Efficient preconditioning of a high-order solver for multiple physics

Matteo Franciolini*, Anirban Garai,† and Scott Murman‡
NASA Ames Research Center, Moffett Field, 94035, United States

This work addresses preconditioning approaches for an implicit high-order solver framework applied to multiple physics. The solver is based on a space-time spectral element method and matrix-free Newton-Krylov solver developed at NASA over the recent years. Within this context, most preconditioning methods are impractical, as the computational time and memory requirements scale poorly with increasing polynomial orders. To improve computational efficiency, we first describe a novel entity-based Block Jacobi preconditioner for the continuous-Galerkin solution of the linear-elasticity and linear-shell equations. Second, we introduce a multigrid algorithm to further reduce time-to-solution on stiff cases arising from continuous- and discontinuous-Galerkin discretizations. Results obtained on relevant single-physics reference solutions, demonstrate the feasibility of the methods, paving the way for high-order solutions of fully coupled multi-physics problems.

I. Introduction

As there are many engineering problems involving the coupling of multiple physics, there is a growing interest in developing solution strategies that account for these effects, leveraging the variety of methods and tools available to accurately solve single physics discretizations. Common approaches involve the use of partitioned solution strategies, which re-use existing single physics solvers through data transfers. Such coupling schemes, which do not require modifications of the single physics solvers, have natural limitations, as they tend to be non-conservative, inaccurate, and numerically unstable [1]. As a consequence, development efforts to achieve robustness and accuracy of the associated single physics solvers are lost. On the other hand, monolithic approaches can be consistent, stable, and high-order accurate, at the expense of a higher software development effort. To this end, recent research at NASA has focused on the development of a high-order monolithic implicit multi-physics solver [2, 3]. The infrastructure supports the discontinuous-, continuous-, and C1-discontinuous-Galerkin numerical methods in a fully space-time manner, along with adjoint and tangent solver capability. One of the goals is to simulate the complex fluid structure interaction of flow past an entry capsule with a parachute attached. This requires simulating multiple physics like the compressible Navier-Stokes equations, the dynamics of a moving body using the linear elasticity equations and the structural response of the thin parachute using the linear shell equations. More recent applications are high-order mesh generation capabilities [4] and the solution of the Eikonal equation [5].

High-order discretizations have many theoretical advantages over conventional numerical methods when solving for multi-physics partial differential equations (PDEs). The objective of the current work is to introduce a preconditioning suite that allows for the efficient solution of linear systems arising from the space-time implicit discretization of multiple physics problems up to 16^{th}-order accuracy in both space and time. The key idea is to build preconditioners in a local-to-each physics fashion, which simplifies the implementation from a software design perspective, while maintaining the coupling between each physics when assembling the linearization of the residual. These individual single physics preconditioning methods can then be combined arbitrarily into a monolithic system by properly considering coupling terms. One of the main issues this work addresses is the prohibitive computational cost at high-order of standard matrix-based factorization algorithms, both from CPU time and memory requirements. To this end, the solver relies upon a matrix-free implementation of the linearization operator to avoid the computation and storage of the Jacobian matrix [6]. An efficient preconditioner still requires the use of parts of the iteration matrix and therefore we focus on devising operators and strategies that maintain the limited memory footprint of the solver.

Previous work has emphasized the development of solvers and preconditioners for the set of Navier–Stokes equations, which is the most costly physics module in the current suite. A novel tensor-product formulation of the preconditioner

*NPP/USRA Research Fellow, NASA Advanced Supercomputing Division. Email: matteo.franciolini@nasa.gov
†Science and Technology Corporation, NASA Advanced Supercomputing Division.
‡NASA Advanced Supercomputing Division.

operator [7] is efficient on "benchmark" cases, however, this alone is not effective enough for stiff linear systems associated with low mach numbers or general mesh elements. To this end, an extension of the method based on multigrid techniques, which relies on efficient coarse order smoothers based on the incomplete lower-upper decomposition (ILU) of the iteration matrix, was recently investigated [8]. This algorithm shows potential for high-order computational fluid dynamics (CFD) simulations providing improvements in computational efficiency in the diffusion dominated regions of the flow.

In this work, we first report recent developments towards an efficient preconditioning method for the elastic solvers of the multi-physics framework. The preconditioner is based on the factorization of the Jacobian matrix associated with degrees of freedom located at the lower-dimensional entities of the mesh, while employing a diagonal approximation of the volumetric terms of the matrix. This results in a computational cost that scales as that of a two-dimensional implicit solver and can be applied to space-time solutions up to 16^{th} order. Second, we further improve the computational efficiency by extending the multigrid preconditioning algorithm previously reported by the authors [8] to the elastic solvers, where the newly designed preconditioner is applied on the GMRES smoothers of the fine levels. Third, we extend the preconditioning method described in [8] for the solution of stiff adjoint problems.

In the following sections, the space-time discretization of the equation sets are briefly introduced, together with a detailed description of the preconditioning operators. Results involving the application of this preconditioner to mesh deformation/optimization cases, and the two-dimensional simulation of a deforming membrane will be presented afterwards. Finally, we will show the performance of the newly designed multigrid preconditioner for the solution of stiff linear systems arising from the elasticity solver, as well as the solution of a adjoint problems arising from the Navier–Stokes equations.

II. The numerical framework

A. Compressible Navier–Stokes equations

The spatial discretization for the Navier–Stokes equations is based upon the discontinuous-Galerkin method described by Diosady and Murman [6]. The method employs an entropy-stable formulation of the compressible Navier–Stokes equations. The discretization is based on a tessellation of the domain Ω into possibly curved non-overlapping elements, K, while the time is partitioned into intervals, here denoted as time-slabs, defined as $I^n = [t^n, t^{n+1}]$. The polynomial approximation space is defined as $\mathcal{V} = \{\boldsymbol{w}, \boldsymbol{w}|_{K\times I} \in [\mathcal{P}_\kappa(K \times I)]^m\}$, with κ equal to the order of polynomial approximation in the space-time domain consisting of piece-wise polynomial functions, where m is the number of equations. We seek a solution $\mathbf{v} \in \mathcal{V}_h$ such that the weak form

$$\begin{aligned} \boldsymbol{r}(\boldsymbol{w}, \mathbf{v}) \quad = \quad & \sum_\kappa \left\{ \int_I \int_\kappa - \left(\boldsymbol{w}_{,t} \cdot \boldsymbol{u} + \nabla \boldsymbol{w} \cdot (\boldsymbol{f}^I - \boldsymbol{f}^V) \right) \right. \\ & + \int_I \int_{\partial\kappa} \boldsymbol{w} \cdot (\widehat{\boldsymbol{f}^I \cdot \boldsymbol{n}} - \widehat{\boldsymbol{f}^V \cdot \boldsymbol{n}}) \\ & \left. + \int_\kappa \boldsymbol{w}(t_-^{n+1}) \cdot \boldsymbol{u}(t_-^{n+1}) - \boldsymbol{w}(t_+^n) \cdot \boldsymbol{u}(t_-^n) \right\} = 0 \end{aligned} \tag{1}$$

is satisfied for all $\boldsymbol{w} \in \mathcal{V}_h$, where $\boldsymbol{u} = \boldsymbol{u}(\mathbf{v})$ as given above. Here $\widehat{\boldsymbol{f}^I \cdot \boldsymbol{n}}$ and $\widehat{\boldsymbol{f}^V \cdot \boldsymbol{n}}$ denote numerical flux functions approximating the inviscid and viscous fluxes, respectively, while $\boldsymbol{n}$ is the outward pointing normal vector. In this work, the inviscid flux is discretized using the method of Ismail and Roe [9], while the viscous flux is discretized using an interior penalty method with stabilization parameters computed following the second form of the Bassi and Rebay scheme [10]. We remark that tensor-product basis functions [11] are employed to reduce the computational complexity of the residual evaluation through the use of the sum-factorization approach. The computational costs can be further reduced by the use of optimized numerical kernels to maximise the throughput of the application on state-of-the-art CPU architectures.

