Efficient Parallel Preconditioners for High-Order Finite Element Discretizations of $H(grad)$ and $H(\mathrm{curl})$ Problems

Junxian Wang[1], Shi Shu[1] and Liuqiang Zhong[2]

[1] School of Mathematics and Computational Science, Xiangtan University, Xiangtan 411105, P.R. China, xianxian.student@sina.com; shushi@xtu.edu.cn
[2] School of Mathematics Sciences, South China Normal University, Guangzhou 510631, P.R. China, zhonglq@xtu.edu.cn

Summary. In this paper, we study preconditioning techniques for both $H(grad)$ and $H(\mathbf{curl})$ problems. For an $H(grad)$ elliptic problem discretized by high-order finite elements with a hierachical basis of k-th order, we design and analyse an parallel AMG preconditioner based on a two-level method and a block Gauss–Seidel smoothing technique. For an $H(\mathbf{curl})$ elliptic problem discretized by high-order edge finite elements, we design a parallel solver based on an auxiliary space preconditioner. Numerical experiments show that the number of iteration for the corresponding PCG methods does not depend on the mesh size, and depend weakly on the order of the finite element space and jumps of the coefficients.

Key words: high-order finite element, jump coefficients, parallel preconditioner, block Gauss-Seidel smoother, auxiliary space preconditioner

1 Introduction

Many problems in fields such as computational electromagnetics and computational fluid dynamics can be essentially be transformed into computations of $H(grad)$ and $H(\mathbf{curl})$ elliptic problems. The finite element method is one of the most commonly used numerical methods for solving these kinds of problems. Efficient parallel algorithms are of great importance because the resulting discrete systems are usually large and the corresponding condition numbers grow rapidly with the refinement of the mesh. High-order finite element methods are of great practical interests because of their better approximation to the analytical solution. But the resulting algebraic systems of equations are more difficult to solve than those derived from linear finite elements.

The classical algebraic multigrid (AMG) method is quite efficient for solving the algebraic system resulting from the discretization of $H(grad)$ elliptic problems discretized by linear finite elements. But it does not work well for the systems obtained

by the high-order finite elements. The reason is that the algebraic mesh graph corresponding to the stiffness matrix obtained by the high-order finite element is quite different from the geometric mesh graph. There exist some improved AMG methods for solving the linear system resulting from high-order finite elements ([2, 4, 6, 7]). But, so far, they are mainly restricted to the smooth coefficient case, and the number of iterations depends on the order of the finite element space. For an $H(\mathbf{curl})$ elliptic problems, [3] designed an auxiliary space preconditioner for the algebraic systems resulting from the linear edge elements of the first type of Nédélec elements. Reference [9] further studied the preconditioners for high-order edge finite element discretizations. Recently, the study of parallel algorithms have attracted more and more researchers' attention, and the corresponding numerical softwares has developed rapidly. The high performance parallel preconditioner package HYPRE is one of the most popular numerical software in the world. BoomerAMG and AMS in HYPRE are two efficient solvers for the $H(grad)$ and $H(\mathbf{curl})$ elliptic problems, discretized by linear elements, respectively.

In this paper, we first consider an $H(grad)$ elliptic problem with jump coefficients. Here we discuss the parallel PCG method for solving the algebraic system derived from high-order finite element based on hierachical bases (HBk). Using a two-level method and a block Gauss–Seidel smoothing technique, we have designed an efficient parallel AMG preconditioner and a corresponding PCG method, and implemented them using HYPRE. Then for an $H(\mathbf{curl})$ elliptic problem discretized by high-order edge finite elements, we have designed a parallel PCG algorithm, whose key idea (see [8]) is to change the construction of the parallel preconditioner for the $H(\mathbf{curl})$ system into the design of several preconditioners for the $H(grad)$ system.

The numerical results indicate that these new algorithms have good algorithmic scalability and the number of iterations does not depend on the mesh size, and depend weakly on the order of the finite element space and the jumps of the coefficients.

