# A performance comparison of continuous and discontinuous Galerkin methods with fast multigrid solvers<sup>\*</sup>

Martin Kronbichler<sup>†</sup> Wolfgang A. Wall<sup>†</sup>

December 12, 2016

#### Abstract

This study presents a fair performance comparison of the continuous finite element method, the symmetric interior penalty discontinuous Galerkin method, and the hybridized discontinuous Galerkin method. Modern implementations of high-order methods with state-of-the-art multigrid solvers for the Poisson equation are considered, including fast matrixfree implementations with sum factorization on quadrilateral and hexahedral elements. For the hybridized discontinuous Galerkin method, a multigrid approach that combines a grid transfer from the trace space to the space of linear finite elements with algebraic multigrid on further levels is developed. Despite similar solver complexity of the matrix-based HDG solver and matrix-free geometric multigrid schemes with continuous and discontinuous Galerkin finite elements, the latter offer up to order of magnitude faster time to solution, even after including the superconvergence effects. This difference is because of vastly better performance of matrix-free operator evaluation as compared to sparse matrix-vector products. A roofline performance model confirms the advantage of the matrix-free implementation.

**Keywords.** High-order finite elements, Discontinuous Galerkin method, Hybridizable discontinuous Galerkin, Multigrid method, Matrix-free method, Highperformance computing

# 1 Introduction

The relative efficiency of various realizations of the discontinuous Galerkin (DG) method as compared to continuous finite elements (continuous Galerkin, CG) has been the subject of a number of recent studies [20, 27, 31, 51]. The hybridizable discontinuous Galerkin (HDG) method [13, 41] has attracted particular interest because it promises a more efficient solution of linear systems than other discontinuous Galerkin methods in terms of the number of degrees of freedom and nonzero entries in the system matrix. As opposed to continuous finite elements or symmetric interior penalty discontinuous Galerkin [3] methods that rely on the primal formulation of the differential equation, the HDG method poses the problem in mixed form using an additional variable for

<sup>\*</sup>This work was partially supported by the German Research Foundation (DFG) under the project "High-order discontinuous Galerkin for the exa-scale" (ExaDG) within the priority program "Software for Exascale Computing" (SPPEXA), grant agreement no. KR4661/2-1 and WA1521/18-1. The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time on the GCS Supercomputer SuperMUC at Leibniz Supercomputing Centre (LRZ, www.lrz.de) through project id pr83te.

<sup>&</sup>lt;sup>†</sup>Institute for Computational Mechanics, Technical University of Munich, Boltzmannstr. 15, 85748 Garching b. München, Germany ({kronbichler,wall}@lnm.mw.tum.de).

the flux. In order to avoid solving a linear system involving both the primal and flux unknowns, an initially counter-intuitive step is taken by introducing yet another variable, the so-called trace variable defined on the mesh skeleton. The numerical fluxes in the mixed system are solely expressed in terms of the trace variable and the local unknowns, avoiding direct coupling between neighboring elements. As a consequence, all element unknowns of the primal and flux variables can be eliminated prior to solving the global linear system by an element-by-element Schur complement, resulting in a global system in terms of the trace variable only. This Schur complement approach is conceptually the same as the technique of static condensation in continuous finite elements that eliminates the unknowns with coupling inside a single element as a means for improving the efficiency of the solution stage, see e.g. [13, 20, 27, 38, 51] and references therein.

Previous efficiency comparisons have found that HDG is a highly competitive option for two-dimensional problems and direct solvers [27, 31]. This work widens the perspective by considering large-scale problems which demand for iterative solvers and optimal complexity preconditioners. In that setting it is not enough to characterize the sparsity structure in the linear system of equations or the nonzero entries in the matrix as a proxy for the cost of one operator evaluation. Instead, the interesting factors for competitive solver times are the preconditioner efficiency and complexity, i.e., the iteration counts, the number of matrix-vector products per iteration, and timings for one matrix-vector product. This work considers multigrid methods which are among the most competitive solvers for elliptic operators on general meshes [19]. Geometric multigrid (GMG) methods combine simple iterative schemes on a hierarchy of coarser meshes. Different error frequencies are attacked on different mesh levels, such that simple iterative schemes that smooth the respective high frequencies on each level can be used. For certain applications, GMG is too restrictive because a mesh coarsening must be explicitly constructed or additional measures need to be taken for more complex differential operators like operator-dependent coarsening or non-standard smoothers [15, 48]. Algebraic multigrid methods are often used as an alternative, in particular on unstructured meshes, but at a somewhat higher cost in case more structure is available [19]. In the context of high-order methods, algebraic multigrid schemes need to be carefully set up in order to not coarsen too aggressively due to the dense coupling of the wider bases [25]. Therefore, *p*-multigrid methods are often considered in the high-order finite element context with a first transfer to a low-order, usually linear, finite element basis [39] before continuing with the algebraic hierarchy construction.

A second aspect that has not been covered by previous performance comparisons is the fact that competitive high-order implementations in the primal formulation are not based on matrices but rather most efficiently implemented by matrix-free operator evaluation. A major reason for favoring matrix-free methods is that most matrix-based iterative solvers are highly memory bandwidth limited when executed on modern processors [44] and alternatives that access less memory can be faster also when performing more arithmetic operations. In the low-order case with linear shape functions, the most competitive matrix-free schemes typically rely on a stencil representation, e.g. the block structure in hierarchical hybrid grids [8, 21]. In case of higher polynomial degrees, on-the-fly evaluation of cell and face integrals with sum factorization is the preferred choice due to a low evaluation complexity. Sum factorization is a technique established by the spectral element community [30, 33, 43] and used in well-established codes like Nek5000 [16], SPECFEM 3D [32], or Nektar++ [50]. These methods are also popular in the DG community [6, 36]. At higher polynomial degrees k in 3D, sum factorization for computing the integrals in matrix-vector products of continuous finite elements has the same  $\mathcal{O}(k^4)$  complexity per element as the matrix after static condensation, but without a direct relation to the stencil width of the matrix. This work uses the framework from

[36] that has been demonstrated to perform particularly well yet being flexible with respect to implementing generic differential operators and providing equally optimized code paths for both continuous and discontinuous Galerkin schemes. The implementation is available through the general-purpose finite element library deal.II<sup>1</sup> [5]. To the best of our knowledge, the numbers presented in this study use the fasted CPU code of this kind available through a common continuous and discontinuous finite element framework. The numbers are up to an order of magnitude faster than what was reported in the recent study [19] with similar general-purpose geometric multigrid schemes, reaching or even outperforming the specialized HPGMG [2] benchmark code. For stateof-the-art implementations of distributed sparse matrix algebra, the Trilinos<sup>2</sup> package [23] is used, providing a fair test bed with mature implementations for both ends. The present study is novel in comparing optimized sum factorization solvers to high-performance and optimal complexity iterative solvers for HDG. Even though the authors in [51] (Remark 1) claim to use some form of sum factorization, the implementation presented in this study leads to vastly different conclusions, showing that previous work has missed some of the relevant aspects.

The remainder of this work is structured as follows. Section 2 introduces the Poisson equation and the discretizations with the continuous finite element method, the symmetric interior penalty method, and the hybridizable discontinuous Galerkin method. An analysis of matrix-vector products of the various methods, including suggested alternatives for HDG as opposed to the sparse trace matrix, are given in Section 3. The computational time to reach a certain level of accuracy as well as performance metrics of the Poisson solvers are given in Section 4. Section 5 summarizes our findings.

# 2 Discretization of Poisson's equation

We consider the Poisson equation as a model problem for elliptic operators,

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

where u is the solution variable,  $\kappa > 0$  is a diffusion coefficient bounded uniformly away from zero, and  $\Omega$  is a bounded subset of *d*-dimensional space  $\mathbb{R}^d$ . The domain boundary  $\partial\Omega$  is partitioned into a Dirichlet portion  $\Gamma_{\rm D}$  where  $u = g_{\rm D}$  and a Neumann portion  $\Gamma_{\rm N}$  where  $-\vec{n} \cdot \kappa \nabla u = g_{\rm N}$  is prescribed, respectively. Here,  $\vec{n}$  denotes the unit outer normal vector on the boundary  $\Gamma_{\rm N}$ .

For discretization, we assume a tesselation  $\mathcal{T}_h$  of the computational domain  $\Omega$  into  $n_e$  elements  $\Omega_e$ , associated with a mesh size parameter h. In this work, we assume a mesh consisting of quadrilateral or hexahedral elements which allow for the most straight-forward and efficient implementation of sum factorization, with lower proportionality constants than tensorial techniques for triangles and tetrahedra [50, 45]. All work known to the authors indicate that matrix-based HDG schemes on quadrilaterals and hexahedra are at least as efficient as on tetrahedra [31, 37, 51], suggesting that our results are unbiased in comparing against the fastest matrix-based options. We assume an element  $\Omega_e$  to be the image of the reference domain  $[-1, 1]^d$  under a polynomial mapping of degree l, based on Gauss-Lobatto support points that are placed according to a manifold

<sup>&</sup>lt;sup>1</sup>http://www.dealii.org, retrieved on November 13, 2016. The implementations used in this study are extensions of the step-37 and step-51 tutorial programs of deal.II, program URLs: https://dealii.org/developer/doxygen/deal.II/step\_37.html and https://dealii.org/developer/doxygen/deal.II/step\_51.html. A note to reviewers of the manuscript: The DG functionality has currently not yet been made available in the deal.II library but it will be merged during December 2016. Thus, this comment will vanish in the final version of the manuscript.

