Preconditioned Eigensolver LOBPCG in *hypre* and PETSc

Ilya Lashuk, Merico Argentati, Evgueni Ovtchinnikov, and Andrew Knyazev

Department of Mathematics, University of Colorado at Denver, P.O. Box 173364, Campus Box 170, Denver, CO 80217, USA.
{ilashuk,rargenta,eovtchin}@math.cudenver.edu,
andrew.knyazev@cudenver.edu

We present preliminary results of an ongoing project to develop codes of the Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) method for symmetric eigenvalue problems for *hypre* and PETSc software packages. *hypre* and PETSc provide high quality domain decomposition and multigrid preconditioning for parallel computers. Our LOBPCG implementation for *hypre* is publicly available in *hypre* 1.8.2b and later releases. We describe the current state of the LOBPCG software for *hypre* and PETSc and demonstrate scalability results on distributed memory parallel clusters using domain decomposition and multigrid preconditioning.

1 Introduction

We implement a parallel algorithm of the Locally Optimal Block Preconditioned Conjugate Gradient Method (LOBPCG) [5, 6] for the solution of eigenvalue problems $Ax = \lambda Bx$ for large sparse symmetric matrices A and $B > 0$ on massively parallel computers for the High Performance Preconditioners (*hypre*) [3] and Portable, Extensible Toolkit for Scientific Computation (PETSc) [2] software libraries. Software for the Preconditioned Eigensolvers is available at http://math.cudenver.edu/~aknyazev/software/CG/ which contains, in particular, our MATLAB and *hypre* codes of LOBPCG. Our native *hypre* LOBPCG version efficiently takes advantage of powerful *hypre* algebraic and geometric multigrid preconditioners. Our native PETSc LOBPCG version gives the PETSc users community an easy access to a customizable code of a high quality modern preconditioned eigensolver.

The LOBPCG method has recently attracted attention as a potential competitor to the Lanczos and Davidson methods due to its simplicity, robustness

and fast convergence. C++ (by R. Lehoucq, U. Hetmaniuk et al. [1, 4], will appear in Anasazi Trilinos), FORTRAN 77 (by Randolph Bank, part of PLTMG 9.0 and above) and FORTRAN 90 (by G. Zèrah, part of ABINIT v4.5 and above, complex Hermitian matrices) implementations of the LOBPCG are being developed by different groups in such application areas as structural mechanics, mesh partitioning and electronic structure calculations.

2 Abstract LOBPCG implementation for *hypre*/**PETSc**

For computing only the smallest eigenpair, we take the block size $m = 1$ and then the LOBPCG gets reduced to a local optimization of a 3-term recurrence:
$x^{(i+1)} = w^{(i)} + \tau^{(i)}x^{(i)} + \gamma^{(i)}x^{(i-1)}$,
$w^{(i)} = T(Ax^{(i)} - \lambda^{(i)}Bx^{(i)})$, $\lambda^{(i)} = \lambda(x^{(i)}) = (x^{(i)}, Ax^{(i)})/(Bx^{(i)}, x^{(i)})$
with properly chosen scalar iteration parameters $\tau^{(i)}$ and $\gamma^{(i)}$. The easiest and most efficient choice of parameters is based on an idea of *local optimality* [5, 6], namely, select $\tau^{(i)}$ and $\gamma^{(i)}$ that minimize the Rayleigh quotient $\lambda(x^{(i+1)})$ by using the Rayleigh–Ritz method. For finding m smallest eigenpairs the Rayleigh–Ritz method on a $3m$–dimensional trial subspace is used during each iteration for the local optimization.

