An Efficient Low Memory Implicit DG Algorithm for Time Dependent Problems

Per-Olof Persson* and Jaime Peraire†

Massachusetts Institute of Technology, Cambridge, MA 02139, U.S.A.

We present an efficient implicit time stepping method for Discontinuous Galerkin discretizations of the compressible Navier-Stokes equations on unstructured meshes. The Local Discontinuous Galerkin method is used for the discretization of the viscous terms. For unstructured meshes, the Local Discontinuous Galerkin method is known to produce non-compact discretizations. In order to circumvent the difficulties accociated with this non-compactness, we represent the irregular matrices arising from the discretization algorithm as a product of matrices with a more structured pattern. Time integration is carried out using backward difference formulas. This leads to a non-linear system of equations to be solved at each timestep. In this paper, we study various iterative solvers for the linear systems of equations that arise in the Newton algorithm. We show that a two-level preconditioner with incomplete LU as a pre-smoother is highly efficient yet inexpensive to compute and to store. It performs particularly well for low Mach number flows, where it is more than a magnitude more efficient than pure two-level or ILU preconditioning. Our methods are demonstrated using three typical test problems with various parameters and timesteps.

I. Introduction

Discontinuous Galerkin (DG) methods have become an attractive alternative to the solution of linear first order hyperbolic equations.[1] The ability to obtain very accurate spatial discretizations on arbitrary unstructured meshes combined with inexpensive explicit time stepping algorithms makes them particularly suited for wave propagation problems in which low dispersion errors are a necessary condition for accuracy. The main drawback of these algorithms however, lies in the small timesteps which are required to fulfill the explicit stability requirement. For polynomial approximations of, say, $p \leq 6$, one can use nodal basis functions with equally spaced nodes and in this case, the maximum timestep size allowed for stability scales like h/p, where h is the characteristic element size. For $p > 6$ on the other hand, alternative basis functions must be used in order to obtain a properly conditioned system and in such cases, the maximum timestep scales typically like h/p^2. This limitation is often too severe for practical applications. We consider below three situations in which explicit time stepping may not be practical for time-dependent calculations:

- for highly adapted meshes having a large disparity in element sizes. This may often be necessary due to geometrical constraints, but also because the solution may contain singularities that need to be appropriately resolved. In such cases, the explicit timestep is determined by the smallest elements. The situation is even worse when dealing with complex geometries. Here, one usually resorts to automatic grid generation techniques. Very often, the quality of the 3D tetrahedral meshes produced is not good due to the presence of small elements or 'slivers'.

- for the solution of viscous problems such as the Navier-Stokes equations using either Local Discontinuous Galerkin (LDG) methods[2] or any of the alternative approaches.[3] Here, the stable timestep size scales like h^2/p^2, if low degree polynomials and equally spaced nodes are used, otherwise a much more restrictive criterion, requiring timesteps proportional to h^2/p^4, must be employed.

*Instructor, Department of Mathematics, MIT, 77 Massachusetts Avenue 2-363A, Cambridge, MA 02139. E-mail: persson@mit.edu. AIAA Member.

†Professor, Department of Aeronautics and Astronautics, MIT, 77 Massachusetts Avenue 37-451, Cambridge, MA 02139. E-mail: peraire@mit.edu. AIAA Associate Fellow.

- for low Mach number problems that are essentially incompressible but solved using a compressible formulation. The DG method produces numerically stable discretizations, but solving them is hard because of the large ratio between the speed of sound and the fluid velocity. Even though the timescale for the physically interesting phenomena is based on the velocity, the timestep for an explicit method will be restricted by the sound waves.

Therefore, in order to develop robust and reliable applications for time dependent problems using DG methods, an implicit discretization appears to be a requirement. On the other hand, the problems of interest are usually very large and therefore, one must attempt to retain the low cost of explicit methods.

Here, we describe an implicit procedure, based on a backward difference approximation of the time derivative. Some implicit/explicit strategies[4,5] have been proposed in which, only the elements that place the more severe timestep restriction are advanced in an implicit manner. Even though our procedure can be combined with an explicit method in a straightforward manner, for simplicity we shall assume that it is applied in the whole domain.

An efficient storage format is important in order to obtain low memory usage and high computational performance. Because the matrices arising from the LDG method have an irregular sparsity pattern, we store them indirectly as a product of several matrices with structured patterns having a dense block format. The matrix-vector products can then be applied using high-performance dense matrix library routines.

For large problems, in particular in 3D, it is too expensive to use direct solution techniques and therefore, iterative methods must be employed. An essential ingredient for these iterative methods is the preconditioner. In this paper we compare the performance of different preconditioners: incomplete factorizations (ILU), coarse scale approximations, and a combination of ILU and a coarse scale method, which appears to be highly effective for our problems.

II. Problem Formulation

A. Equations and Discretization

We consider the time-dependent compressible Navier-Stokes equations,

$$\frac{\partial \boldsymbol{u}}{\partial t} + \nabla \cdot \boldsymbol{F}_i(\boldsymbol{u}) - \nabla \cdot \boldsymbol{F}_v(\boldsymbol{u}, \nabla \boldsymbol{u}) = 0 \ , \tag{1}$$

in a domain Ω, with conservative state variables $\boldsymbol{u}$ and inviscid and viscous flux functions $\boldsymbol{F}_i, \boldsymbol{F}_v$. We discretize $\boldsymbol{u}$ in space by introducing a triangulation of Ω and approximating $\boldsymbol{u}$ by multivariate polynomials of degree p within each element. These polynomials are represented by expansions in a nodal basis, and the coefficients for all the solution components and all the elements are collected into a global solution vector $\boldsymbol{U}$.

