Self-Adapting Software for Numerical Linear Algebra Library Routines on Clusters*

Zizhong Chen[1], Jack Dongarra[1], Piotr Luszczek[1], and Kenneth Roche[1]

Computer Science Department, University of Tennessee Knoxville, 1122 Volunteer Blvd., Suite 203, Knoxville, TN 37996-3450, U.S.A. dongarra@cs.utk.edu

Abstract. This article describes the context, design, and recent development of the LAPACK for Clusters (LFC) project. It has been developed in the framework of Self-Adapting Numerical Software (SANS) since we believe such an approach can deliver the convenience and ease of use of existing sequential environments bundled with the power and versatility of highly-tuned parallel codes that execute on clusters. Accomplishing this task is far from trivial as we argue in the paper by presenting pertinent case studies and possible usage scenarios.

1 Introduction

Driven by the desire of scientists for ever higher levels of detail and accuracy in their simulations, the size and complexity of required computations is growing at least as fast as the improvements in processor technology. Scientific applications need to be tuned to extract near peak performance even as hardware platforms change underneath them. Unfortunately, tuning even the simplest real-world operations for high performance usually requires an intense and sustained effort, stretching over a period of weeks or months, from the most technically advanced programmers, who are inevitably in very scarce supply. While access to necessary computing and information technology has improved dramatically over the past decade, the efficient application of scientific computing techniques still requires levels of specialized knowledge in numerical analysis, mathematical software, computer architectures, and programming languages that many working researchers do not have the time, the energy, or the inclination to acquire. With good reason scientists expect their computing tools to serve them and not the other way around. And unfortunately, the growing desire to tackle highly interdisciplinary problems using more and more realistic simulations on increasingly complex computing platforms will only exacerbate the problem. The challenge for the development of next generation software is the successful management of the complex computing environment while delivering to the scientist the full power of flexible compositions of the available algorithmic alternatives and candidate hardware resources.

* This work is partially supported by the DOE LACSI - Subcontract #R71700J-29200099 from Rice University and by the NSF NPACI – P.O. 10181408-002 from University of California Board of Regents via Prime Contract #ASC-96-19020.

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2659, pp. 665–672, 2003.

With this paper we develop the concept of Self-Adapting Numerical Software (SANS) for numerical libraries that execute in the cluster computing setting. The central focus is the LAPACK For Clusters (LFC) software which supports a serial, single processor user interface, but delivers the computing power achievable by an expert user working on the same problem who optimally utilizes the resources of a cluster. The basic premise is to design numerical library software that addresses both computational time and space complexity issues on the user's behalf and in a manner as transparent to the user as possible. The software intends to allow users to either link against an archived library of executable routines or benefit from the convenience of prebuilt executable programs without the hassle of properly having to resolve linker dependencies. The user is assumed to call one of the LFC routines from a serial environment while working on a single processor of the cluster. The software executes the application. If it is possible to finish executing the problem faster by mapping the problem into a parallel environment, then this is the thread of execution taken. Otherwise, the application is executed locally with the best choice of a serial algorithm. The details for parallelizing the user's problem such as resource discovery, selection, and allocation, mapping the data onto (and off of) the working cluster of processors, executing the user's application in parallel, freeing the allocated resources, and returning control to the user's process in the serial environment from which the procedure began are all handled by the software. Whether the application was executed in a parallel or serial environment is presumed not to be of interest to the user but may be explicitely queried. All the user knows is that the application executed successfully and, hopefully, in a timely manner.

2 Related Efforts

Since the concept of self-adaptation appeared in the literature [27] it has been successfully applied in a wide range of projects. The ATLAS [31] project started as a "`DGEMM()` optimizer" [13] but continues to successfully evolve by including tuning for all levels of Basic Linear Algebra Subprograms (BLAS) [7, 8, 10, 9] and LAPACK [2] as well by making decisions at compilation and execution time. Functionality similar to ATLAS, but much more limited, was included in the PHiPAC [6] project. Iterative methods and sparse linear algebra operations are the main focus of numerous efforts. Some of them [30, 3] target convergence properties of iterative solvers in a parallel setting while others [1, 21, 20, 28, 26] optimize the most common numerical kernels or provide intelligent algorithmic choices for the entire problem solving process [5, 24]. In the area of parallel computing, researchers are offering automatic tuning of generic collective communication routines [29] or specific collectives as in the HPL project [12]. Automatic optimization of the Fast Fourier Transform (FFT) kernel has also been under investigation by many scientists [19, 18, 23]. In grid computing environments [17], holistic approaches to software libraries and problem solving environments such as defined in the GrADS project [4] are actively being tested. Proof of concept

efforts on the grid employing SANS components exist [25] and have helped in forming the approach followed in LFC.