2 A Parallel Preconditioner for the $H(grad)$ System

Let Ω be a simply connected polyhedron in $\mathbb{R}^3$. We consider the variational problem: Find $u \in H_0^1(\Omega)$, such that

$$a(u, v) = (f, v),\ \forall v \in H_0^1(\Omega), \tag{1}$$

where $a(u,v) = \int_\Omega \beta \nabla u \cdot \nabla v dx$, $(f,v) = \int_\Omega f v dx$, and $\beta(x)$ is a bounded positive function, which may contain large discontinuous jumps, and $f \in L^2(\Omega)$.

We discretize the continuous variational problem (1) with k-th order hierachical basis $\{\psi^k, k = 1, \cdots, n_k\}$ proposed in [1], where $n_k = dim(Z_h^k)$ and Z_h^k is the k-th order Lagrangian finite element space. Furthermore we denote the corresponding algebraic system as

$$A_k u_k = f_k. \tag{2}$$

2.1 A Parallel AMG Preconditioner

Let np be the number of processors, A_j^k, u_j^k and f_j^k be the restrictions of A_k, u_k and f_k on the j-th processor, respectively, and n_j^k be the number of rows of A_j^k. Here, we always assume that the indices of first n_j^1 rows of A_j^k corresponds to the indices of the linear element basis. For the l-th component of u_j^k, we call l the local index on the j-th processor, and let the location in u_k be the global index. We also say that the global index belongs to the j-th processor.

We use a two-level method for solving (2), which translates the solving of linear system discretized by high order finite element into the solving of the one corresponding to the linear finite element. We need to generate the parallel matrix of linear finite element A_1 that consists of elements whose row and column indices is correspond to the linear element basis of A_k.

We change the traditional smoother to the block Gauss–Seidel smoother, since the iteration number of a two level-method with the traditional smoother depends on the order of the finite element space. Next, we generate the needed datum for the block Gauss–Seidel smoother. And we need to introduce some notation related to the j-th processor.

(i) Let $W_j = \{1, \cdots, n_j^k\}$ be the local index set for the j-th processor, and V_j be the corresponding global index set. For any $i \in W_j$, we denote

$$S_i = \{l | A_j^k(i,l) > \theta \max_{m \neq i_g} |A_j^k(i,m)|\}, \tag{3}$$

as the i-th block index set, where $\theta \in [0,1]$, $A_j^k(i,m)$ is an element of A_j^k in row i and column m, i_g is the global index of i.

(ii) Mapping array: $g(i), i = 1, \cdots, n_j^k$ where $g(i) = 0$ *or* i, which indicates whether a block smoothing is needed for the $g(i)$-th block index set.

(iii) The expanded matrix $\tilde{A}_j^k$ on the j-th processor is defined as the submatrix formed by those rows of A_k whose indices belong to $\bigcup_{g(i)\neq 0} S_{g(i)}$. The expanded vector $\tilde{f}_j^k$ can be obtained similarly.

Then we can generate the mapping array $g(i)$ and the block index set $S_{g(i)}$ $(for\ g(i) > 0)$, $i = 1, \cdots, n_j^k$ according to some suitable rule as in Algorithm 1.

Algorithm 1 *(generate the mapping array and the block index set)*

Step 1 *Initialize the mapping array* $g(i) = 0,\ i = 1, \cdots, n_j^k$.

Step 2 *Let* $g(i) = i$, *and generate* $S_{g(i)}$ *according to (3),* $i = 1, \cdots, n_j^1$.

Step 3 *Let* $H = V_j \backslash (V_j \cap \bigcup_{i=1}^{n_j^1} S_{g(i)})$,

Step 4 *For a given* $i \in H$, *find its corresponding local index* $i_l \in W_j$. *Set* $g(i_l) = i_l$, *and generate* $S_{g(i_l)}$.