<sup>&</sup>lt;sup>2</sup>http://www.trilinos.org, retrieved on July 20, 2016

description of the computational domain. This enables high-order approximations of curved boundaries and possibly also in the interior of  $\Omega$ . For the methods described below, we denote the bilinear forms associated to integrals over the elements of the triangulation as well as the faces by

$$(a,b)_{\mathcal{T}_h} = \sum_{\Omega_e \in \mathcal{T}_h} \int_{\Omega_e} a \odot b \, d\vec{x}, \quad \langle a,b \rangle_{\partial \mathcal{T}_h} = \sum_{\Omega_e \in \mathcal{T}_h} \sum_{F \in \text{faces}(\Omega_e)} \int_F a \odot b \, d\vec{s}, \quad (2)$$

where a, b can be scalar-valued, vector-valued, or tensor-valued quantities and  $\odot$  denotes the sum of the product in each component.

#### 2.1 Continuous Galerkin approximation

We assume a polynomial approximation of the solution on elements from the space

$$V_h^{\text{CG}} = \left\{ v_h \in H^1(\Omega) : v_h |_{\Omega^e} \in \mathcal{Q}_k(\Omega^e) \; \forall \Omega_e \in \mathcal{T}_h \right\},\tag{3}$$

where  $Q_k(\Omega_e)$  denotes the space of tensor product polynomials of tensor degree k on the element  $\Omega_e$ . In this work, we consider a basis representation by Lagrange polynomials in the nodes of the (k + 1)-point Gauss–Lobatto–Legendre quadrature rule for well-conditioned high order approximation [30]. However, the exact form of the basis is immaterial, as long as it is represented by a tensor product of 1D formulas. The solution space is then restricted to the space  $V_{h,g_{\rm D}}^{\rm CG}$  of functions in  $V_h^{\rm CG}$  which satisfy the boundary condition  $g_{\rm D}$  on  $\Gamma_{\rm D}$  by projection or interpolation.

The discrete finite element version of the Poisson equation (1) is found by multiplication by a test function, integration over  $\Omega$ , integration by parts of the left hand side, and insertion of the Neumann boundary condition. The final weak form is to find a function  $u_h \in V_{h,q_D}^{CG}$  such that

$$(\nabla v_h, \kappa \nabla u_h)_{\mathcal{T}_h} = (v_h, f)_{\mathcal{T}_h} - \langle v_h, g_N \rangle_{\partial T_h \cap \Gamma_N}$$
(4)

holds for all test functions  $v_h \in V_{h,0_D}^{CG}$  that are zero on the Dirichlet boundary.

On each element, the left-hand side gives rise to an element stiffness matrix  $K^e$  and the right-hand side to an element load vector  $b^e$ . These local quantities are assembled into the global stiffness matrix K and the load vector b in the usual finite element way, including the elimination of Dirichlet rows and columns. In case of static condensation, the  $(k - 1)^d$  out of  $(k + 1)^d$  degrees of freedom pertaining to basis functions with support on a single element are eliminated by a Schur complement, reducing the final system size accordingly. We refer to [20, 51] for details.

# 2.2 Symmetric interior penalty discontinuous Galerkin discretization

As a discontinuous Galerkin representative targeting the primal equation amenable to sum factorization, we choose the symmetric interior penalty (SIP) discontinuous Galerkin method [3]. In a discontinuous Galerkin method, only  $L_2$  regularity of the solution is required and no continuity over element boundaries is enforced,

$$V_h^{\mathrm{DG}} = \{ v_h \in L_2(\Omega) : v_h |_{\Omega_e} \in \mathcal{Q}_k(\Omega_e) \ \forall \Omega_e \in \mathcal{T}_h \}.$$
(5)

On each element, the same steps as for continuous Galerkin in terms of multiplication by test functions, integration over an element and integration by parts, are taken. Due to the missing intra-element continuity, the terms  $-v_h \vec{n} \cdot \kappa \nabla u_h$  do not drop out over the interior faces of the mesh and must be connected by a numerical flux on  $\nabla u_h$ . The first step is to take the average  $\frac{1}{2} \left( \nabla u_h^- + \nabla u_h^+ \right)$  of the solution from both elements  $e^-$  and  $e^+$  sharing a face.

In order to ensure adjoint consistency through a symmetric weak form, the term  $-\frac{1}{2}(u_h^- \vec{n}^- + u_h^+ \vec{n}^+) \cdot \kappa \nabla v_h = -\frac{1}{2}(u_h^- - u_h^+)\vec{n}^- \cdot \kappa \nabla v_h$  is added. This term is consistent with the original equation because the difference  $(u_h^- - u_h^+)$  is zero for the analytic solution, see also [24] for a derivation from a first-order system. Finally, a penalty term  $\vec{n}v_h\kappa\sigma(u_h^- - u_h^+)\vec{n}^-$  is added for ensuring coercivity of discrete operator. The penalty parameter  $\sigma = (k+1)^2 \frac{d}{h}$  in d dimensions depends on the inverse mesh size h on uniform meshes and is extended to general meshes by a formula involving surface area and volume from [26]. No tuning with respect to  $\sigma$  is done in this work. As documented in the literature [24], condition numbers and multigrid performance would deteriorate as  $\sigma$  is increased. This gives the following weak form for the DG-SIP method,

$$(\nabla v_h, \kappa \nabla u_h)_{\mathcal{T}_h} - \left\langle v_h \vec{n}, \kappa \frac{\nabla u_h^- + \nabla u_h^+}{2} \right\rangle_{\partial \mathcal{T}_h} - \left\langle \frac{\nabla v_h}{2}, \kappa \vec{n} (u_h^- - u_h^+) \right\rangle_{\partial \mathcal{T}_h}$$
$$+ \left\langle v_h, \kappa \sigma (u_h^- - u_h^+) \right\rangle_{\partial \mathcal{T}_h} = (v_h, f)_{\mathcal{T}_h}$$
(6)

which is to hold for all test functions  $v_h$  in the space  $V_h^{\text{DG}}$ . Note that the bilinear forms  $\langle \cdot, \cdot \rangle_{\partial \mathcal{T}_h}$  visit each interior face twice with opposite directions of the normal vector  $\vec{n}$ , resulting in a symmetric weak form. Boundary conditions are imposed by defining suitable extension values  $u^+$  in terms of the boundary condition and the inner solution value  $u^-$ ,

$$u^{+} = -u^{-} + 2g_{\rm D}, \quad \nabla u^{+} = \nabla u^{-}, \qquad \text{on Dirichlet boundaries,}$$
  
 $u^{+} = u^{-}, \qquad \nabla u^{+} \cdot \vec{n} = -\nabla u^{-} \cdot \vec{n} - 2\frac{g_{\rm N}}{\kappa}, \quad \text{on Neumann boundaries.}$ 
(7)

Thus, additional contributions of known quantities arise that are eventually moved to the right-hand side of the final linear system.

### 2.3 Hybridizable discontinuous Galerkin discretization

For the hybridizable discontinuous Galerkin (HDG) discretization [13], the Poisson equation (1) is rewritten as a first-order system by introducing a flux variable  $\vec{q} = -\kappa \nabla u$  in the equation  $\nabla \cdot \vec{q} = f$ . The discrete solution spaces are  $V_h^{\text{DG}}$  for  $u_h$  and  $(V_h^{\text{DG}})^d$  for  $\vec{q}_h$ . An additional trace variable  $\lambda_h$  that approximates  $u_h$  on the interface between elements is introduced. It is defined by polynomials on the mesh skeleton

$$M_{h}^{\text{tHDG}} = \{\mu_{h} \in L_{2}\left(\mathcal{F}_{h}\right) : \mu_{h}|_{F} \in \mathcal{Q}_{k}(F) \ \forall \text{ faces } F \in \mathcal{F}_{h}\},\tag{8}$$

where  $\mathcal{F}_h$  denotes the collection of all faces in the discretization  $\mathcal{T}_h$ . The functions in  $M_h^{\text{tHDG}}$  are discontinuous between faces (i.e., over vertices in 2D, over vertices and edges in 3D). Besides tensor product polynomials  $\mathcal{Q}_k(F)$ , we will also consider polynomials of complete degree k,  $\mathcal{P}_k(F)$ , in combination with the space  $\mathcal{P}_k(\Omega_e)$ , easily permitted by the discontinuous formulation on F and  $\Omega_e$ .

For deriving the HDG weak form, the system is multiplied by test functions  $\vec{w}_h$ ,  $v_h$ , integrated over element  $\Omega_e$ , and gradient and divergence terms are integrated by parts. For the numerical fluxes, we add a new variable for the first flux,  $\hat{u} = \lambda_h$ , while the second flux is set to  $\hat{\vec{q}} = \vec{q}_h + \tau(u_h - \lambda_h)\vec{n}$ . The system is closed by enforcing continuity on the second numerical flux  $\hat{\vec{q}}$ , which ensures conservativity. The final weak form for the HDG method is to find the values  $\vec{q}_h \in (V_h^{\text{DG}})^d$ ,  $u_h \in V_h^{\text{DG}}$ , and  $\lambda_h \in M_{h,g_D}^{\text{tHDG}}$ , such that

$$\begin{aligned} \left(\vec{w}_{h}, \kappa^{-1}\vec{q}_{h}\right)_{\mathcal{T}_{h}} &- \left(\nabla \cdot \vec{w}_{h}, u_{h}\right)_{\mathcal{T}_{h}} + \langle \vec{w}_{h}, \vec{n}\lambda_{h} \rangle_{\partial\mathcal{T}_{h}} = 0, \\ \left(\nabla v_{h}, \vec{q}_{h}\right)_{\mathcal{T}_{h}} &+ \langle v_{h}, \vec{n} \cdot \vec{q}_{h} + \tau (u_{h} - \lambda_{h}) \rangle_{\partial\mathcal{T}_{h}} = - (v_{h}, f)_{\mathcal{T}_{h}}, \\ &- \langle \mu_{h}, \vec{n} \cdot \vec{q}_{h} + \tau (u_{h} - \lambda_{h}) \rangle_{\partial\mathcal{T}_{h}} = - \langle \mu_{h}, g_{N} \rangle_{\partial\mathcal{T}_{h} \cap \Gamma_{N}}, \end{aligned}$$
(9)

holds for all test functions  $\vec{w}_h \in (V_h^{\text{DG}})^d$ ,  $v_h \in V_h^{\text{DG}}$ , and  $\mu_h \in M_{h,0}^{\text{tHDG}}$ . Dirichlet conditions are imposed strongly on the trace space  $M_h^{\text{tHDG}}$  by projection.