LOBPCG description in [6] skips important details. The complete description of the LOBPCG algorithm as it has been implemented in MATLAB code rev. 4.10 and the *hypre* code 1.9.0b follows:

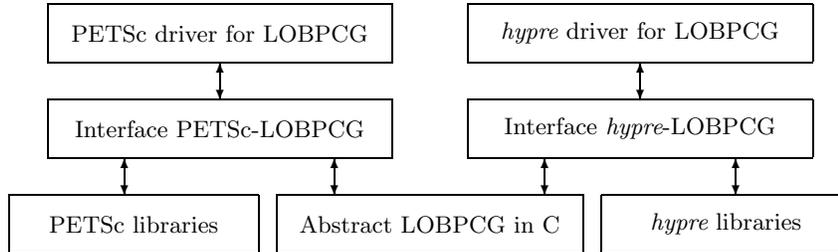**Input:** m starting linearly independent multivectors in $X \in \mathbb{R}^{n \times m}$,
   l linearly independent constraint multivectors in $Y \in \mathbb{R}^{n \times l}$, devices to
   compute $A * X$, $B * X$ and $T * X$.
 1. Allocate memory for ten multivectors
     $W, P, Q, AX, AW, AP, BX, BW, BP, BY \in \mathbb{R}^{n \times m}$.
 2. Apply constraints to X:
     $BY = B * Y$; $X = X - Y * (Y^T * BY)^{-1} * X^T * BY$.
 3. B-orthonormalize X: $BX = B * X$; $R = \text{chol}(X^T * BX)$; $X = X * R^{-1}$;
     $BX = BX * R^{-1}$; $AX = A * X$. ("chol" is the Cholesky decomposition)
 4. Compute the initial Ritz vectors: solve the eigenproblem
     $(X^T * AX) * TMP = TMP * \Lambda$;
     and compute $X = X * TMP$; $AX = AX * TMP$; $BX = BX * TMP$.
 5. Define index set I to be $\{1, \dots, m\}$
 6. **for** $k = 0, \dots, MaxIterations$
 7.   Compute residuals: $W_I = AX_I - BX_I * \Lambda_I$.
 8.   Exclude from index set I those indices which correspond to residual
        vectors for which the norm became smaller than the tolerance.
        If I after that became an empty set, then exit loop.
 9.   Apply preconditioner T to the residuals: $W_I = T * W_I$.
 10   Apply constraints to preconditioned residuals W_I:
        $W_I = W_I - Y * (Y^T * BY)^{-1} * W_I^T * BY$.

11. B-orthonormalize W_I: $BW_I = B * W_I$; $R = \text{chol}(W_I^T * BW_I)$;
       $W_I = W_I * R^{-1}$; $BW_I = BW_I * R^{-1}$.
12.   Compute AW_I: $AW_I = A * W_I$.
13.   **if** $k > 0$
14. B-orthonormalize P_I: $R = \text{chol}(P_I^T * BP_I)$; $P_I = P_I * R^{-1}$;
15.   Update $AP_I = AP_I * R^{-1}$; $BP_I = BP_I * R^{-1}$.
16.   **end if**
       Perform the Rayleigh Ritz Procedure:
           Compute symmetric Gram matrices:
17.   **if** $k > 0$

18. $gramA = \begin{bmatrix} \Lambda & X^T * AW_I & X^T * AP_I \\ \cdot & W_I^T * AW_I & W_I^T * AP_I \\ \cdot & \cdot & P_I^T * AP_I \end{bmatrix}$.

19. $gramB = \begin{bmatrix} I & X^T * BW_I & X^T * BP_I \\ \cdot & I & W_I^T * BP_I \\ \cdot & \cdot & I \end{bmatrix}$.

20.   **else**

21. $gramA = \begin{bmatrix} \Lambda & X^T * AW_I \\ \cdot & W_I^T * AW_I \end{bmatrix}$.

22. $gramB = \begin{bmatrix} I & X^T * BW_I \\ \cdot & I \end{bmatrix}$.

