

# Towards an Automatic and Application-Based Eigensolver Selection

Yeliang Zhang  
Electrical and Computer Engineering Dept.  
The University of Arizona  
Tucson, AZ 85721  
Email: zhang@ece.arizona.edu

Xiaoye S. Li  
Osni Marques  
Lawrence Berkeley National Lab  
Computational Research Division  
1 Cyclotron Road, MS 50F-1650  
Berkeley, CA 94720-8139  
Email: xsli@lbl.gov, oamarques@lbl.gov

## Abstract

The computation of eigenvalues and eigenvectors is an important and often time-consuming phase in computer simulations. The recent efforts in developing eigensolver libraries provide users the reasonable algorithms without the need for them to spend much time in programming. Yet, given the variety of the numerical algorithms that are available to the domain scientists, it is a daunting task for them to choose the “best” algorithm suited for a particular application. As simulations become increasingly sophisticated and larger, it becomes infeasible for the potential user to try out every reasonable algorithm configuration in a timely fashion. There is a need for an intelligent engine that can guide the user through the maze of various solvers with various configurations.

In this paper, we introduce a methodology aiming at determining the best solver based on the application type and the matrix properties. We combine the decision tree and the intelligent engine to select a solver and preconditioner combination for the application submitted by the user. We also discuss how our system interface is implemented with the third party numerical libraries. In the case study, we demonstrate the feasibility and usefulness of our system with a simplified linear solving system. Our experiment shows that our intelligent engine is quite adept in choosing a suitable algorithm for different applications.

## I. INTRODUCTION AND MOTIVATION

The computation of eigenvalues and eigenvectors is an important and often time-consuming phase in computer simulations. Without being exhaustive, eigenvalues and eigenvectors are used in the study of nuclear reactor dynamics (stability of neutron fluxes [9]), in finite element dynamic analysis of structural models (e.g., seismic simulations of civil infrastructure [10], [11]), in the design of the next generation of particle accelerators [12], in the definition of a set of eigenfaces in biometric-based identification systems [13], in the solution of Schrödinger’s equation in chemistry and physics [14], in the design of microelectromechanical systems (MEMS [15]), and in the study of conformational changes of proteins [16]. Because of the need for higher levels of simulation details and accuracy, 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 of solving eigenvalue problems, various useful numerical algorithms that are suitable for solving large-scale eigenvalue problems are developed. Some of these numerical libraries also have parallel implementations [17], [18], [19], [20]. With the growing availability of solvers, domain scientists are no longer facing the problem of no available algorithms to use but too many algorithms to choose from. (In this paper, word “algorithm” is interchangeable with “solver” because we are focusing on eigenvalue problem solving algorithm.) However, little consideration is given to provide a robust and effective way to sift out a best solver for a particular application. For sequential or parallel algorithms, a suitable choice of solver may have an impact of order of magnitude on an application compared to a bad one. We identify the following challenges to choose a “best” solver:

- Given the vast number of algorithmic and computer architectures, how to facilitate a naïve user to choose an algorithm “best” suited for a particular application. Different algorithms have different convergence behaviors, memory requirements, and trade-offs between accuracy and performance. It is difficult and tedious to track this metrics and select an algorithm based on these metrics manually. Ideally, it should be done automatically.

- There is no *unified software framework* targeted for high-end computers that facilitates swapping among different algorithm implementations. The *Eigentemplates* book [21] provides an excellent algorithmic guideline. Yet, in order to fully appreciate that book a potential user must be equipped with sufficient knowledge of numerical analysis and programming skills, in particular on parallel computers. What is needed to bridge the gap between *Eigentemplates* and the end user is the software tool that contains efficient and scalable implementations of those algorithms. The libraries currently available are limited in scope, scattered at different places with different interfaces, see Table I for a brief survey of a variety of eigenvalue packages.
- There are usually a large number of parameters associated with each algorithm, which can be adjusted in order to make the algorithm perform more efficiently. For example, the electronic structure calculation codes often employ conjugate gradient minimization based eigensolvers that require inner-outer iterations, where the number of iterations is set by the user and usually requires information about the target problem, and therefore may greatly impact the computational performance. Providing there is a collection of libraries with a unified interface, as simulations become increasingly sophisticated and larger, it becomes infeasible for the potential user to try out every reasonable algorithm configuration in a timely fashion.

