1.4, physical and economic arguments were put forward to illustrate the necessity for parallel computing. The potential feasibility of these problems using iterative methods was then established. Consideration should also be given to the impact that a parallel iterative method can have on the required execution time of a model problem.

Consider, again, solution of the flow around a periodic array of cylinders in three dimensions. Assuming that we require 100 steps for solution of the problem obtained from the largest mesh in Section 1.3, current frontal technology would require a large number of operations, measured in millions of floating point operations (MFLOP) as discussed in Section 3.2.1. Using the operation count for LU-decomposition of a banded matrix gives

\[ Nb^2 \times 100 = 6.5 \times 10^9 \text{ MFLOP} \]

Solving this problem on a single-processor computer capable of 50 MFLOP/s, corresponding approximately to a 500 Mhz Pentium III machine or a single node of an IBM SP2 [52], would then consume 36,111 hours or 4.2 years of computational time. By contrast, a properly preconditioned iterative method will need to operate only on the nonzero portion of the linear system and will require a number of iterations far fewer than then dimension of the linear system, typically \( O(100) \) [166]. A conservative estimate can be made then by suggesting that the number of operations required per solution will be \( O(100 \times Nb) \). Using an iterative method to solve this system would then require
\[ N_b \times 100 \times 100 = 1.9 \times 10^8 \text{ MFLOP}. \] (3.2)

Solving this system on the same serial computer using an iterative method would then require approximately 778 hours. This is still far too long to wait for a potentially useful result. However, if we assume that the iterative method can be implemented in parallel at perfect efficiency, then, on a 32 processor parallel machine composed of these processors, this problem, which takes years using conventional technology can be done in around 24 hours! This startling difference is the reason for interest in parallel methods. Moore’s law is commonly put forward as a reason to not become involved in the complications associated with parallel computing [133]. After all, if the inherent speed of processors is doubling every year to 18 months (select your favorite interpretation), there seems little reason not to assume that whatever problem you wish to solve will become tractable in the near future [133].

First of all, this does not change the fact that by using parallel computing you have available to you more CPU horsepower than would otherwise be available to you at any given time. Furthermore, there are physical limits, such as the speed of light, that limit how fast a particular processor can handle data. Additionally, there is the so-called “Moore’s second law.” -- This corresponds to Gordon Moore’s observation that, while the speed of CPUs has been doubling every 18 months, the cost of building semiconductor fabrication plants is doubling every 3.5 years [133]. As a result, the price per transistor has been predicted to bottom out in the first decade of this millennium, causing an economic end to Moore’s law. This would follow a pattern which has been observed in many other aspects of life. After many years of exponential growth in their speed, commercial airliners’ top speeds have been frozen at around Mach 0.6 since the 1960s. While faster speeds are possible, supersonic airline travel has never become exceedingly profitable. A similar fate likely awaits the transistor density of microchips.
However, this author believes that the speed of computation will continue to increase even after the number of transistors per chip ceases to double every year or so. In his novel, *Faster: The Acceleration of Just About Everything*, James Gleik explores the periods of exponential growth observed in various human endeavors, for example, the speed of car travel. He observes that, when these metrics level off, that, in general, underlying performance continues to increase: In the example of air travel, we no longer see an acceleration in airplane speed but are be better connected to people overseas than we have ever been in the past, by fax, phone, email, or video conference. The “speed” with which we can commune with Europe has continued to increase. Similarly, we will continue to see the speed of computation increase as parallelism becomes more and more highly exploited.

The purpose of this Chapter is to present some of the hardware and software issues associated with parallel computing. First, parallel computing hardware and the move towards commodity computing clusters is explored in Section 3.2. Software issues, including message passing techniques, data distribution for parallelism, and performance metrics are considered in Section 3.3. Combined with the discussion of iterative methods and preconditioners in Section 2.2, this sets the background for the novel parallel preconditioner presented in Chapter 4.
3.2 Parallel Computing Hardware

Complex scientific computing has traditionally been carried out on large and expensive supercomputers or specialized workstations. A few years ago, the steady increase in speed of the Intel Pentium chip, among others, signaled an upcoming paradigm shift away from specialized workstations to commodity computation. It is only now being recognized that the speed, memory, and I/O requirements of scientific computing can be done using commonly available processors and parts.

Underlying this trend is a shift in the perception of the target granularity of parallel systems. Parallel computer grain size is determined by the amount of memory per processor [68]. A coarse-grained parallel computer is one which has much memory per processor, while a fine-grained one has little memory per processor. The trend has been towards coarse-grained parallelism in research, as the amount of memory available to a particular processor has risen dramatically.

For example, each node of an Intel iPSC/860 hypercube parallel computer, introduced in 1987, had 8 megabytes (MB) of memory and achieved 10 MFLOP/s on the LINPACK DGEFA 1000x1000 benchmark discussed in Section 3.2.1 [52,126]. By 1992, the Connection Machine CM-5 had 32 MB of memory per processor, and achieved 22.1 MFLOP/s on the LINPACK DGEFA 1000x1000 benchmark [52,124]. The commodity cluster used in Chapter 5 for viscoelastic simulations in this thesis, installed in 1999 and described in Section 5.3.1, has 500 MB of memory and is capable of approximately 50 MFLOP/s on the LINPACK DGEFA 1000x1000 benchmark. The trend, therefore, is towards coarser-grained computers with increasingly large amounts of memory relative to processor speed. This is inline with the predictions of artificial intelligence researchers for the requirements of intelligent machines. Hans Moravec, for example, calculates that a human-equivalent computer would require a computational speed of 10 trillion operations per second and a
memory of 36 terabytes [134]. For the commodity cluster described in Section 5.3.1 to maintain the same memory ratio a human brain, it would need to have 4 gigabytes of memory, almost an order of magnitude more. This suggests that the trend towards increasingly coarse granularity is likely to continue.

3.2.1 Performance

Advances in processor speed alone would be insufficient to motivate interest in commodity computing. It is important to also recognize that advances in shared memory, allowing processors to address a single block of transparently, and networking permit multiple processors to be brought to bear more effectively in parallel than in the past. In order to characterize these advances, it is useful to consider benchmarks for processor and communication speed. The purpose of benchmarks is performance prediction. They are used by corporations and government agencies as a basis for capability design, by computer manufacturers to predict product value, and by individuals to gauge value when evaluating purchases.

In this thesis, benchmarks are used to compare parallel systems in terms of processing speed and communication speed. Processing speed is typically quoted in millions of floating point operations per second (MFLOP/s) while communication speed is measured in a bandwidth of megabytes per second (MB/s).

In any given year, dozens of processor performance benchmarks may be used to compare computation speed. A good survey which benchmarks the processor performance benchmarks has been done by Gustafson and Todi in [89]. In this thesis, a linear algebra benchmark which has been used since 1979 for measuring CPU performance, the LINPACK benchmark suite, is utilized for comparisons [52]. Several types of benchmarks are included in this suite, ranging from benchmarks based on solving linear systems of equations, to ones based on eigenvalue problems, linear least squares and singular value prob-
lems. It is the former which is most commonly used, in a module known as DGEFA [52]. Several variants of this module have been used, based on the size of the linear system solved, and in this thesis, a 1000x1000 linear system benchmark is used. This is referred to as the LINPACK DGEFA 1000x1000 benchmark problem [52].

The bandwidth available to a processor for communication with other processors is another performance metric of interest to researchers developing parallel computer systems. This field is much less developed than that for processor speed, but one of the earliest attempts to benchmarks MPI communication speed was done by Mucci, et al. in the MPBench package [135]. The MPBench package is a suite of benchmarking routines to evaluate the performance of MPI communication speed on both shared memory and networked systems. For the purpose of comparison in this thesis, the maximum bandwidth reported by the bandwidth of send test is used to characterize communication speed [135].

Example values for both of these benchmarks have been tabulated in Table 3.1, below. Note that the first row in this Table, the Sun HPC 5000, corresponds to the system used for the results obtained in Chapter 4. This system is described in detail in Section 4.3. The

<table>
<thead>
<tr>
<th></th>
<th>Processor Speed (LINPACK DGEFA 1000x1000) MFLOP/s</th>
<th>Communication Bandwidth (MPBench Bandwidth of Send) KB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun HPC 5000 with Gigaplane shared memory</td>
<td>52.9</td>
<td>100,000</td>
</tr>
<tr>
<td>500 MHz Pentium III Xeon with Gigabit Ethernet</td>
<td>58.35</td>
<td>30,000</td>
</tr>
<tr>
<td>IBM SP2 with high performance switch</td>
<td>44.88</td>
<td>32,000</td>
</tr>
<tr>
<td>Single HP J2240</td>
<td>19.5</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Table 3.1: Typical processor speeds and communication bandwidths measured using the LINPACK DGEFA 1000x1000 benchmark [52] and the MPBench Bandwidth of Send benchmarks [135].
second row corresponds to the system used for the viscoelastic simulations in Chapter 5, and is described in detail in Section 5.3.1.

3.2.2 Commodity Computing

The LINPACK suite of benchmarks gives insight on the current state of commodity computing. On the LINPACK DGEEA 1000x1000 benchmark problem, the IBM SP2 in the Department of Electrical Engineering is capable, on a single node, of sustaining 58.35 MFLOP/s [52]. A Pentium Pro released in 1997 running at 200 Mhz under Linux, by comparison, is capable of 19.73 MFLOPs [52]. A newer Pentium III released in 1999 with a 500 Mhz clock sustains 44.88 MFLOPS [52]. Thus, with a 500 Mhz Pentium III, one can obtain 75% of the performance of an IBM SP2 at a significantly lower cost per processor.

Networking, memory, and disk prices have all fallen in tandem, allowing the assembly of potentially powerful system at extremely modest cost. At the time of this document’s writing, a 500 MHz Pentium III with 1 GB of memory and 9 GB of disk space is approximately $3000 [48]. A gigabit networking switch capable of handling 18 nodes costs around $30,000 and gigabit NICs are around $150 [100]. A sixteen node parallel computer could be assembled from commodity parts for under $100,000 which could outperform a 12 node IBM SP2 costing over $1,000,000 [94]. The numerical experiments in Chapter 5 bear out that this technology is quickly maturing and that this type of performance is now possible.

The move towards standardized parallel computing platforms has been occurring for some time. Early computers and platforms required the use of specialized languages for parallelism. For example, the Intel IPSC/860 hypercube [97] used a communication library called “NX” which the user had to rely on for programming parallel codes [98]. The Thinking Machines parallel computers used “Connection Machine Fortran” as the vendor-supplied language [179]. As these machines went out of production, codes written
for them became obsolete as many of the features defined as standard in these languages were not available on other platforms. In 1992, to help remedy this problem, the Message-Passing Interface Forum was formed to define a standard for message passing systems on parallel computers [184]. While agreement on the first standard was not achieved until years later, MPI became the de-facto standard as it was quickly adopted by various computer manufacturers. On current supercomputers, MPI performs generally as well as the "vendor-specific" language and in many cases is the 'official' communication language of the platform [69]. This portability gives a programmer insurance that his code will not become obsolete.

Availability of MPI is one factor enabling the construction of commodity supercomputers and code portability. Another is the move away from specialized topologies for supercomputer architectures. Hypercube, ring and trees architectures have all been popular for periods of time, particularly the former [21]. This meant that codes had to be written not only to minimize communication but also to control where their messages were communicated to. In the past several years, networking has advanced such that either randomized or all-to-all architectures are the most dominant models for modern parallel computers [1,5]. For the coarse-grained parallel computers which are commonly utilized today, code is no longer generally designed to track the 'nearest neighbor' to a node, leading to code that is more inherently portable.

Commodity PC clusters possess a number of advantages over their supercomputer counterparts. Low cost is clearly an advantage. Furthermore, they are easily adapted to workgroup demands, easily reconfigured between serial queues and parallel systems. Finally, because they are composed from commodity parts, they less susceptible to the effect of specialty supercomputer vendors either going out of business (Thinking
Machines Corporation, FPS, MasPar Computer Corporation, Kendall Square Research) or being bought by other vendors (Cray, Convex).

The performance of commodity parallel computers is only now reaching critical mass. Early work with Beowulf-class systems with 16-64 Pentium nodes showed their cost effectiveness using 100-BaseT networking [186]. Later work established that commodity computing through $1,000,000 should be justifiable [17]. In the past year, the effectiveness of commodity supercomputing has been established out through 1024 processors by a Gordon Bell prize-winner from Barry Smith's research group at Argonne National Labs [5]. The topology of our commodity cluster resembles that of the Gordon Bell prize-winner, but on a much smaller scale, as described in Section 5.3.1. Both consist of dual-processor SMP Pentium III machines where one processor is used for computation and the other for communication. Where ASCI Red is connected with a proprietary interconnect, we have utilized Gigabit Ethernet for the high efficiencies obtained in Chapter 5. Furthermore, using both processors for computation and allowing them to contend for the network interface card has been show to improve overall performance while decreasing efficiency. This is later shown in Chapter 5.
3.3 Implementation of a Parallel Solver
Having discussed some of the hardware issues associated with parallel computers, we now explore some of the software implementation issues encountered. The construction of a parallel solution method for mathematical models governed by partial differential equations requires the selection of several inter-dependent choices. The selection of a spatial discretization method, such as described in Sections 2.4 and 2.5, a time discretization method, described in detail in Sections 2.6 and 2.7, linear solution method, examined in Section 2.2, and target computer architecture, as described in Section 3.2, all interact to constrain the development of a parallel algorithm.

We consider the mathematical model for a physical process as a set of partial differential equations expressed by some nonlinear differential operator. This form is represented here for the variable $u(x, t)$ as

$$\frac{\partial u}{\partial t} = F(u(x, t))$$  \hspace{1cm} (3.3)

where $F(u)$ is the nonlinear differential operator. This equation is defined on some domain for some time with initial conditions. The solution of Eq. 3.3 requires either full solution in space and time, as described in Section 2.6, or steady state solution, as described in Section 2.7. In either case, the solution of the discrete problem requires the solution of a linear system, which is the most time-consuming part of the simulation, as described in Section 1.3.

Iterative methods for accomplishing this linear system solution were reviewed in Section 2.2. These methods, without preconditioning, require scalar-vector products, matrix-vector products and vector dot products as time consuming parts of their implementation. The remainder of this Section provides background for how data distribution is performed so that these operations may be accomplished in parallel. First, message passing is dis-
cussed in Section 3.3.1. This is followed by a detailed description of how matrices and vectors are actually distributed in parallel in Section 3.3.2. This distribution is controlled by a domain decomposition, spectral nested dissection, which is described fully in Section 3.3.3. Finally, parallel efficiency metrics are reviewed in Section 3.3.4.

3.3.1 Message Passing

As was first discussed in Section 1.6, the most basic question regarding message passing is whether or not to explicitly pass messages or not. Implicit, data-parallel languages exist, such as Sisal, SR, or Fortran 90, which allow the user to simply declare what ranges of a vector or matrix are distributed in parallel [49,70,157]. The resulting code is referred to as threaded because the computer program begins as a serial instruction sequence, or thread, which in turn spawns additional threads [95]. Consider the execution of a simple computation, represented by the computation graph in Figure 3.1. Each node in the graph represents a computation within a thread, with the dashed edges representing the forking of child threads, while the solid edges represent parent-child pair re-joinings. These threads are then scheduled by the operating system. If a Last-In-First-Out (LIFO) stack is utilized, the nodes are executed in a depth-first order [137]. For this example, this would result in a maximum of three active threads available for parallelism. If a First-In-First-Out (FIFO) stack is utilized, the nodes are executed in a breadth-first order [137]. In this example, this would yield a maximum of seven active threads available for parallelism. This type of code takes the burden off of the user for dictating communication, but, as was shown in Section 1.6, it is generally at the expense of performance for complicated codes. Furthermore, unless the threads scheduler is designed carefully, even potentially highly parallel algorithms will suffer poor performance [137].
Figure 3.1: Example computation graph for a threaded code. The dashed arrows represent the fork of a child thread, while the left-to-right solid edges represent a join between parent and child. Vertical arrows represent serial dependencies.
With MPI, communication is asynchronous and fits the Multiple-Instruction-Multiple-Data model [21,184]. A call to the MPI routine “MPI_Get_rank” returns the processor’s position in the parallel machine. Based on it’s ID, the program is then free to branch and do whatever it has been instructed to do, without regard for the other processors. The solver presented in Chapter 4 of this thesis does this often, periodically synchronizing with a call to “MPI_Barrier” which instructs all processors to wait until each of them reaches an “MPI_Barrier” command. Communication is done asynchronously whenever possible, and wait commands are introduced only when communication of a particular type must be finished before proceeding.

While it is more difficult to program in parallel using MPI than using a data-parallel language, the resulting code is faster and more portable than a threaded code. Additionally, the parallel structures used in this code lend it some data-parallel readability, primarily through the use of the Portable, Extensible Toolkit for Scientific Computation, PETSc [12].
3.3.2 Parallel Structures

As was shown in Section 2.2.2, the primary operations that compose the backbone of an iterative solver for linear systems are scalar-vector, vector-vector and matrix-vector products. This Section will show how proper data distribution allows these operations to be accomplished in parallel, describes the actual structures used to store matrices and vectors, and points to the need for domain decomposition in these layouts.

The basic idea behind data distribution for parallel layouts is that both matrices and vectors are distributed by rows to the processors, such that the first $N_1$ rows of a vector or matrix are found on processor 1, the next $N_2$ rows are on processor 2 and so on. For scalar-vector and vector-vector operations, this is sufficient to determine a parallel layout. Figure 3.2 illustrates a scalar-vector product in parallel using this distribution. Immediately, each processor may begin work on its own piece of the vector, with processor 1 multiplying $\alpha$ by each of its elements of $x$ ($x_1$ and $x_2$) and storing the result in its elements of $r$ ($r_1$ and $r_2$), processor 2 operating on its parts of the vectors, and so on.

\[
\begin{array}{ccc}
\text{Processor 1} & \alpha \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} & \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} \\
\text{Processor 2} & \alpha \cdot \begin{bmatrix} x_3 \\ x_4 \end{bmatrix} & \begin{bmatrix} r_3 \\ r_4 \end{bmatrix} \\
\text{Processor 3} & \alpha \cdot \begin{bmatrix} x_5 \\ x_6 \end{bmatrix} & \begin{bmatrix} r_5 \\ r_6 \end{bmatrix}
\end{array}
\]

Figure 3.2: A parallel scalar-vector product. Each processor holds two rows of the vectors and has the scalar. As a result each processor can perform this operation independently of the others.
Assuming that the result is acceptable in a parallel format, this operation is completely parallel.

A vector-vector product similarly is accomplished in a straightforward manner in parallel as illustrated in Figure 3.3. Again, each processor starts operating on its block in parallel, with each processor generating its local dot product and communicating the result. While there is communication required, by performing this step asynchronously, work and communication are allowed to overlap and the work is accomplished in parallel. This is considered a highly parallel operation especially for large global vector sizes.

Consider a parallel matrix-vector product, as illustrated in Figure 3.4. Again, the matrix is distributed with a number of its rows being assigned to each processor. Now each processor may begin work in parallel, with processor-1 multiplying block $u_{11} - u_{22}$ by its components of the vector $x$ and placing the result in the local entries of vector $r$. However, notice that for the blocks which are off the diagonal this cannot be accomplished in parallel. For example, in order for the solution from processor-3 to be complete, it must obtain matrix values from processor-1 as defined by block $u_{51} - u_{62}$. This shows that matrix-vector products are not inherently parallel. In order for these products to be accomplished completely in parallel, the off-diagonal blocks must be zero! The method by which entries in the off-diagonal blocks are minimized is related to the decomposition of the physical domain that the problem is defined on, which is the topic of Section 3.3.3.

Examining data distribution provides insight for the reason why some of the operations encountered in direct methods do not parallelize efficiently. After generating a factorization, the final step of an direct solution method is backward elimination, where the solution vector is obtained sequentially as discussed in Section 2.2.1. This operation is illustrated in Figure 3.5. Here, processor-3 must begin by solving for $x_6$. This is then followed on that same processor by solution for $x_5$. No other processors can perform any
Figure 3.3: A parallel dot product. Each processor holds two rows of the vectors. Each processor begins by generating the local dot product from the vector. This is followed by communication between the processors as the result is updated for the off-processor values. This is an example where Gustafson's law clearly applies. The interprocessor communication does not scale with problem size but with the number of processors. As a result each processor can perform this operation independently of the others.
| Processor 1 | \[ u_{11} \quad u_{12} \quad u_{13} \quad u_{14} \quad u_{15} \quad u_{16} \] | \[ x_1 \] | \[ r_1 \] |
| Processor 2 | \[ u_{21} \quad u_{22} \quad u_{23} \quad u_{24} \quad u_{25} \quad u_{26} \] | \[ x_2 \] | \[ r_2 \] |
| Processor 3 | \[ u_{31} \quad u_{32} \quad u_{33} \quad u_{34} \quad u_{35} \quad u_{36} \] | \[ x_3 \] | \[ r_3 \] |
| | \[ u_{41} \quad u_{42} \quad u_{43} \quad u_{44} \quad u_{45} \quad u_{46} \] | \[ x_4 \] | \[ r_4 \] |
| | \[ u_{51} \quad u_{52} \quad u_{53} \quad u_{54} \quad u_{55} \quad u_{56} \] | \[ x_5 \] | \[ r_5 \] |
| | \[ u_{61} \quad u_{62} \quad u_{63} \quad u_{64} \quad u_{65} \quad u_{66} \] | \[ x_6 \] | \[ r_6 \] |

Figure 3.4: A parallel matrix vector product. Each processor may begin working in parallel by multiplying its diagonal block by the local members of its right-hand-sides and storing that part of the solution in the right hand side. The off-diagonal blocks of the matrix, however, require communication either to obtain components of \( x \) from to assign values to components of \( r \).
operations until processor 3 completes! This causes poor parallel efficiencies for backwards elimination, as can be seen in the parallel efficiencies associated with this step for the state-of-the-art direct solution method discussed in Section 1.5.1, Concurrent Factorization and Storage (CFS) [126]. The time required for backward elimination of the CFS algorithm is reported Table 3.6. Notice that as the number of processors is increased, the time required for this step actually increases. Yet all parallel direct methods would still require such a step.