The parameter  $\tau$  ensures stability if chosen  $\tau > 0$  [41]. However, selecting  $\tau \propto \kappa$  is advantageous because it gives optimal convergence rates k + 1 in both the solution and the flux. An element-by-element post-processing can then be performed to recover a solution  $u^*$  that converges at rate k + 2 [13]. Even though superconvergence has not been proved for quadrilaterals and hexahedra (which need special projection properties, as opposed to tetrahedra [14]), we observed superconvergence for all constant-coefficient elliptic test cases with shape-regular but otherwise arbitrary quadrilateral and hexahedral meshes, including the test case from [31] where the authors report only rates k + 1 on quadrilaterals. Note that the superconvergence of HDG essentially contributes to the high efficiency as compared to other methods documented in previous studies [27, 31, 51].

Following the notation in [41], the individual terms in Equation (9) are expanded in terms of the basis functions and put into matrix-vector form. The system reads

$$\begin{pmatrix} A & B^{\mathrm{T}} & C^{\mathrm{T}} \\ B & D & G^{\mathrm{T}} \\ -C & -G & H \end{pmatrix} \begin{pmatrix} \vec{Q}_h \\ \vec{U}_h \\ \Lambda_h \end{pmatrix} = \begin{pmatrix} \vec{0} \\ \vec{R}_f \\ \vec{R}_{g_{\mathrm{N}}} \end{pmatrix}.$$
 (10)

The upper left  $2 \times 2$  block is block-diagonal over elements and can be condensed out before solving the linear system. Thus, only a symmetric positive definite linear system  $K\Lambda_h = \vec{R}$  needs to be solved with

$$K = H + \begin{pmatrix} C & G \end{pmatrix} \begin{pmatrix} A & B^{\mathrm{T}} \\ B & D \end{pmatrix}^{-1} \begin{pmatrix} C^{\mathrm{T}} \\ G^{\mathrm{T}} \end{pmatrix}, \quad \vec{R} = \vec{R}_{g_{\mathrm{N}}} + \begin{pmatrix} C & G \end{pmatrix} \begin{pmatrix} A & B^{\mathrm{T}} \\ B & D \end{pmatrix}^{-1} \begin{pmatrix} \vec{0} \\ \vec{R}_{f} \end{pmatrix}.$$
(11)

# 3 Performance of operator evaluation

In this work, we consider high-performance matrix-free evaluation of matrixvector products whenever possible, relying on fast integration facilities established in spectral elements [30, 33]. While these methods had originally only been used in the high-degree context with  $k \ge 4$ , recent high-performance realizations taking the architecture of modern parallel computers into account have shown that matrix-free methods outperform sparse matrix kernels by several times already for k = 2 on quadrilaterals and hexahedra [10, 36].

In a matrix-free setting, the matrix-vector product is interpreted as a weak form that is tested by all basis functions in an element-by-element way. This corresponds to computing a residual vector on each element that is assembled into the global solution vector, given a function  $u_h$  associated to the input vector [36]. If denote by  $\vec{z} = L\vec{y}$  the global operator evaluation and by  $\vec{z}_e = L_e\vec{y}_e$  the contribution on element  $\Omega_e$ , the *i*-th component of the matrix-vector product is given by quadrature

$$(z_e)_i = \int_{\Omega_e} \kappa \nabla \phi_i \nabla u_h^{y_e} d\vec{x} \approx \sum_q \nabla_{\vec{\xi}} \phi_i \left(J^e\right)^{-1} \left(w_q \det(J^e)\kappa\right) \left(J^e\right)^{-\mathrm{T}} \nabla_{\vec{\xi}} u_h^{y_e}.$$
 (12)

In this expression,  $u_h^{y_e}$  is the finite element function associated to the nodal solution values  $\vec{y_e}, J$  is the Jacobian of the transformation from the reference to the real cell, and  $w_q$  the quadrature weight. In this work, we use Gauss–Legendre quadrature with k + 1 points per coordinate direction which exactly evaluates integrals on Cartesian geometries. The error on curved meshes does not affect convergence orders for elliptic problems in the current setting apart from the usual variational crime [9].

For the interpolation of the nodal values  $y_e$  to the quadrature points for  $\nabla_{\vec{\xi}} u_h^{y_e}$  in Equation (12) as well as the multiplication by the test function gradients  $\nabla_{\vec{\xi}} \phi_i$  for all test functions  $i = 1, \ldots, (k+1)^d$ , the sum factorization technique is used [36]. This approach replaces the direct interpolation over all points in d dimensions by a series of d one-dimensional interpolations. This reduces the evaluation complexity per element from  $\mathcal{O}((k+1)^{2d})$  operations in the naive matrix-vector product with a Kronecker matrix to  $\mathcal{O}(d(k+1)^{d+1})$  operations. The complexity of DG face integrals is only  $\mathcal{O}((k+1)^d)$ , i.e., linear in the number of unknowns [6].

Our implementation [34, 36] is available through the deal.II finite element library [5] and specifically targets modern computer architecture where access to main memory is usually the bottleneck for PDE-based kernels. This means that all operations on an element according to Equation (12) are done in close temporal proximity to service the sum factorization kernels from fast L1 caches. The complexity of one-dimensional kernels in sum factorization is further cut down into half by a high-degree optimization based on the even-odd decomposition [33]. A memory optimization is applied for Cartesian and affine meshes where the Jacobian of the transformation from the reference to the real cell is constant throughout the whole cell and needs only be kept once. In case cells are not affine, it is fastest [36] to pre-compute the Jacobian on all quadrature points and all cells prior to solving linear systems and loading the coefficients in each matrix-vector product, despite the relatively high memory transfer. The experiments below consider both the memory-intensive general mesh case and the simple Cartesian mesh case as they show different performance. For sparse linear algebra, the Epetra backend of Trilinos is used due to its mature state [23].

### 3.1 Variants of matrix-vector products

Since the continuous finite element method, DG-SIP, and the HDG trace matrix all involve different matrix sizes but the error is most closely related to the number of elements, the most appropriate metric for comparison would be the cost per element. On the other hand, we want to acknowledge the ability of higher order basis functions to use coarser meshes. Thus, we report the numbers in this section as the cost per element divided by  $k^d$ , the number of unique degrees of freedom (DoF) per element on a continuous finite element space. In other words, if a matrix-vector product on  $n_e$  elements of degree k takes  $t_{\rm mv}$ seconds, we report the quantity

Equivalent DoFs/s = 
$$\frac{n_e k^d}{t_{\rm mv}}$$
, (13)

uniformly across all discretization schemes. Obviously, this selection does not realistically represent discretization accuracy, a topic we postpone to Section 4. The measurements in this section have been performed on a fully utilized node of dual-socket Intel Xeon E5-2690 v4 (Broadwell) processors with  $2 \times$ 14 cores running at 2.6 GHz and eight memory channels. By using the full node, a fair balance between arithmetic bound kernels (sum factorization) and memory bound kernels (sparse matrix-vector products) is achieved. Reported memory bandwidth of this setup is approximately 130 GB/s in the STREAM triad benchmark [40] or sparse matrix-vector products, whereas the theoretical arithmetic peak is 940 GFLOP/s when measured at the AVX base frequency of 2.1 GHz. All C++ code has been compiled with the GNU compiler gcc, version 6.1, with optimization target AVX2 (Haswell). The minimum runtime out of five experiments is presented.

