Implementation of a preconditioned eigensolver using Hypre

Andrew V. Knyazev [1],* and Merico E. Argentati [1]

[1] *Department of Mathematics, University of Colorado at Denver, USA*

SUMMARY

This paper describes a parallel preconditioned algorithm for solution of partial eigenvalue problems for large sparse symmetric matrices on massively parallel computers, taking advantage of advances in the scalable linear solvers, in particular in multigrid technology and in incomplete factorizations, developed under the Hypre project, at the Lawrence Livermore National Laboratory (LLNL), Center for Applied Scientific Computing. The algorithm implements a "matrix free" locally optimal block preconditioned conjugate gradient method (LOBPCG), suggested earlier by the first author, to compute one or more of the smallest eigenvalues and the corresponding eigenvectors of a symmetric matrix.

We discuss our Hypre specific implementation approach for a flexible parallel algorithm, and the capabilities of the developed software. We demonstrate parallel performance on a set of test problems, using Hypre algebraic multigrid and other preconditioners. The code is written in MPI based C-language and uses Hypre and LAPACK libraries. It has been included in Hypre starting with revision 1.8.0b, which is publicly available on the Internet at LLNL web site. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: Eigenvalue; eigenvector; symmetric eigenvalue problem; preconditioning; sparse matrices; parallel computing; conjugate gradients; algebraic multigrid; additive Schwarz; incomplete factorization; Hypre.

AMS(MOS) subject classifications. 65F15, 65N25, 65Y05.

1. Introduction

We implement a parallel preconditioned algorithm of the Locally Optimal Block Preconditioned Conjugate Gradient Method (LOBPCG) [3, 4] for the solution of eigenvalue problems

$$Ax = \lambda x$$

for large sparse symmetric matrices A on massively parallel computers. We take advantage of advances in the scalable linear solvers, in particular in multigrid technology and in incomplete factorizations, developed under the High Performance Preconditioners (Hypre) project [5], at the Lawrence Livermore

*Correspondence to: Department of Mathematics, University of Colorado at Denver, P.O. Box 173364, Campus Box 170, Denver, CO 80217-3364. *E-mail*: andrew.knyazev@cudenver.edu

Contract/grant sponsor: Center for Applied Scientific Computing, Lawrence Livermore National Laboratory

Contract/grant sponsor: National Science Foundation; contract/grant number: DMS 0208773

National Laboratory (LLNL), Center for Applied Scientific Computing. The solver allows for the utilization of a set of Hypre preconditioners for solution of an eigenvalue problem.

We discuss the implementation approach for a flexible "matrix-free" parallel algorithm, and the capabilities of the developed software, see [1] for details. We test the performance of the software on a few simple test problems.

The LOBPCG Hypre software has been integrated into the Hypre software at LLNL and has been included in the recent Beta Release: Hypre–1.8.0b. Hypre is publicly available on the Internet at LLNL web site to download. Software for the Preconditioned Eigensolvers is available through the Internet page
`http:// www-math.cudenver.edu/ ~aknyazev/software/CG/`
which contains, in particular, the MATLAB code of LOBPCG.

The rest of the paper is organized as follows. Section 2 contains a complete and detailed description of the LOBPCG algorithm as implemented in Hypre. We discuss our Hypre specific implementation approach for a flexible parallel algorithm and the capabilities of the developed software in Sections 3-4. We demonstrate parallel performance on a set of test problems, using Hypre supported preconditioners in Section 5. The final Section 6 contain the conclusions.

2. The Detailed Description of the LOBPCG Algorithm

Below is a detailed description of the LOBPCG algorithm that has been implemented in the Hypre code. The earlier published LOBPCG description in [4] is not nearly as complete as the description below.

LOBPCG Algorithm in Hypre

Input: m starting linearly independent vectors in $X \in \mathbf{R}^{\mathbf{n}\times\mathbf{m}}$, devices to compute $A*x$ and $T*x$.

1. Allocate memory for the six matrices $X, AX, W, AW, P, AP \in \mathbf{R}^{\mathbf{n}\times\mathbf{m}}$.
2. Initial settings for loop: $[Q,R] = \mathrm{qr}(X); X = Q; AX = A*X; k = 0$.
3. Compute initial Ritz vectors and initial residual:
   $[TMP,\Lambda] = eig(X^T*AX); X = X*TMP; AX = AX*TMP;$
   $W = AX - X*\Lambda$.
4. Get norm of individual residuals and store in norm$R \in \mathbf{R}^{\mathbf{m}}$.
5. **do while** $(\max(\mathrm{norm}R) > \mathrm{tol})$
6. Find index set I of norm$R$ for norm$R >$ tol. The index set I defines those residual vectors for which the norm is greater than the tolerance. Refer to the column vectors defined by I, that are a subset of the columns of W and P, by W_I and P_I.
7. Apply preconditioner: $W_I = T*W_I$.
8. Orthonormalize W_I: $[Q,R] = \mathrm{qr}(W_I); W_I = Q$.
9. Update AW_I: $AW_I = A*W_I$.
10. **if** $k > 0$
11. Orthonormalize P_I: $[Q,R] = \mathrm{qr}(P_I); P_I = Q;$
12. Update AP_I: $AP_I = AP_I*R^{-1}$.
13. **end if**
14. Complete Rayleigh Ritz Procedure:
15. **if** $k > 0$

Copyright © 2005 John Wiley & Sons, Ltd.

16. Compute $G = \begin{bmatrix} \Lambda & X^T * AW_I & X^T * AP_I \\ W_I^T * AX & W_I^T * AW_I & W_I^T * AP_I \\ P_I^T * AX & P_I^T * AW_I & P_I^T * AP_I \end{bmatrix}$.
17. Compute $M = \begin{bmatrix} I & X^T * W_I & X^T * P_I \\ W_I^T * X & I & W_I^T * P_I \\ P_I^T * X & P_I^T * W_I & I \end{bmatrix}$.
18. **else**
19. Compute $G = \begin{bmatrix} \Lambda & X^T * AW_I \\ W_I^T * AX & W_I^T * AW_I \end{bmatrix}$.
20. Compute $M = \begin{bmatrix} I & X^T * W_I \\ W_I^T * X & I \end{bmatrix}$.
21. **end if**
22. Solve the generalized eigenvalue problem: $G * y = \lambda M * y$ and store the first m eigenvalues in increasing order in the diagonal matrix Λ and store the corresponding m eigenvectors in Y.
23. **if** $k > 0$
24. Partition $Y = \begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \end{bmatrix}$ according to the number of columns in X, W_I, and P_I, respectively.
25. Compute $X = X * Y_1 + W_I * Y_2 + P_I * Y_3$.
26. Compute $AX = AX * Y_1 + AW_I * Y_2 + AP_I * Y_3$.
27. Compute $P_I = W_I * Y_2 + P_I * Y_3$.
28. Compute $AP_I = AW_I * Y_2 + AP_I * Y_3$.
29. **else**
30. Partition $Y = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix}$ according to the number of columns in X and W_I respectively.
31. Compute $X = X * Y_1 + W_I * Y_2$.
32. Compute $AX = AX * Y_1 + AW_I * Y_2$.
33. Compute $P_I = W_I$.
34. Compute $AP_I = AW_I$.
35. **end if**
36. Compute new residuals: $W_I = AX_I - X_I * \Lambda_I$.
37. Update residual normR for columns of W in W_I.
38. $k = k + 1$
39. **end do**

