Received: 27 November 2019 | Revised: 25 September 2020 | Accepted: 15 October 2020

DOI: 10.1002/nla.2348

RESEARCH ARTICLE

WILEY

Matrix-free preconditioning for high-order H(curl) discretizations

Andrew T. Barker | **Tzanio Kolev**

Center for Applied Scientific Computing,
Lawrence Livermore National Laboratory,
Livermore, California

Correspondence
Andrew T. Barker, Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA 94550.
Email: atb@llnl.gov

Funding information
Laboratory Directed Research and Development, Grant/Award Number: 20-ERD-002; U. S. Department of Energy, Grant/Award Number: DE-AC52-07NA27344

Abstract

The greater arithmetic intensity of high-order finite element discretizations makes them attractive for implementation on next-generation hardware, but assembly of high-order finite element operators as matrices is prohibitively expensive. As a result, the development of general algebraic solvers for such operators has been an open research challenge. Fast matrix-free application of high-order operators has received significant attention in the literature in the context of Poisson-type problems, but preconditioners and solvers for inverting more general operators are not very well-developed. In this paper, we consider the problem of preconditioning a definite Maxwell operator at high polynomial order without assembling a matrix. We show that given efficient preconditioners for high-order H^1 finite element problems on the same mesh, efficient H(curl) preconditioners can be constructed in an auxiliary space framework. We demonstrate the resulting preconditioners in a practical setting with tensor-product basis functions on an unstructured mesh of quadrilaterals. Our approach uses a sparsified H^1 solver constructed on a low-order mesh of the nodal points of the underlying high-order space, and we show that the resulting H(curl) preconditioner is effective at very high polynomial orders for two-dimensional model problems with complicated geometry, varying piecewise constant coefficients, and curved elements. The resulting preconditioner scales with nearly optimal $O(p^{d+1})$ floating point operation count and optimal $O(p^d)$ memory transfer requirements, outperforming existing Maxwell preconditioners in the high-order regime.

KEYWORDS

auxiliary space solvers, finite elements, high-order methods, Maxwell equations, multigrid

1 | INTRODUCTION

Naive assembly of a single element matrix for a differential operator of polynomial order p costs $O(p^{3d})$ operations (where d is the spatial dimension), which is not feasible for moderate p and is disastrous for high p.[1] If explicit assembly can be avoided, high polynomial order is attractive because it has high arithmetic complexity and therefore makes more efficient use of emerging architectures with fine-grained parallelism, such as accelerators and GPUs.[2,3] In particular, the greater accuracy per degree of freedom in high-order approximations leads to reduced memory movement at the cost of increased

https://doi.org/10.1002/nla.2348

operations, but these additional operations are precisely the kind of local, parallelizable kernels where modern hardware architectures excel. As a result of these considerations, significant effort has gone into unassembled or "matrix-free" application of high-order finite element operators.[4-6] For tensor-product elements (quadrilaterals in two dimensional, 2D, and hexahedra in three dimensional, 3D) with tensor-product basis functions, the action of an operator can be computed in $O(p^{d+1})$ operations. This type of fast operator application is used extensively in explicit time-stepping approaches to fluid flow and other more complicated multiphysics applications.[7-10]

In many practical situations, for example any physics that involves significant diffusion, we need implicit or semi-implicit time-stepping, which implies that we need not only an operator's action but an approximation to its inverse. For explicitly assembled matrices, algebraic multigrid (AMG) preconditioners have been a standard tool of scientific computing because of their flexibility, efficiency, and implementation in robust scalable software,[11-13] but this type of solver is not generally available in the matrix-free case. Approaches involving domain decomposition or geometric multigrid show good promise for preconditioning H^1-type problems without assembling an explicit matrix.[14-20] An emerging theme in these references for diffusion-type problems is preconditioners that can be applied in $O(p^{d+1})$ operations, which is the same asymptotic complexity as an operator-vector multiply.

To our knowledge, fast matrix-free preconditioners for high-order Maxwell operators have not been previously considered in the literature. The key difficulty in Maxwell-type problems, whether with matrix assembly or matrix-free, is handling the large nullspace of the operator. Fortunately, we have a rigorous characterization of that nullspace in the de Rham complex, and perhaps more importantly, we can represent it accurately in practice with finite elements that conform to a discrete de Rham complex. The resulting state-of-the-art for preconditioning in $H(\mathrm{curl})$ is based on an auxiliary space framework which leverages H^1 Poisson-like solvers to handle the nullspace of the Maxwell operator.[21,22] This work provides theoretical guarantees for preconditioning of high-order Maxwell problems, but practical implementation requires high-order auxiliary space solvers that cannot be efficiently assembled.

In this paper we consider fast preconditioners for high-order Maxwell-type problems in an auxiliary space framework, where the underlying operators are never explicitly constructed as matrices. We develop parallel preconditioners that can be applied with nearly optimal $O(p^{d+1})$ operations (the same as applying the forward operator) and which require optimal $O(p^d)$ memory transfer ($O(1)$ per degree of freedom). Our solver shows optimal scaling with respect to spatial discretization size h, is applicable to general geometries and unstructured grids, and performs well on problems with varying piecewise constant coefficients, as well as on curved finite elements. Our particular implementation relies on a low-order refined preconditioner for the auxiliary H^1 problems,[18,23,24] but the framework can accommodate any of the unassembled multigrid or domain decomposition solvers in the literature, and in particular, using low-order approximations in a preconditioner does not affect the accuracy of the original problem. Though our numerical implementation is currently in two dimensions, the methods are fully applicable to three-dimensional problems.

We fix a simple problem setting and notation in section 2 and describe the ingredients and theory behind the auxiliary space Maxwell solver in section 3. One of the most crucial ingredients is unassembled fast preconditioners for the auxiliary H^1 problems, which we describe in section 4. In section 5 we describe in more detail how fast practical application of some of the ingredients of the overall preconditioner can be implemented in software, and numerical results are reported in section 6.

2 | PROBLEM SETTING

Let $\Omega \subset \mathbb{R}^d$ be a bounded, convex domain with smooth boundary in d spatial dimensions, where $d = 2$ is the focus of this paper. We consider the definite Maxwell problem of finding $u \in H_0(\mathrm{curl})$ to satisfy

$$(\alpha \,\mathrm{curl}\, u, \mathrm{curl}\, v) + (\beta u, v) = (f, v), \tag{1}$$

for all $v \in H_0(\mathrm{curl})$, where $\alpha(x), \beta(x)$ are real-valued functions with $\beta > 0$, and f is a given source term. Here $H_0(\mathrm{curl})$ denotes the space of $H(\mathrm{curl})$ functions with vanishing tangential trace on the boundary $\partial\Omega$. This problem has important applications in electromagnetics,[25] and also has the known difficulty of a large nullspace of the curl operator.

Our discretization uses high-order Nedelec edge elements on a mesh of quadrilaterals with possibly curved sides,[26] where we denote the curl-conforming Nedelec space by Q_h and the H_0^1-conforming Lagrange space by V_h. This discretization leads ultimately to the linear algebra problem of finding $u_h \in Q_h$ such that

$$A_h u_h = f_h, \tag{2}$$

where A_h is the discrete operator representation of the bilinear form (1) and f_h is the discrete representation of the linear form (f, v).

The remainder of the paper is concerned with the effective preconditioning of the operator A_h in cases where it is not available as an assembled matrix.

3 | AUXILIARY SPACE MAXWELL SOLVERS

The nullspace of the curl operator in (1) is very large compared to the low-dimensional nullspace of Poisson-like or elasticity problems, and as such any preconditioner for the Maxwell problem needs to pay special attention to that nullspace. In particular, the classical de Rham sequence characterizes the nullspace of the curl operator as the range of the ∇ operator from an H^1 space,

$$\begin{tikzcd} H^1 \arrow[r, "\nabla"] & H(\text{curl}) \arrow[r, "\nabla\times"] & H(\text{div}) \arrow[r, "\nabla\cdot"] & L_2 \end{tikzcd}$$

where the nullspace of each operator is the range of the previous operator in the sequence. For curl-conforming finite elements in the Nedelec space Q_h and the Lagrange space V_h, a corresponding diagram also holds in the discrete setting.