Fig. 1 compares the three discretization methods and several implementations of the matrix-vector product:

- CG matrix-free: A standard continuous Galerkin approximation with tensor product basis functions of degree k and Gaussian quadrature on  $(k+1)^d$  points according to Sec. 2.1 evaluated in a matrix-free way.
- CG static condensation matrix: Uses the most efficient sparse matrix representation obtained by static condensation of the  $(k-1)^d$  cell-interior degrees of freedom in continuous elements [27].
- DG-SIP matrix-free: Symmetric interior penalty DG method according to Sec. 2.2 with matrix-free implementation through sum factorization. No matrix is considered due to its low efficiency [27].
- HDG trace matrix: Sparse matrix-vector product with the trace matrix.
- HDG trace matrix post: Takes the increased accuracy of HDG by superconvergent post-processing into account. For this label, the data from "HDG trace matrix" measured at degree k - 1 is reported in terms of the equivalent degrees of freedom in Equation (13) of one degree higher, k<sup>d</sup>.
- HDG trace matrix-free: This approach considers an alternative implementation of the trace matrix-vector product where the matrix system (11) is expanded in terms of all contributing matrices rather than explicitly forming the Schur complement. This allows most matrix-vector products to be implemented in a matrix-free way, using a scheme originally proposed in [37] for fast computation of HDG residuals:
  - 1. Matrix-free multiplication by  $\begin{pmatrix} C^{\mathrm{T}} \\ G^{\mathrm{T}} \end{pmatrix}$  on input  $\Lambda_h$ .
  - 2. Application of the inverse matrix  $\begin{pmatrix} A & B^{\mathrm{T}} \\ B & D \end{pmatrix}^{-1}$  by an inner Schur complement that computes the nodal values  $\vec{U}_h$ ,  $\vec{Q}_h$  given the vectors  $\vec{R}_q$ ,  $\vec{R}_u$  from step 1:

$$\vec{U}_h = \left(D - BA^{-1}B^{\mathrm{T}}\right)^{-1} \left(\vec{R}_u - BA^{-1}\vec{R}_q\right),$$
$$\vec{Q}_h = A^{-1} \left(\vec{R}_q - B^{\mathrm{T}}\vec{U}_h\right).$$

In this equation, the matrix-vector multiplications by  $A^{-1}, B$ , and  $B^{\mathrm{T}}$  can all be implemented by matrix-free evaluation, including the inverse vector mass matrix  $A^{-1}$  for which fast sum factorization techniques exist [37]. Only the  $(k+1)^d \times (k+1)^d$  matrix  $(D-BA^{-1}B^{\mathrm{T}})^{-1}$  needs to be explicitly stored.

3. Matrix-free multiplication by  $\begin{pmatrix} C & G \end{pmatrix}$  and the mass matrix H.

This approach is counter-intuitive because it defeats the original purpose of the static condensation-type elimination of degrees of freedom. This approach is slower than the trace matrix-vector products for all degrees in 2D according to Fig. 1. However, up to twice as high performance until degree five is recorded in 3D due to much reduced memory transfer. The higher complexity  $(k + 1)^6$  per element of the multiplication by the dense matrix  $(D - BA^{-1}B^{T})^{-1}$  dominates over the trace complexity  $(k + 1)^4$ for k > 5.

• HDG mixed matrix-free: This approach replaces the condensation into the trace matrix by applying the numerical fluxes in a more classical DG way in terms of a system in mixed form in degrees of freedom for u and  $\vec{q}$ ,

$$\begin{pmatrix} A & B^{\mathrm{T}} \\ B & D \end{pmatrix} + \begin{pmatrix} C^{\mathrm{T}} \\ G^{\mathrm{T}} \end{pmatrix} H^{-1} \begin{pmatrix} C & G \end{pmatrix},$$



Figure 1: Number of degrees of freedom processed per second on 28 Broadwell cores as a function of the polynomial degree for various discretizations and implementations of the matrix-vector product of the 2D and 3D Laplacian.

This is the form taken for explicit time integration in many DG schemes, including HDG [37]. All matrix-vector products can be performed by sum factorization, including evaluation of  $H^{-1}$  that is an inverse face mass matrix [37] or alternatively can be implemented by point-wise fluxes.

# 3.2 Throughput analysis

The throughput of the matrix-vector products in terms of the number of equivalent degrees of freedom processed per second is measured at large matrix sizes of around 10 million where the solution vectors (and all other global data) need to be fetched from main memory rather than caches.

Fig. 1 shows the measurements in two and three space dimensions on both Cartesian meshes with constant coefficients and curved meshes with variable coefficients. In all tests, continuous finite elements show the best performance, apart from linear shape functions. As a point of reference, the operator evaluation with our implementation for  $Q_2$  shape functions on general grids is considerably faster at 340 million degrees of freedom per second than HPGMG<sup>3</sup> at 140 million degrees of freedom per second ( $64^3$  mesh), both run on the same hardware. This shows both the high level of optimization and the benefit of pre-computed Jacobians [36]. For  $k \geq 2$ , continuous elements provide more than twice the throughput of the next-best method, the DG-SIP method with matrix-free implementation. This goes against the preconception of the DG community [13] which attribute an implementation advantage to DG methods due to more structure and favorable vectorization properties. Note that the throughput of the DG-SIP method at 400 million equivalent degrees of freedom

<sup>&</sup>lt;sup>3</sup>https://hpgmg.org, retrieved on July 20, 2016

per second on the 3D Cartesian mesh translates to around 500–800 million DG degrees of freedom processed per second at polynomial degrees two through eight. A remarkable aspect of the matrix-free schemes is that throughput in terms of degrees of freedom per second appears approximately constant as the polynomial degree increases, despite the theoretical  $\mathcal{O}(k)$  complexity per degree of freedom. One reason for this behavior is that face integrals at cost  $\mathcal{O}(1)$  are dominating at lower degrees  $k \leq 4$  for DG-SIP and the HDG mixed form. Secondly, the even-odd decomposition [33] limits the cost increase at higher degrees. Finally, better arithmetic utilization is achieved for higher polynomial degree.

The results also show that HDG with post-processing (black dashed line) delivers approximately the same performance as matrix-free DG-SIP in two space dimensions. Without post-processing, HDG falls considerably behind the matrix-free schemes in 2D. In 3D, all matrix-based schemes are by an order of magnitude and more slower than the matrix-free schemes. Our results show that the post-processed HDG solution at assumed convergence rate k + 2 performs similarly to the statically condensed CG matrix at rate k + 1 because both have the same  $k^{d-1}$  unique degrees of freedom per face. In other words, the effect of superconvergence is offset by the discontinuous trace solution spaces (8) and embedded discontinuous Galerkin with traces based on the skeleton of the continuous finite element method as used in [42] actually appear more favorable in terms of accuracy efficiency. In the remainder of this study, the statically condensed CG results can be taken as a proxy for the performance of embedded DG methods.

### 3.3 Performance modeling

In order to document our unbiased comparison, Fig. 2 puts the achieved performance into perspective by a roofline performance model. The performance boundaries are the memory bandwidth limit (diagonal line to the left) and the arithmetic throughput limit (horizontal line to the upper right) in terms of the FLOP/byte ratio of the respective kernel [44]. The FLOP/byte ratio is measured by an optimistic assumption to memory access that only counts the global data structures that need to be streamed at least once per matrix-vector product, assuming perfect caching of re-usable data such as vector entries accessed by several elements and no other bottlenecks in the memory hierarchies. The arithmetic operations per element have been derived by formulas similar to the ones from [37] and verified by counting instructions of a run of the element kernel through the Intel Software Development Emulator.<sup>4</sup>

The results in Fig. 2 show that the sparse matrix-vector product displayed in the lower left corner is very close to the theoretical performance maximum of the compressed row storage scheme. The only available optimization of the matrixbased scheme except for a stencil representation in the constant-coefficient affine mesh case would be to use the DG matrix structure where blocks of entries are addressed by one index. In Trilinos, the class Epetra\_VbrMatrix implements such a scheme rather than the pointwise indirect addressing of the compressed row storage in Epetra\_CrsMatrix. This would increase the throughput by up to 50%, moving to a FLOP/byte ratio of 0.25. However, inefficiencies and limitations in the Trilinos implementation prohibit its use in general software such as a multigrid solver. Even if such an implementation were available, the performance model show today's hardware does not permit matrix-based methods to match the matrix-free implementations. Turning to the matrix-free implementations, Fig. 2 reveals a clear difference between the Cartesian case where the memory transfer is mostly due to the solution vector and some index data, and the curved mesh case where the Jacobian transformation also needs

<sup>&</sup>lt;sup>4</sup>https://software.intel.com, Version 7.45, AVX2 (Haswell) mode, retrieved on May 19, 2016.



Figure 2: Roofline model for the evaluation of the 3D Laplacian with different variants on a dual-socket Intel Xeon E5-2690 v4 (Broadwell), 28 cores. Polynomial degrees k = 1, 2, 4, 8 are considered. Small arrows indicate the behavior for increasing polynomial degrees.



Figure 3: Latency study of matrix-vector products for Laplacian on a  $80^3$  mesh with  $Q_2$  elements, involving 4.1 million DoFs.

to be loaded. The former is computation bound, whereas the latter resides in the memory bound region, albeit further to the right than sparse matrix kernels. Due to the considerably more complex kernel structures with many short loops and different operations, the achieved performance is not as close to the theoretical performance bounds as for the sparse matrix-vector kernel. Given that our implementation clearly outperforms the benchmark code HPGMG, the numbers can be considered extremely good nonetheless.

#### 3.4 Latency analysis

A second ingredient to practical solver performance is the latency of matrixvector products. This is relevant for multigrid schemes where a series of coarser representations appear and need to be processed quickly. This section reports results from the SuperMUC Phase 1 system (2 × 8 core Intel Xeon E5-2680 Sandy Bridge CPU running at 2.7 GHz, Infiniband FDR10 interconnect). Fig. 3 shows that the matrix-free variants scale down to  $10^{-4}$  seconds where network latency becomes dominant, a similar result as was reported for HPGMG [2]. This number needs to be compared to a single point-to-point latency of around  $10^{-6}$  seconds.

