A DOMAIN-SPECIFIC LANGUAGE FOR THE HYBRIDIZATION AND STATIC CONDENSATION OF FINITE ELEMENT METHODS*

THOMAS H. GIBSON†, LAWRENCE MITCHELL‡, DAVID A. HAM†, AND COLIN J. COTTER†

Abstract. In this paper, we introduce a domain-specific language (DSL) for concisely expressing localized linear algebra on finite element tensors, and its integration within a code-generation framework. This DSL is general enough to facilitate the automatic generation of cell-local linear algebra kernels necessary for the implementation of static condensation methods and local solvers for a variety of problems. We demonstrate its use for the static condensation of continuous Galerkin problems, and systems arising from hybridizing a finite element discretization. Additionally, we demonstrate how this DSL can be used to implement local post-processing techniques to achieve superconvergent approximation to mixed problems. Finally, we show that these hybridization and static condensation procedures can act as effective preconditioners for mixed problems. We use the DSL in this paper to implement high-level preconditioning interfaces for the hybridization of mixed problems, as well generic static condensation. Our implementation builds on the solver composability of the PETSc library by providing reduced operators, which are obtained from locally assembled expressions, with the necessary context to specify full solver configurations on the resulting linear systems. We present some examples for model second-order elliptic problems, including a new hybridization preconditioner for the linearized system in a nonlinear method for a simplified atmospheric model.

Key words. Domain-specific language, automatic code generation, hybridization, static condensation, mixed finite elements.

AMS subject classifications. 65F08, 65K05, 65M60, 68N19.

1. Introduction.

The development of simulation software is becoming an increasingly important aspect of modern scientific computing. Such software requires a vast range of knowledge spanning several disciplines, ranging from abstract mathematics and computer science to software engineering and high-performance computing. Developing advanced finite element techniques are particularly complicated from both implementation and mathematical perspectives. Software projects developing automatic code-generation systems has become quite popular in recent years, as such systems help create a separation of concerns which focuses on a particular complexity independent from the rest. Examples of such projects include FreeFEM++ [31], Sundance [42], the FEniCS Project [39], Feel++ [45], and Firedrake [46].

The finite element method (FEM) is a mathematically robust framework for computing solutions of partial differential equations (PDEs), with a formulation that is highly amenable to code-generation techniques. The description of the weak formulation of the PDEs, together with the discrete spaces from which solution approximations are constructed, is sufficient to characterize the problem completely. Both the FEniCS and Firedrake projects employ the *Unified Form Language* (UFL) [1] to specify the finite element integral forms and discrete function spaces necessary to properly define the finite element problem. UFL is a highly expressive domain-specific language (DSL) embedded in Python which provides the information which

* **Funding:** This work was supported by the Engineering and Physical Sciences Research Council [grant numbers EP/M011054/1, EP/L000407/1, EP/L016613/1]; and the Natural Environment Research Council [grant number NE/K008951/1]

†Department of Mathematics, Imperial College London, South Kensington Campus, London SW7 2AZ, UK (t.gibson15@imperial.ac.uk, david.ham@imperial.ac.uk, colin.cotter@imperial.ac.uk)

‡Department of Computing and Department of Mathematics, Imperial College London, South Kensington Campus, London SW7 2AZ, UK (lawrence.mitchell@imperial.ac.uk)

finite element code-generation systems require.

There are classes of finite element discretizations which admit discrete systems that can be solved more efficiently by directly manipulating local tensors. For example, the static condensation technique for the reduction of global finite element systems [29, 34] produces smaller globally-coupled linear systems by eliminating interior unknowns to arrive at an equation for the facet[1] degrees of freedom only. Alternatively, hybridized finite element methods [3, 14, 18] introduce Lagrange multipliers enforcing certain continuity constraints on finite element functions. Static condensation can then be applied to the augmented system to produce a reduced equation for the multipliers. Methods of this type are often accompanied by local post-processing techniques that produce superconvergent approximations [10, 20, 21].

The main objective of this paper is to provide a high-level description of elemental linear algebra for code-generation systems. In doing so, we provide a means to enable rapid development of hybridization and static condensation techniques within an automatic code-generation framework. Our work is implemented in the Firedrake finite element framework and the PETSc solver library. Firedrake's solver abstraction builds on the PETSc's formalism for solving discrete systems, using the latter's Python interface, petsc4py [24].

The rest of the paper is organized as follows. In section 2, we introduce our new DSL within the Firedrake software package [46] which allows concise expression of localized linear algebra operations on finite element tensors. We provide some contextual examples for static condensation and hybridization, including a discussion on post-processing. We then outline in section 3 how the Python interface with PETSc can be used in tandem with our DSL to automate the hybridization and static condensation of finite element systems as preconditioners. We demonstrate our implementations for manufactured test problems in section 4. Subsection 4.3 demonstrates the composability of our implementation of a hybridized mixed method as a preconditioner for the linearized system of a nonlinear method for the rotating shallow water equations. Conclusions follow in section 5.

1.1. Notation. Let $\mathcal{T}_h$ denote a tesselation of $\Omega \subset \mathbb{R}^n$, the computational domain, consisting of polygonal elements K associated with a mesh size parameter h, and $\partial\mathcal{T}_h = \{e \in \partial K : K \in \mathcal{T}_h\}$ the set of facets of $\mathcal{T}_h$. We denote the integral forms over $\mathcal{T}_h$ and any facet set $\Gamma \subset \partial\mathcal{T}_h$ by

$$(u,v)_K = \int_K u \cdot v \,\mathrm{d}\boldsymbol{x}, \qquad \langle u,v\rangle_e = \int_e u \cdot v \,\mathrm{d}s, \tag{1}$$

$$(u,v)_{\mathcal{T}_h} = \sum_{K\in\mathcal{T}_h} (u,v)_K, \qquad \langle u,v\rangle_\Gamma = \sum_{e\in\Gamma} \langle u,v\rangle_e, \tag{2}$$

where $\cdot$ should be interpreted as standard multiplication for scalar functions or a dot product for vector functions.

For any double-valued vector field $\boldsymbol{w}$ on a facet $e \in \partial\mathcal{T}_h$, we define the jump of its normal component across e by

$$[\![\boldsymbol{w}]\!]_e = \begin{cases} \boldsymbol{w}|_{e^+} \cdot \boldsymbol{n}_{e^+} + \boldsymbol{w}|_{e^-} \cdot \boldsymbol{n}_{e^-}, & e \in \partial\mathcal{T}_h \setminus \partial\Omega \equiv \mathcal{E}_h^\circ \\ \boldsymbol{w}|_e \cdot \boldsymbol{n}_e, & e \in \partial\mathcal{T}_h \cap \partial\Omega \equiv \mathcal{E}_h^\partial \end{cases} \tag{3}$$

[1] "Facets" here describes the topological entity of the mesh which is one less than the total space dimension. In two dimensions, these are edges, and in three dimensions the are polygonal faces of each cell of the mesh.

where $+$ and $-$ denotes the positive and negative restrictions respectively. Whenever the facet domain is clear by the context, we omit the subscripts and will instead write $[\![\boldsymbol{w}]\!]$.

2. A language for linear algebra on local finite element tensors. We present an expressive language for dense linear algebra on the elemental matrix systems arising from finite element problems. The language, which we call *Slate*, inherits typical mathematical operations performed on matrices and vectors, hence the input syntax is comparable to high-level linear algebra software (e.g. MATLAB). The Slate language provides basic abstract building blocks which can be used by a specialized compiler for linear algebra to generate low-level code implementations.

Slate is designed around the Unified Form Language (UFL) [1, 39], a DSL embedded in Python which provides abstract representations of finite element forms. UFL expressions are interpreted by a *form compiler*, which translates UFL into compiled code for the local assembly of a form over the cells and facets of a mesh. The FEniCS project employs the *FEniCS Form Compiler* (FFC) [35, 40], while Firedrake uses the *Two-Stage Form Compiler* (TSFC) [33]. It is the job of the FEM framework to take the resulting code and assemble the local contributions of the finite element form over the mesh. These local data objects must then be inserted into global data structures before handing over to a linear solver. These operations are handled by the PyOP2 [47] and DOLFIN [41] frameworks in Firedrake and FEniCS respectively.

2.1. An overview of Slate. To clarify conventions and scope of Slate, we start by considering a general form. With $\mathcal{T}_h$, $\mathcal{E}_h^\circ$, and $\mathcal{E}_h^\partial$ denoting the sets of cells, interior facets, and exterior facets respectively, suppose we have a finite element form:

$$a(\boldsymbol{c};\boldsymbol{v}) = \sum_{K\in\mathcal{T}_h}\int_K \mathcal{I}^c(\boldsymbol{c};\boldsymbol{v})\mathrm{d}\boldsymbol{x} + \sum_{e\in\mathcal{E}_h^\circ}\int_e \mathcal{I}^{f,\circ}(\boldsymbol{c};\boldsymbol{v})\mathrm{d}s + \sum_{e\in\mathcal{E}_h^\partial}\int_e \mathcal{I}^{f,\partial}(\boldsymbol{c};\boldsymbol{v})\mathrm{d}s, \tag{4}$$

where $\mathrm{d}\boldsymbol{x}$ and $\mathrm{d}s$ denote appropriate integration measures. Equation (4) is uniquely determined by its lists (possibly of 0-length) of arbitrary coefficient functions $\boldsymbol{c} = (c_0, \cdots, c_p)$ in the associated finite element spaces, arguments $\boldsymbol{v} = (v_0, \cdots, v_q)$ describing any test or trial functions, and its integrand expressions for each integral type: $\mathcal{I}^c$, $\mathcal{I}^{f,\circ}$, $\mathcal{I}^{f,\partial}$. The form $a(\boldsymbol{c};\boldsymbol{v})$ describes a finite element form *globally* over the entire problem domain. The contribution of (4) in each cell K of the mesh $\mathcal{T}_h$ is simply

$$a(\boldsymbol{c};\boldsymbol{v})|_K = \int_K \mathcal{I}^c(\boldsymbol{c};\boldsymbol{v})\mathrm{d}\boldsymbol{x} + \sum_{e\in\partial K\setminus\partial\Omega}\int_e \mathcal{I}^{f,\circ}(\boldsymbol{c};\boldsymbol{v})\mathrm{d}s + \sum_{e\in\partial K\cap\partial\Omega}\int_e \mathcal{I}^{f,\partial}(\boldsymbol{c};\boldsymbol{v})\mathrm{d}s. \tag{5}$$

We call (5) the *cell-local* contribution of $a(\boldsymbol{c};\boldsymbol{v})$. Equation (5) produces an element tensor which is mapped into a global data structure. However, one may want produce a new local tensor by algebraically manipulating different element tensors. This is precisely the job of our new DSL.

The Slate language consists of two primary abstractions for linear algebra: (1) terminal element tensors corresponding to multi-linear integral forms, or assembled data; and (2) expressions consisting of operations on terminal tensors or existing Slate expressions. A summary of Slate nodes are listed below:

- `Tensor`$(a(\boldsymbol{c};\boldsymbol{v}))$
  associates a form, expressed in UFL, with its local element tensor:

$$A^K \leftarrow a(\boldsymbol{c};\boldsymbol{v})|_K, \text{ for all } K \in \mathcal{T}_h. \tag{6}$$

The number of arguments $\boldsymbol{v}$ determine the *rank* of `Tensor`, i.e. scalars, vectors, and matrices are produced from 0-forms, 1-forms, and 2-forms[2] respectively.

- `AssembledVector`(f)
  where f is some finite element function. The result associates a function with its local coefficient vectors.
- `Block`(A^K, $\boldsymbol{i}$)
  where A^K is a `Tensor` corresponding to a mixed form and $\boldsymbol{i}$ are indices over the test and trial spaces associated with the mixed tensor. The result is a block of A^K corresponding to the indices $\boldsymbol{i}$. For example, if a matrix A corresponds to the bilinear form $a : V \times W \to \mathbb{R}$, where $V = V_0 \times \cdots \times V_n$ and $W = W_0 \times \cdots \times W_m$ are product spaces consisting of finite element spaces $\{V_i\}_{i=0}^n$, $\{W_i\}_{i=0}^m$, then the cell-local tensors have the form:

$$A^K = \begin{bmatrix} A^K_{00} & A^K_{01} & \cdots & A^K_{0m} \\ A^K_{10} & A^K_{11} & \cdots & A^K_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ A^K_{n0} & A^K_{n1} & \cdots & A^K_{nm} \end{bmatrix}. \tag{7}$$

  The associated submatrix of (7) with indices $\boldsymbol{i} = (\boldsymbol{p}, \boldsymbol{q})$, $\boldsymbol{p} = \{p_1, \cdots, p_r\}$, $\boldsymbol{q} = \{q_1, \cdots, q_c\}$, is

$$A^K_{\boldsymbol{pq}} = \begin{bmatrix} A^K_{p_1 q_1} & \cdots & A^K_{p_1 q_c} \\ \vdots & \ddots & \vdots \\ A^K_{p_r q_1} & \cdots & A^K_{p_r q_c} \end{bmatrix} \leftarrow \texttt{Block}(A^K, (\boldsymbol{p}, \boldsymbol{q})), \tag{8}$$

  where $\boldsymbol{p} \subset \{1, \cdots, n\}$, $\boldsymbol{q} \subset \{1, \cdots, m\}$.
- `TensorOp`(operands)
  implements all unary and binary operations on Slate objects. These include the following:
  - *Binary operations*:
    `Add`(A, B): The addition of two tensors of equal rank and shape;
    `Mul`(A, B): Standard multiplication of two tensors of appropriate shape.
  - *Unary operations*:
    `Negative`(A): The additive inverse of a local tensor;
    `Transpose`(A): The transpose of a local tensor;
    `Inverse`(A): The inverse of a local square tensor.

By construction, each `Tensor` is uniquely defined by its associated form, and performing any number of binary or unary operations on the `Tensor` objects produces a *Slate expression*. The composition of all the operations presented gives us the necessary algebraic framework needed for the class of problems presented in this paper.

Slate expressions are handled by a *linear algebra compiler*, which employs TSFC for local assembly and generates a dense linear algebra kernel to be iterated cell-wise. Our compiler generates C++ code, using the templated library Eigen [28] for dense linear algebra. During execution, the local computations in each cell are mapped

[2]As with UFL, Slate is capable of abstractly representing arbitrary rank tensors. However, only rank ≤ 2 tensors are typically used in most finite element applications and therefore we currently only generate code for those ranks.

into global data objects via appropriate indirection mappings. Figure 1 provides an illustration of the complete tool-chain.

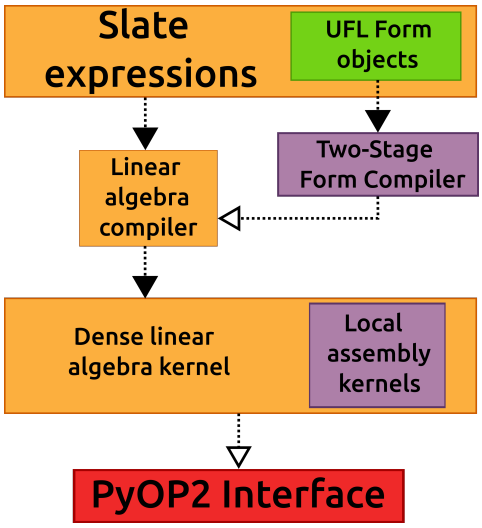


FIG. 1. *The Slate language wraps forms, expressed in UFL, describing the finite element problem. The Slate expressions are handed over to a specialized linear algebra compiler, which produces TSFC kernels for local element assembly, and a single "macro" kernel which performs the dense linear algebra. The resulting kernels are passed to the PyOP2 interface, which wraps the Slate kernel in a mesh-iteration kernel. Parallel scheduling and code generation occurs after the PyOP2 layer.*

2.2. Examples. We now present a few examples and discuss solution methods which require element-wise manipulations of finite element systems and their specification in Slate. We stress here that Slate is not limited to these model problems; rather these examples were chosen for clarity and to demonstrate key features of the Slate language. In section 3 and section 4, we discuss more intricate ways the Slate DSL is used in custom preconditioners for linear systems, including some numerical tests ranging from manufactured problems to a simplified atmospheric model.

2.2.1. Static condensation of CG methods. As a first example, consider the positive-definite Helmholtz problem given by:

$$-\nabla \cdot (\kappa \nabla p) + cp = f \text{ in } \Omega, \tag{9}$$

$$p = p_0 \text{ on } \partial\Omega_D, \tag{10}$$

$$-\kappa \nabla p \cdot \boldsymbol{n} = g \text{ on } \partial\Omega_N, \tag{11}$$

where $\partial\Omega_D \cup \partial\Omega_N = \partial\Omega$ and κ, $c : \Omega \to \mathbb{R}^+$ are positive-valued coefficients. The H^1-discretization is formed by tessellating Ω into $\mathcal{T}_h$, multiplying by a test function and integrating by parts in the usual finite element way. The resulting weak formulation reads: find $p_h \in V_h$ such that

$$a(\kappa, c; \phi, p_h) \equiv (\nabla\phi, \kappa\nabla p_h)_{\mathcal{T}_h} + (\phi, cp_h)_{\mathcal{T}_h} = (\phi, f)_{\mathcal{T}_h} - \langle \phi, g \rangle_{\partial\Omega_N} \equiv F(f, g; \phi) \tag{12}$$

for all $\phi \in V_{h,0}$, where V_h is a continuous piecewise polynomial space incorporating the condition $p_h = p_0$ on $\partial\Omega_D$, and $V_{h,0}$ is a continuous piecewise polynomial space whose functions have zero trace on $\partial\Omega_D$. With P denoting the vector degrees of

freedom, solving (12) requires the solution of the symmetric system of the form:

$$AP = F. \tag{13}$$

The static condensation technique of Guyan and Irons [29, 34] reduces the size of the global matrix A by partitioning the problem into separate systems for the degrees of freedom interior to each element, and the degrees of freedom associated with elemental boundaries. This produces a system of the form:

$$\begin{bmatrix} A_{00} & A_{0\partial} \\ A_{\partial 0} & A_{\partial\partial} \end{bmatrix} \begin{Bmatrix} P_0 \\ P_\partial \end{Bmatrix} = \begin{Bmatrix} F_0 \\ F_\partial \end{Bmatrix}, \tag{14}$$

where the subscripts 0 and ∂ denote elemental interior and boundary coupling respectively. The interior unknowns are then condensed out of (14) via a Schur complement, leaving a smaller globally-coupled system for the interface unknowns:

$$\left(A_{\partial\partial} - A_{\partial 0} A_{00}^{-1} A_{0\partial}\right) P_\partial = F_\partial - A_{\partial 0} A_{00}^{-1} F_0. \tag{15}$$

The matrix A_{00} has block-diagonal structure by our choice of degrees of freedom. Therefore the system in (15) can be assembled by performing element-wise operations on the local finite element matrices and vectors. Once P_∂ is determined, P_0 can be recovered by solving the system:

$$A_{00} P_0 = \left(F_0 - A_{0\partial} P_\partial\right). \tag{16}$$

By the same argument as before, (16) can be evaluated by solving a sequence of local systems in each element of $\mathcal{T}_h$.

There are a number of ways to obtain the partitioned matrix system in (14). The most natural way using UFL is to construct the appropriate finite element basis functions by introducing the spaces:

$$V_h^0 = \{\phi \in V_h : \phi|_K \in V_h(K), \phi|_{\partial K} = 0, \forall K \in \mathcal{T}_h\}, \tag{17}$$

$$V_h^\partial = \{\phi \in V_h : \phi|_K \in V_h(K), \phi|_{\partial K} = 0 \implies \phi|_K = 0, \forall K \in \mathcal{T}_h\}, \tag{18}$$

where V_h^0 and V_h^∂ are function spaces corresponding to functions in V_h *restricted* to element interiors and boundaries respectively. Note that the space V_h^∂ must now enforce the boundary condition in (10). We can now rewrite the problem in (12) as the following: find $(p_h^0, p_h^\partial) \in V_h^0 \times V_h^\partial$ such that

$$\widehat{a}\left(\kappa, c; \left(\phi^0, \phi^\partial\right), \left(p_h^0, p_h^\partial\right)\right) = \widehat{F}(f, g; \phi^0, \phi^\partial) \tag{19}$$

holds for all test functions $(\phi^0, \phi^\partial) \in V_h^0 \times V_{h,0}^\partial$, where

$$\widehat{a}\left(\kappa, c; \left(\phi^0, \phi^\partial\right), \left(p_h^0, p_h^\partial\right)\right) \equiv \begin{cases} a(\kappa, c; \phi^0, p_h^0) + a(\kappa, c; \phi^0, p_h^\partial), \\ a(\kappa, c; \phi^\partial, p_h^0) + a(\kappa, c; \phi^\partial, p_h^\partial), \end{cases} \tag{20}$$

$$\widehat{F}(f, g; \phi^0, \phi^\partial) \equiv \begin{cases} F(f, g; \phi^0), \\ F(f, g; \phi^\partial). \end{cases} \tag{21}$$

See Listing 5 in Appendix A.1 for a UFL formulation of (19).