Output: Eigenvectors X and eigenvalues Λ.

3. LOBPCG Hypre Implementation Strategy

The Hypre eigensolver code is written in MPI, e.g., [2], based C–language and uses Hypre and LAPACK libraries. It has been originally tested with Hypre version 1.6.0 and is being incorporated

Copyright © 2005 John Wiley & Sons, Ltd.

into Hypre starting with the 1.8.0b release. The user interface or Applications Program Interface (API) to the solver is implemented using Hypre style function calls. The matrix–vector multiply and the preconditioned solve are done through user supplied functions. This approach provides significant flexibility, and the implementation illustrates that the LOBPCG algorithm can successfully and efficiently use parallel libraries.

Partition of vector data across the processors is determined by user input consisting of an initial array of parallel vectors. The same partitioning that is used for these vectors, is then used to setup and run the eigensolver. Partitioning of the matrix A and the preconditioner (also determined by the user) must be compatible with the partitioning of these initial vectors.

Preconditioning is implemented through calls to the Hypre preconditioned conjugate gradient method (PCG). Specifically, the action $x = Tb$ of the preconditioner T on a given vector b is performed by calling one of Hypre iterative linear solvers for the following linear algebraic system: $Ax = b$. Thus, we do not attempt to use shift-and-invert strategy, but instead simply take T to be a preconditioner for A.

Since it is assumed in LOBPCG that the matrix A and the preconditioner T are both symmetric positive definite matrices, we only considered the Hypre PCG iterative method for inner iterations. Specifically, LOBPCG has been tested with Hypre PCG solvers with the following preconditioning:

- AMG–PCG: algebraic multigrid
- DS–PCG: diagonal scaling
- ParaSails–PCG: approximate inverse of A is generated by attempting to minimize $\|I - AM\|_F$
- Schwarz–PCG: additive Schwarz
- Euclid–PCG: incomplete LU

4. **Hypre LOBPCG Software Implementation Details**

Hypre supports four conceptual interfaces: Struct, SStruct, FEM and IJ. At present, LOBPCG is only implemented using the IJ interface. This algebraic interface supports applications with general sparse matrices.

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

The test driver ij_es.c is a modified version of the Hypre driver ij.c and retains all the Hypre functionality and capabilities of ij.c. We anticipate that drives ij_es.c and ij.c will be merged in the future Hypre releases. The ij_es.c driver is used to run the eigensolver software with control through command line parameters. Various test matrices can be internally generated that consist of different types of Laplacians, or a Matrix Market format file may be read as input. The user can also choose the preconditioner. An initial set of vectors can be generated randomly, as an identity matrix of vectors, or a Matrix Market file may be input, with it's columns used as initial vectors. Various other input parameters can be specified such as the convergence tolerance, the maximum number of iterations and the maximum number of inner iterations for the Hypre solver. Of course, a user would normally write a code for a specific application using the LOBPCG API. In this case, the test driver ij_es.c can serve as a template.

The routine lobpcg.c implements the main algorithm and the API routines. The routine lobpcg_matrix.c contains the functions that implement parallel Hypre vector operations. It also implements a parallel Modified Gram–Schmidt (MGS) orthonormalization algorithm. The routine lobpcg_utilities.c implements miscellaneous functions such as reading matrix files in the Matrix Market

Copyright © 2005 John Wiley & Sons, Ltd.

Prepared using nlaauth.cls

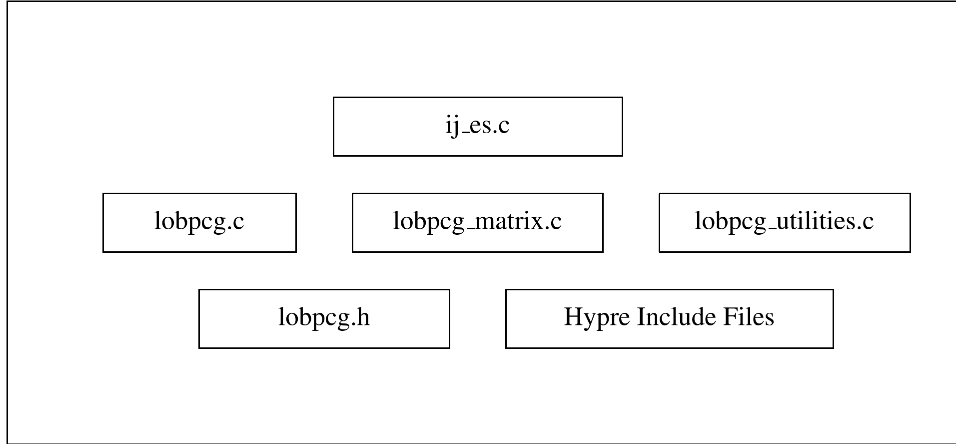

Figure 1. LOBPCG Hypre software modules

format and broadcasting the data to all processors.

Make files for various hardware configurations and compilers are provided. LOBPCG code has been compiled and run on CU Denver cluster using scali and mpich libraries and using gcc, pgcc and mpicc compilers. The code has also been compiled and run on various LLNL Open Computing Facility Production Systems including ASCI blue, M&IC tera, gps and lx. LOBPCG has been tested on several LLNL clusters using Compaq and IBM hardware, running Unix and/or Linux.

5. **Hypre LOBPCG Numerical Results**

5.1. Basic Accuracy of Algorithm

The following formula is used here to conform to the way 3D 7–Point Laplacians are generated in Hypre.