Step 5 *If* $H := H \backslash (H \cap \bigcup_{g(i)\neq 0} S_{g(i)})$ *is nonempty, then goto Step 4.*

In Algorithm 2, we generate the expanded matrix $\tilde{A}_j^k$ and the expanded vector $\tilde{f}_j^k$ on the j-th processor in order to reduce the communication when the block Gauss–Seidel iteration has been carried out.

Algorithm 2 *(generate $\tilde{A}_j^k$ and $\tilde{f}_j^k$)*

Step 1 *Set $\tilde{A}_j^k := A_j^k$. Let $\tilde{S} = \bigcup_{g(i)\neq 0} S_{g(i)} = V_j \bigcup \tilde{S}_0$, where $\tilde{S}_0$ denotes the set of global indices belonging to those neighboring processors.*

Step 2 *Loop through all the neighboring processors, let $\tilde{S}_0^m$ be the set of those element in $\tilde{S}_0$ which is on the current neighboring processor m.*

2.1 *Form $A_m^0(f_m^0)$ by those rows of $A_m^k(f_m^k)$ whose global indices belongs to $\tilde{S}_0^m$. Then send $A_m^0(f_m^0)$ to processor j.*

2.2 *On processor j, receive $A_m^0(f_m^0)$ and append to $\tilde{A}_j^k(\tilde{f}_j^k)$.*

Step 3 *Generate the mapping array $p(n), n = 1, \cdots, |\tilde{S}|$, which is from the set of row numbers of $\tilde{A}_j^k$ to its corresponding global index set.*

Now, using $\tilde{A}_j^k$ and $\tilde{f}_j^k$, we can generate all the block diagonal matrices $A_{j,k}^{l,d}$, nondiagonal matrices $A_{j,k}^{l,nd}$ and right hand side vectors $f_{j,k}^{l,d}$ on processor j when $g(l) \neq 0$. Then, we construct the parallel block Gauss–Seidel smoothing algorithm as follows.

Algorithm 3 *(block Gauss–Seidel iteration)*

Assume that $\tilde{u}_j^k$ is the result from the block Gauss–Seidel iteration on the current processor j. Let S_{max} be the maximum iteration number.

Step 1 *Let the initial guess be $\tilde{u}_j^k = 0$ and $sm = 0$.*

Step 2 *Let $sm = sm+1$. If $sm > 1$, then update $\tilde{u}_j^k$ for those components belonging to its neighboring processors.*

Step 3 *For any $l \in \{1, \cdots, n_j^k\}$ with $g(l) \neq 0$, solve $A_{j,k}^{l,d} u_{j,k}^{l,d} = f_{j,k}^{l,d} - A_{j,k}^{l,nd} \tilde{\tilde{u}}_j^k$ for $u_{j,k}^{l,d}$, where $\tilde{\tilde{u}}_j^k$ is a zero extension of $\tilde{u}_j^k$, and update $\tilde{u}_j^k$ by $u_{j,k}^{l,d}$.*

Step 4 *If $sm < S_{max}$, then goto step 2.*

Step 5 *Use those components of $\tilde{u}_j^k$, whose indices belong to processor j, to form the vector $\tilde{u}_j^{k,s}$, then gather $\tilde{u}_j^{k,s}$ to form the parallel solution $\tilde{u}_k^{(s)}$.*

Based on the above block Gauss–Seidel smoother, we develop the corresponding parallel two-level method in Algorithm 4 for solving (2). Let it_{max} denote the maximum iteration number, and tol denote the stopping criteria.

