18th AIAA Computational Fluid Dynamics Conference
25 - 28 June 2007, Miami, FL

AIAA 2007-3942

Discontinuous Galerkin solutions of the Navier-Stokes Equations using Linear Multigrid Preconditioning

Laslo T. Diosady *, David L. Darmofal †
Aerospace Computational Design Laboratory, Massachusetts Insititute of Technology

A Newton-Krylov method is developed for the solution of the steady compressible Navier-Stokes equations using a Discontinuous Galerkin (DG) discretization on unstructured meshes. An element Line-Jacobi preconditioner is presented which solves a block tridiagonal system along lines of maximum coupling in the flow. An incomplete block-LU factorization (Block-ILU(0)) is also presented as a preconditioner, where the factorization is performed using a reordering of elements based upon the lines of maximum coupling. This reordering is shown to be superior to standard reordering techniques (Nested Dissection, One-way Dissection, Quotient Minimum Degree, Reverse Cuthill-Mckee) especially for viscous test cases. The Block-ILU(0) factorization is performed in-place and a novel algorithm is presented for the application of the linearization which reduces both the memory and CPU time over the traditional dual matrix storage format. A linear p-multigrid algorithm using element Line-Jacobi and Block-ILU(0) smoothing is presented as a preconditioner to GMRES. The linear multigrid preconditioner is shown to significantly improve convergence in terms of the number of linear iterations as well as to reduce the total CPU time required to obtain a converged solution.

I. Introduction

Discontinuous Galerkin (DG) discretizations have become increasingly popular for achieving accurate solutions of conservation laws. Specifically, DG discretizations have been widely used to solve the Euler and Navier-Stokes equations for convection-dominated problems.[1–6] DG methods are attactive since the elementwise discontinuous representation of the solution provides a natural way of achieving higher-order accuracy on arbitrary, unstructured meshes. A detailed overview of DG methods for the discretization of the Euler and Navier-Stokes equations is provided by Cockburn and Shu.[5] They, among others,[7,8] have noted that while DG discretizations have been extensively studied, development of solution methods ideally suited for solving these discretizations have lagged behind. In this work a Newton-GMRES approach using a linear multigrid preconditioner is developed as a solution method for DG discretizations of the steady state Navier-Stokes equations.

The use of multigrid to solve DG discretizations of compressible flows was first presented by Fidkowski[9] and Fidkowski et al.[7] Fidkowski et al. used a p-multigrid scheme with an element-line smoother to solve the non-linear system of equations. The Newton-GMRES approach has been widely used for finite volume discretizations of the Euler and Navier-Stokes equations.[10–16] In the context of DG discretizations, GMRES was first used to solve the steady 2D compressible Navier-Stokes equations by Bassi and Rebay.[3,17] GMRES has also been used for the solution of the linear system arising at each iteration of an implicit time stepping scheme for the DG discretization of the time dependent Euler or Navier-Stokes equations.[18–20] Persson and Peraire[20] developed a two level scheme as a preconditioner to GMRES to solve the linear system at each step of an implicit time stepping scheme. They used an ILU(0) smoother for the desired p and solved a coarse grid problem ($p = 0$ or $p = 1$) exactly. Recently, several other authors have used p-multigrid methods to solve DG discretizations of the Euler or Navier-Stokes equations.[8,21–23] Natase and Mavriplis[8,22] used both p-multigrid (where coarse solutions are formed by taking lower order approximations within each element), and hp-multigrid, where an h-multigrid scheme was used to provide a solution update for the $p = 0$ approximation. Natase and Mavriplis used this hp-multigrid scheme with an element Block-Jacobi smoother to solve the non-linear system as well as to solve the linear system arising from a Newton scheme.

*Doctoral candidate, 77 Massachusetts Ave. 37-401, Cambridge MA, 02139, diosady@mit.edu
†Associate Professor, Senior Member AIAA, 77 Massachusetts Ave. 37-401, Cambridge MA, 02139, darmofal@mit.edu

This work presents a linear p-multigrid scheme as a preconditioner to GMRES for the solution of the steady state Euler and Navier-Stokes equations using a DG discretization. While results presented here are used to solve steady state problems, the methods are also suitable for solving time dependent problems. An overview of the DG discretization and the Newton-Krylov approach for solving systems of non-linear conservation laws is presented in Section II. Section III presents the Block-Jacobi, Line-Jacobi and Block-ILU(0) stationary iterative methods that are used as single-level preconditioners or as smoothers on each level of the linear multigrid preconditioner. By considering the Block-ILU preconditioner as a stationary iterative method, a memory efficient implementation is developed which requires no additional storage for the incomplete factorization, while reducing the total time required per linear iteration compared to the traditional dual matrix storage format. Section IV presents a new matrix reordering algorithm for the Block-ILU factorization based upon lines of maximum coupling between elements in the flow. This line reordering algorithm is shown to significantly improve the convergence behaviour, especially for viscous problems. Section V presents the linear multigrid algorithm and discusses memory considerations involved in the development of a memory efficient preconditioner. Finally, Section VI presents numerical results which show that the linear multigrid algorithm reduces both the number of linear iterations and the time required to obtain a converged solution.

II. Solution Method

II.A. DG Discretization

The time dependent, compressible Navier-Stokes equations using index notation are given by:

$$\partial_t u_k + \partial_i F_{ki}(\mathbf{u}) - \partial_i F^v_{ki}(\mathbf{u}) \quad = \quad 0, \qquad k \in [1, n_s] \tag{1}$$

where u_k is the k^{th} component of the conservative state vector $\mathbf{u} = [\rho, \rho v_i, \rho E]$, ρ is the density, v_i are the component of the velocity, and E is the total energy. For two- and three- dimensional flows, $n_s = 4$ and 5, respectively (assuming turbulence modeling or other equations are not included). $F_{ki}(\mathbf{u})$ and $F^v_{ki}(\mathbf{u})$ are inviscid and viscous flux components, respectively, such that Equation (1) is a compact notation for the conservation of mass, momentum, and energy.

The DG discretization of the Navier-Stokes equations is obtained by choosing a triangulation T_h of the computational domain Ω composed of triangular elements κ, and obtaining a solution in $\mathcal{V}^p_h$, the space of piecewise polynomials of order p, which satisfies the weak form of the equation. We define $\mathbf{u}_h$ to be the approximate solution in $(\mathcal{V}^p_h)^{n_s}$, while $\mathbf{v}_h \in (\mathcal{V}^p_h)^{n_s}$ is an arbitrary test function. The weak form is obtained by multiplying Equation (1) by the test functions and integrating over all elements. The weak form is given by

$$\sum_{\kappa \in T_h} \int_\kappa v_k \partial_t u_k dx + \mathcal{R}_h(\mathbf{u}_h, \mathbf{v}_h) = 0, \tag{2}$$

where,

$$\mathcal{R}_h(\mathbf{u}_h, \mathbf{v}_h) \quad = \quad \sum_{\kappa \in T_h} \left[\mathbb{E}_\kappa(\mathbf{u}_h, \mathbf{v}_h) + \mathbb{V}_\kappa(\mathbf{u}_h, \mathbf{v}_h) \right] \tag{3}$$

$$\mathbb{E}_\kappa(\mathbf{u}_h, \mathbf{v}_h) \quad = \quad -\int_\kappa \partial_i v_k F_{ki} dx + \int_{\partial\kappa} v^+_k \hat{F}_{ki}(\mathbf{u}^+_h, \mathbf{u}^-_h) \hat{n}_i ds \tag{4}$$

and $\mathbb{V}_\kappa(\mathbf{u}_h, \mathbf{v}_h)$ is the discretization of the viscous terms. In Equation (4), $()^+$ and $()^-$ denote values taken from the inside and outside faces of an element, while $\hat{n}$ is the outward-pointing unit normal. $\hat{F}_{ki}(\mathbf{u}^+_h, \mathbf{u}^-_h)\hat{n}_i$ is the Roe numerical flux function approximating $F_{ki}\hat{n}_i$ on the element boundary faces.[24] The viscous terms, $\mathbb{V}_\kappa(\mathbf{u}_h, \mathbf{v}_h)$ are discretized using the BR2 scheme of Bassi and Rebay.[3] The BR2 scheme is used because it achieves optimal order of accuracy while maintaining a compact stencil with only nearest neighbour coupling. Further details of the discretization of the viscous terms may be found in Fidkowski et al.[7]

The discrete form of the equations is obtained by choosing a basis for the space $\mathcal{V}^p_h$. The solution vector $\mathbf{u}_h(x,t)$ may then be expressed as a linear combination of basis functions $\mathbf{v}_{h_i}(x)$ where the coefficients of expansion are given by the discrete solution vector $\mathbf{U}_h(t)$, such that:

$$\mathbf{u}_h(x,t) \quad = \quad \sum_i \mathbf{U}_{h_i}(t) \mathbf{v}_{h_i}(x) \tag{5}$$

Two sets of basis functions are used in the context of this work: a nodal Lagrange basis and a hierarchical basis. Further details of the bases may be found in Fidkowski et al.[7]

Given a basis for the space $\mathcal{V}_h^p$, the weak form of the Navier-Stokes equations given in Equation (2) can be written in semi-discrete form as:

$$\mathcal{M}_h \frac{d\mathbf{U}_h}{dt} + R_h(\mathbf{U}_h(t)) \quad = \quad 0, \tag{6}$$

where R_h is the discrete non-linear residual such that $R_h(\mathbf{U}_h)_i = \mathcal{R}_h(\mathbf{u}_h, \mathbf{v}_{h_i})$, while $\mathcal{M}_h$ is the mass matrix given by

$$\mathcal{M}_{h_{ij}} = \int_\kappa \mathbf{v}_{h_i} \mathbf{v}_{h_j} dx. \tag{7}$$

Since the basis functions are piecewise polynomials which are non-zero only within a single element, the mass matrix is block-diagonal.