The sparse matrix-vector product of HDG already saturates at around  $5 \cdot 10^{-4}$  seconds. Investigation of this issue revealed sub-optimal MPI commands in the data exchange routines of the Epetra sparse matrix [23], involving a global

barrier operation in addition to point-to-point communication. However, the latency could not be reduced to less than  $2 \cdot 10^{-4}$  seconds even when changing the Trilinos source code. We conclude that given the right implementation, no advantage of the discontinuous data structures with less connectivity to neighbors appears on modern high-performance implementations of matrix-vector products, as opposed to direct solvers [27], showing that the results from [51] are not general. When going to considerably higher order elements than the  $Q_2$  basis reported in Fig. 3, the time to process a single element, or rather, a batch of four to eight elements due to vectorization over several elements in our implementation [36], overlays the communication latency. For DG-SIP, this transition occurs at k = 6 and at around k = 8 for CG.

# 4 Performance comparison of multigrid solvers

In this section, we analyze modern multigrid solvers and record the solution accuracy as a function of computing time. All solvers use a multigrid V-cycle as a preconditioner for a conjugate gradient iteration, which increases solver robustness [48, 47]. The iteration is stopped once the residual norm goes below  $10^{-9}$ times the right hand side norm. For coarse and low-order discretizations, this tolerance could be relaxed as e.g. done by the full multigrid cycle in HPGMG [2], but we refrain from this optimization for ease of comparison. The comparisons focus on the three-dimensional case where large-scale iterative solvers are essential. The trend for 2D is similar to 3D, but the advantage of the matrix-free schemes is less. In 2D, they provide two to eight times better efficiency, solving for 10 and 20 million unknowns per seconds on 28 cores for the continuous and discontinuous matrix-free solvers. The HDG solver reaches between 1 and 3 million degrees of freedom per second.

## 4.1 Multigrid solvers for matrix-free methods

Due to their fast matrix-vector products, a polynomial Chebyshev accelerated pointwise Jacobi smoother in a geometric multigrid cycle is the natural choice [1, 2, 36] for the CG and DG-SIP realizations. Besides matrix-vector products, the Chebyshev smoother only needs access to the matrix diagonal that is pre-computed and stored before solving. In addition, an estimate of the largest eigenvalue  $\tilde{\lambda}_{max}$  of the Jacobi-preconditioned matrix is used to make the Chebyshev iteration address modes with eigenvalue in the interval  $[0.06\tilde{\lambda}_{max}, 1.2\tilde{\lambda}_{max}]$ . The eigenvalue estimation is done by 15 iterations with the conjugate gradient method. In the numbers reported below, the setup cost of the multigrid ingredients is ignored, just as we ignore the cost for assembling the HDG trace matrix. We note that the setup of the matrix-free variants is proportional to two to four V-cycles, considerably less than the matrix creation and assembly for HDG. In the context of nonlinear systems where the system matrix changes rapidly, the advantage of matrix-free schemes will thus be even larger than what is reported here, a property exploited in [35].

For pre- and post-smoothing, a polynomial degree of five in the Chebyshev method is used, involving five matrix-vector products. The level transfer is based on the usual geometric embedding operations and also implemented by tensorial techniques. For the solver on the coarse grid, the Chebyshev iteration is selected, now with parameters such the a-priori error estimate for the Chebyshev iteration [49] ensures an error below  $10^{-3}$ . The implementation uses the multigrid facilities of the deal.II finite element library [28, 29], including adaptively refined meshes with hanging nodes in a massively parallel context based on a forest-of-tree data layout and Morton cell ordering [4, 11].

### 4.2 Multigrid for HDG

In the context of the HDG trace system, off-the-shelf AMG solvers such as Trilinos ML [18] work suboptimally or even fail because of a pronounced nondiagonally dominant character of the matrix together with wide stencils due to the high-order basis. To overcome these limitations, this work adopts a variation of the method proposed in [12], where the combination of a high-order HDG trace space with a continuous finite element discretization involving linear basis functions on the same mesh was proposed. This concept is closely related to p-multigrid methods [39] where the structure of a high-order basis is used by first going to a low-order basis with fewer unknowns rather than going to coarser meshes as in h-multigrid approaches. The transfer between the HDG trace space and linear finite elements is realized by the embedding operator that maps linear shape functions onto the trace polynomials as well as its transpose. As opposed to the work [12] that constructs a genuine discretization on the linear finite element space, we select a Galerkin coarse grid operator [48]. As shown below, the iteration count is only around 15-20, much better than 55-75 reported in [12] for similar tolerances that are probably to systematic gaps between different discretizations. The hierarchy is then continued by algebraic multigrid. Since the connectivity of the matrix is the same as for linear finite elements and the matrix is (almost) diagonally dominant, optimal or close-tooptimal performance of the AMG inside the *p*-AMG scheme can be expected.

Due to a strong non-diagonally dominant matrix structure, optimal multigrid performance in HDG cannot be obtained with point-relaxation smoothers. Instead, block-relaxation scheme with blocks combining all degrees of freedom on a face or incomplete factorizations are necessary. (Iteration numbers grow approximately as  $h^{-2/3}$  with point Gauss–Seidel smoothing.) Due to its robustness, ILU(0) is selected as a smoother both for the HDG trace matrix as well as in the AMG levels. As soon as the level matrix size goes below 2000, a direct coarse solver is invoked. For the HDG stabilization parameter, we select  $\tau = 5 \max{\kappa(\vec{x}), \vec{x} \in \text{face}}$  as a balance between solver efficiency and accuracy throughout this study, see also [31].

For the HDG trace matrix, we consider the representation by a sparse matrix because it easily combines with the ILU(0) for the smoother, even though somewhat higher performance would be available in 3D for  $1 \le k \le 5$  with a matrix-free implementation according to Fig. 1. Operator evaluation with the mixed form of HDG seems promising due to the considerably faster matrix-vector product reported in Fig. 1, but the saddle point form is more challenging to handle, requiring strong ingredients such as overlapping Schwarz smoothers or block factorizations [7].

For comparison, we also consider an iterative solver based on the statically condensed matrix for continuous elements. The standard ML-AMG V-cycle with one sweep of ILU(0) for pre- and post-smoothing on all levels is chosen. As seen from Table 1 below, no optimal iteration numbers are obtained in this case for higher polynomial degrees. However, it serves as a point of reference for matrix-based approaches with black-box preconditioning. Alternatively, a similar *p*-multigrid scheme with similar iteration counts as for the HDG trace system could also be considered.

### 4.3 Three-dimensional example with smooth solution

We consider the Poisson equation with analytic solution

$$u(\vec{x}) = \left(\frac{1}{\alpha\sqrt{2\pi}}\right)^3 \sum_{j=1}^3 \exp\left(-\|\vec{x} - \vec{x}_j\|^2 / \alpha^2\right),$$
(14)

given as a sum of three Gaussians centered at the positions  $\vec{x}_j \in \{(-0.5, 0.5, 0.25)^{\mathrm{T}}, (-0.6, -0.5, -0.125)^{\mathrm{T}}, (0.5, -0.5, 0.5)^{\mathrm{T}}\}$  and of width  $\alpha = \frac{1}{5}$ . The equation is



Figure 4: Visualization of parts of the three-dimensional shell geometry with the variable coefficient (15) on a mesh consisting of 393k elements in total.

solved on two domains,

- the unit cube,  $\Omega = (-1, 1)^3$ , with the surfaces at  $x_e = -1$  subject to Neumann boundary conditions and the surfaces at  $x_e = +1$  subject to Dirichlet boundary conditions, e = 1, 2, 3, using constant diffusivity  $\kappa = 1$ , and
- a full spherical shell in 3D with inner radius 0.5 and outer radius 1.0, using a polynomial approximation of degree 5 along a spherical manifold for all elements, and a strongly varying diffusivity

$$\kappa(\vec{x}) = \kappa(x_1, \dots, x_d) = 1 + 10^6 \prod_{e=1}^3 \cos(2\pi x_e + 0.1e),$$
(15)

inspired by the variable coefficient case in [47] but using a shift 0.1*e* in order to eliminate any potential spatial symmetries in the coefficients. Dirichlet conditions are set on all boundaries. A visualization of approximately one eighth of a sample 3D mesh along with the coefficient is given in Fig. 4. Geometric multigrid methods start with an initial mesh consisting of six elements.

In both cases, the boundary conditions  $g_{\rm D}$  and  $g_{\rm N}$  as well as the value of the forcing f in (1) are set such that the analytic solution (14) is obtained.

Figs. 5 and 6 list the accuracy over the computational time for the constantcoefficient case with Cartesian mesh and the variable-coefficient case with curved mesh and high-order mappings. Results appearing in the lower left corner of these plots combine high accuracy with low computational time. The numbers are from experiments on a full node with 28 cores of Intel Xeon E5-2690 v4 Broadwell CPUs. Continuous finite elements show the best efficiency over the whole range of polynomial degrees. The next best method for  $2 \leq k \leq 4$ , the DG-SIP method, is two to six times less efficient. The best matrix-based schemes is the statically condensed finite element method, slightly ahead of the HDG method with post-processing. Both results are approximately three to five times slower than DG-SIP and around 20 times slower than the matrix-free CG implementation for  $k \geq 3$ . All results are along the optimal convergence rate curves at  $\mathcal{O}(h^{k+1})$  for the primal solution  $u_h$  and  $\mathcal{O}(h^{k+2})$  for the post-processed solution in HDG, including the variable-coefficient curved mesh cases.