B. Linear elasticity equations

The code infrastructure is based on the work of Diosady and Murman [12]. We here recall the basic elements of the discretization strategy, while pointing to the previous work for further details. In order to obtain a volume mesh displacement, we solve the equations of linear elasticity given the prescribed displacement at the boundaries. The linear

elasticity equations are:

$$\sigma_{ij,j} = 0. \tag{2}$$

Here σ_{ij} is the Cauchy stress tensor given by:

$$\sigma_{ij} = 2\mu\epsilon_{ij} + \lambda\epsilon_{kk} \tag{3}$$

where $\mu = \frac{E}{2(1+\nu)}$ and $\lambda = \mu\frac{2\nu}{1-2\nu}$ are the Lamé constants given as a function of the Young's Modulus, E, and Poisson ratio, ν. The strain tensor, ϵ is given by:

$$\epsilon_{ij} = \tfrac{1}{2}\left(u_{i,j} + u_{j,i}\right) \tag{4}$$

where $\boldsymbol{u}$ is the displacement field. A compact representation of the stress tensor may be given by $\sigma_{ij} = \mathbf{C}^{ijkl}u_{i,j}$, where $\mathbf{C}^{ijkl}$ is the stiffness tensor.

To obtain a space-time discretization, we partition the spatial domain Ω into non-overlapping elements, κ, and the time domain is divided into intervals, here time-slabs, $I^n = [t^n, t^{n+1}]$, similarly to what is described for the Navier–Stokes equations. For each time slab, we fix the temporal coordinate and solve for the 3-spatial coordinates of the displacement of the space-time mesh from a reference initial spatial mesh. We apply a continuous-Galerkin discretization of (2) over the initial mesh. Let us define

$$\mathcal{V} = \left\{\boldsymbol{w} \in \mathcal{H}_1(\Omega)\times\mathcal{L}_2(I)), \boldsymbol{w}|_\kappa \in [\mathcal{P}(\kappa\times I)]^3\right\},$$

as the space-time finite-element solution space consisting of C^0 continuous piece-wise polynomial functions on each element. We seek solutions for the mesh displacements, $\boldsymbol{u} \in \mathcal{V}_E$ satisfying

$$\boldsymbol{r}(\boldsymbol{w}, \mathbf{v}) = \sum_\kappa \left\{ -\int_I\int_{\kappa^0} \boldsymbol{w}_{i,j}\mathbf{C}^{ijkl}\boldsymbol{u}_{k,l} + \int_I\int_{\partial\kappa^0\cap\partial\Omega} \boldsymbol{w}_i\widehat{\mathbf{C}^{ijkl}\boldsymbol{u}_{k,l}}\boldsymbol{n}_j \right\} = 0. \tag{5}$$

It is worth noting that we solve for the displacements for steady-state cases, while for unsteady cases we solve for the velocity field at each time slab, while the displacements are obtained as $\boldsymbol{u}(t) = \boldsymbol{u}(0) + \int_0^T \mathbf{v}dt$. The specified displacements are weakly enforced on the boundaries of the fluid mesh.

Using a particular choice of basis to discretize the space $\mathcal{V}$, equation (5) leads to a system of linear equations which must be solved for the mesh displacement. Since our goal is to use the mesh deformation strategy as part of an unsteady fluid-structure interaction simulation, efficient solution of the resulting system of equations is required. Here we rely on the tensor-product formulation of the basis functions like we do when solving for the compresisble Navier-Stokes equations. The computation of the residual semilinear form involves computing integrals over the space-time mesh elements. However, degrees of freedom associated with mesh entities are shared between two or more mesh elements. For this reason, the global vector of degrees of freedom is mapped into a local-to-each element arrangement before computing (5), while the opposite operation is performed on the residual vector. These gather/scatter operations from global to local memory layout allow to efficiently reuse the same optimized tensor-product computational kernels as for the discontinuous-Galerkin solver in order to further reduce the computational cost on state-of-the-art CPU architectures.

C. Linear-shell equations

The linear-shell equations model the parachute fabrics as a shell structures in a thin curved domain, and it computes the velocity ($\mathbf{x}_{,t}$) for the unsteady linear-shell simulations with respect to the undeformed mid-surface, represented by the ξ_1, ξ_2 curvilinear coordinates, of thickness h in the ξ_3 direction. The linear-shell equation reads as:

$$\begin{gathered} -A_3\left[\Sigma_{\alpha\beta}\right]_{,\xi_\alpha\xi_\beta} - \frac{1}{2}\left(A_\alpha\left[\Pi_{\alpha\beta}\right]_{,\xi_\beta} + A_\beta\left[\Pi_{\alpha\beta}\right]_{,\xi_\alpha}\right) \\ -\frac{A_{\alpha,\xi_\beta}\times A_2 + A_2\times A_3 A_{\alpha,\xi_\beta}\cdot A_3}{||A_1\times A_2||}\left[\Sigma_{\alpha\beta}\right]_{,\xi_1} - \frac{A_1\times A_{\alpha,\xi_\beta} + A_3\times A_1 A_{\alpha,\xi_\beta}\cdot A_3}{||A_1\times A_2||}\left[\Sigma_{\alpha\beta}\right]_{,\xi_2} \\ -q + \rho h x_{,tt} = 0, \end{gathered} \tag{6}$$

where $\Sigma_{\alpha\beta} = \left[\frac{Eh^3}{12(1-\nu^2)}C^{\alpha\beta\gamma\delta}B_{\gamma\delta}\right]$, $\Pi_{\alpha\beta} = \left[\frac{Eh}{1-\nu^2}C^{\alpha\beta\gamma\delta}H_{\gamma\delta}\right]$ and A_is are the undeformed mid-surface basis vectors. $H_{\alpha\beta}$, $B_{\alpha\beta}$, $C^{\alpha\beta\gamma\delta}$, q, E, ν and ρ are the membrane strain, bending strain, rank-4 symmetric tensor, load on the mid-surface,

Young's Modulus, Poisson ratio, and density respectively. Further details of the linear-shell equations can be found in Burgess et al. [13].