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

that is defined on an $n_x \times n_y \times n_z$ cube with $1 \leq i \leq n_x$, $1 \leq j \leq n_y$ and $1 \leq k \leq n_z$.

In Table I we present numerical data for a 3D 7–Point $100 \times 100 \times 100$ Laplacian, and in Table II we present data for a 3D 7–Point $100 \times 101 \times 102$ Laplacian The numerical output and theoretical results are compared. In both cases LOBPCG computes the first smallest 20 eigenvalues. The initial eigenvectors are chosen randomly. We set the stopping tolerance (the norm of the maximum residual) equal to the default of (10^{-6}). In both cases for all eigenvalues the maximum relative error is less than 10^{-9}. In the first case we have eigenvalues with multiplicity and in the second case the eigenvalues are distinct, but form clusters.

In Table III we present numerical data for a 3D 7–Point $200 \times 200 \times 200$ Laplacian, and in Table IV we present data for a 3D 7–Point $200 \times 201 \times 202$ Laplacian. The numerical output and theoretical results are compared. In both cases LOBPCG computes the first smallest 50 eigenvalues. The initial eigenvectors are chosen randomly. We set the stopping tolerance (the norm of the maximum residual)

Copyright © 2005 John Wiley & Sons, Ltd.

Computed by LOBPCG	Theoretical	Abs. Error	Rel. Error
2.902306248105975e-003	2.902306248071610e-003	3.4365e-014	1.1841e-011
5.803676564900589e-003	5.803676564859043e-003	4.1546e-014	7.1585e-012
5.803676564933479e-003	5.803676564859043e-003	7.4436e-014	1.2826e-011
5.803676565005030e-003	5.803676564859043e-003	1.4599e-013	2.5154e-011
8.705046881746096e-003	8.705046881646476e-003	9.9620e-014	1.1444e-011
8.705046881747718e-003	8.705046881646476e-003	1.0124e-013	1.1630e-011
8.705046881890860e-003	8.705046881646476e-003	2.4438e-013	2.8074e-011
1.063617489417226e-002	1.063617489401058e-002	1.6168e-013	1.5201e-011
1.063617489419002e-002	1.063617489401058e-002	1.7943e-013	1.6870e-011
1.063617489434838e-002	1.063617489401058e-002	3.3780e-013	3.1759e-011
1.160641719850057e-002	1.160641719843391e-002	6.6664e-014	5.7437e-012
1.353754521119737e-002	1.353754521079801e-002	3.9935e-013	2.9500e-011
1.353754521131213e-002	1.353754521079801e-002	5.1412e-013	3.7977e-011
1.353754521135058e-002	1.353754521079801e-002	5.5256e-013	4.0817e-011
1.353754521161252e-002	1.353754521079801e-002	8.1451e-013	6.0166e-011
1.353754521164719e-002	1.353754521079801e-002	8.4918e-013	6.2728e-011
1.353754521171971e-002	1.353754521079801e-002	9.2170e-013	6.8084e-011
1.643891554531294e-002	1.643891552758545e-002	1.7727e-011	1.0784e-009
1.643891554587873e-002	1.643891552758545e-002	1.8293e-011	1.1128e-009
1.643891554649134e-002	1.643891552758545e-002	1.8906e-011	1.1501e-009

Table I. Computed Eigenvalues vs Theoretical – 3D 7–Point Laplacian ($100 \times 100 \times 100$)

Computed by LOBPCG	Theoretical	Abs. Error	Rel. Error
2.846228742602503e-003	2.846228742589029e-003	1.3474e-014	4.7338e-012
5.636061670068701e-003	5.636061669504445e-003	5.6426e-013	1.0012e-010
5.691010695398921e-003	5.691010695232621e-003	1.6630e-013	2.9221e-011
5.747599059530075e-003	5.747599059376461e-003	1.5361e-013	2.6727e-011
8.480843622170175e-003	8.480843622148038e-003	2.2137e-014	2.6102e-012
8.537431986377912e-003	8.537431986291878e-003	8.6034e-014	1.0077e-011
8.592381012111345e-003	8.592381012020053e-003	9.1292e-014	1.0625e-011
1.028289957613539e-002	1.028289957607353e-002	6.1855e-014	6.0153e-012
1.042931557927938e-002	1.042931557925173e-002	2.7646e-014	2.6508e-012
1.058009738856764e-002	1.058009738852800e-002	3.9642e-014	3.7468e-012
1.138221393902417e-002	1.138221393893547e-002	8.8700e-014	7.7928e-012
1.312768152890886e-002	1.312768152871712e-002	1.9174e-013	1.4606e-011
1.318426989316461e-002	1.318426989286096e-002	3.0365e-013	2.3031e-011
1.321914850542536e-002	1.321914850616715e-002	7.4179e-013	5.6115e-011
1.333068589642131e-002	1.333068589603917e-002	3.8214e-013	2.8666e-011
1.336993031623735e-002	1.336993031544341e-002	7.9394e-013	5.9382e-011
1.342487934125382e-002	1.342487934117159e-002	8.2235e-014	6.1255e-012
1.602905184954465e-002	1.602905184550456e-002	4.0401e-012	2.5205e-010
1.612051882680304e-002	1.612051882295458e-002	3.8485e-012	2.3873e-010
1.621471227359417e-002	1.621471226808701e-002	5.5072e-012	3.3964e-010

Table II. Computed Eigenvalues vs Theoretical – 3D 7–Point Laplacian ($100 \times 101 \times 102$)

equal to the default of (10^{-6}). Again in the first case we have eigenvalues with multiplicity and in the second case the eigenvalues are distinct, but form clusters. To test orthogonality we compute the Frobenious norm of the eigenvectors $\|V^T V - I_{m\times m}\|$ where $V \in \mathbf{R^{n\times m}}$ contains the eigenvectors. For

Copyright © 2005 John Wiley & Sons, Ltd.

Prepared using nlaauth.cls

the data in Table III the Frobenious norm is $= 2.130140e-13$. The maximum eigenvalue relative error is approximately $1.0e-9$. For the data in Table IV the Frobenious norm is $= 1.650817e-12$. The maximum eigenvalue relative error is approximately $1.0e-8$.

5.2. Performance Versus the Number of Inner Iterations

Let us remind the reader that in Hypre LOBPCG we execute a preconditioner $x = Tb$ by calling a Hypre preconditioned conjugate gradient method to solve $Ax = b$ with one of Hypre built-in preconditioners. Thus, 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 Hypre preconditioned conjugate gradient method might accelerate the convergence only if we do not make too many inner iterations. In other words, for a given matrix A and a particular choice the Hypre preconditioner, there should be an optimal number of inner iterations.

