A Grid Computing Environment for Enabling Large Scale Quantum Mechanical Simulations

Jack J. Dongarra[1] and Padma Raghavan[1]

Department of Computer Science
The University of Tennessee
1122 Volunteer Blvd.
Knoxville, TN 37996-3450
{dongarra, padma}@cs.utk.edu

Abstract. This paper describes work-in-progress towards developing a simulation environment that utilizes recent advances in in the areas of grid middleware and computational kernels. Our goal is to develop an environment suitable for composing and deploying an overall high-performance, flexible and robust software solution for large-scale quantum mechanical simulations.

1 Introduction

Research in recent years has advanced the state of computational technology for enabling large scale science and engineering applications along two broad fronts. The first concerns the hardware and middleware infrastructure where the evolution is towards computational grids; disparate ensembles of high-performance computers, clusters, networks, and storage can now be integrated to form powerful unified systems [5, 6, 10, 11]. The second concerns the large number of fundamental computational kernels that have been developed for parallel and distributed scientific computing. These have emerged from a variety of research on new parallel algorithms and software development and include advances in both dense and sparse matrix computations [1, 4, 15, 20, 22]. We plan to develop a simulation environment that utilizes recent advances in in the areas of grid middleware and computational kernels. Our environment is geared towards composing and deploying an overall high-performance, flexible and robust software solution for certain large-scale applications of interest.

We believe the main problem facing application developers is that of composing an overall high-performance solution by selecting judiciously from the array of available alternate methods for underlying subproblems. A typical large scale application requires the solution of several subproblems of differing granularity with differing amounts of parallelism, computation and communication requirements. Consequently, a simple parallel model of computation involving the same number of processors from start to end cannot result in an efficient solution. Furthermore, there are a variety of solution techniques for a given problem; the choice of the best alternative often depends on the problem characteristics as

well as hardware and network speeds. Additionally, problem characteristics can change dramatically within the life of the same simulation. Finally, the overall characteristics of the application may allow various ways of decomposing it into subproblems further compounding the problems of composing an overall efficient solution.

We plan to develop a pipelined-parallel software architecture to harness the power of computational grids to enable large scale simulations. At the University of Tennessee, NSF has recently funded a five-year effort for building a "Scalable Intra campus Research Grid," henceforth called SInRG [8]. Our simulation environment will be developed on this grid. Our focus will be simulations involving large-scale eigenvalue computations associated with sparse matrices. These occur in a variety of molecular dynamics applications. Quantum nanotechnology simulations based on a "generalized tight binding molecular dynamics" [16, 18, 21] are extremely compute-intensive; for example, each time-step in a simulation may require the solution of the standard eigenvalue problem to compute all the eigenvalues and eigenvectors of a symmetric, positive definite sparse matrix. Molecular dynamics for restricted closed-shell Hartree-Fock approximation through the Roothan equations [23] also require similar computations.

2 Computational Problem

The central problem in molecular dynamics applications of interest is that of computing $O(N)$ eigenvalues and eigenvectors of an $N \times N$ symmetric positive definite matrix. Such computations are intrinsically expensive; for $N \times N$ matrices, the storage requirements grow as N^2 while the number of operations grow as N^3 when the matrices are treated as *dense*. The constant in the N^3 cost term is larger for the generalized eigenvalue problem but the solution methods for the two problems are closely related. For molecular dynamics models of interest with several thousand atoms, the matrix dimension N is in the range 10,000 to 50,000. The eigenvalue problem has to be solved in each time-step of the simulation and a simulation typically involves a thousand time-steps. Consequently, for matrix dimensions of $\approx 10,000$, a simulation with one thousand time-steps requires computations in the order of 10^{15}. Making such simulations tractable is challenging and must necessarily involve utilizing performance gains from all possible enhancements including those from algorithmic improvements, efficient utilization of hardware resources, and selective composition of solution alternatives.

The standard eigenvalue problem can be generically stated as computing: $Hx = \lambda x$. Given that our application needs $O(N)$ eigenvalues and eigenvectors, *direct* methods are of primary interest. The direct solution process consists of the following three main steps:

1. Transform the matrix to a tridiagonal matrix T using orthogonal transformations.
2. Compute eigenvalues and eigenvectors of T.

3. The eigenvalues of T are the eigenvalues of the original matrix; the eigenvectors of the latter are computed by multiplying the eigenvectors of T by the orthogonal matrix composed of transforms used in the first step in the conversion to tridiagonal form.

