A High-Order Discontinuous Galerkin Multigrid Solver for Aerodynamic Applications

by

Krzysztof J. Fidkowski

B.S., Aerospace Engineering (2003)
B.S., Physics (2003)
Massachusetts Institute of Technology

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aerospace Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2004

Author .....
Department of Aeronautics and Astronautics
May 14, 2004

Certified by .....
David L. Darmofal
Associate Professor
Thesis Supervisor

Accepted by .....
Edward M. Greitzer
H N Slater Professor of Aeronautics and Astronautics
Chair, Committee on Graduate Students

A High-Order Discontinuous Galerkin Multigrid Solver for Aerodynamic Applications

by

Krzysztof J. Fidkowski

Submitted to the Department of Aeronautics and Astronautics
on May 14, 2004, in partial fulfillment of the
requirements for the degree of
Master of Science in Aerospace Engineering

Abstract

Results are presented from the development of a high-order discontinuous Galerkin finite element solver using p-multigrid with line Jacobi smoothing. The line smoothing algorithm is presented for unstructured meshes, and p-multigrid is outlined for the nonlinear Euler equations of gas dynamics. Analysis of 2-D advection shows the improved performance of line implicit versus block implicit relaxation. Through a mesh refinement study, the accuracy of the discretization is determined to be the optimal $O(h^{p+1})$ for smooth problems in 2-D and 3-D. The multigrid convergence rate is found to be independent of the interpolation order but weakly dependent on the grid size. Timing studies for each problem indicate that higher order is advantageous over grid refinement when high accuracy is required. Finally, parallel versions of the 2-D and 3-D solvers demonstrate close to ideal coarse-grain scalability.

Thesis Supervisor: David L. Darmofal
Title: Associate Professor

Acknowledgments

I would like to recognize and thank the people that made this research possible. First and foremost, I would like to thank my advisor, David Darmofal, for his enthusiasm about the project and insight in the face of problems. His countless ideas motivated me at times when I believed to be near a dead end. In addition, I am thankful to Jaime Peraire, Bob Haimes, and the rest of the Project X team (Matthieu, Todd, Paul, Mike, James) for their abundant contributions to the success of the project.

I would like to give a special thanks to my wife, Christina, for her companionship and support throughout the past year. Of course, I'm thankful to my parents and brothers for supporting me in all my academic and non-academic endeavours.

This work was made possible in part by the Computational Science Graduate Fellowship provided by the U.S. Department of Energy and administered by the Krell Institute.

Contents

1 Introduction **13**

1.1 Motivation 13

1.2 Background 18

1.2.1 Discontinuous Galerkin Methods 18

1.2.2 Multigrid for the Euler and Navier-Stokes Equations 18

2 Discontinuous Galerkin Method **21**

2.1 Discretization 21

2.2 Element Types, Interpolation, and Geometry Representation 23

2.3 Numerical Integration 24

3 Multigrid Solver **25**

3.1 Line-Implicit Smoother 25

3.2 Line Creation 26

3.3 Robustness and Under-Relaxation 30

3.4 p-Multigrid 32

3.4.1 Motivation 32

3.4.2 FAS and Two-Level Multigrid 32

3.4.3 V-cycles and FMG 34

3.4.4 Operator Definition 35

3.5 Storage and Implementation 37

4 Stability Analysis **39**

4.1 One-dimensional Analysis 39

4.2 Two-dimensional Analysis 43

5 Results **47**

5.1 Two-dimensional Results 47

5.1.1 Ringleb Flow . . . 48
5.1.2 Flow over a Gaussian Bump . . . 50
5.1.3 Joukowski Airfoil . . . 53
5.2 Three-dimensional Results . . . 59
5.2.1 Three-dimensional Ringleb Flow . . . 59
5.2.2 Flow in a Duct with a Gaussian Perturbation . . . 62

6 Parallelization **67**
6.1 Background . . . 67
6.2 Implementation . . . 68
6.2.1 Data Structure . . . 69
6.2.2 Parallel Solution Algorithm . . . 70
6.2.3 Grid Partitioning and Joining . . . 71
6.3 Scalability Results . . . 72
6.4 Conclusions . . . 74

7 Conclusions **75**

A Entropy Fix for the Roe Flux **77**

B Boundary Conditions **79**

C Hierarchical Basis **81**
C.1 Kernel Functions . . . 81
C.2 Hierarchical Basis in Two-dimensions . . . 82
C.3 Hierarchical Basis in Three-dimensions . . . 84

List of Figures

1-1 Results from the first drag prediction workshop: total drag variation. Source: Levy *et al* [26]. Reproduced with permission. 14

3-1 Possible line configuration: (a) after Stage I and (b) after Stage II. 28
3-2 Lines in flow over a Gaussian bump. 30
3-3 Lines in flow over a Joukowski airfoil. 31
3-4 Diagram of storage in the multigrid algorithm. 38

4-1 1-D Lagrange basis functions on the reference element for $p = 2$. 41
4-2 Smoothing footprint for (a) $p = 0$ and (b) $p > 0$. 43
4-3 Block Jacobi footprint for $p = 1$, $N = 24$: (a) $\alpha = 1^o$, (b) $\alpha = 25^o$. 44
4-4 Line Jacobi footprint for $p = 1$, $N = 24$: (a) $\alpha = 1^o$, (b) $\alpha = 25^o$. 45

5-1 Description of Ringleb flow. 48
5-2 Ringleb flow: accuracy vs. CPU time. 49
5-3 Ringleb flow: FMG convergence history. 50
5-4 Ringleb flow (1408 elements): FMG history with full convergence on each level. 51
5-5 Domain for flow over a Gaussian bump. 51
5-6 Gaussian bump: accuracy vs. CPU time. 52
5-7 Gaussian bump: FMG convergence history. 52
5-8 Gaussian bump (2348 elements): FMG history with full convergence on each level. 53
5-9 Domain for flow over a Joukowski airfoil (not to scale). 54
5-10 Close-up of the intermediate mesh around the Joukowski airfoil. Total mesh size is 3896 elements. 54
5-11 Joukowski airfoil: accuracy vs. CPU time. 55
5-12 Joukowski airfoil: FMG convergence history. 55
5-13 Joukowski airfoil (3896 elements): FMG history with full convergence on each level. 56

5-14 Joukowski airfoil: accuracy convergence for various M. 57
5-15 Joukowski airfoil: Mach contours for $p = 1$ and $p = 3$ interpolation at $M = 0.2$ and $M = 0.002$. For $p = 1$ the accuracy decrease is evident at low M, while for $p = 3$ it is not. 58
5-16 Joukowski airfoil (3896 elements): multigrid convergence for $p = 3$. 59
5-17 Joukowski airfoil (3896 elements), $M = 0.2$ and $p = 3$: comparison of convergence rates of various solvers. 60
5-18 Description of Ringleb flow in 3-D. 60
5-19 3-D Ringleb flow: accuracy vs. CPU time. 61
5-20 3-D Ringleb flow: FMG convergence history. 62
5-21 3-D Ringleb flow (2560 elements): FMG history with full convergence on each level. 63
5-22 Flow over a Gaussian perturbation in 3-D. 63
5-23 3-D Gaussian bump: accuracy vs. CPU time. 64
5-24 3-D Gaussian bump: FMG convergence history. 65
5-25 3-D Gaussian bump (1920 elements): FMG history with full convergence on each level. 65

6-1 Connectivity of two sub-domains via Boundary Face Groups. 69
6-2 Grid splitting utility applied to a Gaussian bump grid. 71
6-3 Parallel scalability of (a) block-implicit and (b) line-implicit smoothing. $p = 2$ interpolation was used for all cases. 72
6-4 Joukowski airfoil (3896 elements): parallel convergence of the line-implicit smoother for $p = 0$ interpolation. 73
6-5 3-D Gaussian bump (1920 elements): time to solution using parallel FMG with residual-based switching. $p = 2$ interpolation on fine level. 74

C-1 Reference triangle. 82
C-2 Reference tetrahedron. 84

List of Tables

2.1 Quadrature order requirements for the 2-D Euler equations using conservative state variables: p is the interpolation order, and q is the element geometry order. 24
2.2 Quadrature order requirements for the 3-D Euler equations. 24

3.1 Approximate storage requirements per equation in 2-D and 3-D for N elements. 37

C.1 Two-dimensional hierarchical basis functions for various orders. 83
C.2 Three-dimensional hierarchical basis functions for various orders. 85

Chapter 1

Introduction

1.1 Motivation

While Computational Fluid Dynamics (CFD) has matured significantly over the past decades, computational costs are extremely large for aerodynamic simulations of aerospace vehicles. In this applied aerodynamics context, the discretization of the Euler and/or Navier-Stokes equations is almost exclusively performed by finite volume methods. The pioneering work of Jameson began this evolution to the status quo [21, 18, 20, 19]. During the 1980's, upwinding mechanisms were incorporated into these finite volume algorithms leading to increased robustness for applications with strong shocks, and perhaps more importantly, to better resolution of viscous layers due to decreased numerical dissipation in these regions [42, 35, 43, 36, 44]. The 1990's saw major advances in the application of finite volume methods to Navier-Stokes simulations, in particular to the Reynolds-Averaged Navier-Stokes (RANS) equations. Significant gains were made in the use of unstructured meshes and solution techniques for viscous problems [2, 34, 28, 31]. While these algorithmic developments have resulted in an ability to simulate aerodynamic flows for very complex problems, the time required to achieve sufficient accuracy in a reliable manner places a severe constraint on the application of CFD to aerospace design.

The accuracy of many finite-volume methods currently used in aerodynamics is at best second order, meaning that the error decreases as $O(h^2)$, where h is a measure of the grid spacing. As a practical matter, however, the accuracy of these methods on more realistic problems appears to be less than this, ranging between first and second order. A question of interest is whether this accuracy is adequate for engineering purposes. Results from two AIAA CFD drag prediction workshops [26, 23] suggest that this may not be the case given current grid sizes and gridding methods. The first Drag Prediction Workshop (DPW I) compared results of three wind tunnel tests to current state-of-the-art CFD methods for

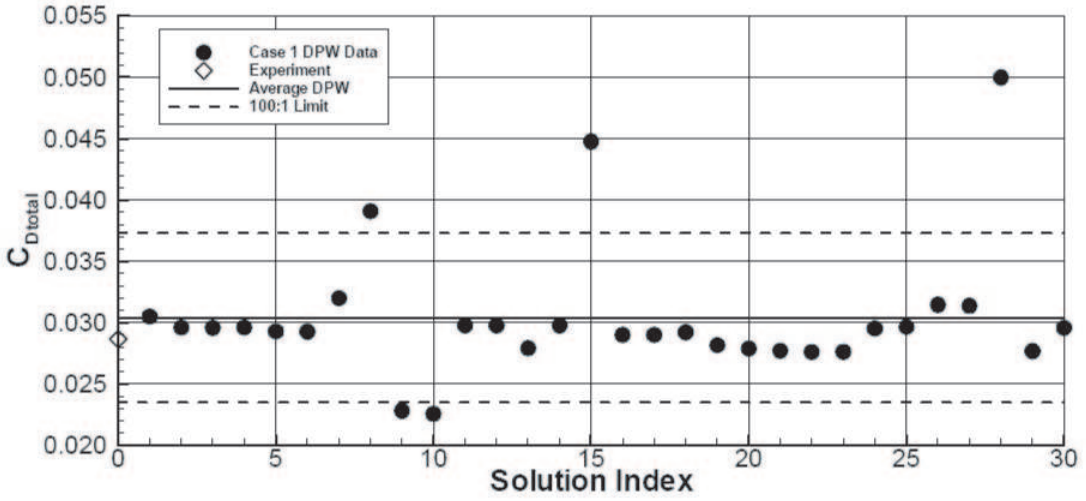

Figure 1-1: Results from the first drag prediction workshop: total drag variation. Source: Levy *et al* [26]. Reproduced with permission.

force and moment prediction. Figure 1-1 shows the variation in the total drag between experiment and CFD for the first test case: a DLR-F4 wing-body configuration at $M_\infty = 0.75, C_L = 0.5$, and $Re_c = 3 \times 10^6$. The solution index in the figure corresponds to the different CFD submissions, encompassing variations in grid type (multiblock structured, unstructured, overset) and turbulence modeling (Spalart-Allmaras, Wilcox $k-\omega$, Menter's SST $k-\omega$, $k-\epsilon$, etc.). Even after neglecting the outliers, which deviate by as much as 200 drag counts from the experiment (1 drag count $= 1 \times 10^{-4}$ variation in the drag coefficient, C_D), the CFD results still show a spread of about 40 drag counts.

To demonstrate the importance of 1 drag count of error, a simple calculation is presented showing the effect of a change in C_D by 1 drag count for a typical long-range passenger aircraft. This calculation parallels that of Vassberg [45], who considered the effect of a 1% change in drag on range of long-range jet aircraft and arrived at a similar answer. The aircraft under consideration is a Boeing 747-400 with the following weight breakdown: $W_L = 493,000$ lb, $W_F = 382,000$ lb, and $W_P = 99,000$ lb, where W_L is the zero-fuel landing weight, W_F is the fuel weight, and W_P is the payload weight. Using the Breguet range equation (V = cruise velocity, SFC = Specific Fuel Consumption),

$$R = \frac{C_L}{C_D} \frac{V}{SFC} \ln \left(\frac{W_L + W_F}{W_L} \right), \tag{1.1}$$

the effect of a 1 drag count change on the allowable payload, W_P, is estimated for a constant

range, R. Assuming a typical cruise C_D of 0.03, 1 drag count represents a 0.33% increase in drag. To maintain the same range, $\ln\left(\frac{W_L+W_F}{W_L}\right)$ has to increase by 0.33%, to first order. If the configuration is fuel volume limited such that W_F is constant, the payload has to decrease to reduce W_L. The reduction in payload turns out to be 2.1% or 2100 lb. Allotting 250 lb per passenger, the cost of 1 drag count in this case turns out to be over 8 passengers (out of approximately 400 initial capacity). Alternatively, if the lost payload weight can be replaced by fuel, keeping $W_L + W_F$ constant, the loss in payload is 930 lb, or almost 4 passengers, with the additional cost of 930 lb in fuel. Thus, even seemingly small changes in C_D can have a significant impact on the revenue of a long-range aircraft. In terms of industry profitability, the ability to predict C_D accurately in the early design stages of an aircraft is crucial.

Returning to the results of DPW I, the uncertainty in C_D for the case presented is unacceptable for industry drag prediction. Although surprising, these results do not necessarily indicate the inaccuracies of the CFD discretizations themselves. First, some of the outliers in the figure have been attributed to either errors in runs by the participants or to the use of incorrect parameters [26]. Second, the chief complaint by the participants was that the standard grids provided for the case were not adequate. Although an effort was undertaken to make the grids indicative of those used in industry, they were also made simple enough to maximize participation. In the end, one of the conclusions of DPW I was that the grids provided were not fine enough to resolve the necessary flow features.

A second workshop, DPW II, was carried out in part to address the need for better grids. In DPW II, the first test case was a similar wing-body configuration, the DLR-F6. Three sets of standard grids were provided (for each grid type) with the intermediate grids constructed based on industry standards. The intermediate grids were about 1.5 to 2 times larger than those used in DPW I and based on size, were more in line with Vassberg's gridding recommendations [45]. The drag results on these intermediate grids show a lower spread of 14 counts in drag prediction [23], which translates into 50-110 passengers for the 747-400 using the above analysis. Although the drag results improved from DPW I, the grid refinement may not convey the whole story. A follow-up study by Lee-Rausch *et al* [24] indicates that the various codes converged to different solutions in terms of shock location and separation patterns. Furthermore, the DPW II results for a second configuration, which included a nacelle and pylon, showed an increase in drag increment with grid refinement. Lee-Rausch *et al* also revisited the DPW I case and demonstrated that for some of the unstructured grid codes, grid refinement resulted in an increase in the variation of forces and moments [24]. One explanation offered by these authors is that the numerical schemes or other code-to-code differences may have a more significant impact than was previously

thought. These observations suggest that even at the industry-level standards put forth by the workshops, the current CFD practices do not reliably yield engineering-required accuracy.

The development of a practical higher-order solution method could help alleviate this accuracy problem by significantly decreasing the computational time required to achieve an acceptable error level. To better demonstrate the potential of higher-order methods, we make the following assumptions:

- The error, E, in the solution (or an output of the solution) is $O(h^p)$.
- The number of elements, N_{el}, in the grid is related to the cell size by $N_{el} = O(h^{-d})$, where d is the spatial dimension of the problem.
- Higher-order accuracy is achieved by increasing the number of unknowns per element (in a finite element manner), N_{dof}, which scales as $N_{dof} = O(p^d)$.
- The number of floating point operations (or the work), W, required to solve the discrete system is $O(N_{el}N_{dof}^w) = O(p^{wd}/h^d)$ where w is the complexity of the solution algorithm.
- The time required to complete a single operation is $1/F$, and hence the total time for solution of the system of equations if $T = W/F$.

Combining these assumptions, the time to achieve a specified error is given by

$$T = O\left(\left(p^w/E^{1/p}\right)^d/F\right).$$

Taking the log of this relationship yields

$$\log T = d\left(-\frac{1}{p}\log E + w\log p\right) - \log F + \text{constant}.$$

If the accuracy requirements are stringent such that $E << 1$, and if the solution complexity is moderate, we expect that the $\log E$ term will dominate the $\log p$ term. Thus, the time required will depend exponentially on p and d. This reasoning demonstrates the potentially significant benefit of improving the order of accuracy. Furthermore, since T scales only inversely with the computational speed, F, small changes in p or w can be as significant as increasing computational power.

Numerous reasons exist for why current finite-volume algorithms are not practical at higher order and have remained second-order. The root cause of many of these difficulties lies in the extended stencils that these algorithms employ. For finite volume discretizations

that explicitly add numerical dissipation, the extended stencils arise from the higher-order stabilization terms. For finite volume algorithms that introduce stabilization through upwinding, the extended stencils arise through the local interpolants used to increase accuracy. These extended stencils contribute to difficulties in:

- *Stable iterative algorithms.* A well-known fact is that the iterative solution of these discretizations requires multi-stage methods and/or implicitness beyond a locally implicit scheme. Another common iterative approach employs backward Euler in which the Jacobian of the higher-order discretization is replaced by a lower-order approximation. Unfortunately, Mavriplis [29] has shown that the use of lower-order approximations severely limits the convergence rate attainable for higher-order finite-volume simulations of complex problems even when the lower-order systems are solved exactly.

- *Memory requirements.* Extended stencils degrade the sparsity of the linearized systems of equations used in implicit solution methods. This increased fill results in very high memory requirements and is the reason that lower-order approximations are often utilized.

- *Parallelization.* Similar to the increased memory requirements, the large support of the extended stencils also increases the communication bandwidth required for parallel computations. However, when the number of cells in each processor is large (as is common in the coarse-grain parallelism used today), this effect may be minimal.

By contrast, finite element formulations introduce higher-order effects compactly within the element. Thus, viewed from the element level, the stencils are not extended for higher-order finite element discretizations. Recently, Venkatakrishnan *et al* [46] showed that higher-order finite element schemes have significant advantages for smooth inviscid and viscous flows; however, they also delineated several remaining challenges that must be addressed before higher-order methods will be robust and efficient for practical applications containing shocks or other under-resolved flow features. While Venkatakrishnan *et al* considered both continuous and discontinuous Galerkin discretizations, this work focuses solely on discontinuous Galerkin methods. For discontinuous Galerkin (DG) formulations, the element-to-element coupling exists only through the flux at the shared boundaries between elements. This limited coupling for DG discretizations is an enabling feature that permits the development of an efficient higher-order solver that may lead to *significant* improvements in the turn-around time for reliably accurate aerodynamic simulations.

In this thesis, a multigrid solution algorithm is presented for higher-order DG discretizations. Although the results shown are for inviscid flows, the solution algorithm is designed

to extend naturally to high Reynolds number viscous problems [33]. First, a description of the DG discretization for the Euler equations is given in Chapter 2. A p-multigrid algorithm is then described in which the coarse levels are formed from lower-order discretizations (using a hierarchical basis) and in which the smoother is line-element block Jacobi. Stability analysis indicates that the eigenvalues of the iterative algorithm are relatively insensitive to p but are dependent on h. An important result from this analysis is that element block Jacobi schemes are stable regardless of the order of approximation without the need for multi-staging (which is not true for higher-order methods using extended stencils). Numerical results are presented for several 2-D and 3-D problems, demonstrating that high accuracy solutions are obtained in less computational time using higher-order discretizations than using highly refined grids. Finally, a parallel implementation of the solver is presented together with almost ideal scalability results on a coarse-grain cluster.

1.2 Background

1.2.1 Discontinuous Galerkin Methods

Discontinuous Galerkin methods have been developed extensively for hyperbolic conservation laws including the Euler equations and, to a lesser extent, for diffusive operators necessary for the full Navier-Stokes equations. Bassi and Rebay have done numerous studies of the DG discretization for Euler and Navier-Stokes equations [4, 5, 6]. Their results show the realizability of higher-order accuracy for the Euler and Navier-Stokes equations using a DG discretization. In addition, they demonstrate several key points in the implementation of DG, such as the requirement of higher-order boundary representations in the geometry [4] and a compact discretization of the viscous terms [6].

Further details on DG methods are given by Cockburn and Shu [10], who review DG discretizations for convection-dominated problems oriented towards Runge-Kutta solution methods. Their work outlines the state-of-the-art in this field and provides useful information on the discretization, choice of flux function, limiting, and boundary treatment.

1.2.2 Multigrid for the Euler and Navier-Stokes Equations

The use of multigrid for the solution of the Euler equations was pioneered by Jameson in 1983, who demonstrated a significant convergence speedup in the solution of 2-D transonic flows on structured meshes [18]. In 1988, Mavriplis outlined one method of performing multigrid on unstructured triangular meshes using a sequence of unrelated coarse and fine meshes [27]. On various problems, he was able to accelerate the convergence by an order of

magnitude, which is comparable to structured multigrid algorithms. Although Mavriplis's method required re-gridding at each coarsening, it clearly illustrated the realizability of multigrid on unstructured meshes. Mavriplis and Jameson [30] then extended this work to the Navier-Stokes equations and achieved similar results.

Allmaras [1] presents a thorough 2-D analysis of various multigrid preconditioners for the advection/diffusion problem as well as for the linearized Navier-Stokes equations. His Fourier analysis is similar to that employed for the line-implicit scheme presented in Chapter 4. Allmaras's goal was to determine the combinations of preconditioners and coarsening strategies that would result in the effective damping of all error modes on at least one grid during a multigrid cycle. He found that for conventional multigrid, alternating direction implicit (ADI) relaxation is necessary to smooth the error components with high frequency modes in either of the two dimensions since these modes cannot be represented on a fully coarsened grid. For semi-coarsening, semi-implicit line-Jacobi relaxation can be used to smooth the high frequency modes in one direction. Alternatively, point-implicit relaxation can be used if the grid is semi-coarsened in both directions separately, which results in a sequence of grids from which the prolongated corrections have to be combined. These requirements for the proper elimination of all modes lead to the expected multigrid convergence improvements but at a sizeable cost. In particular, extending the conventional multigrid recommendations to three dimensions would require alternating *plane*-implicit relaxation in three directions.

Further work in multigrid for the Navier-Stokes equations is provided by Pierce and Giles [34], who looked into efficient preconditioned multigrid methods for Euler and Navier-Stokes equations. In contrast to the all-inclusive damping outlined by Allmaras, Pierce and Giles presented two less expensive methods for the Euler and turbulent Navier Stokes equations. For the Euler equations, they found that in practice, standard scalar preconditioning with full coarsening demonstrates relatively good convergence rates despite failing to satisfy all the damping requirements. For the Navier-Stokes equations, they found that the combination of block-Jacobi preconditioning and semi-coarsening in the direction normal to the wall yields computational savings of an order of magnitude over the standard full-coarsening with scalar preconditioning [34]. Their analysis shows that the effectiveness of this scheme is due to a balance between streamwise convection and normal diffusion in asymptotically stretched boundary layer cells, which enables the preconditioner to damp all convective modes. Coarsening normal to the wall then damps the acoustic modes. In this manner, the dominant error modes are effectively smoothed, resulting in improved convergence.

In his more recent work, Mavriplis [29] discusses the advantages and disadvantages of linear versus non-linear multigrid. Although linear multigrid is in many cases cheaper, it

is also less robust when the initialization is not close to the final solution. In addition, performing linear multigrid requires storage of the full Jacobian, which becomes prohibitive for large problems. This observation in part motivated the choice of non-linear multigrid for this work, as described in Chapter 3.

Finally, regarding the use of multigrid methods in practice, Brandt [8] has outlined numerous barriers to achieving "Textbook Multigrid Efficiency" (TME). According to Brandt, TME means solving a discrete partial differential equation problem in computational work that is only a small (less than 10) multiple of the operation count in evaluating the residual of the discretized system of equations. Brandt notes that current RANS solvers employing multigrid are far from this optimum, requiring on the order of 1500 residual evaluations to converge outputs to within one percent of their final values. Achieving TME would mean a possible two orders of magnitude improvement in convergence. This observation illustrates that much work is still needed in the development of an efficient multigrid solver for aerodynamic applications.

Chapter 2

Discontinuous Galerkin Method

2.1 Discretization

We now describe the DG discretization of the compressible Euler equations (for additional details, consult the review by Cockburn and Shu [10] and the references therein). For generality, we describe the 3-D case. The Euler equations of gas dynamics are given by

$$\mathbf{u}_t + \nabla \cdot \mathcal{F}(\mathbf{u}) = 0, \tag{2.1}$$

where $\mathbf{u}$ is the conservative state vector,

$$\mathbf{u} = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ \rho E \end{pmatrix},$$

and $\mathcal{F} = (\mathbf{F}^x, \mathbf{F}^y, \mathbf{F}^z)$ is the inviscid flux,

$$\mathbf{F}^x = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ \rho uH \end{pmatrix}, \quad \mathbf{F}^y = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ \rho vH \end{pmatrix}, \quad \mathbf{F}^z = \begin{pmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ \rho wH \end{pmatrix}. \tag{2.2}$$

The total enthalpy is given by $H = E + p/\rho$, and the equation of state is

$$p = (\gamma - 1)\left[\rho E - \frac{1}{2}\rho\left(u^2 + v^2 + w^2\right)\right]. \tag{2.3}$$

The 2-D equations are obtained by setting $w = 0$ and neglecting the z-component of the momentum equation.

