The LAPACK for Clusters Project: an Example of Self Adapting Numerical Software*

Zizhong Chen† Jack Dongarra‡† Piotr Luszczek† Kenneth Roche†

June 14, 2003

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 especially 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.

Self-Adapting Numerical Software (SANS) systems are intended to meet this significant challenge [1]. In particular, the LAPACK For Clusters (LFC) project, that we describe in this paper, focuses on issues related to solving linear systems for dense matrices on highly parallel computers. Empirical studies [2] of computing solutions to linear systems of equations demonstrated the viability of the method finding that (on the clusters tested) there is a problem size that serves as a threshold. For problems greater in size than this threshold, the time saved by the self-adaptive method scales with the parallel application justifying the approach. In other words, the user saves time employing the self-adapting software.

This paper is organized as follows. Section 2 provides a

*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.

†Innovative Computing Laboratory, Computer Science Department, University of Tennessee Knoxville, {dongarra,luszczek,roche,zchen}@cs.utk.edu

‡Computational Science and Mathematics Division, Oak Ridge National Laboratory, Tennessee

general discussion of self adaptation and its relation to numerical software and algorithms. Section 3 introduces and gives some details on LFC while sections 4 and 5 describe the design and implementation of LFC in much greater detail. Section 6 attempts to substantiate the claims from the preceding sections with experimental data. Finally, sections 7 and 8, respectively, conclude the paper and acknowledge those that helped in its preparation.

2 Related Work

Achieving optimized software in the context described here is an $\mathcal{NP}$-hard problem [3, 4, 5, 6, 7, 8, 9]. Nonetheless, self-adapting software attempts to tune and approximately optimize a particular procedure or set of procedures according to details about the application and the available means for executing the application.

The ability to adapt to various circumstances may be perceived as choosing from a collection of algorithms and parameters to solve a problem. Such a concept has been appearing in the literature [10] and currently is being used in a wide range of numerical software components. The ATLAS [11] project started as a "`DGEMM` optimizer" [12] but continues to successfully evolve by including tuning for all levels of Basic Linear Algebra Subroutines (BLAS) [13, 14, 15, 16] and LAPACK [17] as well as by making decisions at compilation and execution time. Similar to ATLAS, but perhaps more limited, functionality was included in the PHiPAC [18] project. Iterative methods and sparse linear algebra operations are the main focus of numerous efforts. Some of them [19, 20] target convergence properties of iterative solvers in a parallel setting while others [21, 22, 23, 24, 25] optimize the most common numerical kernels or provide intelligent algorithmic choices for the entire problem solving process [26, 27]. In the area of parallel computing, researchers are offering automatic tuning of generic collective communication routines [28] or specific ones as in the HPL project [29]. Automatic optimization of the Fast Fourier Transform (FFT) kernel has also been under investigation by many scientists [30, 31, 32]. In grid computing environments [33], holistic approaches to software libraries and problem solving environments such as defined in the GrADS project [34] are actively being tested. Proof of concept efforts on the grid employing SANS components exist [35] and have helped in forming the approach followed in LFC.

3 LFC Overview

With this paper we develop the concept of Self-Adapting Numerical Software for numerical libraries that execute in the cluster computing setting. The central focus is the 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 transparent to the user. 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 resolving 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 in less time 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 explicitly queried. All the user knows is that the application executed successfully and, hopefully, in a timely manner.

LFC addresses user's problems that may be stated in terms of numerical linear algebra. The problem may otherwise be dealt with one of the LAPACK [17] routines supported in LFC. In particular, suppose that the user has a system of n linear equations with n unknowns, $Ax = b$. Three common factorizations apply for such a system and are currently supported by LFC (see Table 1 for details).

LFC assumes that only a C compiler, an Message Passing Interface (MPI) [36, 37, 38] implementation such as MPICH [39] or LAM MPI [40], 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" [41].

4 LFC' Selection Processes

Given that decompositional techniques are methods of choice for dense linear systems [42], LFC follows a two-step procedure to obtain a solution: it first factorizes the system matrix and then performs substitution with the factors.

Currently, LFC chooses its factorization algorithm among the following:

- piped-recursive (serial environment),
- in-memory blocked (serial environment),
- distributed memory, right-looking (parallel environment).