To discretize Equation (6) in time, we introduce a time integration scheme given by:

$$\mathbf{U}_h^{m+1} \quad = \quad \mathbf{U}_h^m - \left(\frac{1}{\Delta t} \mathcal{M}_h + \frac{\partial R_h}{\partial \mathbf{U}_h} \right)^{-1} R_h(\mathbf{U}_h^m). \tag{8}$$

A steady state solution of the Navier-Stokes equations is given by $\mathbf{U}_h$ satisfying:

$$R_h(\mathbf{U}_h) \quad = \quad 0. \tag{9}$$

The steady state solution is obtained by using the time integration scheme given in Equation (8) and increasing the time step Δt, such that $\Delta t \to \infty$. Directly setting $\Delta t = \infty$ is the equivalent of using Newton's method to solve Equation (9), however convergence is unlikely if the initial guess is far from the solution. On the other hand, if the solution is updated using Equation (8), then the intermediate solutions approximate physical states in the time evolution of the flow, and convergence is more likely.

II.B. Linear System

The time integration scheme given by Equation (8) requires the solution of a large system of linear equations of the form $A\mathbf{x} = \mathbf{b}$ at each time step, where

$$A = \frac{1}{\Delta t} \mathcal{M}_h + \frac{\partial R_h}{\partial \mathbf{U}_h} \qquad \mathbf{x} = \Delta \mathbf{U}_h^m \qquad \mathbf{b} = -R_h(\mathbf{U}_h^m). \tag{10}$$

The matrix A is commonly refered to as the Jacobian matrix. Since the Jacobian matrix is derived from the DG discretization, the Jacobian matrix has a block-sparse structure with N_e block rows of size n_b, where N_e is the number of elements in the triangulation T_h, while n_b is the number of unknowns for each element. Here $n_b = n_s \times n_m$, where n_m is the number of modes per state. n_m is a function of the solution order p and the spatial dimension, as summarized in Table 1. Each block row of the Jacobian matrix has a non-zero

p	n_m, 2D	n_m, 3D
0	1	1
1	3	4
2	6	10
3	10	20
4	15	35
p	$\frac{(p+1)(p+2)}{2}$	$\frac{(p+1)(p+2)(p+3)}{6}$

Table 1. Number of modes per element, n_m, as a function of solution order, p

diagonal block, corresponding to the coupling of states within each element, and n_f off-diagonal non-zero blocks corresponding to the coupling of states between neighbouring elements, where n_f is the number of faces per element (3 and 4 for 2D triangular and 3D tetrahedral elements, respectively). When the time step, Δt, is small, the Jacobian matrix is block-diagonally dominant and the linear system is relatively easy to solve iteratively. On the other hand as the time step increases the coupling between neighbouring elements becomes increasingly important and the linear system generally becomes more difficult to solve.

II.C. Linear Solution Method

The block-sparse structure of the Jacobian matrix and the large number of unknowns suggest the use of an iterative method, more specifically a Krylov-subspace method, to solve the linear system. Since the Jacobian matrix is non-symmetric (though structurally symmetric), the method of choice is the restarted GMRES[25,26] algorithm which finds an approximate solution, $\tilde{\mathbf{x}}$, in the Krylov subspace, $\mathcal{K} = \{\mathbf{b}, A\mathbf{b}, A^2\mathbf{b}, ...A^{n-1}\mathbf{b}\}$, that minimizes the L-2 norm of the linear residual $\mathbf{r} = \mathbf{b} - A\tilde{\mathbf{x}}$.

The convergence of the GMRES algorithm has been shown to be strongly dependent upon eigenvalues of the Jacobian matrix, A.[25–27] To improve the convergence properties of GMRES, a preconditioner is used which transforms the linear system $A\mathbf{x} = \mathbf{b}$ into a related system $P^{-1}A\mathbf{x} = P^{-1}\mathbf{b}$ with better convergence properties. Though the preconditioner, P, is presented as a matrix, any iterative method may be used as a preconditioner.

II.D. Residual Tolerance Criterion

When solving the DG discretization of the steady-state Navier-Stokes equations using the time stepping scheme presented in Equation (8), it is often unnecessary to solve the linear system of equations exactly at each iteration. When the time step is small, or the solution estimate is far from the exact solution, the linear system only needs to be solved to a limited tolerance, which depends upon the non-linear residual. Kelley and Keyes[28] considered three phases of a time stepping scheme to solve the steady state Euler equations: the initial, midrange, and terminal phases. Kelley and Keyes proved super-linear convergence of the non-linear residual in the terminal phase of an inexact Newton iteration given sufficient reduction of the linear residual in each iteration. In this section, an exit criterion is developed for the solution of the linear system to realize the super-linear convergence during the terminal phase. To develop this exit criterion, we consider the convergence of Newton's method to solve Equation (9), such that the solution update is given by:

$$\mathbf{U}_h^{m+1} = \mathbf{U}_h^m - \left(\frac{\partial R_h}{\partial \mathbf{U}_h}\right)^{-1} R_h(\mathbf{U}_h^m), \tag{11}$$

where $\mathbf{U}_h^m$ is the approximate solution at iteration m of the Newton's method. Defining $\epsilon_h^m = \mathbf{U}_h - \mathbf{U}_h^m$ to be the solution error at iteration m, quadratic convergence of the error can be proven as $\epsilon_h^m \to 0$. Namely,

$$\left|\left|\epsilon_h^{m+1}\right|\right| \quad = \quad C_1 \left|\left|\epsilon_h^m\right|\right|^2, \tag{12}$$

for some constant C_1.[28] Similarly quadratic convergence of the solution residual is observed,

$$\left|\left|R_h(\mathbf{U}_h^{m+1})\right|\right| \quad = \quad C_2 \left|\left|\mathcal{R}_h(\mathbf{U}_h^m)\right|\right|^2, \tag{13}$$

for some different constant C_2. Based on this observation, an estimate of the reduction in the solution residual may be given by:

$$\frac{\left|\left|R_h(\mathbf{U}_h^{m+1})\right|\right|}{||R_h(\mathbf{U}_h^m)||} \sim \left(\frac{||R_h(\mathbf{U}_h^m)||}{\left|\left|R_h(\mathbf{U}_h^{m-1})\right|\right|}\right)^2 = (d^m)^2, \tag{14}$$

where $d^m = \frac{||R_h(\mathbf{U}_h^m)||}{\left|\left|R_h(\mathbf{U}_h^{m-1})\right|\right|}$, is the decrease factor of the non-linear residual at iteration m. When the expected decrease of the non-linear residual is small, it may not be necessary to solve the linear system at each Newton step exactly to get an adequate solution update. It is proposed that the linear system given by $A_h\mathbf{x}_h = \mathbf{b}_h$ should have a reduction in linear residual proportional to the expected decrease in the non-linear residual. Defining the linear residual at linear iteration k to be $\mathbf{r}_h^k = \mathbf{b}_h - A_h\mathbf{x}_h^k$, the linear system is solved to a tolerance of:

$$\frac{||\mathbf{r}_h^n||}{||\mathbf{r}_h^0||} \quad \leq \quad K(d^m)^2, \tag{15}$$

where K is a user defined constant, typically chosen in the range $K = [10^{-3}, 10^{-2}]$. Since the linear residual is not available at each GMRES iteration and computing this linear residual can be computationally expensive, the preconditioned linear residual norm, $\left|\left|P^{-1}(\mathbf{b}_h - A_h\mathbf{x}_h^k)\right|\right|$, is used, which can be computed essentially for

free at each GMRES iteration. The reduction in the preconditioned residual also provides an estimate of the reduction of the norm of the linear solution error, $\left|\left|A_h^{-1}b - \mathbf{x}_h^k\right|\right|$, since

$$\frac{\left|\left|(A_h^{-1}\mathbf{b}_h - \mathbf{x}_h^k)\right|\right|}{\left|\left|(A_h^{-1}\mathbf{b}_h - \mathbf{x}_h^0)\right|\right|} \quad \leq \quad \kappa\left(P^{-1}A_h\right)\frac{\left|\left|P^{-1}(\mathbf{b}_h - A_h\mathbf{x}_h^k)\right|\right|}{||P^{-1}(\mathbf{b}_h - A_h\mathbf{x}_h^0)||}, \tag{16}$$

where $\kappa\left(P^{-1}A_h\right)$ is the condition number of $P^{-1}A_h$. With increasingly effective preconditioning, $P^{-1}A_h$ approaches the identity matrix and the reduction in the preconditioner residual norm more closely approximates the reduction in the linear solution error.

Since the non-linear residual may increase at some iteration m, the tolerance for the linear system presented in Equation (15) is modified to be:

$$\frac{\left|\left|P^{-1}\mathbf{r}_h^n\right|\right|}{||P^{-1}\mathbf{r}_h^0||} \quad \leq \quad K\left(\min\left\{1, d^m\right\}\right)^2. \tag{17}$$

This criterion for the reduction of the linear residual is then used to determine n, the number of GMRES iterations to perform each Newton step.