The problem of handling matrices and vectors in parallel is simplified using The Portable, Extensible Toolkit for Scientific Computation, PETSc [11,12,13]. This is an object oriented toolkit that defines vectors and matrices as objects with properties such as length, number of local rows, and so on. Because it is based on MPI and is object-oriented, it is possible to use these structures in customized code and to customize the PETSc code to accommodate sophisticated solvers and techniques. In fact, in the course of work on this

<table>
<thead>
<tr>
<th>Processor 1</th>
<th>[ u_{11} \quad u_{12} \quad u_{13} \quad u_{14} \quad u_{15} \quad u_{16} ]</th>
<th>[ x_1 ]</th>
<th>[ r_1 ]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 \quad u_{22} \quad u_{23} \quad u_{24} \quad u_{25} \quad u_{26}</td>
<td>x_2 \quad x_3 \quad \bullet \quad x_4</td>
<td>r_2 \quad r_3 \quad r_4</td>
</tr>
<tr>
<td>Processor 2</td>
<td>0 \quad 0 \quad u_{33} \quad u_{34} \quad u_{35} \quad u_{36}</td>
<td>\bullet \quad x_4</td>
<td>r_3 \quad r_4</td>
</tr>
<tr>
<td></td>
<td>0 \quad 0 \quad 0 \quad u_{44} \quad u_{45} \quad u_{46}</td>
<td>\bullet \quad x_4</td>
<td>r_5 \quad r_6</td>
</tr>
<tr>
<td>Processor 3</td>
<td>0 \quad 0 \quad 0 \quad 0 \quad u_{55} \quad u_{56}</td>
<td>\bullet \quad x_5</td>
<td>r_5 \quad r_6</td>
</tr>
<tr>
<td></td>
<td>0 \quad 0 \quad 0 \quad 0 \quad 0 \quad u_{66}</td>
<td>\bullet \quad x_6</td>
<td>r_5 \quad r_6</td>
</tr>
</tbody>
</table>

Figure 3.5: Parallel backwards elimination. This cannot be done with good efficiency as both processors 1 and 2 cannot proceed until processor 3 solves for its local components of \( \mathbf{x} \). Then, only processor 2 can proceed. This dependency is one reason that parallel direct methods are not highly efficient.
Figure 3.6: Time ($T$) required for backwards elimination of CFS as a function of number of unknowns ($N$) for various numbers of processors. [126]
thesis, the author contributed a number of things to this project, including an interface to a pivoting ILU-dt routine.

The package is predicated on an object oriented programming style, which is defined by the principles of data encapsulation, polymorphism, and inheritance [13,109]. Data encapsulation or data hiding refers to writing objects or data structures such that the application code does not directly access the underlying data in the object. The application code effects the data in the objects by making subroutine calls to change the data. Polymorphism refers to techniques that allow one to call the same function from the application level of the code to perform a specific operation regardless of the underlying data structure used to store the data. Finally, inheritance is the process of defining new objects by either adding properties to objects which already exist or combining properties of several types of objects. By basing PETSc on object orientation, the authors sought to manage the complexity of parallel numerical software through an object-oriented programming style, hiding details of message passing when performing basic operations on these objects without concealing the parallelism [13]. Some of the objects included in PETSc are depicted in figure 3.7 relative to the level of abstraction provided. Because of the object orientation used within PETSc, adding functionality is straightforward: In order to add a drop tolerance ILU preconditioner, such as described in Section 2.2.3, a module was added to the preconditioner object which built on the objects below it [11].

Parallel vectors and matrices are stored in modified versions of their serial counterparts [11,13]. Vectors are stored row-wise, and each vector has associated with it a local number of rows. Matrices are stored in the compressed sparse row format, where only nonzero values are stored. This storage method is illustrated in Figure 3.8. With this method, the matrix is stored in three vectors, VAL, which is double-precision, COL and
Figure 3.7: Organization of the PETSc library
$A = \begin{bmatrix}
1 & 0 & 0 & 2 & 0 \\
3 & 4 & 0 & 0 & 5 \\
6 & 0 & 7 & 8 & 9 \\
0 & 0 & 10 & 11 & 0 \\
0 & 0 & 0 & 0 & 12
\end{bmatrix}$

Is stored as . . .

$\text{VAL} = [1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 10 \ 11 \ 12]$
$\text{COL} = [1 \ 4 \ 1 \ 2 \ 5 \ 1 \ 3 \ 4 \ 5 \ 3 \ 4 \ 5]$
$\text{ROW} = [1 \ 3 \ 6 \ 10 \ 12 \ 13]$

Figure 3.8: Storage of a 6x6 matrix in compressed sparse row format (CSR).
ROW, both of which are integer. VAL stores the non-zero values of the matrix, traversed in row-wise fashion. COL, then, stores the column indices of the elements in the VAL vector. That is, if $VAL(k) = a_{i,j}$ then $COL(k) = j$. ROW, then, stores the locations in the VAL vector that start a row. If $VAL(k) = a_{i,j}$ then $ROW(i) \leq k < ROW(i + 1)$. The last entry of ROW is defined to be equal to the total number of nonzeros plus one.

For parallel matrices, two modifications are made to this scheme. First of all, each processor only holds information about the contiguous rows of the matrix which have been assigned to it. Secondly, because the off-diagonal blocks should be minimized in our decomposition for good parallel efficiency, two CSR matrices are actually created on each processor. One for the on-diagonal block which defines inter-processor coupling and another for the off-diagonal ones that define required communication. An example of this decomposition is shown for the first processor in Figure 3.9. Minimizing the number of entries in the off-diagonal blocks requires a domain decomposition of the problem geome-

\[
\begin{pmatrix}
6 & 2 & 0 \\
0 & 8 & 35 \\
13 & 0 & 4 \\
\end{pmatrix}
\begin{pmatrix}
4 & 0 & 12 & 0 & 1 \\
7 & 0 & 0 & 15 & 0 \\
16 & 17 & 18 & 0 & 0 \\
\end{pmatrix}
\begin{pmatrix}
11 & 0 & 14 \\
0 & 8 & 0 \\
0 & 0 & 0 \\
\end{pmatrix}
\begin{pmatrix}
2 & 4 & 5 & 0 & 0 \\
5 & 3 & 5 & 0 & 0 \\
22 & 3 & 4 & 24 & 0 \\
\end{pmatrix}
\begin{pmatrix}
55 & 43 & 33 \\
0 & 0 & 0 \\
39 & 0 & 0 \\
\end{pmatrix}
\begin{pmatrix}
0 & 0 & 2 & 0 & 0 \\
0 & 0 & 11 & 0 & 0 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
6 & 2 & 0 \\
0 & 8 & 35 \\
13 & 0 & 4 \\
\end{pmatrix}
\begin{pmatrix}
4 & 0 & 12 \\
7 & 0 & 0 \\
16 & 17 & 18 \\
\end{pmatrix}
\]

Figure 3.9: Each processor represents its portion of a parallel matrix as two submatrices. One containing values along the diagonal, the other containing off-diagonal blocks.
try, and manner in which this is accomplished is the subject of the following Section for the model problem of natural convection in a fluid filled cavity.
3.3.3 Domain Decomposition for Parallel Computing

Before discussing how the number of entries in the off-diagonal blocks seen in the matrix structures presented in Section 3.3.2 is minimized using domain decomposition, it is necessary to consider the manner in which entries in the linear system matrix arise. Each row in the matrix corresponds to an equation in the discrete problem. This fact suggests that because finite elements have local support, meaning that elements rely on only adjacent elements for contributions to their equations, a domain decomposition based on the problem geometry should be effective [112,115]. This is explored in this Section.

Consider the origin of the linear system arising in the steady state solution of the natural convection in a cavity problem defined in detail in Section 2.5. Here, the equations of motion are discretized over a physical geometry using finite elements. The unknowns at a particular node interact with other unknowns, either at that particular node or nearby, and these interactions determine that row's nonzero pattern.

As seen in Section 2.5, this flow, illustrated in Figure 3.10, is governed by the Boussinesq equations of mass, momentum and energy transfer:

\[ \nabla \cdot \mathbf{v} = 0 \]  \hspace{1cm} (3.4)

\[ \sqrt{\text{Gr}} \ \mathbf{v} \cdot \nabla \mathbf{v} - \nabla^2 \mathbf{v} = \sqrt{\text{Gr}} \ \mathbf{e}_3 T - \nabla P \]  \hspace{1cm} (3.5)

\[ \sqrt{\text{Gr}} \mathbf{v} \cdot \nabla T - \text{Pr}^{-1} \nabla^2 T = 0 \]  \hspace{1cm} (3.6)

The sparsity structure of the linear system that results from the discretization of these equation is shown in the “natural ordering” in Figure 3.11. In this form, the equations of mass, momentum, and energy are most easily “seen” in the matrix structure. However, if we consider the row-wise assignment of this system to multiple processors, one can see that the assignment in this order causes much off-processor communication, and in fact
Figure 3.10: Schematic for convective flow of a variable density fluid in a closed cavity.

contains a small block which corresponds to of the local right-hand-sides and unknown-vector entries residing entirely off-processor from the matrix entries!

One might also be tempted to consider using band-width minimization to determine the matrix ordering, knowing that this keeps all entries in a band along the diagonal of the matrix. However, this, too causes large blocks to be in the off-diagonal blocks as is illustrated in Figure 3.12.

In order to minimize the number of entries in the off-diagonal blocks, we have to consider what causes entries to lie in that region. Most simply, these entries correspond to interactions between a particular processor's equations and unknowns that lie on other processors. Because finite element discretization has local support, generating a decomposition of the original geometry into a number of pieces that have the minimum possible perimeter interface with other domains will minimize the number of entries in the off-diagonal blocks -- therefore minimizing the communication required for matrix-vector
Figure 3.11: Sparsity structure of the Jacobian linear system resulting from the natural ordering and distribution onto 3 processors. Results are for the natural convection problem described in Section 2.5. Off-diagonal blocks are not minimized.
Figure 3.12: Sparsity structure resulting from banded ordering. While this image is exaggerated, bandwidth minimization does not minimize the number of entries in the off-diagonal blocks.
operations. Furthermore, so that the workload is balanced among processors, it is desirable that the areas assigned to each processor is equal. The goals are to:

1. Minimize the number of cut edges between nodes

2. Balance the number of nodes per processor

Consider a mesh with 4x4 nodes and two processors on which we wish to divide the problem. Several possible partitioning schemes are presented in Figure 3.13. Scheme A is unacceptable because the workload is unbalanced between the processors. Scheme B is unacceptable because the communication is not minimized between processors. What is desired is a consistent method for obtaining a decomposition such as C for general geometries, where both communication is minimized and balanced sets are generated.

Expressing these goals mathematically, we desire is a way to minimize the function:

\[ f(x) = \frac{1}{4} \sum_{E_{i,j}} (x_i - x_j)^2 \]  \hspace{1cm} (3.7)

where \( E_{i,j} \) indicates an edge between node \( i \) and \( j \) exists and \( x_i \in \{-1, 1\} \). Minimizing
the function \( f(x) \), therefore, minimizes the number of edges cut between the two generated sets, -1 and 1. Balanced sets are desired, so the constraint \( x^T 1 = 0 \) is added. This problem has been shown to be NP-complete, requiring more than exponential time to solve [74].

As a result, numerous heuristics have been proposed to approximate the best solution to this problem. Coordinate bisection and inertial partitioning methods were the earliest methods used, dividing a domain correspondingly along the longest axis of expansion or by using a line through the geometric center of the domain that minimizes the moment of inertia of the nodes [115]. Neither of these methods is very effective, however, for general, complicated geometry, rarely resulting in a minimized number of cut edges between the two generated sets [115]. Another approach is the geometric method of random circles due to Miller, et al. [132]. Instead of making planar cuts as in the coordinate bisection, it makes spherical cuts (or circular in 2 dimensions). These cuts are made by projecting the vertices onto the surface of a sphere in one higher dimension and then making a planar cut on that sphere. The result is that these cuts are transformed into spherical cuts when projected back into the lower dimension. The random circles technique can generate much better cuts than coordinate bisection, but still requires coordinate information [132].

The method of spectral bisection is used in this thesis and is based on an eigenvector problem which is generated from Eq. 3.7 [90,91,115]. The Laplacian of a graph, \( L \), is defined as a matrix whose entries are -1 if an edge exists between two nodes \( i \) and \( j \), equal to the degree of the node if \( i = j \), and zero otherwise. Our problem, described in (3.7) may be rewritten as a minimization problem

\[
\text{Minimize} \left( \frac{1}{4} x^T L x \right)
\]

subject to
\[ x^T 1 = 0, x_i = \pm 1 \]  

(3.9)

However, this problem is NP-complete [74], so the problem is restated by seeking to minimize

\[
\text{Minimize} \left( \frac{1}{4} x^T L x \right)
\]

(3.10)

subject to

\[ x^T 1 = 0, x^T x = n \]

(3.11)

where \( n \) is the number of nodes in the mesh. The decomposition is generated from the solution vector by sorting and splitting by amplitude. This eigenvector problem is the same problem which would arise from treating our nodal mesh as a network of masses located at the nodes and connected by springs at the edges [90,91]. The solution, therefore corresponds to the second vibrating mode of the mesh. This is shown for a one-dimensional mesh in Figure 3.14, and gives an intuitive sense of how this method should generate balanced, minimally connected geometries. The resulting partitionings have been shown empirically to be of better quality than alternative decomposition methods, both in terms of minimizing the number of cut edges, generating balanced sets, and generating the fastest parallel code using benchmark parallel algorithms [91, 115].

Applying this method recursively to the discretization of the natural convection problem results in the decomposition for four processors shown in Figure 3.15 along with the associated sparsity structure. The sparsity structure has few entries in the off-diagonal blocks. Nested spectral bisection is used for the solver put forth in this thesis, using the CHACO library [90].
First vibrational mode

Second vibrational mode

Figure 3.14: First and second vibrational modes of a spring. The second vibrational mode corresponds to the decomposition obtained from spectral dissection. The positive regions would be assigned to one processor, the negative ones to the second.
Figure 3.15: Decomposition (a) and subsequent Jacobian sparsity structure (b) resulting from recursive bisection of a cavity onto four processors using spectral nested dissection.
3.3.4 Parallel Performance Metrics

The same general goals of exploiting sparsity and minimizing computations apply to both serial and parallel computations. Other objectives also arise, as alluded to in Section 3.3.3, such as minimizing communication and balancing the load between processors. It is desirable to determine a metric that compares the performance on multiple processors in parallel to the performance of the algorithm on fewer, usually one, processor. This metric, referred to as an efficiency, gives the basis for determining the efficiency of a parallel method.

Often, the computational speed of a processor is measured in millions of floating point operations per second or MFLOP/s as discussed in Section 3.2.1, so one might naively assume that it is desirable to maximize the speed of that an algorithm performs on a single processor. Furthermore, one might believe that it is desirable that, when a program is parallelized, that it's computational speed in MFLOPS should scale linearly, ideally. Efficiency defined this way would be expressed as

\[ \varepsilon = \frac{R_s}{PR_1} \]  \hspace{1cm} (3.12)

This metric, however, does not imply good method performance as it falsely implies good performance when a method's complexity increases with the addition of processor. For example, the parallel direct method, CFS, is described often in terms of it's efficiency based on this metric [126,127]. However, efficiencies of over 90% based on this metric of parallel efficiency translate to efficiencies of less than 60% when based on speedup [126].

The end user of a computer simulation probably cares less about how hard a computer has had to work to obtain a solution than with how long it takes to complete the simulation. With this in mind we return to Amdahl's law for a definition of the efficiency [4]
\[ S = \frac{t_1}{t_p} = \frac{1}{f + s} \]  \hspace{1cm} (3.13)
\[ \varepsilon = \frac{1}{f + s \times P} = \frac{S}{P} \]  \hspace{1cm} (3.14)

where \( \varepsilon \) measures the efficiency of the method based on how much faster a particular size of problem may be solved when multiple processors are used. As \( P \to \infty \), this metric is necessarily bounded as computation becomes dominated by communication and serial processing. The increased complexity of the CFS algorithm has driven the speedup down while the operation count remained flat.

Amdahl's law was used early to argue against parallel programming's promise until Gustafson [87] realized that, as problem size is increased, \( S \) typically decreases. For this reason, he proposed to allow the problem size increase with increasing numbers of processors, using the rationale that a parallel algorithm typically is used to solve larger problems than are tractable on a single processor. A Gustafson efficiency is then calculated as:

\[ \varepsilon_G = \frac{t_1}{t_N} \]  \hspace{1cm} (3.15)

where the number of unknowns per processor is held constant.

Speedups based on computation rate are misleading because the cover for algorithms which grow increasingly complex as the number of processors increases. Furthermore, the amount of peak on a processor that is obtainable on a particular computer is highly variable. Instead, for a good parallel algorithm, several conditions must be satisfied:

1. The solution of larger problems should be feasible with the increasing number of processors

2. Communication should be kept to a minimum
3. Speedup-based efficiency should be high. It should be possible to solve current problems faster using multiple processors

4. Gustafson-style scaled efficiency should also be high. This means that larger problems than were previously tractable will be solvable without method degradation

5. Absolute solution time should be low, relative to the state-of-the-art

6. The numerical method should be portable

All of these factors need to be considered to evaluate the utility of a parallel solution method. These are considered for the method developed in the next Chapter.
Chapter 4

A Block Complement and Additive Levels Method (BCALM) Preconditioner for Iterative, Parallel Solution of Finite Element Discretization of Viscous Partial Differential Equations

4.1 Introduction: Iterative Methods for Viscous Flow and the Natural Convection Problem

Problems in viscous fluid flow involving heat and mass transfer are of interest for modelling both materials processing and manufacturing processes, and are typically composed of a set of strongly coupled differential equations. For steady state solutions, these equations are discretized using either finite difference, finite element, or spectral methods which leads to large sets of nonlinear algebraic equations. Newton's method, discussed in Section 2.8, or some other type of fixed-point iteration may then be used to solve this nonlinear equation set, usually necessitating the solution of a large linear algebraic problem at each iteration. Solving this linear system often consumes the largest amount of computer time, and thus developing techniques for rapid solution of these systems remains of great interest to researchers.

Direct solution methods for solution of linear algebraic equations are attractive because of their robustness and general applicability, as seen in Section 2.2.1 [7,76,78]. Indeed, LU-decomposition techniques such as the frontal decomposition method MA42 remain popular for the serial solution of banded linear systems [61]. However, these methods have memory requirements that scale with the number of unknowns, $N$, and the bandwidth, $B_w$, of the linear system as $O(NB_w^2)$, necessitating out-of-core problem solu-

1. A paper based on the material in this chapter has been submitted for publication in Chemical Engineering Science
tion for two dimensional problems and potentially intractable problem sizes in three spatial dimensions due to the increasing bandwidth [79,80,122]. Furthermore, direct methods are not inherently parallel, leading to multi-frontal schemes that degrade as factorization proceeds from the sub-domains that are internal to a particular processor, to supernodes spanning multiple processors [126]. Parallel direct methods also are not scalable for linear problems that do not require repeated factorization, because the backward substitution and forward elimination operations do not exhibit high parallel efficiencies and are the only operations necessary once factorization has occurred [59,60,122,127].

The purpose of this paper is to present a robust algorithm for iterative solution the linear systems that arise from the discretization of viscous flow problems. Specifically, we consider the solution of sets of linear equations

\[ Jx = b \]  \hspace{1cm} (4.1)

where the matrix \( J \in \mathbb{R}^{N \times N} \) and \( x, b \in \mathbb{R}^{N} \). The linear system \( J \) is presumed to result from the discretization of some incompressible flow problem composed of a set of viscous equations

\[ Cu \cdot \nabla u - \nabla^2 u = -\nabla p + g(u) \]  \hspace{1cm} (4.2)

and a continuity equation

\[ \nabla \cdot u = 0 \]  \hspace{1cm} (4.3)

where \( u \) is the velocity field and \( p \) is the pressure. The matrix \( J \) resulting from Newton’s method is a sparse, asymmetric, and indefinite system due to the nonlinear convective terms. For the examples in the Chapter, the steady state solution of natural convection in a fluid-filled cavity using Newton’s method is used as a model problem. This problem is described in detail in Section 2.5, and provides a good model for the Stokes-like systems.
which arise in viscoelastic flow, as discussed in Section 4. Additionally, this system has been widely studied and allows comparison of the developed method with other solution techniques [126,152].

In this Chapter, the Krylov iterative solver selected is Bi-CGSTAB [185], a fast and smoothly convergent variant of the bi-conjugate gradient method that has been shown useful for a variety of ill-conditioned problems [148]. This method is described in detail in Section 2.2.2, and may be found in Figure 2.14. If applied in unpreconditioned form to the Jacobian linear system which arises for this model problem, thousands of iterations are required as seen in Table 4.1 which is discussed in Section 4.3 [34]. For this reason, a robust preconditioner is needed, as discussed in Section 2.2.3.

The remainder of this Chapter is organized as follows. Section 4.2 describes the solution method, for the natural convection problem presented in Section 2.5 and solved at steady state using Newton's method, presented in Section 2.8. Section 4.2.1 then, describes the block preconditioner which is derived using a Schur complement approach. A discussion of data organization for parallelism in Section 4.2.2 highlights material presented in Section 3.3 in the context of the model problem. Preconditioning the viscous operator using additive Schwarz is discussed in Section 4.2.3. Finally, these pieces are assembled in Section 4.2.4, where the implementation is described. Results are presented in section 4.3, with a discussion of implementation details and a demonstration of the method's robustness and parallel efficiency. Section 4.4 presents some conclusions from this work.
4.2 A Robust Preconditioner: The Block Complement and Additive Levels Method (BCALM) Preconditioner

The iterative solution of the linear systems, Eq. (4.1), that arise from the discretization of viscous flow problems is only practical if an effective preconditioner for the system is found. The preconditioner for a linear system is usually generated either by approximately representing the linear system in some easily invertible fashion, \( \tilde{J}u = v \), or by approximating the solution of the linear system, \( Ju = v \). Our technique uses a combination of these methods. First, the problem physics determine a block approximation to the original linear system using the Schur complement of the pressures, as discussed in Section 4.2.1. The resulting block system is utilized as a preconditioner, using both approximations to these resulting blocks and approximate solution methods to define a parallelizable preconditioner.

