hypre: a Library of High Performance Preconditioners

Robert D. Falgout and Ulrike Meier Yang

Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, P.O.Box 808, L-560 Livermore, CA 94551

Abstract. *hypre* is a software library for the solution of large, sparse linear systems on massively parallel computers. Its emphasis is on modern powerful and scalable preconditioners. *hypre* provides various conceptual interfaces to enable application users to access the library in the way they naturally think about their problems. This paper presents the conceptual interfaces in *hypre*. An overview of the preconditioners that are available in *hypre* is given, including some numerical results that show the efficiency of the library.

1 Introduction

The increasing demands of computationally challenging applications and the advance of larger more powerful computers with more complicated architectures have necessitated the development of new solvers and preconditioners. Since the implementation of these methods is quite complex, the use of high performance libraries with the newest efficient solvers and preconditioners becomes more important for promulgating their use into applications with relative ease.

hypre has been designed with the primary goal of providing users with advanced scalable parallel preconditioners. Issues of robustness, ease of use, flexibility and interoperability have also been very important. It can be used both as a solver package and as a framework for algorithm development. Its object model is more general and flexible than the current generation of solver libraries [7].

hypre also provides several of the most commonly used solvers, such as conjugate gradient for symmetric systems or GMRES for nonsymmetric systems to be used in conjunction with the preconditioners.

Design innovations have been made to enable application users access to the library in the way that they naturally think about their problems. For example, applications developers that use structured grids, typically think of their problems in terms of stencils or grids. *hypre*'s users do not have to learn complicated sparse matrix structures; instead *hypre* does the work of building these data structures through various conceptual interfaces. The conceptual interfaces currently implemented include stencil-based structured/semi-structured interfaces, a finite-element based unstructured interface, and a traditional linear-algebra based interface.

The first part of this paper describes these interfaces and the motivations behind their design. The second part gives an overview of the preconditioners that are currently in the library with brief descriptions of the algorithms and some highlights of their performance characteristics. Since space is limited, it is not possible to describe the algorithms in detail, but various references are included for those who are interested in further information. The paper concludes with some remarks on additional software and improvements of already existing codes that are planned to be included in *hypre* in the future.

2 Conceptual Interfaces

Each application to be implemented lends itself to natural ways of thinking of the problem. If the application uses structured grids, a natural way of formulating it would be in terms of grids and stencils, whereas for an application that uses unstructured grids and finite elements it is more natural to access the preconditioners and solvers via elements and element stiffness matrices. Consequently the provision of various interfaces facilitates the use of the library.

Conceptual interfaces also decrease the coding burden for users. The most common interface used in libraries today is a linear-algebraic one. This interface requires that the user compute the mapping of their discretization to row-column entries in a matrix. This code can be quite complex, e.g. consider the problem of ordering the equations and unknowns on the composite grids used in structured adaptive mesh refinement (SAMR) codes. The use of a conceptual interface merely requires the user to input the information that defines the problem to be solved, leaving the forming of the actual linear system as a library implementation detail hidden from the user.

Another reason for conceptual interfaces, maybe the most compelling one, is that they provide access to a large array of powerful scalable linear solvers that need the extra information beyond just the matrix. For example, geometric multigrid (GMG) can not be used through a linear-algebraic interface, since it is formulated in terms of grids.

Similarly, in many cases, these interfaces allow the use of other data storage schemes with less memory overhead and provide for more efficient computational kernels.