Algorithm 4 *(Two-Level method)*

Step 1 *For a given initial guess $u_k^{(0)}$, use HYPRE to get an initial residual vector $r_k^{(0)} = f_k - A_k u_k^{(0)}$, and let $res0 = \|r_k^{(0)}\|$, $l = 1$.*

Step 2 *Presmoother: Use Algorithm 3 for (2) with $S_{max} = m_1$ and the initial guess $u_k^{(l-1)}$ to get the parallel iterative vector u_t.*

Step 3 *Correction*

3.1 Use HYPRE to get the vector $r_k = f_k - A_k u_t$*, and get the parallel vector* f_1 *from the first* n_j^1 *components of* r_k *on the* j*-th processor.*

3.2 Use BoomerAMG to solve $A_1 e = f_1$*, and obtain the solution* e_t*.*

3.3 Add e_t *and* u_t *to get a new parallel vector* $\tilde{u}_t$*.*

Step 4 Postsmoother: Use Algorithm 3 for (2) with $S_{max} = m_1$ *and initial guess* $\tilde{u}_t$ *to get the parallel iterative vector* $u_k^{(l)}$*.*

Step 5 Use HYPRE to get $res = \|f_k - A_k u_k^{(l)}\|$*. If* $\frac{res}{res0} < tol$ *or* $l = it_{max}$*, then exit. Otherwise, set* $l := l + 1$*, and goto step2.*

Combining the above algorithms, we obtain a new parallel AMG method for solving the system (2) resulting from the $H(grad)$ problem.

In Algorithm 4, by setting $it_{max} = 1$, we can obtain a TLB-AMG-p preconditioner for solving (2), and denote the PCG method using TLB-AMG-p as a preconditioner for TLB-AMG-CG-p.

2.2 Numerical Experiments

Consider the model problem (1) with $\Omega = (0,1)^3$, $\beta = \beta_0$ in Ω_J and $\beta = 1$ in $\Omega \backslash \Omega_J$, where β_0 is a positive constant, and Ω_J is a subdomain of Ω.

We decompose Ω into $n \times n \times n$ hexahedra $\Omega_k (k = 1, \cdots, n^3)$. All the hexahedra constitute the desired domain decomposition. To get the final triangulation of Ω, we decompose each hexahedra mentioned above into $m \times m \times m$ smaller hexahedra and divide each smaller hexahedra into 6 tetrahedra. All the tetrahedra constitute the triangulation $\mathcal{T}_h = n(m)$ of the domain Ω.

We like to remark that each subdomain Ω_k corresponds to one processor.

Example 1. (A floating subdomain case) Let $n = 3$ and $\Omega_J = (\frac{1}{3}, \frac{2}{3})^3$.

Example 2. (No floating subdomain case) Let $n = 2$ and $\Omega_J = (0, \frac{1}{2})^3$.

We apply TLB-AMG-CG-p to solve the system (2) corresponding to Example 1. In our test, we set $S_{max} = 2$ in Algorithm 3, one V-cycle in BoomerAMG, $\theta = 0.0$ in (3).

Both in Tables 1 and 2, the data below " k" is the order of finite element basis, the " $n(m)$" row denotes the tetrahedron meshes, the datum in the row of " β_0" are the values of the coefficient β in Ω_J.

The numerical results for Example 1 are presented in Table 1. A relative reduction of the preconditioned norm of the residual vector by a factor of 10^{-8} was used as termination criterion.

Table 1. No. of iterations for Example 1 with TLB-AMG-CG-p.

$n(m)$	3(4)			3(6)		
β_0	10^{-5}	1	10^5	10^{-5}	1	10^5
$k=2$	4	4	5	4	5	6
$k=3$	5	5	6	5	5	6
$k=4$	5	6	9	5	5	9

From Table 1, we find that the iteration number of TLB-AMG-CG-p does not depend on the mesh size, depends weakly on the order of basis function and the jumps of the coefficients for a floating subdomain case.

In Table 2, we present the numerical results obtained by BoomerAMG-CG(with BoomerAMG as a preconditioner) for Example 2.

Table 2. No. of iterations for Example 2 with BoomerAMG-CG.

$n(m)$	$2(4)$			$2(6)$		
β_0	1	10^3	10^5	1	10^3	10^5
$k=2$	65	101	88	72	43	42
$k=3$	16	16	16	15	15	15
$k=4$	29	28	27	27	25	25
$k=5$	>200	>200	>200	>200	>200	>200

In view of Table 2, we find that BoomerAMG-CG is sensitive to the mesh size, the order of basis functions and the jumps of the coefficients for the non-floating subdomain case. So our TLB-AMG-CG-p is more robust and efficient.