4.2.1 A Schur Complement Block Preconditioner

Because the system of equations resulting from the solution of the natural convection problem using Newton’s method, as described in Section 2.8, is indefinite and asymmetric, using a Krylov iterative solver without a preconditioner is computationally expensive [161]. An effective preconditioner for this linear system, is best discovered by closely examining the equations. Consider ordering the equations such that the vector of unknowns is expressed as

\[
x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}
\]  

(4.4)

where, the pressure unknowns are in the partitioned vector \( x_2 \), and all other unknowns are in the partitioned vector \( x_1 \). The resulting Jacobian linear system, Eq. 4.1, is written in this block structure as
\[
\begin{bmatrix}
A & B \\
C & D
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix}
= 
\begin{bmatrix}
b_1 \\
b_2
\end{bmatrix}
\] (4.5)

where, the blocks \( A \) and \( B \) correspond to the discretization of the equations of energy and momentum, (2.87)-(2.88), while blocks \( C \) and \( D \) correspond to the discretization of the continuity equation, (2.89). The block \( D \) is identically \( 0 \) with one entry set as \( d_{11} = 1 \) so that a globally well-defined pressure field is obtained.

The path to a preconditioner is defined by considering the Schur complement of Eq. (4.5). Schur complement algorithms are a general family of algebraic techniques from domain decomposition. These algorithms, developed by Przemieniecki in [149] for structural analysis, were developed as methods for decoupling the boundary nodes of subdomains from internal nodes. This was accomplished in [149] by rewriting their linear system of interest as in Eq. (4.5), with \( x_1 \) corresponding to unknowns internal to each subdomain and \( x_2 \) corresponding to unknowns on the boundaries. The Schur complement, then, is a matrix which decouples the partitioned vector \( x_2 \) from partitioned vector \( x_1 \). Applied recursively, this yields a crude method for parallelism [15].

In our case, the Schur complement of the pressures instead decouples the pressures from the rest of the unknowns. This is formed by taking the first row times \(-CA^{-1}\) and adding it to the second. The resulting Schur complement linear system has the form:

\[
\begin{bmatrix}
A & B \\
0 & S
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix}
= 
\begin{bmatrix}
b_1 \\
b_2 - CA^{-1}b_1
\end{bmatrix}
\] (4.6)

The Schur complement \( S \) of the pressure variables \( x_2 \) is defined as:

\( S \equiv D - CA^{-1}B \) (4.7)
Solution for this system of equations can be constructed by solving first for \( x_2 \), and subsequently for \( x_1 \) [123]. This motivates splitting the preconditioner into two separate operations. One preconditioner is generated for the Schur complement linear system and applied to the pressures, \( x_2 \). Another preconditioning operation is used for the viscous operator, \( A \), and applied to the remainder of the unknowns.

The Schur complement solution technique has been used for Stokes flow problems, however the inversion of the \( A \) operator required to compute \( S \) is time consuming and causes \( S \) to be very dense [39]. Research on preconditioning this Schur complement linear system has generated effective preconditioning schemes. Both Cahouet and Chabard [33] and Ng, et al. [140] performed numerical experiments with the generalized Stokes’ problem and found that a preconditioner for \( S \) which avoids the inversion of \( A \) and the subsequent dense matrix structure may be formed by taking

\[
\tilde{S} \equiv D - CB.
\]  

(4.8)

Work by Mathew [123] proposed an effective preconditioner of the form

\[
\tilde{S} \equiv \text{diagonal}(S) = \text{diagonal}(D - CA^{-1}B)
\]  

(4.9)

due to the rapid die-off of entries near the diagonal when \( A \) is an elliptic operator. Furthermore, in some cases, the diagonal of \( S \) is easily obtained [39]. This is not generally the case, so we combine these two approximation into the easily obtained

\[
\tilde{S} \equiv \text{diagonal}(D - CB)
\]  

(4.10)

The resulting preconditioner for the linear system, Eq. (4.5), is
\[
\begin{bmatrix}
A & B \\
0 & \tilde{S}
\end{bmatrix}
\begin{bmatrix}
u_1 \\
u_2
\end{bmatrix}
= 
\begin{bmatrix}
v_1 \\
v_2
\end{bmatrix}
\] (4.11)

where

\[
\tilde{S} = \text{diagonal}(D - CB)
\] (4.12)

While this solution method has been implemented in serial to obtain a solution using, for example, a direct method for inversion of \( A \), we desire a method that is readily parallelizable. This is accomplished using domain decomposition on the operator \( A \), as described in the following section.

4.2.2 Data Distribution for Parallelism

Examining the Bi-CGStab algorithm described in Section 2.2.2, we see that, in order for the overall method to be scalable, both inner products and matrix-vector multiplications must be scalable. This is accomplished by distributing the vectors and matrices to the processors row-wise, such that each processor has the same range of rows from each matrix and vector, as described in Section 3.3.2. Local inner products may be computed on each processor before summing the results across all the processors. Computing matrix-vector operations in parallel is slightly more complicated. The distribution, for example, of a matrix-vector product to 3 processors can be written in block form as

\[
\begin{bmatrix}
M_{11} & M_{12} & M_{13} \\
M_{21} & M_{22} & M_{23} \\
M_{31} & M_{32} & M_{33}
\end{bmatrix}
\begin{bmatrix}
s_1 \\
s_2 \\
s_3
\end{bmatrix}
= 
\begin{bmatrix}
t_1 \\
t_2 \\
t_3
\end{bmatrix}
\] (4.13)

The components \( \{M_{ij}\}, \{s_i\}, \{t_i\} \) all are data stored on the \( i \)th processor.

In this scheme, the blocks \( \{M_{ii}\} \) represent couplings between the local matrix entries and local vectors components on processor \( i \), while the off-diagonal blocks, \( \{M_{ij}\}_{i \neq i} \) represent off-processor dependencies. In order for matrix vector products to be scalable, the
rows must be distributed to the processors in such a manner that the number of entries in
the off-diagonal blocks, \( \{ M_{ij}\}_{i \neq j} \), is minimized. This is accomplished by assigning all
unknowns at a node to a particular processor using spectral dissection, as described in Sec-
tion 3.3.3.

Because all of the unknowns at a particular point are distributed to a single processor
based on the resulting mapping, the basic operations involved in Bi-CGStab are scalable,
as shown in detail in Sections 3.3.2 and 3.3.3. Furthermore, the operation of inverting the
block \( \tilde{S} \) in the preconditioner described in Section 4.2.1 is also scalable, as each processor
needs only to invert the locally owned diagonal entries. However, in order to complete this
method, the approximate inversion of \( A \) needs to be carried out in a scalable manner. This
is described in the following section.

4.2.3 Preconditioning the Viscous Operator

The operator \( A \) is a set of discretized elliptic equations corresponding to the viscous
part of the overall operator

\[
J = \begin{bmatrix} A & B \\ C & D \end{bmatrix}
\]  

(4.14)

where the operator \( A + B \) corresponds to the discretized Eqs. (2.87) and (2.87). As was
discussed in detail in Section 2.2.3, one method of approximately inverting an elliptic
operator which lends itself to ready parallelization is the additive Schwarz method. Shown
to be effective for a range of elliptic operators [145,154], this parallelizable scheme may
be understood as utilizing the domain decomposition, generated using spectral dissection
to parallelize the basic iterative method operations, to define a splitting of the overall
domain, \( \Omega \), into a number of sub-domains such that
\[ \Omega = \sum_{i=1}^{P} \Omega_i \]  

(4.15)

where \( P \) corresponds to the number of processors to be utilized. Each processor solves the problem restricted to its sub-domain and contributes its result to the overall solution. As presented in detail in Section 2.2.3, the approximate inverse to \( A \) is obtained by approximately inverting

\[ A = \sum_{i=1}^{P} R_i A R_i^T \]  

(4.16)

where \( R_i \) is a restriction operator that zeros matrix elements that do not correspond to unknowns in domain \( \Omega_i \), i.e. assigned to the \( i \)th processor. Similarly, \( \{ R_i^T \} \) is a prolongation operator that extends the result by zeros to be well-defined over the entire domain, \( \Omega \). The matrix to be inverted on each processor is

\[ A_{\Omega_i} \approx R_i A R_i^T . \]  

(4.17)

The resulting approximate inversion of \( A \) is then

\[ A^{-1} = \sum_{i=1}^{P} R_i^T A_{\Omega_i}^{-1} R_i . \]  

(4.18)

As explained in Section 2.2.3, additive Schwarz, alone, is effective in capturing detail in the solution field, as each domain utilizes the boundary information for the particular subdomain [154]. However, the solution at any point in the domain for an elliptic operator is influenced by all boundary values [166]. For this reason, additive Schwarz is utilized with one level of overlap and a Galerkin coarse grid, as described in Section 2.2.3.
As discussed in section 2.2.3, the choice of coarse space for the coarse grid operator can be made in a number of ways. For this problem’s regular geometry, we choose to generate a Galerkin coarse grid and then both formulate the equations over and impose the boundary conditions on this coarse grid. The restriction operator for the coarse grid, by which values are moved from the fine grid to the coarse grid, is achieved by simple insertion. The corresponding prolongation is achieved by bilinear interpolation, avoiding costly computation during implementation of this preconditioner. For additional details on multilevel additive Schwarz, see Section 2.2.3 and the references therein.

The solution of the coarse grid problem may be considered in parallel. However, due to the small dimension of this problem, serial inversion is used. As will be seen in Section 4.3, this does not effect the method’s overall efficiency.

4.2.4 A Block Complement and Additive Levels Method (BCALM) Preconditioner for Viscous Flow Equations

The preconditioner for the Jacobian linear system, \( \tilde{J}u = \nu \), is accomplished using two, separate approximate inversions of the operators \( S \) and \( A \). This is implemented in four steps:

1. \( 1 - \nu = \begin{bmatrix} \nu_1 \\ \nu_2 \end{bmatrix} \)

2. Solve \( \tilde{S}u_2 = \nu_2 \) where \( \tilde{S} = \text{diagonal}(D - CB) \), as presented in Section 4.2.1.

3. Solve \( Ar_1 = s_1 - Br_2 \) using additive Schwarz

4. \( r = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix} \)
The inversion of two types of linear systems is required in accomplishing step 2. The linear system corresponding to the Galerkin coarse grid must be solved, \( A_{\Omega} = R_{0}A R_{0}^{T} \), and the Jacobian linear system obtained on each processor, \( A_{\Omega} = R A R_{i}^{T} \), corresponding to each local subdomain, must be inverted as described in Section 2.2.3. While it is tempting to use full LU decomposition in both of these steps, generating the exact inverse to these matrices does not yield a preconditioner which is exact. Instead, an incomplete LU decomposition, based on the idea of dropping fill terms if they are smaller than a given drop-tolerance [188][189], is utilized as described in detail in Section 2.2.3. A good approximation to the inverse is obtained, while reducing operations and memory required.
4.3 Demonstrating the Preconditioner's Effectiveness and Efficiency

The implementation of this preconditioned BiCGStab iterative solver is simplified by using routines and structures from a suite of tools for scalable scientific computing, the Portable Extensible Toolkit for Scientific Computation (PETSc) [12], as described in Section 3.3. These libraries are layered on top of Message Passing Interface (MPI) for message passing, and additional message passing was also accomplished using MPI, as detailed in Section 3.3.1. No machine-specific calls were made, so the code is highly portable. Performance testing was done on an eight-node Sun Ultra HPC 5000 with 2 Gbytes of memory.

The steady state natural convection problem, described in Section 2.5, was solved using Newton's method, described in Section 2.8. For all runs, Newton's method was considered converged when \( \| x_k - x_{k+1} \| < 10^{-5} \) where \( x_{k-1} \) and \( x_k \) are two successive approximate solutions generated for the system. The internal Bi-CGStab iterations were terminated when either a relative tolerance of \( 1 \times 10^{-5} \) or an absolute tolerance of \( 1 \times 10^{-8} \) was achieved, and a drop tolerance of \( 1 \times 10^{-5} \) was used. From a zero initial guess, these conditions resulted in Newton's method converging in between 2 and 5 iterations, depending on the Grashof number, the number of unknowns, and the domain decomposition.

4.3.1 Method Robustness

In order to gauge the robustness of this method, we consider its sensitivity to factors that could adversely affect its performance -- the magnitude of the \( \text{Gr} \), the number of unknowns in the discretization and the number of processors used in the calculation. The sensitivity of the method to the number of processors is considered in Section 4.3.3. Sensitivity to both the Grashof number and number of unknowns is considered here.
As the Grashof number increases from $Gr=1$ to $Gr=10,000$, the Jacobian linear system goes from being symmetric to being highly asymmetric. This results in the linear system being harder to converge in terms of the Newton iteration requiring two iterations for $Gr=1$ and up to five iterations for $Gr=10,000$. This is the case regardless of the method chosen to solve the linear system, Eq. (4.1), assuming it is solved accurately.

Straightforward application of a Krylov-Schwarz method to Eq. (4.1) is prohibitively expensive [34]. Furthermore, the number of iterations required for convergence increases dramatically with both $Gr$ and the number of unknowns for a given geometry [34]. These trends are illustrated in Table 4.1.

However, by utilizing the BCALM preconditioner described in Section 4.2.4, $Gr$ does not significantly effect the iteration count, as demonstrated in Table 4.2. Going from $Gr=1\times10^0$ to $Gr=1\times10^4$ without preconditioning results in a threefold increase in the number of iterations required to converge a problem with 30,000 unknowns, while, with preconditioning on four processors, the BCALM preconditioned Bi-CGStab algorithm requires only 30% more iterations. As seen in this table, increasing the number of unknowns results in a lowering of the effect of $Gr$ on BiCGStab iteration count.

Furthermore, a glance down the columns of the table reveals that the number of iterations required for the method to converge the linear system scales sub-linearly with the number of unknowns in the problem. This is an important observation, as if the relationship were linear, then the method would be as computationally expensive as a direct method. This trend is further illustrated in Figure 4.1, where the number of BiCGStab iterations required per Newton's method step is shown as a function of the number of unknowns in the system. The plot reveals that the relationship is approximately $O(N_{eq}^{0.32})$, so the resulting method is effective for extremely large problems.
<table>
<thead>
<tr>
<th>Finite Element Mesh</th>
<th>Finite Element Mesh</th>
</tr>
</thead>
<tbody>
<tr>
<td>32 X 32</td>
<td>48 X 48</td>
</tr>
<tr>
<td>Neq = 13,764</td>
<td>Neq = 30,628</td>
</tr>
<tr>
<td>Gr</td>
<td>AIC</td>
</tr>
<tr>
<td>1</td>
<td>1293</td>
</tr>
<tr>
<td>100</td>
<td>1376</td>
</tr>
<tr>
<td>10000</td>
<td>3173</td>
</tr>
</tbody>
</table>

Table 4.1: AIC per Newton’s iteration for solution of the Jacobian linear system using unpreconditioned Bi-CGStab Krylov algorithm for several finite element mesh resolutions and Grashof numbers (Gr). Where $N_{eq}$ is the number of linear equations. Data from [34].
<table>
<thead>
<tr>
<th>4 Processors</th>
<th>Unknowns</th>
<th>Gr=1</th>
<th>Gr=100</th>
<th>Gr=10,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>32x32</td>
<td>13764</td>
<td>79</td>
<td>69</td>
<td>98</td>
</tr>
<tr>
<td>48x48</td>
<td>30628</td>
<td>103</td>
<td>126</td>
<td>134</td>
</tr>
<tr>
<td>64x64</td>
<td>54148</td>
<td>120</td>
<td>116</td>
<td>156</td>
</tr>
<tr>
<td>96x96</td>
<td>121156</td>
<td>156</td>
<td>142</td>
<td>207</td>
</tr>
<tr>
<td>8 Processors</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48x48</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64x64</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>88x88</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>136x136</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: Average iteration count per Newton’s iteration for various Grashof numbers and fixed problem conditions.
Figure 4.1: Plot showing sublinear relationship between iteration count and number of unknowns.
4.3.2 Choice of Timing Metric

The amount of time required for solution of a fixed problem with a particular drop tolerance and convergence criteria using BCALM preconditioned BiCGStab is directly related to the average iteration count (AIC) for solving the linear system, Eq. (4.1), at each Newton step. This value is shown in Figure 4.2 as a function of number of sub-domains for several problem sizes, using a drop tolerance of $1 \times 10^{-5}$. This figure illustrates that, for a given problem size, the addition of processors does not degrade the robustness of the method. This asymptotic behavior has also been observed when using two-level additive Schwarz as a preconditioner for ill-conditioned elliptic systems [111,145], and is expected because the condition number for these problems has been shown to be independent of the number of subdomains for various less complicated problems [4,28,41].

The number of Krylov iterations required per Newton’s iteration does fluctuate slightly with the exact structure of the decomposition. Because this occurs while the AIC is known to be stable with the addition of processors, a metric which best compares problems on varying numbers of processors is the average time per iteration (ATI), the average iteration count per Newton’s iteration divided by time required for the step. This metric is used in the following sections to gauge method performance.

4.3.3 Efficiency Results

One common way of examining the efficiency of a parallel method is by calculating the speedup for a particular problem size as more processors are used to solve the problem. This metric represents how much faster a problem which can be solved on one processor will execute on multiple processors. Typically, this value is calculated as $S \equiv t_1 / t_N$ where $t_1$ is the time required for the method on one processor and $t_N$ is the time required on $N$ processors. Dividing by the number of processors, then, yields an efficiency. In each case presented, the domain is dissected into a number of domains equal to
Figure 4.2: Average Iteration Count (AIC) as a function of number of sub-domains for the various discretizations of the natural convection in a cavity problem. For the 16x16 discretization, a 4x4 coarse grid was chosen, while for both the 32x32 and 64x64 discretizations, a 8x8 coarse grid was selected.
the number of processors, using Chaco to perform nested spectral sectioning [91]. Several problem sizes were run at a Grashof number of 10,000 with a 4x4 coarse grid, the speedup was calculated, and the efficiencies are reported in table 4.3. Superlinear speedups are observed, in some cases, due to the reduction of the problem dimension on each processor with the increasing number of processors. This gives the speedup metric an "unfair" characteristic, where there is a slight reduction in execution steps that give the parallel code an "unfair" advantage over the single processor case [163]. Since the work per processor is diminishing relative to the work required per processor, the speedup of the method is expected to ultimately decrease with increasing numbers of processors. This is consistent with Ahmdal's law [4].

Because the speedup metric only gives information about the method's performance with problems that can already be solved on a single processor, it is informative to examine the efficiency of the method when then number of unknowns per processors is held constant, calculating the so-called scaled or Gustafson efficiency, $\varepsilon_G \equiv t_1/t_N$ [87,88]. This gives a method of examining the method's performance in the more likely scenario where the problem cannot be run on a single processor machine.

Because holding the number of unknowns per processor exactly the same while generating a conforming coarse grid is not possible, this efficiency is calculated as $\eta \equiv N_{eq,N}t_1/N_{eq,1}t_N$, keeping $N_{eq,N}$ and $N_{eq,1}$ within 10% of each other for comparison.

The factorization work per processor is held constant in this case, so, as the number of processors is increased and, therefore, the overall problem size, the efficiency of the method is impacted by the increasing number of iterations required for problem solution. The Gustafson efficiency is calculated for various per-processor problem sizes and the number of processors in Table 4.4. As expected, the Gustafson efficiency is relatively constant with increasing numbers of processors because the overlap per processor relative to
<table>
<thead>
<tr>
<th>Elements per side</th>
<th>Unknowns</th>
<th>P=2</th>
<th>P=4</th>
<th>P=6</th>
<th>P=8</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>30628</td>
<td>76%</td>
<td>92%</td>
<td>88%</td>
<td>89%</td>
</tr>
<tr>
<td>56</td>
<td>41556</td>
<td>80%</td>
<td>89%</td>
<td>77%</td>
<td>76%</td>
</tr>
<tr>
<td>64</td>
<td>54148</td>
<td>87%</td>
<td>87%</td>
<td>91%</td>
<td>79%</td>
</tr>
<tr>
<td>72</td>
<td>68404</td>
<td>79%</td>
<td>84%</td>
<td>87%</td>
<td>91%</td>
</tr>
<tr>
<td>80</td>
<td>84324</td>
<td>82%</td>
<td>94%</td>
<td>91%</td>
<td>97%</td>
</tr>
<tr>
<td>88</td>
<td>101908</td>
<td>93%</td>
<td>96%</td>
<td>100%</td>
<td>91%</td>
</tr>
<tr>
<td>104</td>
<td>142068</td>
<td>102%</td>
<td>102%</td>
<td>97%</td>
<td>90%</td>
</tr>
<tr>
<td>120</td>
<td>188884</td>
<td>88%</td>
<td>94%</td>
<td>96%</td>
<td>93%</td>
</tr>
<tr>
<td>136</td>
<td>242356</td>
<td>98%</td>
<td>105%</td>
<td>98%</td>
<td>95%</td>
</tr>
</tbody>
</table>

Table 4.3: Efficiency calculated from problem speedup for various problem sizes with Gr=10,000 with a 8x8 coarse grid and drop tolerance of $1 \times 10^{-5}$ for the solution on both the coarse grid and the sub-blocks.
<table>
<thead>
<tr>
<th>Unknowns/P</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>3556</td>
<td>77%</td>
<td>81%</td>
<td>82%</td>
<td>96%</td>
</tr>
<tr>
<td>7828</td>
<td>88%</td>
<td>97%</td>
<td>83%</td>
<td>86%</td>
</tr>
<tr>
<td>13764</td>
<td>80%</td>
<td>95%</td>
<td>103%</td>
<td>106%</td>
</tr>
<tr>
<td>30628</td>
<td>91%</td>
<td>119%</td>
<td>109%</td>
<td>106%</td>
</tr>
</tbody>
</table>

Table 4.4: Gustav efficiency calculated for Gr=10,000 with a 8x8 coarse grid and drop tolerance of 1E-5 for both the solution on both the coarse grid and the sub-blocks.
the number of operations required is also constant. Superlinear speedup is observed because the cost of performing the factorization, fixed for a given per-processor problem size, is amortized over an increasing iteration count.

