

MANYCORE PARALLEL COMPUTING FOR A HYBRIDIZABLE  
DISCONTINUOUS GALERKIN NESTED MULTIGRID METHOD\*MAURICE S. FABIEN<sup>†</sup>, MATTHEW G. KNEPLEY<sup>‡</sup>, RICHARD T. MILLS<sup>§</sup>, AND  
BÉATRICE M. RIVIÈRE<sup>†</sup>

**Abstract.** We present a parallel computing strategy for a hybridizable discontinuous Galerkin (HDG) nested geometric multigrid (GMG) solver. Parallel GMG solvers require a combination of coarse-grain and fine-grain parallelism to improve time-to-solution performance. In this work we focus on fine-grain parallelism. We use Intel’s second generation Xeon Phi (Knights Landing) many-core processor. The GMG method achieves ideal convergence rates of 0.2 or less, for high polynomial orders. A matrix free (assembly free) technique is exploited to save considerable memory usage and increase arithmetic intensity. HDG enables static condensation, and due to the discontinuous nature of the discretization, we developed a matrix vector multiply routine that does not require any costly synchronizations or barriers. Our algorithm is able to attain 80% of peak bandwidth performance for higher order polynomials. This is possible due to the data locality inherent in the HDG method. Very high performance is realized for high order schemes, due to good arithmetic intensity, which declines as the order is reduced.

**Key words.** multigrid, discontinuous Galerkin methods, high performance, accelerators

**AMS subject classifications.** 65M55, 65N30, 35J15

**DOI.** 10.1137/17M1128903

**1. Introduction.** Multigrid methods are among the most efficient solvers for linear systems that arise from the discretization of partial differential equations. The effectiveness of this multilevel technique was first observed by Fedorenko [21] in 1964, and popularized by Brandt [11] in 1977. Traditionally, multigrid methods have been applied to low order finite difference, finite volume, and continuous finite element approximations discretizations [53], [56], [24], [12]. The discontinuous Galerkin (DG) discretization brings with it many advantages: it can handle complex geometries, has access to *hp*-adaptivity, is capable of satisfying local mass balance (ideal for flow problems and hyperbolic PDEs), and is highly parallelizable due to the lack of continuity constraints between elements. However, DG methods often have more degrees of freedom than their continuous counterpart. In addition, high order discretizations rapidly increase the condition number of the discretization operator, which poses a challenge for any linear solver [7].

Originally multigrid methods were developed at a time when access to parallelism was limited, and iterative methods that had less concurrency but better convergence

---

\*Submitted to the journal’s Software and High-Performance Computing section May 4, 2017; accepted for publication (in revised form) January 3, 2019; published electronically March 12, 2019.

<http://www.siam.org/journals/sisc/41-2/M112890.html>

**Funding:** This work was supported by the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant ACI-1053575. The work of the first author was supported by the Ken Kennedy Institute and the Ken Kennedy-Cray graduate Fellowship, the Richard Tapia Center for Excellence & Equity, and the Rice Graduate Education for Minorities program.

<sup>†</sup>Department of Computational and Applied Mathematics, Rice University, Houston, TX 77005 ([fabien@rice.edu](mailto:fabien@rice.edu), [riviere@caam.rice.edu](mailto:riviere@caam.rice.edu)).

<sup>‡</sup>Department of Computer Science and Engineering, University of Buffalo, Buffalo, NY 14260 ([knepley@buffalo.edu](mailto:knepley@buffalo.edu)).

<sup>§</sup>Computer Science and Mathematics Division, Argonne National Laboratory, Portland, OR 97210 ([rtmills@anl.gov](mailto:rtmills@anl.gov)).

rates were favored. The multilevel nature of multigrid can cause challenges in its parallelization; fine grids have ample data to work with, which can lead to load balancing problems, something not encountered at coarser levels. Moreover, traditional multigrid is a multiplicative method, that is, each level must be completely processed before moving to the next. Additive multigrid methods allow for the simultaneous processing of all levels, but the trade-off between concurrency and robustness is often not ideal [8].

Multiplicative multigrid is well known to have sequential complexity; for  $N$  data points a single cycle costs  $\mathcal{O}(N)$  floating point operations. However, the parallel complexities of V- and F-cycles are polylogarithmic. For multiplicative multigrid, a natural heterogeneous computing strategy is to process fine levels up to a threshold on a coprocessor (or accelerator) and have the remaining coarse levels be processed by coarse-grain parallelism. In this paper all computations are done on a single Xeon Phi coprocessor (Knights Landing), as we only focus on the fine-grain parallelism. However, an offloading model is a natural extension of this work.

In this paper we consider a nested multigrid technique for high order hybridizable discontinuous Galerkin (HDG) methods which combines  $p$ -multigrid and  $h$ -multigrid. The HDG method is capable of static condensation, which significantly reduces the total number of degrees of freedom compared to classical DG methods. This has important consequences for linear solvers like multigrid, since the cost of multigrid grows with the number of degrees of freedom. Moreover, due to the discontinuous nature of HDG methods, we show that we can efficiently leverage the massive parallelism that manycore processors offer. Through roofline performance modeling and tuning, we show that our implementation results in efficient device utilization as well as significant speedups over a serial implementation. An interesting benefit of the Xeon Phi coprocessor is that it allows us to use traditional parallel programming paradigms like OpenMP, MPI, and pthreads. Consequently, we can take advantage of the massive fine-grain parallelism that the Xeon Phi offers by utilizing these traditional parallel paradigms with significantly limited software intrusion.

The remainder of the paper is organized as follows. Section 2 introduces the model PDE. The relevant finite element notation and HDG discretization are described in section 3. In section 4, we explain how we implement the nodal tensor product basis, quadrature, and barycentric interpolation. Numerical experiments verifying the correct convergences rates for the HDG discretization are conducted in section 5. The core components of the multigrid solver we use are discussed in section 6. This includes the description of intergrid transfer operators, relaxation, coarse grid operators, and the standard multigrid cycle. In section 7 we show through computational experiments that our multigrid method obtains very good convergence rates and error reduction properties. The performance of our algorithm in a parallel setting is evaluated in section 8.

## 2. Model problem.

Consider the elliptic problem

$$(1) \quad -\nabla \cdot (\mathbf{K} \nabla u) = f \quad \text{in } \Omega,$$

$$(2) \quad u = g_D \quad \text{on } \partial\Omega,$$

where  $\Omega$  is an open domain in  $\mathbb{R}^2$  and  $\partial\Omega$  denotes the boundary of the domain. Dirichlet datum  $g_D$  is imposed on the boundary. The function  $f$  is the prescribed source function, and the matrix  $\mathbf{K}$  is symmetric positive definite with piecewise constant entries. The div-grad operator in (1) appears in several problems in engineering such as multiphase flow in porous media. It will provide insight into how a DG multigrid

solver performs on modern architectures. Moreover, it acts as a gateway to construct very efficient numerical methods for time-dependent PDEs that require pressure solves or implicit time stepping.

**3. Discretization.** HDG methods were designed to address the concern that DG schemes generate more degrees of freedom (DOFs) when compared to continuous Galerkin techniques. For standard DG methods, each DOF is coupled with all other DOFs on a neighboring element. By introducing additional unknowns along element interfaces, the HDG method is able to eliminate all DOFs that do not reside on the interfaces. As such, a significantly smaller linear system is generated, and HDG gains much of its efficiency at higher orders [37]. It turns out that the HDG method also has a number of attractive properties, namely, the capability of efficient implementations, optimal convergence rates in the potential and flux variables, as well as the availability of a postprocessing technique that results in the superconvergence of the potential variable. HDG methods are a subset of DG methods, so they still retain favorable properties, e.g., local mass balance, ease of  $hp$ -adaptivity, and the discontinuous nature of the solution variables. A thorough analysis of HDG methods can be found in [17], [20], and [18].

A number of works are available on multigrid for DG methods. Interior penalty methods are the most commonly analyzed; for instance, see [14], [13], [22], [3], and [2]. Most of these works are theoretical, and while they are able to prove convergence, the numerical experiments show rates below what is typically expected from GMG in this model setting. For the interior penalty class of DG methods, it was found that specialized smoothers and careful tuning of the stability parameter were required for better convergence results (see [33], [32]). Local Fourier analysis (local mode analysis [12]) was applied to interior penalty DG methods in [26], [28], and [27]. However, convergence rates were in the range of 0.4 to 0.6 for low order discretizations ( $p \leq 2$ ). In addition, local Fourier analysis is applied to a two-level multigrid scheme and is used as a heuristic to estimate GMG performance. Further, it is well known that two-grid optimality does not always imply V-cycle optimality (see [44]).

It should be noted that the HDG class of discretizations is quite large, similar to that of standard DG methods. In [20], a unified framework is developed (similar to the work of Arnold et al. in [6]) to create a taxonomy of HDG methods. For instance, one can obtain HDG methods by utilizing one of the following: Raviart–Thomas DG, Brezzi–Douglas–Marini DG, local DG, or interior penalty DG. In this work we have no need to distinguish between the various HDG methods, because we employ the local discontinuous Galerkin (LDG) family of hybridizable methods [18]. As such, since the LDG method is a mixed technique, one needs to reformulate the underlying equation (1) as a first order system by introducing an auxiliary variable  $\mathbf{q}$ :

$$(3) \quad \mathbf{q} = -\mathbf{K}\nabla u \quad \text{in } \Omega,$$

$$\mathbf{K}^{-1}\mathbf{q} + \nabla u = 0 \quad \text{in } \Omega,$$

$$\nabla \cdot \mathbf{q} = f \quad \text{in } \Omega,$$

$$(4) \quad u = g_D \quad \text{on } \partial\Omega.$$

We now describe the HDG method. Let  $\mathcal{E}_h$  be a subdivision of  $\Omega$ , made of quadrilaterals,  $K$ , of maximum diameter  $h$ . The unit normal vector outward of  $K$  is denoted by  $\mathbf{n}_K$ . The mesh skeleton is denoted by  $\Gamma_h$ , which is the union of all the edges. We further decompose the mesh skeleton as  $\Gamma_h = \Gamma_h^o \cup \Gamma_h^\partial$ , where  $\Gamma_h^\partial$  denotes the set of all edges on the boundary of the domain, and  $\Gamma_h^o$  the set of all interior

edges. The broken Sobolev space is represented by  $H^1(\mathcal{E}_h)$ ; it consists of piecewise  $H^1$  functions on each mesh element. We use the following shorthand notation for the  $L^2$  inner-product on mesh elements and edges:

$$(5) \quad (u, v)_{\mathcal{E}_h} = \sum_{K \in \mathcal{E}_h} \int_K uv \, dx, \quad \langle u, v \rangle_{\Gamma_h} = \sum_{K \in \mathcal{E}_h} \int_{\partial K} uv \, ds \quad \forall u, v \in H^1(\mathcal{E}_h),$$

$$(6) \quad \langle \mathbf{w} \cdot \mathbf{n}, v \rangle_{\Gamma_h} = \sum_{K \in \mathcal{E}_h} \int_{\partial K} \mathbf{w}|_K \cdot \mathbf{n}_K v|_K \, ds \quad \forall (\mathbf{w}, v) \in H^1(\mathcal{E}_h)^2 \times H^1(\mathcal{E}_h).$$

The underlying approximation spaces for the HDG method are as follows:

$$\begin{aligned} W_h &= \{w \in L^2(\Omega) : w|_K \in \mathbb{Q}_p(K) \quad \forall K \in \mathcal{E}_h\}, & \mathbf{V}_h &= W_h \times W_h, \\ M_h &= \{\mu \in L^2(\Gamma_h) : \mu|_e \in \mathbb{Q}_p(e) \quad \forall e \in \Gamma_h\}, & M_h^p &= M_h, \end{aligned}$$

where  $\mathbb{Q}_p(K)$  is the standard finite element space for quadrilaterals. That is,  $\mathbb{Q}_p(K)$  is the tensor product of polynomials of degree  $p$  on each variable. The same definition (6) applies to  $\langle \mathbf{w} \cdot \mathbf{n}, \mu \rangle_{\Gamma_h}$  for functions  $\mathbf{w} \in H^1(\mathcal{E}_h)^2$  and  $\mu \in M_h$ .

