Generating Customized Eigenvalue Solutions Using Lighthouse

Luke Groeninger
Department of Computer Science
University of Colorado Boulder
luke.groeninger@colorado.edu

Ramya Nair
Department of Computer Science
University of Colorado, Boulder
ramya.nair@colorado.edu

Sa-Lin Cheng Bernstein
Computation Institute
University of Chicago
salinb@uchicago.edu

Javed Hossain
Department of Computer Science
University of Colorado, Boulder
javed.hossain@colorado.edu

Elizabeth R. Jessup
Department of Computer Science
University of Colorado, Boulder
jessup@colorado.edu

Boyana Norris
Computation Institute
University of Chicago
boyana@uchicago.edu

INTRODUCTION

Eigenproblems have become a fundamental part of science and engineering. Indeed, at least one thing you have done today was made possible because of the solution to an eigenproblem. Whether it was because you used a search engine or speech recognition software, drove across a bridge, or flew on an aircraft, the reality is that eigenproblems are now responsible for many everyday services on which you depend. The result of this demand is that scientists and engineers across many disciplines must rely on high-performance numerical software to solve eigenproblems quickly and efficiently. However, exploiting the latest high-performance computational techniques for solving eigenproblems remains difficult. Our work addresses this problem in a way that enables users to find customized software routines for solving eigenproblems without the steep learning curve that is traditionally associated with selecting and using the appropriate solution method. This poster presents new contributions in three areas: (1) taxonomies of dense and sparse eigensolvers available in LAPACK and SLEPc, (2) user interfaces for searching the taxonomies, and (3) performance-based recommendation of solution methods based on machine-learning analysis.

1. CHALLENGES IN SELECTING EIGENSOLVERS

The two largest challenges faced by users are the wide variety of algorithm and software options from which to choose and the task of integrating specific libraries into their projects. The wide breadth of choices means that users face many options that are optimized for differing matrix properties and hardware conditions. In order to select appropriate solvers, users must understand enough about their problems and their computer systems. Unfortunately, even if a user does know enough to select an algorithm, they also have to know how to implement it. Given that linear algebra libraries are domain-specific and intended for experienced developers, it requires a good understanding of programming and software engineering to implement the desired solution. Additionally, the increasing use of parallel systems and clusters in high performance computing means that users may also need to understand how to develop software specifically for those platforms in order to actually achieve the results they seek.

2. APPROACH

The Lighthouse project addresses the aforementioned skills gap by enabling users without the requisite numerical computation or programming experience to find the right linear algebra routines for their projects. Built using modern web design techniques, Lighthouse guides a user to an efficient solution via a variety of methods including simple guided searches, advanced keyword and filter based searches, annotated documentation, pre-optimized code templates, and a user-friendly interface to expose all of these features. The underlying technology of Lighthouse is a taxonomy of linear algebra routines. Previously, Lighthouse had support only for a limited subset of linear algebra problems for solving linear systems. As a result, the implementation portion of our research dealt with adding eigensolvers to the Lighthouse framework in a way that it is easily accessible to users.

The software libraries used to solve eigenvalue problems focus on specific subsets of the eigenproblem. Hence, our task was split into two separate taxonomies: a taxonomy for eigensolvers optimized for dense matrices using LAPACK and a taxonomy optimized for sparse matrices using SLEPc. Given that a user may not know all the information necessary to find an appropriate solution, in these taxonomies we focus on finding the minimal set of questions required to identify an eigensolver that will deliver the results that the user is looking for. After determining which eigensolvers will provide the desired solutions, additional optional questions are used to find the most efficient routines among the available implementations. The eigensolver taxonomies are implemented in the existing Lighthouse framework, extending the existing taxonomy and interfaces, for dense and sparse linear system solution methods.

2.1 The Organization of Lighthouse

2.1.1 LAPACK for Dense Eigensolvers

As implemented in Lighthouse, the ubiquitous LAPACK is one of the most widely used linear algebra software libraries. LAPACK provides optimized dense matrix eigensolver implementations based on the traditional QR algorithm, the Divide and Conquer algorithm, the Relatively Robust Representations algorithm, and the Multiple Relatively Robust Representations algorithms [1]. The primary challenge in implementing support for LAPACK eigensolver routines in Lighthouse comes from the large number of routines included in it. As the routines vary based on

everything from specific algorithms used to the specific format for input matrices, picking the correct routine requires a good understanding of how LAPACK itself is organized in addition to the problem that the user wants to solve.