Therefore, there is a need for an *intelligent engine* that can guide the user through the maze of various solvers with various configurations and a heuristic database that records all the historical data to provide the *intelligent engine* online information to make a judicious decision .

Name	Method	Version	Date	Language	Parallel
ARPACK	Implicitly Restarted Arnoldi/Lanczos	2	1996	F77	MPI
BLZPACK	Block Lanczos, PO+SO	04/00	2000	F77	MPI
DVDSON	Davidson	-	1995	F77	-
GUPTRI	Generalized Upper Triangular Form	-	1999	F77	-
IETL	Power/RQI, Lanczos-Cullum	2.1	2003	C++	-
JDBSYM	Jacobi-Davidson (symmetric)	0.14	1999	C	-
LANCZOS	Lanczos (Cullum, Willoughby)	-	1992	F77	-
LANZ	Lanczos, PO	1.0	1991	F77	-
LASO	Lanczos	2	1983	F77	-
LOBPCG	Preconditioned Conjugate Gradient	4.10	2004	C	MPI
LOPSI	Subspace Iteration	1	1981	F77	-
MPB	Conjugate Gradient / Davidson	1.4.2	2003	C	-
NAPACK	Power Method	-	-	F77	-
PDACG	Deflation-accel. conjugate gradient	-	2000	F77	MPI
PLANSO	Lanczos, PO	.10	1997	F77	MPI
QMRPACK	Nonsymmetric Lanczos with lookahead	-	1996	F77	-
SLEPc	Power/RQI, Subspace, Arnoldi	2.2.1	2004	C/F77	MPI
SPAM	Subspace Projected Approx. Matrix	-	2001	F90	-
SRRIT	Subspace Iteration	1	1997	F77	-
SVDPACK	SVD via Lanczos, Ritzit & Trace Minim.	-	1992	C/F77	-
TRLAN	Lanczos, dynamic thick-restart	1.0	1999	F90	MPI
Underwood	Block Lanczos	-	1992	F77	-

TABLE I  
A SKETCHY SURVEY OF EIGENVALUE LIBRARIES

The objective of this paper is to address some of the challenges aforementioned, and present our design and methodology towards an automatic application-based eigensolver selection toolbox “EIGADEPT” using an intelligent engine. We implemented a small system in our case study to validate our methodology. Furthermore, to provide expert advice to the user, we implemented an intelligent engine in our case study using a small training set from Matrix Market [25]. The intelligent engine updates the database adaptively as simulations proceed.

In Section 2, we discuss our methodology to implement our EIGADEPT, how EIGADEPT will be used, what are its major components and how they are connected. In Section 3, we describe a case study using our methodology to solve linear systems using various algorithms implemented in PETSc [22], [23] and an intelligent engine. We outline our plans for the eigensolver selection using our intelligent engine in Section 4. Section 5 illustrates some related works and states the difference between our project and the others. Section 6 concludes the paper with future work.

## II. METHODOLOGY

Choosing an appropriate algorithm (and its parameters) depends on the mathematical properties of the problem, the desired spectral information, and the available operations and their costs. The eigenvalue template book [21] provides some valuable “decision trees” based on the above information (e.g., Tables 4.1 and 5.1 in it). Given the large, high-dimensional algorithm/parameter space, a simple decision-tree mechanism may be insufficient. Furthermore, such information may not be available at runtime or may be costly to obtain. For example, the decision tree needs to know if the matrix is positive definite. To answer this question, it requires considerable computation which might not be wanted by the user at runtime. Our EIGADEPT intends to address the dilemma of “less information but accurate solving” by providing algorithm recommendations based on previous knowledge.