Discretizing Eq. (1) in space using the DG method leads to a system of ordinary differential equations (ODEs) of the form[6]

$$\boldsymbol{M}\dot{\boldsymbol{U}} = \boldsymbol{F}_v(\boldsymbol{U}) - \boldsymbol{F}_i(\boldsymbol{U}) \equiv \boldsymbol{R}(\boldsymbol{U}) \ . \tag{2}$$

Here, $\boldsymbol{M}$ is the block-diagonal mass-matrix (no dependencies between elements or solution components) and the residual vector $\boldsymbol{R}$ is a nonlinear function of $\boldsymbol{U}$. The viscous terms are handled using the LDG method.[2]

The system of ODEs (2) is integrated in time by choosing an initial vector $\boldsymbol{U}(0) = \boldsymbol{U}_0$ and a timestep Δt. Time stepping using an appropriate scheme produces approximations $\boldsymbol{U}(n\Delta t) \approx \boldsymbol{U}_n$. An explicit technique such as the popular fourth-order Runga-Kutta scheme (RK4) would have a timestep restriction determined by the eigenvalues of the matrix $\boldsymbol{M}^{-1} d\boldsymbol{R}/d\boldsymbol{U}$. For many applications this restriction is severe, and therefore implicit techniques are prefered.

The backward differentiation formula[7] of order k (BDF-k) approximates the time derivative at timestep n by a combination of the current solution and k previous solutions: $\dot{\boldsymbol{U}} \approx \frac{1}{\Delta t} \sum_{i=0}^{k} \alpha_i \boldsymbol{U}_{n-i}$, and it requires the residual $\boldsymbol{R}$ to be evaluated using the current solution $\boldsymbol{U}_n$. The system of equations (2) then becomes:

$$\boldsymbol{R}_{\mathrm{BDF}}(\boldsymbol{U}_n) \equiv \boldsymbol{M} \sum_{i=0}^{k} \alpha_i \boldsymbol{U}_{n-i} - \Delta t \boldsymbol{R}(\boldsymbol{U}_n) = 0 \ . \tag{3}$$

We use a damped Newton's method to solve these equations. An initial guess $\boldsymbol{U}_n^{(0)}$ is formed using the k previous solutions, and iterates $\boldsymbol{U}_n^{(j)}$ are evaluated by computing corrections according to the linearized

equation:

$$J(\boldsymbol{U}_n^{(j)})\Delta\boldsymbol{U}_n^{(j)} = \boldsymbol{R}_{\mathrm{BDF}}(\boldsymbol{U}_n^{(j)}) \ . \tag{4}$$

The new iterates are obtained as $\boldsymbol{U}_n^{(j+1)} = \boldsymbol{U}_n^{(j)} + \beta\Delta\boldsymbol{U}_n^{(j)}$, where β is a damping parameter determined by forcing the residual to decrease. The iterations are continued until the residual is sufficiently small. The Jacobian J is obtained by differentiation of Eq. (3) (dropping the iteration superscript):

$$J(\boldsymbol{U}_n) = \frac{d\boldsymbol{R}_{\mathrm{BDF}}}{d\boldsymbol{U}_n} = \alpha_0\boldsymbol{M} - \Delta t\frac{d\boldsymbol{R}}{d\boldsymbol{U}_n} \equiv \alpha_0\boldsymbol{M} - \Delta t\boldsymbol{K}. \tag{5}$$

In this paper, we will focus on the solution of the linear system of equations (4) using iterative methods and deliberately ignore other issues related to the convergence of the Newton method, such as the determination of the step sizes Δt, etc. We will consider three test problems with a wide range of timesteps Δt and different flow properties. The constant α_0 is assumed to be exactly one for simplicity (which is the case for $k = 1$ and a good approximation for higher k).

B. Jacobian Sparsity Pattern and Representation

Our system matrix $\boldsymbol{A} = \boldsymbol{M} - \Delta t\boldsymbol{K}$ is sparse with a non-trivial block structure. It can be represented in a general sparse matrix format, such as the compressed column format,[8] but then we would not take advantage of the large dense submatrices (for example to obtain high performance in the matrix-vector products).

To analyze the sparsity structure in D dimensions, assume there are n_s nodes in each element and n_{es} nodes on each face. For a simplex element, $n_s = \binom{p+D}{D}$ and $n_{es} = \binom{p+D-1}{D-1}$. Furthermore, assume there are n_c solution components ($n_c = D + 2$ for the compressible Navier-Stokes equations) and n_t elements. The total number of unknowns is then $n = n_s n_c n_t$. Clearly we can store the solution vector u in one contiguous piece of memory, and it is convenient to treat it as a three-dimensional array u_{i_s,i_c,i_t} with indices i_s, i_c, i_t for node, component, and element, respectively.

A block diagonal matrix can easily be stored in a block-wise dense format. The mass matrix $\boldsymbol{M}$ has $n_c n_t$ blocks of size n_s-by-n_s along the diagonal, and we represent it by an array M_{i_s,j_s,i_t} of dimension n_s-by-n_s-by-n_t, where $M_{:,:,i_t}$ is the elemental mass matrix for element i_t. We do not duplicate these blocks for all n_c solution components since they are identical. In this format it is trivial to apply $\boldsymbol{M}$ times a vector by high-performance dense library routines, and also to compute the inverse $\boldsymbol{M}^{-1}$ block-wise.

We split the stiffness matrix $\boldsymbol{K}$ into a viscous part $\boldsymbol{K}_v$ and an inviscid part $\boldsymbol{K}_i$:

$$\boldsymbol{K} = \boldsymbol{K}_v - \boldsymbol{K}_i = \frac{d\boldsymbol{F}_v}{d\boldsymbol{U}} - \frac{d\boldsymbol{F}_i}{d\boldsymbol{U}} \ . \tag{6}$$

The sparsity structure of the inviscid matrix $\boldsymbol{K}_i$ consists of n_t blocks of size $n_s n_c$-by-$n_s n_c$ along the diagonal (larger blocks than for $\boldsymbol{M}$ since the components are connected) plus connections between the nodes on neighboring element faces. In the DG formulation, the diagonal blocks correspond to the volume integrals plus the self-terms from the face integrals, and the off-diagonal entries correspond to the face integrals between elements.

We store the block diagonal part of $\boldsymbol{K}_i$ in a three-dimensional array $D_{i_s i_c,j_s j_c,i_t}$, which has a structure similar to that of the mass matrix. The off-diagonal parts are stored in a four-dimensional array $C_{i_{es} i_c,j_{es} j_c,i_e,i_t}$ of dimension $n_{es}n_c$-by- $n_{es}n_c$-by-$(D+1)$-by-n_t, where i_e is the face index in element i_t. Note that these dense blocks are smaller than the block diagonal ($n_{es}n_c$ compared to $n_s n_c$). Again we can see that applying $\boldsymbol{K}_i$ times a vector can be done efficiently using this format, although the implementation is slightly more complex since the element connectivities are required, and since the face nodes are generally not consecutive in memory.

