Efficient discontinuous Galerkin implementations and preconditioners for implicit unsteady compressible flow simulations

Efficient discontinuous Galerkin implementations and preconditioners for implicit unsteady compressible flow simulations

Matteo Franciolini[a,b,c,*], Krzysztof J. Fidkowski[a], Andrea Crivellini[b]

[a] *Department of Aerospace Engineering, University of Michigan, 1320 Beal Ave, 48109 Ann Arbor (MI), United States*
[b] *Department of Industrial Engineering and Mathematical Sciences, Polytechnic University of Marche, Via Brecce Bianche 12, 60131 Ancona, Italy*
[c] *NASA Ames Research Center, 94035 Moffett Field (CA), United States*

Abstract

This work presents and compares efficient implementations of high-order discontinuous Galerkin methods: a modal matrix-free discontinuous Galerkin (DG) method, a hybridizable discontinuous Galerkin (HDG) method, and a *primal* formulation of HDG, applied to the implicit solution of unsteady compressible flows. The matrix-free implementation allows for a reduction of the memory footprint of the solver when dealing with implicit time-accurate discretizations. HDG reduces the number of globally-coupled degrees of freedom relative to DG, at high order, by statically condensing element-interior degrees of freedom from the system in favor of face unknowns. The *primal* formulation further reduces the element-interior degrees of freedom by eliminating the gradient as a separate unknown. This paper introduces a p-multigrid preconditioner implementation for these discretizations and presents results for various flow problems. Benefits of the p-multigrid strategy relative to simpler, less expensive, preconditioners are observed for stiff systems, such as those arising from low-Mach number flows at high-order approximation. The p-multigrid preconditioner also shows excellent scalability for parallel computations. Additional savings in both speed and memory occur with a matrix-free/reduced version of the preconditioner.

Keywords:
high-order, DG, HDG, p-multigrid, preconditioners, parallel efficiency

1. Introduction

In recent years, high-order discontinuous Galerkin (DG) methods have garnered attention in the field of Computational Fluid Dynamics. With increasing availability of high-performance computing (HPC) resources, the use of high-order methods for unsteady flow simulations has become popular. The success of these methods can be attributed to attractive dispersion and diffusion properties at high orders, ease of parallelization thanks to their compact stencil,

*Corresponding author
Email address: matteo.franciolini@nasa.gov (Matteo Franciolini)

and accuracy on unstructured meshes around complex geometries. However, the implementation of an efficient solution strategy for high-order DG methods is still a subject of active research, especially for unsteady flow problems involving the solution of the Navier–Stokes (NS) equations.

Several previous studies, see for example [1, 2], demonstrated that implicit schemes in the context of high-order spatial discretizations are necessary to efficiently overcome the strict stability limits of explicit time integration schemes [3]. Implicit strategies require the solution of a large system of equations, which is typically performed with iterative solvers such as the generalized minimial residual method (GMRES). The choice of the preconditioner is a key aspect of the strategy and has been explored extensively in the literature, see for example [4, 5, 6, 7]. Among those, the use of multilevel algorithms to precondition a flexible GMRES solver [8] has been demonstrated as a promising choice for both compressible [4, 9, 10, 6] and incompressible flow problems [11, 12]. Superior iterative performance compared to single-grid preconditioned solvers has been observed, as well as better parallel efficiency on distributed-memory architectures.

The application of implicit time integration strategies to DG discretizations is hindered by computational time and memory expenses associated with the assembly and storage of the residual Jacobian matrix. Although the Jacobian is sparse, the number of non-zero entries scales as k^{2d}, where k is the approximation order and d is the spatial dimension. Thus, the costs grow rapidly with approximation order, particularly in three-dimensional problems. Motivated by this scaling, previous works [13, 14, 15, 12] considered the possibility of a reduced matrix storage ("matrix-free") implementation of the iterative solver. This implementation avoids the allocation of the Jacobian matrix but still requires the allocation of a preconditioner operator which in some cases may still be quite large. In this context, the use of multilevel matrix-free strategies with cheap element-wise block-Jacobi preconditioners on the finest level appears to balance computational efficiency and memory considerations with iterative performance for stiff systems. The latter is relevant to solvers applied to DG discretizations, for which the condition number scales as $O(h^{-2})$ [16], where h is the mesh dimension.

The size of the DG linear system can be reduced through hybridizable discontinuous Galerkin (HDG) methods, which have been recently considered as an alternative to the standard discontinous Galerkin discretization [17, 18]. HDG methods introduce an additional trace variable on the mesh faces but can reduce the number of globally-coupled degrees of freedom relative to DG, when a high order of polynomial approximation is employed. The reduction occurs through a static condensation of the element-interior degrees of freedom, exploiting the block structure of the HDG Jacobian matrix. Thanks to this operation, the memory footprint of the solver scales as $k^{2(d-1)}$. Additionally, HDG methods exhibit superconvergence properties of the gradient variable in diffusion-dominated regimes. On the other hand, they increase the number of operations local to each element, both before and after the linear system solution. While several works have compared the accuracy and cost of HDG versus continuous [19, 20] and discontinuous [21, 18] Galerkin methods, a comparison considering the efficiency of iterative solvers applied to the solution of unsteady flows is missing in this context. In fact, it is worth pointing out that, whereas HDG reduces the number of globally-coupled degrees of freedom relative to DG, at high approximation orders, its element-local operation count is non-

trivial. This is particularly the case for viscous problems, in which the state gradient is approximated as a separate variable in a *mixed*-type fashion. An alternative approach is to only approximate the state, and to obtain the gradient when needed by differentiating the state. This leads to the *primal* HDG formulation [22, 23], which we also consider in this work. The advantage of *primal* HDG relative to standard, *mixed* HDG lies mainly in the reduction of element-local operations, which translates into improved computational performance.

While for DG the use of multilevel strategies to deal with ill-conditioned systems has been previously studied, their use in HDG contexts appears not to have yet been explored, especially for unsteady flow problems. A multilevel technique has been introduced in the context of an h-multigrid strategy built using the trace variable projection on a continuous finite element space [16]. In addition, the use of an algebraic multigrid method applied to a linear finite element space obtained by Galerkin projection has been proposed in the context of elliptic problems [24]. A similar idea is also considered to speed-up the iterative solution process in HDG [25].

The present work focuses on the comparison of implicit solution strategies in the context of unsteady flow simulations for the three aforementioned implementations, *i.e.*, DG, *mixed* HDG, and *primal* HDG. The comparison includes efficient preconditioning, such as p-multigrid, to deal with the solution of stiff linear systems arising from high-order time discretizations. In particular, an efficient algorithm to inherit the coarse space operators at a low computational cost in the context of HDG is presented for the first time. The scalability of the linear solution process is also considered and compared to standard single-grid preconditioners, such as a incomplete lower-upper factorization, ILU(0) [6]. The efficiency of the different solution strategies and the overall memory footprint is assessed on two-dimensional laminar compressible flow problems. The results of these cases demonstrate that (i) the different discretizations attain similar error levels, (ii) the use of a multilevel strategy reduces the number of linear iterations in all cases tested, (iii) only for the DG discretizations is this advantage reflected in the CPU time, and iv) the *primal* HDG and p-multigrid matrix-free DG solvers yield comparable solution times and memory footprints, faster than standard, single-grid preconditioners like ILU(0) applied to DG.

The paper is structured as follows. Section 2 presents the differential equations, and Sections 3 and 4 show the spatial and temporal discretizations used in this work. Section 6 presents the p-multigrid preconditioner implementations. Section 7 reports numerical experiments using different preconditioning strategies, including ILU(0), block Jacobi, and p-multigrid. These experiments are performed on a range of test cases, including two-dimensional airfoils and a circular cylinder. The methods are compared in terms of iterations, computational time, and memory footprint. To clarify our nomenclature, we report in Table 1 the meaning of each abbreviation that will be used throughout the text.

DG	standard discontinuous Galerkin discretization
MB	matrix-based (F)GMRES
MF	matrix-free (F)GMRES
MFL	matrix-free (F)GMRES with preconditioner lagging
*m*HDG	*mixed* hybridizable discontinuous Galerkin discretization
*p*HDG	*primal* hybridizable discontinuous Galerkin discretization
BJ	element-by-element block Jacobi preconditioner
BILU	partition-by-partition block ILU(0) preconditioner

Table 1: Nomenclature used throughout the text.

2. Governing equations

This work considers solutions of the compressible Navier–Stokes (NS) equations, which can be written as

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \vec{v}) &= 0, \\ \frac{\partial}{\partial t}(\rho \vec{v}) + \nabla \cdot (\rho \vec{v} \otimes \vec{v} + p\mathbf{I}) &= \nabla \cdot \boldsymbol{\tau}, \\ \frac{\partial}{\partial t}(\rho e_0) + \nabla \cdot (\rho \vec{v} h_0) &= \nabla \cdot (\vec{v} \cdot \boldsymbol{\tau} - \vec{q}), \end{aligned} \tag{1}$$

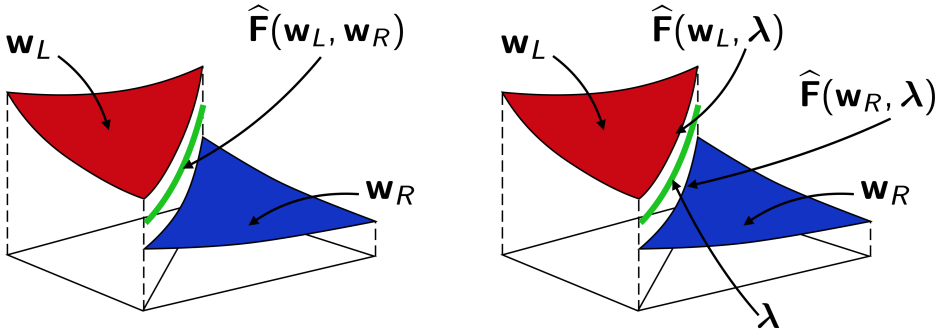
with $\vec{v} \in \mathbb{R}^d$ the velocity, and d the number of space dimensions. The total energy e_0, total enthalpy h_0, pressure p, total stress tensor $\boldsymbol{\tau}$, and heat flux $\vec{q}$ are given by

$$\begin{aligned} &e_0 = e + (\vec{v} \cdot \vec{v})/2, \qquad h_0 = e_0 + p/\rho, \qquad p = (\gamma - 1)\rho e, \\ &\boldsymbol{\tau} = 2\mu \left(\mathbf{S} - \tfrac{1}{3} (\nabla \cdot \vec{v}) \mathbf{I} \right), \qquad \vec{q} = -\tfrac{\mu}{\mathrm{Pr}} \nabla h, \qquad \mathbf{S} = \tfrac{1}{2} \left(\nabla \vec{v} + (\nabla \vec{v})^T \right). \end{aligned}$$

Here e is the internal energy, h is the enthalpy, $\gamma = c_p/c_v$ is the ratio of gas specific heats, μ is the viscosity, Pr is the molecular Prandtl number, and $\mathbf{S}$ is the mean strain-rate tensor. The space and time discretizations are outlined in the following sections.

3. Spatial discretization

Three versions of a modal discontinuous Galerkin (DG) finite element method are considered in this work. The first one is a standard DG implementation, which employs basis functions defined in the reference element space. The second one, commonly referred to as the hybridizable discontinuous Galerkin (HDG) method, introduces an additional set of variables defined on the mesh element interfaces to reduce the globally coupled degrees of freedom compared to DG, as shown in Figure 1. This implementation explicitly uses a mixed form for the gradient states (also known as the *dual variable*), *i.e.* the gradients are used as an additional element-wise variable, and increase the accuracy of the gradient evaluation. A third implementation, *primal* HDG [22], reduces the computational costs of the solver by removing the dual variable from the equations. This approach is desirable when an increased accuracy of



(a) DG (b) HDG

Figure 1: Schematic comparison of the solution approximation and flux evaluation in DG versus HDG.

the gradient variable is not strictly necessary or achievable for the problem. In all cases, two types of basis functions in the reference space have been considered: polynomial functions of maximum degree equal to k as well as tensor product functions of degree k in each dimension. In this work the former is employed within triangular grids, and the number of degrees of freedom per equation per element is $n_v^\ell = \prod_{i=1}^{d} (k+i)/i$. The latter approach is used for quadrilateral mesh elements, and $n_v = (k+1)^d$.

In all cases, the discretization is based on an approximation Ω_h of the domain Ω and a triangulation $\mathcal{T}_h = \{K\}$ of Ω_h made by a set of n_e non-overlapping elements, denoted by K. Here, $\mathcal{F}_h^i$ stands for the set of internal element faces, $\mathcal{F}_h^b$ the set of boundary element faces and $\mathcal{F}_h = \mathcal{F}_h^i \cup \mathcal{F}_h^b$ their union. We also define

$$\Gamma_h^i = \bigcup_{F \in \mathcal{F}_h^i} F, \qquad \Gamma_h^b = \bigcup_{F \in \mathcal{F}_h^b} F, \qquad \Gamma_h = \Gamma_h^i \cup \Gamma_h^b. \tag{2}$$

where F denotes a generic mesh element face. Following Brezzi et al. [26], we also introduce the average trace operator, which on a generic internal face $F \in \mathcal{F}_h^i$ is defined as $\{\cdot\} \overset{\text{def}}{=} \frac{(\cdot)^+ + (\cdot)^-}{2}$, where $(\cdot)$ denotes a generic scalar or vector quantity. This definitions can be suitably extended to domain boundary faces by accounting for the weak imposition of boundary conditions.

3.1. Discontinuous Galerkin

In compact form, the system (1) can be expressed as

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \vec{\mathbf{F}}_c(\mathbf{u}) + \nabla \cdot \vec{\mathbf{F}}_v(\mathbf{u}, \nabla \mathbf{u}) = \mathbf{0}, \tag{3}$$

where $\mathbf{u} \in \mathbb{R}^m$ is the vector of conservative variables and $\vec{\mathbf{F}}_c, \vec{\mathbf{F}}_v \in \mathbb{R}^{m \times d}$ are the inviscid and viscous fluxes, with m the number of equations and d the number of space dimensions. Note that the diffusive fluxes are connected to the gradients of $\mathbf{u}$ via the functional relation $\vec{\mathbf{F}}_d = -\mathbf{K}\nabla\mathbf{u}$, with $\mathbf{K}$ the diffusivity tensor.

The state vector is approximated by a polynomial expansion with no continuity constraints imposed between adjacent elements, *i.e.*$\mathbf{u}_h \in [\mathcal{V}_h]^m$ where

$$\mathcal{V}_h = \{\phi_h \in L_2(\Omega) : \phi_h|_K \in \mathbb{P}_k, \forall K \in \mathcal{T}_h\}, \tag{4}$$

and k is the order of polynomial approximation. The weak form of (3) follows from multiplying the PDE by the set of test functions in the same approximation space, integrating by parts, and coupling elements via numerical fluxes. The variational formulation for each element $K \in \mathcal{T}_h$ reads: find $\mathbf{u}_h \in [\mathcal{V}_h]^m$ such that

$$\begin{aligned} \int_K \mathbf{w}_h \cdot \frac{\partial \mathbf{u}_h}{\partial t} \, \mathrm{d}K - \int_K \nabla_h \mathbf{w}_h : \vec{\mathbf{F}}(\mathbf{u}_h, \nabla_h \mathbf{u}_h) \, \mathrm{d}K \\ + \int_{\partial K} \mathbf{w}_h^+ \cdot \widehat{\mathbf{F}}(\mathbf{u}_h^\pm, \nabla_h \mathbf{u}_h^\pm, \vec{n}^+) \, \mathrm{d}\sigma - \int_{\partial K} \nabla \mathbf{w}_h^+ : (\mathbf{K}^+ \cdot \vec{n}^+)(\mathbf{u}_h^+ - \hat{\mathbf{u}}_h) \, \mathrm{d}\sigma = 0, \end{aligned} \tag{5}$$

for all $\mathbf{w}_h \in [\mathcal{V}_h]^m$. Here, $\vec{\mathbf{F}}$ is the sum of the inviscid and viscous flux functions, while $\widehat{\mathbf{F}}$ is the numerical flux and $\hat{\mathbf{u}}_h = (\mathbf{u}_h^+ + \mathbf{u}_h^-)/2$. Note that the quantities $(\cdot)^+$ and $(\cdot)^-$ denote element interior and element neighbor quantities, respectively.

