EigAdept - The Expert Eigensolver Toolbox

Xiaoye S. Li[1], Osni A. Marques[1] and Yeliang Zhang[2]

[1] Lawrence Berkeley National Laboratory,
1 Cyclotron Road, MS 50F-1650, Berkeley, CA 94720-8139, USA
[xsli,oamarques]@lbl.gov
[2] Electrical and Computer Engineering Dept., The University of Arizona
Tucson, AZ 85721
zhang@ece.arizona.edu

Abstract. The computation of eigenvalues and eigenvectors is an important and often time-consuming phase in a large range of computer simulations. Recent efforts in the development of eigensolver libraries have given users robust algorithms without the need for users to spend much time in programming. Yet, given the variety of numerical algorithms that are available to domain scientists, choosing the "best" algorithm for increasingly sophisticated and larger applications is a daunting task. In this paper, we discuss a methodology and a software toolbox that aims at guiding the user through the maze of various eigensolvers with different configurations, and at determining the best eigensolver based on the application type and the matrix properties.

1 Introduction and Motivation

The computation of eigenvalues and eigenvectors is an important and often time-consuming phase in computer simulations, including the study of nuclear reactor dynamics, finite element dynamic analysis of structural models, design of particle accelerators, the solution of Schrödinger's equation in chemistry and physics, the design of microelectromechanical systems, etc. Because of the need for higher accuracy and higher level of details in the models, the size and complexity of the computations grow as fast as the advancement of the computer hardware.

In order to cope with the increasing need for solving large-scale eigenvalue problems, various (sequential and parallel) numerical algorithms have been developed [1, 2]. With the growing availability of good implementations, domain scientists are no longer facing the problem of a lack of algorithms to use but rather too many algorithms to choose from. (In this paper, "algorithm" is interchangeable with "eigensolver".) However, little consideration is given to mechanisms that provide an effective way for a user to sift out the more appropriate eigensolver for a particular application. In general, a suitable choice may have an order of magnitude impact on performance compared to a bad choice.

We identify the following challenges for the selection of the "best" eigensolver: *i*) Given the vast number of algorithms and computer architectures, how to facilitate the selection process for a non-expert user. Different algorithms have

different convergence behaviors, memory requirements, and trade-offs between accuracy and performance. It is therefore difficult and tedious to manually track these metrics; ideally, the selection should be done automatically. *ii*) There is no unified software framework that facilitates swapping among different algorithm implementations. The *Eigentemplates* book [3] provides an excellent algorithmic guideline but to fully appreciate that book a user must be equipped with sufficient knowledge of numerical analysis and programming skills, in particular on high-end computers. *iii*) Usually, there are a large number of parameters associated with each algorithm, which can be adjusted in order to make the algorithm perform more efficiently. Even if there were a collection of libraries with a unified interface, it becomes unfeasible for a potential user to try out every algorithm configuration in a timely fashion.

The objective of this work is to address some of the aforementioned challenges, and present a design and methodology towards an automatic application-based eigensolver selection toolbox, EigAdept, which also incorporates an intelligent engine. In the following sections we summarize the system architecture, how it is implemented and used, what are its major components and how the components are connected.

2 Methodology

Choosing an appropriate eigensolver (and its parameters) depends on the mathematical properties of the problem, the desired spectral information, and the available operations and their costs. The Eigentemplates book [3] provides valuable "decision trees" based on such information but given the large, high-dimensional algorithm/parameter space, a simple decision-tree mechanism may be insufficient. Furthermore, some information may not be available at runtime or may be costly to obtain. EigAdept intends to address the dilemma of "less information but accurate solving" by providing algorithm recommendations based on previous knowledge. Its main components are (see [4] for details):

1. A *Universal User Interface*. For different algorithms with different parameters, it is much easier for the user to invoke a unique interface (function), which will then select and execute the algorithm, and return the result to the user. This process is transparent to the user, although the user can also specify her preferences for any desired algorithm or execution criteria (such as memory limit, convergence rate, iteration counts, etc.)
2. A *Data Analyzer*. It receives the input data submitted by the user, and extracts and passes the necessary information required by the intelligent engine (described below) to make an algorithm-wise decision. If some information is missing, the intelligent engine still tries to find a suitable algorithm providing a similar problem has been previously solved.
3. *Decision Trees*. It is a repository of decision trees incorporated from [3].
4. A *Relational Database*. The database initially contains some training sets from which the best eigensolver for certain applications are known beforehand. Information about each new application solved is stored in the database

with all the application properties and eigensolver information. The database gradually improves its contents as more problems are solved, thereby making the algorithm prediction increasingly accurate. The database is implemented with MySQL, which is a free open source database software [5].