III. In-Place Preconditioning

III.A. Stationary Iterative Methods

Stationary iterative methods used to solve the system of linear equations $A\mathbf{x} = \mathbf{b}$ involve splitting the matrix A into two parts such that $A = M + N$, where M in some sense approximates the matrix A and is relatively easy to invert. Since an iterative scheme is typically used directly as a preconditioner to GMRES, M is commonly refered to as the preconditioning matrix. Applying a stationary iterative method, $\mathbf{x}$ is updated using

$$\mathbf{x}^{k+1} \quad = \quad (1-\omega)\mathbf{x}^k + \omega M^{-1}(\mathbf{b} - N\mathbf{x}^k), \tag{18}$$

where ω is the under relaxation factor. An equivalent form of Equation (18) is

$$\mathbf{x}^{k+1} \quad = \quad \mathbf{x}^k + \omega M^{-1}\mathbf{r}^k, \tag{19}$$

where $\mathbf{r}^k$ is the linear residual given by

$$\mathbf{r}^k \quad = \quad \mathbf{b} - A\mathbf{x}^k. \tag{20}$$

In practice, stationary iterative methods involve a preprocessing stage and an iterative stage. The iterative stage involves repeated solution updates according to Equation (18) or Equation (19), where Equation (18) is used if the application of N is computationally less expensive than the application of A, otherwise Equation (19) is used. In addition, if the stationary iterative method is used as a smoother for linear multigrid, then the iterative stage will involve repeated calculation of the linear residual, $\mathbf{r}$, using Equation (20). In the preprocessing stage the matrix A is factorized such that the application of M^{-1}, M, N and A in Equations (18), (19), and (20) may be evaluated at a fraction of the computational cost of the preprocessing stage. In our implementation, the preprocessing stage is performed in place such that the original matrix A is rewritten with a factorization F. As a result the iterative method uses only the memory required to store the original matrix A, with no additional memory storage required for M, M^{-1} or N.

III.B. Block-Jacobi Solver

The first and most basic stationary iterative method used in this work is a Block-Jacobi solver. The Block-Jacobi solver is given by choosing M to be the block-diagonal of the matrix A. In the preprocessing stage each diagonal block is LU factorized and the factorization, F, is stored, where

$$F = \begin{bmatrix} LU(A_{11}) & A_{12} & A_{13} \\ A_{21} & LU(A_{22}) & A_{23} \\ A_{31} & A_{32} & LU(A_{33}) \end{bmatrix}. \tag{21}$$

This factorization allows for the easy application of both M and M^{-1} during the iterative stage. N is given by the off-diagonal blocks of A which are not modified in the preprocessing stage. Table 2 gives the asymptotic operation counts per element for forming F (given A), as well as the application of M^{-1}, M, N and A. The operation counts presented in Table 2 are asymptotic estimates, in that lower order terms in n_b have been ignored. The application of A is computed as the sum of the applications of M and N. Thus, the Block-Jacobi iterative step uses Equation (18), since the application of A is computationally more expensive than the application of N.

Operation	Operation Count	2D	3D
Form F	$\frac{2}{3}n_b^3$	$\frac{2}{3}n_b^3$	$\frac{2}{3}n_b^3$
$x = M^{-1}x$	$2n_b^2$	$2n_b^2$	$2n_b^2$
$y = Mx$	$2n_b^2$	$2n_b^2$	$2n_b^2$
$y = Nx$	$2n_f n_b^2$	$6n_b^2$	$8n_b^2$
$y = Ax$	$2(n_f + 1)n_b^2$	$8n_b^2$	$10n_b^2$

Table 2. Block-Jacobi solver asymptotic operation count per element

III.C. Line-Jacobi Solver

The second stationary iterative method presented in this work is a Line-Jacobi solver. The Line-Jacobi solver is given by forming lines of maximum coupling between elements and solving a block-tridiagonal system along each line. The coupling between elements is determined by using a $p = 0$ discretization of the scalar transport equation:

$$\nabla \cdot (\rho u \phi) - \nabla \cdot (\mu \nabla \phi) = 0 \tag{22}$$

The lines are formed by connecting neighbouring elements with maximum coupling. For purely convective flows, the lines are in the direction of streamlines in the flow. For viscous flows solved using anisotropic grids, the lines within the boundary layer are often in non-streamline directions. Further details of the line formation algorithm are presented in the theses of Fidkowski[9] and Oliver.[29]

For the Line-Jacobi solver, M is given by the block-tridiagonal systems corresponding to the lines of maximum coupling, while N is given by the blocks associated with the coupling between elements across different lines. In the preprocessing stage, M is factorized using a block-variant of the Thomas algorithm given by:

$$F = \begin{bmatrix} LU(A_{11}) & A_{12} & A_{13} \\ A_{21} & LU(A_{22}^{'}) & A_{23} \\ A_{31} & A_{32} & LU(A_{33}^{'}) \end{bmatrix} \tag{23}$$

where, $A_{22}^{'} = A_{22} - A_{21}A_{11}^{-1}A_{12}$ and $A_{33}^{'} = A_{33} - A_{32}A_{22}^{'-1}A_{23}$. The corresponding LU factorization of M is given by:

$$M = \begin{bmatrix} A_{11} & A_{12} & \\ A_{21} & A_{22} & A_{23} \\ & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} I & & \\ A_{21}A_{11}^{-1} & I & \\ & A_{32}A_{22}^{'-1} & I \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} & \\ & A_{22}^{'} & A_{23} \\ & & A_{33}^{'} \end{bmatrix} \tag{24}$$

The factorization given by Equation (23) is stored as opposed to the LU factorization given by Equation (24) to reduce the computational cost of the preprocessing stage. The reduction in computational cost of storing the factorization given by Equation (23) is offset by an increase in the computational cost of applying M and M^{-1} during the iterative stage. The total computational cost for both the preprocessing and iterative stages using the factorization given by Equation (23) is lower than the LU factorization given by Equation (24), as long as the total number of linear iterations is less than the block size, n_b.

Table 3 gives the asymptotic operation counts per element for the preprocessing stage as well as the application of M^{-1}, M, N and A. The application of A is once again computed as a sum of the applications of M and N. As with the Block-Jacobi solver, the solution update for the Line-Jacobi solver is given by Equation (18), since the application of N is computationally less expensive than the application of A.

Operation	Operation Count	2D	3D
Form F	$\frac{14}{3}n_b^3$	$\frac{14}{3}n_b^3$	$\frac{14}{3}n_b^3$
$x = M^{-1}x$	$8n_b^2$	$8n_b^2$	$8n_b^2$
$y = Mx$	$8n_b^2$	$8n_b^2$	$8n_b^2$
$y = Nx$	$2(n_f - 2)n_b^2$	$2n_b^2$	$4n_b^2$
$y = Ax$	$2(n_f + 2)n_b^2$	$10n_b^2$	$12n_b^2$

Table 3. Line-Jacobi solver asymptotic operation count per element

III.D. Block-ILU Solver

The final iterative method presented in this work is a block incomplete-LU factorization (Block-ILU). ILU factorizations have been successfully used as preconditioners for a variety of aerodynamic problems.[10,12–16,20] Typically the LU factorization of a sparse matrix will have a sparsity pattern with significantly more non-zeros, or fill, than the original matrix. The principle of an incomplete-LU factorization is to produce an approximation of the LU factorization of A, which requires significantly less fill than the exact LU factorization. The incomplete LU factorization, $\tilde{L}\tilde{U}$, is computed by performing Gaussian elimination on A but ignoring values which would result in additional fill. The fill level, k, indicates the distance in the sparsity graph of the neighbours in which coupling may be introduced in the ILU(k) factorization. In the context of this work ILU(0) is used, hence no additional fill outside the sparsity pattern of A is permitted. To simplify the notation, for the remainder of this work we use ILU to denote an ILU(0) factorization unless explicitly stated otherwise.

Though incomplete-LU factorizations are widely used, most implementations store both the linearization A and the incomplete factorization $\tilde{L}\tilde{U}$. Since in most aerodynamic applications the majority of the memory is used for the storage of the linearization and its factorization, such duplicate memory storage may limit the size of the problems which may be solved on a given machine.[12,20,30] In this section, an algorithm is developed that performs the incomplete-LU factorization in-place, such that no additional memory is required for the storage of the factorization. This in-place storage format is an enabling feature which allows for the solution of larger and more complex problems on a given machine. Assuming the majority of the memory is used for the storage of the Jacobian matrix and the Krylov vectors, the increase in the size of the problem which may be solved on a given machine is given by $\frac{2+\eta}{1+\eta}$, where η is the ratio of the memory required to store the Krylov vectors to the memory required to store the Jacobian matrix. For a typical range $\eta \in [0.1, 1.0]$, this represents an increase of 50-90% in the size of problem which may be solved.

To develop an ILU implementation where the memory usage is no greater than that required for the Jacobian, we consider the ILU factorization as a stationary iterative method. In the context of stationary iterative methods, M is given by the product $\tilde{L}\tilde{U}$. It can be easily shown that A differs from M only where fill is dropped in the incomplete LU factorization. Correspondingly, N is given by a matrix containing all fill which was ignored in the ILU factorization. To construct an in-place storage for ILU, note that both A and N may be reconstructed from $\tilde{L}\tilde{U}$ given the original sparsity pattern of A. Namely, A may be computed by taking the product $\tilde{L}\tilde{U}$ and ignoring those values not within the original sparsity pattern. Similarly N can be computed by taking the values of $-\tilde{L}\tilde{U}$ outside the sparsity pattern of A. Though recomputing A and N in this manner is possible, it is impractical since the computational cost is of the same order as the original ILU factorization and requires additional memory storage. Fortunately, only the application of A or N is required, and these products can be computed efficiently using $\tilde{L}$ and $\tilde{U}$.

The remainder of this section describes the implementation and computational efficiency of the in-place Block-ILU solver. The operation count estimates for the Block-ILU solver is based on the assumption that no three elements in the computational grid all neighbour one another. While this assumption may be violated for some computational grids, such violations occur infrequently, such that the analysis based on this assumption is sufficient. The actual implementation does not make this assumption.

