PRECONDITIONING MATRIX-FREE HIGH-ORDER FINITE ELEMENT OPERATORS

JEREMY L. THOMPSON*

Abstract. Global sparse matrices are no longer a good representation of high-order finite element operators. Matrix-free operators offer superior performance, both with respect to FLOPs needed for evaluation and the memory transfer needed for a matrix-vector product. However, matrix-free methods require iterative solvers, which are sensitive to the condition number of the operator. Matrix-free operators with tensor product bases have the best efficiency at high-order, but the ill-conditioning slows iterative solver convergence. Preconditioners can accelerate convergence of these solvers with high-order operators. We discuss preconditioners that can be efficiently derived from matrix-free operators or formulated as matrix-free operators themselves, to include diagonal or block diagonal based techniques such as Jacobi and Chebyshev, multi-level techniques such a p-multigrid, and domain decomposition techniques such as Additive Schwartz and BDDC.

1. Introduction. High-order finite element methods offer advantages over low-order finite elements; *hp* finite elements offer high accuracy and exponential convergence [6], [28], [30]. However, high-order finite elements are less common because the operator or its Jacobian rapidly loses sparsity as the order is increased. Matrix-free implementation of high-order finite elements can provide the benefits of high-order methods without relying upon matrix sparsity for efficient implementation.

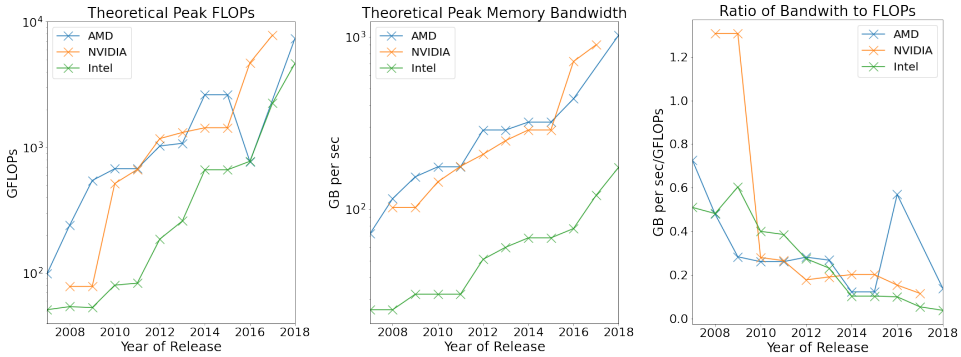
High Performance Computing (HPC) hardware has seen improvements in Floating Point Operations per second (FLOPs) that outstrip improvements in memory and network bandwidth, as highlighted in McCalpin's invited talk an Supercomputing 2016 [26]. These developments make memory and network bandwith a key limiting factor in the performance of HPC code. Under these hardware constraints, global sparse matrices are no longer a good representation of a high-order finite element operators. Matrix-free operators offer superior performance, both with respect to FLOPs needed for evaluation and the memory transfer needed for a matrix-vector product. However, matrix-free methods require iterative solvers, which are sensitive to the high condition numbers of high-order operator.

In Section 2, we discuss the specific hardware limitations that constrain the performance of HPC application codes. In Section 3, we provide notation to describe arbitrary PDEs for matrix-free implementation and discuss the performance matrix-free implementation compared to assembled matrices for high-order finite elements on 3D hexahedral meshes. In Section 4, we discuss preconditioning techniques for high-order finite elements and highlight areas where future work can bring these techniques to new applications or improve the performance of these preconditioners.

2. Hardware Limitations. Two key performance metrics for HPC hardware are FLOPs and memory and network bandwidth. FLOPs is the more widely popularized of these two metrics; the Top 500 [27] list tracks the 500 supercomputers with the highest peak FLOPs, as measured by High-Performance Linpack (HPL) [29]. HPL measures the performance when solving random dense linear systems in double precision via LU factorization and measures maximum achievable FLOPs.

Other benchmarks, such as High-Performance Geometric Multigrid (HPGMG) [2] and High-Performance Conjugate Gradient (HPCG) [8], measure performance based upon solving a more complex benchmark problem. The disparity between the FLOPs

*Department of Applied Mathematics, University of Colorado Boulder, Boulder, CO (jeremy.thompson@colorado.edu



(a) Maximum FLOPs (b) Maximum Bandwidth (c) Ratio

Fig. 1: System Balance

achieved in benchmarks such as HPGMG and HPCG and the peak FLOPs measured by HPL is partially explained by the growing gap between FLOPs and memory and network bandwidth.

Over the last thirty years, the peak FLOPs for new HPC hardware has been increasing more rapidly than memory bandwidth and network bandwidth, for both CPUs and GPUs. As discussed in McCalpin's Supercomputing 2016 invited talk [26], peak FLOPs per socket have been increasing at a rate of 50-60% per year while memory bandwidth has only been increasing at a rate of approximately 23% per year and network bandwidth has only been increasing at a rate of approximately 20% per year. FLOPs have improved twice as much as memory and network bandwidth. This problem is exacerbated by network latency, which is decreasing at a rate of approximately 20% per year, and memory latency, which is *increasing* at a rate of approximately 20%per year.

