Implementation of a Scalable Preconditioned Eigenvalue Solver Using Hypre

Andrew V. Knyazev and Merico E. Argentati (speaker)

Department of Mathematics and

Center for Computational Mathematics

University of Colorado at Denver

Acknowledgement

Supported by Lawrence Livermore National Laboratory, Center for Applied Scientific Computing (LLNL–CASC).

We would like to thank Rob Falgout, Charles Tong, Panayot Vassilevski and other members of the Hypre team for their help.

Implementation of LOBPCG Eigensolver Using Hypre

1. Background concerning algorithm
2. Hypre software library
3. LOBPCG Hypre implementation strategy
4. User interface (API)
5. Compiling and Testing
6. Initial Performance Results
7. Conclusions

Background

- LOBPCG - Locally Optimal Block Preconditioned Conjugate Gradient Method
- Authors of code for Hypre – Andrew Knyazev & Merico Argentati both from the University of Colorado at Denver
- Algorithm is described in: A. V. Knyazev, Toward the Optimal Preconditioned Eigensolver: Locally Optimal Block Preconditioned Conjugate Gradient Method. SIAM Journal on Scientific Computing 23 (2001), no. 2, pp. 517-541.

LOBPCG Algorithm

- LOBPCG solver finds the smallest eigenvalues of a symmetric positive definite matrix
- The algorithm is matrix free since the multiplication of a vector by the matrix A and an application of the preconditioner T to a vector are needed only as functions
- For computing only the smallest eigenpair, the algorithm implements a local optimization of a 3-term recurrence

$$x^{n+1} \in \mathrm{span}\{x^n,\ x^{n-1},\ T(Ax^n - \lambda(x^n)x^n)\}$$

- If finding m smallest eigenpairs of A, the Rayleigh-Ritz method on a $3m$–dimensional trial subspace, is used during each iteration for the local optimization

Previously Developed LOBPCG Software

- MATLAB
- C-language based on LAPACK/BLAS libraries
- Initial PETSc version

Hypre (High Performance Preconditioners)

- Hypre is a software library for solving large, sparse linear systems on massively parallel computers
- The primary goal is to provide users with advanced parallel preconditioners
- The Hypre libraries are designed to provide robustnesss, ease of use, flexibility and interoperability

LOBPCG Hypre Implementation

- C-language
- Hypre libraries (hypre-1.6.0) and LAPACK/BLAS libraries
- Implementation is based on Hypre Linear Algebraic (IJ) Interface for applications with sparse linear systems
- User interface to solver is a “Hypre style” API
- User provided functions for MATVEC multiply and preconditioner/solve
- LOBPCG Hypre implementation utilizes Hypre parallel vector manipulation routines and MGS for orthonormalization

LOBPCG Hypre Implementation (cont.)

- Test driver IJ_eigen_solver.c, is modified version of IJ_linear_solvers.c which retains much of its functionality/capability
- Test driver allows for input of Matrix Market files and internally generated matrices (Laplacians of several different types and sizes)
- Shell script IJ_eigen_solver.sh provides basic suite of tests
- LOBPCG Hypre based code will be included in the next **Beta Release** of the **Hypre Project**

LOBPCG Software Implemented Using Hypre

IJ_eigen_solver.c

lobpcg.c | lobpcg_matrix.c | lobpcg_utilities.c

lobpcg.h | Hypre Include Files

LOBPCG User Interface (API)

```
I.  Setup Functions
HYPRE_LobpcgCreate(HYPRE_LobpcgData *lobpcg);
HYPRE_LobpcgSetup(HYPRE_LobpcgData lobpcg);
HYPRE_LobpcgSetVerbose(HYPRE_LobpcgData lobpcg);
HYPRE_LobpcgSetMaxIterations(HYPRE_LobpcgData lobpcg,int max_iter);
HYPRE_LobpcgSetTolerance(HYPRE_LobpcgData lobpcg,double tol);
HYPRE_LobpcgSetBlocksize(HYPRE_LobpcgData lobpcg,int bsize);
HYPRE_LobpcgSetSolverFunction(HYPRE_LobpcgData lobpcg,
    int (*FunctSolver)(HYPRE_ParVector x,HYPRE_ParVector y));
HYPRE_LobpcgDestroy(HYPRE_LobpcgData lobpcg);

II. Lobpcg Solver
HYPRE_LobpcgSolve(HYPRE_LobpcgData lobpcgdata,
    int (*FunctA)(HYPRE_ParVector x,HYPRE_ParVector y),
    HYPRE_ParVector *v,double **eigval);

III. Output Functions
HYPRE_LobpcgGetEigval(HYPRE_LobpcgData lobpcg,double **eigval);
HYPRE_LobpcgGetResvec(HYPRE_LobpcgData lobpcg,double ***resvec);
HYPRE_LobpcgGetEigvalHistory(HYPRE_LobpcgData lobpcg,double ***eigvalhistory);
```