In the preprocessing stage, the block incomplete-LU factorization of A is performed in-place where A is

replaced by the factorization F. An example of one step of the factorization is given below:

$$\begin{bmatrix} A_{11} & & A_{13} & & A_{15} & A_{16} \\ & A_{22} & & & & \\ A_{31} & & A_{33} & & & \\ & & & A_{44} & & \\ A_{51} & & & & A_{55} & \\ A_{61} & & & & & A_{66} \end{bmatrix} \Rightarrow \begin{bmatrix} LU(A_{11}) & & A_{13} & & A_{15} & A_{16} \\ & A_{22} & & & & \\ (A_{31}A_{11}^{-1}) & & A_{33}^{'} & & & \\ & & & A_{44} & & \\ (A_{51}A_{11}^{-1}) & & & & A_{55}^{'} & \\ (A_{61}A_{11}^{-1}) & & & & & A_{66}^{'} \end{bmatrix}$$

Where $A_{33}^{'} = A_{33} - A_{31}A_{11}^{-1}A_{13}$, $A_{55}^{'} = A_{55} - A_{51}A_{11}^{-1}A_{15}$, and $A_{66}^{'} = A_{66} - A_{61}A_{11}^{-1}A_{16}$. Based on the assumption that no three elements all neighbour one another, only two of the blocks A_{ij}, A_{ik}, and A_{jk} may be non-zero for any $i \neq j \neq k$. This implies that when eliminating row i only elements A_{ji} and A_{jj}, $j \geq i$ are modified. In addition, fill is ignored at A_{jk} and A_{kj}, if elements $j, k > i$ both neighbour element i. In the general case where the assumption is violated, A_{jk} and A_{kj} are non-zero, and these terms are modified in the Block-ILU factorization such that: $A_{jk}^{'} = A_{jk} - A_{ji}A_{ii}^{-1}A_{ik}$ and $A_{kj}^{'} = A_{kj} - A_{ki}A_{ii}^{-1}A_{ij}$. The number of non-zero blocks in the matrix N is given by $\sum_{i=1}^{N_e} \tilde{n}_{f_i}(\tilde{n}_{f_i} - 1)$ where, $\tilde{n}_{f_i}$ is the number of larger ordered neighbours of element i. While the number of non-zero blocks is dependent upon the ordering of the elements in the ILU factorization, it is possible to obtain an estimate by assuming an ordering exists where, $\tilde{n}_{f_i} = \lceil \frac{i}{N_e} n_f \rceil$. The corresponding estimate for the number of non-zero blocks in N is $N_e(n_f^2 - 1)/3$.

In the iterative stage, the application of M^{-1} is performed using backward and forward substitution of $\tilde{L}$ and $\tilde{U}$. The application of A is performed by multiplying by those components of $\tilde{L}$ and $\tilde{U}$ which would not introduce fill outside the original sparsity pattern of A. Similarly, the application of N may be performed by multiplying by the components of $\tilde{L}$ and $\tilde{U}$ which introduce fill outside the original sparsity pattern of A.

The application of A and N is best illustrated with a simple example. Consider the 3×3 matrix A below, and the corresponding ILU factorization, $\tilde{L}\tilde{U}$:

$$A = \begin{bmatrix} 4 & 5 & -6 \\ 8 & 3 & 0 \\ -12 & 0 & 26 \end{bmatrix} \qquad \tilde{L} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -3 & 0 & 1 \end{bmatrix} \qquad \tilde{U} = \begin{bmatrix} 4 & 5 & -6 \\ 0 & -7 & 0 \\ 0 & 0 & 8 \end{bmatrix}$$

The corresponding matrices M, N and F are given by:

$$M = \begin{bmatrix} 4 & 5 & -6 \\ 8 & 3 & -12 \\ -12 & -15 & 26 \end{bmatrix} \qquad N = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 12 \\ 0 & 15 & 0 \end{bmatrix} \qquad F = \begin{bmatrix} 4 & 5 & -6 \\ 2 & -7 & 0 \\ -3 & 0 & 8 \end{bmatrix}$$

The application of A to a vector x, may be performed by multiplying x by those components of $\tilde{L}$ and $\tilde{U}$ which would not introduce fill outside the original sparsity pattern of A. For the sample matrix, fill was ignored in the ILU factorization at (2,3) and (3,2) when eliminating row 1. Hence, for the sample matrix the application of A may be performed as follows:

$$\begin{aligned} y_1 = &\quad \tilde{U}_{11}x_1 + \tilde{U}_{12}x_2 + \tilde{U}_{13}x_3 && = 4x_1 + 5x_2 - 6x_3 \\ y_2 = &\quad \tilde{L}_{21}\tilde{U}_{11}x_1 + \tilde{L}_{21}\tilde{U}_{12}x_2 + \cancel{\tilde{L}_{21}\tilde{U}_{13}x_3} + \tilde{U}_{22}x_2 && = 2(4x_1) + 2(5x_2) - 7x_2 \\ y_3 = &\quad \tilde{L}_{31}\tilde{U}_{11}x_1 + \cancel{\tilde{L}_{31}\tilde{U}_{12}x_2} + \tilde{L}_{31}\tilde{U}_{13}x_3 + \tilde{U}_{33}x_3 && = -3(4x_1) - 3(-6x_3) + 8x_3 \end{aligned}$$

Clearly, the operation count for computing the application of A in this manner is more expensive than simply applying A in the original form. However, it is important to recognize that in the case of block matrices, each of the terms $\tilde{L}_{ij}$ and $\tilde{U}_{ij}$ are matrices and x_i's are vectors, and hence the (matrix-vector) multiplications become significantly more expensive than the (vector) additions. Hence, to leading order, the computational cost is given by the number of matrix-vector multiplications. The total number of multiplications may be reduced by recognizing that certain products ($\tilde{U}_{11}x_1$, $\tilde{U}_{12}x_2$, $\tilde{U}_{13}x_3$) are repeated. Taking advantage of the structure of the matrix A, based on the assumption that no three elements neighbour one another, it is possible to show that the application of A using $\tilde{L}\tilde{U}$ may be performed at a computational cost of $2(\frac{3}{2}n_f + 1)n_b^2 N_e$.

The application of N is performed by multiplying those components of $\tilde{L}$ and $\tilde{U}$ which would introduce fill outside the original sparsity pattern of A. For the sample matrix, fill was ignored at (2,3) and (3,2) when eliminating row 1. Hence, the application of N to a vector x may be performed as follows:

$$\begin{aligned} y_1 = & & = 0 \\ y_2 = & -\tilde{L}_{21}\tilde{U}_{13}x_3 & = -2(-6x_3) = 12x_3 \\ y_3 = & -\tilde{L}_{31}\tilde{U}_{12}x_2 & = 3(5x_2) = 15x_2 \end{aligned}$$

Once again, the computational cost is dominated by (matrix-vector) multiplications, and additional efficiency may be attained by recognizing that some products may be repeated. The operation count for the application of N is a function of $\tilde{n}_{f_i}$, the number of larger ordered faces of each element. While the operation count for the application of N is dependent upon the ordering of the elements in the ILU factorization, it is possible to obtain an estimate by assuming an ordering exists where, $\tilde{n}_{f_i} = \lceil \frac{i}{N_e} n_f \rceil$. The corresponding estimate for the operation count for applying N is given by $2/3(n_f + 4)(n_f - 1)n_b^2 N_e$.

This estimate of the operation count for the application of N tends to overestimate actual operation counts for practical computational grids. A revised estimate for the application of N may be obtained by considering a particular reordering algorithm based on lines of maximum coupling which is presented in Section IV. Using the ordering of the elements based upon lines effectively reduces the number of free faces for all but the first element in each line since at least one of the faces corresponds to a lower ordered neighbour. The revised estimate for the operation count for the application of N may then be obtained by replacing n_f by $n_f - 1$ in the initial estimate given above. Namely, the revised estimate for the operation count is given by: $\frac{2}{3}(n_f + 3)(n_f - 2)n_b^2 N_e$.

Dim	Type	# Elements	p	Timing
2D	**Estimate**			**0.50**
	Structured	2432	1	0.78
	Unstructured	7344	1	0.84
	Cut Cell	1250	1	0.69
	Structured	2432	4	0.51
	Unstructured	7344	4	0.52
	Cut Cell	1250	4	0.46
3D	**Estimate**			**0.93**
	Structured	1920	1	0.86
	Unstructured	45417	1	1.02
	Cut Cell	2883	1	0.98
	Structured	1920	3	0.77
	Cut Cell	2883	3	0.85

Table 4. Revised timing estimate for application of N for in-place Block-ILU(0) normalized by a Jacobian vector product

Table 4 shows this revised estimate of the operation count for the application of N normalized by the operation count for the application of A using the traditional dual matrix storage format, for both 2D and 3D problems. Table 4 also shows timing results from several sample 2D and 3D problems. For each grid, timing results are presented for $p = 1$ as well as the largest value of p for which the Jacobian matrix could fit into memory on a single machine. For the $p = 1$ cases the actual timing results exceed the revised estimate. However, for large p the actual timing results closely match the revised estimate in 2D, and are bounded by the revised estimate in 3D. The poorer performance for the $p = 1$ cases may be attributed to the effects of lower order terms in n_b, which become significant since the block size for the $p = 1$ solution is relatively small.