4.3.4 Comparison with Concurrent Factorization and Storage

To complete the results sections, it is desirable to compare the BCALM preconditioned iterative method described in this chapter to a sparse direct solver, the concurrent factorization and storage (CFS) scheme [128]. The CFS method, based on the incomplete nested dissection ordering of unknowns [77,78], recursively partitions the domain into subdomains and separators, as discussed in Section 1.5. This method was shown to be faster than a commercial iterative code for the natural convection problem for all Grashof numbers [126].

The comparison is between a serial implementation of each code, and the coarse space for the BCALM preconditioner is slightly different as decomposition routines had not yet been generalized for arbitrary geometries [34]. In both cases, the decomposition corresponds to a 16-piece decomposition. These runs were performed on a single node of the IBM SP2 with 512 MB of memory using double precision arithmetic, and a comparison is reported in table 4.5. For the problem sizes run, BCALM preconditioned Bi-CGSTab is faster than CFS in all cases except for the smallest problem and the highest Gr. As problem size increases, the performance difference becomes more extreme.

The advantage of using iterative methods goes beyond execution time for some specific problem in serial. The economy of memory usage and ease of more efficient implementation also makes iterative methods very attractive. For example, table 4.5 is limited to a 31,000 unknown problem because the direct method had already run out of memory, while the iterative method required less than 20% of the memory available on the machine.
<table>
<thead>
<tr>
<th>Discretization</th>
<th>Ununknowns</th>
<th>Gr</th>
<th>CFS</th>
<th>BCALM / BiCGSTAB</th>
</tr>
</thead>
<tbody>
<tr>
<td>4x4</td>
<td>268</td>
<td>1</td>
<td>0.08</td>
<td>0.03</td>
</tr>
<tr>
<td>4x4</td>
<td>268</td>
<td>100</td>
<td>0.08</td>
<td>0.04</td>
</tr>
<tr>
<td>4x4</td>
<td>268</td>
<td>10000</td>
<td>0.08</td>
<td>0.09</td>
</tr>
<tr>
<td>8x8</td>
<td>948</td>
<td>1</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>8x8</td>
<td>948</td>
<td>100</td>
<td>0.4</td>
<td>0.2</td>
</tr>
<tr>
<td>8x8</td>
<td>948</td>
<td>10000</td>
<td>0.4</td>
<td>0.3</td>
</tr>
<tr>
<td>16x16</td>
<td>3556</td>
<td>1</td>
<td>3.0</td>
<td>0.9</td>
</tr>
<tr>
<td>16x16</td>
<td>3556</td>
<td>100</td>
<td>3.0</td>
<td>0.9</td>
</tr>
<tr>
<td>16x16</td>
<td>3556</td>
<td>10000</td>
<td>3.0</td>
<td>1.4</td>
</tr>
<tr>
<td>32x32</td>
<td>13764</td>
<td>1</td>
<td>95.0</td>
<td>4.7</td>
</tr>
<tr>
<td>32x32</td>
<td>13764</td>
<td>100</td>
<td>95.0</td>
<td>5.7</td>
</tr>
<tr>
<td>32x32</td>
<td>13764</td>
<td>10000</td>
<td>95.0</td>
<td>5.9</td>
</tr>
<tr>
<td>48x48</td>
<td>30628</td>
<td>1</td>
<td>827.0</td>
<td>13.5</td>
</tr>
<tr>
<td>48x48</td>
<td>30628</td>
<td>100</td>
<td>827.0</td>
<td>15.0</td>
</tr>
<tr>
<td>48x48</td>
<td>30628</td>
<td>10000</td>
<td>827.0</td>
<td>18.2</td>
</tr>
</tbody>
</table>

Table 4.5: Comparison of the CFS sparse direct solver to BCALM preconditioned BiCGSTAB.
Furthermore, the BCALM preconditioned iterative method is more scalable than CFS. Parallel efficiencies for CFS based on parallel speedup are reported in table 4.6. Note that, based on execution time, these efficiencies are far lower than reported in [126]. Contrast these observed efficiencies with those obtained for BCALM preconditioned Bi-CGSTab in Table 4.3. The performance of CFS is degraded by the increasing algorithm complexity with increasing numbers of processors and unknowns.
<table>
<thead>
<tr>
<th>Discretization</th>
<th>Unknowns</th>
<th>Speedup</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>32x32</td>
<td>13764</td>
<td>2.3</td>
<td>57%</td>
</tr>
<tr>
<td>32x64</td>
<td>27300</td>
<td>2.7</td>
<td>67%</td>
</tr>
<tr>
<td>48x64</td>
<td>40724</td>
<td>2.8</td>
<td>70%</td>
</tr>
</tbody>
</table>

Table 4.6: Efficiencies for four processor runs on the IBM SP2. Efficiency is based on speedup instead of operation count. For contrast, when based on computation rate as in [126], the efficiency is 83% for the problem with 13,764 unknowns.
4.4 Conclusions

The largest limitation that iterative methods possess is their lack of robustness as a class of solution methods. In general, they neither terminate in a finite number of steps, nor are guaranteed to converge at all.

This Chapter lays out a robust method for iterative solution of linear equation sets which arise in the context of finite element discretizations of complex viscous flow problems. Based on a popular message passing library, it is portable to the vast majority of parallel computers in use today. The combination of both block preconditioning techniques based on Schur complement reduction and multilevel additive Schwarz allows the generation of a sophisticated preconditioner appropriate for the complex equations of mixed type that arise when modelling natural convection in a cavity and other viscous flows. Furthermore, the design of the preconditioner allows the efficient exploitation of parallelism. When coupled with a Krylov subspace method, the accurate solution of indefinite and asymmetric equation sets that result from the discretization of large classes of differential equations may be obtained.

Schur complements and multilevel additive Schwarz play important roles in the solution of large, sparse, linear systems of equations. Schur complement reduction has traditionally been utilized as a domain decomposition technique in combination with direct solvers to partition and reduce the size of the linear system. Additive Schwarz has typically been used as a preconditioner for the solution of symmetric linear systems which arise from the discretization of elliptic operators. Here, these notions have been expanded by defining a parallel block preconditioner for the fully coupled linear system that arises from the discretization of general viscous flow problems.

Through the utilization of spectral dissection to assign nodes to processors in a minimally connected manner, the scalability of the method is ensured. The Schur complement
and additive Schwarz parts of the preconditioner are parallelized, and the inclusion of the serial solution of the coarse grid problem does not impact the overall method's scalability. As nearest neighbor mapping was not utilized, the method is best suited to parallel topologies that have efficient all-to-all networking, as is most common in the parallel computers in use today.

The results for the natural convection problem in this Chapter document the excellent performance of the BCALM preconditioned Krylov solver put forward. The incorporation of a block preconditioning scheme based on a Schur complement of the pressures yields a robust method which is insensitive to the magnitude of the Grashof number. Furthermore, the resulting scheme is highly parallelizable, yielding both single-problem and Gustafson efficiencies over 90% for large problem sizes. The method is much faster than even a state-of-the-art direct method, CFS. Finally, the preconditioner does not exploit either the natural convection problem specifically nor the regular geometry encountered. As will be seen in Chapter 5, the method is highly generalizable.
Chapter 5

The Parallel Simulation of Transient Viscoelastic Flows

5.1 Introduction

The ultimate goal of this thesis is the application of the parallel processing methods developed here, and described in Chapter 5, to the calculation of viscoelastic flows governed by complex differential constitutive equations. The background for these equations is described in Sections 1.1, 2.3 and 2.4. Calculations are presented which focus on the Oldroyd-B model as a simple, prototypical model that is well established for benchmark calculations. As described in Section 5.1.1, these calculations are extremely difficult at reasonable values of Deborah number because of the steep boundary layers in stress that form and the subtle coupling between the velocity and stress fields. These features are well portrayed in the benchmark problem used here -- the two-dimensional flow around a cylinder in a channel.

5.1.1 Flow Around a Cylinder in a Channel: Two-Dimensional Benchmark

The flow of viscoelastic fluid around a cylinder in a channel has received attention in recent years [31,116,117,125] and is relevant to the work in this thesis as the $L = \infty$ limit of the case of an infinite linear array of cylinders, presented in Section 5.3. The geometry for this flow is characterized in Figure 5.7 by the channel width $H$ and the cylinder radius $R$. The magnitude of elastic contributions to the flow is characterized by the Weissenberg number, $We = \frac{\lambda \langle \nu \rangle}{r}$, where $\lambda$ is the zero-shear-rate relaxation time of the polymer, $r$ is the radius of the cylinder, and $\langle \nu \rangle$ is the average velocity in the unobstructed channel. Note that for our geometry, $De = We$ as shown in Eqs. 5.13 and 5.14.

The contours of the stream function, and the normal and shear stress components for a Newtonian fluid are shown in Figure 5.1 as calculated by Liu [116]. Symmetry about the
Figure 5.1: Solution fields for Newtonian flow around a single cylinder in a channel with $H=2$. (a) Ten equally spaced contours are shown for the streamlines. Twenty equally spaced contours between the minimum and maximum values are shown for (b) the normal stress $\tau_{xx}$ and (c) the shear stress $\tau_{xy}$. In each plot, the maximum and minimum locations are denoted by an X and a dot, respectively. From [116].
plane \( x = 0 \) is found for the axial velocity \( v_x \) and shear stress \( \tau_{xy} \), and antisymmetry is observed about \( x = 0 \) for \( v_y \) and \( \tau_{xz} \). Fluid accelerates as it moves towards the gap region at \( x = 0 \). The channel wall causes pronounced shearing in the gap region, so the maximum and minimum of shear stress occurs in the gap region on the channel wall and the surface of the cylinder, respectively. The maximum normal stress, \( \tau_{xx} \), occurs upstream on the cylinder surface and the minimum normal stresses occurs downstream on the cylinder surface.

Liu [116] compared the viscoelastic models for Oldroyd-B [22], presented in Section 1.1, along with those for FENE-P [22], FENE-CR [42] and Giesekus models [22] for flow around a confined cylinder. The qualitative features of this flow are summarized here. As the Weissenberg number is increased, elastic effects result in a progressive modification of the velocity field around the cylinder, as illustrated in Figure 5.2 by Liu [116]. The dimensionless centerline velocity is affected. Velocity profiles upstream of the cylinder are compared, normalized with the average velocity across the channel, \( \langle v \rangle \), and is not affected as \( \text{We} \) is increased. In the wake of the cylinder, however, a slight increase in strain rate is observed relative to Newtonian flows. The polymer solutions exhibit a pronounced downstream shift in the velocity relative to Stokes flow. The viscoelastic models predict that the velocity of the fluid moving away from the cylinder recovers more slowly to its ultimate value than Newtonian.

The axial stress, \( \tau_{pzz} \), is shown in Figure 5.3. There are four regions where polymer molecules are highly extended. Polymers first become deformed in the compressive region \( (\tau_{pzz} > 0) \) upstream of the cylinder, where the flow is nearly planar elongation with a negative elongation rate in the \( x \)-direction. Near the front stagnation point, the flow is slow and the macromolecules have sufficient residence time to become highly stretched. Two additional regions of extension are found near the surfaces of the cylinder and the channel.
Figure 5.2: Axial velocity $v_x$ along the symmetry line for flow around a single cylinder in a channel with $H = 2$ (a) upstream and (b) downstream of the cylinder. Results are for $We = 1.0$. From [116].
Figure 5.3: Comparisons of (a) axial normal stress $\tau_{xx}$ and (b) average molecular extension for flow around a single cylinder ($H=2$) at $We=0.5$ for three viscoelastic models. From [116].
Finally, in the wake of the cylinder, large molecular extension is observed along the stagnation streamline where polymer molecules are stretched by the planar elongational flow near the rear stagnation point are advected downstream, forming a "birefringent strand" before relaxing back to equilibrium.

Renardy [151] studied the flow of an upper convected Maxwell fluid [22] past a cylinder in the limit of high De for a fixed Newtonian velocity field. Renardy then integrated the stresses in this fixed velocity field and studied the asymptotic behavior of the resulting stress field in the limit of high De. In his work, the width of the boundary layer near the cylinder surface was predicted to be $O(\text{We}^{-1})$ while the thickness of the birefringent strand in the wake was predicted to be $O(\text{We}^{-2})$. These predictions are compared with the current results in Section 5.3.

Planar stagnation flow of a Maxwell fluid was considered by Phan-Thein [146] and an exact solution was obtained. The flow geometry for this problem is that of opposing flow on a horizontal plate that meets at a line of symmetry and then flows upwards, away from the plate. A stagnation point exists on the plate surface at the plane of symmetry. Using a similarity solution, Phan-Thein's work predicted that stress at the stagnation point should scale as

$$O\left(\frac{1}{2\text{De} - 1}\right).$$

(5.1)

for this flow and is singular at $\text{De} = 0.5$. While this is not the same geometry considered here, planar stagnation flow may be considered as the limiting flow at the cylinder surface where curvature is negligible. Phan-Thein's solution is discussed in the context of the current work in Section 5.3.

A sophisticated experimental and theoretical investigation of the viscoelastic instability observed in the flow of a Boger fluid around an isolated cylinder in a channel was pre-
sented by Öztekin et al. [141]. Here a laser Doppler velocimeter was used to determine the critical value of the Weissenberg number for the geometry, and the extent of the flow instability. This was done for several aspect ratios, $H/R$. These experimental results were then compared with a linear stability analysis performed with the Oldroyd-B and Giesekus fluids on a wedge-shaped region in the cylinder wake. A flow transition was predicted by the linear stability analysis that agreed qualitatively with the trends in the experimental data.

The geometry modelled by Öztekin et al.'s [141] linear stability analysis was flow around a cylinder of radius $R$ confined inside a concentric porous cylinder of radius $R_w$, as shown in Figure 5.4. The porous outer cylinder enforces a uniform velocity $U$ at a distance $R_w$ from the center of the inner cylinder. The form of the solution postulated by Öztekin is:

![Figure 5.4: Flow geometry for the linear stability analysis presented by Öztekin et al. [73] for flow of an Oldroyd-B fluid around cylinder of radius $R$ confined in a concentric porous cylinder of radius $R_w$. A uniform streaming velocity $U$ is enforced at the surface of the porous cylinder. This figure is taken from reference [141].](image-url)
\[ v_r = f(r) \cos \theta \]  \hspace{1cm} (5.2)

The ratio of the radius of the porous cylinder to the radius of the inner cylinder \( R_w/R \) was intended to have the same influence on the flow as changing the ratio of the channel half-width to the radius of the cylinder \( H/R \) in the flow around a cylinder in a channel. Öztpek solves for \( f(r) \) as

\[ f(r) = A + B \ln r + C r^2 + \frac{D}{r^2} \]  \hspace{1cm} (5.3)

where

\[ A = -(C + D) \]  \hspace{1cm} (5.4)

\[ B = 2(D - C) \]  \hspace{1cm} (5.5)

\[ C = \Lambda D \]  \hspace{1cm} (5.6)

\[ D = \frac{1}{\left[ \Lambda(-1 - 2 \ln h + h^2) + \left( -1 + 2 \ln h + \frac{1}{h^2} \right) \right]} \]  \hspace{1cm} (5.7)

\[ \Lambda = \frac{(h^2 - 1)}{(h^2 - h^4)} \]  \hspace{1cm} (5.8)

In Section 5.3, a comparison is included between Öztpek et al. and the simulations in this thesis.

In terms of accuracy, none of the four viscoelastic models examined in Liu [116] accurately predict either the long birefringent wake nor the displacement of the centerline axial velocity profile downstream from the cylinder observed in experiment. Furthermore, these models do not predict the increased drag force on the cylinder observed in experiment. This most likely due to the inability of these models to capture the extensional rheology that characterizes this geometry. For example, measurements of the extensional viscosity for PIB Boger fluid asymptote at large strains, reaching a finite upper limit. By comparison, Oldroyd-B predicts an extensional viscosity that grows without bound. Constitutive
equations which are modified to predict bounded extensional viscosity such as FENE-CR, FENE-P, and Giesekus, however, actually predict birefringence, axial velocity profiles and drag less accurately than Oldroyd-B.

This need for better rheological models has been answered, in part, by work done by Ghosh [82,105]. This new model provides a better description of transient extensional rheology by using input from stochastic simulations of multi-bead spring models to better capture the hysteresis in the uncoiling of the polymer. Predictions from this model have been promising, based on modelling flow around cylinders between parallel plates. Both the birefringent strand and increased drag forces are better represented for Deborah numbers up to 0.9, however, it has been infeasible to increase the Deborah number beyond this limit and further test this model.

As the Deborah number in these numerical simulations is increased, the gradient in the stresses field near the surface of the cylinder grows. Consider the velocity gradient between the top of the cylinder and the wall for the case of flow around an isolated cylinder [105], illustrated in Figure 5.5. Even with increasing Deborah number, the gradient in the region nearer the cylinder than 0.1 R remains flat, corresponding to the viscometric boundary layer. To resolve the viscometric boundary layer in the velocity field, therefore, requires elements smaller than this length scale. This is likely all that must be considered for the case for Newtonian flows. However for polymeric flows, the stress does not level off until the flow is much closer to the cylinder. This trend is shown in Figure 5.6. Furthermore, the disparity between width of the viscometric region for the velocity and stress gradients becomes more severe with increasing Deborah number. The thickness of the viscometric boundary layer for the stresses is often two orders of magnitude thinner than that for the viscometric boundary layer for the velocities.
Figure 5.5: The asymptotic behavior of the velocity gradient near the cylinder surface for flow around an isolated cylinder corresponding to a beta of 0.59 and a lambda of 0.793. The viscometric boundary layer extends approximately 1E-1 away from the cylinder surface for all Weissenberg numbers considered. [105]
Figure 5.6: The asymptotic behavior of the stress field near the cylinder surface for flow around an isolated cylinder corresponding to a beta of 0.59 and a lambda of 0.793. The viscometric boundary layer is closer to the cylinder than observed for the velocity gradient. Furthermore, as the Weissenberg number is increased, the magnitude of the stresses near the surface becomes both larger and closer to the surface. As a result, the viscometric boundary layer must either be completely missed by a mesh or a number of elements must be placed in the layer to resolve it. [105]
This behavior of the stresses near the cylinder surface has recently been blamed for causing a peculiar effect when modelling viscoelastic flows: As the mesh becomes increasingly refined, the maximum Deborah number which can be simulated to a steady state has been observed to decrease. However, recent work by Joo [105] has shown that this most likely occurs due to the size of the elements reaching the same dimension of thickness as that of the stress boundary layer. By placing a number of elements within this very thin boundary layer, this work demonstrated that, for contraction flow, increasing the mesh refinement allows higher Deborah numbers to be simulated for steady state flows.

With previous numerical methods, attempting to put elements as small as $1 \times 10^{-4}$ into the mesh for flow around a periodic array of cylinders rendered the problem intractable due to the large number of unknowns in the resulting discretization. In the recent work by Smith [167], the smallest elements considered for this problem were $O(1 \times 10^{-2} R)$, limiting the Deborah numbers considered to $De = 0.85$.

5.1.2 Flow Around a Cylinder in a Channel: Experiments and Flow Instabilities

This study is motivated by the experimental observations of the flow of polyisobutylene Boger fluids by McKinley et al. [125] and by Byars [31] for an isolated cylinder and by Liu [116,117] for arrays of cylinders. For both the isolated cylinder and the arrays of cylinders, a transition to a three-dimensional cellular structure in the cylinder wake is observed at a critical value of $De$. Varying the spacing between the cylinders leads either to a shear dominated flow, for close cylinder spacings, or to a mixture of shear and elongational kinematics, for the widely spaced cylinders and the isolated cylinder.

5.1.3 Chapter Outline

The purpose of this Chapter is to demonstrate the utility that BCALM preconditioned iterative methods have in viscoelastic flow modelling. First, Section 5.2 presents a review of previous attempts at using sophisticated methods to render larger viscoelastic flow
problems tractable. The parallel time dependent simulation of viscoelastic flow around a cylinder is presented in Section 5.3. First, the solution of problems with a smaller mesh size than previously considered are presented, showing that with the addition of elements near the surface of the cylinder it is possible to drive the system to higher Deborah numbers than previously possible with increasing mesh resolution. The parallel performance of the method is then described. The application of the algorithm to the linear stability problem is covered in Section 5.4, documenting its parallel performance.
5.2 A History of Solving Large Viscoelastic Flow Problems

No numerical approximation to the viscoelastic flow problem has created a substitute for enormous mesh refinement of the approximation to capture fine-scale structure of the flow field and, hopefully, be the basis for predictions of flow stability. A discussion of the development of numerical methods and of test calculations “in search of high Deborah number solutions” follows. While parallel solver technology has not been utilized previously to solve large viscoelastic flow problems, sophisticated solver technology has been brought to bear on these problems in the past. Additionally, research on iterative methods and parallelism in other fields provides some of the background on which the BCALM preconditioner and its application to the viscoelastic flow problem is based.