The viscous matrix $\boldsymbol{K}_v$ is more complicated. The LDG method introduces a new set of unknowns $\boldsymbol{q} = \nabla\boldsymbol{u}$ and discretizes a system of first-order equations using standard DG methods. At the element faces, the numerical fluxes are chosen by upwinding in two opposite directions for the two sets of equations. This leads to the following form of the matrix:[2]

$$\boldsymbol{K}_v = \frac{d\boldsymbol{F}_v}{d\boldsymbol{U}} = \frac{\partial\boldsymbol{F}_v}{\partial\boldsymbol{U}} + \frac{\partial\boldsymbol{F}_v}{\partial\boldsymbol{Q}}\boldsymbol{M}^{-1}\boldsymbol{C}. \tag{7}$$

Here, the partial derivative matrices and $\boldsymbol{C}$ all have the same structure as $\boldsymbol{K}_i$, that is, block diagonal plus face contributions, and they can be stored as before. Note that the dimension of $\boldsymbol{Q}$ is D times larger than that of $\boldsymbol{U}$, and so are the matrices $\partial \boldsymbol{F}_v / \partial \boldsymbol{Q}$ and $\boldsymbol{C}$. The mass matrix $\boldsymbol{M}^{-1}$ is the same as before (but again D times as many blocks). Also, C is much sparser since it does not connect the different components (except for certain boundary conditions, which can be treated by small separate arrays).

A problem with the LDG discretization is that the stencil is wider than for the inviscid part. Although each of the individual matrices in Eq. (7) only connects to neighbors, the matrix product might connect neighbors' neighbors as well. This makes it harder to store $\boldsymbol{K}_v$ directly in a compact dense format. Instead, we store the matrix implicitly by the individual matrices in Eq. (7), and work with this formulation throughout the solution procedure. In our iterative solvers we must be able to apply the following operations with $\boldsymbol{K}_v$:

1. Apply the matrix-vector product $\boldsymbol{K}_v \boldsymbol{p}$
2. Form the block-diagonal of $\boldsymbol{K}_v$ for preconditioning
3. Form the incomplete LU factorization of $\boldsymbol{K}_v$ for preconditioning

We perform all of these operations using our split format. The matrix-vector product is easily applied by multiplication of each of the matrices in the product in Eq. (7) from right to left. The block-diagonal of $\boldsymbol{K}_v$ is computed directly by a blocked matrix multiplication. Finally, the incomplete factorization requires both diagonal and off-diagonal blocks, but since it is approximate anyway we choose to ignore the connections between neighbors' neighbors and store it in a compact block-format. This approximation might decrease the performance of the incomplete factorization, but as we show later it still performs well, in particular as a smoother for a low-degree preconditioner.

Using these techniques we avoid the wider stencil, which is one of the main disadvantages with the LDG method.

III. Implicit Solution Procedure

A. Iterative Solvers

We use iterative solvers for the linear system $\boldsymbol{A}\boldsymbol{u} = \boldsymbol{b}$. The matrix $\boldsymbol{A}$ is unsymmetric, so the popular Conjugate Gradient method can not be used (unless applied to the normal equations, which makes the system ill-conditioned). A simple method is Jacobi's method, or block Jacobi, which only requires computation of residuals and solution of block diagonal systems. We also consider unsymmetric Krylov subspace methods: the Quasi-Minimal Residual method (QMR), the Conjugate Gradient Squared method (CGS), the Generalized Minimal RESidual method (GMRES), and restarted GMRES with restart value m, GMRES(m). Among these, GMRES converges faster since it minimizes the true residual in each Krylov subspace, but its storage and computational cost increases with the number of iterations. GMRES with restarts is an attempt to resolve this, however it is well known that its convergence may occasionally stagnate. The two methods CGS and QMR are variants of the biorthogonalization method BiCG, and are also cheap to store and apply. See Barrett et al[8] for more details on these methods.

B. Preconditioning

In general the Krylov subspace methods must be preconditioned to perform well. This amounts to finding an approximate solver for $\boldsymbol{A}\boldsymbol{u} = \boldsymbol{b}$, which is relatively inexpensive to apply. For our block structured matrices, a natural choice is to set matrix entries outside the diagonal blocks to zero and invert the resulting matrix. This block-diagonal preconditioner does a decent job, but it turns out that we can do significantly better.

1. Incomplete Factorizations

A more ambitious approach is to factorize $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$ by Gaussian elimination. Obviously, if this is done without approximations, the problem is solved exactly and the iterative solver becomes obsolete. But this requires large amounts of additional storage for the fill-in (matrix entries that are zero in $\boldsymbol{A}$ but non-zero in $\boldsymbol{U}$ or $\boldsymbol{L}$), and also large computational costs.

A compromise is to compute an incomplete factorization, such as the ILU(0) factorization.[9] Here, no new matrix entries are allowed into the factorization (they are simply set to zero during the factorization), which makes the algorithm cheap to apply, and it requires about the same memory storage as A itself. The resulting factorization $\tilde{L}\tilde{U}$ might approximate A well, but its performance is hard to analyze. Alternatives which perform better keep some of the fill-in in the factorization, for example the neighbors' neighbors, or elements larger than a threshold value.

With our block-structured matrices it is natural to use a blocked incomplete factorization. The incomplete factorization provides a way around the wider stencil of the LDG method by simply ignoring the additional matrix entries.

2. Multi-Level Schemes and Low-Degree Corrections

Another technique for solving $\boldsymbol{A}\boldsymbol{u} = \boldsymbol{b}$ approximately is to use multi-level methods, such as the multigrid method.[10] Problems on a coarser scale are considered, either by using a coarser mesh (h-multigrid) or, for high-order methods, by reducing the polynomial degree p (p-multigrid[11,12]). An approximate error is computed on this coarse scale, which is applied as a correction to the fine scale solution. In the multigrid method a hierarchy of levels is used. On each level, a few iterations of a smoother (such as Jacobi's method) are applied to reduce the high-frequency errors.