The HDG method seeks an approximation  $(\mathbf{q}_h, u_h, \lambda_h) \in \mathbf{V}_h \times W_h \times M_h$  of the exact solution  $(\mathbf{q}|_{\Omega}, u|_{\Omega}, u|_{\Gamma_h \setminus \partial\Omega_D})$  such that

$$(7) \quad (\mathbf{q}_h, \mathbf{v})_{\mathcal{E}_h} - (u_h, \nabla \cdot \mathbf{v})_{\mathcal{E}_h} + \langle \lambda_h, \mathbf{v} \cdot \mathbf{n} \rangle_{\Gamma_h^o} = -\langle P_h g_D, \mathbf{v} \cdot \mathbf{n} \rangle_{\partial\Omega},$$

$$- (\mathbf{q}_h, \nabla w)_{\mathcal{E}_h} + \langle \mathbf{q}_h \cdot \mathbf{n}, w \rangle_{\Gamma_h} + \langle \tau(u_h - \lambda_h), w \rangle_{\Gamma_h^o} + \langle \tau u_h, w \rangle_{\partial\Omega}$$

$$(8) \quad = (f, w)_{\mathcal{E}_h} + \langle \tau P_h g_D, w \rangle_{\partial\Omega},$$

$$(9) \quad \langle \mathbf{q}_h \cdot \mathbf{n}, \mu \rangle_{\Gamma_h} + \langle \tau(u_h - \lambda_h), \mu \rangle_{\Gamma_h^o} + \langle \tau u_h, \mu \rangle_{\partial\Omega} = \langle \tau P_h g_D, \mu \rangle_{\partial\Omega}$$

for all  $(\mathbf{v}, w, \mu) \in \mathbf{V}_h \times W_h \times M_h$ . The factor  $\tau$  is a local stabilization term that is piecewise constant on  $\mathcal{E}_h$ . In the above,  $P_h g_D$  is the  $L^2$ -projection of  $g_D$ , defined by

$$\int_e P_h g_D \mu = \int_e g_D \mu \quad \forall \mu \in \mathbb{Q}_p(e), \quad \forall e \in \partial\Omega.$$

Introducing additional unknowns at first glance does not appear to add much benefit. However, the HDG method allows us to eliminate the unknowns  $\mathbf{q}_h$  and  $u_h$  using (7) and (9) in an element-by-element manner to arrive at a weak formulation in terms of  $\lambda_h$  only. Let  $\mathbf{Q}$ ,  $\mathbf{U}$ , and  $\boldsymbol{\Lambda}$  denote the vectors of DOFs for the numerical solutions  $\mathbf{q}_h$ ,  $u_h$ , and  $\lambda_h$ , respectively. Since the solutions  $\mathbf{q}_h$  and  $u_h$  are discontinuous, we can denote by  $\mathbf{Q}_K$  and  $\mathbf{U}_K$  the part of the vectors  $\mathbf{Q}$ ,  $\mathbf{U}$  corresponding to the DOFs located on  $K$ . We also denote by  $\boldsymbol{\Lambda}_K$  the part of the vector  $\boldsymbol{\Lambda}$  that corresponds to the DOFs located on the boundary of  $K$ . We can express (7) and (8) in matrix form:

$$(10) \quad \mathbf{A}_K \mathbf{Q}_K - \mathbf{B}_K^T \mathbf{U}_K + \mathbf{C}_K \boldsymbol{\Lambda}_K = \mathbf{R}_K \quad \forall K \in \mathcal{E}_h,$$

$$(11) \quad \mathbf{B}_K \mathbf{Q}_K + \mathbf{D}_K \mathbf{U}_K + \mathbf{E}_K \boldsymbol{\Lambda}_K = \mathbf{F}_K \quad \forall K \in \mathcal{E}_h.$$

Further, one can obtain the local DOFs:

$$(12) \quad \begin{bmatrix} \mathbf{Q}_K \\ \mathbf{U}_K \end{bmatrix} = \begin{bmatrix} \mathbf{A}_K & -\mathbf{B}_K^T \\ \mathbf{B}_K & \mathbf{D}_K \end{bmatrix}^{-1} \left( - \begin{bmatrix} \mathbf{C}_K \\ \mathbf{E}_K \end{bmatrix} \boldsymbol{\Lambda}_K + \begin{bmatrix} \mathbf{R}_K \\ \mathbf{F}_K \end{bmatrix} \right).$$

The above inverse is well defined, and elimination of the local DOFs for  $\mathbf{q}$  and  $u$  can be done in parallel, independently of one another [20]. Equation (9) can be written in matrix form. Fix an interior edge  $e \in \Gamma_h$  that is shared by elements  $K_1$  and  $K_2$  and denote by  $\Lambda_{K_1 K_2}$  the set of DOFs for the unknown  $\lambda_h$ , which lies on the edge  $e$ :

$$(13) \quad \mathbf{G}_{K_1 e} \mathbf{Q}_{K_1} + \mathbf{G}_{K_2 e} \mathbf{Q}_{K_2} + \tau \mathbf{H}_{K_1 e} \mathbf{U}_{K_1} + \tau \mathbf{H}_{K_2 e} \mathbf{U}_{K_2} - 2\tau \mathbf{M}_e \Lambda_{K_1 K_2} = \mathbf{0}.$$

If the edge  $e$  lies on the boundary  $\partial\Omega \cap \partial K$ , we have

$$\mathbf{G}_{K e} \mathbf{Q}_K + \tau \mathbf{H}_{K e} \mathbf{U}_K = \mathbf{S}_e.$$

Now we summarize a general HDG assembly and solve procedure:

- (i) Use (12) to assemble the local problems for  $\Lambda_K$ .
- (ii) Assemble the local problems for  $\Lambda_K$  into a global system matrix using (13).
- (iii) Solve the global system matrix for  $\Lambda$ .
- (iv) Using the newly solved for  $\Lambda$  in (13), reconstruct  $\mathbf{U}$  and  $\mathbf{Q}$ .
- (v) Postprocess  $U$  and  $Q$  to obtain superconvergence.

Step (ii) in our framework is technically assembly free; we do not directly assemble and store the global system matrix. Instead, we exploit the unassembled local problems from (12) to generate a matrix vector multiplication routine. The resulting routine utilizes many small but dense matrices, instead of a large sparse matrix. See [54] for a survey of different finite element assembly strategies (matrix free or otherwise). Although the focus of our paper is not assembly, an algorithm was developed in [36] to accelerate the assembly of the HDG global system matrix.

**4. Basis functions.** We select a tensor product basis for the space  $\mathbb{Q}^p(K)$ . To easily facilitate high order approximations, we invoke a nodal basis with nodes that correspond to roots of Jacobi polynomials (Gauss–Legendre–Lobatto (GLL)). Various operators in the HDG discretization and multigrid method may require evaluation of the nodal basis at points that are not nodal. Some authors use a modal–nodal transformation to deal with such evaluations (see [35]). Another technique is to simply use the fast and stable *barycentric interpolation* (see [10]). This allows one to stay in the Lagrange basis and not have to resort to a generalized Vandermonde matrix. That is, barycentric interpolation allows for the stable evaluation of the Lagrange basis at any point in its domain. Given  $N$  grid points, the setup cost is  $\mathcal{O}(N^2)$  to generate the barycentric weights, and an  $\mathcal{O}(N)$  cost for each evaluation. Due to the discontinuous nature of the HDG approximation, evaluations occur elementwise, so the stability of barycentric interpolation is perhaps more important than the cost of reevaluation. Since barycentric interpolation allows for arbitrary evaluation, one can select quadrature points that differ from the nodal interpolation points. Below we introduce the barycentric interpolation formula used in this work.

Let  $v$  be a polynomial of degree  $n$  that interpolates the scattered data  $\{f_0, f_1, \dots, f_n\}$  through the points  $\{x_0, x_1, \dots, x_n\}$ . The second form of the barycentric formula is

$$v(x) = \frac{\sum_{j=0}^n \frac{W_j}{x - x_j} f_j}{\sum_{j=0}^n \frac{W_j}{x - x_j}},$$

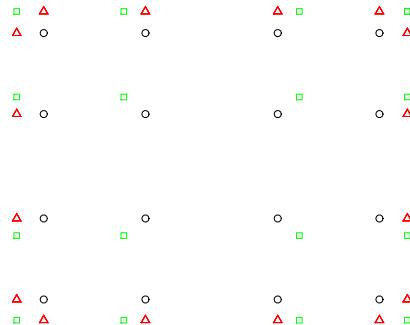


FIG. 1. *GLL nodes* (open squares), *GL nodes* (open circles), and *GL surface nodes* (open triangles).

where  $W_j$  are the barycentric weights. Interpolation points  $x_i$  arising from classical orthogonal polynomials have explicit formulas for their barycentric weights [52], [55]. As an example, GLL and Gauss-Legendre (GL) nodes have the following barycentric weights:

$$W_j^{\text{GLL}} = (-1)^j \sqrt{w_j^{\text{GL}}}, \quad W_j^{\text{GL}} = (-1)^j \sqrt{x_j^{\text{GL}} w_j^{\text{GL}}},$$

where  $w_j$  and  $x_j$  (with appropriate superscripts) are the GL and GLL quadrature nodes and weights. On the reference element, nodal basis functions are defined using a tensor product of a 1D spectral grid. In the case of GLL nodes,  $\{\xi_i^{(p)}\}_{i=0}^p$  are zeros of a particular family of Jacobi polynomials (see [35]), where  $-1 \leq \xi_i \leq 1$ . We adopt the standard that the reference element is  $[-1, 1]$  in 1D,  $[-1, 1] \times [-1, 1]$  in 2D, and  $[-1, 1] \times [-1, 1] \times [-1, 1]$  in 3D. See Figure 1 for a sample spectral grid. In 2D the tensor product can be written as

$$(\xi_i, \eta_j) \stackrel{\text{def}}{=} (\xi_i^{(p)}, \eta_j^{(p)}), \quad i, j = 0, 1, \dots, p,$$

where  $p$  is the polynomial degree associated with  $Q^p$ . On each element, we define the basis functions as tensor products:

$$\phi_I^e(\xi, \eta) = \ell_i(\xi) \ell_j(\eta),$$

with  $I = i + j(P+1)$  (lexicographic ordering, and  $\ell_i$  is the Lagrange polynomial that is nodal at GLL node  $i$ ). In 1D, if there are  $p+1$  GLL nodes, then there are  $p+1$  associated basis functions. In 2D, we take the tensor product of the 1D GLL nodes, which results in  $(p+1)^2$  GLL nodes, and  $(p+1)^2$  associated basis functions. In the index  $I$ , if  $i = j = p$ , then  $I = p + p(p+1) = p^2 + 2p$ , and including the index contribution from  $i = j = 0$ , we have a total of  $p^2 + 2p + 1 = (p+1)^2$  basis functions in 2D as well. For brevity, let the lexicographic ordering  $I = i + j(p+1) =: (i, j)$ . Let the reference element be given by  $\square = [-1, 1] \times [-1, 1]$ . Just as in the 1D case, we assume that

$$u(x, y) \Big|_{\Omega^e} \approx \sum_{i=0}^p \sum_{j=0}^p u_{ij}^e \phi_I^e(\xi, \eta), \quad (\xi, \eta) \in \square,$$

$$\phi_I^e(\xi_m, \eta_n) = \delta_{im} \delta_{jn}, \quad i, j, n, m \in \{0, \dots, p\},$$

$$(\xi, \eta) = \text{GLL quadrature points } \in \square.$$

Using a change of variables to map from the physical element to the reference element, we have

$$\begin{aligned} \iint_{\square} u(\xi, \eta) d\xi d\eta &= \int_{-1}^1 \left( \int_{-1}^1 u(\xi, \eta) d\xi \right) d\eta \\ &\approx \sum_{i=0}^P \sum_{j=0}^P w_i w_j u(\xi_i, \eta_j), \end{aligned}$$

where  $w_i, w_j$  are the GLL quadrature weights in the  $x$  and  $y$  directions. We want to map an arbitrary element to this element, which requires a change of variables  $\xi = \xi(x, y)$ ,  $\eta = \eta(x, y)$ . The Jacobian of this mapping is

$$\mathcal{J}^e(\xi, \eta) = \begin{vmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{vmatrix} = \left| \frac{\partial x}{\partial \xi} \frac{\partial y}{\partial \eta} - \frac{\partial y}{\partial \xi} \frac{\partial x}{\partial \eta} \right|.$$