A well-known and very effective approach for handling this nullspace is auxiliary space preconditioning.[21] We follow in particular the auxiliary Maxwell space (AMS) approach of Reference 22, which is well-suited for a fairly general algebraic setting with complicated geometries, unstructured meshes, and varying coefficients. The decomposition of Hiptmair and Xu[21] establishes that any $u_h \in Q_h$ can be expressed as

$$u_h = v_h + G_h q_h + \Pi_h w_h, \tag{3}$$

where $q_h \in V_h, w_h \in V_h^d$ and $v_h \in Q_h$ is small in an appropriate sense. Here the discrete gradient $G_h : V_h \to Q_h$ represents the gradient operator ∇, and the projection Π_h projects $(V_h)^d \to Q_h$.

The decomposition (3) leads naturally to a preconditioning strategy with auxiliary spaces for the range of G_h and Π_h and a smoother for the (small) v_h component. More precisely, the ingredients we need to develop and implement in a matrix-free setting include:

1. The original curl–curl operator $A_h : Q_h \to Q_h$ from (2), the operator we intend to precondition;
2. the vector interpolant $\Pi_h : (V_h)^d \to Q_h$ and its transpose Π_h^T;
3. the discrete gradient $G_h : V_h \to Q_h$ and its transpose G_h^T;
4. a smoother $S_h : Q_h \to Q_h$ on the Nedelec degrees of freedom;
5. and auxiliary space solvers in $(H^1)^d$ and H^1, denoted by $\tilde{L}_{\Pi_h}^{-1} : (V_h)^d \to (V_h)^d$ and $\tilde{L}_{G_h}^{-1} : V_h \to V_h$, respectively, which we consider in more detail below.

Once these ingredients are implemented in a fast setting, we can use them to construct an AMS preconditioner. Several variants of AMS cycles exist. In this paper we focus on the following symmetric cycle:

Given A_h, f_h and an initial guess x_0 (often we choose $x_0 = 0$):

1. Smooth:

$$x_a \leftarrow S_h(f_h - A_h x_0).$$

2. Correct in H^1 (restrict with G_h^T, solve, interpolate):

$$x_b \leftarrow x_a + G_h \tilde{L}_{G_h}^{-1} G_h^T (f_h - A_h x_a).$$

3. Correct in $(H^1)^d$ (restrict with Π_h^T, solve, interpolate) :

$$x_c \leftarrow x_b + \Pi_h \tilde{L}_{\Pi_h}^{-1} \Pi_h^T (f_h - A_h x_b).$$

4. Correct in H^1 (restrict with G_h^T, solve, interpolate):

$$x_d \leftarrow x_c + G_h \tilde{L}_{G_h}^{-1} G_h^T (f_h - A_h x_c).$$

5. Smooth:

$$x \leftarrow S_h(f_h - A_h x_d).$$

We denote the action of this AMS cycle by $x = M_h^{-1} f_h$. The remainder of this paper focuses on the implementation and performance of these fast, matrix-free ingredients.

4 | LOW-ORDER REFINED PRECONDITIONERS

The auxiliary space preconditioners $\tilde{L}_{\Pi_h}^{-1}, \tilde{L}_{G_h}^{-1}$ that we use in a practical AMS cycle involve several different spaces and operators, as well as certain approximations. We describe in this section the operators analyzed in auxiliary space theory, the approximations we use in practice, and the precise relationships among them.

In a discrete setting, the natural auxiliary space operators for constructing a provably spectrally equivalent preconditioner M_h^{-1} in the sense of the fictitious space lemma (theorem 2.2 of Reference 21) are $L_{\Pi_h} = \Pi_h^T A_h \Pi_h$ and $L_{G_h} = G_h^T A_h G_h$. In the low-order case,[22] these operators are formed explicitly as matrices and AMG is applied to those matrices. The fictitious space lemma still applies for high-order discretizations, but the operators cannot be assembled efficiently. In our high-order matrix-free setting, we can in fact apply these operators efficiently in an unassembled format by applying their three factors in sequence, but applying a solver or preconditioner that approximates their inverse is not quite as straightforward.

For relatively general application on unstructured, possibly curved meshes with varying coefficients, we use the idea of low-order refined preconditioning introduced by Orszag in a spectral element setting[24] and further analyzed in several other places.[18,23] To begin, we rediscretize the following Poisson problems on the auxiliary Lagrange spaces:

$$a_{G_h}(u, v) = (\beta \nabla u, \nabla v), \tag{4}$$

$$a_{\Pi_h}(\mathbf{u}, \mathbf{v}) = (\alpha \nabla \mathbf{u}, \nabla \mathbf{v}) + (\beta \mathbf{u}, \mathbf{v}) = \alpha \int_\Omega \nabla \mathbf{u} : \nabla \mathbf{v}\, dx + \int_\Omega \beta \mathbf{u} \cdot \mathbf{v}\, dx, \tag{5}$$

where for a vector function $\mathbf{u}$ we define $\nabla \mathbf{u}$ as a tensor, that is, $[\nabla \mathbf{u}]_{ij} = \partial[\mathbf{u}]_i / \partial x_j$. If the bilinear forms above are taken in the high-order Lagrange spaces $u, v \in V_h; \mathbf{u}, \mathbf{v} \in V_h^d$ then we write the corresponding operators as $L_{G_h}^p$ for a_{G_h} and $L_{\Pi_h}^p$ for a_{Π_h}. These operators themselves are high-order and therefore prohibitively expensive to assemble, so we approximate them again. The basic idea is to construct a low-order mesh on the *nodal points* of the original high-order mesh, as in Figure 1. The first-order Lagrange finite element space on the mesh of nodal points is denoted by $V_{h,1}$. This low-order refined space has the same number of degrees of freedom as the original high-order space V_h, and in fact if we use a nodal basis for both V_h and $V_{h,1}$, then the operator connecting the degrees of freedom on the spaces is simply the identity. However, the resulting system matrix on the low-order space $V_{h,1}$ is much sparser than the high-order space. In particular, a single element matrix in the high-order space V_h has $O(p^d)$ nonzeros per row, while the corresponding low-order element matrix has $O(1)$ nonzeros per row. In addition, the low-order space requires lower order quadrature, compounding the savings for matrix assembly. As a result, explicit matrices can be efficiently assembled for the low-order space $V_{h,1}$.

Then taking $u, v \in V_{h,1}; \mathbf{u}, \mathbf{v} \in V_{h,1}^d$ in (4), (5), we denote the corresponding operators as $L_{G_h}^1$ and $L_{\Pi_h}^1$. Since these operators can be efficiently assembled, we can use standard matrix-based AMG to approximate their inverses. The approximate inverses given by (assembled) AMG solvers on the low-order refined mesh are denoted by $\tilde{L}_{G_h}^{-1} \approx (L_{G_h}^1)^{-1}$ and $\tilde{L}_{\Pi_h}^{-1} \approx (L_{\Pi_h}^1)^{-1}$. This notation has already been used in the definition of the AMS cycle M_h^{-1}. (We never form, apply, or use in any way the operators $L_{G_h}^p, L_{\Pi_h}^p$ in practice. They are introduced only to clarify concepts and exposition.)

Another way of thinking about the low-order refined technique is that it trades a mesh with high p, where traditional AMG scales unfavorably, for a mesh with small h, which is the regime in which traditional AMG scales very well. We note, however, that since the low-order mesh is used to precondition the original high-order operator, we do not lose any of