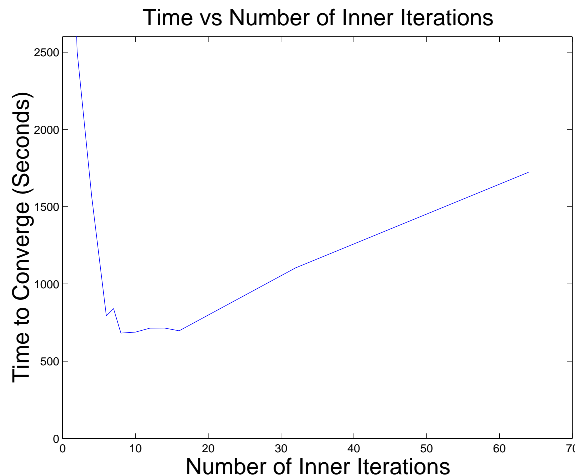

Figure 2. Performance versus the number of inner iterations

In our first numerical example we try to find this optimal number for a 7–Point 3–D Laplacian with $n = 200$, $nz = 55,760,000$ and with the Schwarz–PCG preconditioner. We run the problem on 10 processors. We run LOBPCG until a tolerance of 10^{-6} is achieved. We measure the execution time as we vary the quality of the preconditioner by changing the maximum number of iterations (inner iterations) that are used in the Hypre solver. The results of this experiment are displayed in Figure 2. We find that on this problem the optimal number of inner iterations is approximately 10.

5.3. Different Preconditioners

In the next numerical example we use a 7–Point 3–D Laplacian with $n = 100$, $nz = 6,940,000$, the blocksize equal to 10 and we vary the type of preconditioner. We run the problem on 10 processors using LLNL ASCI blue using Hypre version Hypre-1.6.0. For the one case where no preconditioner is used we iterate until a maximum of 500 iterations is achieved. When a preconditioner is used we run LOBPCG until a tolerance of 10^{-10} is achieved. We set the maximum number of inner iterations equal

Copyright © 2005 John Wiley & Sons, Ltd.

Prepared using nlaauth.cls

to 3 (the default). We illustrate convergence for six different runs as follows: no preconditioner, DS–PCG, ParaSails–PCG, Schwarz–PCG, Euclid–PCG and AMG–PCG. The results of this experiment are displayed in Figure 4. We plot the results in order of improving convergence. We also give the run time in seconds. For this problem the AMG–PCG preconditioner has the fastest run time and converges in the smallest number of iterations which is 30 in 2047 seconds.

In the next numerical example we use a 7–Point 3–D Laplacian with $n = 100$, $nz = 6,940,000$, the blocksize equal to 10 and we vary the type of preconditioner. We run the problem on 10 processors using LLNL ASCI blue using Hypre version Hypre-1.6.0. For the one case where no preconditioner is used we iterate until a maximum of 500 iterations is achieved. When a preconditioner is used we run LOBPCG until a tolerance of 10^{-10} is achieved. We vary the maximum number of inner iterations for each preconditioner setting this to the estimated optimal value that miminizes run time. We illustrate convergence for six different runs as follows with the maximum number of inner iterations shown in parenthesis: no preconditioner (0), DS–PCG (16), ParaSails–PCG (14), Schwarz–PCG (8), Euclid–PCG (6) and AMG–PCG (1). The results of this experiment are displayed in Figure 3. We plot the results in the same order as in Figure 4. We also give the run time in seconds. For this problem the AMG–PCG preconditioner has the fastest run time and converges in the smallest number of iterations which is 31 in 1352 seconds.

It is apparent from these figures that both the number of iterations and the execution time can be decreased significantly by choosing the maximum number of inner iterations optimally. In the case of the AMG–PCG preconditioner, the the number of iterations to converged increased by 1, but the execution time decreased when the maximum number of inner iterations was reduced from 3 to 1.

Copyright © 2005 John Wiley & Sons, Ltd.