Compiling and Testing

- Make files for various hardware configurations and compilers are provided
- A test program IJ_eigen_solver.sh, similar to IJ_linear_solvers.sh, has been developed
- LOBPCG code has been compiled and tested on the CU Denver cluster using scali and mpich libraries and using gcc, pgcc and mpicc compilers

Compiling and Testing (cont.)

- The beowulf cluster at CU Denver includes:
  - 36 nodes, 2 processors each
  - each node: 2 PIII 933MHz, 2GB memory
  - linux Redhat 7.2
  - SCI Dolpin interconnect, management interconnect
- Lobpcg has been compiled and tested on a subset of the OCF Production Systems at LLNL, including ASCI blue, M&IC tera, gps and lx

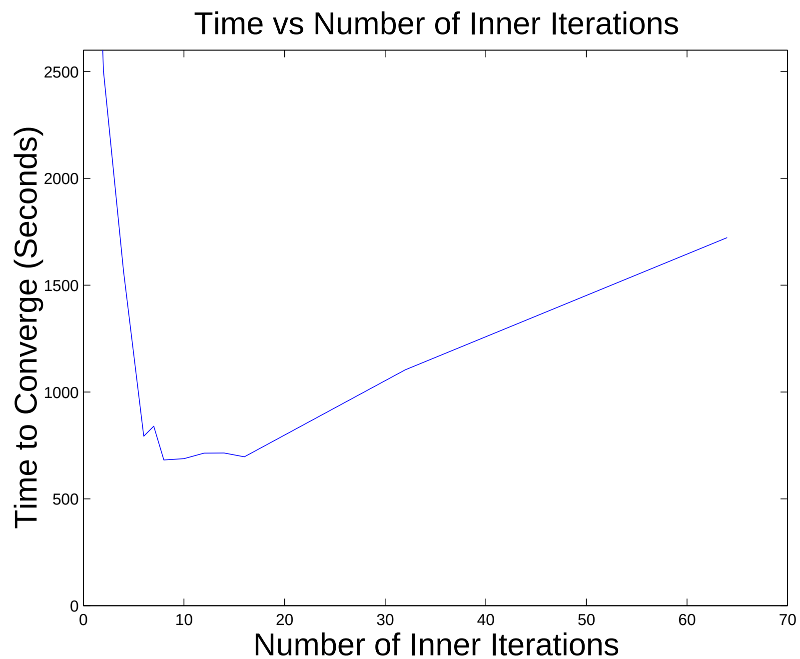
Compiling and Testing (cont.)

- Testing has been done using a variety of internally generated Laplacians and several matrix market files that were used as input
- Quality and accuracy of code, using multiple processors, seems to be quite good based in this limited testing
- Preconditioning is implemented through calls to Hypre preconditioned PCG linear solvers. In our tests, only a few (2-10) inner iterations typically provide the best performance and increasing the number of the inner iterations does not improve the final convergence

