To CG or to HDG: a comparative study in 3D

Sergey Yakovlev[1], David Moxey[2], Robert M. Kirby[3] and Spencer J. Sherwin[4]

Abstract

Since the inception of discontinuous Galerkin (DG) methods for elliptic problems, there has existed a question of whether DG methods can be made more computationally efficient than continuous Galerkin (CG) methods. Fewer degrees of freedom, approximation properties for elliptic problems together with the number of optimization techniques, such as static condensation, available within CG framework make it challenging for DG methods to be competitive until recently. However, with the introduction of a static-condensation-amenable DG method – the hybridizable discontinuous Galerkin (HDG) method – it has become possible to perform a realistic comparison of CG and HDG methods when applied to elliptic problems. In this work, we extend upon an earlier 2D comparative study, providing numerical results and discussion of the CG and HDG method performance in three dimensions. The comparison categories covered include steady-state elliptic and time-dependent parabolic problems, various element types and serial and parallel performance. The postprocessing technique, which allows for superconvergence in the HDG case, is also discussed. Depending on the linear system solver used and the type of the problem (steady-state vs time-dependent) in question the HDG method either outperforms or demonstrates a comparable performance when compared with the CG method. The HDG method however falls behind performance-wise when the iterative solver is used, which indicates the need for an effective preconditioning strategy for the method.

Keywords: High-Order Finite Elements, Spectral/*hp* Elements, Discontinuous Galerkin Method, Hybridization, Parallel Computing, Postprocessing, Superconvergence

1 Introduction

Due to the large number of numerical methods for solving partial differential equations (PDEs), computational scientists and engineers are often faced with a choice of method to solve a given scientific problem. Typically, this choice is not based purely on numerical properties of a given method such as its asymptotic convergence rates, ability to handle complicated geometry or specific feature-capturing properties, but also on other more practical aspects such as robustness, time-to-implement and computational cost. In previous work [30] the authors presented a comparative study of the performance of the continuous and discontinuous Galerkin methods in the context of symmetric second-order elliptic PDEs, with the goal of providing guidance as to why one may select one versus the other based on these criteria. The

[1]sergeyyak@gmail.com, Scientific Computing and Imaging (SCI) Institute, Univ. of Utah, Salt Lake City, UT, USA.

[2]d.moxey@imperial.ac.uk, Department of Aeronautics, Imperial College London, London, UK

[3]kirby@cs.utah.edu, School of Computing and Scientific Computing and Imaging (SCI) Institute, Univ. of Utah, Salt Lake City, UT, USA; Corresponding Author.

[4]s.sherwin@imperial.ac.uk, Department of Aeronautics, Imperial College London, London, UK

performance results and comparison were based on the data obtained from two-dimensional numerical simulations. In this paper we significantly extend the scope of the aforementioned 2D study by comparing the performance of CG and DG methods both in serial and parallel in three dimensions.

The CG method has a rich history, having been utilised in a large number of numerical studies, and so we refer the interested reader to [25, 50–52] for a comprehensive discussion of the formulation and implementation of the method. Whilst DG methods are a more recent development, they too are now widely used as a spatial discretisation and are especially popular when considering convection dominated systems, as a DG discretisation often results in beneficial properties such as local conservation [11]. However, in recent years there has been much interest in creating efficient implicit DG discretisations of elliptic operators, given that DG methods do not need to necessarily enforce a continuous solution function as is the case in CG, which makes them more amenable to applications such as shock capturing when coupled with appropriate stabilisation.

The discretisation of elliptic operators has traditionally fallen into the realm of the CG method, where the size of the matrix system is substantially smaller than the equilvalent DG system since degrees of freedom associated with vertices, edges and faces which connect elements only appear once in the global system. Furthermore, through the application of the lesser-known *static condensation* (or *substructuring*) technique, the size of the matrix system can be further reduced. In this approach, given an appropriate choice of basis functions which make a distinction between boundary and interior modes, the CG system can be condensed by taking the Schur complement of the boundary system one (or indeed multiple) times. Whilst this reduced boundary system has no special structure in general, the interior system is block diagonal and can be trivially inverted. At higher orders, and particularly in three dimensions, this dramatically reduces execution time, and has meant that implicit DG methods have struggled to compete from the perspective of computational efficiency.

Recently however, the hybridized DG (HDG) method, introduced by Cockburn et al. [15] attempts to address this issue by reducing the size of the linear system to be solved in exchange for an additional cost incurred during its construction. In essence, the HDG method applies a static condensation technique within the DG framework, so that the only globally coupled degrees of freedom are those located on the mesh skeleton or trace space, greatly reducing the global system size. One additional benefit of the HDG method is its superconvergence property, whereby a solution obtained at polynomial order p can converge with order $p + 2$ through the application of a local post-processing technique on each element.

The HDG method has proven to be a popular method and has, in recent years, been applied in the context of steady-state diffusion [9, 10, 16, 18], Maxwell's equations [32, 33, 36], convection-diffusion problems [7, 13, 34, 35], linear elasticity [45], Timoshenko beam model [4, 5], elastodynamics [37], Stokes equations [12, 17, 19, 34], compressible [26, 49] and incompressible Navier-Stokes, and Oseen equations [6, 37, 39, 40]. We note, however, that the majority of these works focus either on the theoretical aspects of the method such as formulation and analysis for a specific equation type, or the specific benefits such as accurately captured solution features that the HDG method can offer.

The goal of our work is to perform an assessment of the performance of the HDG method in 3D and compare it with the CG method, which can be considered the performance "meter stick" in the realm of numerical methods for elliptic PDEs. On one hand, such a comparison will provide scientific computing practitioners with the HDG method performance information and guide them in the choice of the method to solve a given problem; on the other hand, our work will serve as a feedback for numerical computing theoreticians by improving the understanding of the benefits and shortcomings of the theoretically proven properties under a set of specific implementation choices.

To our knowledge, this is the first work that encompasses a broad set of factors and comparison categories: this study (a) is performed in 3D; (b) uses various element types; (c) accounts for the use of local postprocessing in the HDG method; (d) accounts for static condensation in the CG discretisation; and (e) discusses large-scale parallel performance and implementation choices.

We note that there is a limited amount of published work which documents some of these features. Of the few published studies that discuss performance-related topics, most focus on one particular aspect of the points above. For example, [41] measures scalability of the 2D HDG method for the compressible Navier-Stokes equations up to 32 cores, and [24] derives theoretical floating point operation counts for CG, DG and HDG schemes in 2D and 3D. However, from a practical perspective, there is still a pressing need to examine how these methods perform through directly measuring their execution times, in both the setting of large-scale computational resources that are needed for real-world 3D simulations, and in a manner that provides a fair comparison against existing benchmark CG discretisations.

As we see it, there are two main aspects of the "fair comparison" of two numerical methods: the "fairness" and the scope and depth of the comparison. In our case, we strive towards fairness by using the same object-oriented spectral element framework Nektar++ [3] as the foundation for the implementation of both methods, which guarantees that both solvers will have the same basic underlying functionality in terms of numerical quadrature, elemental matrix operations and linear system solvers.

In terms of the scope of this work, our intention is to study problems that are likely to arise in common use cases. We therefore focus on elliptic steady-state and parabolic time-dependent problems, as these often form the building blocks of more complex systems and can therefore give a good indicator as to the performance of each method in a wider range of applications. In particular, the following aspects are covered:

- serial performance for smaller test cases using direct solvers;
- parallel performance up to 4,096 cores for larger test cases using iterative solvers;
- the effect of structured meshes of hexahedra and tetrahedra in serial and parallel;
- the effect of unstructured meshes in parallel to assess scalability of each method; and
- the effect of HDG postprocessing to achieve superconvergence of the solution field (for the cases where superconvergence property holds).

The scope of this work therefore encompasses a wide range of potential application areas and significantly builds upon the two-dimensional results that have been previously presented.

The results presented in this paper demonstrate that, for steady-state second-order elliptic problems, the HDG method (with postprocessing and resulting superconvergence) outperforms the CG method starting at a polynomial order between one and three, depending on mesh size and elements shape. The HDG method demonstrates at worst competitive and at best superior performance (depending on the linear system solver chosen) when compared to the CG method in a time-dependent parabolic PDE setting. In parallel execution, where an iterative linear system solver is used, the HDG method is significantly outperformed by the CG method even when postprocessing is utilised. This performance gap indicates the need for the robust preconditioners developed specifically for the HDG method, a research area that is beginning to be addressed (see [14, 41]). The additional degrees of freedom that are present in the HDG system between common edges and vertices of faces have an additional effect on performance. However, on a per-iteration basis (with the use of postprocessing in HDG case) both methods have approximately the same performance characteristics up to large core counts.

We also note that not all of these parameters are relevant to a given application area. Superconvergence of the HDG method may not always be easily achievable, or indeed achievable at all, particularly in case of nonlinear problems. In the interests of fairness, we base our conclusions here on the best case scenario for each method. Wherever possible, we also discuss relative performance when superconvergence is not taken into account.

The paper is organized as follows. In Section 2 we describe domain partitioning, finite element spaces and polynomial expansions. This is followed by the formulations of the CG and HDG methods in Section 3. We give an outline of our parallelization strategy in Section 4. Section 5 contains numerical comparison of the CG and HDG methods performance in both serial and parallel settings. The paper is concluded by the contribution summary in Section 6.

2 Domain Partitioning, Finite Element Spaces and Polynomial Expansions

In this section we introduce the preliminaries that will be used for the CG and HDG methods formulation in Section 3. We define the partitioning of the domain in Section 2.1, the finite element spaces in Section 2.2 and the polynomial expansions used in Section 2.3.

2.1 Partitioning of the Domain

We begin by discretizing our domain. We assume $\mathcal{T}(\Omega)$ is a three-dimensional tessellation of Ω. Let $\Omega^e \in \mathcal{T}(\Omega)$ be a non-overlapping element within the tessellation such that if $e_1 \neq e_2$ then $\Omega^{e_1} \bigcap \Omega^{e_2} = \emptyset$. We denote the number of elements (or cardinality) of $\mathcal{T}(\Omega)$ by N_{el}. Let $\partial\Omega^e$ denote the boundary of the element Ω^e (i.e. $\bar{\Omega}^e \setminus \Omega^e$) and $\partial\Omega^e_i$ denote an individual face of $\partial\Omega^e$ such that $1 \leq i \leq N^e_b$ where N^e_b denotes the number of faces of element e. We then denote by Γ the set of boundaries $\partial\Omega^e$ of all the elements Ω^e of $\mathcal{T}(\Omega)$. Finally, we denote by N_Γ the number of faces (or cardinality) of Γ.

For simplicity, we assume that the tessellation $\mathcal{T}(\Omega)$ consists of conforming elements. We say that Γ^l is an *interior face* of the tessellation $\mathcal{T}(\Omega)$ if there are two elements of the tessellation, Ω^e and Ω^f, such that $\Gamma^l = \Omega^e \cap \Omega^f$ and the area of Γ^l is not zero. We say that Γ^l is a *boundary face* of the tessellation $\mathcal{T}(\Omega)$ if there is an element of the tessellation, Ω^e, such that $\Gamma^l = \Omega^e \cap \partial\Omega$ and the area of Γ^l is not zero.

As it will be useful later, let us define a collection of index mapping functions, that allow us to relate the local faces of an element Ω^e, namely, $\partial\Omega^e_1, \dots, \partial\Omega^e_{N^e_b}$, with the global faces of Γ, that is, with $\Gamma^1, \dots, \Gamma^{N_\Gamma}$. Thus, since the j-th face of the element Ω^e, $\partial\Omega^e_j$, is the l-th face Γ^l of the set of edges Γ, we set $\sigma(e,j) = l$ so that we can write $\partial\Omega^e_j = \Gamma^{\sigma(e,j)}$. Similarly, since the interior face Γ^l is the intersection of the boundaries of the two elements Ω^e and Ω^f, we set $\eta(l,+) = e$ and $\eta(l,-) = f$ so that we can write $\Gamma^l = \partial\Omega^{\eta(l,+)} \cap \partial\Omega^{\eta(l,-)}$. Here the $\pm$ convention is arbitrary.