Uniqueness and local conservation of the solution are achieved by the use of proper numerical interface fluxes. The Roe [27] approximate Riemann solver is employed for the inviscid part $\widehat{\mathbf{F}}_c$, while the second form of Bassi and Rebay (BR2) [28] is employed for the viscous part, $\widehat{\mathbf{F}}_v$. Following BR2, the numerical viscous flux is given by

$$\widehat{\mathbf{F}}_v \left(\mathbf{u}_h^\pm, \nabla_h \mathbf{u}_h^\pm, \vec{n}^+ \right) \overset{\text{def}}{=} \{ \vec{\mathbf{F}}_v (\mathbf{u}_h, \nabla_h \mathbf{u}_h) \} \cdot \vec{n} + \eta_F \{ \vec{\boldsymbol{\delta}}_F (\mathbf{u}_h^+ - \mathbf{u}_h^-) \} \cdot \vec{n}^+, \tag{6}$$

where, according to [29, 26], the penalty factor η_F must be greater than the number of faces of the elements. The auxiliary variable $\vec{\boldsymbol{\delta}}$ is determined from the jump of $\mathbf{u}_h$, via the solution of the following auxiliary problem:

$$\int_K \vec{\boldsymbol{\tau}}_h^+ : \vec{\boldsymbol{\delta}}_F^+ \, \mathrm{d}K = \frac{1}{2} \int_F \vec{\boldsymbol{\tau}}_h^+ : (\mathbf{K}^+ \cdot \vec{n}^+) \, (\mathbf{u}_h^+ - \mathbf{u}_h^-) \, \mathrm{d}\sigma, \quad \forall \boldsymbol{\tau}_h \in [\mathcal{V}_h]^{d \times m}. \tag{7}$$

At the boundary of the domain, the numerical flux function appearing in equation (5) must be consistent with the boundary conditions of the problem. In practice, this is accomplished by properly defining a boundary state which accounts for the boundary data and, together with the internal state, allows for the computation of the numerical fluxes and the lifting operator on the portion Γ_h^b of the boundary Γ_h, see [28, 30].

A system of ordinary differential equations for the degrees of freedom (DoFs) arising from Equation (5) can be compactly written in the form

$$\mathbf{M} \frac{d\mathbf{W}}{dt} + \mathbf{R}(\mathbf{W}) = \mathbf{0}, \tag{8}$$

where $\mathbf{M}$ is the block-diagonal spatial mass matrix, $\mathbf{W}$ is the vector of the DoFs of the problem, and $\mathbf{R}$ is the spatial residual vector.

3.2. Hybridizable discontinuous Galerkin

3.2.1. Mixed form

The second spatial discretization considered in this work is the HDG method in mixed form (*m*HDG), see [18]. A system of first-order partial differential equations can be obtained from (1) by introducing $\vec{\mathbf{q}} \in \mathbb{R}^{m \times d}$,

$$\begin{aligned} \vec{\mathbf{q}} - \nabla \mathbf{u} &= \vec{\mathbf{0}}, \\ \frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot \mathbf{F}_c(\mathbf{u}) + \nabla \cdot \mathbf{F}_v(\mathbf{u}, \vec{\mathbf{q}}) &= \mathbf{0}. \end{aligned} \tag{9}$$

The HDG discretization approximates the state and gradient variables as $\mathbf{u}_h \in [\mathcal{V}_h]^m$ and $\vec{\mathbf{q}}_h \in [\mathcal{V}_h]^{m\times d}$, with $\mathcal{V}_h$ defined in (4). An additional trace variable, $\boldsymbol{\lambda}_h \in [\mathcal{M}_h]^m$, is defined on the faces, using the space

$$\mathcal{M}_h = \left\{\mu \in L_2(\mathcal{F}_h^i) : \mu|_F \in \mathbb{P}_k, \forall F \in \mathcal{F}_h^i\right\}, \tag{10}$$

where $\mathbb{P}_k$ is the space of polynomials of order k on face F. We remark that the trace variable is defined on the internal faces only, while a properly defined boundary value is used for the flux computation on $\mathcal{F}_h^b$.

The weak form is obtained by weighting the equations in (9) with appropriate test functions, integrating by parts, and using the interface variable $\boldsymbol{\lambda}_h$ for the face state. Consistent and stable numerical fluxes are required at the mesh element interfaces. The variational formulation reads: find $\mathbf{u}_h \in [\mathcal{V}_h]^m$, $\vec{\mathbf{q}}_h \in [\mathcal{V}_h]^{m\times d}$, $\boldsymbol{\lambda}_h \in [\mathcal{M}_h]^m$, such that

$$\int_K \vec{\mathbf{v}}_h : \vec{\mathbf{q}}_h \,\mathrm{d}K + \int_K (\nabla_h \cdot \mathbf{v}_h) \cdot \mathbf{u}_h \,\mathrm{d}K - \int_{\partial K} (\vec{\mathbf{v}}_h^+ \cdot \vec{n}^+) \cdot \boldsymbol{\lambda}_h \,\mathrm{d}\sigma = \mathbf{0}, \tag{11}$$

$$\int_K \mathbf{w}_h \cdot \frac{\partial \mathbf{u}_h}{\partial t} \,\mathrm{d}K - \int_K \nabla_h \mathbf{w}_h : \vec{\mathbf{F}}(\mathbf{u}_h, \vec{\mathbf{q}}_h) \,\mathrm{d}K + \int_{\partial K} \mathbf{w}_h^+ \cdot \widehat{\mathbf{F}}(\mathbf{u}_h^+, \vec{\mathbf{q}}_h^+, \boldsymbol{\lambda}_h, \vec{n}^+) \,\mathrm{d}\sigma = \mathbf{0}, \tag{12}$$

$$\int_{\partial K} \boldsymbol{\mu}_h \cdot \left\{\widehat{\mathbf{F}}(\mathbf{u}_h^+, \vec{\mathbf{q}}_h^+, \boldsymbol{\lambda}_h, \vec{n}^+) + \widehat{\mathbf{F}}(\mathbf{u}_h^-, \vec{\mathbf{q}}_h^-, \boldsymbol{\lambda}_h, \vec{n}^-)\right\} \,\mathrm{d}\sigma = \mathbf{0}, \tag{13}$$

for all $\mathbf{w}_h \in [\mathcal{V}_h]^m$, $\vec{\mathbf{v}}_h \in [\mathcal{V}_h]^{m\times d}$, $\boldsymbol{\mu}_h \in [\mathcal{M}_h]^m$. The third equation, which weakly imposes flux continuity across interior faces, is required to close the system. We remark that, when using the mixed form, the same theoretical convergence rate is observed for the state variable $\mathbf{u}_h$ and the gradient variable $\vec{\mathbf{q}}_h$. In diffusion-dominated regimes, this allows for a local post-processing of the state to a higher order [17].

In HDG, the numerical flux function $\widehat{\mathbf{F}}$, which is the sum of the inviscid and viscous fluxes, is defined as

$$\widehat{\mathbf{F}}(\mathbf{u}_h, \vec{\mathbf{q}}_h, \boldsymbol{\lambda}_h, \vec{n}) = \vec{\mathbf{F}}(\boldsymbol{\lambda}_h, \vec{\mathbf{q}}_h) \cdot \vec{n} + \boldsymbol{\tau}(\boldsymbol{\lambda}_h, \mathbf{u}_h, \vec{n}), \tag{14}$$

where $\boldsymbol{\tau} = \boldsymbol{\tau}_c + \boldsymbol{\tau}_v$ is a stabilization term for both the inviscid and viscous parts of the flux. In this work $\boldsymbol{\tau}_c$ is chosen in a Roe-like fashion as

$$\boldsymbol{\tau}_c = \left|\vec{\mathbf{F}}_c'(\boldsymbol{\lambda}_h) \cdot \vec{n}\right| (\mathbf{u}_h - \boldsymbol{\lambda}_h), \tag{15}$$

while the viscous stabilization term $\boldsymbol{\tau}_v$ is based on the BR2 scheme,

$$\boldsymbol{\tau}_v = \eta_F \vec{\boldsymbol{\delta}}_F(\mathbf{u}_h - \boldsymbol{\lambda}_h) \cdot \vec{n}, \tag{16}$$

with $\vec{\boldsymbol{\delta}}_F$ the lifting operator applied to the jump $(\mathbf{u}_h - \boldsymbol{\lambda}_h)$, and η_F the stabilization factor.

Similarly to DG, at the boundary of the domain, the numerical flux function is made consistent with the boundary conditions of the problem through the definition of a boundary state which accounts for the boundary data and,

together with the internal state, allows for the computation of numerical fluxes and the lifting operator on the portion Γ_h^b of the boundary Γ_h.

Defining $\mathbf{R}^Q$, $\mathbf{R}^U$ and $\mathbf{R}^\Lambda$ as the residual vectors arising from Equations (11), (12) and (13), the discretized system of nonlinear equations can be written as

$$\begin{aligned} \mathbf{R}^Q &= \mathbf{0}, \\ \mathbf{M}^U \frac{d\mathbf{U}}{dt} + \mathbf{R}^U &= \mathbf{0}, \\ \mathbf{R}^\Lambda &= \mathbf{0}. \end{aligned} \tag{17}$$

where $\mathbf{M}^U$ is the element-based mass matrix. The compact form of (17) can be written using the solution vector of the discrete unknowns, $\mathbf{W} = [\mathbf{Q}; \mathbf{U}; \mathbf{\Lambda}]$, and the concatenated vector of residuals $\mathbf{R} = [\mathbf{R}^Q; \mathbf{R}^U; \mathbf{R}^\Lambda]$,

$$\mathbf{M}\frac{d\mathbf{W}}{dt} + \mathbf{R}(\mathbf{W}) = \mathbf{0}, \tag{18}$$

where the matrix $\mathbf{M}$ is given by

$$\mathbf{M} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{M}^U & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}. \tag{19}$$

3.2.2. Primal form

A variant of the mixed hybridizable discontinuous Galerkin method presented in Section 3.2.1 is the *primal* HDG (*p*HDG) method and follows the work in [22, 23]. In *p*HDG, the dual variable is eliminated by introducing the definition of the gradient in (12).

The *p*HDG discretization approximates the variable $\mathbf{u}_h \in [\mathcal{V}_h]^m$, with $\mathcal{V}_h$ defined in Equation (4). The trace variable $\boldsymbol{\lambda}_h \in [\mathcal{M}_h]^m$ is still employed for hybridization, and the variational formulation reads: find $\mathbf{u}_h \in [\mathcal{V}_h]^m$, $\boldsymbol{\lambda}_h \in [\mathcal{M}_h]^m$ such that

$$\begin{aligned} &\int_K \mathbf{w}_h \cdot \frac{\partial \mathbf{u}_h}{\partial t}\, \mathrm{d}K - \int_K \nabla_h \mathbf{w}_h : \vec{\mathbf{F}}(\mathbf{u}_h, \nabla_h \mathbf{u}_h)\, \mathrm{d}K \\ &+ \int_{\partial K} \mathbf{w}_h^+ \cdot \widehat{\mathbf{F}}(\mathbf{u}_h^+, \nabla_h \mathbf{u}_h^+, \boldsymbol{\lambda}_h, \vec{n}^+)\, \mathrm{d}\sigma - \int_{\partial K} \nabla \mathbf{w}_h^+ : (\mathbf{K}^+ \cdot \vec{n}^+)(\mathbf{u}_h^+ - \hat{\mathbf{u}}_h)\, \mathrm{d}\sigma = \mathbf{0}, \end{aligned} \tag{20}$$

$$\int_{\partial K} \boldsymbol{\mu}_h \cdot \left\{ \widehat{\mathbf{F}}(\mathbf{u}_h^+, \vec{\mathbf{q}}_h^+, \boldsymbol{\lambda}_h, \vec{n}^+) + \widehat{\mathbf{F}}(\mathbf{u}_h^-, \vec{\mathbf{q}}_h^-, \boldsymbol{\lambda}_h, \vec{n}^-) \right\} \mathrm{d}\sigma = \mathbf{0}, \tag{21}$$

for all $\mathbf{w}_h \in [\mathcal{V}_h]^m$, $\boldsymbol{\mu}_h \in [\mathcal{M}_h]^m$. We note that (20)–(21) are not obtained by just substituting $\vec{\mathbf{q}}_h = \nabla \mathbf{u}_h$ from (11)–(13). In fact the fourth term of (20) arises from the elimination of the variable $\vec{\mathbf{q}}_h$. This term ensures symmetry and adjoint-consistency of the *primal* HDG discretization. This type of discretization, involving a smaller number of element-wise degrees of freedom than mixed HDG, does not suffer significantly from overhead costs of dealing with the gradients: eliminating the dual variable and adding the adjoint-consistency term typically results in a faster solver.

We note that in this case, the gradients are of one order lower accuracy than the state variable $\mathbf{u}_h$. Numerical flux functions, stabilizing terms, and boundary condition enforcement are defined in the same manner as in mixed HDG.

Defining $\mathbf{R}^U$ and $\mathbf{R}^\Lambda$ as the residuals vectors arising from (20)–(21), the ODE system of equations can be written as

$$\begin{aligned} \mathbf{M}^U \frac{d\mathbf{U}}{dt} + \mathbf{R}^U &= \mathbf{0}, \\ \mathbf{R}^\Lambda &= \mathbf{0}, \end{aligned} \tag{22}$$

where $\mathbf{M}^U$ is the elemental mass matrix. Therefore, the compact form of (22) is written using the solution vector of discrete unknowns, $\mathbf{W} = [\mathbf{U}; \mathbf{\Lambda}]$, and the concatenated vector of residuals, $\mathbf{R} = [\mathbf{R}^U; \mathbf{R}^\Lambda]$,

$$\mathbf{M}\frac{d\mathbf{W}}{dt} + \mathbf{R}(\mathbf{W}) = \mathbf{0}, \tag{23}$$

where the matrix $\mathbf{M}$ is given by

$$\mathbf{M} = \begin{bmatrix} \mathbf{M}^U & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}. \tag{24}$$

4. Temporal discretization

The temporal discretization used in this work is an explicit-first stage, singly-diagonal-implicit Runge–Kutta (ESDIRK) scheme. The general formulation of the scheme for (18) is

$$\begin{aligned} \mathbf{M}\mathbf{W}^i &= \mathbf{M}\mathbf{W}^n - \Delta t \sum_{j=1}^{i} a_{ij}\mathbf{R}(\mathbf{W}^j), \\ \mathbf{W}^{n+1} &= \mathbf{W}^n + \Delta t \sum_{i=1}^{s} \beta_i \mathbf{W}^i, \end{aligned} \tag{25}$$

for $i = 1, ..., s$ where s is the number of stages, a_{ij} and b_i are the coefficients of the scheme, and n is the time index. Within each stage, the solution of a non-linear system is required. This is performed by the Newton-Krylov method, which requires the solution of a sequence of linear systems within each stage. In this regard, the k^{th} Newton–Krylov iteration assumes the form

$$\left(\frac{\mathbf{M}}{a_{ii}\Delta t} + \frac{\partial \mathbf{R}}{\partial \mathbf{W}}\right)(\mathbf{W}^i_{k+1} - \mathbf{W}^i_k) = -\frac{\mathbf{M}}{a_{ii}\Delta t}(\mathbf{W}^i_k - \mathbf{W}^n) - \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}}\mathbf{R}(\mathbf{W}^j) - \mathbf{R}(\mathbf{W}^i_k), \tag{26}$$

with $i = 1, ..., s$. In this work the third-order ESDIRK3 scheme [31] is employed. The method involve three non-linear solutions, following an explicit first stage.