3 LAPACK for Clusters Overview

The LFC software addresses the motivating factors from the previous section in a self-adapting fashion. LFC assumes that only a C compiler, an MPI [14–16] implementation , and some variant of the BLAS routines, be it ATLAS or a vendor supplied implementation, is installed on the target system. Target systems are intended to be "Beowulf like". There are essentially three components to the software: data collection routines, data movement routines, and application routines.

4 Typical Usage Scenario

The steps involved in a typical LFC run start with a user's problem that may be stated in terms of linear algebra. The problem statement is addressable with one of the LAPACK routines supported in LFC. For instance, suppose that the user has a system of n linear equations with n unknowns, $Ax = b$. There is a parallel computing environment that has LFC installed. The user is, for now, assumed to have access to at least a single node of said parallel computing environment. This is not a necessary constraint - rather a simplifying one. The user compiles the application code (that calls LFC routines) linking with the LFC library and executes the application from a sequential environment. The LFC routine executes the application returning an error code denoting success or failure. The user interprets this information and proceeds accordingly.

On the LFC software side, a decision is made upon how to solve the user's problem by coupling the cluster state information with a knowledge of the particular application. Specifically, a decision is based upon the scheduler's ability to successfully predict that a particular subset of the available processors on the cluster will enable a reduction of the total time to solution when compared to serial expectations for the specific application and user parameters. The relevant times are the time that is spent handling the user's data before and after the parallel application plus the amount of time required to execute the parallel application. If the decision is to solve the user's problem locally (sequentially) then the relevant LAPACK routine is executed. On the contrary, if the decision is to solve the user's problem in parallel then a process is forked that will be responsible for spawning the parallel job and the parent process waits for its return in the sequential environment. The selected processors are allocated (in MPI), the user's data is mapped (block cyclically decomposed) onto the processors (the data may be in memory or on disk), the parallel application is executed (e.g. ScaLAPACK), the data is reverse mapped, the parallel process group is freed, and the solution and control are returned to the user's process.

5 Performance Results

Figure 1 demonstrates the strength of the self-adapting approach of the LFC software. The problem sizes tested were 512, 1024, 2048, 4096, 8192, 12288, 14000. LFC chose 2, 3, 6, 8, 12, 16, 16 processes for these problems respectively. The oracle, which in theory knows the best parameters, utilized 4, 4, 8, 10, 14, 16, 16 processes respectively. The runs were conducted on a cluster of eight Intel Pentium III, 933 MHz dual processors, connected with a 100 Mb/s switch. In each run the data was assumed to start on disk and was written back to disk after the factorization. In the parallel environment, both the oracle and LFC utilized the I/O routines from ROMIO to load/store the data and the ScaLAPACK routine `PDGESV()` for the application code.

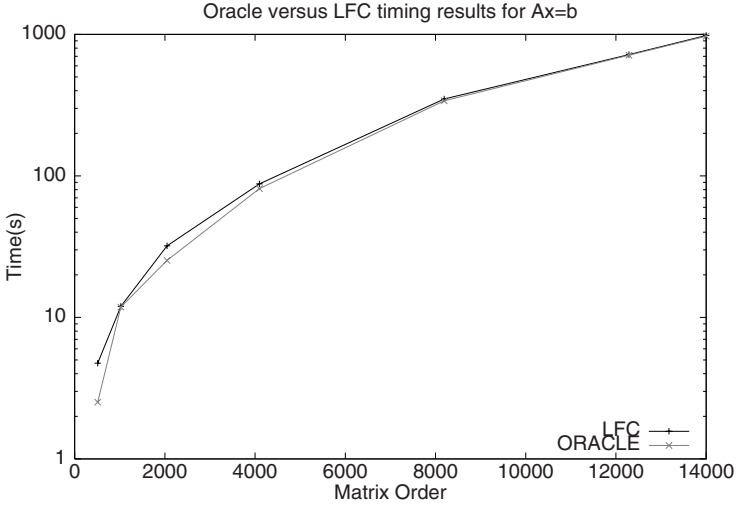

Fig. 1. Parellel execution times of the linear solver run by the oracle and LFC on a cluster.