Prepared using nlaauth.cls

Computed by LOBPCG	Theoretical	Abs. Error	Rel. Error
7.328583561111907e-004	7.328583560819685e-004	2.9222e-014	3.9874e-011
1.465657036482683e-003	1.465657036456151e-003	2.6532e-014	1.8102e-011
1.465657036504189e-003	1.465657036456151e-003	4.8038e-014	3.2776e-011
1.465657036555275e-003	1.465657036456151e-003	9.9124e-014	6.7631e-011
2.198455716865076e-003	2.198455716830332e-003	3.4744e-014	1.5804e-011
2.198455716948983e-003	2.198455716830332e-003	1.1865e-013	5.3970e-011
2.198455717027431e-003	2.198455716830332e-003	1.9710e-013	8.9654e-011
2.686789266037899e-003	2.686789265965112e-003	7.2786e-014	2.7090e-011
2.686789266056451e-003	2.686789265965112e-003	9.1338e-014	3.3995e-011
2.686789266298282e-003	2.686789265965113e-003	3.3317e-013	1.2400e-010
2.931254397296861e-003	2.931254397204514e-003	9.2347e-014	3.1504e-011
3.419587946372111e-003	3.419587946339294e-003	3.2817e-014	9.5969e-012
3.419587946391159e-003	3.419587946339294e-003	5.1865e-014	1.5167e-011
3.419587946403376e-003	3.419587946339294e-003	6.4082e-014	1.8740e-011
3.419587946447446e-003	3.419587946339294e-003	1.0815e-013	3.1627e-011
3.419587946458649e-003	3.419587946339294e-003	1.1936e-013	3.4903e-011
3.419587946476946e-003	3.419587946339294e-003	1.3765e-013	4.0254e-011
4.152386626793977e-003	4.152386626713476e-003	8.0501e-014	1.9387e-011
4.152386626838803e-003	4.152386626713476e-003	1.2533e-013	3.0182e-011
4.152386626871016e-003	4.152386626713476e-003	1.5754e-013	3.7940e-011
4.395956739058093e-003	4.395956738956095e-003	1.0200e-013	2.3203e-011
4.395956739115164e-003	4.395956738956095e-003	1.5907e-013	3.6185e-011
4.395956739164134e-003	4.395956738956095e-003	2.0804e-013	4.7325e-011
4.640720175941161e-003	4.640720175848256e-003	9.2905e-014	2.0019e-011
4.640720176003971e-003	4.640720175848256e-003	1.5571e-013	3.3554e-011
4.640720176064025e-003	4.640720175848256e-003	2.1577e-013	4.6495e-011
5.128755419455116e-003	5.128755419330278e-003	1.2484e-013	2.4341e-011
5.128755419489028e-003	5.128755419330278e-003	1.5875e-013	3.0953e-011
5.128755419563034e-003	5.128755419330278e-003	2.3276e-013	4.5382e-011
5.128755419626326e-003	5.128755419330278e-003	2.9605e-013	5.7723e-011
5.128755419652193e-003	5.128755419330278e-003	3.2191e-013	6.2767e-011
5.128755419828899e-003	5.128755419330278e-003	4.9862e-013	9.7221e-011
5.373518856398788e-003	5.373518856222438e-003	1.7635e-013	3.2818e-011
5.373518856522068e-003	5.373518856222438e-003	2.9963e-013	5.5761e-011
5.373518857079027e-003	5.373518856222438e-003	8.5659e-013	1.5941e-010
5.861554099838080e-003	5.861554099704460e-003	1.3362e-013	2.2796e-011
5.861554099864227e-003	5.861554099704460e-003	1.5977e-013	2.7257e-011
5.861554099951332e-003	5.861554099704460e-003	2.4687e-013	4.2117e-011
6.349887649026199e-003	6.349887648839240e-003	1.8696e-013	2.9443e-011
6.349887649165296e-003	6.349887648839240e-003	3.2606e-013	5.1348e-011
6.349887649389846e-003	6.349887648839240e-003	5.5061e-013	8.6711e-011
6.349887649599634e-003	6.349887648839240e-003	7.6039e-013	1.1975e-010
6.349887649683784e-003	6.349887648839240e-003	8.4454e-013	1.3300e-010
6.349887651282868e-003	6.349887648839240e-003	2.4436e-012	3.8483e-010
6.592741930912540e-003	6.592741929540924e-003	1.3716e-012	2.0805e-010
6.592741932120729e-003	6.592741929540924e-003	2.5798e-012	3.9131e-010
6.592741932257311e-003	6.592741929540924e-003	2.7164e-012	4.1203e-010
6.594651088573609e-003	6.594651085731400e-003	2.8422e-012	4.3099e-010
7.082686333686548e-003	7.082686329213422e-003	4.4731e-012	6.3156e-010
7.082686334263617e-003	7.082686329213422e-003	5.0502e-012	7.1303e-010

Table III. Computed Eigenvalues vs Theoretical – 3D 7–Point Laplacian ($200 \times 200 \times 200$)

Copyright © 2005 John Wiley & Sons, Ltd.

Computed by LOBPCG	Theoretical	Abs. Error	Rel. Error
7.256560050821676e-004	7.256560050253854e-004	5.6782e-014	7.8250e-011
1.444087866437634e-003	1.444087866387744e-003	4.9890e-014	3.4548e-011
1.451217941392075e-003	1.451217941348242e-003	4.3833e-014	3.0204e-011
1.458454685488341e-003	1.458454685399567e-003	8.8774e-014	6.0869e-011
2.169649802770521e-003	2.169649802710601e-003	5.9920e-014	2.7617e-011
2.176886546806222e-003	2.176886546761926e-003	4.4296e-014	2.0348e-011
2.184016621821864e-003	2.184016621722424e-003	9.9440e-014	4.5531e-011
2.641283120686505e-003	2.641283120632874e-003	5.3631e-014	2.0305e-011
2.660292840210907e-003	2.660292840102666e-003	1.0824e-013	4.0688e-011
2.679586915034578e-003	2.679586914908529e-003	1.2605e-013	4.7040e-011
2.902448483210256e-003	2.902448483084783e-003	1.2547e-013	4.3230e-011
3.366845057100329e-003	3.366845056955731e-003	1.4460e-013	4.2948e-011
3.374081801131233e-003	3.374081801007056e-003	1.2418e-013	3.6803e-011
3.378724701571519e-003	3.378724701465025e-003	1.0649e-013	3.1519e-011
3.393091520585909e-003	3.393091520476848e-003	1.0906e-013	3.2142e-011
3.398018776333513e-003	3.398018776270888e-003	6.2625e-014	1.8430e-011
3.405148851314911e-003	3.405148851231386e-003	8.3525e-014	2.4529e-011
4.099643737568510e-003	4.099643737329912e-003	2.3860e-013	5.8200e-011
4.111523381875465e-003	4.111523381839207e-003	3.6258e-014	8.8187e-012
4.123580712749051e-003	4.123580712593744e-003	1.5531e-013	3.7663e-011
4.316955043867267e-003	4.316955043799349e-003	6.7919e-014	1.5733e-011
4.352588258303715e-003	4.352588258135675e-003	1.6804e-013	3.8607e-011
4.388754388229686e-003	4.388754387899513e-003	3.3017e-013	7.5232e-011
4.575919955822750e-003	4.575919955710155e-003	1.1260e-013	2.4606e-011
4.595214030583404e-003	4.595214030516019e-003	6.7385e-014	1.4664e-011
4.614223750127816e-003	4.614223749985810e-003	1.4201e-013	3.0776e-011
5.042516980226320e-003	5.042516980122206e-003	1.0411e-013	2.0647e-011
5.049753724316828e-003	5.049753724173531e-003	1.4330e-013	2.8377e-011
5.071020119652128e-003	5.071020119498034e-003	1.5409e-013	3.0387e-011
5.085386938932781e-003	5.085386938509857e-003	4.2292e-013	8.3165e-011
5.107186249587937e-003	5.107186249261871e-003	3.2607e-013	6.3844e-011
5.114316324448457e-003	5.114316324222369e-003	2.2609e-013	4.4207e-011
5.308718636271618e-003	5.308718636084337e-003	1.8728e-013	3.5278e-011
5.320775967245566e-003	5.320775966838874e-003	4.0669e-013	7.6435e-011
5.332655611528657e-003	5.332655611348169e-003	1.8049e-013	3.3846e-011
5.775315660627379e-003	5.775315660496388e-003	1.3099e-013	2.2681e-011
5.803818800031267e-003	5.803818799872216e-003	1.5905e-013	2.7405e-011
5.832748185673050e-003	5.832748185584729e-003	8.8322e-014	1.5142e-011
6.251591879300317e-003	6.251591878876629e-003	4.2369e-013	6.7773e-011
6.268215373938566e-003	6.268215373743164e-003	1.9540e-013	3.1173e-011
6.270885954113541e-003	6.270885953682493e-003	4.3105e-013	6.8738e-011
6.304381503892885e-003	6.304381503507001e-003	3.8588e-013	6.1209e-011
6.306519169078511e-003	6.306519168018819e-003	1.0597e-012	1.6803e-010
6.323391223430504e-003	6.323391222976793e-003	4.5371e-013	7.1751e-011
6.470702318936291e-003	6.470702318482879e-003	4.5341e-013	7.0071e-011
6.527694875012630e-003	6.527694874065828e-003	9.4680e-013	1.4504e-010
6.529850866239660e-003	6.529850865593299e-003	6.4636e-013	9.8986e-011
6.585539579459185e-003	6.585539578484342e-003	9.7484e-013	1.4803e-010
6.984390562135336e-003	6.984390559250812e-003	2.8845e-012	4.1300e-010
6.996448015022004e-003	6.996447890005350e-003	1.2502e-010	1.7869e-008