Fig. 1 illustrates the idea of conceptual interfaces. On the left are specific interfaces with algorithms and data structures that take advantage of more specific information. On the right are more general interfaces, algorithms and data structures. Note that the more specific interfaces also give users access to general solvers like algebraic multigrid (AMG) or incomplete LU factorization (ILU). The top row shows various concepts: structured grids, composite grids, unstructured grids or just plain matrices. In the second row, various solvers/ preconditioners are listed. Each of those requires different information from the user, which is provided through the conceptual interfaces. Geometric multigrid, e.g., needs a structured grid and can only be used with the left most interface, AMGe [2], an algebraic multigrid method, needs finite element information, whereas

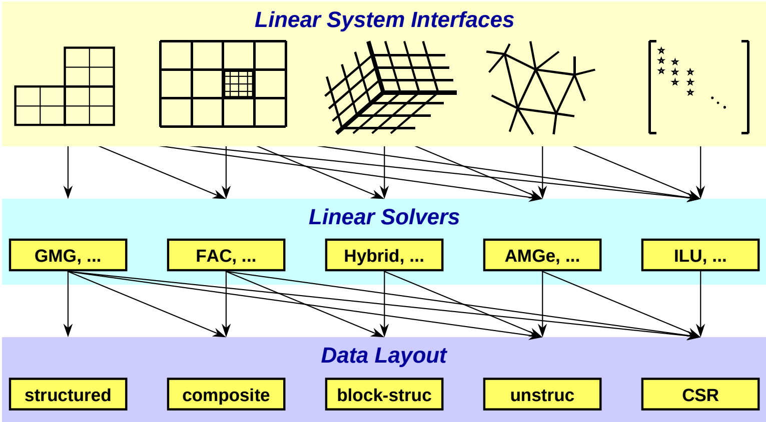

Fig. 1. Graphic illustrating the notion of conceptual interfaces.

general solvers can be used with any interface. The bottom row contains a list of data layouts or matrix/vector storage schemes that can be used for the implementation of the various algorithms. The relationship between linear solver and storage scheme is similar to that of interface and linear solver.

hypre currently supports four conceptual interfaces: a structured-grid system interface, a semi-structured-grid system interface, a finite-element interface and a linear-algebraic interface.

Note that *hypre* does not partition the problem, but builds the internal parallel data structures (often quite complicated) according to the partitioning of the application that the user provides.

2.1 Structured-Grid System Interface (Struct)

This interface is appropriate for scalar applications whose grids consists of unions of logically rectangular grids with a fixed stencil pattern of nonzeros at each grid point. It also enables users access to *hypre*'s most efficient scalable solvers for scalar structured-grid applications, such as the geometric multigrid methods SMG and PFMG. See also Sections 3.1 and 3.2. The user defines the stencil and the grid; the right hand side and the matrix are then defined in terms of the stencil and the grid.

2.2 Semi-Structured-Grid System Interface (SStruct)

This interface is appropriate for applications whose grids are mostly structured, but with some unstructured features, e.g. block structured grids (such as shown in Fig. 2), composite grids in structured adapative mesh refinement (AMR) applications, and overset grids. It additionally allows for more general PDEs than the Struct interface, such as multiple variables (system PDEs) or multiple variable types (e.g. cell centered, face centered, etc.). The user needs to define stencils, grids, a graph that connects the various components of the final grid, the right hand side and the matrix.

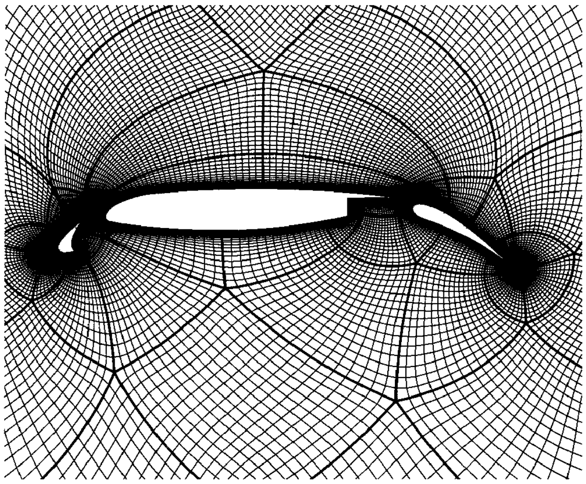

Fig. 2. An example block-structured grid, distributed across many processors.

