SIAM J. SCI. COMPUT.
Vol. 29, No. 5, pp. 2224–2239

BLOCK LOCALLY OPTIMAL PRECONDITIONED EIGENVALUE XOLVERS (BLOPEX) IN HYPRE AND PETSC[*]

A. V. KNYAZEV[†], M. E. ARGENTATI[†], I. LASHUK[†], AND E. E. OVTCHINNIKOV[‡]

Abstract. We describe our software package Block Locally Optimal Preconditioned Eigenvalue Xolvers (BLOPEX) recently publicly released. BLOPEX is available as a stand-alone serial library, as an external package to PETSc (Portable, Extensible Toolkit for Scientific Computation, a general purpose suite of tools developed by Argonne National Laboratory for the scalable solution of partial differential equations and related problems), and is also built into *hypre* (High Performance Preconditioners, a scalable linear solvers package developed by Lawrence Livermore National Laboratory). The present BLOPEX release includes only one solver—the Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) method for symmetric eigenvalue problems. *hypre* provides users with advanced high-quality parallel multigrid preconditioners for linear systems. With BLOPEX, the same preconditioners can now be efficiently used for symmetric eigenvalue problems. PETSc facilitates the integration of independently developed application modules, with strict attention to component interoperability, and makes BLOPEX extremely easy to compile and use with preconditioners that are available via PETSc. We present the LOBPCG algorithm in BLOPEX for *hypre* and PETSc. We demonstrate numerically the scalability of BLOPEX by testing it on a number of distributed and shared memory parallel systems, including a Beowulf system, SUN Fire 880, an AMD dual-core Opteron workstation, and IBM BlueGene/L supercomputer, using PETSc domain decomposition and *hypre* multigrid preconditioning. We test BLOPEX on a model problem, the standard 7-point finite-difference approximation of the 3-D Laplacian, with the problem size in the range of 10^5–10^8.

Key words. conjugate gradient, iterative method, preconditioning, multigrid, domain decomposition, parallel computing, eigenvalue, LOBPCG, BLOPEX, *hypre*, PETSc, BlueGene, Beowulf

AMS subject classifications. 65F15, 65N25, 65N55, 65Y05

DOI. 10.1137/060661624

1. Introduction.

We describe a new software package, Block Locally Optimal Preconditioned Eigenvalue Xolvers (BLOPEX) revision 1.0, recently publicly released. BLOPEX is available as a stand-alone serial library, which can be downloaded from http://math.cudenver.edu/~aknyazev/software/BLOPEX/, as an external package to the Portable, Extensible Toolkit for Scientific Computation (PETSc) (see Balay et al. [2]), and is built into the High Performance Preconditioners (*hypre*) library (see Falgout, Jones, and Yang [8, 9]). Jose Roman has also written a Scalable Library for Eigenvalue Problem Computations (SLEPc) (see Hernandez, Roman, and Vidal [10]) interface to our *hypre* BLOPEX.

[*]Received by the editors May 31, 2006; accepted for publication (in revised form) July 2, 2007; published electronically September 28, 2007. Preliminary results of this work have been presented at the 11th and 12th Copper Mountain Conferences on Multigrid Methods and Preconditioning (2003), the SIAM Conference on Parallel Processing for Scientific Computing (2004), and the 16th International Conference on Domain Decomposition Methods (2005), and have appeared in Knyazev and Argentati [20] and Lashuk et al. [23].

http://www.siam.org/journals/sisc/29-5/66162.html

[†]Department of Mathematical Sciences, University of Colorado at Denver and Health Sciences Center, P.O. Box 173364, Campus Box 170, Denver, CO 80217-3364 (Andrew.Knyazev@cudenver.edu, http://math.cudenver.edu/~aknyazev/, Merico.Argentati@cudenver.edu, http://math.cudenver.edu/~rargenta/, Ilya.Lashuk@cudenver.edu). The work of these authors was partially supported by NSF awards CNS 0420985, DMS 0208773, and DMS 0612751 and by the Lawrence Livermore National Laboratory.

[‡]Harrow School of Computer Science, University of Westminster, London HA1 3TP, UK (Evgueni.Ovtchinnikov@na-net.ornl.gov).

The native *hypre* BLOPEX version takes advantage of powerful *hypre* multigrid preconditioners, both algebraic multigrid (AMG, called BoomerAMG in *hypre*) and geometric or structured multigrid (SMG). BLOPEX in PETSc gives the PETSc community easy access to the customizable code of a preconditioned eigensolver.

Currently, BLOPEX includes only one solver—the Locally Optimal Block Preconditioned Conjugate Gradient (LOBPCG) method (see Knyazev [17, 18]) for eigenproblems $Ax = \lambda Bx$ with large, possibly sparse, symmetric matrix A and symmetric positive definite matrix B. Preconditioned eigenvalue solvers in general (see Knyazev [16, 17]) and in particular the LOBPCG method have recently attracted attention as potential competitors to (block-) Lanczos methods. LOBPCG has been implemented using different computer languages in a number of other software packages such as: C++ in Anasazi Trilinos (see Arbenz et al. [1] and Heroux et al. [11]) and in NGSolve (see Zaglmayr [30, sections 7.2–7.4]); C in PRIMME (see Stathopoulos and McCombs [25]); FORTRAN 77 in PLTMG (see Bank [3]); FORTRAN 90 in ABINIT (see Bottin et al. [5]) and in PESCAN (see Tomov et al. [26]); and Python in SciPy by Robert Cimrman and in PYFEMax by Peter Arbenz and Roman Geus. LOBPCG has been independently tested in Yamada et al. [27, 28] for the Fermion–Habbard model on earth simulator CDIR/MPI; in Arbenz et al. [1] and Borzí and Borzí [4] using AMG preconditioning for vibration problems; and in Tomov et al. [26] and Yang, Meza, and Wang [29] for electronic structure calculations. Our *hypre* BLOPEX version has been used in Bramble, Kolev, and Pasciak [6] for the Maxwell problem.

Section 2 contains a complete and detailed description of the LOBPCG algorithm as implemented in BLOPEX. We discuss our specific abstract implementation approach in section 3. Parallel performance on distributed and shared memory systems using domain decomposition and multigrid preconditioning is discussed in section 4. Technical information, in the appendices, includes descriptions of variables, installation and testing instructions, a list of computers used for testing, and acronyms.

2. LOBPCG and its implementations.

In subsection 2.1, we briefly describe the ideas behind the LOBPCG method, mostly following Knyazev [17, 18]. In subsection 2.2, we present a complete description of the LOBPCG algorithm as implemented in BLOPEX. Deflation by hard and soft locking is suggested in subsection 2.3.

2.1. LOBPCG ideas and description.

2.1.1. The problem and the assumptions.

We consider the problem of computing the m smallest eigenvalues and the corresponding eigenvectors of the generalized eigenvalue problem $Ax = \lambda Bx$, with symmetric (Hermitian in the complex case) matrices A and B, the latter assumed to be positive definite. For a given approximate eigenvector x, we use the traditional Rayleigh quotient $\lambda(x) = (x, Ax)/(x, Bx)$ in the standard scalar product as an approximation to the corresponding eigenvalue. We emphasize that only matrix B is assumed to be positive definite. There is no such requirement on A; moreover, in all formulas in this section we can replace A with $A + \alpha B$ and the formulas are invariant with respect to a real shift α.