Let $\mathcal{V}_h^p$ be the space of discontinuous vector-valued polynomials of degree p on a subdivision T_h of the domain Ω into elements such that $\Omega = \bigcup_{\kappa \in T_h} \kappa$. The DG discretization of the Euler equations is of the following form: find $\mathbf{u}_h \in \mathcal{V}_h^p$ such that $\forall \mathbf{v}_h \in \mathcal{V}_h^p$,

$$\begin{aligned} B(\mathbf{v}_h, \mathbf{u}_h) \equiv \quad & \sum_{\kappa \in T_h} \Big\{ \int_\kappa \mathbf{v}_h^T \left(\mathbf{u}_h\right)_t d\mathbf{x} - \int_\kappa \nabla \mathbf{v}_h^T \cdot \mathcal{F}(\mathbf{u}_h) d\mathbf{x} \\ & + \int_{\partial\kappa \setminus \partial\Omega} \mathbf{v}_h^{+T} \mathcal{H}(\mathbf{u}_h^+, \mathbf{u}_h^-, \hat{\mathbf{n}}) ds \\ & + \int_{\partial\kappa \cap \partial\Omega} \mathbf{v}_h^{+T} \mathcal{H}_{\mathrm{bd}}(\mathbf{u}_h^+, \mathbf{u}_h^-, \hat{\mathbf{n}}) ds \Big\} = 0, \end{aligned} \tag{2.4}$$

where $\mathcal{H}(\mathbf{u}_h^+, \mathbf{u}_h^-, \hat{\mathbf{n}})$ and $\mathcal{H}_{\mathrm{bd}}(\mathbf{u}_h^+, \mathbf{u}_h^-, \hat{\mathbf{n}})$ are inviscid numerical flux functions for interior and boundary edges, respectively. The $()^+$ and $()^-$ notation indicates the trace value taken from the interior and exterior of the element, respectively, and $\hat{\mathbf{n}}$ is the outward-pointing normal of the element. The numerical flux function used to evaluate the boundary flux on $\partial\Omega$ need not coincide with that used for the interior edges. For the interior flux, the Roe-averaged flux function was used [35]. An entropy fix was employed on the flux function as outlined in Appendix A. The boundary conditions on $\partial\Omega$ are imposed weakly by constructing an exterior boundary state on $\partial\Omega$ that is a function of the inner state and of the boundary condition data, $\mathbf{u}_h^-(\mathbf{u}_h^+, \mathrm{BCData})$. A description of all the boundary conditions used in this thesis is given in Appendix B.

The final discrete form of the DG discretization is constructed by selecting a basis for $\mathcal{V}_h^p$. Specifically, a set of element-wise discontinuous functions $\{\phi_j\}$ is introduced, such that each ϕ_j has local support on only one element. The solution to the DG discretization has the form,

$$\mathbf{u}_h(t, x) = \sum_j \overline{\mathbf{u}}_j(t)\phi_j(x).$$

Even though our interest lies in the steady-state solution, the presence of the unsteady term is useful for improving the initial transient behavior of the solver. A simple backward Euler

discretization in time is used so that the final discrete system is

$$\mathcal{M}\frac{1}{\Delta t}\left(\overline{\mathbf{u}}^{n+1}-\overline{\mathbf{u}}^{n}\right)+\mathbf{R}(\overline{\mathbf{u}}^{n+1})=0, \tag{2.5}$$

where $\mathcal{M}$ is the mass matrix and $\mathbf{R}$ is the residual vector representing the final three terms of (2.4). In Chapter 3, we will drop the over-bar notation for the discrete solution vector.

2.2 Element Types, Interpolation, and Geometry Representation

The primitive elements are triangles in 2D and tetrahedra in 3D. Each element can be mapped to either the reference triangle or to the reference tetrahedron. Two basis types were implemented for the interpolation of the state: a Lagrange basis and a hierarchical basis. The Lagrange basis consists of functions φ_i, whose values on a set of equally-spaced nodes n_j within the reference element are given by δ_{ij}. In this basis, the sets of functions for different orders are disjoint. On the other hand, in the hierarchical basis, the set of basis functions for order $p-1$ is a subset of the set for order p. Higher order interpolation is achieved by adding the appropriate number of new, higher-order, basis functions. The particular choice of the hierarchical basis functions used is described in Appendix C.

Geometry representation on the domain boundary was achieved via high-order curved elements. In a high-order element, the position of additional nodes corresponding to the reference-element nodes used in the Lagrange basis are specified, and the local-to-global mapping is given by

$$\mathbf{x}=\sum_j \phi_j(\xi)\mathbf{x}_j, \tag{2.6}$$

where ξ is the coordinate vector in the reference element, $\mathbf{x}$ is the global coordinate vector, φ_j is the Lagrange basis function associated with node j, and $\mathbf{x}_j$ is the specified global coordinate vector for node j. For curved boundary geometries, the additional non-interior nodes are placed on the domain boundary.

An adverse effect of using curved elements is that curved elements no longer achieve the same order of interpolation on the global element as on the reference element. This effect arises because the mapping in (2.6) introduces higher-order components into the global interpolation, with the consequence that the basis for interpolating the desired order in $\mathbf{x}$ is no longer complete. However, the effect was not found to degrade the accuracy at an observable level on the smooth problems considered in the accuracy studies. On highly

curved elements, one solution is to increase the order of interpolation for those elements to a level for which the desired order can be achieved.

2.3 Numerical Integration

Integration over the elements or over the element boundaries is performed through numerical quadrature. That is, an integral over a line segment, triangle, or tetrahedron is calculated as

$$\int_A f(\mathbf{x}) dA \approx \sum_g \omega_g f(\mathbf{x}_g), \tag{2.7}$$

where ω_g is the weight and $\mathbf{x}_g$ is the coordinate associated with Gauss quadrature point g. Quadrature requirements depend on the order of interpolation and increase on curved elements to capture the higher-order changes in the Jacobian of the local-to-global mapping. The sufficiency of the quadrature used was verified through the accuracy studies presented in Chapter 5 and is summarized in Tables 2.1 and 2.2. The number of quadrature points necessary for a given order depends on the quadrature rule used. An extensive listing of 2-D and 3-D quadrature rules is given in Solín *et al* [39].

Table 2.1: Quadrature order requirements for the 2-D Euler equations using conservative state variables: p is the interpolation order, and q is the element geometry order.

	Edge (1-D)			Interior (2-D)		
	q=1	q=2	q=3	q=1	q=2	q=3
$p=0$	1	3	5	-	-	-
$p=1$	5	5	7	3	3	4
$p=2$	5	7	7	4	5	6
$p=3$	7	9	9	6	7	8

Table 2.2: Quadrature order requirements for the 3-D Euler equations.

	Face (2-D)			Interior (3-D)		
	q=1	q=2	q=3	q=1	q=2	q=3
$p=0$	3	4	4	-	-	-
$p=1$	4	5	6	4	5	6
$p=2$	6	7	8	5	6	7
$p=3$	8	9	10	7	7	8

Chapter 3

Multigrid Solver

To solve the nonlinear system $\mathbf{R}(\mathbf{u}) = 0$, a p-multigrid scheme is used with line-Jacobi smoothing. A generic iterative scheme can be written as

$$\mathbf{u}^{n+1} = \mathbf{u}^n - \mathbf{P}^{-1}\mathbf{R}\left(\mathbf{u}^n\right), \tag{3.1}$$

where the preconditioner, $\mathbf{P}$, is an approximation to $\frac{\partial \mathbf{R}}{\partial \mathbf{u}}$. Two preconditioners were considered: an elemental block-Jacobi smoother in which the unknowns on each element are solved for simultaneously, and an elemental line block-Jacobi smoother in which the unknowns on each line of elements are solved for simultaneously. The line smoother, as well as the multigrid solver based on it, will be discussed in detail.

3.1 Line-Implicit Smoother

The motivation for using a line smoother is that in strongly convective systems the transport of information proceeds along characteristic directions. Solving implicitly on lines of elements connected along these directions alleviates the stiffness associated with strong convection. Also, for viscous flows, the line solver is an important ingredient in removing the stiffness associated with regions of high grid anisotropy which are frequently required in viscous layers [1, 28]. In such cases, the lines are formed between elements with the strongest coupling, including the effects of both convection and diffusion. In either regime, the implementation of such a smoother requires the ability to connect elements into lines and to solve implicitly on each line.

The line creation algorithm is described in detail in Section 3.2. Given a set of N_l lines, or disjoint sets of adjacent elements, the state updates are obtained by solving N_l block tridiagonal systems constructed from the linearized flow equations. Let n_l, $1 \leq l \leq N_l$,

denote the number of elements in line l, and let $\mathbf{M}_l$ denote the linear system for line l. $\mathbf{M}_l$ is a block $n_l \times n_l$ matrix, where the dimension of each block is the number of unknowns per element. The on-diagonal blocks of $\mathbf{M}_l$ consist of the local Jacobians associated with the elements on the line. The off-diagonal blocks of $\mathbf{M}_l$ represent the influence of the states in the off-line elements on the residuals in the on-line elements. Only the block-tridiagonal entries are included in $\mathbf{M}_l$ although it is possible for a line of elements to "wrap back" such that the true matrix structure is not tridiagonal. However, ignoring these off-diagonal blocks was not found to cause a substantial loss in performance.

The final form of the preconditioner based on the elemental line smoother is augmented by the addition of the unsteady term,

$$\mathbf{P} = \mathbf{M} + \frac{1}{\Delta t}\mathcal{M}, \tag{3.2}$$

where $\mathbf{M}$ is the entire set of line matrices. The addition of the time term corresponds to solving for a finite time step, Δt, in the unsteady problem. Mathematically, this addition makes the system more diagonally dominant and hence better conditioned for the iterative method. Physically, instead of steady-state, the equations now represent the evolution of the system at a time increment of Δt. The time term is used to help alleviate transients during the solution process. As the solution begins to converge, $\Delta t \to \infty$. A discussion of how Δt is set is given in Section 3.3.

Inversion of $\mathbf{P}$ uses a block-tridiagonal algorithm in which the block diagonal is LU decomposed. As the dominant cost of the line solver (especially for higher-order schemes) is the LU decomposition of the diagonal, the computational cost of the line smoother scales in the same manner as for the simpler elemental block-Jacobi. It will be shown in Section 5.1.3 that, per iteration, the line smoother is approximately twice as slow as the block smoother. However, also shown will be the significantly better performance of the line smoother due to the increased implicitness along strongly-coupled directions.

3.2 Line Creation

The effectiveness of the line smoother depends on the length of the lines and, for inviscid flows, on their alignment with the convective direction. In general, these criteria are difficult to achieve on irregular triangular meshes unless the flow pattern is known ahead of time and the mesh is constructed accordingly.

Much work has been done in the past in the area of line creation on unstructured meshes. For viscous problems, Mavriplis used a geometry-based algorithm to improve the iterative

method in boundary layer regions. Specifically, smoothing on highly-stretched was done using a line-implicit scheme connecting the stretched elements [28, 31]. Okusanya developed a nodal line creation algorithm in which the lines were made to follow directions of maximum node-to-node coupling [32]. The coupling was taken directly from the discretization and represented the connectivity between nodes based on convection and diffusion.

In this work, a line-creation algorithm is presented similar to that developed by Okusanya, but for elements instead of nodes. The algorithm is shown to yield a unique set of lines on a general irregular mesh for a given flow state, independent of the starting (seed) element used.

The first step in the line generation is the construction of an element connectivity matrix $C(j,k)$, which is a sparse, symmetric matrix that contains information on the strength of the coupling between the elements or, equivalently, between the blocks of the Jacobian matrix. For the Euler equations, coupling is determined by the direction of convection. As such, the inter-element connectivity is taken as the integral of the square of the mass flux over the shared face,

$$C(j,k) = \int_e (\rho\vec{v}\cdot\hat{n})^2 ds.$$

In practice, the matrix $C(j,k)$ is formed during residual calculation when the inter-elemental fluxes are available, resulting in marginal additional cost.

An alternative is to use a $p = 0$ discretization of the scalar transport equation (e.g. entropy) and to base the connectivities on the off-diagonal entries of the Jacobian. This approach is necessary for viscous problems, in which coupling due to diffusive effects must be considered. Oliver *et al* present the details of this method for the Navier-Stokes equations [33].

Given the connectivities in $C(j,k)$, the line creation algorithm is employed. In the following description of the two-stage process, let $N(j;f)$ denote the element adjacent to element j, across face f. In addition, let $F(j)$ denote the set of faces enclosing element j.

Stage I: Line Creation

1. Obtain a seed element i
2. Call MakePath(i) - *Forward Path*
3. Call MakePath(i) - *Backward Path*
4. Return to (1). The algorithm is finished when no more seed elements exist.

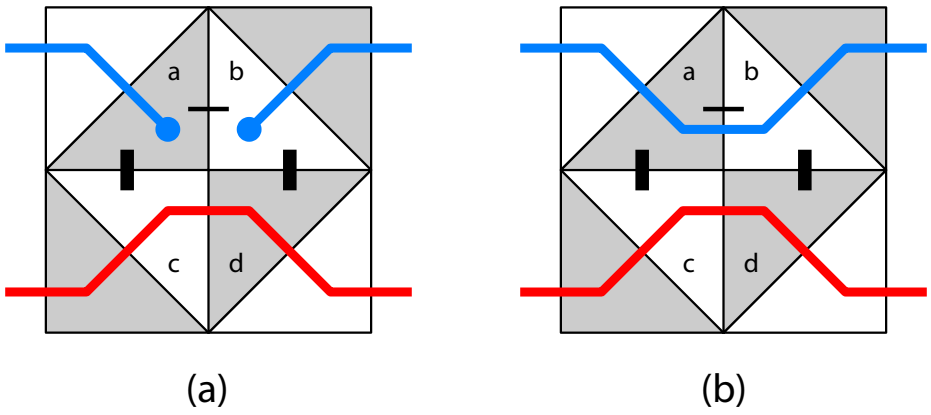

Figure 3-1: Possible line configuration: (a) after Stage I and (b) after Stage II.

MakePath(j)

While path not terminated:

For element j, pick the face $f \in F(j)$ with highest connectivity, such that element $k = N(j; f)$ is not part of the current line. Terminate the path if any of the following conditions hold:

- face f is a boundary face
- element k is already part of a line
- $C(j, k)$ is not one of the top two connectivities in element k

Otherwise, assign element j to the current line, set $j = k$, and continue.

Following Stage I, it is possible that endpoints of two lines are adjacent to each other. Such a configuration is illustrated in Figure 3-1a. In the figure, elements a and b are the endpoints of two different lines. a and b are not connected because $C(a, b)$ is not in the top two connectivities in either element (see Lemma 1), where connectivity is denoted by the thickness of the bar on the shared edge. Also, a is not connected to c because $C(a, c)$ is not in the top two connectivities in c. Similarly for b and d. Since for best performance it is desirable to use lines of maximum length, Stage II is employed to extend the length of the lines created in Stage I.

Stage II: Line Connection

1. Loop through endpoint elements, j, of all lines. Denote by $H_j \subset F(j)$ the set of faces h of j that are boundary faces or that have $N(j; h)$ as a line endpoint.

2. Choose $h \in H_j$ of maximum connectivity. If h is not a boundary face, let $k = N(j; h)$.

3. If k has no other neighboring endpoints of higher connectivity, and no boundary faces of higher connectivity, then connect the two lines to which j and k belong.

Applying Stage II to the example in Figure 3-1a results in the connection of a and b, as shown in Figure 3-1b. Both stages of the algorithm result in unique line decompositions of the domain, independent of the seed elements used. The proof proceeds as follows.

Lemma 1 Following Stage I, a given pair of neighboring elements, j, k, are connected if and only if $C(j,k)$ is one of the top two connectivities in j and in k.

Proof: Assume j and k are connected after Stage I. Without loss of generality, assume that j was visited first in the algorithm. $C(j,k)$ must be in the top two connectivities of j, since only those are considered for the Forward Path or the Backward Path. In addition $C(j,k)$ must be in the top two connectivities of k, since a connection from j to k will not be made if k has two or more higher connectivities.

Conversely, assume that $C(j,k)$ is one of the top two connectivities in j and in k. Again, without loss of generality, assume that j is visited first in the algorithm. Element k will be considered either in the Forward Path or the Backward Path. Element k cannot be part of another line because it has not yet been visited in the algorithm. Since $C(j,k)$ is one of the top two connectivities in k, elements j and k will be connected.

Lemma 2 Two neighboring endpoints, j and k, of distinct lines will be joined in Stage II to form one line if and only if $C(j,k)$ is maximum over the set of faces in H_j and H_k.

Proof: Follows directly by the Stage II algorithm.

Note, in a situation of equal connectivities and ties in which there exists an element that does not have a unique set of top two connectivities, the Lemmas are not well-posed, and the line decomposition may not be unique. However, this situation is not expected to occur for practical problems except possibly immediately after flow initialization.

Theorem 1 For a given domain and connectivity matrix, the line decompositions after Stage I and after Stage II are unique, provided that each element has a unique set of "top two connectivities".

Proof: The uniqueness of the line decomposition after Stage I follows from Lemma 1, since the Lemma provides a rule for connecting a given pair of neighboring elements regardless of the seed elements used for the algorithm. Similarly, the uniqueness after Stage II follows from Lemma 2, since the Lemma dictates whether two line endpoints will be connected regardless of the order in which the endpoints are visited.

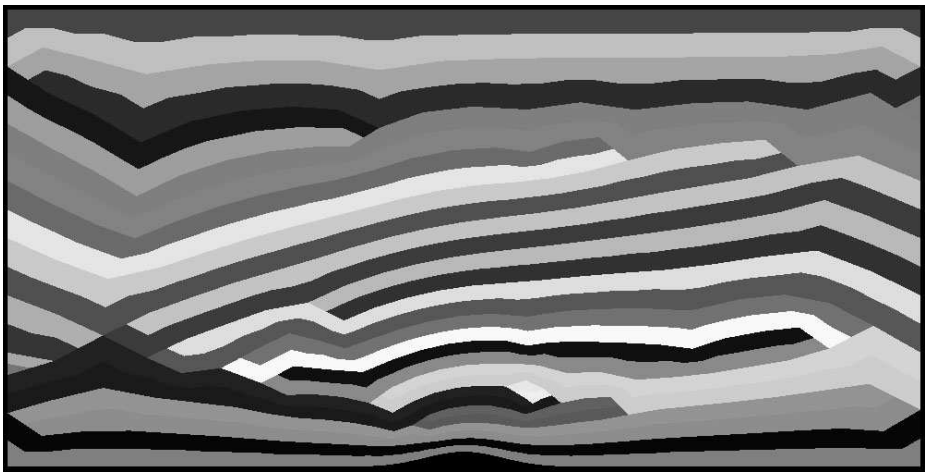

Figure 3-2: Lines in flow over a Gaussian bump.

The lines for two flow cases are shown in Figures 3-2 and 3-3. Figure 3-2 shows the lines for a 2-D duct flow with a Gaussian perturbation on the bottom wall, while Figure 3-3 shows the lines for a 2-D Joukowski airfoil case. Details on these cases are given in Chapter 5.

3.3 Robustness and Under-Relaxation

One of the key goals in designing the solver was robustness. Since the smoother uses a preconditioner based on a linearized form of the governing nonlinear equations, failure can occur if the initial guess is not close to the final solution. In practice, this failure manifests itself through the appearance of non-physical states such as negative density or pressure. To avoid such occurrences, two different strategies are used depending on the degree of nonlinear behavior.

As a first step, the state update is limited through under-relaxation,

$$\mathbf{u}^{n+1} = \mathbf{u}^n + \alpha d\mathbf{u}, \tag{3.3}$$

where $d\mathbf{u}$ is the update obtained from (3.1). The under-relaxation factor is calculated as the greatest α, $0 < \alpha \leq \alpha_0$, that keeps the density and pressure changes under a fraction, η_{max}, of the current values over all the elements. Generally, the same α is used for all elements. In practice, however, a few elements can exist which require a much lower α than the rest of the domain. Using the same small α globally in this case would unnecessarily slow the progress to the solution. To resolve this problem, a minimum global α, α_{min}, is used on all elements except locally for those elements which violate the η_{max} constraint.

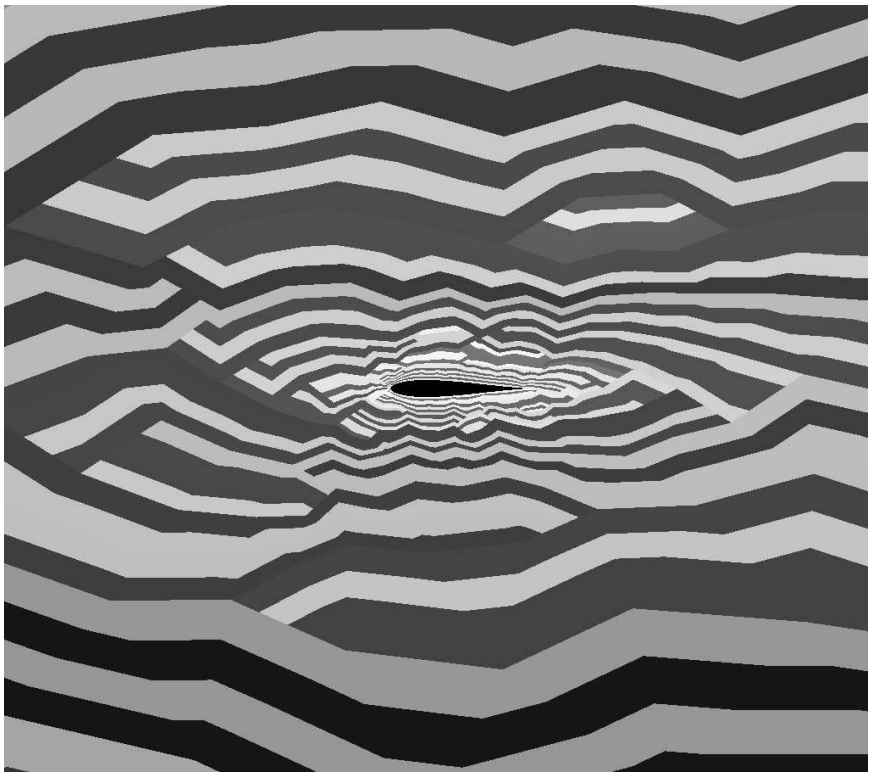

Figure 3-3: Lines in flow over a Joukowski airfoil.

The calculation of α results in a minimal computational overhead relative to the smoother and prevents failure during the initial solution transients for many problems. $\alpha_0 = 0.5$ is used for optimal smoothing of certain high-frequency modes, as demonstrated in Chapter 4. In practice, $\eta_{max} = 0.1$ and $\alpha_{min} = 0.001$ are used.

Even with under-relaxation, however, the solver can fail in the initial steps of a difficult problem through the inability to limit changes in the pressure. Since pressure is a nonlinear function of the conservative state variables, an iterative method is used to determine the α required to keep pressure changes below the factor η_{max}. Specifically, α is first set to the largest value, $0 < \alpha \leq \alpha_0$, that keeps the density changes below the η_{max} factor. If the requested update results in an unallowable pressure change at any Lagrange node point, α is multiplied by $\eta_\alpha < 1$. This check is repeated up to $n_{\alpha,max}$ times, if necessary. If no acceptable α is found after $n_{\alpha,max}$ iterations, the second strategy is employed: on each successive failure of under-relaxation, Δt is lowered by a user-defined factor, η_t, until under-relaxation is applied successfully. If the new Δt does not cause an under-relaxation failure in the next n_{lag} iterations, it is increased by the factor η_t. The process continues until Δt is successively increased back to its original maximum value, Δt_{max}. The typical factors used are $\eta_\alpha = 0.5$, $n_{\alpha,max} = 10$, $\eta_t = 10$, $n_{lag} = 10$, and $\Delta t_{max} = 10^{10} t_{ref}$, where t_{ref} is

a reference time scale of the problem (e.g. $t_{ref} = l_{ref}/V_\infty$, where l_{ref} is a reference length scale).

3.4 p-Multigrid

3.4.1 Motivation

p-multigrid was used in conjunction with the line smoother to increase the performance of the solver. In standard multigrid techniques, solutions on spatially coarser grids are used to correct solutions on the fine grid. This method is motivated by the observation that most smoothers are poor at eliminating low-frequency error modes on the fine grid. However, these low-frequency error modes can be effectively corrected by smoothing on coarser grids, in which these modes appear as high frequency. In p-multigrid, the idea is similar, with the exception that lower-order interpolants serve as the "coarse grids" [37, 17].

p-multigrid fits naturally within the framework of high-order DG discretizations. Additional coarse grid information is not required since the same spatial grid is used by all levels. In addition, a hierarchical basis can be used, eliminating the duplication of state storage at each level. The transfer operators between the grids, prolongation and restriction, are local and only need to be stored for a reference element. These operators become trivial in the case of a hierarchical basis, and they reduce computational time by simplifying the implementation.

3.4.2 FAS and Two-Level Multigrid

To solve the nonlinear system in question, the Full Approximation Scheme (FAS), introduced by Brandt [7], was chosen as the multigrid method. Much of the following description is adapted from Briggs [9].

Consider the discretized system of equations given by

$$\mathbf{R}^p(\mathbf{u}^p) \quad = \quad \mathbf{f}^p,$$

where $\mathbf{u}^p$ is the discrete solution vector for p^{th} order interpolation on a given grid, $\mathbf{R}^p(\mathbf{u}^p)$ is the associated nonlinear system, and $\mathbf{f}^p$ is a source term (zero for the fine-level problem). Let $\mathbf{v}^p$ be an approximation to the solution vector and define the discrete residual, $\mathbf{r}^p(\mathbf{v}^p)$, by

$$\mathbf{r}^p(\mathbf{v}^p) \quad \equiv \quad \mathbf{f}^p - \mathbf{R}^p(\mathbf{v}^p).$$

In a basic two-level multigrid method, the exact solution on a coarse level is used to correct the solution on the fine level. This correction scheme is given as follows:

- Restrict the state and residual to the coarse level: $\mathbf{v}_0^{p-1} = \tilde{I}_p^{p-1}\mathbf{v}^p$, $\mathbf{r}^{p-1} = I_p^{p-1}\mathbf{r}^p$.
- Solve the coarse level problem: $\mathbf{R}^{p-1}(\mathbf{v}^{p-1}) = \mathbf{R}^{p-1}(\mathbf{v}_0^{p-1}) + \mathbf{r}^{p-1}$.
- Prolongate the coarse level error and correct the fine level state: $\mathbf{v}^p = \mathbf{v}^p + I_{p-1}^p(\mathbf{v}^{p-1} - \mathbf{v}_0^{p-1})$.