Listing 1 here displays abridged code using Slate to form the system in (15). The syntax of Slate is designed to enable one to naturally and concisely express the

```
1  A = Tensor(a)
2  F = Tensor(F)
3  S = A.block((1, 1)) - A.block((1, 0)) * A.block((0, 0)).inv * A.block((0, 1))
4  E = F.block((1,)) - A.block((1, 0)) * A.block((0, 0)).inv * F.block((0,))
5  pd = Function(Vd)                  # Function to store the result: P_∂
6  Smat = assemble(S, bcs=[...])      # Assemble and solve: SP_∂ = E
7  Evec = assemble(E)
8  solve(Smat, pd, Evec)
9
10 po = Function(Vd)                  # Function to store the result: P_0
11 PD = AssembledVector(pd)           # Coefficient vector for P_∂
12 assemble(A.block((0, 0)).inv * (F.block((0,)) - A.block((0, 1)) * PD), po)
```

LISTING 1

Slate expressions (highlighted in orange) for the Schur complement system in (15), as well as the local reconstruction of the interior unknowns described by (16). Here, the finite element forms `a` *and* `F` *in lines 1 and 2 are previously defined UFL expressions for (20), (21) respectively. In this example, the argument field index for the interior element space is denoted by 0, and 1 for the facet element space.*

desired algebraic expressions. Calling `block` on the Slate tensors associated with the UFL forms produces the corresponding blocks of the partitioned matrix and vector (lines 3–4). Equation (15) is the only system which requires a global solve (line 8). Strong boundary conditions should be provided during the Schur complement operator assembly (line 6). Line 12 displays the Slate expression for locally assembling the interior degrees of freedom by directly inverting the local matrices of A_{00} onto the relevant right-hand side.

2.2.2. Hybridized mixed methods. Here we demonstrate the use of Slate to facilitate the implementation of hybridized finite element methods. We use the mixed formulation of the second-order system (9)–(11) as our example:

$$\mu \boldsymbol{u} + \nabla p = 0 \quad \text{in } \Omega, \tag{22}$$

$$\nabla \cdot \boldsymbol{u} + cp = f \quad \text{in } \Omega, \tag{23}$$

$$p = p_0 \quad \text{on } \partial\Omega_D, \tag{24}$$

$$\boldsymbol{u} \cdot \boldsymbol{n} = g \quad \text{on } \partial\Omega_N, \tag{25}$$

where $\mu = \kappa^{-1}$ and $\boldsymbol{u} = -\kappa\nabla p$. Traditional mixed methods seek a solution $(\boldsymbol{u}_h, p_h)$ in the finite dimensional spaces

$$\boldsymbol{U}_h = \{\boldsymbol{w} \in \boldsymbol{H}(\mathrm{div}; \Omega) : \boldsymbol{w}|_K \in \boldsymbol{U}(K), \forall K \in \mathcal{T}_h, \boldsymbol{w} \cdot \boldsymbol{n} = g \text{ on } \partial\Omega_N\}, \tag{26}$$

$$V_h = \{\phi \in L^2(\Omega) : \phi|_K \in V(K), \forall K \in \mathcal{T}_h\} \tag{27}$$

respectively. The space $\boldsymbol{U}_h$ consists of $\boldsymbol{H}(\mathrm{div})$-conforming piecewise vector polynomials, where choices of $\boldsymbol{U}(K)$ typically include the Raviart-Thomas (RT), Brezzi-Douglas-Marini (BDM), or Brezzi-Douglas-Fortin-Marini (BDFM) elements [12, 13, 43, 48]. The implementation of $\boldsymbol{H}(\mathrm{div})$ finite element spaces within a code-generation framework is discussed in [49].

The mixed finite element formulation of (22)–(25) reads as follows: find $(\boldsymbol{u}_h, p_h) \in \boldsymbol{U}_h \times V_h$ satisfying

$$(\boldsymbol{w}, \mu\boldsymbol{u}_h)_{\mathcal{T}_h} - (\nabla \cdot \boldsymbol{w}, p_h)_{\mathcal{T}_h} = -\langle \boldsymbol{w} \cdot \boldsymbol{n}, p_0 \rangle_{\partial\Omega_D} \tag{28}$$

$$(\phi, \nabla \cdot \boldsymbol{u}_h)_{\mathcal{T}_h} + (\phi, cp_h)_{\mathcal{T}_h} = (\phi, f)_{\mathcal{T}_h} \tag{29}$$

for all $(\boldsymbol{w}, \phi) \in \boldsymbol{U}_{h,0} \times V_h$, where $\boldsymbol{U}_{h,0}$ is the space of functions in $\boldsymbol{U}_h$ whose normal components vanish on $\partial\Omega_N$. With U and P denoting the vector of degrees of freedom for $\boldsymbol{u}_h$ and p_h respectively, computing the solution of (28)–(29) requires solving the saddle point system:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{Bmatrix} U \\ P \end{Bmatrix} = \begin{Bmatrix} G \\ F \end{Bmatrix}. \tag{30}$$

Methods to efficiently invert such systems include $\boldsymbol{H}(\mathrm{div})$-multigrid [4] (requiring complex overlapping-Schwarz smoothers), global Schur complement factorizations (which require an approximation to the inverse of the elliptic Schur complement operator), or auxiliary space multigrid [32]. Here, we focus on a solution method using *hybridization* [3, 9, 11, 12, 14].

For the hybridized mixed method, the discrete solution spaces are V_h for p_h and $\widehat{\boldsymbol{U}}_h$ for $\boldsymbol{u}_h$, where

$$\widehat{\boldsymbol{U}}_h = \{\boldsymbol{w} \in [L^2(\Omega)]^n : \boldsymbol{w}|_K \in \boldsymbol{U}(K), \forall K \in \mathcal{T}_h\}. \tag{31}$$

The vector finite element space $\widehat{\boldsymbol{U}}_h$ is a subspace of $[L^2(\Omega)]^n$ consisting of local $\boldsymbol{H}(\mathrm{div})$ functions, but normal components are no longer continuous on $\partial\mathcal{T}_h$. Lagrange multipliers are introduced as an auxiliary trace variable in the function space:

$$M_h = \{\gamma \in L^2(\partial\mathcal{T}_h) : \gamma|_e \in M(e), \forall e \in \partial\mathcal{T}_h\}, \tag{32}$$

where $M(e)$ denotes a polynomial space defined on each facet e.

Deriving the hybridized mixed system is accomplished by standard integration by parts over each element K. The trace function λ_h is introduced in surface integrals approximating p_h on elemental boundaries. An additional constraint equation is added to close the system. The hybridized mixed formulation reads: find $(\boldsymbol{u}_h, p_h, \lambda_h) \in \widehat{\boldsymbol{U}}_h \times V_h \times M_h$ such that

$$a(\mu, c; (\boldsymbol{w}, \phi), (\boldsymbol{u}_h, p_h)) + b(\boldsymbol{w}, \lambda_h) = F(p_0, f; \boldsymbol{w}, \phi) \tag{33}$$

$$b(\boldsymbol{u}_h, \gamma) = G(g; \gamma), \tag{34}$$

for all $(\boldsymbol{w}, v, \gamma) \in \widehat{\boldsymbol{U}}_h \times V_h \times M_{h,0}$, where $M_{h,0}$ denotes the space of traces vanishing on $\partial\Omega_D$, and the finite element forms are defined as:

$$a(\mu, c; (\boldsymbol{w}, \phi), (\boldsymbol{u}_h, p_h)) = \begin{cases} (\boldsymbol{w}, \mu\boldsymbol{u}_h)_{\mathcal{T}_h} - (\nabla \cdot \boldsymbol{w}, p_h)_{\mathcal{T}_h} \\ (\phi, \nabla \cdot \boldsymbol{u}_h)_{\mathcal{T}_h} + (\phi, c u_h)_{\mathcal{T}_h} \end{cases} \tag{35}$$

$$b(\boldsymbol{u}_h, \gamma) = \langle \gamma, [\![\boldsymbol{u}_h]\!] \rangle_{\partial\mathcal{T}_h \setminus \partial\Omega_D} \tag{36}$$

$$F(p_0, f; \boldsymbol{w}, \phi) = \begin{cases} -\langle [\![\boldsymbol{w}]\!], p_0 \rangle_{\partial\Omega_D} \\ (\phi, f)_{\mathcal{T}_h} \end{cases} \tag{37}$$

$$G(g; \gamma) = \langle \gamma, g \rangle_{\partial\Omega_N}. \tag{38}$$

We refer the reader to Listing 6 in Appendix A.2 for an example of how one may formulate (35)–(38) in UFL.

Arising from the hybridized-mixed formulation, we have the augmented system of linear equations with the general form:

$$\begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix} \begin{Bmatrix} X \\ \Lambda \end{Bmatrix} = \begin{Bmatrix} F \\ G \end{Bmatrix}, \tag{39}$$

where $X = \begin{bmatrix} U & P \end{bmatrix}^T$ and Λ are the vectors of degrees of freedom for the flux, scalar, and trace unknowns. Equation (39) defines a 3×3 block system which is block-sparse by our choices of $\widehat{\boldsymbol{U}}_h$, V_h, and M_h. If the space of Lagrange multipliers M_h is chosen appropriately, then the solution U in (39), albeit sought a priori in a discontinuous space, will coincide with its $\boldsymbol{H}(\mathrm{div})$-conforming counterpart in (30) via:

$$b(\boldsymbol{u}_h, \gamma) = G(g; \gamma), \text{ for all } \gamma \in M_{h,0}. \tag{40}$$

Equation (40) enforces both continuity of the normal components of $\boldsymbol{u}_h$ across elemental boundaries, as well as the Neumann condition on $\partial\Omega_N$. As a result, the formulations in (33)–(34) and (28)–(29) are mathematically equivalent [3].

The hybridized problem has a number of immediate advantages:

1. By our choice of function spaces, the operator:

$$A = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \tag{41}$$

is block-sparse. As a result, we can simultaneously eliminate U and P by performing element-wise static condensation to (39), producing a significantly smaller problem for Λ only:

$$S\Lambda = E, \tag{42}$$

where S and E are given by

$$S = BA^{-1}B^T \tag{43}$$

$$E = BA^{-1}F - G. \tag{44}$$

2. The matrix S is sparse, symmetric, and positive-definite [18]. Furthermore, S is a discrete elliptic operator and therefore we can solve (42) using standard algebraic multigrid (AMG) preconditioning for elliptic problems [27].
3. Once Λ is computed, both U and P can be recovered locally. Using (41) and noting that the matrix B is of the form $B = \begin{bmatrix} K & 0 \end{bmatrix}$, we can derive elemental systems for U and P:

$$A_{00}U = (G - A_{01}P - K^T\Lambda), \tag{45}$$

$$(A_{11} - A_{10}A_{00}^{-1}A_{01})P = (F - A_{10}A_{00}^{-1}(G + K^T\Lambda)). \tag{46}$$

An expression for the Schur complement can be formed by expanding out the block matrix products and inverse:

$$S = \begin{bmatrix} K & 0 \end{bmatrix} \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix}^{-1} \begin{bmatrix} K^T \\ 0 \end{bmatrix}. \tag{47}$$

However, it can be quite cumbersome in general to write out (47) in terms of individual matrices. Instead, Slate can operate directly on the defined block matrices. See Listing 2 for the corresponding Slate code for the static condensation of (39) and local recovery in (45)–(46).

2.2.3. Hybridized discontinuous Galerkin methods. The hybridized discontinuous Galerkin (HDG) method is a natural extension of its mixed counterpart. The unification of hybridized methods is discussed in [18], and the HDG method was

```
1  # Local element tensors corresponding to the relevant forms
2  A = Tensor(a)
3  B = Tensor(b)
4  F = Tensor(F)
5  G = Tensor(G)
6  S = B * A.inv * B.T
7  E = B * A.inv * F - G
8  lambdah = Function(M)           # Function to store the result: Λ
9  Smat = assemble(S, bcs=[...])  # Assemble and solve: SΛ = E
10 Evec = assemble(E)
11 solve(Smat, lambdah, Evec)
12
13 ph = Function(V)                # Function to store the result: P
14 uh = Function(U)                # Function to store the result: U
15 Lv = AssembledVector(lambdah)    # Coefficient vector for: Λ
16 A00 = A.block((0, 0))            # Blocks for two-stage reconstruction
17 A01 = A.block((0, 1))
18 A10 = A.block((1, 0))
19 A11 = A.block((1, 1))
20 K = B.block((0, 1))
21 assemble((A11 - A10 * A00.inv * A01).inv * (G - A10 * A00.inv * (F + K.T * Lv)), ph)
22 assemble(A00.inv * (F - A01 * AssembledVector(p_h) - K.T * Lv, uh)
```

LISTING 2

Slate code for the Schur complement system and the local solves for the scalar and flux unknowns. The form arguments `a`, `b`, `F`, *and* `G` *in lines 2–5 correspond to UFL expressions for the forms in (35)–(38). The element-wise assembly of the Schur complement system occurs in lines 9 and 10. Lines 22 and 21 correspond to equations (45) and (46). Note that any vanishing conditions on the trace space should be provided during operator assembly in line 9.*

studied numerically in [20, 21, 37]. Here, we follow [18, 20] and present a specific HDG method, namely the LDG-H method for (28)–(29).

The LDG-H method seeks flux and scalar approximations in the spaces $\boldsymbol{U}_h = [V_h]^n$ and V_h, respectively, with V_h being the space of discontinuous polynomials. Next, we introduce the numerical fluxes:

$$\widehat{\boldsymbol{u}}_h = \boldsymbol{u}_h + \tau\,(p_h - \widehat{p}_h)\,\boldsymbol{n}, \quad \widehat{p}_h = \lambda_h, \tag{48}$$

where $\lambda_h \in M_h$ is a function approximating the trace of p on $\partial\mathcal{T}_h$ and τ is a positive function that may vary on each facet $e \in \partial\mathcal{T}_h$.

The full LDG-H formulation is obtain via integrating by parts, which produces the following problem: find $(\boldsymbol{u}_h, p_h, \lambda_h) \in \boldsymbol{U}_h \times V_h \times M_h$ such that

$$(\boldsymbol{w}, \mu\boldsymbol{u}_h)_{\mathcal{T}_h} - (\nabla\cdot\boldsymbol{w}, p_h)_{\mathcal{T}_h} + \langle[\![\boldsymbol{w}]\!], \lambda_h\rangle_{\partial\mathcal{T}_h} = 0, \tag{49}$$

$$-(\nabla\phi, \boldsymbol{u}_h)_{\mathcal{T}_h} + \langle\phi, [\![\widehat{\boldsymbol{u}}_h]\!]\rangle_{\partial\mathcal{T}_h} + (\phi, cp_h)_{\mathcal{T}_h} = (\phi, f)_{\mathcal{T}_h} \tag{50}$$

$$\langle\gamma, [\![\widehat{\boldsymbol{u}}_h]\!]\rangle_{\partial\mathcal{T}_h\setminus\partial\Omega_D} = \langle\gamma, g\rangle_{\partial\Omega_N}, \tag{51}$$

$$\langle\gamma, \lambda_h\rangle_{\partial\Omega_D} = \langle\gamma, p_0\rangle_{\partial\Omega_D}, \tag{52}$$

for all $(\boldsymbol{w}, \phi, \gamma) \in \boldsymbol{U}_h \times V_h \times M_h$. Equation (51) enforces continuity of the numerical flux $\widehat{\boldsymbol{u}}$ on the mesh skeleton, and (52) weakly applies the Dirichlet condition. Note that the choice of τ has a significant influence on the theoretical convergence rates in the computed solutions, as discussed in [20]. We elaborate further and numerically demonstrate the effects of τ in subsection 4.2. See Appendix A.3 for a UFL formulation of (49)–(52).

The LDG-H system has more degrees of freedom than its hybridized mixed counterpart, but its matrix-form exhibits the same useful properties as (39). The matrix system arising from (49)–(52) has the general form:

$$\begin{bmatrix} A & B & K \\ C & D & L \\ M & N & Q \end{bmatrix} \begin{Bmatrix} U \\ P \\ \Lambda \end{Bmatrix} = \begin{Bmatrix} G \\ F \\ H \end{Bmatrix}. \tag{53}$$

By our function spaces, the upper-left 2×2 block has block-sparse structure and we may eliminate U and P element-wise to obtain a reduced system for Λ:

$$S\Lambda = E, \tag{54}$$

where

$$S = Q - \begin{bmatrix} M & N \end{bmatrix} \begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} \begin{bmatrix} K \\ L \end{bmatrix}, \tag{55}$$

$$E = H - \begin{bmatrix} M & N \end{bmatrix} \begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} \begin{Bmatrix} G \\ F \end{Bmatrix}. \tag{56}$$

Equation (54) shares the same properties as the Schur complement system of (42) for hybridized mixed method.

```
# Element tensors defining the local 3-by-3 block system
R = Tensor(a)
Z = Tensor(L)

# Extracting blocks of R and Z to form the reduced system
Q = R.block((2, 2))
H = Z.block((2,))
S = Q - R.block((2, (0, 1))) * R.block(((0, 1), (0, 1))).inv * R.block(((0, 1), 2))
E = H - R.block((2, (0, 1))) * R.block(((0, 1), (0, 1))).inv * Z.block(((0, 1),))
lambdah = Function(M)           # Function to store the result: Λ
Smat = assemble(S, bcs=[...])  # Assemble and solve: SΛ = E
Evec = assemble(E)
solve(Smat, lambdah, Evec)
```

LISTING 3

Slate code for assembling (54) via extracting the relevant mixed blocks of the full 3×3 block system, given UFL expressions `a`, `L` *for (49)–(51). Arguments of the mixed space* $\boldsymbol{U}_h \times V_h \times M_h$ *are indexed by 0, 1, and 2 respectively. As with the hybrid-mixed method, vanishing conditions on the trace variables should be provided as boundary conditions during operator assembly.*

Listing 3 displays the Slate code for assembling Schur complement system in (54). Providing the appropriate argument indices via the `block` method on Slate tensors extracts the corresponding mixed blocks in equations (55) and (56). Note that it is indeed possible to rewrite the Slate expressions purely in terms of the individual blocks of (53). However, this is quite cumbersome and the expressions are far more intelligible when written in this way.

As before, we may reconstruct U and P locally once Λ is computed. This can be accomplished in two stages, first computing P locally by solving:

$$(D - CA^{-1}B)P = G - CA^{-1}F - (L - CA^{-1}K)\Lambda, \tag{57}$$

followed by:

$$AU = F - BP - K\Lambda. \tag{58}$$

See Listing 4 for the Slate expressions corresponding to equations (57) and (58). The local systems are constructed from the individual blocks of the system in (54), following similarly from the hybridized mixed method.

```
A = R.block((0, 0))                # Individual blocks for the local solves
B = R.block((0, 1))
C = R.block((1, 0))
D = R.block((1, 1))
K = R.block((0, 2))
L = R.block((1, 2))
F = Z.block((0,))
G = Z.block((1,))

Sd = D - C * A.inv * B              # Intermediate expressions
Sl = L - C * A.inv * K
Lv = AssembledVector(lambdah)       # Coefficient vector for Λ
ph = Function(V)                    # Function to store the result: P
uh = Function(U)                    # Function to store the result: U

# Solve for P and U
assemble(Sd.inv * (G - C * A.inv * F - Sl * Lv), ph)
assemble(A.inv * (F - B * AssembledVector(ph) - K * Lv), uh)
```

LISTING 4
Slate code for locally reconstructing the scalar and flux unknowns of the LDG-H method.