Using data from [33], we can see in Figure 1 that AMD, NVIDIA, and Intel top of the line hardware has a steady decrease in the maximum memory bandwidth compared to FLOPs over the last 13 years. HPC applications need to be careful to control the memory bandwidth required for their codes to better realize the FLOPs capabilities of HPC hardware.

3. Matrix-Free Finite Elements. High-order finite elements implemented in a matrix-free fashion are one way to address the common performance bottlenecks for modern HPC hardware. In this section we develop notation to describe high-order matrix-free finite elements on unstructured meshes. This notation is applicable to both linear and non-linear PDEs. Furthermore, we compare the performance of matrix-free implementations and assembled matrices.

3.1. Notation. The development of this notation will largely follow [3].

Let $\{X_i\}_{i=1}^P$ denote the Legendre-Gauss-Lobatto (LGL) nodes of degree $P-1$ on the reference interval $[-1,1]$ while $\{q_i\}_{i=1}^Q$ and $\{w_i\}_{i=1}^Q$ denote the quadrature points and quadrature weights corresponding to a Q point quadrature rule. If we consider Lagrange basis functions $\{\phi_i\}_{i=1}^P$, we can construct matrices $B_{ij} = \phi_j(q_i)$, $D_{ij} = \partial_x \phi_j(q_i)$, and $W_{ij} = w_i \delta_{ij}$, representing interpolation to the quadrature points, computation of derivatives at the quadrature points, and quadrature weights, respec-

tively.

We can define the corresponding matrices for 3D problems via tensor products

$$\begin{aligned}\mathbf{B} &= B \otimes B \otimes B \\ \mathbf{D}_0 &= D \otimes B \otimes B \\ \mathbf{D}_1 &= B \otimes D \otimes B \\ \mathbf{D}_2 &= B \otimes B \otimes D \\ \mathbf{W} &= W \otimes W \otimes W.\end{aligned} \tag{3.1}$$

The basis operations 3.1 are defined on a reference element $\hat{K} = [-1,1]^3$. In the finite element and spectral element methods, we partition the domain Ω into a set of E elements, denoted $\{K^e\}_{e=1}^E$ with coordinate mapping to the reference element given by $X : \hat{K} \to K^e$. The Jacobian of this mapping is given by $J_{ij} = \partial x_i / \partial X_j$, where X is the reference coordinates and x the physical coordinates. We can invert the Jacobian and compute the derivatives of the physical coordinates in the reference space at every quadrature point.

$$\mathbf{D}_i^e = \Lambda\left(\frac{\partial X_0}{\partial x_i}\right)\mathbf{D}_0 + \Lambda\left(\frac{\partial X_1}{\partial x_i}\right)\mathbf{D}_1 + \Lambda\left(\frac{\partial X_2}{\partial x_i}\right)\mathbf{D}_2 \tag{3.2}$$

where $\Lambda(X)_{ij} = X_i \delta_{ij}$ expresses pointwise multiplication of J_{ij}^{-1} at quadrature points as a diagonal matrix. With this coordinate mapping, element integration weights become $\mathbf{W}^e = W\Lambda(|J^e(q)|)$.

When using an assembled matrix to represent a finite element operator, a global assembly operator is defined as $\mathcal{E} = [\mathcal{E}^e]$, where $\mathcal{E}^e$ represents local restriction operators extracting degrees of freedom that correspond to element e from the global solution vector. Notice that these local restriction operators do not assume a structured mesh, a conforming mesh, or consistent polynomial order bases for each element.

With these definitions, we can represent the Galerkin system of equations corresponding to the weak form of arbitrary second order PDEs. The weak form of PDEs is linear in test functions and can be expressed as pointwise operations where functions of u and ∇u are contracted with v and ∇v.

Given the weak form of an arbitrary PDE

$$\begin{gathered}\text{find } u \in V \text{ such that for all } v \in V \\ \langle v, u\rangle = \int_\Omega v \cdot f_0(u, \nabla u) + \nabla v : f_1(u, \nabla u) = 0\end{gathered} \tag{3.3}$$

where $\cdot$ represents contraction over fields and $:$ represents contraction over fields and spatial dimensions. The corresponding Galerkin system of equations is

$$\sum_e \mathcal{E}\left[(\mathbf{B}^e)^T \mathbf{W}^e \Lambda\left(f_0\left(u^e, \nabla u^e\right)\right) + \sum_{i=0}^{d-1}\left(\mathbf{D}_i^e\right)^T \Lambda\left(f_1\left(u^e, \nabla u^e\right)\right)\right] = 0 \tag{3.4}$$

where $u^e = \mathbf{B}^e \mathcal{E}^e u$ and $\nabla u^e = \{\mathbf{D}_i^e \mathcal{E}^e u\}_{i=0}^{d-1}$.