I_p^{p-1} is the residual restriction operator, and I_{p-1}^p is the state prolongation operator. $\tilde{I}_p^{p-1}$ is the state restriction operator and is not necessarily the same as residual restriction. Alternatively, the FAS coarse level equation can be written as

$$\begin{aligned} \mathbf{R}^{p-1}(\mathbf{v}^{p-1}) &= I_p^{p-1}\mathbf{f}^p + \tau_p^{p-1}, \\ \tau_p^{p-1} &\equiv \mathbf{R}^{p-1}(\tilde{I}_p^{p-1}\mathbf{v}^p) - I_p^{p-1}\mathbf{R}^p(\mathbf{v}^p). \end{aligned}$$

The first equation differs from the original coarse level equation by the presence of the term τ_p^{p-1}, which improves the correction property of the coarse level. In particular, if the fine level residual is zero, the coarse level correction is zero since $\mathbf{v}^{p-1} = \mathbf{v}_0^{p-1}$.

The two-level correction scheme resembles defect correction, in which the solution to a linear or non-linear system is found by iterating with an approximate system that is simpler to solve. Details on defect correction, including convergence analysis, can be found in Désidéri *et al* [14], and the references therein. We present a summary of the method and its relationship to two-level p-multigrid.

Consider a linear system arising from, for example, a second-order finite difference discretization:

$$\mathbf{A}_2(\mathbf{u}) = \mathbf{f}, \tag{3.4}$$

and let $\mathbf{A}_1(\mathbf{u}) = \mathbf{f}$ be a first-order discretization. The defect correction method allows one to obtain successively better solutions to the second-order system by iteratively solving the first-order system. The iteration can be written as

$$\mathbf{A}_1(\mathbf{v}^{m+1}) = \mathbf{f} + \mathbf{A}_1(\mathbf{v}^m) - \mathbf{A}_2(\mathbf{v}^m), \tag{3.5}$$

where $\mathbf{v}^m$ is an approximation to the solution $\mathbf{u}$ at iteration m. At each such iteration, one solves the simpler system of equations, $\mathbf{A}_1$, and only computes the term $\mathbf{A}_2(\mathbf{v}^m)$ for the

second-order system. By inspection, a fixed point of the iteration is a solution to (3.4). The similarity with the two-level correction scheme is evident when the latter is written as

$$\mathbf{R}^{p-1}\left((\mathbf{v}^{p-1})^{m+1}\right) = I_p^{p-1}\mathbf{f}^p + \mathbf{R}^{p-1}\left(\tilde{I}_p^{p-1}(\mathbf{v}^p)^m\right) - I_p^{p-1}\mathbf{R}^p\left((\mathbf{v}^p)^m\right), \quad (3.6)$$

$$(\mathbf{v}^p)^{m+1} = (\mathbf{v}^p)^m + I_{p-1}^p\left((\mathbf{v}^{p-1})^{m+1} - \tilde{I}_p^{p-1}(\mathbf{v}^p)^m\right). \quad (3.7)$$

A key difference between (3.5) and (3.6) is that while in (3.5) the solution vector is the same length for $\mathbf{A}_1$ and $\mathbf{A}_2$, in (3.6) $\mathbf{v}^{p-1}$ and $\mathbf{v}^p$ are of different sizes. Hence, state and residual restriction operators are necessary to transform vectors on level p to level $p-1$. In addition, an extra prolongation step, (3.7), is required to transform the correction from level $p-1$ to level p. Thus, one can view the two-level p-multigrid scheme as defect correction in which the approximate system is of smaller dimension.

3.4.3 V-cycles and FMG

To make multigrid practical, the basic two level correction scheme is extended to a V-cycle and to full multigrid (FMG). In a V-cycle, a sequence of one or more coarse levels is used to correct the solution on the fine level. Descending from the finest level to the coarsest, a certain number of pre-smoothing steps, ν_1, is performed on each level before the problem is restricted to the next coarser level. On the coarsest level, the problem is either solved directly or smoothed a relatively large number of times, ν_c. Ascending back to the finest level, ν_2 post-smoothing steps are performed on each level after prolongation. Each such V-cycle constitutes a multigrid iteration.

Using plain V-cycles to obtain a high-order solution requires starting the smoothing iterations on the highest order approximation. As this level contains the largest number of degrees of freedom, smoothing on it is the most expensive. An alternative is to first obtain an approximation to the solution using the coarser levels before smoothing on the finest level. This is the premise behind FMG in which V-cycles on successively finer levels are used to approximate the solution on the finest level. By the time the solution is prolongated to the finest level, it is usually a close approximation to the final solution with the exception of certain high frequency errors that can be smoothed efficiently on that level. In an effective multigrid scheme - one in which the smoother, transfer operators, and coarse level approximation spaces are well matched - FMG should require only a few V-cycles on each level before prolongating to the next finer level [7]. In practice, this behavior can be tested by using a known output to track the error at each multigrid iteration.

A decision that has to be made in the FMG algorithm is when to start iterating on

the next finer level. Converging the solution fully on each level is not practical because the discretization error on the coarser levels is usually well above machine zero. Although one can perform a constant number of V-cycles on each level, an alternative is to prolongate when a residual-based criterion is met. The criterion used is described as follows. At the end of a V-cycle, the current residual and its L_1 norm, $|r^p|_{L_1}$, are known. The solution vector is prolongated to the $p+1$ level and the residual, r^{p+1}, is calculated along with its L_1 norm $|r^{p+1}|_{L_1}$. Iteration on the next finer level commences when $|r^p|_{L_1} < \eta_r |r^{p+1}|_{L_1}$. If this condition is not met, another V-cycle at level p is carried out. The extra prolongation and computation of r^{p+1} at the end of each V-cycle add slightly to the computational cost. However, the benefit is that low-order approximations are not unnecessarily converged when a high-order solution is desired. In practice $\eta_r = 0.5$ is used.

3.4.4 Operator Definition

The transfer operators used in the multigrid scheme are now defined. Let Ω denote the entire domain, and let ϕ_i^p denote the i^{th} basis function of order p in a global ordering over all the basis functions in the discretization. Since the approximation spaces are nested, ϕ_i^{p-1} can be expressed in terms of ϕ_j^p,

$$\phi_i^{p-1} = \sum_j \alpha_{ij}^{p-1} \phi_j^p. \tag{3.8}$$

The prolongation operator, I_{p-1}^p, transfers changes in $\mathbf{v}^{p-1}$ (in V-cycle multigrid) and the solution itself (in FMG when moving to a higher level) to level p. Thus, a fine-level representation is required of a coarse-level approximation. That is, we wish to calculate $\mathbf{v}^p$, whose components are given by

$$v_j^p = \sum_i \left(I_{p-1}^p \right)_{ji} v_i^{p-1}, \tag{3.9}$$

such that, for all points in the domain,

$$\sum_j v_j^p \phi_j^p \;\; = \;\; \sum_i v_i^{p-1} \phi_i^{p-1}. \tag{3.10}$$

In (3.10), (3.9) is used to substitute for v_j^p and (3.8) to substitute for ϕ_i^{p-1}:

$$\sum_j \sum_i \left(I_{p-1}^p \right)_{ji} v_i^{p-1} \phi_j^p \;\; = \;\; \sum_j \sum_i \alpha_{ij}^{p-1} v_i^{p-1} \phi_j^p.$$

Since a state representation is unique in the basis ϕ_j^p, it follows that

$$\left(I_{p-1}^p\right)_{ji} = \alpha_{ij}^{p-1} \Rightarrow I_{p-1}^p = (\alpha^{p-1})^T. \tag{3.11}$$

To form the residual restriction operator, I_p^{p-1}, the definition of the residual vector is used,

$$R_j^p(\mathbf{v}^p) = B(\phi_j^p, \mathbf{v}^p).$$

Given $\mathbf{R}^p(\mathbf{v}^p)$ we wish to determine $\mathbf{R}^{p-1}(\mathbf{v}^p) = I_p^{p-1}\mathbf{R}^p(\mathbf{v}^p)$. Using the linearity of B in the first input and (3.8), the components $R_i^{p-1}(\mathbf{v}^p)$ can be written as

$$\begin{aligned} R_i^{p-1}(\mathbf{v}^p) &= B(\phi_i^{p-1}, \mathbf{v}^p) \\ &= \sum_j \alpha_{i,j} B(\phi_j^p, \mathbf{v}^p) \\ &= \sum_j \alpha_{i,j} R_j^p. \end{aligned}$$

Thus, the residual restriction operator is

$$I_p^{p-1} = \alpha^{p-1}. \tag{3.12}$$

Finally, the state restriction operator, which is used to transfer $\mathbf{v}^p$ to $\mathbf{v}^{p-1}$ via $\mathbf{v}^{p-1} = \tilde{I}_p^{p-1}\mathbf{v}^p$, is determined by enforcing state equality between the coarse and fine levels in a weak form. Specifically, we seek $\mathbf{v}^{p-1} \in \mathcal{V}_h^{p-1}$ such that

$$\int_\Omega \mathbf{w}^{p-1}\mathbf{v}^{p-1} d\Omega = \int_\Omega \mathbf{w}^{p-1}\mathbf{v}^p d\Omega, \qquad \forall \mathbf{w}^{p-1} \in V_h^{p-1}. \tag{3.13}$$

Using the basis ϕ^{p-1} for $\mathcal{V}_h^{p-1}$ and ϕ^p for $\mathcal{V}_h^p$, (3.13) is equivalent to

$$\begin{aligned} \int_\Omega \phi_k^{p-1} \sum_i \mathbf{v}_i^{p-1}\phi_i^{p-1} d\Omega &= \int_\Omega \phi_k^{p-1} \sum_j \mathbf{v}_j^p \phi_j^p d\Omega \\ \sum_i \mathcal{M}_{k,i}^{p-1}\mathbf{v}_i^{p-1} &= \sum_j \mathcal{N}_{k,j}^{p-1}\mathbf{v}_j^p \\ \mathbf{v}_i^{p-1} &= (\mathcal{M}^{p-1})^{-1}\mathcal{N}^{p-1}\mathbf{v}_j^p \\ \tilde{I}_p^{p-1} &= (\mathcal{M}^{p-1})^{-1}\mathcal{N}^{p-1}, \end{aligned} \tag{3.14}$$

$$\mathcal{M}_{k,i}^{p-1} = \int_\Omega \phi_k^{p-1}\phi_i^{p-1} d\Omega, \qquad \mathcal{N}_{k,j}^{p-1} = \int_\Omega \phi_k^{p-1}\phi_j^p d\Omega.$$

Although the operators have been defined in a global sense, the local compact support for the basis functions allows these operators to be calculated on a reference element and to be applied element-wise throughout the domain. Since the transfer operators are applied in the reference element, no special consideration is necessary for curved elements.

The operators presented are sparse globally, but take on the form of possibly dense matrices locally on each element. Using a Lagrange basis, the local operators are dense matrices. However, using a hierarchical basis, I_{p-1}^{p} is the identity matrix with zero rows appended, and I_{p}^{p-1} is the identity matrix with zero columns appended. The state restriction operator, $\tilde{I}_{p}^{p-1}$, consists of the identity matrix with non-zero columns appended. These non-zero columns result from the calculation of $(\mathcal{M}^{p-1})^{-1}\mathcal{N}^{p-1}$ and represent the coupling between the p and $p-1$ order basis functions. If an orthogonal basis were used, these columns would consist of zeros.

3.5 Storage and Implementation

The greatest storage requirement comes from the line preconditioner, which is essentially equal in size to the full Jacobian. The benefit of storing the full Jacobian is that doing so allows multiple linear iterations per one inversion of the diagonal blocks. We have found that linear iterations benefit overall computational time; however, storage of the full Jacobian leads to excessive memory requirements for problems in which the element count and interpolation order are large. Hence, a memory-lean version of the line solver was written in which the Jacobian is stored only for one line of elements at a time. Table 3.1 lists the memory requirements for the lean and non-lean implementations in 2-D and 3-D. The elements are assumed to be triangles in 2-D and tetrahedra in 3-D. The storage for the lean Jacobian depends on the length of the longest line (i.e. grid geometry and flow direction), and is therefore only approximate.

Table 3.1: Approximate storage requirements per equation in 2-D and 3-D for N elements.

	Per-element	Solution	Non-lean $\frac{\partial \mathbf{R}}{\partial \mathbf{u}}$	Lean $\frac{\partial \mathbf{R}}{\partial \mathbf{u}}$
2-D	$n(p) = \frac{(p+1)(p+2)}{2}$	$Nn(p)$	$3N[n(p)]^2$	$\approx 3N^{1/2}[n(p)]^2$
3-D	$n(p) = \frac{(p+1)(p+2)(p+3)}{6}$	$Nn(p)$	$4N[n(p)]^2$	$\approx 4N^{1/3}[n(p)]^2$

In addition to the memory savings of the lean line solver, the state updates obtained for each line can be applied as each line is processed, resulting in a Gauss-Seidel type iterative

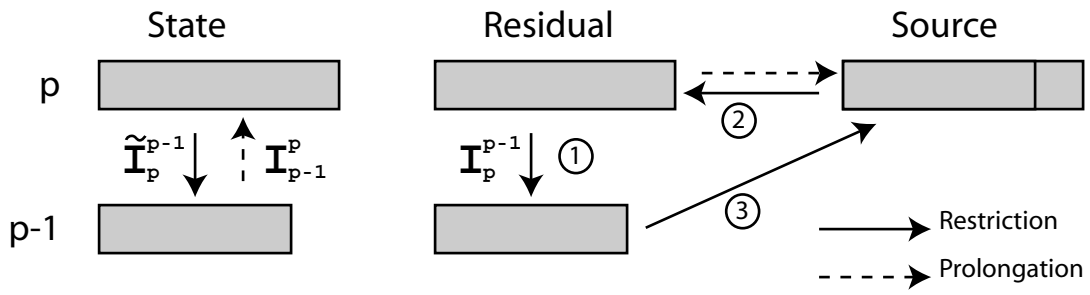

Figure 3-4: Diagram of storage in the multigrid algorithm.

scheme. This method was implemented and showed slightly faster convergence rates for the cases tested in Chapter 5.

Storage is also one consideration in the implementation of the multigrid scheme. For each level, state, residual, and source vectors were defined. However, since the residual data can be overwritten, it is not necessary to allocate three separate vectors for each level. Figure 3-4 shows the implementation used for a general basis.

As shown, only one source vector of adequate size is allocated. During restriction, the state vector is transferred directly via $\mathbf{v}^{p-1} = \tilde{I}_p^{p-1}\mathbf{v}^p$. Three transfers then take place involving the residual and source terms. First, $\mathbf{R}^p$ is restricted via $\mathbf{R}^{p-1} = I_p^{p-1}\mathbf{R}^p$. Second, the source term used by level p is stored in the level p residual vector. Finally, the coarse level residual is transferred to the source term: $\mathbf{f}^{p-1} = \mathbf{R}^{p-1}$. Analogous steps are taken when transferring to the next coarser level. During prolongation, the state $\mathbf{v}^p$ is corrected by interpolating the difference between the coarse level solution and the restriction of the fine level solution. In addition, the source term is restored from the residual vector. Since prolongation introduces a correction to the solution, the update goes through the standard under-relaxation process.

Chapter 4

Stability Analysis

The objective of this chapter is to determine the stability of block-implicit and line-implicit relaxation applied to DG for convection-dominated flows. To this end, Fourier analysis is performed on the advection problem in one and two dimensions on simple domains with periodic boundary conditions. The footprints of the relaxation operators are calculated and analyzed for stability.

The advection problem in one and two dimensions is given by

$$\begin{aligned} \vec{V} \cdot \nabla u &= f(\vec{x}), \\ a u_x &= f(x) \qquad (1-D), \\ a u_x + b u_y &= f(x,y) \quad (2-D). \end{aligned} \tag{4.1}$$

In this problem, the velocity $\vec{V}$ is constant, u is the unknown concentration variable, and f is a source function. The problem is defined on the interval $[-1,1]$ ($[-1,1] \times [-1,1]$ in 2-D) with periodic boundary conditions.

4.1 One-dimensional Analysis

In 1-D, the [-1,1] interval is discretized into N elements, κ_r, $r = 1 \ldots N$, each of size $h = 2/N$, where N is an even integer. Within each element, $(p+1)$ basis functions are defined, where p is the interpolation order. Since each basis function has local support on only one element, the total number of unknowns is $N(p+1)$. For similarity with the Euler equations, a concentration flux is defined by $\mathcal{F}(u) \equiv au$. Letting $\mathcal{V}_h^p$ be the space of discontinuous polynomials of degree p on the given subdivision, the DG discretization of

(4.1) reads: find $u_h \in \mathcal{V}_h^p$ such that $\forall v_h \in \mathcal{V}_h^p$,

$$B(v_h, u_h) \equiv \mathcal{H}(u_h)v_h|_{\kappa_R} - \mathcal{H}(u_h)v_h|_{\kappa_L} - \int_\kappa \mathcal{F}(u_h)v_{h,x}dx = \int_\kappa v_h f(x)dx. \tag{4.2}$$

κ_R and κ_L denote the right and left element boundaries, and full-upwinding is used for the inter-element flux:

$$\mathcal{H}(u_h) = \frac{a}{2}\left(u_{h,L} + u_{h,R}\right) - \frac{|a|}{2}\left(u_{h,R} - u_{h,L}\right), \tag{4.3}$$

where $u_{h,L}$ and $u_{h,R}$ refer to values of u_h taken from the left and right elements at an interface. Periodic boundary conditions are enforced by identifying the left boundary of the first element with the right boundary of the last element. Using a basis, $\{\phi_k\}$, for $\mathcal{V}_h^p$, u_h can be represented as $u_h(x) = \sum_k u_k\phi_k(x)$. (4.2) can then be written compactly as $\mathbf{Au} = \mathbf{f}$, where $\mathbf{u}$ is the discrete vector of the u_k, and $\mathbf{f}$ is the discrete source vector with components $f_k = \int_\kappa \phi_k f(x)dx$.

For an elemental block Jacobi smoother, the linear system is separated into $\mathbf{A} = \mathbf{M} - \mathbf{N}$, where $\mathbf{M}$ is the elemental block diagonal component of $\mathbf{A}$. With reference to the basic iterative scheme (3.1), $\mathbf{M}$ acts as the preconditioner $\mathbf{P}$. Fourier (Von Neumann) analysis is used to determine the relaxation footprint, which consists of the eigenvalues of $-\mathbf{M}^{-1}\mathbf{A} = \mathbf{M}^{-1}\mathbf{N} - \mathbf{I} = \mathbf{S} - \mathbf{I}$, where $\mathbf{S} \equiv \mathbf{M}^{-1}\mathbf{N}$. Let $\mathbf{u}$ denote the exact discrete solution vector and $\mathbf{v}^m$ an approximation at the m^{th} iteration. Defining $\mathbf{e}^m \equiv \mathbf{v}^m - \mathbf{u}$, the basic iterative step can be written as

$$\mathbf{e}^{m+1} = \mathbf{S}\mathbf{e}^m. \tag{4.4}$$

Because of the periodic boundary conditions, the eigenvectors of $\mathbf{S}$ are sinusoidal on the elements, indexed by r. Thus, the error eigenvectors take on the form

$$\mathbf{e}^m(\theta_j) = \begin{bmatrix} \bar{\mathbf{e}}_1^m(\theta_j) \\ \vdots \\ \bar{\mathbf{e}}_r^m(\theta_j) \\ \vdots \\ \bar{\mathbf{e}}_N^m(\theta_j) \end{bmatrix}, \qquad \bar{\mathbf{e}}_r^m(\theta_j) \;=\; \bar{\mathbf{v}}^m(\theta_j)e^{ir\theta_j}, \tag{4.5}$$

where $\theta_j = j\pi h$ is the mode of the eigenvector, with $j \in \{-N/2+1, \ldots, N/2\}$. $\bar{\mathbf{e}}_r^m$ represents a vector of size $(p+1)$ corresponding to the portion of the error eigenvector on element r.

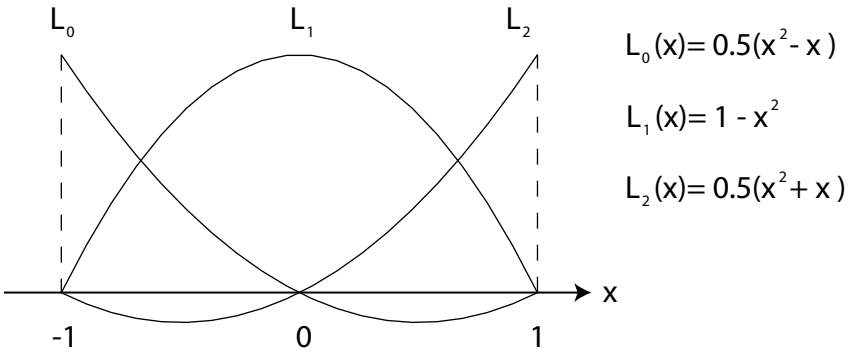

Figure 4-1: 1-D Lagrange basis functions on the reference element for $p = 2$.

Substituting (4.5) into (4.4) yields an expression for the relaxation of the θ_j eigenvector:

$$\bar{\mathbf{e}}_r^m \quad = \quad \widetilde{\mathbf{S}}^m(\theta_j)\bar{\mathbf{v}}^0 e^{ir\theta_j}. \tag{4.6}$$

$\widetilde{\mathbf{S}}(\theta_j)$ is a $(p+1)\times(p+1)$ matrix corresponding to $\mathbf{S}$ for error modes that are sinusoidal on the elements. For stability, the $(p+1)$ eigenvalues of $\widetilde{\mathbf{S}}(\theta_j)$ must be less than 1 in absolute value. For this 1-D problem, the eigenvalues and eigenvectors can be computed analytically, with the result that the elemental block Jacobi scheme is stable independent of order. The derivation of this result is given for the Lagrange basis, which is depicted in Figure 4-1 for $p = 2$.

The analytical result is that for any order p, the only nonzero eigenvalues of $\widetilde{\mathbf{S}}$ are $\lambda = e^{\pm i\theta_j}$. The derivation begins by expressing the error equation, $\mathbf{A}\mathbf{e}^m = \mathbf{r}^m \equiv \mathbf{f} - \mathbf{A}\mathbf{v}^m$, in stencil form. Specifically, for any element r, the error equation can be written as

$$\widehat{\mathbf{A}}^L\bar{\mathbf{e}}_{r-1}^m + \widehat{\mathbf{A}}^0\bar{\mathbf{e}}_r^m + \widehat{\mathbf{A}}^R\bar{\mathbf{e}}_{r+1}^m \quad = \quad \bar{\mathbf{r}}_r^m, \tag{4.7}$$

where if $r = 1$, $r-1$ refers to element N, and if $r = N$, $r+1$ refers to element 1. $\bar{\mathbf{r}}_r^m$ is the residual vector on element r. The $(p+1)\times(p+1)$ matrices $\widehat{\mathbf{A}}^L$, $\widehat{\mathbf{A}}^0$, and $\widehat{\mathbf{A}}^R$ are obtained from (4.2) and (4.3). They are listed here for $p = 2$:

$$\widehat{\mathbf{A}}^L = \frac{1}{2}\begin{pmatrix} 0 & 0 & -(|a|+a) \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad \widehat{\mathbf{A}}^R = \frac{1}{2}\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -(|a|-a) & 0 & 0 \end{pmatrix}, \tag{4.8}$$

$$\widehat{\mathbf{A}}^0 = \frac{1}{2}\begin{pmatrix} |a| & \frac{4}{3}a & -\frac{1}{3}a \\ -\frac{4}{3}a & 0 & \frac{4}{3}a \\ \frac{1}{3}a & -\frac{4}{3}a & |a| \end{pmatrix}.$$

Substituting the sinusoidal error form, (4.5), into (4.7) yields

$$\begin{aligned} \widehat{\mathbf{A}}^L \bar{\mathbf{e}}_r^m e^{-i\theta_j} + \widehat{\mathbf{A}}^0 \bar{\mathbf{e}}_r^m + \widehat{\mathbf{A}}^R \bar{\mathbf{e}}_r^m e^{i\theta_j} &= \bar{\mathbf{r}}_r^m \\ (\widehat{\mathbf{A}}^L e^{-i\theta_j} + \widehat{\mathbf{A}}^0 + \widehat{\mathbf{A}}^R e^{i\theta_j}) \bar{\mathbf{e}}_r^m &= \bar{\mathbf{r}}_r^m \end{aligned}$$

The off-block-diagonal terms in the original system $\mathbf{A}$ are associated with $\widehat{\mathbf{A}}^L$ and $\widehat{\mathbf{A}}^R$ in the stencil representation. Thus, for block-Jacobi smoothing, the elemental matrices $\widetilde{\mathbf{M}}$ and $\widetilde{\mathbf{N}}$ corresponding to $\mathbf{M}$ and $\mathbf{N}$ in the case of sinusoidal error modes are

$$\widetilde{\mathbf{M}} = \frac{1}{2}\begin{pmatrix} |a| & \frac{4}{3}a & -\frac{1}{3}a \\ -\frac{4}{3}a & 0 & \frac{4}{3}a \\ \frac{1}{3}a & -\frac{4}{3}a & |a| \end{pmatrix}, \qquad \widetilde{\mathbf{N}} = \frac{1}{2}\begin{pmatrix} 0 & 0 & (|a|+a)e^{-i\theta_j} \\ 0 & 0 & 0 \\ (|a|-a)e^{i\theta_j} & 0 & 0 \end{pmatrix}.$$

From the form of $\widetilde{\mathbf{N}}$, the product $\widetilde{\mathbf{S}} = \widetilde{\mathbf{M}}^{-1}\widetilde{\mathbf{N}}$ results in the following column representation of $\widetilde{\mathbf{S}}$:

$$\widetilde{\mathbf{S}} = [(|a|-a)\mathbf{s}_0, 0, \ldots, 0, (|a|+a)\mathbf{s}_p], \tag{4.9}$$

where $\mathbf{s}_0$ and $\mathbf{s}_p$ are nonzero columns. Thus, the interior and upwind basis functions within each element are eigenvectors of $\widetilde{\mathbf{S}}$, with eigenvalue 0. For a given θ_j, the eigenvector corresponding to the nonzero eigenvalue turns out to be $\bar{\mathbf{v}} = [1, 1, \ldots, 1]^T$. This statement is verified by substituting this $\bar{\mathbf{v}}$ into the eigenvalue problem:

$$\begin{aligned} \widetilde{\mathbf{M}}^{-1}\widetilde{\mathbf{N}}\bar{\mathbf{v}} &= \lambda\bar{\mathbf{v}} \\ \widetilde{\mathbf{N}}\bar{\mathbf{v}} &= \lambda\widetilde{\mathbf{M}}\bar{\mathbf{v}} \end{aligned}$$

$$\widetilde{\mathbf{N}}\bar{\mathbf{v}} = \frac{1}{2}\begin{pmatrix} (|a|+a)e^{-i\theta_j} \\ 0 \\ \vdots \\ 0 \\ (|a|-a)e^{i\theta_j} \end{pmatrix}, \qquad \widetilde{\mathbf{M}}\bar{\mathbf{v}} = \frac{1}{2}\begin{pmatrix} (|a|+a) \\ 0 \\ \vdots \\ 0 \\ (|a|-a) \end{pmatrix}.$$

By inspection, the eigenvalues are $\lambda = e^{-i\theta_j}$ for $a > 0$, and $\lambda = e^{i\theta_j}$ for $a < 0$. Hence, for each θ_j, the only nonzero eigenvalue of $\widetilde{\mathbf{S}}$ lies on the complex unit circle independent

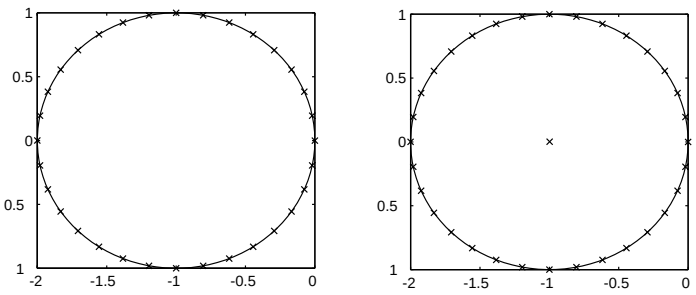

Figure 4-2: Smoothing footprint for (a) $p = 0$ and (b) $p > 0$.

of p. The footprint of the relaxation operator is readily obtained from the eigenvalues of $\widetilde{\mathbf{S}}$ by $\lambda(-\widetilde{\mathbf{M}}^{-1}\widetilde{\mathbf{A}}) = \lambda(\widetilde{\mathbf{S}}) - 1$. Figure 4-2 shows the footprints of the $p = 0$ and $p > 0$ preconditioned operators.