2.2 The Finite Element Spaces

Next, we define the finite element spaces associated with the partition $\mathcal{T}(\Omega)$. To begin, for a three-dimensional problem we set

$$V_h := \{v \in L^2(\Omega): \quad v|_{\Omega^e} \in P(\Omega^e) \quad \forall\, \Omega^e \in \mathcal{T}(\Omega)\}, \tag{1a}$$

$$\mathbf{\Sigma}_h := \{\tau \in [L^2(\Omega)]^3 : \tau|_{\Omega^e} \in \mathbf{\Sigma}(\Omega^e) \quad \forall\, \Omega^e \in \mathcal{T}(\Omega)\}, \tag{1b}$$

$$\mathcal{M}_h := \{\mu \in L^2(\Gamma): \quad \mu|_{\Gamma^l} \in P(\Gamma^l) \quad \forall\, \Gamma^l \in \Gamma\}, \tag{1c}$$

where $P(\Gamma^l) = \mathcal{T}_P(\Gamma^l)$, $P(\Omega^e) = \mathcal{N}_P(\Omega^e)$ are the spaces of polynomials of total degree P defined on a standard triangular and tetrahedral regions, and $P(\Gamma^l) = \mathcal{Q}_P(\Gamma^l)$, $P(\Omega^e) = \mathcal{H}_P(\Omega^e)$ are the spaces of tensor-product polynomials of degree P defined on a standard quadrilateral and hexahedral regions correspondingly. The above polynomial spaces are defined as

$$\begin{aligned}
\mathcal{T}_P(\Gamma^l) &= \{\xi_1^p \xi_2^q; \quad 0 \le p+q \le P; (x_1,x_2) \in \Gamma^l; -1 \le \xi_1, \xi_2; \xi_1 + \xi_2 \le 1\},\\
\mathcal{Q}_P(\Gamma^l) &= \{\xi_1^p \xi_2^q; \quad 0 \le p,q \le P; (x_1,x_2) \in \Gamma^l; -1 \le \xi_1, \xi_2 \le 1\},\\
\mathcal{N}_P(\Omega^e) &= \{\xi_1^p \xi_2^q \xi_3^r; 0 \le p+q+r \le P; (x_1,x_2,x_3) \in \Omega^e; -1 \le \xi_1, \xi_2, \xi_3; \xi_1 + \xi_2 + \xi_3 \le 1\},\\
\mathcal{H}_P(\Omega^e) &= \{\xi_1^p \xi_2^q \xi_3^r; 0 \le p,q,r \le P; (x_1,x_2,x_3) \in \Omega^e; -1 \le \xi_1, \xi_2, \xi_3 \le 1\},
\end{aligned}$$

where $x_i = x_i(\xi_1,\xi_2)$ for $\mathcal{T}_P(\Gamma^l)$ and $\mathcal{Q}_P(\Gamma^l)$ and $x_i = x_i(\xi_1,\xi_2,\xi_3)$ for $\mathcal{N}_P(\Omega^e)$ and $\mathcal{H}_P(\Omega^e)$. Similarly $\mathbf{\Sigma}(\Omega^e) = [\mathcal{N}_P(\Omega^e)]^3$ or $\mathbf{\Sigma}(\Omega^e) = [\mathcal{H}_P(\Omega^e)]^3$.

2.3 Elemental Polynomial Expansion Bases

In our numerical implementation, we have applied a spectral/*hp* element discretization which is described in detail in [28]. Here we briefly describe the C^0-continuous hexahedral and tetrahedral expansions within the standard regions which we have adopted in this work. We have chosen this type of basis since it provides the C^0 continuity required for the CG method as well as allows the decomposition of these expansions into an *interior* and *boundary* modes [28, 43], which is also beneficial to the HDG method implementation.

A commonly used hierarchical C^0 polynomial expansion [28, 42] is based on the tensor product of the integral of Legendre polynomials or equivalently generalized Jacobi polynomials $P_p^{1,1}(\xi)$ such that

$$\phi_{i(pqr)}(\boldsymbol{x}(\boldsymbol{\xi})) = \psi_p^a(\xi_1)\psi_q^a(\xi_2)\psi_r^a(\xi_3) \quad 0 \le p,q,r \le P$$

where

$$\psi_p^a(z) = \begin{cases} \frac{1-z}{2} & p = 0 \\ \frac{1-z}{2}\frac{1+z}{2}P_p^{1,1}(z) & 0 < p < P \\ \frac{1+z}{2} & p = P \end{cases},$$

$\boldsymbol{\xi} = (\xi_1, \xi_2, \xi_3)$, $\boldsymbol{x} = (x_1, x_2, x_3)$ and $\boldsymbol{x}(\boldsymbol{\xi})$ represents the mapping from the standard region $\Omega_{\rm st} = \{-1 \le \xi_1, \xi_2, \xi_3 \le 1\}$ to Ω^e.

Within a tetrahedral domain a compatible C^0 expansion can also be developed and is based on an orthogonal expansion described by Sherwin and Karniadakis [43]. This C^0 expansion takes the form of a generalized tensor product

$$\phi_{i(pqr)}(\boldsymbol{x}(\boldsymbol{\xi})) = \psi_p^a(\eta_1)\psi_{pq}^b(\eta_2)\psi_{pqr}^c(\eta_3)$$

where

$$\psi_{pq}^b(z) = \begin{cases} \psi_q(z) & p = 0, 0 \le q \le P \\ \left(\frac{1-z}{2}\right)^{p+1} & 1 \le p < P, q = 0 \\ \left(\frac{1-z}{2}\right)^{p+1}\left(\frac{1+z}{2}\right)P_{q-1}^{2p+1,1}(z) & 1 \le p < P, 1 \le q + p < P \\ \psi_q(z) & p = P, 0 \le q \le P \end{cases}$$

$$\psi_{pqr}^c(z) = \begin{cases} \psi_{qr}^b(z) & p = 0, 0 \le q, r \le P \\ \psi_{pr}^b(z) & q = 0, 0 \le p, r \le P \\ \left(\frac{1-z}{2}\right)^{p+q+1} & 1 \le p, q \le P, r = 0 \\ \left(\frac{1-z}{2}\right)^{p+q+1}\left(\frac{1+z}{2}\right)P_{r-1}^{2p+2q+1,1}(z) & 1 \le p < P, 1 \le p + q + r < P \\ \psi_{pr}^b(z) & q = P, 0 \le p, r \le P \\ \psi_{qr}^b(z) & p = P, 0 \le q, r \le P \end{cases}$$

and we use a collapsed coordinate system

$$\eta_1 = 2\frac{(1+\xi_1)}{(1-\xi_2)} - 1, \qquad \eta_2 = 2\frac{1+\xi_2}{1-\xi_3} - 1, \qquad \eta_3 = \xi_3.$$

Once again, $\boldsymbol{x}(\boldsymbol{\xi})$ represents a mapping from $\Omega_{\rm st} = \{-1 \le \xi_1, \xi_2, \xi_3;\ \xi_1 + \xi_2 + \xi_3 \le 1\}$ to Ω^e. This expansion is the extension of the triangular C^0 expansion, originally proposed by Dubiner [21] and is detailed in [28,43]. The expansion used on the element faces, corresponding to the finite element space $\mathcal{M}_h$, is denoted by $\zeta_{i(pq)}$. Just as in the 3D case described above, $\zeta_{i(pq)}$ is the tensor product expansion on the quadrilateral faces and the generalized tensor product expansion on the triangular faces.

3 The CG and HDG Methods

We will formulate the CG and HDG methods for the following elliptic diffusion problem with mixed Dirichlet and Neumann boundary conditions:

$$\begin{aligned} -\nabla^2 u(\boldsymbol{x}) &= f(\boldsymbol{x}), & \boldsymbol{x} &\in \Omega, \\ u(\boldsymbol{x}) &= g_D(\boldsymbol{x}), & \boldsymbol{x} &\in \partial\Omega_D, \\ \boldsymbol{n}\cdot\nabla u(\boldsymbol{x}) &= g_N(\boldsymbol{x}), & \boldsymbol{x} &\in \partial\Omega_N, \end{aligned} \tag{2}$$

where $\partial\Omega_D \bigcup \partial\Omega_N = \partial\Omega$ and $\partial\Omega_D \bigcap \partial\Omega_N = \emptyset$. The formulation above can be generalized in many ways and lead to a variety of different systems. In light of the comprehensive implementation discussion of both methods in 2D setting, presented in [30], and considering that the extension to 3D is a trivial task we limit ourselves to a rather basic formulation/implementation discussion in this section and refer an interested reader to the original 2D paper for more details.

Both the statically condensed CG method and the HDG method can be viewed as following the same pipeline: construction of a collection of elemental (local) operators which are then judiciously assembled through a static-condensation-aware procedure to yield a global system whose solution determines the degrees of freedom on the boundary of the elements. This boundary system is significantly smaller than the full system one would solve without employing the Schur complement (the linear algebra underpinning of the static-condensation procedure) of the corresponding assembled global system. Once the solution has been obtained on the boundaries of the elements, the primary solution over each element can be determined independently through a forward-application of the elemental operators.

It is well known that the approximation u^{CG} given by the CG method is an element of the space of continuous functions in V_h satisfying

$$\begin{aligned} u^{\mathrm{CG}} &= \mathcal{I}_h(g_D) \quad \text{on } \partial\Omega_D, \\ \int_\Omega \nabla v \cdot \nabla u^{\mathrm{CG}} d\boldsymbol{x} &= \int_{\partial\Omega_N} v\, g_N ds + \int_\Omega v\, f d\boldsymbol{x}, \end{aligned} \tag{3}$$

for all test functions $v \in V_h^0$ where

$$V_h^0 = \{v \in V_h \mid v = 0 \text{ on } \partial\Omega_D\}.$$

Here $\mathcal{I}_h$ is a suitably defined interpolation operator whose image is the space of traces on $\partial\Omega_D$ of functions in V_h^0.

In order to define the HDG method we have to start by rewriting the original problem (2) in auxiliary or mixed form as two first-order differential equations by introducing an auxiliary flux variable $\boldsymbol{q} = \nabla u$. This gives us:

$$\begin{aligned} -\nabla \cdot \boldsymbol{q} &= f(\boldsymbol{x}) & \boldsymbol{x} &\in \Omega, \\ \boldsymbol{q} &= \nabla u(\boldsymbol{x}) & \boldsymbol{x} &\in \Omega, \\ u(\boldsymbol{x}) &= g_D(\boldsymbol{x}) & \boldsymbol{x} &\in \partial\Omega_D, \\ \boldsymbol{q} \cdot \boldsymbol{n} &= g_N(\boldsymbol{x}) & \boldsymbol{x} &\in \partial\Omega_N. \end{aligned}$$

The HDG method seeks an approximation to $(u, \boldsymbol{q})$, $(u^{\mathrm{HDG}}, \boldsymbol{q}^{\mathrm{HDG}})$, in the space $V_h \times \boldsymbol{\Sigma}_h$, and determines it by requiring that

$$\sum_{\Omega^e \in \mathcal{T}(\Omega)} \int_{\Omega^e} (\nabla v \cdot \boldsymbol{q}^{\mathrm{HDG}})\, d\boldsymbol{x} - \sum_{\Omega^e \in \mathcal{T}(\Omega)} \int_{\partial\Omega^e} v\, (\boldsymbol{n}^e \cdot \widetilde{\boldsymbol{q}}^{\mathrm{HDG}})\, ds = \sum_{\Omega^e \in \mathcal{T}(\Omega)} \int_{\Omega^e} v\, f\, d\boldsymbol{x}, \tag{5a}$$

$$\sum_{\Omega^e \in \mathcal{T}(\Omega)} \int_{\Omega^e} (\boldsymbol{w} \cdot \boldsymbol{q}^{\mathrm{HDG}})\, d\boldsymbol{x} = - \sum_{\Omega^e \in \mathcal{T}(\Omega)} \int_{\Omega^e} (\nabla \cdot \boldsymbol{w})\, u^{\mathrm{HDG}}\, d\boldsymbol{x} + \sum_{\Omega^e \in \mathcal{T}(\Omega)} \int_{\partial\Omega^e} (\boldsymbol{w} \cdot \boldsymbol{n}^e)\, \widetilde{u}^{\mathrm{HDG}}\, ds, \tag{5b}$$

for all $(v, \boldsymbol{w}) \in V_h(\Omega) \times \boldsymbol{\Sigma}_h(\Omega)$, where the numerical traces $\widetilde{u}^{\mathrm{HDG}}$ and $\widetilde{\boldsymbol{q}}^{\mathrm{HDG}}$ are defined in terms of the approximate solution $(u^{\mathrm{HDG}}, \boldsymbol{q}^{\mathrm{HDG}})$.

The remainder of this section will be split in three parts: local problems, global formulation and postprocessing. In the first two parts we will present the formulation of the two methods side by side to highlight the similarities between the statically condensed CG method and the HDG method. In the third part we will outline the local postprocessing procedure for the HDG method.

3.1 Local Problems

By a local problem we mean the procedure through which we express the numerical solution on an element through the solution on its boundary. In other words, we want to solve the original Equation (2) on a single element under the assumption that the Dirichlet boundary conditions for that element are known.