To discretize the second-order derivative in Eqn. 6, a C^1-discontinuous-Galerkin spectral-element methodology was developed. We partition the spatial undeformed mid-surface Ω_s into non-overlapping elements, κ_s, and the time domain is divided into intervals, here time-slabs, $I^n = [t^n, t^{n+1}]$ to obtain a space-time discretization. We define

$$\mathcal{V} = \left\{ \boldsymbol{w} \in \mathcal{H}_1(\Omega_s) \times \mathcal{L}_2(I)), \boldsymbol{w}|_{\kappa_s} \in [\mathcal{P}(\kappa_s \times I)]^3 \right\}$$

the space-time finite-element space consisting of C^0 continuous piece-wise polynomial functions on each element. We then seek solutions for the mesh velocity, $\mathbf{x}_{,t} \in \mathcal{V}_E$ satisfying

$$\begin{aligned} \boldsymbol{r}(\boldsymbol{w}, \dot{\mathbf{x}}) = \sum_{\kappa_s} \{ \int_{\kappa_s} \int_t \frac{1}{2} \left(A_\alpha \cdot \boldsymbol{w}_{,\xi_\beta} + A_\beta \cdot \boldsymbol{w}_{,\xi_\alpha} \right) \left[\Pi_{\alpha\beta} \right] \\ + \int_{\kappa_s} \int_t \left(-A_3 \cdot \boldsymbol{w}_{,\xi_\alpha \xi_\beta} + \boldsymbol{w}_{,\xi_1} \cdot \frac{A_{\alpha,\xi_\beta} \times A_2 + A_2 \times A_3 A_{\alpha,\xi_\beta} \cdot A_3}{||A_1 \times A_2||} + \boldsymbol{w}_{,\xi_2} \cdot \frac{A_1 \times A_{\alpha,\xi_\beta} + A_3 \times A_1 A_{\alpha,\xi_\beta} \cdot A_3}{||A_1 \times A_2||} \right) \left[\Sigma_{\alpha\beta} \right] \\ - \int_{\kappa_s} \int_t \boldsymbol{q} \cdot \boldsymbol{w} - \int_{d\Omega_s} \int_t \boldsymbol{N} \cdot \boldsymbol{w} + \int_{\kappa_s} \int_t \rho h \mathbf{x}_{,tt} \cdot \boldsymbol{w} + \int_{d\kappa_s} \int_t \mathcal{J}_{\Sigma_{\alpha\beta}} \} = 0, \end{aligned} \tag{7}$$

where $\boldsymbol{N}$ is the traction at the shell boundaries and $\mathcal{J}_{\Sigma_{\alpha\beta}}$ is the jump terms arising from the second-order derivative terms [13]. Similar to the linear elasticity, the displacement $\mathbf{x}$ are obtained as $\mathbf{x}(t) = \mathbf{x}(0) + \int_0^T \mathbf{x}_{,t} dt$.

D. Implicit time integration

The space-time fully coupled linear system is solved using a matrix-free Newton-Krylov algorithm. A single Newton update is obtained through the solution of

$$\left(\frac{\partial \boldsymbol{R}^*}{\partial \boldsymbol{U}} \right) \Delta \boldsymbol{U}^k \quad = \quad -\boldsymbol{R}^*(\boldsymbol{U}^k), \tag{8}$$

with $\boldsymbol{R}^*(\boldsymbol{U})$ is the unsteady residual and $\Delta \boldsymbol{U}^k$ is the Newton's update. We remark that the vectors $\boldsymbol{R}^*$ and $\boldsymbol{U}$ may span multiple physics and multiple discretization methods, which are fully coupled by appropriate boundary conditions as reported in [2]. Eq. (8) represents a linear system that we solve using a preconditioned restarted Flexible Generalized Minimal Residual (FGMRES) algorithm [14]. One advantage of such a method is that the residual Jacobian matrix $\partial \boldsymbol{R}^* / \partial \boldsymbol{U}$ is not required explicitly, only its product with the residual update. This task is accomplished within the following steps:

1. Evaluation of the state, gradient, linearized state and linearized gradient ($\boldsymbol{U}$, $\nabla \boldsymbol{U}$, $\boldsymbol{V}$ and $\nabla \boldsymbol{V}$) at the quadrature points.
2. Evaluation of the linearized fluxes ($\boldsymbol{F}_{\text{Lin}} = \frac{\partial \boldsymbol{F}}{\partial \boldsymbol{U}} \boldsymbol{V} + \frac{\partial \boldsymbol{F}}{\partial \nabla \boldsymbol{U}} \nabla \boldsymbol{V}$) at the quadrature points.
3. Multiplication of the linearized fluxes with the gradient of the basis functions.

The use of a matrix-free implementation decouples the use of the Jacobian to compute the Fréchet derivatives with respect to its use for preconditioning, and increases the flexibility of the solver.

III. Entity-based preconditioning for the continuous-Galerkin discretization

A. Method

For the structual solver, which is based on a continuous-Galerkin (CG) formulation, preconditioning is based on the factorization of the Jacobian blocks associated with the degrees of freedom residing on the lower-dimensional entities. Following standard CG practices, we divide the degrees of freedom by entity type: vertices (v), edges (e), faces (f) and

volumes (κ). This reflects on the Jacobian matrix, which can be divided in a 4x4 entity-block matrix:

$$\frac{\partial \mathbf{R}}{\partial \mathbf{U}} = \begin{bmatrix} \mathbf{A}_{v,v} & \mathbf{A}_{v,e} & \mathbf{A}_{v,f} & \mathbf{A}_{v,\kappa} \\ \mathbf{A}_{e,v} & \mathbf{A}_{e,e} & \mathbf{A}_{e,f} & \mathbf{A}_{e,\kappa} \\ \mathbf{A}_{f,v} & \mathbf{A}_{f,e} & \mathbf{A}_{f,f} & \mathbf{A}_{f,\kappa} \\ \mathbf{A}_{\kappa,v} & \mathbf{A}_{\kappa,e} & \mathbf{A}_{\kappa,f} & \mathbf{A}_{\kappa,\kappa} \end{bmatrix}. \tag{9}$$

Each block $\mathbf{A}_{i,j}$ contains the dependence of the residual vector associated with a given entity i with respect to a degree of freedom located on an entity j. In order to build a preconditioner, we first neglect the mixed entity-blocks, i.e. we only consider the block diagonal of (9). This not only reduces the storage, but considerably simplifies the computation and factorization of the preconditioner. This savings allows to factorize each entity sub-section of the matrix independently, increasing the computational efficiency of the method. Note that each one of the entity-wise Jacobian blocks represents a sparse matrix. $\mathbf{A}_{v,v}$, $\mathbf{A}_{e,e}$ and $\mathbf{A}_{f,f}$ have a sparsity pattern associated with the mesh element connectivity, and can be stored in the block-compressed sparse-row format (BSR). This increases the computational efficiency of the factorization and the triangular solve with respect to a standard compressed sparse-row format (CSR) by reducing indirect addressing and using optimized matrix-matrix products. On the other hand, the Jacobian block associated with the volumes $\mathbf{A}_{\kappa,\kappa}$ are dense and block-diagonal by construction with the use of our modal basis functions.