Table IV. Computed Eigenvalues vs Theoretical – 3D 7–Point Laplacian ($200 \times 201 \times 202$)

Copyright © 2005 John Wiley & Sons, Ltd.

Prepared using nlaauth.cls

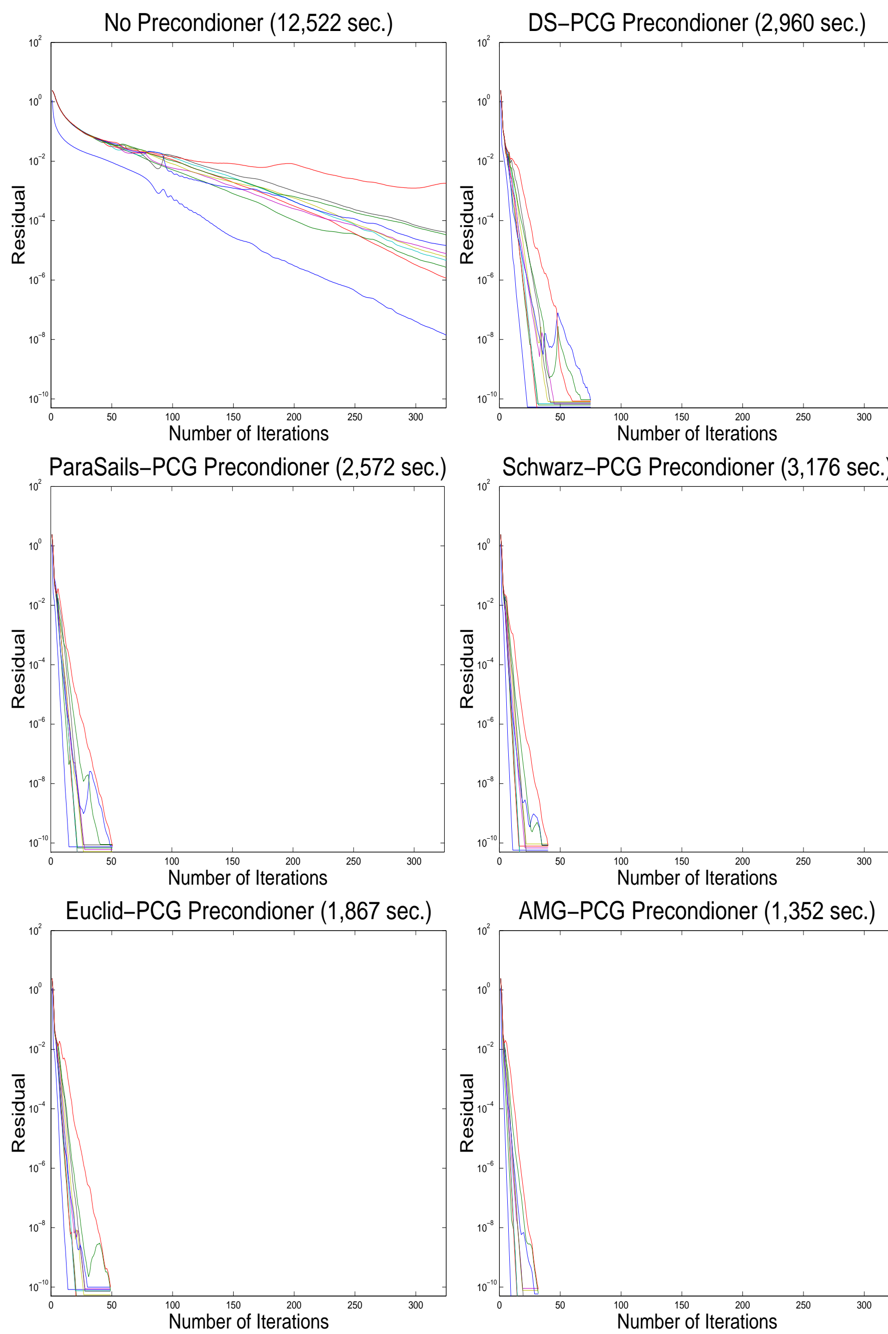

Figure 3. Convergence rates for different preconditioners (optimize inner iter.)

Copyright © 2005 John Wiley & Sons, Ltd.