To accelerate the convergence, we introduce a preconditioner T, which is typically a function that for a given vector x produces Tx. The existing theory (see, e.g., [18, 21]) of the LOBPCG method is based on several assumptions: T needs to be linear, symmetric, and positive definite. Under these assumptions a specific convergence rate bound is proved in [18, 21]. Numerical tests suggest that *all these requirements on T are necessary* in order to guarantee this convergence rate. The use of preconditioning that does not satisfy the requirements above not only can slow down the convergence, but may also lead to a breakdown of the method and severe instabilities in the code, resulting in incorrect results—however, this should not be mistakenly interpreted as bugs in the code.

2.1.2. Single-vector LOBPCG. For computing only the smallest eigenpair, i.e., if $m = 1$, the LOBPCG method takes the form of a 3-term recurrence:

$$\begin{aligned} &x^{(i+1)} = w^{(i)} + \tau^{(i)}x^{(i)} + \gamma^{(i)}x^{(i-1)}, \\ &w^{(i)} = T(Ax^{(i)} - \lambda^{(i)}Bx^{(i)}),\ \lambda^{(i)} = \lambda(x^{(i)}) = (x^{(i)}, Ax^{(i)})/(Bx^{(i)}, x^{(i)}) \end{aligned} \tag{2.1}$$

with properly chosen scalar iteration parameters $\tau^{(i)}$ and $\gamma^{(i)}$. The easiest, and perhaps the most efficient, choice of parameters is based on an idea of *local optimality* (see Knyazev [14, 17, 18]), namely, select $\tau^{(i)}$ and $\gamma^{(i)}$ that minimize the Rayleigh quotient $\lambda(x^{(i+1)})$ by using the Rayleigh–Ritz procedure in the 3-D trial subspace spanning $x^{(i)}$, $w^{(i)}$, and $x^{(i-1)}$. We do not describe the Rayleigh–Ritz procedure using an arbitrary basis of the trial subspace here and assume that the reader is familiar with it.

If $\gamma^{(i)} = 0$, method (2.1) turns into the classical preconditioned locally optimal steepest descent method, so one can think of method (2.1) as an attempt to accelerate the steepest descent method by adding an extra term to the iterative recurrence; see, e.g., Knyazev [14, 17]. The general idea of accelerating an iterative sequence by involving the previous approximation is very natural and is well known. Let us note here that if instead of the previous approximation $x^{(i-1)}$ we use the previous preconditioned residual $w^{(i-1)}$ in formula (2.1), we get a different method that in practical tests does not converge as fast as method (2.1).

When implementing the Rayleigh–Ritz procedure in the trial subspace spanning $x^{(i)}$, $w^{(i)}$, and $x^{(i-1)}$, we need to be careful in choosing an appropriate basis that leads to reasonably conditioned Gram matrices. The current eigenvector approximation $x^{(i)}$ and the previous eigenvector approximation $x^{(i-1)}$ get closer to each other in the process of iterations, so special measures need to be taken in method (2.1) to overcome the potential numerical instability due to the round-off errors. Using both vectors $x^{(i)}$ and $x^{(i-1)}$ as basis vectors of the trial subspace leads to ill-conditioned Gram matrices, and the Rayleigh–Ritz method can produce spurious eigenpairs.

A simple fix is suggested in [18]: the 3-term recurrence that contains the current eigenvector approximation, the preconditioned residual, and the implicitly computed difference between the current and the previous eigenvector approximations:

$$\begin{array}{rclcl} x^{(i+1)} & = & w^{(i)} + \tau^{(i)}x^{(i)} & + & \gamma^{(i)}p^{(i)},\ w^{(i)} = T(Ax^{(i)} - \lambda^{(i)}Bx^{(i)}), \\ p^{(i+1)} & = & w^{(i)} & + & \gamma^{(i)}p^{(i)},\ p^{(0)} = 0,\ \lambda^{(i)} = \lambda(x^{(i)}). \end{array} \tag{2.2}$$

We have $x^{(i+1)} \in \operatorname{span}\{w^{(i)},\ x^{(i)},\ p^{(i)}\} = \operatorname{span}\{w^{(i)},\ x^{(i)},\ x^{(i-1)}\}$ as the directions $p^{(i+1)} = x^{(i+1)} - \tau^{(i)}x^{(i)}$; therefore, the new formula (2.2) is mathematically equivalent to (2.1) in exact arithmetic. We choose in (2.2) the scalar iteration parameters $\tau^{(i)}$ and $\gamma^{(i)}$ as above, i.e., minimizing the Rayleigh quotient $\lambda(x^{(i+1)})$.

We note that formula (2.2) is not quite the ultimate fix; e.g., it is possible in principle that at the initial stage of the iterations $\tau^{(i)}$ is small, so that $p^{(i+1)}$ is too close to $x^{(i+1)}$, and the Rayleigh–Ritz procedure may fail. The proper choice of the basis in the trial subspace $\operatorname{span}\{w^{(i)}, x^{(i)}, p^{(i)}\}$ for the Rayleigh–Ritz procedure in (2.2), having in mind that vectors $w^{(i)}$ and $p^{(i)}$ converge to zero, is not a trivial issue; see Hetmaniuk and Lehoucq [12] for possible causes of LOBPCG instability.

The locally optimal choice of the step sizes in the LOBPCG allows an easy generalization from the single-vector version (2.1) or (2.2) to the block version described next in subsection 2.1.3, where a block of m vectors is iterated simultaneously. We return to the discussion of the choice of the basis in subsection 2.2, where it becomes even more vital with the increase of the block size m.

2.1.3. The LOBPCG basics. A block version of LOBPCG for finding the $m \geq 1$ smallest eigenpairs is suggested in Knyazev [16, 17]:

$$x_j^{(i+1)} \in \operatorname{span}\left\{x_1^{(i-1)},\ x_1^{(i)},\ T(A-\lambda_1^{(i)}B)x_1^{(i)}, \dots,\ x_m^{(i-1)},\ x_m^{(i)},\ T(A-\lambda_m^{(i)}B)x_m^{(i)}\right\},$$

where $x_j^{(i+1)}$ is computed as the jth Ritz vector, $j = 1, \dots, m$, corresponding to the jth smallest Ritz value in the Rayleigh–Ritz procedure on a $3m$-dimensional trial subspace. The block method has the same problem of having close vectors in the trial subspace as the single-vector version (2.1) discussed in the previous subsection, for which [18] suggested the same fix using the directions p.

As in other block methods, the block size should be chosen, if possible, to provide a large gap between first m eigenvalues and the rest of the spectrum, as this typically leads to a better convergence; see [18, 21, 24]. Block methods generally handle clusters in the spectrum and multiple eigenvalues quite well, and the LOBPCG is no exception. An attempt should be made to include the whole cluster of eigenvalues in the block, while for multiple eigenvalues this is not essential at all. The smallest eigenvalues often converge faster, and the few eigenvalues with the noticeably slower convergence usually are the largest in the block and typically correspond to a cluster of eigenvalues that is not completely included in the block. A possible cure is to use a flexible block size a bit larger than the number of eigenpairs actually wanted and to ignore poorly converging eigenpairs.