3.1.1 The CG method

We begin by noting that, if we assume that the function λ, which belongs to the space $\mathcal{M}_h^0 = \{v \in \mathcal{M}_h \mid v \in C^0\}$ of continuous functions in $\mathcal{M}_h$, is known, the equation satisfied by the restriction of u^{CG} to an arbitrary element $\Omega^e \in \mathcal{T}_h$ is the solution of the following local problem:

$$\begin{aligned} u^{\mathrm{CG}} &= \lambda && \text{on } \partial\Omega^e, \\ \int_{\Omega^e} \nabla v \cdot \nabla u^{\mathrm{CG}} d\boldsymbol{x} &= \int_{\Omega^e} v\, f d\boldsymbol{x} && \text{for all } v \in P(\Omega^e) \text{ such that } v = 0 \text{ on } \partial\Omega^e. \end{aligned} \tag{6}$$

This formulation follows from the standard global formulation of the CG method by taking the test functions v different from zero only on the element Ω^e . This implies that, if we define by U_λ and by U_f the local solutions to (6) when $f = 0$ and when $\lambda = 0$, respectively (*i.e.* the homogeneous and heterogeneous solutions), we can write

$$u^{\mathrm{CG}} = U_\lambda + U_f. \tag{7}$$

The discrete problem represented by Equation (6) can also be recast into an elemental matrix problem. To do so, we first define

$$u^{\mathrm{CG}} = \sum_n \phi_n^e(\boldsymbol{x})\, \underline{\hat{u}}^{\mathrm{CG}}[n] = \sum_n \phi_n^b(\boldsymbol{x})\, \underline{\hat{u}}^b[n] + \sum_n \phi_n^i(\boldsymbol{x})\, \underline{\hat{u}}^i[n] \tag{8}$$

where the ϕ_n^i are functions which are defined to be zero on the element boundary $\partial\Omega^e$ and ϕ_n^b are functions that have support on the element boundaries. The array $\underline{\hat{u}}^{\mathrm{CG}}[n]$ holds the degrees of freedom (modes) of the solution; $\underline{\hat{u}}^b[n]$ and $\underline{\hat{u}}^i[n]$ holds the degrees of freedom (modes) on the boundaries and in the interior, respectively. We next introduce

$$\mathbb{L}[n,m] = \int_{\Omega^e} \nabla\phi_n \cdot \nabla\phi_m d\boldsymbol{x}, \quad \mathbb{L} = \begin{bmatrix} \mathbb{L}^{b,b} & \mathbb{L}^{b,i} \\ \mathbb{L}^{i,b} & \mathbb{L}^{i,i} \end{bmatrix}, \quad \underline{f}[n] = \int_{\Omega^e} \phi_n\, f d\boldsymbol{x} \tag{9}$$

where the superscripts on the matrix $\mathbb{L}$ correspond to the decomposition of the functions ϕ_n^e into the sets ϕ_n^b and ϕ_n^i. We can now restate Equations (6) as

$$\underline{\hat{v}}^T \mathbb{L} \underline{\hat{u}} = \underline{\hat{v}}^T \underline{f} \tag{10}$$

where $v = \sum_n \phi_n^e(\boldsymbol{x})\, \underline{\hat{v}}[n]$. Considering Equation (7), we can express U_λ and U_f in terms of their approximating expansions as follows: $U_\lambda = \sum_n \phi_n^e(\boldsymbol{x}) \underline{\hat{U}}_\lambda[n] = \sum_n \phi_n^b(\boldsymbol{x}) \underline{\hat{U}}_\lambda^b[n] + \sum_n \phi_n^i(\boldsymbol{x}) \underline{\hat{U}}_\lambda^i[n]$ and $U_f = \sum_n \phi_n^e(\boldsymbol{x}) \underline{\hat{U}}_f[n] = \sum_n \phi_n^b(\boldsymbol{x}) \underline{\hat{U}}_f^b[n] + \sum_n \phi_n^i(\boldsymbol{x}) \underline{\hat{U}}_f^i[n]$. Let $\lambda = \sum_n \phi_n^b(\partial\Omega^e) \underline{\hat{\lambda}}[n]$. By substituting these expressions into Equation (10) and solving for $\underline{\hat{U}}_\lambda$ assuming $f = 0$ with known boundaries based upon λ, and solving for $\underline{\hat{U}}_f$ assuming $\lambda = 0$ with known right-hand-side f, we arrive at (respectively):

$$\underline{\hat{U}}_\lambda = \begin{bmatrix} \mathbb{I} \\ -(\mathbb{L}^{i,i})^{-1}\mathbb{L}^{i,b} \end{bmatrix} \underline{\hat{\lambda}}, \quad \underline{\hat{U}}_f = \begin{bmatrix} 0 & 0 \\ 0 & (\mathbb{L}^{i,i})^{-1} \end{bmatrix} \underline{f}. \tag{11}$$

If we know f and λ (or equivalently $\underline{f}$ and $\underline{\hat{\lambda}}$) we would be able to construct the solution u^{CG}, and so it therefore remains to find a way to characterize λ.

3.1.2 The HDG method

We begin by assuming that the function $\lambda := \widetilde{u}^{\mathrm{HDG}} \in \mathcal{M}_h$ is known, for any element Ω^e, from the global formulation of the HDG method. The restriction of the HDG solution to the element Ω^e, $(u^e, \boldsymbol{q}^e)$ is then the function in $V_h(\Omega^e) \times \boldsymbol{\Sigma}_h(\Omega^e)$ and satisfies the following equations:

$$\int_{\Omega^e} (\nabla v \cdot \boldsymbol{q}^e)\, d\boldsymbol{x} - \int_{\partial\Omega^e} v\, (\boldsymbol{n}^e \cdot \widetilde{\boldsymbol{q}}^e)\, ds = \int_{\Omega^e} v\, f\, d\boldsymbol{x}, \tag{12a}$$

$$\int_{\Omega^e} (\boldsymbol{w} \cdot \boldsymbol{q}^e)\, d\boldsymbol{x} = -\int_{\Omega^e} (\nabla \cdot \boldsymbol{w})\, u^e\, d\boldsymbol{x} + \int_{\partial\Omega^e} (\boldsymbol{w} \cdot \boldsymbol{n}^e)\, \lambda\, ds, \tag{12b}$$

for all $(v, \boldsymbol{w}) \in V_h(\Omega^e) \times \boldsymbol{\Sigma}(\Omega^e)$. To allow us to solve the above equations locally, the numerical trace of the flux is chosen in such a way that it depends only on λ and on $(u^e, \boldsymbol{q}^e)$:

$$\widetilde{\boldsymbol{q}}^e(\boldsymbol{x}) = \boldsymbol{q}^e(\boldsymbol{x}) - \tau(u^e(\boldsymbol{x}) - \lambda(\boldsymbol{x}))\boldsymbol{n}^e \qquad \text{on } \partial\Omega^e \tag{12c}$$

where τ is a positive function. For the HDG method taking τ to be positive ensures that the method is well defined. The results in [9] indicate that a reasonable choice of τ is to be of order one. Note that τ is a function of the set of borders of the elements of the discretization; hence, it is allowed to be different per element and per edge. Thus, if we are dealing with the element whose *global number* is e, we denote the value of τ on the edge whose *local number* is i by $\tau^{e,i}$.

Similar to the CG formulation in Section 3.1.1 we denote by $(U_\lambda, \boldsymbol{Q}_\lambda)$ and $(U_f, \boldsymbol{Q}_f)$ the solution of the (12a) and (12b) local problem when $f = 0$ and when $\lambda = 0$, respectively, and define our approximation to be

$$(u^{\mathrm{HDG}}, \boldsymbol{q}^{\mathrm{HDG}}) = (U_\lambda, \boldsymbol{Q}_\lambda) + (U_f, \boldsymbol{Q}_f).$$

We note that for the HDG decomposition, unlike in the CG case, the local problem involve an approximation over $\partial\Omega^e$. However similar to the CG problem solution to (12a) and (12b) when $f = 0$ allows us to express $U_\lambda, \boldsymbol{Q}_\lambda$ in terms of λ.

Let us start by defining

$$u^e(\boldsymbol{x}) = \sum_{j=1}^{N_u^e} \phi_j^e(\boldsymbol{x})\, \underline{\hat{u}}^e[j], \qquad q_k^e(\boldsymbol{x}) = \sum_{j=1}^{N_q^e} \phi_j^e(\boldsymbol{x})\, \underline{\hat{q}}_k^e[j], \qquad \lambda^l(\boldsymbol{x}) = \sum_{j=1}^{N_\lambda^l} \zeta_j^l(\boldsymbol{x})\, \underline{\hat{\lambda}}^l[j],$$

where $u^e(\boldsymbol{x}) : \Omega^e \to \mathbb{R}$, $\boldsymbol{q}^e(\boldsymbol{x}) : \Omega^e \to \mathbb{R}^3$ and $\lambda^l(\boldsymbol{x}) : \Gamma^l \to \mathbb{R}$.

After inserting the finite expansion of the trial functions into Equations (12a) and (12b), and using the hybridized definition of the flux given in Equation (12c), the equations for the local solvers can be written in matrix form as:

$$\left[(\mathbb{D}_1^e)^T (\mathbb{D}_2^e)^T (\mathbb{D}_3^e)^T\right] \begin{bmatrix} \underline{\hat{q}}_1^e \\ \underline{\hat{q}}_2^e \\ \underline{\hat{q}}_3^e \end{bmatrix} - \sum_{l=1}^{N_b^e} \left[\widetilde{\mathbb{E}}_{1l}^e\, \widetilde{\mathbb{E}}_{2l}^e\, \widetilde{\mathbb{E}}_{3l}^e\right] \begin{bmatrix} \underline{\hat{q}}_1^e \\ \underline{\hat{q}}_2^e \\ \underline{\hat{q}}_3^e \end{bmatrix} + \sum_{l=1}^{N_b^e} \tau^{e,l} \left[\mathbb{E}_l^e\, \underline{\hat{u}}^e - \mathbb{F}_l^e\, \underline{\hat{\lambda}}^{\sigma(e,l)}\right] = \underline{f}^e \tag{13a}$$

$$\mathbb{M}^e \underline{\hat{q}}_k^e = -(\mathbb{D}_k^e)^T \underline{\hat{u}}^e + \sum_{l=1}^{N_b^e} \widetilde{\mathbb{F}}_{kl}^e\, \underline{\hat{\lambda}}^{\sigma(e,l)} \qquad k = 0, 1, 2 \tag{13b}$$

where $\underline{f}^e[i] = (\phi_i^e, f)_{\Omega^e}$ and the matrices are defined as follows:

$$\begin{aligned} \mathbb{D}_k^e[i,j] &= \left(\phi_i^e, \frac{\partial \phi_j^e}{\partial x_k}\right)_{\Omega^e}, & \mathbb{E}_l^e[i,j] &= \left\langle \phi_i^e, \phi_j^e \right\rangle_{\partial\Omega_l^e}, & \mathbb{F}_l^e[i,j] &= \left\langle \phi_i^e, \zeta_j^{\sigma(e,l)} \right\rangle_{\partial\Omega_l^e}, \\ \mathbb{M}^e[i,j] &= \left(\phi_i^e, \phi_j^e\right)_{\Omega^e}, & \widetilde{\mathbb{E}}_{kl}^e[i,j] &= \left\langle \phi_i^e, \phi_j^e n_k^e \right\rangle_{\partial\Omega_l^e}, & \widetilde{\mathbb{F}}_{kl}^e[i,j] &= \left\langle \phi_i^e, \zeta_j^{\sigma(e,l)} n_k^e \right\rangle_{\partial\Omega_l^e}. \end{aligned}$$

Note, that the choice of the trace expansion that matches the elemental expansion restricted to particular face, that is $\zeta_i^{\sigma(e,l)}(s) = \phi_{k(i)}(s)$ (which is typical of a C^0 expansion basis defined in Section 2.3), insures that $\mathbb{E}_l^e$ contains the same entries as $\mathbb{F}_l^e$ and, similarly, $\widetilde{\mathbb{E}}_{kl}^e$ contains the same entries as $\widetilde{\mathbb{F}}_{kl}^e$.

Finally, if we concatenate all the unknowns into one vector $\underline{\hat{v}}^e = (\underline{\hat{u}}^e, \underline{\hat{q}}_1^e, \underline{\hat{q}}_2^e, \underline{\hat{q}}_3^e)^T$ and introduce $\underline{w}^e = (\underline{f}^e, 0, 0, 0)^T$, we can write equations (13) as:

$$\mathbb{A}^e \underline{\hat{v}}^e + \mathbb{C}^e \underline{\hat{\lambda}}^e = \underline{w}^e. \tag{14}$$

From Equation (14) it follows that:

$$[\underline{\hat{U}}_\lambda, \underline{\hat{\boldsymbol{Q}}}_\lambda]^T = -(\mathbb{A}^e)^{-1} \mathbb{C}^e \underline{\hat{\lambda}}^e \quad \text{and} \quad [\underline{\hat{U}}_f, \underline{\hat{\boldsymbol{Q}}}_f]^T = (\mathbb{A}^e)^{-1} \underline{w}^e.$$

Again, just like in the CG method formulation, once we know λ and f, we can find u^{HDG} and $\boldsymbol{q}^{\mathrm{HDG}}$.

3.2 Global Formulation

In Sections 3.1.1-3.1.2 we have expressed the solution within an element through the solution on its boundary for both numerical methods. Now we need to find a way to characterize λ globally in order to obtain the statically condensed trace system. Our goal here is to derive a restriction of the global equation for λ to one element of the following form: $\mathbb{K}^e\underline{\hat{\lambda}}^e = \underline{F}^e$, where $\underline{F}^e$ contains the local contributions (such as forcing terms and Neumann boundary condition terms) to the right-hand-side of the global linear system. Once we have the local $\mathbb{K}^e$ matrices, we can either assemble them into the global linear system matrix or keep them "as is" together with the global-to-local map, depending on the choice of the linear system solver. If we choose to assemble the global system matrix, we can do so using the global-to-local spreading operator $\mathcal{A}$, that takes the unique trace space coefficient values $\underline{\Lambda}$ and "spreads" them to the local (elemental) face coefficients vector $\underline{\Lambda}^l$. If we denote the "portion" of $\mathcal{A}$ that corresponds to the particular element by $\mathcal{A}^e$ ($\mathcal{A}^e\underline{\Lambda} = \underline{\hat{\lambda}}^e$) then the assembled system matrix can be expressed as $\mathbf{K} = \sum_{e=1}^{|\mathcal{T}(\Omega)|}(\mathcal{A}^e)^T\mathbb{K}^e\mathcal{A}^e$ and the right hand side as $\underline{F} = \sum_{e=1}^{|\mathcal{T}(\Omega)|}(\mathcal{A}^e)^T\underline{F}^e$. We are then left with the following problem to solve: $\mathbf{K}\underline{\Lambda} = \underline{F}$.