Figure 2 illustrates the fact that the situation is more complicated than just selecting the right grid aspect ratio (e.g. the number of process rows divided by the number of process columns). Sometimes it might be beneficial to use a smaller number of processors. This is especially true if the number of processors is a prime number which leads to a flat process grid and thus very poor performance on many systems. It is unrealistic to expect that non-expert users will correctly make the right decisions here. It is either a matter of having expertise or experimental data to guide the choice and our experiences suggest that perhaps a combination of both is required to make good decisions consistently. Another point stressed by the figure is the widening gap (excluding merge points at prime processor numbers) between the worst and the optimal resource

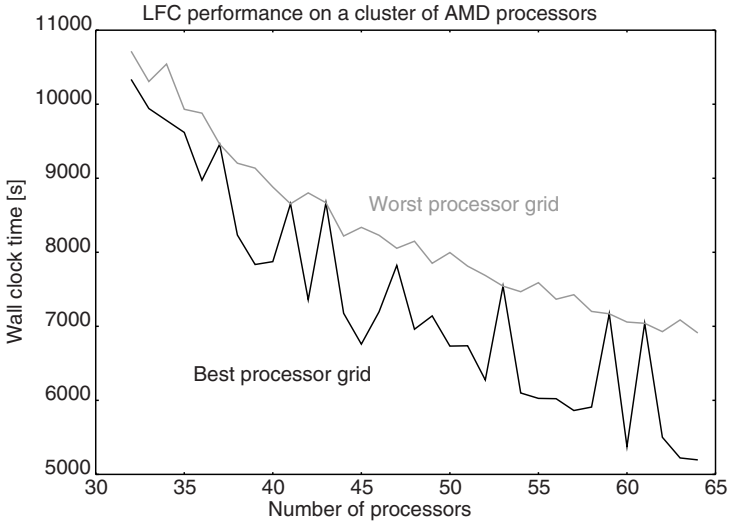

Fig. 2. Timing results for solving a linear system of order 70000 with the best and worst possible rectangular processor grid topologies reported.

choices for increasing number of processors. This shows the increasing chance for a user to use the hardware and software in a suboptimal way as more powerful computers become available.

As a side note, with respect to experimental data, it is worth mentioning that the collection of data for Figure 2 required a number of floating point operations that would compute the LU factorization of a square dense matrix of order almost three hundred thousand. Matrices of that size are usually suitable for supercomputers (the slowest supercomputer on the Top500 [22] list that factored such a matrix was on position 16 in November 2002) – an unlikely target machine for majority of users.

6 Conclusions and Future Work

As computing systems become more powerful and complex it becomes a major challenge to tune applications for high performance. We have described a concept and outlined a plan to develop numerical library software for systems of linear equations which adapts to the user's problem and the computational environment in an attempt to extract near optimum performance. This approach has applications beyond solving systems of linear equations and can be applied to most other areas where users turn to a library of numerical software for their solution.

At runtime our software makes choices at the software and hardware levels for obtaining a best parameter set for the selected algorithm by applying expertise from the literature and empirical investigations of the core kernels on the

target system. The algorithm selection depends on the size of the input data and empirical results from previous runs for the particular operation on the cluster. The overheads associated with this dynamic adaptation of the user's problem to the hardware and software systems available can be minimal.

The results presented here show unambiguously that the concepts of self adaptation can come very close to matching the performance of the best choice in parameters for an application written for a cluster. As Figure 1 highlights, the overhead to achieve this is minimal and the performance levels are almost indistinguishable. As a result, the burden on the user is removed and hidden in the software.

This paper has given a high level overview of the concepts and techniques used in self adapting numerical software. There are a number of issues that remain to be investigated in the context of this approach [11]. Issues such as adapting to a changing environment during execution, reproducibility of results when solving the same problem on differing numbers of processors, fault tolerance, rescheduling in the presence of additional load, dynamically migrating the computation, etc all present additional challenges which are ripe for further investigation. In addition, with Grid computing becoming mainstream, these concepts will find added importance [4].

Acknowledgements.
We wish to thank the Ohio Supercomputing Center (OSC), the Computational Science and Mathematics Division at Oak Ridge National Laboratory (XTORC cluster), the Center for Computational Sciences at Oak Ridge National Laboratory (Cheetah, Eagle), the Dolphin donation cluster (part of the SinRG program at the University of Tennessee Knoxville), the San Diego Supercomputing Center (SDSC), and the National Energy Research Scientific Computing Center (NERSC) for research conducted on their resources. We also wish to thank NPACI, the National Partnership for the Advancement of Computational Infrastrucure, for including LFC in its NPACkage.

References

1. R. Agarwal, Fred Gustavson, and M. Zubair. A high-performacne algorithm using preprocessing for the sparse matrix-vector multiplication. In *Proceedings of International Conference on Supercomputing*, 1992.
2. E. Anderson, Z. Bai, C. Bischof, Suzan L. Blackford, James W. Demmel, Jack J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and Danny C. Sorensen. *LAPACK User's Guide*. Society for Industrial and Applied Mathematics, Philadelphia, Third edition, 1999.
3. Richard Barrett, Michael Berry, Jack Dongarra, Victor Eijkhout, and Charles Romine. Algorithmic bombardment for the iterative solution of linear systems: A poly-iterative approach. *Journal of Computational and Applied Mathematics*, 74(1-2):91–109, 1996.