2.3 Finite Element Interface (FEI)

This is appropriate for users who form their systems from a finite element discretization. The interface mirrors typical finite element data structures, including element stiffness matrices. Though this interface is provided in *hypre* , its definition was determined elsewhere [8]. This interface requires the definition of the element stiffness matrices and element connectivities. The mapping to the data structure of the underlying solver is then performed by the interface.

2.4 Linear-Algebraic System Interface (IJ)

This is the traditional linear-algebraic interface. The user needs to define the right hand side and the matrix in the general linear-algebraic sense, i.e. in terms of row and column indices. This interface provides access only to the most general data structures and solvers and as such should only be used when none of the grid-based interfaces is applicable.

3 Preconditioners

This section gives an overview of the preconditioners currently available in *hypre* via the conceptual interfaces. *hypre* also provides solvers to be used in conjunction with the preconditioners such as Jacobi, conjugate gradient and GMRES.

Great efforts have been made to generate highly efficient codes. Of particular concern has been the scalability of the solvers. Roughly speaking, a method is scalable if the time required to produce the solution remains essentially constant as both the problem size and the computing resources increase. All methods implemented here are generally scalable per iteration step, the multigrid methods are also scalable with regard to iteration count.

All the solvers use MPI for parallel processing. Most of them have also been threaded using OpenMP, making it possible to run *hypre* in a mixed message-passing/threaded mode, of potential benefit on clusters of SMPs.

3.1 SMG

SMG is a parallel semicoarsening multigrid solver targeted at the linear systems arising from finite difference, finite volume, or finite element discretizations of the diffusion equation

$$\nabla \cdot (D \nabla u) + \sigma u = f \tag{1}$$

on logically rectangular grids. The code solves both 2D and 3D problems with discretization stencils of up to 9-point in 2D and up to 27-point in 3D. For details on the algorithm and its parallel implementation/performance see [21, 3, 10]. SMG is a particularly robust method. The algorithm semicoarsens in the z-direction and uses plane smoothing. The xy plane solves are effected by one V-cycle of the 2D SMG algorithm, which semicoarsens in the y-direction and uses line smoothing

3.2 PFMG

PFMG is a parallel semicoarsening multigrid solver similar to SMG. It is described in detail in [1, 10]. PFMG uses simple pointwise smoothing instead of plane smoothing. As a result, it is less robust than SMG, but more efficient per V-cycle. The largest run with PFMG as a preconditioner for conjugate gradient was applied to a problem with 1 billion unknowns on 3150 processors of the ASCI Red computer and took only 54 seconds. Recently we added a PFMG solver for systems of PDEs available through the semi-structured interface.

3.3 *BoomerAMG*

BoomerAMG is a parallel implemenation of algebraic multigrid. It requires only the linear system. *BoomerAMG* uses two types of parallel coarsening strategies. The first one, refered to as RS-based coarsening, is based on the highly sequential coarsening strategy used in classical AMG [20]. To obtain parallelism, each processor coarsens independently, followed by various strategies for dealing with the processor boundaries. Obviously, this approach depends on the number of processors and on the distribution of the domain across processors. The second type of coarsening, called CLJP-coarsening [9], is based on parallel maximum independent set algorithms [19, 16] and generates a processor independent coarsening. CLJP-coarsening has proven to be more efficient for truly unstructured grids, whereas RS-based coarsenings lead to better results on structured problems. For more detailed information on the implementation of the CLJP coarsening scheme see [11]. For a general description of the coarsening schemes and the interpolation used within *BoomerAMG* as well as various numerical results, see [12].

BoomerAMG provides classical pointwise smoothers, such as weighted Jacobi relaxation, a hybrid Gauß-Seidel/ Jacobi relaxation scheme and its symmetric variant. It also provides more expensive smoothers, such as overlapping Schwarz smoothers, as well as access to other methods in *hypre* such as ParaSails, PILUT and Euclid. These smoothers have shown to be effective for certain problems for which pointwise smoothers have failed, such as elasticity problems [22].