In this formulation, the element restriction operators and basis operators can represent different element geometries and different degree polynomial bases, providing a flexible description for arbitrary meshes. The pointwise representation of the weak

form given by f_0 and f_1 does not depend upon geometry or polynomial degree of the bases and is the same for all elements. Furthermore, this notation can be extended to handle separate fields with different bases, such as with mixed finite element methods.

Dirichlet boundary conditions are represented in the element restriction operation by enforcing the specified values on the constrained nodes. Neumann or Robin boundary conditions are represented by adding boundary integral terms in the same form as 3.4 with appropriate basis and element restriction operators. Boundary integrals internal to the domain Ω, such as face integrals in Discontinuous Galerkin methods, can also be represented using additional terms with corresponding bases and element restrictions.

3.2. Linearization. When the PDE 3.3 is linear, the pointwise functions f_0 and f_1 are also linear and Krylov subspace methods can be used to solve the Galerkin 3.4. When the PDE is non-linear, the Jacobian of the residual evaluator given in 3.4 can be represented in a similar fashion, based upon the weak form

$$\langle v, J\left(u\right) w\rangle = \int_{\Omega} \left[v^T \nabla v^T\right] \left[\begin{array}{cc} f_{0,0} & f_{0,1} \\ f_{1,0} & f_{1,1} \end{array}\right] \left[\begin{array}{c} w \\ \nabla w \end{array}\right] \tag{3.5}$$

where $f_{i,0} = \frac{\partial f_i}{\partial u}\left(u, \nabla u\right)$ and $f_{i,1} = \frac{\partial f_i}{\partial \nabla u}\left(u, \nabla u\right)$. If these pointwise functions $f_{i,j}$ are not available analytically, they can be computed via algorithmic differentiation or finite differencing. With these pointwise functions, we can describe Jacobian-free Newton-Krylov methods, which are used to solve non-linear PDEs. Jacobian-free Newton-Krylov methods were summarized, with preconditioning strategies, by Knoll and Keyes in [19].

3.3. Performance. To demonstrate the performance benefits of high-order finite elements implemented in a matrix free fashion, we explore the specific case of the Helmholz equations. The strong form of the Helmholz equations is given by

$$\nabla^2 f = -k^2 f \tag{3.6}$$

The corresponding weak form is given by

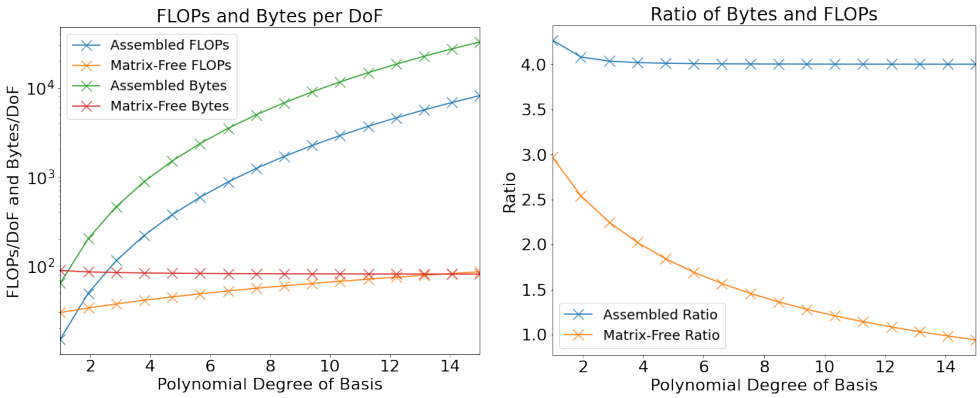
$$\int_{\Omega} \nabla v : \nabla u - k^2 v \cdot u = 0 \tag{3.7}$$

with a Galerkin system of equations

$$\sum_e \mathcal{E} \left[\left(\mathbf{B}^e\right)^T \mathbf{W}^e \Lambda \left(\left(-k^2\right) \mathbf{B}^e \mathcal{E}^e u\right) + \sum_{i=0}^{d-1} \left(\mathbf{D}_i^e\right)^T \mathbf{W}^e \Lambda \left(\left\{\mathbf{D}_i^e \mathcal{E}^e u\right\}_{i=0}^{d-1}\right) \right] = 0 \tag{3.8}$$

The total floating point operations and matrix entries required to apply the matrix-vector product for the operator representing the Galerkin 3.8 depends upon a specific partitioning of the domain Ω into elements. For simplicity, we compare the total floating point operations and matrix entries required to apply the operator representing 3.8 for a single high-order element with arbitrary hexahedral geometry. While fewer operations and matrix entries will be required on a domain decomposed into multiple elements due to the shared nodes on element boundaries, this comparison will adequately illustrate the relative merits of matrix-free implementations for high-order operators.