For $p = 0$, the eigenvalues are identical to those obtained from the traditional upwind finite-difference scheme, as the schemes are identical. Under-relaxation, introduced in Section 3.3, places the eigenvalues associated with the high-frequency modes close to -1 when $\alpha = 0.5$, so that the smoother is effective at eliminating these modes. For higher order, the additional eigenvalues all lie at -1, implying that the smoother will converge at a rate that is independent of the order and guaranteeing p-independent multigrid as well.

4.2 Two-dimensional Analysis

In 2-D, the area $[-1, 1] \times [-1, 1]$ is discretized into N^2 square elements. For the basis, the tensor product of the 1-D Lagrange basis is used, so that $\phi_{\alpha\beta}(x, y) = \phi_\alpha(x)\phi_\beta(y)$, where α and β are local element indices. Discretizing the equations as in 1-D results in the system $\mathbf{Au} = \mathbf{f}$.

With periodic boundary conditions, Fourier stability analysis is carried out as in 1-D. In the 2-D case, the eigenvectors of $\mathbf{S}$ are characterized by two modes, $(\theta_j, \theta_k) = (j\pi h, k\pi h)$, where $j, k \in \{-N/2+1, \ldots, N/2\}$ and $h = 2/N$. Indexing the elements by the ordered pair of integers (r, s), the smoothing operator eigenvectors can be represented as

$$\bar{\mathbf{e}}^m_{rs}(\theta_j, \theta_k) \quad = \quad \bar{\mathbf{v}}^m(\theta_j, \theta_k)e^{ir\theta_j + is\theta_k}, \tag{4.10}$$

where $\bar{\mathbf{e}}^m_{rs}$ and $\bar{\mathbf{v}}^m$ are now vectors of size $(p+1)^2$. Substituting for $\mathbf{e}$ in the basic iterative method yields the following expression for the error at the m^{th} iteration:

$$\bar{\mathbf{e}}^m_{rs}(\theta_j, \theta_k) \quad = \quad \widetilde{\mathbf{S}}^m(\theta_j, \theta_k)\bar{\mathbf{v}}^0(\theta_j, \theta_k)e^{ir\theta_j + is\theta_k}. \tag{4.11}$$

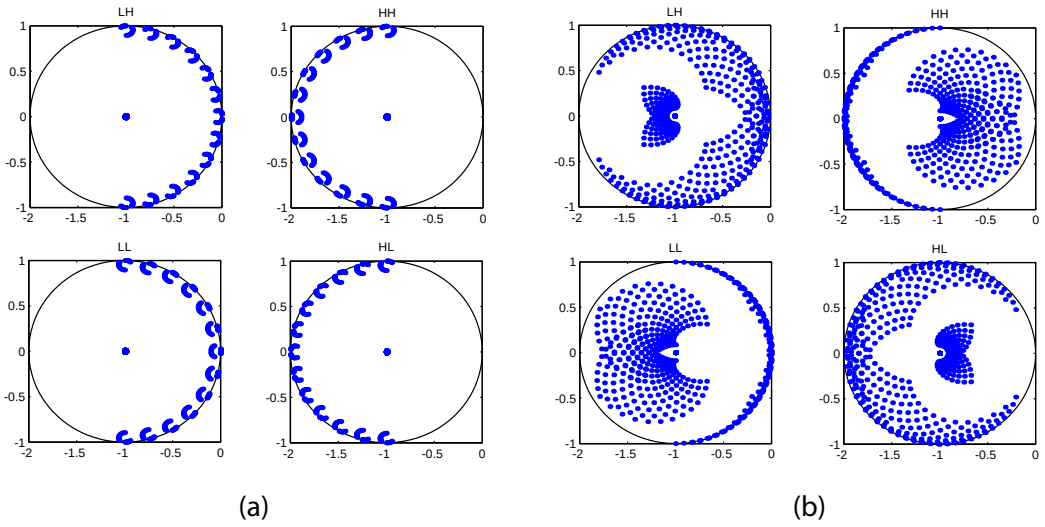

Figure 4-3: Block Jacobi footprint for $p = 1$, $N = 24$: (a) $\alpha = 1^o$, (b) $\alpha = 25^o$.

$\widetilde{\mathbf{S}}(\theta_j, \theta_k)$ is now a $(p+1)^2 \times (p+1)^2$ iteration matrix. While the eigenvalues of $\widetilde{\mathbf{S}}$ cannot be calculated analytically, numerical results show that $|\rho(\widetilde{\mathbf{S}})| \leq 1$ for all flow angles $\alpha = \tan^{-1}(b/a)$, for both block and line preconditioners. For multigrid studies, it is useful to differentiate between low (L) and high (H) frequency modes. L modes are defined to be those with $-\pi/2 < \theta \leq \pi/2$. All other values of θ correspond to H modes. This separation is ideal for h-multigrid in which the grid size is halved on each finer grid. For p-multigrid, the ideal separation is not clear, but the stated separation is used for simplicity. With this distinction, the eigenvalues are separated based on the mode pair (θ_j, θ_k) of the eigenvectors: LL, LH, HL, or HH. Figure 4-3 shows this separation for the cases of $\alpha = 1^o$ and $\alpha = 25^o$ using the block Jacobi smoother.

For $\alpha = 1^o$ (near horizontal flow), the eigenvalues are close to those for the one-dimensional case. The modes least affected by under-relaxed Jacobi smoothing are those with footprint eigenvalues closest to 0 in the complex unit disk. As expected, for both angles, the HH modes are effectively reduced by the iterative method. The LL modes are densely clustered near 0 for both, and, as in 1D, some form of multigrid is required to reduce these errors. For $\alpha = 1^o$ and somewhat for $\alpha = 25^o$, the HL modes are effectively reduced by the smoother, but the LH modes are not reduced, a consequence of the flow being aligned more in the x-direction. Since the LH modes contain high frequency components, they cannot be represented on uniformly-coarsened grids to be affected by multigrid. The presence of these errors stalls the convergence and degrades the performance of multigrid with block-Jacobi smoothing.

A line smoother is capable of reducing the "LH" modes - i.e. the modes that are

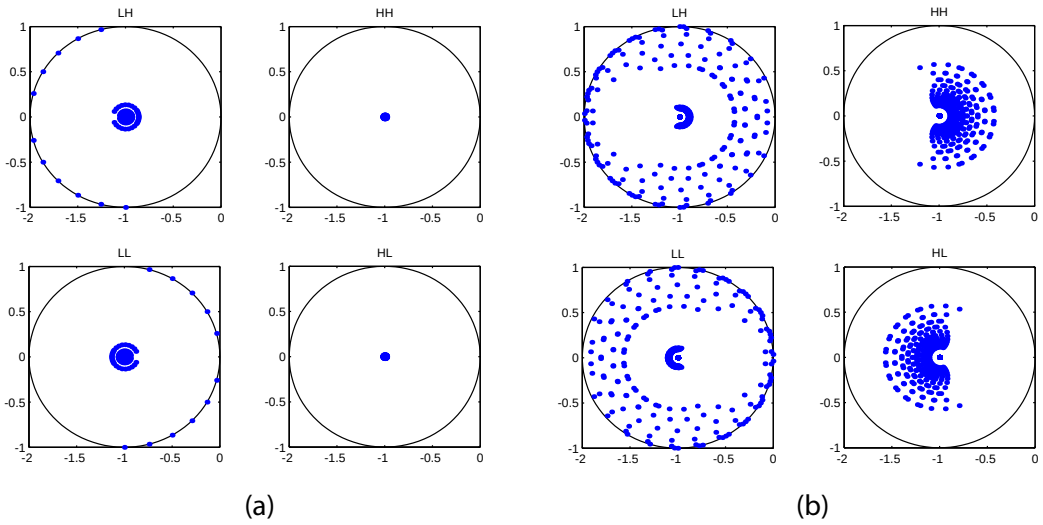

Figure 4-4: Line Jacobi footprint for $p = 1$, $N = 24$: (a) $\alpha = 1^o$, (b) $\alpha = 25^o$.

low frequency in the flow direction and high frequency in the transverse direction. In the analysis, the line updates are performed in Jacobi fashion in that the unknown values for all elements not on the line are held constant when calculating the update. Line smoothing is most effective in cases where the lines are aligned with the flow direction. Figure 4-4 shows the footprint of line smoothing for the cases of $\alpha = 1^o$ and $\alpha = 25^o$, using horizontal (x-aligned) lines. For $\alpha = 1^o$, the line smoother almost completely eliminates the HH and HL modes. In addition, under-relaxed line smoothing eliminates the LH modes for this case. For $\alpha = 25^o$, line Jacobi is more effective at reducing the HH, HL, and LH modes in comparison to block Jacobi. However, smoothing of the LH modes still results in some eigenvalues close to 0, a consequence of the 25^o difference between the flow angle and the line angle. Alignment of lines with the flow direction is therefore crucial for good multigrid performance of the line smoother.

Chapter 5

Results

This chapter presents accuracy and solver performance results for various smooth problems in two and three dimensions. For each problem, uniform grid refinement was performed to study the accuracy of the discretization. Performance and convergence rate were determined by timing the full multigrid scheme. The following parameters were used in all the cases:

- - Hierarchical basis
- - $\nu_1 = 4$ pre-smoothing sweeps and $\nu_2 = 4$ post-smoothing sweeps
- - $\nu_c = 100$ sweeps on the coarsest level, $p = 0$
- - Memory-lean line solver
- - Residual-based level switching criterion for FMG
- - Initialization with a converged solution on $p = 0$

All timing runs were performed on an Intel Pentium 4 2.53 GHz system with 512 MB RAM.

5.1 Two-dimensional Results

Three problems were studied in two dimensions: Ringleb flow, flow over a Gaussian bump, and flow over a Joukowski airfoil. In the following sections each problem is described and the accuracy and performance results are presented.

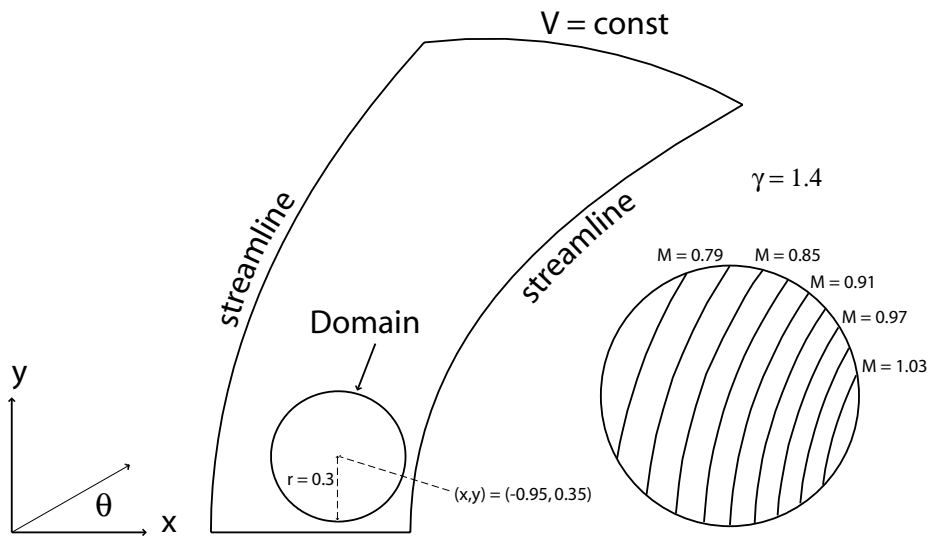

Figure 5-1: Description of Ringleb flow.

5.1.1 Ringleb Flow

Ringleb flow is an exact solution of the Euler equations obtained using the hodograph method. The streamlines and iso-Mach lines for a typical Ringleb solution domain are shown in Figure 5-1.

The relevant transformation equations between the Cartesian variables (x, y) and the hodograph variables (V, θ) are

$$\begin{aligned}
\Psi &= \frac{1}{V}\sin(\theta), \\
c^2 &= 1 - \frac{\gamma - 1}{2}V^2, \\
J &= \frac{1}{c} + \frac{1}{3c^3} + \frac{1}{5c^5} - \frac{1}{2}\log\frac{1+c}{1-c}, \\
\rho &= c^{2/(\gamma-1)}, \\
x &= \frac{1}{2\rho}\left[\frac{1}{V^2} - 2\Psi^2\right] + \frac{J}{2}, \\
y &= \pm\frac{\Psi}{\rho V}\cos(\theta).
\end{aligned}$$

Since the exact flow state can be determined for any (x, y), the domain is taken to be a circle inside the regular Ringleb domain, as shown in Figure 5-1. The boundary condition is imposed by setting the exact state on the exterior of the domain, and using the Riemann approximate flux function. An accuracy study was performed using a set of three hierarchical grids (88, 352, and 1408 elements). Orders of interpolation ranging from $p = 0$ to $p = 3$ were used, and the output of interest was the L_2 norm of the error. Each case was

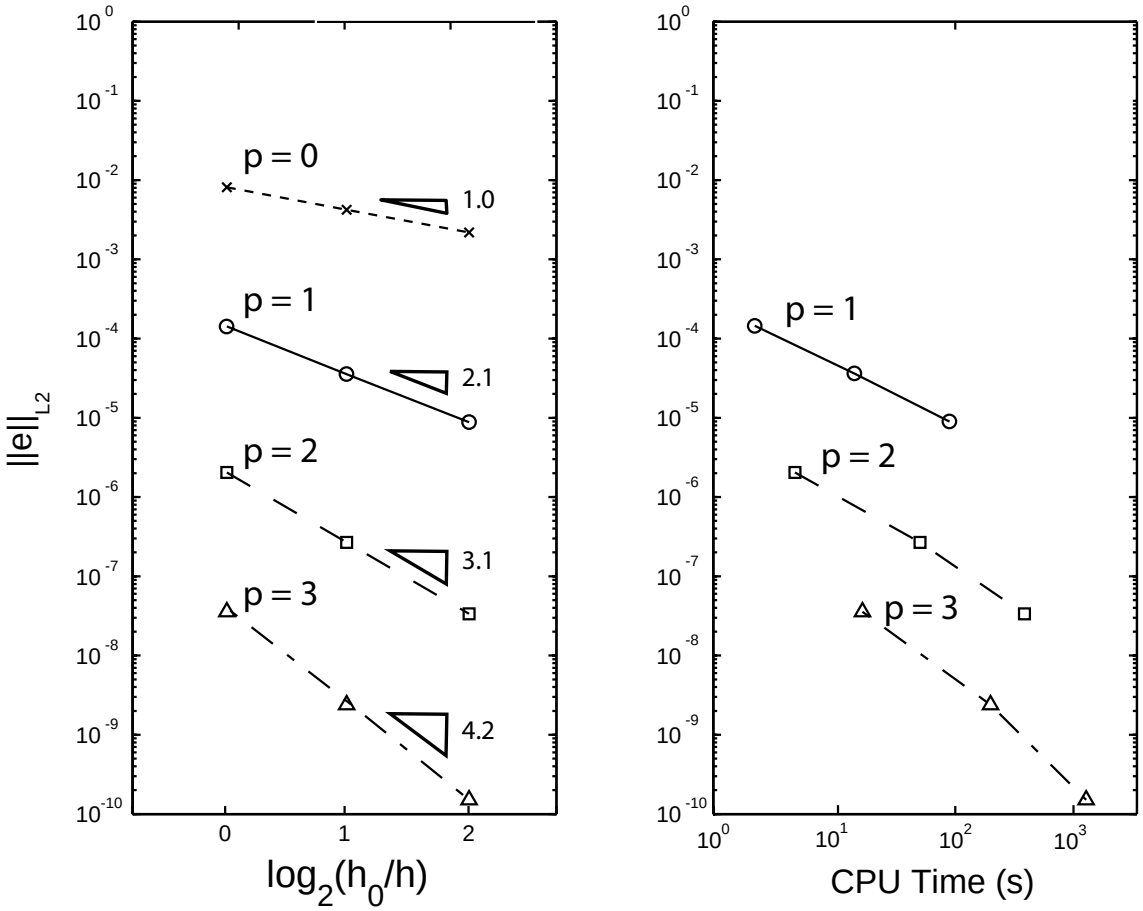

Figure 5-2: Ringleb flow: accuracy vs. CPU time.

converged to machine zero residual for the $p = 3$ discretization.

Figure 5-2 shows the solution accuracy versus grid size and order. Optimal accuracy convergence of $p + 1$ is attained, in that $||e||_{L_2} = Ch^{p+1}$. Figure 5-2 also shows the error plotted versus CPU time to solution. A solution was taken to be converged when the error norm came within 1 percent of its final value, determined by converging the solution to machine zero residual beforehand. The advantage of high order interpolation is clear: a $p = 3$ solution on the coarsest grid yields the same accuracy as a $p = 2$ solution on a grid 16 times the size in a time of 17 seconds as compared to 352 seconds.

The error and residual convergence histories for $p = 3$ FMG solutions on each grid are given in Figure 5-3. Grid dependence is evident, showing one of the drawbacks of the solution algorithm. However, order independence can be seen from the asymptotic multigrid rates shown in Figure 5-4a. These rates were obtained by converging each level of FMG to machine zero residual. An analogous plot of the L_2 error norm is given in Figure 5-4b,

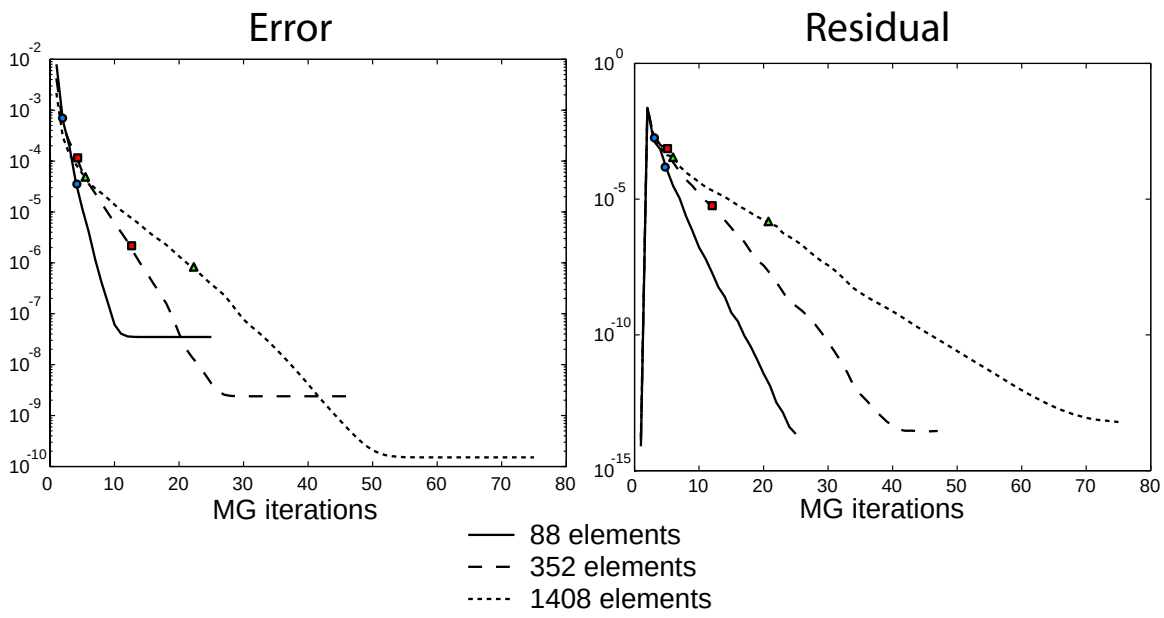

Figure 5-3: Ringleb flow: FMG convergence history.

showing that error truncation is reached well before the residual reaches machine zero.

5.1.2 Flow over a Gaussian Bump

The second test problem is that of channel flow over a Gaussian bump. The problem setup is depicted in Figure 5-5. The channel height is 12σ, the channel length is 24σ, and the bump height is 0.4σ, where σ is the standard deviation of the Gaussian. The bump geometry was represented using cubic curved elements on the boundary. Wall boundary conditions were enforced on the top and bottom channel boundaries. At the outflow, the static pressure was set, and at the inflow, the total temperature, total pressure, and flow angle (0^o) were prescribed, resulting in a free-stream Mach number of $M = 0.2$. The output of interest in this case was the L_2 norm of the entropy error, $||S - S_{fs}||_{L_2}$, where S_{fs} is the free-stream entropy.

Again, three hierarchical grids (587, 2348, and 9392 elements) were used in a convergence study. The results are shown in Figure 5-6. As in the Ringleb case, optimal error convergence of $p + 1$ is attained. Figure 5-6 also shows the accuracy versus CPU time for each run. The advantage of higher order for obtaining accurate solutions is again evident.

The entropy error and residual convergence histories are shown in Figure 5-7. Grid dependence is apparent but not significant. Order independence is shown in Figure 5-8a, which shows the residual history for FMG with full convergence on each level. The accompanying step-like error plot in Figure 5-8b illustrates that error truncation is attained

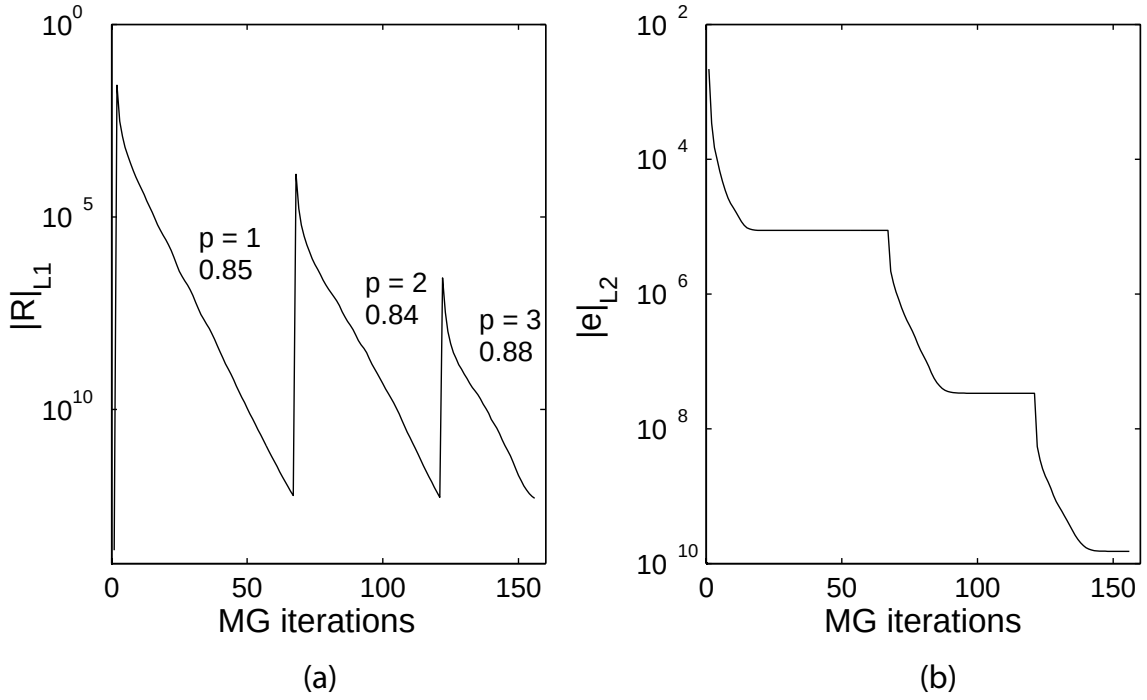

Figure 5-4: Ringleb flow (1408 elements): FMG history with full convergence on each level.