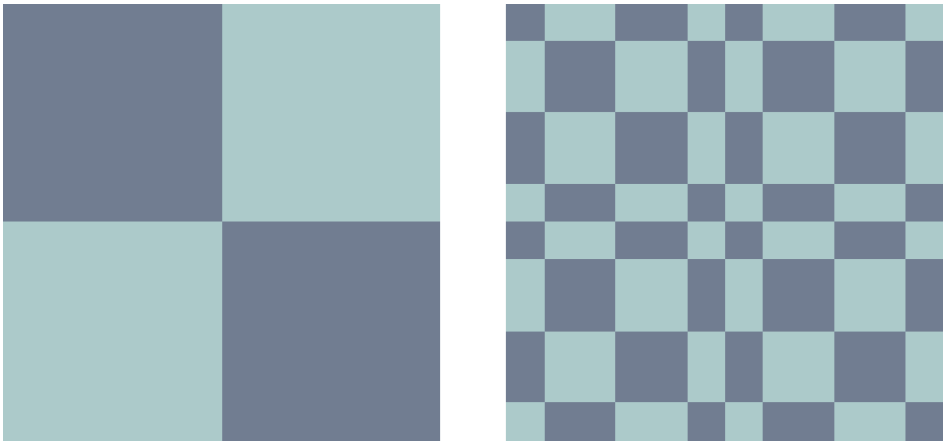


FIGURE 1 A high-order mesh and its corresponding low-order refined mesh

$$\begin{tikzcd}
 & & L_{\Pi_h} \arrow[r, leftrightarrow, "{[22,\ \text{Theorem } 3.2]}"] & L^p_{\Pi_h} \arrow[r, leftrightarrow, "{[23,\ 27]}"] & L^1_{\Pi_h} \arrow[r, leftrightarrow, "\text{AMG literature}"] & \tilde{L}^{-1}_{\Pi_h} \\
A_h \arrow[r, leftrightarrow, "{[22,\ \text{Theorem } 4.1]}"] & M_h^{-1} \arrow[ur, dotted] \arrow[dr, dotted] & & & & \\
 & & L_{G_h} \arrow[r, leftrightarrow, "{[22,\ \text{Theorem } 3.2]}"] & L^p_{G_h} \arrow[r, leftrightarrow, "{[23,\ 27]}"] & L^1_{G_h} \arrow[r, leftrightarrow, "\text{AMG literature}"] & \tilde{L}^{-1}_{G_h}
\end{tikzcd}$$

FIGURE 2 Relationships between some of the operators and approximate operators used to construct our auxiliary space preconditioner. Solid lines indicate spectral equivalence between operators, while the dotted lines simply indicate that M_h^{-1} requires other operators for its construction

the accuracy associated with high-order methods. One potential difficulty is that the elements become quite anisotropic for high p, which may affect AMG performance—we find below in our numerical results that this does not seem to be a problem for constant coefficients α, β but may present mild difficulties when coefficients vary.

Well-known results from the spectral element literature show that $L^1_{G_h}, L^1_{\Pi_h}$ are spectrally equivalent to (respectively) $L^p_{G_h}, L^p_{\Pi_h}$ with constants that are independent of the polynomial order p.[23,27] The equivalence of $L^p_{G_h}, L^p_{\Pi_h}$ with L_{G_h}, L_{Π_h} is established in (theorem 3.2 of Reference 22). The only thing remaining is to establish the quality of AMG V-cycles $\tilde{L}^{-1}_{G_h}, \tilde{L}^{-1}_{\Pi_h}$ as approximations to $(L^1_{G_h})^{-1}, (L^1_{\Pi_h})^{-1}$, which is the subject of a vast AMG literature[12,13,28,29] and is out of scope for this paper, although it will be established numerically in section 6.

The theoretical relationships between all these operators is summarized in Figure 2.

5 | TENSORIZED IMPLEMENTATION

For a fast implementation to precondition an operator where an explicit matrix is not available, we need fast, matrix-free applications of each of the operators in section 3. In particular, we need fast tensorized routines for A_h, G_h, G_h^T, Π_h, Π_h^T, and S_h, as well as a fast way to apply $\tilde{L}^{-1}_{G_h}, \tilde{L}^{-1}_{\Pi_h}$ which have already been described. Each of these operators should be applied with computational complexity $O(p^{d+1})$ (to match the forward operator-vector product) and optimal memory transfers $O(p^d)$.

Our particular implementation depends on a tensor-product representation of basis functions on tensor-product finite elements (i.e., quadrilaterals and hexahedra). In this paper we are focused on quadrilateral elements in two dimensions, $d = 2$, though all of the algorithms can be easily extended to $d = 3$. The structure of the high-order Nedelec element we consider has two sets of basis functions, one "horizontal" tensor product (whose vector basis functions have 0 y-component on the reference element) and one "vertical" tensor product. Each set is a tensor product of a "closed" integration rule (Gauss–Lobatto) and an "open" integration rule (Gauss–Legendre), see Figure 3.

We choose nodal degrees of freedom for the high-order Nedelec space, instead of the more traditional integral moments, because they naturally inherit the tensor-product structure of the element. This is critical for the efficient

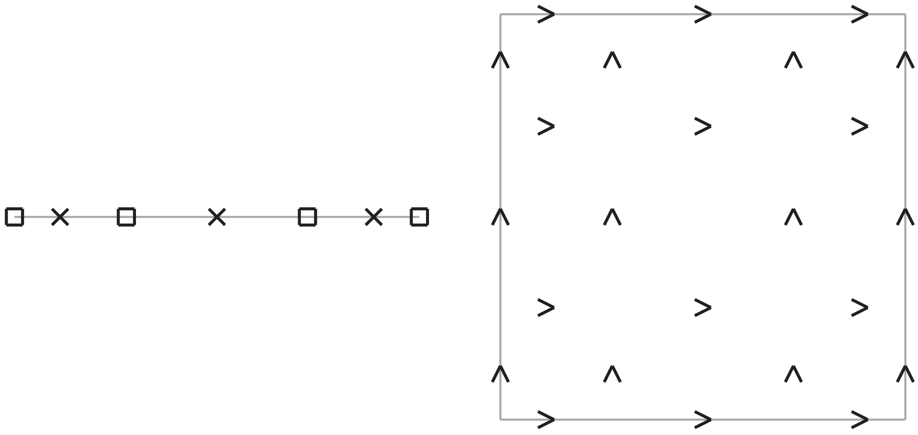


FIGURE 3 On the left, Gauss–Lobatto (closed) points indicated by squares and Gauss–Legendre (open) points indicated by crosses on a one-dimensional interval. On the right, nodes for third order Nedelec H(curl) basis functions on a quadrilateral element. Nedelec degrees of freedom are inner products with unit vectors that have the indicated position and direction. The degrees of freedom of the "horizontal" basis functions are located at tensor products of Gauss–Lobatto with Gauss–Legendre points, while the "vertical" basis functions are located at the transposed set of points

implementation of the intergrid and Nedelec operators discussed in the rest of this section. Since details about the Nedelec space and the specific definition of the nodal high-order Nedelec degrees of freedom play an important role in the following, we provide some details below.

A Nedelec finite element function $\psi \in Q_h$ is transformed from a physical element E to the reference element $\hat{E}$ using the Piola transformation:

$$\psi(x) = J_E(\hat{x})^{-T}\hat{\psi}(\hat{x}) \ . \tag{6}$$

Here J_E is the Jacobian of the transformation $\Phi_E : \hat{E} \mapsto E, \hat{x}$ is a point in reference space, x is a point in physical space, and $x = \Phi_E(\hat{x})$. The Piola transformation (6) is also used to transform the gradients of nodal functions $v \in V_h$:

$$\nabla v(x) = J_E(\hat{x})^{-T}\hat{\nabla}\hat{v}(\hat{x}) \ . \tag{7}$$

In the familiar 3D setting, the curl of Nedelec fields is transformed with the corresponding $H(div)$ transformation

$$\mathrm{curl}\,\psi(x) = \frac{1}{|J_E(\hat{x})|}J_E(\hat{x})\ \widehat{\mathrm{curl}}\ \ \hat{\psi}(\hat{x}) \ , \tag{8}$$

but in our 2D setting this simplifies significantly to

$$\mathrm{curl}\,\psi(x) = \frac{1}{|J_E(\hat{x})|}\widehat{\mathrm{curl}}\ \ \hat{\psi}(\hat{x}) \ , \tag{9}$$

which can be derived from (8) by thinking of the 3D Jacobian as the 2D Jacobian extended by an identity on the diagonal and regarding the 2D curl as the z-component of the 3D curl.

While the above facts are classical, the basis and degrees of freedom we use for the high-order Nedelec space are less standard. Specifically, for a fixed element E, the degrees of freedom are dot products of $\psi \in Q_h$ with a set of tangent vectors t_i at a set of points x_i. The points are mapped from Cartesian products of open and closed points on the reference element, $x_i = \Phi_E\hat{x}_i$ as indicated in Figure 3. The tangent vectors are tangentially mapped from the unit reference element vectors in Figure 3, that is, $t_i = J_E(\hat{x}_i)\hat{t}_i$. Thus, the degree of freedom associated with the point $x_i \in E$ is

$$\psi(x_i) \cdot t_i = \psi(x_i) \cdot J_E(\hat{x}_i)\hat{t}_i. \tag{10}$$

Note that due to (6), this could also be written as $\hat{\psi}(\hat{x}_i) \cdot \hat{t}_i$, because in reference space the degrees of freedom are just the dot products of the (Piola) transformed Nedelec functions with the unit vectors $\hat{t}_i$ at the points $\hat{x}_i$. Note also, that we need to flip the sign of Nedelec degrees of freedom on edges that have different local and global orientation, when switching from element to global degrees of freedom.