Table 5 shows the asymptotic operation count per element for the preprocessing stage and components of the iterative stage for the Block-ILU solver using the in-place storage format. Note that if the Block-ILU factorization $\tilde{L}\tilde{U}$ is stored as a separate matrix such that the original matrix A is still available, the cost of computing $y = Ax$ is $2(n_f + 1)N_e n_b^2$. Based on the operation counts presented in Table 5, a linear

Operation	Operation Count	2D	3D
Form F	$2(n_f+1)n_b^3$	$8n_b^3$	$10n_b^3$
$x = M^{-1}x$	$2(n_f+1)n_b^2$	$8n_b^2$	$10n_b^2$
$y = Mx$	$2(n_f+1)n_b^2$	$8n_b^2$	$10n_b^2$
$y = Nx$ (Initial Estimate)	$\frac{2}{3}(n_f+4)(n_f-1)n_b^2$	$9\frac{1}{3}n_b^2$	$16n_b^2$
$y = Nx$ (Revised Estimate)	$\frac{2}{3}(n_f+3)(n_f-2)n_b^2$	$4n_b^2$	$9\frac{1}{3}n_b^2$
$y = Ax$	$2(\frac{3}{2}n_f+1)n_b^2$	$11n_b^2$	$14n_b^2$
$y = Ax$ (Full Storage)	$2(n_f+1)n_b^2$	$8n_b^2$	$10n_b^2$

Table 5. Block-ILU solver asymptotic operation count per element

iteration in 2D should be performed using Equation (18) since the application of A is more expensive than the application of N. Based on the initial estimate for the application of N, in 3D it appears as though the cost of applying A is less than applying N and hence a linear iteration should be performed using Equation (19). However, in practice a linear iteration in 3D is also performed using Equation (18) since the revised timing estimate for the application of N is less than the application of A.

III.E. Timing Performance

In the previous sections, timing estimates were presented in terms of the operations counts for the different components of each solver. To verify the validity of these estimates actual timing results were obtained using a sample 2D test grid with 2432 elements using a $p = 4$ discretization. The actual and estimated timing results are presented in Table 6 where the time has been normalized by the cost of a single matrix vector product of the Jacobian matrix. As shown in Table 6 the actual timing results closely match the estimates based on operation counts.

Operation	Block-Jacobi		Line-Jacobi		Block-ILU	
	Estimate	Actual	Estimate	Actual	Estimate	Actual
$x = M^{-1}x$	0.25	0.39	1.00	1.24	1.00	1.16
$y = Nx$	0.75	0.76	0.25	0.28	0.50	0.51
$y = Ax$	1.00	1.14	1.25	1.34	1.38	1.43

Table 6. Solver asymptotic operation count per element normalized by a Jacobian vector product

Table 7 gives the asymptotic operation counts for the different solvers presented in this work. As shown in Table 7, the operation count of performing a linear iteration using the in-place storage format is 25% and 5% less than that using the traditional dual matrix storage format for 2D and 3D respectively. The in-place matrix storage format is superior to the traditional dual matrix storage format since the application of N is computationally less expensive than the application of A. In this case, the dual storage format could be modified to store M and N as opposed to M and A, so that a linear iteration may be performed according to Equation (18). A linear iteration could then be performed faster using the modified dual matrix storage format than the in-place matrix storage format. However, the modified dual matrix storage format would require computing N in the preprocessing stage, such that the total computational time for both the preprocessing and iterative stages would still be faster using the in-place storage format if fewer than approximately $3n_b$ linear iterations are performed.

III.F. In-place ILU Factorization of General Matrices

The in-place ILU algorithm developed in this section has been tailored for DG discretizations and may not be generally applicable to sparse matrices arising from other types of discretizations. While the application of A and N may be computed using the ILU factorization for any sparse matrix, the use of an in-place factorization may be unfeasible due to the number of operations required. The number of non-zero blocks in N and correspondingly, the computational cost for the application of N scales with the square of the number

Preconditioner	2D	3D
Block Jacobi	8	10
Line Jacobi	10	12
Block-ILU In-Place	12	$19\frac{1}{3}$
Block-ILU Dual Storage	16	20

Table 7. Linear iteration asymptotic operation count per element (in multiples of n_b^2)

of off-diagonal blocks in the stencil of A. Similarly, if the assumption that no three elements neighbour one another is removed, the operation count for the application of A using the ILU factorization also scales with the square of the number of off-diagonal blocks in the stencil. The in-place ILU algorithm is feasible for DG discretizations since there is only nearest neighbour coupling, resulting in a stencil with few off-diagonal blocks. On the other hand, discretizations such as high-order finite volume discretizations have much wider stencils, involving 2nd and 3rd order neighbours,[6,16] making the in-place ILU factorization algorithm unfeasible.

IV. ILU Reordering

In the development of an efficient Block-ILU(0) preconditioner for DG discretizations, the ordering of the equations and unknowns in the linear system is critical. Matrix reordering techniques have been widely used to reduce fill in the LU factorization for direct methods used to solve large sparse linear systems.[26] These reordering techniques have also been used with ILU preconditioners of Krylov methods.[12,13,16,31] Benzi et al[31] performed numerical experiments comparing the effect of different reordering techniques on the convergence of three Krylov subspace methods used to solve a finite difference discretization of a linear convection-diffusion problem. They showed that reordering the system of equations can both reduce fill for the incomplete factorization, and improve the convergence properties of the iterative method.[31] Blanco and Zingg[12] compared Reverse Cuthill-Mckee, Nested Dissection, and Quotient Minimum Degree reorderings for ILU(k) factorizations of a finite volume discretization of the Euler Equations. They showed that the Reverse Cuthill-Mckee reordering reduced the fill and resulted in faster convergence for ILU(2). Similarly, Pueyo and Zingg[13] used Reverse Cuthill-Mckee reordering to reduce fill and achieve faster convergence for the finite volume discretization of the Navier-Stokes equations. In the context of ILU(0) factorizations, no additional fill is introduced, hence reordering the system of equations effects only the convergence properties of the iterative method. However, Benzi et al[31] showed that even for ILU(0), reordering the systems of equations can significantly reduce the number of GMRES iterations required to reach convergence.

The effect of several standard reordering techniques are examined in this section. The numerical results for the matrix reordering algorithms were determined using the PETSc package for numerical linear algebra.[32–34] The matrix reordering algorithms presented are those available in the PETSc package; namely Reverse Cuthill-Mckee, Nested-Dissection, One-Way Dissection and Quotient Minimum Degree. In addition, the natural ordering produced by the grid generation is employed. Finally, a new matrix reordering algorithm based on lines of maximum coupling within the flow is developed.

IV.A. Line Reordering

The lines of maximum coupling described in Section III.C may be used to order the elements for ILU preconditioning. Specifically, the elements can be ordered as they are traversed along each line. We note that this does not produce a unique reordering, since each line may be traversed in either the forward or backward directions or the lines themselves may also be reordered. While a systematic approach may be developed to choose an optimal permutation for the lines, the natural ordering produced by the line creation algorithm is used for the test cases presented. For these test cases, reordering the lines according to the standard reordering techniques (Reverse Cuthill-Mckee, Nested-Dissection, One-Way Dissection and Quotient Minimum Degree) or reversing the direction of the lines from the natural ordering did not significantly impact the convergence rate.

IV.B. Numerical Results

To investigate the effectiveness of a reordering based upon lines, numerical results are presented for two representative test cases: an inviscid transonic flow and a subsonic viscous flow. The convergence plots are presented in terms of the number of linear iterations since the computational cost of performing the ILU(0) factorization or a single linear iteration is independent of the matrix reordering when using the traditional dual matrix storage format.

The first test case is an Euler solution of the transonic flow over the NACA 0012 airfoil at a freestream Mach number of $M = 0.75$ and angle of attack of $\alpha = 2.0°$. The flow is solved using a $p = 4$ discretization on an unstructured mesh with 7344 elements. Figure 1 shows the convergence plot of the non-linear residual starting from a converged $p = 3$ solution. The fastest convergence is achieved using the reordering based on lines, which requires only 946 linear iterations for a 10 order drop in residual. One-Way Dissection and Reverse Cuthill-Mckee algorithms also perform well requiring only 1418 and 1611 iterations to converge respectively. On the other hand, Quotient Minimum Degree and Nested Dissection reorderings result in convergence rates which are worse than the natural ordering of the elements.

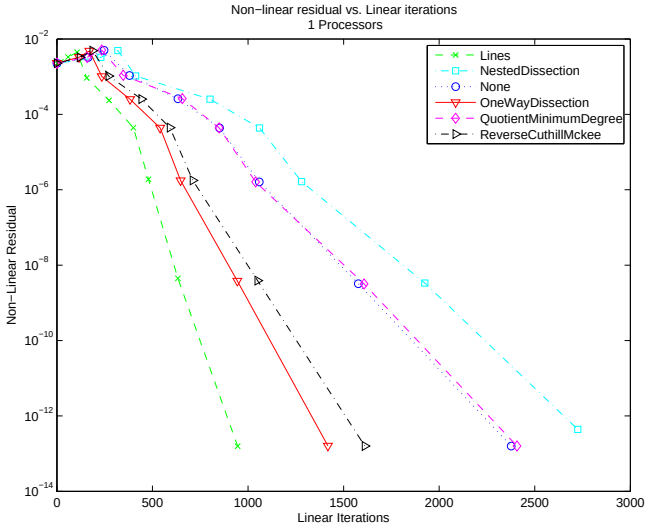

Figure 1. Non-linear residual vs linear iterations using the Block-ILU(0) preconditioner with different reordering techniques for a transonic Euler solution of the flow about the NACA0012 airfoil

The second test case is a Navier-Stokes solution of the subsonic flow over the NACA0012 airfoil at zero angle of attack with a freestream Mach number of $M = 0.5$ and a Reynolds number of $Re = 1000$. A $p = 4$ solution is obtained on a computational mesh with 2432 elements, where the solution procedure is restarted from a converged $p = 3$ solution. Figure 2 presents the convergence plot of the non-linear residual versus linear iterations. The reordering based upon lines is superior to all other reorderings; requiring only 341 iterations to converge. The second best method for this test case is the natural ordering of elements which requires 1350 iterations. The natural reordering performs well for this test case since a structured mesh is used (though the solution procedure does not take advantage of the structure), and hence the natural ordering of the elements involves some inherent structure. Among the other reordering algorithms, Reverse Cuthill-Mckee performs best, requiring 1675 iterations, followed by One-Way Dissection, Quotient Minimum Degree and finally Nested Dissection.