5. Linear system solution

The linear system of ODEs can be solved numerically using iterative solvers. To this end, we apply the generalized minimal residual (GMRES) method to the system in (26). The solution process differs between standard DG and HDG, as the latter discretization takes advantage of the introduction of face unknowns in order to reduce the size of the matrix to be allocated. This section provides details of the implementation.

5.1. Discontinuous Galerkin discretization

The linear system arising from a DG discretizations takes the following general form

$$\mathcal{K}\mathbf{x} + \mathbf{b} = \mathbf{0}, \tag{27}$$

where $\mathcal{K} = (\mathbf{M}/(a_{ii}\Delta t) + \partial\mathbf{R}/\partial\mathbf{W})$ is the iteration matrix, $\mathbf{x} = \Delta\mathbf{W}$ is the vector of degrees of freedom updates, and $\mathbf{b}$ is the right-hand side. The GMRES implementation can follow two approaches. The first one is denoted as matrix-based (MB) and consists of computing and storing the iteration matrix explicitly to perform matrix-vector products as needed within the iterative solution. A second, matrix-free (MF) approach takes advantage of the structure of the matrix vector products, which can be approximated using the matrix-free formula

$$\left(\frac{\mathbf{M}}{a_{ii}\Delta t} + \frac{\partial\mathbf{R}}{\partial\mathbf{W}}\right)\Delta\mathbf{W} \approx \frac{\mathbf{M}}{a_{ii}\Delta t}\Delta\mathbf{W} + \frac{\mathbf{R}(\mathbf{W} + h\Delta\mathbf{W}) - \mathbf{R}(\mathbf{W})}{h}, \tag{28}$$

with

$$h = \varepsilon\frac{\sqrt{1 + ||\Delta\mathbf{W}||}}{||\mathbf{W}||} \tag{29}$$

and $\varepsilon \approx 10^{-7}$. The latter approach offers several advantages over the former, as the Jacobian matrix is no longer required to maintain the temporal accuracy of the solution. The GMRES solver still requires a preconditioning matrix, which generally needs to be stored. However, this matrix can be approximated or frozen for a certain number of iterations without losing the formal order of accuracy of the time integration scheme, thus providing an improvement in computational efficiency. For example, when using a block-Jacobi preconditioning approach, the memory footprint required for the Jacobian matrix can be one order of magnitude lower, as only the memory for the on-diagonal blocks needs to be allocated. The additional residual evaluation, which are required in (28), have a computational cost similar to a matrix-vector product for high-order polynomials. Further details can be found in previous work [15, 13].

5.2. Hybridizable discontinuous Galerkin discretization

5.2.1. Mixed form

The system in (26) can be conveniently arranged using the definition of element-interior and face DoFs. To this end, considering first the mixed form of the HDG discretization, it is convenient to define the following elemental block matrices at stage i of the Newton-Krylov method

$$\begin{aligned}
\mathbf{A}^{QQ} &= \frac{\partial\mathbf{R}^{Q}}{\partial\mathbf{Q}}, \quad \mathbf{A}^{QU} = \frac{\partial\mathbf{R}^{Q}}{\partial\mathbf{U}}, & \mathbf{B}^{Q\Lambda} &= \frac{\partial\mathbf{R}^{Q}}{\partial\mathbf{\Lambda}}, \\
\mathbf{A}^{UQ} &= \frac{\partial\mathbf{R}^{U}}{\partial\mathbf{Q}}, \quad \mathbf{A}^{UU} = \frac{\mathbf{M}^{U}}{a_{ii}\Delta t} + \frac{\partial\mathbf{R}^{U}}{\partial\mathbf{U}}, & \mathbf{B}^{U\Lambda} &= \frac{\partial\mathbf{R}^{U}}{\partial\mathbf{\Lambda}}, \\
\mathbf{C}^{\Lambda Q} &= \frac{\partial\mathbf{R}^{\Lambda}}{\partial\mathbf{Q}}, \quad \mathbf{C}^{\Lambda U} = \frac{\partial\mathbf{R}^{\Lambda}}{\partial\mathbf{U}}, & \mathbf{D} &= \frac{\partial\mathbf{R}^{\Lambda}}{\partial\mathbf{\Lambda}},
\end{aligned} \tag{30}$$

while the right hand side of Eq. (26) is obtained as

$$\begin{aligned}
\mathbf{f}^{Q} &= -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} \mathbf{R}^{Q}(\mathbf{Q}^{j}, \mathbf{U}^{j}, \mathbf{\Lambda}^{j}) - \mathbf{R}^{Q}(\mathbf{Q}_{k}^{i}, \mathbf{U}_{k}^{i}, \mathbf{\Lambda}_{k}^{i}), \\
\mathbf{f}^{U} &= -\frac{\mathbf{M}^{U}}{a_{ii}\Delta t}\left(\mathbf{U}_{k}^{i} - \mathbf{U}^{n}\right) - \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} \mathbf{R}^{U}(\mathbf{Q}^{j}, \mathbf{U}^{j}, \mathbf{\Lambda}^{j}) - \mathbf{R}^{U}(\mathbf{Q}_{k}^{i}, \mathbf{U}_{k}^{i}, \mathbf{\Lambda}_{k}^{i}), \\
\mathbf{g} &= -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} \mathbf{R}^{\Lambda}(\mathbf{Q}^{j}, \mathbf{U}^{j}, \mathbf{\Lambda}^{j}) - \mathbf{R}^{\Lambda}(\mathbf{Q}_{k}^{i}, \mathbf{U}_{k}^{i}, \mathbf{\Lambda}_{k}^{i}).
\end{aligned} \tag{31}$$

The full system of equations can be therefore written in the compact form as

$$\begin{bmatrix} \mathbf{A}^{QQ} & \mathbf{A}^{QU} & \mathbf{B}^{Q\Lambda} \\ \mathbf{A}^{UQ} & \mathbf{A}^{UU} & \mathbf{B}^{U\Lambda} \\ \mathbf{C}^{\Lambda Q} & \mathbf{C}^{\Lambda U} & \mathbf{D} \end{bmatrix} \begin{pmatrix} \Delta\mathbf{Q} \\ \Delta\mathbf{U} \\ \Delta\mathbf{\Lambda} \end{pmatrix} + \begin{pmatrix} \mathbf{f}^{Q} \\ \mathbf{f}^{U} \\ \mathbf{g} \end{pmatrix} = \mathbf{0}. \tag{32}$$

5.2.2. Primal form

In the *primal* formulation, the elemental block matrices related to the gradient variables are no longer present in the linear system. Moreover, the right-hand side can be evaluated via the following equations

$$\begin{aligned}
\mathbf{f}^{U} &= -\frac{\mathbf{M}^{U}}{a_{ii}\Delta t}\left(\mathbf{U}_{k}^{i} - \mathbf{U}^{n}\right) - \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} \mathbf{R}^{U}(\mathbf{U}^{j}, \mathbf{\Lambda}^{j}) - \mathbf{R}^{U}(\mathbf{U}_{k}^{i}, \mathbf{\Lambda}_{k}^{i}), \\
\mathbf{g} &= -\sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} \mathbf{R}^{\Lambda}(\mathbf{U}^{j}, \mathbf{\Lambda}^{j}) - \mathbf{R}^{\Lambda}(\mathbf{U}_{k}^{i}, \mathbf{\Lambda}_{k}^{i}).
\end{aligned} \tag{33}$$

that do not depend anymore on the internal DoFs $\mathbf{Q}$. The linear system for the primal HDG discretization assumes the form

$$\begin{bmatrix} \mathbf{A}^{UU} & \mathbf{B}^{U\Lambda} \\ \mathbf{C}^{\Lambda U} & \mathbf{D} \end{bmatrix} \begin{pmatrix} \Delta\mathbf{U} \\ \Delta\mathbf{\Lambda} \end{pmatrix} + \begin{pmatrix} \mathbf{f}^{U} \\ \mathbf{g} \end{pmatrix} = \mathbf{0}, \tag{34}$$

5.3. Static condensation and back-solve

Considering the block structure of the matrices appearing in (32) and (34), the solution of the system can involve a smaller number of DoFs by statically condensing out the element-interior variables. Partitioning the matrix into element-interior and face components, $[\mathbf{A}, \mathbf{B}; \mathbf{C}, \mathbf{D}]$, and similarly for the right-hand side vector, $[\mathbf{f}; \mathbf{g}]$, the Schur-complement linear system reads

$$\underbrace{(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B})}_{\mathcal{K}} \Delta\mathbf{\Lambda} = \underbrace{(\mathbf{g} - \mathbf{C}\mathbf{A}^{-1}[\mathbf{f}])}_{\mathbf{b}}, \tag{35}$$

which assumes the same form as system (27) and can be solved using a GMRES algorithm. The definition of each block can be found from (32) and (34). The static condensation is an operation that involves matrix-matrix products for the iteration matrix, as well as matrix-vector products for the right-hand side. Fortunately, the compact structure of the residual Jacobian prevents us from having to allocate global matrices for the computation of the condensed matrix, *i.e.* the operations described in Eq. (35) are local to each element. In addition, the computation of $\mathbf{A}^{-1}$ can

be performed in place. By doing so, we do not increase the memory footprint of the HDG implementation during the solve.

After the solution of (35), the interior states have to be recovered for the residual evaluation in the next time step. This operation is commonly referred to as the *back solve* and assumes the following form

$$\Delta \mathbf{U} = -\mathbf{A}^{-1} \left(\mathbf{f} + \mathbf{B} \Delta \mathbf{\Lambda} \right). \tag{36}$$

Our implementation choice of assembling the condensed matrix on-the-fly requires re-evaluation of the inverse of the matrix $\mathbf{A}$ in an element-wise fashion during the back-solve.

As a final remark for the two solvers, we point out that for both the *mixed* and *primal* form of HDG, memory allocation and time spent on the global solve are lower than that of a DG solver due to the smaller number of globally-coupled degrees of freedom at high orders. On the other hand, the inversion of the $\mathbf{A}$ block-structured matrix of equation (35), although local to each element, increases the amount of element-wise operations.

6. Multigrid preconditioning

The use of a p-multigrid strategy to precondition a flexible implementation [8] of GMRES is explored in the context of the spatial discretizations presented. The basic multigrid idea is to exploit iterative solvers to smooth-out high-frequency components of the error with respect to an unknown exact solution. Such iterative solvers are not sufficiently effective at damping low-frequency error components, and to this end, an iterative solution built using coarser problems can be useful to shift, via system projection, the low-frequency modes towards the upper side of the spectrum. This simple and effective strategy allows us to obtain satisfactory rates of convergence over the entire range of error frequencies.

In p-multigrid the coarse problems are built based on lower-order discretizations with respect to the original problem of degree k. We consider L coarse levels denoted by the index $\ell = 0, ..., L$ and indicate the fine and coarse levels with $\ell = 0$ and $\ell = L$, respectively. The polynomial degree of level ℓ is k_ℓ and the polynomial degrees of the coarse levels are chosen such that $k_\ell < k_{\ell-1}$, with $k_0 = k$. These orders are used to build coarser linear systems $\mathcal{K}_\ell \mathbf{x}_\ell = \mathbf{b}_\ell$. In order to avoid additional integration of the residuals and Jacobians on the coarse level, we employ subspace inheritance of the matrix operators assembled on the finest space to build the coarse space operators $\mathcal{K}_i$ for both DG and HDG discretizations. This choice involves projections of the matrix operators and right hand sides, which are computed only once on the finest level. Compared to subspace non-inheritance, which requires the re-evaluation of the Jacobians in proper coarser-space discretizations of the problem, inheritance is cheaper in processing and memory. Although previous work has shown lower convergence rates when using such cheaper operators [32, 11, 12], especially in the context of elliptic problems and incompressible flows, we found these operators sufficiently efficient for our target problems involving the compressible NS equations, as will be demonstrated in the results section.

In the context of standard discontinuous Galerkin discretizations, see for example [9, 6, 10, 11, 12], the p-multigrid approach has been thoroughly investigated and exploited in several ways, *e.g.* h-, p-, and hp-strategies. On the other

Algorithm 1 MG_{full}

for $\ell = L, 0, -1$ **do**
  if $\ell = L$ **then**
    $\mathbf{b}_\ell = \mathbf{I}_0^\ell \mathbf{b}_0$
    SOLVE $\mathbf{A}_\ell \mathbf{x}_\ell^{FMG} = \mathbf{b}_\ell$
  else
    $\mathbf{b}_\ell = \mathbf{I}_0^\ell \mathbf{b}_0$
    $\tilde{\mathbf{x}}_\ell = \mathbf{I}_{\ell+1}^\ell \mathbf{x}_{\ell+1}^{FMG}$
    $\mathbf{x}_\ell^{FMG} = MG_V(\ell, \mathbf{b}_\ell, \tilde{\mathbf{x}}_\ell)$
  end if
end for
return $\mathbf{x}_\ell^{FMG}$

Algorithm 2 $MG_{\mathcal{V}}(\ell, \mathbf{b}_\ell, \mathbf{x}_\ell)$

if $\ell = L$ **then**
  SOLVE $\mathbf{A}_\ell \overline{\mathbf{x}}_\ell = \mathbf{b}_\ell$
else
  $\overline{\mathbf{x}}_\ell$=SMOOTH$(\mathbf{x}_\ell, \mathbf{A}_\ell, \mathbf{b}_\ell)$
  $\mathbf{r}_\ell = \mathbf{b}_\ell - \mathbf{A}_\ell \overline{\mathbf{x}}_\ell$
  $\mathbf{r}_{\ell+1} = \mathbf{I}_\ell^{\ell+1} \mathbf{r}_\ell$
  $\mathbf{e}_{\ell+1}$=$MG_V(\ell + 1, \mathbf{r}_{\ell+1}, \mathbf{0})$
  $\hat{\mathbf{x}}_\ell = \overline{\mathbf{x}}_\ell + \mathbf{I}_{\ell+1}^\ell \mathbf{e}_{\ell+1}$
  $\overline{\mathbf{x}}_\ell$=SMOOTH$(\hat{\mathbf{x}}_\ell, \mathbf{A}_\ell, \mathbf{b}_\ell)$
end if
return $\overline{\mathbf{x}}_\ell$

hand, the use of p-multigrid for HDG has not been as widely studied. See, for example, preliminary works [16, 24, 25] related to this research area. The definition of the restriction and prolongation operators, as well as the coarse grid operators and right hand sides, is not straightforward when considering the statically condensed system. We will therefore first introduce the concept of subspace inheritance for a standard DG solver and then extend it to HDG.

We employ a full multigrid (FMG) $\mathcal{V}$-cycle solver, outlined in Algorithm 1. The FMG cycle constructs a good initial guess for a $\mathcal{V}$-cycle iteration, which starts on the fine space. To do so, the solution is initially obtained on the coarsest level (L), and then prolongated to the next refined one ($L-1$). At this point, a standard $\mathcal{V}$-cycle is called, such that an improved approximation of the solution can be used for the $\mathcal{V}$-cycle at level $L-2$. This procedure is repeated until the $\mathcal{V}$-cycle on the finest level is completed. The single $\mathcal{V}$-cycle is outlined in Algorithm 2. Starting from a level ℓ, the solution is initially smoothed using an iterative solver (SMOOTH). The residual of the solution, $\mathbf{r}_\ell$, is then computed and projected to the coarser level $\ell + 1$, where another $\mathcal{V}$-cycle is recursively called to obtain a coarse-grid correction $\mathbf{e}_{\ell+1}$. This quantity is prolongated to level ℓ and used to correct the solution to be smoothed again. When the coarsest level is reached, the problem is solved with a larger number of iterations to decrease as much as possible the solution error at a low computational cost.