Hence, integrating over an arbitrary element  $\Omega^e$  ( $\cup_{e=1}^E \Omega^e$ ),

$$\begin{aligned} \iint_{\Omega^e} u(x, y) dx dy &= \iint_{\square} u^e(\xi, \eta) \mathcal{J}^e(\xi, \eta) d\xi d\eta \\ &\approx \sum_{i=0}^P \sum_{j=0}^P w_i w_j u^e(\xi_i, \eta_j) \mathcal{J}_{ij}^e, \end{aligned}$$

where  $\mathcal{J}^e(\xi_i, \eta_j) = \mathcal{J}^e(\xi, \eta)$ . The above approach assumed that the interpolation points and quadrature points were the same (classic spectral element method). However, GLL quadrature rule for  $p+1$  points is only exact for polynomials of degree  $2p-1$ . If higher order quadrature is needed, the GL quadrature rule for  $p+1$  points is exact for polynomials of degree  $2p+1$ . Then one can fix the GLL interpolation points, and evaluate the Lagrange basis functions at GL quadrature points if desired. To do this, we utilize the barycentric formula

$$\ell_j(x) = \frac{\frac{W_j^{\text{GL}}}{x - x_j^{\text{GLL}}}}{\sum_{k=0}^p \frac{W_k^{\text{GL}}}{x - x_k^{\text{GLL}}}}.$$

Barycentric interpolation enables the stable and fast evaluation of the Lagrange polynomial basis  $\ell_j$  anywhere in its domain.

**5. HDG discretization.** Before proceeding with the results of the HDG GMG method, we verify numerically that the HDG discretization provides the expected optimal  $L^2$  convergence rates;  $p+1$  for *both* the potential  $u_h$  and its flux  $\mathbf{q}_h$ . Moreover, with the use of a local postprocessing filter [17], we can achieve superconvergence of the potential  $u_h$ , so that it converges in the  $L^2$  norm with the rate  $p+2$ . HDG does fall under the umbrella of stabilized DG methods, so the parameter  $\tau$  in (8) and (9) needs to be specified. The local stability parameter is piecewise constant, defined facet by facet. For the model problem we set  $\Omega = [0, 1] \times [0, 1]$ ,  $\partial\Omega_D = \partial\Omega$  ( $\partial\Omega_N = \emptyset$ ),  $\mathbf{K}(x, y) = \tanh(x+y) + 1$ . The domain  $\Omega$  is partitioned into  $N \times N$  squares. A manufactured solution is used to examine the error:  $u(x, y) = x(x-1)y(y-1)\exp(-x^2 - y^2)$ , and the corresponding forcing function  $f$  is determined from  $u$ .

For the model Poisson problem it turns out that  $\tau \equiv 1$  on every facet provides optimal convergence rates. In Table 1, it is apparent that the expected convergence rates are met. Moreover, the postprocessed potential  $u_h^*$  results in a rate of  $p + 2$ . The postprocessed flux  $\mathbf{q}_h^*$  converges in a rate of  $p + 1$ , but the errors are smaller than that of  $\mathbf{q}_h$ .

TABLE 1  
*Errors and convergence rates of the HDG scheme on a Cartesian mesh of  $N \times N$  elements.*

$p$	$N$	$\ u_h - u\ _{L^2(\Omega)}$		$\ u_h^* - u\ _{L^2(\Omega)}$		$\ q_h - q\ _{L^2(\Omega)}$		$\ \mathbf{q}_h^* - \mathbf{q}\ _{L^2(\Omega)}$	
		Error	Rate	Error	Rate	Error	Rate	Error	Rate
6	2	1.05e-07	-	3.02e-09	-	2.39e-07	-	1.12e-07	-
	4	8.51e-10	6.95	1.13e-11	8.06	1.95e-09	6.94	8.41e-10	7.06
	8	6.97e-12	6.93	4.45e-14	7.99	1.61e-11	6.92	6.65e-12	6.98
	16	5.60e-14	6.96	7.12e-16	5.97	1.41e-13	6.84	6.36e-14	6.71
	32	8.79e-16	5.99	5.35e-16	4.12	2.11e-13	-5.82	1.47e-13	-1.21
5	2	7.53e-07	-	3.50e-08	-	1.77e-06	-	9.80e-07	-
	4	1.69e-08	5.47	3.03e-10	6.85	3.91e-08	5.50	1.81e-08	5.76
	8	2.90e-10	5.87	2.43e-12	6.96	6.72e-10	5.86	2.95e-10	5.94
	16	4.72e-12	5.94	1.92e-14	6.98	1.10e-11	5.94	4.69e-12	5.97
	32	7.52e-14	5.97	3.68e-16	5.70	2.05e-13	5.74	1.01e-13	5.53
4	2	1.46e-05	-	7.04e-07	-	3.32e-05	-	1.79e-05	-
	4	4.92e-07	4.89	1.09e-08	6.02	1.13e-06	4.88	5.49e-07	5.03
	8	1.65e-08	4.90	1.73e-10	5.97	3.80e-08	4.89	1.76e-08	4.96
	16	5.37e-10	4.94	2.75e-12	5.98	1.24e-09	4.94	5.62e-10	4.97
	32	1.71e-11	4.97	4.33e-14	5.99	3.98e-11	4.97	1.78e-11	4.98
3	2	8.61e-05	-	7.58e-06	-	2.05e-04	-	1.35e-04	-
	4	7.78e-06	3.47	2.63e-07	4.85	1.80e-05	3.52	9.86e-06	3.78
	8	5.49e-07	3.82	8.63e-09	4.93	1.27e-06	3.82	6.59e-07	3.90
	16	3.63e-08	3.92	2.77e-10	4.96	8.43e-08	3.91	4.28e-08	3.95
	32	2.33e-09	3.96	8.80e-12	4.98	5.43e-09	3.96	2.73e-09	3.97

**6. Multigrid algorithm.** Multigrid methods work by eliminating a wide range of frequencies from the estimated error between the exact solution and the multigrid iterate. Any multigrid method has three key components: relaxation, grid transfers, and coarse grid operators (course grid correction). In the following subsections our selection of these components is presented. The geometric multigrid method leverages information about the underlying geometry, discretization, and PDE. This results in an optimal solver (see [53], [12]) that is coupled to the discretization; only  $\mathcal{O}(N)$  floating point operations are required to reduce the error to discretization level accuracy. Moreover, GMG offers a uniform convergence rate. That is, the rate of convergence of GMG is not dependent on the mesh size.

In direct contrast to GMG, algebraic (black box) solvers rely entirely on the discretization matrix. The trade-off with algebraic solvers (e.g., Krylov methods) is that they are decoupled from the discretization, but their convergence is very sensitive to the condition number of the discretization matrix. Refinement in  $h$  or  $p$  increases said condition number, and in turn the algebraic solver requires even more iterations to reach a desired tolerance. In this context, very efficient preconditioners are needed to assist the algebraic solvers. Multigrid methods are also very popular as preconditioners [22].

The finite element method brings with it two ways to potentially increase accuracy:  $h$  refinement and  $p$  refinement. Since GMG is coupled to the discretization,

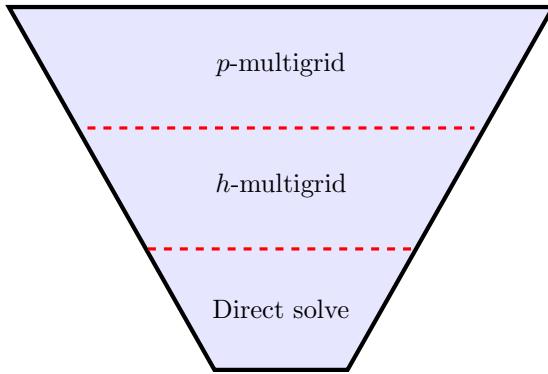


FIG. 2. Diagram of our  $hp$ -multigrid algorithm.

it can leverage  $h$  refinement and/or  $p$  refinement (refinement in  $p$  means increasing/decreasing the polynomial order). This implies that two different classes of multigrid operators are needed: one for  $h$  refinement, the other for  $p$  refinement. For simplicity, we do not consider  $hp$ -GMG (refining  $h$  and  $p$  simultaneously). In  $h$ -GMG the polynomial order  $p$  is fixed, and the hierarchy of grid spaces is determined by increasing/decreasing the mesh size. In  $p$ -GMG the mesh size  $h$  is fixed, and the hierarchy of grid spaces is determined by their increasing/decreasing the polynomial order. The combination of these three refinement strategies offers interesting and potentially efficient ways of dealing with high order and very high order approximations (see [2], [34]).

We use a nested multigrid strategy. The multigrid method starts with  $p$ -multigrid, which fixes the mesh, and the fine and coarse grids are formed by increasing/decreasing the polynomial order. Once the polynomial order reaches  $p = 0$ , which corresponds to a face-centered finite volume scheme, we continue with the  $h$ -multigrid scheme as suggested in [23], [19], [51]. This technique exhibits  $h$  independent convergence. It projects the HDG  $p = 0$  approximation into an appropriate space of continuous functions, so that standard  $h$ -multigrid for continuous Galerkin can be used for the coarser levels. A diagram of our general nested multigrid strategy is given in Figure 2.

We use rediscretization for  $p$ -multigrid to form the hierarchical grids; this way we do not store the sparse linear systems for the polynomial orders  $p > 0$ . The corresponding grid transfer operators are also implemented in a matrix-free manner; as the canonical grid transfer operators are discontinuous across element interfaces, they are completely data parallel.

For  $p = 0$ , and subsequent  $h$ -multigrid levels, we store the linear systems in compressed row storage format. On the coarsest level (matrix of about dimension 2000) of the  $h$ -multigrid scheme, we employ the Math Kernel Library-optimized PARDISO direct solver [48]. Other options are available for the coarse grid solve, for instance, replacing the direct solve with a number of relaxation steps, or continuing the multigrid hierarchy with an algebraic multigrid method.

For problems where geometric multigrid is not suitable, or generating a meaningful geometric hierarchy is not possible, one can replace  $h$ -multigrid with algebraic multigrid or an appropriate coarse grid solver. In these cases,  $p$ -multigrid is still applicable as the mesh is fixed.