The architecture of EIGADEPT is shown in Figure 1. The main components of our EIGADEPT system are:

- 1) A *Universal User Interface*. For various numerical algorithms available with different parameters, it’s easier for the user to call different algorithms with one universal interface. Since most of the legacy scientific codes are programmed in FORTRAN, we provide a universal FORTRAN interface in our EIGADEPT system. The user call this fortran subroutine in their own application to trigger the EIGADEPT system to find a suitable algorithm and return the result to the user. All the selection procedure and execution are transparent to the user. The user could also list his priority for any desired algorithm characteristics such as memory limit, convergence rate, iteration counts, etc. These information will be useful to choose a most suitable algorithm if there are several candidates.
- 2) A *Data Analyzer*. It receives the input data submitted from the user, strips the necessary information required by the intelligent engine to make algorithm-wise decision and passes these information to the intelligent engine. For example, it will decide if the matrix provided is sparse or symmetric. Data Analyzer will provide the intelligent engine all the necessary information that can be collected without substantial computation. For example, determining if a matrix is transposable is costly, so Data Analyzer may not provide such information to the intelligent engine. Though this information is missing, the intelligent engine is still able to find a suitable algorithm provided a similar problem was solved previously.
- 3) *Decision Trees*. It is a repository of decision trees incorporated from [21].
- 4) A *relational database*. The database initially contains some training sets from which the best solvers and preconditioners are known beforehand for certain applications. The training set is obtained from our prior work with various applications. Each new application solved by the intelligent engine will be stored in the database with all the application properties and algorithm information. A database record stores the information regarding the input data such as symmetry, sparsity, the best solver and preconditioner, the numerical library that provides this solver and preconditioner, the parameter setting, the execution time using the best solver and preconditioner, and the application type, etc. The database gradually improves its contents as more problems are solved by the intelligent engine, thereby making the algorithm prediction increasingly accurate. In particular, the database can be *adapted* at runtime through the repeated solutions of similar eigensystems form a specific application domain. The database is implemented by MySQL which is a free open source database with a reliable C API.
- 5) An *Intelligent Engine*. This is the central part of EIGADEPT system. After receiving the information from Data Analyzer, the intelligent engine will search the decision tree first for an algorithm. If there is a match, it will directly search the underlying numerical libraries for a matching subroutine and pass the user’s input data with the proper parameters to that subroutine. After the execution, the intelligent engine will return to the user the result and the runtime statistics. If the decision tree does not give a choice, the intelligent engine

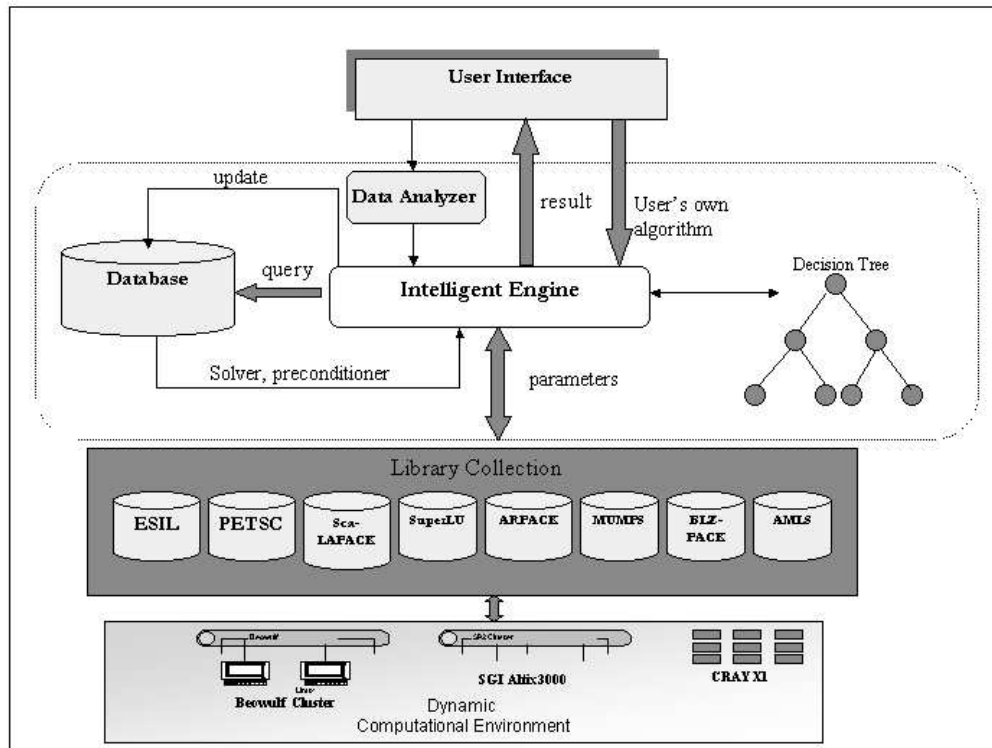


Fig. 1. EIGADEPT Architecture

will query the database for a suitable algorithm based on the matrix properties and application type. There could be three outcomes with this query:

- Exactly one record in the database fits all the specifications provided by the intelligent engine. Then the solver, the preconditioner and the numerical library will be selected.
- More than one record in the database fit the specifications. Then the intelligent engine will try to find an algorithm based on the user's preferred algorithm characteristics such as least amount of memory, and convergence rate.
- No record is found in the database. We will provide our default solving algorithm to the user application. This decision may be wrong, therefore, when there are available free resources, we will solve the problem with different algorithms other than the default one. We call it *off-line* execution. If the off-line execution suggests a better algorithm for this application, we will update our database with this algorithm and all the parameters used.

After choosing the algorithm to use, the intelligent engine passes the function call with the proper parameters to the corresponding numerical library underneath and returns the execution results to the user when finished.

Complementary to the system's intelligent engine, the user can provide his own choice of algorithm, then the system will directly use this algorithm from the numerical libraries connected to the system without consulting the decision tree and database. If the user-specified algorithm is not available in the current library repository, the intelligent engine will ask the user to provide his own implementation or returns an error message. Each execution will be fed back to the database by the intelligent engine for future reference. Since the algorithm chosen from the decision tree or the database may or may not be optimal, the off-line execution and feedback

mechanisms provide adaptation for the algorithm selection.

The intelligent engine is programmed in C because of its portability and low overhead. Another reason we are using C is because many modern, popular numerical libraries such as PETSc is programmed in C/C++, our C program will be easy to be incorporated into other applications which are using these libraries. For the libraries programmed in Fortran, our intelligent engine can issue subroutine calls with C-Fortran interface.

- 6) *Numerical Libraries*. This is a collection of the currently available libraries for the eigenvalue problems. They are implemented by different programming languages and with different calling sequences. Once an algorithm is chosen by the intelligent engine, it is the intelligent engine’s responsibility to pass the needed parameters to 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.

### III. CASE STUDY

In this section, we present a case study using our system to select an iterative solver and a preconditioner for solving linear system  $Ax = b$ . For our training set, we choose 20 small size matrices from Matrix Market [25] which are solvable by various iterative methods. These matrices come from such application domains as acoustic scattering, astrophysics, biochemistry, fluid flow, and quantum physics, etc. We include matrices from different domains for the sake of generality. Although the training set contains only limited types of applications, it is a good starting point for the solver and preconditioner selection for the user application. Information regarding the new application problems and their performance using the solver/preconditioner chosen by the intelligent engine will be added into database.

For our case study, we use PETSc [22] as the underneath library because PETSc is widely used and provides a uniform interface to many linear solvers. The intelligent engine and relational database are implemented with C and MySQL respectively. The decision tree used is shown in Figure 2 which is taken from [1]. The user triggers matrix solving through our universal interface, which has the following prototype:

```
solve ( mp1, mp2, mp3, ..., A, b )
```

where  $mp_i (i = 1...n)$  contains the matrix properties such as  $mp_1 : symmetric$ ,  $mp_2 : sparse$ , etc. For those matrix properties that the user does not know, they can be left as NULL. The *Data Analyzer* receives the function call from the user interface, and checks if it can provide further information for the NULL fields. It then passes all the matrix properties to the intelligent engine after analyzing the input information.

After receiving all the information it needs, the intelligent engine first searches decision tree to find out if there is a suitable solver for this problem. In most of the case, the decision tree search does not result in a leaf because of information missing. From that point, the intelligent engine will query the database to find a matrix with similar properties and its previous solving algorithm. In this case, the database will return the solver and preconditioner from PETSc and their needed parameters. The intelligent engine will use these information to solve the application. In some cases, the decision tree is sufficient to find out the matrix solver but the preconditioner choice remains open. The database can be used to decide the right preconditioner.

We tested our intelligent system with three matrices from real applications. Table II tabulates the main properties of the three matrices, including application domain, data type, size and number of nonzeros.