3.2.1 The CG method

It is reasonably straightforward to see that λ is the element of the space $\mathcal{M}_h^0$ such that

$$\begin{aligned} \lambda = \mathcal{I}_h(g_D) & \qquad \text{on } \partial\Omega_D, \\ \int_\Omega \nabla U_\mu \cdot \nabla U_\lambda = \int_\Omega U_\mu\, f + \int_{\partial\Omega_N} U_\mu\, g_N & \quad \text{for all } \mu \in \mathcal{M}_h^0 \text{ such that } U_\mu = \mu \text{ on } \partial\Omega^e, \end{aligned} \tag{15}$$

where we note that U_λ is related to λ through problem (6) when $f = 0$ and U_μ is similarly related to μ. Indeed, to see that the weak formulation (15) holds, insert the expression of the approximate solution u^{CG} given by Equation (7) into the standard formulation of the CG method given by Equation (3), take the test function v to be U_μ, and note that we have $\int_\Omega \nabla U_f \cdot \nabla U_\mu = 0$, by definition of the local solutions U_f and U_μ. This last result can also be demonstrated by evaluating $\int_\Omega \nabla U_f \cdot \nabla U_\mu = \underline{\hat{U}}_f^T \mathbb{L}\underline{\hat{U}}_\mu$ using the definitions (9) and (11).

We can further highlight the connection between Equation (15) and the statically condensed CG problem by considering the elemental contribution to (15) using the matrix form we introduced above. We can express the component of problem (15) restricted to element Ω^e as

$$\underline{\hat{\mu}}^T \begin{bmatrix} \mathbb{I}, & -\mathbb{L}^{b,i}(\mathbb{L}^{i,i})^{-1} \end{bmatrix} \begin{bmatrix} \mathbb{L}^{b,b} & \mathbb{L}^{b,i} \\ \mathbb{L}^{i,b} & \mathbb{L}^{i,i} \end{bmatrix} \begin{bmatrix} \mathbb{I} \\ -(\mathbb{L}^{i,i})^{-1}\mathbb{L}^{i,b} \end{bmatrix} \underline{\hat{\lambda}} = \underline{\hat{\mu}}^T \left[\underline{g}_N^e + \underline{f}^b - \mathbb{L}^{b,i}(\mathbb{L}^{i,i})^{-1}\,\underline{\mathrm{f}}^i\right]$$

where $\mu = \sum_n \phi_n^b(\partial\Omega^e)\underline{\hat{\mu}}$ and $\underline{g}_N^e[n]$ is defined as $\langle g_N, \phi_n^b\rangle_{\partial\Omega^e\cap\partial\Omega_N}$ on the elements where $\partial\Omega^e \cap \partial\Omega_N \neq \emptyset$ and as zero otherwise. Multiplying out this equation then leads us to the standard elemental Schur complement formulation of the statically condensed problem for $\underline{\hat{\lambda}}$:

$$\left[\mathbb{L}^{b,b} - \mathbb{L}^{b,i}(\mathbb{L}^{i,i})^{-1}\mathbb{L}^{i,b}\right]\underline{\hat{\lambda}} = \underline{g}_N^e + \underline{f}^b - \mathbb{L}^{b,i}(\mathbb{L}^{i,i})^{-1}\,\underline{\mathrm{f}}^i, \tag{16}$$

which can then be assembled into the global trace linear system $\mathbf{K}\underline{\Lambda} = \underline{F}$, where

$$\mathbf{K} = \sum_{e=1}^{|\mathcal{T}(\Omega)|} (\mathcal{A}^e)^T \mathbb{K}^e \mathcal{A}^e = \sum_{e=1}^{|\mathcal{T}(\Omega)|} (\mathcal{A}^e)^T \left[\mathbb{L}^{b,b} - \mathbb{L}^{b,i}(\mathbb{L}^{i,i})^{-1}\mathbb{L}^{i,b}\right] \mathcal{A}^e$$

and

$$\underline{F} = \sum_{e=1}^{|\mathcal{T}(\Omega)|} (\mathcal{A}^e)^T \left[\underline{g}_N^e + \underline{f}^b - \mathbb{L}^{b,i}(\mathbb{L}^{i,i})^{-1}\,\underline{\mathrm{f}}^i\right].$$

3.2.2 The HDG method

To determine λ in case of the HDG method, we require that the boundary conditions be weakly satisfied *and* that the normal component of the numerical trace of the flux $\widetilde{\boldsymbol{q}}$ given by Equation (12c) be single valued. This renders this numerical trace *conservative*, a highly-valued property for this type of methods; see [2].

So, we say that λ is the element of $\mathcal{M}_h$ such that

$$\lambda = \mathsf{P}_h(g_D) \qquad \text{on } \partial\Omega_D, \tag{17a}$$

$$\sum_{\Omega_e \in \mathcal{T}_h} \int_{\partial\Omega^e} \mu\, \widetilde{\boldsymbol{q}}^e \cdot \boldsymbol{n}^e = \int_{\partial\Omega_N} \mu\, g_N, \tag{17b}$$

for all $\mu \in \mathcal{M}_h^0$ such that $\mu = 0$ on $\partial\Omega_D$. Here P_h denotes the L^2-projection into the space of restrictions to $\partial\Omega_D$ of functions of $\mathcal{M}_h$.

After defining $\underline{g_N}^l[i]$ to be $\left\langle g_N, \zeta_i^l \right\rangle_{\Gamma^l \cap \partial\Omega_N}$ on Neumann boundary faces and zero otherwise, we can write Equation (17b) restricted to a single face in the matrix form:

$$\left[\tilde{\bar{\mathbb{F}}}_1^{l,e}\, \tilde{\bar{\mathbb{F}}}_2^{l,e}\, \tilde{\bar{\mathbb{F}}}_3^{l,e}\right] \begin{bmatrix} \underline{\hat{q}}_1^e \\ \underline{\hat{q}}_2^e \\ \underline{\hat{q}}_3^e \end{bmatrix} + \left[\tilde{\bar{\mathbb{F}}}_1^{l,m}\, \tilde{\bar{\mathbb{F}}}_2^{l,m}\, \tilde{\bar{\mathbb{F}}}_3^{l,m}\right] \begin{bmatrix} \underline{\hat{q}}_1^m \\ \underline{\hat{q}}_2^m \\ \underline{\hat{q}}_3^m \end{bmatrix} + (\tau^{e,i} + \tau^{m,j})\bar{\mathbb{G}}^l \underline{\hat{\lambda}}^l - \tau^{e,i}\bar{\mathbb{F}}^{l,e}\underline{u}^e - \tau^{m,j}\bar{\mathbb{F}}^{l,m}\underline{u}^m = \underline{g_N}^l,$$

where we are assuming that $l = \sigma(e,i) = \sigma(m,j)$, that is, that the elements e and m have the common internal face Γ^l. The matrices are defined as follows:

$$\bar{\mathbb{F}}^{l,e}[i,j] = \left\langle \zeta_i^l, \phi_j^e \right\rangle_{\Gamma^l} \qquad \tilde{\bar{\mathbb{F}}}_k^{l,e}[i,j] = \left\langle \zeta_i^l, \phi_j^e n_k^e \right\rangle_{\Gamma^l} \qquad \bar{\mathbb{G}}^l[i,j] = \left\langle \zeta_i^l, \zeta_j^l \right\rangle_{\Gamma^l}.$$

Using the notation introduced in Section 3.1.2 Equation (17b) for a single face can be written in a more compact form as:

$$\mathbb{B}^e \underline{v}^e + \mathbb{G}^e \underline{\hat{\lambda}}^e + \mathbb{B}^m \underline{v}^m + \mathbb{G}^m \underline{\hat{\lambda}}^m = \underline{g}_N^l. \tag{18}$$

If we sum the face contributions from Equation (18) over all the faces in the mesh, concatenate the individual face Neumann conditions $\underline{g}_N^l$ into $\underline{g}_N$ and replace $\underline{\hat{\lambda}}^e$ by $\mathcal{A}^e\underline{\Lambda}$ we get

$$\sum_{e=1}^{|\mathcal{T}(\Omega)|} (\mathcal{A}^e)^T \left[\mathbb{B}^e \underline{v}^e + \mathbb{G}^e \mathcal{A}^e \underline{\Lambda}\right] = \underline{g}_N. \tag{19}$$

The last step is to express $\underline{v}^e$ from Equation (14) (the local problem) and plug it into Equation (19) above. This will give us our global system for the trace unknowns:

$$\mathbf{K}\underline{\Lambda} = \underline{F}, \tag{20}$$

where

$$\mathbf{K} = \sum_{e=1}^{|\mathcal{T}(\Omega)|} (\mathcal{A}^e)^T \mathbb{K}^e \mathcal{A}^e = \sum_{e=1}^{|\mathcal{T}(\Omega)|} (\mathcal{A}^e)^T \left[\mathbb{G}^e - \mathbb{B}^e(\mathbb{A}^e)^{-1}\mathbb{C}^e\right] \mathcal{A}^e$$

and

$$\underline{F} = \underline{g}_N - \sum_{e=1}^{|\mathcal{T}(\Omega)|} (\mathcal{A}^e)^T \mathbb{B}^e (\mathbb{A}^e)^{-1}\underline{w}. \tag{21}$$

3.3 The HDG postprocessing

To end the presentation of the HDG method, we highlight how one can postprocess the approximate solution to obtain a new approximation of the scalar variable with the order of convergence increased by one when the polynomial degree P is larger than zero. For a detailed discussion of the postprocessing in the context of the HDG method, including the flux postprocessing, we refer the reader to [16] and the references therein. Below we will give a brief outline of the postprocessing technique we used in our implementation.

The postprocessed numerical solution u_*^e on the element Ω^e is the function in $\mathcal{T}_{P+1}(\Omega^e)$ defined by

$$(\nabla u_*^e, \nabla w)_{\Omega^e} = (\boldsymbol{q}^e, \nabla w)_{\Omega^e} \quad \forall \mathcal{T}_{P+1}(\Omega^e), \tag{22a}$$

$$(u_*^e, 1)_{\Omega^e} = (u^e, 1)_{\Omega^e}. \tag{22b}$$

We note that, if $\boldsymbol{q}^e$ converges with order $P+1$ and the average of u^e on each element superconverges with order $P+2$, then the postprocessing u_*^e converges with order $P+2$, when $P \geq 1$. The conditions for superconvergence imposed upon the finite element space depending on the element shape are discussed in [18]. Numerical results demonstrating the L^2 errors before and after postprocessing for hexahedral and tetrahedral elements (see Section 5) indicate that the technique works as intended for both types of elements. In our implementation we have expressed $\boldsymbol{q}^e$, used in Equation (22a), through λ^e and u^e using Equation (12b). This was done to avoid solving local problems for $\boldsymbol{q}^e$ in case only scalar variable u^e is required.

4 Parallel Implementation

In a three dimensional formulation, both CG and HDG discretisations lead to the creation of matrix systems which, for even moderately sized meshes at low polynomial orders, quickly become too large to store or solve in a practially useful amount of time on a single processor. Strategies for dividing the problem amongst multiple processors are therefore essential to consider simulations for even relatively small problems. The goal of this section is to describe a parallelisation strategy for both HDG and CG systems, and through the examination of a model unsteady diffusion problem, compare the relative performance and scalability of these methods across a large number of processors in the following section.

4.1 Parallelisation strategy

Whilst finite element methods are conceptually simple to parallelise owing to the existing elemental decomposition of the domain, there are many numerical and technical hurdles to overcome when implementing an efficient and scalable parallel algorithm. Namely we must partition the domain ‘evenly’, so that each process receives a computationally equal-sized part of the whole problem, and also design an algorithm to solve the resulting matrix systems in parallel without a single process needing to store the entirity of the spatial operator.

The focus of this work is on the latter problem. However we note that the approach taken here is to construct the dual graph of the mesh, where each node of the graph represents an element and edges connecting nodes denote the connection of two elements through the trace space. This graph can then passed through a partitioner such as [8, 29] in order to determine an appropriate number of subgraphs, each of which forms the mesh partition for a given process. When the mesh is hybrid (i.e. heterogeneous element type) we assign an appropriate weight to each element, such as the number of local degrees of freedom, so that in the resulting matrix inversion each process receives a computationally equal portion of the full problem.

The formulation of both CG and HDG algorithms leads to the construction of a large sparse matrix system

$$\mathbf{K}\underline{\Lambda} = \underline{F} \tag{23}$$

for the statically condensed variable λ, where $\mathbf{K}$ represents the discrete Laplacian operator. Parallel algorithms for inverting this system may be broadly categorised as either direct [1,46], in which the matrix is inverted through a procedure such as Cholesky decomposition, or iterative, whereby the solution is obtained through repeatedly applying the operator $\mathbf{K}$ in some manner in order to converge to a solution.

In parallel, iterative algorithms such as the preconditioned conjugate gradient method [20] have a distinct advantage over direct methods in that we can leverage the finite element construction of $\mathbf{K}$ in order to more readily parallelise the problem. Let $\widetilde{\mathbf{K}} = \bigoplus_{e=1}^{N_{\text{el}}} \mathbb{K}^e$ denote the block-diagonal matrix with each block $\mathbb{K}^e$ being the local action of $\mathbf{K}$ on an element Ω^e. Then, by utilising the global-to-local operator $\mathcal{A}$ we can rewrite (23) as

$$\mathbf{K}\underline{\Lambda} = [\mathcal{A}^\top \widetilde{\mathbf{K}} \mathcal{A}]\underline{\Lambda},$$

where $\underline{\Lambda}^l = \mathcal{A}\underline{\Lambda}$ is the vector of local coefficients of $\underline{\Lambda}$.

The use of this assembly mapping allows us to readily parallelise an iterative solve of either the CG or HDG system. In order to calculate the matrix multiplication $\underline{u} = \mathbf{K}\underline{v}$ we apply the following procedure:

1. Prior to the solve, each process p constructs the matrix $\widetilde{\mathbf{K}}_p$: the direct sum of the local elemental matrices contained in the partition belonging to this process.