**6.1.  $p$ -multigrid grid transfer.** Finite element methods pair well with multilevel methods, due to the flexibility in the choice of underlying spaces. Consider the nested discontinuous finite element spaces  $\Omega^H \subset \Omega^h$ . The mapping from a coarse space ( $\Omega^H$ ) to a fine space ( $\Omega^h$ ) is called prolongation, or interpolation. There are many ways to build a prolongation mapping; however, in the context of finite element methods there is a canonical choice.

The canonical prolongation operator uses the fact that  $\Omega^H \subset \Omega^h$ . That is, any coarse grid function can be expanded as a linear combination of fine grid functions. Thus, we take the prolongation operator to be the natural embedding from  $\Omega^H$  into  $\Omega^h$ . As such,  $P : \Omega^H \rightarrow \Omega^h$ ,

$$(14) \quad P(u_H) = u_H \quad \forall u_H \in \Omega^H.$$

The mapping from a fine space to a coarse space is called restriction, or subsampling. It is not as trivial to select a restriction operator. However, since finite element methods are based on a weak formulation, a canonical restriction is defined by an  $L^2$  projection. The restriction is given by  $R : (\Omega^h)' \rightarrow (\Omega^H)'$ ,

$$(15) \quad (Rv_h, u_H) = (v_h, u_H) \quad \forall v_h \in (\Omega^h)', \quad \forall u_H \in \Omega^H.$$

Notice that  $(Rv_h, u_H) = (v_h, R^*u_H) = (v_h, Pu_H)$ . One can infer from this that  $R^* = P$ . Selecting the restriction operator in this way is deliberate; it is done to ensure that the multigrid operator is symmetric. This allows the multigrid method to be a suitable preconditioner for symmetric iterative methods.

The grid transfer operators are defined on the reference element, and for DG methods the canonical prolongation and restriction operators are data independent. For  $p$ -multigrid we apply the grid transfer operators in a matrix-free manner. These grid transfer operators are defined in (14) and (15), where it is understood that for  $p \geq 0$ , the coarse space is  $\Omega^H = M_h^p$ , and the fine space is  $\Omega^h = M_h^{p+1}$ .

We note that for the HDG method the grid transfer operators are defined on a lower dimension. This is due to static condensation reducing the unknowns (dimension  $d$ ) to the mesh skeleton (dimension  $d - 1$ ). For instance, classical DG methods in 2D would have a prolongation operator  $P : Q^p \rightarrow Q^{p+1}$  of size  $(p + 2)^2 \times (p + 1)^2$ . For HDG in 2D, the prolongation operator would be  $P$  of size  $(p + 2) \times (p + 1)$ , as static condensation reduces the unknowns to 1D. Nodal basis functions give rise to dense grid transfer operators, whereas modal basis functions have sparse binary grid transfer operators.

**6.2.  $h$ -multigrid grid transfer.** The  $h$ -multigrid scheme we use is taken from [23], [19], [51]. The main idea is to project the  $p = 0$  HDG solution into the space of continuous functions, then continue with a standard geometric multigrid for piecewise linear continuous finite elements. It is also possible to project the  $p = 1$  HDG solution to the piecewise linear finite element space. However, in [51] and [19], it is found that this leads to larger iteration counts and poor convergence rates compared to the  $p = 0$  projection. Our numerical experiments also agree with their findings.

Let  $\mathcal{E}_k$  denote a uniform mesh of quadrilateral elements at level  $k$ . The coarsest mesh is denoted by  $k = 0$ . Geometric refinements of the coarsest mesh are then made, for a total of  $J$  meshes. The mesh for the original discretization (finest mesh) is denoted by  $k = J - 1$ . For simplicity we use uniform meshes so that a geometric hierarchy can be used. Set

$$M_k = \{v : \Omega \rightarrow \mathbb{R} : v \text{ is continuous } v|_{\partial\Omega} = 0, v|_K \in Q^1(K) \forall K \in \mathcal{E}_{k+1}\}$$

for  $0 \leq k \leq J - 1$ , and  $M_J = M_h^0$ . Note that the spaces have a nonnested property  $M_0 \subset M_1 \subset \dots \subset M_{J-1} \not\subset M_J$ . As such, a specialized prolongation operator has to be used [23]. It is defined as  $P_k : M_{k-1} \rightarrow M_k$  ( $2 \leq k \leq J$ ),

$$(16) \quad P_k v = \begin{cases} v & \text{if } k < J, \\ \Pi_{M_J}(v|_{\Gamma_h}) & \text{if } k = J, \end{cases}$$

where  $\Pi_{M_J}$  is the  $L^2$ -orthogonal projection onto  $M_J$ . The corresponding restriction operator  $R_{k-1} : M_k \rightarrow M_{k-1}$  follows from (14) ( $1 \leq k \leq J - 1$ ),

$$(17) \quad (R_{k-1} w, u)_{k-1} = (w, P_k u)_k \quad \forall u, w \in M_{k-1}.$$

**6.3. Coarse grid operator.** The multigrid method utilizes a hierarchy of grids. As such, there is a need for a hierarchy of discretizations. In other words, on a coarse grid we have a reduced model of our original discretization. This reduced model needs an appropriate discretization (coarse grid operator) for the coarse grid. Two of the most common coarse grid operators are subspace inheritance and subspace noninheritance.

Subspace inheritance defines coarse grid operators by  $\mathbf{A}^H = \mathbf{R}\mathbf{A}^h\mathbf{P}$ , where  $\mathbf{A}^H$  is the coarse grid operator,  $\mathbf{A}^h$  is the fine grid operator, and  $\mathbf{R}$  and  $\mathbf{P}$  are the restriction and prolongation operators, respectively. On each coarse grid information from the previous level is being inherited. This technique has the disadvantage that continuity may be implicitly enforced for sufficiently coarse grids. Subspace noninheritance simply uses rediscritization; on each coarse grid  $\Omega^H$  the discretization operator is built for the dimension associated with  $\Omega^H$ .

In the  $p$ -multigrid setting, the coarse grid problems are formed by discretizing our problem for smaller polynomial degrees. With  $p$ -multigrid there is a subtle choice that is to be made for its refinement strategy. The coarse spaces for  $p$ -multigrid are obtained by decreasing the polynomial order  $p$ . How one should decrease the polynomial order is not well agreed upon. A couple of coarsening ratios have been proposed in the context of continuous  $hp$ -multigrid (see [43]):

- geometric: assumes  $p$  is a power of two,  $p, p/2, p/4, \dots, 2, 1, 2, 4, \dots$ ;
- odd: assumes  $p$  is an odd natural number;
- even: assumes  $p$  is an even natural number;
- full arithmetic:  $p, p-1, p-2, \dots, 2, 1, 0, 1, 2, \dots$

The most flexible coarsening ratio is full arithmetic, which allows for arbitrary polynomial degrees.

To define the  $p$ -multigrid hierarchy we use full arithmetic coarsening. We selected full arithmetic coarsening since we are using multigrid as a solver, not as a preconditioner. In this case, there are more opportunities to eliminate high frequency modes. For high order DG methods, traditional parallelizable relaxation schemes do not properly eliminate high frequency modes [32, 33, 22, 3, 14, 13]. Full arithmetic coarsening was found to be as effective as more rapid coarsening for problems with moderate polynomial degrees [43, 34].

For  $p > 0$ , we employ a matrix-free procedure. In the  $h$ -multigrid setting, the coarse grid problems are formed by subspace inheritance. For low order problems, there is little benefit for matrix-free operations, so we store the  $p = 0$  matrix and any subsequent matrices that arise from the  $h$ -multigrid algorithm. This amounts to the Galerkin triple product  $\mathbf{A}_{k-1} = \mathbf{R}_{k-1}\mathbf{A}_k\mathbf{P}_k$  (for  $1 \leq k \leq J - 1$ ), where  $\mathbf{R}_{k-1}$  and  $\mathbf{P}_k$  are as defined in (17) and (16), respectively.

**6.4. Relaxation.** Relaxation (also called smoothing) plays an integral part in multigrid techniques. High frequencies are damped using a relaxation method, leaving low frequencies to be resolved by coarse grid correction. In some sense, multigrid can be thought of as a way to accelerate the convergence of a relaxation method; typically the relaxation methods used in multigrid have poor (or no) convergence properties. The main purpose of a relaxation method is to smooth the error, not necessarily make it small. Standard choices for relaxation are pointwise/block stationary iterative methods (Jacobi, damped Jacobi, Gauss–Seidel, successive over relaxation, etc.), and polynomial-type preconditioners.

A general overview of iterative methods can be found in [46]. For lower order discretizations (in the context of this work, third order or less), standard relaxation methods tend to work well. However, for higher order discretizations, loss of uniform convergence is experienced. A larger number of relaxation steps or more powerful smoothers are required to retain uniform convergence in the high order case.

In a parallel setting, polynomial smoothers were shown in [1] to perform better in terms of time to solution. With the advent of manycore devices, some authors have examined parallel asynchronous iterations (see [5], [4]). Sparse approximate inverses (SPAI) have exhibited promising results when used as preconditioners and smoothers in multigrid (see [9], [15]). Moreover, the construction of SPAI operators are inherently parallel and require only the application of a matrix vector product.

Relaxation for high order DG methods typically requires specialized treatment. This was found to be the case for HDG in our experiments, as well as in [51] and [58]. In particular, block-type methods work better than pointwise or polynomial smoothers. In the context of our work, we have found that SPAI and additive Schwarz domain decomposition methods work very well as smoothers. In section 7 we present multigrid convergence results for an SPAI preconditioner. The additive Schwarz smoother has similar convergence properties depending on the overlap size. These more expensive smoothers are used in our work as they enable us to retain robust multigrid convergence. Section 7 discusses this matter further.

**6.5. Sparse approximate inverse relaxation.** Sparse approximate inverses have exhibited promising results when used as preconditioners and smoothers in multigrid (see [9], [15]). Moreover, the construction of SPAI operators are inherently parallel and require only the application of a matrix vector product. SPAI seeks an approximation  $\mathbf{M}$  of  $\mathbf{A}^{-1}$  such that

$$\min_{\mathbf{M} \in \mathcal{S}} \|\mathbf{I} - \mathbf{M}\mathbf{A}\|_F,$$

where  $\mathcal{S}$  is a set of sparse matrices. By using the Frobenius norm, a great deal of concurrency is exposed:

$$\|\mathbf{I} - \mathbf{M}\mathbf{A}\|_F^2 = \sum_{i=1}^n \|\vec{\mathbf{e}}_i^T - \mathbf{M}_i^T \mathbf{A}\|_2^2.$$

Each of the  $n$  least squares problems is independent of one another. Storage and computational savings can be obtained by leveraging the sparsity of  $\mathbf{A}$  and  $\mathbf{M}$ . In addition, since the HDG method gives rise to a symmetric operator, the factorized sparse approximate inverse (FSAI) can be utilized to obtain further improvements. The use of SPAI as a smoother will require its explicit storage, but the application is that of a matrix vector multiply:

$$\vec{x}_{\text{SPAI}}^{(k+1)} = \vec{x}_{\text{SPAI}}^{(k)} + \mathbf{M}(\vec{f} - \mathbf{A}\vec{x}_{\text{SPAI}}^{(k)}).$$

**6.6. Additive Schwarz domain decomposition relaxation.** Increasing the polynomial order in finite element methods causes the condition number of the discretization operator to grow. This stretches the spectrum of the discretization operator, and traditional smoothers can no longer damp a sufficiently wide range of high frequencies. To combat this, one can increase the number of smoothing steps, or seek a more powerful smoother. In some cases, domain decomposition methods can provide a very powerful smoother for multigrid techniques. In [41], [49], and [50], variations of the Schwarz domain decomposition method are used as multigrid smoothers for high order finite element discretizations.