A simpler and cheaper version of this is to use a two-grid scheme and compute a low-degree correction to the problem. In our DG setting, we project the residual $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{u}$ to $p = 1$ using an orthogonal Koornwinder expansion.[13] With a $p = 1$ discretization we solve for an approximate error, which is then prolongated to the original polynomial order and used as a correction to the solution. Before and after this procedure we perform a step of block Jacobi (damped with a factor 2/3 to make it a smoother).

3. The p1-ILU(0) Preconditioner

During our experiments with different preconditioners, we have observed that the block ILU(0) preconditioner and the low-degree preconditioner complement each other, and sometimes one is better than the other and vice-versa. It is well known that an ILU factorization can be used as a smoother for multigrid methods,[14,15] and it has been reported that it performs well for the Navier-Stokes equations, at least in the incompressible case using low-order discretizations.[16] Inspired by this, we use the block ILU(0) as a pre-smoother for a two-level scheme, which turns out to be many times better than pure ILU(0) or pure two-level scheme with a Jacobi smoother, without being significantly more expensive.

This preconditioner is essentially the p1-correction, but using ILU(0) instead of block Jacobi for the pre-smoothing. Below is a high-level description of the algorithm, for approximately solving $\boldsymbol{A}\boldsymbol{u} = \boldsymbol{b}$:

Solve $\tilde{L}\tilde{U}\boldsymbol{u}' = \boldsymbol{b}$ with block ILU(0) factorization $\boldsymbol{A} \approx \tilde{L}\tilde{U}$
Compute the residual $\boldsymbol{r}' = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{u}'$
Compute $p = 1$ projection $\boldsymbol{r}'_L$ of $\boldsymbol{r}'$ using the Koornwinder basis
Solve exactly (e.g. with direct solver) $\boldsymbol{A}_L\boldsymbol{e}'_L = \boldsymbol{r}'$, with $\boldsymbol{A}_L$ projected from $\boldsymbol{A}$
Compute prolongation $\boldsymbol{e}'$ from $\boldsymbol{e}'_L$
Add correction $\boldsymbol{u}'' = \boldsymbol{u}' + \boldsymbol{e}'$
Compute $\boldsymbol{u}$ by applying a Jacobi step to $\boldsymbol{u}''$

The only difference between this algorithm and the Jacobi smoothed p1-correction is the first step, which requires about the same computational cost as a Jacobi step, except for the calculation of $\tilde{L}\tilde{U}$ which is done only once per system.

IV. Results

A. Test Problems

We have studied three simplified test problems which we believe are representative for a large class of real-world problems. They are as follows:

1. Inviscid flow over duct. Structured, almost uniform mesh (figure 1). A total of 384 elements.

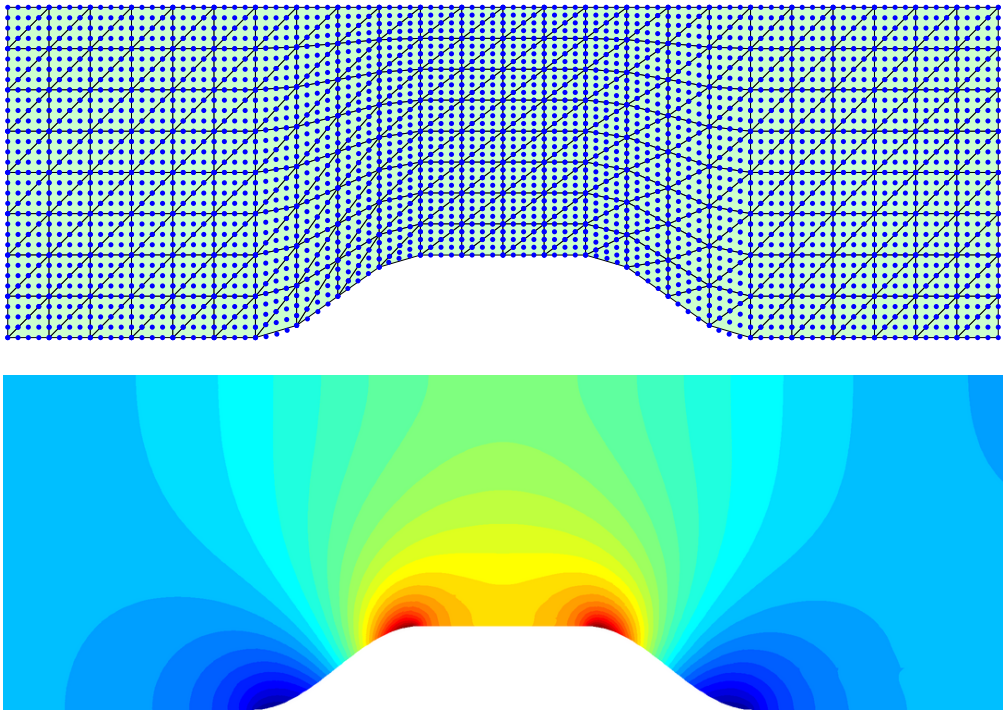

Figure 1. The inviscid flow over duct test problem at Mach 0.2. The entire mesh with DG nodes for $p = 4$ (top) and the Mach number of the solution (bottom).

2. Flat plate boundary layer, Reynolds number 50,000 based on the plate length. Graded and anisotropic mesh (Figure 2), with maximum stretching ratio of about 20. A total of 224 elements.

3. Flow around NACA0012 wing, angle of attach 5 degrees. Reynolds number 1000 based on the freestream condition and on the profile chord. Somewhat graded isotropic mesh (Figure 3). A total of 735 elements.

We solve for a steady-state solution to each problem for different Mach numbers, which we compute linearizations $\boldsymbol{K}$ around. We then consider the solution of $\boldsymbol{Au} = \boldsymbol{b}$ with $\boldsymbol{A} = \boldsymbol{M} - \Delta t \boldsymbol{K}$ and random right-hand side $\boldsymbol{b}$.

B. Iterative Solvers

First we solve the NACA test problem at Mach 0.2 using five different iterative solvers: Block-Jacobi, QMR, CGS, GMRES(20), and GMRES. The four Krylov solvers are preconditioned with the block diagonal to get a fair comparison, with similar computational cost for all methods. We count matrix-vector products instead of iterations, since the QMR and the CGS methods require two matrix-vector products per iteration. During the iterations we study the norm of the true error of the solution (not the residual), and we iterate until a relative error of 10^{-5} is obtained.