2. When we are required to perform a matrix multiplication applied to a vector of local coefficients $\underline{v}_p^l$, we first calculate $\underline{u}_p^l = \widetilde{\mathbf{K}}_p \underline{v}_p^l$.

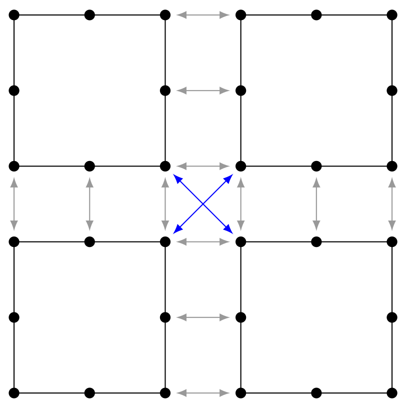

Figure 1: Illustration of communication patterns in CG and HDG, where each element is on a different process. HDG communication (gray arrows only) requires only pairwise communication. However CG requires communication between shared vertices (blue and gray arrows).

3. Each process then applies a local assembly operation $\mathcal{A}_p^\top$ to determine the unique coefficients $\underline{u}_p = \mathcal{A}_p^\top \underline{u}_p$ which lie on the process.
4. Finally, each process performs an inter-process assembly operation by sending its contributions to any other processors which have common coefficients. Any contributions received from other processors are added to the vector $\underline{u}_p$.
5. At the end of this procedure, each process has the global coefficients $\underline{u}$.

The HDG method has an advantage over the CG method in this procedure, since in the formulation of the HDG method we utilise the trace space in order to decouple elements. This implies that the inter-process communication is always pairwise, whereas in the CG method, any processes which share a vertex or edge must perform a communication as illustrated in Figure 1. In the worst case, for small unstructured tetrahedral meshes this may result in an all-to-all communication if every element shares a common vertex or edge, although such scenarios are unlikely. For the unstructured examples we consider in the following section, vertex valencies of up to 44 are observed and communication patterns can therefore be very demanding.

However, CG also has the advantage of smaller elemental matrix sizes. In unsteady problems, the majority of the computational cost of the simulation comes from inverting the matrix system, which in the PCG algorithm can be broken down into the time taken for the local block matrix-vector products $\widetilde{\mathbf{K}}_p \underline{v}_p^l$ and the communication cost incurred from the assembly process.

In order to give a qualitative idea of how these matrix sizes may affect performance a priori, in Figure 2 we compare the rank of an elemental block $\mathbb{K}^e$ for the CG method (N_{CG}) compared to the HDG method (N_{HDG}) as a function of polynomial order p and element type. The two bar plots show a comparison at equal polynomial order (left) and one order lower for HDG (right) in order to show the computational effects that the postprocessing technique

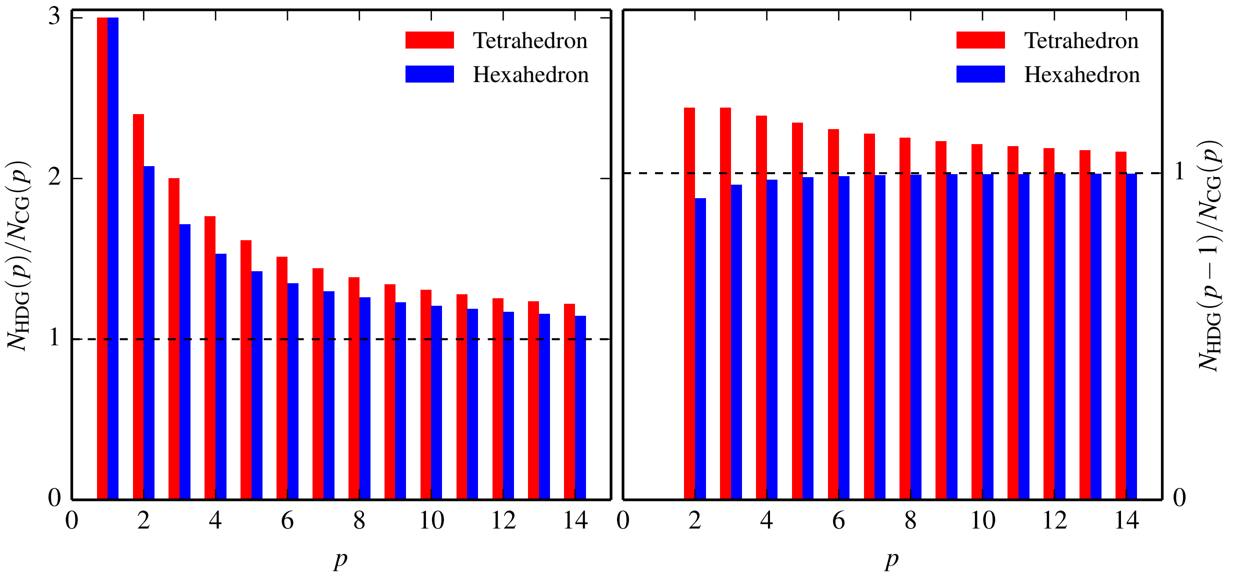

Figure 2: Ratio of HDG to CG local matrix sizes for hexahedral and tetrahedral with (right) and without (left) postprocessing taken into account. The dotted line indicates where matrix dimensions are equal.

can achieve. We clearly see that when compared at equivalent order, HDG local matrices are always larger than the equivalent boundary-condensed CG matrices. This effect is most prohibitive for tetrahedral elements, where even at polynomial order 10 tetrahedral elements still have a dimension ratio of around $\frac{4}{3}$. When HDG postprocessing is used, hexahedral matrices are marginally smaller than their CG counterparts, but tetrahedral elements still have a significant overhead.

There is, therefore, a balance to be struck between the possible increase in performance from HDG communication versus the larger matrix sizes which occur in the HDG formulation. It is also clear that for problems that do not permit superconvergence, HDG will suffer from far larger elemental matrices than its CG counterparts, particularly at low polynomial orders. We will investigate these properties by performing a series of numerical experiments to determine the weak and strong scalability of each method in the following section.

5 Numerical Results

In this section we compare the performance of the CG and HDG methods using elliptic PDE in three dimensions as a test case. We will start by comparing numerical errors for both methods while solving the steady-state Helmholtz equation (postprocessing is employed in HDG case). Next, we will discuss the serial implementation performance and its dependence on the choice of the direct linear system solver, using the steady-state Helmholtz problem as well as the time-dependent heat equation as a benchmark. We will make a few remarks regarding the performance of the preconditioned conjugate gradient (PCG) linear system solver for both numerical methods. Finally, we will conclude the results section by the parallel implementation performance discussion.

We now provide some of the experimental specifics. Tests for the serial implementation were run on the machine equipped with 80 Intel(R) Xeon(R) E7-4870 2.40GHz processors and 750 GB of RAM running OpenSUSE 12.2 (x86_64). The code was compiled using gcc 4.8.1. Tests for the parallel implementation were run on HECToR, the UK national supercomputer service, which is a Cray XE6 machine comprising of 32-core nodes consisting of two 16-core Interlagos AMD Opteron processors together with with 32GB of RAM and are connected using a Cray Gemini interconnect. Tests were performed using between 1 and 128 nodes (32 and 4,096 cores respectively).

When considering the serial implementation we have opted for a direct linear system solver under the assumption that serial codes are typically used for relatively small problems where the use of direct solver is both permissible storage-wise and effective. For the parallel implementation we utilise a PCG iterative method. Due to the existence of the extensive preconditioning-related body of work for the CG method [22, 44] and a relative scarcity of such research in case of the HDG method, we opt to use a Jacobi preconditioner for both numerical methods so that neither method is unfairly biased and communication costs due to the preconditioner are minimal. The HDG method results were obtained with parameter τ set to one.

In many test cases we will be dealing with regular meshes. Hexahedral meshes are generated through a tensor product of one-dimensional evenly spaced segments, and tetrahedral meshes are generated by splitting each hexahedron into six tetrahedra. We adopt the use of an abbreviated notation n^3 and $n^3 \times 6$ instead of more commonly used $n \times n \times n$ and $n \times n \times n \times 6$ to denote regular hexahedral and tetrahedral meshes respectively.

Remark 1 *We note that the performance of a numerical method depends not only on the choice of linear solver but also on some of the optimization choices that were made regarding the method implementation. In order to assemble a finite element operator or evaluate the result of such operator action, one can consider one of the three strategies:* global *(based on the global interpretation of the spectral/hp finite element method),* sum-factorization *(that exploits the tensorial nature of the basis) and* local matrix *approach. Any of the three strategies can become optimal for a particular problem choice and parameter range. Generally, since higher order spectral/hp finite element methods favor sum-factorization performance-wise, it is our optimization strategy of choice for this section. More details regarding the above optimization strategies can be found in [48].*

Remark 2 *In [18], the space for the numerical flux $\boldsymbol{q}$ that ensures superconvergence of the HDG method on a hexahedral element is slightly larger than the standard tensor-product space $\boldsymbol{Q}^k$ (for details, see Table* 6 *in [18]). In our implementation, we use the $\boldsymbol{Q}^k$ space to represent the flux variable and still observe superconvergence properties on hexahedral elements numerically.*

5.1 Postprocessing, Errors and Convergence Orders

To verify each method, we begin by examining the L^2 errors for both the CG and HDG methods, with the use of postprocessing in the latter case. As a test case we use a Helmholtz

N_{CG}	Hexahedra		Tetrahedra		N_{HDG}
	L^2_{CG}	L^2_{HDG}	L^2_{CG}	L^2_{HDG}	
3	1.944e-02	8.301e-02	1.686e-01	1.323e-01	2
4	2.126e-03	6.046e-03	5.522e-02	4.367e-02	3
5	1.833e-04	3.449e-04	1.673e-02	1.373e-02	4
6	1.325e-05	2.504e-05	4.431e-03	3.981e-03	5
7	8.201e-07	1.460e-06	1.085e-03	9.973e-04	6
8	4.460e-08	7.863e-08	2.388e-04	2.283e-04	7
9	2.151e-09	3.828e-09	4.707e-05	4.562e-05	8

Table 1: L^2 errors of the scalar variable u for CG and HDG methods for Helmholtz equation on 9^3 hexahedral mesh and $6^3 \times 6$ tetrahedral mesh.

equation of the form

$$\begin{aligned} \nabla^2 u(\boldsymbol{x}) - \mu u(\boldsymbol{x}) &= f(\boldsymbol{x}) \qquad \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x}) &= g_D(\boldsymbol{x}) \quad \boldsymbol{x} \in \partial\Omega_D, \end{aligned}$$

where $\mu = 1$, $\Omega = [0,1]^3$ and $f(\boldsymbol{x})$ and $g_D(\boldsymbol{x})$ are selected to give an exact solution of the form

$$u(\boldsymbol{x}) = \sin(5\pi x)\sin(5\pi y)\sin(5\pi z).$$

We first measure the L^2 errors of the scalar variable u for the CG method and the postprocessed L^2 errors of u for the HDG method on regular tetrahedral and hexahedral meshes comprising of 1296 and 729 elements respectively. Corresponding results are presented in Table 1, where N_{CG} and N_{HDG} denote the number of 1D modes in each direction of the tensorial 3D expansion. The results for the HDG method are shifted by one polynomial order for the ease of comparison and to reflect the superconvergence property of the HDG method. We can see that the postprocessing technique indeed raises the convergence order by one and the HDG method with polynomials of order P produces errors comparable with those produced by the CG method with polynomials of order $P+1$.

5.2 Serial performance

Having verified each method we now analyse the timing data in order to measure the performance of each method. We start by considering the time it takes to solve the steady-state Helmholtz problem using a serial implementation of both methods. Execution time measured includes all the stages of a typical finite element code: we start by loading the mesh from a file and initializing all the data structures and end by the numerical solution evaluation at the quadrature points. In case of the HDG method, the postprocessing procedure and evaluation of the postprocessed solution is included in the runtime. We note that the serial implementation of the steady-state problem postprocessing accounts for around $20-30\%$ of the execution time for $P=1$, depending on the mesh used. This quickly drops below 10% as the polynomial order increases and levels off at around $3-5\%$ of the total execution time at

N_{CG}	DSC			DMSC			N_{HDG}
	CG	HDG	$\frac{\mathrm{HDG}}{\mathrm{CG}}$	CG	HDG	$\frac{\mathrm{HDG}}{\mathrm{CG}}$	
3	4.92	6.07	1.24	7.46	4.96	0.66	2
4	80.96	48.44	0.60	48.32	31.51	0.65	3
5	564.31	243.17	0.43	297.90	169.73	0.57	4
6	2661.04	947.12	0.36	1103.36	791.51	0.72	5
7	8765.75	2709.26	0.31	3397.99	2642.24	0.78	6
8	25450.47	6917.71	0.27	8789.20	7051.14	0.80	7
9	58228.23	16177.59	0.28	19580.75	17786.10	0.91	8

Table 2: Execution time for the CG and HDG methods on 9^3 hexahedral mesh, using direct static condensation (DSC) and direct multi-level static condensation (DMSC) solvers.

higher polynomial orders. The run time contribution of the postprocessing will be smaller in the case of a time-dependent problem, as it is done only once, and in case the of a parallel implementation since there is no inter-element dependency for this operation.

The statically condensed linear system is solved directly using the Cholesky factorisation implementation in LAPACK and dense format matrices. Before assembling the matrix, we reorder using the reverse Cuthill-McKee algorithm to reduce the bandwidth of the system and the resulting factorisation. We have found that this combination of dense format and RCM is efficient, often outperforming sparse-format approaches due to the lower matrix sparsity found at higher polynomial orders.