Fig. 5 also includes HDG results with polynomials of complete degree up to  $k, \mathcal{P}_k$ , spanned by orthogonal Legendre polynomials, rather than tensor-product space  $\mathcal{Q}_k$ . This space skips the higher order mixed terms such as xy, xz, yz, xyz in  $\mathcal{Q}_1$  elements and thus tightly selects polynomials exactly up to degree k. The  $\mathcal{P}_k$  basis reduces both the number of unknowns by going from  $(k + 1)^2$  polynomials per face to (k + 1)(k + 2)/2 and also the nonzero entries per row,



Figure 5: Accuracy over solver time for 3D Laplacian on the unit cube.

reducing the cost per element by up to a factor of four. The computational results in Fig. 5 show that the decrease in solver times for the tight polynomial space comes with a decrease in solution accuracy: Even though the solution still converges optimally at rate k + 1, the error constants are higher and more elements are needed for reaching the same accuracy. Thus, no savings can be achieved this way. This observation is in line with results e.g. in [37, 51] comparing tetrahedral to hexahedral element shapes, where a similar or slightly better efficiency per degree of freedom of hexahedral elements was demonstrated in the context of matrix-based HDG.

Table 1 lists the number of iterations to reduce the linear residual by  $10^{-9}$  with the preconditioned conjugate gradient method as well as solver throughput. Up to 16 million DoFs per second can be processed with the matrix-free continuous finite element implementation on 28 cores on Cartesian meshes, and 7.75 million unknowns per second on a curved mesh. Throughout  $2 \le k \le 6$ , the throughput is above 13 million unknowns per second on the Cartesian mesh and 6 million unknowns per second on the curved mesh. For comparison, we measured 6.86 million DoFs with HPGMG and polynomial degree 2 on the same system, again slower than our implementation. This is despite a coarser iteration tolerance with fewer iterations of HPGMG that relies on a full multigrid cycle rather than a V-cycle that further reduces the number of operations on the finest level [2]. As reported in [47], there is a slight increase in iteration counts as the polynomial degree k increases, but high-order methods appear highly attractive nonetheless.

Table 1 confirms that DG-SIP provides the second highest solver throughput. Note that the cost per degree of freedom is almost independent of the polynomial degree for the matrix-free multigrid solvers, confirming the results of matrix-vector products in Sec. 3. The factor between the fastest realization (at degree between two and four) and the slowest one is only approximately two. For the matrix-based HDG and statically condensed CG methods, we notice a distinct decrease in throughput as the polynomial degree increases. This is a



Figure 6: Accuracy over solver time for 3D Laplacian on the sphere with highorder curved boundaries and variable coefficients according to Eq. (15).

direct consequence of the fact that the matrix rows are more densely populated for higher degrees, with the cost being directly proportional to this number. Even though the number of unknowns goes down with static condensation as compared to matrix-free evaluation, Fig. 1 shows that this reduction is not nearly enough for competitive performance. In other words, the gap between the matrix-free implementations and the matrix-based schemes widens as the degree increases.

Note that the smaller iteration counts for the geometric multigrid approaches are due to a more expensive smoother that involves five matrix-vector products rather than only one forward and backward substitution in the ILU of HDG. The HDG solver with *p*-AMG is highly competitive in terms of the total number of operator evaluations to reach the prescribed tolerance of  $10^{-9}$ . For example, the HDG solver involves 45 operator evaluations and 30 ILU applications on the finest level in the Cartesian case with k = 2 of Table 1 that lists 15 iterations, as compared to 65 matrix-vector products for the CG solver with GMG preconditioning at 5 iterations and 156 matrix-vector products for DG-SIP at 12 iterations. It is rather the different performance of matrix-vector products that favors the matrix-free schemes.

Fig. 7 lists the computational time required to reach a fixed relative accuracy of  $10^{-6}$  for various polynomial degrees on a Cartesian mesh with 28 Broadwell cores, both in two and three space dimensions. Note that the 3D numbers for k = 1 need significantly more memory and computational resources than what is available the (fat-memory) single node with 512 GB used for the present tests, requiring  $3.7 \cdot 10^{11}$  degrees of freedom for continuous elements or  $4.4 \cdot 10^{12}$  degrees of freedom in the trace system. For reasons of comparison, extrapolations of the computational time recorded at around  $10^8$  degrees of freedom to accuracy  $10^{-6}$  have been used under the justified assumption of optimal iteration counts and convergence rates. The efficiency dramatically increases as the polynomial degree is risen, enabling the solution to a tolerance of  $10^{-6}$  in less than 10

Table 1: Number of iterations for matrix sizes close to ten million degrees of freedom as well as absolute performance in terms of degrees of freedom solved per second on 28 Broadwell cores. Conjugate gradient tolerances:  $10^{-9}$ .

	CG mat-free		CG stat cond		DG-SIP mat-free		HDG trace matrix	
$_{k}$	its	$\mathrm{DoFs/s}$	its	$\mathrm{DoFs/s}$	its	$\mathrm{DoFs/s}$	its	DoFs/s
3D Cartesian mesh, constant coefficients								
1	4	$7.89\cdot 10^6$	11	$3.94\cdot 10^6$	14	$3.07 \cdot 10^{6}$	16	$2.08 \cdot 10^{6}$
2	5	$1.35 \cdot 10^{7}$	21	$1.17\cdot 10^6$	12	$5.17 \cdot 10^{6}$	15	$1.14 \cdot 10^{6}$
3	5	$1.45\cdot 10^7$	22	$6.17\cdot 10^5$	10	$6.42\cdot 10^6$	17	$6.15\cdot 10^5$
4	5	$1.58 \cdot 10^{7}$	25	$3.46\cdot 10^5$	10	$6.33 \cdot 10^6$	17	$4.11 \cdot 10^5$
5	5	$1.52\cdot 10^7$	26	$2.29\cdot 10^5$	11	$5.03 \cdot 10^6$	20	$2.48\cdot 10^5$
6	5	$1.43\cdot 10^7$	27	$1.98\cdot 10^6$	11	$5.06\cdot 10^6$	20	$1.87\cdot 10^5$
7	5	$1.25\cdot 10^7$	26	$1.30\cdot 10^5$	13	$4.04\cdot 10^6$	21	$1.40\cdot 10^5$
8	5	$1.17\cdot 10^7$	27	$9.90\cdot 10^4$	12	$3.64 \cdot 10^6$	21	$1.13 \cdot 10^5$
3D curved mesh, variable coefficients								
1	5	$3.13 \cdot 10^{6}$	14	$3.38 \cdot 10^{6}$	13	$2.51 \cdot 10^{6}$	217	$1.56 \cdot 10^{5}$
2	5	$7.75\cdot 10^6$	23	$1.06\cdot 10^6$	11	$3.36\cdot 10^6$	32	$5.43\cdot10^5$
3	6	$6.51\cdot 10^6$	22	$6.40\cdot10^5$	10	$4.15\cdot 10^6$	46	$2.31\cdot 10^5$
4	5	$7.38\cdot 10^6$	24	$3.68\cdot 10^5$	12	$3.51 \cdot 10^{6}$	30	$2.32 \cdot 10^5$
5	7	$6.88\cdot 10^6$	23	$2.65\cdot 10^5$	13	$3.12\cdot 10^6$	27	$1.82\cdot 10^5$
6	8	$6.08\cdot 10^6$	24	$1.85\cdot 10^5$	15	$2.65\cdot 10^6$	24	$1.57\cdot 10^5$
7	10	$4.90\cdot 10^6$	25	$1.35\cdot 10^5$	17	$2.35 \cdot 10^6$	27	$1.09\cdot 10^5$
8	11	$4.47 \cdot 10^{6}$	25	$1.12 \cdot 10^{5}$	21	$1.95 \cdot 10^{6}$	22	$1.14 \cdot 10^{5}$



Figure 7: Time to reach a discretization accuracy of  $10^{-6}$  as a function of the polynomial degree.

seconds for degree k = 3 with the matrix-free CG method and the post-processed HDG method at k = 3 in three dimensions, or 0.01 seconds in 2D.

When increasing the polynomial degree further, different saturation points appear in 3D. Matrix-based schemes mainly suffer from the aforementioned increase of nonzero entries per row, despite the number of DoFs still going down, a behavior also described in [19]. The faster matrix-free schemes with approximately constant timings per unknown run into latency issues instead, including the coarse grid solver. Furthermore, the granularity of the mesh sizes that are tested in the pattern  $4^3$ ,  $8^3$ ,  $12^3$ ,  $16^3$ ,  $20^3$ ,  $24^3$ ,  $28^3$ ,  $32^3$ ,  $40^3$ ,  $48^3$  (and continuing with multiples of 4, 5, 6, 7 times a power of two) result in selecting the mesh  $12^3$  for all of k = 6, 7, 8, whereas the next coarser size  $8^3$  is too coarse unless  $k \ge 9$ .

### 4.4 Three-dimensional example with non-smooth solution

We now consider the Laplacian on a cube  $(-1, 1)^3$  with a slit along the plane  $\{x = 0, y < 0, -1 < z < 1\}$ . The solution is given by

$$u(x, y, z) = r^{\frac{1}{2}} + \sin(0.5\phi)$$
 with  $r = \sqrt{x^2 + y^2}, \phi = \arctan\left(\frac{y}{x}\right) + \pi,$ 



Figure 8: Singular solution. Time to reach a discretization accuracy of  $10^{-4}$  on the uniform mesh and  $10^{-5}$  on the adaptive mesh as a function of the polynomial degree on 28 cores.

and constant in z-direction. Fig. 8 shows the solution around the singularity on a slice at z = 0 with elevation along the function value. Due to the singularity, convergence rates in the  $L_2$  norm are only linear in the mesh size, irrespective the polynomial degree. Fig. 8 lists the time to reach a discretization accuracy of  $10^{-4}$  on uniform meshes and  $10^{-5}$  with adaptive meshes which are created by successively refining the 15% of the elements with the largest jump in the gradient over element boundaries as a simple error estimator [17]. The results confirm the considerably higher efficiency of the matrix-free schemes both in the uniform and adaptive mesh case. Furthermore, the methods become more efficient as the polynomial degree is increased also on the adaptive mesh, as opposed to the matrix-based methods that level off and high order methods do not pay off. Also note that DG-SIP is more efficient than continuous elements on the uniform mesh due to better solution accuracy around the singularity, allowing for coarser meshes.