The results are shown in figure 4 for the timesteps $\Delta t = 10^{-3}$ and $\Delta t = 10^{-1}$ (the explicit timestep limit is about $\Delta t_0 = 10^{-5}$). For the small timestep (left plot) we note that all solvers perform well, with block-Jacobi and QMR requiring about twice as many matrix-vector products than the other three solvers. CGS is highly erratic, but usually performs about as good as GMRES. The restarted GMRES gives only a slightly slower convergence than the more expensive full GMRES.

The relative behaviour of the four Krylov solvers for the larger and the smaller timesteps is analogous. However, the block-Jacobi solver diverges after a few hundred iterations.

Based on these observations, and the fact that similar results are obtained for other problem types, we choose to only use the GMRES(20) solver in the remainder of the paper. The block-Jacobi solver is too

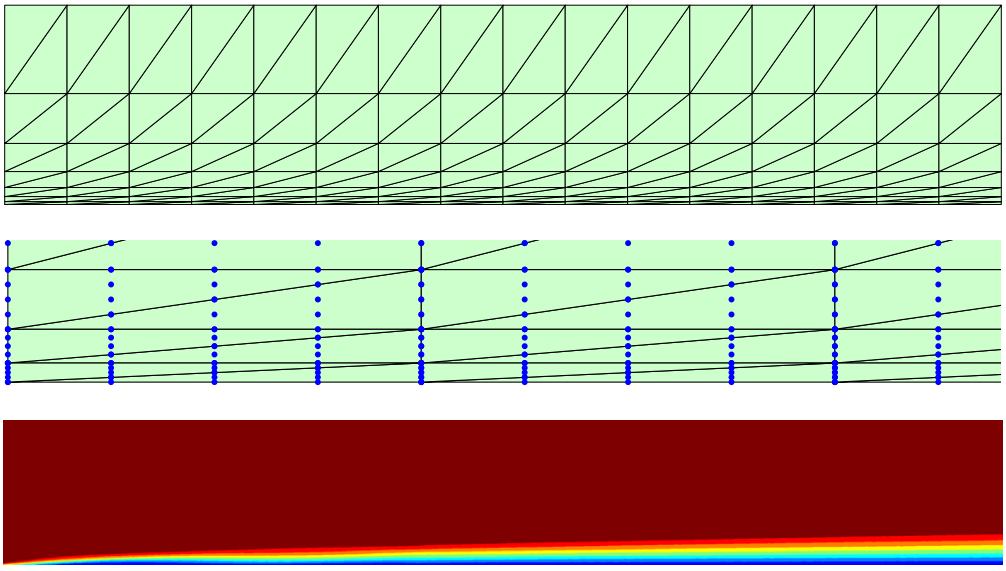

Figure 2. The boundary layer test problem at Mach 0.2 and Reynolds number 50,000. The entire mesh (top), a close-up of the lower-left corner with DG nodes for $p = 4$ (middle), and the Mach number of the solution in the close-up view (bottom).

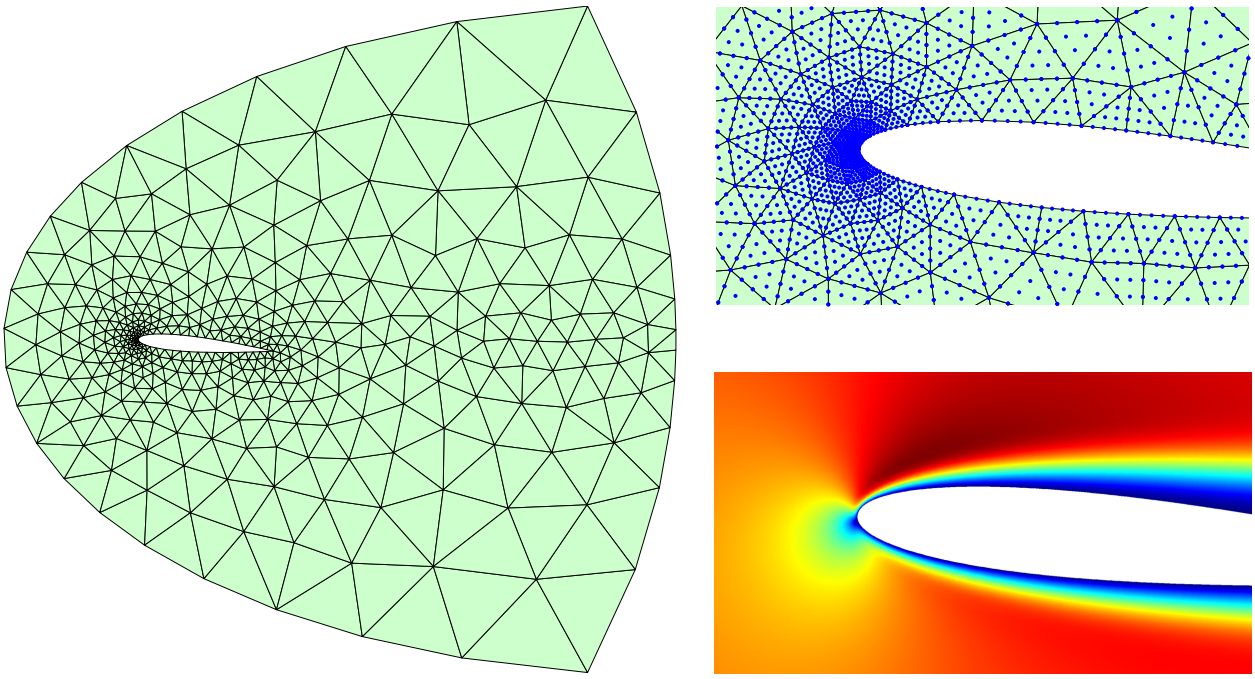

Figure 3. The NACA0012 test problem at Mach 0.2 and Reynolds number 1,000. The entire mesh (left), a close-up of the wing with DG nodes for $p = 4$ (top right), and the Mach number of the solution (bottom right).