Below we describe the tensorized matrix-free implementation of some of the important operators in our preconditioner. In particular, we describe matrix-free discrete gradient G_h and discrete projection Π_h implementations, and also the application of an operator-vector multiply for the Nedelec mass matrix and curl–curl matrix (i.e., the discretizations of the (u, v) term and $(\text{curl}u, \text{curl}v)$ terms in (1), respectively).

To describe our tensor algorithms, we will use multi-index notation $i = (i_1, i_2)$ for basis function and quadrature indices, where i indexes into a d-dimensional tensor product and i_k indexes into one of the one-dimensional (1D) components of the tensor product. In our tensor product setting, we will often refer to the one-dimensional nodal basis functions ϕ^o, ϕ^c, where the nodal points for ϕ^o are on an "open" Gauss–Legendre set of points and ϕ^c is on a "closed" Gauss–Lobatto set of points that include the endpoints of the interval (see the left side of Figure 3). Similarly, the points at which these basis functions are evaluated can be denoted x^o, x^c for clarity.

5.1 | Matrix-free intergrid operators

We describe here the "matrix-free" tensorized implementation of the discrete gradient operator $G_h : V_h \to Q_h$ and the discrete projection $\Pi_h : V_h^d \to Q_h$ which connect the Nedelec space Q_h to the Lagrange auxiliary spaces used in our auxiliary space preconditioner.

Note that these operators map degrees of freedom (dofs) to degrees of freedom, so the rows (indexed by $i = (i_1, i_2)$) of both these operators are associated with nodal points of the two Nedelec tensor products of Figure 3, while the columns (indexed by $j = (j_1, j_2)$) are associated with the tensor Lagrange basis functions of closed Gauss–Lobatto points.

As discussed in Reference 22, to compute the entries of the discrete gradient, we need to evaluate the gradient of a basis function from V_h in the dof of Q_h. Therefore, we can decompose the discrete gradient as

$$G_h = \begin{pmatrix} G_h^{\text{h}} \\ G_h^{\text{v}} \end{pmatrix}.$$

Combining (7) and (10) we see that the entries of G_h are given simply by

$$\hat{\nabla}\hat{v}(\hat{x}_i) \cdot \hat{t}_i,$$

and therefore

$$\begin{aligned} [G_h^{\text{h}}]_{i=(i_1,i_2),j=(j_1,j_2)} &= q_i \phi_{j_2}^c(x_{i_2}^c)(\phi_{j_1}^c)'(x_{i_1}^o) \\ [G_h^{\text{v}}]_{i=(i_1,i_2),j=(j_1,j_2)} &= q_i (\phi_{j_2}^c)'(x_{i_2}^o)\phi_{j_1}^c(x_{i_1}^c). \end{aligned}$$

Here $q_i \in \{-1, 1\}$ represents the orientation of the Nedelec dof at the point x_i; for a general unstructured mesh, this information must be provided in some form for every Nedelec dof. Define

$$C_{kj}^{oc} = (\phi_j^c)'(x_k^o), \quad B_{kj}^{cc} = \phi_j^c(x_k^c), \quad B_{kj}^{oc} = \phi_j^c(x_k^o).$$

The superscript notation here indicates that the rows of C^{oc} are associated with open (Nedelec) nodal points, while the columns are associated with closed (Lagrange) nodal points, and similarly for B^{cc}, B^{oc}. Note that B^{cc} is actually the identity but we use this notation to clarify the tensor product structure of the algorithms. The matrices B^{oc}, B^{cc}, C^{oc} are fixed once for all, having the same values and structure for each element of the mesh. In addition, since these operators are constructed on a one-dimensional interval and used in combination to represent multidimensional basis functions, the setup for this tensor product form of G_h costs $O(p^2)$ independent of d.

To describe the algorithm for applying the operator, let $V = v_{j_1,j_2} \in V_h$ be an H^1 vector we want to premultiply by Ghh. Then

$$\begin{aligned}(G^{\mathrm{h}}_h V)_{(i_1,i_2)} &= q_i \sum_{j_1}\sum_{j_2} (\phi^c_{j_1})'(x^o_{i_1})\phi^c_{j_2}(x^c_{i_2}) v_{j_1,j_2}\\ &= q_{i_1,i_2}\sum_{j_1}\sum_{j_2} C^{oc}_{i_1,j_1} B^{cc}_{i_2,j_2} v_{j_1,j_2}\\ &= q_{i_1,i_2}\sum_{j_1} C^{oc}_{i_1,j_1}\sum_{j_2} B^{cc}_{i_2,j_2} v_{j_1,j_2}.\end{aligned}$$

The evaluation of the output vector for all i_1, i_2 involves two tensor contractions over j_1 and j_2 and has a total cost of $O(p^3)$ because of its sum factorization structure. The vertical basis Ghv is similar.

For the transpose, let $Z = Z_{i_1,i_2} \in Q_h$ be the H(curl) vector we want to premultiply by $(G^{\mathrm{h}}_h)^T$. Then

$$\begin{aligned}[(G^{\mathrm{h}}_h)^T Z]_{(j_1,j_2)} &= \sum_{i_1}\sum_{i_2} q_{i_1,i_2}(\phi^c_{j_1})'(x^o_{i_1})\phi^c_{j_2}(x^c_{i_2}) z_{i_1,i_2}\\ &= \sum_{i_1}\sum_{i_2} q_{i_1,i_2} C^{oc}_{i_1,j_1} B^{cc}_{i_2,j_2} z_{i_1,i_2}\\ &= \sum_{i_1} C^{oc}_{i_1,j_1}\sum_{i_2} q_{i_1,i_2} B^{cc}_{i_2,j_2} z_{i_1,i_2},\end{aligned}$$

which is again two contractions for a cost of $O(p^3)$, and again the vertical basis is similar.

The operator G_h depends only on the topology (not the geometry) of the mesh nodes, and in general does not depend on the actual element sizes, coefficients, or element curvature. The operator Π_h, on the other hand, does depend on the particular geometry of the mesh. We can think of $\Pi_h : V^d_h \to Q_h$ as a d-by-d block operator, with the top half corresponding to "horizontal" Nedelec dofs on the reference element and the bottom corresponding to "vertical" Nedelec dofs. The left and right sides of the operator correspond to x and y components of the Lagrange dofs. That is,

$$\Pi_h = \begin{pmatrix} \Pi^{\mathrm{h},x}_h & \Pi^{\mathrm{h},y}_h \\ \Pi^{\mathrm{v},x}_h & \Pi^{\mathrm{v},y}_h \end{pmatrix}.$$

Given $\mathbf{v} \in V^d_h$ its projection on the Nedelec dofs according to (10) is

$$\mathbf{v}(x_i) \cdot t_i = \hat{\mathbf{v}}(\hat{x}_i) \cdot J_E(\hat{x}_i)\hat{t}_i.$$

Note that unlike Nedelec fields, $\mathbf{v}$ transforms to reference space with a simple (not Piola) transformation, so we get an additional Jacobian term in Π_h compared to G_h. Therefore, we can write

$$(\Pi^{\mathrm{h},x}_h)_{i=(i_1,i_2),j=(j_1,j_2)} = \phi^c_{j_1}(x^o_{i_1})\phi^c_{j_2}(x^c_{i_2})(J_E t_{i_1,i_2})^x,$$

where the rows i correspond to Nedelec nodal points and the columns j correspond to Lagrange nodal points. Here (as before) $\phi^c_{j_k}$ is a 1D Lagrange basis function for the (closed, Gauss–Lobatto) nodal point j_k, t is the tangent vector associated with the nodal point i on the reference element, and J_E is the Jacobian of the element transformation. The notation $(J_E t_{i_1,i_2})^x$ indicates that we take the reference tangent vector for the point i, premultiply with the Jacobian, and then take the x-component. The tensor product form of one of the components can then be written

$$\begin{aligned}[(\Pi^{\mathrm{h}}_h)V]_{i_1,i_2} &= \sum_{j_1}\sum_{j_2} B^{oc}_{i_1,j_1} B^{cc}_{i_2,j_2}(J_E t_{i_1,i_2}) v_{j_1,j_2}\\ &= (J_E t_{i_1,i_2})\sum_{j_1} B^{oc}_{i_1,j_1}\sum_{j_2} B^{cc}_{i_2,j_2} v_{j_1,j_2},\end{aligned}$$

and the other components of Π_h are similar. The action of the transpose operator can be implemented in a similar way.