With a polynomial basis of degree $P-1$, a 3D hexahedral element has P^3 nodes. Therefore, the assembled matrix representing the Galerkin system of equations 3.8



(a) FLOPs and Bytes per DoF

(b) Ratio of Bytes to FLOPs

Fig. 2: Performance per DoF

has P^6 floating point values. Additionally, applying a matrix-vector product requires $\mathcal{O}\left(2P^6 - P^3\right)$ floating point operations.

On the other hand, a matrix-free representation of the Galerkin system of equations 3.8 can exploit tensor contractions to reduce total required operator entries and floating point operations. With a P point quadrature rule, the 1D basis operators, B and D, have $2P^2$ entries, the inverses of the Jacobians of the coordinate mappings requires d^2P^3 entries, and the element quadrature weights require P^3 entries. Including the spatial frequency, $\left(d^2+1\right)P^3 + 2P^2 + 1$ floating point values are required. Using tensor contractions to interpolate the nodal values to the quadrature space requires $\mathcal{O}\left(P^{d+1}\right)$ operations, and computing derivatives at the quadrature points requires an additional $\mathcal{O}\left(P^{d+1} + \left(2d^2-1\right)P^3\right)$ operations. With appropriate precomputation of the geometric factors from the coordinate mapping, the pointwise application of the weak form and transpose basis operators can be applied in an additional $\mathcal{O}\left(2P^{d+1} + (d+1)P^3\right)$ operations. In total, the matrix-vector requires $\mathcal{O}\left(4P^{d+1} + \left(2d^2+d+1\right)P^3\right)$ floating point operations.

For simplicity, we considered an arbitrary P point quadrature rule. The analysis is similar for over-integration, $Q > P$, or under-integration, $Q < P$. If the P LGL points are also used for the quadrature rule, then $2P^3$ floating point operations are required to apply the basis operators, as $B = I$. If the PDE has multiple fields with the same basis, then B and D only need to be stored once for each field that shares this basis.

As shown in Figure 2, the number of values required per node to represent the Galerkin system of equations 3.8 in a matrix-free fashion is constant with respect to polynomial order, while the number of values required for the assembled matrix representing the operator grows exponentially. The number of operations required to compute the matrix-vector product in a matrix-free fashion per node grows only linearly while the number of operations grows cubically for the assembled matrix. This means that the amount of data movement required per degree of freedom decreases for matrix-free implementations while it remains relatively constant for assembled matrices.

We assumed a mesh with hexahedral elements for this analysis. In comparison to generation of simplex meshes, generation of high quality hexahedral meshes is a time intensive process. However, as the formulation 3.4 can handle meshes comprised of different finite element geometries, it is possible to generate meshes comprised predominately of high quality hexahedral elements with initial refinement of a simplex mesh without the costly process of fully converting a simplex mesh into hexahedral elements. Thus, the performance benefits of high-order matrix-free finite elements can be realized without substantial additional effort in generating a mesh exclusively composed of high quality hexahedral elements.

4. Preconditioning. As discussed in Section 3, high-order matrix-free finite elements offer performance benefits in comparison to assembled sparse matrix representations. When using matrix-free formulations, iterative solvers are required to solve the Galerkin system of equations. We are primarily interested in Krylov subspace methods such as Conjugate Gradient, first developed by Hestens and Stiefel [15], and related methods by Lanczos [20] and [21]. Krylov subspace methods are a natural fit for matrix-free finite elements, as these methods only require matrix-vector products to populate the Krylov subspace

$$\mathcal{K}(r_0, A, k) = \{r_0, Ar_0, \ldots, A^{k-1}r_0\} \tag{4.1}$$

which is used to construct increasingly accurate iterates x^k that approach the true solution x.

The iteration count to reach convergence of Krylov subspace methods is based upon condition number of the operator [23] and high-order finite element operators have notoriously poor condition numbers [17]. In this section we discuss preconditioners to control the condition number of high-order finite elements implemented in a matrix-free fashion. With these preconditioners, we can reduce total iteration count and thus total time to solution for these operators.

Suppose we are solving the linear system given by

$$Ax = b \tag{4.2}$$

via a Krylov subspace method. This linear system may come from the Galerkin system of equations of our PDE of interest or the Jacobian of our PDE of interest. We can improve the convergence of our Krylov method for this system by solving the preconditioned system

$$\left(M_L^{-1} A M_R^{-1}\right)(M_R x) = M_L^{-1} b \tag{4.3}$$

via our Krylov method instead.

We will investigate left preconditioning, where $M_R = I$ and $M_L^{-1} \approx A^{-1}$. Therefore, we will adopt the notation $M_L = M$. Many of the preconditioning methods we discuss can be used as iterative solvers. In these cases, a small number of iterations of the preconditioner M are used to improve the Conjugate Gradient iterate x^k.