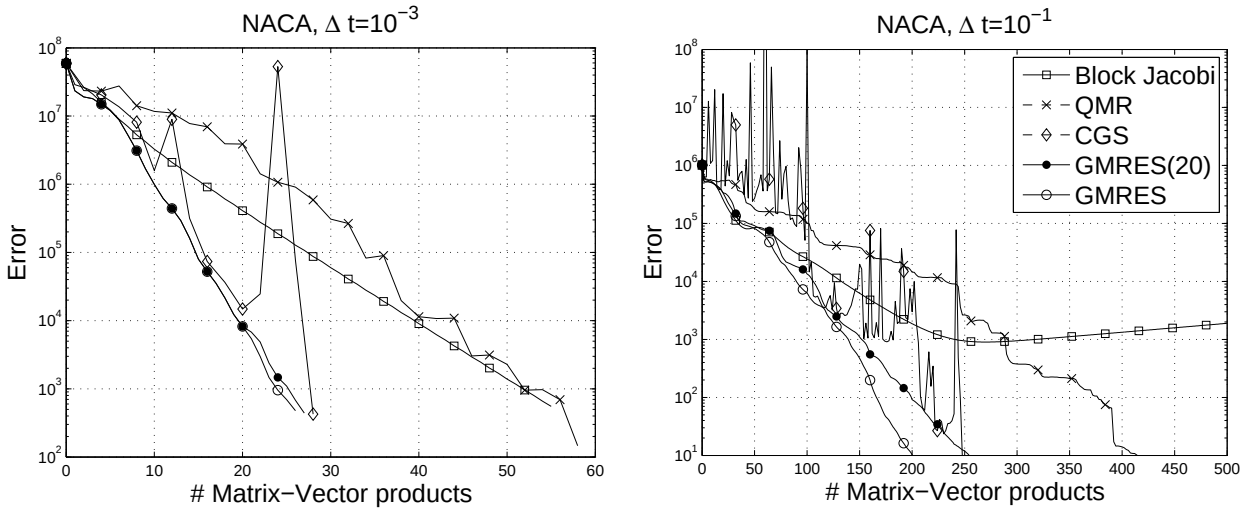

Figure 4. Convergence of various iterative solvers for the NACA problem with $\Delta t = 10^{-3}$ (left) and $\Delta t = 10^{-1}$ (right).

unreliable for large timesteps, and although it is simpler to implement it is not significantly faster than the other solvers if the cost is dominated by the matrix-vector multiplications. Also, it is not clear if the convergence of block-Jacobi can be improved by better preconditioning.

The full GMRES has the disadvantage that the cost increases with the number of iterations, both the storage and the computations. We have not observed any stagnation of the restarted GMRES(20) for our problems, and its slightly slower convergence is well compensated by the lower cost of the method. The solvers QMR and CGS are good alternatives since they are inexpensive and converge relatively fast.

C. Preconditioners for GMRES(20)

Next, we study the effect of different preconditioners on the convergence of the GMRES(20) method. In figure 5, we show the convergence of the p1-ILU(0) preconditioner, the pure ILU(0) and the Jacobi smoothed p1-correction preconditioners, as well as the diagonal block preconditioner used in the previous section. The problem is again the NACA model, but with timestep $\Delta t = 1.0$ in both plots, and Mach numbers 0.2 (left plot) and 0.01 (right plot).

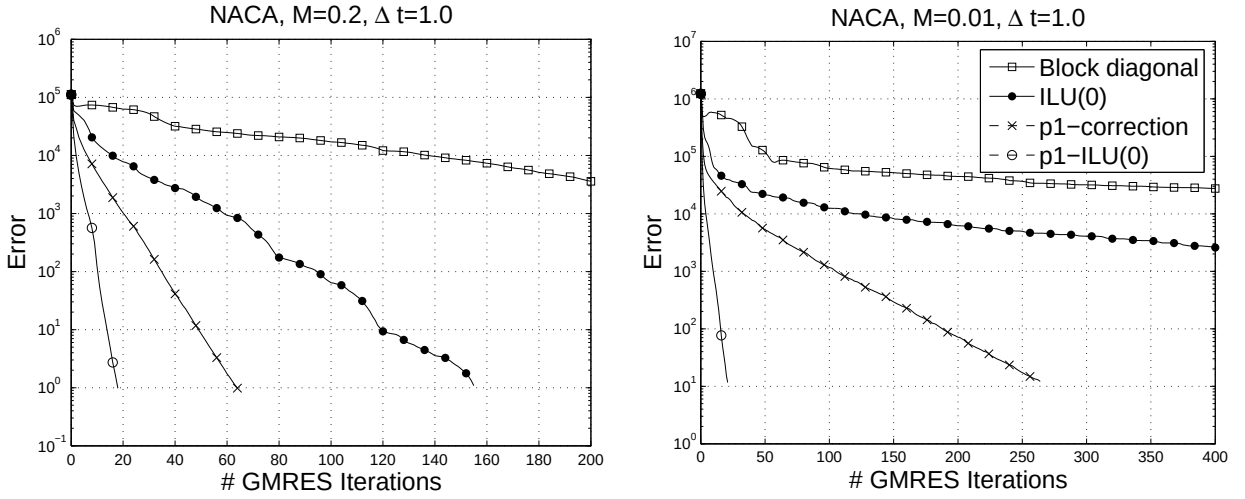

Figure 5. Convergence of GMRES(20) using various preconditioners for the NACA problem for Mach 0.2 (left) and Mach 0.01 (right). The timestep $\Delta t = 1.0$ in both models.

As before, we plot the error but now as a function of the number of GMRES iterations, since the performance of the preconditioners is implementation dependent and somewhat complex to analyze. A few points can be noted. The block diagonal preconditioner is cheaper than the other ones, possibly up to an order of magnitude. Applying ILU(0) is about the same cost as a matrix-vector product. The p1-correction is likely to be somewhat more expensive, depending on the cost of solving the reduced system. Finally, the p1-ILU(0) preconditioner is about the same cost as the Jacobi smoothed p1-correction, since one step of ILU(0) is about the same cost as a block-Jacobi step.