The block size m should not be so big that the costs of the Rayleigh–Ritz procedure in the $3m$-dimensional trial subspace dominate the costs of iterations. If a large number of eigenpairs is needed, deflation is necessary; see subsection 2.3. The optimal choice of the block size m is problem-, software-, and computer-dependent; e.g., it is affected by such details as the amount of the CPU cache and the use of optimized BLAS library functions in the multivector implementation; see section 3.

2.2. The detailed description of the LOBPCG algorithm. The description of the LOBPCG algorithm as implemented in our BLOPEX-1.0 code follows:

```
Input: $m$ starting linearly independent vectors in $X \in \mathbb{R}^{n\times m}$, $l$ linearly independent
    constraint vectors in $Y \in \mathbb{R}^{n\times l}$, devices to compute $A*X$, $B*X$, and $T*X$.
 1. Allocate memory $W, P, Q, AX, AW, AP, BX, BW, BP \in \mathbb{R}^{n\times m}, BY \in \mathbb{R}^{n\times l}$.
 2. Apply the constraints to $X$:
    $BY = B*Y;\ X = X - Y*\left(Y^T*BY\right)^{-1}*((BY)^T*X)$.
 3. $B$-orthonormalize $X$: $BX = B*X; R = \mathrm{chol}(X^T*BX); X = X*R^{-1}$;
    $BX = BX*R^{-1}$; $AX = A*X$. (Note: "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 the index set $J$ of active iterates to be $\{1,\dots,m\}$.
 6. for $k = 0,\dots,MaxIterations$:
 7.   Compute the residuals: $W_J = AX_J - BX_J*\Lambda_J$.
 8.   Exclude from the index set $J$ the indices that correspond to residual
```

```
         vectors for which the norm has become smaller than the tolerance.
         If $J$, then becomes empty; exit loop.
 9.      Apply the preconditioner $T$ to the residuals: $W_J = T * W_J$.
 10      Apply the constraints to the preconditioned residuals $W_J$:
         $W_J = W_J - Y * \left(Y^T * BY\right)^{-1} * ((BY)^T * W_J)$.
 11.     Compute $BW_J$ and $B$-orthonormalize $W_J$: $BW_J = B * W_J$;
         $R = \mathrm{chol}(W_J^T * BW_J)$; $W_J = W_J * R^{-1}$; $BW_J = BW_J * R^{-1}$.
 12.     Compute $AW_J$: $AW_J = A * W_J$.
 13.     if $k > 0$
 14.         $B$-orthonormalize $P_J$: $R = \mathrm{chol}(P_J^T * BP_J); P_J = P_J * R^{-1}$;
 15.         Update $AP_J = AP_J * R^{-1}$; $BP_J = BP_J * R^{-1}$.
 16.     end if
         Perform the Rayleigh–Ritz procedure:
             Compute symmetric Gram matrices:
 17.         if $k > 0$
 18.             $gramA = \begin{bmatrix} \Lambda & X^T * AW_J & X^T * AP_J \\ \cdot & W_J^T * AW_J & W_J^T * AP_J \\ \cdot & \cdot & P_J^T * AP_J \end{bmatrix}$.
 19.             $gramB = \begin{bmatrix} I & X^T * BW_J & X^T * BP_J \\ \cdot & I & W_J^T * BP_J \\ \cdot & \cdot & I \end{bmatrix}$.
 20.         else
 21.             $gramA = \begin{bmatrix} \Lambda & X^T * AW_J \\ \cdot & W_J^T * AW_J \end{bmatrix}$.
 22.             $gramB = \begin{bmatrix} I & X^T * BW_J \\ \cdot & I \end{bmatrix}$.
 23.         end if
 24.         Solve the generalized eigenvalue problem:
             $gramA * C = gramB * C * \Lambda$, where the first $m$ eigenvalues in
             increasing order are in the diagonal matrix $\Lambda$ and the
             $gramB$-orthonormalized eigenvectors are the columns of $C$.
             Compute Ritz vectors:
 25.         if $k > 0$
 26.             Partition $C = \begin{bmatrix} C_X \\ C_W \\ C_P \end{bmatrix}$ according to the number of columns in
                 $X$, $W_J$, and $P_J$, respectively.
 27.             Compute $P = W_J * C_W + P_J * C_P$;
                 $AP = AW_J * C_W + AP_J * C_P$; $BP = BW_J * C_W + BP_J * C_P$.
 28.             $X = X * C_X + P; AX = AX * C_X + AP; BX = BX * C_X + BP$.
 29.         else
 30.             Partition $C = \begin{bmatrix} C_X \\ C_W \end{bmatrix}$ according to the number of columns in
                 $X$ and $W_J$, respectively.
 31.             $P = W_J * C_W; AP = AW_J * C_W; BP = BW_J * C_W$.
 32.             $X = X * C_X + P; AX = AX * C_X + AP; BX = BX * C_X + BP$.
 33.         end if
 37. end for
Output: The eigenvectors $X$ and the eigenvalues $\Lambda$.
```

For description of all LOBPCG BLOPEX variables see Appendix A.

In the complex case, the transpose needs to be replaced by the adjoint. Only double-precision real arithmetic is supported in the current release 1.0 of BLOPEX.

The algorithm is *matrix-free* in the sense that the matrices A and B are not needed in the algorithm and not stored in the code, but rather are accessed only through matrix-vector product functions provided by the user. Matrix-free codes are needed in applications where the matrices A and B are never explicitly formed, e.g., in finite element method software packages for partial differential equations.

The algorithm uses only one block application of the preconditioner T (step 9) and one block matrix-vector product Bx (step 11) and Ax (step 12) per iteration. This is achieved by storing and manipulating $9m$ vectors ($6m$ vectors if $B = I$). If there is not enough memory for $9m$ vectors, or if m is so large that linear algebra operations with $9m$ vectors are more costly than multiplication by A and B, e.g., if A and B are very sparse, the algorithm can be modified so that only $3m$ vectors are stored, but then additional block matrix-vector products Ax and Bx are required per iteration. Such a modification currently is not yet available in BLOPEX.

Choosing a good basis for the Rayleigh–Ritz method becomes even more delicate with a larger number m of vectors in the block. Ill-conditioned Gram matrices in the Rayleigh–Ritz procedure may result in algorithm failure, inaccurate computations of the Ritz pairs, or spurious Ritz pairs. As it can be seen in the LOBPCG algorithm in steps 11–22, we explicitly compute X, P, and W and form the basis of the Rayleigh–Ritz procedure in the following way: X is left untouched since it already has B-orthonormal columns, made of the Ritz vectors, while the vectors inside both W and P are B-orthonormalized in steps 11 and 14, correspondingly, so we put the identities on the block diagonal of the matrix *gramB* (steps 19 and 22).

For the B-orthonormalization of W and P in steps 11 and 14, the cheapest orthonormalization technique, using the Cholesky decomposition of the Gram matrix, has been chosen. This technique is known to be of poor quality in the presence of the round-off errors. Nevertheless, in our experience, the Cholesky-based B-orthonormalization of W and P reduces the risk of the Rayleigh–Ritz procedure failure while being considerably cheaper than the stable B-orthonormalization of all $3m$ vectors in the basis described, e.g., in Hetmaniuk and Lehoucq [12].