2.2.4. Local post-processing. Local post-processing techniques for the construction of superconvergent approximations were discussed within the context of mixed methods in [3, 10, 52], and discontinuous Galerkin methods in [20, 21]. Here, we present two post-processing techniques for producing local solutions of higher approximation order. The Slate code follows naturally from previous discussions in subsection 2.2.2 and subsection 2.2.3, using the standard set of operations on local tensors. Listings and further discussion on these techniques can be found in Appendix B.

Post-processing of the scalar solution. There are number of post-processing techniques for enhancing the accuracy of the scalar approximation of the hybridized mixed method and its DG variant. We present a modified version of the procedure presented by Stenberg in [52], and highlighted within the context of hybridizing eigenproblems in [19].

Let $\mathcal{P}_k(K)$ denote a polynomial space of degree $\leq k$ on an element $K \in \mathcal{T}_h$. Then for a given pair of computed solutions $\boldsymbol{u}_h, p_h$ of the hybridized methods, we define the post-processed scalar $p_h^\star \in \mathcal{P}_{k+1}(K)$ as the unique solution of the local problem:

$$(\nabla w, \nabla p_h^\star)_K = -(\nabla w, \kappa^{-1}\boldsymbol{u}_h)_K, \tag{59}$$

$$(v, p_h^\star)_K = (v, p_h)_K, \tag{60}$$

for all $(w, v) \in \mathcal{P}_{k+1}^{\perp,l}(K) \times \mathcal{P}_l(K)$, $0 \leq l \leq k$. Here, the space $\mathcal{P}_{k+1}^{\perp,l}(K)$ denotes the L^2-orthogonal complement of $\mathcal{P}_l(K)$. This post-processing method directly uses the definition of the flux $\boldsymbol{u}_h = -\kappa\nabla p_h$ to construct the local problem above. In practice, the space $\mathcal{P}_{k+1}^{\perp,l}(K)$ may be constructed using an orthogonal hierarchical basis, and solving (59)–(60) amounts to inverting a local symmetric positive definite system.

At the time of this work, neither FEniCS nor Firedrake supports the construction of such a finite element basis. However, we can use a standard Lagrange multiplier

method (for example, see [38, §4.7]) to enforce the orthogonality constraint. The local problem then becomes the mixed system: find $(p_h^\star, \psi) \in \mathcal{P}_{k+1}(K) \times \mathcal{P}_l(K)$ such that

$$(\nabla w, \nabla p_h^\star)_K + (w, \psi)_K = -(\nabla w, \kappa^{-1}\boldsymbol{u}_h)_K, \tag{61}$$

$$(\phi, p_h^\star)_K = (\phi, p_h)_K, \tag{62}$$

for all $(w, \phi) \in \mathcal{P}_{k+1}(K) \times \mathcal{P}_l(K)$, $0 \leq l \leq k$. The local problems (61)–(62) and (59)–(60) are equivalent, with the Lagrange multiplier ψ enforcing orthogonality of test functions in $\mathcal{P}_{k+1}(K)$ with functions in $\mathcal{P}_l(K)$.

It has been shown that the post-processing method above produces a new approximation which superconvergences at a rate of $k+2$ for the hybridized mixed methods [3, 19, 52]. For the LDG-H method, similar results are proven when $\tau = \mathcal{O}(1)$ and $\tau = \mathcal{O}(h)$, but only $k+1$ convergence is achieved when $\tau = \mathcal{O}(1/h)$ [20, 21]. We demonstrate this numerically in section 4.

Post-processing of the flux. For the post-processing of the flux for the LDG-H method, we use the numerical trace $\widehat{\boldsymbol{u}}_h$ from (48). The technique we outline here follows from [21], and produces a new flux with better conservation properties.

Let $\mathcal{T}_h$ be a mesh consisting of simplices. On each element $K \in \mathcal{T}_h$, we define a new function $\boldsymbol{u}_h^\star$ to be the unique element of the local Raviart-Thomas space $[\mathcal{P}_k(K)]^n + \boldsymbol{x}\mathcal{P}_k(K)$ satisfying

$$(\boldsymbol{r}, \boldsymbol{u}_h^\star)_K = (\boldsymbol{r}, \boldsymbol{u}_h)_K, \tag{63}$$

$$\langle \mu, \boldsymbol{u}_h^\star \cdot \boldsymbol{n} \rangle_e = \langle \mu, \widehat{\boldsymbol{u}}_h \cdot \boldsymbol{n} \rangle_e, \tag{64}$$

for all $(\boldsymbol{r}, \mu) \in [\mathcal{P}_{k-1}(K)]^n \times \mathcal{P}_k(e)$, $e \in \partial K$. This local problem produces a new flux $\boldsymbol{u}_h^\star$ with the following properties:

1. $\boldsymbol{u}_h^\star$ converges at the *same* rate as $\boldsymbol{u}_h$ for all choices of τ producing a solvable system for (49)–(51). However,
2. $\boldsymbol{u}_h^\star \in \boldsymbol{H}(\mathrm{div}; \Omega)$. That is,

$$[\![\boldsymbol{u}_h^\star]\!]_e = 0 \text{ for all } e \in \mathcal{E}_h^\circ. \tag{65}$$

3. Furthermore, the divergence of $\boldsymbol{u}_h^\star$ convergences at a rate of $k+1$.

We demonstrate the rates of of convergence of the post-processed flux for varying choices of τ in section 4.

3. Static condensation is a preconditioner. The solver abstraction of Firedrake follows the PETSc formalism for solving and preconditioning linear systems. PETSc [6, 7] is a fully-featured library which provides access to a large range of its own and third party implementations of solver algorithms. For example, PETSc provides access to direct solvers like LU, with the possibility of utilizing external packages like MUMPS [2], or UMFPACK [25]. PETSc also provides many robust block preconditioners via the `fieldsplit` option [15] for mixed problems. Together with the Python interface with PETSc via petsc4py [24], composing solvers in Firedrake becomes a straightforward process.

Following PETSc's abstraction for preconditioning linear systems, one can think of (left) preconditioning the matrix equation in residual form:

$$r = r(A, b) \equiv b - Ax = 0 \tag{66}$$

by an operator $\boldsymbol{P}$ (which may not necessarily be linear) as a transformation into an *equivalent* system of the form

$$\boldsymbol{P}r = \boldsymbol{P}(b - Ax) = 0. \tag{67}$$

Given a current iterate x_i, the residual at the i-th iteration is simply the expression $r_i \equiv b - Ax_i$. $\boldsymbol{P}$ acts on the residual to produce an approximation to the error $\epsilon_i \equiv x - x_i$. The solver converges if

$$\|r_m\|_2 = \|\boldsymbol{P} r_{m-1}(A,b)\|_2 \leq \max(\epsilon_{\text{rtol}}\|b\|_2, \epsilon_{\text{atol}}), \tag{68}$$

where the norms are measured in the standard vector 2-norm. In the case that $\boldsymbol{P}$ is an application of an exact inverse, the residual is converted into an exact (up to numerical round-off) residual. In this way, direct methods are expressed in PETSc's Krylov method interface by specifying "preconditioner only" as a `ksp_type` (e.g. `-ksp_type preonly -pc_type lu`) [6].

We denote the application of particular Krylov method for the linear system (66) as $\mathcal{K}_x(r(A,b))$. Upon preconditioning the system via $\boldsymbol{P}$ as in (67), we write

$$\mathcal{K}_x(\boldsymbol{P} r(A,b)). \tag{69}$$

If (69) is solved directly via the application of A^{-1}, then $\boldsymbol{P}r(A,b) = A^{-1}b - Ix$, where I is the identity matrix. So, we have that $\mathcal{K}_x(\boldsymbol{P}r(A,b)) = \mathcal{K}_x(r(I, A^{-1}b))$ produces the exact solution of (66) in a single iteration of $\mathcal{K}$.

3.1. Interfacing with PETSc via custom preconditioners. The implementation of preconditioners for these systems requires manipulation not of assembled matrices, but rather their symbolic representation. To do this, we use the preconditioning infrastructure developed in [36], which gives preconditioners written in Python access to the symbolic problem description. We manipulate this appropriately and provide operators assembled from Slate expressions to PETSc for further algebraic preconditioning. Using this technique, we have developed a static condensation solver for continuous Galerkin methods, a hybridized mixed context for $\boldsymbol{H}(\text{div}) \times L^2$ mixed problems, and a generic static condensation interface for hybridized systems (both hybridized mixed and HDG methods). The advantage of writing even the latter as a preconditioner is the ability to switch out the solution scheme for the system, even when nested inside a larger set of coupled equations.

3.1.1. A static condensation preconditioner. The Firedrake preconditioner `SCCG` is a generic static condensation interface whose `setUp` method takes an H^1 problem, and uses the Slate DSL to write and form the condensed system for the facet degrees of freedom. The preconditioning operator $\boldsymbol{P}$ here is the inverse of the Schur-complement factorization of the left-hand side operator in (14):

$$\boldsymbol{P} = \begin{bmatrix} I & A_{00}^{-1}A_{0\partial} \\ 0 & I \end{bmatrix} \begin{bmatrix} A_{00}^{-1} & 0 \\ 0 & S^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ A_{\partial 0}A_{00}^{-1} & I \end{bmatrix}, \tag{70}$$

where S is the Schur-complement left-hand side of (15).

Normally when using the `fieldsplit` options together with a Schur complement preconditioner in PETSc, two Krylov solvers are needed to invert both S and the upper-left block A_{00}. However, only S requires an iterative method; the matrix A_{00} has block-diagonal structure and therefore recovery of the interior unknowns is performed element-wise using a Slate description of the local problem (see (16)).

The incoming co-vector b is defined on the H^1 finite element space, and is partitioned into interior (b_0) and facet (b_∂) sections. The right-hand side for the reduced system is assembled pointwise via:

$$\tilde{b} = b_\partial - A_{\partial 0}A_{00}^{-1}b_0, \tag{71}$$

where $A_{\partial 0}A_{00}^{-1}b_0$ is assembled locally using Slate. This preconditioner requires a single global solver for the facet unknowns x_∂:

$$\mathcal{K}_{x_\partial}(\boldsymbol{P}_1 r(S, \tilde{b})) \tag{72}$$

where $\boldsymbol{P}_1$ is another preconditioning operator for the system associated with S.

This static condensation procedure composes naturally with PETSc in that additional solver specifications and preconditioning for (72) is completely configurable through standard PETSc options. Once both the interior and facet solutions are solved for, the results are placed back into an H^1 residual function $(x_\partial, x_0) \to x$. The residual of the problem is checked to verify convergence of the solver.

3.1.2. Preconditioning mixed methods via hybridization. The preconditioner `HybridizationPC` takes an $\boldsymbol{H}(\mathrm{div}) \times L^2$ system and forms the hybridized problem by manipulating the UFL objects representing the discretized PDE. This includes replacing argument spaces with their discontinuous counterparts, introducing test functions on an appropriate trace space, and providing operators assembled from Slate expressions. This is an alternative to using standard `fieldsplit` options.

More precisely, let $Ax = b$, $x \leftarrow (x_U, x_P)$, where x_U and x_P are the flux and scalar unknowns respectively, be the mixed saddle point problem. Then this preconditioner replaces $Ax = b$ with the augmented system:

$$\begin{bmatrix} \widetilde{A} & B^T \\ B & 0 \end{bmatrix} \begin{Bmatrix} \widetilde{x} \\ x_\lambda \end{Bmatrix} = \begin{Bmatrix} \widetilde{b} \\ b_\lambda \end{Bmatrix} \tag{73}$$

where $\widetilde{b} \leftarrow (\widetilde{b}_U, b_P)$, $\widetilde{b}_U, b_P$ are the right-hand sides for the flux and scalar equations respectively, and $\widetilde{\cdot}$ indicates modified matrices and co-vectors with discontinuous functions. Here, B is an off-diagonal block matrix containing the contributions of the multipliers, and $\widetilde{x} \leftarrow (\widetilde{x}_U, x_P)$ are the new discontinuous unknowns.

The preconditioning operator for (73) has the form:

$$\widetilde{\boldsymbol{P}} = \begin{bmatrix} I & \widetilde{A}^{-1}B^T \\ 0 & I \end{bmatrix} \begin{bmatrix} \widetilde{A}^{-1} & 0 \\ 0 & S^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ B\widetilde{A}^{-1} & I \end{bmatrix}, \tag{74}$$

where S is the Schur complement matrix $S = -B\widetilde{A}^{-1}B^T$. Note that (74) is actually a 3×3 block factorization, with the two fields for the flux and scalar unknowns associated with the block $\widetilde{A}$. The only globally coupled system that needs an iterative solver is the reduced problem for the Lagrange multipliers:

$$\mathcal{K}_{x_\lambda}(\boldsymbol{P}_1 r(S, E)), \tag{75}$$

where E is the reduced right-hand size of the hybridized mixed system. Once x_λ is computed, the flux and scalar fields are reconstructed element-wise via solving the local problems in (45) and (46).

Since the flux is constructed in a discontinuous space $\widehat{\boldsymbol{U}}_h$, we must project the computed solution into $\boldsymbol{U}_h \subset \boldsymbol{H}(\mathrm{div})$. This can be done cheaply via local facet averaging. The resulting solution is then $x \leftarrow (\Pi_{\mathrm{div}}\widetilde{x}_U, x_P)$, where x_P, $\Pi_{\mathrm{div}}\widetilde{x}_U$ are the scalar and projected flux solutions respectively. This ensures the residual for the original mixed is properly evaluated to test for convergence.

We note here that assembly of the right-hand side E in (75) requires special attention. The situation we are given is that we have $b_U = b_U(\boldsymbol{w})$ for $\boldsymbol{w} \in \boldsymbol{U}_h$, but require $\widetilde{b}_U(\widehat{\boldsymbol{w}})$ for $\widehat{\boldsymbol{w}} \in \widehat{\boldsymbol{U}}_h$. For consistency, we also require for any $\boldsymbol{w} \in \boldsymbol{U}_h$ that

$$\widetilde{b}_U(\boldsymbol{w}) = b_U(\boldsymbol{w}). \tag{76}$$

We can construct such a $\widetilde{b}_U$ satisfying (76) using the local definition:

$$\widetilde{b}_U(\widehat{\boldsymbol{\psi}}_i) = \frac{b_U(\widehat{\boldsymbol{\psi}}_i)}{N_i}, \quad \widehat{\boldsymbol{\psi}}_i \in \widehat{\boldsymbol{U}}_h, \tag{77}$$

where N_i is the number of cells that the degree of freedom corresponding to the basis function $\boldsymbol{\psi}_i \in \boldsymbol{U}_h$ touches. For $\boldsymbol{H}(\mathrm{div})$-elements, $N_i = 2$ for facet nodes, and 1 otherwise. By construction of space $\widehat{\boldsymbol{U}}_h$, we have for $\boldsymbol{\psi}_i \in \boldsymbol{U}_h$:

$$\boldsymbol{\psi}_i = \begin{cases} \widehat{\boldsymbol{\psi}}_i^+ + \widehat{\boldsymbol{\psi}}_i^- & \boldsymbol{\psi}_i \text{ associated with a facet node,} \\ \widehat{\boldsymbol{\psi}}_i & \boldsymbol{\psi}_i \text{ associated with an interior node,} \end{cases} \tag{78}$$

where $\widehat{\boldsymbol{\psi}}_i, \widehat{\boldsymbol{\psi}}_i^{\pm} \in \widehat{\boldsymbol{U}}_h$, and $\widehat{\boldsymbol{\psi}}_i^{\pm}$ are functions corresponding to the positive and negative restrictions associated with the i-th facet node.[3] Using (77) and (78), we can verify that our construction of $\widetilde{b}_U$ satisfies (76).

3.1.3. A static condensation interface for hybridization. It may be the case that one formulates the hybridized system of three fields directly and wishes to only solve the problem via a Schur complement method. In this case, we created a static condensation interface `HybridSCPC` which takes a hybridized linear system $Ax = b$, where A and b have the general block form described in (53), and forms the Schur complement system for the trace variables using operators generated from Slate expressions. As before, a single global solve is required for the trace unknowns, and local reconstructions are performed to retrieve the discontinuous flux and scalar solutions.

4. Numerical results. We now present results utilizing the Slate DSL and our custom Firedrake preconditioners for a set of test problems. All numerical studies conducted in this paper were performed on a 32-core workstation, with each core being a Intel E5-2450v2 (Xeon) CPU at 2.5GHz. Solver configurations for the test problems in this section are provided in Appendix E.

4.1. A three-dimensional positive-definite Helmholtz equation. We consider solving a positive definite Helmholtz equation in three dimensions on a regular unit-cube mesh consisting of tetrahedral elements. We solve a simple pure-Neumann problem:

$$-\nabla \cdot \nabla p + p = f \text{ in } \Omega = [0,1]^3, \quad \nabla p \cdot \boldsymbol{n} = g \text{ on } \partial\Omega, \tag{79}$$

where f and g are chosen such that the analytic solution is simply the trigonometric function $p(x, y) = \cos(3\pi x)\cos(3\pi y)\cos(3\pi z)$. We solve (79) using our static condensation preconditioner in subsection 3.1.1.

We perform a convergence study using Lagrange elements of degree 4, 5, 6, and 7 on refined tetrahedral meshes. The meshes are divided into 2^r cells in all spatial directions. The elliptic problem on the facet nodes is inverted using conjugate gradients, preconditioned with using hypre's boomerAMG algebraic multigrid method (AMG) [5]. To ensure an accurate result, we specify a relative tolerance of 10^{-13}. Figure 2 shows the convergence of the L^2 error as the mesh is refined.

[3] These are the two "broken" parts of $\boldsymbol{\psi}_i$ on a particular facet connecting two elements. That is, for two adjacent cells, a basis function in $\boldsymbol{U}_h$ for a particular facet node can be decomposed into two basis functions in $\widehat{\boldsymbol{U}}_h$ defined on their respective sides of the facet.

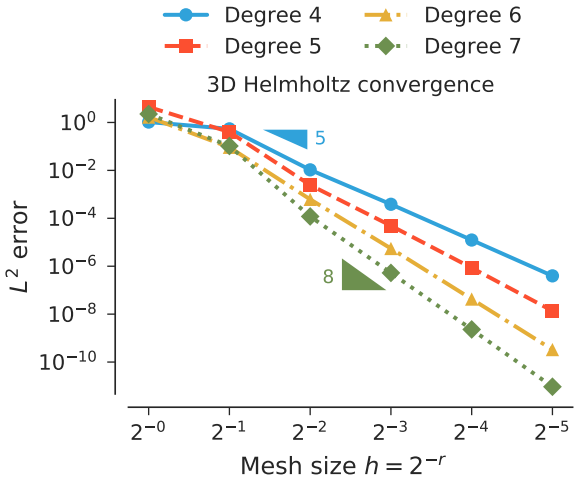


FIG. 2. *Error convergence rates of the continuous Galerkin method for degrees 4, 5, 6, and 7. The solution was obtained using the static condensation method. Slope markers are added to assist in comprehension. For all degrees, we achieve the expected $k+1$ rate of convergence in the L^2 errors between the computed solution and the analytic result.*

Currently, we compute local inverses directly without exploiting the use of factorization methods, like QR or LU with pivoting. Therefore, it is likely that our implementation is introducing small errors during operator assembly. To investigate this, we measure the reduction in the problem residual after a single application of our static condensation preconditioner. While reductions slowly decrease as the resolutions increase, we observe significant reductions overall (the residual is reduced down to 10^{-12} for all degrees at the highest resolution). Isolated calculations can be found in Appendix C.1.

We remark here that the setup cost of this preconditioner is quite high, since it requires the assembly of dense inverses of high-degree element tensors. In three-dimensions especially, the overall algorithmic intensity grows rapidly, and how we compute local matrix inverses has a significant impact. This suggests a future direction for optimizations in Firedrake. A more thorough performance analysis is needed to draw further conclusions on when using static condensation is advantageous for problems of this type. See [44] for a more comprehensive study.