4.1. Jacobi and Chebyshev. Jacobi iterations for an assembled linear operators produce a new approximate solution x^k via

$$x_i^k = \left(b_i - \sum_{j \neq i} a_{ij} x_j^{k-1}\right) / a_{ii} \tag{4.4}$$

For matrix-free implementations, these values are not directly available, as computation of the full assembled linear operator for preconditioning defeats the benefits of matrix-free methods; however, the true operator diagonal can be efficiently computed.

For Jacobi preconditioning based upon the true diagonal of the operator A, we have $M = \text{diag}\,(A)$ and

$$x_i^k = b_i / a_{ii} \tag{4.5}$$

The more diagonally dominant the linear operator A is, the better M^{-1} approximates the true inverse A^{-1}. For multi-field PDEs, point block diagonal assembly offers a middle ground between diagonal assembly and operator assembly for operators that are insufficiently diagonally dominant for Jacobi preconditioning based upon the true diagonal to be an effective preconditioner.

The Chebyshev semi-iterative method, analyzed extensively by Golub and Varga [14], can be viewed as an improvement of the Jacobi, or related Gauss-Seidel, techniques. In the Chebyshev method, we generate iterates of the form

$$y^k = \omega_k \left(y^{k-1} - y^{k-2} + \gamma r^{k-1} \right) + y^{k-2} \tag{4.6}$$

where $\omega_k = 2\frac{2-\beta-\alpha}{\beta-\alpha}\frac{c_{k-1}(\mu)}{c_k(\mu)}$, $\gamma = 2\ (2-\alpha-\beta)$, $\hat{M} r^{k-1} = b - A y^{k-1}$, and c_i are the Chebyshev polynomials with $\mu = 1 + \frac{1-\beta}{\beta-\alpha}$. The eigenvalues of A are in the range $[\alpha, \beta]$. $\hat{M}$, $y^0 = x^0$ and $y^1 = x^1$ come from another iterative preconditioning technique such as Jacobi.

Efficient Jacobi and Chebyshev implementations are important as smoothers for multigrid preconditioners. As discussed in [1], the Chebyshev semi-iterative method is a particularly good choice as a smoother for multigrid. In multigrid preconditioning, the smoother targets the high-energy components, which corresponds to the larger eigenvalues in the spectrum of the operator. It is easier to provide adequate estimates of the maximal eigenvalue and Chebyshev is not too sensitive to estimation of this parameter.

While Jacobi and Chebyshev techniques are well established, there is some work to be done in providing efficient operator diagonal or block diagonal assembly, especially in the context of matrix-free Jacobians derived numerically or algorithmically.

4.2. P-Multigrid. Multigrid methods are popular multi-level techniques that provide resolution independent convergence rates. p-type multigrid, developed by Ronquist and Patera [32], is a natural choice for high-order finite elements on an unstructured mesh and can be implemented with operators represented as in 3.4. Ronquist and Patera declared p-multigrid *sensibly independent* of number of elements and polynomial degree of the element bases. Multigrid can be used as an independent solver, but we investigate the use of multigrid as a preconditioner for a Krylov method.

There has been work by Heys, Manteuffel, McCormick, and Olson demonstrating the feasibility of algebraic multigrid for high-order finite elements [16]; however, algebraic multigrid requires assembly of the finite element operator, which defeats the benefits of matrix-free implementation. There are examples of using h-multigrid, such as [5]; however p-multigrid offers more flexibility with respect to meshes in comparison to h-multigrid as it does not require aggregation of multiple elements into larger elements.

For p-multigrid with nodal bases, the prolongation operator interpolates to a

higher order basis nodes and is defined by

$$P_{p-1}^{p} = m_p^{-1} \sum_e \mathcal{E}_p^T \mathbf{B}_{p-1}^{p} \mathcal{E}_{p-1} \tag{4.7}$$

where $\mathbf{B}_{p-1}^{p}$ interpolates from a basis of degree $p-1$ to degree p and $m_p = \mathcal{E}_p^T \mathcal{E}_p 1$ counts the multiplicity of shared nodes between elements. The restriction operator is defined as the transpose of the prolongation operator, $R_{p-1}^{p} = \left(P_{p-1}^{p}\right)^T$. These operators can be implemented matrix-free, in the same fashion as 3.4.

With these components we can implement p-multigrid, using the algorithm described by May, Sanan, Rupp, Knepley, and Smith in [25]

Algorithm 4.1 Multigrid Algorithm

1: Compute x^k
2: $x^k \leftarrow x^k + \hat{M}^{-1}\left(b - Ax^k\right)$ ▷ pre-smooth m times
3: $r = R\left(b - Ax^k\right)$ ▷ restrict the residual
4: $A_c e = r$ ▷ Solve on coarse grid (may involve additional levels)
5: $x^k \leftarrow x^k + Pe$ ▷ prolongate error
6: $x^k \leftarrow x^k + \hat{M}^{-1}\left(b - Ax^k\right)$ ▷ post-smooth m times

With even modest order finite elements, such as degree 4, p-multigrid can substantially reduce the size of the global solution vector by a factor of $P^3/8$, which makes assembly of the finite element operator on the coarse grid tractable so that direct solvers such as Algebraic Multigrid can be used to solve the coarse problem.