Let us note that matrices P and W converge to zero; what is even worse is that different columns of P and W converge to zero with different speeds, so matrices plugged into the Cholesky decomposition are extremely poorly scaled. It is crucial for the stability that the code of the Cholesky decomposition used here is numerically invariant with respect to matrix scaling; otherwise, the explicit normalization of columns of P and W is necessary prior to the call of the Cholesky decomposition. Such a subtle property as scaling invariance, which we rely on here, apparently may be affected by aggressive optimization flags during the compilation of the Cholesky decomposition code.

We also have tested (in MATLAB) a version in which we explicitly compute the block diagonal of the matrix *gramB* instead of just setting the blocks to be the identities. Such an approach is similar to a known idea of repeated orthonormalization and results in more attainable accuracy. However, in our tests we have found that the attainable accuracy of our Algorithm 2.2 is already adequate, so this is not implemented in our BLOPEX LOBPCG.

Numerical software development is often an act of a balanced compromise between performance, reliability, and accuracy. Performance has been our main priority in

BLOPEX release 1.0, but we preserve reasonable reliability for common situations and achieve the accuracy typical for large-scale applications in engineering. Design of efficient and reliable implementations of the LOBPCG algorithm for BLOPEX is underway.

2.3. LOBPCG deflation by hard and soft locking. When several eigenvectors are computed simultaneously, it is often the case that some eigenvectors converge faster than others. To avoid the unnecessary computational work, one "locks" the eigenvectors that have already converged within a required tolerance while continuing to iterate the others. Locking approximately computed eigenvectors is not well studied theoretically. In this subsection, we introduce hard and soft locking and explain how locking is used in the LOBPCG algorithm, following the presentation of Knyazev [19]. The idea of soft locking is very natural and may have been known to some practitioners, developing codes for symmetric eigenvalue problems, even prior to Knyazev [19], where the term "soft locking" was apparently first introduced, but we are not aware of any earlier references.

A commonly used locking technique, known as deflation by restriction, is to keep the locked eigenvectors unchanged and those that are still iterated orthogonal to the locked ones. This technique is usually called "locking," but here it is referred to as "hard locking" to distinguish it from a new type of locking described next.

A different technique, called here "soft locking," can be used in simultaneous iterations combined with the Rayleigh–Ritz procedure. The idea of this technique is to remove the residuals of the locked vectors from the computation but continue using the vectors themselves in the Rayleigh–Ritz procedure, which can change them. The orthogonality of the soft locked vectors to iterative vectors follows from the well-known fact that the Ritz vectors corresponding to different Ritz values are orthogonal, and those corresponding to multiple Ritz values can be chosen to be orthogonal.

Compared to hard locking, soft locking is computationally more expensive. The advantage of soft locking is that it provides more accurate results. First and foremost, as the number of hard locked vectors increases, the attainable accuracy of the iterated ones may decrease, which may prevent them from reaching the required tolerance. With soft locking, the attainable accuracy of the iterated vectors does not depend on the number and the accuracy of soft locked ones. Second, if the Rayleigh quotient of kth locked vectors is less than the $(k+1)$th exact eigenvalue, then the accuracy of the first k locked vectors will increase in the course of the iterations, owing to the participation in the Rayleigh–Ritz procedure, even though their role in the iterative loop is now reduced to only this procedure. Soft-locking accuracy can be analyzed using known error bounds for the Rayleigh–Ritz method (see, e.g., [13, 15, 22]).

Below we describe the hard and soft locking as used in the BLOPEX implementation of LOBPCG and present preliminary numerical results demonstrating the effectiveness of soft locking compared to traditional hard locking.

Let us consider the single-vector LOBPCG method with no locking as described by (2.1), which we repeat here for the reader's convenience:

$$x^{(i+1)} = w^{(i)} + \tau^{(i)} x^{(i)} + \gamma^{(i)} x^{(i-1)},\ w^{(i)} = T(Ax^{(i)} - \lambda^{(i)} Bx^{(i)}).$$

Let P be a projector on the complement to the span of already computed eigenvectors. In LOBPCG with hard locking, we redefine $w^{(i)} = PT(Ax^{(i)} - \lambda^{(i)} Bx^{(i)})$ so that all iterates are now within the range of P if the initial guess in chosen within the range. Traditionally, P is orthogonal in the B-based scalar product and the complement is orthogonal also in the B-based scalar product, since this allows the easiest and most inexpensive implementation. This is what we use in step 10 of the LOBPCG algorithm in subsection 2.2. The constraints, given by the columns of the matrix Y, are in this case the already approximately computed eigenvectors that we want to deflate.

If the locked eigenvectors are computed exactly, then it is easy to analyze the influence of classical hard locking and to show that it is equivalent to simply removing the locked eigenpairs from consideration. If the locked eigenvectors are only approximate, the range of P is not exactly an invariant subspace of the matrix pencil $A-\lambda B$, and the analysis of the influence of the approximation quality on locking becomes much more complicated; see, e.g., D'yakonov and Knyazev [7].

As the number of locked vectors increases, their inaccuracy may affect the attainable accuracy of iterated vectors. To avoid this, we use soft locking in the LOBPCG algorithm in subsection 2.2. We denote by X_J the matrix that contains only the active (not soft locked) iterative eigenvector approximations as columns, while all m iterative vectors X are included in the Rayleigh–Ritz procedure. Only X_J participates in iterations; i.e., having $\Lambda_J^{(i)}$ and $X_J^{(i)}$ we compute only $W_J^{(i)} = T(AX_J^{(i)} - BX_J^{(i)}\Lambda_J^{(i)})$, where $\Lambda_J^{(i)}$ is a diagonal matrix of approximate eigenvalues corresponding to $X_J^{(i)}$. We include the complete X, but only W_J and P_J, in the basis of the Rayleigh–Ritz procedure in steps 18–22 and 26–32 of the LOBPCG algorithm in subsection 2.2.

The positive influence of soft locking is especially noticeable when the stopping tolerance is not so small. Figure 2.1 presents the results of soft locking when the residual norm reaches 10^{-1} in LOBPCG with $m = 10$ for a model problem. We observe that the active eigenpairs continue to converge and that the eigenpairs, which are already soft locked, still improve in accuracy.

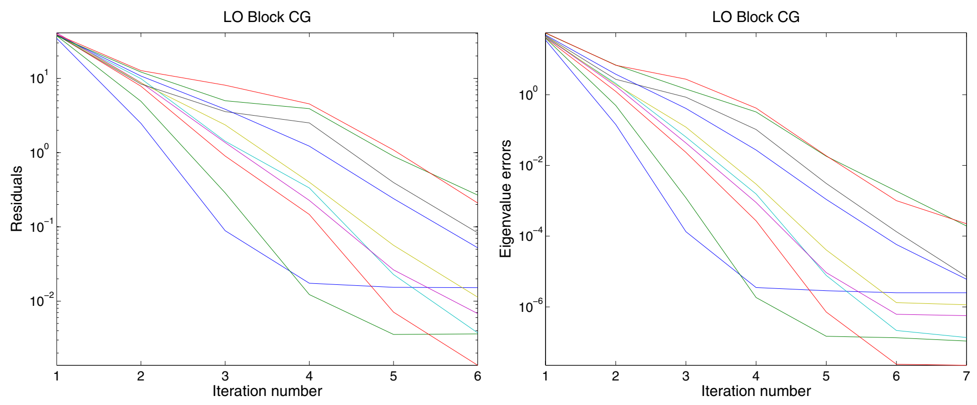

FIG. 2.1. *Residual norms (left) and eigenvalue errors (right) for* 10 *eigenpairs in LOBPCG with soft locking. Soft locked (the bottom* 6*) eigenpairs continue to improve.*

To summarize, we use the classical hard locking in LOBPCG through the constraints and soft locking inside the LOBPCG code. If the user wants to compute so many eigenpairs that it is not reasonable to include all of them at once in the block, the user should determine an appropriate block size m for the given hardware and call LOBPCG with the block size m several times, putting the previously computed eigenvectors into the constraints for the next LOBPCG call. The LOBPCG MATLAB code includes an example of such a call in the help information.