### 4.5 Scalability in the massively parallel context

The parallel behavior of the solvers is shown in Fig. 9 by a strong scaling experiment and in Fig. 10 by a weak scaling experiment. All codes have been parallelized with pure MPI according to the techniques described in [4, 11, 36]. Two large-scale parallel systems have been used, SuperMUC Phase 1 consisting of up to 9216 nodes with  $2 \times 8$  cores (Intel Xeon E5-2680 Sandy Bridge, 2.7 GHz), and SuperMUC Phase 2 consisting of up to 512 nodes with  $2 \times 14$  cores (Intel Xeon E5-2697 v3 Haswell, 2.6 GHz). In Fig. 9, we observe ideal strong scaling of the geometric multigrid solvers until a lower threshold of approximately 0.05–0.1 seconds where communication latency becomes dominant. We note that the scaling in the GMG solvers for CG and DG-SIP saturates below  $30\,000$  degrees of freedom per core in both panels of Fig. 9. Also note the wide range of problem sizes with almost two orders of magnitude going from saturated scaling to the size that still fits into approximately 2 GB RAM memory per core, much more than for the matrix-based realization. On the other hand, the HDG solver is already saturated at around 0.5 seconds. This breakdown is due to the non-ideal behavior of the ML-AMG part also reported in [46]. The scaling of the HDG solver is relatively good until matrix sizes go below 5000 rows per core. Note that on a smaller  $64^3$  mesh, nearly linear scaling down to approximately 0.2s has been obtained.

The weak scaling plots in Fig. 10 display the time to solve a linear system



Figure 9: Strong scaling experiment on SuperMUC Phase 1 (SandyB) and SuperMUC Phase 2 (Haswell) on up to 147 456 cores.

with one million degrees of freedom per core for the three chosen discretizations as the number of processors and the problem size increase at the same rate. Between the smallest and largest configuration in the weak scaling tests, parallel efficiencies of 75%, 78%, 75%, and 87% have been measured for the CG method on SuperMUC Phase 1 and Phase 2 as well as DG-SIP on SuperMUC Phase 1 and Phase 2, respectively. For the HDG linear solvers, the weak scaling is somewhat worse, reaching 40% when going from 28 to 14 336 Haswell cores of SuperMUC Phase 2. Approximately half of the decrease in efficiency is due to the increase in solver iterations from 18 to 29, and the other half is due to inefficiencies in the AMG hierarchy.

The largest computation on 147 456 cores achieved an arithmetic throughput of about 1.4 PFLOP/s for DG-SIP, out of a theoretical peak of 3.2 PFLOP/s on SuperMUC Phase 1. To put the obtained results into perspective, we compare the CG solution time of 1.52s on 288 cores and 2.04s on 147 456 cores to the numbers from [19] which were obtained on the Stampede system with the same Intel Xeon E5-2680 Sandy Bridge processors, outperforming both the 2.5s for HPGMG at  $Q_2$  elements and more than 20s for GMG implementation of the authors from [19] on fourth degree elements. In addition, we could solve a DG-SIP system with one million degrees of freedom per core in 1.92s on SuperMUC Phase 2. Note that the computational time per core on the Sandy Bridge and Haswell systems, respectively, is similar for the continuous elements (limited mainly by indirect addressing into vectors), whereas there is a considerably advantage of the Haswell system with fused multiply-add instruction and faster L1 cache access for DG-SIP.

# 5 Conclusions

In this study, a performance comparison between continuous and discontinuous Galerkin methods has been presented. The work has concentrated on state-of-the-art multigrid solvers for the Laplacian as the prototype elliptic equation. As opposed to previous studies that focused on direct solvers, our experiments show that the primal formulation in terms of continuous finite elements or discontinuous Galerkin symmetric interior penalty methods allows for up to an order of magnitude more efficient solution than the HDG method in 3D, also when including superconvergence of HDG. In two space dimensions, the performance gap is approximately a factor of two to five times for polynomial degrees  $2 \le k \le 5$ . When comparing the continuous finite element implementation



Figure 10: Weak scaling experiment on SuperMUC Phase 1 (SandyB) and SuperMUC Phase 2 (Haswell) using a grain size of 1 million unknowns per core and  $Q_3$  elements in 3D.

against the interior penalty discontinuous Galerkin method, a performance advantage of a factor of two to three for the continuous case has been recorded.

Our results are due to the beneficial properties of modern sum factorization implementations on quadrilateral and hexahedral meshes. Depending on the structure of the equations and the behavior of the solution, either continuous finite elements or symmetric interior penalty discontinuous Galerkin methods are the preferred choice. We found that the time to solution per degree of freedom is almost constant for polynomial degrees between two and eight at a similar rate as for the HPGMG benchmark, and better than matrix-based multigrid schemes on linear finite elements. Thus, the polynomial degree can be chosen as high as the meshing of the geometry allows for without compromising throughput. The promising results in this study motivate to pursue developments of matrix-free solvers in non-elliptic contexts, where the Jacobi-related techniques used here are not sufficient and matrix-based methods use Gauss–Seidel or ILU smoothers.

Our conclusions go against the results of previous efficiency studies and are mainly explained by the different performance of matrix-vector products. In particular, counting degrees of freedom or arithmetic operations is not enough to judge application performance. The higher performance is due to a reduced memory transfer as compared to memory-limited sparse matrix kernels. A roofline performance model has been developed which shows that our results are close to the performance limits of the underlying hardware, making the conclusions general without bias towards any of the methods. Moreover, the performance model allows for predicting performance on other HPC systems with different machine balances. Note that reaching the high performance numbers recorded for the matrix-free solvers needs careful implementation that is only realistic for large finite element libraries that can distribute the development burden over many applications. However, we expect the developments from [2, 10, 36] to become mainstream library components of finite element codes in the future, similarly to high-performance dense linear algebra implementations of BLAS and LAPACK. In order to exploit the fast matrix-free approaches also in the context of HDG, alternative evaluation schemes have been presented in this work. The HDG mixed system involving the primal variable and the flux is clearly faster than the sparse matrix-vector product with the trace matrix, despite considerably fewer degrees of freedom in the latter. Given appropriate solvers, these approaches promise best HDG performance.

# Acknowledgments

The authors would like to thank Katharina Kormann and Niklas Fehn for discussions about the manuscript and Timo Heister and Guido Kanschat on the multigrid implementation in deal.II.

# References

- M. ADAMS, M. BREZINA, J. HU, AND R. TUMINARO, Parallel multigrid smoothing: polynomial versus Gauss-Seidel, J. Comput. Phys., 188 (2003), pp. 593-610, doi:10.1016/S0021-9991(03)00194-3.
- [2] M. ADAMS, J. BROWN, J. SHALF, B. VAN STRAALEN, E. STROHMAIER, AND S. WILLIAMS, *High-performance geometric multigrid*, 2016, https://hpgmg.org.
- [3] D. 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, doi:10.1137/S0036142901384162.
- [4] W. BANGERTH, C. BURSTEDDE, T. HEISTER, AND M. KRONBICHLER, Algorithms and data structures for massively parallel generic finite element codes, ACM Trans. Math. Softw., 38 (2011), doi:10.1145/2049673.2049678.
- [5] W. BANGERTH, D. DAVYDOV, T. HEISTER, L. HELTAI, G. KANSCHAT, M. KRONBICHLER, M. MAIER, B. TURCKSIN, AND D. WELLS, *The deal.II library, version 8.4*, J. Numer. Math., 24 (2016), pp. 135–141, doi:10.1515/jnma-2016-1045, www.dealii.org.
- [6] P. BASTIAN, C. ENGWER, D. GÖDDEKE, O. ILIEV, O. IPPISCH, M. OHLBERGER, S. TUREK, J. FAHLKE, S. KAULMANN, S. MÜTHING, AND D. RIBBROCK, *EXA-DUNE: Flexible PDE Solvers, Numerical Meth*ods and Applications, Springer International Publishing, Cham, 2014, pp. 530–541, doi:10.1007/978-3-319-14313-2\_45.
- [7] M. BENZI, G. H. GOLUB, AND J. LIESEN, Numerical solution of saddle point problems, Acta Numerica, 14 (2005), pp. 1–137, doi:10.1017/S096249290400.
- [8] B. BERGEN, T. GRADL, U. RÜDE, AND F. HÜLSEMANN, A massively parallel multigrid method for finite elements, Comput. Sci. Eng., 8 (2006), pp. 56–62, doi:10.1109/MCSE.2006.102.
- S. C. BRENNER AND L. R. SCOTT, The mathematical theory of finite element methods, Springer-Verlag, New York, 3rd ed., 2008, doi:10.1007/978-0-387-75934-0.
- [10] J. BROWN, Efficient nonlinear solvers for nodal high-order finite elements in 3D, J. Sci. Comput., 45 (2010), pp. 48–63, doi:10.1007/s10915-010-9396-8.
- [11] C. BURSTEDDE, L. C. WILCOX, AND O. GHATTAS, p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees, SIAM J. Sci. Comput., 33 (2011), pp. 1103–1133, doi:10.1137/100791634.
- [12] B. COCKBURN, O. DUBOIS, J. GOPALAKRISHNAN, AND S. TAN, Multigrid for an HDG method, IMA J. Numer. Anal., 34 (2014), pp. 1386–1425, doi:10.1093/imanum/drt024.