It is worth noting that it would still be impractical to use the full block diagonal matrix $\mathbf{A}_{\kappa,\kappa}$ at very high-order of accuracy, since the space and time complexity of factorization algorithms scale poorly with polynomial order. In order to circumvent this issue, we store its diagonal approximation, which reduces substantially the computational cost of the method. Once the reduced matrix has been assembled, we construct the preconditioner as in Equation (10) by computing the incomplete LU with zero filling (ILU0) factorization of each matrix independently, which can be performed at a very low computational cost. For the volume terms, the ILU reduces to the computation of the reciprocal element of the point diagonal. We refer to this approach as Entity Block-ILU0 (EILU0).

$$P^{-1} = \begin{bmatrix} \text{ILU0}(\mathbf{A}_{v,v}) & & & \\ & \text{ILU0}(\mathbf{A}_{e,e}) & & \\ & & \text{ILU0}(\mathbf{A}_{f,f}) & \\ & & & \text{diag}(\mathbf{A}_{\kappa,\kappa})^{-1} \end{bmatrix}. \tag{10}$$

A further reduced version of the preconditioner can be obtained by neglecting the off-diagonal blocks of the Jacobian matrices $\mathbf{A}_{v,v}$, $\mathbf{A}_{e,e}$ and $\mathbf{A}_{f,f}$. In this case, the Block-ILU0 reduce to the LU factorization of each diagonal block of the matrix, which significantly reduces the CPU time and memory requirements. In the remainder of the paper, we will refer to this approach as Entity Block-Jacobi (EBJ). Although an eigenvalue analysis showed that the spectral properties of this block-diagonal matrix do not allow to construct a simple Jacobi-iterative solver, its use within a GMRES linear solver proved to be very effective to reduce the computational costs.

B. Performance

The first test case here considered is the deformation of a box-in-box mesh [12], which involves embedding a box of size (0.2 × 0.2) within a unit box (1 × 1) and rotating the interior box by a prescribed motion. We consider here mesh motions on a 2^{nd}-, 4^{th}- and 8^{th}-order meshes with a fixed number of mesh elements (24), as depicted in Figure 1. To focus on the performance of the solver rather than the efficacy of the mesh deformation algorithm, the solver employs a linear elastic model with constant Young modulus E and Poisson ratio ν throughout the domain. Figure 2 shows the convergence plot of the test case for $N = 2, 4, 8$ order solutions. The Entity Block Jacobi preconditioner provides considerable improvements with respect to the unpreconditioned case, and, as expected, the advantages increase with the increased stiffness at higher solution order, with the reduction of total CPU time greater than two orders of magnitude.

Next, we examine the performance of the preconditioner operator in a test case involving mesh deformation of a low-pressure turbine airfoil presented in [15]. Starting with an initial mesh, we apply perturbations along the surface normal to represent surface roughness effects, and use the elasticity approach to propagate the surface deformations into the domain. Figure 3 shows the deformation applied at the surface of the turbine blade, as well as the deformed mesh near the leading edge of the blade. This approach is associated with a fixed point (steady-state) mesh displacement. Figure 4 compares the method with respect to the unpreconditioned case. It can be observed that the entity-based preconditioner again reduces the CPU time by roughly two orders of magnitude.

Fig. 1 Box-in-box test case. Initial mesh (left) and $\theta = 30°$deformed mesh (right) coloured by the determinant of the scaled Jacobian for the 8th order case.

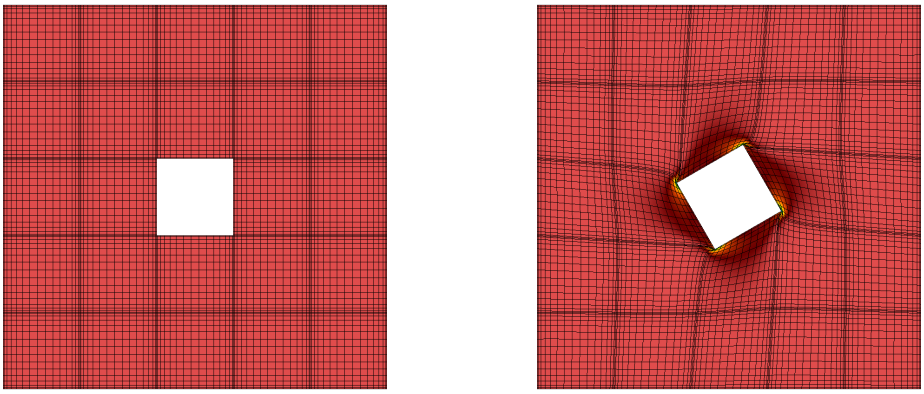


Fig. 2 Preconditioner performance on the mesh deformation algorithm applied to the box-in-box case. Comparison between the Entity Block Jacobi (EBJ) preconditioner and unpreconditioned solutions.

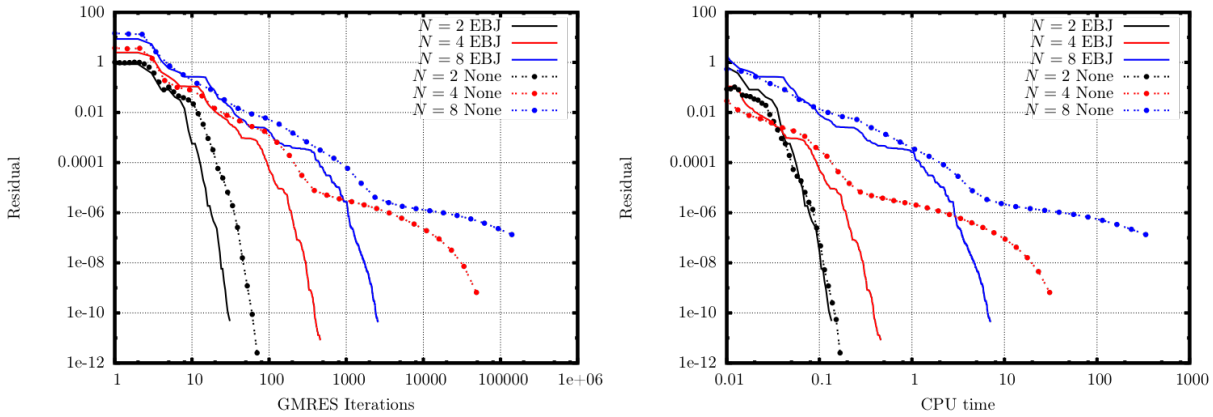


Fig. 3 Geometric mesh deformation of the LPT airfoil. Surface normal displacement (left) and its volumetric propagation within the domain (right).

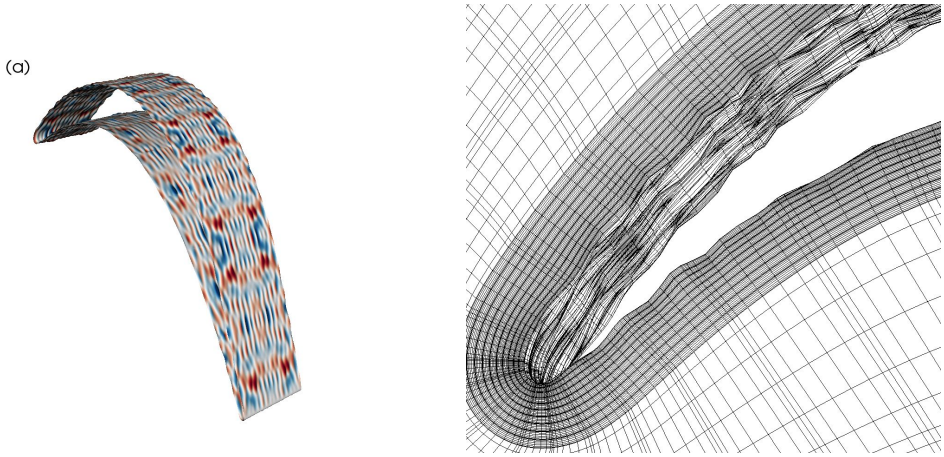


Fig. 4 Performance of the mesh deformation algorithm applied to the LPT airfoil up to $N = 16$ order. Comparison between the Entity Block Jacobi (EBJ) preconditioner and unpreconditioned solutions.