In the sequential environment, a stand-alone variant of the relevant LAPACK routines form the backbone of the serial applications in LFC. Achieving high performance in a sequential environment might seem trivial for expert users. Thus, we provide linker interface to enable such users to take advantage their favorite BLAS library. However, less experienced users could possibly have problems while dealing with linker dependencies. For such users, we provide an executable binary that is correctly built and capable of solving a linear system in a child process with data submitted through a system pipe. Two overheads result from such an approach: the time spent in `fork(2)` and `exec(3)` system calls and copying the data between separate process' address spaces. Intuitively (and empirically as shown later on), both overheads will have a lesser impact with increasing dimension of the matrix (the system calls, data copying and linear solver have computational complexities $O(1)$, $O(n^2)$, and $O(n^3)$ respectively).

The LFC' parallel solver have a stand-alone variant of the relevant ScaLAPACK and Basic Linear Algebra Communication Subprograms (BLACS). This allows leveraging a large body of expertise as well as software design and engineering. It also allows developers to focus on new issues and address common problems encountered by users.

Once a decision has been made and the parallel algorithm to be used, there are still possibilities to be accounted for in order to achieve desirable levels of performance. This phase of selection concerns various parameters of the aforementioned parallel algorithm. However, it is not merely a final code tuning exercise as the wrong choices may have disastrous effects in performance terms. The most important possibilities include:

- block size,
- processor count,
- logical process grid aspect ratio,
- processor set.

LAPACK and ScaLAPACK require a tuning parameter – a block size – which is crucial to attaining high performance. If LAPACK's functionality was embedded in an archived library which was supplied by the vendor then the burden of selecting the block size would have been removed from the user. However, if the vendor supplies only a BLAS library then the block size selection is to be made by the user and there is a possibility of degrading the performance by inappropriate choice. Thus, all the effort that went into tuning the BLAS may be wasted. It is possible to solve the problem in a sequential environment because of theoretical advances [43, 44, 45] in the decompositional approach in matrix computations. But in a parallel setting, the procedure is still not mature enough [46] and consequently there is a need for extra effort when selecting parameters that will define the parallel runtime environment for the specific application.

As mentioned earlier, in the parallel environment, proper choice of resources is harder than in the sequential setting due to a larger set of parameters to selected properly. ScaLAPACK Users' Guide [47] provides the following equation for predicting the total time T spent in one of its linear solvers (LL^T, LU, or QR) on matrix of size n in N_P processes [48]:

$$T(n,N_P) = \frac{C_f n^3}{N_P} t_f + \frac{C_v n^2}{\sqrt{N_P}} t_v + \frac{C_m n}{N_B} t_m \qquad (1)$$

where:

Driver	C_f	C_v	C_m
LU	$2/3$	$3+1/4\log_2 N_P$	$N_B(6+\log_2 N_P)$
LL^T	$1/3$	$2+1/2\log_2 N_P$	$4+\log_2 N_P$
QR	$4/3$	$3+\log_2 N_P$	$2(N_B\log_2 N_P+1)$

Table 1: Performance parameters of ScaLAPACK. All costs entries correspond to a single right-hand side; LU, LL^T and QR correspond to `PxGESV`, `PxPOSV`, and `PxGELS` routines, respectively.

- t_f time per floating-point operation (matrix-matrix multiplication flop rate is a good starting approximation)
- t_m corresponds to latency
- $1/t_v$ corresponds to bandwidth
- C_f corresponds to number of floating-point operations (see Table 1)
- C_v and C_m correspond to communication costs (see Table 1)

In contrast, for a single processor the equation is:

$$T_{seq}(n) = C_f n^3 t_f \tag{2}$$

The equation (1) yields surprisingly good predictions (more homogeneous the system, the more accurate the prediction). The surprise factor comes from the number of simplifications that were made in the model which was used to derive the equation. The hard part in using the equation is measuring system parameters which are related to some of the variables in the equation. The hardship comes from the fact that these variables do not correspond directly to typical hardware specifications and cannot be obtained through simple tests. In a sense, this situation may be regarded as if the equation had some hidden constants which are to be discovered in order to obtain reasonable predictive power. At the moment we are not aware of any reliable way of acquiring those parameters and thus we rely on parameter fitting approach that uses timing information from previous runs.

Our experimental results (presented later on) illustrate the fact that it is far from sufficient to choose the right number of processors. The key is to select the right aspect ratio of the logical process grid (unfortunately equation (1) does not account for it), i.e. the number of process rows divided by the number of process columns. In particular, sometimes it might be better for overall performance to use a smaller number of processors than the total number of the available ones.

Lastly there exist influential aspects in decision making process that are worth mentioning:

- per-processor performance (parallel efficiency),
- resource utilization (memory),
- time to solution (user perspective).