While the large majority of viscoelastic flow simulations have been accomplished with algorithms based on frontal LU decomposition methods for solvers [47,116,117,159,176], as early as 1986 iterative methods were applied to solving the linear systems that aries in the modelling of viscoelastic flow. That year, flow of a second order fluid past a cylinder, discretized spatially using Galerkin finite elements, was modelled at steady state using Gauss-Seidel to solve the system of equations that resulted from successive approximation. [162]. At that point, however, Krylov methods which could be applied to complex, nonsymmetric and indefinite linear systems had yet to be developed. That year marked the publication of GMRES, the first Krylov method to achieve notoriety that was both capable of being applied to these systems and guaranteed to converge in a finite number of steps [156]. By 1989, Fortin and Fortin [65] applied this method to model stick-slip flow of an Oldroyd-B fluid. Discretization in this work was carried out using Lasaint-Raviart finite element method [65]. The resulting nonlinear equations for steady state flow were solved using Newton’s method. Convergence of their calculations was limited, however, to Deborah numbers less than 4.5 for this problem, where frontal methods were capable of con-
verging this discretization through a Deborah number of 20 on the same mesh [121]. Later work incorporated GMRES into a steady-state solution technique for the modified EVSS discretization of a PTT fluid [67]. Failure of GMRES in this method was noted for contraction flow and was attributed to the high stresses near singularities in the flow geometry [67]. Tsai and Liu [182] took a different approach to the steady-state solution of the stick-slip flow of an Oldroyd-B fluid discretized using the finite element method of Marchal and Crochet [121]. They applied Bi-CGSTAB and GMRES to the fully coupled linear systems that arise during Newton’s method solution, but were still incapable of reproducing the high-Deborah number flows associated with the frontal method based simulations [182]. Furthermore, this method required an LU-factorization for the preconditioner, so the resulting scheme was not faster than nor less memory intensive than standard frontal techniques. Using a block preconditioner scheme, Baaijens [8] was able to model the steady state axi-symmetric contraction flow of an upper-convected Maxwell fluid using a DEVSS-DG finite element method coupled with Newton’s method over a wide range of Deborah number. The method, however, degrades with both increasing Deborah number and number of unknowns.

Overcoming the shortcomings encountered in applying iterative methods in these previous works is a goal of this thesis. In the works discussed above, stabilizing the effect of Deborah number on convergence of the linear iterative solver was a fundamental challenge. Furthermore, none of the preconditioning techniques put forth in these methods are generally applicable beyond the framework put forward for viscoelastic flow. Furthermore, while noting that iterative methods are attractive due to their low memory requirements and the ability for parallel computation, none of the previous work actually exploited this capability. Parallelization of the preconditioning schemes put forward in the prior work could not be accomplished effectively. In the work by Tsai and Liu [182] a
frontal method factorization was used as a preconditioner. In the work by Baaijens [8], an LU factorization was built into the block preconditioner. As a result, the problem sizes (level of mesh refinement) attempted in the previous work are very modest. The largest problem reported by Tsai and Liu [182] in 1995 had 29,177 unknowns while the largest problem by Baaijens [8] in 1998 had 62,128 unknowns.

The parallel viscoelastic flow code has been developed by Dou and Phan-Thien [53], based on finite volume discretization and the SIMPLER algorithm for pressure-velocity coupling in the generalized viscous problem. However, the largest problem attempted by these researchers has approximately 100,000 degrees-of-freedom. This limit is likely due to their observed decrease in convergence rate with increasing numbers of processors and unknowns. This algorithm, it should be emphasized, is a domain decomposition technique and not a solver: The SIMPLE and SIMPLER algorithms dictate a solution ordering to which a Gauss-Seidel method is applied [53]. Finally, a review of recent papers co-authored by Baaijens [24], Phan-Thien [54], Fortin [66] and Khomami [168] reveals that iterative methods are not being used with time dependent simulations. This is likely due to the shortcomings noted above combined with the repeated application of an iterative method that a timestepping technique requires.

The goal of the current work has been to provide a preconditioner that may be combined with a Krylov subspace iterative method to address these shortcoming. Specifically, the design of the BCALM preconditioner has been implemented such as to be:

- Insensitive to flow parameters
- Generalizable for viscous flow problems
- Useful in parallel to allow the solution of previously intractable problems
- Useful for both steady-state and transient flow problems
This is without precedent in the field, yet the BCALM preconditioner, coupled with a Krylov iterative method as described in Chapter 4, provides a novel starting point.

Examination of the linear systems that arise from discretizing the conservation equations, Eqs. 2.72-2.74, suggests the most obvious reason that straightforward application of an iterative method to these linear systems might be unsuccessful is the block singularity introduced into the linear system by the continuity equation. Consider ordering the equations such that the vector of unknowns is expressed

\[
x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}
\]  

(5.9)
as in Chapter 4 for the natural convection problem. Here, the pressure unknowns are in the partitioned vector \( x_2 \), and all other unknowns are in \( x_1 \). For a steady-state solution, the linear system of equations that results from Newton’s method:

\[
\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}.
\]  

(5.10)

This is the same form observed in Eq. 4.5 for the natural convection problem, and straightforward application of a BCALM-preconditioned Krylov iterative method would be a ready extension of the methods described in Chapter 4. As seen in Chapter 4 for the natural convection problem, utilizing this block complement framework for the current work has allowed generalization of the formulation in a way that is relatively insensitive to problem-specific parameters. This chapter extends the BCALM-preconditioned iterative method put forward for steady-state simulations in Chapter 4 to time dependent flow.

The ideas of block complement reduction [149], multilevel Schwarz techniques [166], and inexact solvers [164], described in detail in Chapter 2, all contributed to the method described here. By combining these elements a powerful preconditioned iterative solution
scheme has been developed, utilizing the BCALM preconditioned Krylov methods. The effectiveness of this method for the time-dependent simulation of viscoelastic fluid flows is established below.
5.3 The Parallel Solution of Transient Viscoelastic Flow Around a Periodic Array of Cylinders

The steady creeping flow of a viscoelastic liquid past a linear array of cylinders along the centerline of a rectangular channel is considered. The problem is shown schematically in Figure 5.7. The flow is two dimensional with the primary flow in the x-direction. Here, the radius of each cylinder is , the channel half-width is $H_C$, and the center-to-center cylinder spacing is $L_C$. The physical configuration is then specified by two dimensionless quantities:

$$L \equiv \frac{L_C}{R_C} \quad (5.11)$$

$$H \equiv \frac{H_C}{R_C} \quad (5.12)$$

The characteristic velocity is taken to be the average $x$-component of the velocity across the unobstructed channel, $\langle v \rangle$. By taking $\lambda$ as the zero-shear-rate relaxation time of the polymer solution, the flow is controlled by one of two dimensionless groups, the Deborah and Weissenberg numbers:

![Figure 5.7: Schematic diagram for flow past a linear, periodic array of cylinders confined between parallel plates. The cylinders have radius $R_C$, and the geometry is specified by the cylinder-to-cylinder spacing $L_C$ and the channel half-height $H_C$. Computations were performed on the whole channel in the unit cell.](image)
\[ \text{De} \equiv \frac{\lambda \langle v \rangle}{R_c} \quad (5.13) \]

\[ \text{We} \equiv \frac{\lambda \langle v \rangle}{H_c - R_c} \quad (5.14) \]

For the geometry considered, \( H_C = 2R_c \), so these two quantities are the same.

As described in Section 2.6.2, the RK-4 timestepping scheme is applied to the governing equations for the DEVSS-G formulation of an Oldroyd-B fluid [167]. The governing equations, as previously presented in Section 2.3, are written as

\[ \tau_p + \text{De} \left[ \frac{\partial \tau_p}{\partial t} + v \cdot \nabla (\tau_p - G^T \cdot \tau_p - \tau_p \cdot G) \right] = (1 - \beta) [G + G^T] \quad (5.15) \]

\[- \nabla^2 \nu + (1 - \beta) \nabla \cdot [G + G^T] + \nabla \cdot \tau_p + \nabla p = 0 \quad (5.16) \]

\[ G \cdot \nabla \nu = 0 \quad (5.17) \]

\[ \nabla \cdot \nu = 0 \quad (5.18) \]

These equations are discretized using discontinuous Galerkin finite elements for the constitutive equation, Eq. (5.15), as discussed in Section 2.4.3, and standard Galerkin finite elements for the conservation equations, Eqs. (5.16)-(5.17), and constraint, Eqs. (5.18), as discussed in 2.4.2.

As discussed in Section 2.6.2, application of the fourth order Runge-Kutta timestepping method (RK4) requires that the algebraic, kinematic variables be separated from the differential expression for the stress. This results in the system recovered from the discretization of Eqs. (5.15)-(5.18):

\[ M \cdot \frac{d\tau_p}{dt} = \frac{1}{\text{De}} f(\tau_p, \nu, G) \quad (5.19) \]

\[ g(\tau_p, \nu, p, G) = 0 \quad (5.20) \]
In Equation (5.19), $f$ is the discretization of the constitutive equation, and $g$ is a Stokes-like problem for the kinematic variables resulting from the discretization of Eqs. (5.16)-(5.18). Equation (5.20) is invertible for the kinematic unknowns ($v, p, G$), so Eq. (5.20) is written as

$$
\begin{bmatrix}
v \\
p \\
G
\end{bmatrix} = g(\tau_p)
$$

(5.21)

Equation (5.21) is substituted into Eq. (5.19) to write a function in terms of the differential variable alone, as is necessary for the application of Runge-Kutta. We define a new function $\hat{f}$ as

$$
\hat{f}(\tau_p) \equiv M^{-1}f(\tau_p, g(\tau_p)).
$$

(5.22)

The Fourth order Runge-Kutta algorithm is written as

$$
\begin{align*}
k_1 &= \Delta t \hat{f}(\tau_p^n) \\
k_2 &= \Delta t \hat{f}(\tau_p^n + \frac{k_1}{2}) \\
k_3 &= \Delta t \hat{f}(\tau_p^n + \frac{k_2}{2}) \\
k_4 &= \Delta t \hat{f}(\tau_p^n + k_3) \\
\tau_p^{n+1} &= \tau_p^n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{align*}
$$

(5.23)

Each step of Eq. (5.23) then requires the implicit solution of a Stokes-like problem for the kinematic variables. The constitutive equation is then evaluated explicitly for the stresses. Each of these steps is accomplished in parallel.

The Stokes-like problem for the kinematic variables is solved using a BCALM-preconditioned Krylov subspace method, either GMRES or BiCGStab. Discretization of Eq. (5.16)-(5.18) leads to the solution of a linear system of the form
\[
\begin{bmatrix}
A & B \\
C & D
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2
\end{bmatrix}
= 
\begin{bmatrix}
b_1 \\
b_2
\end{bmatrix}
\]

(5.24)

Here, the blocks \( A \) and \( B \) correspond to the discretization of the conservation and constraint equations, Eqs. (5.16)-(5.17), and blocks \( C \) and \( D \) correspond to the discretization of the continuity equation, (5.18). With this structure, the preconditioner developed in Chapter 4 is readily applied using a matrix lumping coarse space instead of a Galerkin coarse space. Both of these coarse spaces are described in detail in Section 2.2.3.

For the explicit update of the stresses, the elements decouple on an element-by-element basis. Therefore, this step is both extremely fast and completely parallelizeable. Results for this Section are divided in two parts. The first Section presents results based on simulating problems with larger number of unknowns than were previously tractable. Then, the efficiency of the parallel time-dependent RK4 scheme with the imbedded BCALM preconditioned Krylov solver for the Stokes-like problem are presented; in these calculations the discontinuous Galerkin explicit stress update is used to approximate the stresses.

5.3.1 Pentium Cluster Description

To allow comparison with other researchers, it is important to characterize the computer system on which the computations presented in this Chapter were performed. This requires more explanation than for the case of the calculations in Chapter 4, because, unlike a system which is fully installed by the vendor, the operating system, compilers utilized, drivers utilized, as well as the hardware selected all interact to effect the method’s performance.

These calculations were performed on a cluster of sixteen, dual-processor Dell Precision 610 workstations [48]. The performance of this cluster is documented in Section 3.2.1. Each workstation is configured with one Giga-byte of RAM, and two Pentium III
Xeon-based CPUs, each with one Mega-byte of on-chip cache. These machines have a CPU clock speed of 500 MHz, and each processor is capable of 58.4 MFLOP/s on the LINPACK DGEFA 1000x1000 benchmark problem described in Section 3.2.1 [52]. Networking between the individual workstations was accomplished using 1000 Mbps Gigabit Ethernet (1000BASE-SX) [96]. Each workstation has a Packet Engines G-NIC II adapter card and the workstations are connected to each other via a Packet Engines Power Rail 2200 networking switch [142].

Each workstation runs the Linux 2.2.16 operating system [180], however support for these network adapters is not included in the base kernel distribution. The driver for this network card is based on a driver originally written by Donald Becker [16], but it has been rewritten for stability by Eric Kasten [108] and further modified by Keith Underwood [183] and this author. The FORTRAN and C compilers used are from the EGCS-2.91.66 distribution for Linux [9]. The MPI library from Notre Dame, LAM MPI version 6.3.2 was utilized [20].

5.3.2 Pushing for Higher Deborah Numbers: The Parallel Simulation of Large Viscoelastic Flow Problems

Previously, due to limitations on the ability of frontal solution techniques to be applied to extremely large problems, researchers simulating viscoelastic flow were limited to \(O(1\times10^5)\) degrees of freedom. For the case of flow around a periodic array of cylinders, this left researchers with a choice. Either use a coarse mesh and completely miss the boundary layer physics or attempt to resolve, as much as possible, the boundary layers through mesh refinement. Both of these techniques have drawbacks.

The first approach, using a coarse mesh for simulation purposes, was used by Sun et al. for testing a new finite element formulation, DAVSS-G/DG in [174]. Broadly, these researchers sought to modify the DEVSS-G/DG method by incorporating an adaptive vis-
cosity, defined as a function of polymeric stresses and gradients in the flow field. By leaving the grid relatively coarse near the cylinder surface, it is possible to drive the flow to relatively high Deborah numbers. Furthermore, it is still possible to obtain good representations of drag force on the cylinder. However, the representation of the solution near the cylinder surface is very poor for high De. This effect is illustrated in Figure 5.8, where the adaptive viscosity is plotted as a function of distance from the cylinder surface. The poor solution results from the steepness of the stress rise in the wake of the cylinder, and this poor solution representation was also observed in the velocity gradient field and eventually the stress field. This oscillation was attributed to the inability of the mesh to resolve the extremely steep behavior of the stress field near the cylinder. As a result, the solutions obtained are inappropriate for use as a steady-state base state for linear stability calculations.

Due to the interest in performing linear stability analysis on the steady state solution obtained from flow around a cylinder, other researchers have attempted to refine the mesh near the cylinder surface to resolve the viscometric boundary layer. However, increasing mesh refinement led to a decrease in the maximum attainable Deborah number. Work by Smith [167] suggested that this effect is caused by steep gradients in the $G_{xx}$ field and investigated the relationship between $G$ and $\tau_p$ along the center-plane downstream from the cylinder.

Smith observed that, close to the stagnation point, the contribution to the r-component of the momentum equation from the $r\theta$-component of the polymer stress becomes large. He suggested that this effect was due to the convection of the stress from the top and bottom of the cylinder into the cylinder wake, illustrated in Figure 5.9. Smith’s work also observed a similar structure in $\tau_{r\theta}$. While the shear stress is zero along the center-plane,
Figure 5.8: Oscillations in the adaptive viscosity near the cylinder surface. Similar oscillations were also reported in the stress field. These oscillations result in a steady-state solution which is inappropriate for use in linear stability analysis, and are likely due to the inability of the mesh used to resolve the viscometric boundary layer in the stresses [174].
Figure 5.9: Contours of $\tau_{pr\theta}$ for flow of an Oldroyd-B fluid around an isolated cylinder. Results were calculated with mesh B for $We = 0.833$ and $\beta = 0.59$. Taken from [167].

the gradients in $\tau_{pr\theta}$ in the $\theta$-direction are large. This effect is illustrated in Figure 5.10, where $\tau_{pr\theta}$ and $\tau_{r\theta\theta}$ are plotted along the curve $\theta = 0.034$ (radians) is plotted. The $\theta\theta$-component of the polymer stress made an $O(1)$ contribution to the momentum balance, as seen in this Figure. While the magnitude of the $r\theta$-component of $\tau_p$ is small, the gradients are significant, the variation in $\tau_{pr\theta}$ making an $O(10)$ contribution to the momentum equation. Qualitatively, it is observed that both $\tau_{pr\theta}$ and $\tau_{r\theta\theta}$ change on the same length scale as $G_{xx}$ in the cylinder wake.

These observations caused Smith [167] to conjecture that the numerical difficulties in calculating flow around a cylinder for large values of $De$ are not due to the development of the large normal stresses in the purely elongational flow along the center-plane, but rather due to convection of stress from the shearing region between the cylinder and the wall into the cylinder wake. He then observed that resolution of the gradients in $G$ will require the
Figure 5.10: The rθ-component (filled symbols) and the θθ-component (open symbols) of the polymer stress at an angle of 0.034 radians away from the center-plane in the cylinder wake for values of the Weissenberg number of 0.667 (▲), 0.75 (♦) and 0.833 (●). Taken from [167].

Careful calculation of $\tau_{r\theta}$, which develops along the entire surface of the cylinder. As a result, Smith's calculations were limited to a Deborah number less than 0.85, compared to 1.05 for DEVSS-G/SUPG in the work by Sun, et. al [174]. Smith characterized the mesh refinement needed to resolve this phenomena as "excessive" due to its expense.

This numerical difficulty was also observed by Phan-Thien and Dou for the DEVSS-Omega formulation for flow around widely spaced cylinders [53]. The normalized drag force on the cylinder was determined for five meshes, ranging in number of nodes from 1234 in mesh M1 to 58,550 in mesh M5. The drag force calculated there is shown in Figure 5.11 as a function of De. The extremely coarse meshes could be driven to De > 2.0. However, for mesh M4, convergence to steady-state was limited to the maximum Deborah number, De = 0.8. With the finest mesh used in [53], converged solutions were obtained
Figure 5.11: The normalized drag force on a cylinder for five mesh sizes. Increasing mesh refinement results in method failure at successively lower Deborah numbers, yet the method is $O(h)$ accurate. Taken from [53].
only to \( \text{De} = 0.4 \). The trend of a restricted domain of convergence with increasing mesh refinement is obvious.

Furthermore, because of the low-order accuracy of the finite volume method of Phan-Thien and Dou, these researchers extrapolated their solution to \( h=0 \) for comparison with other researchers. This puts these researchers in the uncomfortable position of having a solution which is increasingly accurate as the mesh size is decreased, but seeing mesh refinement prevent them from converging to high Deborah number flows.

**5.3.3 Mesh Refinement to Capture Boundary Layers**

The purpose of the discussion in this Section is to show that, through the use of parallelism additional mesh refinement may be carried out that was not previously possible. With the resulting mesh, stress boundary layers are better resolved, and the value of De attainable will not decrease, hopefully, with the increasing mesh refinement.

The flow calculations were carried out using the mesh illustrated in Figure 5.12. In this mesh, the smallest elements (scaled with Re) are \( O(1\times10^{-3}) \), compared with the smallest elements considered by Sun et. al. [174] \( O(1\times10^{-1}) \) or Smith [167] \( O(1\times10^{-2}) \). In all cases reported here, \( H_c = 2 \), \( L_c = 18 \), \( R_c = 1 \), \( \langle v \rangle = 0.00319 \) and \( h_{chr} = 0.00319 \) and the dimensionless form presented in Eqs. (2.51)-(2.53) is utilized. The fourth-order Runge-Kutta timestepping (RK4) was utilized with the BCALM-preconditioned GMRES for the implicit, Stokes-like solve and DG was utilized for the explicit stress updates. Krylov iterations were terminated when the absolute tolerance was below \( O(1\times10^{-3}) \).

The large mesh dimension used in prior simulations resulted in aphysical solutions for lower Deborah numbers with increasing mesh refinement. For example, consider the flow around a cylinder predicted by Joo [105]. The flow was considered around a cylinder using a relatively coarse mesh, with the finest elements of size \( O(5\times10^{-2}) \) near the cylinder surface. The trace of the configuration tensor, \( \langle QQ \rangle \), over the solution field obtained
Figure 5.12: The highly-resolved mesh used to resolve the stress gradients in the viscometric boundary layer. This mesh has 23,598 elements and 187,816 nodes. Discretization of viscoelastic flow with DEVSS/DG results in 751,110 total unknowns with 281,850 velocity, pressure or gradients unknowns and 469,260 stress unknowns. The detail shows the smallest elements, which are O(1E-3).
from this simulation is plotted for $De = 1.0$ in Figure 5.13. Examination of the flow field in Figures 5.14 and 5.15 reveals two regions where the trace of the configuration tensor, $\langle \mathbf{QQ} \rangle$, is negative and aphysical. By contrast, the case of $De=1.0$ was considered on the new mesh, and the trace of the configuration tensor is plotted around the cylinder in the flow field in Figure 5.16. Much finer detail is visible in the solution, and the configuration tensor is positive-definite throughout the flow field.

These regions, where the solution on the coarse mesh has gone aphysical, correspond to interesting sharp gradients in the stress field near the viscometric boundary layer. The configuration tensors $\langle \mathbf{QQ} \rangle_x$ and $\langle \mathbf{QQ} \rangle_y$ and the pressure for the flow field are shown in Figures 5.17-5.19 for $De = 0.5$ and $De = 1.0$. The development of the birefringent strand and sharp stress gradients near the cylinder surface are seen in these figures. Integration of transient state to a steady state also was attempted for $De = 1.1$ using the initial state of $De=1.0$. This simulation, however, did not reach a steady state. Instead, the stress field increased linearly in time near the rear stagnation point, a phenomena which is discussed later.