4.2. Hybridized methods for the Poisson equation. We seek a solution to the Dirichlet problem for the Poisson equation as a first-order system:

$$\boldsymbol{u} + \nabla p = 0 \text{ in } \Omega = [0,1]^2, \quad \nabla \cdot \boldsymbol{u} = f \text{ in } \Omega, \quad p = p_0 \text{ on } \partial\Omega, \tag{80}$$

where f and p_0 are chosen so that the analytic solution is the sinusoid $p(x,y) = \sin(\pi x)\sin(\pi y)$ and its negative gradient. We solve this problem by hybridizing the mixed formulation of (80), and employ our static condensation preconditioner in 3.1.3. The trace variables are obtained with a direct method: LU with MUMPS providing the factorization algorithms [2].

The standard error estimates for the RT and BDM mixed methods are well-known [3, 12, 16, 26, 48]. To verify our implementation of the hybridized-mixed preconditioner, we run a convergence study using the setup above. Additionally, we compute post-processed scalar solutions using the method described in (61)–(62).

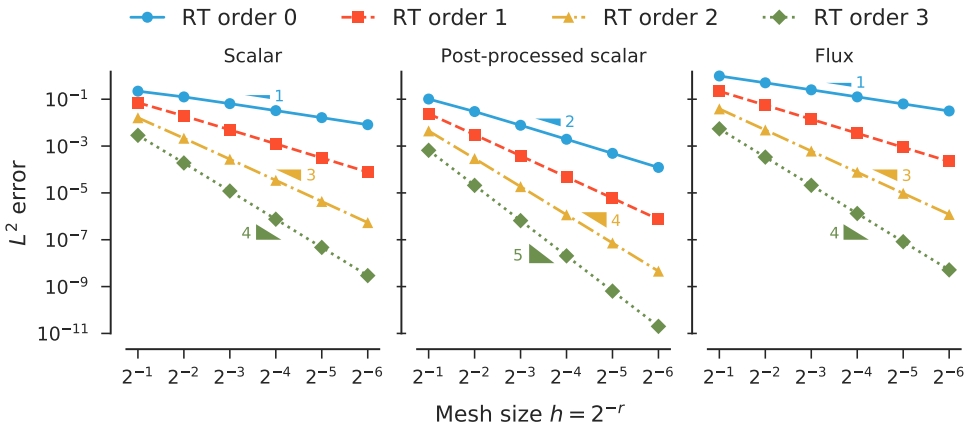


FIG. 3. *Error convergence rates of the hybridized RT method of orders 0, 1, 2, and 3. We observe the expected theoretical rates for the scalar and flux solutions of the standard RT method. Additionally, we can clearly see the effects of post-processing the scalar solution, yielding superconvergent results.*

Figure 3 displays the convergence rates for the hybridized RT method. We repeat the same study for the BDM method on simplices and RT method on quadrilaterals, which can be found in Appendix C.2.

We repeat this same study for the LDG-H method with varying choices of τ in order to examine how τ influences the convergence rates. The expected theoretical rates for the LDG-H method given a particular order of τ is discussed in [20]. In all our experiments, we use the post-processing methods described in subsection 2.2.4 to produce approximations $p_h^\star$ and $\boldsymbol{u}_h^\star$. We consider three cases: $\tau = 1$, $\tau = \frac{1}{h}$, and $\tau = h$. The convergence rates for each LDG-H method for approximation degrees 1, 2, and 3 are summarized in Appendix C.3.

For $\tau = 1$ and $\tau = h$, we expect the post-processed scalar $p_h^\star$ to superconverge at a rate of $k+2$ for $k \geq 1$. This superconvergence is lost when $\tau = \frac{1}{h}$. No matter the choice of τ, the post-processed flux $\boldsymbol{u}_h^\star$ converges at the same rate as $\boldsymbol{u}_h$. Furthermore, the errors in the post-processed flux are actually smaller than $\boldsymbol{u}_h$, indicating a slight increase in accuracy while maintaining the convergence order of the flux. Similar behavior was observed in [21]. We isolate the cases for $\tau = h$ and $\tau = \frac{1}{h}$ and display the convergence rates in Figure 4.

4.3. A nonlinear method for the shallow water equations. A primary motivator for our interest in hybridized methods revolves around developing efficient solvers for problems in geophysical flows. In this last section, we present some results integrating the shallow water equations on the sphere using Test Case 5 (the mountain test case) from [53]. We use the framework of compatible finite elements [22, 23].

4.3.1. Semi-implicit solver. We start with the vector-invariant rotating nonlinear shallow water system defined on a two-dimensional spherical surface Ω embed-

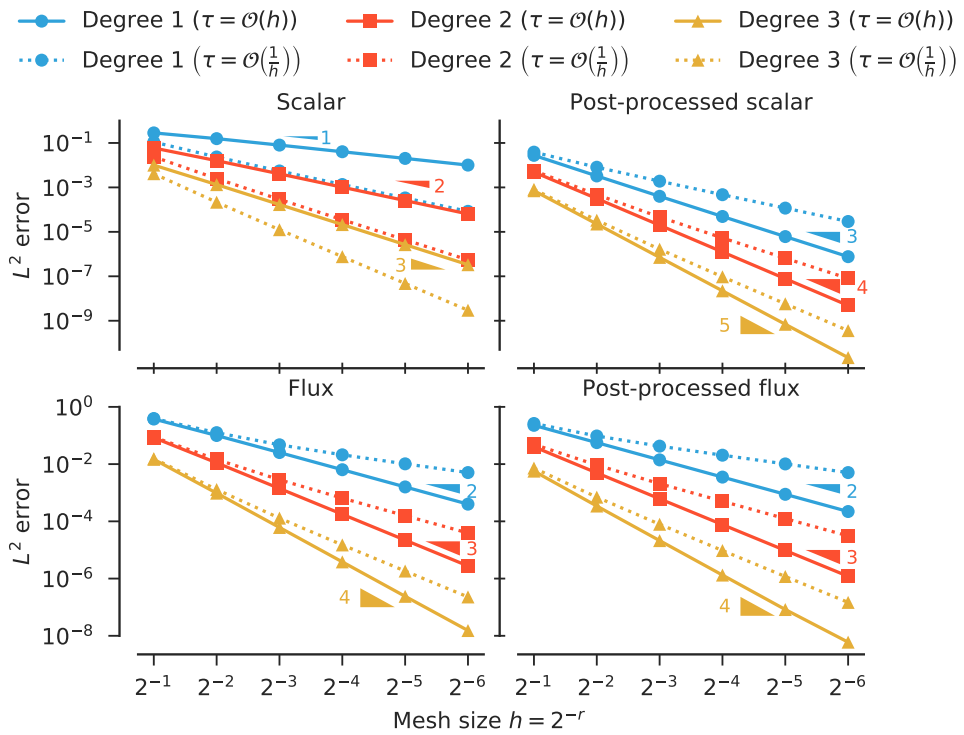


FIG. 4. *Error convergence rates for the LDG-H method with $\tau = h$ and $\tau = \frac{1}{h}$. This illustrates the sensitivity of this discretization subject to appropriate choices of stabilization parameter τ. We see no change in the convergence rates between the scalar and post-processed scalar solutions when $\tau = \frac{1}{h}$. Superconvergence is achieved when taking $\tau = h$. The post-processed flux rates in both cases match the rates of the unprocessed flux.*

ded in $\mathbb{R}^3$:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \left(\nabla^\perp \cdot \boldsymbol{u} + f\right)\boldsymbol{u}^\perp + \nabla\left(g\left(D + b\right) + \frac{1}{2}|\boldsymbol{u}|^2\right) = 0, \tag{81}$$

$$\frac{\partial D}{\partial t} + \nabla \cdot (\boldsymbol{u} D) = 0, \tag{82}$$

where $\boldsymbol{u}$ is the fluid velocity, D is the depth field, f is the Coriolis parameter, g is the acceleration due to gravity, b is the bottom topography, and $(\cdot)^\perp \equiv \hat{\boldsymbol{k}} \times \cdot$, with $\hat{\boldsymbol{k}}$ being the unit normal to the surface Ω. For some tessellation, $\mathcal{T}_h$, our semi-discrete mixed method for (81)–(82) seeks approximations $(\boldsymbol{u}_h, D_h) \in \boldsymbol{U}_h \times V_h \subset \boldsymbol{H}(\mathrm{div}) \times L^2$

satisfying:

$$\left(\boldsymbol{w}, \frac{\partial \boldsymbol{u}_h}{\partial t}\right)_{\mathcal{T}_h} - \left(\nabla^\perp\left(\boldsymbol{w}\cdot\boldsymbol{u}_h^\perp\right), \boldsymbol{u}_h^\perp\right)_{\mathcal{T}_h} + \left(\boldsymbol{w}, f\boldsymbol{u}_h^\perp\right)_{\mathcal{T}_h} + \langle[\![\boldsymbol{n}^\perp\boldsymbol{w}\cdot\boldsymbol{u}_h^\perp]\!], \tilde{\boldsymbol{u}}_h^\perp\rangle_{\partial\mathcal{T}_h} - \left(\nabla\cdot\boldsymbol{w}, g\left(D_h+b\right)+\frac{1}{2}|\boldsymbol{u}_h|^2\right)_{\mathcal{T}_h} = 0, \tag{83}$$

$$\left(\phi, \frac{\partial D_h}{\partial t}\right)_{\mathcal{T}_h} - \left(\nabla\phi, \boldsymbol{u}_h D_h\right)_{\mathcal{T}_h} + \langle[\![\phi\boldsymbol{u}_h]\!], \tilde{D}_h\rangle_{\partial\mathcal{T}_h}, = 0, \tag{84}$$

for all $(\boldsymbol{w}, \phi) \in \boldsymbol{U}_h \times V_h$, where $\tilde{\cdot}$ indicates that the value of the function should be taken from the upwind side of each facet.

The timestepping scheme follows a prediction-correction/Picard iteration semi-implicit approach, where predictive values of the relevant fields are determined at each time-step, and corrective updates are generated by solving the linear system (linearized about a state of rest) for $(\Delta\boldsymbol{u}_h, \Delta D_h) \in \boldsymbol{U}_h \times V_h$, given by

$$(\boldsymbol{w}, \Delta\boldsymbol{u}_h)_{\mathcal{T}_h} + \frac{\Delta t}{2}\left(\boldsymbol{w}, f\Delta\boldsymbol{u}_h^\perp\right)_{\mathcal{T}_h} - \frac{\Delta t}{2}\left(\nabla\cdot\boldsymbol{w}, g\Delta D_h\right)_{\mathcal{T}_h} = -R_{\boldsymbol{u}}[\boldsymbol{u}_h^{n+1}, D_h^{n+1}; \boldsymbol{w}], \tag{85}$$

$$(\phi, \Delta D_h)_{\mathcal{T}_h} + \frac{H\Delta t}{2}(\phi, \nabla\cdot\Delta\boldsymbol{u}_h)_{\mathcal{T}_h} = -R_D[\boldsymbol{u}_h^{n+1}, D_h^{n+1}; \phi], \tag{86}$$

for all $(\boldsymbol{w}, \phi) \in \boldsymbol{U}_h \times V_h$, where H is the mean layer depth, and $R_{\boldsymbol{u}}$ and R_D are residual linear forms that vanish when $\boldsymbol{u}_h^{n+1}$ and D_h^{n+1} are solutions to the implicit midpoint rule time discretisation of the semi-discrete equations. The solution $(\Delta\boldsymbol{u}_h, \Delta D_h)$ is then used to update the iterative values of $\boldsymbol{u}_h^{n+1}$ and D_h^{n+1} according to $(\boldsymbol{u}_h^{n+1}, D_h^{n+1}) \leftarrow (\boldsymbol{u}_h^{n+1} + \Delta\boldsymbol{u}_h, D_h^{n+1} + \Delta D_h)$, having initially chosen $(\boldsymbol{u}_h^{n+1}, D_h^{n+1}) = (\boldsymbol{u}_h^n, D_h^n)$. For the interested reader, Appendix D outlines the procedure in more detail.

For each Picard iteration, solving the implicit linear system requires the solution of the indefinite saddle-point system:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}\begin{Bmatrix} \Delta U \\ \Delta D \end{Bmatrix} = \begin{Bmatrix} R_{\boldsymbol{u}} \\ R_D \end{Bmatrix}. \tag{87}$$

In staggered finite difference models, the standard approach for solving (87) is to neglect the Coriolis term and eliminate the velocity unknowns ΔU to obtain a discrete elliptic problem for which smoothers like Richardson or relaxation methods are convergent. This is more problematic in the compatible finite element framework, since A is not diagonal. Instead, we use the preconditioner described in subsection 3.1.2 to form the hybridized problem and eliminate both ΔU and ΔD locally.

4.3.2. Atmospheric flow over a mountain (Williamson, test case 5). We use the finite element spaces $BDM_2 \times DG_1$, on an octahedral sphere mesh of radius $R_\odot = 6.37122 \times 10^6$m. The test case is initialized with depth and velocity fields in geostrophic balance:

$$D(\boldsymbol{x}; 0) = D_0 - \left(R_\odot\Omega_\odot u_0 + \frac{u_0^2}{2}\right)\frac{z^2}{gR_\odot^2}, \quad \boldsymbol{u}(\boldsymbol{x}; 0) = \frac{u_0}{R_\odot}\left\{-y \quad x \quad 0\right\}^T, \tag{88}$$

where $\Omega_\odot = 7.292 \times 10^{-5}\text{s}^{-1}$ is the angular rotation rate of the Earth, $u_0 = 20\text{ms}^{-1}$ is the maximum amplitude of the zonal wind, and $D_0 = 5960\text{m}$. An isolated mountain is placed with its center at latitude $\frac{\pi}{6}$ and longitude $-\frac{\pi}{2}$, given by the expression: $b = 2000\text{m}\left(1 - \frac{r}{R_0}\right)$, where $R_0 = \frac{\pi}{9}$ and $r^2 = \min[R_0^2, (x_{\text{lat}} - \frac{\phi}{6})^2 + (x_{\text{lon}} + \frac{\pi}{2})^2]$.

We run for a fixed number of time-steps, and compare the simulation time using two difference solver configurations for the implicit linear system. First, we use an approximated Schur complement preconditioner for GMRES of the form:

$$\begin{bmatrix} I & 0 \\ CA^{-1} & I \end{bmatrix} \begin{bmatrix} A^{-1} & 0 \\ 0 & \widehat{S}^{-1} \end{bmatrix} \begin{bmatrix} I & -A^{-1}B \\ 0 & I \end{bmatrix},$$

where $\widehat{S} = D - C\widehat{A}^{-1}B$, and $\widehat{A} = \text{diag}(A)$ is a diagonal approximation to the velocity mass matrix. The approximate Schur complement $\widehat{S}$ is inverted using conjugate gradients preconditioned with PETSc's smoothed aggregation multigrid (GAMG). The solver for $\widehat{S}$ is configured to reduce the preconditioned residual norm by a factor of 10^8. A^{-1} is computed approximately using a single application of incomplete LU.

TABLE 1

PETSc performance summary for a 20 time-step run of the Williamson 5 test case. We use a midpoint method for time integrating the equations. The simulation was run using 16-cores on the workstation described above. The mesh was generated by refining a regular octahedron seven times, giving 131072 cells. The DOF count of the problem is just over 1 million (1,376,256 total unknowns for velocity and depth). We do not display timings associated with writing data or problem setup. Instead, we display the total time to complete the profile runs. We observe significant speed-up when using hybridization over preconditioned GMRES.

Simulation stage	**Hybridization**		**Approx. Schur**		**Speedup factor**
	Time (s)		Time (s)		
Total run-time	3.463e+02		1.623e+03		4.687
	Time (s)	% total	Time (s)	% total	
Advection	1.2326e+02	35.6 %	1.2445e+02	7.7 %	
Linear solver	2.0588e+02	59.4 %	1.4812e+03	91.3 %	7.194

Next, we use a "preonly" application of our hybridization preconditioner. We use GAMG-preconditioned conjugate gradients with a relative residual tolerance of 10^{-8} to invert the Lagrange multiplier system. Table 1 displays the timings for our profiling run using the two configurations mentioned above. We expect the overall simulation time to be dominated by the linear solve for the correction updates. When using hybridization, we observe a significant reduction in time spent during the implicit solve stage compared to the preconditioned GMRES approach, as well as overall simulation time.

We present more specific timings in Table 2. In particular, we examine the overall time spent during setup and application of each preconditioner. For both preconditioners, the times spent during setup are comparable. However, the application of hybridization is nearly an order of magnitude faster. This is primarily because we reduce the number of required "outer" iterations to zero. Hybridization is performing an exact Schur complement factorization of the original problem. We also measure the reductions in the true-residual of the linear system (87). Our hybridized method reduces the residual down by a factor of 10^8 on average, which coincides with the specified relative tolerance for the Krylov method on the trace system.

TABLE 2

Setup, application, and solve times for both solver configurations. These are cumulative times in each stage of the two preconditioners throughout the entire profile run. We display the average iteration count (rounded to nearest the integer) for both the outer GMRES and the inner CG+GAMG Krylov solves. The solve time for the preconditioned GMRES configuration consists of numerically inverting both A and $\widehat{S}$ for each GMRES iteration. For the hybridization preconditioner, this time includes both inverting the Schur complement system for the trace solution, and element-wise reconstructions.

Configuration	**PC setup**	**PC apply**	**Krylov solve**	**Outer its.**	**Inner its.**
	Time (s)	Time (s)	Time (s)		
Hybridization	8.6671e+01	1.7799e+02	1.9148e+02	None	5
Approx. Schur	5.8638e+01	1.4232e+03	1.4666e+03	10	5

5. Conclusions and future outlooks. We have presented Slate, and shown how this language can be used to create concise mathematical representations of localized linear algebra on the tensors corresponding to finite element forms. We have shown how this DSL can be used in tandem with UFL in Firedrake to implement solution approaches making use of static condensation, hybridization, and localized post-processing. In particular, this framework alleviates much of the difficulty in implementing such methods within intricate numerical code. In this way, Slate can be used to help enable the rapid development and exploration of new hybridization and static condensation techniques.

Our approach to preconditioner design revolves around its composable nature, in that these Slate-based implementations can be seamlessly incorporated into complicated solution schemes. In particular, there is current research in the design of dynamical cores for numerical weather prediction using implementations of hybridization and static condensation with Slate [8, 51]. The performance of such methods for geophysical flows is a subject of ongoing research. Slate is a developing project and continues to expand in its features, including extending compiler technology to generate code suitable for vectorization and implementation on different computer architectures.

Acknowledgments. The authors would like to thank Miklós Homolya for his helpful input on developing Slate as a language and suggestions for designing its compiler.

Code availability. We cite archives of the exact software versions used to produce the results in this paper. The Slate-based preconditioners from subsection 3.1 can be found in [61]. For all components of the Firedrake project, we used the recent versions: COFFEE [54], FIAT [55], FInAT [56], Firedrake [57], PETSc [58], petsc4py [59], PyOP2 [60], TSFC [63], and UFL [64]. The numerical experiments, including raw data and plotting scripts, are available as [62].

Appendix A. UFL listings. This section presents listings of UFL [1] formulations of the problems covered in subsection 2.2. The purpose here is to assist in comprehension and provide further context, in light of the examples using the Slate DSL, where UFL code has been omitted for brevity. We outline each method in the order in which they are presented.

A.1. Partitioned H^1-formulation. The first example covered starts with an H^1 weak formulation of an elliptic problem, as specified in (9)–(11): given an appropriate finite element space $V_h \subset H^1$, find $p_h \in V_h$ such that

$$a(\kappa, c; p_h, \phi) = F(f, g; \phi), \text{ for all } v \in V_{h,0}, \tag{89}$$

where

$$a(\kappa, c; p_h, \phi) = (\nabla\phi, \kappa\nabla p_h)_{\mathcal{T}_h} + (\phi, cp_h)_{\mathcal{T}_h} \tag{90}$$

$$F(f, g; \phi) = (\phi, f)_{\mathcal{T}_h} - \langle \phi, g\rangle_{\partial\Omega_N}. \tag{91}$$

The static condensation technique for problems of this type is accomplished by partitioning the coefficient vector for p_h into coefficients for basis functions interior to each cell, and on the trace (or skeleton) of the mesh.