BoomerAMG can also be used for solving systems of PDEs if given the additional information on the multiple variables per points. The function or 'unknown' approach coarsens each physical variable separately and interpolates only within variables of the same type. By exploiting the system nature of the problem, this approach often leads to significantly improved performance, lower memory usage and better scalability. See Table 1 which contains results for a structured 2-dimensional elasticity problem on the unit square, run on the ASCI Blue Pacific computer.

grid size	# of procs.	scalar *BoomerAMG* time (# of its.)	systems *BoomerAMG* time (# of its)
80×80	1	42.4(58)	4.1 (8)
160×160	4	130.4(112)	6.3 (9)
320×320	16	317.5(232)	8.6(10)
640×640	64	1238.2(684)	14.4(13)

Table 1. Test results for a 2-dimensional model elasticity problem

Table 2 contains results for a 3-dimensional elasticity problem on a thin plate with a circular hole in its center. The problem has 215,055 variables and was run on 16 processors of the ASCI White computer. The results show that for this problem *BoomerAMG* as a solver is not sufficient, but it does make an effective preconditioner.

3.4 ParaSails

ParaSails is a parallel implementation of a sparse approximate inverse preconditioner. It approximates the inverse of A by a sparse matrix M by minimizing the Frobenius norm of $I - AM$. It uses graph theory to predict good sparsity patterns for M. ParaSails has been shown to be an efficient preconditioner for many problems, particularly since the minimization of the Frobenius norm of $I - AM$ can be decomposed into minimization problems for the individual rows of $I - AM$,

Solvers	# of its.	total time in secs.
scaled CG	1665	34.8
ParaSails-CG	483	26.6
scalar *BoomerAMG*	n.c.	-
scalar *BoomerAMG*-CG	53	28.9
systems *BoomerAMG*	78	40.6
systems *BoomerAMG*-CG	19	12.3

Table 2. Test results for an elasticity problem

leading to a highly parallel algorithm. A detailed description of the algorithm can be found in [4] and implementation details in [5]. Particular emphasis has been placed on a highly efficient implementation that incorporates special, more efficient treatment of symmetric positive definite matrices and load balancing. The end result is a code that has a very scalable setup phase and iteration steps. See Table 3, which shows test results for ParaSails applied to the 3-dimensional constant coefficient anisotropic diffusion problem $0.1u_{xx} + u_{yy} + 10u_{zz} = 1$ with Dirichlet boundary conditions. The local problem size is $60 \times 60 \times 60$. Unlike multigrid, convergence is not linearly scalable, and the number of iterations will increase as the problem size increases. However, ParaSails is a general purpose solver and can work well on problems where multigrid does not.

# of procs	# of its.	setup time	solve time	time per it.
1	107	12.1	75.3	0.70
8	204	13.8	247.9	1.22
64	399	15.4	536.6	1.34
216	595	15.8	856.4	1.44
512	790	17.4	1278.8	1.62
1000	979	17.1	1710.7	1.75

Table 3. Scalability of ParaSails with increasing problem size (216,000 per proc.)

3.5 PILUT

PILUT is a parallel preconditioner based on Saad's dual-threshold incomplete factorization algorithm. It uses a thresholding drop strategy as well as a mechanism to control the maximum size of the ILU factors. It uses the Schur-complement approach to generate parallelism. The original code was written by Karypis and Kumar for the T3D [18]. This version differs from the original version in that it uses MPI and more coarse-grain parallelism.

3.6 Euclid

Euclid is a scalable implementation of the Parallel ILU algorithm. It is best thought of as an "extensible ILU preconditioning framework", i.e. Euclid can support many variants of ILU(k) and ILUT preconditionings. Currently it supports Block Jacobi ILU(k) and Parallel ILU(k) methods. Parallelism is obtained via local and global reorderings of the matrix coefficients. A detailed description of the algorithms can be found in [14, 15].