5.2 | Matrix-free Nedelec operators

To describe a tensor-based implementation of Nedelec mass and curl-curl operators $Q_h \to Q_h$, define

$$B^{qo}_{k,i} = \phi^o_i(x^q_k), \quad B^{qc}_{k,i} = \phi^c_i(x^q_k),$$

as 1D Lagrange basis functions associated with Nedelec degrees of freedom on either the open Gauss–Legendre or closed Gauss–Lobatto nodal points, evaluated at the (always open Gauss–Legendre) quadrature points x_k. We have the similar 1D derivative matrices

$$C^{qo}_{k,i} = (\phi^o_i)'(x^q_k), \quad C^{qc}_{k,i} = (\phi^c_i)'(x^q_k).$$

The notation here indicates that these matrices have rows associated with quadrature points, and columns associated with either open or closed nodesets for the Nedelec tensor products. These B and C matrices are the same for each element of the mesh, depending only on the basis functions and quadrature rules chosen.

For the operator A_h, we begin with the curl–curl term. A "horizontal" Nedelec vector basis function ψ evaluated at the quadrature point indexed by $k=(k_1,k_2)$ on the reference element can be written

$$\psi^k_i = \begin{pmatrix} B^{qo}_{k_1,i_1} B^{qc}_{k_2,i_2} \\ 0 \end{pmatrix}. \tag{11}$$

We have

$$\text{curl}\ \psi^k_i = -B^{qo}_{k_1,i_1} C^{qc}_{k_2,i_2}.$$

Now we want to consider the action of the curl–curl matrix $A_{i,j} = (\text{curl}\ \psi_i, \text{curl}\ \psi_j)$ on a vector $Z = Z_{j_1,j_2} \in Q_h$ on a single physical element E.

We have from (9)

$$\begin{aligned}(AZ)_{i_1,i_2} &= \sum_{j_1}\sum_{j_2}\int_E \text{curl}\ \psi_i\ \text{curl}\ \psi_j Z_{j_1,j_2}\, dx \\ &= \sum_{j_1}\sum_{j_2}\int_{\hat{E}} \det(J_E)(\det(J_E^{-1})\ \hat{\text{curl}}\ \ \hat{\psi}_i)(\det(J_E^{-1})\ \hat{\text{curl}}\ \ \hat{\psi}_j) Z_{j_1,j_2}\, d\hat{x},\end{aligned}$$

and then assuming for the moment that both ψ_i, ψ_j represent horizontal basis functions and using (11) we can write

$$\begin{aligned}(AZ)_{i_1,i_2} &= \sum_{j_1}\sum_{j_2}\sum_{k_1}\sum_{k_2} w_k \det(J_k^{-1}) B^{qo}_{k_1,i_1} C^{qc}_{k_2,i_2} B^{qo}_{k_1,j_1} C^{qc}_{k_2,j_2} Z_{j_1,j_2} \\ &= \sum_{k_1} B^{qo}_{k_1,i_1} \sum_{k_2} d_{k_1,k_2} C^{qc}_{k_2,i_2} \sum_{j_1} B^{qo}_{k_1,j_1} \sum_{j_2} C^{qc}_{k_2,j_2} Z_{j_1,j_2}\end{aligned}$$

where $d_{k_1,k_2} = w_k \det(J_k^{-1})$ and w_k is the integration weight at quadrature point k. This d coefficient is different for every quadrature point on the entire mesh, and calculating and storing it is the main setup cost for this matrix-free operator. Note, however, that this storage cost is optimal $O(p^d)$, which is proportional to the number of dofs. For different combinations of horizontal and vertical basis functions, the tensor contractions are similar with some B and C tensors exchanged in the above description.

We next describe an algorithm for applying a Nedelec mass matrix without assembly, which is another part of the operator A_h. In this setting the rows and columns of the operator are both associated with Nedelec dofs as in Figure 3. Let

$$J_k^{-1} = \begin{pmatrix} d^{00}_k & d^{01}_k \\ d^{10}_k & d^{11}_k \end{pmatrix},$$

where J_E is the element Jacobian for the reference finite element mapping in two dimensions and $k = (k_1, k_2)$ is the index of a (tensor product) quadrature rule.

Recalling the definition (11), we have

$$J_k^{-1}\psi_i^k = \begin{pmatrix} d_k^{00} B^{qo}_{k_1,i_1} B^{qc}_{k_2,i_2} \\ d_k^{10} B^{qo}_{k_1,i_1} B^{qc}_{k_2,i_2} \end{pmatrix}.$$

Finally, define the scalar $g_k^{\mathrm{h}} = \det(J_k) w_k((d_k^{00})^2 + (d_k^{10})^2)$. This quantity g_k^{h} is different on each quadrature point of the mesh, and represents the main storage cost for the mass operator. As for the curl–curl operator, this storage cost is optimal $O(p^d)$.

Now the action of an element mass matrix on a physical element E on a vector $Z = Z_{i_1,i_2} \in Q_h$, if i and j both index into horizontal Nedelec basis functions, can be written using (6)

$$\begin{aligned}
(MZ)_{i_1,i_2} &= \sum_{j_1}\sum_{j_2}\int_E \psi_i \cdot \psi_j Z_{j_1,j_2}\, dx \\
&= \sum_{j_1}\sum_{j_2}\int_{\hat{E}} \det(J_E)(J_E^{-T}\hat{\psi}_i)\cdot(J_E^{-T}\hat{\psi}_j) Z_{j_1,j_2}\, dx \\
&= \sum_{k_1}\sum_{k_2}\sum_{j_1}\sum_{j_2} g_k^{\mathrm{h}} B^{qo}_{k_1,i_1} B^{qc}_{k_2,i_2} B^{qo}_{k_1,j_1} B^{qc}_{k_2,j_2} Z_{j_1,j_2} \\
&= \sum_{k_1} B^{qo}_{k_1,i_1} \sum_{k_2} B^{qc}_{k_2,i_2} g^{\mathrm{h}}_{k_1,k_2} \sum_{j_1} B^{qo}_{k_1,j_1} \sum_{j_2} B^{qc}_{k_2,j_2} Z_{j_1,j_2},
\end{aligned}$$

which can be evaluated by a series of tensor contractions for a computational cost of $O(p^{d+1})$ and optimal memory transfer $O(p^d)$. For vertical basis functions, the open and closed basis tensors get switched around a bit and the scalar g_k^{v} has a slightly different form, but the basic algorithm is very similar.

To implement a point Jacobi preconditioner for the Nedelec operator A_h, we also need to be able to efficiently assemble the diagonal of the operators A, M. This turns out to be fairly straightforward. For example, for the mass matrix we have

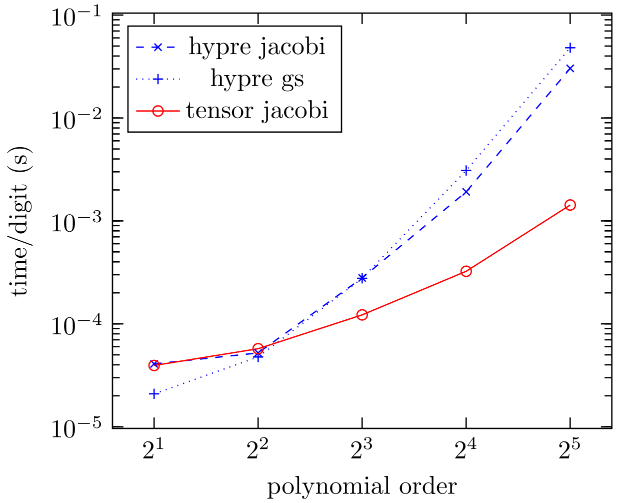
$$\begin{aligned}
M_{ii} &= \int_E \psi_i \cdot \psi_i\, dx = \int_{\hat{E}} \det(J_E)(J_E^{-T}\psi_i)\cdot(J_E^{-T}\psi_i)\, dx \\
&= \sum_{k_1}\sum_{k_2} \det(J_k) w_k \begin{pmatrix} d_k^{00} B^{qo}_{k_1,i_1} B^{qc}_{k_2,i_2} \\ d_k^{10} B^{qo}_{k_1,i_1} B^{qc}_{k_2,i_2} \end{pmatrix} \cdot \begin{pmatrix} d_k^{00} B^{qo}_{k_1,i_1} B^{qc}_{k_2,i_2} \\ d_k^{10} B^{qo}_{k_1,i_1} B^{qc}_{k_2,i_2} \end{pmatrix} \\
&= \sum_{k_1}\sum_{k_2} \det(J_k) w_k ((d_k^{00})^2 + (d_k^{10})^2)(B^{qo}_{k_1,i_1} B^{qc}_{k_2,i_2})^2 \\
&= \sum_{k_1} (B^{qo}_{k_1,i_1})^2 \sum_{k_2} g_{k_1,k_2} (B^{qc}_{k_2,i_2})^2,
\end{aligned}$$

which can be evaluated by two tensor contractions for a computational cost of $O(p^{d+1})$. For a vertical basis function, the "o" and "c" above are swapped and the g_k coefficient takes a slightly different form.