Hypre Preconditioners Tested with LOBPCG

- 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

LOBPCG Performance vs Preconditioner Iterations



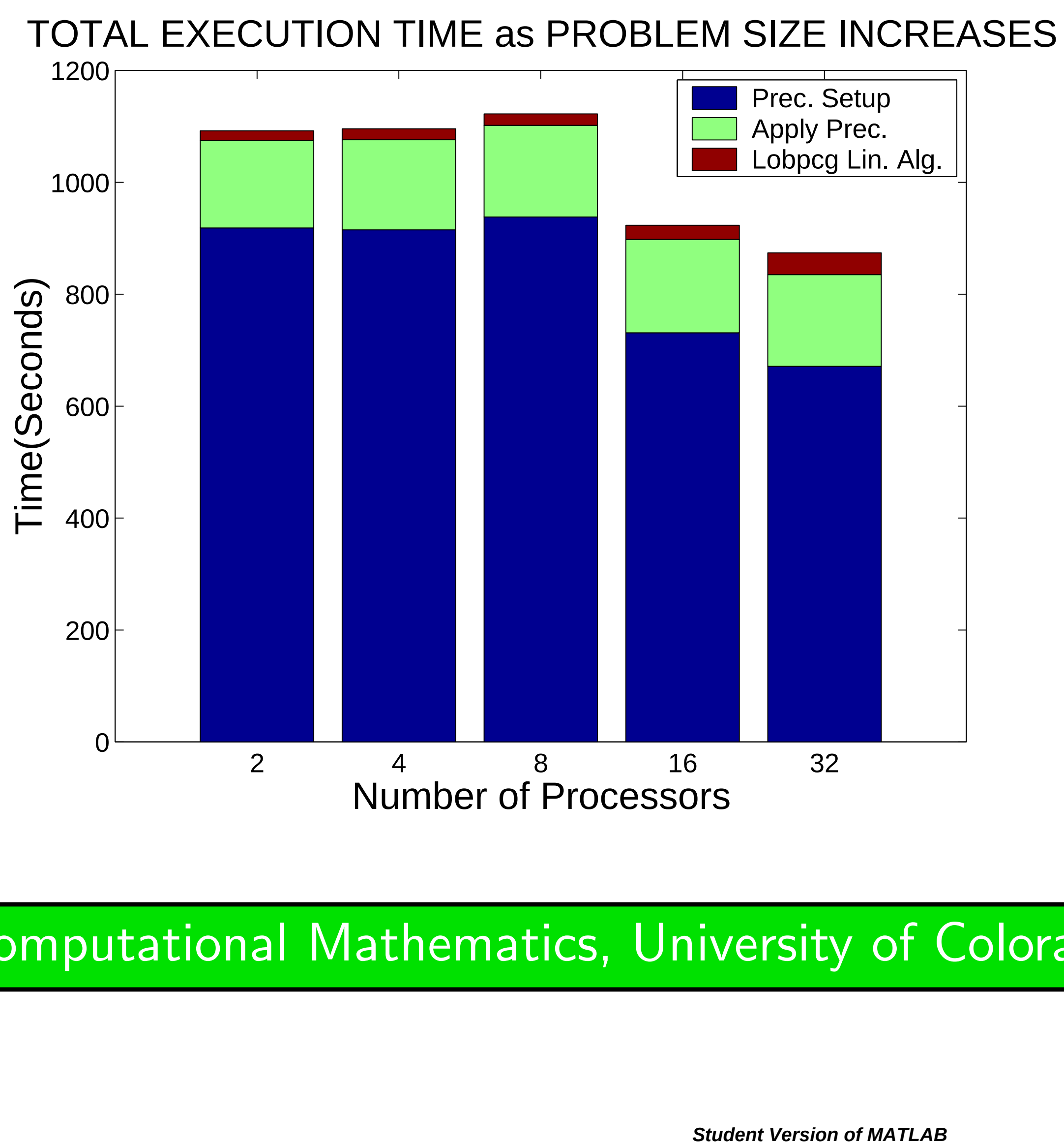
7–Point 3-D Laplacian, 8,000,000 x 8,000,000 matrix, 55,760,000 nz, Preconditioner/Solve: Schwarz-PCG, 10 Processors.

Timing Results – Scalability

Nproc	Nz	Prec. setup	Apply Prec.	LOBPCG Lin. Alg.
2	6,940,000	918.5	155.7	17.7
4	13,907,376	914.9	161.1	19.4
8	27,986,067	937.9	163.9	20.3
16	55,760,000	730.9	166.9	25.3
32	112,000,000	671.0	163.8	38.9

7–Point 3–D Laplacian, Block size: 1, Lobpcg iterations: 10, Inner (pcg) interations: 3, Preconditioner/Solve: Schwarz-PCG.

Timing Results – Scalability (cont.)



Conclusions

- Implementation illustrates that the LOBPCG matrix free algorithm can be implemented using parallel libraries
- User interface routines
  - provide ease of use for a variety of users
  - have been developed with the goal of using the Hypre standard API
  - provide flexible capability for the user to provide MATVEC multiply and preconditioned solver functions, which leverage the power of Hypre preconditioners
- Initial scalability looks promising, but more testing is needed by other users on larger problems
- Enhancements are possible to improve robustness, efficiency and readability/clarity of code