Apps	Matrix	Type	Size	nnz
Reaction-Diffusion	rdb3200l	real, unsymmetric	3200 × 3200	18880
BCS Structural Engineering	bcsstk26	real, symmetric	1922 × 1922	30336
Bounded Finline Dielectric Waveguide	bfw782b	real, symmetric	782 × 782	5982

TABLE II  
PROPERTIES OF THE APPLICATION MATRICES

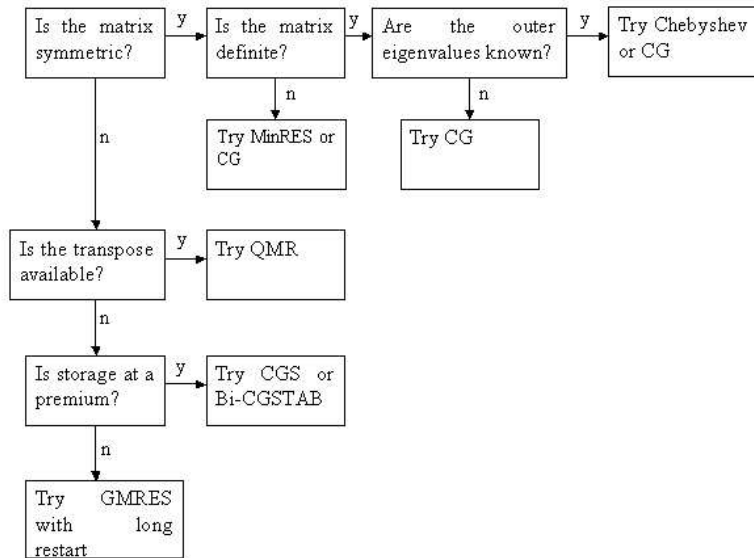


Fig. 2. Decision Tree For Iterative Methods (Taken from [1])

For the sake of simplicity, we used only one processor in this experiment. From all the solvers and preconditioners provided by PETSc, we chose eight solvers: Conjugate Gradient (cg), BiConjugate Gradient (bicg), Generalized Minimal Residual (gmres), BiCGSTAB (bcgs), Conjugate Gradient Squared (cgs), Transpose-Free Quasi-Minimal Residual (tfqmr), Conjugate Residual (cr) and Least Squares Method (lsqr), and four preconditioners: No Preconditioner (none), Jacobi (jacobi), Block Jacobi (bjacobi) and Additive Schwarz (asm). Altogether, there are 32 solver/preconditioner combinations to choose from. The user provides the input linear system through the uniform interface and the intelligent engine selects a suitable solver and preconditioner, executes them, and returns the results to the user. As a comparison, we also run all 32 solver/preconditioner algorithm combinations. Table III summarizes the results, which shows the best and worst solution times, the percentage of failures (non-converged) among all the solver/preconditioner combinations. The solver and preconditioner resulting in the best and worst performance are given in the parenthesis. For these three linear systems, the algorithm selected by our intelligent engine is always the best. Without this intelligent system, a user with little numerical experience may decide to use any of the 32 combinations, which may fail to converge or take long time to converge. For example, for rdb32001, 56.2% solver/preconditioner combinations do not converge. Thus, based only on random selection without any guidance, the application may have to try many times before finding one algorithm that converges. Even if the user chosen solver/preconditioner algorithm converges, the convergence rate varies greatly. The worst/best performance ratios are 8.73, 7.29 and 1.46 for rdb32001, bcsstk26 and bfw782b, respectively.

Matrix	Best time	Worst time	Automatic Selection	% of Failures
rdb32001	2.54 (bicg + none)	22.16 (bcgs + asm)	bicg + none	56.2%
bcsstk26	1.62 (cr + bjacobi)	11.78 (bicg + none)	cr + bjacobi	43.8%
bfw782b	0.0645 (cr + jacobi)	0.0942 (lsqr + asm)	cr + jacobi	9.3%

TABLE III

PERFORMANCE OF THE ITERATIVE METHODS AND OUR INTELLIGENT ENGINE. TIMES ARE IN SECONDS.

We also recorded the runtime overhead of our intelligent engine. For all the three applications, the overhead from intelligent engine is less than 1% of the total execution time.