The plots show clearly that the p1-ILU(0) preconditioner is superior to the other methods. For Mach 0.2, the Jacobi smoothed p1-correction and the pure ILU(0) methods converge more than four times slower. The block-diagonal preconditioner is very inefficient for these large timesteps.

For the lower Mach number 0.01, all methods converge much slower (note the different scaling on the x-axis), with the exception of p1-ILU(0) which appears to converge fast independently of the Mach number. We will study this effect further in section E below.

D. Timestep Dependence

It is clear that the behavior of the iterative solvers on our system matrix $A = M - \Delta t K$ is highly dependent on the timestep Δt. For small values (about the same as the explicit timestep), $A \approx M$ and all of our preconditioners will solve the problem exactly. When we use an implicit solver we hope that we will be able to take much larger timesteps without a corresponding linear increase in cost. As $\Delta t \to \infty$, the system A is approximately a constant times $\boldsymbol{K}$, and we are essentially solving for a steady-state solution.

To study this timestep dependence, we solve all three model problems for Δt ranging from the explicit limit to high, steady-state like values. For each problem we solve for Mach numbers 0.2 and 0.01. We use GMRES(20) with the same four preconditioners as in the previous section.

The results can be seen in figure 6. For each problem and preconditioner, we plot the number of GMRES iterations required to reach a relative error of 10^{-5}. The dashed lines in the plots correspond to the explicit cost, that is, the number of timesteps an explicit solver would need to reach a given time. An implicit solver should be significantly below these curves in order to be efficient. Note that this comparison is not very precise, since the overall expense depends on the relative cost between residual evaluation and matrix-vector products/preconditioning.

All solvers converge in less than ten iterations for sufficiently small timesteps (about the explicit limit). As Δt increases, the block-diagonal preconditioner becomes less effective, and it fails to converge in less than 1000 iterations for timesteps larger than a few magnitudes times the explicit limit (except for the inviscid Mach 0.2 problem). The ILU(0) and the p1-correction preconditioners both perform well, and considering the lower cost of ILU(0) they are comparable. The p1-ILU(0) preconditioner is always more than a factor of two better than the other methods for large timesteps. This difference is again more pronounced for low Mach numbers, as shown in the three plots on the right.

E. Mach Number Dependence

To investigate how the convergence depends on the Mach number, we solve the NACA problem with $\Delta t = 1.0$ for a wide range of Mach numbers. The plot in figure 7 shows again that all methods performs worse as the Mach number decreases, except our p1-ILU(0) preconditioner. In fact, it appears to make the number of GMRES iterations almost independent of the Mach number, which is remarkable.

V. Conclusions

We have shown a way to timestep DG problems implicitly using efficient memory storage and fast iterative solvers. The discretized DG/LDG problems are stored block-wise, and in particular we circumvent the wider stencil of the LDG method by keeping the matrix factors that arise from the discretization. We studied several iterative solvers, and concluded that the restarted GMRES(20) works well in general. In particular, when preconditioned with our p1-ILU(0) preconditioner, it performs consistently well on all our test problems. It also made the convergence essentially independent of the Mach number.

The proposed method performs well for a number of problems with a large range of Reynolds and Mach numbers. We think that this approach is a good candidate for the solution of the more complex equations appearing in turbulence modeling. This will be the subject of future research.