In this work the smoothers consist of preconditioned GMRES solvers. Any combination of single grid preconditioners can be coupled with an iterative solver to properly operate as a smoother in a multigrid strategy. Devising a methodology to optimally set the multigrid preconditioner is beyond the scope of the present work. However, previous work [12] has shown that an optimal and scalable solver can be obtained using an aggressive preconditioner on the coarsest space discretization, where the factorization of the matrix can be performed at a low computational cost, and the system has to be solved with a higher accuracy. On the other hand, cheaper operators can be used on the finest levels of the discretization, where the systems need not be solved to a high degree of accuracy. In the following

subsections, the details are given about how the matrices and vectors are restricted and prolongated between multigrid levels.

6.1. DG subspace inheritance

Let us define a sequence of approximation spaces $\mathcal{V}_\ell \subseteq \mathcal{V}_h$ on the same triangulation $\mathcal{T}_h$, ℓ being a multigrid level, with $\mathcal{V}_h = \mathcal{V}_0 \supset \mathcal{V}_1 \supset ... \supset \mathcal{V}_L$ and L the number of coarse levels. Note that $\mathcal{V}_L$ denotes the coarsest space. In our p-multigrid setting, each approximation space is defined similarly to (4), but using k_ℓ polynomials, with $k_0 > ... > k_\ell > ... > k_L$.

In this context, the prolongation operator can be defined as $\mathcal{I}_{\ell+1}^{\ell} : \mathcal{V}_{\ell+1} \to \mathcal{V}_\ell$ such that

$$\sum_{K\in\mathcal{T}_h} \int_K \left(\mathcal{I}_{\ell+1}^{\ell} u_{\ell+1} - u_{\ell+1}\right) dK = 0, \qquad \forall u_{\ell+1} \in \mathcal{V}_{\ell+1}. \tag{37}$$

Similarly, the restriction operator can be defined as the L_2 projection $\mathcal{I}_{\ell}^{\ell+1} : \mathcal{V}_\ell \to \mathcal{V}_{\ell+1}$ such that

$$\sum_{K\in\mathcal{T}_h} \int_K \left(\mathcal{I}_{\ell}^{\ell+1} u_\ell - u_\ell\right) v_{\ell+1} dK = 0, \qquad \forall (u_\ell, v_{\ell+1}) \in \mathcal{V}_\ell \times \mathcal{V}_{\ell+1}. \tag{38}$$

Such a definition can be extended to operate on vector functions $\mathbf{u}_h \in [\mathcal{V}_\ell]^m$ component-wise, *i.e.* $\mathcal{I}_{\ell}^{\ell+1}\mathbf{u}_h = \left\{\mathcal{I}_{\ell}^{\ell+1} u_i\right\}$.

Regarding coarse-space matrices, discrete matrix operators $\mathbf{I}_{\ell}^{\ell+1} \in \mathbb{R}^{p\times q}$, with $p = n_e n_{\mathrm{v}}^{\ell} m$, $q = n_e n_{\mathrm{v}}^{\ell+1} m$ and n_{v}^{ℓ} the number of DoFs on level ℓ, have to be considered to inherit the fine-space iteration matrix $\mathcal{K}_0$. This matrix can be restricted up to level ℓ via $\mathcal{K}_\ell = (\mathbf{I}_0^\ell)\mathcal{K}_0(\mathbf{I}_\ell^0)$. Fortunately, the explicit assembly of the operators can be avoided and the projection can be applied for each matrix block b of size $(n_{\mathrm{v}}^{\ell} m)^2$, which assumes the following form

$$\mathcal{K}_{\ell+1}^{b} = \mathbf{M}_{\ell+1,\ell} \mathcal{K}_\ell^b \left(\mathbf{M}_{\ell+1,\ell}\right)^T, \tag{39}$$

where

$$(\mathbf{M}_{\ell+1,\ell}) = (\mathbf{M}_{\ell+1})^{-1} \int_K \boldsymbol{\phi}^{\ell+1} \otimes \boldsymbol{\phi}^{\ell} \, \mathrm{d}K, \qquad \mathbf{M}_{\ell+1} = \int_K \boldsymbol{\phi}^{\ell+1} \otimes \boldsymbol{\phi}^{\ell+1} \, \mathrm{d}K. \tag{40}$$

Here, $\boldsymbol{\phi}^{\ell}$ denotes the set of basis functions defined in element K of order k_ℓ. We point out that, thanks to the use of basis functions defined in a reference element, the projection operators are identical for each element and pair of polynomial orders, and they can be used in the same way for the on-diagonal and off-diagonal blocks of the iteration matrix. As the prolongation operators can be obtained from the restriction by the transpose, $\mathbf{I}_{\ell+1}^{\ell} = \left(\mathbf{I}_{\ell}^{\ell+1}\right)^T$, the method requires the allocation of only L matrices of size $n_v^{\ell+1} \times n_v^{\ell}$ for each different type of element, which is inexpensive from a memory footprint viewpoint.

6.2. HDG subspace approximate-inheritance

In HDG, the globally-coupled unknowns are those related to the face DoFs, and the iteration matrix is obtained through static condensation, see Equation (35), which allows us to solve the system for the face unknowns only. Theoretically speaking, for element-interior degrees of freedom, the same operators of Section 6.1 can be employed,

while those for faces degrees of freedom can be obtained through similar considerations. In this case, the sequence of approximation spaces $\mathcal{M}_\ell \subseteq \mathcal{M}_h$ is properly defined on the interior mesh element faces, $\mathcal{F}_h$, with ℓ a multigrid level. To this end, we define $\mathcal{M}_h = \mathcal{M}_0 \supset \mathcal{M}_1 \supset ... \supset \mathcal{M}_L$. The prolongation operator is now $\mathcal{J}^\ell_{\ell+1} : \mathcal{M}_{\ell+1} \to \mathcal{M}_\ell$, defined by

$$\sum_{F\in\mathcal{F}_h} \int_F \left(\mathcal{J}^\ell_{\ell+1}\lambda_{\ell+1} - \lambda_{\ell+1}\right) \mathrm{d}\sigma = 0, \qquad \forall \lambda_{\ell+1} \in \mathcal{M}_{\ell+1}. \tag{41}$$

The restriction operator is defined as $\mathcal{J}^{\ell+1}_\ell : \mathcal{M}_\ell \to \mathcal{M}_{\ell+1}$ such that

$$\sum_{F\in\mathcal{F}_h} \int_F \left(\mathcal{J}^{\ell+1}_\ell \lambda_\ell - \lambda_\ell\right) \mu_{\ell+1} \, \mathrm{d}\sigma = 0, \qquad \forall (\lambda_\ell, \mu_{\ell+1}) \in \mathcal{M}_\ell \times \mathcal{M}_{\ell+1}. \tag{42}$$

These definitions can also be extended to operate on vector functions $\boldsymbol{\lambda}_h \in [\mathcal{M}_\ell]^M$ and are assumed to act component-wise.

Applying the same subspace-inheritance idea used for DG, one obtains the coarse space condensed HDG matrix and right hand side through the application of element-interior and face DoF projections,

$$\mathcal{K}_\ell = \left((\mathbf{J}^\ell_0)\mathbf{D}_0(\mathbf{J}^0_\ell) - \left[(\mathbf{J}^\ell_0)\mathbf{C}_0(\mathbf{I}^0_\ell)\right]\left[(\mathbf{I}^\ell_0)\mathbf{A}_0(\mathbf{I}^0_\ell)\right]^{-1}\left[(\mathbf{I}^\ell_0)\mathbf{B}_0(\mathbf{J}^0_\ell)\right]\right), \tag{43}$$

$$\mathbf{b}_\ell = \left((\mathbf{J}^\ell_0)\mathbf{g}_0 - \left[(\mathbf{J}^\ell_0)\mathbf{C}_0(\mathbf{I}^0_\ell)\right]\left[(\mathbf{I}^\ell_0)\mathbf{A}_0(\mathbf{I}^0_\ell)\right]^{-1}\left[(\mathbf{I}^\ell_0)\mathbf{f}_0\right]\right). \tag{44}$$

We point out that this operation involves the application of mixed element-interior and face degrees of freedom Galerkin projections prior to the static condensation of the system. Thus, it comes with an increased operation count compared to DG subspace inheritance. To minimize the number of operations involved in the projection, we introduce an *approximate*-inherited approach where the coarse space matrices and right-hand sides are obtained by simply applying face projections to the condensed matrices and vectors on the finest space. In other words, we obtain $\mathcal{K}_\ell$ and $\mathbf{b}_\ell$ through

$$\mathcal{K}_\ell = (\mathbf{J}^\ell_0)\mathcal{K}_0(\mathbf{J}^0_\ell), \tag{45}$$

$$\mathbf{b}_\ell = (\mathbf{J}^\ell_0)\mathbf{b}_0. \tag{46}$$

This coarse space matrix and right-hand side will in general differ from those of Equation (43) and (44), as the element-interior operator in the Schur complement term is projected differently.

Similar considerations have to be made for the residual evaluation on the coarse levels, which are required in the projection from level ℓ to $\ell + 1$. The residual on a level ℓ should be computed as

$$\mathbf{r}_\ell = \left(\mathbf{R}^\wedge_\ell - \left[(\mathbf{J}^\ell_0)\mathbf{C}_0(\mathbf{I}^0_\ell)\right]\left[(\mathbf{I}^\ell_0)\mathbf{A}^{QQ}_0(\mathbf{I}^0_\ell)\right]^{-1}\mathbf{R}^{QU}_\ell,\right) \tag{47}$$

and this requires the evaluation of the element-interior degrees of freedom from the face unknowns. This operation would require a back-solve on the coarse levels, which increases the amount of operations within each multigrid cycle.

To reduce the operation count as much as possible we again rely on the computation of an approximate projection that is evaluated using the working variable $\Delta\mathbf{\Lambda}$ only:

$$\mathbf{r}_\ell = \mathcal{K}_\ell \Delta\mathbf{\Lambda}_\ell + \mathbf{b}_\ell. \tag{48}$$

Despite the use of those approximations compared to DG subspace inheritance, such a strategy exhibits very good performance in HDG solutions of the compressible Navier–Stokes equations, as will be demonstrated in the results section.

Similarly to DG, the global assembly of the projection matrix $\mathbf{J}_0^\ell$ for face degrees of freedom is not strictly required for HDG, since the projection is applied to each block b of the matrix of size $(n_\mathrm{v}^\ell m)^2$, with n_v^ℓ related to face degrees of freedom on level ℓ. The projection of each block b of the iteration matrix assumes the form

$$\mathcal{K}_{\ell+1}^b = \mathbf{N}_{\ell+1,\ell} \mathcal{K}_\ell^b \left(\mathbf{N}_{\ell+1,\ell}\right)^T, \tag{49}$$

where

$$\left(\mathbf{N}_{\ell+1,\ell}\right) = \left(\mathbf{N}_{\ell+1}\right)^{-1} \int_F \boldsymbol{\mu}^{\ell+1} \otimes \boldsymbol{\mu}^\ell \, \mathrm{d}\sigma, \qquad \mathbf{N}_{\ell+1} = \int_F \boldsymbol{\mu}^{\ell+1} \otimes \boldsymbol{\mu}^{\ell+1} \, \mathrm{d}\sigma. \tag{50}$$

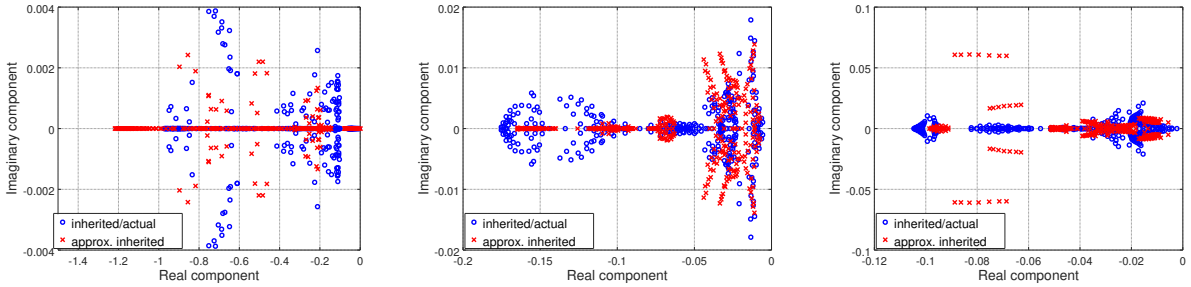
In this case $\boldsymbol{\mu}^\ell$ denotes the set of basis functions defined in the element face F of order k_ℓ. Thanks to the use of basis functions defined in the reference element of the space discretization, similar considerations to those related to element-interior degrees of freedom projection hold true.

To assess the impact of the approximate inheritance approximation on the HDG coarse-space condensed matrix, $\mathcal{K}$, we conduct an eigenvalue analysis of the inherited and approximate-inherited operators for a simplified test case. The problem of interest is scalar advection-diffusion in a unit square domain ($L = 1$), with advective velocity $\vec{v} = (0.8, 0.6)$, diffusivity $\nu = |\vec{v}|L/Pe$, unit Dirichlet boundary condition on the bottom boundary, and homogenous zero Dirichlet boundary conditions on all other boundaries. Note, Pe is the P'ecl'et number, defined by $Pe = |\vec{v}|L/\nu$.

The computational grid consists of uniform NxN squares, each subdivided into two triangles, for a total of $2N^2$ elements. We use $N = 8$, a fine-space order of $p_0 = 2$, and a coarse-space order of $p_1 = 1$. For various Pe, we compute both the inherited condensed matrix, $\mathcal{K}_1^{\text{inherited}}$ via (43), and the approximate-inherited condensed matrix, $\mathcal{K}_1^{\text{approx}}$ via (45). Note that for this linear problem, the inherited matrix is equivalent to the condensed matrix computed directly on the coarse space. Figure 2 compares the eigenvalue spectra of the condensed matrices for three different P'ecl'et numbers. The eigenvalues are not identical, but they are located in generally the same areas and have similar real and imaginary components. Iterative solution properties of the matrices are therefore expected to be similar.

6.3. Preconditioning options and memory footprint

In this work we exploit single-grid preconditioners both for benchmarking and to precondition the smoothers of the multigrid strategy. Two operators will be considered in this work. The first and simplest one will be here labelled as element-wise block-Jacobi (BJ). The BJ preconditioner extracts the block-diagonal portion of the iteration matrix and factorizes it, in a local-to-each element fashion, using the PLU factorization. This cheap and memory saving

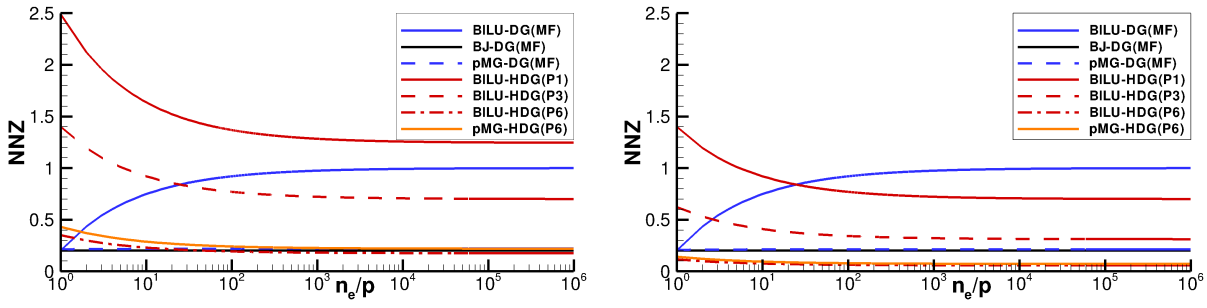


(a) $Pe = 10$ (b) $Pe = 100$ (c) $Pe = 500$

Figure 2: Eigenvalues of the inherited and approximate-inherited condensed HDG matrices, $\mathcal{K}$, on the coarse $p = 1$ space for a scalar advection-diffusion test case.

preconditioner becomes more effective as the time step of the discretization decreases, *i.e.* the matrix becomes more diagonally dominant and the condition number decreases. In addition, the BJ preconditioner is applied in the same way in serial and parallel computations.