The first region of interest which we consider is the boundary layer around the cylinder, away from the front and rear stagnation points. While the development of this region is apparent in the contour plots (Figures 5.17-5.19), the features are better illustrated by showing the stress and velocities as a function of distance from the cylinder surface. This is examined in Figures 5.20 and 5.21, where the velocity, velocity gradients, and stresses are shown as a function of the distance from the cylinder surface between the top of the cylinder and the channel wall. Here, $\zeta = R_C - y$ is the distance from the cylinder. The velocity profile near the cylinder wall is nearly Newtonian, as seen in Figure 5.20, with a linear velocity profile and a constant gradient. The stress field, however, is significantly effected by the fluid elasticity at this value of $De$, which may be seen in the field plots in
Figure 5.13: Contours of trace $\langle Q Q \rangle$ around the cylinder for flow around a linear array of cylinders ($L=18, H=2, R=1$). Results are for Oldroyd-B at $De=1.0$ on a coarse mesh used by Joo [105]. At regions in front of and in the wake of the cylinder, the configuration tensor has lost positive definiteness.
Figure 5.14: Detail of contours of trace $\langle QQ \rangle$ in wake of the cylinder for flow around a linear array of cylinders ($L=18$, $H=2$, $R=1$). Results are for Oldroyd-B at $De=1.0$ on a coarse mesh used by Joo [105]. The negative regions in the wake correspond to regions where $\langle QQ \rangle_{xx}$ and $\langle QQ \rangle_{yy}$ become negative.
Figure 5.15: Detail of contours of trace $\langle QQ \rangle$ in front of the cylinder for flow around a linear array of cylinders ($L=18$, $H=2$, $R=1$). Results are for Oldroyd-B at $De=1.0$ on a coarse mesh used by Joo [105]. The negative regions in the wake correspond to regions where $\langle QQ \rangle_{yy}$ becomes negative.
Figure 5.16: Contours of trace $\langle QQ \rangle$ around the cylinder for flow around a linear array of cylinders ($L=18$, $H=2$, $R=1$). Results are for Oldroyd-B at $De=1.0$ on the present mesh. The configuration tensor is positive definite throughout the flow field due to mesh refinement.
Figure 5.17: Contours of \(<\Omega^2>\) around the cylinder for flow around a linear array of cylinders \(L=18, \theta=2, R=1\). Results are for Oldroyd-B at \(D_e=0.5\) and \(D_e=1.0\) on the present mesh. The development of both the bi-refringent strand in the wake of and the large stress gradients in front of the cylinder may be seen with increasing Deborah number.
Figure 5.18: Contours of \( \langle QQ \rangle_{yy} \) around the cylinder for flow around a linear array of cylinders \((L=18, H=2, R=1)\). Results are for Oldroyd-B at De=0.5 and De=1.0 on the present mesh. The development of both the birefringent strand in the wake of and the large stress gradients in front of the cylinder may be seen with increasing Deborah number.
Figure 5.19: Contours of pressure around the cylinder for flow around a linear array of cylinders ($L=18$, $H=2$, $R=1$). Results are for Oldroyd-B at $De=0.5$ and $De=1.0$ on the present mesh.
Figure 5.20: The asymptotic behavior of the velocity and velocity gradient in the space between the cylinder and the top of the channel. The flow is near-Newtonian, with linear velocity profile near the cylinder and constant velocity gradient.
Figure 5.21: The asymptotic behavior of the stress in the space between the cylinder and the top of the channel. The development of a region of increased stress due to the convection of stretched polymers from the front of the cylinder is seen to develop with increasing Deborah number.
Figures 5.17 and 5.18. As De is increased from 0.5 to 1.0, a region just outside of the viscometric boundary layer develops where stretched polymers are convected from the stagnation point at the front of the cylinder to the top of the cylinder. This second structure increases the width of the boundary layer. This effect is illustrated in Figure 5.22, where the region in which stresses have reached 98% of their maximum value are shown. Going from $De = 0.5$ to $De = 1.0$ results in a decrease in the width of the boundary layer. However, in calculations at $De = 1.1$, the secondary effects of convection from the front of the cylinder cause the boundary layer thickness to grow larger. This is contrary to the asymptotic analysis of Renardy in [151] which suggests that the width of this boundary layer is $O(De^{-1})$. Renardy’s analysis, [151], considered only shearing near the cylinder surface, and cannot predict the secondary effects caused by the convection around the cylinder into the boundary layer. In fact, in the limit $De >> 1$, our simulations suggest that the secondary effects would significantly distort the viscometric boundary layer.

The second region of interest is the structure of the wake. The effect of increasing De on the stresses and velocities along the centerline in the wake are shown in Figures 5.23 and 5.24. The birefringent strand moves close to the cylinder with increasing Deborah number, while the effect of this structure propagates further into the wake. Furthermore, changes to the width of the wake may be illustrated by taking a cut across the wake where it has reached its largest magnitude. This is shown in Figure 5.25. The thickness of the wake scales even more sharply than suggested by the asymptotic analysis of Renardy [151], who estimated that the width of this wake structure decreased as $O(De^{-2})$. In our calculations the wake was observed to scale with $O(De^{-1.2})$. However, Renardy’s analysis was predicated on the assumption that the velocity field is Newtonian, an assumption which may be clearly seen as incorrect in Figure 5.24.
Figure 5.22: Detail of the asymptotic behavior of the stress in the space between the cylinder and the top of the channel. The region shown is represents where the stresses have reached 98% of their maximum value. Due to secondary elongational effects, this region does not decrease monotonically.
Figure 5.23: Stress profiles along the centerline in the wake of the cylinder for flow around a linear array of cylinders ($L=18, H=2, R=1$). Results are for Oldroyd-B at various Deborah numbers on the present mesh. Stress gradients become increasingly steep and the birefringent strand propagates further into the wake for increasing Deborah number.
Figure 5.24: Velocity profiles along the centerline in the wake of the cylinder for flow around a linear array of cylinders ($L=18$, $H=2$, $R=1$). Results are for Oldroyd-B at various Deborah numbers on the present mesh. Stress gradients become increasingly steep and the birefringent strand propagates further into the wake for increasing Deborah number.
Figure 5.25: The normalized magnitude of the stresses across the wake of the cylinder at the location of the stress maximum. As the Deborah number increases, the gradients in the stresses become much sharper and the dimension of the wake becomes smaller. For a Deborah number of 1.1, the structure was not resolved.
The analysis of Renardy [151], while interesting, does not seem to be accurate for the case of flow of an Oldroyd-B fluid around a cylinder. Most significantly, the assumption of a Newtonian flow field is flawed. Furthermore, the boundary layer analysis in [151] does not capture the strong convective influence that the stresses near the cylinder have on the viscometric boundary layer.

Increasing $De$ beyond 1.0 results in the configuration tensor becoming negative and therefore a physical for this mesh. This is due to the configuration tensor becoming negative valued in the wake of the cylinder. There are two possible explanations for this behavior. Since the thickness of the stress boundary structure is predicted to scale as $O(De^{-2})$ [151] and has been observed to scale even more sharply as $O(De^{-4})$, resolution of this structure becomes increasingly expensive with increasing Deborah number. With the current mesh, the sharp gradient is only represented by a single element, suggesting that additional mesh refinement may be necessary. This region of negative wake was also observed by Dou and Phan-Thien [54], as illustrated in Figure 5.26. They suggested that resolution of this structure may be possible by further refining the mesh.

The computed drag on the cylinder should also be considered as a macroscopic metric to gauge the simulation’s performance. The drag force on the cylinder per unit length is calculated by integrating the total stress over the surface of the cylinder:

$$F_N = \int_{0}^{2\pi} (p \cos \theta + \tau_{r\theta} \sin \theta + \beta G_{r\theta} \sin \theta) r \, Rd\theta$$  \hspace{1cm} (5.25)

The drag force value calculated in the current work are compared in Table 5.1 to the simulations run by Liu [117], Sun [174] and Dou and Phan-Thien [54]. Good agreement is seen up to $De = 1.0$. The three terms in the drag calculation above may be understood as a form drag contribution, $\int p \cos \theta$, a skin drag contribution from the solvent, $\int \beta G_{r\theta} \sin \theta$ and a
Figure 5.26: Results obtained from Dou and Phan-Thien [54] for flow around a cylinder in a channel using DAVSS-Omega. For high Deborah number, the first normal stress difference, \( N_1 \), becomes negative due to the stresses becoming aphysical in the wake of the cylinder.
Table 5.1: Comparison of the dimensionless drag per unit length obtained by various authors for the Oldroyd-B fluid and the h/R=2.0 geometry. Note that the results from Dou et al. [54] represent an extrapolation to a mesh size of zero. The results are from Dou et al. [54], Sun et al. [174], and Liu et al. [117].

<table>
<thead>
<tr>
<th>De</th>
<th>Current</th>
<th>Dou et al.</th>
<th>Sun et al.</th>
<th>Liu et al.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>118.8</td>
<td>118.0</td>
<td>119.1</td>
<td>119.5</td>
</tr>
<tr>
<td>1.0</td>
<td>117.8</td>
<td>118.8</td>
<td>119.3</td>
<td>119.5</td>
</tr>
<tr>
<td>1.1</td>
<td>118.0</td>
<td>119.7</td>
<td>120.4</td>
<td>N/A</td>
</tr>
</tbody>
</table>

The skin drag contribution from the polymer, \( \int \tau_{\omega \theta} \sin \theta \). The contributions to the total drag for the current calculations are compared with the Newtonian flow values in Table 5.2. If there were no interaction between the polymeric and solvent stress fields near the cylinder surface, the solvent contribution to the skin drag would be expected to remain constant with increasing De. Instead, the solvent contribution decreases with increasing Deborah number.

The refinement of the mesh very close to the cylinder surface yielded resolution of fine structure in the stress which has not been previously been reported. This has allowed the flow field to be developed up to De = 1.0, which represents an increase in possible Deborah number with increasingly refined mesh beyond the results obtained by Smith [167].

Table 5.2: The form and skin contributions to the dimensionless drag per unit length for the Oldroyd-B fluid and the h/R=2.0 geometry. The polymer presence clearly influences the solvent behavior.

<table>
<thead>
<tr>
<th>De</th>
<th>Total</th>
<th>Form Drag</th>
<th>Skin Drag</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Total</td>
<td>Form Drag</td>
<td>Polymer</td>
</tr>
<tr>
<td>0.0</td>
<td>132.3</td>
<td>92.2</td>
<td>16.5</td>
</tr>
<tr>
<td>0.5</td>
<td>118.8</td>
<td>80.6</td>
<td>15.6</td>
</tr>
<tr>
<td>1.0</td>
<td>117.8</td>
<td>82.6</td>
<td>14.3</td>
</tr>
<tr>
<td>1.1</td>
<td>118.0</td>
<td>83.9</td>
<td>13.9</td>
</tr>
</tbody>
</table>
Still, using this mesh to compute a steady-state solution through time integration to $De = 1.1$ was not possible. Smith [167] attributed the numerical instability to either the convection of stresses near the rear stagnation point or in the boundary layer. However, in the current work, both of these features have been resolved. Phan-Thein [146] suggested a numerical instability for planar stagnation flow at $De = 0.5$, as discussed in Section 5.1.1, however the current calculations have passed this limit. Curvature may push this instability to a different $De$, as this $\varepsilon$-order effect was not considered.

The comparison of the base states calculated here with the base state calculated by Öztekin et al. [141] shows that there is a region where Öztekin's similarity solution is valid. The azimuthal dependence of the radial component of the velocity solution by Öztekin et al. [141], given in Eq. 5.2, is compared with the results computed here in Figures 5.27-5.29 for $De = 0.0$, $De = 0.5$ and $De = 0.75$. Likely due to the coarseness of his mesh, Smith [167] did not see these valid regions. The radial dependence of the radial component of Öztekin's solution, given in Eq. 5.3, is compared to the result computed here for $De = 0.0$ in Figures 5.30 and 5.31. As Figures 5.27-5.31 show, the region where Öztekin analysis is valid is limited to small $R$ and small $\theta$. This means that the predicted instability would be highly localized and possibly influenced by the surrounding mixed flow. This could have the effect of delaying the onset of the predicted instability to a higher $De$ than the 0.5 predicted in [141].

5.3.4 Method Efficiency

Having established that the parallel BCALM-preconditioned Krylov method allows the solution of problems with mesh refinement which were previously intractable, we now turn our attention to the method’s performance. While the program is MPI-based for maximum portability and the code discussed in the following has been utilized on multiple platforms, including the IBM SP2 and the Silicon Graphics Onyx2 Infinite Reality system,
Figure 5.27: Comparison of the azimuthal velocity dependency of Öztokin et al.'s [141] similarity solution to the current calculations for $L = 18$ and $De = 0.0$. The region where Öztokin’s solution is valid near the cylinder decreases in size with increasing $De$. 
Figure 5.28: Comparison of the azimuthal velocity dependency of Öztékin et al.'s [141] similarity solution to the current calculations for $L = 18$ and $De = 0.5$. The region where Öztékin's solution is valid near the cylinder decreases in size with increasing $De$. 
Figure 5.29: Comparison of the azimuthal velocity dependency of Öztækin et al.'s [141] similarity solution to the current calculations for $L = 18$ and $De = 0.75$. The region where Öztækin's solution is valid near the cylinder decreases in size with increasing $De$. 
Figure 5.30: Comparison of the radial velocity dependency of Öztekin et al.'s [141] similarity solution to the current calculations for $L = 18$ and $De = 0.0$. 
Figure 5.31: Comparison of the radial velocity dependency of Öztékin et al.'s [141] similarity solution to the current calculations for $L = 18$ and $De = 0.0$. 
these results correspond to timings and parallel efficiencies obtained from a cluster of 16
dual-processor 500 Mhz Pentium III machines interconnected with gigabit ethernet. These
machines are capable of approximately 50 MFLOP/s per processor in practice, compared
with approximately the approximately 80 MFLOP/s per processor performance of the
IBM SP2 installed here at MIT [52]. The use of gigabit ethernet approximately matches
the speed of the SP2 high-speed switch, 30,000 bps [1]. As a result, a relatively high-per-
formance machine has been assembled using commodity parts. While the computers are
dual processor machines, due to the poor performance of memory copy on the Intel X86
platform, one processor is dedicated to computation and the other is dedicated to commu-
nication for the purpose of the efficiency timings. This is consistent with the method
which is used to extract maximum performance from the world’s first teraflop computer,
Accelerated Strategic Computing Initiative - Red (ASCI-Red). This supercomputer,
installed at Sandia national laboratory, is used by the Department of Defense and the
Department of Energy to perform calculations for maintaining the nuclear readiness of the
U.S. arsenal without violating the test ban treaty. This machine consists of 7,264 dual pro-
cessor Pentium Pro machines. For the highest demonstrable efficiencies they, too, are uti-
lized in this manner [5].

For these studies, four different meshes were used, as illustrated in Figures 5.33-5.35.
For the DEVSS/DG discretization of an Oldroyd B fluid, mesh SM1 has 60,900 total
unknowns, with 37,910 of them stresses. Mesh SM2 has 113,464 unknowns with 70,690
of them stresses, mesh SM3 has 226,130 unknowns with 141,040 of them stresses and
mesh SM4 has 422,052 unknowns with 263,410 of them stresses.

Evaluation of the parallel method may be characterized in three major ways. First, the
parallel method is compared to the previously used serial, frontal method. Next, the
robustness of the method is evaluated. Finally the parallel efficiency of the method is
Figure 5.32: Mesh SM1. This mesh has 1,930 elements and 15,240 nodes. Discretization of viscoelastic flow with DEVSS/DG results in 60,900 total unknowns with 22,990 velocity, pressure or gradients unknowns and 37,910 stress unknowns.

Figure 5.33: Mesh SM2. This mesh has 3,582 elements and 42,774 nodes. Discretization of viscoelastic flow with DEVSS/DG results in 113,464 total unknowns with 42,774 velocity, pressure or gradients unknowns and 70,690 stress unknowns.
Figure 5.34: Mesh SM3. This mesh has 7,122 elements and 56,556 nodes. Discretization of viscoelastic flow with DEVSS/DG results in 226,130 total unknowns with 85,090 velocity, pressure or gradients unknowns and 141,040 stress unknowns.

Figure 5.35: Mesh SM4. This mesh has 13,273 elements and 105,532 nodes. Discretization of viscoelastic flow with DEVSS/DG results in 422,052 total unknowns with 158,642 velocity, pressure or gradients unknowns and 263,410 stress unknowns.
explored. For these runs, a Deborah number of 0.5 was used along with $H_C = 1$ and $L_C = 18$. Krylov iterations of BiCGStab for used for the Stokes solve and were considered converged if either a relative tolerance of $1 \times 10^{-12}$ or an absolute tolerance of $1 \times 10^{-3}$ was achieved. One domain per processor was used to define both the additive Schwarz decomposition and the coarse grid. Consideration of the method under these criteria illustrates the effectiveness of the parallel scheme put forth in this thesis.

A comparison of the parallel time-stepping scheme to the previously utilized frontal scheme was first performed to both establish the method's accuracy and to show that the method outperforms current technology in a serial sense. The fastest algorithm in the world is not very useful if it cannot obtain the correct solution. Similarly, if a parallel method is many times slower than a serial method then it may not be very useful.

The steady-state solution obtained from the parallel method for the four meshes agrees with the solutions obtained from the frontal method. Evolution towards this state can be seen in a comparison of the 2-norm, defined as

$$
\|x\|_2 = [|x_1|^2 + |x_2|^2 + \ldots + |x_n|^2]^{1/2},
$$

(5.26)

of the deviation of the velocities vectors from that obtained from the Newtonian case. This quantity was obtained from both methods at each timestep for the first 100 timesteps and is illustrated for both the velocity Figure 5.36. From this plot it can be seen that the velocity solution obtained are is in agreement with that obtained from the frontal method. Because the correct velocities, pressures and gradients are obtained from the implicit solution of the Stokes-like problem, the correct stresses should also be obtained. The 2-norm of the stresses obtained from both the frontal method and the parallel method are shown in Figure 5.37. For both Figures, the solution obtained from the iterative method is indistinguishable from that obtained from the frontal method.
Figure 5.36: A comparison of the 2-norm of the deviation of the velocity solution from the parabolic solution that would result in an unobstructed channel. The solution obtained from the serial, frontal code previously used in the group is compared to the solution obtained from the parallel iterative method for the case of a Deborah number of 0.5 and $L_C$ of 18.
Figure 5.37: The solution obtained from the serial, frontal code previously used in the group is compared to the solution obtained from the parallel iterative method for the case of a Deborah number of 0.5 and $L_C$ of 18.
<table>
<thead>
<tr>
<th></th>
<th>Time required for 100 timesteps (Hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Frontal Method</td>
</tr>
<tr>
<td>Mesh SM1</td>
<td>0.7</td>
</tr>
<tr>
<td>Mesh SM2</td>
<td>1.8</td>
</tr>
<tr>
<td>Mesh SM3</td>
<td>27.1</td>
</tr>
<tr>
<td>Mesh SM4</td>
<td>N/T</td>
</tr>
</tbody>
</table>

Table 5.3: The time required for 100 timesteps for each of the meshes. The serial, frontal method compared to an eight-processor implementation of the parallel iterative method. For small problems, the methods are comparable, but as the problem size and the bandwidth increases, the parallel iterative version becomes much faster than the frontal method. The solution of SM4 was not possible with the frontal method.

Knowing that the correct solution is obtained, we now turn our attention to the speed of the parallel method relative to the frontal method. The time required for 100 timesteps was determined for both an eight-processor parallel run and for the sequential frontal method. These values are tabulated in Table 5.3. For the smaller meshes, the two methods are comparable. With eight times more processing power, the parallel method is approximately seven times faster than the frontal method. However for the larger meshes, meshes three and four, the parallel iterative method is much faster than the comparable frontal method. This is due to the increasingly large bandwidth that the frontal method must use, which makes the largest mesh intractable using conventional methods. The new parallel method is both an accurate and fast solution method relative to current technology.

The method also is robust, being insensitive to both model parameters and solver parameters. The key model parameter which we wish to establish insensitivity to is the Deborah number. This parameter controls the overall stiffness of the resulting system. The
total number of iterations required for the Stokes’ problem solve during the first timestep for each of the meshes is shown as a function of the Deborah number in Figure 5.38 for the parallel scheme running on 16 processors. As can be seen, the number of iterations is very insensitive to Deborah number over a range of Deborah numbers much larger than would be stable for the explicit timestepping scheme. This is sensible for two reasons. First, application of the BCALM preconditioner successfully improves the condition of the linear system. Furthermore, the greatest influence of the Deborah number is in the stress field. The impact on the linear system is therefore through the right-hand side of the Stokes linear system, helping maintain the iterative method’s insensitivity to changes in De. Increases to the De are implemented by increasing the relaxation time for the fluid and, therefore, the characteristic time for the fluid relative to the characteristic time for the flow. While increasing De increases the effect of the stress field on the velocity field, because the timestep is nondimensionalized with the characteristic time of the flow, the velocity field actually changes less per timestep with increasing De.

The numerical method also is insensitive to changes in the key parameters in the preconditioned solution method. The main parameters introduced by this scheme are the number of processors in the system and the drop tolerance for the inexact solvers. First, the coarse grid performance with the addition of processors is evaluated by examining the number of Krylov iterations required as a function of the number of subdomains in the linear system. This is done both for the case of a stationary initial condition of zero in Figure 5.39 and for taking the previous timestep as the initial condition and refining it in Figure 5.40. BiCGStab is selected as the Krylov method for these runs. As can be seen in Figure 5.40, for the largest meshes, the number of iterations required actually decreases with the addition of processors. This is shown clearly in Figure 5.41 for the case of refinement of the previous timestep solution to the current timestep with the largest mesh, SM4. This is
Figure 5.38: The total number of iterations required for BiCGStab to converge the Stokes-like subproblem in RK4/DG during the first timestep. The iteration count is stable for a range of Deborah numbers much larger than the RK4/DG time-stepping scheme.
Figure 5.39: The number of iterations required for BiCGStab to converge the Stokes-like subproblem in RK4/DG from an initial guess of zero during the first timestep. The iteration count is stable and for the largest meshes the addition of processors actually improves convergence.
Figure 5.40: The number of iterations required for BiCGStab to converge the Stokes-like subproblem in RK4/DG using the previous timestep as an initial guess during the second timestep. The iteration count is stable and for the largest meshes the addition of processors actually improves convergence.
Figure 5.41: Detail: The number of iterations required for BiCGStab to converge the Stokes-like subproblem in RK4/DG using the previous timestep as an initial guess during the second timestep for the largest mesh considered in this Section. The iteration count clearly improves convergence in this case.
consistent with both the results observed by Knoll, et. al. [112], who computed a similar coarse grid scheme through 32 sub-blocks in a serial code, and by Jenssen and Weinerfelt [101], who computed with a similar coarse grid preconditioning scheme with up to 128 processors.