Since in the previous section it was demonstrated that the postprocessing does increase the numerical solution order by one, we shift the timing results of the CG method by one; that is, we for example compare the CG solution of order 5 with the HDG solution of order 4. However, from this table, one may also infer relative performance by looking at offset rows for problems where superconvergence is not available.

Runtime data is presented in Tables 2 and 3 for regular hexahedral and tetrahedral meshes using both direct static condensation (DSC) and direct multi-level static condensation (DMSC) solution strategies. We apply the multi-level algorithm in a local fashion as described in [43], so that the global system is constructed at only the lowest level. Time is measured in seconds and $\frac{\mathrm{HDG}}{\mathrm{CG}}$ denotes the ratio of corresponding runtimes. As these results are calculated in serial, we deliberately restrict our results to consider polynomial orders in the range $P \leq 8$. As the runtimes indicate, very high polynomial orders quickly become computationally intractable on a single processor.

Analyzing the results presented in Tables 2 and 3 we can see that superconvergence achieved through postprocessing allows the HDG method to be competitive with the CG method and outperform the latter as the polynomial order increases. In case of the DMSC solver, the HDG method consistently outperforms CG from a polynomial order of one, whereas for the DSC solver the HDG method outperforms the CG method from the third or fourth polynomial degree, depending on the element type. We do however reiterate that the observed behavior is only valid for the direct solve of the statically condensed linear system.

The use of multi-level static condensation results in some interesting performance obser-

N_{CG}	DSC			DMSC			N_{HDG}
	CG	HDG	$\frac{\mathrm{HDG}}{\mathrm{CG}}$	CG	HDG	$\frac{\mathrm{HDG}}{\mathrm{CG}}$	
3	0.73	3.28	4.52	3.46	2.92	0.84	2
4	4.86	12.29	2.53	14.02	9.05	0.65	3
5	42.04	46.71	1.11	59.39	29.30	0.49	4
6	220.04	144.82	0.66	154.89	92.03	0.59	5
7	819.23	399.07	0.49	777.43	297.04	0.38	6
8	2282.90	885.17	0.39	2057.15	796.64	0.39	7
9	5747.78	1852.25	0.32	4081.03	1712.52	0.42	8

Table 3: Execution time for the CG and HDG methods on $6^3 \times 6$ tetrahedral mesh, using direct static condensation (DSC) and direct multi-level static condensation (DMSC) solvers.

N_{CG}	DSC			DMSC			N_{HDG}
	CG	HDG	$\frac{\mathrm{HDG}}{\mathrm{CG}}$	CG	HDG	$\frac{\mathrm{HDG}}{\mathrm{CG}}$	
3	3.3687	3.7979	1.13	1.9001	2.3727	1.25	2
4	22.1862	19.2747	0.87	7.4255	6.2906	0.85	3
5	106.7327	77.9770	0.73	25.8663	20.8083	0.80	4
6	311.8601	220.7129	0.71	82.3704	75.9543	0.92	5
7	831.7730	378.4930	0.46	164.9608	157.2287	0.95	6
8	2148.8804	1020.5737	0.47	313.3912	354.5623	1.13	7
9	3512.1737	1916.7294	0.55	594.2661	637.5448	1.07	8

Table 4: Average time taken to perform 100 timesteps of the diffusion equation for the CG and the HDG methods on a 9^3 hexahedral mesh. Direct static condensation (DSC) and direct multi-level condensation (DMSC) solvers are used. Time is measured in seconds.

vations. Overall, it is clear that CG benefits far more from using DMSC than HDG. This performance difference can be attributed to the method that is used to determine the degrees of freedom that form in each level of static condensation. We first construct a graph that represents the mesh connectivity of the global system. The graph is then passed to a nested bisection algorithm within the partitioning software METIS [29]. The resulting separatrix is then inverted in a bottom-up manner as described in [47] to form each level of static condensation.

In the HDG method, the coupling of connectivity through element faces, but not edges and vertices, means that this graph has a far less complex structure when compared to its CG counterpart. In turn, the nested bisection algorithm does not recurse to as many levels as can be achieved in the CG method. Since the linear system to be solved resides at the lowest level, the HDG formulation results in an increased number of degrees of freedom at this level when compared to the CG. Indeed in severe cases, the lack of recursion means that the overheads of having multiple levels can sometimes increase execution time. An example of this can be seen in the the simple hexahedral mesh of Table 2 at $N_{\mathrm{HDG}} = 6$ and 7, where HDG with DMSC has a longer runtime than DSC.

5.3 Unsteady diffusion equation

In the case of an unsteady equation, setup costs are often negligible when compared to the number of timesteps required to produce a solution up to the final required time.

The exemplar elliptic PDE used for these tests is the traditional diffusion equation for a scalar variable u,

$$\frac{\partial u}{\partial t} - \nabla^2 u = 0, \tag{24}$$

which occurs frequently in the modelling of various physical phenomena, and thus can give an indication of how the CG and HDG methods perform in the setting of a more complex system, such as operator splitting schemes for the Navier-Stokes equations [27].

As in the previous section we take the domain to be a cube $[0,1]^3$ and enforce homogeneous Dirichlet boundary conditions. For the purposes of error comparison we utilise an exact solution and initial condition of the form

$$u(\boldsymbol{x},0) = \sin(\pi x)\sin(\pi y)\sin(\pi z), \quad u(\boldsymbol{x},t) = e^{-3\pi^2 t}u(\boldsymbol{x},0).$$

Since the initialization (and postprocessing for the HDG method) are performed only once for each simulation, and the timestep size as well as the final time may vary from simulation to simulation, we will compare the time each method requires to perform N timesteps, rather than the total simulation runtime. In this particular case we have used the second-order diagonally implicit Runge-Kutta (DIRK2) [23] scheme for time integration, the timestep was taken to be 10^{-6} to minimise temporal error and we run the simulation for 1000 timesteps of which the first 100 are discarded to exclude the initialization time from our comparison. We average the time it takes to perform 100 timesteps over the remaining 900 steps.

Whilst HDG postprocessing time is excluded from the analysis, the orders of both methods are shifted by one with respect to one another to account for HDG superconvergence as in the previous section. Results for the 9^3 hexahedral mesh are presented in Table 4.

We observe that under these conditions and factoring out the initialization and post-processing time, the HDG method outperforms the CG method when DSC solver is used. Each method exhibits similar performance characteristics when the DMSC solver is used: the ratio of runtimes is within $1.00 \pm 10\%$ range for higher polynomial orders. We note that for the unsteady diffusion equation, we are not inverting the full linear system matrix at each time-integration step (or rather stage, since we are using DIRK2 scheme) but do the backward/forward substitution using the precomputed Cholesky factors. This accounts for the difference in solve time ratios ($\frac{HDG}{CG}$) between the steady-state and unsteady cases.

Also, it is important to note that even with minor modifications to equation (24), one can lose the superconvergence property. In this case, we observe that the CG system is significantly faster than that of the HDG scheme, by a factor between 2 and 4 depending on polynomial order.

5.4 Iterative linear system solver

We conclude the discussion of the serial implementation performance by making a few remarks on the use of the iterative solver, namely PCG with diagonal preconditioner, for both numerical methods. Based on the numerical experiments performed, there are key points that can be observed:

1. There is a dependence between the HDG method parameter τ, that penalizes the jump in the flux $\boldsymbol{q}$ on the face shared by the two neighbor elements, and the number of PCG iterations required to achieve a given tolerance.

2. While using just a diagonal preconditioner for the HDG and the CG methods, we can observe that the statically condensed linear system matrix for the CG method requires substantially fewer iterations for PCG to converge than its HDG counterpart. This illustrates that the HDG method requires a specifically designed preconditioner to be competitive in the realm of iterative solvers.

Let us consider the example of the data from which the above conclusions were drawn. In Table 5 we provide the number of iterations of the PCG solver required to solve a statically condensed linear system to within a relative tolerance of $\epsilon = 10^{-9}$. Again, the CG method polynomial order is shifted by one to account for HDG postprocessing. To demonstrate the reliance of the iteration count on τ, we vary τ between 1 and 10,000 and compare this against the CG method with a Jacobi preconditioner used for both methods. We observe that there is a slight reduction (followed by a growth) in the iteration count for the HDG method with the parameter τ value increase with an optimal value around 100. Even with the one order shift between the two methods, the conditioning of the linear system matrix produced by the HDG method is much worse than in case of its CG counterpart. This therefore calls for the development of an efficient HDG preconditioner as future work, in the same manner as [38] has done for DG methods for the compressible Navier-Stokes equations.

To emphasise this point, in Table 6 we consider some alternative forms of preconditioner for both HDG and CG systems. For each element of the trace, we collect the matrix entries of the global system $\boldsymbol{\Lambda}$, invert each block and assemble them in a block diagonal matrix to

N_{HDG}	Value of τ in the HDG method					CG	N_{CG}
	1	10	100	1,000	10,000		
2	130	133	168	305	494	51	3
3	323	317	327	555	1465	98	4
4	520	514	501	601	1326	148	5
5	766	753	715	758	1379	197	6
6	959	951	930	934	1345	247	7
7	1210	1189	1170	1132	1428	303	8
8	1422	1400	1366	1360	1426	357	9

Table 5: Number of iterations taken for a PCG solve with diagonal preconditioner to converge to within a relative tolerance of $\epsilon = 10^{-9}$. Both the CG method and HDG method with various values of parameter τ are considered on a $5^3 \times 6$ tetrahedral mesh.

form a block-Jacobi preconditioner. This leads to very little improvement for the CG system, but for the HDG significantly improves the iteration count. To further highlight the effect of the preconditioner, we also consider two choices for the CG scheme: a low energy block preconditioner, in which a basis transformation is applied to make the resulting global system more diagonally-dominant, and a linear space conditioner for the coarse linear finite element space which is combined with the low energy preconditioner. With these choices we see even more improvement over the HDG system.

However, each of these choices has different costs in terms of set-up, communication and the action of the preconditioner itself on the global coefficients. For example, the HDG system needs to use a more expensive block preconditioner to attain the same iteration count as the Jacobi-preconditioned CG system. This is important because in the following section, we will report timings of the matrix solve which are performed using the iterative solver.

Therefore, for the purposes of drawing a fair comparison in these experiments, we need a metric which gives a sense of relative cost between the schemes. We will therefore use a simple Jacobi preconditioner for both methods, and measure time which is averaged over both a number of timesteps and also the total number of PCG iterations required to converge to the solution. This per-iteration timing value gives an indication as to how each method would perform in the presence of an optimal preconditioner, and also allows us to consider the effects of communication when taking into account the parallel matrix-vector multiplications.

5.5 Parallel performance

We begin our discussion of the parallel HDG performance results by first describing the test problem that has been used to benchmark the HDG scheme. Firstly, we posit that the use of a parallel solver is most likely to occur in time-dependent problems. Our tests therefore focus on this aspect and do not include any set-up costs which may be associated with matrix construction. As before then, we examine the properties of the unsteady diffusion solver as defined in Section 5.3. We measure per-iteration timings as noted at the end of the previous section.

N_{HDG}	HDG block	CG block	CG low energy	CG linear space	N_{CG}
2	130	51	37	29	3
3	157	98	53	35	4
4	203	150	62	38	5
5	223	204	69	40	6
6	245	252	74	41	7
7	258	295	78	42	8
8	279	345	82	44	9

Table 6: Number of iterations taken for a PCG solve with various preconditioners to within a relative tolerance of $\epsilon = 10^{-9}$. Both the CG method and HDG method with $\tau = 1$ are considered on a $5^3 \times 6$ tetrahedral mesh.

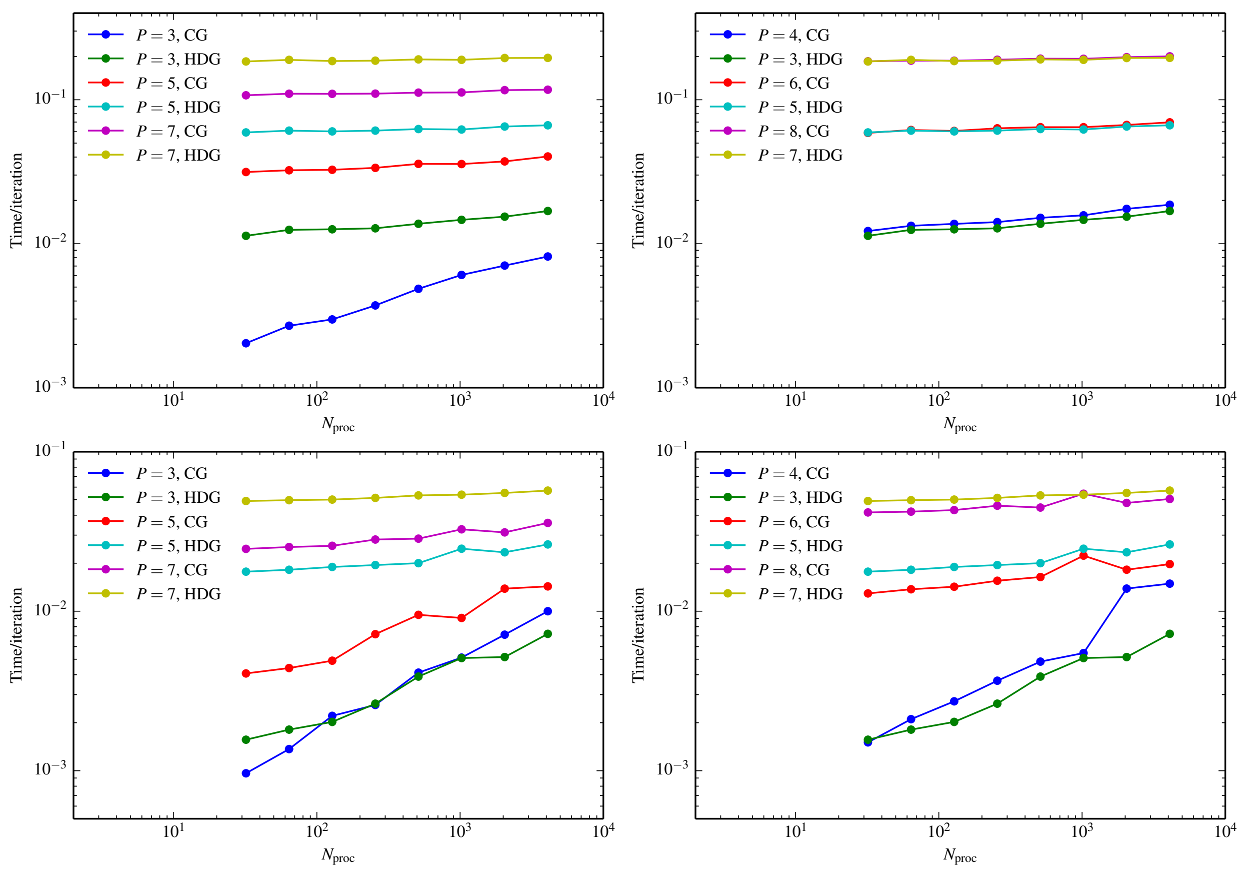

Figure 3: Weak scalings showing time per iteration for between 1 and 128 nodes (32 and 4,096) processors for hexahedral mesh (top) and tetrahedral mesh (bottom). The different figures show effects with (left) and without (right) HDG postprocessing.