Clearly, reordering the elements according to the lines of maximum coupling results in superior convergence for both inviscid and viscous test cases. The advantages of the line reordering algorithm is especially obvious in the viscous case where reordering according to lines results in a convergence rate nearly 5 times faster than the standard matrix reordering algorithms available in the PETSc package. Due to the clear success of the line reordering algorithm for these two sample problems, the line reordering method is used for the remainder of the work presented here.

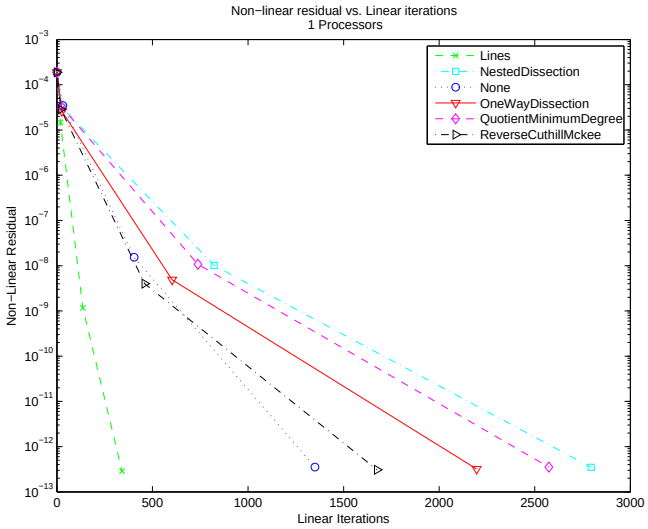

Figure 2. Non-linear residual vs linear iterations using the Block-ILU(0) preconditioner with different reordering techniques for a Navier-Stokes solution of the flow about the NACA0012 airfoil

V. Linear Multigrid

Multigrid algorithms are used to accelerate the solution of systems of equations arising from the discretization of a PDE-based problem by applying corrections based on a coarser discretization with fewer degrees of freedom. The coarse discretization may involve a computational mesh with fewer elements (h-multigrid) or a lower order solution space (p-multigrid). The DG discretization naturally lends itself to a p-multigrid formulation as a coarser solution space may be easily created by using a lower order polynomial interpolation within each element. Multigrid algorithms may be used to directly solve a non-linear system of equations (non-linear multigrid), or to solve the system of linear equations arising at each step of Newton's method (linear multigrid). This section presents a linear p-multigrid algorithm which is used as a preconditioner to GMRES and makes use of the stationary iterative methods presented in Section III as linear smoothers on each multigrid level.

V.A. Linear Multigrid Algorithm

The basic two-level linear-multigrid algorithm is presented below. While only a two-level system is presented here, in general the multigrid formulation involves multiple solution levels.

- Perform pre-smoothing: $\tilde{\mathbf{x}}_h^k = (1-\omega)\mathbf{x}_h^k + \omega M_h^{-1}(\mathbf{b}_h - N_h\mathbf{x}_h^k)$
- Compute linear residual: $\tilde{\mathbf{r}}_h^{\tilde{k}} = \mathbf{b}_h - A_h\tilde{\mathbf{x}}_h^k$
- Restrict linear residual: $\mathbf{b}_H = I_H^h\tilde{\mathbf{r}}_h^k$, where I_H^h is the restriction operator
- Define coarse level correction: $\mathbf{x}_H^0 = 0$
- Perform coarse level smoothing: $\mathbf{x}_H^{j+1} = (1-\omega)\mathbf{x}_H^j + \omega M_H^{-1}(\mathbf{b}_H - N_H\mathbf{x}_H^j)$
- Prolongate coarse level correction: $\hat{\mathbf{x}}_h^k = \tilde{\mathbf{x}}_h^k + I_h^H\mathbf{x}_H$, where I_h^H is the prolongation operator
- Perform post-smoothing: $\mathbf{x}_h^{k+1} = (1-\omega)\hat{\mathbf{x}}_h^k + \omega M_h^{-1}(\mathbf{b}_h - N_h\hat{\mathbf{x}}_h^k)$

As presented in Section II.A, the solution space for the DG discretization is given by $\mathcal{V}_h^p$, the space of piecewise polynomials of order p spanned by the basis functions $\mathbf{v}_{h_i}$. The corresponding coarse solution space is given by $\mathcal{V}_h^{p-1}$, the space of piecewise polynomials of order $p-1$ spanned by the basis functions $\mathbf{v}_{H_k}$.

Since $\mathcal{V}_h^{p-1} \in \mathcal{V}_h^p$, the coarse level basis functions may be expressed as a linear combination of the fine level basis functions:

$$\mathbf{v}_{H_k} = \sum_i \alpha_{ik} \mathbf{v}_{h_i}. \tag{25}$$

The matrix of coefficients α_{ik} form the prolongation operator I_h^H. The coefficients of expansion may also be used to define the restriction operator by considering the restriction of a component of the residual:

$$\mathcal{R}_h(\mathbf{u}_h, \mathbf{v}_{H_k}) = \mathcal{R}_h(\mathbf{u}_h, \sum_i \alpha_{ik} \mathbf{v}_{h_i}) = \sum_i \alpha_{ik} \mathcal{R}_h(\mathbf{u}_h, \mathbf{v}_{h_i}). \tag{26}$$

Hence the restriction operator is given by $I_H^h = \left(I_h^H\right)^T$. In our implementation of the linear multigrid algorithm, the coarse grid Jacobian A_H is given by a simple Galerkin projection of the fine grid Jacobian:

$$A_H = I_H^h A_h I_h^H. \tag{27}$$

V.B. Memory Considerations

For a linear multigrid preconditioner significant additional memory is required for the storage of the lower order Jacobians on each multigrid level. Table 8 shows the additional memory required for all lower order Jacobians in terms of the fine grid Jacobian for $p = 1 \rightarrow 5$.

Several authors[9,30] have argued that a linear multigrid preconditioner may be unfeasible for large problems due to the additional memory cost of storing these lower order Jacobians. Alternatively, others have advocated for skipping multigrid levels to reduce memory usage. For example, Persson and Peraire[20] employed a multi-level scheme where only $p = 0$ and $p = 1$ corrections were applied. Though the linear multigrid method may require significant additional memory for the storage of the lower order Jacobians, faster convergence of the GMRES method is expected and hence fewer Krylov vectors may be required to obtain a converged solution. Hence, to provide a memory equivalent comparison between a single- and multi-level preconditioner, the total memory usage for the Jacobians and Krylov vectors must be considered. In the context of a restarted GMRES algorithm this is equivalent to increasing the GMRES restart value for the single level preconditioner so that the total memory used by the single and multi-level preconditioners is the same. Table 8 also gives the additional memory for the storage of all lower order Jacobians for the linear multigrid solver in terms of the number of solution vectors on the fine grid. These values may also be viewed as the additional number of GMRES vectors allocated for the single-level preconditioner to provide a memory equivalent comparison with the multigrid preconditioner.

	% Fine Jacobian		Solution Vectors	
Solution Order	2D	3D	2D	3D
$p = 1$	11.1%	6.25%	5	6
$p = 2$	27.7%	17.0%	27	43
$p = 3$	46.0%	29.3%	74	146
$p = 4$	64.9%	42.2%	156	369
$p = 5$	84.1%	55.5%	283	778

Table 8. Additional memory usage for lower order Jacobians for linear multigrid as a percent of the fine grid Jacobian and number of fine grid solution vectors

VI. Numerical Results

The performance of the three preconditioners presented in Section III, as well as the linear multigrid preconditioner presented in Section V are evaluated using three representative test cases: an inviscid transonic flow, a turbulent viscous subsonic flow, and a laminar viscous flow.

VI.A. Inviscid transonic flow over NACA0012 airfoil, $M = 0.75$, $\alpha = 2^\circ$

The first test case is an Euler solution of the transonic flow over the NACA0012 airfoil at an angle of attack of $\alpha = 2^\circ$ with a free-stream Mach number of $M = 0.75$. This flow is solved using a $p = 4$ discretization on an unstructured mesh with 7344 elements. The solution procedure is initialized with a $p = 3$ solution of the flow. A GMRES restart value of 40 is used for the linear multigrid preconditioner while a memory equivalent GMRES restart value of 200 is used for the single-level preconditioners. The number of linear iterations taken in each Newton step is determined by the tolerance criterion specified in Equation (17) up to a maximum of 10 GMRES outer iterations. Table 9 shows the convergence results for the different preconditioners in terms on the number of non-linear Newton iterations, linear iterations, GMRES outer iterations and CPU time. The convergence history of the non-linear residual versus CPU time is given in Figure 3. The residual tolerance criterion developed in Section II.C ensures sufficient convergence of the linear system in each Newton step so that quadratic convergence of the non-linear residual is observed for all preconditioners except Block-Jacobi. Additionally, the residual tolerance criterion developed in Section II.C ensures that the convergence history of the non-linear residual in terms of non-linear iterations is the same for these preconditioners. The difference in behaviour for the Block-Jacobi preconditioner is due to stalling of the restarted GMRES algorithm, which prevents a sufficient convergence of the linear system to obtain quadratic convergence.