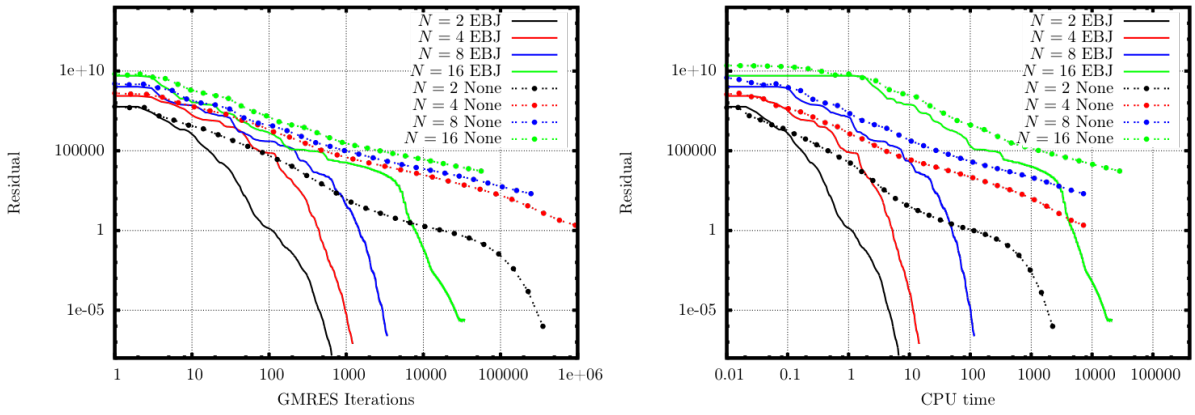
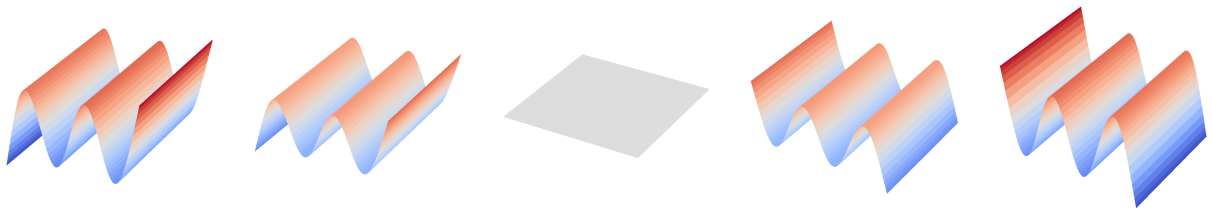
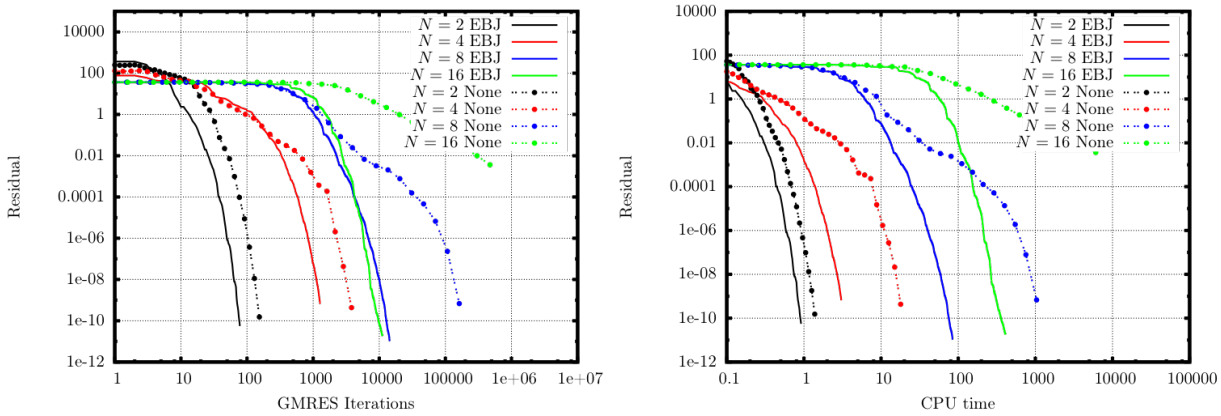


Fig. 5 Unconstrained plate subject to external periodic forcing in phase with the 4^{th}-vibration mode, coloured by the vertical displacement. The mesh is made by 8x8 elements, while $\tilde{t}$ indicates the non-dimensional time.



(a) $\tilde{t} = 0.0$ (b) $\tilde{t} = 0.125$ (c) $\tilde{t} = 0.25$ (d) $\tilde{t} = 0.375$ (e) $\tilde{t} = 0.5$

Fig. 6 Preconditioner performance on the linear-shell simulation of an unsteady free-free plate for different polynomial orders $N = 2, 4, 8, 16$. Comparison between the Entity Block Jacobi (EBJ) preconditioner and unpreconditioned solutions.



Finally, we examine the performance of the preconditioner on the linear-shell simulation of an unconstrained plate using 4^{th}-order discretization in time, see Figure 5. The preconditioner strategy for the linear-shell follows the methodology employed for the linear-elasticity, with the addition that the Jacobian matrix is modified to take into account the high-order derivative contribution. Note that in lower-dimensional problems, such as the linear-shell solver

on a surface instead of a volume, the "volumes" are the faces of the mesh, so the discussion of Eqn. (10) is consistently reduced by the topological dimension. Similarly to the previous test cases, we report results for different polynomial orders in space on the same number of mesh elements. Figure 6 shows significant improvements in the computational efficiency, where the highest order case ($N = 16$) converges several orders of magnitude faster than the case without a preconditioner.

IV. Multigrid preconditioning

A. Method

A multigrid preconditioner has been introduced by the authors in [8]. A stack of nested discretization spaces, $\mathcal{V}_0 \subset \ldots \mathcal{V}_\ell \subset \ldots \mathcal{V}_c$, with $\mathcal{V}_0 = \mathcal{V}$, is constructed by lowering the order of the element-wise polynomial approximation of the solution and applying an iterative solver in each one of them. On coarse spaces, low-frequency modes of the error shift towards the upper side of the spectrum, and hence are damped more effectively by the iterative solver. The approach to obtain coarse spaces follows a matrix-free framework, where the L_2-projection of the state is used to apply the linearization of the residual to the vector of unknowns of the linear system on the coarse level. The L_2-projection operator for our discontinuous-Galerkin spectral-element solver is local to each element,

$$\int_I \int_\kappa \boldsymbol{w}_{\ell+1} \boldsymbol{w}_{\ell+1} \mathbf{U}_{\ell+1} = \int_I \int_\kappa \boldsymbol{w}_{\ell+1} \boldsymbol{w}_\ell \mathbf{U}_\ell \tag{11}$$