Trivially, the owner of the system is interested in optimal resource utilization while the user expects the shortest time to obtain the solution. Instead of aiming at optimization of either the former (by maximizing memory utilization and sacrificing the total solution time by minimizing the number of processes involved) or the latter (by using all the available processors) LFC attempts to balance the two. In the future, we envisage more general scheduling policies to take over this balancing and provide a better control of how the resources are used.

5 Resource Monitoring and Data Mapping

LFC allows to choose various variants of monitoring:

- interface to Network Weather Service (NWS),
- LFC monitoring daemon, and
- manual (user-guided) resource specification.

LFC uses discovery and adaptivity to assist the user in problem solving. Cluster state information is continually being assembled. The services provided by the NWS [49] may be utilized by LFC, provided that the NWS sensors have been configured to monitor the same resources that are used by parallel MPI applications.

A solution catered specifically to LFC' needs has also been developed. While conceptually similar, the solution addresses the monitoring problem from a standpoint of a

parallel MPI application. The following steps are repeated by the information gathering daemon process: a processor discovery routine is invoked that accounts for the existence of candidate resources, the available physical memory per processor is assessed, the time-averaged CPU load of each processor in a node is assessed, read/write times per processor to/from the local and network disks is assessed, point-to-point and global communications latencies and bandwidths on the cluster are assembled, and the core kernel of matrix-matrix multiplication is studied per processor. In addition to this data gathering cycle, there is an interest in the one-time discovery of the underlying memory hierarchy. Random access loads and stores with uniform and non-uniform stride help with this discovery.

In addition, there exists also a choice of a largely simplified solution for restrictive environments. In such a setting, the user manually (and presumably in a static manner) describes the computational environment to be used by LFC.

Data mapping component of LFC moves the data describing a linear system between sequential and parallel environments. The former being the initial user setting and the latter being the one chosen by LFC for execution. The component takes care of data transformation from their initial layout into the one more suitable for parallel execution – 2D block cyclic distribution [50].

6 Experimental Results

First, we present experimental results on a single processor that show the array of possibilities available for the user of LFC. In the sequential environment, a standalone (i.e. without user-visible linker dependencies) variant of the relevant LAPACK routines form the backbone of the serial applications in LFC. Achieving high performance in such an environment might be questionable as it involves overheads of system calls and data copying as it was mentioned earlier. The following data sheds some light on the impact that these overheads have on the real world performance of a linear solver. Figure 1 shows the performance data. The figure compares the following types of solvers:

- Recursive LU solver with optimized (in this case from ATLAS) BLAS – an expert user case,

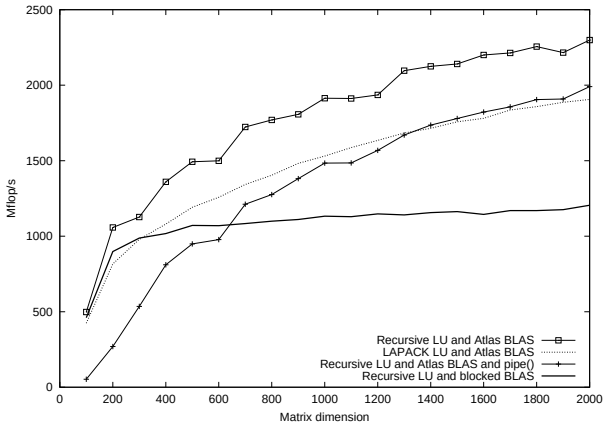

Figure 1: Performance comparison between the standard LAPACK's linear solve routine and the same routine executed in a separate process created with `fork(2)` and `exec(3)` system calls (matrix and right hand side data are sent through a system pipe, only solution data is sent back). The tested machine had an Intel Pentium 4 2.4 GHz processor with Atlas 3.4.1.

- LAPACK (block iterative) LU solver with optimized (in this case from ATLAS) BLAS – an expert LAPACK user case,

- Recursive LU solver with optimized (in this case from ATLAS) BLAS invoked through a system pipe – a case for an LFC user without knowledge of BLAS linking requirements,

- Recursive LU solver with blocked BLAS [51] – a case for any user on a RISC-based system without optimized BLAS.

For large enough matrices there exists about 10%-20% of performance drop and the difference decreases as the matrix size increases. We believe that for many users this is a price worth paying for convenience and certainty.