Since UFL strictly works from the weak formulation of PDEs, and code-generation systems like Firedrake [46] and FEniCS [39] work directly from UFL, we must perform the partitioning at this level. That is, we introduce finite element spaces which possess the right construction of basis functions for interior and boundary degrees of freedom respectively. We start by introducing the spaces V_h^0 and V_h^∂, as defined in (17) and (18) respectively. These are function spaces corresponding to functions in V_h *restricted* to element interiors (V_h^0) and boundaries (V_h^∂).

Note that we have the following decomposition $V_h = V_h^0 \oplus V_h^\partial$. Then it follows that $p_h \in V_h$ has the form $p_h = p_h^0 + p_h^\partial$. We perform a similar splitting of test and trial functions of V_h to produce the rewritten form of (89) as a *mixed* system: find $(p_h^0, p_h^\partial) \in V_h^0 \times V_h^\partial$ such that

$$\widehat{a}\left(\kappa, c; \left(\phi^0, \phi^\partial\right), \left(p_h^0, p_h^\partial\right)\right) = \widehat{F}(f, g; \phi^0, \phi^\partial) \tag{92}$$

holds for all test functions $(\phi^0, \phi^\partial) \in V_h^0 \times V_{h,0}^\partial$, where

$$\widehat{a}\left(\kappa, c; \left(\phi^0, \phi^\partial\right), \left(p_h^0, p_h^\partial\right)\right) \equiv \begin{cases} a(\kappa, c; \phi^0, p_h^0) + a(\kappa, c; \phi^0, p_h^\partial), \\ a(\kappa, c; \phi^\partial, p_h^0) + a(\kappa, c; \phi^\partial, p_h^\partial), \end{cases} \tag{93}$$

$$\widehat{F}(f, g; \phi^0, \phi^\partial) \equiv \begin{cases} F(f, g; \phi^0), \\ F(f, g; \phi^\partial). \end{cases} \tag{94}$$

Equation (92) is the starting point for statically condensing the resulting matrix system, as described in (14) of the manuscript. Using UFL, it is straightforward to express the forms in (92). Since the forms are linear in test and trial arguments, we can simplify the UFL code. See Listing 5 for a complete UFL description of the partitioned problem.

A.2. Hybrid-mixed formulation. To form the hybridized-mixed formulation of the first-order system:

$$\mu\boldsymbol{u} + \nabla p = 0 \quad \text{in } \Omega, \tag{95}$$

$$\nabla \cdot \boldsymbol{u} + cp = f \quad \text{in } \Omega, \tag{96}$$

$$p = p_0 \quad \text{on } \partial\Omega_D, \tag{97}$$

$$\boldsymbol{u} \cdot \boldsymbol{n} = g \quad \text{on } \partial\Omega_N, \tag{98}$$

we introduce the finite element spaces $\widehat{\boldsymbol{U}}_h$, V_h, and M_h, where V_h is the space of discontinuous piece-wise polynomials, and $\widehat{\boldsymbol{U}}_h$, M_h are as defined in (31) and (32). With $\boldsymbol{u}_h$, p_h and λ_h being approximations to $\boldsymbol{u}$, p, and $p|_{\partial\mathcal{T}_h}$ respectively, the hybridized-mixed method reads: find $(\boldsymbol{u}_h, p_h, \lambda_h) \in \widehat{\boldsymbol{U}}_h \times V_h \times M_h$ such that

$$(\boldsymbol{w}, \mu\boldsymbol{u}_h)_{\mathcal{T}_h} - (\nabla \cdot \boldsymbol{w}, p_h)_{\mathcal{T}_h} + \langle [\![\boldsymbol{w}]\!], \lambda_h\rangle_{\partial\mathcal{T}_h \setminus \partial\Omega_D} = -\langle [\![\boldsymbol{w}]\!], p_0\rangle_{\partial\Omega_D} \tag{99}$$

$$(\phi, \nabla \cdot \boldsymbol{u}_h)_{\mathcal{T}_h} + (\phi, cu_h)_{\mathcal{T}_h} = (\phi, f)_{\mathcal{T}_h} \tag{100}$$

$$\langle \gamma, [\![\boldsymbol{u}_h]\!]\rangle_{\partial\mathcal{T}_h \setminus \partial\Omega_D} = \langle \gamma, g\rangle_{\partial\Omega_N} \tag{101}$$

```
1  # Decomposed Lagrange finite element space
2  element = FiniteElement("Lagrange", triangle, degree)
3  V = FunctionSpace(mesh, element)
4  Vo = FunctionSpace(mesh, element["interior"])
5  Vd = FunctionSpace(mesh, element["facet"])
6  W = Vo * Vd
7
8  # Test/trial arguments and coefficients
9  vo, vd = TestFunctions(W)
10 uo, ud = TrialFunctions(W)
11 c = Coefficient(V)
12 f = Coefficient(V)
13 g = Coefficient(V)
14 kappa = Coefficient(V)
15 u = uo + ud
16 v = vo + vd
17
18 # Finite element forms
19 a = dot(grad(v), kappa*grad(u))*dx + inner(v, c*u)*dx
20 L = inner(v, f)*dx - inner(v, g)*ds
```

LISTING 5

UFL code for the partitioned Helmholtz system as described in (19) of subsection 2.2.1. We treat this system as a mixed problem, with a mixed space consisting of the restrict Lagrange element spaces (lines 2–6). Test and trial functions are defined on each component space, and we write the finite element forms as functions of the sum of each component argument. Expanding of the forms in lines 19 and 20 occurs during form compilation.

for all $(\boldsymbol{w}, v, \gamma) \in \widehat{\boldsymbol{U}}_h \times V_h \times M_{h,0}$, where $M_{h,0}$ denotes the space of traces vanishing on $\partial\Omega_D$.

Expressing the relevant forms for (99)–(101) in UFL can be done by defining the expressions:

$$a(\mu, c; (\boldsymbol{w}, \phi), (\boldsymbol{u}_h, p_h)) = \begin{cases} (\boldsymbol{w}, \mu\boldsymbol{u}_h)_{\mathcal{T}_h} - (\nabla \cdot \boldsymbol{w}, p_h)_{\mathcal{T}_h} \\ (\phi, \nabla \cdot \boldsymbol{u}_h)_{\mathcal{T}_h} + (\phi, c u_h)_{\mathcal{T}_h} \end{cases} \tag{102}$$

$$b(\boldsymbol{u}_h, \gamma) = \langle \gamma, [\![\boldsymbol{u}_h]\!] \rangle_{\partial\mathcal{T}_h \setminus \partial\Omega_D} \tag{103}$$

$$F(p_0, f; \boldsymbol{w}, \phi) = \begin{cases} -\langle [\![\boldsymbol{w}]\!], p_0 \rangle_{\partial\Omega_D} \\ (\phi, f)_{\mathcal{T}_h} \end{cases} \tag{104}$$

$$G(g; \gamma) = \langle \gamma, g \rangle_{\partial\Omega_N}. \tag{105}$$

Here, we note that the facet integrals in (99) and (101) have elemental tensors which are simply the transposes of each other. We therefore need only write an expression for one of these to write the necessary Slate expressions, as discussed in subsection 2.2.2. Listing 6 summarizes how these forms can be expressed in UFL.

A.3. LDG-H formulation. The LDG-H method for solving the system (95)–(97) is formed by first introducing the numerical fluxes

$$\widehat{\boldsymbol{u}}_h = \boldsymbol{u}_h + \tau\,(p_h - \widehat{p}_h)\,\boldsymbol{n}, \tag{106}$$

$$\widehat{p}_h = \lambda_h \in M_h, \tag{107}$$

where $\boldsymbol{n}$ is the outward normal to elemental boundaries, and τ is some positive-valued function that may vary on facets of the mesh. We note that the definitions in (106)–(107) are specific to the LDG-H method.

```
# Discontinuous RT, DG, and trace finite element spaces
RT = FiniteElement("RT", triangle, 1)
U = FunctionSpace(mesh, BrokenElement(RT))
V = FunctionSpace(mesh, FiniteElement("DG", triangle, 0))
M = FunctionSpace(mesh, FiniteElement("HDiv Trace", triangle, 0))
W = U * V

# Test/trial arguments and coefficients
u, p = TrialFunctions(W)
w, phi = TestFunctions(W)
gamma = TestFunction(M)
n = FacetNormal(mesh)
p0 = Coefficient(V)
c = Coefficient(V)
f = Coefficient(V)
g = Coefficient(V)
mu = Coefficient(V)

# Finite element forms
a = dot(w, mu*u)*dx - div(w)*p*dx + phi*div(u)*dx + phi*c*p*dx
b = gamma*jump(u, n=n)*dS + gamma*dot(u, n)*ds(2)
F = phi*f*dx - dot(w, n)*p0*ds(1)
G = gamma*g*ds(2)
```

LISTING 6

UFL code defining the relevant finite element forms for the hybrdized mixed method. This example features the hybridized lowest order RT method. We arbitrarily label the Dirichlet boundary by an integer 1, and 2 for the Neumann portion using subdomain notation in UFL.

Using L^2-finite element spaces for $\boldsymbol{u}$ and p, the full LDG-H formulation reads: find $(\boldsymbol{u}_h, p_h, \lambda_h) \in \boldsymbol{U}_h \times V_h \times M_h$ such that

$$(\boldsymbol{w}, \mu \boldsymbol{u}_h)_{\mathcal{T}_h} - (\nabla \cdot \boldsymbol{w}, p_h)_{\mathcal{T}_h} + \langle [\![\boldsymbol{w}]\!], \lambda_h \rangle_{\partial \mathcal{T}_h} = 0, \tag{108}$$

$$-(\nabla \phi, \boldsymbol{u}_h)_{\mathcal{T}_h} + \langle \phi, [\![\widehat{\boldsymbol{u}}_h]\!] \rangle_{\partial \mathcal{T}_h} + (\phi, c p_h)_{\mathcal{T}_h} = (\phi, f)_{\mathcal{T}_h} \tag{109}$$

$$\langle \gamma, [\![\widehat{\boldsymbol{u}}_h]\!] \rangle_{\partial \mathcal{T}_h \setminus \partial \Omega_D} = \langle \gamma, g \rangle_{\partial \Omega_N}, \tag{110}$$

$$\langle \gamma, \lambda_h \rangle_{\partial \Omega_D} = \langle \gamma, p_0 \rangle_{\partial \Omega_D}, \tag{111}$$

for all $(\boldsymbol{w}, \phi, \gamma) \in \boldsymbol{U}_h \times V_h \times M_h$. Notice that all boundary conditions are applied naturally in the finite element forms. Writing (108)–(111) in UFL is shown in Listing 7.

Appendix B. Post-processing. Here we focus on how Slate can be used to define local systems arising from the post-processing techniques described in subsection 2.2.4. To establish our context, suppose we solved the following three-field hybridized finite element problem defined on a tessellation $\mathcal{T}_h$: find $(\boldsymbol{u}_h, p_h, \lambda_h) \in \boldsymbol{U}_h \times V_h \times M_h$ such that

$$a(\boldsymbol{w}, \phi, \gamma; \boldsymbol{u}_h, p_h, \lambda_h) = L(\boldsymbol{w}, \phi, \gamma) \tag{112}$$

for all $(\boldsymbol{w}, \phi, \gamma) \in \boldsymbol{U}_h \times V_h \times M_h$. We remain ambiguous whether this is a hybridized-mixed or an LDG-H formulation, described in subsection 2.2.2 and subsection 2.2.3 respectively, as most of our discussion applies to both cases. We will be specific when necessary. We also assume (112) arises within the context of mixed first-order systems. That is, $\boldsymbol{u}_h, p_h, \lambda_h$ are approximations to $\boldsymbol{u} = -\kappa \nabla p$, p, and $p|_{\partial \mathcal{T}_h}$ respectively.

```
# Equal-order discontinuous Galerkin spaces
element = FiniteElement("Discontinuous Lagrange", triangle, degree)
U = VectorFunctionSpace(mesh, element)
V = FunctionSpace(mesh, element)
M = FunctionSpace(mesh, FiniteElement("HDiv Trace", triangle, degree))
W = U * V * M

# Test/trial arguments and coefficients
u, p, lambdar = TrialFunctions(W)
w, phi, gamma = TestFunctions(W)
n = FacetNormal(mesh)
p0 = Coefficient(V)
c = Coefficient(V)
f = Coefficient(V)
g = Coefficient(V)
mu = Coefficient(V)

# LDG-H fluxes, stability parameter, and forms. Note that
# τ can also be a function of the facet area.
tau = Constant(1)
phat = lambdar
uhat = u + tau*(p - phat)*n
a = dot(w, mu*u)*dx - div(w)*p*dx - dot(grad(phi), u)*dx + phi*c*p*dx +\
2*avg(lambdar*dot(w, n))*dS + lambdar*dot(w, n)*ds +\
2*avg(phi*dot(uhat, n))*dS + phi*dot(uhat, n)*ds +\
2*avg(gamma*dot(uhat, n))*dS + gamma*dot(uhat, n)*ds(2) +\
gamma*lambdar*ds(1))
L = phi*f*dx + gamma*g*ds(2) + gamma*p0*ds(1)
```

LISTING 7

UFL code defining the LDG-H method in (49)–(51). The choice of function spaces are the traditional discontinuous Lagrange spaces of equal order. While τ in this listing is simply 1, it is indeed possible to take τ to be a function of the facet area of the mesh.

B.1. Post-processing of p_h. Post-processing techniques for the scalar solution p_h of (112) has been studied extensively, dating back to Stenberg in [52]. Several post-processing methods have been constructed which yields a more accurate scalar approximation, however we will simply focus on the method described in (59)–(60), which was also featured in [19].

Assuming that V_h is constructed in the usual way, with elemental functions in $\mathcal{P}_k(K)$, $K \in \mathcal{T}_h$, then (59)–(60) produces a new approximation $p_h^\star$ which is locally $\mathcal{P}_{k+1}(K)$. As discussed in subsection 2.2.4, we rewrite (59)–(60) using a Lagrange multiplier method: find $(p_h^\star, \psi) \in \mathcal{P}_{k+1}(K) \times \mathcal{P}_l(K)$ such that

$$(\nabla w, \nabla p_h^\star)_K + (w, \psi)_K = -(\nabla w, \kappa^{-1}\boldsymbol{u}_h)_K, \tag{113}$$

$$(\phi, p_h^\star)_K = (\phi, p_h)_K, \tag{114}$$

for all $(w, \phi) \in \mathcal{P}_{k+1}(K) \times \mathcal{P}_l(K)$, $0 \leq l \leq k$.

Explicitly in terms of linear algebra, (113)–(114) produces the local system:

$$\begin{bmatrix} A & C^T \\ C & 0 \end{bmatrix} \begin{Bmatrix} P^\star \\ \Psi \end{Bmatrix} = \begin{Bmatrix} U \\ P \end{Bmatrix}, \tag{115}$$

where $P^\star$ and Ψ are the vectors of degrees of freedom for $p_h^\star$ and ψ respectively. Since ψ only acts as a Lagrange multiplier to enforce an orthogonality constraint on functions in $\mathcal{P}_{k+1}(K)$ with $P_l(K)$, we only need to return $P^\star$ once (115) is solved

in each element. This post-processing method works for both hybridized-mixed and HDG-type problems. We demonstrate this numerically in both subsection 4.2 and Appendix C.

Expressing (113)–(114) in UFL [1] requires slight modifications. UFL currently can only express finite element problems globally over the problem domain. However, this is not an issue. To ensure locality of the post-processing problem, we may simply use global discontinuous-Galerkin spaces and construct the relevant finite element forms in the usual way using standard UFL notation. Listing 8 displays the UFL code for expressing the system in (113)–(114), including the Slate tensors for solving the local systems. These are solved locally by directly inverting the operator in (115) onto the right-hand side.

```
1  # Solve the hybridized problem for u_h, p_h, λ_h
2  U = FunctionSpace(mesh, ...)
3  V = FunctionSpace(mesh, ...)
4  M = FunctionSpace(mesh, ...)
5  W = U * V * M
6  a = ...
7  L = ...
8  w = Function(W)
9  solve(a == L, w, ...)
10 u_h, p_h, lambda_h = w.split()
11
12 # Define finite element spaces for the post-processing problem:
13 # l should be an integer with 0 ≤ l ≤ k,
14 # where k is the approximation degree of the scalar solution.
15 k = ...
16 l = ...
17 PP = FunctionSpace(mesh, "DG", k + 1)
18 Pl = FunctionSpace(mesh, "DG", l)
19 Wpp = PP * Pl
20
21 # Define elemental matrices/vectors for the local problems
22 pp, psi = TrialFunctions(Wpp)
23 w, phi = TestFunctions(Wpp)
24
25 a_pp = inner(grad(w), grad(pp))*dx + inner(w, psi)*dx +\
26        inner(phi, pp)*dx
27 L_pp = -inner(grad(w), mu*u_h)*dx +\
28        inner(phi, p_h)*dx
29 A = Tensor(a_pp)
30 b = Tensor(L_pp)
31
32 pstar = Function(PP)         # Function to store the result of p_h^*
33 assemble((A.inv * b).block((0,)), pstar)
```

Listing 8

UFL and Slate code for writing out the scalar post-processing problem in (113)–(114). The resuling local systems are solved by directly inverting the full matrix onto the right-hand side, shown in line 33. We return only the block corresponding to the post-processed solution. Slate objects and syntax are highlighted in orange.

B.2. Post-processing of $\boldsymbol{u}_h$. We follow the discussion in subsection 2.2.4 concerning post-processing of the flux. We restrict our attention to a mesh $\mathcal{T}_h$ consisting of simplicial elements K. Moreover, the method described here uses the numerical flux $\widehat{\boldsymbol{u}}_h$ of the LDG-H method, defined as a function of $\boldsymbol{u}_h$, p_h, and λ_h. As a result, this discussion applies specifically to the LDG-H method, summarized in subsection 2.2.3.

The post-processing method described in (63)–(64) defines a new flux $\boldsymbol{u}_h^\star$ which is locally a unique element of the Raviart-Thomas space: $RT_k(K) = [\mathcal{P}_k(K)]^n + \boldsymbol{x}\mathcal{P}_k(K)$. To summarize here, we wish to compute $\boldsymbol{u}_h^\star \in RT_k(K)$ for all K such that

$$(\boldsymbol{r}, \boldsymbol{u}_h^\star)_K = (\boldsymbol{r}, \boldsymbol{u}_h)_K, \tag{116}$$

$$\langle \gamma, \boldsymbol{u}_h^\star \cdot \boldsymbol{n} \rangle_e = \langle \gamma, \widehat{\boldsymbol{u}}_h \cdot \boldsymbol{n} \rangle_e, \tag{117}$$

for all $(\boldsymbol{r}, \gamma) \in [\mathcal{P}_{k-1}(K)]^n \times \mathcal{P}_k(e)$, $e \in \partial K$. Equation (116)–(117) produces a locally square system of equations for the degrees of freedom for the new flux approximation:

$$KU^\star = R. \tag{118}$$

The new flux $\boldsymbol{u}_h^\star$, while it converges at the same rate as $\boldsymbol{u}_h$, possesses better conservation properties. Namely, $\boldsymbol{u}_h^\star \in \boldsymbol{H}(\mathrm{div}; \mathcal{T}_h)$ in the sense that its interior jump is zero:

$$[\![\boldsymbol{u}_h^\star]\!]_e = 0 \text{ for all } e \in \mathcal{E}_h^\circ. \tag{119}$$

Furthermore, its divergence converges at a rate of $k+1$. See Appendix C.2 and Appendix C.3 for our numerical study.

Similarly to the scalar post-processing method, its representation and implementation in UFL using Slate is achieved by writing a global discontinuous formulation of (116)–(117), using a discontinuous Raviart-Thomas space for the post-processed flux to ensure locality.

```
34 # Define the numerical flux using the computed trace, scalar and flux:
35 n = FacetNormal(mesh)
36 uhat = u_h + tau*(p_h - lambda_h)*n
37
38 # Define finite element spaces for the post-processing problem:
39 RTd = FunctionSpace(mesh, "DRT", k + 1)
40 DG = VectorFunctionSpace(mesh, "DG", k - 1)
41 UU = DG * M                    # Reuse the trace space from before
42
43 # Define elemental matrices/vectors for the local problems
44 up = TrialFunction(RTd)
45 r, gamma = TestFunctions(UU)
46
47 a_up = inner(r, up)*dx +\
48        gamma('+')*jump(up, n=n)*dS + gamma*dot(up, n)*ds
49 L_up = inner(r, u_h)*dx +\
50        gamma('+')*jump(uhat, n=n)*dS + gamma*dot(uhat, n)*ds
51 K = Tensor(a_up)
52 R = Tensor(L_up)
53
54 ustar = Function(RTd)          # Function to store the result of u_h^*
55 assemble(K.inv * R, ustar)
```

LISTING 9

UFL and Slate code for writing out the flux post-processing problem in (116)–(117). The resuling local systems are solved by directly inverting the full matrix onto the right-hand side, shown in line 55.