with $\boldsymbol{w}_\ell \in \mathcal{V}_\ell$, $\boldsymbol{w}_{\ell+1} \in \mathcal{V}_{\ell+1}$ and $\mathcal{V}_\ell \subset \mathcal{V}_{\ell+1}$. Hence, the coarse space state can be obtained as a sequence of matrix-vector products

$$\boldsymbol{U}_{\ell+1} = \mathbf{M}_{\ell+1}^{-1} \Phi_{\ell+1}^T \mathbf{W}_\ell \Phi_\ell \boldsymbol{U}_\ell = \mathcal{I}_\ell^{\ell+1} \boldsymbol{U}_\ell \tag{12}$$

where Φ is the interpolation matrix, $\mathbf{W}$ is the diagonal matrix of the Gauss-quadrature weights, and $\mathbf{M}$ is the mass matrix. The subscript ℓ, $\ell + 1$ represents the multigrid level. Similarly, the prolongation operator, defined as the natural injection of the coarse DoFs into the fine space solution, is obtained as an L_2 projection between the coarse and the fine space:

$$\boldsymbol{U}_\ell = \mathbf{M}_\ell^{-1} \Phi_\ell^T \mathbf{W}_\ell \Phi_{\ell+1} \boldsymbol{U}_{\ell+1} = \mathcal{I}_{\ell+1}^\ell \boldsymbol{U}_{\ell+1} \tag{13}$$

Finally, the restriction operator, to be applied to residual vectors, can be obtained as the transpose of the prolongation. That is, considering the symmetry of the mass matrix,

$$\boldsymbol{R}_{\ell+1} = \Phi_{\ell+1}^T \mathbf{W}_\ell \Phi_\ell \mathbf{M}_\ell^{-1} \boldsymbol{R}_\ell = (\mathcal{I}_{\ell+1}^\ell)^T \boldsymbol{R}_\ell \tag{14}$$

Note that Φ and $\mathbf{M}$ are applied using the same tensor-product sum-factorization approach used for the residual evaluation.

The projection, restriction and prolongation operators are defined on elements, however, when we apply the multigrid preconditioner in a continuous-Galerkin solver, the global degrees of freedom are ordered according to the entities. Hence, before and after applying the element-wise operators, we gather/scatter the DoFs from the global ordering to the elemental ordering and vice-versa. During the first gather operation, shared degrees of freedom are effectively duplicated in the elemental representation of the problem. Hence, no continuity requirement is enforced during the application of the restriction, prolongation, and projection operators. The high-order interior modes of the C0-continuous piecewise polynomial functions are orthogonal with each other, but non-orthogonal to the modes of topologically lower entities, which results in different values of the shared entities in different elements. To ensure continuity, shared-entity values are averaged during the scatter operation. Note that the average operation is only required in the restriction and in the L_2-projection operators, as the prolongation simply pads the higher order modes with zeros .

Here we provide an overview of the sequence of operations involved in p-multigrid iterations. The recursive p-multigrid V-cycle for the problem of Eq. (8) on level ℓ is reported in Algorithm 1. To obtain an application of the p-multigrid preconditioner the multilevel iteration is invoked on the fine problem. One MG_V iteration requires two applications of the smoother on the finest level (one pre- and one post-smoothing) and one application of the coarse level smoother independently from the number of levels.

The choice of the smoother plays an important aspect for the computational efficiency of the method. Traditionally, stable operators with optimal smoothing properties have been used to achieve high multigrid efficiency. Notable examples are the Jacobi iterative method or a multi-stage Runge-Kutta. However, these methods may suffer from stability issues, which are typically prevented by the use of under-relaxation factors. On the other hand, more complex smoothing

algorithms, such as few iterations of a GMRES solver, are stable by construction and can work with arbitrarily complex problems, and for this reason have been used in this work. The GMRES smoothers are able to maintain the matrix-free nature of the solver, but still require the use of a preconditioner to efficiently smooth high error frequencies in each multigrid level.

Algorithm 1 $\overline{\boldsymbol{U}}_\ell = \mathrm{MG}_V(l, \widetilde{\boldsymbol{R}}^*_\ell, \boldsymbol{U}_\ell)$

if $(\ell = L)$ **then**
  $\overline{\boldsymbol{U}}_\ell = \mathrm{solve}(\mathrm{lin}_\ell, \widetilde{\boldsymbol{R}}^*_\ell, 0)$
end if
if $(\ell < L)$ **then**
  Pre-smoothing:
  $\overline{\overline{\boldsymbol{U}}}_\ell = \mathrm{smooth}(\mathrm{lin}_\ell, \widetilde{\boldsymbol{R}}^*_\ell, \boldsymbol{U}_\ell)$
  Coarse grid correction:
  $\boldsymbol{D}_\ell = \widetilde{\boldsymbol{R}}^*_\ell - \mathrm{lin}_\ell(\overline{\overline{\boldsymbol{U}}}_\ell)$
  $\boldsymbol{D}_{\ell+1} = (\mathcal{I}^\ell_{\ell+1})^T \boldsymbol{D}_\ell$
  $\boldsymbol{E}_{\ell+1} = \mathrm{MG}_V(\ell+1, \boldsymbol{D}_{\ell+1}, 0)$
  $\widehat{\boldsymbol{U}}_\ell = \overline{\overline{\boldsymbol{U}}}_\ell + \mathcal{I}^\ell_{\ell+1} \boldsymbol{E}_{\ell+1}$
  Post-smoothing:
  $\overline{\boldsymbol{U}}_\ell = \mathrm{smooth}(\mathrm{lin}_\ell, \widetilde{\boldsymbol{R}}^*_\ell, \widehat{\boldsymbol{U}}_\ell)$
end if
return $\overline{\boldsymbol{U}}_\ell$

In order to maintain efficiency, a strategy is adopted for multigrid preconditioning that utilizes an efficient "weaker" form of the preconditioned GMRES smoother on the finest levels, and a "stronger" preconditioner for the coarser levels [8, 16]. Thus, the difficulties with memory availability and computational cost associated with polynomial scaling at high order are avoided, while leveraging the advantages offered by improved Jacobian approximations at lower orders. To employ this strategy for Navier–Stokes, smoothers are preconditioned using a tensor-product alternating directions implicit (ADI) operator [7] on the fine space ($N > 2$), and an incomplete LU factorization with zero-filling (Block-ILU0) on the coarsest space $\mathcal{V}_c$. Similarly, for the elasticity, the Entity Block-Jacobi preconditioner (EBJ) is used in all but the coarsest space, where the Entity Block-ILU0 is employed. Note that for a continuous-Galerkin coarse space the only non-zeros degrees of freedom are located on the mesh vertices, and therefore the Entity Block-ILU0 corresponds to the Block-ILU0 of the full Jacobian matrix.