6 | NUMERICAL RESULTS

Below we present numerical results showing the robustness and scalability (especially with respect to polynomial order p) of our unassembled auxiliary space Maxwell preconditioner. To summarize, our smoother S_h is a single sweep of matrix-free damped Jacobi with damping 1/4; the interpolation operators $G_h, G_h^T, \Pi_h, \Pi_h^T$ are implemented in a tensorized matrix-free form; the auxiliary space solvers are based on AMG for the low-order refined $L^1_{\Pi_h}, L^1_{G_h}$ operators, sometimes with an intermediate CG iteration; and the overall AMS preconditioner M_h^{-1} is used as a preconditioner for an outer conjugate gradient iteration. The outer CG iteration is stopped at relative residual reduction of 10^{-12}. The implementation uses the MFEM package[30,31] and is fully parallel, but the results below focus on scalability with respect to p and

FIGURE 4 Time per digit of residual reduction for application of matrix-based and tensor-based smoothers for the Nedelec operator A_h from (2)



not parallel scalability. The parameters for BoomerAMG are the defaults in the MFEM interface to hypre, so that we use HMIS coarsening with one level of aggressive coarsening, and a one-sweep ℓ_1-Gauss–Seidel smoother.

6.1 | Components in isolation

Before presenting results for the full auxiliary space solver applied to the problem (1), we consider the performance of some of its underlying components in isolation, starting with smoothers.

A smoother for A_h is essential for the basic auxiliary framework (3). A local Fourier analysis for (1) in the geometric multigrid context is performed in Reference 32. The smoothers they consider are of Hiptmair type, and their analysis of point smoothers (such as Jacobi or Gauss–Seidel) is only to show that these smoothers are insufficient for a standard geometric multigrid. In our auxiliary space setting the requirements on the smoother are quite different. In particular, in Reference 22 it is shown that the smoother in an auxiliary space solver should be spectrally equivalent to an operator related to a scaled Nedelec mass matrix, which in turn is spectrally equivalent to the diagonal of A_h. In Reference 33, 7.11, the auxiliary space smoother is required to be convergent in the A_h inner product, and a scaled identity is proposed as a smoother. As a result, point smoothers can be expected to work well in the AMS cycle. The default option for AMS in hypre is Gauss–Seidel, but that choice is not available here because we cannot form an explicit matrix, so we rely instead on a damped Jacobi smoother.

Since the coefficients in the problem are already included in the diagonal of A_h we find (from extensive numerical experiments not shown) that a damping parameter of 1/4 works well for a wide variety of coefficients, problem sizes, and polynomial order. A more theoretically grounded strategy would be to choose the damping based on the absolute row sum of the operator, or based on the number of nonzeros in a row (cf, Reference 33, proposition 1.10), but in the high-order setting these give pessimistic bounds that are not very useful in practice. Another potential smoothing strategy is polynomial smoothing, which would likely have many advantages, but is out of scope for this paper where the focus is on the overall auxiliary-space multigrid cycle.

In the high-order context, using a damped Jacobi smoother not only saves us the cost of assembling the matrix, but applying a matrix-free smoother is also faster. In Figure 4 we plot cost per accuracy, or more precisely, computational time per digit of residual reduction for three smoothers: the Jacobi and Gauss–Seidel preconditioners from hypre, and our tensorized matrix-free Jacobi preconditioner. The matrix-free tensor preconditioner is faster than Gauss–Seidel for $p > 4$, and more importantly its cost increases as a much slower rate with respect to p.

Next we consider the low-order refined preconditioner $L^1_{G_h}$ for the elliptic-type auxiliary problem defined by the triple product $L_{G_h} = G_h^T A_h G_h$ in isolation from the rest of the AMS cycle. In Table 1 we show the convergence factor for the triple product operator when preconditioned by hypre BoomerAMG assembled for both the high-order triple-product and the low-order refined rediscretization, using a toy 2×2 mesh of squares on the domain $\Omega = [0,1]^2$. The effectiveness of this approximation is shown in the table, where the low-order refined technique still leads to acceptable convergence rates for constant coefficients and orders up to $p = 16$. For a case with varying piecewise constant coefficients (see subsection 6.3

	$\beta = 1$		$\beta = 10^4$	
Order	L_{G_h}	$L^1_{G_h}$	L_{G_h}	$L^1_{G_h}$
2	0.18	0.38	0.05	0.34
4	0.20	0.47	0.21	0.38
8	0.18	0.52	0.21	0.42
16	0.21	0.56	0.22	0.55

TABLE 1 Convergence factor for conjugate gradient on the operator $L_{G_h} = G_h^T A_h G_h$ preconditioned with BoomerAMG built for either the matrix $G_h^T A_h G_h$ (assembled) or the matrix $L^1_{G_h}$ (low-order refined)

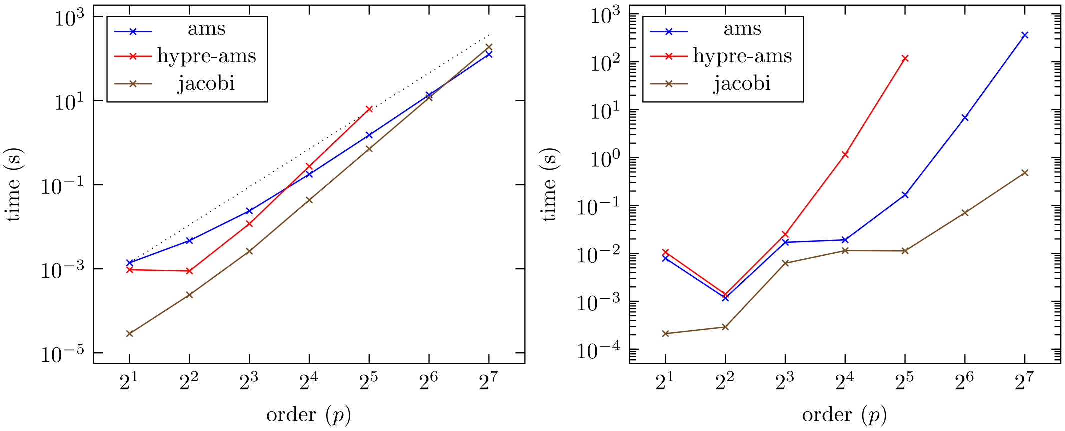


FIGURE 5 Solve time on the left, setup (including assembly) time on the right, for varying polynomial order p and preconditioning strategies for the outer CG iteration. The dotted black line on the left is $O(p^3) = O(p^{d+1})$ scaling, which the matrix-free auxiliary Maxwell space preconditioner matches

for details), the results are also quite reasonable. This approximation of a solve on the (high-order) auxiliary space by a preconditioner application from the low-order refined space is perhaps the weakest link in the chain of equivalences outlined in Figure 2. More accurate auxiliary space solves would lead to very low iteration counts, as can be seen in the "tensor-∞" lines in Table 5 below. However, more accurate solves in a high-order H^1 space are also much more expensive and their implementation is a research topic in its own right. The low-order refined approach used here is relatively easy to implement, and the corresponding V-cycle is very fast to apply in practice even if the resulting iteration count is not quite as low as could be achieved by some other method.