Prepared using nlaauth.cls

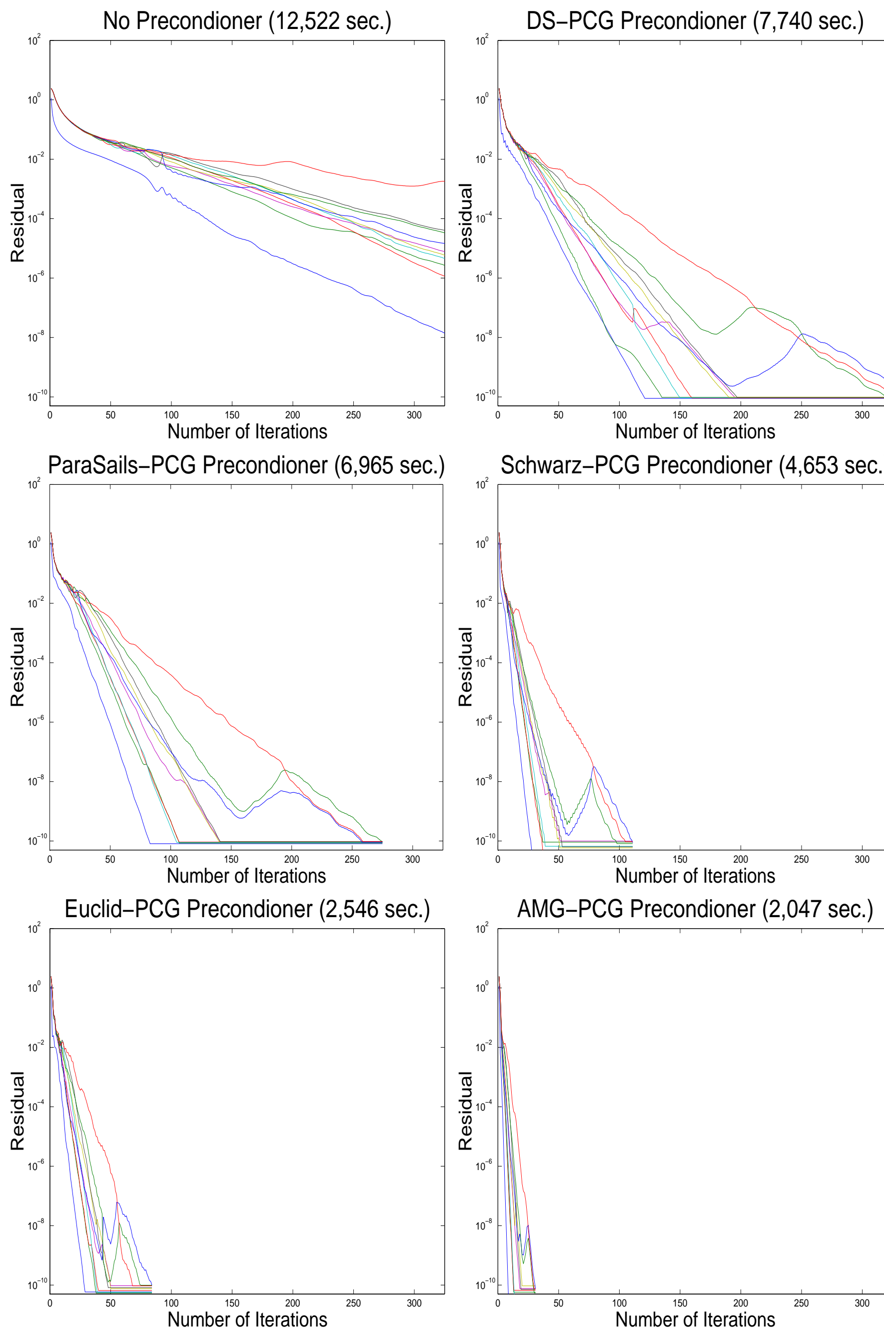

Figure 4. Convergence rates for different preconditioners

Copyright © 2005 John Wiley & Sons, Ltd.

5.4. *Scalability on a Beowulf Cluster*

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 where n is the number of grid points on each side of a cube. We vary n so that the number of non–zeroes nz, is proportional to the number of processors, varying n from $n = 100$ to $n = 252$. We set the block size to 1, set the maximum LOBPCG iterations to 10, set the maximum number of (inner) iterations of the preconditioned solver to 3, and use the Schwarz–PCG preconditioner. We measure the time for setup of the preconditioner, the time for applying the preconditioner and the time for remaining LOBPCG work. The preconditioner is setup before execution of LOBPCG. The time for application of the preconditioner is the sum of the time for each preconditioned/solve step during execution of the LOBPCG algorithm. This data is summarized in Table V and displayed as a bar chart in Figure 5.

We observe good scalability for this problem solved on the Beowulf cluster at CU Denver. This cluster includes 36 nodes, two PIII 933MHz processors and 2GB memory per node, running Linux RedHat and a 7.2SCI Dolpin interconnect.

Nproc	n	nz	Precond. Setup	Apply Precond.	LOBPCG Linear Alg.	Total (sec)
2	100	6,940,000	918.5	155.7	17.7	1091.9
4	126	13,907,376	914.9	161.1	19.4	1095.4
8	159	27,986,067	937.9	163.9	20.3	1122.1
16	200	55,760,000	730.9	166.9	25.3	923.1
32	252	112,000,000	671.0	163.8	38.9	873.7

Table V. Timing results – scalability

5.5. *Scalability on IBM ASCI blue at LLNL*

Our next scalability test is similar to the previous one. We use a 7–Point 3–D Laplacian and vary the problem size so that $n = 100, 126, 159, 200, 252$ to make the total number of non-zeros proportional to the number of processors. The first line in Table VI corresponds to 4 processors since IBM ASCI blue has 4 processors per node. We set the block size to 1, the maximum number of Hyper Solver PCG iterations to 10 (it was 3 in Subsection 5.4, but 10 gives a better LOBPCG convergence according to Subsection 5.2) and we let the LOBPCG iterations run until convergence using the default tolerance of 10^{-6} (in Subsection 5.4 we stop after 10 iterations). We use the same Schwarz–PCG preconditioner. The detailed data for this experiment is summarized in Table VI. As we can see from this table, more iterations are required as the problem becomes larger.

In Figure 6, on the left graph, we plot the total time to converge as the problem size and number of processors increase. This time includes 1) the time to setup the preconditioner, 2) the accumulated time to apply the preconditioner during LOBPCG execution and 3) the accumulated time for matrix vector multiplies and other linear algebra during LOBPCG execution. As we can see the time increases as the problem size increases, with the time being dominated by the application of the preconditioner. The accumulated time for matrix vector multiplies and other linear algebra is a relatively small proportion of the total time.

In Figure 6, on the right graph, we plot the total time/iteration as the problem size and number

Copyright © 2005 John Wiley & Sons, Ltd.