23.   **end if**
24.   **Solve the generalized eigenvalue problem:**
           $gramA * Y = gramB * Y * \Lambda$, where the first m eigenvalues in
           increasing order are in the diagonal matrix Λ and the corresponding
           $gramB$-orthonormalized eigenvectors are columns of Y.
           Compute Ritz vectors:
25.   **if** $k > 0$

26.   Partition $Y = \begin{bmatrix} Y_X \\ Y_W \\ Y_P \end{bmatrix}$ according to the number of columns in
               X, W_I, and P_I, respectively.
27.   Compute $P = W_I * Y_W + P_I * Y_P$;
               $AP = AW_I * Y_W + AP_I * Y_P$; $BP = BW_I * Y_W + BP_I * Y_P$.
28. $X = X * Y_X + P$; $AX = AX * Y_X + AP$; $BX = BX * Y_X + BP$.
29.   **else**

30.   Partition $Y = \begin{bmatrix} Y_X \\ Y_W \end{bmatrix}$ according to the number of columns in
               X and W_I respectively.
31. $P = W_I * Y_W$; $AP = AW_I * Y_W$; $BP = BW_I * Y_W$.
32. $X = X * Y_X + P$; $AX = AX * Y_X + AP$; $BX = BX * Y_X + BP$.
33.   **end if**
37. **end for**
Output: Eigenvectors X and eigenvalues Λ.

The LOBPCG eigensolver code is written in C-language and calls a few LAPACK subroutines. The matrix–vector multiply and the preconditioner call are done through user supplied functions. The main LOBPCG code is abstract in the sense that it works only through an interface that determines the particular software environment: *hypre* or PETSc, in order to call parallel (multi)vector manipulation routines.

A block diagram of the high-level software modules is given in Figure 1.

PETSc driver for LOBPCG		*hypre* driver for LOBPCG
Interface PETSc-LOBPCG		Interface *hypre*-LOBPCG
PETSc libraries	Abstract LOBPCG in C	*hypre* libraries

Fig. 1. LOBPCG *hypre*/PETSc software modules

hypre supports four conceptual interfaces: Struct, SStruct, FEM and IJ. At present, LOBPCG has been tested with all but the FEM interface. *hypre* test drivers for LOBPCG are simple extensions of the *hypre* test drivers for linear system. We anticipate that both types of drives will be merged in the post 1.9.0b *hypre* release.

We do not use shift-and-invert strategy. Preconditioning is implemented directly as well as through calls to the *hypre*/PETSc preconditioned conjugate gradient method (PCG). Specifically, in the latter case the action $x = Tb$ of the preconditioner T on a given vector b is performed by calling a few steps of PCG to solve $Ax = b$.

LOBPCG-*hypre* has been tested with all available *hypre* PCG-capable preconditioners in Struct, SStruct and IJ interfaces, most notably, with IJ AMG–PCG algebraic multigrid, IJ DS–PCG diagonal scaling, IJ additive Schwarz–PCG, and Struct PFMG-PCG geometric multigrid. LOBPCG-PETSc has been tested with PETSc native Additive Schwarz and PETSc linked IJ AMG from *hypre*.

3 *hypre*/**PETSc LOBPCG Numerical Results**

3.1 Basic Accuracy of Algorithm

In these tests LOBPCG computes the smallest 50 eigenvalues of 3D 7–Point $200 \times 200 \times 200$ and $200 \times 201 \times 202$ Laplacians. In the first case we have eigenvalues with multiplicity and in the second case the eigenvalues are distinct, but clustered. The initial eigenvectors are chosen randomly. We set the stopping tolerance (the norm of the maximum residual) equal to 10^{-6}. The numerical output and exact eigenvalues are compared. In both cases for all eigenvalues the maximum relative error is less than 10^{-8} and the Frobenius norm $\|V^T V - I_{m \times m}\| < 10^{-12}$, where $V \in \mathbb{R}^{n \times m}$ contains the approximate eigenvectors. These tests suggest that LOBPCG is cluster robust, i.e. it does not miss (nearly) multiple eigenvalues.

In some tests, the LOBPCG code becomes instable because of ill-conditioned Gram matrices, which is typically a result of bad initial guesses, e.g., generated by a poor quality random number generator. When the ill-conditioning appears restarts are helpful. The simplest restart is to drop the matrix P from the basis of the trial subspace. Such restarts improve the stability of the LOBPCG code as observed in MATLAB tests, and are planned to be implemented in a future *hypre*/PETSc LOBPCG revision.