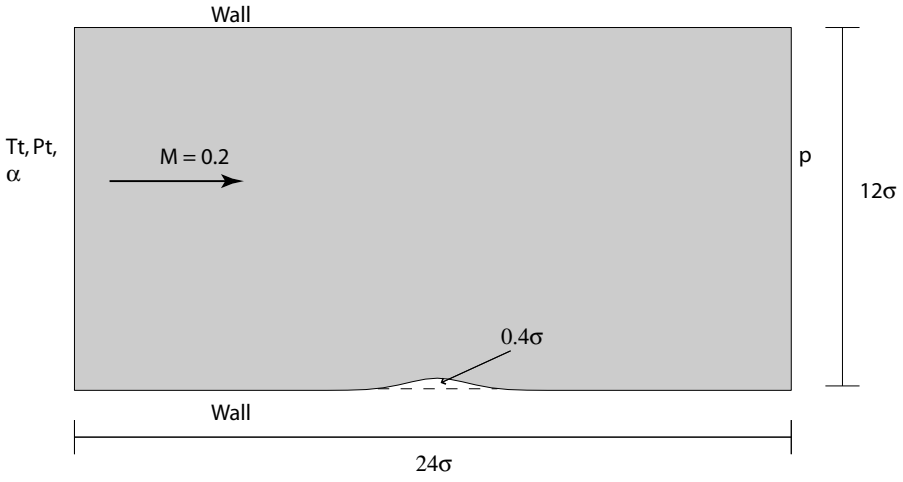

Figure 5-5: Domain for flow over a Gaussian bump.

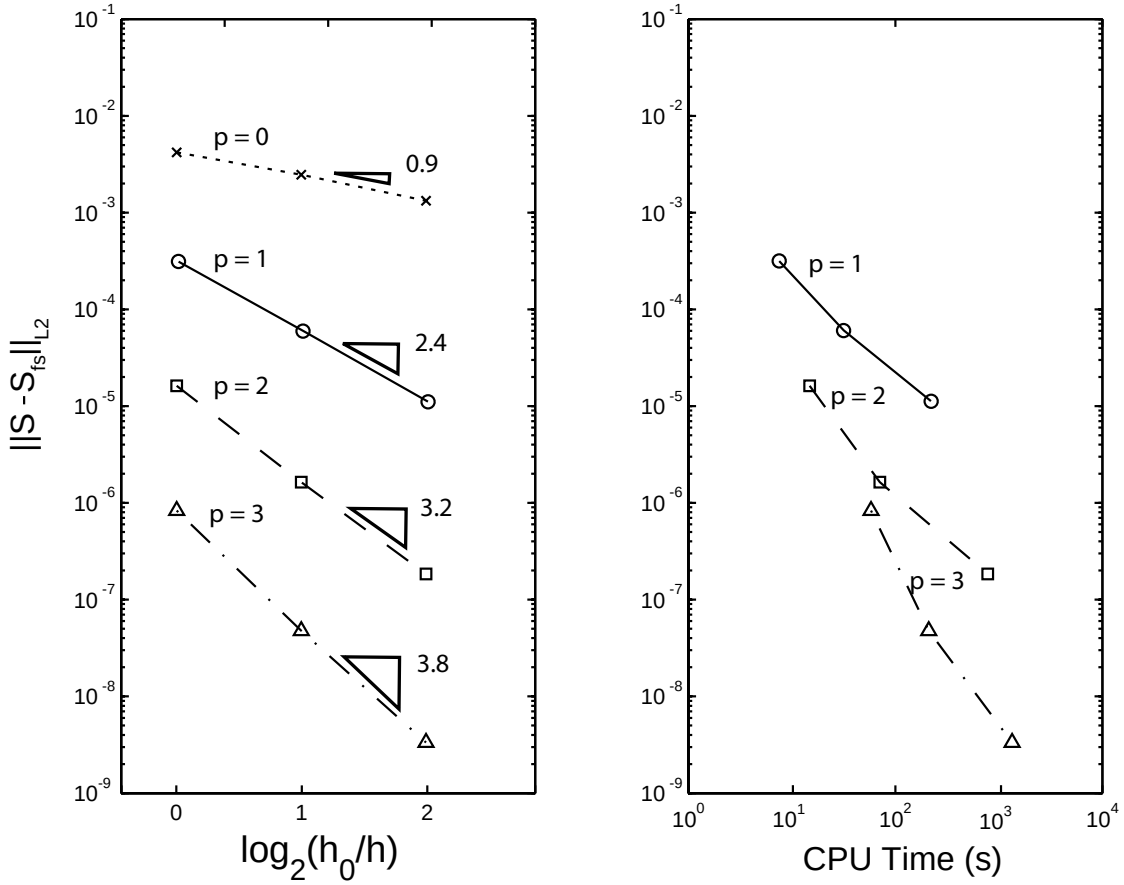

Figure 5-6: Gaussian bump: accuracy vs. CPU time.

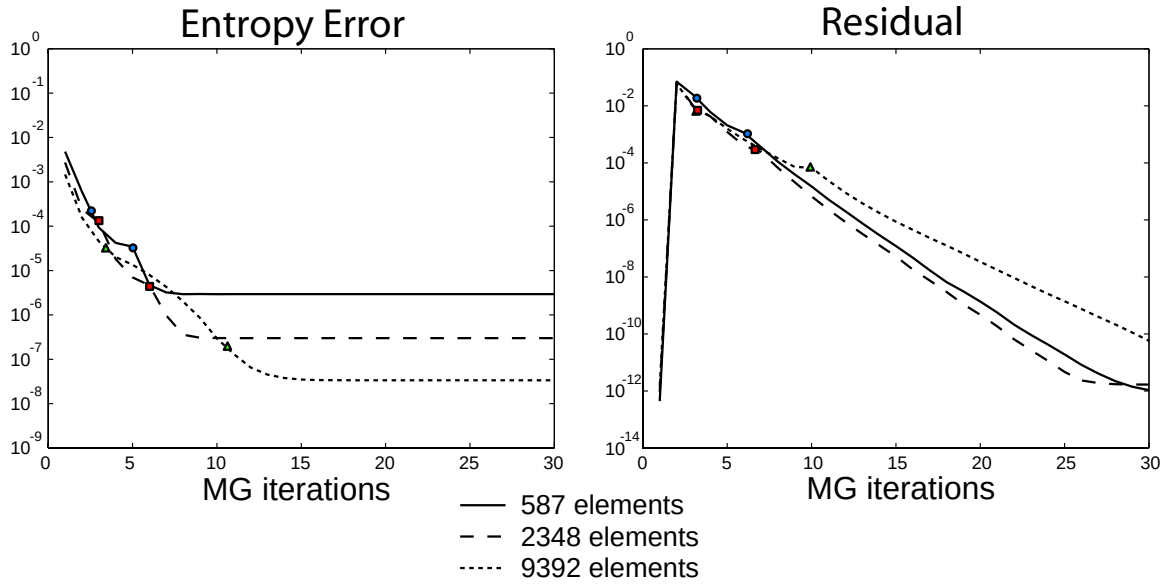

Figure 5-7: Gaussian bump: FMG convergence history.

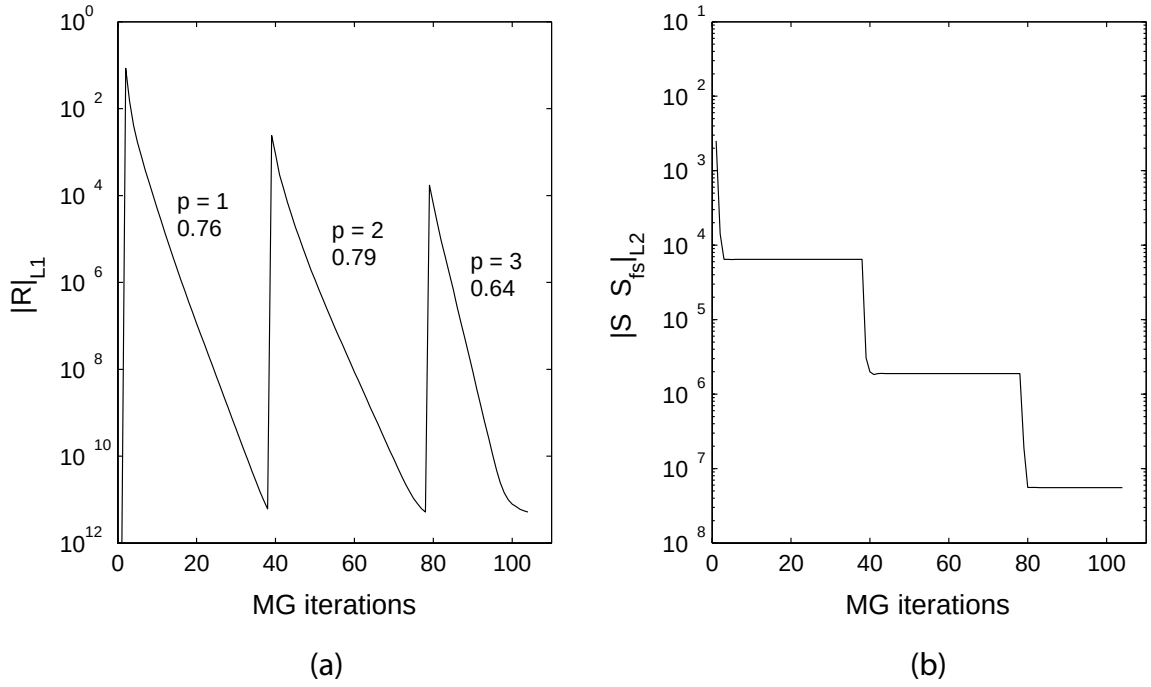

Figure 5-8: Gaussian bump (2348 elements): FMG history with full convergence on each level.

with only a few multigrid iterations after transfer to a finer level, which is characteristic of an effective multigrid scheme.

5.1.3 Joukowski Airfoil

The third test problem is a 12 percent thick Joukowski airfoil at $M = 0.2$ and $\alpha = 0^o$. The airfoil was created using a standard Joukowski transformation. The computational domain is shown in Figure 5-9, and a portion of an intermediate-sized mesh is shown in Figure 5-10. Cubic curved elements were used adjacent to the airfoil to represent the geometry. Total temperature, total pressure, and flow angle were specified at the inlet, static pressure was specified at the outlet, and the free-stream state was prescribed at the top and bottom domain boundaries. For this case, the output of interest was the airfoil drag coefficient, which is zero for the exact inviscid solution.

The results of an accuracy study (grid sizes of 974, 3896, and 15584 elements) are shown in Figure 5-11. Optimal convergence is roughly attained, although the error on the finest grid $p = 3$ solution appears to bottom out. This effect is likely a consequence of a singularity caused by the inviscid flow assumption and a finite trailing edge angle (due to relatively coarse gridding). We expect this effect to lessen with the introduction of viscous modeling.

The error and residual histories are shown in Figure 5-12. The drag convergence ap-

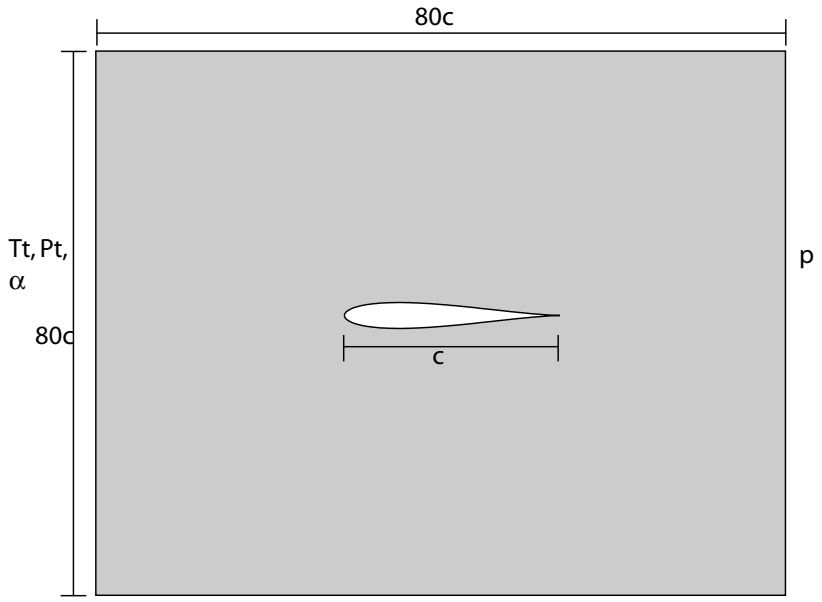

Figure 5-9: Domain for flow over a Joukowski airfoil (not to scale).

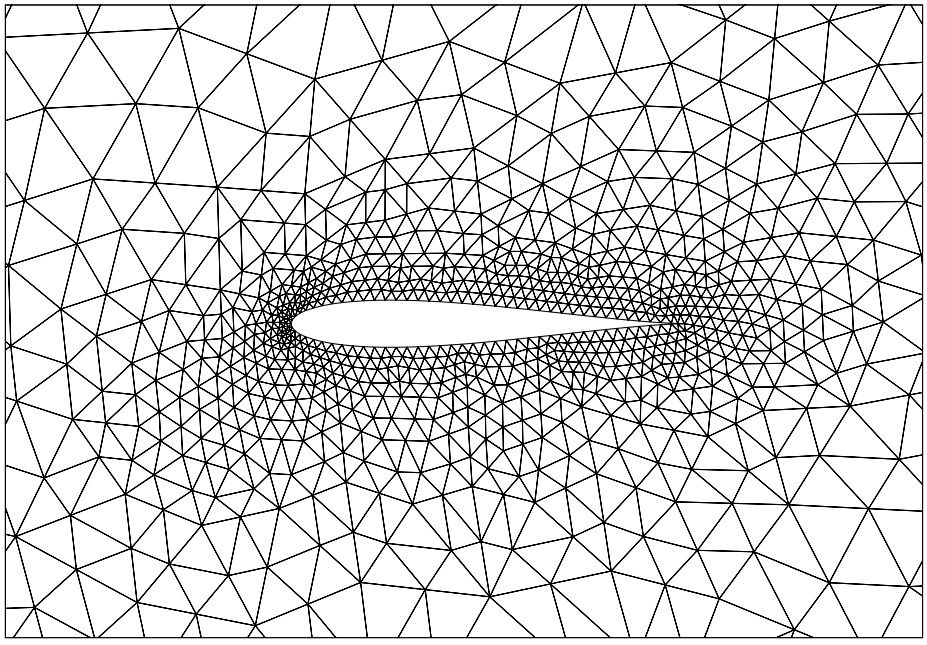

Figure 5-10: Close-up of the intermediate mesh around the Joukowski airfoil. Total mesh size is 3896 elements.

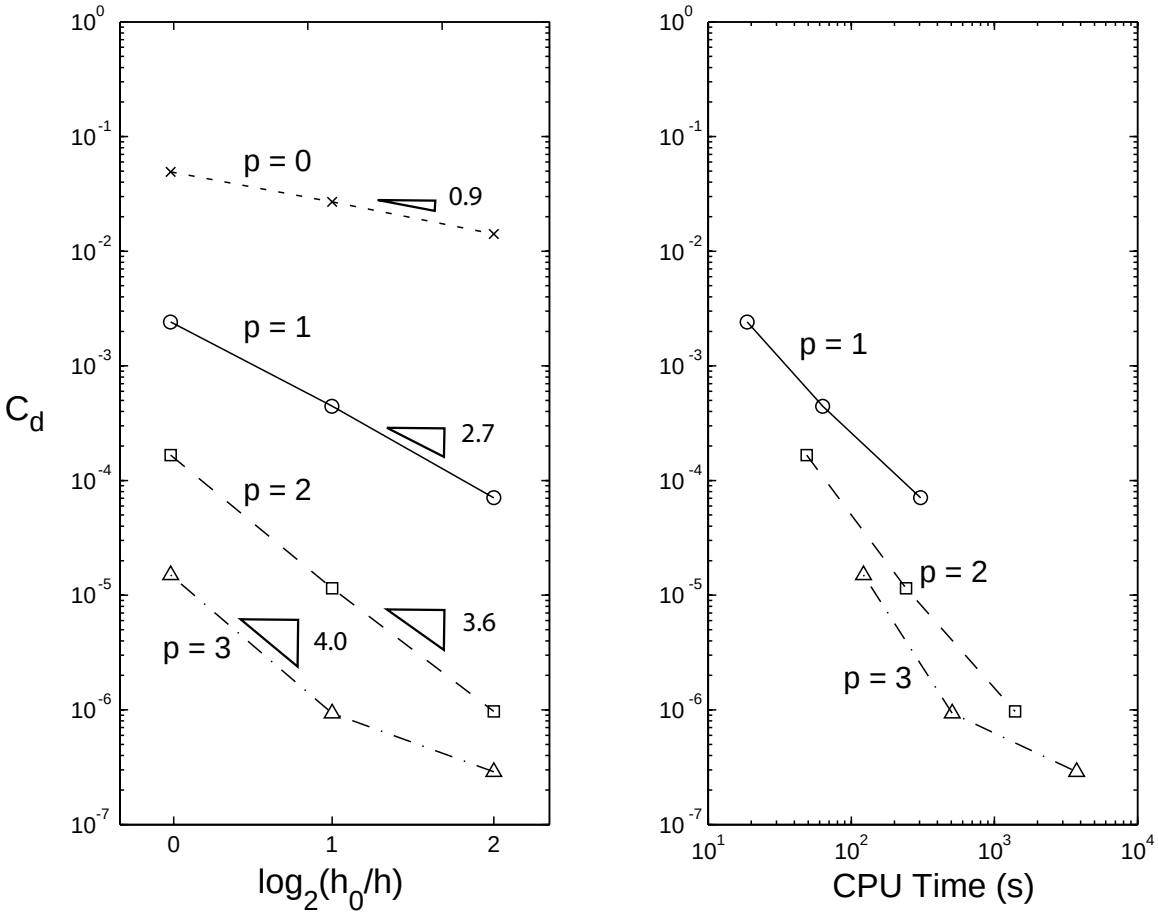

Figure 5-11: Joukowski airfoil: accuracy vs. CPU time.

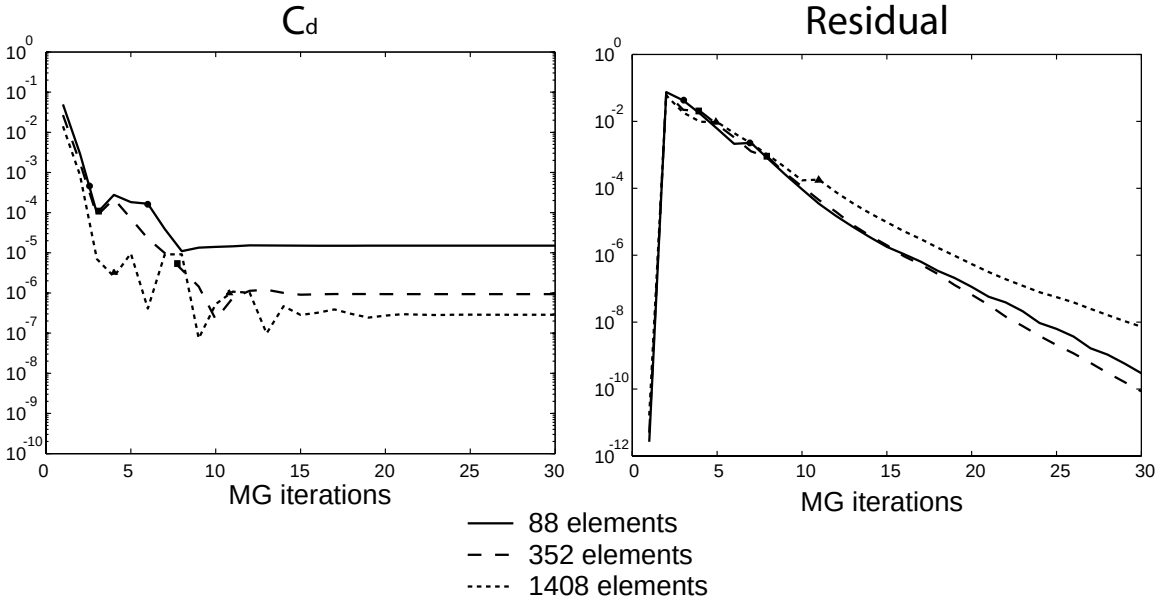

Figure 5-12: Joukowski airfoil: FMG convergence history.

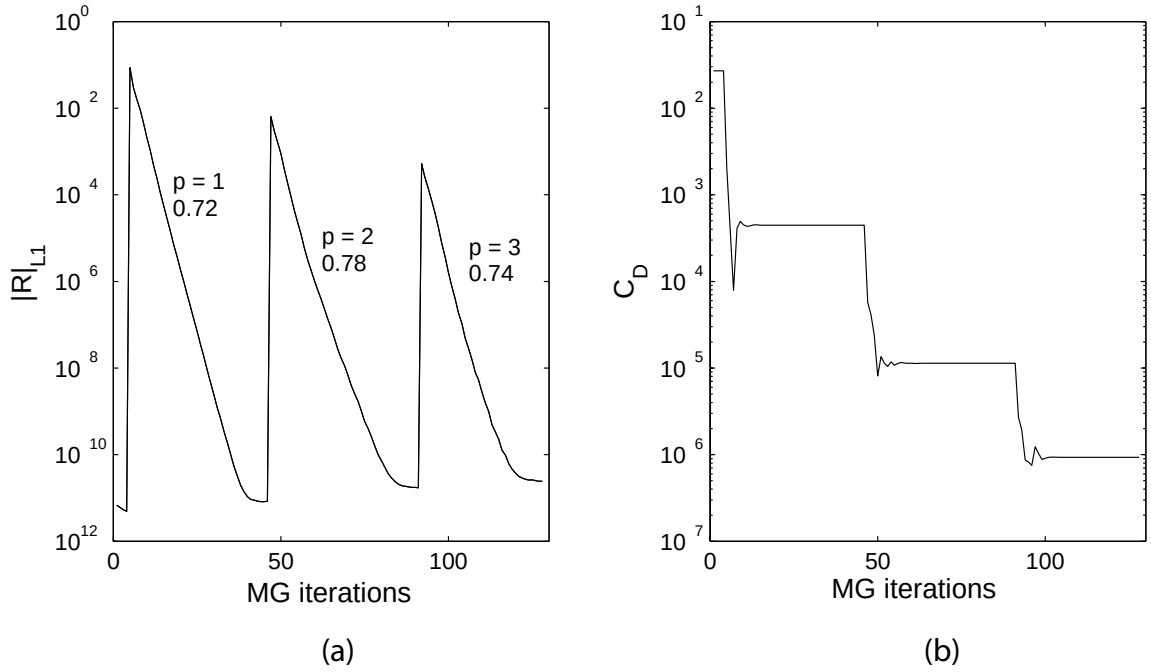

Figure 5-13: Joukowski airfoil (3896 elements): FMG history with full convergence on each level.

pears more oscillatory than the previous cases, but still proceeds faster than the residual convergence. Grid dependence exists, but does not appear to be strong. Figure 5-13 shows the residual and error histories for FMG with full convergence on each level. The order-independence is again clear, as is the quick convergence of the drag output.

A low Mach number accuracy and convergence study was performed for the Joukowski airfoil. For low Mach numbers, the compressible Euler equations describe nearly incompressible flow. This limit poses a problem for the numerical solution process in terms of slow convergence and decreased accuracy [41, 12]. Numerous investigations in this area have shed light on the problem, and the behavior is generally well understood. The root cause for the convergence degradation lies in the vastly different wave speeds in the system at low Mach numbers. While acoustic modes propagate at the speed of sound, convective disturbances propagate at the flow velocity, which is much lower. Since it is the acoustic modes that dictate the effective "time step" of the iterative smoother, the convergence of low Mach number cases suffers as convective disturbances take many iterations to traverse the domain [12].

In addition to slow convergence, many schemes that introduce dissipation also suffer from decreased accuracy [47]. This effect is due to an imbalance at low Mach numbers between the dissipation terms and the terms of the Euler equations. This problem is often solved by the same method used to treat the slow convergence, which is local preconditioning.

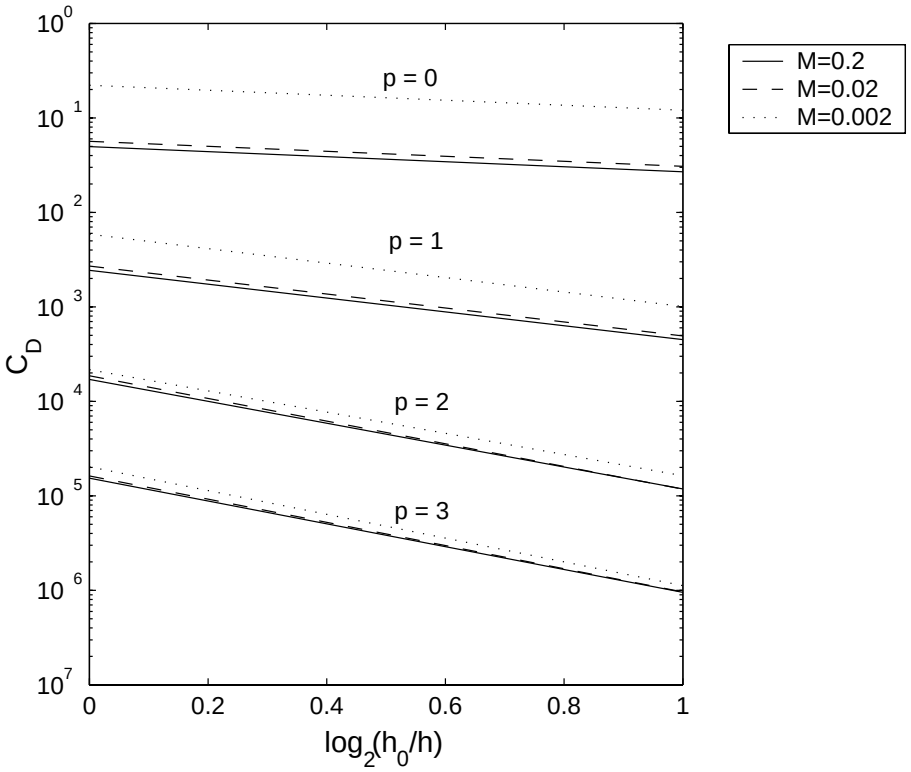

Figure 5-14: Joukowski airfoil: accuracy convergence for various M.