#### IV. PLAN FOR EIGENSOLVER

An automatic choice of an optimal eigensolver is in many ways more difficult than an automatic choice of an iterative linear solver. In eigenvalues analyses, some of the important data to be taken into account are: the dimension of the problem, number of eigenvalues (and/or eigenvectors) required, location of the required solutions in the eigenvalue spectrum (i.e. smallest, largest, close to a reference value, etc.), accuracy of the required solutions, and availability of approximate solutions (obtained from previous simulations with similar problems for example). In contrast to iterative linear solvers, the available memory is also important because it can influence the effectiveness of restarting strategies implemented in several algorithms. 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.

We will 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. Algorithms of this type have already been implemented in various parallel libraries, including ARPACK (implicitly restarted Arnoldi method), BLZPACK (block Lanczos method), JaDa (block Jacobi-Davidson method), and TRLAN (thick-restart Lanczos method). Table I lists the other candidate implementations. We plan to enhance some of these eigensolvers with shift-and-invert capabilities using existing scalable sparse direct linear solvers, such as SuperLU and MUMPS. Parallel implementations for some other newly emerged algorithms are not yet available, which is the case of the automatic multilevel substructuring (AMLS) method. The AMLS method is often used in structural engineering applications, and is suitable for large numbers of processors because of its domain decomposition nature.

The fundamental obstacle is that the aforementioned solvers have different interfaces and parameters. Therefore, we need to effectively and efficiently package them so as to deal with their distinct requirements and possibly outputs. In addition, we have already identified a number of applications that can be used to further study and validate the ideas discussed in this paper, in particular structural analyses of civil infrastructures, the simulation of microelectromechanical systems (MEMS), the design of new materials at a nano-scale, the design of the next generation linear accelerators, and the study of conformational changes of proteins. In the following, we briefly summarize the characteristics of these applications.

In structural analyses of civil infrastructure the matrices result from structural systems idealized by finite element models. They are usually sparse and of very large dimension depending on the desired level of details. The number of eigenvalues and eigenvector required in typical applications may vary from a few in the lower end of the eigenvalue spectrum to a few hundred in a particular range (seismic analyses).

In MEMS simulations, beams, electrostatic gaps, circuit elements, and other elements are modeled by small, coupled systems of differential equations. Three kinds of analysis are usually performed: static (solving nonlinear systems), modal (eigenanalysis), and transient (solving nonlinear ODEs). The modal analysis often reduces to solving a generalized sparse eigenvalue problem, and the matrices can be complex symmetric. One or a few interior eigenmodes are of particular interest.

In the study of electronic properties of new materials at a nano-scale the eigenvalue problems of interest are complex Hermitian and appear in different flavors. The configuration-interaction (CI) treatment of excited or many body problems requires the diagonalization of large dense matrices. For a multiple exciton CI Hamiltonian the matrices involved are sparse and highly structured. In both cases, one seeks to compute some of the small eigenvalues together with the corresponding eigenvectors.

In the design of next-generation particle accelerators, there is a need to compute approximate cavity resonance frequencies (RF) and the electromagnetic field associated with a cavity structure. The problem is challenging because of the wide distribution of the eigenvalue spectrum, the need for finding the relatively small interior eigenvalues of a generalized eigenvalue problem, and the large problem size. Typically, there are a number of eigenvalues very

close to zero and the eigenvalues of interest are the (hundreds) smallest nonzero eigenvalues, together with their associated eigenvectors.

The study of collective motions of proteins provides insights on the conformational changes molecules experiment during chemical reactions. An approximation for the history of motion of the protein can be obtained through the superposition of collective variables, or normal modes coordinates. These modes and their corresponding frequencies are obtained as solutions of a real symmetric problem obtained from the potential energy and atomic masses of the proteins. Usually, the normal modes associated with the smallest frequencies (the smallest eigenvalues) are responsible for most of the amplitude of the atomic displacements of proteins. The goal of web-based services like eINemo is to allow for the computation of low frequency normal modes of proteins given in PDB format, with different levels of detail. Therefore, this application is also suitable for our automatic eigensolver.