Preconditioner	Newton Iter	Linear Iter	GMRES Outer	Time(s)
Block-Jacobi	10	15024	78	13596
Line-Jacobi	9	3836	23	3925
Block-ILU	9	971	10	1184
LinearMG w/ Block-Jacobi	9	1511	40	3873
LinearMG w/ Line-Jacobi	9	301	11	1417
LinearMG w/ Block-ILU	9	142	9	934

Table 9. Convergence results of the inviscid transonic NACA0012 test case

Using the single-level Block-ILU preconditioner significantly reduces the number of linear iterations required to converge compared to the single-level Line-Jacobi and Block-Jacobi preconditioners. This improved convergence using the Block-ILU preconditioner ensures that the GMRES restart value is reached only once. On the other hand, the GMRES restart value is reached in each Newton iteration for the Block-Jacobi preconditioner and all but the first three Newton iteration for the Line-Jacobi preconditioner. The repeated restarting of the GMRES algorithm degrades the convergence rate and leads to the stalling of the GMRES algorithm using the Block-Jacobi preconditioner. While both the preprocessing and the iterative stages of the Block-ILU preconditioner are more expensive than the corresponding stages of the Line-Jacobi or Block-Jacobi preconditioners, the significant reduction in the number of linear iterations ensures that the Block-ILU preconditioner achieves fastest convergence in terms of CPU time.

The linear multigrid preconditioners with Block-Jacobi, Line-Jacobi and Block-ILU smoothing significantly reduce the number of linear iterations required to achieve convergence compared to the corresponding single-level preconditioners. The improved convergence rate in terms of the number of linear iterations ensure that the GMRES restart value is not reached as often for the multi-level preconditioners despite the memory equivalent GMRES restart value being five times smaller than the single-level preconditioners. This ensures that GMRES stall is not seen with the linear multigrid preconditioner using Block-Jacobi smoothing. Additionally, the GMRES restart value is reached only twice for the linear multigrid preconditioner with Line-Jacobi smoothing.

Though the linear multigrid preconditioner significantly reduces the number of linear iterations required to converge this problem, the cost of each application of the linear multigrid preconditioner is more expensive than the single level preconditioner. However, fastest convergence in terms of CPU time is achieved using the linear multigrid preconditioner with Block-ILU smoothing which performs about 20% faster than the single level Block-ILU preconditioner.

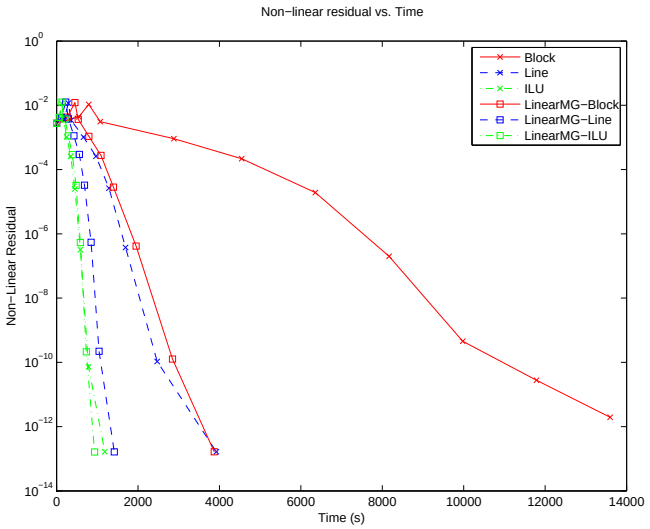

Figure 3. Convergence plot of the inviscid transonic NACA0012 test case

VI.B. Turbulent viscous subsonic flow over a flat plate, $M = 0.25$, $Re = 10^7$

The second test case is a Reynolds-Averaged Navier-Stokes (RANS) solution of a subsonic, $M = 0.25$ flow over a flat plate at a Reynolds number of $Re = 10^7$. The flow is discretized using a $p = 3$ solution on an adapted unstructured anisotropic grid with 466 elements. The flow is initialized with an interpolated $p = 3$ solution based on a previous grid in the adaptation procedure.[35] A GMRES restart value of 40 is used for the linear multigrid preconditioner while a memory equivalent GMRES restart value of 120 is used for the single-level preconditioners. The convergence data is summarized in Table 10, while Figure 4 gives the convergence plot.

Preconditioner	Newton Iter	Linear Iter	GMRES Outer	Time(s)
Block-Jacobi	20	23334	194	1663
Line-Jacobi	16	15527	153	1333
Block-ILU	14	1730	22	326
LinearMG w/ Block-Jacobi	20	7914	198	1996
LinearMG w/ Line-Jacobi	15	2682	72	1203
LinearMG w/ Block-ILU	14	434	15	356

Table 10. Convergence results of the turbulent viscous flat plate test case

The Block-ILU preconditioner performs significantly better than the other single level preconditioners for this test case. The Block-Jacobi solver is unable to converge this problem since the restarted GMRES algorithm stalls. The Line-Jacobi preconditioner also suffers due to the stalling of the restarted GMRES algorithm, such that two additional Newton iterations are required in order to converge the non-linear residual. On the other hand for the Block-ILU preconditioner, the GMRES restart value is reached in only half of the Newton iterations and stalling does not occur.

As in the inviscid test case, the use of the linear multigrid preconditioner significantly reduces the number of linear iterations compared to the single-level preconditioners. For linear multigrid with Block-Jacobi and Line-Jacobi smoothing, this implies that a better solution update is obtained prior to the stalling of the GMRES algorithm. Hence, linear multigrid with Block-Jacobi smoothing is able to reduce the non-linear residual further than using only Block-Jacobi preconditioning. While, linear multigrid with Line-Jacobi smoothing requires one less Newton iteration than the Line-Jacobi preconditioner in order to converged the non-linear residual. Though the linear multigrid preconditioner with Block-ILU smoothing significantly reduces the number linear iterations compared to the single level Block-ILU preconditioner, fastest convergence in terms

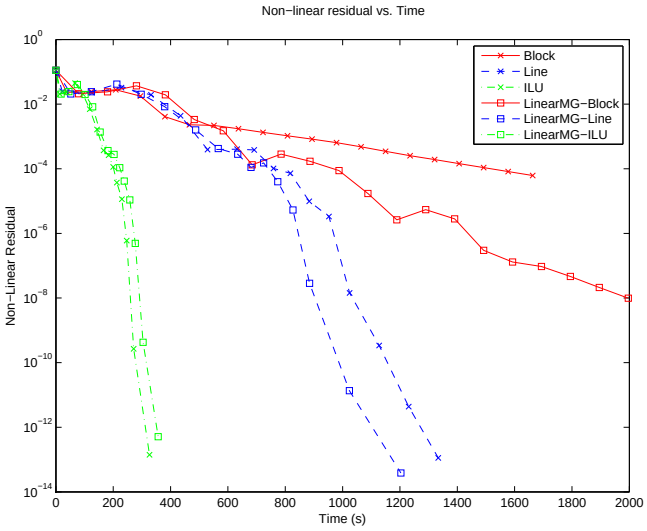

Figure 4. Convergence plot of the turbulent viscous flat plate test case

of CPU time is seen by the single-level Block-ILU preconditioner.

VI.C. Viscous Subsonic flow over NACA0005 airfoil, $M = 0.4$, $\alpha = 0°$, $Re = 50000$

The final test case is a Navier-Stokes solution of a subsonic, $M = 0.4$ flow over the NACA0005 airfoil at zero angle of attack with Reynolds number $Re = 50000$. The flow is discretized using a $p = 3$ solution on an anisotropically adapted unstructured cut-cell mesh with 3470 elements.[36] The flow is initialized with an interpolated $p = 3$ solution based on a previous grid in the adaptation procedure. Once again, a GMRES restart value of 40 is used for the linear multigrid preconditioner while a memory equivalent GMRES restart value of 120 is used for the single-level preconditioners. The convergence data is summarized in Table 11, while Figure 5 gives the convergence plot.

Preconditioner	Newton Iter	Linear Iter	GMRES Outer	Time(s)
Block-Jacobi	26	21524	185	8144
Line-Jacobi	18	8585	79	3679
Block-ILU	17	2559	31	1376
LinearMG w/ Block-Jacobi	17	2722	76	2628
LinearMG w/ Line-Jacobi	17	1130	38	2028
LinearMG w/ Block-ILU	17	600	23	1206

Table 11. Convergence results of the viscous subsonic NACA0005 test case

To achieve fast convergence for this viscous test case, it is necessary that the preconditioner sufficiently resolves the coupling between elements in the boundary layer. Since the Block-Jacobi preconditioner ignores all inter-element coupling, the restarted GMRES algorithm stalls and the linear system is not sufficiently solved such that several additional Newton iterations are required to converge the non-linear residual. On the other hand, the Line-Jacobi and Block-ILU preconditioners which make use of the lines of maximum coupling within the flow are able to sufficiently converge the linear system at each Newton step and super-linear convergence of the non-linear residual is observed.