3 A Parallel Preconditioner for the $H(\mathrm{curl})$ Problem

Consider the variational problem: Find $\boldsymbol{u} \in H_0(\mathbf{curl}; \Omega)$, such that

$$a(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{f}, \boldsymbol{v}),\ \forall \boldsymbol{v} \in H_0(\mathbf{curl}; \Omega), \tag{4}$$

where $a(\boldsymbol{u}, \boldsymbol{v}) = \int_\Omega (\alpha \nabla \times \boldsymbol{u} \cdot \nabla \times \boldsymbol{v} + \beta \boldsymbol{u} \cdot \boldsymbol{v}) dx,\ (\boldsymbol{f}, \boldsymbol{v}) = \int_\Omega \boldsymbol{f} \cdot \boldsymbol{v} dx, \alpha(\boldsymbol{x}), \beta(\boldsymbol{x})$ are two bounded positive functions, and $\boldsymbol{f} \in (L^2(\Omega))^3$.

Use the finite element space $\boldsymbol{V}_h^{k,j} (k \geq 1, j = 1, 2)$ for problem (4), where $\boldsymbol{V}_h^{k,1}$ and $\boldsymbol{V}_h^{k,2}$ are the k^{th}-order element spaces of the first family and second family of Nédélec elements, respectively(see [5]), then we obtain a linear system

$$A_h^{k,j} U_h^{k,j} = F_h^{k,j}. \tag{5}$$

Below we develop a parallel PCG algorithm for solving (5).

3.1 A Parallel Preconditioner for (5)

Let $B_h^{k,j}$ be the HX-ho preconditioner for $A_h^{k,j}$ (see [8]). As an example, we take $k = 1$ and $j = 2$. Let $I_h^l : \boldsymbol{V}_h^{1,1} \mapsto Z_{h,l} (l = 1, 2, 3)$, $I_h^{g,2} : \boldsymbol{V}_h^{1,2} \mapsto \nabla Z_h^2$ be the corresponding interpolation matrices where $(Z_h^1)^3 := \sum_{l=1}^3 Z_{h,l}$. Given a vector y, we can give an algorithm for solving $w = B^{1,2} y$ as Algorithm 5.

Algorithm 5 *(HX-ho preconditioner)*

Step 1 (Setup):

1.1 Form (in parallel) matrices I_h^l, $I_h^{g,2}$ and get the matrix $A^{1,1}$ with $A^{1,2}$;

1.2 Generate matrices $A_h^l = I_h^l A^{1,1}(I_h^l)^T, l = 1,2,3$ *and* $A_h^{g,2} = I_h^{g,2} A^{1,2} (I_h^{g,2})^T$.

Step 2 (Solve):

2.1 Use zero as an initial guess, and apply m *times parallel symmetric Gauss–Seidel iteration for* $A^{1,2}\tilde{u} = y$ *to obtain the solution* $\tilde{u}$*;*

2.2 Generate a vector y_1 *by extracting those* y *components which belong to* $\boldsymbol{V}_h^{1,1}$*. Compute* $u_l = B_h^l I_h^l y_1 (l = 1,2,3)$ *where* B_h^l *can be chosen as the TLB-AMG-p* ($k = 1$) *preconditioner of* A_h^l*, calculate the vector* u_l' *by zero extension of* $(I_h^l)^T u_l$*;*

2.3 Compute $\tilde{u}_{g,2} = B_h^{g,2} I_h^{g,2} y$ *where* $B_h^{g,2}$ *can be chosen as the TLB-AMG-p* ($k = 2$) *preconditioner of* $A_h^{g,2}$*;*

2.4 Construct $w = \tilde{u} + \sum_{i=1}^{3} u_l' + (I_h^{g,2})^T \tilde{u}_{g,2}$.

We denote HX-ho-CG-p for the PCG method using Algorithm 5 as a preconditioner. Below we present some examples to show the efficiency and robustness of HX-ho-CG-p.