Euclid has been shown to be very scalable with regard to setup time and triangular solves. Fig. 3 shows results for a 5 point 2D convection diffusion problem with 256×256 unknowns per processor.

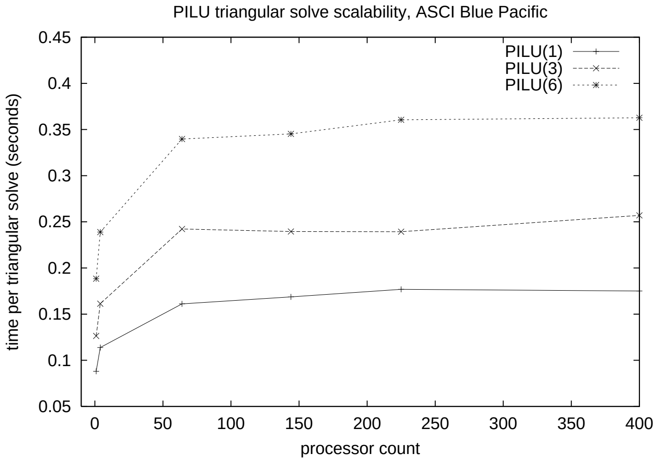

Fig. 3. Some scalability results for Euclid

4 Additional Information

The *hypre* library can be downloaded by visiting the *hypre* home page at the URL `http://www.llnl.gov/CASC/hypre`. It can be built by typing `configure` followed by `make`. There are several options that can be used with configure. For information on how to use those, one needs to type `configure --help`. Although *hypre* is written in C, it can also be called from Fortran. More specific information on *hypre* and how to use it can be found in the users manual and the reference manual, which are also available at the same URL.

5 Conclusions and Future Work

Overall, *hypre* contains a variety of highly efficient preconditioners and solvers, available via user-friendly conceptual interfaces. Nevertheless, it is a project in progress. As new research leads to better and more efficient algorithms, new preconditioners will be added and old preconditioners will be improved.

On the list of new codes to be made available shortly is AMGe, an algebraic multigrid method based on the use of local finite element stiffness matrices [2, 17]. This method has proven to be more robust and to converge faster than classical AMG for some problems, e.g. elasticity problems. This code will be available directly through the FEI interface.

Various improvements are planned for *BoomerAMG*. Classical Gauß-Seidel relaxation as well as multiplicative Schwarz smoothers are some of the numerically most efficient methods, i.e. they lead to good convergence for AMG for some problems, but are also highly sequential. Plans are to add multi-coloring techniques to obtain a parallel Gauß-Seidel smoother and parallel multiplicative Schwarz smoothers, as well as introduce smoothing and overrelaxation parameters to increase convergence of the currently available parallel smoothers. New research [6] has shown that through the use of certain geometric components, better coarsenings can be developed that may lead to better convergence and lower memory requirements for certain problems. Investigations are underway to make these new techniques available to the users.

Acknowledgments

This paper would not have been possible without the many contributions of the *hypre* library developers: Edmond Chow, Andy Cleary, Van Henson, David Hysom, Jim Jones, Mike Lambert, Jeff Painter, Charles Tong and Tom Treadway. This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.