The standard additive Schwarz method takes the form

$$\mathbf{M} = \sum_{e=1}^{|\Gamma_h|} \mathbf{R}_e^T \mathbf{A}_e^{-1} \mathbf{R}_e,$$

where  $\mathbf{R}_e$  is a binary restriction operator that transfers a global datum to a local datum. The operator  $\mathbf{A}_e$  is the discretization operator restricted on the facet  $e$ . Notice that this is a single-level Schwarz method, and as such it is not as effective as their two-level counterparts. Weighting the single-level Schwarz method has been demonstrated to produce effective smoothers (see [41], [50], [40]). The weighting appears to be successful, but is also somewhat experimental. The inverse of a diagonal counting matrix is used in [41], and a type of bump function is used in [50] (both of these weightings are used without a convergence theory). For a weighting matrix  $\mathbf{W}$ , the additive Schwarz smoother is applied as follows:

$$\vec{x}_{\text{ASM}}^{(k+1)} = \vec{x}_{\text{ASM}}^{(k)} + \mathbf{W} \mathbf{M} (\vec{f} - \mathbf{A} \vec{x}_{\text{ASM}}^{(k)}).$$

On each level of the  $p$ -multigrid hierarchy we store smoother  $\mathbf{M}$  as dense blocks and apply them in a unassembled manner. The size of these blocks depends on the polynomial order and overlap size.

**6.7. Standard multigrid V-cycle.** Our nested multigrid V-cycle is presented in Algorithms 1 and 2. This is often referred to as the V-cycle. We use the following notation:  $S^\nu(\cdot)$  denotes  $\nu$  steps of a relaxation method. In the  $p$ -multigrid cycle the smoother  $S^\nu(\cdot)$  is either SPAI or the additive Schwarz relaxation. In section 7 we use SPAI, and in section 8 we use the additive Schwarz relaxation. For  $h$ -multigrid we use the Chebyshev smoother [46] as the problem is low order.

---

**Algorithm 1**  $v^h \leftarrow hMG(v^h, r^h)$ .

---

$\mathbf{A}^h$ ,  $\mathbf{R}$ ,  $\mathbf{P}$  are stored in sparse matrix format.

$\mathbf{R}$  and  $\mathbf{P}$  are as defined in (17) and (16).

The smoother  $S$  is a polynomial relaxation (Chebyshev).

- 1:  $v^h \leftarrow S^{\nu_1}(\mathbf{A}^h, r^h, v^h)$
  - 2: **if** On coarsest level **then**
  - 3:    $v^H \leftarrow (\mathbf{A}^H)^{-1} v^H$  (bottom level solver, e.g., direct solve or AMG)
  - 4: **else**
  - 5:    $r^H \leftarrow \mathbf{R}(r^h - \mathbf{A}^h v^h)$
  - 6:    $v^H \leftarrow 0$
  - 7:    $v^H \leftarrow hMG(v^H, r^H)$
  - 8:    $v^h \leftarrow v^h + \mathbf{P} v^H$
  - 9:  $v^h \leftarrow S^{\nu_2}(\mathbf{A}^h, r^h, v^h)$
-

---

**Algorithm 2**  $v^h \leftarrow pMG(v^h, r^h)$ .

---

$\mathbf{A}^h$ ,  $\mathbf{R}$ ,  $\mathbf{P}$  are applied in a matrix-free manner.

$\mathbf{R}$  and  $\mathbf{P}$  are as defined in (15) and (14).

The smoother  $S$  is either SPAI or the additive Schwarz relaxation.

- 1:  $v^h \leftarrow S^{\nu_1}(\mathbf{A}^h, r^h, v^h)$
  - 2: **if** On coarsest level ( $p = 0$ ) **then**
  - 3:    $v^H \leftarrow hMG(v^H, r^H)$  (bottom level solver, is  $h$ -multigrid)
  - 4: **else**
  - 5:    $r^H \leftarrow \mathbf{R}(r^h - \mathbf{A}^h v^h)$
  - 6:    $v^H \leftarrow 0$
  - 7:    $v^H \leftarrow pMG(v^H, r^H)$
  - 8:  $v^h \leftarrow v^h + \mathbf{P}v^H$
  - 9:  $v^h \leftarrow S^{\nu_2}(\mathbf{A}^h, r^h, v^h)$
- 

**7. Multigrid convergence.** In [2], uniform convergence in  $h$  and  $p$  was established for a DG  $hp$ -multigrid algorithm. However, in order to guarantee  $p$  (or  $h$ ) independent convergence, the number of relaxation steps per level grows quadratically in  $p$ . Numerical experiments suggest that for our HDG  $p$ -multigrid, the relaxation steps per level need to grow linearly (in  $p$ ) to guarantee  $p$  independent convergence. In [19],  $h$  independent convergence was proven for the HDG  $h$ -multigrid algorithm we use.

Figures 3(a) and 3(b) display the results for  $2 \leq p \leq 8$  and fine mesh with  $(2^5)^2$  elements. We employ an FSAI smoother on each level, with  $\nu_1 = \nu_2 = 2$  pre- and postsmothing steps. The FSAI smoother is constructed so that it results in an approximate inverse with an operator complexity of unity ( $\text{nnz}[\mathbf{M}] = \text{nnz}[\mathbf{A}]$ ). Subspace noninheritance is used to generate coarse grid operators. For the  $p$ -GMG phase of the GMG method, we use full arithmetic coarsening. To measure the convergence rate, we keep track of the two norm of the fine grid residual between successive iterations:

$$\rho_k = \frac{\|(r^h)^{(k)}\|_2}{\|(r^h)^{(k-1)}\|_2}.$$

We can see that the results are quite good, even for a modest FSAI smoother. All of the convergence rates are under 0.22 and tend to cluster in the range 0.13–0.15 for even polynomial degrees. This observation perhaps indicates that even- $p$  coarsening or geometric- $p$  will not only be more efficient (more rapid coarsening), but also more accurate.

Figure 3 seems to suggest that our method is not exhibiting a  $p$  independent convergence rate. However, we recall that, according to [2], uniform convergence in  $p$  is guaranteed only if the number of relaxation steps per level grows quadratically in  $p$  (we set  $\nu_1 = \nu_2 = 2$ ). For multigrid methods there is a subtle observation to be made: uniform convergence is an excellent property; but if the uniform convergence rate is near one, this is not an efficient solver. Moreover, if the number of smoothing steps is sufficiently large, a great deal of efficiency is lost. For reasonable smoothing steps, in [14] and [2] the authors demonstrate convergence rates of 0.85 (or worse). In order to drop the convergence rates down to 0.5, upwards of 20 smoothing steps every level need to be taken.

What is needed for efficiency is as follows: uniform convergence, a fast convergence rate (problem dependent), and a small number of uniform smoothing steps. Satisfying

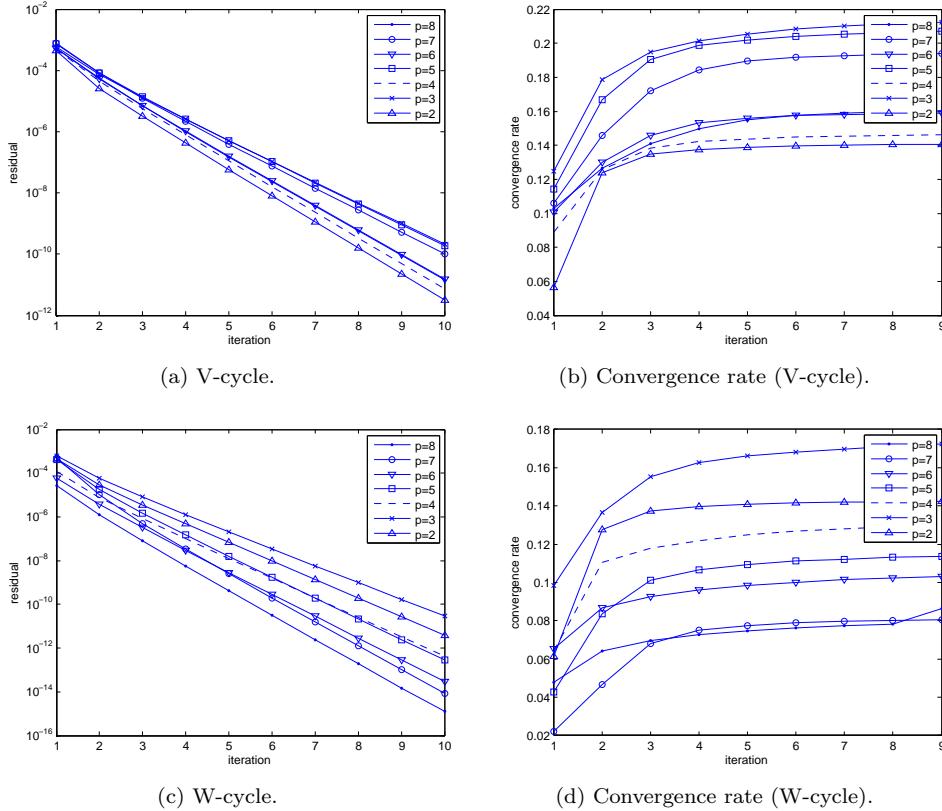
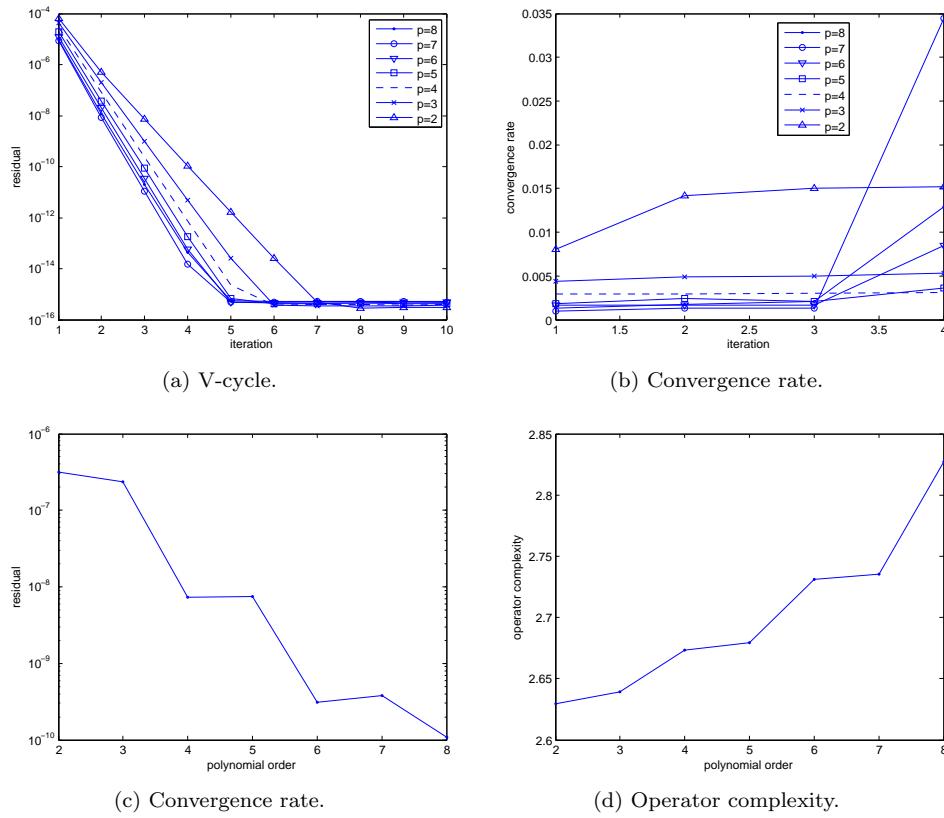


FIG. 3. GMG for HDG (SPAI-1 smoother).

these three properties in practice for high order DG methods is not trivial. Some potential avenues to explore are stronger (but more expensive) smoothers, different coarse grid transfer operators, different multigrid schedules, as well as different grid hierarchies. This direction of inquiry is beyond the scope of our paper. In Figure 4 we do show that a more expensive smoother essentially yields uniform convergence with a fast convergence rate and a fixed number of smoothing steps.