Fortunately, the nature of dense matrices means that various algorithms and functions that LAPACK supports only differ in terms of performance and desired outputs and generally are suitable for operating on a variety of input matrix formats. As most combinations of source matrices and desired outputs lead to a small set of suitable driver functions, the taxonomy can be implemented as a simple decision tree that leads to individual functions. By using this approach, users are able to easily find a suitable function without needing to know unnecessary details, making it appealing to novice users.

2.1.2 SLEPc for Sparse Eigensolvers

SLEPc[2] (Scalable Library for Eigenvalue Problem Computations) is a PETSc[3] (Portable, Extensible Toolkit for Scientific Computation) compatible library that provides highly optimized parallel solvers for sparse eigenproblems. SLEPc implements a wide variety of eigensolver algorithms including Krylov-Schur, Generalized Davidson, Jacobi Davidson, Arnoldi, Lanczos, and Rayleigh quotient conjugate gradient. SLEPc also provides flexibility to change eigensolver parameters such as number of eigenvalues/eigenvectors desired, type of matrix, convergence criteria, portion of eigenvalue spectrum and so on. Though such flexible options give better control, the actual implementation to solve an eigenproblem is more difficult than with LAPACK as some combinations of options may adversely affect performance while other combinations may not converge on a solution at all. We solved this issue by experimentally determining the most appropriate implementation of these solvers so that the user does not need to understand the specific algorithms themselves.

To build the taxonomy we experimentally determined the performance of each eigensolver based on a variety of input and output parameters. Since similar input sets might result in very different solvers, we ran these solvers over a large number of input and output permutations using different source matrices and tabulated the results. For real matrices alone, 10000+ experimental results are collected. The most appropriate solver for each input case is then decided based on the convergence and accuracy of the eigenvalues and eigenvectors.

In order to make sense of this large amount of data, we used machine learning techniques such as decision tree analysis [4], to predict the most suitable eigensolver based on the problem characteristics. We used the tabulated results containing the most appropriate solver, as the training set to generate a decision tree using MATLAB. The generated decision tree is then used for prediction of the best solver for general cases. The predictor takes care of various input characteristics such as portion of eigenvalue spectrum, number of eigenvalues sought, number of processors and many more. Decision tree results were then verified using cross-validation.

This analysis has already been conducted for real matrices and many interesting results are noted. For example, while finding the smallest real eigenvalue for symmetric matrices, the Lanczos eigensolver seem to give better results for most cases, whereas for nonsymmetric matrices, KrylovSchur seems optimum for most cases. After cross-validation, the error for this tree (for real matrices) is 0.0333 and the standard error is 0.0038.

This predictor, after completion, is used to decide the eigensolver for user specific cases. This functionality will be incorporated into the Lighthouse framework so that we generate the code for the resultant SLEPc eigensolver according to the user's choice of input and output conditions. This generated code is in the form of C and script files and can be used by simply adding it into the desired application's code base.

3. CONCLUSION

It is our belief that our solution appropriately solves the problems that we identified. By deriving the optimal functions for SLEPc routines through machine-learning methods and organizing LAPACK routines into a taxonomy, we have created a novel capability that reduces the barriers to using these high-performance eigensolvers, increases developer's productivity and results in faster solution times than ad hoc solution approaches. By integrating eigenproblem capabilities into the Lighthouse taxonomy web-based infrastructure, we are providing an easy-to-use interface for discovering and effective use of state-of-the-art numerical methods for solving dense and sparse eigenproblems. We are also enabling access to a broad community of users who may not otherwise be able to take advantage of high-performance implementations of modern eigensolvers.

4. REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen. LAPACK Users Guide Third Edition. Society for Industrial and Applied Mathematics, 1999. DOI=http://dx.doi.org/10.1137/1.9780898719604.

[2] C. Campos, J. E. Roman, E. Romero and A. Tomas. SLEPc Users Manual. Tech. Rep. DSIC-II/24/02 - Revision 3.3, Universitat Politècnica de València, 2012.

[3] Satish Balay, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G.Knepley, Lois Curfman McInnes, Barry F.Smith, and Hong Zhang. PETSc Webpage, 2001. http://www.mcs.anl.gov/petsc.

[4] Jiawei Han, Micheline Kamber, and Jian Pei. Data mining concepts and techniques, third edition. Morgan Kaufmann Publishers, Waltham, Mass., 2012.