5.5.1 Weak scaling

We initially examine the weak scaling of both methods, whereby as the number of processors increases, the problem size in terms of both number of elements and polynomial order is kept constant on each processor. An ideal numerical method which scales perfectly should therefore show the same runtime as the number of processors increases.

On each processor we consider two meshes: a 3^3 mesh of hexahedra and a $2^3 \times 6$ mesh of tetrahedra. The number of elements is deliberately fixed to be the same at each polynomial order in order to assess the relative communication costs of each method. At lower polynomial orders, there are very few degrees of freedom per process, and therefore communication costs are the dominant force in the timings of each simulation. Scalability will therefore be very poor. At high polynomial order however, local matrix sizes are much larger and therefore communication costs form less of the overall runtime; we can therefore expect much better scaling.

In this series of simulations we can therefore observe both whether the HDG method gains any benefits in execution time from the simpler communication patterns, and additionally if the use of postprocessing offers an advantage for the HDG scheme at equivalent polynomial order to overcome the larger matrix sizes which are inherent to the formulation.

The timings for hexahedral and tetrahedral meshes are shown in Figure 3. On the left hand side we see CG and HDG methods compared at the same order, and on the right we see timings for CG which are one polynomial order higher to observe the effects of postprocessing of the HDG results. We can draw two immediate conclusions from these data. Firstly, the impact of a larger local matrix size for the HDG method is clearly seen even at low polynomial order on the left hand side, with significantly longer runtimes. However, when postprocessing is implemented, this difference is not as pronounced for the tetrahedron and almost identical for the hexahedron as predicted by the illustration given in Figure 2.

The second observation is that asides from $P = 2$ on the tetrahedral mesh, the HDG method and CG method scale with very similar trends. We may surmise that certainly at higher polynomial orders, the use of an efficient parallel gather-scatter operation allows the CG method to scale as effectively as the HDG method in these tests. However it is interesting to note that at $P = 2$ for a tetrahedral mesh, where the CG method must communicate to a wider range of processors given the higher valency of vertices and edges than the hexahedral mesh, the HDG method does scale with a smaller slope than the CG method. This indicates the potential for effective communication performance of HDG at lower orders, and on meshes of tetrahedra where communication patterns are more complex for the CG method. This aspect is particularly important, since many problems of practical interest in complex three-dimensional geometries utilise unstructured meshes with high vertex valency. We therefore investigate this aspect in particular in the following section.

5.5.2 Strong scaling

In the second series of simulations, we perform a strong scaling test in which the global problem size is kept constant as the number of processors increases. This is a more practical test, in the sense that a given problem is more likely to be divided onto more processors by an end-user in order to obtain a decrease in execution time. It is also more difficult

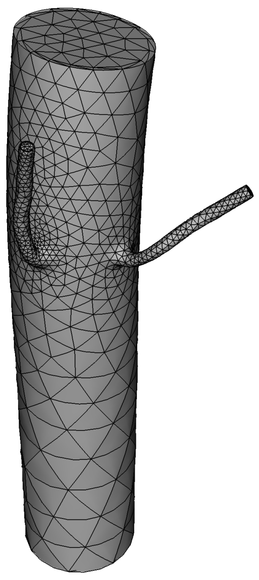
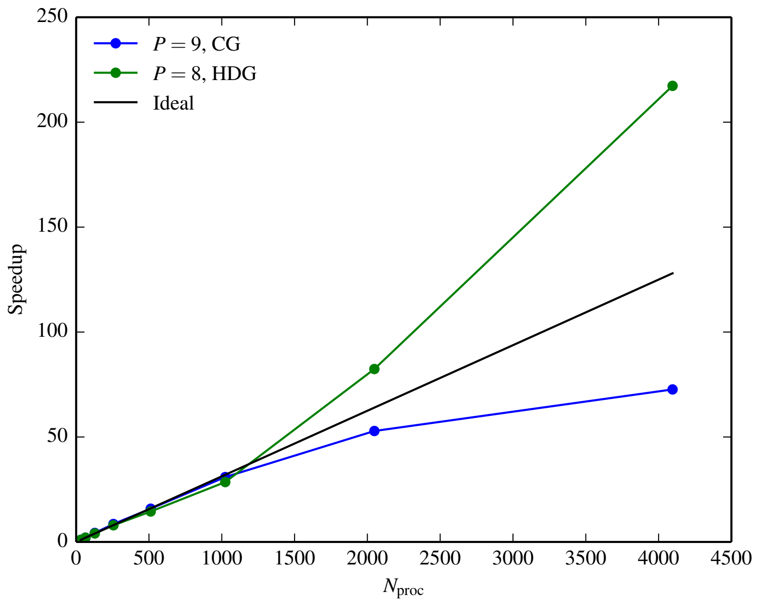

Figure 4: Strong scaling of time-dependent diffusion problem between 1 and 128 nodes (32 and 4,096 processors respectively) at polynomial order $P = 9$ for both CG and HDG (right) on a unstructured mesh of a rabbit aorta intercostal pair (left).

to attain optimal performance in this test at lower polynomial orders, since at some point the latency and bandwidth of communication in combination with the small per-process computation cost will in general limit scalability. To this end we consider the scaling of each method on two unstructured curvilinear tetrahedral meshes which are used in the study of biological flows. The first mesh, which we examine at a high polynomial order, is a small mesh of an intercostal pair of a rabbit aorta comprising of approximately 8,500 tetrahedra. At a lower order we instead examine the full mesh of the aortic arch which instead contains approximately 1.5×10^5 elements.

The strong scaling of each method for the intercostal pair is shown in Figure 4. The speedup quoted on the vertical axis is relative to the time taken on a single 32-processor node. At high polynomial order we clearly see that both methods scale well. Even at 4,096 processors, where each process contains only two or three tetrahedra, efficiency remains high. At times the scaling is seen to be super-linear; whilst counterintuitive, this is commonly seen on modern multi-core hardware [31] due to resource contention. At lower numbers of nodes, the matrix-vector multiplication used in the iterative solve requires intensive memory access, meaning that much of the processor time is spent fetching data from the main memory store, where the limiting factor is memory bandwidth. As the problem size decreases however, matrices are able to be stored on processor cache which is far quicker to access, meaning that per-processor execution time is greatly reduced and super-linear scaling can be observed.

To validate this effect is not specific to HECToR, we have performed a replica experiment on ARCHER, the successor to HECToR. ARCHER is a Cray XC30 system, with each node possessing two 2.7 GHz 12-core E5-2697v2 processors, with an improved interconnect.

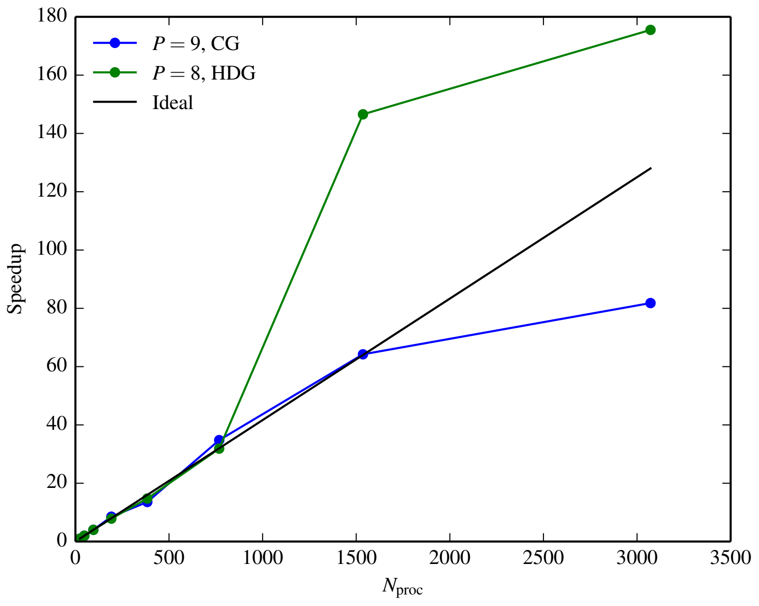

Figure 5: Strong scaling of time-dependent diffusion problem between 24 and 4,096 processors at polynomial order $P = 9$ for both CG and HDG (right) on a unstructured mesh of a rabbit aorta intercostal pair (left).

Figure 5 shows that a similar and indeed more exaggerated behaviour occurs when scaling up to 128 nodes (3,072 cores).

The main observation that can be drawn from Figures 4 and 5 however is that the HDG method generally outperforms the CG method in terms of strong scalability. We note that for the intercostal mesh, the average vertex valency is 12 and maximum valency 44. When the number of elements per core is low, CG communication patterns are therefore significantly complex. The results here go some way to demonstrating that the simplified communication patterns of the HDG method can lead to a more efficient implementation when considering per-iteration timings.

To demonstrate the scalability of each method at lower polynomial orders we consider the same series simulation but now performed for the full rabbit aortic arch. In this case, communication patterns will be significantly more complex due to the larger mesh size. We therefore consider only results obtained from ARCHER as this has the a faster interconnect. In Figure 6 we see that a similar picture of scalability occurs for this mesh as well, although due to the higher communication costs, the super-linear effect is far less pronounced.

5.5.3 Summary of parallel results

Overall we may conclude that in parallel, the HDG and CG methods observe similar or slightly higher execution times per iteration, but only if HDG postprocessing is utilised. However, we note that in general the number of iterations that is taken to converge to the solution in the PCG method is higher for HDG than CG. However, with a better preconditioning strategy, we may expect that iteration counts for both methods can be reduced. We note

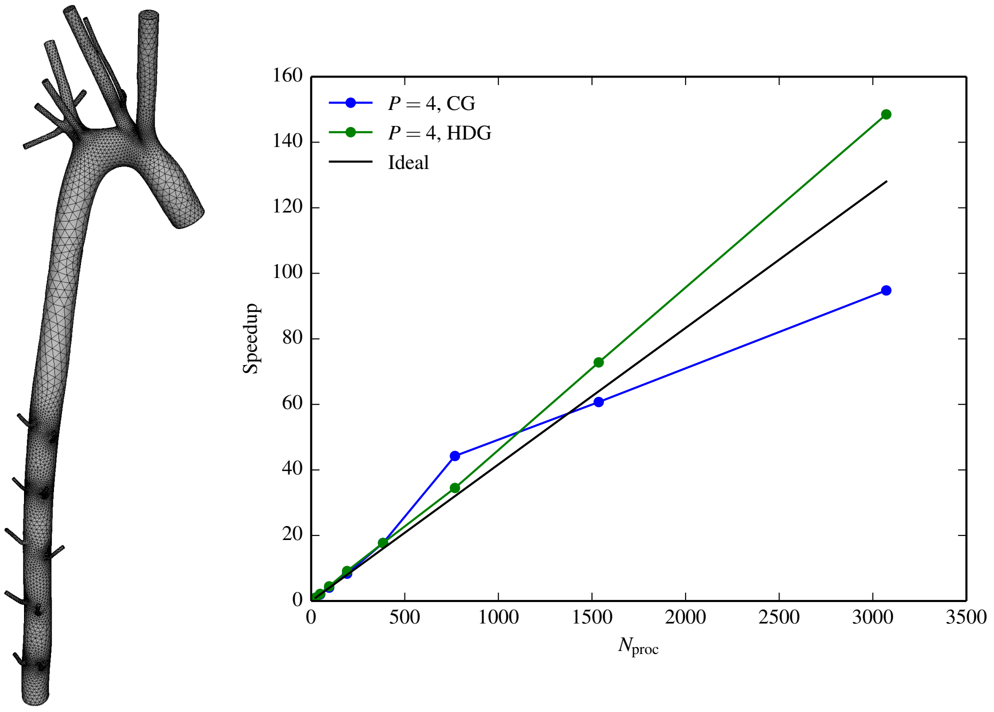

Figure 6: Strong scaling of time-dependent diffusion problem between 1 and 128 processors at polynomial order $P = 4$ for both CG and HDG (right) on a unstructured mesh of a rabbit aortic arch (left).

that in terms of strong scaling, HDG has some benefits over CG where the communication cost is high – either from a low number of elements per process at high order, or a large number of unstructured tetrahedra per process at low order. Furthermore we note that many common CG preconditioning strategies rely on the solution of the coarse linear space at each iteration of the PCG algorithm, particularly when solving Poisson-type problems occuring in for example CFD applications, which further hinder strong scaling of the method [22]. If HDG preconditioning strategies can be developed which avoid this then the difference in scalability for real-world flow problems may be far more pronounced.