## V. RELATED WORK

There exists a number of other research projects seeking the goals similar to ours but using different approaches. The Self-Adaptive Numerical Software (SANS) system [2], [3] contains an intelligent engine, a history database, a network scheduler and the underlying adaptable libraries. SANS uses user provided metadata to automatically analyze the problem and then uses a scripting language to compose a “p-oy-algorithm”. For problems the information cannot be obtained in quickly, SANS searches a heuristic database storing previous performance data for self-tuning rules. Netsolve [4], [5] is a client-agent-server system which provides remote access to hardware and software resources from a variety of scientific problem solving environments. After the client submits a function request, from a list of all available servers, the agent finds an available server in which the demand of the software request can be met. Then the server executes the function for the client and returns the results. Unlike SANS and our system, Netsolve does not provide suggestions to the user about which solver will be the best and it loyally selects the server hosting the service that the client requests to fulfill the task.

Grid-TLSE [24] provides user a web interface for sparse matrix computation on the grid. The main part is an expert site consist of: writing the procedures for the expertise, inclusion of the sparse matrix software, building a database including a bibliography on sparse matrix software and collections of sparse matrices (Rutherford-Boeing and PARASOL data sets). The Scalable Multimethod Sparse Solvers project (SuperSolvers) [7] aims at developing new sparse solution schemes, their analysis, and applications to computational modeling and simulation. SuperSolvers incorporate hybrid solvers (using flexible incomplete sparse factorization preconditioners with a range from pure iterative to pure direct method) and composite solvers (using a sequence of basic solution schemes on a single linear system and an adaptive solver dynamically selecting a sparse solution scheme to match changing numerical attributes across iterations of a long running simulation). SuperSolvers focus on selecting a robust and scalable solver tailored to meet application demands. In [8], the authors developed a Linear System Analyzer (LSA) based on component programming paradigms. Users can choose available linear system solver components defined by IDL. As a prototype to exploit the usefulness of component architecture on distributing scientific problem solving, LSA does not give much advice to the users on how to select the solvers and preconditioners to reduce the runtime and improve performance. Thus for a naive user, LSA will not be able to deliver a best optimized result.

EIGADEPT is focusing on eigenvalue and eigenvector applications. Unlike SANS, EIGADEPT is not using metadata from the user to describe the input data. Instead, information comes from the heuristic database and the data analyzer analyzing 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 his/her application code if he/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.

## VI. FUTURE WORK

More numerical libraries will be incorporated into the system and we will focus on eigenproblems requirements. So far, we have not considered the memory limit, the available computing resources and the user’s time requirement. The user will be able to specify this kind of information through the input interface and such information will affect the algorithm selection. The information regarding the application execution environment and the underlying architecture will be used to choose the right implementation. In order to ensure scalable and transportable

performance of the eigensolvers, we will identify and isolate the performance critical kernels, capture the important hardware characteristics, including memory/communication latency and bandwidth, cache size and incorporate these information into our knowledge base.

We will conduct more research on our data analyzer, because the more information it provides, the closer is the selected algorithm to the best. Some information such as “Is this matrix’ transpose available?” or “Is this matrix positive definite?” are useful, but to answer these questions needs intensive computation. We need to find an effective approach to finding an accurate guess with low overhead.

We will wrap our intelligent as a CCA [28] model. A great deal of efforts have been put on building numerical algorithms with CCA. With the CCA providing/using port mechanism, our intelligent engine can couple with the other newly developed eigensolver components directly. The intelligent engine can even be used outside our eigensolver context through the well-defined component interface to accommodate different scientific applications.

#### ACKNOWLEDGMENT

This work was supported by the Director, Office of Advanced Scientific Computing Research, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract number DE-AC03-76SF00098.

We would like to thank Parry Husbands and Tony Drummond of Lawrence Berkeley National Laboratory for their valuable suggestions on our system design.