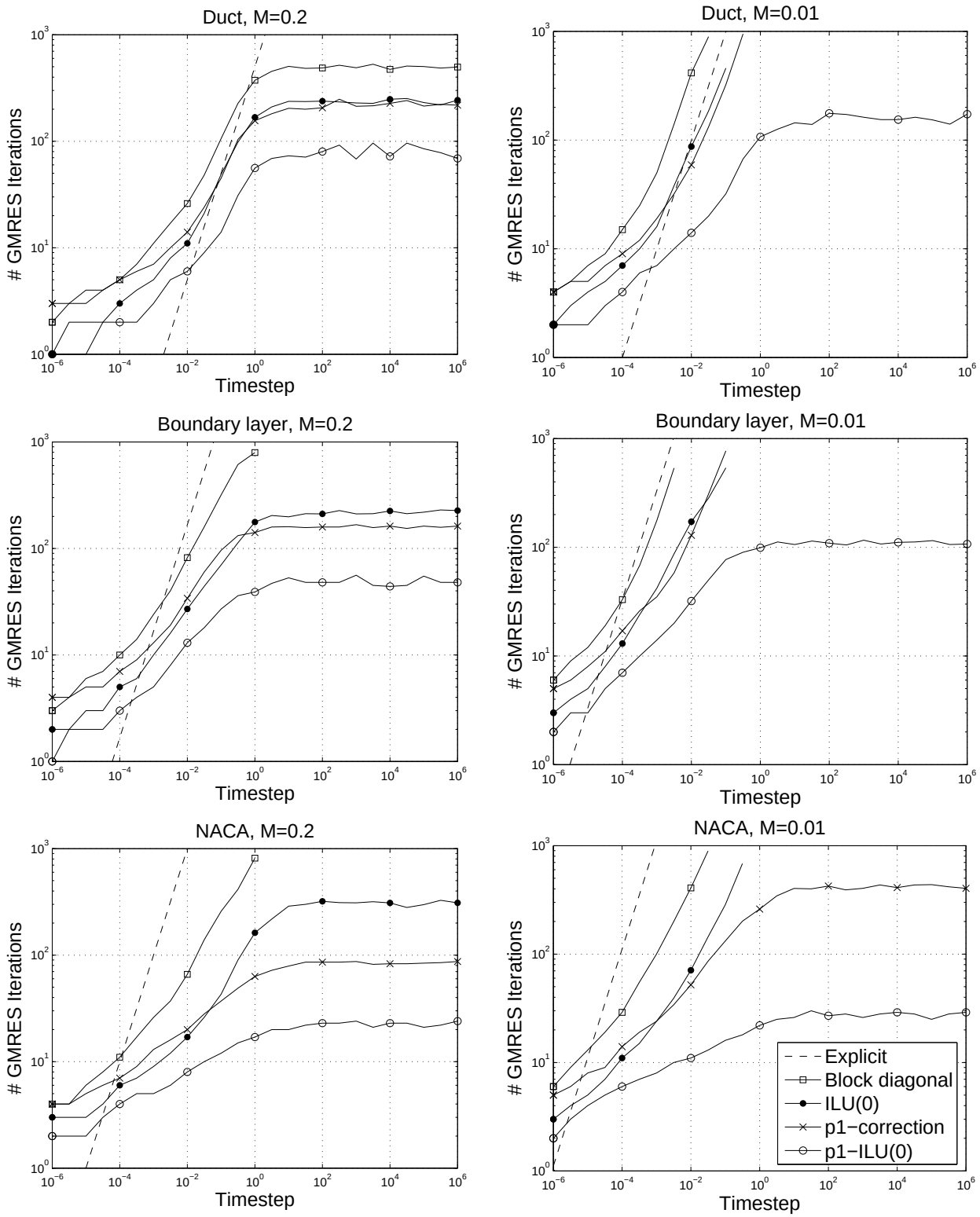

Figure 6. Convergence of GMRES(20) with various preconditioners, as a function of the timestep Δt. The problems are, from top to bottom, the duct problem, the boundary layer problem, and the NACA problem, with Mach numbers 0.2 (left plots) and 0.01 (right plots).

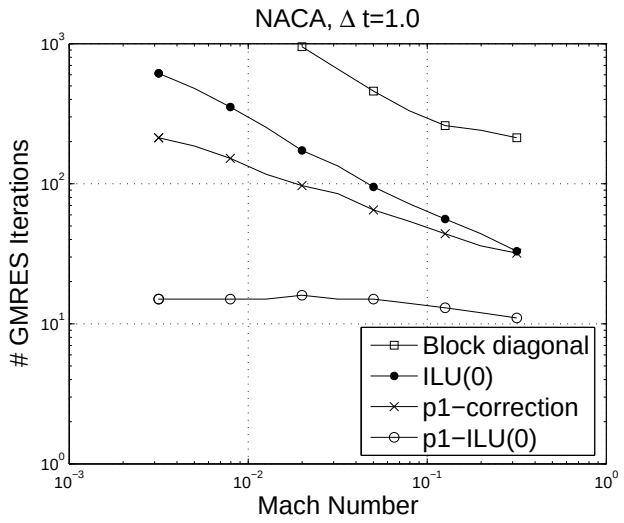

Figure 7. Convergence of GMRES(20) for the NACA problem, as a function of the Mach number. The p1-ILU(0) preconditioner makes the convergence almost independent of the Mach number, while the other solvers perform significantly worse for small values.

References

[1]Hesthaven, J. S. and Warburton, T., "Nodal high-order methods on unstructured grids. I. Time-domain solution of Maxwell's equations," *J. Comput. Phys.*, Vol. 181, No. 1, 2002, pp. 186–221.

[2]Cockburn, B. and Shu, C.-W., "The local discontinuous Galerkin method for time-dependent convection-diffusion systems," *SIAM J. Numer. Anal.*, Vol. 35, No. 6, 1998, pp. 2440–2463 (electronic).

[3]Arnold, D. N., Brezzi, F., Cockburn, B., and Marini, L. D., "Unified analysis of discontinuous Galerkin methods for elliptic problems," *SIAM J. Numer. Anal.*, Vol. 39, No. 5, 2001/02, pp. 1749–1779 (electronic).

[4]Ascher, U. M., Ruuth, S. J., and Spiteri, R. J., "Implicit-explicit Runge-Kutta methods for time-dependent partial differential equations," *Appl. Numer. Math.*, Vol. 25, No. 2-3, 1997, pp. 151–167, Special issue on time integration (Amsterdam, 1996).

[5]Kennedy, C. A. and Carpenter, M. H., "Additive Runge-Kutta schemes for convection-diffusion-reaction equations," *Appl. Numer. Math.*, Vol. 44, No. 1-2, 2003, pp. 139–181.

[6]Cockburn, B. and Shu, C.-W., "Runge-Kutta discontinuous Galerkin methods for convection-dominated problems," *J. Sci. Comput.*, Vol. 16, No. 3, 2001, pp. 173–261.

[7]Shampine, L. F. and Gear, C. W., "A user's view of solving stiff ordinary differential equations," *SIAM Rev.*, Vol. 21, No. 1, 1979, pp. 1–17.

[8]Barrett, R., Berry, M., Chan, T. F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., and der Vorst, H. V., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*, SIAM, Philadelphia, PA, 1994.

[9]Meijerink, J. A. and van der Vorst, H. A., "An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix," *Math. Comp.*, Vol. 31, No. 137, 1977, pp. 148–162.

[10]Hackbusch, W., *Multigrid methods and applications*, Vol. 4 of *Springer Series in Computational Mathematics*, Springer-Verlag, Berlin, 1985.

[11]Rønquist, E. M. and Patera, A. T., "Spectral element multigrid. I. Formulation and numerical results," *J. Sci. Comput.*, Vol. 2, No. 4, 1987, pp. 389–406.

[12]Fidkowski, K., Oliver, T., Lu, J., and Darmofal, D., "p-Multigrid solution of high-order discontinuous Galerkin discretizations of the compressible Navier-Stokes equations," *J. Comput. Phys.*, Vol. 207, No. 1, 2005, pp. 92–113.

[13]Koornwinder, T. H., "Askey-Wilson polynomials for root systems of type BC," *Hypergeometric functions on domains of positivity, Jack polynomials, and applications (Tampa, FL, 1991)*, Vol. 138 of *Contemp. Math.*, Amer. Math. Soc., Providence, RI, 1992, pp. 189–204.

[14]Wesseling, P., "A robust and efficient multigrid method," *Multigrid methods (Cologne, 1981)*, Vol. 960 of *Lecture Notes in Math.*, Springer, Berlin, 1982, pp. 614–630.

[15]Wittum, G., "On the robustness of ILU-smoothing," *Robust multi-grid methods (Kiel, 1988)*, Vol. 23 of *Notes Numer. Fluid Mech.*, Vieweg, Braunschweig, 1989, pp. 217–239.

[16]Elman, H. C., Howle, V. E., Shadid, J. N., and Tuminaro, R. S., "A parallel block multi-level preconditioner for the 3D incompressible Navier-Stokes equations," *J. Comput. Phys.*, Vol. 187, No. 2, 2003, pp. 504–523.