We remark here, since Slate designed to be a general-purpose tool for local finite element systems, many more post-processing schemes are possible which are not covered in this paper. The methods presented in subsection 2.2.4 were simply chosen

to complement our numerical results. There is ongoing work pertaining to the implementation and studying the effects of various methods for more general problems using Slate. We refer the reader to [3, 10, 19, 20, 21, 52] for an overview of some post-processing methods one could explore further in our framework, some of which are covered here.

Appendix C. Convergence studies. We use our previously defined notation in section 1, where $\mathcal{T}_h$ denotes a tessellation of Ω, and $\mathcal{E}_h^\circ$, $\mathcal{E}_h^\partial$ denotes the set of interior and exterior facets respectively.

C.1. H^1-Helmholtz. Here, we present more detailed numerical results to compliment the discussion in subsection 4.1 for the three-dimension Helmholtz problem in (79). The H^1-formulation reads: find $p_h \in V_h$ such that

$$(\nabla\phi, \nabla p_h)_{\mathcal{T}_h} + (\phi, p_h)_{\mathcal{T}_h} = (\phi, f)_{\mathcal{T}_h} \tag{120}$$

for all $\phi \in V_h$.

The mesh $\mathcal{T}_h$ of the unit cube $[0,1]^3$ consists of tetrahedral cells, which is regularly refined during the convergence test. For each approximation degree $k = 4, 5, 6$, and 7, we solve (120) using our static condensation preconditioner from subsection 3.1.1. The complete solver configuration is described in Appendix E.1.

Figure 2 is a visualization of Table 3, which confirms that our solution method, using our implementation of static condensation, is achieving the expected theoretical rates. However, a more telling indicator is how well our preconditioner is reducing the problem residual after application. Table 4 displays the reduction factor for the finest mesh ($r = 5$) consisting of 196608 tetrahedral cells.

As discussed in subsection 4.1, there are a number of possible directions for future optimizations. While we suspect small errors are being introduced during local operator assembly (as a result of inverting high-degree element tensors inefficiently), we still observe significant reductions in the problem residual. This suggests that we are indeed solving (120) sufficiently, but more importantly our preconditioner is performing an exact Schur complement factorization (subject to rounding errors).

C.2. Hybrid-mixed methods. In this section, we complement the results in subsection 4.2 by providing complete convergence studies for the hybridized variants of some mixed methods for the model Dirichlet problem:

$$\boldsymbol{u} + \nabla p = 0 \text{ in } \Omega = [0,1]^2, \quad \nabla \cdot \boldsymbol{u} = f \text{ in } \Omega, \quad p = 0 \text{ on } \partial\Omega, \tag{121}$$

where f is chosen so that the analytic solution is simply $p(x, y) = \sin(\pi x)\sin(\pi y)$.

Recall that, for suitable finite element spaces such that the second Brezzi condition [30] is satisfied, the fully-discrete mixed formulation of the model Poisson problem reads as follows: find $(\boldsymbol{u}_h, p_h) \in \boldsymbol{U}_h \times V_h$ such that

$$(\boldsymbol{w}, \boldsymbol{u}_h)_{\mathcal{T}_h} - (\nabla \cdot \boldsymbol{w}, p_h)_{\mathcal{T}_h} = 0, \tag{122}$$

$$(\phi, \nabla \cdot \boldsymbol{u}_h)_{\mathcal{T}_h} = (\phi, f)_{\mathcal{T}_h}, \tag{123}$$

for all $(\boldsymbol{w}, \phi) \in \boldsymbol{U}_h \times V_h$.

Equation (122)–(123) can be rewritten as a hybrid-mixed method by introducing the discontinuous variant of $\boldsymbol{U}_h$, $\widehat{\boldsymbol{U}}_h$, and an appropriate trace space M_h such that functions in M_h belong to the same polynomial space as $\boldsymbol{u}_h \cdot \boldsymbol{n}|_e$, for all $e \in \partial\mathcal{T}_h$. The

TABLE 3
Convergence history for the H^1 method of the three-dimensional Helmholtz system using our static condensation preconditioner. For each approximation degree k, the mesh consists of regularly-refined tetrahedral cells. We approach the expected convergence rate of $k+1$ in the L^2-errors as we refine the mesh.

H^1 Helmholtz			
k	mesh	$\|p - p_h\|_{L^2(\Omega)} \leq \mathcal{O}(h^{k+1})$	
	r	L^2-error	rate
4	0	1.040e+00	—
	1	5.410e-01	0.943
	2	1.054e-02	5.681
	3	3.847e-04	4.777
	4	1.239e-05	4.957
	5	3.958e-07	4.968
5	0	4.443e+00	—
	1	3.993e-01	3.476
	2	2.436e-03	7.357
	3	4.902e-05	5.635
	4	8.544e-07	5.842
	5	1.369e-08	5.963
6	0	1.676e+00	—
	1	9.205e-02	4.186
	2	6.178e-04	7.219
	3	5.499e-06	6.812
	4	4.262e-08	7.011
	5	3.321e-10	7.004
7	0	2.256e+00	—
	1	1.042e-01	4.437
	2	1.195e-04	9.768
	3	5.260e-07	7.827
	4	2.301e-09	7.837
	5	9.331e-12	7.946

TABLE 4
Residual reductions (maximum over all mesh) for the three-dimensional Helmholtz problem subsection 4.1 using our static condensation preconditioner.

Degree	Reductions (finest mesh): $\frac{\|b-Ax^*\|_2}{\|b\|_2}$
4	1.003e-12
5	1.92e-12
6	3.412e-12
7	6.49e-12

resulting three-field hybridized problem reads: find $(\boldsymbol{u}_h, p_h, \lambda_h) \in \widehat{\boldsymbol{U}}_h \times V_h \times M_h$ such that

$$(\boldsymbol{w}, \boldsymbol{u}_h)_{\mathcal{T}_h} - (\nabla \cdot \boldsymbol{w}, p_h)_{\mathcal{T}_h} + \langle [\![\boldsymbol{w}]\!], \lambda_h \rangle_{\mathcal{E}_h^\circ} = 0, \tag{124}$$

$$(\phi, \nabla \cdot \boldsymbol{u}_h)_{\mathcal{T}_h} = (\phi, f)_{\mathcal{T}_h}, \tag{125}$$

$$\langle \gamma, [\![\boldsymbol{u}_h]\!] \rangle_{\mathcal{E}_h^\circ} = 0, \tag{126}$$

for all $(\boldsymbol{w}, \phi, \gamma) \in \widehat{\boldsymbol{U}}_h \times V_h \times M_{h,0}$, where $M_{h,0}$ is the space of traces vanishing on $\partial\Omega_D$.

For the Raviart-Thomas (RT) and Brezzi-Douglas-Marini (BDM) mixed methods for (122)–(123), the convergence rates are well-known. Its hybridized variants inherit these rates, as both formulations were shown to be equivalent problems [3]. Furthermore, post-processing methods using the computed fields can produce new approximations that converge at accelerated rates. While there are a number or post-processing methods available, we restrict our attention to the procedures described in Appendix B.

For our convergence study on simplicial elements, we use a uniform mesh of $[0,1]^2$ with square sides of size 2^{-r}, which are then divided into two triangles per square cell. The expected theoretical rates for the hybridized RT method of order k are $k+1$ in the L^2-norm for both the scalar and flux approximations [3, 48]. With the post-processing method described in (113)–(114), we expect the new scalar approximation to superconverge at a rate of $k+2$, for all $k \geq 0$.

Table 5 summarizes our findings for the RT-H method on triangular cells. We repeat the same study for the RT method on quadrilateral cells (RTCF-H) in Table 6, which should have the same convergence rates at the RT method on triangles. We can see from the tables that these convergence rates are achieved in full agreement with the theoretical rates.

TABLE 5

Convergence history for the hybridized Raviart-Thomas method of order k on simplices, for $k = 0, 1, 2, 3$. The computed scalar and flux approximations are expected to converge at a rate of $k+1$. With post-processing of the scalar solution, we expect superconvergence at a rate of $k+2$.

RT-H method							
k	mesh	$\|p - p_h\|_{L^2(\Omega)} \leq \mathcal{O}(h^{k+1})$		$\|\boldsymbol{u} - \boldsymbol{u}_h\|_{\boldsymbol{L}^2(\Omega)} \leq \mathcal{O}(h^{k+1})$		$\|p - p_h^\star\|_{L^2(\Omega)} \leq \mathcal{O}(h^{k+2})$	
	r	L^2-error	rate	L^2-error	rate	L^2-error	rate
0	1	2.207e-01	—	9.818e-01	—	1.017e-01	—
	2	1.254e-01	0.815	5.019e-01	0.968	2.975e-02	1.774
	3	6.476e-02	0.954	2.516e-01	0.996	7.724e-03	1.945
	4	3.264e-02	0.988	1.259e-01	0.999	1.949e-03	1.986
	5	1.635e-02	0.997	6.295e-02	1.000	4.885e-04	1.997
	6	8.180e-03	0.999	3.148e-02	1.000	1.222e-04	1.999
1	1	7.114e-02	—	2.187e-01	—	2.406e-02	—
	2	1.934e-02	1.879	5.566e-02	1.974	3.063e-03	2.974
	3	4.941e-03	1.968	1.400e-02	1.991	3.832e-04	2.999
	4	1.242e-03	1.992	3.512e-03	1.995	4.779e-05	3.003
	5	3.109e-04	1.998	8.800e-04	1.997	5.962e-06	3.003
	6	7.776e-05	2.000	2.203e-04	1.998	7.443e-07	3.002
2	1	1.615e-02	—	3.870e-02	—	4.435e-03	—
	2	2.159e-03	2.903	4.874e-03	2.989	2.925e-04	3.922
	3	2.745e-04	2.975	6.113e-04	2.995	1.854e-05	3.980
	4	3.446e-05	2.994	7.665e-05	2.996	1.164e-06	3.994
	5	4.313e-06	2.998	9.599e-06	2.997	7.283e-08	3.998
	6	5.392e-07	3.000	1.201e-06	2.998	4.554e-09	3.999
3	1	2.869e-03	—	5.447e-03	—	6.573e-04	—
	2	1.894e-04	3.921	3.376e-04	4.012	2.094e-05	4.972
	3	1.200e-05	3.980	2.108e-05	4.002	6.558e-07	4.997
	4	7.526e-07	3.995	1.319e-06	3.998	2.049e-08	5.000
	5	4.708e-08	3.999	8.251e-08	3.998	6.403e-10	5.000
	6	2.943e-09	4.000	5.160e-09	3.999	2.001e-11	5.000

TABLE 6

Convergence history for the hybridized Raviart-Thomas method of order k on quadrilateral cells, for $k = 0, 1, 2, 3$. We expect the same rates of convergence as the H-RT method on simplicies, detailed in Table 5.

RTCF-H method							
k	mesh	$\|p - p_h\|_{L^2(\Omega)} \leq \mathcal{O}(h^{k+1})$		$\|\boldsymbol{u} - \boldsymbol{u}_h\|_{\boldsymbol{L}^2(\Omega)} \leq \mathcal{O}(h^{k+1})$		$\|p - p_h^\star\|_{L^2(\Omega)} \leq \mathcal{O}(h^{k+2})$	
	r	L^2-error	rate	L^2-error	rate	L^2-error	rate
0	1	2.894e-01	—	1.054e+00	—	1.378e-01	—
	2	1.568e-01	0.884	5.127e-01	1.039	3.840e-02	1.844
	3	7.974e-02	0.976	2.531e-01	1.018	9.864e-03	1.961
	4	4.003e-02	0.994	1.261e-01	1.005	2.483e-03	1.990
	5	2.003e-02	0.999	6.298e-02	1.001	6.218e-04	1.998
	6	1.002e-02	1.000	3.148e-02	1.000	1.555e-04	1.999
1	1	6.203e-02	—	2.024e-01	—	1.681e-02	—
	2	1.606e-02	1.949	5.098e-02	1.989	2.071e-03	3.021
	3	4.052e-03	1.987	1.276e-02	1.998	2.582e-04	3.004
	4	1.015e-03	1.997	3.191e-03	2.000	3.225e-05	3.001
	5	2.539e-04	1.999	7.979e-04	2.000	4.031e-06	3.000
	6	6.349e-05	2.000	1.995e-04	2.000	5.038e-07	3.000
2	1	8.350e-03	—	2.667e-02	—	1.531e-03	—
	2	1.070e-03	2.964	3.376e-03	2.982	9.473e-05	4.015
	3	1.346e-04	2.991	4.233e-04	2.996	5.895e-06	4.006
	4	1.685e-05	2.998	5.295e-05	2.999	3.680e-07	4.002
	5	2.107e-06	2.999	6.620e-06	3.000	2.299e-08	4.000
	6	2.634e-07	3.000	8.276e-07	3.000	1.437e-09	4.000
3	1	8.326e-04	—	2.638e-03	—	1.116e-04	—
	2	5.304e-05	3.972	1.670e-04	3.981	3.482e-06	5.003
	3	3.331e-06	3.993	1.047e-05	3.995	1.087e-07	5.001
	4	2.084e-07	3.998	6.550e-07	3.999	3.396e-09	5.000
	5	1.303e-08	4.000	4.094e-08	4.000	1.061e-10	5.000
	6	8.146e-10	4.000	2.559e-09	4.000	3.317e-12	5.000

Additionally, we run the same numerical study on the hybridized BDM discretization, starting with the lowest order method on simplices. Our findings for the BDM-H method are presented in Table 7. Again, for all k we achieve the expected rates of convergence in the L^2-norm for the scalar and flux approximations, which are k and $k+1$ respectively [14]. When using scalar post-processing, we also obtain accelerated rates of convergence in the post-processed scalar. A noticeable difference here is that the lowest-order method ($k = 1$) with scalar post-processing produces a new approximation converging at a rate of $k + 1$, while subsequent orders of $k > 1$ produce new scalars converging at a rate of $k+2$. This is in full agreement with the theory, and this behavior is summarized in a more general context in [16] using L^p-error estimates.

C.3. LDG-H method. We now carry out numerical experiments to validate our implementation by verifying the theoretical properties of the LDG-H method for (121). In particular, we examine how select choices of stabilization parameter can influence the convergence rates of the computed solutions.

The LDG-H method seeks approximations $\boldsymbol{u}_h$ and p_h in the finite-dimensional discontinuous spaces $\boldsymbol{U}_h = [V_h]^2$ and V_h respectively, with V_h being the space of discontinuous Lagrange polynomials. As discussed in subsection 2.2.3, we introduce

TABLE 7

Convergence history for the hybridized Brezzi-Douglas-Marini method of order k on simplices, for $k = 1, 2, 3$. For all k, we expect the scalar and flux approximations to converge at a rate of k and $k+1$, respectively. When using scalar post-processing, the lowest-order ($k = 1$) method achieves $k+1$ convergence in the new scalar. For orders $k > 1$, superconvergence in the post-processed scalar is achieved at a rate of $k + 2$.

BDM-H method							
k	mesh	$\|p - p_h\|_{L^2(\Omega)} \leq \mathcal{O}(h)$		$\|\boldsymbol{u} - \boldsymbol{u}_h\|_{\boldsymbol{L}^2(\Omega)} \leq \mathcal{O}(h^2)$		$\|p - p_h^\star\|_{L^2(\Omega)} \leq \mathcal{O}(h^2)$	
	r	L^2-error	rate	L^2-error	rate	L^2-error	rate
1	1	2.578e-01	—	6.323e-01	—	1.160e-01	—
	2	1.320e-01	0.966	1.833e-01	1.787	3.274e-02	1.824
	3	6.567e-02	1.007	4.776e-02	1.940	8.491e-03	1.947
	4	3.276e-02	1.003	1.208e-02	1.984	2.144e-03	1.986
	5	1.637e-02	1.001	3.029e-03	1.995	5.373e-04	1.996
	6	8.182e-03	1.000	7.580e-04	1.999	1.344e-04	1.999
		$\|p - p_h\|_{L^2(\Omega)} \leq \mathcal{O}(h^k)$		$\|\boldsymbol{u} - \boldsymbol{u}_h\|_{\boldsymbol{L}^2(\Omega)} \leq \mathcal{O}(h^{k+1})$		$\|p - p_h^\star\|_{L^2(\Omega)} \leq \mathcal{O}(h^{k+2})$	
2	1	7.386e-02	—	1.072e-01	—	1.291e-02	—
	2	1.950e-02	1.921	1.465e-02	2.871	9.590e-04	3.750
	3	4.951e-03	1.978	1.882e-03	2.960	6.312e-05	3.925
	4	1.243e-03	1.994	2.374e-04	2.987	4.003e-06	3.979
	5	3.110e-04	1.999	2.977e-05	2.995	2.513e-07	3.994
	6	7.776e-05	2.000	3.726e-06	2.998	1.573e-08	3.998
3	1	1.629e-02	—	1.806e-02	—	1.275e-03	—
	2	2.164e-03	2.913	1.195e-03	3.917	4.096e-05	4.960
	3	2.747e-04	2.978	7.560e-05	3.983	1.272e-06	5.009
	4	3.447e-05	2.994	4.741e-06	3.995	3.964e-08	5.004
	5	4.313e-06	2.999	2.966e-07	3.998	1.238e-09	5.001
	6	5.392e-07	3.000	1.855e-08	3.999	3.869e-11	5.000

the numerical flux terms:

$$\widehat{\boldsymbol{u}}_h = \boldsymbol{u}_h + \tau\,(p_h - \widehat{p}_h)\,\boldsymbol{n}, \tag{127}$$

$$\widehat{p}_h = \lambda_h, \tag{128}$$

where $\lambda_h \in M_h$ is a function approximating the trace of the scalar variable on $\partial\mathcal{T}_h$ and $\tau > 0$ is a stabilization parameter that, in general, may vary on each facet $e \in \partial\mathcal{T}_h$. The full LDG-H formulation reads: find $(\boldsymbol{u}_h, p_h, \lambda_h) \in \boldsymbol{U}_h \times V_h \times M_h$ such that

$$(\boldsymbol{w}, \boldsymbol{u}_h)_{\mathcal{T}_h} - (\nabla \cdot \boldsymbol{w}, p_h)_{\mathcal{T}_h} + \langle [\![\boldsymbol{w}]\!], \lambda_h \rangle_{\partial\mathcal{T}_h} = 0, \tag{129}$$

$$-(\nabla\phi, \boldsymbol{u}_h)_{\mathcal{T}_h} + \langle \phi, [\![\widehat{\boldsymbol{u}}_h]\!] \rangle_{\partial\mathcal{T}_h} = (\phi, f)_{\mathcal{T}_h}, \tag{130}$$

$$\langle \gamma, [\![\widehat{\boldsymbol{u}}_h]\!] \rangle_{\mathcal{E}_h^\circ} = 0, \tag{131}$$

$$\langle \gamma, \lambda_h \rangle_{\partial\Omega_D} = 0, \tag{132}$$

for all $(\boldsymbol{w}, \phi, \gamma) \in \boldsymbol{U}_h \times V_h \times M_h$.

The theoretical convergence rates for the LDG-H method vary depending on the choice of τ. Therefore we run our numerical study as before, but take various choices of τ. Table 8 summarizes the expected convergence rates for the LDG-H method with various orders of τ, which is also discussed in more detail in [17, 20]. We consider three cases: $\tau = 1$ (Table 9), $\tau = h$ (Table 10), and $\tau = \frac{1}{h}$ (Table 11), where h denotes the facet area. In all LDG-H experiments, we use both scalar and flux post-processing methods described in Appendix B to generate new approximations: $p_h^\star$ and $\boldsymbol{u}_h^\star$.