6.2 | Constant coefficients

The first results for the full Maxwell problem (1) are on a toy 2×2 mesh of squares on the domain $\Omega = [0,1]^2$ with constant coefficients $\alpha = \beta = 1$. We approximate $L^{-1}_{\Pi_h}$ with a single BoomerAMG V-cycle built for the operator $L^1_{\Pi_h}$, and we approximate $L^{-1}_{G_h}$ with a single CG iteration preconditioned with a BoomerAMG V-cycle built for $L^1_{G_h}$. The results for this very simple problem are in Figure 5 and Table 2. They show that our matrix-free approach leads to a reasonable and practical preconditioner even for extremely high polynomial orders where it is prohibitively difficult to construct a matrix. In particular, the scaling with respect to order for the inverse of the operator in Figure 5 is proportional to the nearly optimal $O(p^{d+1})$ timing of the forward application of the same operator. Though the iteration counts in Table 2 grow somewhat with p, the growth is moderate, and very high p is still possible with this approach.

To better understand the overall scaling of the matrix-free preconditioner, we break down the solve time into several components in Table 3, which reports totals over the entire solve for the smoother S_h, residual calculations within the AMS preconditioner (but not in the outer conjugate gradient iteration), and the auxiliary solves $\tilde{L}^{-1}_{G_h}$ and $\tilde{L}^{-1}_{\Pi_h}$. For low-order problems ($p \leq 8$), initialization time can be significant and variable, so we run the solver 10 times and take the average for the solve timings. We note that residual calculations and $\tilde{L}^{-1}_{G_h}$ represent by far the greatest expense in the cycle, but that they both scale with the expected $O(p^{d+1})$ at high order.

TABLE 2 Iteration counts, 2^2 elements, various polynomial orders p

	Matrix-free		Assembled
Order	**Tensor AMS**	**Tensor Jacobi**	**hypre AMS**
2	6	3	4
4	10	7	6
8	12	17	8
16	16	48	8
32	22	112	8
64	27	247	—
96	30	383	—
128	31	521	—

Abbreviation: AMS, auxiliary Maxwell space.

TABLE 3 Timings in seconds for several parts of the matrix-free auxiliary Maxwell space cycle corresponding to the runs in Figure 5 and Table 2

Order	Smoothers	Residuals	$\tilde{L}_{G_h}^{-1}$	$\tilde{L}_{\Pi_h}^{-1}$	Total
2	1.9e-05	1.1e-04	8.9e-04	5.6e-05	1.4e-03
4	3.4e-05	1.2e-03	2.2e-03	1.9e-04	4.7e-03
8	6.8e-05	7.3e-03	1.1e-02	9.9e-04	2.4e-02
16	1.9e-04	6.0e-02	8.1e-02	6.7e-03	1.8e-01
32	7.8e-04	5.5e-01	6.9e-01	3.2e-02	1.5e+00
64	4.6e-03	5.1e+00	6.2e+00	1.5e-01	1.4e+01
128	3.4e-02	4.7e+01	5.7e+01	8.2e-01	1.3e+02

TABLE 4 Iteration counts, polynomial order $p = 16$, various numbers of elements

	Matrix-free		Assembled
Elements	**Tensor AMS**	**Tensor Jacobi**	**hypre AMS**
2^2	16	48	8
4^2	17	110	9
8^2	17	237	8
16^2	17	491	8
32^2	18	1037	8

To ensure that our preconditioner continues to be scalable with respect to h, we do a set of experiments with fixed $p = 16$ and varying element size h, still on a uniform square grid. These results show that our matrix-free AMS cycle maintains optimality with respect to mesh size h, with details shown in Table 4 and Figure 6. This example is included partially to show that simple Jacobi preconditioning is not a reasonable choice in this regime, even if it works reasonably well for the tiny four element problems previously presented.

6.3 | Varying coefficients

One attractive property of the AMS framework is its algebraic nature, which allows it to be used easily with unstructured grids and varying coefficients α, β. Here we show how our matrix-free AMS cycle behaves with a varying coefficient β. The numerical results here are on a uniform 8×8 grid of squares, with piecewise constant coefficient as described in Figure 7.

In this setting, a single V-cycle on the low-order refined space may not be a good enough preconditioner for the auxiliary space operator. So we perform an auxiliary space solve with a CG iteration on the natural auxiliary space operators

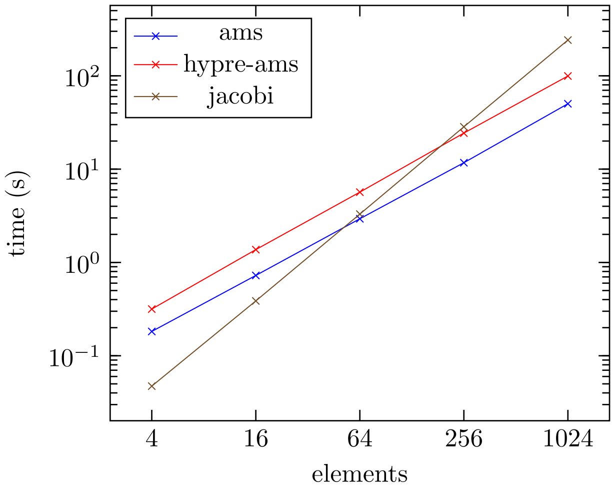


FIGURE 6 Solve time for h-scaling. Both assembled auxiliary Maxwell space (AMS) (hypre-ams) and matrix-free AMS show optimal scaling in this regime, while Jacobi of course does not

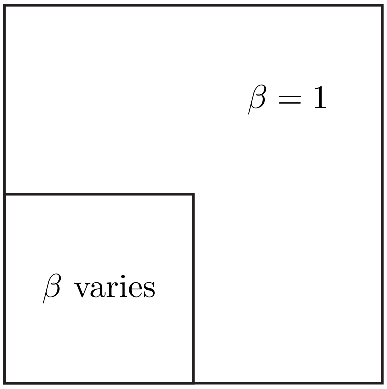


FIGURE 7 Varying piece-wise constant coefficient example setup

TABLE 5 Number of outer CG iterations for example with varying piece-wise constant coefficient

	Contrast in β				
Solver	**1e+00**	**1e+01**	**1e+02**	**1e+04**	**1e+08**
$p = 4$					
hypre	6	8	8	9	9
tensor-1	14	15	18	21	24
tensor-4	5	6	7	9	10
tensor-∞	2	3	3	3	3
$p = 8$					
hypre	9	11	11	11	10
tensor-1	15	16	20	21	33
tensor-4	6	6	10	9	12
tensor-∞	2	3	3	3	3

L_{Π_h}, L_{G_h}, preconditioned with (assembled) AMG built for the low-order refined operators. The results are summarized in Table 5 and Table 6, where "tensor-n" refers to an auxiliary space CG solve with n iterations, where if $n = \infty$ we solve to machine precision for comparison. We see that very high contrast induces some minor difficulty in the auxiliary space solve, but that the overall AMS cycle is still robust under the assumption that the inner solves are accurate. These results underscore the importance of an accurate auxiliary space solve more generally, and that the outer iteration counts in the simpler cases could be improved by iterating or replacing the low-order refined V-cycle at the cost of increased computational time.

TABLE 6 Solve time in seconds for the example with varying piece-wise constant coefficient

	Contrast in β				
Solver	**1e+00**	**1e+01**	**1e+02**	**1e+04**	**1e+08**
$p=4$					
hypre	0.02	0.02	0.03	0.03	0.03
tensor-1	0.07	0.07	0.08	0.09	0.11
tensor-4	0.07	0.06	0.07	0.09	0.10
tensor-∞	0.31	0.40	0.41	0.38	0.39
$p=8$					
hypre	0.33	0.32	0.32	0.32	0.30
tensor-1	0.62	0.60	0.76	0.83	1.22
tensor-4	0.49	0.49	0.77	0.74	0.92
tensor-∞	2.65	3.49	3.51	3.67	3.56

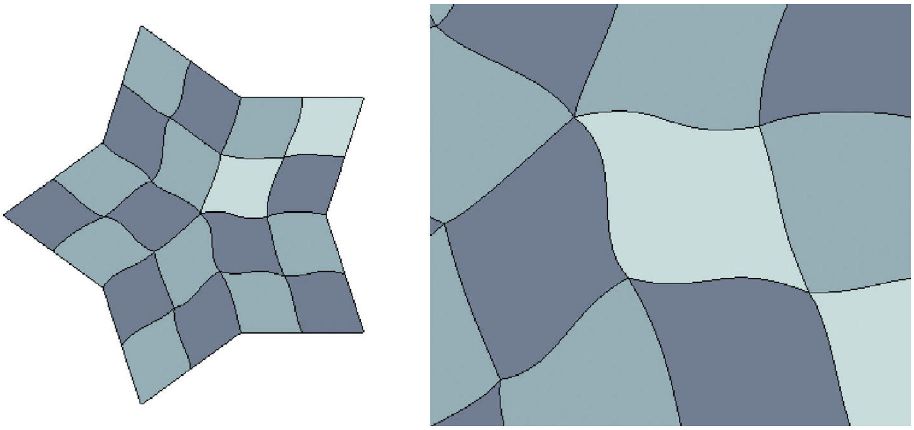


FIGURE 8 A mesh with curved elements, here with a finite element mapping in Q^3

6.4 | Nonuniform and curved meshes

Here we demonstrate the effectiveness of the tensor-product based preconditioner on a mesh with curved elements, as in Figure 8. These are standard finite element spaces where the mapping from the reference element to the physical element is of high polynomial order, and all the tensor manipulations outlined in section 5 go through with only the modification of increasing integration order to account for the higher-order polynomial in the Jacobian. The use of curved element geometries is perhaps unnecessary for our simple model problem, but these type of geometries have important advantages in complicated multiphysics settings, see for example, Reference 9.