For a detailed discussion and overview of fundamental eigensolution techniques, two excellent sources are the books Demmel and Parlett [7, 19]. Good serial and parallel implementation exist in form of packages LAPACK [1] and ScaLAPACK [4].

For the three step solution process described above there are several algorithmic choices for each step. The simplest model might be to treat to the matrix as dense; now the choice of kernels for steps 1 and 3 are obvious. However, step 2 could be performed using at least three broad classes of methods: (a) based on bisection and inverse iteration, (b) using the QR method, and (c) using divide and conquer. The performance of each alternative depends to a large extent on the eigenspectrum of the problem as well as machine characteristics such as the computation to communication ratio.

The matrices arising from simulations of interest are sparse; i.e., and $N \times N$ matrix has typically some cN nonzeroes where c is a small constant. Sparsity of the matrix can be exploited for the first step; if the matrix can be put into a *band* form with a bandwidth of b, the intrinsic cost of the first step decreases from N^3 to b^2N. The best algorithm for converting a banded matrix to tridiagonal form on parallel computers is still under research.

One interesting aspect of our application is that although $O(N)$ eigenvalues and eigenvectors are needed, they are not needed all at once. That is, it suffices to compute and use the eigenvectors one at a time. Hence at the very least , by reorganization of the underlying computation, the space requirement could be reduced from $O(N^2)$ to $O(bN)$ when band methods are used for step 1. A first alternative would be to compute only eigenvalues of the tridiagonal matrix T in the second step. The computations of eigenvectors could be postponed to the third step, where as earlier the eigenvectors of T could be calculated and then used to compute eigenvectors of the original matrix. Furthermore, a key issue in the parallel implementation of step (2) using divide-and-conquer (alternative c above) relates to the data-distribution of the eigenvectors as well as their re-orthogonalization. By moving eigenvector computation to the third step, one can easily explore models where the eigenvalues are divided into several groups (spectrum slicing) and the eigenvector computation for each group proceeds independently on a single processor. This could be done using explicit parallelization and LAPACK [1] kernels for eigenvector computation. Yet another interesting alternative would be to compute eigenvectors of the original matrix directly using inverse iterations. This could be especially advantageous when the sparsity of the matrix is utilized. This choice in turn leads to other alternatives, for example, a wide variety of choices for the sparse linear solution scheme for inverse iteration [2].

Our development effort is geared towards exploring such alternatives for the subproblems in order to compose an overall efficient solution. Another aspect

of our approach relates to overcoming the traditional problem with decreasing speedups on increasing the numbers of processors using fixed-problem size. Consider for example, the simple solution process which involves treating the matrix as dense and using the routines from ScaLAPACK to solve the overall problem on a multiprocessor or a cluster. In our earlier work [17], we took exactly this approach and enabled a relaxation of a 1061 atom carbon cluster that forms part of a "knee" junction with interesting metal-semiconductor contact for connecting nanotubes of different diameters. The matrix dimension was 4244 and using 8-16 processors of a NOW with Intel Pentium-II processors and a Myrinet switch [9], computation time for a single time-step drops to under several minutes (speedups compared to one processor execution were nearly ideal). However, the overall simulation which required nearly 800 time-steps took several days of non-stop execution. Furthermore, this time cannot be reduced by simply increasing the number of processors; this lowers the per-processor utilization and we observed a slowdown with as few as 32 processors.

By developing a software pipeline, we can tackle as many different simulations as the number of pipeline stages. Now the actual time for a single time-step of any given simulation could be reduced by a factor equal to the number of pipeline stages. Each pipeline stage will be deployed on disjoint groups of processors. Parallelism within each stage will be exploited using the message passing model and MPI. The overall solution will be composed using NetSolve which will also be used to deploy the application on the SInRG computational grid.

3 Developing a Software Environment

The primary building block of the SInRG architecture is the Grid Service Cluster (GSC). A GSC is an ensemble of hardware and software constructed and administered by a single research group but also optimized to make its resources easily available for the overall user community. A GSC is a concentration of (possibly specialized) computing resources in an advanced local to wide-area network. Each GSC has three basic hardware components: a high-speed data switch capable of providing at least 1Gb/s per link, a data-storage unit connected to the switch, and computational resources customized for specific research. The latter could be an SMP, an MPP, a cluster of workstations, etc. Six GSCs are being established, with each one having typically in excess of the raw computing power of the state-of-the-art cluster of 32-node multiple-CPU computers. The total raw computing power over all GSCs will be in the range of Teraops/second. We next describe NetSolve, the software environment on SInRG and then outline our strategy for developing our simulation environment. We view this project as a precursor to the development of general-purpose software component technology based on object-oriented methods, an emerging field of research [12, 13].