While less important because it is selectable, it is also desirable to show that the imbedded iterative method is not sensitive to the drop tolerance selected for the inexact solver in the preconditioner. The total number of iterations required for unrestarted GMRES to converge each mesh during the first timestep is shown in Figure 5.42. As is seen, there is a minimum drop tolerance which can be used for a given system which determines the point where a large amount of the factorization is lost. This trend is illustrated by considering the number of nonzeros in the factorization, shown in Figure 5.43. Taking the drop tolerance too large does not improve the solution performance, but only represents a minor memory inefficiency for the parallel method. Still, for the meshes considered, a drop tolerance of \( O(10^{-4}) \) saves approximately 20% of the maximum required memory and has a minimal effect on iteration count. Due to the small dimension of the coarse grid problem, this system is solved exactly.

One common way of examining the efficiency of a parallel method is by calculating the speedup for a particular problem size as more processors are used to solve the problem. This metric represents how much faster a problem which can be solved on one processor will execute on multiple processors. Typically, this value is calculated as \( S \equiv t_1/t_N \) where \( t_1 \) is the time required for the method on one processor and \( t_N \) is the time required on \( N \) processors. In each case presented, the domain is dissected into a number of domains equal to the number of processors, using Chaco to perform nested spectral sectioning. Several problem sizes were run at \( De = 0.5 \) with a lumped coarse grid and the speedup calculated and reported for the solution of the Stokes-like subproblem using preconditioned
Figure 5.42: The total number of iterations required for GMRES to converge the Stokes-like subproblem during the first timestep for a Deborah number of 0.5 and for a 16 processor run as a function of drop tolerance. Beyond a critical drop tolerance, the number of iterations required stabilizes.
Figure 5.43: The number of nonzeros in each sub-block ILU factorization of the viscous operator of the Stokes-like subproblem for a Deborah number of 0.5 and for a 16 processor run. Beyond the same critical drop tolerance where the iteration count stabilizes, the number of nonzeros in the factorization is also flat. For mesh SM1, the “A” subblock corresponding to the viscous operator has 1100901 nonzeros. For SM2, it has 2077361, for SM3 it has 4184115, and for SM4 it has 7858793.
GMRES and a drop tolerance of 1E-6 for the first, factorization step and subsequent refinement steps in Figures 5.44-5.45. The efficiencies for the discontinuous Galerkin updates are reported as a function of the number of processors in Figure 5.46, and performance is seen to be near ideal. For the factorization time, some superlinear speedups are observed, due to the reduction of the problem dimension on each processor with the increasing number of processors. This gives the speedup metric an "unfair" characteristic, where there is a slight reduction in execution steps that give the parallel code an "unfair" advantage over the single processor case [163]. However this is not the case for either the subsequent iterations or the discontinuous Galerkin updates. In the case of subsequent iterations, the work per processor is diminishing relative to the communication required per processor, the speedup of the method is expected to ultimately decrease with increasing numbers of processors. This is consistent with Ahmdal's law [4]. Furthermore, these parallel efficiencies are comparable to those obtained in Section 4.3 for the case of natural convection in a fluid filled cavity.

The use of both processors on each of the dual processor machines in the cluster also was considered. As observed by other researchers, this is less efficient due to inefficient memory usage on Intel x86 platforms and contention for control of the network card between the processors [5][178]. A comparison of the results obtained above to those obtained by using 16 processors internal to 8 machines was performed to examine the method's efficiency and is illustrated in Figures 5.47-5.49. For small problems, the method's performance is much more abysmal, with the addition of processors actually hurting the speedup. However, for the largest problems, the performance on 32 processors is better than with 16 while the efficiency is lower. In practice, this is the only decision that would be made. Should a researcher use 16 processors in 16 machines or all 32? For
Figure 5.44: Speedup for the initial factorization of the Stokes-like linear system at a Deborah number of 0.5 using a drop tolerance for each sub-problem of 1E-6. Superlinear speedups are possible at this step due to the scaling of the subproblem factorization dimension.
Figure 5.45: Speedup for the subsequent solution of the Stokes-like linear system at a Deborah number of 0.5 using the factorization obtained with drop tolerance for each sub-problem of 1E-6. As there is no "unfair" advantage, the method's efficiency for a fixed problem size will decrease with the addition of processors.
Figure 5.46: Speedup for the explicit discontinuous Galerkin update at a Deborah number of 0.5. As the DG update may be accomplished in parallel, the deterioration with the addition of processors is minimal.
Figure 5.47: Speedup through 32 processors for the initial factorization of the Stokes-like linear system at $De = 0.5$ using a drop tolerance for each sub-problem of $1E-6$. Because very little communication is required for this factorization, the efficiencies remain high.
Figure 5.48: Speedup through 32 processors for the subsequent solution of the Stokes-like linear system at De = 0.5 using the factorization obtained with drop tolerance for each subproblem of 1E-6. As communication is required in this step, for small problems running multiple processors per box rapidly degrades performance.
Figure 5.49: Speedup on up to 32 processors for the explicit discontinuous Galerkin update at $D_e = 0.5$ using the factorization obtained with drop tolerance for each sub-problem of $1E-6$. As the DG update may be accomplished in parallel, the deterioration with the addition of processors is minimal.
large problems, the results in Figures 5.47-5.49 suggest that all of the processors should be utilized for fastest performance while the efficiency will be lower.
5.4 The parallelization of linear stability analyses

While the portability of the block-complement and additive levels method preconditioner for Krylov iterative methods has already been evidenced by its ready implementation in the time dependent viscoelastic flow problem and for the steady-state natural convection problem in Section 4.3, it is interesting to further demonstrate its utility in application to the linear stability problem. In this context, a steady state two-dimensional solution is obtained, usually through the timestepping RK4/DG scheme described in 2.6.2, and then the stability of the solution to three-dimensional perturbations analyzed.

The motivation for studying the stability of the two dimensional base state is to attempt to reproduce the instabilities observed in the flow of PIB Boger fluid by McKinley et al. [125] and Byars [31] for an isolated cylinder and by Liu [116,117] for an array of cylinders. For both the isolated cylinder and the arrays of cylinders, a transition to a three-dimensional cellular structure in the cylinder wake is observed at a critical De.

A steady-state, two-dimensional base flow is obtained from the timestepping scheme at long time as

$$
(\nu''(x_1, x_2), p''(x_1, x_2), G''(x_1, x_2), \tau_p''(x_1, x_2))
$$

(5.27)

Linearization of the governing equations Eqs. (5.15)-(5.18) about Eq. (5.27) accomplished by substituting into these equations Eqs. (5.15)-(5.18) an expression for a small deviation $\varepsilon \ll 1$ from the steady state:

$$
\begin{bmatrix}
\nu(x_1, x_2, x_3, t) \\
P(x_1, x_2, x_3, t) \\
G(x_1, x_2, x_3, t) \\
\tau_p(x_1, x_2, x_3, t)
\end{bmatrix} =
\begin{bmatrix}
\nu''(x_1, x_2) \\
P''(x_1, x_2) \\
G''(x_1, x_2) \\
\tau_p''(x_1, x_2)
\end{bmatrix} + \varepsilon
\begin{bmatrix}
\rho(x_1, x_2, t) \\
\rho(x_1, x_2, t) \\
\rho(x_1, x_2, t) \\
\rho(x_1, x_2, t)
\end{bmatrix}
$$

(5.28)
Here \( (\vartheta, \varpi, \hat{G}, \hat{\tau}_p) \) are the perturbed quantities. When these expressions are substituted into (5.15)-(5.18), expressions for the perturbed quantities are recovered in terms of the steady state as

\[
\tau_p + D e (\frac{\partial \varpi_p}{\partial t} + \nu'' \cdot \nabla \hat{\tau}_p + (-\{ \varpi'' \cdot \hat{G} \}) - \{ \varpi'' \cdot \hat{G} \}^T + \nabla \cdot \nabla \hat{\tau}_p - \{ \varpi'' \cdot G'' \} - \{ \varpi'' \cdot \hat{G}'' \}^T) = -(1 - \beta)(\hat{G} + \hat{G}^T)
\]

\[
-\nabla^2 \hat{\varpi} + \hat{\tau}_p + (1 - \beta) \nabla \cdot (\hat{G} + \hat{G}^T) + \nabla \varpi = 0
\]

\[
\hat{G} = \nabla \vartheta
\]

\[
\nabla \cdot \hat{\varpi} = 0
\]

By expressing the perturbed quantities as in Eq. (5.28) the disturbance of a steady state flow to perturbations is recovered. It is also possible to examine the response of the base state to perturbations by writing the form of the perturbation in the neutral direction as a Fourier component with wavenumber \( k \). The functions for the perturbation are complex-valued functions written as

\[
\begin{bmatrix}
\vartheta(x_1, x_2, x_3, t) \\
\varpi(x_1, x_2, x_3, t) \\
G(x_1, x_2, x_3, t) \\
\tau_p(x_1, x_2, x_3, t)
\end{bmatrix}
= \begin{bmatrix}
\vartheta''(x_1, x_2) \\
\varpi''(x_1, x_2) \\
G''(x_1, x_2) \\
\tau_p''(x_1, x_2)
\end{bmatrix}
+ \varepsilon Re \begin{bmatrix}
\vartheta(x_1, x_2, t) \\
\varpi(x_1, x_2, t) \\
G(x_1, x_2, t) \\
\tau_p(x_1, x_2, t)
\end{bmatrix}
\]

\[
e^{-ikx_1}
\]

The variables \( (\vartheta, \varpi, \hat{G}, \hat{\tau}_p) \) scale the in-phase and out-of-phase components of the perturbation in the neutral direction and additional unknowns are introduced as

276
\[
\begin{bmatrix}
\hat{\nu}_{x, \sin} \\
\hat{\nu}_{x, \cos} \\
\hat{\nu}_{y, \sin} \\
\hat{\nu}_{y, \cos} \\
\hat{\nu}_{z, \sin} \\
\hat{\nu}_{z, \cos}
\end{bmatrix}
\begin{bmatrix}
\hat{G}_{x, \sin} \\
\hat{G}_{x, \cos} \\
\hat{G}_{y, \sin} \\
\hat{G}_{y, \cos} \\
\hat{G}_{z, \sin} \\
\hat{G}_{z, \cos}
\end{bmatrix}
\hat{p} = 
\begin{bmatrix}
p_{\sin} \\
p_{\cos}
\end{bmatrix}
\hat{\mathbf{t}} = 
\begin{bmatrix}
\hat{t}_{x, \sin} \\
\hat{t}_{x, \cos} \\
\hat{t}_{y, \sin} \\
\cdots \\
\hat{t}_{z, \sin} \\
\hat{t}_{z, \cos}
\end{bmatrix}
\tag{5.34}
\]

The resulting linearized equations are discretized using the Galerkin finite elements for the conservation equations along with either discontinuous Galerkin elements for the constitutive equation or SUPG.

The parallelization of two linear stability codes is considered. First, a two-dimensional linear stability code based on RK4 timestepping and a discontinuous Galerkin formulation for the stress. Then a three dimensional linear stability code formulated using theta-method timestepping and SUPG is presented. In both cases, a BCALM preconditioned Krylov method is used for the Stokes'-like updates for the conservative variables, but the methods differ in parallelization of the stress updates.

\subsection*{5.4.1 A Parallel, Two-Dimensional Linear Stability Code Using Fourth-order Runge Kutta Timestepping and Discontinuous Galerkin.}

A two-dimensional parallel linear stability code based on a discontinuous Galerkin formulation for the stresses and fourth order Runge-Kutta timestepping is readily implemented from the transient viscoelastic code presented in Section 2.6.2. The same number of unknowns are present, the only difference being that the quantities represent the perturbed quantities instead of the actual field variables.

Rewriting the system recovered from the discretization of the perturbed equations Eqs. (5.29)–(5.32) as a set of ODEs:
\[ M \cdot \frac{d\hat{\tau}_p}{dt} = \frac{1}{De} f(\hat{\tau}_p, \varphi, \hat{G}) \]  
(5.35)

\[ g(\hat{\tau}_p, \varphi, \rho, \hat{G}) = 0 \]  
(5.36)

In Eq. (5.35), \( f \) is the discretization of the constitutive equation, and \( g \) is a Stokes-like problem for the kinematic variables resulting from the discretization of Eqs. (5.30)-(5.32). Equation (5.36) is invertible for the kinematic unknowns, so this may be written:

\[
\begin{bmatrix}
\varphi \\
\rho \\
\hat{G}
\end{bmatrix} = g(\hat{\tau}_p) 
\]  
(5.37)

Equation (5.37) is then substituted into Eq. (5.35). This allows us to write an function in terms of the stress alone, as is necessary for the application of Runge-Kutta. We define a new function, \( \hat{f} \):

\[
\hat{f}(\hat{\tau}_p) \equiv M^{-1} f(\hat{\tau}_p, g(\hat{\tau}_p)) 
\]  
(5.38)

At this point we have recovered exactly the same problem structure as was seen in the transient viscoelastic code presented in Section 2.6.2. As a result, we expect that, for a similarly sized problem, the parallel efficiencies should be similar.

This is borne out by the speedups observed for this method. The linear stability of the two dimensional meshes presented in Section 5.3.4 was considered at a Deborah number of 0.5, and the speedups obtained from the initial factorization, subsequent linear solve and discontinuous Galerkin updates, shown in Figures 5.50-5.52. These speedups, from the linear stability code, are compared to those obtained in Section 5.3.4 for the full transient code. Again, speedup is calculated as \( S \equiv t_1/t_N \) where \( t_1 \) is the time required for the method on one processor and \( t_N \) is the time required on \( N \) processors. As in Section 5.3.4 for the time dependent code, the domain is dissected into a number of domains equal to the number of processors, using Chaco to perform nested spectral sectioning along with a
Figure 5.50: Speedup for the initial factorization of the Stokes-like linear system for both the transient code and linear stability code based on the RK4/DG timestepping method. Runs were performed at De = 0.5 using a drop tolerance for each sub-problem of 1E-6. In both cases, superlinear speedups are possible at this step due to the scaling of the subproblem factorization dimension. The two cases for each mesh are virtually indistinguishable due to the similarity of the implementation.
Figure 5.51: Speedup for the subsequent solution of the Stokes-like linear system for both the transient code and linear stability code based on the RK4/DG timestepping method. Runs were performed at De = 0.5 using the factorization obtained with a drop tolerance for each sub-problem of 1E-6. Because the two problems are so closely related, it is impossible to distinguish between the transient runs and the linear stability runs.
Figure 5.52: Speedup for the explicit discontinuous Galerkin update of the stress field for both the transient code and linear stability code based on the RK4/DG timestepping method. Runs were performed at $De = 0.5$ using the factorization obtained with drop tolerance for each sub-problem of $1E-6$. As the DG update may be accomplished in parallel, the deterioration with the addition of processors is minimal for both types of runs. Since this step has high efficiency for all the number of processors and unknowns considered, the lines are very close to each other. A slight improvement is seen with increasing numbers of unknown, the speedup from 14.2 to 15.9.
lumped coarse grid. The speedup for these calculations are virtually identical to those
obtained for the parallel transient code. Therefore, like the nonlinear parallel two dimen-
sional transient code, the linear stability code is highly parallel.

5.4.2 A Parallel Three Dimensional Linear Stability Code Using Theta Method
Timestepping and SUPG.

A three-dimensional parallel linear stability code based on a SUPG discretization for
the stresses and theta timestepping requires a slight modification of the transient viscoelas-
tic code is presented in Section 5.3. A larger number of unknown types are present, and
the stress field cannot be solved for explicitly in a discontinuous fashion. As presented in
Sections 2.6.1, the timestepping may be understood as dividing the timestep into three
segments:

\[
M \cdot \frac{u^{n+\theta} - u^n}{\theta \Delta t} = A_1(u^{n+\theta}) + A_2(u^n)
\]

\[
M \cdot \frac{u^{n+\theta} - u^n}{(1 - 2\theta)\Delta t} = A_1(u^{n+\theta}) + A_2(u^{n+1-\theta})
\]

\[
M \cdot \frac{u^{n+1} - u^{n+\theta}}{\theta \Delta t} = A_1(u^{n+1}) + A_2(u^{n+1-\theta})
\]

(5.39)

The operator splitting presented in this Section is used, the only difference being that the
quantities represent the perturbed quantities instead of the actual field variables:

\[
A_1 \equiv \begin{bmatrix}
-\omega \tau_p \\
-\nabla^2 \hat{\nu} + \nabla \cdot \tau_p + \nabla p + (1 - \beta) \nabla \cdot (\hat{G} + \hat{G}^T) \\
\nabla \cdot \phi \\
\hat{G} - \nabla \phi
\end{bmatrix}
\]

(5.40)

\[
A_2 \equiv \begin{bmatrix}
-(1 - \omega) \tau_p De(\hat{\nu} \cdot \nabla \tau_p - \{ \tau_p \cdot \hat{G} \} - \tau_p \cdot \hat{G}^T - (1 - \beta)(\hat{G} + \hat{G}^T)
\end{bmatrix}
\]
For the Stokes-like subproblem, the iterative method BCALM-preconditioned BiCGStab method was used. This implicit update of the conservative variables is treated in exactly the same manner as the Stokes-like solve encountered in the fourth-order Runge-Kutta scheme for the nonlinear equations, and these speedup-based efficiencies are plotted along with the times required for factorization and subsequent iterative update of the stokes problem in Figure 5.53. As can be seen in this figure, the Stokes solve is highly parallel.

Since the implicit update for the stresses does not decouple, this problem was solved with using preconditioned BiCGStab, also. Because the stress problem is not block singular, two-level additive Schwarz was utilized without the block complement operation.

The linear stability of flow around closely spaced cylinders was considered by Smith [167]. The purpose of this work was to reproduce the instability observed by Liu [116] for a closely spaced linear array of cylinders. Smith’s simulations were successful in capturing this instability, however the largest cylinder spacing considered in [167] was \( L = 3.25 \). This limitation was due to the solver technology - using a larger spacing requires a larger number of unknowns and the \( L = 3.25 \) case, with \( O(1 \times 10^5) \) unknowns, was the largest possible.

An example mesh for this flow is shown in Figure 5.54. Flow was considered for this mesh for \( De = 1.7 \). For this mesh, the time required for 300 timesteps was compared between the frontal method code and the parallel scheme. The frontal method required 8.2 hours whereas the parallel iterative method running on eight processors required 1.3 hours. The parallel iterative method again is approximately the same speed for this modestly sized problem as the frontal method.

The stress problem is much smaller than the stokes update, consisting of 30,780 unknowns in this problem compared to 111,354 unknowns in the Stokes-like problem.
Figure 5.53: The speedups observed for the theta-method SUPG linear stability code. Again, high parallel efficiency is observed for BCALM-preconditioned BiCGStab, used for the Stokes solve. A lower efficiency is observed for the stress solve due to its small size.
Figure 5.54: The mesh used to study the flow of an Oldroyd-B fluid around a closely-spaced array of cylinders. This mesh has 2470 elements and 10,009 nodes. Discretization of viscoelastic flow with DEVSS/SUPG results in 142,134 total unknowns with 111,354 velocity, pressure or gradient unknowns and 30,780 stress unknowns.
Furthermore, the sine and cosine components may be solved using the same linear system, further reducing the dimension of the subproblem. As a result, for the case of an 8 processor run under the same conditions used above, the time spent in the stress solve is $\frac{1}{44}$ the time spent in the subsequent Stokes-like problem. The parallel efficiency of this subproblem solve is also illustrated in Figure 5.53. Due to the small size of this subproblem, the efficiencies are smaller than observed for the Stokes problem. Because this calculation is such a small part of the overall algorithm, the low efficiency is tolerated.

With this efficient method, larger cylinder spacing may be considered. The critical De for the additional cylinder spacing of $L = 3.5$ is computed, and this new result is shown along with those from [167] in Figure 5.55.
Figure 5.55: The effect of cylinder spacing on critical De for flow of an Oldroyd-B fluid past a closely spaced array of cylinders. The result for L=3.5 was not obtainable by Smith [167] due to the number of unknowns that arise. Data for L < 3.5 taken from [167].
5.5 Conclusions

The results presented in this Chapter document the excellent performance of BCALM-pre-conditioned Krylov methods, explained in detail in Chapter 4, for the complex two-dimensional problem of transient viscoelastic flow of an Oldroyd-B fluid around an array of cylinders in a channel. This problem is extremely difficult, numerically, due to the steep stress gradients in the boundary layer near the cylinder.

We have calculated the flow of an Oldroyd-B fluid, discretized using the DEVSS/DG finite element method, to steady-state using a fourth-order Runge-Kutta timestepping method for $L = 18$. By using a mesh with a minimum spacing smaller than $O(1 \times 10^{-3})$, previously unresolved features in the cylinder boundary layer and wake are visible in the current results. The result of resolving these features is that the maximum $De$ attainable, $De = 1.0$, is higher than that reported by Smith [167], whose mesh was an order-of-magnitude more coarse. Increasing the mesh resolution has enabled the simulation of higher Deborah number flows. The solution of the resulting problem, with $O(1 \times 10^4)$ unknowns, would have been intractable with prior solution technologies.