The second preconditioner is the incomplete lower-upper factorization with zero-fill, ILU(0). In particular, the minimum discarded fill reordering proposed in [5] is employed. The algorithm was shown to be suited for stiff spatial discretizations, and it allows for an in-place factorization [6]. When it is applied in parallel, the ILU(0) is performed separately on each square, partition-wise block of the iteration matrix. In this case this preconditioner will be labelled as block-ILU(0) (BILU). As a natural downside, BILU loses preconditioning efficiency when it is applied in parallel. A variant that compensates for this effect is the Additive Schwarz method, which extends the partition-wise block of the Jacobian with a number of overlapping elements between the mesh partitions. This algorithm, employed in the context of incompressible Navier–Stokes equations, increases the memory footprint of the solver when a few elements per partition are used [12]. We do not use such an approach in the present study of compressible flow.



(a) Full-order basis functions (b) Tensor-product basis functions

Figure 3: Allocated number of non-zeros (NNZ) non-dimensionalized by the memory allocation of the DG Jacobian matrix. DG matrix-free solvers compared to HDG for different values of polynomial orders and preconditioning type. p-multigrid preconditioning (pMG) is assumed to be that of Table 3.

We now provide estimates of the memory footprint of the solver as well as the computational time spent evaluating the matrix operators. It is worth pointing out that, for DG, the matrix assembly time, as well as its operation count, is a function of the number of non-zeros of the matrix itself, while HDG involves the overhead costs of the static condensation and back solve. Considering a square, two-dimensional, bi-periodic domain made of quadrilateral elements, we obtain the results shown in Figure 3, where the number of non-zeros (NNZ), non-dimensionalized by the number of nonzeros of the Jacobian arising from the DG discretization, is reported as a function of the number of elements per partition, n_e/p. It can be observed that:

1. The memory footprint of DG, matrix-based as well as HDG solvers is always equal to that of a Jacobian matrix, and this value is a function of the polynomial order for HDG. This is due to the fact that the preconditioner is always evaluated using the same memory as the Jacobian matrix.
2. For a matrix-free, DG discretization, the allocation involves only the preconditioner operator. When BJ is considered, NNZ reduces by 80% with respect to the allocation of a full DG Jacobian. As $n_e/p \to 1$, the NNZ of the BILU solver approaches that of the element-wise block Jacobi, while for $n_e/p >> 1$ it tends to be that of a Jacobian matrix. This is due to the fact that as the domain is partitioned, the ILU(0) factorization is performed in the squared, partition-wise block of the iteration matrix and therefore, in a matrix-free fashion, the off-partition blocks can be neglected during the assembly phase.
3. The p-multigrid (pMG) matrix-free preconditioning approach applied to a DG discretization, here assumed to be that of Table 3, requires a memory footprint in line with that of an element-wise block Jacobi method, as already observed in [12]. In fact, when using lower-order polynomial spaces with $k_\ell << k$, the size of those matrices is considerably smaller than that of the finest space since they scale with k^{2d}.
4. For HDG, only for high-order polynomials is NNZ reduced with respect to the iteration matrix of a DG method. For $k = 6$, a memory footprint in line with that of a BJ, matrix-free approach is observed, while for $k = 1, 3$ the memory is considerably larger. It is worth pointing out that the use of full-order basis functions (Figure 3(a)) and tensor-product basis functions (Figure 3(b)) only affects the NNZ ratio in the HDG case: in particular, the memory footprint reduction when using a tensor-product basis is larger due to the higher amount of element-wise unknowns compared to internal face unknowns.
5. The NNZ ratio for HDG increases when $n_e/p \to 1$. The reason for this is that the face-to-element ratio within the computational mesh of the domain partition increases.

Finally, it is important to remark that for a matrix-free iterative solver employed in DG contexts, it is possible to optimize the matrix assembly evaluation to compute only the blocks required by the preconditioner. For example, for BJ, the evaluation of the off-diagonal blocks of the Jacobian can be neglected, while for pMG matrix-free with BJ on the finest space, the off-diagonal blocks could be computed at a reduced polynomial order consistent with that of the coarser spaces.

7. Numerical results on a model test case

We present numerical experiments to assess the performance of the HDG discretizations in comparison to DG. First, Navier–Stokes solutions of a vortex transported by uniform flow at $M = 0.05$ and $Re = 100$ are reported. The objective is i) to show the convergence rates of the solver both in space and time; ii) to investigate the effects of grid refinement for the approximate-inherited approach proposed for HDG, providing mesh-independent convergence rates; and iii) compare the effects of polynomial order, time step size and space discretization on the parallel performance of the solution strategy.

7.1. Test case description

The test case is a modified version of the VI1 case studied in the 5$^{\text{th}}$ International Workshop on High Order CFD Methods [33], and it consists of a two-dimensional mesh of the domain $(x, y) \in [0, 0.1]\times[0, 0.1]$ with periodic boundary conditions on each side. The flow initialization involves the definition of the following state

$$\begin{aligned} u &= U_\infty \left(1 - \beta\left(\frac{y - Y_c}{R}\right) e^{-r^2/2}\right) \\ v &= U_\infty \beta \left(\frac{x - X_c}{R}\right) e^{-r^2/2} \\ T &= T_\infty - \left(\frac{U_\infty^2 \beta^2}{2C_p}\right) e^{-r^2} \end{aligned} \tag{51}$$

with the heat capacity at constant pressure $C_p = R_{\text{gas}}\gamma/(\gamma - 1)$, the non dimensional distance to the initial vortex core position $r = \sqrt{(x - X_c)^2 + (y - Y_c)^2}/R$, X_c, Y_c the coordinates of the vortex center, and the free stream velocity $U_\infty = M_\infty \sqrt{\gamma R_{\text{gas}} T_\infty}$. The fluid pressure p, temperature T, and density ρ are prescribed to ensure a steady solution of the problem in the freestream co-moving frame, *i.e.*$\rho_\infty = p_\infty/R_{\text{gas}}T_\infty$, $\rho = \rho_\infty(T/T_\infty)^{1/(\gamma-1)}$, $p = \rho R_{\text{gas}} T$. The parameters where chosen such that $M_\infty = 0.05, \beta = 1/50$ and $R = 0.005$. In contrast to the inviscid-flow case studied in the workshop, the governing equations in the present study are Navier-Stokes, with $Re = 100$ based on the domain size.

7.2. Assessment of the solution accuracy

Numerical experiments have been performed to assess the output error, both in space and time. The meshes for the study consist of regular quadrilaterals. The mesh density ranges from 2×2 to 64×64, while the polynomial order range is $k \in \{1, 2, 3, 4, 5, 6\}$. The L_2 state error was computed relative to the solution on a 128×128, $\mathbb{P}_6$ space discretization, after one convective period, T. Contour plots of the solution at the initial and final states are shown in Figure 4. Figure 5(a) reports space discretization errors. The tests were performed using a very small time step size, $T/\Delta t = 4000$, and the ESDIRK3 scheme to ensure a negligible time discretization error, with an absolute tolerance on the non-linear system of 10^{-10}, and a relative tolerance of 10^{-5} on GMRES. Even though DG suffers less than HDG from pre-asympthotic behaviour on such a smooth solution, all three implementations show comparable error levels

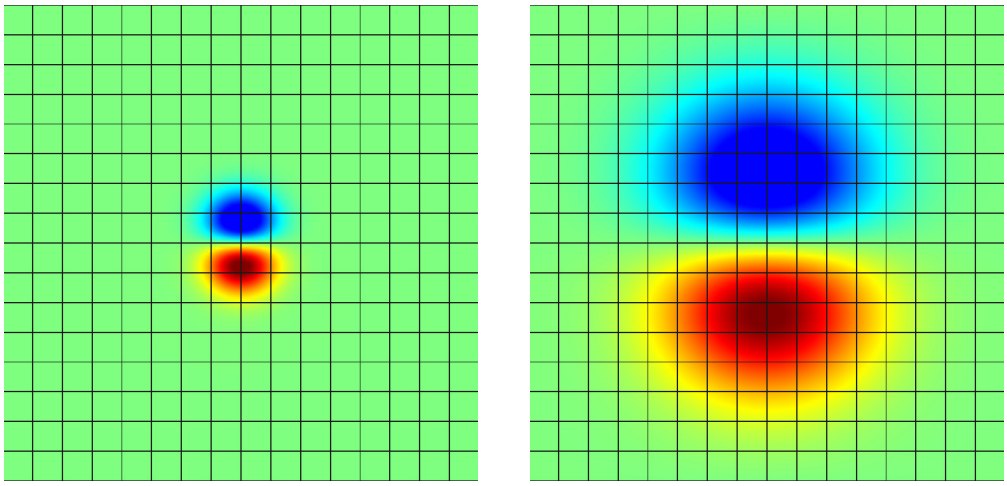
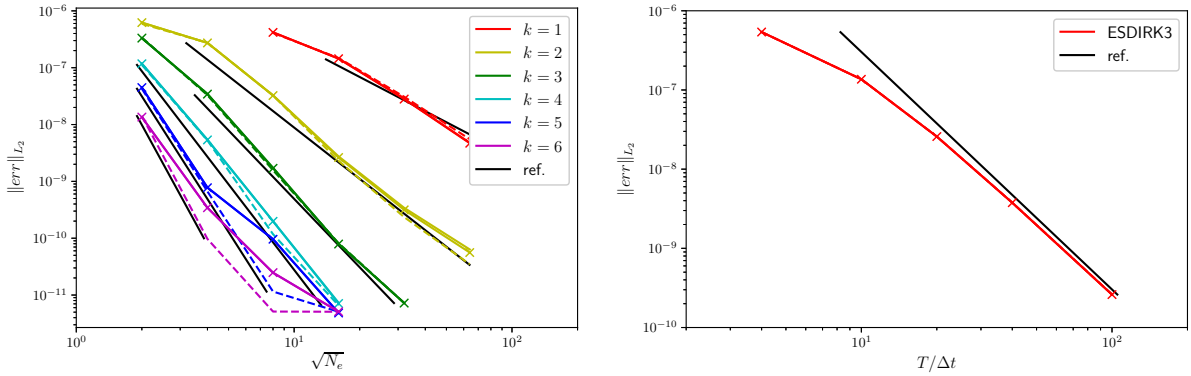


Figure 4: Convected vortex at $Re = 100$, $M = 0.05$. Mach number contours. Solution at $t = 0$ (left) and $t = T$ (right).



(a) Space

(b) Time

Figure 5: L_2 solution error. Laminar vortex test case at $Re = 100$, $M = 0.05$. Convergence rates for the DG (dashed), *mHDG* (solid) and *p*HDG (crosses) discretizations versus theoretical estimates in space and time.

and converge with the theoretical convergence rates for every polynomial approximation shown. As a consequence of such analysis, and considering that both the DG and HDG implementations share the same code base, we will consider only the CPU time as a measure of the time-to-solution efficiency.

Regarding the time integration scheme, the convergence rates of ESDIRK3 are also reported in Figure 5(b), and these were obtained using the 16×16 grid, with $\mathbb{P}_6$ polynomials for the three space discretization strategies. The theoretical third-order convergence rate of the ESDIRK3 time integration scheme can be observed for all three space discretizations with comparable error levels.

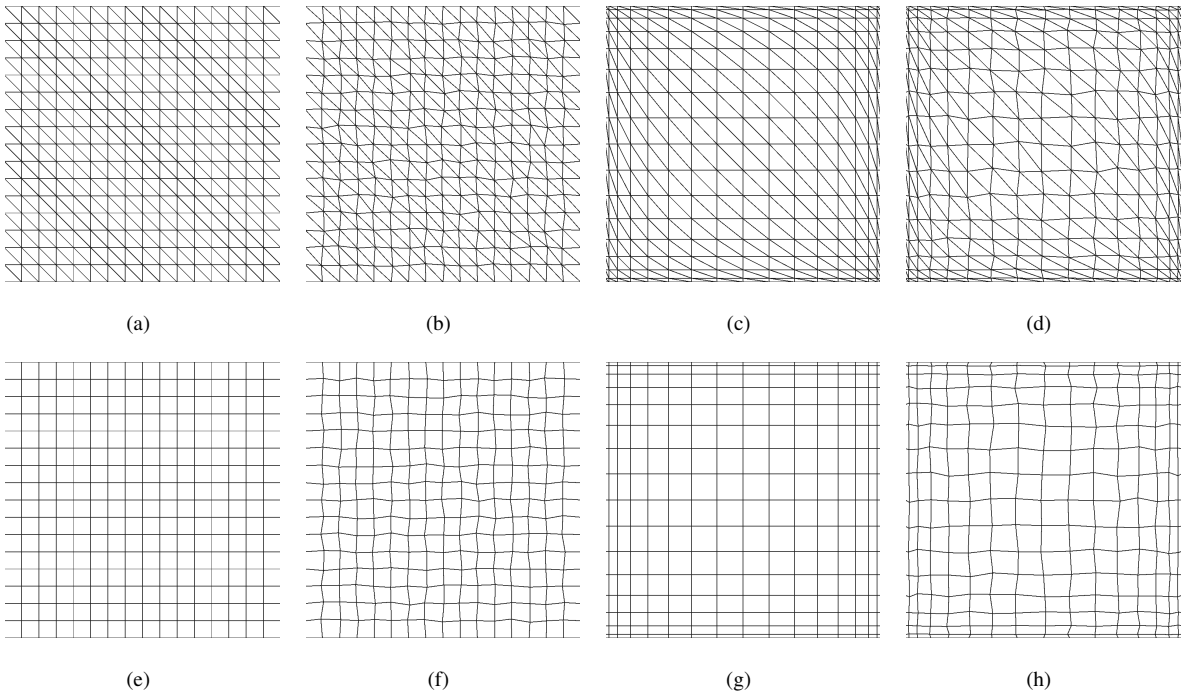


Figure 6: Convected vortex at $Re = 100$, $M = 0.05$. Examples of triangular and quadrilateral meshes involving i) regular elements; ii) randomly distorted elements; iii) regular and clustered elements; and iv) clustered-distorted elements.

7.3. Assessment of the approximate-inherited multigrid approach for HDG

To demonstrate the efficacy of the multigrid approach proposed herein for HDG, we report numerical experiments obtained by reducing the mesh element size for different element types. Four mesh sequences are considered in the study, obtained as i) regular elements; ii) randomly distorted elements; iii) regular and clustered elements; and iv) clustered-distorted elements. The study was performed on both triangular and quadrilateral elements, see Figure 6. We remark that the random mesh perturbation was limited to 10% of the minimum dimension of the element, and that the clustering has been obtained by placing the mesh element nodes using Gauss-Lobatto rules. A full multigrid strategy has been employed for the study. The strategy combines three multigrid levels, and for each of them a BILU-preconditioned GMRES smoother is considered. To avoid the impact of domain decomposition, all computations are performed in serial. Three smoothing iterations were performed on each level, while 400 iterations were employed on the coarsest level to ensure that the results are not polluted by a lack of coarse-level resolution. This configuration was found to be optimal for the present serial computations, and it is consistent with previous work in the literature, see [12].