References

1. Ashby, S., Falgout, R.: A parallel multigrid preconditioned conjugate gradient algorithm for groundwater flow simulations. Nuclear Science and Engineering **124** (1996) 145–159
2. Brezina, M., Cleary, A., Falgout, R., Henson, V., Jones, J., Manteuffel, T., McCormick, S, Ruge, J.: Algebraic multigrid based on element interpolation (AMGe). SIAM J. Sci. Comput. **22** (2000) 1570–1592
3. Brown, P., Falgout, R., Jones, J.: Semicoarsening multigrid on distributed memory machines. SIAM J. Sci. Comput. **21** (2000) 1823-1834
4. Chow, E.: A priori sparsity patterns for parallel sparse approximate inverse preconditioners. SIAM J. Sci. Comput. **21** (2000) 1804-1822
5. Chow, E.: Parallel implementation and practical use of sparse approximate inverses with a priori sparsity patterns. Int'l J. High Perf. Comput. Appl. **15** (2001) 56–74
6. Chow, E.: An unstructured multigrid method based on geometric smoothness. submitted to Num. Lin. Alg. Appl. Also available as Lawrence Livermore National Laboratory technical report UCRL-JC-145075 (2001)
7. Chow, E., Cleary, A., Falgout, R.: Design of the *hypre* preconditioner library. In Henderson, M., Anderson, C., Lyons, S., eds: Proc. of the SIAM Workshop on Object Oriented Methods for Inter-operable Scientific and Engineering Computing (1998) SIAM Press

8. Clay, R. et al.: An annotated reference guide to the Finite Element Interface (FEI) specification, version 1.0. Technical Report SAND99-8229, Sandia National Laboratories, Livermore, CA (1999)
9. Cleary, A., Falgout, R., Henson, V., Jones, J.: Coarse-grid selection for parallel algebraic multigrid. in Proc. of the 5th Intern. Sympos. on Solving Irregularly Structured Problems in Parallel, Lecture Notes in Computer Science **1457** (1998) 104–115
10. Falgout, R., Jones, J.: Multigrid on massively parallel architectures. In Dick, E., Riemslagh, K., and Vierendeels, J., eds: Multigrid Methods VI, Lecture Notes in Computational Science and Engineering, vol. **14** (2000) 101–107, Berlin. Springer
11. Gallivan, K., Yang, U. M.: Efficiency issues in parallel coarsening schemes. LLNL technical report (2001)
12. Henson, V. E., Yang, U. M.: *BoomerAMG*: a parallel algebraic multigrid solver and preconditioner. To appear in Applied Numerical Mathemaitics. Also available as LLNL technical report UCRL-JC-133948 (2000)
13. Henson, V.E., Vassilevski, P.: Element-free AMGe: General algorithms for computing interpolation weights in AMG. to appear in SIAM J. Sci. Comput. Also available as LLNL technical report UCRL-JC-139098
14. Hysom, D., Pothen, A.: Efficient parallel computation of ILU(k) preconditioners. SC99, ACM (1999), CDROM, ISBN #1-58113-091-0, ACM Order #415990, IEEE Computer Society Press Order # RS00197
15. Hysom, D., Pothen, A.: A scalable parallel algorithm for incomplete factor preconditioning. SIAM J. Sci. Comput. **22** (2001) 2194–2215
16. Jones, M., Plassman, P.; A parallel graph coloring heuristic. SIAM J. Sci. Comput. **14** (1993) 654–669
17. Jones, J., Vassilevski, P.: AMGe based on element agglomeration. to appear in SIAM J. Sci. Comput. Also available as LLNL technical report UCRL-JC-135441
18. Karypis, G., Kumar, V.: Parallel threshold-based ILU factorization. Technical Report 061 (1998) University of Minnesota, Department of Computer Science/ Army HPC Research Center, Minneapolis, MN
19. Luby, M.: A simple parallel algorithm for the maximal independent set problem. SIAM J. on Computing **15** (1986) 1036–1053
20. Ruge, J., Stüben, K.: Algebraic Multigrid (AMG). in McCormick, S., ed. Multigrid Methods, Frontiers in Applied Mathematics vol. **3** (1987) 73–130, SIAM, Philadelphia
21. Schaffer, S.: A semi-coarsening multigrid method for elliptic partial differential equations with highly discontinuous and anisotropic coefficients. SIAM J. Sci. Comput. **20** (1998) 228–242
22. Yang, U. M.: On the use of Schwarz smoothing in AMG. 10th Copper Mt. Conf. Multigrid Meth.. Also available as LLNL technical report UCRL-VG-142120 (2001)