We compare the unassembled high-order algorithm outlined above to the fully assembled matrix-based AMS available in Hypre for several polynomial orders on the mesh in Figure 8, with results shown in Table 7. The iteration count is significantly higher for the unassembled version, but solving the problem is still practical here even at very high polynomial order, where assembly becomes a bottleneck in the traditional approach.

7 | CONCLUSION

We have shown the possibility of implementing an auxiliary space Maxwell solver for electromagnetics problems in a matrix-free setting, that is, without explicit assembly of high-order finite element matrices. The resulting solver is practical for very high polynomial order in two dimensions, scales optimally with respect to h, and shows good promise for efficient implementation on next-generation hardware. Getting accurate auxiliary space solves at a cost that scales well

TABLE 7 Results on a curved mesh

	Iterations		Solve (s)		Setup (s)	
Order	**Tensor**	**Hypre**	**Tensor**	**Hypre**	**Tensor**	**Hypre**
2	12	5	0.01	0.01	0.02	0.01
4	14	8	0.03	0.01	0.01	0.01
8	18	14	0.17	0.14	0.01	0.15
16	36	19	2.07	3.16	0.09	6.71
32	90	26	32.19	66.30	0.32	655.61

in p for problems with varying coefficients is a challenge, but given such a H^1 auxiliary space solver the underlying AMS framework is robust to a variety of coefficients, curved meshes, and complex geometries.

ACKNOWLEDGEMENTS

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 and was partially supported by the LLNL-LDRD Program under Project No. 20-ERD-002. This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

CONFLICT OF INTEREST

This work does not have any conflicts of interest.

ORCID

Andrew T. Barker https://orcid.org/0000-0003-3572-911X

REFERENCES

1. Löhner R. Improved error and work estimates for high-order elements. Int J Numer Methods Fluids. 2013;72(11):1207–1218.
2. Kronbichler M, Kormann K, Pasichnyk I, Allalen M. Fast matrix-free discontinuous Galerkin kernels on modern computer architectures. Proceedings of the International Supercomputing Conference. New York, NY: Springer; 2017. p. 237–255.
3. Fischer PF, Min M, Rathnayake T, et al. Scalability of high-performance PDE solvers. Int J High Perform Comput Appl. 2020;34(5):562–586.
4. Brown J. Efficient nonlinear solvers for nodal high-order finite elements in 3D. J Sci Comput. 2010;45(1):48–63.
5. Kronbichler M, Kormann K. A generic interface for parallel cell-based finite element operator application. Comput Fluids. 2012;63:135–147.
6. Kronbichler M, Kormann K, Fehn N, Munch P, Witte J. A Hermite-like basis for faster matrix-free evaluation of interior penalty discontinuous Galerkin operators; 2019. arXiv:190708492.
7. Deville MO, Fischer PF, Mund EH. High-order methods for incompressible fluid flow. Cambridge, MA: Cambridge University Press, 2002.
8. libCEED: code for efficient extensible discretizations [Software]; https://github.com/CEED/libCEED.
9. Dobrev VA, Kolev TV, Rieben RN. High-order curvilinear finite element methods for Lagrangian hydrodynamics. SIAM J Sci Comput. 2012;34(5):B606–B641.
10. Tomov S, Abdelfattah A, Barra V, et al. CEED-MS32 milestone report: performance tuning of CEED software and 1st and 2nd wave applications. Zenodo. Center for efficient exascale discretization (CEED), Exascale computing project (ECP); 2019.
11. hypre: High-performance preconditioners [Software]; computation.llnl.gov/casc/hypre.
12. Henson V, Yang UM. BoomerAMG: a parallel algebraic multigrid solver and preconditioner. Appl Numer Math. 2002;41:155–177.
13. Xu J, Zikatanov L. Algebraic multigrid methods. Acta Numerica. 2017;26:591–721.
14. Ainsworth M, Jiang S. Preconditioning the mass matrix for high order finite element approximation on triangles. SIAM J Numer Anal. 2019;57(1):355–377.

15. Austin TM, Brezina M, Jamroz B, Jhurani C, Manteuffel TA, Ruge J. Semi-automatic sparse preconditioners for high-order finite element methods on non-uniform meshes. J Comput Phys. 2012;231(14):4694–4708.
16. Chalmers N, Warburton T. Low-order preconditioning of high-order triangular finite elements. SIAM J Sci Comput. 2018;40(6):A4040–A4059.
17. Kronbichler M, Ljungkvist K. Multigrid for matrix-free high-order finite element computations on graphics processors. ACM Trans Parallel Comput. 2019;6(1):2.
18. Olson L. Algebraic multigrid preconditioning of high-order spectral elements for elliptic problems on a simplicial mesh. SIAM J Sci Comput. 2007;29(5):2189–2209.
19. Pazner W, Persson PO. Approximate tensor-product preconditioners for very high order discontinuous Galerkin methods. J Comput Phys. 2018;354:344–369.
20. Sundar H, Stadler G, Biros G. Comparison of multigrid algorithms for high-order continuous finite element discretizations. Numer Linear Algebra Appl. 2015;22(4):664–680.
21. Hiptmair R, Xu J. Nodal auxiliary space preconditioning in H(curl) and H(div) spaces. SIAM J Numer Anal. 2007;45:2483–2509.
22. Kolev TV, Vassilevski PS. Parallel auxiliary space AMG solver for H(curl) problems. J Comput Math. 2009;27:604–623.
23. Lottes JW, Fischer PF. Hybrid multigrid/Schwarz algorithms for the spectral element method. J Sci Comput. 2005;24(1):45–78.
24. Orszag SA. Spectral methods for problems in complex geometries. J Comput Phys. 1980;37(1):70–92.
25. Monk P. Finite element methods for Maxwell's equations. Oxford, UK: Oxford University Press, 2003.
26. Nedelec JC. Mixed finite elements in R^3. Numer Math. 1980;35:315–341.
27. Parter SV, Rothman EE. Preconditioning Legendre spectral collocation approximations to elliptic problems. SIAM J Numer Anal. 1995;32(2):333–385.
28. Cleary A, Falgout R, Henson V, et al. Robustness and Scalability of Algebraic Multigrid. SIAM J Sci Comput. 2000;21(5):1886–1908.
29. Mandel J, McCormick S, Ruge J. An algebraic theory for multigrid methods for variational problems. SIAM J Numer Anal. 1988;25(1):91–110.
30. MFEM: modular finite element methods [Software]; https://mfem.org.
31. Anderson R, Andrej J, Barker A, et al. MFEM: a modular finite element library. Comput Math Appl. 2020. https://doi.org/10.1016/j.camwa.2020.06.009.
32. Boonen T, Van Lent J, Vandewalle S. Local Fourier analysis of multigrid for the curl-curl equation. SIAM J Sci Comput. 2008;30(4):1730–1755.
33. Vassilevski PS. Multilevel block factorization preconditioners: matrix-based analysis and algorithms for solving finite element equations. New York, NY: Springer, 2008.

How to cite this article: Barker AT, Kolev T. Matrix-free preconditioning for high-order *H*(curl) discretizations. *Numer Linear Algebra Appl.* 2021;28:e2348. https://doi.org/10.1002/nla.2348