3.2 Performance Versus the Number of Inner Iterations

Let us remind the reader that we can execute a preconditioner $x = Tb$ directly or by calling PCG to solve $Ax = b$. We do not attempt to use shift-and-invert strategy, but instead simply take T to be a preconditioner for A. Therefore, we can expect that increasing the number of "inner" iterations of the PCG might accelerate the overall convergence, but only if we do not make too many iterations. In other words, for a given matrix A and a particular choice a preconditioner, there should be an optimal finite number of inner iterations.
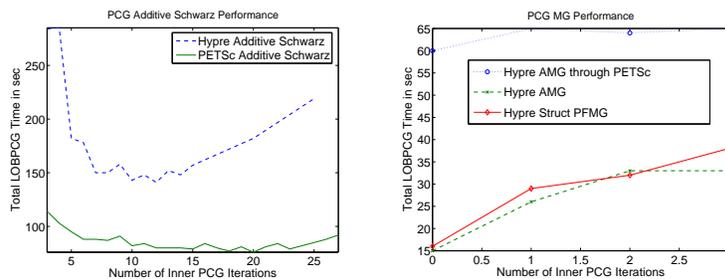
Fig. 2. Performance versus the number of inner iterations. 7–Point 3-D Laplacian, 1,000,000 unknowns. Dual 2.4-GHz Xeon 4GB.

In numerical example illustrated on Figure 2, we try to find this optimal number for the Schwarz–PCG and AMG-PCG preconditioners in *hypre* and PETSc. We measure the execution time as we vary the quality of the preconditioner by changing the maximum number of inner iterations in the corresponding PCG solver. We find that on this problem the optimal number of inner iterations is approximately $10-15$ for Schwarz-PCG, but AMG-PCG works best if AMG is applied directly as a preconditioner, without even initializing the AMG-PCG function.

Our explanation of this behavior is based on two facts. First, the Schwarz method is somewhat cheaper, but not of such a good quality, compared to AMG in these tests. Moreover, the costs for matrix vector multiplies and multivector linear algebra in LOBPCG is a relatively small proportion of the AMG application, but comparable to the computational cost of Schwarz here. Second, one PCG iteration is less computationally expensive compared to one LOBPCG iteration because of larger number of linear algebra operations with multivectors in the latter. A single direct application of AMG as the preconditioner in LOBPCG gives enough improvement in convergence to make it the best choice, while Schwarz requires more iterations that are less time consuming if performed through PCG, rather than by direct application in LOBPCG.

3.3 LOBPCG Performance vs. Block Size

We test both *hypre* and PETSc LOBPCG codes on a 7–Point 3-D Laplacian with 2,000,000 unknowns with *hypre* AMG Preconditioner on Sun Fire 880, 6 CPU 24GB system by increasing the block size m, i.e. the number of computed eigenvectors, from 1 to 16. We observe that the growth of the total CPU time with the increase of the block size is linear, from approximately 100 sec for $m = 1$ to 2,500 sec for $m = 16$. We expect that for larger m the complexity term $m^2 n$ becomes visible. We note, however, that neither *hypre* nor PETSc currently has efficiently implemented multivectors, e.g., in the current implementation the number of MPI communications in computation of the Gram matrices grows with m. An efficient implementation of main multivector functions is crucial in order to significantly reduce the overall costs for large m.

3.4 Scalability with the Schwarz–PCG and Multigrid–PCG preconditioners

We test scalability by varying the problem size so it is proportional to the number of processors. We use a 7–Point 3–D Laplacian and set the block size $m = 1$.

For the Schwarz–PCG, we set the maximum number of inner iterations of the PCG to 10. The tests are performed on the Beowulf cluster at CU Denver that includes 36 nodes, two PIII 933MHz processors and 2GB memory per

node, running Linux RedHat and a 7.2SCI Dolpin interconnect and on MCR cluster (dual Xeon 2.4-GHz, 4 GB nodes) at LLNL. In all these tests, the time per iteration is reasonably scalable, but the number of LOBPCG iterations grows with the problem size i.e., the Schwarz–PCG preconditioner in *hypre* and in PETSc is not optimal in this case.