While the theoretical foundation for p-multigrid implemented in a matrix-free fashion on unstructured meshes has existed for quite some time, there do not appear to be many examples of these techniques being combined for practical problems. We are collaborating on a paper implementing these techniques in the context of Neo-Hookean hyperelasticity at finite strain. This is a new contribution to the solid mechanics community, which typically uses low order finite elements and assembled sparse matrices. Overall, the use of high-order finite elements with p-multigrid is under-explored for solid mechanics problems, especially with high-contrast or nearly incompressible materials.

4.3. Domain Decomposition. Domain decomposition methods are another popular class of preconditioners for high-order finite elements. Fisher, with others, has used overlapping Schwarz for high-order finite elements or spectral elements with fluid dynamics problems, such as in [10] and [11]. As with h-type multigrid, overlapping Schwarz techniques require additional information about the mesh in order to provide overlapping subdomains based upon element boundaries. Also, the number of nodes required for overlapping is significant at high-order. With a modest 2 node overlap, approximately $(P+4)^3$ nodes are required in a subdomain compared to P^3 in an element.

Balancing Domain Decomposition by Constraints (BDDC), first developed by Dohrmann [7], is technique for non-overlapping domain decomposition. In BDDC, the coarse problem is solved on a reduced set of shared nodes between subdomains, using element corners and edges in 3D. BDDC is closely related to Finite Element Tearing and Interconnecting (FETI), developed by Farhat and Roux [9], and subclasses of these two methods can be shown to be the same method [13], [18], and [31].

For sufficiently high-order elements, we can treat each element as a subdomain. We partition the degrees of freedoms into two groups, those on the interface Γ and

those in the interior I. We can therefore formulate the BDDC preconditioner, as seen in [4], as

$$M^{-1} = \left(R_1^T - \mathcal{H} J_D\right) \hat{A}^{-1} \left(R_1 J_D^T \mathcal{H}\right) \tag{4.8}$$

where $\mathcal{H}$ is the direct sum of local operators $\mathcal{H}^{(i)} = -\left(A_{II}^{(i)}\right)^{-1} \left(A_{\Gamma I}^{(i)}\right)^T$ that map the jump over subdomain interfaces J_D to subdomain interiors by solving a local Dirichlet problem and giving zero for other values so

$$\left(J_D^T v\left(x\right)\right)^{(i)} = \sum_{j \in \mathcal{N}_x} \left(\delta_j\left(x\right) v^{(i)}\left(x\right) - \delta_i\left(x\right) v^{(j)}\left(x\right)\right) \forall x \in \Gamma_i \tag{4.9}$$

with $\delta_i\left(x\right) = 1/|\mathcal{N}_x|$ and $\mathcal{N}_x$ is the set of indices of subdomains that have x on their boundary. The function $\delta_i\left(x\right)$ is used to create the scaled injection operator R_1 such that interior values have 1 and interface values have $|\mathcal{N}_x|$ entries each set to $\delta_i\left(x\right)$.

The operator $\widetilde{A}^{-1}$ represents a subdomain solver. This subdomain solver can be a direct method or an inexact solver, as discussed by Li and Widlun in [22]. We are interested in investigating separable approximate inverses based on the Fast Diagonalization Method (FDM) for non-separable problems.

Lynch, Rice, and Thomas introduced FDM in [24]. FDM directly solves separable linear equations based upon tensor products of lower dimension operators. For a single element, the Galerkin system of equations 3.8 for the 3D Helmholz problem 3.6 can be rewritten as

$$A = \sum_{i=0}^{d-1} \mathbf{K}_i - k^2 \mathbf{M} \tag{4.10}$$

where $\mathbf{M} = \mathbf{B}^T \mathbf{W} \mathbf{B}$, $\mathbf{K}_i = \mathbf{D}_i^T \mathbf{W} \mathbf{D}_i$, and so on. In this example, we neglect the terms arising from the coordinate mapping as they can result in a non-separable operator.