Table 2 report the results on triangular and quadrilateral mesh elements. The numerical experiment consists of one time period such that $T/\Delta t = 10$ using the ESDIRK3 scheme, with T the convective period of the problem. The average number of iterations as well as the average convergence rate (CR) ρ are reported. The CR is defined as $\rho = (r_{IT}/r_0)^{1/IT}$ with IT the average number of GMRES iterations, r_0 and r_{IT} the residuals at the first and IT^{th} iteration

	Reg Tri		Dis Tri		Reg Grad Tri		Dis Grad Tri	
n_e	IT_a	ρ_a	IT_a	ρ_a	IT_a	ρ_a	IT_a	ρ_a
$16^2 \cdot 2$	3.000	0.0060	2.833	0.0056	3.000	0.0120	3.000	0.0131
$32^2 \cdot 2$	3.000	0.0123	3.000	0.0119	3.500	0.0245	3.333	0.0195
$64^2 \cdot 2$	2.500	0.0058	3.000	0.0132	3.667	0.0315	4.000	0.0422
$128^2 \cdot 2$	2.333	0.0047	2.667	0.0086	3.500	0.0290	4.167	0.0560
	Reg Quad		Dis Quad		Reg Grad Quad		Dis Grad Quad	
n_e	IT_a	ρ_a	IT_a	ρ_a	IT_a	ρ_a	IT_a	ρ_a
16^2	2.833	0.0053	2.333	0.0033	3.000	0.0069	3.000	0.0057
32^2	2.833	0.0101	2.833	0.0081	3.000	0.0087	3.000	0.0096
64^2	3.167	0.0183	3.000	0.0153	3.833	0.0386	3.667	0.0336
128^2	3.333	0.0246	3.500	0.0273	4.333	0.0637	4.500	0.0659

Table 2: Laminar vortex test case at $Re = 100$, $M = 0.05$. h-independence test on tri-element (top) and quad-element (bottom) meshes. of a FGMRES(MG_{full}) solver built on three levels. GMRES(BILU) smoothers with 400 iterations on the coarsest level $\ell = 2$. 3 smoothing iterations were employed for $\ell = \{0, 1\}$.

respectively. In all of the numerical experiments, the p-multigrid strategy appears to work optimally as the number of iterations only slightly grows with the number of mesh elements, even for distorted and graded mesh sequences.

7.4. Evaluation of the solver efficiency

We report results of numerical experiments devoted to assess the performance of the p-multigrid preconditioning strategy for HDG in comparison to other operators, as well as state-of-the-art preconditioned DG discretizations. In particular, we take as a reference the matrix-based, BILU preconditioned DG discretization and we aim at reporting insights on the computational time of the solution, in order to provide an overall idea of the computational efficiency of the solver. The space discretization relies on the 16×16 mesh made by regular quadrilaterals and two polynomial orders, *i.e.* $k = \{3; 6\}$. As for the time discretization, non dimensional time steps of $\Delta t = \{1; 0.1\}$ are employed. In both cases, a single time step is computed, which corresponds to three non-linear problems. We observe that our Newton solver converges in two iterations, which means that the CPU time and the average number of iterations are evaluated considering a total of six linear system solutions. Each linear system is solved up to a non-preconditioned relative linear tolerance of 10^{-5}, while an absolute tolerance of 10^{-10} was used for the nonlinear solver.

Special attention is hereby given to the parallel efficiency of the computations, both in terms of CPU time and average number of GMRES iterations. To this extent, the ideas reported in [12] on the choice of the number of levels and the smoothing type have been proposed. A very similar behaviour of the solver has been presently observed and thus we explicitly refer to that work for a more in-depth discussion about the effects of the smoothing. We here report only the general idea: a scalable multigrid strategy to precondition linear systems arising from the Navier–Stokes

Level	Order	Solver	Preconditioner	Iterations
1	6	GMRES	BJ	10
2	2	GMRES	BJ	10
3	1	GMRES	BILU	30

Table 3: Computational settings for the p-multigrid smoothers employed within the paper.

equations can be obtained by the use of simple BJ-preconditioned GMRES smoothers for all of the levels except the coarse one, where more powerful preconditioners can be cheaply introduced on the smoothers due to the low computational cost of the coarse matrix factorization.

In the numerical experiments we rely on a three-level multigrid strategy. In both the cases, $k_{1,2} = \{2, 1\}$ have been used on the coarser spaces smoothed with BJ- and BILU-preconditioned GMRES solvers, respectively. $\{10, 30\}$ smoothing iterations have been employed, which were found to be sufficient to provide optimal results both in serial and in parallel computations for HDG and DG as well. On the finest space, 10 smoothing iterations of GMRES(BJ) were used. The computational settings are summarized in Table 3. We remark that these settings have been found to be sufficiently computationally efficient for all of the numerical experiments reported in this paper.

Table 4 compares the performance of the multigrid preconditioner to those of BILU for $k = 3$ using $\Delta t = 1$ (left) and $\Delta t = 0.1$ (right), for three space discretizations, *i.e.* DG, mHDG and pHDG. The table is designed in such a way that the merits of the discretization are presented along vertical blocks, while the effects of the preconditioner and time step size are presented along horizontal blocks. For all three space discretization, the performance of the BILU preconditioner degrades in view of the considerable increase in the number of GMRES iterations moving from serial to parallel runs. On the other hand, the multigrid preconditioning strategy shows higher parallel efficiencies in all cases, with the increase in number of iterations either very low (for the largest time step) or non-existent (for the smallest time step). In fact, with the exception of the coarsest space solution, the algorithm is not affected by the domain decomposition.

Regarding the CPU time, we report the speed-up values non-dimensionalized by the CPU time of the DG, matrix-based and BILU-preconditioned computation. For the largest time step size, switching from single-grid BILU to multigrid provides consistent benefits on all three space discretizations in view of a higher parallel efficiency of the approach. In fact, despite the speed-up factor being lower for serial computations, it peaks at 32 cores. The strategy provides a speedup of 2.09 for DG, 2.21 and 3.05 for mHDG and pHDG, respectively.

For the smaller time step, the right-hand side of Table 4, the situation is slightly different. In fact, the reduced conditioning of the matrix reduces the iterative solution times, as well as increases the relative cost of the matrix assembly. For mHDG, where the assembly costs are the highest, we observed speed-up values below one. Conversely, pHDG still outperforms the reference, providing a speed-up factor in the range $[1.29, 1.75]$ due to the smaller number of operations during the Jacobian assembly with respect to the *mixed* form. However, as opposed to what happens for DG, where an improvement in computational efficiency is still observed, the use of p-multigrid does not benefit the

Discr.	$k=3$, $\Delta t=1$						$k=3$, $\Delta t=0.1$					
Solver	BILU-DG			MG_{full}-DG			BILU-DG			MG_{full}-DG		
n_p	Time	E	IT_a	SU_{MB}	E	IT_a	Time	E	IT_a	SU_{MB}	E	IT_a
1	38.52		81.00	1.03		3.50	23.51		27.83	0.88		2.17
2	27.53	0.70	137.33	1.43	0.97	3.50	15.12	0.78	53.17	1.08	0.96	2.17
4	15.10	0.64	154.83	1.51	0.94	3.50	7.68	0.77	55.33	1.06	0.92	2.17
8	8.72	0.55	181.17	1.58	0.85	3.67	4.15	0.71	65.50	1.04	0.84	2.17
16	6.03	0.40	229.50	1.78	0.69	4.00	2.23	0.66	72.33	0.99	0.74	2.17
32	5.82	0.21	284.50	2.09	0.42	4.67	1.57	0.47	81.83	0.96	0.51	2.17
Solver	BILU-mHDG			MG_{full}-mHDG			BILU-mHDG			MG_{full}-mHDG		
n_p	SU_{MB}	E	IT_a	SU_{MB}	E	IT_a	SU_{MB}	E	IT_a	SU_{MB}	E	IT_a
1	1.49		45.17	1.23		2.50	0.96		20.33	0.79		2.00
2	1.75	0.82	77.17	1.54	0.88	2.50	1.03	0.83	36.00	0.88	0.87	2.00
4	1.63	0.70	95.50	1.50	0.78	2.50	0.91	0.72	41.00	0.80	0.78	2.00
8	1.67	0.62	113.17	1.56	0.70	2.50	0.90	0.66	46.83	0.78	0.70	2.00
16	1.76	0.47	138.67	1.75	0.57	2.83	0.79	0.54	52.67	0.68	0.56	2.00
32	1.93	0.27	183.50	2.21	0.37	3.50	0.81	0.39	64.50	0.70	0.42	2.00
Solver	BILU-pHDG			MG_{full}-pHDG			BILU-pHDG			MG_{full}-pHDG		
n_p	SU_{MB}	E	IT_a	SU_{MB}	E	IT_a	SU_{MB}	E	IT_a	SU_{MB}	E	IT_a
1	2.45		44.33	1.95		2.00	1.62		44.33	1.19		2.00
2	2.83	0.81	75.17	2.48	0.89	2.00	1.75	0.84	75.17	1.36	0.89	2.00
4	2.60	0.68	94.17	2.26	0.74	2.50	1.55	0.73	94.17	1.22	0.79	2.00
8	2.57	0.58	112.33	2.34	0.67	2.50	1.51	0.66	112.33	1.19	0.71	2.00
16	2.67	0.44	136.83	2.60	0.53	2.67	1.29	0.52	136.83	1.05	0.58	2.00
32	2.59	0.22	182.83	3.05	0.32	3.50	1.38	0.40	182.83	1.05	0.41	2.00

Table 4: Computational efficiency comparison using DG and HDG discretizations. Laminar vortex test case at $Re = 100$, $M = 0.05$, discretized using 16×16 mesh with $\mathbb{P}_3$ polynomials. Two time steps, $\Delta t = \{1; 0.1\}$ using the ESDIRK3 scheme are reported. SU_{MB} stands for the speed-up factor relative to the DG, matrix-based, BILU-preconditioned computation, E is the parallel efficiency and IT_a the average number of GMRES iterations.

performance of the solver.

Table 5 shows the same results for a $k = 6$ space discretization. Similar observations to those reported in Table 4 can be made on the overall parallel performance of the preconditioning strategies here considered, *i.e.* the increase in the number of iterations of the preconditioner reflects the performance degradation of the solution strategy when n_p increases. However, in this case, the advantages arising from the use of a multigrid preconditioning strategy are more evident. In fact, for the largest time step size, the speed-up values are higher than those reported in Table 4 involving 3^{rd} order polynomials. In particular, when using 32 cores, the speedup reaches 2.65, 1.87 and 3.84 for DG, mHDG and pHDG respectively. In contrast to what was observed previously, when reducing the time step size, the advantages of using a multigrid strategy still appear evident for DG, which provides speed-ups in the range $[1.86, 2.29]$. While for higher-order polynomials and smaller time steps a *mixed* HDG implementation provides performance in line with

Discr.	$k=6$, $\Delta t=1$						$k=6$, $\Delta t=0.1$					
Solver	BILU-DG			MG_{full}-DG			BILU-DG			MG_{full}-DG		
n_p	Time	E	IT_a	SU_{MB}	E	IT_a	Time	E	IT_a	SU_{MB}	E	IT_a
1	533.69		87.33	1.73		5.67	390.37		32.00	1.90		3.00
2	381.18	0.70	190.83	2.37	0.96	5.83	247.56	0.79	80.83	2.29	0.95	3.00
4	198.53	0.67	213.50	2.42	0.94	5.83	123.33	0.79	85.33	2.25	0.94	3.00
8	109.00	0.61	257.50	2.69	0.95	5.33	62.84	0.78	99.17	2.24	0.92	3.00
16	63.14	0.53	313.17	2.55	0.78	6.33	32.33	0.75	105.83	2.04	0.81	3.00
32	47.91	0.35	392.33	2.65	0.53	7.17	19.61	0.62	110.67	1.86	0.61	3.00
Solver	BILU-mHDG			MG_{full}-mHDG			BILU-mHDG			MG_{full}-mHDG		
n_p	SU_{MB}	E	IT_a	SU_{MB}	E	IT_a	SU_{MB}	E	IT_a	SU_{MB}	E	IT_a
1	1.46		46.50	1.45		2.67	1.08		23.17	1.07		2.17
2	1.75	0.84	112.50	1.76	0.85	2.67	1.16	0.85	50.67	1.16	0.85	2.17
4	1.58	0.73	139.50	1.61	0.75	2.67	1.01	0.74	54.50	1.01	0.74	2.17
8	1.58	0.66	155.83	1.61	0.68	2.83	0.94	0.68	62.17	0.93	0.68	2.17
16	1.52	0.55	203.83	1.59	0.58	3.67	0.83	0.58	71.83	0.83	0.58	2.17
32	1.74	0.42	268.50	1.87	0.45	5.17	0.80	0.46	82.50	0.80	0.46	2.17
Solver	BILU-pHDG			MG_{full}-pHDG			BILU-pHDG			MG_{full}-pHDG		
n_p	SU_{MB}	E	IT_a	SU_{MB}	E	IT_a	SU_{MB}	E	IT_a	SU_{MB}	E	IT_a
1	3.15		45.83	3.11		2.50	2.36		22.17	2.30		2.00
2	3.71	0.83	106.50	3.81	0.86	2.50	2.52	0.84	48.50	2.50	0.86	2.00
4	3.35	0.71	133.00	3.46	0.75	2.67	2.20	0.74	52.50	2.18	0.75	2.00
8	3.35	0.65	149.33	3.46	0.68	2.67	2.04	0.67	59.50	2.03	0.69	2.00
16	3.19	0.54	193.33	3.39	0.58	3.50	1.80	0.58	69.17	1.78	0.58	2.00
32	3.38	0.37	257.00	3.84	0.43	5.17	1.73	0.46	79.67	1.71	0.46	2.00

Table 5: Computational efficiency comparison using DG and HDG discretizations. Laminar vortex test case at $Re = 100$, $M = 0.05$, discretized using 16×16 mesh with $\mathbb{P}_6$ polynomials. Two time steps, $\Delta t = \{1; 0.1\}$ using ESDIRK3 are reported. SU_{MB} stands for the speed-up factor referred to the DG, matrix-based, BILU-preconditioned computation, E is the parallel efficiency and IT_a the average number of GMRES iterations.

that of a matrix-based, BILU-DG solver, the *primal* implementation exhibits better performance overall, even when it shows an overall lower parallel efficiency. In particular, BILU-pHDG is the best performing solution strategy, providing speed-up values in the range $[1.73, 2.36]$. In this case, p-multigrid performs similarly to BILU.

Table 6 reports for comparison the speed-up values obtained using a matrix-free implementation of the iterative solver for the DG space discretization. In particular, we compute the matrix-based speedup SU_{MB} using as a reference the matrix-based, BILU-DG solver to show the performance compared to the reference algorithm. The performance is evaluated using the same preconditioners reported in Tables 4 and 5. Considering the single-grid matrix-free, BILU-DG numerical experiments, one can summarize that:

1. For $k = 3$, switching MB to MF penalizes the CPU time. The reason for this is two-fold: first, the serial computation is 35% slower for MF than for MB. Second, when the solver is applied in parallel, the matrix-free

		$\Delta t = 1$				$\Delta t = 1/10$			
		BILU-DG		MG_{full}-DG		BILU-DG		MG_{full}-DG	
Case	n_p	MF	MFL	MF	MFL	MF	MFL	MF	MFL
$k = 3$	1	0.73	0.97	0.76	0.89	0.84	1.53	0.67	0.85
	2	0.68	0.80	1.02	1.21	0.74	1.09	0.81	1.04
	4	0.65	0.72	1.07	1.28	0.72	1.03	0.78	1.02
	8	0.61	0.73	1.13	1.32	0.68	0.91	0.80	1.03
	16	0.56	0.63	1.20	1.36	0.61	0.79	0.72	0.90
	32	0.59	0.62	1.41	1.59	0.58	0.71	0.69	0.86
$k = 6$	1	1.03	2.10	1.84	2.56	1.00	3.10	1.97	3.36
	2	0.99	1.47	2.42	3.18	0.99	2.03	2.33	3.98
	4	0.96	1.37	2.41	3.32	0.98	1.92	2.23	3.78
	8	0.93	1.25	2.69	3.78	0.95	1.75	2.18	3.65
	16	0.85	1.06	2.35	3.11	0.90	1.56	1.90	3.15
	32	0.78	0.98	2.37	3.03	0.86	1.41	1.70	2.77

Table 6: Computational efficiency of a matrix-free DG strategy. Laminar vortex test case at $Re = 100$, $M = 0.05$, discretized using 16×16 mesh with $\mathbb{P}_3$ and $\mathbb{P}_6$ polynomials. Two time steps, $\Delta t = \{1; 0.1\}$ using the ESDIRK3 scheme are reported. SU_{MB} stands for the speed-up factor referred to the matrix-based, BILU-DG computation reported in Tables 4 and 5, SU_{MF} is the speedup computed considering the matrix-free BILU-DG settings. Results obtained by lagging the preconditioner evaluation (MFL) for the entire solution process, *i.e.* six linear systems, are also reported.

implementation seems to be less parallel efficient, since the speed-up factor decreases with an increasing the number of processors.