Next, we show results from experiments conducted with LFC' linear solver in a parallel setting. Figure 2 illustrates the relationship between the number of processors and the aspect ratio of the logical process grid (the number of process rows divided by the number of process columns). Sometimes it might be benefi-

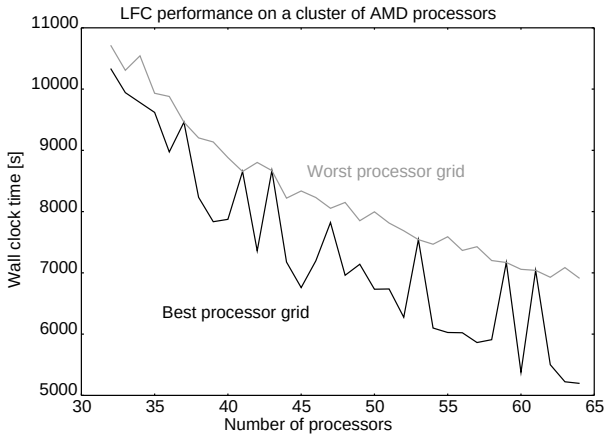

Figure 2: Time to solution of a linear system of order 70000 with the best and worst aspect ratios of the logical process grid. (Each processor was an AMD Athlon 1.4 GHz with 2 GB of memory; the interconnect was Myrinet 2000.)

Hardware specifications	
CPU type	Intel Pentium 4
CPU clock rate	2400 MHz
System bus clock rate	400 MHz
Level 1 data cache	8 KB
Level 1 instruction cache	12k μops*
Level 2 unified cache	256 KB
SMP CPUs per box	2
Main memory per SMP box	2048 MB
Ethernet Interconnect (switched)	100 Mbps
Single CPU performance	
Peak floating-point performance	4800 Mflop/s
Matrix-matrix multiply – `DGEMM`	≈3200 Mflop/s
Matrix-vector multiply – `DGEMV`	≈470 Mflop/s

*Intel has not made public the exact size of the trace cache in bytes; they've only said how many μops of unspecified size it holds.

Table 2: Parameters of the Pentium 4 cluster (running Linux OS) that was used in tests. The `DGEMV` and `DGEMM` rates were measured with ATLAS 3.4.1 (with SSE2 extensions enabled).

cial not to use all the available 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 in every case. 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. As a side note, with respect to the 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 [52] list that factored such a matrix was on position 16 in November 2002).

The following graphs show experimental results from a Pentium 4 cluster. The detailed description of the cluster is summarized in Table 2. Figures 3 and 4 compare performance of the LFC' parallel on 30 and 60 CPUs of the Pentium 4 cluster for matrices of dimension between 10000 and 20000. For matrix size 20000, 30 processors (provided the aspect ratio of the logical aspect grid is chosen optimally) have nearly 50% better efficiency. However, the time to solution for the same matrix size may be nearly 50% shorter for 60 processors. Figures 5 and 6 attempt to show the large extent to which the aspect ratio of the logical process grid influences per-processor performance of the LFC' parallel solver. The graphs in the figures shows data for various numbers of processors (between 40 and 64) and consequently does not represent a function since, for example, ratio 1 may be obtained with 7 by 7 and 8 by 8 process grids (within the specified range of number of processors). Figure 5 shows performance of parallel Gaussian elimination algorithm while Figure 6 shows performance of parallel Cholesky code. A note on relevance of data from Figures 5 and 6: they are only relevant for the LFC' parallel linear solvers that are based on the solvers from ScaLAPACK [47], they would not be indicative of performance of a different solver, e.g. HPL [29] which uses different communication patterns and consequently behaves differently with respect to the aspect ratio.

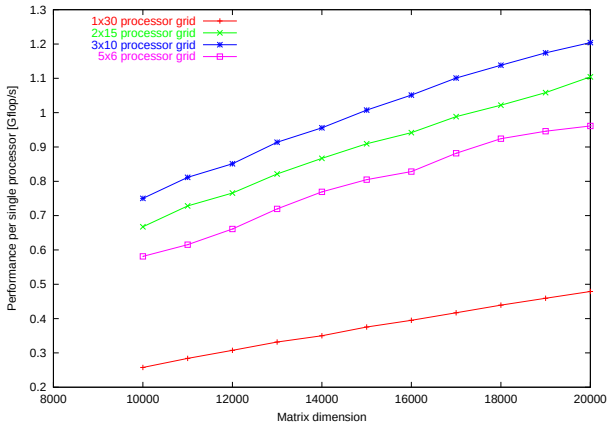

Figure 3: Per-processor performance of the LFC' parallel linear solver on the Pentium 4 cluster using 30 CPUs and various aspect ratios of the logical process grid.