5. An *Intelligent Engine*. This is the central part of the EigAdept system. It receives information from the data analyzer and searches the decision tree for an appropriate algorithm. If the decision tree does not lead to any match, the intelligent engine queries the database for a suitable eigensolver based on the application characteristics (and possibly the user's preferred algorithm properties).
6. *Numerical Libraries*. This is a collection of the currently available libraries providing eigensolvers. Once an algorithm is chosen by the intelligent engine, it is the intelligent engine's task to pass the parameters required by the underlying library.
7. *Computational Environment*. This is where the application is executed. It can be a stand alone desktop, a Linux cluster, or a hybrid of MPP and shared memory machines.

3 Current and Future Work

Automatically choosing an optimal eigensolver is in many ways more difficult than doing so for linear solvers. In eigenvalue analyses, some of the important data to be taken into account include the dimension of the problem, number of eigenvalues (and/or eigenvectors) required, location of the required solutions in the spectrum (i.e. smallest, largest, close to a reference value, etc.), accuracy of the required solutions, availability of approximate solutions (e.g., obtained from previous simulations with similar problems) [3], available memory (which can influence the effectiveness of restarting strategies), etc. For many large-scale applications an iterative scheme is the method of choice. However, some simulations may require increasing number of eigenvalues to be computed from one run to another. In this case, we may reach a breaking point where switching to a direct method becomes a viable alternative [1].

We focus on a class of eigensolvers based on projection methods, which transform the original eigenvalue problem into a problem associated with an appropriate subspace of much reduced dimension, and find the best approximations from this reduced subspace. These methods are amenable to scalable implementations and have already been implemented in various libraries, including PARPACK [6], BLZPACK [7], JaDa [8], and TRLan [9]. We will enhance some of these eigensolvers with shift-and-invert capabilities using existing scalable sparse direct linear solvers, such as SuperLU [10] and MUMPS [11].

There exists a number of research projects seeking goals similar to ours but using different approaches [12, 13] and offering different functionalities. Our work targets large-scale eigenvalue applications without requiring any special format to describe the input data. Instead, information comes from the heuristic database and the data analyzer that examines the user input data. The intelligent

engine acts as a middleware to connect the user application and the numerical libraries. The user does not need to change her application code if she wants to use another library. The combination of on-line and off-line mechanisms adaptively increase the "wisdom" of the intelligent engine algorithm selection procedure.

EigAdept is implemented in C++, which provides for a convenient way of adding new capabilities. Polymorphism enables EigAdept to solve a problem using the same library but with different input parameters. It also allows for good interoperability with libraries written in C or Fortran. At the time of this writing, three major classes have been implemented: `LINSolver` (Linear Solver)class, `EIGSolver` (Eigenvalue solver) class and `Options` (Options for EIGSolver) class. For sparse matrices, we support both Compress Row Storage (CRS) and Compressed Column Storage (CCS) schemes. Our future work will focus on better defining these classes and the information to be stored in the relational database.

References

1. L. S. Blackford, J. Choi, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, ScaLAPACK Users' Guide, SIAM, Philadelphia, 1997.
2. V. Hernandez, J. Roman, A. Tomas and V. Vidal, A Survey of Software for Sparse Eigenvalue Problems, SLEPc Technical Report STR-6, Universidad Politecnica de Valencia, 2005.
3. R. Barret et. al., Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, SIAM, Philadelphia, 1994.
4. Y. Zhang, X.S. Li and O. Marques, Towards an Automatic and Application-Based Eigensolvers Selection, LACSI Symposium 2005, October 11-13, Santa Fe, NM.
5. L. Welling and L. Thomson, MySQL Tutorial, MySQL Press, 2003.
6. R. Lehoucq, K. Maschhoff, D. Sorensen and Chao Yang, Parallel ARPACK, http://www.caam.rice.edu/software/ARPACK.
7. O. Marques, BLZPACK: Description and User's Guide, Technical Report TR/PA/95/30, CERFACS, Toulouse, France, 1995.
8. A. Stathopoulos and J. R. McCombs, A Parallel, Block, Jacobi-Davidson Implementation for Solving Large Eigenproblems on Coarse Grain Environments, Proc. of the Int. Conf. on Parallel and Distributed Processing, 1999.
9. K. Wu and H. Simon, Thick-restart Lanczos method for large symmetric eigenvalue problems, SIAM J. Matrix Anal. Appl., 2001(22):602–616.
10. J. W. Demmel, J. R. Gilbert and X. S. Li, SuperLU Users' Guide, Technical Report LBNL-44289, Lawrence Berkeley National Laboratory, 1999.
11. P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent and J. Koster, MUltifrontal Massively Parallel Solver (MUMPS Version 4.3), Users' Guide, ENSEEIHT-IRIT, Toulouse, France, 2003.
12. J. Dongarra and V. Eijkhout, Self-Adapting Numerical Software for Next Generation Applications, The International Journal of High Performance Computing Applications, Volume 17 Number 2 Summer 2003.
13. K. Seymour, A. Yarkhan, S. Agrawal, and J. Dongarra. NetSolve: Grid Enabling Scientific Computing Environments, Grid Computing and New Frontiers of High Performance Processing, L. Grandinetti, Ed., Elsevier, 2005.