6 Conclusions

In this paper, we have presented a comprehensive overview of the performance properties of both statically-condensed CG and HDG methods both in serial and in parallel with different solver strategies, and across a range of test problems which have been deliberately chosen to indicate performance in a variety of theoretically interesting and practically relevant problems. The field of potential categories and problem types one may use for numerical method comparison is vast and by no means does this work claim to cover them all. However, where the problem of interest is time-dependent and does not involve the reconstruction of matrix systems, or elliptic and within scope of direct solvers, this work gives a good indication as to how each method will perform.

The results demonstrate that in serial execution where direct solvers are used, HDG either outperforms or exhibits extremely similar execution times as the CG method. For steady-state elliptic problems, HDG outperforms CG at anywhere between polynomial orders one and three, with the time taken to generate matrices, solve the elliptic problem and produce the solution field ranging between 25% and 90% of the equivalent CG runtime, depending on the choice of solver and element type. For the unsteady diffusion problem, whilst this difference is less pronounced if multi-level static condensation is used, HDG is still able to provide an equivalent performance level to the CG method. We note that in these problems, the local postprocessing method which allows one to obtain a superconverged solution field is crucial in allowing the HDG method to perform far better than the CG method.

On the other hand, when an iterative solver is used to invert the matrix system, the HDG method can struggle to attain the same performance level as CG. In this case the use of postprocessing is essential for the HDG method to exhibit equivalent performance characteristics. This arises from the difference in size between local boundary matrices for the methods, so that the action of the matrix operator becomes more expensive in the case of the HDG method. Indeed, even when postprocessing is used, in the HDG formulation some elements such as the tetrahedron still have larger boundary matrices and thus increased execution times. When the iterative solver is applied in parallel we therefore see similar execution times to the CG method. In parallel, where an iterative solver is the preferred choice, this therefore has the consequence of reduced performance and HDG methods have little advantage over their CG counterparts. One exception to this however is in problems where very few elements are stored per process, where communication costs are higher and the simplified communication patterns present in the HDG formulation can offer better scalability in comparison to CG.

One should keep in mind that the conclusions regarding the comparative performance of two methods when iterative solver is used are based on the per-iteration time data. If one considers the time it takes to solve the linear system using an iterative solver, the CG method is clearly ahead of the HDG method in terms of the number of iterations it requires to converge to a solution. This clearly points to the fact that the development of good preconditioning strategies for the HDG method is one of the stepping stones on the way to its competitive parallel performance.

Acknowledgments: The work was supported by the Department of Energy (DOE NETL DE-EE004449) and under NSF OCI-1148291. DM acknowledges support from the EU FP7 project IDIHOM under grant No. 265780. SJS additionally acknowledges Royal Academy of Engineering support under their research chair scheme.

References

[1] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A Fully Asynchronous Multifrontal Solver Using Distributed Dynamic Scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[2] D.N. Arnold, F. Brezzi, B. Cockburn, and D. Marini. Unified analysis of discontinuous Galerkin methods for elliptic problems. *SIAM J. Numer. Anal.*, 39:1749–1779, 2002.

[3] C.D. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. de Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot, H. Xu, Y. Mohamied, C. Eskilsson, B. Nelson, P. Vos, C. Biotto, R. M. Kirby, and S. J. Sherwin. Nektar++: An open-source spectral/hp element framework. *Computer Physics Communications*, 192:205–219, 2015.

[4] Fatih Celiker, Bernardo Cockburn, and Ke Shi. Hybridizable discontinuous Galerkin methods for Timoshenko beams. *J. Sci. Comput.*, 44(1):1–37, 2010.

[5] Fatih Celiker, Bernardo Cockburn, and Ke Shi. A projection-based error analysis of HDG methods for Timoshenko beams. *Math. Comput.*, 81(277), 2012.

[6] Aycil Cesmelioglu, Bernardo Cockburn, Ngoc Cuong Nguyen, and Jaume Peraire. Analysis of HDG methods for Oseen equations. *J. Sci. Comput.*, 55(2):392–431, 2013.

[7] Yanlai Chen and Bernardo Cockburn. Analysis of variable-degree HDG methods for convection-diffusion equations. Part I: general nonconforming meshes. *IMA Journal of Numerical Analysis*, 32(4):1267 – 1293, 2012.

[8] Cédric Chevalier and François Pellegrini. PT-Scotch: A tool for efficient parallel graph ordering. *Parallel Computing*, 34(6):318–331, 2008.

[9] B. Cockburn, B. Dong, and J. Guzmán. A superconvergent LDG-Hybridizable Galerkin method for second-order elliptic problems. *Math. Comp.*, 77(264):1887–1916, 2007.

[10] B. Cockburn, J. Guzmán, and H. Wang. Superconvergent discontinuous Galerkin methods for second-order elliptic problems. *Math. Comp.*, 78:1–24, 2009.

[11] B. Cockburn and C.-W. Shu. Runge-Kutta Discontinuous Galerkin Methods for convection-dominated problems. *J. Sci. Comput.*, 16:173–261, 2001.

[12] Bernardo Cockburn and Jintao Cui. Divergence-free HDG methods for the vorticity-velocity formulation of the stokes problem. *J. Sci. Comput.*, 52(1):256–270, 2012.

[13] Bernardo Cockburn, Bo Dong, Johnny Guzmán, Marco Restelli, and Riccardo Sacco. A hybridizable discontinuous Galerkin method for steady-state convection-diffusion-reaction problems. *SIAM J. Scientific Computing*, 31(5):3827–3846, 2009.

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

[15] Bernardo Cockburn, Jayadeep Gopalakrishnan, and Raytcho Lazarov. Unified hybridization of discontinuous Galerkin, mixed, and continuous Galerkin methods for second order elliptic problems. *SIAM J. Numer. Anal.*, 47(2):1319–1365, February 2009.

[16] Bernardo Cockburn, Jayadeep Gopalakrishnan, and Francisco-Javier Sayas. A projection-based error analysis of HDG methods. *Math. Comput.*, 79(271):1351–1367, 2010.

[17] Bernardo Cockburn, Ngoc Cuong Nguyen, and Jaume Peraire. A comparison of HDG methods for Stokes flow. *J. Sci. Comput.*, 45(1-3):215–237, 2010.

[18] Bernardo Cockburn, Weifeng Qiu, and Ke Shi. Conditions for superconvergence of HDG methods for second-order elliptic problems. *Math. Comput.*, 81(279), 2012.

[19] Bernardo Cockburn and Ke Shi. Conditions for superconvergence of HDG methods for Stokes flow. *Math. Comput.*, 82(282), 2013.

[20] James W Demmel, Michael T Heath, and Henk A Van Der Vorst. Parallel numerical linear algebra. *Acta Numerica*, 2:111–197, 1993.

[21] M. Dubiner. Spectral methods on triangles and other domains. *J. Sci. Comp.*, 6:345–390, 1991.

[22] Leopold Grinberg, D Pekurovsky, SJ Sherwin, and George E Karniadakis. Parallel performance of the coarse space linear vertex solver and low energy basis preconditioner for spectral/hp elements. *Parallel Computing*, 35(5):284–304, 2009.

[23] E. Hairer, S. P. Norsett, and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems.* Springer, 2nd edition, 1991.

[24] Antonio Huerta, Aleksandar Angeloski, Xevi Roca, and Jaime Peraire. Efficiency of high-order elements for continuous and discontinuous galerkin methods. *International Journal for Numerical Methods in Engineering*, 96(9):529–560, 2013.

[25] T. J. R. Hughes. *The finite element method: linear static and dynamic finite element analysis.* Prentice-Hall, 1987.

[26] Alexander Jaust, Jochen Schuetz, and Michael Woopen. A Hybridyzed Discontinuous Galerkin Method for Unsteady Flows with Shock-Capturing. In *44th AIAA Fluid Dynamics Conference*, AIAA Aviation. American Institute of Aeronautics and Astronautics, 2014.

[27] G. E. Karniadakis, M. Israeli, and S. A. Orszag. High-order splitting methods for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 97(2):414–443, 1991.

[28] G.E. Karniadakis and S.J. Sherwin. *Spectral/hp element methods for CFD - 2^{nd} Edition.* Oxford University Press, UK, 2005.

[29] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1999.

[30] Robert M. Kirby, Spencer J. Sherwin, and Bernardo Cockburn. To CG or to HDG: A Comparative Study. *J. Sci. Comput.*, 51(1):183 – 212, Apr 2012.

[31] Michael Lange, Gerard Gorman, Michele Weiland, Lawrence Mitchell, and James Southern. Achieving efficient strong scaling with PETSc using hybrid MPI/OpenMP optimisation. In *Supercomputing*, pages 97–108. Springer, 2013.

[32] Stephane Lanteri and Ronan Perrussel. An implicit hybridized discontinuous Galerkin method for time-domain Maxwell's equations. Rapport de recherche RR-7578, INRIA, March 2011.

[33] L. Li, S. Lanteri, and R. Perrussel. A hybridizable discontinuous Galerkin method for solving 3D time-harmonic Maxwells equations. In Andrea Cangiani, Ruslan L. Davidchack, Emmanuil Georgoulis, Alexander N. Gorban, Jeremy Levesley, and Michael V. Tretyakov, editors, *Numerical Mathematics and Advanced Applications 2011*, pages 119–128. Springer Berlin Heidelberg, 2013.

[34] N.C. Nguyen, J. Peraire, and B. Cockburn. An implicit high-order hybridizable discontinuous Galerkin method for linear convectiondiffusion equations. *Journal of Computational Physics*, 228(9):3232 – 3254, 2009.

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

[36] Ngoc Cuong Nguyen, Jaume Peraire, and Bernardo Cockburn. Hybridizable discontinuous Galerkin methods for the time-harmonic Maxwell's equations. *J. Comput. Physics*, 230(19):7151–7175, 2011.

[37] Ngoc Cuong Nguyen, Jaume Peraire, and Bernardo Cockburn. An implicit high-order hybridizable discontinuous Galerkin method for the incompressible Navier-Stokes equations. *J. Comput. Physics*, 230(4):1147–1170, 2011.

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

[39] Sander Rhebergen and Bernardo Cockburn. A space-time hybridizable discontinuous Galerkin method for incompressible flows on deforming domains. *J. Comput. Physics*, 231(11):4185–4204, 2012.

[40] Sander Rhebergen, Bernardo Cockburn, and Jaap J. W. van der Vegt. A space-time discontinuous Galerkin method for the incompressible Navier-Stokes equations. *J. Comput. Physics*, 233:339–358, 2013.

[41] Xevi Roca, Ngoc C Nguyen, and Jaime Peraire. Scalable parallelization of the hybridized discontinuous galerkin method for compressible flow. In *21st AIAA Computational Fluid Dynamics Conference*, 2013.

[42] S.J. Sherwin. Hierarchical hp finite elements in hybrid domains. *Finite Elment Analysis and Design*, 27:109, 1997.

[43] S.J. Sherwin and G.E. Karniadakis. A new triangular and tetrahedral basis for high-order (hp) finite element methods. *Int. J. Numer. Meth. Eng.*, 38(22):3775–3802, 1995.

[44] Spencer J Sherwin and Mario Casarin. Low-Energy Basis Preconditioning for Elliptic Substructured Solvers Based on Unstructured Spectral/hp Element Discretization. *Journal of Computational Physics*, 171(1):394–417, 2001.

[45] S.-C. Soon, B. Cockburn, and Henryk K. Stolarski. A hybridizable discontinuous Galerkin method for linear elasticity. *International Journal for Numerical Methods in Engineering*, 80(8):1058–1092, 2009.

[46] Henry M Tufo and Paul F Fischer. Fast parallel direct solvers for coarse grid problems. *Journal of Parallel and Distributed Computing*, 61(2):151–177, 2001.

[47] Peter Vos. *From h to p efficiently: optimising the implementation of spectral/hp element methods*. PhD thesis, Imperial College London, 2011.

[48] Peter E. J. Vos, Spencer J. Sherwin, and Robert M. Kirby. From h to p efficiently: Implementing finite and spectral/hp element methods to achieve optimal performance for low- and high-order discretisations. *J. Comput. Physics*, 229(13):5161–5181, 2010.

[49] Michael Woopen, Thomas Ludescher, and Georg May. A Hybridyzed Discontinuous Galerkin Method for Turbulent Compressible Flow. In *44th AIAA Fluid Dynamics Conference*, AIAA Aviation. American Institute of Aeronautics and Astronautics, 2014.

[50] O.C. Zienkiewicz and R.L. Taylor. *The Finite Element Method: Fluid Mechanics*, volume 3. Butterworth-Heinemann, Oxford, 5th edition, 2000.

[51] O.C. Zienkiewicz and R.L. Taylor. *The Finite Element Method: Solid Mechanics*, volume 2. Butterworth-Heinemann, Oxford, 5th edition, 2000.

[52] O.C. Zienkiewicz and R.L. Taylor. *The Finite Element Method: The Basis*, volume 1. Butterworth-Heinemann, Oxford, 5th edition, 2000.