3. Abstract BLOPEX implementation for PETSc and *hypre*.

PETSC and *hypre* are software libraries for solving large systems on massively parallel computers. Our native PETSc BLOPEX version gives the PETSc user community easy access to the customizable code of a modern preconditioned eigensolver and an opportunity to easily call *hypre* preconditioners from PETSc. The BLOPEX built-in *hypre* version efficiently takes direct advantage of powerful *hypre* AMG preconditioner BoomerAMG and its geometric multigrid preconditioners.

The BLOPEX library currently includes only the LOBPCG eigensolver. The BLOPEX code is written in C-language and calls a few LAPACK subroutines for dense matrix operations. 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, e.g., PETSc or *hypre*, in order to call parallel (multi)vector manipulation routines. A block diagram of the main modules is given in Figure 3.1.

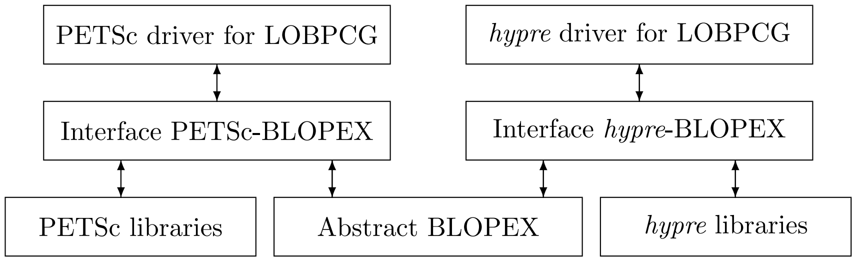

FIG. 3.1. *BLOPEX hypre and PETSc software modules.*

In the abstract part of BLOPEX, the blocks of vectors (X, P, etc.) are represented by an abstract data object we call "multivector." A particular implementation of the multivector is outside of our code, so we only provide an interface to it. Ideally, all operations involving multivectors should be implemented in a block fashion to minimize the number of exchanges between the processors and to take advantage of a highly optimized matrix-matrix multiplication routine `dgemm` in BLAS when computing Gram matrices and updating multivectors. However, to our knowledge such an object is currently not available to users in PETSc and *hypre*, so we had to temporarily use the multivector interfaces as references to individual vectors. When properly implemented multivectors become available in PETSc and *hypre*, we plan to change our interface codes to use the true multivectors, which is expected to lead to greater speed-up due to a smaller number of communications and efficient use of BLAS.

4. BLOPEX LOBPCG numerical results in PETSc and *hypre*.

Here we present results which were accumulated over a considerable length of time. Despite our efforts, the reader may still notice some inconsistency, e.g., some tests may use different tolerances. All tests are performed in double-precision real arithmetic.

4.1. The basic accuracy of the algorithm.

In these tests BLOPEX LOBPCG computes the smallest 50 eigenvalues of 3-D 7-point $200\times200\times200$ and $200\times201\times202$ Laplacians using a direct application of the *hypre* BoomerAMG preconditioning. In the first case we have eigenvalues with multiplicity, and in the second case the eigenvalues are distinct but clustered. The initial approximations are chosen randomly. We set the stopping tolerance (the norm of the maximum residual) equal to 10^{-6}. The numerical output (for the complete data, see Knyazev and Argentati [20]) is compared to the exact eigenvalues

$$\lambda_{i,j,k} = 4\left[\sin\left(\frac{i\pi}{2(n_x+1)}\right)^2 + \sin\left(\frac{j\pi}{2(n_y+1)}\right)^2 + \sin\left(\frac{k\pi}{2(n_z+1)}\right)^2\right]$$

of the 3-D 7-point Laplacians on the $n_x \times n_y \times n_z$ cube with $1 \leq i \leq n_x$, $1 \leq j \leq n_y$, and $1 \leq k \leq n_z$. In both tests for all eigenvalues the maximum relative error is less than 10^{-8}, so LOBPCG is cluster robust, i.e., it does not miss (nearly) multiple eigenvalues, which supports the conclusion of Borzí and Borzí [4].

4.2. Performance versus number of inner iterations. We can execute a preconditioner $x = Tb$ either directly or by calling PCG to solve a linear system, which in our tests is $Ax = b$, so T approximates A^{-1}. Increasing the number of "inner" iterations of the PCG will accelerate the overall convergence, but only if we do not make too many iterations, since even if $T = A^{-1}$, the convergence of the LOBPCG method is still linear. In other words, for a given matrix A and a particular choice of a preconditioner T, there should be an optimal finite (maybe zero, corresponding to the direct application of T) number of inner iterations.

We try to find this optimal number for the Schwarz-PCG and BoomerAMG-PCG preconditioners in *hypre* and PETSc for the 7-point 3-D $100 \times 100 \times 100$ Laplacian with block size $m = 1$. 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. PETSc and *hypre* built-in Schwarz algorithms are different but demonstrate similar behavior. The *hypre* multigrid BoomerAMG and PFMG preconditioners are called from *hypre* test drivers. The *hypre* BoomerAMG preconditioner in addition is called through PETSc. The results for these tests involving different multigrid preconditioners are also similar.

We find that for this problem the optimal number of inner iterations is approximately 10–15 for Schwarz-PCG, but BoomerAMG-PCG works best if BoomerAMG is applied directly as a preconditioner, without even initializing the PCG solve function. Our explanation of this behavior is based on two facts. First, the Schwarz method with the default parameters used here is somewhat cheaper, but not of such good quality when compared to BoomerAMG in these tests. Moreover, the costs for matrix-vector and multivector linear algebra in LOBPCG are relatively small compared to the costs of the BoomerAMG application but is comparable to the costs of applying the Schwarz preconditioner here. Second, one PCG iteration is less computationally expensive compared to one LOBPCG iteration because of the larger number of linear algebra operations with vectors in the latter. A single direct application of BoomerAMG as the preconditioner in LOBPCG results in enough improvement in convergence to make it the best choice, while Schwarz requires more iterations that are less time consuming if performed using PCG rather than directly in LOBPCG.