Numerous local preconditioners have been developed for the Euler equations [41, 25, 12], with very good results. However, a drawback of many of these preconditioners is poor robustness near stagnation points [13]. For SUPG/GLS finite element discretizations Wong *et al* [48] have developed a stabilization matrix which remains well-conditioned in the low Mach number limit.

In the present study, we are interested in the behavior of the DG discretization and the p-multigrid solver at low Mach numbers. To this end, an accuracy and convergence study was performed for the Joukowski airfoil using the first two grids (974 and 3896 elements), with free-stream Mach numbers of $M = 0.2$, $M = 0.02$, and $M = 0.002$. The accuracy plot for C_D is shown in Figure 5-14. Although the error convergence rate remains optimal (p+1), the absolute accuracy in C_D deteriorates with decreasing Mach number. This effect is greatest for the low-order discretizations, and decreases with increasing order. The deterioration is expected, as the upwind flux function used in the DG discretization is poorly conditioned as $M \to 0$. However, as the order is increased, the magnitude of the inter-element jumps decreases for smooth problems, making the poorly scaled dissipation is less detrimental. Figure 5-15 shows the Mach contours as M is reduced from 0.2 to 0.002, for $p = 1$ and $p = 3$ interpolation. While the $p = 1$ contours exhibit a deterioration

in accuracy, especially near the airfoil surface where the dissipation is highest, the $p = 3$ contours remain smooth.

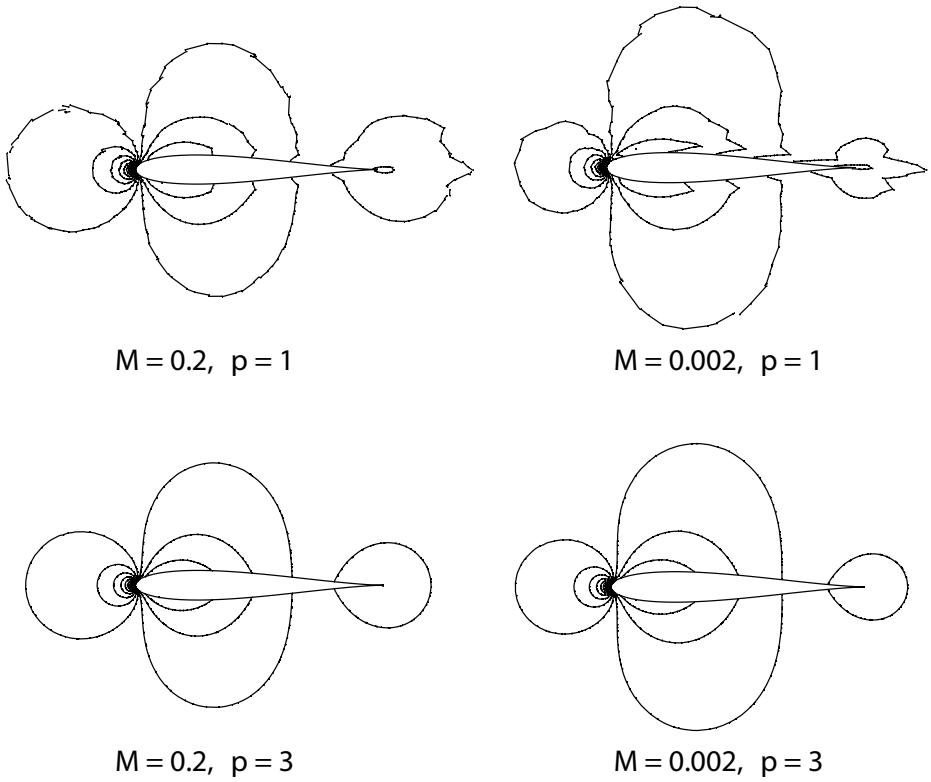

Figure 5-15: Joukowski airfoil: Mach contours for $p = 1$ and $p = 3$ interpolation at $M = 0.2$ and $M = 0.002$. For $p = 1$ the accuracy decrease is evident at low M, while for $p = 3$ it is not.

While the higher-order discretization eliminates the low Mach number accuracy degradation, the multigrid convergence rates still suffer, as shown in Figure 5-16. The plots are for a $p = 3$ solution on the intermediate-sized mesh. For $M = 0.002$, the solution requires over 1500 multigrid iterations to converge to machine zero residual. Local preconditioning could be used to accelerate convergence at low Mach numbers.

In addition to the low Mach number study, a comparison of line-implicit versus block-implicit smoothing was carried out using the intermediate Joukowski airfoil mesh. The results are shown in Figure 5-17 for $p = 3$ interpolation. In terms of CPU time, the line-implicit smoother is about twice as fast as the block-implicit smoother. When used in conjunction with p-multigrid, however, the difference in speed is less pronounced, with the line-implicit multigrid scheme showing slightly faster convergence. This observation is likely due to the fact that the lines on the unstructured mesh are not perfectly aligned with the flow direction, preventing the smoothing of some of the LH modes, as discussed in Section 4.2. The observed behavior is also consistent with Pierce and Giles, who recommended

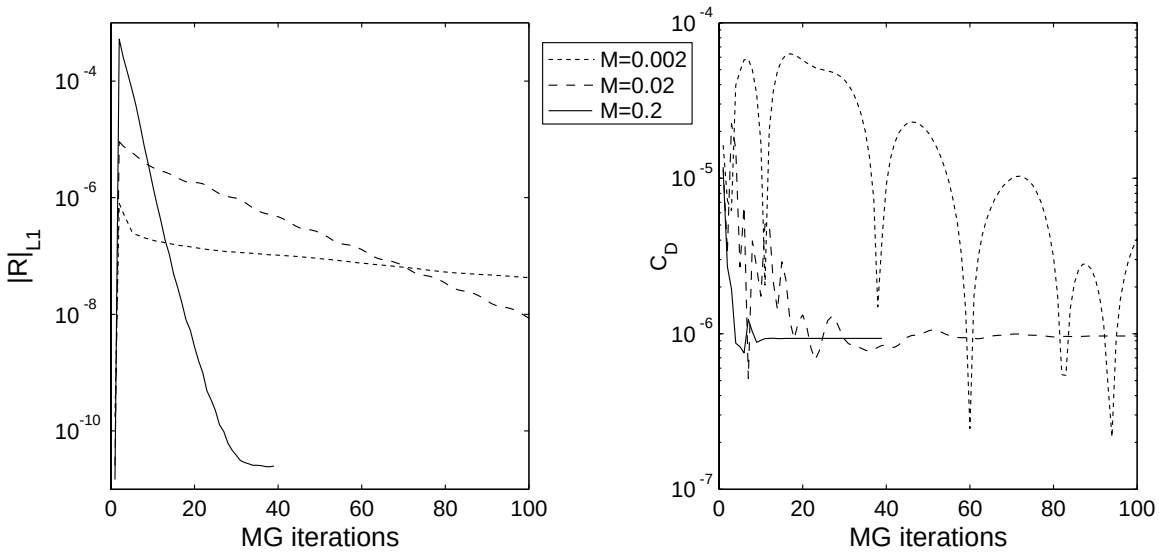

Figure 5-16: Joukowski airfoil (3896 elements): multigrid convergence for $p = 3$.

standard multigrid with simple block relaxation for the Euler equations [34].

5.2 Three-dimensional Results

Two problems were studied in three dimensions: a 3-D extension of Ringleb flow, and flow through a variable-area duct. In the following sections, both problems are described and the associated results are presented.

5.2.1 Three-dimensional Ringleb Flow

The 3-D Ringleb flow case is a trivial extension of the 2-D case. The domain considered is a rectangular box with coordinates as shown in Figure 5-18. 2-D Ringleb flow was imposed in the (x', z) plane, where the (x', y') coordinates are rotated by 45^o from the (x, y) coordinates.

No variation was specified in the y' direction. As in the 2-D case, the boundary conditions were imposed by setting the exact state on the exterior of the domain and using the flux function. The domain was meshed by first uniformly dividing the box into N^3 identical box cells, and then subdividing each cell into five tetrahedra. The cell subdivision is illustrated in the inset of Figure 5-18. Grid refinement for the accuracy study was performed by uniformly refining the overlaying structured mesh, and then subdividing each cell into tetrahedra. Although this method does not result in hierarchical grids, it turns out to be sufficient for an accuracy study, as the same element geometries are employed and the elemental length scale is halved at each refinement.

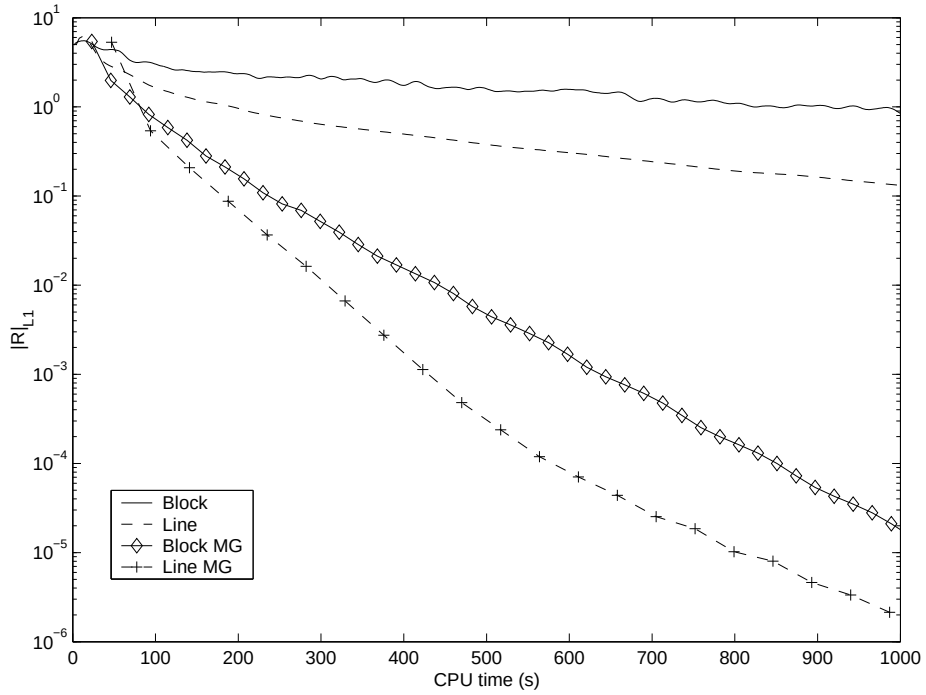

Figure 5-17: Joukowski airfoil (3896 elements), $M = 0.2$ and $p = 3$: comparison of convergence rates of various solvers.

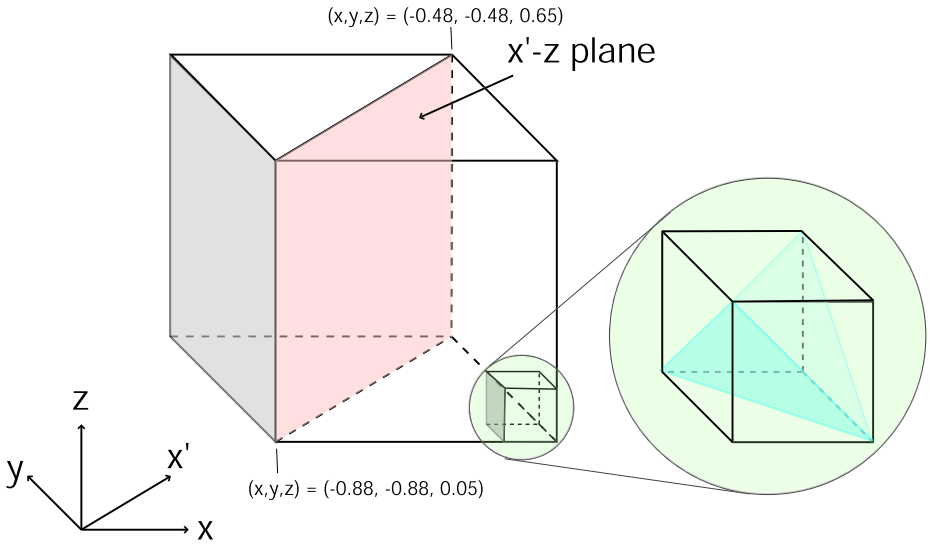

Figure 5-18: Description of Ringleb flow in 3-D.

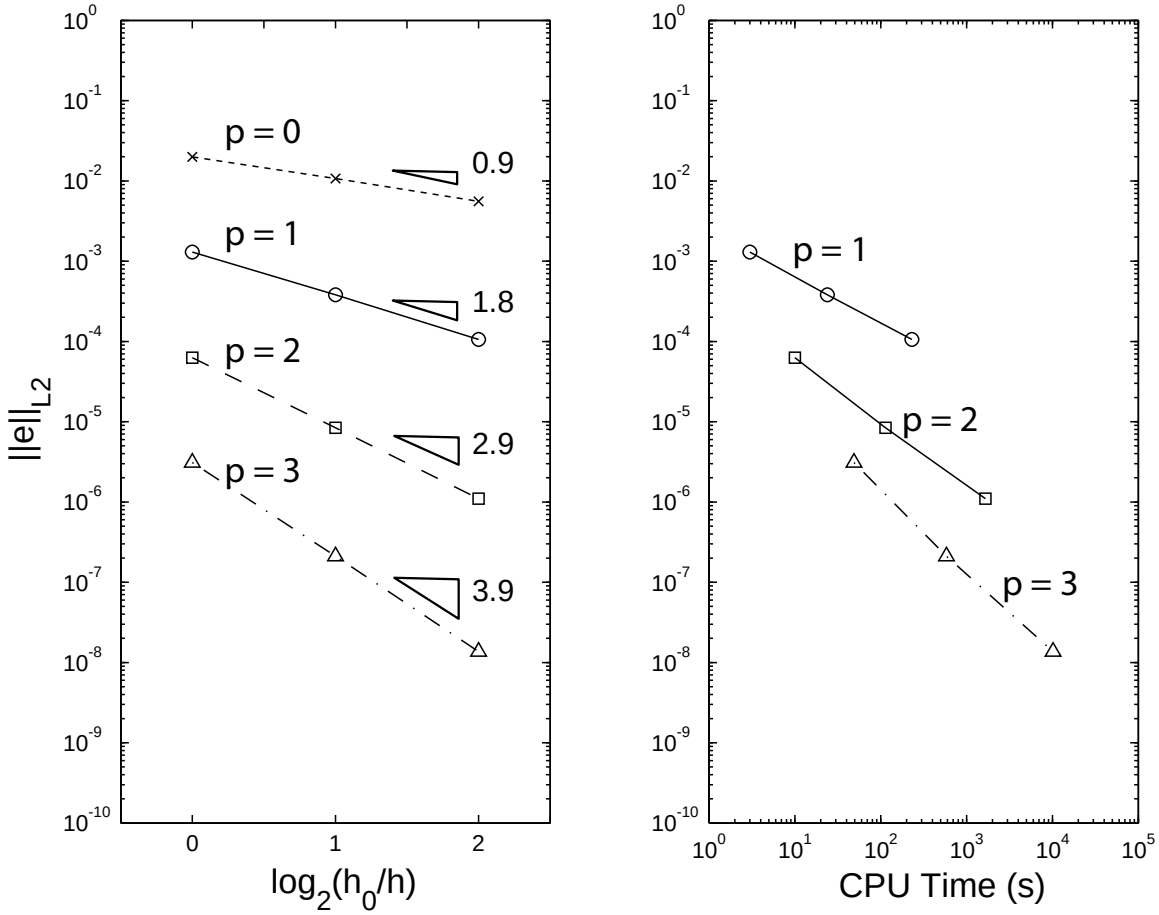

Figure 5-19: 3-D Ringleb flow: accuracy vs. CPU time.

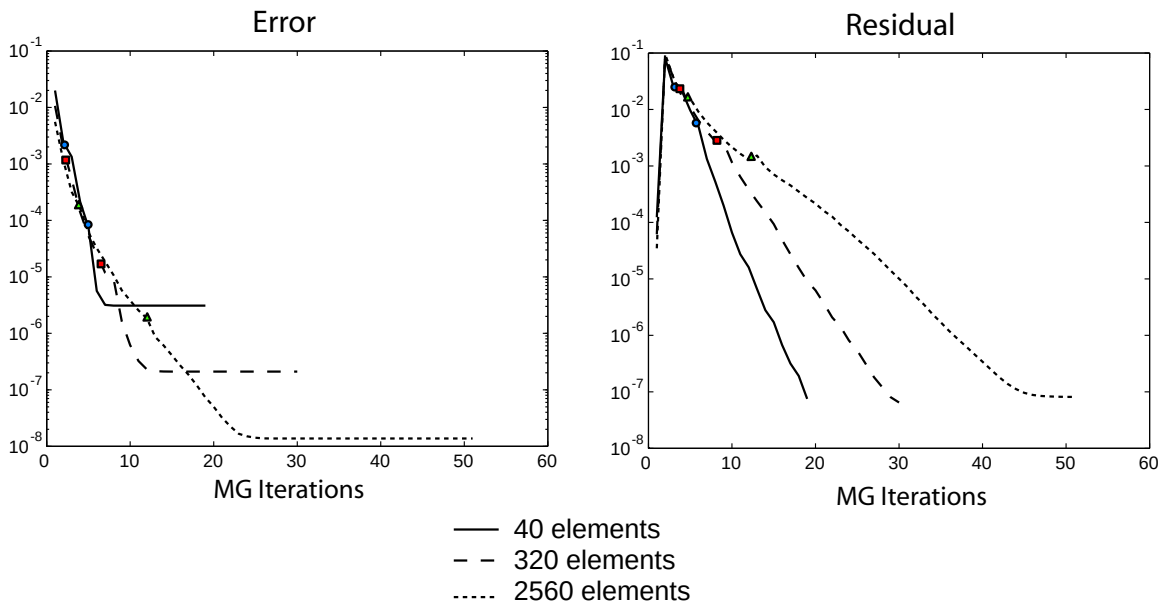

Figure 5-20: 3-D Ringleb flow: FMG convergence history.

An accuracy study was performed using three grids, with sizes of 40 elements ($N = 2$), 320 elements ($N = 4$), and 2560 elements, ($N = 8$) . As in the 2-D case, the L_2 norm of the error served as the output. Figure 5-19 shows that optimal error convergence of $p+1$ is approximately attained. The timing results indicate the benefit of higher order in achieving high accuracy. For example, to achieve an error norm of 10^{-4}, $p = 2$ can be used over $p = 1$ with about 1.5 orders of magnitude decrease in CPU time.

Figure 5-20 shows the FMG error and residual histories, which indicate grid dependence. Figure 5-21 shows the asymptotic convergence rates, illustrating the order independence in (a), and the early error truncation in (b). These results are similar to the 2-D case.

5.2.2 Flow in a Duct with a Gaussian Perturbation

The second problem studied in 3-D was that of subsonic flow inside a duct with a Gaussian perturbation on the bottom wall. The problem setup is depicted in Figure 5-22. Wall boundary conditions were imposed on each of the four duct walls. Cubic curved boundary elements were employed on the bottom wall to fit the Gaussian perturbation. Elements that shared a face or an edge with the bottom surface were curved to fit the geometry. Static pressure was prescribed at the outlet, and at the inflow, the total temperature, total pressure, and flow direction were specified, resulting in a freestream Mach number of 0.2. The output of interest was the L_2 norm of the entropy error over the entire flowfield.

Three grids (240, 1920, and 15360 elements) were constructed for the accuracy study using the refinement procedure outlined for the 3-D Ringleb case. Results from the accuracy

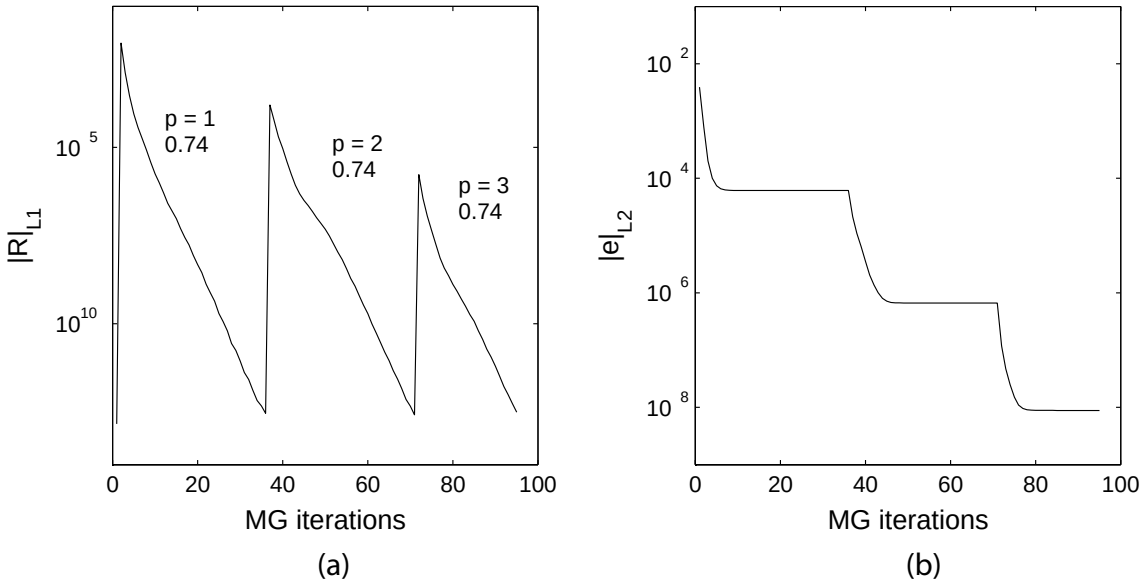

Figure 5-21: 3-D Ringleb flow (2560 elements): FMG history with full convergence on each level.

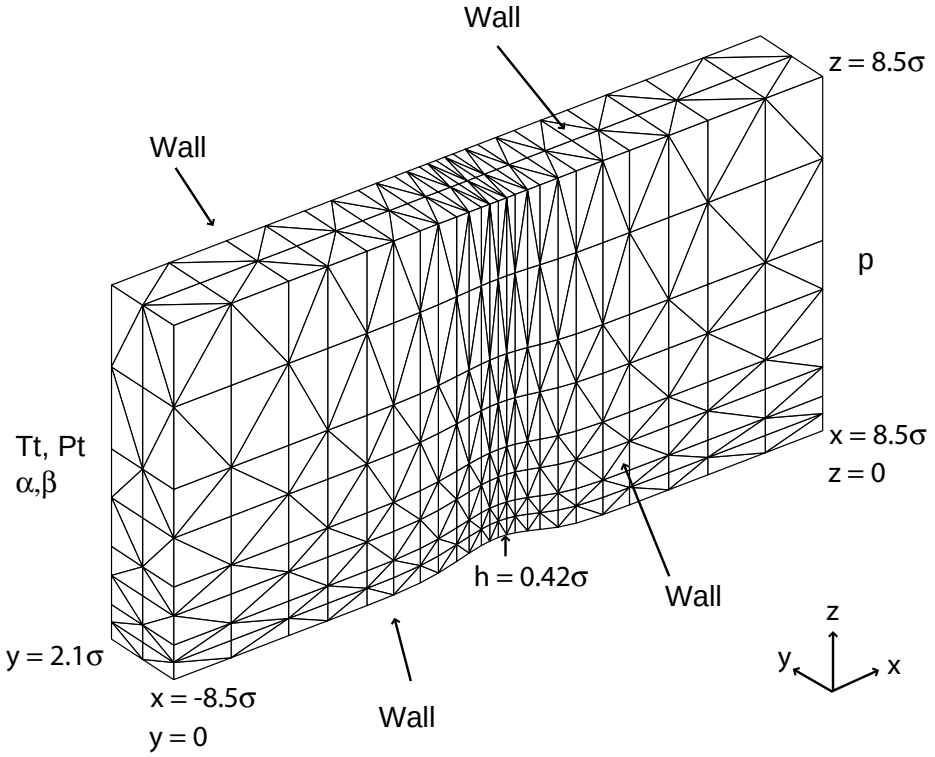

Figure 5-22: Flow over a Gaussian perturbation in 3-D.

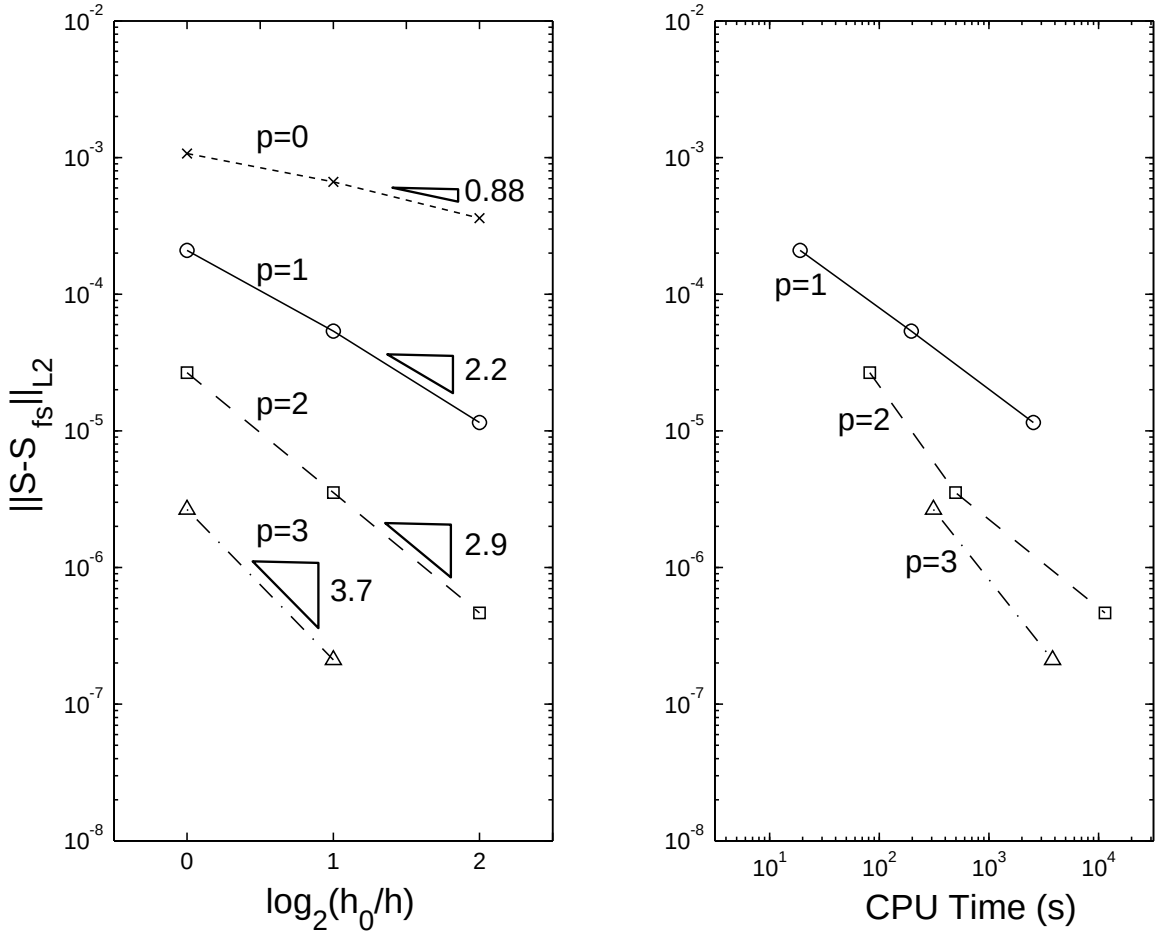

Figure 5-23: 3-D Gaussian bump: accuracy vs. CPU time.

study are shown in Figure 5-23. Convergence was observed to be near the optimal $p+1$. The timing results are similar to the 2-D case and again display the benefit of higher order over grid refinement.