Beyond $De = 1.0$, however, the transient simulation fails to reach a steady-state flow. Most likely, this instability is numerical, due to the current mesh’s inability to resolve the steep gradients in the wake of the cylinder. The width of this structure appears to scale as $O(De^{-1})$, which is more severe than that predicted by Renardy in [151]. However, the current simulation results appear to match the similarity solution predicted by Öztekin et al. [141] in the wake of the cylinder. The valid region of Öztekin’s solution was missed by Smith [167] in his comparison, most likely due to the coarseness of his mesh. Öztekin predicts an instability in this region, however its existence will be affected by the mixed flow observed in this region.
The high performance of the parallel fourth-order Runge-Kutta timestepping code was also examined. The parallel BCALM-preconditioned Krylov method put forth in Chapter 4 is coupled with a parallel discontinuous Galerkin update. The resulting algorithm is faster than the serial, frontal algorithm used in previous viscoelastic simulations, even before exploiting the advantages of parallel computing. This work demonstrates the great flexibility of the current solver and its utility in arbitrary geometries. This is in contrast with a previous parallel attempt, CFS by Mehrabi [126], which required regular geometries for its complex scheduling scheme. Furthermore, the algorithm is shown to be very robust, insensitive to De, the drop-tolerance, and the number of processors utilized.

Very high parallel efficiencies are documented for the method. Both the application of the BCALM-preconditioned Krylov method and the discontinuous Galerkin updates have parallel efficiencies over 95% through sixteen processors, even on moderately sized problems. Furthermore, the algorithm is very flexible, as demonstrated by the incorporation of BCALM-preconditioned Krylov techniques into two linear stability analyses. The first linear stability analysis code uses fourth-order Runge-Kutta timestepping and is therefore a ready extension of the transient simulation code. Again, high parallel efficiencies were observed, and the resulting code enables the solution of previously intractable problems. The second linear stability code utilizes a theta-method timestepping with a SUPG method for the stresses. This code is slightly different in structure, using an application of the BCALM-preconditioned Krylov method to the resulting Stokes' like problem, but requiring the solution of a second system related to the coupled stresses. This problem has allowed the calculation of a transition in the flow of an Oldroyd-B fluid through a closely-spaced linear periodic array of cylinders from a two-dimensional steady-state to a three-dimensional steady state for L = 3.5. These spacings are larger than were possible for
Smith in [167] due to the limitations of his frontal technology. This linear stability code, also, has high parallel efficiency.
Chapter 6

Discussion
The purpose of the research presented in this thesis is to contribute to the eventual understanding of polymer processing through the development of an efficient numerical solver for solving the complex viscoelastic flows that are important to these processes. From a general, process design standpoint, the modelling of a polymer operation can be thought of as a chain of events. First, in the description of the physical process using a graphical user interface of some type to characterize the geometry of the flow configuration. This process geometry is used to define a mesh over which the equations of motion are discretized. This is combined with unique polymer physics which influence these equations’ form through the constitutive equation for the stresses. The result of this discretization is a large system of equations, which must be solved repeatedly to obtain a solution, either by Newton’s method or by time integration. The output is then visualized and interpreted to assist in process design decisions.

Each of these steps has challenges. The question of how to design an interface which allows useful interaction between the engineer and the model is certainly an open question. The input to these research codes currently is in the form of text files, and it would be more useful to have an interactive, graphical interface which could be used to generate complex meshes with proper refinement quickly. Next, the question of how to capture the unique properties of viscoelastic material in the constitutive equation is an open question for research. Early work attempted to use parametric models to capture the shear thinning and normal stress phenomena observed in viscoelastic flow, but these models were unsatisfactory. Ongoing research attempts to explain these behaviors from first principles, resulting in increasingly complex models. The discretization of these governing equations
results in very large linear systems which must be inverted for solution. To date this step has been accomplished with twenty-five year old frontal LU decomposition due to the inability of iterative methods to robustly handle these stiff linear systems. Once a solution has been obtained, it must be somehow usefully visualized. Currently, this is by writing FORTRAN codes that extract the information of interest -- such as drag forces, pressure profiles, and stress contours -- and visualizing the output with utilities such as Tecplot [2].

While these are all interesting research questions, the ones of greatest impact to the field of viscoelastic flow research are contributions the development of new constitutive models. Yet, performing calculations with the most sophisticated constitutive models is impossible without a solver which can efficiently, robustly, and quickly solve the resulting system of equations. It is the latter issue that this thesis addresses.

Our analysis in Section 1.4 estimates the size of problems that researchers would like to solve in three dimensions as $O(1 \times 10^7)$ unknowns -- intractable with conventional serial algorithms and clearly motivating the need for parallel computing. Parallel direct methods, such as Concurrent Factorization and Storage by Mehrabi [126,129], possess several key limitations, as discussed in Section 1.6 and Section 2.2. While robust, these methods have complexity which grows with both the number of unknowns in the system and the number of processors utilized. Furthermore, some basic operations encountered in these methods, such as backwards substitution, do not parallelize efficiently. This motivates the study of parallel iterative methods in this thesis.

Straightforward application of iterative methods to the discretized equations governing viscoelastic flow has not been successful [35]. For this reason, we consider methods for splitting the governing equations by type. This is motivated in detail in Sections 1.1-1.3. Time splitting methods, presented in Section 2.6 and developed in [167,176], exploit operator splitting and are the basis considered for the viscoelastic flow calculations performed
in Chapter 5. The work in this thesis, therefore, fell into two categories -- we develop a parallel solver for the Stokes' like sub-problem associated with the momentum and continuity equations and develop parallel hyperbolic methods, necessary for treatment of the constitutive equation.

One requisite goal of this thesis is to assure that the technology we develop is long-lasting both in term of the intellectual contribution to the field and in terms of the practical utility of the actual computer code. For this reason, a widely portable message passing library, MPI [184], is utilized. As a result, the BCALM-preconditioned Krylov method implemented for this thesis is already being used by other researchers [82, 105] and should not be obsoleted in the near future. This is in contrast with the CFS method of Mehrabi [126], written in the proprietary language “NX” for the now obsolete Intel iPSC/860 [97], which has not been used except as a benchmark since 1995. Note that the selection of explicit parallelism is consistent with the trend in the computer industry away from implicit parallelism. Current generation Pentium-type architectures achieve internal parallelism by using implicit methods which attempt to detect parallelizeable elements in the instruction set. However, for their next-generation chip, the 64-bit Itanium microprocessor, parallelism must be explicitly coded into the instruction set [99]. This represents a move towards recognizing that parallelism must be customized for the algorithm of interest.

In the current work, this has led to the exploration of iterative methods. The memory efficiency of these methods makes them attractive, and by developing an effective preconditioner, based on the problem physics, robustness has been achieved in a general framework. The first problem considered was the solution for natural convection in a fluid filled cavity at steady state with high Grashof numbers. This stiff problem provided a well
understood test problem which challenged the robustness of the solver. BCALM precondi-
tioned BiCGStab was shown to be an robust and efficient solution method.

We study the problem of natural convection in a fluid filled cavity, described in detail
in Section 2.5, as a model for the Stokes'-like subproblem encountered in the modelling of
viscoelastic flow, which arises as described in Section 5.3.

In this context, we develop a robust method for the iterative solution of linear equation
sets which arise in the context of finite element discretizations of complex viscous flow
problems. The combination of both block preconditioning techniques based on Schur
complement reduction, described in Section 4.2.1, and multilevel additive Schwarz,
described in Section 2.2.3, allows the generation of a sophisticated preconditioner appro-
priate for the complex equations of mixed type that arise when modelling viscous flows,
including natural convection in a cavity and viscoelastic flow problems. Furthermore, the
design of the preconditioner, detailed in Section 4.2.4, allows the efficient exploitation of
parallelism. When coupled with a Krylov subspace method, the accurate solution of indef-
inite and asymmetric equation sets that result from the discretization of large classes of
differential equations may be obtained.

Schur complements and multilevel additive Schwarz methods play important roles in
the solution of large, sparse, linear systems of equations. Schur complement reduction has
traditionally been utilized as a domain decomposition technique in combination with
direct solvers to partition and reduce the size of the linear system. The additive Schwarz
method has typically been used as a preconditioner for the solution of symmetric linear
systems which arise from the discretization of elliptic operators. We expand these notions
by defining a parallel block preconditioner for the fully coupled linear system that arises
from the discretization of general viscous flow problems.
Through the utilization of spectral dissection to assign nodes to processors in a minimally connected manner, the scalability of the method is ensured. The Schur complement and additive Schwarz parts of the preconditioner are parallelized, and the inclusion of the serial solution of the coarse grid problem does not impact the overall method’s scalability.

The results for the natural convection problem document the excellent performance of the BCALM preconditioned Krylov solver put forward. The incorporation of a block preconditioning scheme based on a Schur complement of the pressures yields a robust method which is insensitive to the magnitude of the Grashof number. Furthermore, the resulting scheme is highly parallelizeable, yielding both single-problem and Gustafson efficiencies over 90% for large problem sizes. The method is much faster than even a state-of-the-art direct method, CFS. Finally, the preconditioner does not exploit either the natural convection problem specifically nor the regular geometry encountered.

The subsequent application of this technique to viscoelastic flow required the consideration of the problem physics involved. With the application of a timestepping technique to the governing equations, a Stokes-like subproblem is recovered both under the Runge-Kutta and Theta-method schemes considered as shown in Section 5.4. This subproblem lends itself to the straightforward application of BCALM-preconditioned Krylov methods. However, in both cases there is a second problem, related to the constitutive equation which must be parallelized. In the case of RK4/DG, this subproblem decouples element-by-element so that parallelization is straightforward and near ideal speedups are observed. In the case of the theta-method splitting, the stress subproblem is fully coupled, and a Krylov method preconditioned with two-level additive Schwarz has been effectively applied. Efficiencies for this are lower, but the overall method is still highly parallel.

The results in Chapter 5 document the excellent performance of BCALM-preconditioned Krylov methods for the complex two-dimensional problem of transient viscoelastic
flow of an Oldroyd-B fluid around an array of cylinders in a channel. This problem is extremely difficult, numerically, due to the steep stress gradients in the boundary layer near the cylinder.

We calculate the flow of an Oldroyd-B fluid, discretized using the DEVSS/DG finite element method, to steady-state using a fourth-order Runge-Kutta timestepping method for \( L = 18 \). By using a mesh with a minimum spacing smaller than \( O(1 \times 10^{-3}) \), previously unresolved features in the cylinder boundary layer and wake are visible in the current results. The result of resolving these features is that the maximum \( \text{De} \) attainable, \( \text{De} = 1.0 \), is higher than that reported by Smith [167], whose mesh was an order-of-magnitude more coarse. Increasing the mesh resolution has enabled the simulation of higher Deborah number flows. The solution of the resulting problem, with \( O(1 \times 10^5) \) unknowns, would have been intractable with prior solution technologies.

The current simulation results appear to match the similarity solution predicted by Öztékin et al. [141] in the wake of the cylinder. The valid region of Öztékin's solution was missed by Smith [167] in his comparison, most likely due to the coarseness of his mesh. Öztékin predicts a instability in this region, however it appears to be highly localized and its manifestation may be affected by the mixed flow in the wake of the cylinder.

The high performance of the parallel fourth-order Runge-Kutta timestepping code is also examined. The parallel BCALM-preconditioned Krylov method put forth in Chapter 4 is coupled with a parallel discontinuous Galerkin update. The resulting algorithm is faster than the serial, frontal algorithm used in previous viscoelastic simulations, even before exploiting the advantages of parallel computing. This work demonstrates the great flexibility of the current solver and it's utility in arbitrary geometries.

Very high parallel efficiencies are documented for the method. Both the application of the BCALM-preconditioned Krylov method and the discontinuous Galerkin updates have
parallel efficiencies over 95% through sixteen processors, even on moderately sized problems.

Furthermore, the BCALM-preconditioned Krylov algorithm is very flexible, as demonstrated by the incorporation of BCALM-preconditioned Krylov techniques into two linear stability analyses. The first linear stability analysis code uses fourth-order Runge-Kutta timestepping and is therefore a ready extension of the transient simulation code. Again, high parallel efficiencies were observed, and the resulting code enables the solution of previously intractable problems. The second linear stability code utilizes a theta-method timestepping with a SUPG method for the stresses. This code is slightly different in structure, using an application of the BCALM-preconditioned Krylov method to the resulting Stokes' like problem, but requiring the solution of a second system related to the coupled stresses. This problem has allowed the calculation of a transition in the flow of an Oldroyd-B fluid through a closely-spaced linear periodic array of cylinders from a two-dimensional steady-state to a three-dimensional steady state for \( L = 3.5 \). These spacings are larger than was possible for Smith in [167] due to the limitations of his frontal technology. This linear stability code, also, has high parallel efficiency.

The effectiveness of these parallel schemes has been established, and the solution of previously impossible calculations has been shown in this thesis. For the flow around the series of equally spaced cylinders in a channel, this has led to the resolution of previously unresolved structures in the stress field. It should be emphasized that the BCALM preconditioner is fairly general -- it exploits the singular block which appears in the linear system due to the presence of a continuity constraint which would be associated with any conservative model, even beyond viscoelastic fluid flows. This parallel preconditioner allows researchers to solve problems with larger numbers of unknown than previously possible and allows the exploitation of parallel computers. The development of an efficient, robust
parallel code capable of solving the viscoelastic flow problem is a significant contribution to this research field.

Future extensions to the current work are possible to have further impact in viscoelastic flow modelling. Some of these are straightforward extensions of the current work. Others require more effort and research. Two of the most important extensions to this work are in the possible generation of a three dimensional viscoelastic code and in the introduction of more complicated constitutive equations.

Due to the large problem size, current three-dimensional modelling has been limited to observing the influence that particular wavenumbers in the third dimension have on a base two dimensional steady flow. With the solution of problems with millions of unknowns now possible, small three dimensional simulations may be accomplished. This will require the development of a three dimensional code which should be a straightforward extension of the current two dimensional code.

In Section 1.3, we estimated the size of a three dimensional simulation as \( O(i \times 10^7) \) unknowns. Of these unknowns, approximately \( 2.3 \times 10^6 \) will be stress unknowns and \( 7.7 \times 10^6 \) will be unknowns associated with the Stokes'-like subproblem. The minimum memory requirement for this problem will be approximately the sum of the storage required for the unknowns, the storage of the Stokes' linear system, and the storage required for the preconditioner, as the storage required for the elemental linear system associated with the discontinuous Galerkin update is negligible. The storage of unknowns and the Stokes'-like linear system is evenly divided onto the processors. The storage of the preconditioner on each processor is actually less expensive than for the overall linear system, as both the local dimension of this system and the expected number of nonzeros for the additive Schwarz problem per processor scale inversely with the number of processors.
Furthermore, the use of a drop-tolerance preconditioning scheme reduces the levels of fill that occurs to $O(1)$. As a result, the amount of memory required will be

$$
\frac{N_{eq}}{P} + \frac{N_{eq,\text{Stokes}}}{P} n_z + \frac{N_{eq,\text{Stokes}} n_z}{P}.
$$

Here $P$ is the number of processors utilized, $N_{eq}$ is the number of total unknowns in the system, $N_{eq,\text{Stokes}}$ is the number of unknowns in the Stokes’ linear system, and $n_z$ is the number of nonzeros per equation. If we assume that each equation contains contributions from each node in the element, then for the three dimensional element shown in Figure 1.4, then the number of nonzeros per equations will be $n_z = 129$. For a moderately sized parallel computer, such as utilized in Chapter 5 for modelling viscoelastic flow, with 16 processors, each processor would require $O(7\times10^7)$ words of storage, or, at double precision, approximately 2.5 Giga-bytes of memory. This is only slightly more memory than is available in our commodity cluster and a commodity cluster could be assembled with this amount of memory per processor. The three-dimensional code would need to assure parallel storage of all arrays for this to be tractable, which would be a significant accomplishment. Realistically, a 32-node parallel computer with 2 Giga-bytes of memory per processor would be necessary.

The parallelization of the RK4/DG timestepping scheme enables the implementation of increasingly sophisticated models for viscoelastic fluids. Various new types of constitutive models are attractive in light of this high-performance parallel solver. These models introduce additional unknowns or couplings which rendered them previously intractable.

The class of micro/macro models such as studied by Nayak [139], an example of which are liquid crystal polymers, would benefit from this solver technology. These models combine a kinetic theory model for the micro-scale interactions with conservation
equations for the macro-scale phenomena. In these models, a distribution function is used to define the stress as a product of molecule orientation $f$ and hydrodynamic force $F^{(h)}$ as

$$
\tau = \int f F^{(h)} du.
$$

(6.2)

The molecule orientation is generally expressed as a differential equation:

$$
\frac{\partial f}{\partial t} = -\nu \cdot \nabla f + \frac{\partial}{\partial u} g[G,f].
$$

(6.3)

This equation is easily incorporated into the BCALM-preconditioned Krylov methods developed in this thesis. Equation (6.3) is analogous to the energy equation found in the natural convection problem, Eq. (2.81). As a result, the high speedup and increased tractability seen for the viscoelastic calculations performed in Chapter 5 should be seen for this problem, too.

Evolutionary models, such as devised by Ghosh [82], seek to vary the internal configuration variables in the constitutive expression. This model, for example, is very nonlinear and requires the calculation of the largest eigenvalue of the rate of strain tensor at every point in the flow. Finding the Jacobian for such a model might be impossible due to the requirement of finding the largest eigenvalue, rendering a standard finite element discretization coupled with Newton-Raphson to generate a steady state solution impractical. By contrast, time integration with the current parallel method is straightforward because of the explicit expression for the stress. The kinematics are decoupled from the proposed constitutive equation in the parallel DEVSS/DG formulation shown.

An improvement to this scheme would introduce coupled multimodes that consider the constitutive equation as having more than one relaxation time associated with it. As a result, a system of equations couple the stress contributions from the different modes. These methods introduce large number of stress unknowns which as the number of stress
unknowns scales linearly with the number of modes. Prior to the development of DEVSS/DG, this would have resulted in extremely large, intractable linear systems. Instead, with parallel DG, multimode models may be implemented in a straightforward manner and the resulting computation of large number of stresses will be neither time consuming nor memory intensive.

Several other potential extensions would attempt to address a shortcoming of the viscoelastic timestepping methods reported in Chapter 5.

The initial value problem for the ODEs associated with viscoelastic flow are solved numerically using a fixed timestep. Classically, for a fixed timestep, the error in a particular timestep has been shown to go to infinity as the timestep goes to infinity. This can be relaxed for cases where the timestepping can be shown to have an attractor, but these results depend on a sufficiently strong finite-time convergence result. See, for example, [171], and the references therein.

An improvement to this technique is the use of adaptive timestepping methods where the timesteps are chosen inductively so that some estimate of the local error at each step is kept below some user-defined tolerance. Such methods are potentially more efficient than fixed timestepping algorithms while maintaining comparable accuracy.

Until recently, it was not possible to show that, as the user-defined tolerance tended towards zero, the number of timesteps would not tend towards infinity [172]. However, recent research has shown a class of adaptive algorithms which have finite time convergence tendencies [170]. The implementation of an adaptive timestepping scheme may potentially speed the time-dependent calculations further, and should be researched in the context of the time integration schemes used here.

Another extension to the current method would allow multiple blocks per processor. This would lower the memory requirement for the additive Schwarz method and speed the
method. Furthermore, the preconditioner's robustness increases with the addition of subdomains, as shown in Figure 5.41. Allowing multiple blocks per processor would better exploit this trait.

The parallel timestepping techniques presented in Chapter 5 for viscoelastic flow problems represent a major advancement in the ability to solve large viscoelastic flow problems. However, putting a major effort into the immediate construction of three dimensional or more complicated stress formulations without considering a simplification of the problem formulation would be contrary to the observed trends in the computer industry. Some researchers in artificial intelligence, estimating the processor speed and memory required to have equivalent processing power as the human mind, have noted that modern computers are typically insufficiently equipped with memory relative to their processing speed [134]. As a result, the trend towards coarse-grained parallelism is likely to continue and the application of massive parallelism to improve the tractability of these problems is unlikely in the next few years. It is interesting to consider the impact that simplified problem formulations would have when used in conjunction with the current technology.

In the Stokes-like linear sub-problem that dominates the two-dimensional timestepping algorithm presented in Chapter 5, 25% of the unknowns are related to gradients of the velocity, \( \mathbf{G} \). In three dimensions, based on the discretization described in Section 1.3, 8 out of the 33 total unknowns introduced by each element would be gradients of the velocity, \( \mathbf{G} \), again approximately 25%. These gradients, \( \mathbf{G} \), were introduced by Szady [176] for numerical stability, but it is interesting to consider the impact of not having these unknowns present in the Stokes-like subproblem. This might be accomplished either by utilizing a new finite element formulation or a different operator splitting.

First of all, the linear system would be 25% smaller, requiring that much less memory based on a naive implementation of the methods presented in Chapter 5. However, the
resulting Stokes-like sub-problem is symmetric, requiring 50% less storage than the methods utilized in this thesis. As a result, only 37% of the memory required for the original implementation would be necessary. Due to this symmetry, simpler Krylov methods such as Conjugate Gradients could be used [154]. Conjugate Gradients requires far fewer operations than either Bi-CGStab or GMRES, at most requiring 50% of the operations necessary for these methods and likely far fewer, depending on the linear system structure [114]. As a result, for the three-dimensional problem with the finest mesh described in Section 1.3 and discussed earlier in this chapter, instead of requiring 2.5 Giga-bytes memory per processor and 24 hours for problem solution on a 16 processor parallel computer, only 947 Mega-bytes of memory per processor and 4.5 hours would be required. The combination of simplified formulations with both more complex constitutive models and three-dimensional flow simulations will mark the next major advancement in the modelling of viscoelastic flow problems.
References


[183] K. Underwood, hamachi: A Packet Engines GNIC-II Gigabit Ethernet driver v0.15i, keithu@parl.clemson.edu, (3/16/2000).