#### REFERENCES

- [1] R. Barret et. al. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, Society for Industrial and Applied Mathematics, Philadelphia, 1994.
- [2] J. Dongarra and V. Eijkhout. *Self-adapting numerical software and automatic tuning of heuristics*, In Proceedings of the International Conference on Computational Science, June 2–4 2003.
- [3] 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.
- [4] K. Seymour, A. Yarkhan, S. Agrawal, J. Dongarra. *NetSolve: Grid Enabling Scientific Computing Environments*, Grid Computing and New Frontiers of High Performance Processing, Grandinetti, L. eds. Elsevier, 2005.
- [5] S. Agrawal, J. Dongarra, K. Seymour, S. Vadhiyar. *Netsolve: Past, Present, and Future - A Look at a Grid Enabled Server*, Making the Global Infrastructure a Reality, F. Berman, G. Fox, A. Hey eds. Wiley Publishing, 2003.
- [6] [www.netlib.org](http://www.netlib.org)
- [7] S. Bhowmick, L. McInnes, B. Norris and P. Raghavan, *Robust Algorithms and Software for Parallel PDE-Based Simulations*, Proceedings of HPC 2004, The Twelfth Special Symposium on High Performance Computing at the 2004 Advanced Simulation Technologies Conference, Arlington, VA, pp. 37-42, April 2004.
- [8] <http://www.extreme.indiana.edu/pseware/LSA/LSAhome.html>
- [9] G. Verdu, D. Ginestar, V. Vidal, and J. L. Muñoz Cobo. 3D  $\lambda$ -Modes of the Neutron-Diffusion Equation. *Ann. Nucl. Energy*, 21:405–421, 1994.
- [10] F. Y. Cheng. *Matrix Analysis of Structural Dynamics: Applications and Earthquake Engineering*. Marcel Dekker, New York, N.Y., 2000.
- [11] A. K. Chopra. *Dynamics of Structures: Theory and Applications to Earthquake Engineering*. Prentice Hall, Upper Saddle River, N.J., 2nd edition, 2000.
- [12] K. Ko, N. Folwell, L. Ge, A. Guetz, V. Ivanov, L. Lee, Z. Li, I. Malik, W. Mi, C. Ng, and M. Wolf. Electromagnetic systems simulation - from simulation to fabrication. SciDAC Report, 2003. Menlo Park, CA.
- [13] M. Turk and A. Pentland. Eigenfaces for Recognition. Technical report, Vision and Modeling Group, The Media Laboratory, MIT, 1990.
- [14] M. Payne, M. P. Teter, D. C. Allan, T. A. Arias, and J.D. Joannopoulos. Iterative minimization techniques for ab initio total energy calculations: Molecular dynamics and conjugate gradients. *Rev. Mod. Phys.*, 1045, 1992.
- [15] J. V. Clark, D. Bindel, N. Zhou, S. Bhave, , Z. Bai, J. Demmel, and K. S. J. Pister. SUGAR: Advancements in a 3D multi-domain simulation package for MEMS. In *Proceedings of the Microscale Systems: Mechanics and Measurements Symposium*, Portland, OR, June 4 2001.
- [16] O. A. Marques and Y.-H. Sanejouand. Hinge-Bending Motion in Citrate Synthase Arising from Normal Modes Calculations. *Proteins: Structure, Function and Genetics*, 23:557–560, 1995.
- [17] 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.
- [18] Rich Lehoucq, Kristi Maschhoff, Denny Sorensen, and Chao Yang. Parallel ARPACK. [http://www.caam.rice.edu/~kristyn/parpack\\_home.html](http://www.caam.rice.edu/~kristyn/parpack_home.html).
- [19] O. A. Marques. BLZPACK: Description and User’s Guide. Technical Report TR/PA/95/30, CERFACS, Toulouse, France, 1995.

- [20] Kesheng Wu and Horst Simon. Thick-restart lanczos method for large symmetric eigenvalue problems. *SIAM J. Matrix Analysis and Applications*, 22(2):602–616, 2001.
- [21] 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*. SIAM, Philadelphia, 2000.
- [22] <http://www.mcs.anl.gov/petsc>
- [23] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith and H. Zhang. *PETSc Users Manual*
- [24] <http://www.enseeiht.fr/lima/tlse/survey.html>
- [25] <http://math.nist.gov/MatrixMarket/>
- [26] J. Nieplocha, R.J. Harrison, and R.J. Littlefield. Global Arrays: A nonuniform memory access programming model for high-performance computers, *The Journal of Supercomputing*, 10:197-220, 1996.
- [27] <http://www-unix.mcs.anl.gov/mpi/>
- [28] <http://www.cca-forum.org/>