3.1 NetSolve

High-level access across all GSC's and to CPUs within each GSC is provided through NetSolve [5]. NetSolve is a software environment designed to transform

disparate computers and software libraries into a unified, easy-to-access computational service. It aggregates the hardware and software resources of any number of computers that are loosely connected across a network and offers up their combined power through familiar client interfaces such as MATLAB, and C. It uses a *client-agent-server* paradigm to deliver this power to users without revealing the complexity of the underlying system. The user's data is sent to the server, where the programs or numerical libraries operate on it; the result then is sent back to the user's machine.

NetSolve provides the user with a pool of computational resources. These resources are in the form of servers that have access to ready-to-use numerical software. These computational servers can be running on single workstations, networks of workstations, or MPP (Massively Parallel Processor) systems. The user gains access by using any one of the NetSolve client interfaces such as MATLAB. The main function of the NetSolve agent is to process user requests and to choose the most suitable server for the underlying computation. An added advantage is that the agent performs load-balancing among the different resources.

When building NetSolve, one of the challenges was to design a suitable model for the computational servers. Features include uniform access to the software, configurability, and preinstallation. To make the implementation of such a computational server model possible, NetSolve has a general, machine-independent way of describing a specific numerical process, as well as a set of tools to generate new computational modules. The main component is a descriptive language which is used to specify the functionality of a computational server. The description files written in this language can be compiled by NetSolve into computational modules executable on any UNIX or NT platform. This approach allows machine independence as well as the ability to integrate arbitrary software components into NetSolve. Additionally, this framework also allows increased collaboration between research teams across institutions. Description files for a given numerical library need be written only once. These files can then be transferred to other locations and then compiled to create a new stand-alone NetSolve system or to contribute new servers to an existing system. Each time a new description file is created, the capabilities of the entire NetSolve system are increased. A number of description files have been generated for the following numerical libraries: ARPACK, FitPack, ItPack, MinPack, FFTPACK, LAPACK, BLAS, and ScaLAPACK. A Graphical User Interface (GUI) is provided to simplify generation of description files. This interface performs various error checking on user input in the form of choices from a menu. Using this interface is much easier than creating a description file manually, especially as the complexity of the problem increases.

3.2 A Pipelined Parallel Architecture

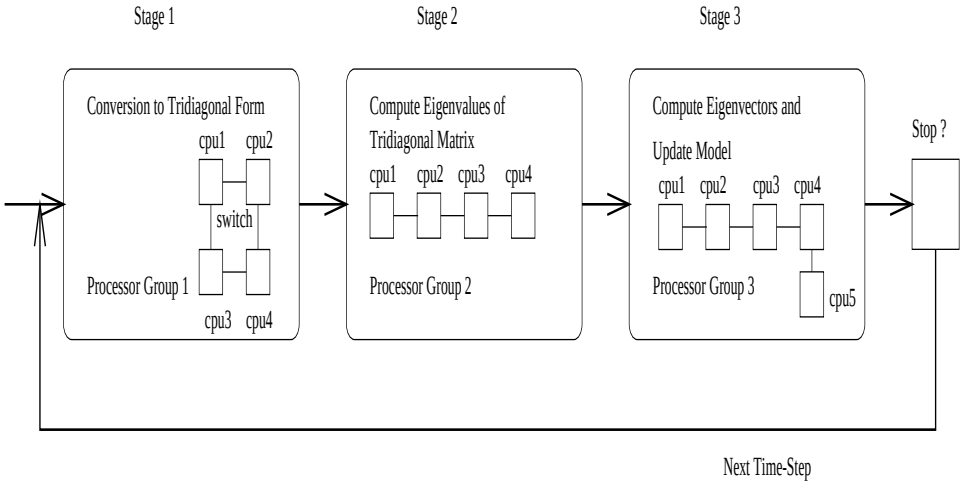
As mentioned in Section 2, the underlying eigenvalue computations can be divided into three main stages. We propose a pipelined parallel architecture in which each stage of the pipeline represents a major stage in the underlying computations. Now each one of the pipeline stages can be made to execute on the

right number of processors such that per-processor efficiency is maintained while the pipeline stages are kept balanced. The pipeline stages are motivated by the three stage solution process described earlier. However, they are somewhat different both to reduce inter-stage communication and to easily allow the use of different kernels within the stages.