In the next set of numerical tests for the same eigenproblem, we use the block size $m = 10$, and we vary the type of PCG preconditioner available in the *hypre* IJ interface together with the number of inner PCG iterations. For the complete test data, see Knyazev and Argentati [20]; here we provide only a summary. We observe that both the number of outer iterations and the execution time can be decreased significantly by choosing the maximum number of inner iterations optimally. For this problem, the BoomerAMG preconditioner applied directly has the fastest run time and converges in the smallest number of iterations, 31. If we use no preconditioning,

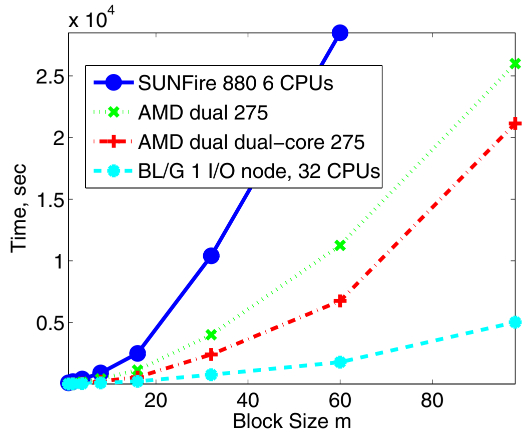

FIG. 4.1. *LOBPCG performance versus block size. Preconditioner: hypre BoomerAMG.*

then in 500 iterations, which take 10 times longer, we still do not reach the required tolerance 10^{-10}, which illustrates the importance of preconditioning.

4.3. LOBPCG performance versus block size. We test *hypre* and PETSc LOBPCG on a 7-point 3-D Laplacian with $128 \times 128 \times 128 \approx 2M$ unknowns using the *hypre* BoomerAMG preconditioner by increasing the block size m from 1 to 98. We run the tests on a shared memory SUN Fire880 system, dual- and single-core AMD Opterons Dual 275 CPUs servers, and a single I/O node of IBM BG/L with 32 CPUs. We find that the *hypre* and PETSc LOBPCG run times are very close and use the same tolerance 10^{-8}; with PETSc the extra overhead is less than a few percent of the total times compared to a direct use of *hypre*, as expected. The growth of the CPU time with the increase of the block size m is approximately linear for this problem up to $m = 16$; then the complexity term m^2n becomes visible and the slower convergence rate for larger eigenvalues starts noticeably affecting the overall timing, as shown in Figure 4.1. An efficient implementation of the multivector, e.g., such as that used in the ABINIT version of the LOBPCG (see Bottin et al. [5]), is expected to significantly improve performance of the LOBPCG with respect to the block size.

4.4. Scalability. For the 7-point 3-D Laplacian we vary the problem size proportionally to the number of processors. We directly apply *hypre* BoomerAMG or PFMG multigrid preconditioner. First, on the MCR cluster the eightfold increase in the number of dual-CPU nodes does not noticeably increase the LOBPCG CPU time, which is approximately 50 sec for BoomerAMG and 180 sec for PFMG with $m = 1$ and tolerance 10^{-8}. The PFMG takes more time overall compared to BoomerAMG because of the larger convergence factor, but each iteration of the former is 50% faster than that of the latter, as we observe in Figure 4.2, which displays the CPU time per iteration for LOBPCG *hypre* code with *hypre* BoomerAMG and PFMG preconditioners in our second test on the Beowulf cluster. We see that the 32-fold increase in the number of dual-CPU nodes just slightly increases the CPU time.

Finally, we solve the 7-point 3-D Laplacian eigenvalue problems for large matrix sizes and for a variety of block sizes m on the BG/L system. We observe a 10–20% variability in the iteration numbers, and thus in the wall clock, due to the randomness of the initial approximations, so we report here typical average results. Tables 4.1 and 4.2 contain information about LOBPCG runs using *hypre* SMG geometric multigrid and tolerance 10^{-6}.

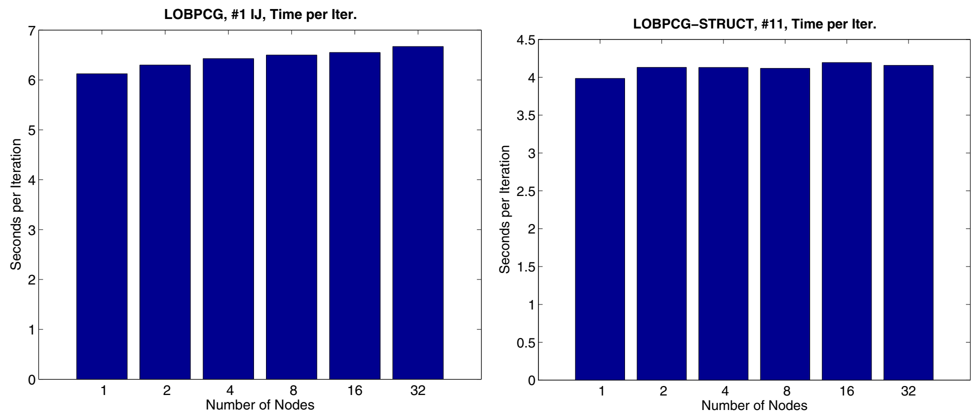

FIG. 4.2. *LOBPCG scalability,* $1M$ *unknowns per node on Beowulf.*

TABLE 4.1
Scalability for $80 \times 80 \times 80 = 512{,}000$ *mesh per CPU,* $m = 1/6$.

# CPUs	Matrix Size	Setup (sec.)	# Iter.	Solve time (sec.)
8	4.096 M	7	9 / 19	62 / 580
64	32.768 M	10	8 / 17	62 / 570
512	0.262144 B	15	7 / 13	56 / 500

TABLE 4.2
Scalability for $50 \times 50 \times 50 = 125{,}000$ *mesh per CPU,* $m = 50$.

# CPUs	Matrix size	Setup (sec.)	# Iter.	Solve time (thous. sec.)
8	1 M	1	34	3
64	8 M	5	32	3
512	64 M	10	29	3

5. Conclusions.

The BLOPEX software is integrated into the *hypre* library and is easily available in PETSc as an external package. BLOPEX is the only currently available software that solves eigenvalue problems by directly using high quality existing *hypre* and PETSc preconditioners. Our abstract C implementation of the LOBPCG in BLOPEX allows easy deployment with different software packages. Our interface routines are based on *hypre* and PETSc standard interfaces and give the user an opportunity to provide matrix-vector multiply and preconditioned solver functions. The code scalability is tested for model problems of large sizes on parallel systems.

Appendix A. Description of all LOBPCG BLOPEX variables.

X (multivector). The iterative approximation to eigenvectors.
A (function). Multiplies the function argument, which is a multivector, by the first matrix of the eigenproblem.
B (function). Multiplies the function argument, which is a multivector, by the second matrix of the eigenproblem.
T (function). Applies the preconditioner to the function argument, a multivector.
Y (multivector). The given constraint that can be used for hard locking. Iterations run in the B-orthogonal complement to the span of Y.

Λ (diagonal matrix, in the actual code given by a vector of the diagonal entries). The iterative approximation to eigenvalues.

W (multivector). Used to store the residuals (step 7), the preconditioned residuals (step 9), and the preconditioned constrained residuals (step 10 and later).

J (index vector). Denotes the set of indexes of active iterates. The starting value for J is $\{1, \ldots, m\}$. If the norm of the residual r_j for some $j \in J$ has become smaller than the tolerance, this index j is excluded from the set J. By X_J we denote a submatrix of X with the columns, the indexes of which are present in the set J—these are the active iterates. The indexes from $\{1, \ldots, m\}$ that are not present in the set J correspond to iterative approximations to eigenvectors, which have been "soft locked."