For the next set of experiments, we allow the FSAI operator complexity to grow ( $\text{nnz}[\mathbf{M}] \approx 2.5 \text{ nnz}[\mathbf{A}]$ ). Figures 4(a), 4(b), 4(c), and 4(d) display the results of this change. For the V-cycle, the residual is reduced to machine precision after only 5 to 7 iterations (Figure 4(a)). The aggressive smoothing also yields stellar convergence rates, below 0.015, as can be ascertained from Figure 4(b).

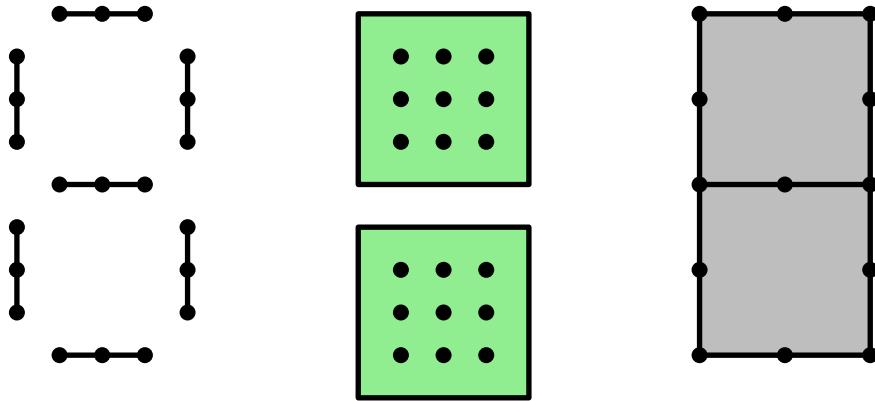
The performance of the V-cycle is better than ideal, so we can easily extend our HDG GMG method to leverage the *full multigrid cycle* (FMG). FMG is well known to be *the* optimal multigrid schedule for linear problems; with a single FMG iteration, the residual is reduced to discretization level error. Moreover, it requires only  $\mathcal{O}(N)$  floating point operations to achieve this accuracy (see [11], [12], [53]). Indeed, Figure 4(c) numerically verifies that a *single* FMG iteration is enough to reach discretization level error. There is something particularly interesting about Figure 4(c)—the FMG iteration performs better for higher orders  $6 \leq p$ . This second experiment of course comes at a price: the FSAI smoothers on each level have an

FIG. 4. *FMG for HDG (aggressive FSAI)*.

operator complexity in the range of 2.65 to 2.85. Such a trade-off may be valuable for problems with highly varying or discontinuous coefficients. Also, the FSAI smoother can be easily tuned to control how aggressively its operator complexity grows.

**8. Performance model.** The storage and assembly of global matrices in finite element methods can be exceedingly prohibitive, especially at higher orders. By leveraging matrix-free algorithms, one can save on memory and convert a memory bound problem (sparse matrix vector multiplication) into a compute bound problem. The authors of [45] found that to improve sparse matrix vector multiplication for HDG methods, specialized storage formats were needed. Since the HDG method can be reduced to a problem on the trace space, this allows for an assembly (matrix-free or otherwise) of the discretization operator in a facet-by-facet manner, instead of an element-by-element manner. The importance of this is that the element-by-element approach requires a synchronization (barrier, atomic, etc.) in order to avoid race conditions. A graph coloring algorithm is typically used in this situation, but only allows for a group of colors to be utilized at any given time. The facet-by-facet approach does not depend on the vertex degree, but only on the element type (how many facets on a given element). Figure 5 displays some example DOFs and connectivity for HDG, DG, and continuous Galerkin methods.

The HDG method is able to exploit facet-by-facet connectivity; given an interior

FIG. 5. *HDG, DG, and continuous Galerkin DOFs.*

facet, only the two elements that share said facet will contribute to its DOFs. An element-to-facet mapping will allow one to gather and scatter the DOFs on a facet. Pseudocode for our matrix vector multiply routine is given in Algorithm 3.

---

**Algorithm 3** Algorithm for HDG matrix vector multiply (assembly free).

---

```

1: procedure MATRIX VECTOR MULTIPLY
2:   Algorithm for HDG matrix vector multiply (assembly free)
3:   Load  $\mathbb{K}_K$  and  $x$ 
4:    $eN \leftarrow \text{Map0(tid)}$  (Given thread id, find the edge it corresponds to)
5:    $(E1, E2) \leftarrow \text{Map1}(eN)$  (Find the elements that share edge)
6:    $(\text{idx1}) \leftarrow \text{Map2}(eN, E1)$  (load local index of DOF on element 1)
7:    $(\text{idx2}) \leftarrow \text{Map3}(eN, E2)$  (load local index of DOF on element 2)
8:    $y_1 = 0$ 
9:   for  $j = 1$  to size( $\mathbb{K}_{E1, eN}$ ) do
10:     $y_1 \leftarrow y_1 + \mathbb{K}_{E1, eN}(\text{idx1}(j), :) \cdot x$ 
11:    $y_2 = 0$ 
12:   for  $j = 1$  to size( $\mathbb{K}_{E2, eN}$ ) do
13:     $y_2 \leftarrow y_2 + \mathbb{K}_{E2, eN}(\text{idx2}(j), :) \cdot x$ 
14:    $y(\text{tid}) = y_1 + y_2$ 

```

---

**8.0.1. Knights Landing (KNL) manycore coprocessor.** In this work we use the second generation Xeon Phi designed by Intel. It is a manycore processor, and the 7000 series has anywhere from 64 to 72 cores, with 4 threads per core. The KNL is an example of a high-throughput low-memory device. The design of the KNL is similar to other manycore devices and accelerators: a large number of cores with lower clock speeds are packed into the unit, enabling a large vector width, as well as having access to a user-manageable fast memory hierarchy. One interesting feature of the KNL is that it can be programmed using traditional parallel paradigms like OpenMP, MPI, and pthreads. In addition, it operates as a native processor. Further details regarding the KNL can be obtained in [31]. For specifications of the KNL used in this work, see Table 2.

TABLE 2  
*KNL testbed specifications.*

Processor number	7210
# of cores	64
Processor base frequency	1.30 GHz
Cache	32 MB L2
RAM	384 GB DDR4
MCDRAM	16 GB
Instruction set	64-bit
Instruction set extension	Intel AVX-512
Operating system	CentOS 7.2
Compiler	Intel Parallel Studio XE 16

**8.0.2. Local matrix generation.** In order to make use of the assembly-free matrix vector multiply, one needs to generate the associated local matrices (in the case of HDG, see (i), (iv), and (v) in section 3). To generate the required local matrices, for simplicity, we use a one core (thread) per element strategy. However, as one increases the polynomial order (beyond  $p = 5$ ), this strategy loses performance. Further improvements may potentially be obtained by utilizing nested parallelism, or linear algebra libraries dedicated for small- or medium-sized matrices (see [25], [36]).

For all computational experiments we set the KNL in quadrant mode. Figures 6 and 7 display the wall-clock and speedup as the polynomial order and number of threads is varied. In Figure 6, one can see that as we increase the number of threads beyond 32 or 64, diminishing returns are very noticeable. To generate the results of Figure 7, we fix the number of threads to 64, and we vary the polynomial order. The comparison with a serial implementation clearly shows that the additional parallelism the KNL offers is beneficial; speedups ranging from 2X to 32X are attained.

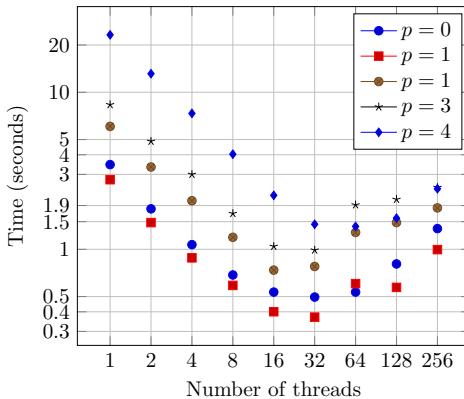


FIG. 6. Local matrix assembly wall-clock.

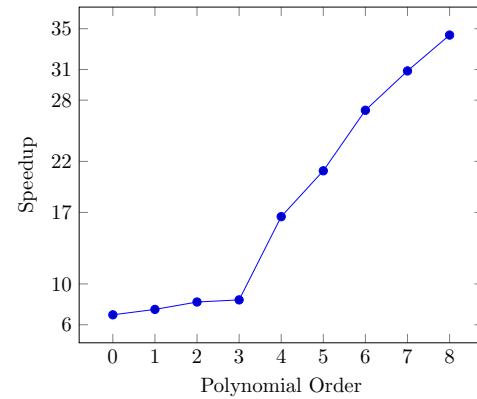
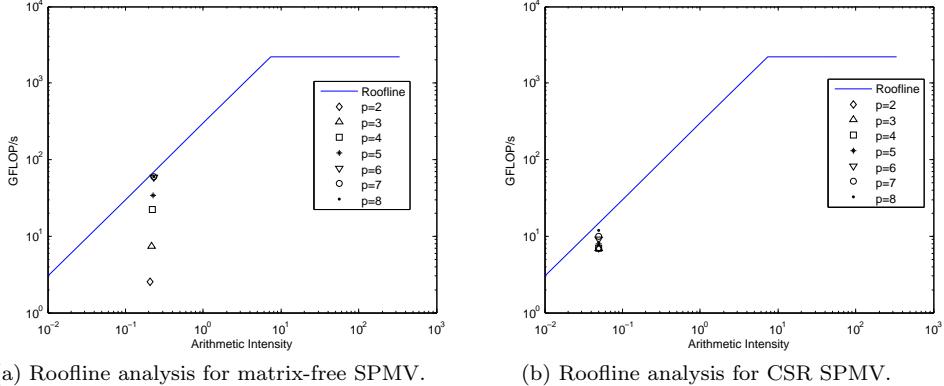


FIG. 7. Local matrix assembly speedup.

**8.1. Roofline analysis.** The KNL (7210 processor number) used in this analysis runs at 2.1 GHz (double precision), a double precision processing power of 2,199 GFLOP/s, and the STREAM memory bandwidth benchmark (triad; see [42]) reports a bandwidth of 300 GB/s. These two metrics provide the performance boundaries for arithmetic throughput and memory bandwidth limits, respectively. For our numerical experiments, the clustering mode is set as quadrant, and cache mode is set to flat.

FIG. 8. *Roofline analysis for matrix-free SPMV and CSR SPMV.*

**8.1.1. Sparse matrix vector multiply (SPMV).** In general, matrix vector multiplication is severely bandwidth bound. For a dense matrix of dimension  $N$ , we can expect  $(2N^2 - N)$  FLOPs and  $8(2N^2 + N)$  MEMOPs (bytes). Thus, for large  $N$ , we can expect an arithmetic intensity of only  $\lim_{N \rightarrow \infty} (2N^2 - N)/(8(2N^2 + N)) = 0.25$ . For finite element problems the underlying discretization matrix is typically sparse, which pushes SPMV further into the bandwidth bound region on the roofline chart. Matrix-free operation can remedy this situation by improving the arithmetic intensity. That is, matrix-free application can shift our bandwidth bound SPMV to a compute bound (or less bandwidth bound) problem.