3.2 Numerical Results

We consider the model problem (4) with $\Omega = (0,1)^3$, $\alpha = \beta = \beta_0$ in Ω_J and $\alpha = \beta = \beta_0$ in $\Omega \backslash \Omega_J$, where β_0 is a positive constant, and $\Omega_J \subset \Omega$.

Example 3. (No floating subdomain case) $\Omega_J = (0, \frac{1}{3})^3, \beta_0 = 10^5$.

Example 4. (A floating subdomain case) $\Omega_J = (\frac{1}{3}, \frac{2}{3})^3, \beta_0 = 10^5$.

The numerical results for the two examples are reported in Table 3, where the datum in the column "np" denote the number of processors, the number $n(m)$ denote the tetrahedral meshes, and the datum left in the column of "$n(m)$" are the corresponding iteration number of HX-ho-CG-p.

A relative reduction of the preconditioner norm of the residual vector by a factor of 10^{-6} was used as termination criterion in HX-ho-CG-p. In our test, we set $m = 3$ in Algorithm 5 and the block Gauss–Seidel iteration is changed to 15 Jacobi iterations with weight $\omega = 0.45$ as a smoother in TLB-AMG-p.

Table 3. No. of iterations for the two examples with HX-ho-CG-p.

np	Example 3			Example 4		
	3(2)	3(4)	3(8)	3(2)	3(4)	3(8)
1	15	16	16	19	21	19
4	16	17	17	24	22	21
8	17	17	17	29	22	21

From Table 3, we find that the new algorithm HX-ho-CG-p has good algorithmic scalability and that the number of iterations is independent of the mesh size, and weakly dependent on the jumps of the coefficients.

Acknowledgments The authors are partially supported by the National Natural Science Foundation of China(Grant No. 10771178), the National Basic Research Program of China(Grant No. 2005CB321702), Key Project of Chinese Ministry of Education(Grant No. 208093). The first author is supported by Hunan Provincial Innovation Foundation For Postgraduate(Grant No. CX2009B121)

References

1. M. Ainsworth and J. Coyle. Hierarchic finite element bases on unstructured tetrahedral meshes. *Int. J. Numer. Methods Eng.*, 58:2103–2130, 2003.
2. J.J. Heys, T.A. Manteuffel, S.F. McCormick, and L.N. Olson. Algebraic multigrid for higher-order finite elements. *J. Comput. Phys.*, 204(2):520–532, 2005.
3. R. Hiptmair and J. Xu. Nodal auxiliary spaces preconditions in $H(\mathbf{curl})$ and $H(div)$ spaces. *SIAM J. Numer. Anal.*, 45(6):2483–2509, 2007.
4. Y. Huang, S. Shu, and X. Yu. Preconditioning higher order finite element systems by algebraic multigrid method of linear elements. *J. Comput. Math.*, 24(5):657–664, 2006.
5. P. Monk. *Finite Element Methods for Maxwell Equations*. Numerical Mathematics and Scientific Computation. Oxford University Press, Oxford, Colorado, USA 2003.
6. J. Ruge. AMG for higher-order discretizations of second-order elliptic problems. In *Eleventh Copper Mountain Conference on Multigrid Methods*, 2003.
7. S. Shu, D. Sun, and J. Xu. An algebraic multigrid method for higher-order finite element discretizations. *Computing*, 77(4):347–377, 2006.
8. J. Xu, L. Chen, and R. Nochetto. *Optimal multilevel methods for $H(grad)$, $H(\mathbf{curl})$ and $H(div)$ systems on graded and unstructured grids*. Summer School, Peking University, July 13, 2009–August 8, 2009, 2009.
9. L. Zhong, S. Shu, D. Sun, and L. Tan. Preconditioners for higher order edge finite element discretizations of Maxwell's equations. *Sci. China Ser. A*, 51(8):1537–1548, 2008.