P (multivector). The LOBPCG "directions."

gramA and *gramB* (matrices). The Gram matrices in the Rayleigh–Ritz procedure.

C (matrix). The matrix of $gramB$-orthonormalized eigenvectors in the Rayleigh–Ritz procedure, used as the matrix of coefficients to compute the Ritz vectors, the new multivectors X, as linear combinations of the basis X, W_J, and P_J.

AX, BX, AP, BP, etc. (multivectors). Implicitly computed results of the application of the corresponding functions to the corresponding vectors. In the absence of the round-off errors, they would be identical to their targets, e.g., $AX = A \star X$.

R (matrix). The upper triangular matrix of the Cholesky factorization, used for the B-orthogonalization via Cholesky.

Appendix B. Installing and testing BLOPEX with PETSc and *hypre*.

Here we give brief but specific information for the reader who wants to install and test our BLOPEX software. This information concerns the current BLOPEX-1.0, PETSc-2.3.2, and *hypre*-2.0.0 releases. We refer to Appendix D for the acronyms and other names not explained here and suggest reading the software manuals for more details.

To install and test BLOPEX and *hypre* in PETSc, download the PETSc source and follow the standard installation procedure with the external packages option. The present procedure is to add "--download-blopex=1 --download-hypre=1" options to the PETSc configuration. BLOPEX and *hypre* sources are then downloaded and compiled automatically. The BLOPEX test driver is already included in PETSc at the src/contrib/blopex/driver directory. For example, in

```
driver -da_grid_x 128 -da_grid_y 128 -da_grid_z 128 -ksp_type preonly -pc_type
hypre -pc_hypre_type boomeramg -n_eigs 10 -seed 2 -tol 1e-6 -itr 5 -full_out 1
```

the eigenproblem for the 7-point 3-D Laplacian is solved by directly using the *hypre* BoomerAMG preconditioner. The last five parameters are self-explanatory.

BLOPEX LOBPCG is built into *hypre* and requires no special configuration options. *hypre* supports four conceptual interfaces: STRUCT, SSTRUCT, FEM, and IJ. LOBPCG has been tested with all but the FEM interface. *hypre*-2.0.0 test drivers for LOBPCG are in the src/test directory. The following are examples with the same options as in the PETSc call above using IJ and STRUCT test drivers:

```
ij -lobpcg -n 128 128 128 -pcgitr 0 -vrand 10 -seed 2 -tol 1e-6 -itr 5 -verb 1
struct -lobpcg -n 128 128 128 -pcgitr 0 -vrand 10 -seed 2 -tol 1e-6 -itr 5 -verb 1
```

Our code can, but is not really intended to, be used with the popular shift-and-invert strategy, since its main advantage compared to traditional eigensolvers is that it uses preconditioning. Our test drivers can call preconditioning directly ("-ksp_type preonly" PETSc option and "-pcgitr 0" *hypre* option) as well as through calls to the *hypre*/PETSc PCG method. 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 $Ax = b$. The BLOPEX LOBPCG code has been tested with all available *hypre* PCG-capable preconditioners in STRUCT, SSTRUCT, and IJ interfaces, with the PETSc native additive Schwarz, and with the PETSc linked BoomerAMG from *hypre*.

Appendix C. Computers used for testing.

AMD workstations Several single- and dual-core AMD two 64 bit 275 CPUs with 16GB shared memory, running Linux, at CCM CU-Denver.

SUN Fire 880 SUN Solaris 9, 6 UltraSPARC III 64 bit CPUs, 24GB shared memory, at CCM CU-Denver.

Beowulf cluster Linux, 36 nodes, dual PIII 933MHz processors and 2GB memory per node with 7.2SCI Dolphin interconnect, at CCM CU-Denver.

MCR cluster Linux, dual Xeon 2.4GHz 4 GB nodes with Quadrics, at LLNL.

Blue Gene/L (BG/L) IBM single-rack with 1024 compute nodes, organized in 32 I/O nodes with 32 compute nodes each. Every compute node is a dual-core chip, containing two 700MHz PowerPC-440 CPUs and 512 MB of memory. We run using the default 1 CPU for computing and 1 CPU for communication on every compute node. Location: NCAR.

Appendix D. Acronyms and other names.

D.1. Organizations.

CCM CU-Denver Center for Computational Mathematics, University of Colorado at Denver and Health Sciences Center, Downtown Denver Campus.

LLNL Lawrence Livermore National Laboratory.

NCAR National Center for Atmospheric Research.

D.2. Software packages.

BLOPEX Block Locally Optimal Preconditioned Eigenvalue Xolvers, a library for large-scale eigenvalue problems, developed by the authors of the paper.

PETSc Portable, Extensible Toolkit for Scientific Computation, tools for the scalable solution of partial differential equations and related problems developed by Argonne National Laboratory; see Balay et al. [2].

hypre High Performance Preconditioners, scalable linear solvers package developed by LLNL; see Falgout, Jones, and Yang [8, 9].

SLEPc Scalable Library for Eigenvalue Problem Computations, a library for large eigenvalue problems, an extension of PETSc; see Hernandez, Roman, and Vidal [10].

D.3. *hypre* interfaces and preconditioners.

IJ A Linear-Algebraic Interface for applications with sparse matrices.

- **BoomerAMG** a parallel implementation of an algebraic multigrid.
- **Schwarz** Agglomeration-based domain decomposition preconditioner.

STRUCT A structured grid interface for applications with logically rectangular grids.

- **SMG** A parallel semicoarsening multigrid solver for the linear systems arising from finite difference, finite volume, or finite element discretizations of the diffusion equation on logically rectangular grids. The code solves both 2-D and 3-D problems with discretization stencils of up to 9-point in 2-D and up to 27-point in 3-D. The algorithm semicoarsens in the z-direction and uses plane smoothing.

PFMG A parallel semicoarsening multigrid solver similar to SMG. The main difference between the two methods is in the smoother: PFMG uses simple pointwise smoothing. As a result, PFMG is not as robust as SMG, but is much more efficient per V-cycle.

SSTRUCT A semistructured grid interface that targets applications with grids that are mostly structured but with some unstructured features.

FEI An unstructured grid interface designed for Finite Element applications.

D.4. Methods.

PCG Preconditioned Conjugate Gradient.

LOBPCG Locally Optimal Block Preconditioned Conjugate Gradient.

AMG Algebraic Multigrid.

D.5. Computer terms.

CPU Central Processing Unit.

MPI Message Passing Interface.

BLAS Basic Linear Algebra Subprograms are routines that provide standard building blocks for performing basic vector and matrix operations.

Acknowledgments. We are grateful to all members of the Scalable Algorithms Group of the Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, and, in particular, to Ulrike Yang, Rob Falgout, Edmond Chow, Charles Tong, and Panayot Vassilevski for their patient support and help. We thank Matthew Knepley and other members of the PETSc team for their help with incorporating our BLOPEX as an external package into the PETSc library. We thank Jose Roman, a member of the SLEPc team, for writing the SLEPc interface to our *hypre* LOBPCG.