The FMG performance results, shown in Figure 5-24, again indicate grid dependence. As in the previous cases, however, the asymptotic convergence rate, Figure 5-25a, remains order independent. The associated step-like error plot in Figure 5-25b shows the rapid error convergence expected from an effective multigrid scheme.

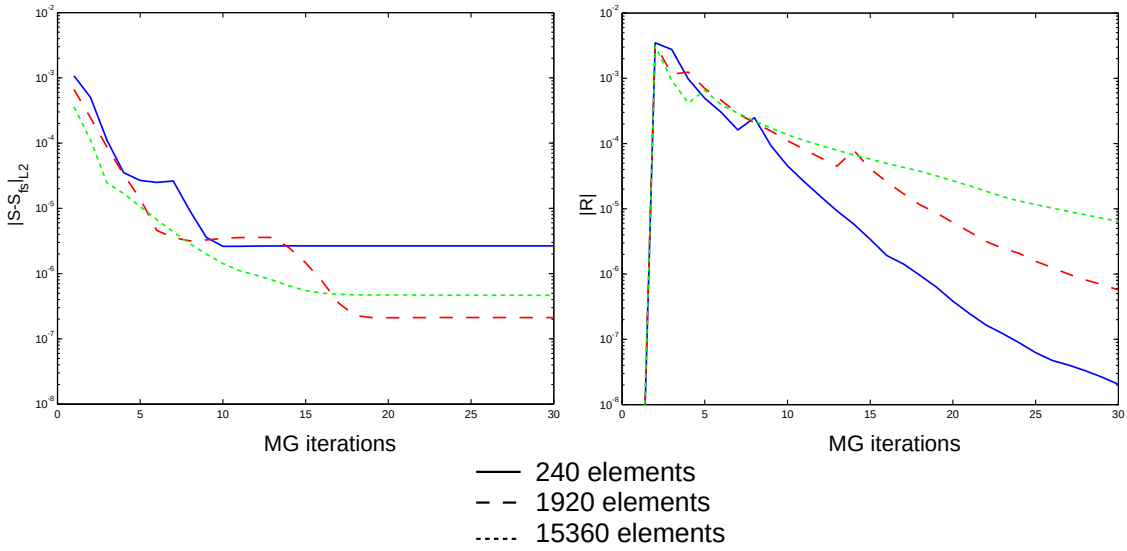

Figure 5-24: 3-D Gaussian bump: FMG convergence history.

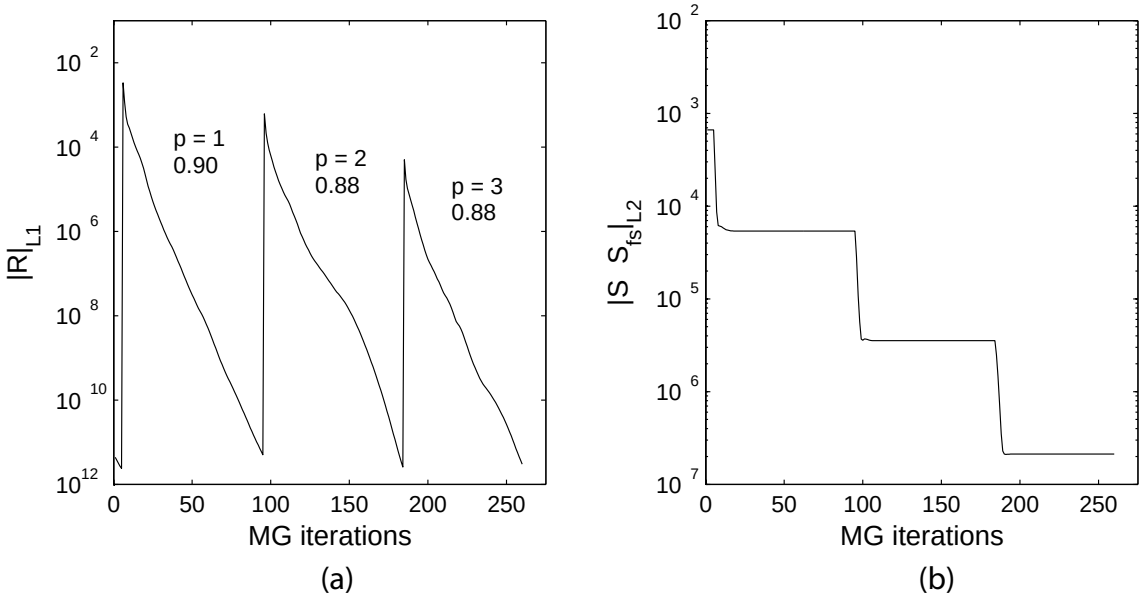

Figure 5-25: 3-D Gaussian bump (1920 elements): FMG history with full convergence on each level.

Chapter 6

Parallelization

The point was made in Chapter 1 that the local nature of the DG discretization facilitates parallelization of the solution algorithm. To validate this claim and to reduce the wall-clock time of the runs, parallel versions of the 2-D and 3-D solvers were implemented. The Message Passing Interface (MPI) library was used to handle communication, and the code was tested on a coarse-grain parallel cluster. This chapter presents the methodology behind the parallelization and the associated scalability results.

6.1 Background

CFD currently makes use of the state of the art in parallel computing to quickly solve very large problems of practical interest. Industry-standard codes can attain close to ideal scalability for large numbers of processors but often at the cost of machine-dependent tuning. In 2001, Gropp *et al* [16] carried out a parallelization case study with an unstructured CFD code representative of the state of practice at NASA. They found that they could achieve optimal parallel efficiency only by tuning to minimize cache-misses for a particular achievable memory bandwidth. Such platform-dependent tuning is undesirable because it makes the code less portable. The parallelization effort in this work shies away from this tuning difficulty in two ways: by using a high-order DG discretization to minimize communication bandwidth, and by focusing on a coarse-grain parallelization in which the communication barrier is much less of an issue.

A significant amount of research has been done in the area of DG parallelization, and parallel solution algorithms have been implemented. Closely related work was done in 1999 by Baggag *et al* [3], who considered the DG discretization of the Euler equations of gas dynamics, and used MPI for parallelization. They tested their code on several architectures, including SGI Origin, IBM SP2, and clusters of SGI and Sun workstations. For the sizes

of problems tested, they were able to achieve near perfect speedup on 4 processors (3.5-3.8 speedup), but they ran into a communication barrier when they tried to scale higher. With 8 processors, the speedup ranged from 3.1 to 7.1, depending on the system architecture. The communication barrier is likely due to the small problem sizes considered (in some cases less than 100 elements per processor) and to the low computation-to-communication ratio. A key difference between this work and that of the authors is that instead of line-implicit multigrid, the authors used an explicit time-marching scheme. An explicit method usually requires more residual evaluations in the solution process and hence reduces the computation-to-communication time ratio. In addition, Baggag *et al* considered only 2-D cases while this work will include both 2-D and 3-D.

A 3-D parallel DG solver was presented by Crivellini *et al* in 2003 [11]. These authors studied acoustic propagation and hence were concerned with the *linear* Euler equations. They also used MPI for communication and were able to achieve decent speedup (7.5 for 10 processors) with approximately 500 elements per processor. They performed their tests on a distributed memory Linux cluster containing 5 nodes, 10 CPU AMD MPI 1.33GHz with standard Ethernet cards and a single switch (100 Mbps). Again, these authors used explicit Runge-Kutta for advancing in time, which requires several residual evaluations per time step and decreases the computation-to-communication ratio compared to an implicit scheme.

More recently, Dong and Karniadakis [15] presented a multilevel parallel model for general high-order numerical methods. Multilevel parallelism takes advantage of modern high-performance computer architectures by partitioning a problem on two or more levels. For example, on the first level the domain can be partitioned among the nodes, while on the second level each subdomain can be partitioned among the multiple processors per node. In DG with line-implicit smoothing, the final level could be the lines of elements, for which block-tridiagonal systems have to be inverted. In their implementation within a spectral element framework, Dong and Karniadakis observed very good scaling of multilevel parallelism and attributed it to the greatly reduced number of processes involved in communications at each level. Such a multilevel model presents an option for extending the coarse-grain parallelization in this work to architectures with large numbers of processors.

6.2 Implementation

An effort was made to introduce parallelization without major changes to the existing source code. To this end, pre-compiler flags were used to minimize code duplication, allowing most of the serial functions to be reused. The file formats for input and output were

retained, with the caveat that multiple input grids were used corresponding to the partitioned domain. Two utility functions were created for splitting and joining grids, allowing for simple conversion between serial and parallel cases.

6.2.1 Data Structure

The existing finite element solver uses an unstructured mesh, which means that the full grid connectivity is stored explicitly. In the grid data structure, the elements are grouped into *Element Groups*. All elements in one Element Group must have the same order of interpolation and be of the same type (curved elements are a different type than linear elements). For example, in a mesh around an airfoil, one group of elements can consist of the interior elements and another of the curved boundary elements.

Boundary Face Groups store information about which element faces form a particular boundary, and they are associated with specific boundary conditions. It is through these boundary face groups that the data for parallel computation is communicated. Specifically, additional storage is created for each Boundary Face Group that is an *Inter-Domain Boundary* - that is, a group that represents element faces neighboring a sub-domain stored on another processor. This storage consists of two types of data for each face on an Inter-Domain Boundary: the orientation of the neighboring element and the state data stored in the neighboring element. An example diagram of this connectivity is shown in Figure 6-1 for the case of a square.

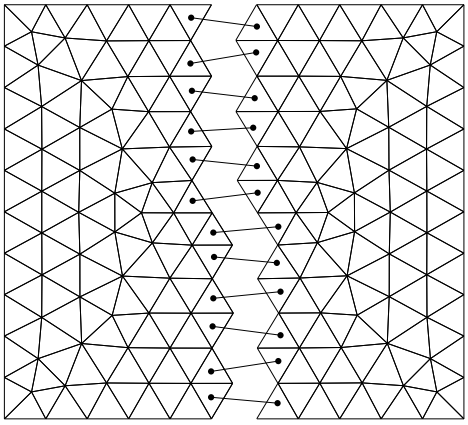

Figure 6-1: Connectivity of two sub-domains via Boundary Face Groups.

The overhead associated with the parallel structure storage is minimal in the DG discretization since only the data for one layer of neighboring elements has to be stored. This is true even for higher order interpolation since the higher order is introduced locally within each element.

6.2.2 Parallel Solution Algorithm

The parallel data structure allows each sub-domain to store the data for the neighboring elements that are not on the processor. In a parallel solver, the inter-domain boundary data has to be communicated among the processors at an appropriate time with the intent that the communication time should not have a significant impact on the computation time for practical problems. The data communication process is evident by considering the parallelization of a basic iterative step.

Each iteration requires the construction of the residual vector, $\mathbf{R}$, and part or all of the Jacobian matrix, $\partial\mathbf{R}/\partial\mathbf{u}$, on every element. This construction is performed by looping over the elements, the interior faces, and the boundary faces. Since no order is required for these loops, for parallelization the loop over the boundary faces is carried out last to allow for the communication of the boundary data. The resulting parallel solution algorithm reads as follows,

Parallel Solution Algorithm - Basic Iterative Step

1 Calculate contribution to $\mathbf{R}$ and $\partial\mathbf{R}/\partial\mathbf{u}$ from the interior faces and the elements.

2 Wait for inter-domain boundary data transmission to complete.

3 Calculate contribution to $\mathbf{R}$ and $\partial\mathbf{R}/\partial\mathbf{u}$ from boundary elements.

4 Calculate $\mathbf{u}^{n+1}$ using the basic iterative scheme (3.1).

5 Begin the transmission of inter-domain boundary data.

In Step 5, MPI non-blocking sends and receives are used so that computation can continue on each processor while the communication is in progress. Ideally, the inter-processor communications finish by Step 2 in the next iteration so that no computation time is lost waiting for communications to complete. Before the first iteration, a call to the parallel initialization function starts the first transmission of boundary data. In addition to the boundary data, the processors also exchange information on the current residual norm (for printout by the root processor), the order of interpolation (for p-multigrid), and the status of computation (e.g. if an error has occurred).

Both the block-implicit and non-lean line-implicit iterative schemes were parallelized using the above algorithm. The lean line solver is a special case in that the residual and Jacobian are computed sequentially for each line while the calculation of the boundary contribution to $\mathbf{R}$ and $\partial\mathbf{R}/\partial\mathbf{u}$ is done at once through a single function. Changing the method of enforcing boundary conditions to suit parallelization would require additional

storage and computation, and further segregation of the code between serial and parallel. Thus, the boundary condition enforcement function was not changed, and the communications were required to end before the calculations of $\mathbf{R}$ and $\partial\mathbf{R}/\partial\mathbf{u}$ started. Although this implementation does not allow as much time for communication at each iteration, with the relatively small amount of communication inherent in parallel DG, the performance was not found to suffer significantly.

With the parallelization of the iterative smoother, p-multigrid was effectively parallelized too. This is because p-multigrid simply consists of smoothing on different interpolation levels and using prolongation and restriction operators to transfer the solution and residual between levels. Specifically, on each level of p-multigrid, the parallel versions of the iterative smoother were employed.

6.2.3 Grid Partitioning and Joining

Two utility programs were written for converting between serial and parallel cases. The first program takes as input a given grid file (possibly with solution data) and splits it into a given number of sub-domains, such that each sub-domain constitutes a viable grid with its appropriate part of the data. The splitting of the grid structure was performed using the functions available in the METIS library [22]. Specifically, a data tree was constructed with the domain elements as the tree nodes and the interior faces as the connectivities. A call to METIS split this tree with the objectives of minimizing the communication requirement and balancing the number of elements per processor. The remainder of the utility program was devoted to extracting the sub-grids in the proper format. An example of the splitting program applied to a 2-D Gaussian bump grid is illustrated in Figure 6-2.

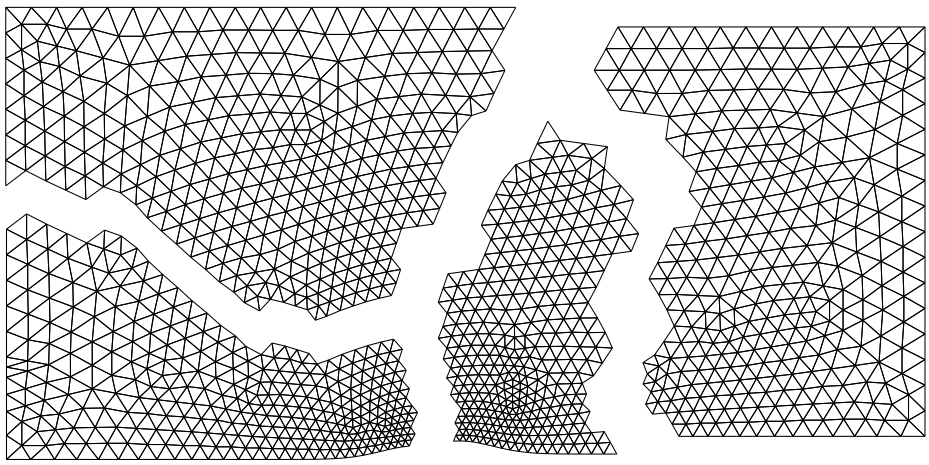

Figure 6-2: Grid splitting utility applied to a Gaussian bump grid.

The second program takes as input a certain number of sub-grids and joins them into a single grid. Applied immediately after the splitting program, it returns the original grid. In practice, the parallel solver is run after splitting a grid causing the data in the sub-grids to change, and the joining program is run last to piece the grids back together.

6.3 Scalability Results

The parallel code was tested in 2-D and 3-D on a 16-node cluster. Each node of the cluster contained four Xeon 2.4 GHz processors with 2 GB shared RAM, and the network connection was 1 Gigabit Ethernet. For the purposes of testing, 10 nodes were used with no more than one processor per node. The basic iterative methods and FMG were tested for parallel scalability and convergence.

The 2-D test case was the Joukowski airfoil using the intermediate mesh (3896 elements), and the 3-D test case was the Gaussian bump using the intermediate mesh (1920 elements). $p = 2$ interpolation was used for both cases. Figure 6-3 shows the scalability of the smoothers for the 2-D case and the 3-D case. The block-implicit scheme is given in 6-3a and the line-implicit scheme in 6-3b. For the line-implicit scheme, the memory-lean implementation was used. The data was obtained from the total solution times using each smoother.

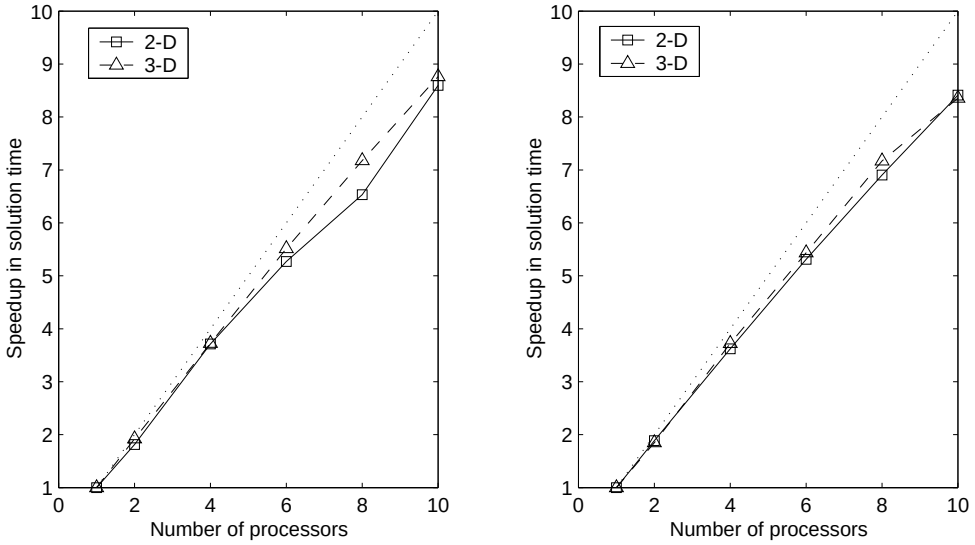

Figure 6-3: Parallel scalability of (a) block-implicit and (b) line-implicit smoothing. $p = 2$ interpolation was used for all cases.

For each parallel case, the speedup is the ratio of the serial solution time to the parallel solution time. The line smoother shows slightly lower scalability than the block smoother,

which can be attributed to the convergence degradation of parallel line smoothing (described next). In fact, per iteration, the line smoother scales better than the block smoother, which has a lower computation to communication ratio. For ten processors, the observed line smoother speedup is around 8.5. Although the block smoother speedup dips to 6.5 on eight processors, it recovers back to nearly 9 on ten processors. The dip may be due to a less favorable mesh partitioning for eight processors compared to ten processors. Nevertheless, for the relatively small mesh sizes tested (about 200 elements per processor in the 3-D, ten-processor case) both smoothers show outstanding coarse-grain scalability.

While the block-implicit scheme produces identical results in serial as in parallel, the line-implicit scheme performance degrades slightly in parallel due to shortened lines. That is, partitioning the domain for parallel computation may cut lines of elements used for the implicit solver, thereby decreasing the solver's performance. Figure 6-4 shows the convergence histories of the line-implicit scheme for the 2-D Joukowski airfoil case using $p = 0$ with varying numbers of processors. The degradation in convergence with increasing processor number, N, is apparent but not considerable: the ten-processor parallel case requires 12% more iterations to converge to machine zero compared to the serial case. This minimal effect is likely due to the coarse parallelization, in which the average line length is not greatly affected by the partitioning. Greater care can be taken in the future to penalize splitting the mesh across lines, although doing so would require a baseline set of lines (and hence a solution) on the mesh, and most likely a higher communication bandwidth.

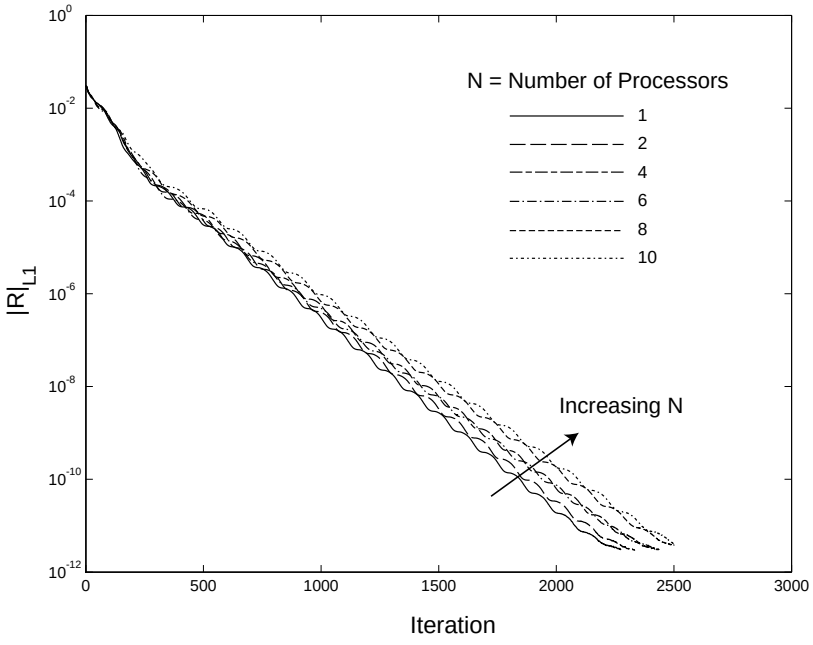

Figure 6-4: Joukowski airfoil (3896 elements): parallel convergence of the line-implicit smoother for $p = 0$ interpolation.

The results of parallel FMG applied to the 3-D bump case are given in Figure 6-5. This figure shows the parallel speedup in CPU time to solution using FMG with line-implicit smoothing and residual-based switching.

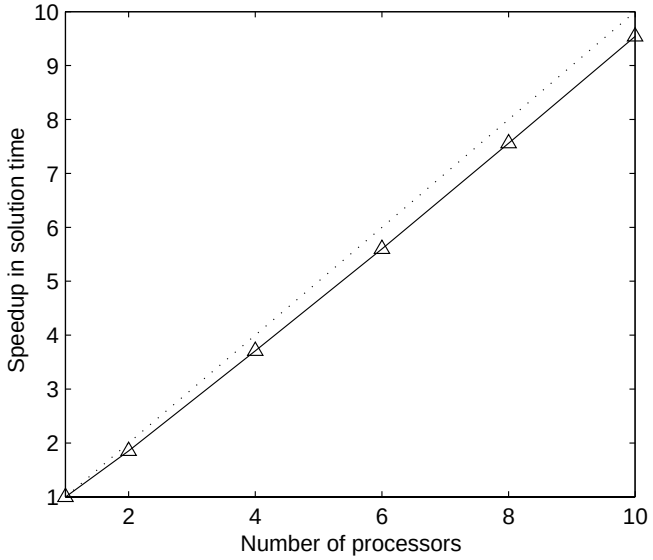

Figure 6-5: 3-D Gaussian bump (1920 elements): time to solution using parallel FMG with residual-based switching. $p = 2$ interpolation on fine level.

The observed speedup for this case is 9.5 for ten processors. This high value is likely due to minimal bandwidth partitioning made possible by the high aspect ratio bump mesh shown in Figure 5-22. In addition, smoothing iterations are more expensive in 3-D than in 2-D for a given p, increasing the computation to communication ratio. Thus, the observed speedup in Figure 6-5 is greater than that of the 2-D line smoother, Figure 6-3b.

6.4 Conclusions

The DG solver under investigation was parallelized using the MPI library, and 2-D and 3-D parallel versions of the code were tested to assess scalability and convergence. Even though relatively small problems were considered (as few as 200 elements per processor), the parallel solvers showed very good scalability and only slight decrease in convergence for the line smoother. The improved scalability over some previous parallel DG implementations can be attributed mostly to the use of an implicit instead of an explicit solution scheme. The implicit schemes used in this work are more expensive per iteration due to the inversion of the preconditioner, allowing for an increased computation to communication ratio. In addition, the cluster used for the tests was equipped with a faster network connection (Gigabit Ethernet) than that used by most previous authors.

Chapter 7

Conclusions

A higher-order solution method was presented for DG applied to the Euler equations. The highlights of this solution method in comparison to most industry-standard CFD methods include: a discontinuous Galerkin finite element discretization, a line-implicit iterative smoother, and a p-multigrid solution algorithm. p-multigrid fits naturally into the local high-order finite element discretization, using local level transfer operators that become trivial to implement in the case of a hierarchical basis. Line smoothing at each level is stable regardless of order and is effective at removing the error modes associated with convection, as predicted by 2-D analysis of an advection problem.

The results for 2-D and 3-D smooth problems demonstrate the advantages, as well as some of the remaining issues of the solution method. First, optimal accuracy convergence was attained for all test cases in two and three dimensions. The outputs of interest included the absolute error norm in the Ringleb cases, the entropy error norm in the Gaussian bump cases, and the drag in the Joukowski airfoil case. Although these results are promising, all the problems considered were of smooth flow, without shocks or discontinuities. Since no effective limiter has yet been implemented for the discretization, the capability of the solver is restricted to shock-free flows.