For the Multigrid–PCG preconditioners, we apply the preconditioners directly, without calling the PCG. We test here *hypre* IJ AMG–PCG algebraic multigrid, *hypre* Struct PFMG-PCG geometric multigrid and PETSc linked IJ AMG from *hypre* on LLNL MCR cluster, see Figure 3.4 left.
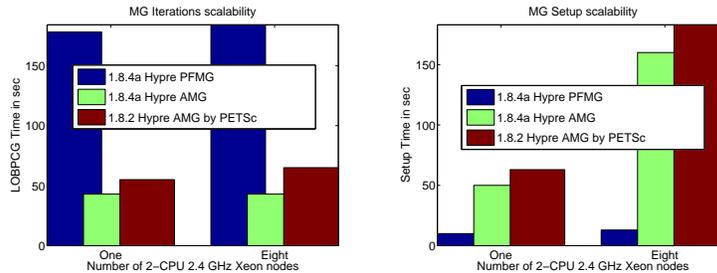
Fig. 3. 7–Point Laplacian, 2,000,000 unknowns per node. Preconditioners: AMG and PFMG. System: LLNL MCR. LOBPCG scalability (left) and preconditioner setup (right).

Good LOBPCG scalability can be seen on Figure 3.4 left. The Struct PFMG takes more time compared to AMG here because of the larger convergence factor. To satisfy the reader curiosity, we also provide the scalability data for the preconditioner setup on Figure 3.4 right.

Conclusions

- We present apparently the world's first parallel code for generalized symmetric definite eigenvalue problems, that can apply the preconditioning directly. The LOBPCG is our method of choice for the preconditioned eigensolver because of its simplicity, robustness and fast convergence.
- Our *hypre*/PETSc LOBPCG code illustrates that the LOBPCG "matrix-free" algorithm can be successfully implemented using parallel libraries that are designed to run on a great variety of multiprocessor platforms.
- In problems we tested with AMG preconditioning, 90%–99% of the computational effort is required for the preconditioner setup and in the applying the preconditioner and thus the LOBPCG scalability is mainly dependent on the scalability of *hypre*/PETSc preconditioning. Initial scalability measurements look promising, but more testing is needed by other users.

- The LOBPCG *hypre* software has been integrated into the *hypre* software at LLNL and has been publicly released in *hypre*–1.8.2b and above, and so is available for users to test.

References

1. Peter Arbenz, Ulrich L. Hetmaniuk, Richard B. Lehoucq, and Raymond S. Tuminaro. A comparison of eigensolvers for large-scale 3D modal analysis using AMG-preconditioned iterative methods. *Int. J. Numer. Meth. Engng.*, 2005. Also available as Sandia National Laboratories Technical report SAND 2005-0282J.
2. Satish Balay, Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, 2004.
3. Robert D. Falgout, Jim E. Jones, and Ulrike Meier Yang. Pursuing scalability for *hypre*'s conceptual interfaces. *ACM Transactions on Mathematical Software*, 2005. For TOMS special issue on the ACTS Collection.
4. Michael A. Heroux, Roscoe A. Bartlett, Vicki E. Howle, Robert J. Hoekstra, Jonathan J. Hu, Tamara G. Kolda, Richard B. Lehoucq, Kevin R. Long, Roger P. Pawlowski, Eric T. Phipps, Andrew G. Salinger, Heidi K. Thornquist, Ray S. Tuminaro, James M. Willenbring, Alan Williams, and Kendall S. Stanley. An overview of the trilinos project. *ACM Transactions on Mathematical Software*, 2005. For TOMS special issue on the ACTS Collection.
5. A. V. Knyazev. Preconditioned eigensolvers: practical algorithms. In Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, editors, *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide*, pages 352–368. SIAM, Philadelphia, 2000.
6. A. V. Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM J. Sci. Comput.*, 23(2):517–541, 2001.