REFERENCES

[1] P. Arbenz, U. L. Hetmaniuk, R. B. Lehoucq, and R. S. Tuminaro, *A comparison of eigensolvers for large-scale 3D modal analysis using AMG-preconditioned iterative methods*, Int. J. Numer. Meth. Engrg., 64 (2005), pp. 204–236.

[2] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, *PETSc Users Manual*, Technical report ANL-95/11 - Revision 2.1.5, Argonne National Laboratory, Argonne, IL, 2004.

[3] R. E. Bank, *PLTMG: A Software Package for Solving Elliptic Partial Differential Equations*, Users' Guide 9.0, Department of Mathematics, University of California at San Diego, San Diego, CA, 2004. Available online at http://cam.ucsd.edu/~reb/reports/b15.pdf.gz.

[4] A. Borzì and G. Borzì, *Algebraic multigrid methods for solving generalized eigenvalue problems*, Int. J. Numer. Meth. Engrg., 65 (2006), pp. 1186–1196.

[5] F. Bottin, S. Leroux, A. Knyazev, and G. Zérah, *Large scale ab initio calculations based on three levels of parallelization*, Computational Materials Science, to appear (http://dx.doi.org/10.1016/j.commatsci.2007.07.019). Also available online at http://arxiv.org/abs/0707.3405.

[6] J. H. Bramble, T. V. Kolev, and J. E. Pasciak, *The approximation of the Maxwell eigenvalue problem using a least-squares method*, Math. Comp., 74 (2005), pp. 1575–1598.

[7] E. G. D'yakonov and A. V. Knyazev, *On an iterative method for finding lower eigenvalues*, Russian J. Numer. Anal. Math. Modelling, 7 (1992), pp. 473–486.

[8] R. D. Falgout, J. E. Jones, and U. M. Yang, *Pursuing scalability for hypre's conceptual interfaces*, ACM Trans. Math. Software, 31 (2005), pp. 326–350.

[9] R. D. Falgout, J. E. Jones, and U. M. Yang, *The design and implementation of hypre, a library of parallel high performance preconditioners*, in Numerical Solution of Partial Differential Equations on Parallel Computers, A. M. Bruaset and A. Tveito, eds., Lect. Notes Comput. Sci. Eng. 51, Springer, Berlin, 2006, pp. 267–294.

[10] V. Hernandez, J. E. Roman, and V. Vidal, *SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems*, ACM Trans. Math. Software, 31 (2005), pp. 351–362.

[11] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley, *An overview of the Trilinos project*, ACM Trans. Math. Software, 31 (2005), pp. 397–423.

[12] U. Hetmaniuk and R. Lehoucq, *Basis selection in LOBPCG*, J. Comput. Phys., 218 (2006), pp. 324–332.

[13] A. V. Knyazev, *Sharp a priori error estimates of the Rayleigh-Ritz method without assumptions of fixed sign or compactness*, Math. Notes, 38 (1986), pp. 998–1002.

[14] A. V. Knyazev, *A preconditioned conjugate gradient method for eigenvalue problems and its implementation in a subspace*, in Numerical Treatment of Eigenvalue Problems, Vol. 5, Internat. Ser. Numer. Math. 96, Birkhäuser, Basel, 1991, pp. 143–154.

[15] A. V. Knyazev, *New estimates for Ritz vectors*, Math. Comp., 66 (1997), pp. 985–995.

[16] A. V. Knyazev, *Preconditioned eigensolvers—An oxymoron?*, Electron. Trans. Numer. Anal., 7 (1998), pp. 104–123.

[17] A. V. Knyazev, *Preconditioned eigensolvers*, in Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide, Z. Bai, J. Demmel, J. Dongarra, A. Ruhe, and H. van der Vorst, eds., SIAM, Philadelphia, 2000, pp. 352–368.

[18] A. V. Knyazev, *Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method*, SIAM J. Sci. Comput., 23 (2001), pp. 517–541.

[19] A. V. Knyazev, *Hard and soft locking in iterative methods for symmetric eigenvalue problems*, Presentation at the Eighth Copper Mountain Conference on Iterative Methods, March 28–April 2, 2004. Available online at http://math.cudenver.edu/~aknyazev/research/conf/cm04_soft_locking/cm04.pdf

[20] A. V. Knyazev and M. E. Argentati, *Implementation of a Preconditioned Eigensolver Using Hypre*, Tech. report, CCM-CU Denver, Denver, CO, 2005. Available online at http://math.cudenver.edu/ccm/reports/rep220.pdf.

[21] A. V. Knyazev and K. Neymeyr, *A geometric theory for preconditioned inverse iteration,* III*: A short and sharp convergence estimate for generalized eigenvalue problems*, Linear Algebra Appl., 358 (2003), pp. 95–114.

[22] A. V. Knyazev and J. E. Osborn, *New a priori FEM error estimates for eigenvalues*, SIAM J. Numer. Anal., 43 (2006), pp. 2647–2667.

[23] I. Lashuk, M. E. Argentati, E. Ovchinnikov, and A. V. Knyazev, *Preconditioned eigensolver LOBPCG in hypre and PETSc*, in Proceedings of the 16th International Conference on Domain Decomposition Methods (2005), O. B. Widlund and D. E. Keyes, eds., Lect. Notes Comput. Sci. Eng. 55, Springer, Berlin, 2007, pp. 635–642.

[24] E. Ovtchinnikov, *Cluster robustness of preconditioned gradient subspace iteration eigensolvers*, Linear Algebra Appl., 415 (2006), pp. 140–166.

[25] A. Stathopoulos and J. R. McCombs, *PRIMME: PReconditioned Iterative Multi-Method Eigensolver: Methods and Software Description*, Tech report WM-CS-2006-08, College of William and Mary, Williamsburg, VA, 2006. Available online at http://www.cs.wm.edu/ andreas/publications/primme.pdf.

[26] S. Tomov, J. Langou, A. Canning, L.-W. Wang, and J. Dongarra, *Comparison of nonlinear conjugate-gradient methods for computing the electronic properties of nanostructure architectures*, in International Workshop on Computational Nanoscience and Technology, Lect. Notes Comput. Sci. 3516, Springer-Verlag, New York, 2005, pp. 317–325.

[27] S. Yamada, T. Imamura, and M. Machida, 16.447 *tflops and* 159*-billion-dimensional exact-diagonalization for trapped Fermion–Hubbard model on the earth simulator*, in Proceedings of the ACM/IEEE Conference on Supercomputing, Seattle, WA, November 12–18, 2005, IEEE Computer Society Press, 2005, p. 44.

[28] S. Yamada, T. Imamura, T. Kano, and M. Machida, *High-performance computing for exact numerical approaches to quantum many-body problems on the earth simulator*, in Proceedings of the ACM/IEEE Conference on Supercomputing (Tampa, FL, November 11–17, 2006), ACM Press, New York, 2006, article 47.

[29] C. Yang, J. C. Meza, and L. Wang, *A constrained optimization algorithm for total energy minimization in electronic structure calculations*, J. Comput. Phys., 217 (2006), pp. 709–721.

[30] S. Zaglmayr, *High Order Finite Element Methods for Electromagnetic Field Computation*, Ph.D. thesis, Johannes Kepler University Linz, Linz, Germany, 2006.