2. For $k = 6$, the penalization decreases. In fact, in serial computation, the same computational time has been recovered, meaning that a matrix-free iteration performs similarly to a matrix-vector product. However, a slightly lower parallel efficiency is still observed.
3. When a small time step is employed, higher speed-up values are achieved for MF compared to MB. In other words, the lower the number of GMRES iterations, the higher the performance. A slightly higher parallel efficiency is also observed.

The possibility of lagging the preconditioner evaluation is also explored and the results are reported in Table 6 (MFL). In particular, we skip the recomputation of the preconditioner for six consecutive iterations, which means that the Jacobian matrix is evaluated only for the first non-linear iteration of the first stage of the ESDIRK3 scheme. By doing so, it is observed that the linear system converges with the same number of iterations, which are not reported for brevity. This shows, at least for this kind of problem, that the preconditioner does not lose its efficiency throughout the stages of the same time step. Moreover, it confirms the powerful properties of the matrix-free iterative strategy, which allow skipping of Jacobian evaluation without degrading the convergence of the linear solver. Obviously, from the CPU time point of view, lagging the preconditioner improves the computational efficiency, since the matrix is evaluated only once. This is reflected by the speed-up values (see the MFL result columns). We point out that the maximum speed-up values are obtained for high orders, when the impact of the Jacobian assembly is large on the

overall CPU time, and for small time step sizes: in this case the conditioning of the matrix and the CPU time spent on the iterative solution process reduce, and so the computational time saved by skipping the Jacobian assembly reflects on the overall efficiency of the method. It is worth pointing out that by doing so, the matrix-free penalization at low orders is reduced, while speed-up values in the range $[1.41, 3.10]$ are achieved for the $k = 6$, $\Delta t = 0.1$ case.

Similar observations hold true when the matrix-free approximation is employed within a multigrid strategy. Also in this case, the problem has been solved using the same number of GMRES iterations with respect to the non-lagging computation. However, from the CPU time view, a higher speed-up value is achieved both in serial and in parallel runs thanks to the optimal multigrid scalability with respect to single-grid BILU preconditioning. The speed-up factor for the largest polynomial order is now in the range $[2.56, 3.78]$ and $[2.77, 3.98]$ for the large and small time step, respectively. Those speed-up values are larger than the ones obtained for a fully matrix-based implementation of the DG discretization, see Table 5. In comparison to HDG, it can be seen that by using the Jacobian lagging option, larger speed-up values are generally achieved for small time step sizes at high order. However, the use of MG_{full}-pHDG may be preferred to MG_{full}-DG, even coupled with matrix-free and preconditioner lagging for the largest time step employed, as better suitability for dealing with very stiff and high-order discretizations has been highlighted.

We remark that, for computational efficiency, we employ the matrix-free implementation of the iterative solver only on the finest space of the multilevel iterative solution, similarly to what has been done in [12]. This choice is consistent with the results obtained for the single-grid preconditioner in Table 6. In fact, the matrix-free iteration cost is similar to that of a matrix-based one only for high orders, while it is larger for lower-order polynomials. It therefore is appropriate to employ matrix-based smoothers on the coarse levels. When discretizing the equations using high orders, this idea seems to work optimally. Note also that the overall memory footprint of the application is dominated by the allocation of the block-Jacobi preconditioner for the finest space smoother, as shown in Figure 3.

7.5. Remarks

We summarize the main points of the previous section. First, we see clearly that having large time step sizes maximizes the advantages of coupling HDG and p-multigrid, since the expense of using static condensation together with a more powerful and expensive preconditioning strategy produce a faster solver only for high condition numbers of the system, while the higher scalability of the multigrid algorithm as opposed to single-grid reflects on the overall scalability of the solver. On the other hand, for small time step sizes, the computational time is dominated by the Jacobian assembly and condensation for HDG, thus the CPU time as well as the parallel efficiency is dominated by the matrix assembly.

For HDG, we demonstrate how the *mixed* and *primal* formulations provide comparable results in terms of error levels by refining both in space and time, despite the fact that the *primal* form provides a one order lower convergence rate for the gradient variable. From the algorithmic point of view, the statically condensed system is typically solved using roughly the same number of iterations. However, the computational time is considerably lower for the pHDG formulation, since it does not deal with the Jacobian entries, albeit local to each element, related to the state gradient.

For this reason, in the remainder of the work, only the *primal* formulation will be employed for benchmarking. We stress that even for small time step sizes, the single-grid preconditioned *p*HDG solver performs fairly well in comparison to the other solution strategies, showing the benefits of the approach for unsteady flow computations.

We remark that the current implementation of the parallelization strategy makes HDG less parallel efficient than DG. This fact is ascribed to the higher amount of duplicate operations due to the integration over halo mesh elements and faces created by the domain decomposition required for the static condensation of the interior degrees of freedom of the Jacobian matrix. On the other hand, in DG the duplicate work involves only partition faces. Other implementation choices, such as those related to the minimization of the duplicate work on the partition boundaries, may be considered in future works.

We finally point out that, despite being appealing, a matrix-free implementation of the hybridizable discontinuous Galerkin method is not at all straightforward for obtaining satisfactory performance in CPU time [24], and thus its development is beyond the scope of the present work.

8. Results on complex test cases

The second family of numerical experiments deals with the solution of two test cases: i) laminar flow over a two-dimensional circular cylinder at $Re = 100$ and $M = 0.2$ [34]; and ii) the solution of the plunging motion of a NACA 0012 airfoil at $Re = 1000$ and Mach number $M = 0.2$. The latter case is solved by using the ALE mesh motion formulation introduced in [18]. Those results are devoted to extend the comparison to more complex unsteady flow problems. In all the cases reported herein, the right-preconditioning approach is still employed such that the convergence of the linear solver is not affected by changing the preconditioning operator, and therefore all the numerical experiments are performed using a similar accuracy.

8.1. Circular cylinder

Laminar flow around a circular cylinder at Mach number $M = 0.2$ and Reynolds number $Re = 100$ has been solved on a grid of $n_e = 960$ mesh elements using a $\mathbb{P}_6$ space discretization. Figure 7 shows a snapshot of the computed Mach number contours. To integrate the governing equations in time, the four-stage, third-order explicit-first-stage, diagonally-implicit Runge–Kutta method (ESDIRK3) was employed. The solution accuracy was assessed by comparing with literature data [34]. To this end, Table 7 shows the averaged drag and lift coefficients, as well as the Strouhal number of the body forces (C_d, C_l, St) for several temporal refinements on the same grid. The coefficients were obtained by averaging a statistically-developed solution over ten shedding periods. An overall good agreement has been found, while a temporal convergence can be observed by using a non-dimensional time step of $\Delta t \leq 0.25$. It is well known [15] that the time step size greatly affects the iterative solution process, and therefore it has to be taken into account to evaluate the performance. Presently, we use the largest time step size that yields both converged body forces and Strouhal numbers, and this maximizes the efficiency of the solution strategy. Considering the results

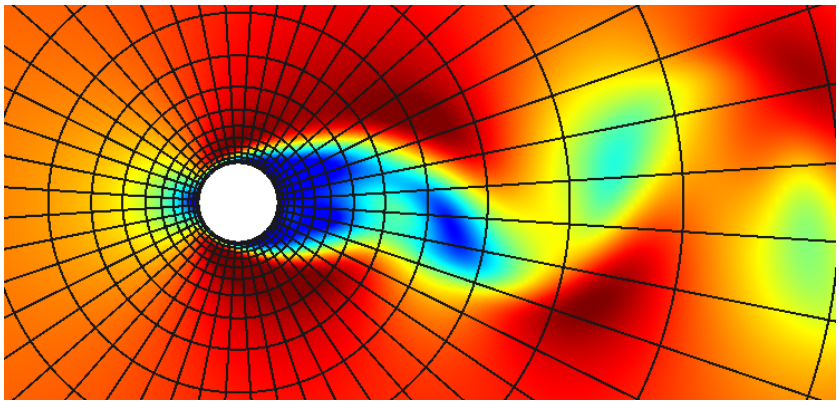


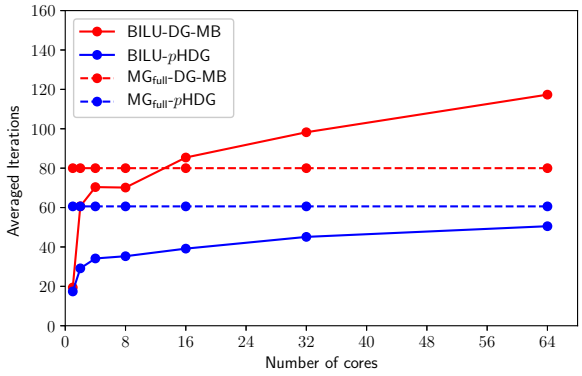
Figure 7: Laminar flow around a circular cylinder at $Re = 100$, $M = 0.2$. Mach number contours.

$(U/L)\Delta t$	C_d	C_l	St
0.5	1.3468	3.383e-03	0.16327
0.25	1.3519	-1.441e-03	0.16410
0.125	1.3527	-1.400e-04	0.16410
0.05	1.3528	-1.718e-04	0.16410
0.025	1.3528	-6.353e-06	0.16410

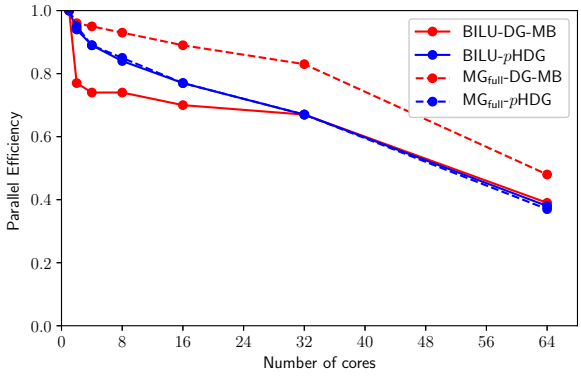
Table 7: Laminar vortex test case at $Re = 100$, $M = 0.05$. Time convergence rates for the DG and HDG spatial discretizations and the ESDIRK3 temporal scheme.

in Table 7, we choose $\Delta t = 0.25$. The parallel performance of the solution strategies introduced in the previous sections is also assessed. To do so, a fully-developed flow field is integrated in time for 10 time steps to compute the average number of GMRES iterations and the convergence rates during the non-linear solution. The computations are performed on the range of 1 to 64 cores (n_p) on a platform based on two 16-core AMD Opteron processors arranged in a two-processor per-node fashion, for a total of 32 cores per node. A fixed relative tolerance of 10^{-6} to stop the GMRES solver is used, as well as an absolute tolerance of 10^{-5} for the Newton-Raphson method.

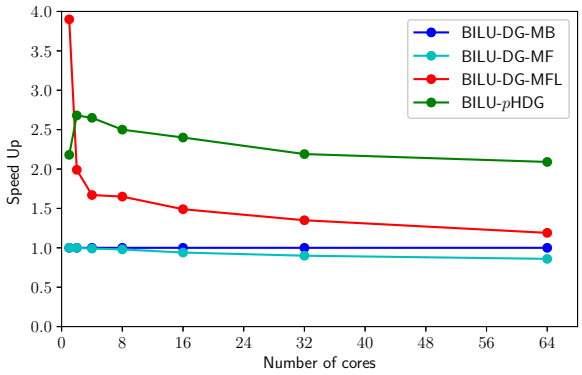
Figure 8 report the results of the computations. As observed for the convected vortex test case, the BILU preconditioner shows an increase in the number of GMRES iterations with increasing number of processes in both the DG and pHDG space discretizations. This can be observed in Figure 8(a), which reports the total number of iterations performed on the finest level of the discretization, averaged throughout the solution process. This behavior is attributed to the way the incomplete lower-upper factorization is performed, as it deals with the square, partition-wise block of the iteration matrix. Therefore, the preconditioner effectiveness naturally decreases as n_p grows, since the number of off-diagonal blocks neglected by the ILU increases with n_p. By switching from MB to MF, the number of iterations remains the same and it is not reported for brevity. On the other hand, the use of a multigrid algorithm with the provided settings results in an ideal algorithmic efficiency, *i.e.*, the number of fine space iterations do not



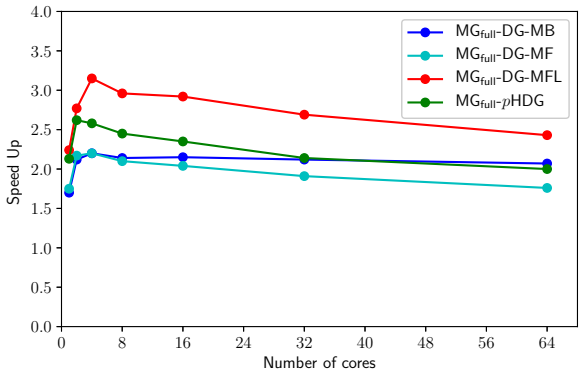
(a) Fine space iterations



(b) Parallel Efficiency



(c) BILU Speed Up



(d) MG$_{\text{full}}$ Speed Up

Figure 8: L_2 solution error. Laminar vortex test case at $Re = 100$, $M = 0.05$. Convergence rates for the DG (dashed), *mHDG* (solid) and pHDG (crosses) discretizations versus theoretical estimates in space and time.

grow by increasing the number of parallel domains, for both DG and pHDG. In the multigrid case, the number of fine space iterations is computed by multiplying the number of outer FGMRES iterations with the number of pre and post smoothing on the finest space, which are reported in Table 3. Note that the computational cost of a multigrid fine space iteration is way cheaper than that of a BILU algorithm, since the element-by-element block Jacobi preconditioner is employed. It is worth also mentioning that in such a practical application the same numerical set-up as that of a DG solver reported in Table 3 has been employed to precondition the system, and that the method proves to work pretty well. Considering pHDG with BILU preconditioning, the iterative solution process still suffers of algorithmic degradation by parallelizing the computation, despite the increase in the number of iterations being smaller than that observed for DG. In addition, we observe a lower number of iterations on average, meaning that the condition number of the matrix has been reduced by the static condensation. The superior parallel efficiency of the multigrid algorithm is clearly visible in Figure 8(b), which reports the strong scalability of the algorithms. While the use of BILU-DG-MB

drops the parallel efficiency below 80% by partitioning the computation on just two computational cores, the multigrid algorithm allows to keep good parallel efficiencies up to 32 cores. The parallel efficiency of the pHDG solver lies somehow in between the two, and it does not change by modifying the preconditioner, which suggests that it is dominated by the scalability of the static condensation/back solve portion of the application.

Figure 8(c) reports the speed up values of the matrix free (MF) and lagged matrix free (MFL) algorithms, which updates the preconditioner operators only once per time step. The performance of the pHDG is also reported. In all of the cases, the reference algorithm is BILU-DG-MB. An improvement in computational efficiency can be observed for the MFL and pHDG cases, which speeds up values in favor of the latter. When switching to the full multigrid preconditioner, see Figure 8(d), we observe an overall increase of performance for the DG algorithms. On the other hand, the costs of pHDG, associated with the static condensation and back solve, cannot be offset simply by the use of a better preconditioner and still dominate the percentage of the overall computational time. As a result, the MG_{full}-DG-MFL strategy performs the best, with speed-up values in the range [2.24, 3.15]. Note that a similar performance between MG_{full}-DG-MB, MG_{full}-DG-MF and pHDG is observed, but while the matrix based one requires a full Jacobian matrix allocation, the matrix free and the hybridizable methods reduce considerably the memory footprint according to what is reported in Figure 3.