- [13] B. COCKBURN, J. GOPALAKRISHNAN, AND R. LAZAROV, Unified hybridization of discontinuous Galerkin, mixed, and continuous Galerkin methods for second order elliptic equations, SIAM J. Numer. Anal., 47 (2009), pp. 1139–1365, doi:10.1137/070706616.
- [14] B. COCKBURN, J. GUZMÁN, AND H. WANG, Superconvergent discontinuous Galerkin methods for second-order elliptic problems, Math. Comput., 78 (2009), pp. 1–24, doi:10.1090/S0025-5718-08-02146-7.
- [15] H. ELMAN, D. SILVESTER, AND A. WATHEN, Finite Elements and Fast Iterative Solvers with Applications in Incompressible Fluid Dynamics, Oxford Science Publications, Oxford, 2005.
- [16] P. F. FISCHER, J. W. LOTTES, AND S. KERKEMEIER, Nek5000 Web page, 2015, https://nek5000.mcs.anl.gov.
- [17] J. P. D. S. R. GAGO, D. W. KELLY, O. C. ZIENKIEWICZ, AND I. BABUŠKA, A posteriori error analysis and adaptive processes in the finite element method: Part II — Adaptive mesh refinement, Int. J. Num. Meth. Engrg., 19 (1983), pp. 1621–1656.
- [18] M. W. GEE, C. M. SIEFERT, J. J. HU, R. S. TUMINARO, AND M. G. SALA, *ML 5.0 Smoothed Aggregation User's Guide*, Tech. Report 2006-2649, Sandia National Laboratories, 2006.
- [19] A. GHOLAMI, D. MALHOTRA, H. SUNDAR, AND G. BIROS, FFT, FMM, or multigrid? A comparative study of state-of-the-art Poisson solvers for uniform and nonuniform grids in the unit cube, SIAM J. Sci. Comput., 38 (2016), pp. C280–C306, doi:10.1137/15M1010798.
- [20] G. GIORGIANI, D. MODESTO, S. FERNÁNDEZ-MÉNDEZ, AND A. HUERTA, *High-order continuous and discontinuous Galerkin methods for wave problems*, Int. J. Numer. Meth. Fluids, 73 (2013), pp. 883–903, doi:10.1002/fld.3828.
- [21] B. GMEINER, U. RDE, H. STENGEL, C. WALUGA, AND B. WOHLMUTH, Performance and scalability of hierarchical hybrid multigrid solvers for stokes systems, SIAM Journal on Scientific Computing, 37 (2015), pp. C143–C168, doi:10.1137/130941353.
- [22] G. HAGER AND G. WELLEIN, Introduction to High Performance Computing for Scientists and Engineers, CRC Press, Boca Raton, 2011.
- [23] M. A. HEROUX, R. A. BARTLETT, V. E. HOWLE, R. J. HOEKSTRA, J. J. HU, T. G. KOLDA, R. B. LEHOUCQ, K. R. LONG, R. P. PAWLOWSKI, E. T. PHIPPS, A. G. SALINGER, H. K. THORNQUIST, R. S. TUMI-NARO, J. M. WILLENBRING, W. A., AND K. S. STANLEY, An overview of the Trilinos project, ACM Trans. Math. Softw., 31 (2005), pp. 397–423, www.trilinos.org.
- [24] J. S. HESTHAVEN AND T. WARBURTON, Nodal discontinuous Galerkin methods: Algorithms, analysis, and applications, vol. 54 of Texts in Applied Mathematics, Springer, 2008, doi:10.1007/978-0-387-72067-8.
- [25] J. J. HEYS, T. A. MANTEUFFEL, S. F. MCCORMICK, AND L. N. OLSON, Algebraic multigrid for high-order finite elements, J. Comput. Phys., 204 (2005), pp. 520–532, doi:10.1016/j.jcp.2004.10.021.
- [26] K. HILLEWAERT, Development of the discontinuous Galerkin method for high-resolution, large scale CFD and acoustics in industrial geometries, PhD thesis, Univ. de Louvain, 2013.

- [27] A. HUERTA, A. ANGELOSKI, X. ROCA, AND J. PERAIRE, Efficiency of high-order elements for continuous and discontinuous Galerkin methods, Int. J. Numer. Meth. Eng., 96 (2013), pp. 529–560, doi:10.1002/nme.4547.
- [28] B. JANSSEN AND G. KANSCHAT, Adaptive multilevel methods with local smoothing for H<sup>1</sup>- and H<sup>curl</sup>-conforming high order finite element methods, SIAM J. Sci. Comput., 33 (2011), pp. 2095–2114, doi:10.1137/090778523.
- [29] G. KANSCHAT, Multi-level methods for discontinuous Galerkin FEM on locally refined meshes, Comput. & Struct., 82 (2004), pp. 2437–2445, doi:10.1016/j.compstruc.2004.04.015.
- [30] G. E. KARNIADAKIS AND S. J. SHERWIN, Spectral/hp element methods for computational fluid dynamics, Oxford University Press, 2nd ed., 2005.
- [31] 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, doi:10.1007/s10915-011-9501-7.
- [32] D. KOMATITSCH AND J. TROMP, Introduction to the spectral element method for three-dimensional seismic wave propagation, Geophys. J. Int., 139 (1999), pp. 806–822, doi:10.1046/j.1365-246x.1999.00967.x.
- [33] D. KOPRIVA, Implementing spectral methods for partial differential equations, Springer, Berlin, 2009.
- [34] K. KORMANN AND M. KRONBICHLER, Parallel finite element operator application: Graph partitioning and coloring, in Proc. 7th IEEE Int. Conf. eScience, 2011, pp. 332–339, doi:10.1109/eScience.2011.53.
- [35] M. KRONBICHLER, A. DIAGNE, AND H. HOLMGREN, A fast massively parallel two-phase flow solver for the simulation of microfluidic chips, Int. J. High Perf. Comput. Appl., in press (2016), doi:10.1177/1094342016671790.
- [36] M. KRONBICHLER AND K. KORMANN, A generic interface for parallel finite element operator application, Comput. Fluids, 63 (2012), pp. 135– 147, doi:10.1016/j.compfluid.2012.04.012.
- [37] M. KRONBICHLER, S. SCHOEDER, C. MÜLLER, AND W. A. WALL, Comparison of implicit and explicit hybridizable discontinuous Galerkin methods for the acoustic wave equation, Int. J. Numer. Meth. Eng., 106 (2016), pp. 712–739, doi:10.1002/nme.5137.
- [38] R. LÖHNER, Improved error and work estimates for high-order elements, Int. J. Numer. Meth. Fluids, 72 (2013), pp. 1207–1218, doi:10.1002/fld.3783.
- [39] H. LUO, J. D. BAUM, AND R. LÖHNER, A p-multigrid discontinuous Galerkin method for the Euler equations on unstructured grids, J. Comput. Phys., 211 (2006), pp. 767–783, doi:10.1016/j.jcp.2005.06.019.
- [40] J. D. MCCALPIN, STREAM: Sustainable memory bandwidth in high performance computers, 1991–2007. A continually updated technical report. http://www.cs.virginia.edu/stream.
- [41] N. C. NGUYEN, J. PERAIRE, AND B. COCKBURN, An implicit highorder hybridizable discontinuous Galkerin method for linear convectiondiffusion equations, J. Comput. Phys., 228 (2009), pp. 3232–3254, doi:10.1016/j.jcp.2009.01.030.
- [42] N. C. NGUYEN, J. PERAIRE, AND B. COCKBURN, A class of embedded discontinuous Galkerin methods for computational fluid dynamics, J. Comput. Phys., 302 (2015), pp. 674–692, doi:10.1016/j.jcp.2015.09.024.

- [43] S. A. ORSZAG, Spectral methods for problems in complex geometries, J. Comput. Phys., 37 (1980), pp. 70–92, doi:10.1016/0021-9991(80)90005-4.
- [44] D. A. PATTERSON AND J. L. HENNESSY, Computer Organization and Design, Morgan Kaufmann, Burlington, 4th ed., 2009.
- [45] J. SCHÖBERL, C++11 implementation of finite elements in NGSolve, Tech. Report ASC Report No. 30/2014, Vienna University of Technology, 2014.
- [46] H. SUNDAR, G. BIROS, C. BURSTEDDE, J. RUDI, O. GHATTAS, AND G. STADLER, *Parallel geometric-algebraic multigrid on unstructured forests* of octrees, in SC12: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2012, doi:10.1109/SC.2012.91.
- [47] H. SUNDAR, G. STADLER, AND G. BIROS, Comparison of multigrid algorithms for high-order continuous finite element discretizations, Numer. Linear Algebra Appl., 22 (2015), pp. 664–680, doi:10.1002/nla.1979.
- [48] U. TROTTENBERG, C. OOSTERLEE, AND A. SCHÜLLER, *Multigrid*, Elsevier Academic Press, London, 2001.
- [49] R. S. VARGA, Matrix iterative analysis, Springer, Berlin, 2nd ed., 2009.
- [50] P. E. J. 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 discretizations, J. Comput. Phys., 229 (2010), pp. 5161–5181, doi:10.1016/j.jcp.2010.03.031.
- [51] 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, doi:10.1007/s10915-015-0076-6.