4. F. Berman. The GrADS project: Software support for high level grid application development. *International Journal of High Performance Computing Applications*, 15:327–344, 2001.
5. A. Bik and H. Wijshoff. Advanced compiler optimizations for sparse computations. *Journal of Parallel and Distributing Computing*, 31:14–24, 1995.
6. J. Bilmes et al. Optimizing matrix multiply using PHiPAC: a portable, high-performance, ANSI C coding methodology. In *Proceedings of International Conference on Supercomputing*, Vienna, Austria, 1997. ACM SIGARC.
7. Jack J. Dongarra, J. Du Croz, Iain S. Duff, and S. Hammarling. Algorithm 679: A set of Level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16:1–17, March 1990.
8. Jack J. Dongarra, J. Du Croz, Iain S. Duff, and S. Hammarling. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16:18–28, March 1990.
9. Jack J. Dongarra, J. Du Croz, S. Hammarling, and R. Hanson. Algorithm 656: An extended set of FORTRAN Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 14:18–32, March 1988.
10. Jack J. Dongarra, J. Du Croz, S. Hammarling, and R. Hanson. An extended set of FORTRAN Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 14:1–17, March 1988.
11. Jack J. Dongarra and Victor Eijkhout. Self-adapting numerical software for next generation applications. Technical report, Innovative Computing Laboratory, University of Tennessee, August 2002. `http://icl.cs.utk.edu/iclprojects/pages/sans.html`.
12. Jack J. Dongarra, Piotr Luszczek, and Antione Petitet. The LINPACK benchmark: Past, present, and future. *Concurrency and Computation: Practice and Experience*, 15:1–18, 2003.
13. Jack J. Dongarra and Clint R. Whaley. Automatically tuned linear algebra software (ATLAS). In *Proceedings of SC'98 Conference*. IEEE, 1998.
14. Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. *The International Journal of Supercomputer Applications and High Performance Computing*, 8, 1994.
15. Message Passing Interface Forum. MPI: A Message-Passing Interface Standard (version 1.1), 1995. Available at: `http://www.mpi-forum.org/`.
16. Message Passing Interface Forum. MPI-2: Extensions to the Message-Passing Interface, 18 July 1997. Available at `http://www.mpi-forum.org/docs/mpi-20.ps`.
17. Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, 1999.
18. M. Frigo. A fast Fourier transform compiler. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, Atlanta, Georgia, USA, 1999.
19. M. Frigo and S. G. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proceedings International Conference on Acoustics, Speech, and Signal Processing*, Seattle, Washington, USA, 1998.
20. E.-J. Im. *Automatic optimization of sparse matrix-vector multiplication*. PhD thesis, University of California, Berkeley, California, 2000.
21. E.-J. Im and Kathy Yelick. Optimizing sparse matrix-vector multiplication on SMPs. In *Ninth SIAM Conference on Parallel Processing for Scientific Computing*, San Antonio, Texas, 1999.

22. Hans W. Meuer, Erik Strohmaier, Jack J. Dongarra, and Horst D. Simon. *Top500 Supercomputer Sites*, 20th edition edition, November 2002. (The report can be downloaded from `http://www.netlib.org/benchmark/top500.html`).
23. D. Mirkovic and S. L. Johnsson. Automatic performance tuning in the UHFFT library. In *2001 International Conference on Computational Science*, San Francisco, California, USA, 2001.
24. Jakob Ostergaard. *OptimQR – A software package to create near-optimal solvers for sparse systems of linear equations.* `http://ostenfeld.dk/~jakob/OptimQR/`.
25. Antoine Petitet et al. Numerical libraries and the grid. *International Journal of High Performance Computing Applications*, 15:359–374, 2001.
26. Ali Pinar and Michael T. Heath. Improving performance of sparse matrix-vector multiplication. In *Proceddings of SC'99*, 1999.
27. J. R. Rice. On the construction of poly-algorithms for automatic numerical analysis. In M. Klerer and J. Reinfelds, editors, *Interactive Systems for Experimental Applied Mathematics*, pages 31–313. Academic Press, 1968.
28. Sivan Toledo. Improving the memory-system performance of sparse matrix-vector multiplication. *IBM Journal of Research and Development*, 41(6), November 1997.
29. Sathish Vadhiyar, Graham Fagg, and Jack J. Dongarra. Performance modeling for self adapting collective communications for MPI. In *Los Alamos Computer Science Institute Symposium (LACSI 2001)*, Sante Fe, New Mexico, 2001.
30. R. Weiss, H. Haefner, and W. Schoenauer. *LINSOL (LINear SOLver) – Description and User's Guide for the Parallelized Version.* University of Karlsruhe Computing Center, 1995.
31. R. Clint Whaley, Antoine Petitet, and Jack J. Dongarra. Automated empirical optimizations of software and the ATLAS project. *Parallel Computing*, 27(1-2):3–35, 2001.