As a final comparison, Table 8 reports the same test case solved using a grid obtained by splitting the quadrilateral elements into triangles. Full-order basis functions were employed. Only the computation with $n_p = 64$ is reported. It is worth pointing out that this space discretization reduces the total number of DoFs by the 23.8% with respect to the previous one. By comparing the computational time and average number of GMRES iterations, similar speed-up values are obtained using the DG-MB solver as well as pHDG. On the other hand, the DG-MF solver seems to be slightly penalized with respect to the MB one, as the speed-up values of the computations drop by a factor between 7 to 15 percent. It is worth pointing out that the matrix-free penalization that arise in this case is in line to what reported in previous studies [15, 12] using broken polynomial spaces, which reduce by increasing the number of DoFs per element, for example in three-dimensional computations. On the other hand, a slightly larger speedup for pHDG computations has been observed, which makes this solution strategy the most convenient from the CPU time point of view.

8.2. *Laminar flow around a heaving and pitching NACA 0012 airfoil*

This test case is the CL1 (heaving and pitching airfoil) proposed for the 5^{th} international workshop on high-order CFD methods (HiOCFD5), see [33]. The simulation involves the compressible Navier–Stokes equations, with $\gamma = 1.4$, $Pr = 0.72$, and constant viscosity, to simulate flow over a moving NACA 0012 airfoil. The airfoil is modified to obtain a closed trailing edge via the equation

$$y(x) = \pm 0.6\left(0.2969\sqrt{x} - 0.1260x - 0.3516x^2 + 0.2843x^3 - 0.1036x^4\right), \tag{52}$$

Preconditioner	BILU					
Case	DG-MB		DG-MF	DG-MFL	*p*HDG	
n_p	Time	IT_a	SU_{MB}	SU_{MB}	SU_{MB}	IT_a
64	322.26	129.050	0.80	1.00	2.40	49.750
Preconditioner	MG_{full}					
Case	DG-MB		DG-MF	DG-MFL	*p*HDG	
n_p	SU_{MB}	IT_a	SU_{MB}	SU_{MB}	SU_{MB}	IT_a
64	1.66	4.525	1.43	1.77	2.05	3.388

Table 8: Circular cylinder test case at $Re = 100$, $M = 0.2$, discretized using 1920 mesh elements with $\mathbb{P}_6$ full-order basis functions. Computational efficiency comparison of DG and *p*HDG solvers using BILU and *p*-multigrid, respectively. SU_{MB} stands for the speed-up factor relative to the matrix-based, BILU-DG computation and IT_a for the average number of GMRES iterations.

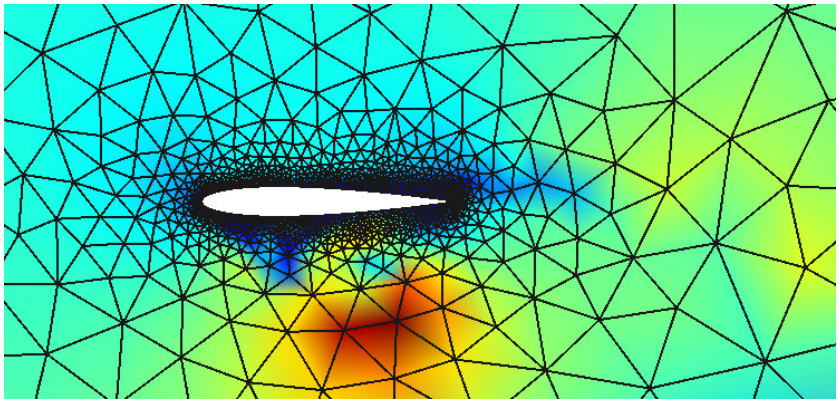


Figure 9: Laminar flow around a heaving NACA 0012 airfoil. $Re = 1000$, $M = 0.2$. Mach number contours at the final time $T = 2$.

with $x \in [0, 1]$. The initial condition is a steady state solution at a free-stream Mach number of $M = 0.2$ and a Reynolds number $Re = 1000$. The motion is a pure plunging motion governed by the following function

$$h(t) = t^2(3 - t)/4, \tag{53}$$

and the output of interest is the total energy and vertical impulse exchanged between the airfoil and the fluid,

$$W = \int_T F_y(t)\dot{h}(t)dt, \qquad I = \int_T F_y(t)dt, \tag{54}$$

where $F_y(t)$ is the vertical force computed on the airfoil surface, and $\dot{h}(t)$ is the time derivative of Eq. (53). In particular, the output is evaluated at a non-dimensional time $T = 2$. The mesh consists of a triangulation of the domain by $n_e = 2137$ elements, shown in Figure 9, and the solution approximation space is $\mathbb{P}_6$.

The time step size of the simulation has been evaluated through a time-convergence analysis of the output quantities, reported in Table 9. Table 10 compares the computational efficiency of the solution strategies in terms of the speed-up factor, SU_{MB}, and the number of iterations, IT_a. Only the computation with the larger number of cores is reported to show the efficiency on large and practical computations. The reference to compute the speed-up is the

$T/\Delta t$	W	I
20	-1.3860	-2.3667
40	-1.3839	-2.3444
80	-1.3837	-2.3391
160	-1.3837	-2.3378

Table 9: Time convergence analysis of the laminar flow around a plunging NACA 0012 airfoil. A satisfactory accuracy is observed for $T/\Delta t = 2$, where $T = 2$ is the length of the simulation and Δt is the time step size.

Preconditioner	BILU					
Case	DG-MB		DG-MF	DG-MFL	pHDG	
n_p	Time	IT_a	SU_{MB}	SU_{MB}	SU_{MB}	IT_a
64	900.19	136.119	0.69	0.81	2.35	49.750
Preconditioner	MG_{full}					
Case	DG-MB		DG-MF	DG-MFL	pHDG	
n_p	SU_{MB}	IT_a	SU_{MB}	SU_{MB}	SU_{MB}	IT_a
64	1.49	4.477	1.06	1.13	2.13	2.754

Table 10: Circular cylinder test case at $Re = 100$, $M = 0.2$, discretized using 2137 mesh elements with $\mathbb{P}_6$ full-order basis functions. Computational efficiency comparison of DG and pHDG solvers using BILU and p-multigrid, respectively. SU_{MB} stands for the speed-up factor referred to the matrix-based, BILU-DG computation and IT_a for the average number of GMRES iterations.

computational time of the matrix-based, BILU-DG solver. By switching from a matrix-based to a matrix-free implementation, the computational strategy is penalized by higher computational cost of a single iterations, see the MF column. By employing the Jacobian lagging (MFL), this penalization is reduced only slightly. On the other hand, the pHDG implementation provides a solver which is more than twice as much as fast, while the system require a considerably lower number of GMRES iterations with respect to the reference. The use of p-multigrid preconditinoing in a DG context provides a speed-up factor of about 1.49 for a fully matrix-based implementation, and the number of iterations drops from 136 to around 4.4 on average. The use of matrix-free in this case still penalizes the solver, which is about 12% faster for the MFL case. Multigrid preconditioning is shown to be robust enough to precondition the linear system arising from the pHDG discretization, since it converges using an average of 2.754 iterations. However, this gain is not reflected on the CPU time, which is slightly higher.

9. Conclusions

The paper compares, within the same framework, the computational efficiency of different high-order discontinuous Galerkin implementations. The first involves a modal discontinuous Galekin method coupled with matrix-based and matrix-free iterative solvers, while the second one considers a hybridizable discontinuous Galerkin implementation, both in the *mixed* form, which allocates the components of the Jacobians related to the gradient variable, and in the *primal* form, which forgoes separate approximation of the gradient variable and symmetrizes the discretization

by adding an additional adjoint-consistency term. The efficiency of the solution strategies is assessed by comparing different single-level preconditioners as well as multilevel ones such as p-multigrid, on a variety of two-dimensional test cases involving laminar viscous flows, including mesh motion, on meshes made by triangular and quadrilateral elements. The effects of parallel efficiency and the use of different basis functions have also been considered. The paper shows that the use of a matrix-free implementation of the iterative solver in the context of implicit discontinuous Galerkin discretizations provides a memory footprint which is in line to that of an HDG method if a block-diagonal preconditioner is employed within the smoother on the finest space. Moreover, the *primal* HDG method becomes more efficient than the *mixed* one, having a lower number of non-zeros Jacobian entries, and it provides comparable error levels. If compared to *p*HDG, the p-multigrid matrix-free solver is competitive in terms of CPU time when the problems involve time marching with small time steps, since the preconditioner evaluation can be lagged. On the other hand, HDG methods require expensive element-wise operations that become a bottleneck in those conditions. Finally, a novel approximate-inherited p-multigrid strategy has also been introduced for HDG. Such a strategy is more robust and efficient for different test cases and mesh types, and it is able to reduce considerably the number of iterations of the solution process. However, this gain in number of iterations is not reflected in the CPU time of the solver. Future work will be devoted to the validation of those strategies on stiff three-dimensional cases involving laminar and turbulent flows, and possibly hybrid RANS-LES models.

Acknowledgements

M. Franciolini acknowledges Dr. Lorenzo Botti from the University of Bergamo for useful discussions and comments. Funding received from the Department of Energy under grant DE-FG02-13ER26146/DE-SC0010341 is also gratefully acknowledged.

References

[1] F. Bassi, A. Crivellini, D. A. Di Pietro, S. Rebay, An implicit high-order discontinuous Galerkin method for steady and unsteady incompressible flows, Computers & Fluids 36 (10) (2007) 1529–1546.

[2] F. Bassi, L. Botti, A. Colombo, A. Ghidoni, F. Massa, Linearly implicit Rosenbrock-type Runge–Kutta schemes applied to the Discontinuous Galerkin solution of compressible and incompressible unsteady flows, Computers & Fluids 118 (2015) 305–320.

[3] J. S. Hesthaven, T. Warburton, Nodal discontinuous Galerkin methods: algorithms, analysis, and applications, Springer Science & Business Media, 2007.

[4] L. Wang, D. J. Mavriplis, Implicit solution of the unsteady Euler equations for high-order accurate discontinuous Galerkin discretizations, Journal of Computational Physics 225 (2) (2007) 1994 – 2015.

[5] P.-O. Persson, J. Peraire, Newton-GMRES preconditioning for discontinuous Galerkin discretizations of the Navier–Stokes equations, SIAM Journal on Scientific Computing 30 (6) (2008) 2709–2733.

[6] L. T. Diosady, D. L. Darmofal, Preconditioning methods for discontinuous Galerkin solutions of the Navier–Stokes equations, Journal of Computational Physics 228 (11) (2009) 3917–3935.

[7] P. Birken, G. Gassner, M. Haas, C. D. Munz, Preconditioning for modal discontinuous Galerkin methods for unsteady 3D Navier–Stokes equations, Journal of Computational Physics 240 (2013) 20–35.

[8] Y. Saad, A flexible inner-outer preconditioned GMRES algorithm, SIAM Journal on Scientific Computing 14 (2) (1993) 461–469.

[9] K. J. Fidkowski, T. A. Oliver, J. Lu, D. L. Darmofal, *p*-Multigrid solution of high-order discontinuous Galerkin discretizations of the compressible Navier–Stokes equations, Journal of Computational Physics 207 (1) (2005) 92–113.

[10] K. Shahbazi, D. J. Mavriplis, N. K. Burgess, Multigrid algorithms for high-order discontinuous Galerkin discretizations of the compressible Navier–Stokes equations, Journal of Computational Physics 228 (21) (2009) 7917–7940.

[11] L. Botti, A. Colombo, F. Bassi, *h*-multigrid agglomeration based solution strategies for discontinuous Galerkin discretizations of incompressible flow problems, Journal of Computational Physics 347 (2017) 382–415.

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

[13] A. Crivellini, F. Bassi, An implicit matrix-free discontinuous Galerkin solver for viscous and turbulent aerodynamic simulations, Computers & fluids 50 (1) (2011) 81–93.

[14] M. Ceze, L. Diosady, S. M. Murman, Development of a high-order space-time matrix-free adjoint solver, in: 54th AIAA Aerospace Sciences Meeting, 0833, 2016.

[15] M. Franciolini, A. Crivellini, A. Nigro, On the efficiency of a matrix-free linearly implicit time integration strategy for high-order Discontinuous Galerkin solutions of incompressible turbulent flows, Computers & Fluids 159 (2017) 276–294.

[16] B. Cockburn, O. Dubois, J. Gopalakrishnan, S. Tan, Multigrid for an HDG method, IMA Journal of Numerical Analysis 34 (4) (2014) 1386–1425.

[17] N. C. Nguyen, J. Peraire, B. Cockburn, An implicit high-order hybridizable discontinuous Galerkin method for nonlinear convection–diffusion equations, Journal of Computational Physics 228 (23) (2009) 8841–8855.

[18] K. J. Fidkowski, A hybridized discontinuous Galerkin method on mapped deforming domains, Computers & Fluids 139 (2016) 80–91.

[19] R. M. Kirby, S. J. Sherwin, B. Cockburn, To CG or to HDG: a comparative study, Journal of Scientific Computing 51 (1) (2012) 183–212.

[20] S. Yakovlev, D. Moxey, R. M. Kirby, S. J. Sherwin, To CG or to HDG: a comparative study in 3D, Journal of Scientific Computing 67 (1) (2016) 192–220.

[21] M. Woopen, A. Balan, G. May, J. Schütz, A comparison of hybridized and standard DG methods for target-based hp-adaptive simulation of compressible flow, Computers & Fluids 98 (2014) 3–16.

[22] J. Dahm, Toward Accurate, Efficient, and Robust Hybridized Discontinuous Galerkin Methods, Phd thesis, University of Michigan, 2017.

[23] P. Devloo, C. Faria, A. Farias, S. Gomes, A. Loula, S. Malta, On continuous, discontinuous, mixed, and primal hybrid finite element methods for second-order elliptic problems, International Journal for Numerical Methods in Engineering .

[24] M. Kronbichler, W. A. Wall, A performance comparison of continuous and discontinuous Galerkin methods with fast multigrid solvers, SIAM Journal on Scientific Computing 40 (5) (2018) A3423–A3448.

[25] J. Schütz, V. Aizinger, A hierarchical scale separation approach for the hybridized discontinuous Galerkin method, Journal of Computational and Applied Mathematics 317 (2017) 500–509.

[26] D. N. Arnold, F. Brezzi, B. Cockburn, L. D. Marini, Unified analysis of discontinuous Galerkin methods for elliptic problems, SIAM J. Numer. Anal. 39 (5) (2002) 1749–1779.

[27] P. L. Roe, Approximate Riemann solvers, parameter vectors, and difference schemes, Journal of computational physics 43 (2) (1981) 357–372.

[28] F. Bassi, S. Rebay, G. Mariotti, S. Pedinotti, M. Savini, A High-Order Accurate Discontinuous Finite Element Method for Inviscid and Viscous Turbomachinery Flows, in: R. Decuypere, G. Dibelius (Eds.), 2nd European Conference on Turbomachinery Fluid Dynamics and Thermodynamics, Technologisch Instituut, Antwerpen, Belgium, 99–108, 1997.

[29] F. Brezzi, G. Manzini, D. Marini, P. Pietra, A. Russo, Discontinuous Galerkin approximations for elliptic problems, Numer. Methods Partial Differential Equations 16 (2000) 365–378.

[30] F. Bassi, S. Rebay, High-Order Accurate Discontinuous Finite Element Solution of the 2D Euler Equations, J. Comput. Phys. 138 (1997) 251–285.

[31] H. Bijl, M. H. Carpenter, V. N. Vatsa, C. A. Kennedy, Implicit time integration schemes for the unsteady compressible Navier–Stokes

equations: laminar flow, Journal of Computational Physics 179 (1) (2002) 313–329.

[32] P. F. Antonietti, M. Sarti, M. Verani, Multigrid algorithms for hp-discontinuous Galerkin discretizations of elliptic problems, SIAM Journal on Numerical Analysis 53 (1) (2015) 598–618.

[33] 5^{th} International Workshop on High–Order CFD Methods, https://how5.cenaero.be/, 2018.

[34] J. R. Meneghini, F. Saltara, C. L. R. Siqueira, J. A. Ferrari Jr, Numerical simulation of flow interference between two circular cylinders in tandem and side-by-side arrangements, Journal of Fluids and Structures 15 (2) (2001) 327–350.