B. Performance

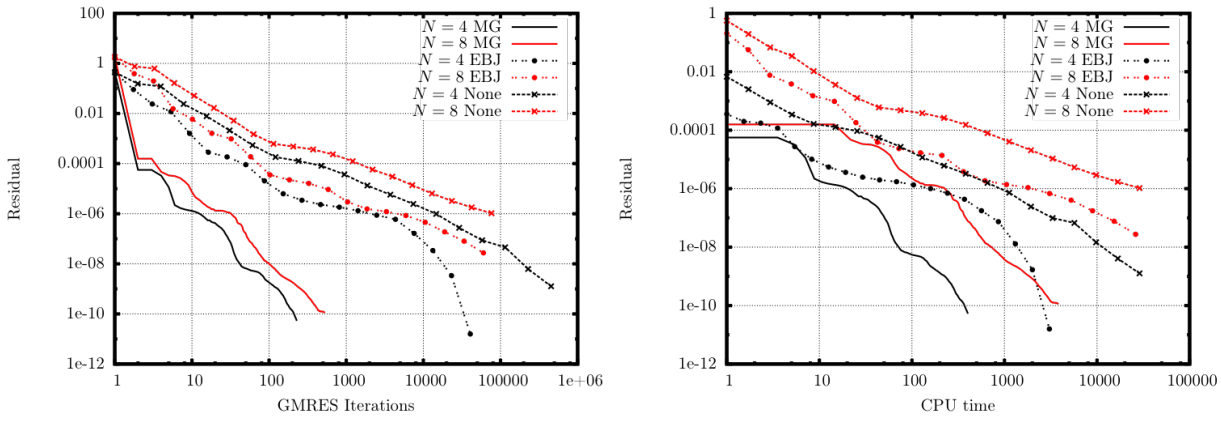
We first assess the efficiency of this solution strategy on stiff problems arising from the continuous-Galerkin solver for unsteady mesh deformation. The test case is taken from the high-order workshop series [17], where a NACA 0012 airfoil is given a time-dependent vertical displacement

$$y(t) = \alpha^2 t^2 \left(\frac{\beta - \alpha t}{\gamma} \right) \tag{15}$$

with $\alpha = 0.2$, $\beta = 3.0$, and $\gamma = 4.0$. We test the algorithm for two spatial polynomial orders, $N = 4, 8$, and we keep the polynomial order in time constant ($N_t = 4$). The multigrid smoothers are instances of preconditioned matrix-free GMRES solvers coupled with the Entity Block-Jacobi preconditioner in all levels but the coarsest, where the Entity ILU0-GMRES algorithm is used. The performance is tightly correlated to the number of smoothing iterations and to the choice of the smoothers. The focus here is to show the the representative improvement from the use of the multigrid algorithm rather than devising an optimized set of smoothing parameters that would be problem dependent.

Figure 7 shows the performance in terms of number of GMRES iterations and CPU time. For both the polynomial orders N a significant saving in terms of GMRES iterations are obtained when switching from single-grid to multigrid, with significant savings in terms of computational time. The $N = 4$ case was roughly 8 times faster, while in the $N = 8$ case we observed savings greater than an order of magnitude.

Fig. 7 Performance of the multigrid preconditioning strategy compared to the single-grid EBJ preconditioner for the NACA mesh deformation test case at $N = 4; 8$.



Second, we assess the performance of the multigrid algorithm for the solution of an adjoint problem arising from the discontinuous-Galerkin solution of the Navier–Stokes equations. The adjoint solution consists of solving one single linear system, which allows to isolate the effect of the preconditioner on performance from that of the nonlinear solver. The test case presented here involves the flow around a NACA 4412 airfoil and computes the adjoint solution with respect to the drag force on the surface of the airfoil. The flow conditions are $M = 0.1$, $\alpha = 12°$, with the computed Reynolds numbers of $Re = 5000$ and 50k, i.e. a lower and higher Reynolds number case to assess the relative importance of diffusion. The numerical tests are obtained on a three-dimensional discretization made by roughly 368k, 2.1M DoFs per equation at $Re = 5000$, 50k, respectively. Contours of velocity magnitude in the center of the planar section are presented in Fig.8 at the relevant Reynolds numbers.

The performance of the multigrid algorithm with respect to the single-grid, tensor-product ADI preconditioner are reported in Figures 9 and 10. For $N = 4$, we use one coarse level $N = 2$, with ADI on the fine and Block-ILU0 on the coarse. For $N = 8$, we use two coarse levels ($N = 4$, $N = 2$), with ADI everywhere but on the coarsest level, where we use Block-ILU0. The results obtained for the $Re = 5000$ case show speed-up values in CPU time around 3, which decreases to roughly 2 for the case at the higher Reynolds number.

Fig. 8 Contours of the velocity magnitude of the flow around a NACA 4412 airfoil at $Re = 5000; 50k$, $M = 0.1$, $\alpha = 12°$.

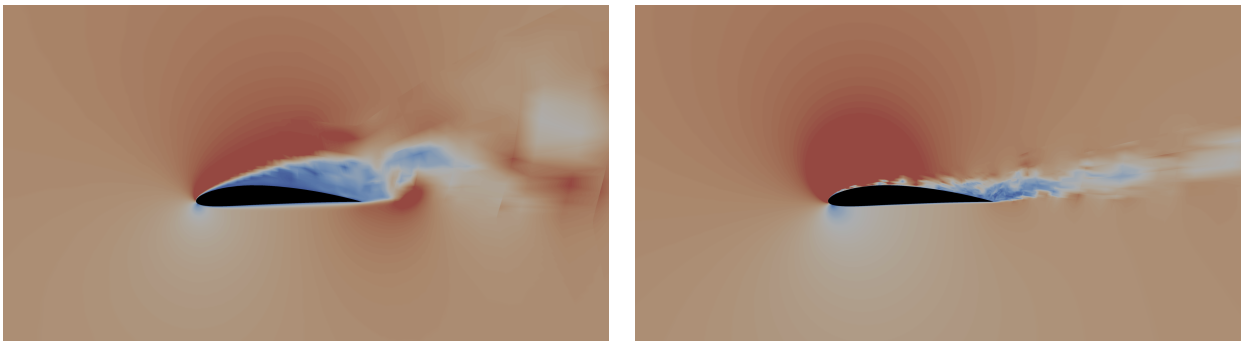


Fig. 9 Performance of the multigrid preconditioning strategy compared to the single-grid ADI preconditioner for the NACA 4412 test case at $Re = 5000$ for $N = 4; 8$.