These operators $\mathbf{M}$ and $\mathbf{K}$ can be written in terms of the 1D operators, $M = B^T W B$ and $K = D^T W D$, as $\mathbf{M} = M \otimes M \otimes M$, $\mathbf{D}_0 = D \otimes M \otimes M$, and so on. Provided that K is symmetric and M is symmetric and positive definite, we can simultaneously diagonalize K and M, yielding $\mathcal{X}^T M \mathcal{X} = I$ and $\mathcal{X}^T K \mathcal{X} = L$. With these pseudoeigenvalues and pseudoeigenvectors, we can rewrite A as

$$A = \boldsymbol{\mathcal{X}} \left(\sum_{i=0}^{d-1} \mathbf{L} - k^2 \mathbf{I}\right) \boldsymbol{\mathcal{X}}^T \tag{4.11}$$

with inverse

$$A^{-1} = \boldsymbol{\mathcal{X}}^T \left(\sum_{i=0}^{d-1} \mathbf{L}_i - k^2 \mathbf{I}\right)^{-1} \boldsymbol{\mathcal{X}} \tag{4.12}$$

where $\boldsymbol{\mathcal{X}} = \mathcal{X} \otimes \mathcal{X} \otimes \mathcal{X}$, $\mathbf{L}_0 = L \otimes I \otimes I$, and so on. Notice that if we treat $\boldsymbol{\mathcal{X}}$ as a basis operation, these inverses can be described and implemented matrix-free, as in 3.4.

With non-separable operators A, a suitable choice of $\left(\sum_{i=0}^{d-1} \mathbf{L}_i - k^2 I_3\right)$ can produce suitable approximate subdomain solvers. Fisher, Miller, and Tufo demonstrated in [12] that while FDM cannot be used for arbitrarily deformed subdomains, defining

the diagonalization over a parallelepiped with average dimensions in each coordinate direction is adequate for preconditioning. For PDEs with arbitrarily deformed subdomains and non-linear pointwise functions f_0 and f_1, early experiments indicate that a suitable approximate inverse might be constructed by correctly selecting pseudoeigenvalues to use with pseudoeigenvectors from the mass and stiffness matrices given above in 4.10. These approximate inverse techniques may be used with other domain decomposition techniques; however, overlapping techniques may require significant additional computation due to the additional nodes in each subdomain.

4.4. Split Preconditioners. These preconditioners are not appropriate for all PDEs. In these cases, splitting these PDE by fields and applying different preconditioners to different fields can yield an effective preconditioner. In this way, the above preconditioners can be composed to handle a wider range of problems. We are collaborating in the development of solver for Neo-Hookean hyperelasticity at finite strain in the incompressible regime. This solver will split the displacement and discontinuous pressure fields, applying p-multigrid as a preconditioner to the displacement fields and block Jacobi as a preconditioner to the discontinuous pressure field.

5. Conclusion. We discussed the performance benefits of high-order matrix-free finite elements. High-order finite element operators solved with Krylov subspace methods require preconditioning to improve convergence and time to solution.

We highlighted several areas for future improvement in preconditioning for high-order finite element operators implemented in a matrix-free fashion. While Jacobi, block Jacobi, and Chebyshev semi-iterative preconditioning is not new, these techniques are important, on their own or as smoothers for other techniques, and there is work to be done on providing efficient operator diagonal or point block diagonal assembly. p-multigrid with matrix-free prolongation and restriction operators is a natural fit for high-order matrix-free finite elements and has proven effective in solid mechanics problems. BDDC is another attractive technique that can be implemented in a matrix-free fashion, and FDM based matrix-free separable approximate inverses may provide suitable subdomain solvers for BDDC and other domain decomposition techniques with non-linear and non-separable problems.

Acknowledgments. This work is supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering and early testbed platforms, in support of the nation's exascale computing imperative.

REFERENCES

[1] M. Adams, M. Brezina, J. Hu, and R. Tuminaro, *Parallel multigrid smoothing: polynomial versus gauss–seidel*, Journal of Computational Physics, 188 (2003), pp. 593–610.

[2] M. Adams, J. Brown, J. Shalf, B. Straalen, E. Strohmaier, and S. Williams, *Hpgmg 1.0: A benchmark for ranking high performance computing systems*, LBNL Technical Report, (2014).

[3] J. Brown, *Efficient nonlinear solvers for nodal high-order finite elements in 3d*, Journal of Scientific Computing, 45 (2010), pp. 48–63.

[4] J. Brown, Y. He, and S. MacLachlan, *Local fourier analysis of balancing domain decomposition by constraints algorithms*, SIAM Journal on Scientific Computing, 41 (2019), pp. S346–S369.

[5] D. Davydov, J.-P. Pelteret, D. Arndt, and P. Steinmann, *A matrix-free approach for finite-strain hyperelastic problems using geometric multigrid*, arXiv preprint arXiv:1904.13131, (2019).

[6] L. Demkowicz, J. T. Oden, W. Rachowicz, and O. Hardy, *Toward a universal hp adaptive finite element strategy, part 1. constrained approximation and data structure*, Computer Methods in Applied Mechanics and Engineering, 77 (1989), pp. 79–112.

[7] C. R. Dohrmann, *A preconditioner for substructuring based on constrained energy minimization*, SIAM Journal on Scientific Computing, 25 (2003), pp. 246–258.

[8] J. Dongarra, M. A. Heroux, and P. Luszczek, *High-performance conjugate-gradient benchmark*, International Journal of High Performance Computing Applications, 30 (2016), pp. 3–10.