As with the previous test cases, the use of the linear multigrid preconditioner significantly reduces the number of linear iterations required to converge the linear system at each Newton step. The GMRES restart value is reached less often in the case of the linear multigrid preconditioners despite the GMRES restart value being three times larger for the single-level preconditioners. This ensures that the linear multigrid preconditioner with Block-Jacobi smoothing is able to solve the linear system sufficiently to converge the

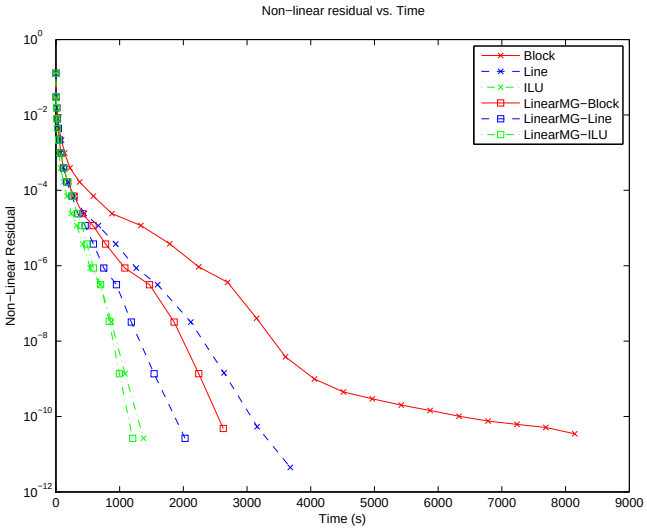

Figure 5. Convergence plot of the viscous subsonic NACA0005 test case

non-linear residual in 17 non-linear iterations as opposed to 26 for the corresponding single-level Block-Jacobi preconditioner. Once again, the fastest convergence in terms of CPU time is achieved using the linear multigrid preconditioner with Block-ILU smoothing.

VII. Conclusions

An efficient, solution algorithm has been presented for the Discontinuous Galerkin discretization of the compressible Navier-Stokes equations on unstructured grids. The algorithm is based on a Newton-Krylov approach with a linear p-multigrid preconditioner using a Block-ILU(0) smoother.

An in-place Block-ILU(0) factorization algorithm has been developed, which has been shown to reduce both the memory and computational cost over the traditional dual matrix storage format. A reordering technique for the Block-ILU(0) factorization, based upon lines of maximum coupling in the flow, has also been developed. The results presented show that this reordering technique significantly reduces the number of linear iterations required to converge compared to standard reordering techniques, especially for viscous test cases.

A linear p-multigrid algorithm has been developed as a preconditioner to GMRES. The linear multigrid preconditioner is shown to significantly reduce the number of linear iterations required to obtain a converged solution compared to a single-level preconditioner. The linear multigrid preconditioner also results in faster convergence in terms of CPU time, as demonstrated by the representative test cases presented.

Acknowledgments

This work was partially supported by funding from The Boeing Company with technical monitor Dr. Mori Mani.

References

[1]Bassi, F. and Rebay, S., "High-order accurate discontinuous finite element solution of the 2D Euler equations," *Journal of Computational Physics*, Vol. 138, No. 2, 1997, pp. 251–285.

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

[3]Bassi, F. and Rebay, S., "GMRES Discontinuous Galerkin solution of the compressible Navier-Stokes equations," *Discontinuous Galerkin Methods: Theory, Computation and Applications*, edited by K. Cockburn and Shu, Springer, Berlin, 2000, pp. 197–208.

[4]Cockburn, B., Karniadakis, G., and Shu, C., "The development of discontinuous Galerkin methods," *Lecture Notes in*

Computational Science and Engineering, Vol. 11, Springer, 2000.

[5]Cockburn, B. and Shu, C.-W., "Runge-Kutta Discontinuous Galerkin methods for convection-dominated problems," *Journal of Scientific Computing*, 2001, pp. 173–261.

[6]Barth, T., "Numerical Methods for Conservation Laws on Structured and Unstructured Meshes," VKI March 2003 Lecture Series, 2003.

[7]Fidkowski, K. J., Oliver, T. A., Lu, J., and Darmofal, D. L., "p-Multigrid Solution of High-Order Discontiunous Galerkin Discretizations of the Compressible Navier-Stokes Equations," *Journal of Computational Physics*, Vol. 207, No. 1, 2005, pp. 92–113.

[8]Nastase, C. R. and Mavriplis, D. J., "High-order discontinuous Galerkin methods using an hp-multigrid approach," *Journal of Computational Physics*, Vol. 213, No. 1, 2006, pp. 330–357.

[9]Fidkowski, K. J., *A High-Order Discontinuous Galerkin Multigrid Solver for Aerodynamic Applications*, Masters thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, June 2004.

[10]Anderson, W., Rausch, R., and Bonhaus, D., "Implicit multigrid algorithms for incompressible turbulent flows on unstructured grids," No. 95-1740-CP, In Proceedings of the 12th AIAA CFD Conference, San Diego CA, 1995.

[11]Cai, X.-C., Gropp, W. D., Keyes, D. E., and Tidriri, M. D., "Newton-Krylov-Schwarz Methods in CFD," In Proceedings of the International Workshop on Numerical Methods for the Navier-Stokes Equations, 1995, pp. 17–30.

[12]Blanco, M. and Zingg, D. W., "A Fast Solver for the Euler Equations on Unstructured Grids using a Newton-GMRES Method," AIAA Paper 1997-0331, Jan. 1997.

[13]Pueyo, A. and Zingg, D. W., "An Efficient Newton-GMRES Solver for Aerodynamic Computations," AIAA Paper 1997-1955, 1997.

[14]Mavriplis, D. J., "On Convergence Acceleration Techniques for Unstructured Meshes," AIAA Paper 1998-2966, 1998.

[15]Knoll, D. A. and Keyes, D. E., "Jacobian-free Newton-Krylov methods: a survey of approaches and applications," *Journal of Computational Physics*, Vol. 193, No. 1, 2004, pp. 357–397.

[16]Nejat, A. and Ollivier-Gooch, C., "Effect of Discretization Order on Preconditioning and Convergence of a Higher-Order Unstructured Newton-Krylov Solver for Inviscid Compressible Flows," AIAA Paper 2007-0719, Jan. 2007.

[17]Bassi, F. and Rebay, S., "Numerical evaluation of two discontinuous Galerkin methods for the compressible Navier-Stokes equations," *International Journal for Numererical Methods in Fluids*, Vol. 40, 2002, pp. 197–207.

[18]Rasetarinera, P. and Hussaini, M. Y., "An Efficient Implicit Discontinuous Spectral Galerkin Method," *Journal of Computational Physics*, Vol. 172, No. 1, 2001, pp. 718–738.

[19]Dolejší, V. and Feistauer, M., "A semi-implicit discontinuous Galerkin finite element method for the numerical solution of inviscid compressible flow," *Journal of Computational Physics*, Vol. 198, No. 1, 2004, pp. 727–746.

[20]Persson, P.-O. and Peraire, J., "An Efficient Low Memory Implicit DG Algorithm for Time Dependent Problems," AIAA Paper 2006-0113, Jan. 2006.

[21]Hillewaert, K., Chevaugeon, N., Geuzaine, P., and Remacle, J.-F., "Hierarchic Multigrid Iteration Strategy for the Discontinuous Galerkin Solution of the Steady Euler Equations," *International Journal for Numerical Methods in Fluids*, Vol. 51, No. 9, 2005, pp. 1157–1176.

[22]Cristian R. Nastase, D. J. M., "High-order Discontinuous Galerkin Methods using a Spectral Multigrid approach," AIAA Paper 2005-1268, Jan. 2005.

[23]Luo, H., Baum, J. D., and Löhner, R., "A p-multigrid discontinuous Galerkin method for the Euler equations on unstructured grids," *Journal of Computational Physics*, Vol. 211, No. 1, 2006, pp. 767–783.

[24]Roe, P. L., "Approximate Riemann Solvers, Parameter Vectors, and Difference Schemes," *Journal of Computational Physics*, Vol. 43, No. 2, 1981, pp. 357–372.

[25]Saad, Y. and Schultz, M. H., "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems," *SIAM Journal on Scientific and Statistical Computing*, Vol. 7, No. 3, 1986, pp. 856–869.

[26]Saad, Y., *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, 1996.

[27]Trefethen, L. N. and Bau, D., *Numerical Linear Algebra*, Society for Industrial and Applied Mathematics, 1997.

[28]Kelley, C. T. and Keyes, D. E., "Convergence Analysis of Pseudo-Transient Continuation," *SIAM Journal of Numerical Analysis*, Vol. 35, No. 2, 1998, pp. 508–523.

[29]Oliver, T. A., *Multigrid Solution for High-Order Discontinuous Galerkin Discretizations of the Compressible Navier-Stokes Equations*, Masters thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, June 2004.

[30]Mavriplis, D. J., "An Assessment of Linear Versus Nolinear Multigrid Methods for Unstructured Mesh Solvers," *Journal of Computational Physics*, Vol. 175, No. 1, 2002, pp. 302–325.

[31]Benzi, M., Szyld, D. B., and van Duin, A., "Orderings for incomplete factorization preconditioning of nonsymmetric problems," *SIAM Journal on Scientific Computing*, Vol. 20, No. 5, 1999, pp. 1652–1670.

[32]Balay, S., Buschelman, K., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Smith, B. F., and Zhang, H., "PETSc Users Manual," Tech. Rep. ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2004.

[33]Balay, S., Gropp, W. D., McInnes, L. C., and Smith, B. F., "Efficient Management of Parallelism in Object Oriented Numerical Software Libraries," *Modern Software Tools in Scientific Computing*, edited by E. Arge, A. M. Bruaset, and H. P. Langtangen, Birkhäuser Press, 1997, pp. 163–202.

[34]Balay, S., Buschelman, K., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Smith, B. F., and Zhang, H., "PETSc Web page," 2007, http://www.mcs.anl.gov/petsc.

[35]Oliver, T. and Darmofal, D., "An Unsteady Adaptation Algorithm for Discontinuous Galerkin Discretizations of the RANS Equations," AIAA Paper 2007-3940, 2007.

[36]Fidkowski, K. and Darmofal, D., "An Adaptive Simplex Cut-Cell Method for Discontinuous Galerkin Discretizations of the Navier-Stokes Equations," AIAA Paper 2007-3941, 2007.