Prepared using nlaauth.cls

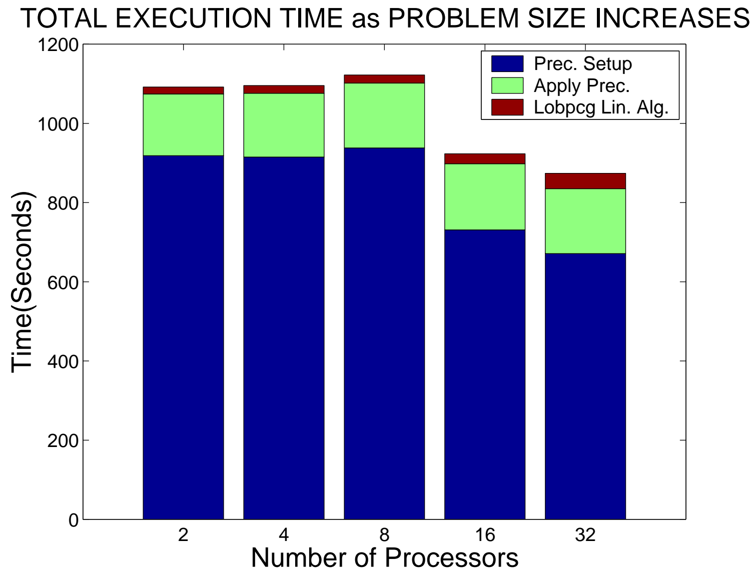

Figure 5. LOBPCG scalability as problem size increases

Nproc	n	Prec. setup (Seconds)	Apply Prec. (Seconds)	LOBPCG Lin. Alg. (Seconds)	Itr	Apply/Itr (Seconds)	Alg/Itr (Seconds)
4	100	342	242.8	14.5	9	26.9	1.61
8	126	352	287.2	17.6	10	28.7	1.76
16	159	349	401.9	28.6	13	30.9	2.20
32	200	456	517.8	49.4	16	32.6	3.09
64	252	377	673.4	84.5	21	32.1	4.03

Table VI. Scalability Data for 3–D Laplacian

of processors increase for 1) the average time/iteration to apply the preconditioner during LOBPCG execution and 2) the average time/iteration for matrix vector multiplies and other linear algebra during LOBPCG execution. Good scalability is exhibited in this graph for the average time/iteration. Again the time/iteration for matrix vector multiplies and other linear algebra is a relatively small proportion of the total time/iteration.

6. **Conclusions**

Let us formulate here the main points of the present paper:

- The novel soft locking scheme, described here, is implemented in the Hypre LOBPCG code and passes the tests performed.

Copyright © 2005 John Wiley & Sons, Ltd.

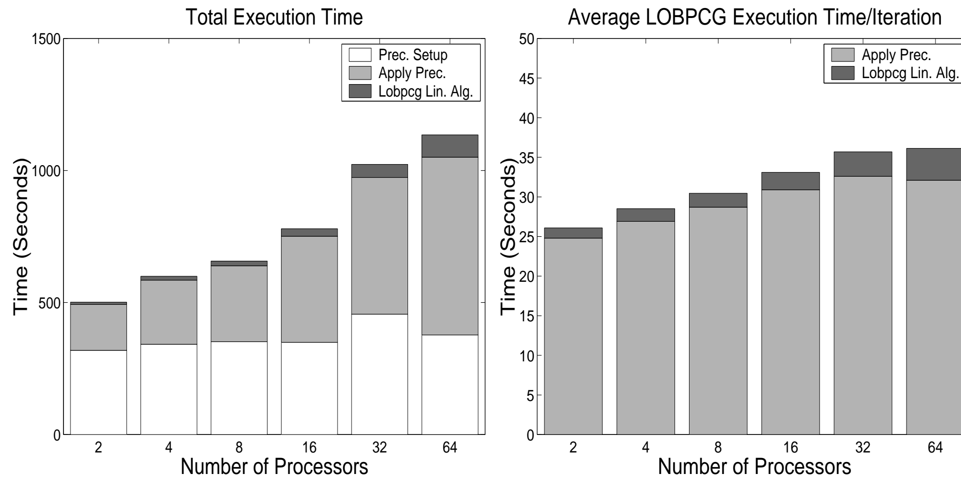

Figure 6. Execution Time to Converge as Problem Size Increases for 3–D Laplacian

- For the vast majority of the eigenvalue problems tested the Hypre LOBPCG code does not experience any problems due to instability because of ill-conditioned Gram matrices. If the ill-conditioning appears, restarts are helpful, when the matrix P is dropped from the basis of the trial subspace. Such restarts further improve the stability of the code and are planned to be implemented in the next Hypre LOBPCG release.
- This implementation 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.
- The users gain a significant leverage and flexibility in solving large sparse eigenvalue problems because the users can provide their own matrix–vector multiply and preconditioned solver functions and/or use the standard Hypre preconditioned solver functions.
- Initial scalability measurements look promising, but more testing is needed by other users. However, scalability is mainly dependent on the scalability within Hypre since most of the computations are involved with building and applying the preconditioner. In problems we tested, 90%–99% of the computational effort is required for building the preconditioner and in the applying the preconditioner during execution of the algorithm.
- Enhancements to improve robustness, efficiency and readability/clarity of code are expected to be made in the future software releases. We also plan to develop Hypre LOBPCG for generalized eigenvalue problems.
- The LOBPCG Hypre software has been integrated into the Hypre software at LLNL and has been included in the recently released Hypre Beta Release – Hypre–1.8.0b, and so is now available for users to test.

ACKNOWLEDGEMENTS

Copyright © 2005 John Wiley & Sons, Ltd.

The authors are very grateful to all members of the Scalable Algorithms Group of the Center for Applied Scientific Computing, Lawrence Livermore National Laboratory and, in particular, to Rob Falgout, Edmond Chow, Charles Tong, and Panayot Vassilevski, for their patient support and help.

Bibliography

[1] Merico E. Argentati. *Principal Angles Between Subspaces as Related to Rayleigh Quotient and Rayleigh–Ritz Inequalities with Applications to Eigenvalue Accuracy and an Eigenvalue Solver*. PhD thesis, University of Colorado at Denver, 2003.

[2] William Gropp, Ewing Lusk, and Anthony Skjellum. *Using MPI: Portable Parallel Programming with the Message–Passing Interface*. The MIT Press, Cambridge, MA, second edition edition, 1999.

[3] 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. Section 11.3. An extended version published as a technical report UCD-CCM 143, 1999, at the Center for Computational Mathematics, University of Colorado at Denver, http://www-math.cudenver.edu/ccmreports/rep143.ps.gz.

[4] A. V. Knyazev. Toward the optimal preconditioned eigensolver: Locally optimal block preconditioned conjugate gradient method. *SIAM J. Sci. Comput.*, 23(2):517–541, 2001.

[5] Lawrence Livermore National Laboratory, Center for Applied Scientific Computing (CASC), University of California. *HYPRE User's Manual - Software Version 1.6.0*, 1998.

Copyright © 2005 John Wiley & Sons, Ltd.

Prepared using nlaauth.cls