[9] C. Farhat and F.-X. Roux, *A method of finite element tearing and interconnecting and its parallel solution algorithm*, International Journal for Numerical Methods in Engineering, 32 (1991), pp. 1205–1227.

[10] P. F. Fischer, *An overlapping schwarz method for spectral element solution of the incompressible navier-stokes equations*, Journal of Computational Physics, 133 (1997), pp. 84–101.

[11] P. F. Fischer and J. W. Lottes, *Hybrid schwarz-multigrid methods for the spectral element method: Extensions to navier-stokes*, in Domain Decomposition Methods in Science and Engineering, Springer, 2005, pp. 35–49.

[12] P. F. Fischer, H. M. Tufo, and N. Miller, *An overlapping schwarz method for spectral element simulation of three-dimensional incompressible flows*, in Parallel Solution of Partial Differential Equations, Springer, 2000, pp. 159–180.

[13] Y. Fragakis and M. Papadrakakis, *The mosaic of high performance domain decomposition methods for structural mechanics: Formulation, interrelation and numerical efficiency of primal and dual methods*, Computer methods in applied mechanics and engineering, 192 (2003), pp. 3799–3830.

[14] G. H. Golub and R. S. Varga, *Chebyshev semi-iterative methods, successive overrelaxation iterative methods, and second order richardson iterative methods*, (1961).

[15] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems1*, Journal of Research of the National Bureau of Standards, 49 (1952).

[16] J. Heys, T. Manteuffel, S. F. McCormick, and L. Olson, *Algebraic multigrid for higher-order finite elements*, Journal of computational Physics, 204 (2005), pp. 520–532.

[17] N. Hu, X.-Z. Guo, and I. Katz, *Bounds for eigenvalues and condition numbers in the p-version of the finite element method*, Mathematics of computation, 67 (1998), pp. 1423–1450.

[18] A. Klawonn and O. Widlund, *Feti and neumann-neumann iterative substructuring methods: connections and new results*, Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences, 54 (2001), pp. 57–90.

[19] D. A. Knoll and D. E. Keyes, *Jacobian-free newton–krylov methods: a survey of approaches and applications*, Journal of Computational Physics, 193 (2004), pp. 357–397.

[20] C. Lanczos, *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*, United States Governm. Press Office Los Angeles, CA, 1950.

[21] C. Lanczos, *Solution of systems of linear equations by minimized iterations*, J. Res. Nat. Bur. Standards, 49 (1952), pp. 33–53.

[22] J. Li and O. B. Widlund, *On the use of inexact subdomain solvers for bddc algorithms*, Computer Methods in Applied Mechanics and Engineering, 196 (2007), pp. 1415–1428.

[23] D. G. Luenberger, *Introduction to linear and nonlinear programming*, vol. 28, Addison-Wesley Reading, MA, 1973.

[24] R. E. Lynch, J. R. Rice, and D. H. Thomas, *Direct solution of partial difference equations by tensor product methods*, Numerische Mathematik, 6 (1964), pp. 185–199.

[25] D. A. May, P. Sanan, K. Rupp, M. G. Knepley, and B. F. Smith, *Extreme-scale multigrid components within petsc*, in Proceedings of the Platform for Advanced Scientific Computing Conference, 2016, pp. 1–12.

[26] J. D. McCalpin, *Memory bandwidth and system balance in hpc systems*, Invited talk, Supercomputing, (2016).

[27] H. Meuer, E. Strohmaier, J. Dongarra, H. Simon, and M. Meuer, *Top 500 list*, 2020, http://www.top500.org/.

[28] J. T. Oden, L. Demkowicz, W. Rachowicz, and T. Westermann, *Toward a universal hp adaptive finite element strategy, part 2. a posteriori error estimation*, Computer methods in applied mechanics and engineering, 77 (1989), pp. 113–180.

[29] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary, *Hpl-a portable implementation of the high-performance linpack benchmark for distributed-memory computers*, (2004), http:

//www.netlib.org/benchmark/hpl/.

[30] W. Rachowicz, J. T. Oden, and L. Demkowicz, *Toward a universal hp adaptive finite element strategy part 3. design of hp meshes*, Computer Methods in Applied Mechanics and Engineering, 77 (1989), pp. 181–212.

[31] D. J. Rixen, C. Farhat, R. Tezaur, and J. Mandel, *Theoretical comparison of the feti and algebraically partitioned feti methods, and performance comparisons with a direct sparse solver*, International Journal for Numerical Methods in Engineering, 46 (1999), pp. 501–533.

[32] E. M. Rønquist and A. T. Patera, *Spectral element multigrid. i. formulation and numerical results*, Journal of Scientific Computing, 2 (1987), pp. 389–406.

[33] K. Rupp, *Cpu-gpu-mic comparision charts*, 2020, https://github.com/karlrupp/cpu-gpu-mic-comparison.