The first stage is responsible for conversion of the matrices to a tridiagonal matrix T. The second stage computes eigenvalues of the tridiagonal matrix; the eigenvectors of T are not computed in this stage. The third stage computes eigenvectors of the original matrix and this may involve using the traditional method of computing eigenvectors of T or new methods to be developed [2]. Our software will be designed is to select the optimal number of processors for each stage of the pipeline; each stage uses a disjoint set of processors. By careful selection of the number of processors per stage, the pipeline can be kept balanced, i.e., with each stage requiring approximately the same amount of time. By working on three different simulations at the same time, we can ensure that one time-step of a simulation will be completed in the time required for a single pipeline stage. If t units of time are required at most by any pipeline stage, then after the first $3t$ units of time, one time-step will be completed for a simulation every t time units. This pipeline architecture is shown in Figure 1.

We have selected the stages so that amount of information to be communicated between stages is typically $O(N)$. We expect the original sparse matrices to be transferred from each stage to the next. These matrices are very sparse and have only $O(N)$ nonzeroes. Additionally, from stage one and two, the tridiagonal matrix must be transferred and this obviously is $O(N)$ amount of data. From stage two to three $O(N)$ eigenvalues need be transmitted. In the last stage the eigenvalues are used to compute the corresponding eigenvectors, however if the simulation is to proceed for another time step all eigenvectors need not be computed and stored all at once. In the molecular dynamics model, each eigenvector and eigenvalue can be used as soon as it is computed to calculate its contribution to the force equations. Only the matrices for the next time-step need to passed from stage three back to one if the simulation is to proceed.

To implement the pipeline we will develop a NetSolve server which will contain suitably encapsulated kernels from ScaLAPACK, LAPACK and sparse solvers such as DSCPACK. The application interface will be through the MATLAB interface to NetSolve. We will begin with a static allocation of processors to pipeline stages as well as a static choice of kernels for each stage. We will migrate to dynamic, on-the-fly selection as the capabilities in NetSolve are enhanced. When resources permit, the three-stage pipelined parallel architecture can be replicated and farmed out to SInRG using NetSolve, thus enabling several independent groups of simulations. By utilizing the performance monitoring features of NetSolve, we will attempt to reduce simulation times through the use of scheduling strategies. Some recent work on application specific scheduling enhancements has yielded promising results [3] and these schemes will be integrated into NetSolve. There are also plans to incorporate fault-tolerance and



Time (a pipeline stage requires at most t units)			
1t	Simulation A, time-step 1		
2t	Simulation B, time-step 1	Simulation A, time-step 1	
3t	Simulation C, time-step 1	Simulation B, time-step 1	Simulation A, time-step 1
4t	Simulation A, time-step 2	Simulation C, time-step 1	Simulation B, time-step 1
5t	Simulation B, time-step 2	Simulation A, time-step 2	Simulation C, time-step 1
6t	Simulation C, time-step 2	Simulation B, time-step 2	Simulation A, time-step 2
7t	Simulation A, time-step 3	Simulation C, time-step 2	Simulation B, time-step 2
8t	Simulation B, time-step 3	Simulation A, time-step 3	Simulation C, time-step 2

Fig. 1. A pipelined parallel software architecture to enable molecular dynamics simulations on a computational grid

visualization capabilities to NetSolve and these features will also be of potential use for our target applications.

4 Concluding Remarks

We plan to use our simulation environment to enable "generalized tight binding molecular dynamics" models of Carbon nanotubes developed by Menon. These nanotechnology simulations concern exploring the properties of complex and three-point, four-point and hetero-junctions of nanotubes to suggest experimentally feasible transistor-like devices [16–18, 21]. These simulations are based on generalized tight-binding molecular dynamics scheme which has been shown to obtain equilibrium geometries for carbon clusters that are in very good agreement with *ab initio* and experimental results. Such simulations are essential for gaining a better understanding of the electronic and material properties of nanoscale clusters to allow design of nanoscale devices in the near future.