TABLE 8

The expected theoretical convergence rates of the LDG-H method with a stability parameter τ of a particular order. Note that the orders of τ considered in this table correspond to the ones in our experiments. There may be other possible choices of τ which yield different rates of convergence.

parameter	expected rates of convergence ($k \geq 1$)			
τ	$\|p - p_h\|_{L^2(\Omega)}$	$\|\boldsymbol{u} - \boldsymbol{u}_h\|_{\boldsymbol{L}^2(\Omega)}$	$\|p - p_h^\star\|_{L^2(\Omega)}$	$\|\boldsymbol{u} - \boldsymbol{u}_h^\star\|_{\boldsymbol{L}^2(\Omega)}$
$\mathcal{O}(1)$	$k+1$	$k+1$	$k+2$	$k+1$
$\mathcal{O}(h)$	k	$k+1$	$k+2$	$k+1$
$\mathcal{O}\left(h^{-1}\right)$	$k+1$	k	$k+1$	k

TABLE 9

Convergence history for the LDG-H method with stabilization parameter $\tau = 1$ for degree $k = 1, 2, 3$.

LDG-H method ($\tau = \mathcal{O}(1)$)									
k	mesh	$\|p - p_h\|_{L^2(\Omega)} \leq \mathcal{O}(h^{k+1})$		$\|\boldsymbol{u} - \boldsymbol{u}_h\|_{\boldsymbol{L}^2(\Omega)} \leq \mathcal{O}(h^{k+1})$		$\|p - p_h^\star\|_{L^2(\Omega)} \leq \mathcal{O}(h^{k+2})$		$\|\boldsymbol{u} - \boldsymbol{u}_h^\star\|_{\boldsymbol{L}^2(\Omega)} \leq \mathcal{O}(h^{k+1})$	
	r	L^2-error	rate	L^2-error	rate	L^2-error	rate	L^2-error	rate
1	1	1.700e-01	—	3.746e-01	—	3.101e-02	—	2.356e-01	—
	2	4.829e-02	1.816	9.985e-02	1.907	3.947e-03	2.974	6.033e-02	1.965
	3	1.256e-02	1.943	2.531e-02	1.980	4.844e-04	3.027	1.516e-02	1.993
	4	3.182e-03	1.981	6.342e-03	1.997	5.960e-05	3.023	3.795e-03	1.998
	5	7.997e-04	1.993	1.586e-03	2.000	7.380e-06	3.014	9.492e-04	1.999
	6	2.003e-04	1.997	3.964e-04	2.000	9.177e-07	3.007	2.373e-04	2.000
2	1	3.621e-02	—	8.362e-02	—	5.063e-03	—	4.249e-02	—
	2	5.023e-03	2.850	1.110e-02	2.913	3.266e-04	3.954	5.369e-03	2.984
	3	6.485e-04	2.953	1.405e-03	2.982	2.047e-05	3.996	6.716e-04	2.999
	4	8.197e-05	2.984	1.760e-04	2.997	1.277e-06	4.002	8.393e-05	3.000
	5	1.029e-05	2.994	2.200e-05	3.000	7.970e-08	4.002	1.049e-05	3.000
	6	1.289e-06	2.997	2.749e-06	3.000	4.977e-09	4.001	1.311e-06	3.000
3	1	6.203e-03	—	1.464e-02	—	7.319e-04	—	6.077e-03	—
	2	4.248e-04	3.868	9.666e-04	3.921	2.334e-05	4.971	3.784e-04	4.005
	3	2.729e-05	3.960	6.114e-05	3.983	7.294e-07	5.000	2.357e-05	4.005
	4	1.722e-06	3.986	3.829e-06	3.997	2.276e-08	5.002	1.471e-06	4.002
	5	1.080e-07	3.995	2.394e-07	4.000	7.102e-10	5.002	9.192e-08	4.001
	6	6.761e-09	3.998	1.496e-08	4.000	2.219e-11	5.000	5.744e-09	4.000

TABLE 10

Convergence history for the LDG-H method with stabilization parameter $\tau = h$, where h denotes the facet area, for degree $k = 1, 2, 3$.

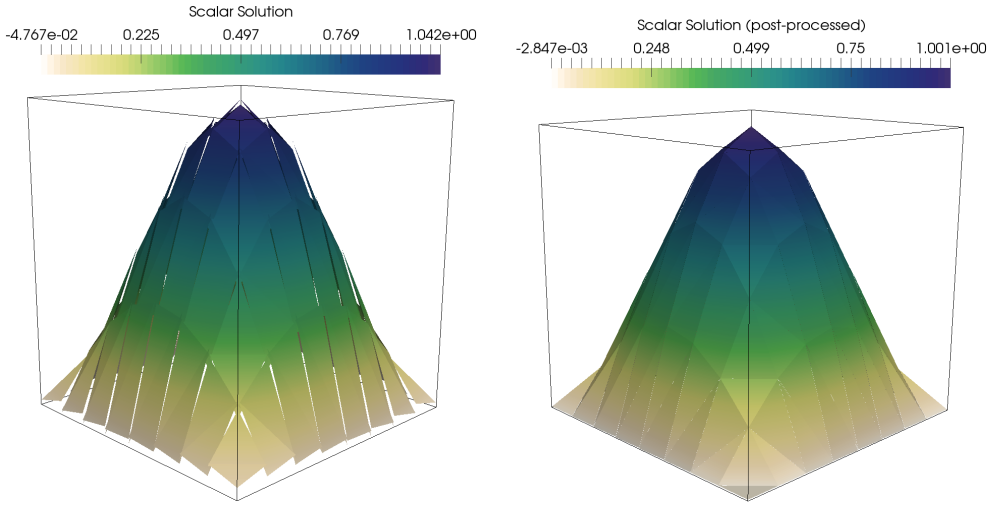
LDG-H method ($\tau = \mathcal{O}(h)$)									
k	mesh	$\|p - p_h\|_{L^2(\Omega)} \leq \mathcal{O}(h^{k})$		$\|\boldsymbol{u} - \boldsymbol{u}_h\|_{\boldsymbol{L}^2(\Omega)} \leq \mathcal{O}(h^{k+1})$		$\|p - p_h^\star\|_{L^2(\Omega)} \leq \mathcal{O}(h^{k+2})$		$\|\boldsymbol{u} - \boldsymbol{u}_h^\star\|_{\boldsymbol{L}^2(\Omega)} \leq \mathcal{O}(h^{k+1})$	
	r	L^2-error	rate	L^2-error	rate	L^2-error	rate	L^2-error	rate
1	1	2.838e-01	—	3.762e-01	—	2.770e-02	—	2.266e-01	—
	2	1.565e-01	0.859	9.999e-02	1.912	3.280e-03	3.078	5.639e-02	2.007
	3	8.012e-02	0.966	2.539e-02	1.978	3.975e-04	3.045	1.406e-02	2.004
	4	4.029e-02	0.992	6.372e-03	1.994	4.917e-05	3.015	3.518e-03	1.999
	5	2.018e-02	0.998	1.595e-03	1.998	6.125e-06	3.005	8.805e-04	1.998
	6	1.009e-02	0.999	3.989e-04	1.999	7.647e-07	3.002	2.203e-04	1.999
2	1	5.925e-02	—	8.398e-02	—	4.888e-03	—	4.030e-02	—
	2	1.596e-02	1.892	1.110e-02	2.919	3.151e-04	3.955	4.940e-03	3.028
	3	4.067e-03	1.973	1.407e-03	2.980	1.988e-05	3.986	6.138e-04	3.009
	4	1.022e-03	1.993	1.765e-04	2.995	1.246e-06	3.996	7.674e-05	3.000
	5	2.557e-04	1.998	2.209e-05	2.999	7.794e-08	3.999	9.603e-06	2.998
	6	6.394e-05	2.000	2.762e-06	3.000	4.872e-09	4.000	1.201e-06	2.999
3	1	1.006e-02	—	1.470e-02	—	7.092e-04	—	5.689e-03	—
	2	1.335e-03	2.913	9.661e-04	3.928	2.259e-05	4.972	3.430e-04	4.052
	3	1.694e-04	2.978	6.112e-05	3.982	7.094e-07	4.993	2.123e-05	4.014
	4	2.126e-05	2.994	3.832e-06	3.996	2.220e-08	4.998	1.326e-06	4.002
	5	2.660e-06	2.999	2.397e-07	3.999	6.937e-10	5.000	8.288e-08	3.999
	6	3.326e-07	3.000	1.528e-08	3.971	2.192e-11	4.984	5.993e-09	3.790

TABLE 11

Convergence history for the LDG-H method with stabilization parameter $\tau = \frac{1}{h}$, where h denotes the facet area, for degree $k = 1, 2, 3$.

LDG-H method ($\tau = \mathcal{O}(h^{-1})$)									
k	mesh	$\|p - p_h\|_{L^2(\Omega)} \leq \mathcal{O}(h^{k+1})$		$\|\boldsymbol{u} - \boldsymbol{u}_h\|_{\boldsymbol{L}^2(\Omega)} \leq \mathcal{O}(h^k)$		$\|p - p_h^\star\|_{L^2(\Omega)} \leq \mathcal{O}(h^{k+1})$		$\|\boldsymbol{u} - \boldsymbol{u}_h^\star\|_{\boldsymbol{L}^2(\Omega)} \leq \mathcal{O}(h^k)$	
	r	L^2-error	rate	L^2-error	rate	L^2-error	rate	L^2-error	rate
1	1	1.096e-01	—	3.920e-01	—	3.857e-02	—	2.627e-01	—
	2	2.339e-02	2.229	1.257e-01	1.641	8.083e-03	2.255	9.616e-02	1.450
	3	5.488e-03	2.091	4.730e-02	1.410	1.907e-03	2.083	4.251e-02	1.177
	4	1.348e-03	2.026	2.113e-02	1.162	4.699e-04	2.021	2.050e-02	1.053
	5	3.353e-04	2.007	1.022e-02	1.048	1.171e-04	2.005	1.015e-02	1.014
	6	8.374e-05	2.002	5.067e-03	1.013	2.924e-05	2.001	5.063e-03	1.004
2	1	2.385e-02	—	8.839e-02	—	5.668e-03	—	4.911e-02	—
	2	2.484e-03	3.263	1.454e-02	2.603	4.724e-04	3.585	9.242e-03	2.410
	3	2.882e-04	3.107	2.838e-03	2.358	4.658e-05	3.342	2.075e-03	2.155
	4	3.525e-05	3.031	6.473e-04	2.132	5.329e-06	3.128	5.023e-04	2.047
	5	4.381e-06	3.008	1.577e-04	2.037	6.489e-07	3.038	1.244e-04	2.013
	6	5.469e-07	3.002	3.916e-05	2.009	8.052e-08	3.010	3.102e-05	2.004
3	1	4.141e-03	—	1.556e-02	—	8.131e-04	—	7.273e-03	—
	2	2.157e-04	4.263	1.297e-03	3.585	3.335e-05	4.608	7.002e-04	3.377
	3	1.251e-05	4.107	1.296e-04	3.323	1.654e-06	4.334	7.907e-05	3.147
	4	7.652e-07	4.031	1.498e-05	3.112	9.507e-08	4.120	9.555e-06	3.049
	5	4.755e-08	4.008	1.834e-06	3.030	5.797e-09	4.036	1.181e-06	3.016
	6	2.968e-09	4.002	2.281e-07	3.007	3.597e-10	4.010	1.470e-07	3.006

We can see in Tables 9, 10, and 11 that these orders are achieved fairly well. It is also interesting to note that, while the post-processed flux $\boldsymbol{u}_h^\star$ maintains the rate of convergence of $\boldsymbol{u}_h$, the error in $\boldsymbol{u}_h^\star$ is slightly smaller than that of $\boldsymbol{u}_h$. Similar behavior was observed in [21] on a different manufactured problem. While scalar post-processing does not yield superconvergence in $p_h^\star$ when $\tau = \frac{1}{h}$, the errors in $p_h^\star$ are smaller than that of p_h. For a visualization of the effects scalar post-processing has, Figure 5 displays the computed scalar-fields ($\tau = 1$) with and without post-processing. We can see the obvious smoothing effect post-processing has on p_h.



(a) Scalar solution. (b) Post-processed scalar solution.

FIG. 5. *Comparison between the scalar solution of the LDG-H method ($\tau = 1$) using discontinuous linear finite elements without (left) and with (right) post-processing. These were computed on a mesh with $r = 3$.*

Appendix D. Nonlinear shallow water solver. In this last section, we provide some supplementary information about the nonlinear shallow water solver described in Section 4.3.

The starting point is the implicit midpoint rule discretisation of Equations (81–82), where we seek $(\boldsymbol{u}_h^{n+1}, D_h^{n+1}) \in \boldsymbol{U}_h \times V_h$ such that

$$\begin{aligned}(133)\quad \left(\boldsymbol{w}, \boldsymbol{u}_h^{n+1} - \boldsymbol{u}_h^n\right)_{\mathcal{T}_h} - \Delta t \left(\nabla^\perp \left(\boldsymbol{w} \cdot \boldsymbol{u}_h^{*\perp}\right), \boldsymbol{u}_h^{*\perp}\right)_{\mathcal{T}_h} + \Delta t \left(\boldsymbol{w}, f \boldsymbol{u}_h^{*\perp}\right)_{\mathcal{T}_h} \\ + \Delta t \langle [\![\boldsymbol{n}^\perp \boldsymbol{w} \cdot \boldsymbol{u}_h^{*\perp}]\!], \tilde{\boldsymbol{u}_h^{*\perp}} \rangle_{\partial \mathcal{T}_h} \\ - \Delta t \left(\nabla \cdot \boldsymbol{w}, g\left(D_h^* + b\right) + \frac{1}{2}|\boldsymbol{u}_h^*|^2\right)_{\mathcal{T}_h} = 0,\end{aligned}$$

$$(134)\quad \left(\phi, D_h^{n+1} - D_h^n\right)_{\mathcal{T}_h} - \Delta t (\nabla \phi, \boldsymbol{u}_h^* D_h^*)_{\mathcal{T}_h} + \Delta t \langle [\![\phi \boldsymbol{u}_h^*]\!], \tilde{D}_h^* \rangle_{\partial \mathcal{T}_h}, = 0,$$

for all $(\boldsymbol{w}, \phi) \in \boldsymbol{U}_h \times V_h$, where $\boldsymbol{u}_h^* = (\boldsymbol{u}_h^{n+1} + \boldsymbol{u}_h^n)/2$ and $D_h^* = (D_h^{n+1} + D_h^n)/2$.

To make a more practical approach, we replace the solution of this problem by a fixed (4, say) number of iterations of a Picard iteration of the form (85–86), where we now describe the construction of the residual functionals $R_{\boldsymbol{u}}$ and R_D. One approach would be to simply define these functionals from (133–134), but this leads to a small critical timestep for stability of the scheme. To make the numerical scheme more stable, we define residuals as follows.

For $R_{\boldsymbol{u}}$, we first solve for $\boldsymbol{v}_h \in \boldsymbol{U}_h$ such that

$$\begin{aligned}(135)\quad (\boldsymbol{w}, \boldsymbol{v}_h - \boldsymbol{u}_h^n)_{\mathcal{T}_h} - \Delta t \left(\nabla^\perp \left(\boldsymbol{w} \cdot \boldsymbol{u}_h^{*\perp}\right), \boldsymbol{v}_h^{\sharp\perp}\right)_{\mathcal{T}_h} + \Delta t \left(\boldsymbol{w}, f \boldsymbol{v}_h^{\sharp\perp}\right)_{\mathcal{T}_h} \\ + \Delta t \langle [\![\boldsymbol{n}^\perp \boldsymbol{w} \cdot \boldsymbol{u}_h^{*\perp}]\!], \tilde{\boldsymbol{v}_h^{\sharp\perp}} \rangle_{\partial \mathcal{T}_h} \\ - \Delta t \left(\nabla \cdot \boldsymbol{w}, g\left(D_h^* + b\right) + \frac{1}{2}|\boldsymbol{u}_h^*|^2\right)_{\mathcal{T}_h} = 0,\end{aligned}$$

for all $\boldsymbol{w} \in \boldsymbol{U}_h$, where $\boldsymbol{v}_h^\sharp = (\boldsymbol{v}_h + \boldsymbol{u}_h^n)/2$. This is a linear variational problem. Then,

$$(136)\quad R_{\boldsymbol{u}}[\boldsymbol{u}_h^{n+1}, D_h^{n+1}; \boldsymbol{w}] = \left(\boldsymbol{w}, \boldsymbol{v}_h - \boldsymbol{u}_h^{n+1}\right)_{\mathcal{T}_h}.$$

Similarly, for R_D we first solve for $E_h \in V_h$ such that

$$(137)\quad (\phi, E_h - D_h^n)_{\mathcal{T}_h} - \Delta t (\nabla \phi, \boldsymbol{u}_h^* E_h^\sharp)_{\mathcal{T}_h} + \Delta t \langle [\![\phi \boldsymbol{u}_h^*]\!], \tilde{E}_h^\sharp \rangle_{\partial \mathcal{T}_h} = 0,$$

for all $\phi \in V_h$, where $E_h^\sharp = (E_h + D_h^n)/2$. This is also a linear problem. Then,

$$(138)\quad R_D[\boldsymbol{u}_h^{n+1}, D_h^{n+1}; \phi] = \left(\phi, E_h - D_h^{n+1}\right)_{\mathcal{T}_h}.$$

The complete scheme is detailed in Algorithm 1.

D.1. Preconditioning via hybridization. Solving the implicit linear system in line 8 of Algorithm 1 requires the solution of the indefinite saddle-point system:

$$(139)\quad \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{Bmatrix} \Delta U \\ \Delta D \end{Bmatrix} = \begin{Bmatrix} R_{\boldsymbol{u}} \\ R_D \end{Bmatrix}.$$

In traditional staggered finite difference methods, a typical procedure is to eliminate the velocity increments from (139) to arrive at a discrete elliptic system. However,

Algorithm 1 Nonlinear procedure for the rotating shallow water system using a semi-implicit time-integration scheme. Lines 7–11 define the Picard cycle, in which successive linear systems are solved for corrective updates.

1: $t_n = 0$
2: $\boldsymbol{u}_h^n \leftarrow \boldsymbol{u}^0$ {Velocity initial condition}
3: $D_h^n \leftarrow D^0$ {Depth initial condition}
4: **while** $t_n < t_{\max}$ **do**
5: $\quad \boldsymbol{u}_h^{n+1} \leftarrow \boldsymbol{u}_h^n$
6: $\quad D_h^{n+1} \leftarrow D_h^n$
7: $\quad$ **for** $i = 1, \cdots, k$ **do**
8: $\quad\quad$ Solve for $(\Delta \boldsymbol{u}_h, \Delta D_h) \in \boldsymbol{U}_h \times V_h$ such that:

$$
\begin{aligned}
(\boldsymbol{w}, \Delta \boldsymbol{u}_h)_{\mathcal{T}_h} + \frac{\Delta t}{2} \left(\boldsymbol{w}, f \Delta \boldsymbol{u}_h^{\perp}\right)_{\mathcal{T}_h} & \\
- \frac{\Delta t}{2} (\nabla \cdot \boldsymbol{w}, g \Delta D_h)_{\mathcal{T}_h} &= -R_{\boldsymbol{u}}[\boldsymbol{u}_h^{n+1}, D_h^{n+1}; \boldsymbol{w}], \\
(\phi, \Delta D_h)_{\mathcal{T}_h} + \frac{\Delta t}{2} (\phi, H \nabla \cdot \Delta \boldsymbol{u}_h)_{\mathcal{T}_h} &= -R_D[\boldsymbol{u}_h^{n+1}, D_h^{n+1}; \phi],
\end{aligned}
$$

$\quad\quad$ for all $(\boldsymbol{w}, \phi) \in \boldsymbol{U}_h \times V_h$ {Linearized system for corrective updates}
9: $\quad\quad \boldsymbol{u}_h^{n+1} \leftarrow \boldsymbol{u}_h^{n+1} + \Delta \boldsymbol{u}_h$
10: $\quad\quad D_h^{n+1} \leftarrow D_h^{n+1} + \Delta D_h$
11: $\quad$ **end for**
12: $\quad \boldsymbol{u}_h^n \leftarrow \boldsymbol{u}_h^{n+1}$
13: $\quad D_h^n \leftarrow D_h^{n+1}$
14: $\quad t_n \leftarrow t_n + \Delta t$
15: **end while**

this is problematic when using compatible finite elements, as the matrix A is not block-diagonal.