In Figure 8(a), we include a roofline analysis of our matrix vector multiply routine in the context of its performance on the Laplacian operator. The roofline analysis [57] allows us to identify bottlenecks, confirm algorithm limitations, as well as gain insight into what we should focus on in terms of optimization. Two different techniques are studied. Sparse matrix storage (CSR format; see Figure 8(b)) has the lowest performance arithmetic intensity, with a theoretical limit of 0.25. The multithreaded Intel Math Kernel Library is used for this approach [30]. For high orders the HDG method has hundreds of nonzeros per row. Slightly better but similar performance was obtained in [45] by using a specialized block sparse matrix storage format on GPUs. The matrix-free technique shifts the arithmetic intensity favorably (see Figure 8(a)); this behavior is typical in spectral methods and spectral elements, due to near constant FLOP and memory requirements per DOF [54], [16], [39].

Figure 9 displays the bandwidth measurements. From Figure 9(a), for lower order polynomials ( $p < 5$ ) on a fixed coarse mesh, we see that a maximum of 50% of the peak bandwidth is achieved. The overhead costs of forking threads is dominating computations for  $p < 5$  on coarser meshes. As  $p$  increases beyond five, the cost of forking threads is no longer dominant. Our algorithm reaches 80% of the peak bandwidth reported by the STREAM benchmark.

Figure 9(b) considers fixing the polynomial degree and increasing the number of elements in the mesh. Here it is evident that  $h$ -refinement improves the bandwidth performance for lower polynomial degrees. To observe further improvements, a new SPMV routine could be designed specifically for lower orders. For instance, in [38], the authors group together the contributions from multiple elements into a thread block in their SPMV-GPU kernel. Their technique is shown to be very high performing.

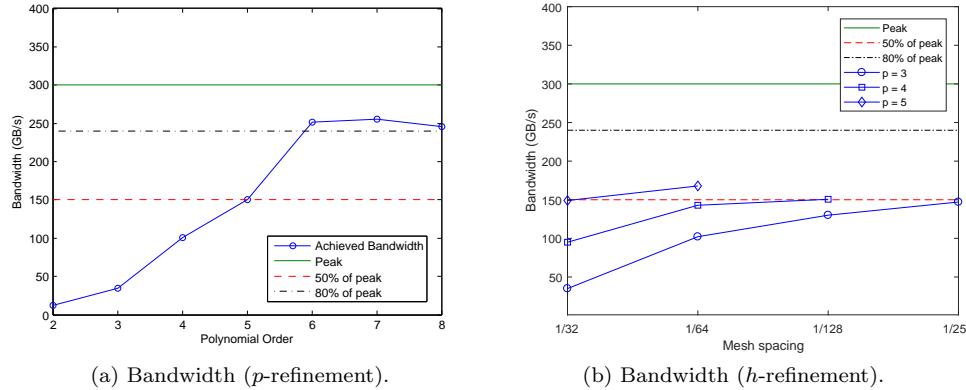


FIG. 9. Achieved bandwidth vs. polynomial order for matrix-free SPMV.

TABLE 3  
Arithmetic intensity and GFLOP/s for the local solvers.

Polynomial order	AI	GFLOP/s
0	0.84	0.0032
1	1.41	1.15
2	2.60	5.129
3	4.33	21.18
4	6.56	53.66

**8.1.2. Projection from volume to surface.** The generation of the local solvers from (12) is completely data parallel and needs no synchronizations or thread communication. It requires a dense linear solve, two matrix vector multiplications, and a single SAXPY (scalar times a vector plus a vector). Roughly the complexity model for this projection operator in terms of FLOPs can be estimated by

$$\begin{aligned} \text{FLOPs} &= 2N^2 - N + (2/3)N^3 + M(2N - 1) + 2M, \\ \text{MEMOPs} &= 8(N^2 + NM + M + N), \end{aligned}$$

where  $N = 4(p+1)^2$  and  $M = 4(k+1)$ . Table 3 collects the results for polynomial order 0, 1, 2, 3, and 4. As we increase the polynomial order, performance increases due to the compute bound nature of the local HDG solvers, as the dense linear solve dominates with  $\mathcal{O}(n^3)$  FLOPs for  $\mathcal{O}(n^2)$  data. We again note that the focus of this work is not on the generation of these local matrices, but we can still obtain reasonable performance utilizing a straightforward implementation.

After solving the trace space system given by (13), if the volume solutions  $u$  and  $q$  are desired, one can invoke (12) to reconstruct them. The cost of this procedure depends on whether one discards the local matrices. In this case, one has to recompute, and the cost is the same as the projection from the surface to volume. If one instead keeps the local matrices, all that is needed is two matrix vector multiplies and a single SAXPY. Ultimately this comes down to a preference of convenience over memory concerns.

**8.1.3. Cost analysis matrix-free vs. matrix-stored.** We consider the cost of our matrix-free approach compared to a matrix-based approach. The previous sections discuss the memory considerations, but there are other measures one can

use. Most importantly, operation counts and time to solution. The operation count (FLOPs) for sparse matrix vector multiplication is  $\mathcal{O}(nnz)$ , where  $nnz$  is the number of nonzero entries in the sparse matrix in question. For the statically condensed HDG method on a structured Cartesian mesh,  $nnz$  for the stiffness matrix is  $C_1 = \mathcal{O}(Nd(p+1)^{d-1} \times (4d-1)(p+1)^{d-1})$  [47], [29] ( $d$  is the dimension, and  $N$  is the total number of elements in the mesh). The operation count for our proposed matrix-free variant (Algorithm 3) is  $C_2 = \mathcal{O}(2|\Gamma_h|(2 \cdot 4(p+1) - 1))$ . In 2D, as  $N \rightarrow \infty$ , the ratio tends to  $C_2/C_1 \rightarrow (16p + 14)/(7(p+1)^2)$ , which indicates that for higher orders, the matrix-free variant is cheaper in terms of operations than the sparse matrix format.

A second important measure is time to solution. Both the matrix-free and matrix-based algorithms require the local solvers. In the matrix-based approach, a nonnegligible amount of time is spent on the matrix assembly, especially for higher orders. We compare the wall-clock matrix-based solve to the wall-clock of the (parallel) assembly. See Figure 10(a). For  $p = 2$ , the assembly time is already more than 40% of the total solve time. Increasing the polynomial order to  $p > 2$ , the assembly time ranges from around 45% to 56% of the total solve time. Comparing the matrix-free solve timings to the matrix-based solve and assembly time, Figure 10(b) shows that for  $p > 2$  we obtain speedups of 1.5 to 2.3 in favor of the matrix-free method.

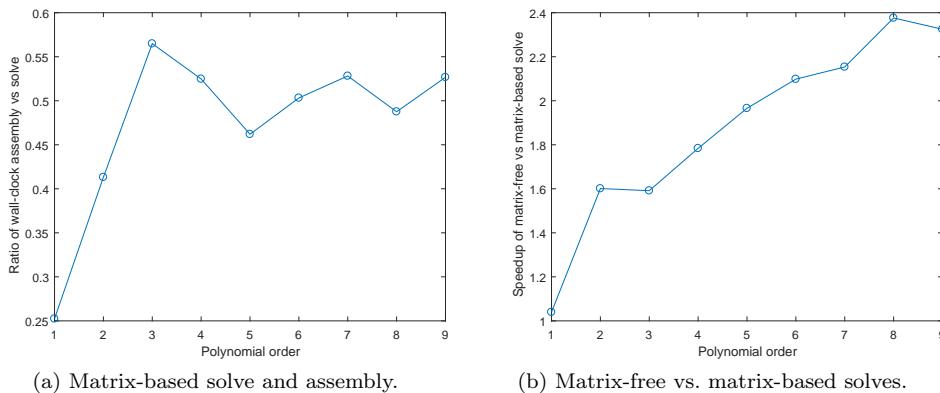


FIG. 10. (a) *Ratio of wall-clock timings for the matrix-based solve and assembly.* (b) *Speedup for matrix-free vs. matrix-based solves.*

We acknowledge that the benefits of matrix-free methods will vanish if efficient preconditioners are not available. In such cases, in order to have more options for robust preconditioners, the matrix will have to be stored.

**9. Conclusions.** We presented a highly efficient multigrid technique for solving a second order elliptic PDE with the HDG discretization. Both the multigrid technique and the HDG discretization are amenable to high-throughput low-memory environments like that provided by the KNL architecture; this was demonstrated by using thorough profiling and utilizing a roofline analysis [57]. The HDG method has much of its computation localized due to the discontinuous nature of the solution technique. Moreover, the HDG method brings static condensation to DG methods, which significantly reduces the number of nonzeros in the discretization operator. This in turn means that less work is required for a linear solver to obtain solutions. Since the HDG method converges with optimal orders for all of the variables it approximates, a local element-by-element postprocessing is available.

Any iterative solver will require sparse matrix vector multiplication (matrix-free or otherwise), including multigrid. Two different approaches were examined for sparse matrix vector multiplication: matrix-based and matrix-free. For high order HDG, the matrix-free technique better utilizes the resources available on the KNL, because of increased arithmetic intensity. In the high order regime, the matrix-based technique requires a sparse matrix storage, which increases time to solution. This is mainly because of additional assembly time, low arithmetic intensity, and erratic memory access patterns for sparse matrix vector multiplication.

Our algorithm is able to attain 80% of peak bandwidth performance for higher order polynomials. This is possible due to the data locality inherent in the HDG method. Very good performance is realized for high order schemes, due to good arithmetic intensity, which declines as the order is reduced. The performance is lower for polynomial orders  $p < 6$ . With an approach similar to the work done in [38], where multiple cells are processed by a thread block, one would be able to increase performance for lower order polynomials. We observed speedups when compared to a multicore CPU for the HDG methods components, namely, volume-to-surface mapping, surface-to-volume mapping, matrix-free matrix vector multiplication, and local matrix generation. The ratio of peak FLOP rates on the two target architectures was roughly 100X, and peak bandwidths was 5X, so this figure fits with our model of the computation. This is possible due to the data locality inherent in the HDG method. Very good performance is realized for high order schemes, due to good arithmetic intensity, which declines as the order is reduced.

An attractive feature of the KNL is that it can be programmed with traditional parallel paradigms like OpenMP, MPI, and pthreads. Thus, one can harness the massive fine-grain parallelism that the KNL offers by utilizing these traditional parallel paradigms with significantly limited intrusion.