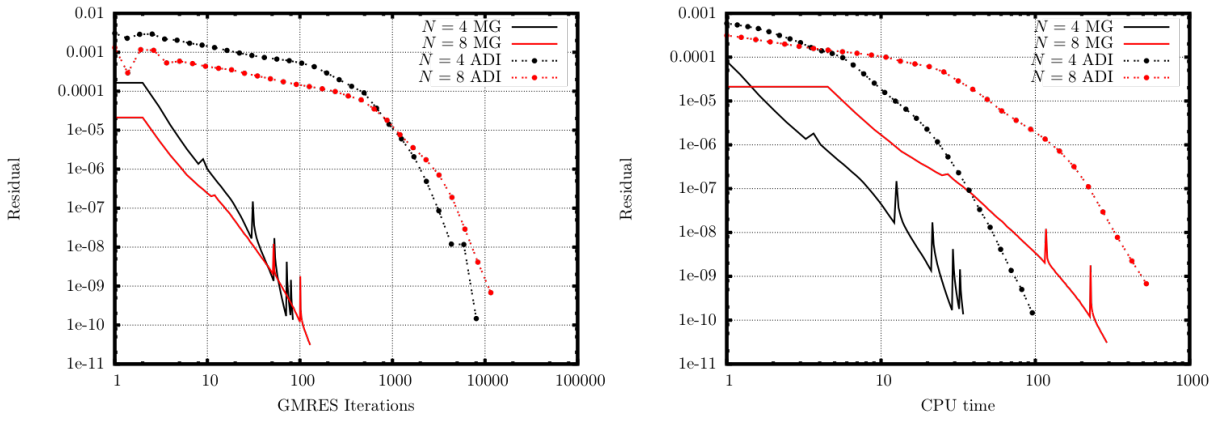
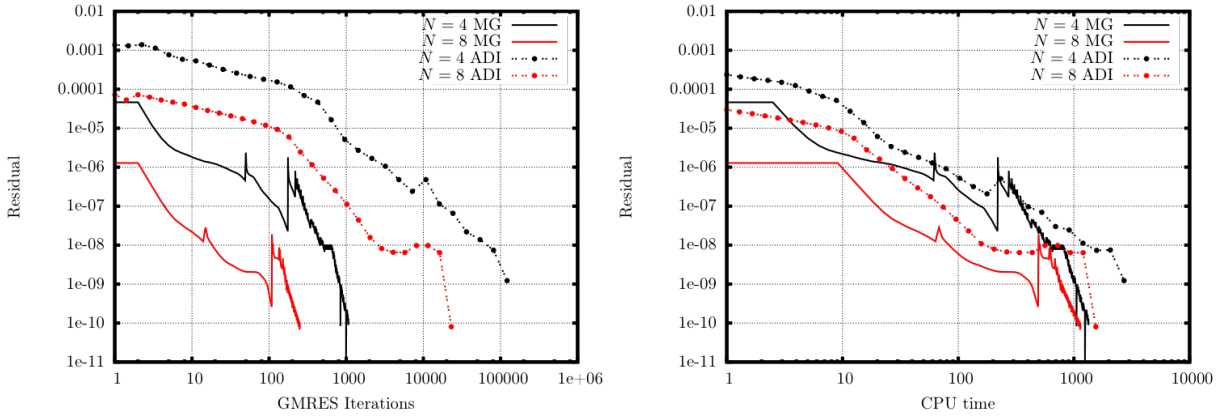


Fig. 10 Performance of the multigrid preconditioning strategy compared to the single-grid ADI preconditioner for the NACA 4412 test case at $Re = 50k$ for $N = 4; 8$.



V. Conclusion

The paper presents an efficient preconditioning strategy for solving high-order discretizations using implicit Newton-Krylov methods. A hierarchy of preconditioning methods is constructed which utilizes strong preconditioners built around approximations of the full residual Jacobian at low order, and efficient matrix-free preconditioners tailored to the physical system/discretization at higher order. This hierarchy is then implemented in a multigrid preconditioning strategy to leverage the strengths of each type of preconditioner at different polynomial approximations. Results across several physical systems and two different discretization strategies demonstrate this approach is competitive in terms of computational effort with explicit methods at CFL=1, while allowing the choice of an arbitrary timestep based upon physical considerations for multiphysics applications. We believe the approach outlined here is general, and can be applied to other discretization methods for high-order systems.

References

[1] Keyes, D. E., McInnes, L. C., Woodward, C., Gropp, W., Myra, E., Pernice, M., Bell, J., Brown, J., Clo, A., Connors, J., et al., "Multiphysics simulations: Challenges and opportunities," *The International Journal of High Performance Computing Applications*, Vol. 27, No. 1, 2013, pp. 4–83.

[2] Carton de Wiart, C., Diosady, L. T., Garai, A., Burgess, N., Blonigan, P., Ekelschot, D., and Murman, S. M., "Design of a modular monolithic implicit solver for multi-physics applications," *56th AIAA Aerospace Sciences Meeting*, 2018, p. 1400.

[3] Diosady, L., Murman, S., and Carton de Wiart, C., "A higher-order space-time finite-element method for moving-body and fluid-structure interaction problems," *10^{th} International Conference of Computational Fluid Dynamics*, 2018.

[4] Marcon, J., Garai, A., Denison, M., and Murman, S., "An Adjoint Elasticity Solver for High-Order Mesh Deformation," 2021, p. 1238.

[5] Flad, D., Pradhan, A., and Murman, S., "Arbitrary Order Solutions for the Eikonal Equation using a Discontinuous Galerkin Method," *To appear in International Journal of Numerical Methods in Engineering*, 2021.

[6] Diosady, L. T., and Murman, S. M., "Higher-order methods for compressible turbulent flows using entropy variables," *53rd AIAA Aerospace Sciences Meeting*, 2015, p. 0294.

[7] Diosady, L. T., and Murman, S. M., "Tensor-product preconditioners for higher-order space–time discontinuous Galerkin methods," *Journal of Computational Physics*, Vol. 330, 2017, pp. 296–318.

[8] Franciolini, M., and Murman, S. M., "Multigrid preconditioning for a space-time spectral-element discontinuous-Galerkin solver," *AIAA Scitech 2020 Forum*, 2020, p. 1314.

[9] Ismail, F., and Roe, P. L., "Affordable, Entropy-consistent Euler flux functions II: entropy production at shocks," *J. Comput. Phys.*, Vol. 228, No. 15, 2009, pp. 5410–5436.

[10] Bassi, F., Rebay, S., Mariotti, G., Pedinotti, S., and Savini, M., "A high-order accurate discontinuous finite element method for inviscid and viscous turbomachinery flows," *Proceedings of the 2nd European Conference on Turbomachinery Fluid Dynamics and Thermodynamics*, Antwerpen, Belgium, 1997, pp. 99–109.

[11] Diosady, L. T., and Murman, S. M., "General element shapes within a tensor-product higher-order space-time discontinuous-Galerkin formulation," *22nd AIAA Computational Fluid Dynamics Conference*, 2015, p. 3044.

[12] Diosady, L. T., and Murman, S. M., "A linear-elasticity solver for higher-order space-time mesh deformation," *2018 AIAA Aerospace Sciences Meeting*, 2018, p. 0919.

[13] Burgess, N. K., Diosady, L. T., and Murman, S. M., "A C^1-discontinuous-Galerkin spectral-element shell structural solver," *AIAA Aviation Forum 2017-3727*, 2017.

[14] Saad, Y., "A flexible inner-outer preconditioned GMRES algorithm," *SIAM Journal on Scientific Computing*, Vol. 14, No. 2, 1993, pp. 461–469.

[15] Garai, A., Diosady, L. T., Murman, S. M., and Madavan, N. K., "Scale-resolving simulations of bypass transition in a high-pressure turbine cascade using a spectral element discontinuous Galerkin method," *Journal of Turbomachinery*, Vol. 140, No. 3, 2018, p. 031004.

[16] Franciolini, M., Botti, L., Colombo, A., and Crivellini, A., "p-Multigrid matrix-free discontinuous Galerkin solution strategies for the under-resolved simulation of incompressible turbulent flows," *arXiv preprint arXiv:1809.00866*, 2018.

[17] "5^{th} High-Order Workshop," `https://how5.cenaero.be/sites/how5.cenaero.be/files/CL1_HeavingAndPitchingAirfoil.pdf`, 2018.