5 Acknowledgments

This work was supported in part through grants ACI-97-21361, CCR-98-18334, and CDA-99-72889 from the National Science Foundation.

References

1. E. Anderson, Z. Bai, S. Blackford, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen: LAPACK Users' Guide, Third Edition. SIAM, Philadelphia, PA (1999)
2. J. Barlow, P. Raghavan, K. Teranishi, C. Yang, and R.C. Ward: Computing Eigenvectors of Sparse Matrices Using Inverse Iterations. In preparation
3. F. Berman and R. Wolski: AppLeS: Application Level Scheduling. See `http://apples.ucsc.edu`
4. L. S. Blackford, J. Choi, A. Cleary, 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, PA (1997)
5. H. Casanova and J. Dongarra: NetSolve: A Network Server for Solving Computational Science Problems. The International Journal of Supercomputing Applications, **11** (1997) 212–223
6. K. M. Chandy, A. Rifkin, P. A. G. Sivilotti, J. Mandelson, M. Richardson, W. Tanaka, and L. Weisman: A Word-Wide Distributed System Using Java and the Internet. Proc. of the Fifth IEEE International Symposium of High-Performance Distributed Computing (1996)
7. J. W. Demmel: Applied Numerical Linear Algebra. SIAM, Philadelphia, PA (1997)
8. J. J. Dongarra, M. W. Berry, M. Beck, J. Gregor, M. A. Langston, T. Moore, J. S. Plank, P. Raghavan, M. G. Thomason, R. C. Ward, and R. M. Wolski: A Scalable Intracampus Research Grid. Available at website: http://www.cs.utk.edu/ sinrg, Funded by NSF-CISE

9. J. J. Dongarra, J. S. Plank, and P. Raghavan: Enabling Technology for High-Performance Heterogeneous Clusters. National Science Foundation, $150,000 (1999)
10. I. Foster and C. Kesselman: Globus: A Metacomputing Infrastructure Toolkit. The International Journal of Supercomputing Applications (1997)
11. G. Fox and W. Furmanski: Web Technologies in High-Performance Distributed Computing. In Computational Grids (1998)
12. D. Gannon, R. Bramley, S. Diwan, B. Temko, N. Mukhi, K. Chiu, M. Govindaraju, M. Yechuri, and J. Villacis: Common Component Architecture. Available at website: http://www.cs.indiana.edu/ccat.
13. D. Gannon, R. Bramley, J. Villacis and A. Whitaker: Using the Grid to Support Software Component Systems. SIAM Conference on Parallel Processing (1999)
14. M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. J. Dongarra: MPI: The Complete Reference. The MIT Press Cambridge, MA (1996)
15. G. Karypis and V. Kumar: METIS: Unstructured graph partitioning and sparse matrix ordering system. Technical Report, Department of Computer Science, University of Minnesota, Minneapolis, MN (1995)
16. M. Menon, E. Richter and K. R. Subbaswamy: Structural and Vibrational Properties of Fullerenes and Nanotubes in a Non-orthogonal Tight-Binding Scheme. J. Chem. Phys. **104** (1996).
17. M. Menon, R. Richter, P. Raghavan and K. Teranishi: Large Scale Quantum Mechanical Simulations of Carbon Wires. Superlattices and Microstructures **27** (2000) 577–581
18. M. Menon and K.R. Subbaswamy: Non-orthogonal Tight-Binding Scheme for Silicon with Improved Transferability. Phys. Rev., **55** (1997)
19. B. Parlett: The Symmetric Eigenvalue Problem. Prentice Hall, Engle-wood Cliffs, NJ (1980)
20. P. Raghavan: DSCPACK: A Domain-Separator Cholesky Package for solving sparse linear systems on multiprocessors and NOWs using C and MPI. Available upon request
21. A. M. Rao, E. Richter, S. Bandow, B. Chase, P. C. Eklund, K. A. Williams, S. Fang, K. R. Subbaswamy, M. Menon, A. Thess, R. E. Smalley, G. Dresselhaus, and M. S. Dresselhouse: Diameter-Selective Raman Scattering from Vibrational Modes in Carbon Nanotubes. Science **275**(1997)
22. B. Smith, L. McInnes, and W. Gropp: PETSc 2.0 user's manual. Mathematics and Computer Science Division, Argonne National Laboratory, Report ANL-95-11-Revision 2.0.22, (1997)
23. R.C. Ward, Talk on applications of eigenvalue computations (1999)