In terms of performance, order-independent convergence was demonstrated for all test cases. However, grid dependence was observed, and found to be significant for the Ringleb and 3-D bump cases. Regardless, for all test cases, the solution error reaches truncation level before the residual reaches machine-zero, and usually in only a few multigrid iterations following an FMG level transfer.

A low Mach number study on the Joukowski airfoil showed that the absolute accuracy level decreases with decreasing Mach number, although the optimal $p+1$ accuracy convergence rate remains. While this result was expected based on previous work, the favorable observation is that the accuracy degradation diminishes for higher interpolation order. The

iterative convergence rate, however, was shown to degrade with decreasing Mach number for all interpolation orders.

Parallelization work demonstrated the benefits and practicality of parallelizing the DG solution method. The combination of the DG discretization and the implicit solution method made the parallelization effective and relatively straightforward to implement. Testing on a coarse-grain cluster showed very good scalability of the basic iterative smoothers and of p-multigrid.

Most importantly, timing results demonstrated the benefit and practicality of using higher order for attaining high accuracy. Using the p-multigrid solution method, low-order discretizations require highly refined grids and more computational time to attain the levels of accuracy of high-order discretizations on spatially coarser grids. The question that remains is how these results will be affected by non-smooth problems and more complex physical models.

Much work remains to be done to make the solution method presented in this work practical. As already mentioned, a limiter is required to stabilize the oscillatory behavior of high-order approximations near discontinuities. The effects of such a limiter on accuracy and convergence are not known, although decreased performance is expected. In addition, physical models for viscosity (already introduced in 2-D [33]) and turbulence are required for the simulation of practical flow cases. Regarding the solver, the run times are currently relatively long due to the expensive preconditioner inversion. This step can potentially be made cheaper by using an approximate inversion and/or by optimizing the inversion process. In addition, h-multigrid in conjunction with p-multigrid may be necessary to help alleviate the observed h dependence.

Appendix A

Entropy Fix for the Roe Flux

The Roe flux was used to approximately solve the Riemann problem at element boundaries. In its standard form, the Roe flux permits expansion shocks as solutions of the approximate problem. As a result, stationary expansion shocks are not dissipated. Thus, an entropy fix, described below, was used to introduce artificial dissipation.

The Roe flux can be written as

$$\mathcal{H}(\mathbf{u}_L, \mathbf{u}_R) = \frac{1}{2}\left(\mathcal{F}_L + \mathcal{F}_R - \hat{\mathbf{T}}|\hat{\mathbf{\Lambda}}|\hat{\mathbf{T}}^{-1}(\mathbf{u}_R - \mathbf{u}_L)\right), \tag{A.1}$$

where $\hat{\mathbf{T}}\hat{\mathbf{\Lambda}}\hat{\mathbf{T}}^{-1}$ is the diagonalization of $\hat{\mathbf{A}}$, which is the Roe-averaged linear approximation of the Jacobian $\partial\mathcal{F}/\partial\mathbf{u}$.

The entropy fix affects eigenvalues (entries of the diagonal matrix $\hat{\mathbf{\Lambda}}$) which are close to zero. This is done by replacing all the components ($\hat{\lambda}$) of $\hat{\mathbf{\Lambda}}$ in A.1 by $\beta(\hat{\lambda})$, where

$$\beta(\hat{\lambda}) = \begin{cases} |\hat{\lambda}| & |\hat{\lambda}| \geq \epsilon \\ (\hat{\lambda}^2 + \epsilon^2)/(2\epsilon) & |\hat{\lambda}| < \epsilon \end{cases}$$

In this work, $\epsilon = 0.01$ was used.

Appendix B

Boundary Conditions

This appendix describes how various boundary conditions were imposed in two and three dimensions. In the following description, the $()^+$ and $()^-$ notation denotes the trace value taken from the interior and exterior of the domain, respectively.

Full state vector

The simplest boundary condition is the specification of the full exterior state vector, $\mathbf{u}^-$. The Riemann solver is then used with the known interior state, $\mathbf{u}^+$, to calculate the flux at the boundary,

$$\mathcal{H}_{\text{bd}}(\mathbf{u}^+, \mathbf{u}^-, \hat{\mathbf{n}}) = \mathcal{H}(\mathbf{u}^+, \mathbf{u}^-, \hat{\mathbf{n}}). \tag{B.1}$$

Flow tangency

To enforce flow tangency, the boundary flux is set to be the pressure contribution to the momentum flux. The pressure is calculated based on the interior density, energy, and tangential velocity (by (2.3)),

$$p = p(\rho^+, (\rho\mathbf{v})^+_{||}, (\rho E)^+), \qquad (\rho\mathbf{v})_{||} = (\rho\mathbf{v}) - (\rho\mathbf{v}\cdot\hat{\mathbf{n}})\hat{\mathbf{n}}. \tag{B.2}$$

In the above, $(\rho\mathbf{v})_{||}$ is the component of the momentum vector tangent to the wall. The boundary flux is then computed using the inviscid flux definition, (2.2).

Subsonic Inflow

A subsonic inflow is enforced by specifying the total temperature, T_t, total pressure, p_t, and flow direction: one angle, α, in 2-D, or two angles, α and β, in 3-D. The boundary flux is determined by constructing $\mathbf{u}^-$ from $\mathbf{u}^+$ and the specified parameters, as follows. First,

the Riemann invariant $J^+(\mathbf{u}^+, \hat{\mathbf{n}})$ is calculated as

$$J^+ = \mathbf{v} \cdot \hat{\mathbf{n}} + \frac{2c}{(\gamma - 1)}, \tag{B.3}$$

where c is the speed of sound based on the interior state. J^+ contains all the information from the interior that is used in constructing the exterior state. The inflow Mach number, M, is calculated from J^+ and the specified parameters by solving the following equation:

$$\left(\gamma R T_t d_n^2 - \frac{\gamma - 1}{2}(J^+)^2\right) M^2 + \left(\frac{4\gamma R T_t d_n}{\gamma - 1}\right) M + \frac{4\gamma R T_t}{\gamma - 1} - (J^+)^2 = 0. \tag{B.4}$$

R is the universal gas constant, and $d_n = \hat{\mathbf{n}}_{in} \cdot \hat{\mathbf{n}}$, where $\hat{\mathbf{n}}_{in}$ is the specified inflow direction. The physically relevant solution ($M \geq 0$) is used. Using M and the specified stagnation quantities, the exterior static temperature (T^-), exterior static pressure (p^-), and exterior density (ρ^-) are calculated. The exterior velocity is found from M and the speed of sound, $c^- = \gamma p^- / \rho^-$. Finally, the total energy, $(\rho E)^-$ is found using the exterior pressure, density, and velocity. In this manner, the exterior state, $\mathbf{u}^-$, is calculated as a function of $\mathbf{u}^+$ and the specified parameters. The boundary flux is then calculated using the Riemann flux with $\mathbf{u}^+$ and $\mathbf{u}^-$ as inputs.

Outflow

Enforcing an outflow condition requires the construction of an exterior state based on the interior state and possibly the specified static pressure, p^-. If the interior velocity normal to the boundary is sonic or supersonic, we set $\mathbf{u}^- = \mathbf{u}^+$.

Otherwise, the exterior state is calculated from $J^+(\mathbf{u}^+)$, the interior entropy $S^+ = S(\mathbf{u}^+)$, and the interior tangential velocity $\mathbf{v}_{||}^+$. The calculation proceeds as follows. First, ρ^- is given by

$$\rho^- = \left(\frac{p^-}{S^+}\right)^{1/\gamma}. \tag{B.5}$$

The normal velocity, $\mathbf{v}_\perp$, is found using J^+ and $c^- = \sqrt{\gamma p^- / \rho^-}$,

$$\mathbf{v}_\perp = J^+ - \frac{2c^-}{\gamma - 1}. \tag{B.6}$$

Setting $\mathbf{v}_{||}^- = \mathbf{v}_{||}^+$ fully defines $\mathbf{v}^-$. $(\rho E)^-$ is then calculated using p^-, ρ^-, and $\mathbf{v}^-$. Having constructed $\mathbf{u}^-$, the boundary flux is computed using the Riemann flux.

Appendix C

Hierarchical Basis

A hierarchical basis was used in one and two dimensions for interpolation of the state and residual. This appendix gives the details on the construction of such a basis for the orders used. Much of the following is taken from Solín *et al* [39].

C.1 Kernel Functions

In defining the hierarchical basis, the following one-dimensional high-order kernel functions are used:

$$\begin{aligned}
\varphi_0(x) &= -2\sqrt{\frac{3}{2}}, \\
\varphi_1(x) &= -2\sqrt{\frac{5}{2}}x, \\
\varphi_2(x) &= -\frac{1}{2}\sqrt{\frac{7}{2}}(5x^2-1), \\
\varphi_3(x) &= -\frac{1}{2}\sqrt{\frac{9}{2}}(7x^2-3)x, \\
\varphi_4(x) &= -\frac{1}{4}\sqrt{\frac{11}{2}}(21x^4-14x^2+1), \\
\varphi_5(x) &= -\frac{1}{4}\sqrt{\frac{13}{2}}(33x^4-30x^2+5)x, \\
&\vdots
\end{aligned}$$

The functions $\varphi_k(x)$ are of order k in x, and arise from a decomposition of the Lobatto shape functions. The details of this decomposition are given in [39].

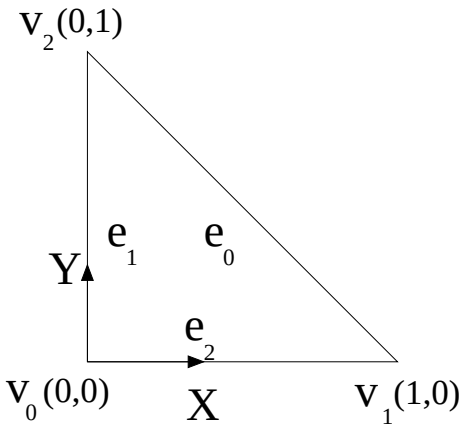

Figure C-1: Reference triangle.

C.2 Hierarchical Basis in Two-dimensions

The reference triangle on which the basis is constructed is shown in Figure C-1. On this triangle, three affine coordinates are defined,

$$\lambda_0(X,Y) = 1 - X - Y, \quad \lambda_1(X,Y) = X, \quad \lambda_2(X,Y) = Y.$$

These coordinates take on the value of 1 on one vertex, and decay linearly to zero on the other two vertices. The sequence of hierarchical basis functions is defined by three types of functions: vertex functions, edge functions, and bubble functions.

Vertex Functions

Three vertex functions are defined as follows:

$$\begin{aligned}\phi^{v_0}(X,Y) &= \lambda_0(X,Y),\\ \phi^{v_1}(X,Y) &= \lambda_1(X,Y),\\ \phi^{v_2}(X,Y) &= \lambda_2(X,Y).\end{aligned}$$

These functions account for $p = 1$ interpolation. For symmetry, the hierarchical basis functions do not begin at $p = 0$.

Edge Functions

For $p \geq 2$, $p-1$ edge functions are defined on each of the three edges. Specifically, for $k = 2, \ldots, p$,

$$\phi_k^{e_0} = \lambda_1 \lambda_2 \varphi_{k-2}(\lambda_2 - \lambda_1),$$

$$\phi_k^{e_1} = \lambda_2 \lambda_0 \varphi_{k-2}(\lambda_0 - \lambda_2),$$
$$\phi_k^{e_2} = \lambda_0 \lambda_1 \varphi_{k-2}(\lambda_1 - \lambda_0).$$

Each edge function vanishes on the two edges with which it is not associated.

Bubble Functions

For $p \geq 3$, $(p-1)(p-2)/2$ bubble functions are defined. These functions are indexed by two indices, n_1 and n_2, with $n_1, n_2 \geq 1$ and $n_1 + n_2 \leq p-1$,

$$\phi_{n_1,n_2}^b = \lambda_0 \lambda_1 \lambda_2 \varphi_{n_1-1}(\lambda_1 - \lambda_0)\varphi_{n_2-1}(\lambda_0 - \lambda_2).$$

The bubble functions vanish on all three edges.

Table C.1 summarizes which basis functions are used for orders of interpolation from $p = 1$ to $p = 5$.

Table C.1: Two-dimensional hierarchical basis functions for various orders.

Order	Vertex	Edge	Bubble	Total
p = 1	3	0	0	3
p = 2	3	3	0	6
p = 3	3	6	1	10
p = 4	3	9	3	15
p = 5	3	12	6	21

Separate vertex, edge, and bubble functions have been used previously, namely by Sherwin and Karniadakis [38], and Szabó and Babuška [40]. The work of these authors differed in the choice of kernel functions and in the definitions of the edge and bubble functions. Specifically, Sherwin and Karniadakis used Jacobi polynomials while Szabó and Babuška used Jacobi polynomials for the edge functions and Legendre polynomials for the bubble functions. In addition, Sherwin and Karniadakis were interested in preserving C^0 continuity between elements and in a consistent product form for ease in integration, and they therefore lost symmetry in their definitions of edge and bubble functions. The edge and bubble functions used in this work were chosen for their symmetry and ease in implementation, and the Lobatto-derived kernel functions were used for their favorable conditioning properties at high order.

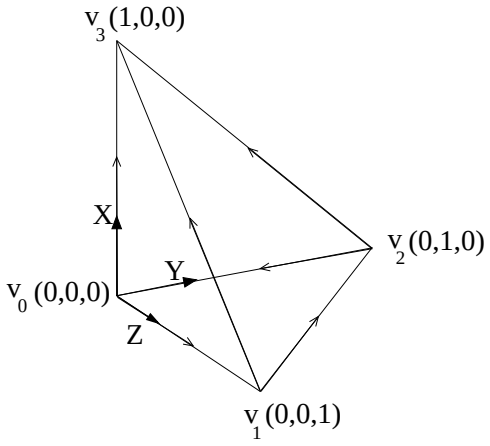

Figure C-2: Reference tetrahedron.

C.3 Hierarchical Basis in Three-dimensions

In three-dimensions the elements are tetrahedra. On the reference tetrahedron, shown in Figure C-2, four affince coordinates are defined,

$$\begin{aligned}
\lambda_0(X,Y,Z) &= 1 - X - Y - Z, \\
\lambda_1(X,Y,Z) &= Z, \\
\lambda_2(X,Y,Z) &= Y, \\
\lambda_3(X,Y,Z) &= X.
\end{aligned}$$

These coordinates take on the value of 1 on one vertex, and decay linearly to zero on the other three vertices. In 3-D, the sequence of hierarchical basis functions is defined by four types of functions: vertex functions, edge functions, face functions, and bubble functions.

Vertex Functions

Four vertex functions are defined as in 2-D and account for $p = 1$ interpolation,

$$\phi^{v_i} = \lambda_i, \qquad i = 1 \ldots 4.$$

Edge Functions

For $p \geq 2$, $p-1$ edge functions are defined on each of the six edges. If $j = 1, \ldots, 6$ is an index over the edges, and $k = 2, \ldots, p$, the basis functions can be expressed as

$$\phi_k^{e_j} = \lambda_{j_0} \lambda_{j_1} \varphi_{k-2}(\lambda_{j_0} - \lambda_{j_1}),$$

where j_0 and j_1 refer to the two vertices that define edge j. In Figure C-2, the arrows on each edge point from j_0 to j_1. Note that each edge function vanishes on the two faces which do not contain that edge.

Face Functions

For $p \geq 3$, $(p-2)(p-1)/2$ face functions are defined on each of the four faces. For a face i, $i = 1, \ldots, 4$, let A, B, and C be the indices of the three vertices which bound the face. For uniqueness, A and C are chosen to have the lowest and highest local index, respectively. The face function associated with face i is then given by

$$\phi_k^{s_i} = \lambda_A \lambda_B \lambda_C \varphi_{n_1-1}(\lambda_B - \lambda_A)\varphi_{n_2-1}(\lambda_A - \lambda_C),$$

where n_1 and n_2 are indices satisfying $n_1, n_2 \geq 1$ and $n_1 + n_2 \leq p-1$.

Bubble Functions

For $p \geq 4$, $(p-3)(p-2)(p-1)/6$ bubble functions are defined. If n_1, n_2, and n_3 are indices satisfying $n_1, n_2, n_3 \geq 1$ and $n_1 + n_2 + n_3 \leq p-1$, the bubble functions can be written as

$$\phi^b_{n_1,n_2,n_3} = \varphi_{n_1-1}(\lambda_0 - \lambda_1)\varphi_{n_2-1}(\lambda_2 - \lambda_1)\varphi_{n_3-1}(\lambda_3 - \lambda_1)\prod_{i=0}^{3}\lambda_i.$$

The bubble functions vanish on each of the four faces of the tetrahedron.

Table C.2 summarizes which basis functions are used for orders of interpolation from $p = 1$ to $p = 5$. Again, Sherwin and Karniadakis [38] also used a 3-D basis separated into vertex, edge, face, and bubble functions. However, as in 2-D, they used Jacobi polynomials for their kernel functions and did not define the edge and face functions symmetrically.

Table C.2: Three-dimensional hierarchical basis functions for various orders.

Order	Vertex	Edge	Face	Bubble	Total
p = 1	4	0	0	0	4
p = 2	4	6	0	0	10
p = 3	4	12	4	0	20
p = 4	4	18	12	1	35
p = 5	4	24	24	4	56

Bibliography

[1] S. R. Allmaras. Analysis of semi-implicit preconditioners for multigrid solution of the 2-D compressible Navier-Stokes equations. AIAA Paper Number 95–1651-CP, 1995.

[2] W. K. Anderson, R. D. Rausch, and D. L. Bonhaus. Implicit/multigrid algorithms for incompressible turbulent flows on unstructured grids. *J. Comput. Phys.*, 128:391–408, 1996.

[3] Abdelkader Baggag, Harold Atkins, and David Keyes. Parallel implementation of the discontinuous Galerkin method. Technical Report Report 99-35, ICASE, 1999.

[4] F. Bassi and S. Rebay. High-order accurate discontinuous finite element solution of the 2-D Euler equations. *Journal of Computational Physics*, 138:251–285, 1997.

[5] F. Bassi and S. Rebay. A high-order discontinuous finite element method for the numerical solution of the compressible Navier-Stokes equations. *Journal of Computational Physics*, 131:267–279, 1997.

[6] F. Bassi and S. Rebay. An implicit high-order discontinuous Galerkin method for the steady state compressible Navier-Stokes equations. *Computational Fluid Dynamics 98, Proceedings from the Fourth European Computational Fluid Dynamics Conference*, 2:1227–1233, 1998.

[7] Achi Brandt. *Guide to Multigrid Development.* Springer-Verlag, 1982.

[8] Achi Brandt. Barriers to achieving textbook multigrid efficiency (TME) in CFD. ICASE Interim Report No. 32 NASA/CR-1998-207647, 1998.

[9] William Briggs, Van Emden Henson, and Steve F. McCormick. *A Multigrid Tutorial, 2nd Ed.* SIAM, 2000.

[10] Bernardo Cockburn and Chi-Wang Shu. Runge-Kutta discontinuous Galerkin methods for convection-dominated problems. *Journal of Scientific Computing*, pages 173–261, 2001.

[11] A. Crivellini and F. Bassi. A three-dimensional parallel discontinuous Galerkin solver for acoustic propagation studies. *International Journal of Aeroacoustics*, 2(2):157–173, 2003.

[12] D. Darmofal and B. van Leer. Local preconditioning: Manipulating mother nature to fool father time, in: M. Hafez, D.A. Caughey (Eds.), *Computing the Future II: Advances and Prospects for Computational Aerodynamics*, Wiley, New York, 1998.

[13] D. L. Darmofal and P. J. Schmid. The importance of eigenvectors for local preconditioners of the Euler equations. *Journal of Computational Physics*, 127:346–362, 1996.

[14] J. A. Désidéri and P. W. Hemker. Convergence analysis of the defect-correction iteration for hyperbolic problems. *SIAM J. Sci. Statist. Comput.*, 16:88–118, 1995.

[15] Suchuan Dong and George E. Karniadakis. Multilevel parallelization models for high-order CFD. AIAA 2004-1087, 2004.

[16] William D. Gropp, Dinesh K. Kaushik, David E. Keyes, and Barry F. Smith. High-performance parallel implicit CFD. *Parallel Computing*, 27:337–362, 2001.

[17] B. T. Helenbrook, D. J. Mavriplis, and H. A. Atkins. Analysis of p-multigrid for continuous and discontinuous finite element discretizations. AIAA Paper 2003-3989, 2003.

[18] A. Jameson. Solution of the Euler equations for two-dimensional transonic flow by a multigrid method. *Applied Mathematics and Computation*, 13:327–356, 1983.

[19] A. Jameson. Computational algorithms for aerodynamic analysis and design. *Applied Numerical Mathematics*, 13:383–422, 1993.

[20] A. Jameson, T. J. Baker, and N. P. Weatherhill. Calculation of inviscid transonic flow over a complete aircraft. AIAA 86-0103, 1986.

[21] A. Jameson, W. Schmidt, and E. Turkel. Numerical simulation of the Euler equations by finite volume methods using Runge-Kutta time stepping schemes. AIAA-81-1259, 1981.

[22] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report TR 95-035, Department of Computer Science, University of Minnesota, 1995. Also available on WWW at URL `http://www.cs.umn.edu/~karypis/papers/mlevel_serial.ps`.

[23] Kelly R. Laflin, John C. Vassberg, Richard A. Wahls, Joseph H. Morrison, Olaf Brodersen, Mark Rakowitz, Edward N. Tinoco, and Jean-Luc Godard. Summary of data from the Second AIAA CFD Drag Prediction Workshop. AIAA Paper 2004-0555, 2004.

[24] E. M. Lee-Rausch, N. T. Frink, D. J. Mavriplis, R. D. Rausch, and W. E. Milholen. Transonic drag prediction on a DLR-F6 transport configuration using unstructured grid solvers. AIAA-2004-0554, 2004.

[25] B. Van Leer, W. T. Lee, and P. L. Roe. Characteristic time-stepping or local preconditioning of the Euler equations. AIAA 91-1552, 1991.

[26] David W. Levy, Thomas Zickuhr, John Vassberg, Shreekant Agrawal, Richard A. Wahls, Shahyar Pirzadeh, and Michael J. Hemsch. Data summary from the First AIAA Computational Fluid Dynamics Drag Prediction Workshop. *Journal of Aircraft*, 40(5):875–882, 2003.

[27] D. J. Mavriplis. Multigrid solution of the 2-D Euler equations on unstructured triangular meshes. *AIAA Journal*, 26:824–831, 1988.

[28] D. J. Mavriplis. Multigrid strategies for viscous flow solvers on anisotropic unstructured meshes. *Journal of Computational Physics*, 145:141–165, 1998.

[29] D. J. Mavriplis. An assessment of linear versus nonlinear multigrid methods for unstructured mesh solvers. *Journal of Computational Physics*, 175:302–325, 2001.

[30] D. J. Mavriplis and A. Jameson. Multigrid solution of the Navier-Stokes equations on triangular meshes. *AIAA Journal*, 28:1415–1425, 1990.

[31] D. J. Mavriplis and S. Pirzadeh. Large-scale parallel unstructured mesh computations for 3-D high-lift analysis. *AIAA Journal of Aircraft*, 36:987–998, 1999.

[32] Tolulope O. Okusanya. *Algebraic Multigrid for Stabilized Finite Element Discretizations of the Navier-Stokes Equations*. PhD dissertation, M.I.T., Department of Aeronautics and Astronautics, June 2002.

[33] Todd A. Oliver, Krzysztof J. Fidkowski, and David L. Darmofal. Multigrid solution for high-order discontinuous Galerkin discretization of the compressible Navier-Stokes equations (to appear). In *Third International Conference on Computational Fluid Dynamics, Toronto, Canada*, 2004.

[34] Niles A. Pierce and Michael B. Giles. Preconditioned multigrid methods for compressible flow calculations on stretched meshes. *Journal of Computational Physics*, 136:425–445, 1997.

[35] P. L. Roe. Approximate Riemann solvers, parametric vectors, and difference schemes. *Journal of Computational Physics*, 43:357–372, 1981.

[36] P. L. Roe. Characteristic-based schemes for the Euler equations. *Ann. Rev. Fluid Mech.*, 18:337–65, 1986.

[37] E. M. Rønquist and A. T. Patera. Spectral element multigrid I. Formulation and numerical results. *J. Sci. Comput.*, 2(4):389–406, 1987.

[38] Spencer J. Sherwin and George E. Karniadakis. A new triangular and tetrahedral basis for high-order finite element methods. *Int. J. Num. Meth. Eng.*, 38:3775–3802, 1995.

[39] Pavel Solín, Karel Segeth, and Ivo DoleŽel. *Higher-Order Finite Element Methods*. Chapman and Hall, 2003.

[40] Barna Szabó and Ivo Babuška. *Finite Element Method*. John Wiley & Sons, 1991.

[41] E. Turkel. Preconditioned methods for solving the incompressible and low speed compressible equations. *Journal of Computational Physics*, 72:277–298, 1987.

[42] B. Van Leer. Flux-vector splitting for the Euler equations. Technical Report 81-11, ICASE, 1981.

[43] B. Van Leer. Upwind-difference methods for aerodynamic problems governed by the Euler equations. *Lectures in Applied Mathematics*, 22, 1985.

[44] B. Van Leer, J. Thomas, P. Roe, and R. Newsome. A comparison of numerical flux formulas for the Euler and Navier-Stokes equations. AIAA Paper 87-1104, 1987.

[45] John C. Vassberg, Mark A. DeHaan, and Tony J. Sclafani. Grid generation requirements for accurate drag predictions based on OVERFLOW calculations. AIAA-2003-4124, 2003.

[46] V. Venkatakrishnan, S. R. Allmaras, D. S. Kamenetskii, and F. T. Johnson. Higher order schemes for the compressible Navier-Stokes equations. AIAA Paper 2003-3987, 2003.

[47] G. Volpe. On the use and accuracy of compressible flow codes at low Mach numbers. AIAA 91-1662, 1991.

[48] J. S. Wong, D. L. Darmofal, and J. Peraire. The solution of the compressible Euler equations at low Mach numbers using a stabilized finite element algorithm. *Computer Methods in Applied Mechanics and Engineering*, 190:5719–5737, 2001.