## REFERENCES

- [1] M. ADAMS, M. BREZINA, J. HU, AND R. TUMINARO, *Parallel multigrid smoothing: Polynomial versus Gauss-Seidel*, J. Comput. Phys., 188 (2003), pp. 593–610.
- [2] P. F. ANTONIETTI, M. SARTI, AND M. VERANI, *Multigrid algorithms for hp-discontinuous Galerkin discretizations of elliptic problems*, SIAM J. Numer. Anal., 53 (2015), pp. 598–618, <https://doi.org/10.1137/130947015>.
- [3] P. F. ANTONIETTI, M. SARTI, AND M. VERANI, *Multigrid algorithms for high order discontinuous Galerkin methods*, in Domain Decomposition Methods in Science and Engineering XXII, Springer, 2016, pp. 3–13.
- [4] H. ANZT, S. TOMOV, J. DONGARRA, AND V. HEUVELINE, *Weighted block-asynchronous iteration on GPU-accelerated systems*, in Euro-Par 2012: Parallel Processing Workshops, Springer, 2013, pp. 145–154.
- [5] H. ANZT, S. TOMOV, M. GATES, J. DONGARRA, AND V. HEUVELINE, *Block-asynchronous multigrid smoothers for GPU-accelerated systems*, Proc. Comput. Sci., 9 (2012), pp. 7–16.
- [6] D. N. ARNOLD, F. BREZZI, B. COCKBURN, AND L. D. MARINI, *Unified analysis of discontinuous Galerkin methods for elliptic problems*, SIAM J. Numer. Anal., 39 (2002), pp. 1749–1779, <https://doi.org/10.1137/S0036142901384162>.
- [7] P. BASTIAN, *Load balancing for adaptive multigrid methods*, SIAM J. Sci. Comput., 19 (1998), pp. 1303–1321, <https://doi.org/10.1137/S1064827596297562>.
- [8] P. BASTIAN, G. WITTUM, AND W. HACKBUSCH, *Additive and multiplicative multi-grid: A comparison*, Computing, 60 (1998), pp. 345–364.
- [9] M. W. BENSON, *Frequency domain behavior of a set of parallel multigrid smoothing operators*, Internat. J. Comput. Math., 36 (1990), pp. 77–88.
- [10] J.-P. BERRUT AND L. N. TREFETHEN, *Barycentric Lagrange interpolation*, SIAM Rev., 46 (2004), pp. 501–517, <https://doi.org/10.1137/S0036144502417715>.
- [11] A. BRANDT, *Multi-level adaptive solutions to boundary-value problems*, Math. Comp., 31 (1977), pp. 333–390.

- [12] A. BRANDT AND O. LIVNE, *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics*, revised ed., Classics Appl. Math. 67, SIAM, Philadelphia, 2011, <https://doi.org/10.1137/1.9781611970753>.
- [13] S. C. BRENNER, J. CUI, AND L.-Y. SUNG, *Multigrid methods for the symmetric interior penalty method on graded meshes*, Numer. Linear Algebra Appl., 16 (2009), pp. 481–501.
- [14] S. C. BRENNER AND J. ZHAO, *Convergence of multigrid algorithms for interior penalty methods*, Appl. Numer. Anal. Comput. Math., 2 (2005), pp. 3–18.
- [15] O. BRÖKER AND M. J. GROTE, *Sparse approximate inverse smoothers for geometric and algebraic multigrid*, Appl. Numer. Math., 41 (2002), pp. 61–80.
- [16] C. CANTWELL, S. SHERWIN, R. KIRBY, AND P. KELLY, *From h to p efficiently: Strategy selection for operator evaluation on hexahedral and tetrahedral elements*, Comput. & Fluids, 43 (2011), pp. 23–28.
- [17] B. COCKBURN, B. DONG, AND J. GUZMÁN, *A superconvergent LDG-hybridizable Galerkin method for second-order elliptic problems*, Math. Comp., 77 (2008), pp. 1887–1916.
- [18] B. COCKBURN, B. DONG, J. GUZMÁN, M. RESTELLI, AND R. SACCO, *A hybridizable discontinuous Galerkin method for steady-state convection-diffusion-reaction problems*, SIAM J. Sci. Comput., 31 (2009), pp. 3827–3846, <https://doi.org/10.1137/080728810>.
- [19] B. COCKBURN, O. DUBOIS, J. GOPALAKRISHNAN, AND S. TAN, *Multigrid for an HDG method*, IMA J. Numer. Anal., 34 (2014), pp. 1386–1425.
- [20] B. COCKBURN, J. GOPALAKRISHNAN, AND R. LAZAROV, *Unified hybridization of discontinuous Galerkin, mixed, and continuous Galerkin methods for second order elliptic problems*, SIAM J. Numer. Anal., 47 (2009), pp. 1319–1365, <https://doi.org/10.1137/070706616>.
- [21] R. P. FEDORENKO, *The speed of convergence of one iterative process*, U.S.S.R. Comput. Math. Math. Phys., 4 (1964), pp. 559–564.
- [22] J. GOPALAKRISHNAN AND G. KANSCHAT, *A multilevel discontinuous Galerkin method*, Numer. Math., 95 (2003), pp. 527–550.
- [23] J. GOPALAKRISHNAN AND S. TAN, *A convergent multigrid cycle for the hybridized mixed method*, Numer. Linear Algebra Appl., 16 (2009), pp. 689–714.
- [24] W. HACKBUSCH, *Multi-Grid Methods and Applications*, Springer Series in Computational Mathematics, Springer, Berlin, Heidelberg, 2013.
- [25] A. HEINECKE, H. PABST, AND G. HENRY, *LIBXSMM: A High Performance Library for Small Matrix Multiplications*, <https://github.com/hfp/libxsmm>.
- [26] P. W. HEMKER, W. HOFFMANN, AND M. H. VAN RAALTE, *Two-level Fourier analysis of a multigrid approach for discontinuous Galerkin discretization*, SIAM J. Sci. Comput., 25 (2003), pp. 1018–1041, <https://doi.org/10.1137/S1064827502405100>.
- [27] P. W. HEMKER, W. HOFFMANN, AND M. H. VAN RAALTE, *Fourier two-level analysis for discontinuous Galerkin discretization with linear elements*, Numer. Linear Algebra Appl., 11 (2004), pp. 473–491.
- [28] P. HEMKER AND M. VAN RAALTE, *Fourier two-level analysis for higher dimensional discontinuous Galerkin discretisation*, Comput. Vis. Sci., 7 (2004), pp. 159–172.
- [29] A. HUERTA, A. ANGELOSKI, X. ROCA, AND J. PERAIRE, *Efficiency of high-order elements for continuous and discontinuous Galerkin methods*, Internat. J. Numer. Methods Engrg., 96 (2013), pp. 529–560.
- [30] Intel Math Kernel Library. Reference Manual, 2009, <https://software.intel.com/en-us/mkl>.
- [31] J. JEFFERS, J. REINDERS, AND A. SODANI, *Intel Xeon Phi Processor High Performance Programming: Knights Landing Edition*, Morgan Kaufmann, 2016.
- [32] K. JOHANSEN, *Multigrid Methods for NIPG*, Tech. Report 0531, ICES, 2005.
- [33] K. JOHANSEN, *A Symmetric Smoother for the Nonsymmetric Interior Penalty Discontinuous Galerkin Discretization*, Tech. Report 0523, ICES, 2005.
- [34] J. L. K. J. FIDKOWSKI, T. A. OLIVER, AND D. L. DARMOFAL, *p-multigrid solution of high-order discontinuous Galerkin discretizations of the compressible Navier-Stokes equations*, J. Comput. Phys., 207 (2005), pp. 92–113.
- [35] G. KARNIADAKIS AND S. J. SHERWIN, *Spectral/hp Element Methods for CFD*, Numer. Math. Sci. Comput., Oxford University Press, 1999.
- [36] J. KING, S. YAKOVLEV, Z. FU, R. M. KIRBY, AND S. J. SHERWIN, *Exploiting batch processing on streaming architectures to solve 2D elliptic finite element problems: A hybridized discontinuous Galerkin (HDG) case study*, J. Sci. Comput., 60 (2014), pp. 457–482.
- [37] R. M. KIRBY, S. J. SHERWIN, AND B. COCKBURN, *To CG or to HDG: A comparative study*, J. Sci. Comput., 51 (2012), pp. 183–212.
- [38] M. G. KNEPLEY, K. RUPP, AND A. R. TERREL, *Finite Element Integration with Quadrature on the GPU*, preprint, <https://arxiv.org/abs/1607.04245>, 2016.
- [39] M. KRONBICHLER AND K. KORMANN, *A generic interface for parallel cell-based finite element*

- operator application*, Comput. & Fluids, 63 (2012), pp. 135–147.
- [40] S. LOISEL, R. NABBEN, AND D. B. SZYLD, *On hybrid multigrid-Schwarz algorithms*, J. Sci. Comput., 36 (2008), pp. 165–175.
  - [41] J. W. LOTTES AND P. F. FISCHER, *Hybrid multigrid/Schwarz algorithms for the spectral element method*, J. Sci. Comput., 24 (2005), pp. 45–78.
  - [42] J. D. MCCALPIN, *STREAM: Sustainable Memory Bandwidth in High Performance Computers*, <https://www.cs.virginia.edu/stream/> (accessed: 2016-08-30).
  - [43] W. F. MITCHELL, *The hp-multigrid method applied to hp-adaptive refinement of triangular grids*, Numer. Linear Algebra Appl., 17 (2010), pp. 211–228.
  - [44] A. NAPOV AND Y. NOTAY, *When does two-grid optimality carry over to the v-cycle?*, Numer. Linear Algebra Appl., 17 (2010), pp. 273–290.
  - [45] X. ROCA, N. C. NGUYEN, AND J. PERAIRE, *GPU-accelerated sparse matrix-vector product for a hybridizable discontinuous Galerkin method*, in Aerospace Sciences Meetings, American Institute of Aeronautics and Astronautics, 2011, pp. 2011–687.
  - [46] Y. SAAD, *Iterative Methods for Sparse Linear Systems*, 2nd ed., SIAM, Philadelphia, 2003, <https://doi.org/10.1137/1.9780898718003>.
  - [47] A. SAMII, N. PANDA, C. MICHOSKI, AND C. DAWSON, *A hybridized discontinuous Galerkin method for the nonlinear Korteweg-de Vries equation*, J. Sci. Comput., 68 (2016), pp. 191–212.
  - [48] O. SCHENK AND K. GÄRTNER, *Solving unsymmetric sparse systems of linear equations with PARDISO*, Future Generation Computer Systems, 20 (2004), pp. 475–487.
  - [49] J. STILLER, *Nonuniformly Weighted Schwarz Smoothers for Spectral Element Multigrid*, preprint, <https://arxiv.org/abs/1512.02390>, 2015.
  - [50] J. STILLER, *Robust Multigrid for High-Order Discontinuous Galerkin Methods: A Fast Poisson Solver Suitable for High-Aspect Ratio Cartesian Grids*, preprint, <https://arxiv.org/abs/1603.02524>, 2016.
  - [51] S. TAN, *Iterative Solvers for Hybridized Finite Element Methods*, Ph.D. Thesis, University of Florida, 2009.
  - [52] L. N. TREFETHEN, *Approximation Theory and Approximation Practice*, SIAM, Philadelphia, 2013.
  - [53] U. TROTTERBERG, C. OOSTERLEE, AND A. SCHÜLLER, *Multigrid*, Academic Press, 2001.
  - [54] P. E. VOS, S. J. SHERWIN, AND R. 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. Phys., 229 (2010), pp. 5161–5181.
  - [55] H. WANG, D. HUYBRECHS, AND S. VANDEWALLE, *Explicit barycentric weights for polynomial interpolation in the roots or extrema of classical orthogonal polynomials*, Math. Comp., 83 (2014), pp. 2893–2914.
  - [56] P. WESSELING, *An Introduction to Multigrid Methods*, Pure Appl. Math. (N.Y.), John Wiley & Sons, New York, 1992.
  - [57] S. WILLIAMS, A. WATERMAN, AND D. PATTERSON, *Roofline: An insightful visual performance model for multicore architectures*, Commun. ACM, 52 (2009), pp. 65–76.
  - [58] S. YAKOVLEV, D. MOXEY, R. M. KIRBY, AND S. J. SHERWIN, *To CG or to HDG: A comparative study in 3D*, J. Sci. Comput., 67 (2016), pp. 192–220.