Our approach here is to precondition (139) by replacing the original mixed system with its hybrid-mixed equivalent form. Upon hybridizing the mixed system, we have the equivalent problem: find $(\Delta \widehat{\boldsymbol{u}}_h, \Delta D_h, \lambda_h) \in \widehat{\boldsymbol{U}}_h \times V_h \times M_h$ such that

$$
\begin{aligned}
(\boldsymbol{w}, \Delta \widehat{\boldsymbol{u}}_h)_{\mathcal{T}_h} + \frac{\Delta t}{2} \left(\boldsymbol{w}, f \Delta \widehat{\boldsymbol{u}}_h^{\perp}\right)_{\mathcal{T}_h} - \frac{\Delta t}{2} (\nabla \cdot \boldsymbol{w}, g \Delta D_h)_{\mathcal{T}_h} & \\
+ \langle [\![\boldsymbol{w}]\!], \lambda_h \rangle_{\partial \mathcal{T}_h} &= \widehat{R}_{\boldsymbol{u}}, \qquad (140) \\
(\phi, \Delta D_h)_{\mathcal{T}_h} + \frac{\Delta t}{2} (\phi, H \nabla \cdot \Delta \widehat{\boldsymbol{u}}_h)_{\mathcal{T}_h} &= R_D, \qquad (141) \\
\langle \gamma, [\![\Delta \widehat{\boldsymbol{u}}]\!] \rangle_{\partial \mathcal{T}_h} &= 0, \qquad (142)
\end{aligned}
$$

for all $(\boldsymbol{w}, \phi, \gamma) \in \widehat{\boldsymbol{U}}_h \times V_h \times M_h$. Note that the space M_h is chosen such that these trace functions when restricted to a facet $e \in \partial \mathcal{T}_h$ are from the same polynomial space as $\Delta \boldsymbol{u}_h \cdot \boldsymbol{n}$ restricted to that same facet. Additionally, it can be shown that λ_h is an approximation to $\Delta t g \Delta D / 2$, which can be exploited to show that the resulting linear system for λ_h can be solved using multigrid techniques [27].

The resulting three-field problem for (140)–(142) has the matrix form:

$$
\begin{bmatrix} \widehat{\boldsymbol{A}} & C^T \\ C & 0 \end{bmatrix} \begin{Bmatrix} \Delta X \\ \Lambda \end{Bmatrix} = \begin{Bmatrix} \widehat{R}_{\Delta X} \\ 0 \end{Bmatrix} \qquad (143)
$$

where $\widehat{\boldsymbol{A}}$ is the discontinuous operator corresponding to the left-hand side matrix in (139), $\Delta X = \left\{\Delta\widetilde{U} \quad \Delta D\right\}^T$, and $R_{\Delta X} = \left\{\widehat{R}_{\boldsymbol{u}} \quad R_D\right\}^T$ are the problem residuals. An explanation on how this procedure is implemented can be found in the main manuscript (see subsection 3.1.2). This discussion includes details on how the resulting computed solutions for (143), primarily the velocity, is projected back into the original $\boldsymbol{H}(\mathrm{div})$-conforming space.

This concludes the summary of our nonlinear procedure for solving (81)–(82) in the Williamson test case 5. Details about the test case itself and the results comparing our solver configurations for the implicit linear system (139) are discussed in subsection 4.3. The solver configurations used in the comparisons are displayed in Appendix E.3.

Appendix E. Full solver parameters.

E.1. Elliptic Helmholtz: static condensation. Here we use the static condensation preconditioner as described in subsection 3.1.1. All local solves and static condensation operations are performed by evaluating the corresponding Slate expressions. For the inner-most `ksp`, we numerically invert the Schur complement operator in the interface problem using the conjugate gradient method preconditioned with hypre's boomerAMG algebraic multigrid implementation. We select more aggressive coarsening techniques, using a method proposed by Ruge and Steuben in [50].

Listing 10
Solver configuration for static condensation

```
-ksp_type preonly
-mat_type matfree
-pc_type python
-pc_python_type scpc.SCCG
-static_condensation_ksp_type cg
-static_condensation_ksp_rtol 1.0e-13
-static_condensation_pc_type hypre
-static_condensation_pc_hypre_type boomeramg
-static_condensation_pc_hypre_boomeramg_no_CF True
-static_condensation_pc_hypre_boomeramg_coarsen_type HMIS
-static_condensation_pc_hypre_boomeramg_interp_type ext+i
-static_condensation_pc_hypre_boomeramg_P_max 4
-static_condensation_pc_hypre_boomeramg_agg_nl 1
```

E.2. Hybridized Mixed and LDG-H Poisson: static condensation. The three-field systems of the hybridized mixed and LDG-H methods is condensed to a system for the trace using the implementation described in 3.1.3. The reduced Schur complement is inverted using an LU-factorization, with MUMPS providing the factorization algorithm [2].

Listing 11
Solver configuration for static condensation of hybridized problems

```
-ksp_type preonly
-mat_type matfree
-pc_type python
-pc_python_type scpc.SCHybrid
-hybrid_sc_ksp_type preonly
-hybrid_sc_pc_type lu
```

```
-hybrid_sc_pc_factor_mat_solver_package mumps
```

E.3. Linear solver: hybridization and static condensation.

E.3.1. Approximate Schur complement. For the performance comparison of the implicit solve in subsection 4.3, we solve the linearized system using GMRES. A full Schur complement factorization with a diagonally lumped velocity mass matrix (`-selfp`) is used to precondition the system. The upper-left matrix block is inverted approximately using block-jacobi-ILU. For the approximate Schur complement, we use the conjugate gradient method preconditioned with geometric-agglomerated algebraic multigrid (GAMG). We set a maximum of two Chebyshev iterations, preconditioned with block-Jacobi-ILU, for each multigrid level.

LISTING 12
Solver configuration for approximate Schur complement approach

```
-ksp_type gmres
-pc_type fieldsplit
-pc_fieldsplit_type schur
-pc_fieldsplit_schur_fact_type FULL
-pc_fieldsplit_schur_precondition selfp
-pc_fieldsplit_0_ksp_type preonly
-pc_fieldsplit_0_pc_type bjacobi
-pc_fieldsplit_0_sub_pc_type ilu
-pc_fieldsplit_1_ksp_type cg
-pc_fieldsplit_1_ksp_rtol 1.0e-8
-pc_fieldsplit_1_pc_type gamg
-pc_fieldsplit_1_mg_levels_ksp_type chebyshev
-pc_fieldsplit_1_mg_levels_ksp_max_it 2
-pc_fieldsplit_1_mg_levels_pc_type bjacobi
-pc_fieldsplit_1_mg_levels_sub_pc_type ilu
```

E.3.2. Hybridization and static condensation. The preconditioner in subsection 3.1.2 transforms the mixed system into its hybridized form by introducing Lagrange multipliers and replacing form arguments with discontinuous functions. Once the hybridized flux is reconstructed, the result is projected back into the $\boldsymbol{H}(\mathrm{div})$ finite element space via local facet averaging.

LISTING 13
Solver configuration for hybridization preconditioner

```
-ksp_type preonly
-mat_type matfree
-pc_type python
-pc_python_type scpc.HybridizationPC
-hybridization_ksp_type cg
-hybridization_ksp_rtol 1.0e-8
-hybridization_pc_type gamg
-hybridization_mg_levels_ksp_type chebyshev
-hybridization_mg_levels_ksp_max_it 2
-hybridization_mg_levels_pc_type bjacobi
-hybridization_mg_levels_sub_pc_type ilu
```

[1] M. S. Alnæs, A. Logg, K. B. Ølgaard, M. E. Rognes, and G. N. Wells, *Unified form language: A domain-specific language for weak formulations of partial differential equations*, ACM Transactions on Mathematical Software (TOMS), 40 (2014), p. 9.

[2] P. R. Amestoy, I. S. Duff, and J.-Y. L'excellent, *Multifrontal parallel distributed symmetric and unsymmetric solvers*, Computer methods in applied mechanics and engineering, 184 (2000), pp. 501–520.

[3] D. N. Arnold and F. Brezzi, *Mixed and nonconforming finite element methods: implementation, postprocessing and error estimates*, ESAIM: Mathematical Modelling and Numerical Analysis, 19 (1985), pp. 7–32.

[4] D. N. Arnold, R. S. Falk, and R. Winther, *Multigrid in h(div) and h(curl)*, Numerische Mathematik, 85 (2000), pp. 197–217, https://doi.org/10.1007/s002110000137.

[5] A. H. Baker, R. D. Falgout, T. Gamblin, T. V. Kolev, M. Schulz, and U. M. Yang, *Scaling algebraic multigrid solvers: On the road to exascale*, in Competence in High Performance Computing 2010, Springer, 2011, pp. 215–226.

[6] S. Balay, S. Abhyankar, M. Adams, P. Brune, K. Buschelman, L. Dalcin, W. Gropp, B. Smith, D. Karpeyev, D. Kaushik, et al., *Petsc users manual revision 3.7*, tech. report, Argonne National Lab.(ANL), Argonne, IL (United States), 2016.

[7] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, *Efficient management of parallelism in object-oriented numerical software libraries*, in Modern software tools for scientific computing, Springer, 1997, pp. 163–202.

[8] W. Bauer and C. J. Cotter, *Energy-enstrophy conserving compatible finite element schemes for the shallow water equations on rotating domains with boundaries*, arXiv preprint arXiv:1801.00691, (2018).

[9] D. Boffi, F. Brezzi, M. Fortin, et al., *Mixed finite element methods and applications*, vol. 44, Springer, 2013.

[10] J. H. Bramble and J. Xu, *A local post-processing technique for improving the accuracy in mixed finite-element approximations*, SIAM Journal on Numerical Analysis, 26 (1989), pp. 1267–1275.

[11] F. Brezzi, D. Boffi, L. Demkowicz, R. Durán, R. Falk, and M. Fortin, *Mixed finite elements, compatibility conditions, and applications*, Springer, 2008.

[12] F. Brezzi, J. Douglas, R. Durán, and M. Fortin, *Mixed finite elements for second order elliptic problems in three variables*, Numerische Mathematik, 51 (1987), pp. 237–250.

[13] F. Brezzi, J. Douglas, and L. D. Marini, *Two families of mixed finite elements for second order elliptic problems*, Numerische Mathematik, 47 (1985), pp. 217–235.

[14] F. Brezzi and M. Fortin, *Mixed and hybrid finite element methods*, vol. 15, Springer Science & Business Media, 2012.

[15] J. Brown, M. G. Knepley, D. A. May, L. C. McInnes, and B. Smith, *Composable linear solvers for multiphysics*, in Parallel and Distributed Computing (ISPDC), 2012 11th International Symposium on, IEEE, 2012, pp. 55–62.

[16] Z. Chen, *Lp-posteriori error analysis of mixed methods for linear and quasilinear elliptic problems*, in Modeling, Mesh Generation, and Adaptive Numerical Methods for Partial Differential Equations, Springer, 1995, pp. 187–199.

[17] B. Cockburn, *Static condensation, hybridization, and the devising of the hdg methods*, in Building Bridges: Connections and Challenges in Modern Approaches to Numerical Partial Differential Equations, Springer, 2016, pp. 129–177.

[18] B. Cockburn, J. Gopalakrishnan, and R. Lazarov, *Unified hybridization of discontinuous galerkin, mixed, and continuous galerkin methods for second order elliptic problems*, SIAM Journal on Numerical Analysis, 47 (2009), pp. 1319–1365.

[19] B. Cockburn, J. Gopalakrishnan, F. Li, N.-C. Nguyen, and J. Peraire, *Hybridization and postprocessing techniques for mixed eigenfunctions*, SIAM Journal on Numerical Analysis, 48 (2010), pp. 857–881.

[20] B. Cockburn, J. Gopalakrishnan, and F.-J. Sayas, *A projection-based error analysis of hdg methods*, Mathematics of Computation, 79 (2010), pp. 1351–1367.

[21] B. Cockburn, J. Guzmán, and H. Wang, *Superconvergent discontinuous galerkin methods for second-order elliptic problems*, Mathematics of Computation, 78 (2009), pp. 1–24.

[22] C. J. Cotter and J. Shipton, *Mixed finite elements for numerical weather prediction*, Journal of Computational Physics, 231 (2012), pp. 7076–7091.

[23] C. J. Cotter and J. Thuburn, *A finite element exterior calculus framework for the rotating shallow-water equations*, Journal of Computational Physics, 257 (2014), pp. 1506–1526.

[24] L. D. Dalcin, R. R. Paz, P. A. Kler, and A. Cosimo, *Parallel distributed computing using python*, Advances in Water Resources, 34 (2011), pp. 1124–1139.

[25] T. A. Davis, *Algorithm 832: Umfpack v4. 3—an unsymmetric-pattern multifrontal method*,

ACM Transactions on Mathematical Software (TOMS), 30 (2004), pp. 196–199.
[26] J. Douglas and J. E. Roberts, *Global estimates for mixed methods for second order elliptic equations*, Mathematics of computation, 44 (1985), pp. 39–52.
[27] J. Gopalakrishnan and S. Tan, *A convergent multigrid cycle for the hybridized mixed method*, Numerical Linear Algebra with Applications, 16 (2009), pp. 689–714.
[28] G. Guennebaud, B. Jacob, M. Lenz, et al., *Eigen v3, 2010*, URL http://eigen. tuxfamily. org, (2015).
[29] R. J. Guyan, *Reduction of stiffness and mass matrices*, AIAA journal, 3 (1965), p. 380.
[30] F. Hartmann, *The discrete babuška-brezzi condition*, Archive of Applied Mechanics, 56 (1986), pp. 221–228.
[31] F. Hecht, *New development in freefem++*, Journal of numerical mathematics, 20 (2012), pp. 251–266.
[32] R. Hiptmair and J. Xu, *Nodal auxiliary space preconditioning in h(curl) and h(div) spaces*, SIAM Journal on Numerical Analysis, 45 (2007), pp. 2483–2509, https://doi.org/10.1137/060660588.
[33] M. Homolya, L. Mitchell, F. Luporini, and D. A. Ham, *Tsfc: a structure-preserving form compiler*, 2017, https://arxiv.org/abs/1705.03667.
[34] B. Irons, *Structural eigenvalue problems-elimination of unwanted variables*, AIAA journal, 3 (1965), pp. 961–962.
[35] R. C. Kirby and A. Logg, *A compiler for variational forms*, ACM Transactions on Mathematical Software (TOMS), 32 (2006), pp. 417–444.
[36] R. C. Kirby and L. Mitchell, *Solver composition across the PDE/linear algebra barrier*, SIAM Journal on Scientific Computing, (2017), https://arxiv.org/abs/1706.01346. To appear.
[37] R. M. Kirby, S. J. Sherwin, and B. Cockburn, *To cg or to hdg: a comparative study*, Journal of Scientific Computing, 51 (2012), pp. 183–212.
[38] M. G. Larson and F. Bengzon, *The Finite Element Method: Theory, Implementation, and Practice*, Springer-Verlag, Berlin Heidelberg, 2010, https://doi.org/10.1007/978-3-642-33287-6.
[39] A. Logg, K.-A. Mardal, and G. Wells, *Automated solution of differential equations by the finite element method: The FEniCS book*, vol. 84, Springer Science & Business Media, 2012.
[40] A. Logg, K. B. Ølgaard, M. E. Rognes, and G. N. Wells, *Ffc: the fenics form compiler*, Automated Solution of Differential Equations by the Finite Element Method, (2012), pp. 227–238.
[41] A. Logg and G. N. Wells, *Dolfin: Automated finite element computing*, ACM Transactions on Mathematical Software (TOMS), 37 (2010), p. 20.
[42] K. Long, R. Kirby, and B. van Bloemen Waanders, *Unified embedded parallel finite element computations via software-based fréchet differentiation*, SIAM Journal on Scientific Computing, 32 (2010), pp. 3323–3351.
[43] J.-C. Nédélec, *Mixed finite elements in 3*, Numerische Mathematik, 35 (1980), pp. 315–341.
[44] D. Pardo, J. Álvarez-Aramberri, M. Paszynski, L. Dalcin, and V. M. Calo, *Impact of element-level static condensation on iterative solver performance*, Computers & Mathematics with Applications, 70 (2015), pp. 2331–2341.
[45] C. Prudhomme, V. Chabannes, G. Pena, and S. Veys, *Feel++: finite element embedded language in c++*, Free Software available at http://www.feelpp.org. Contributions from A. Samake, V. Doyeux, M. Ismail and S. Veys.
[46] F. Rathgeber, D. A. Ham, L. Mitchell, M. Lange, F. Luporini, A. T. McRae, G.-T. Bercea, G. R. Markall, and P. H. Kelly, *Firedrake: automating the finite element method by composing abstractions*, ACM Transactions on Mathematical Software (TOMS), 43 (2016), p. 24.
[47] F. Rathgeber, G. R. Markall, L. Mitchell, N. Loriant, D. A. Ham, C. Bertolli, and P. H. Kelly, *Pyop2: A high-level framework for performance-portable simulations on unstructured meshes*, in High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion:, IEEE, 2012, pp. 1116–1123.
[48] P.-A. Raviart and J.-M. Thomas, *A mixed finite element method for 2-nd order elliptic problems*, in Mathematical aspects of finite element methods, Springer, 1977, pp. 292–315.
[49] M. E. Rognes, R. C. Kirby, and A. Logg, *Efficient assembly of h(div) and h(curl) conforming finite elements*, SIAM Journal on Scientific Computing, 31 (2009), pp. 4130–4151.
[50] J. W. Ruge, *Algebraic multigrid*, Multigrid methods, 3 (1987), pp. 73–130.
[51] J. Shipton and C. Cotter, *Higher-order compatible finite element schemes for the nonlinear rotating shallow water equations on the sphere*, arXiv preprint arXiv:1707.00855, (2017).

[52] R. Stenberg, *Postprocessing schemes for some mixed finite elements*, ESAIM: Mathematical Modelling and Numerical Analysis, 25 (1991), pp. 151–167.
[53] D. L. Williamson, J. B. Drake, J. J. Hack, R. Jakob, and P. N. Swarztrauber, *A standard test set for numerical approximations to the shallow water equations in spherical geometry*, Journal of Computational Physics, 102 (1992), pp. 211–224.
[54] Zenodo/COFFEE, *COFFEE: A Compiler for Fast Expression Evaluation*, Nov. 2017, https://doi.org/10.5281/zenodo.597585.
[55] Zenodo/FIAT, *FIAT: The Finite Element Automated Tabulator*, Jan. 2018, https://doi.org/10.5281/zenodo.1135105.
[56] Zenodo/FInAT, *FInAT: a smarter library of finite elements*, Jan. 2018, https://doi.org/10.5281/zenodo.1135106.
[57] Zenodo/Firedrake, *Firedrake: an automated finite element system*, Jan. 2018, https://doi.org/10.5281/zenodo.1135096.
[58] Zenodo/PETSc, *PETSc: Portable, Extensible Toolkit for Scientific Computation*, Jan. 2018, https://doi.org/10.5281/zenodo.1135103.
[59] Zenodo/petsc4py, *petsc4py: The Python interface to PETSc*, Jan. 2018, https://doi.org/10.5281/zenodo.1135099.
[60] Zenodo/PyOP2, *OP2/PyOP2: Framework for performance-portable parallel computations on unstructured meshes*, Jan. 2018, https://doi.org/10.5281/zenodo.1155736.
[61] Zenodo/SCPC, *SCPC: Static condensation and hybridization in Firedrake & PETSc*, Jan. 2018, https://doi.org/10.5281/zenodo.1135108.
[62] Zenodo/Tabula-Rasa, *Tabula-Rasa: experimentation framework for hybridization and static condensation*, Jan. 2018, https://doi.org/10.5281/zenodo.1163390.
[63] Zenodo/TSFC, *TSFC: The Two Stage Form Compiler*, Jan. 2018, https://doi.org/10.5281/zenodo.1135098.
[64] Zenodo/UFL, *UFL: The Unified Form Language*, Jan. 2018, https://doi.org/10.5281/zenodo.1135104.