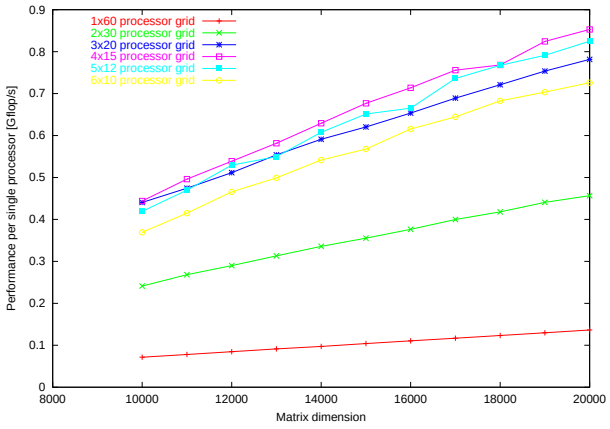

Figure 4: Per-processor performance of the LFC' parallel linear solver on the Pentium 4 cluster using 60 CPUs and various aspect ratios of the logical process grid.

7 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 users problem and the computational environment in an attempt to extract near optimum performance. This approach has applications beyond solving systems of 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 users 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 our experiments indicate, 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.

8 Acknowledgments

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 and Eagle), resources of the Scalable Intracampus Research Grid (SInRG) Project at the University of Tennessee supported by the National Science Foundation CISE Research Infrastructure Award EIA-9972889, 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 Infrastructure, for including LFC in its NPACkage.

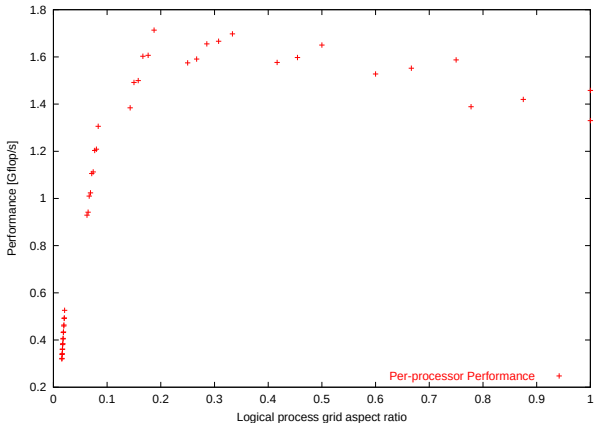

Figure 5: Per-processor performance of the LFC' parallel linear LU solver on the Pentium 4 cluster as a function of the aspect ratio of the logical process grid (matrix size was 72000 and the number of CPUs varied between 48 and 64).

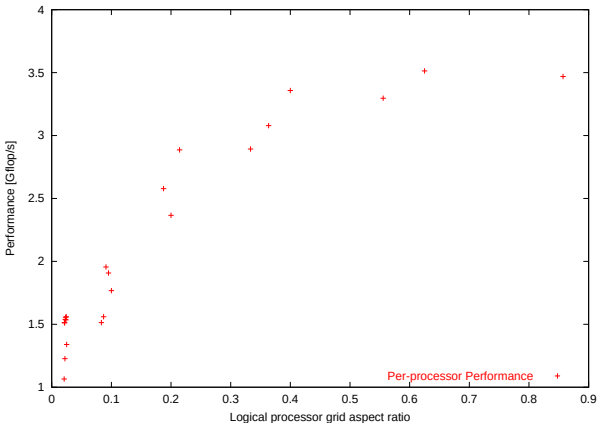

Figure 6: Per-processor performance of the LFC' parallel Cholesky (LL^T) linear solver on the Pentium 4 cluster as a function of the aspect ratio of the logical process grid (matrix size was 72000 and the number of CPUs varied between 40 and 48).

References

[1] Jack J. Dongarra and Victor Eijkhout. Self adapting numerical algorithms for next generation applications. *International Journal of High Performance Computing Applications*, 17(2):125–132, 2003. ISSN 1094-3420.

[2] Kenneth J. Roche and Jack J. Dongarra. Deploying parallel numerical library routines to cluster computing in a self adapting fashion. In *Parallel Computing: Advances and Current Issues*. Imperial College Press, London, 2002.

[3] E. Amaldi and V. Kann. On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems. *Theoretical Computer Science*, 209:237–260, 1998.

[4] Pierluigi Crescenzi and Viggo Kann (editors). A compendium of NP optimization problems. `http://www.nada.kth.se/theory/problemlist.html`.

[5] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

[6] J. Gergov. Approximation algorithms for dynamic storage allocation. In *Proceedings of the 4th Annual European Symposium on Algorithms*, pages 52–56. Springer-Verlag, 1996. Lecture Notes in Computer Science 1136.

[7] D. Hochbaum and D. Shmoys. A polynomial approximation scheme for machine scheduling on uniform processors: using the dual approach. *SIAM Journal of Computing*, 17:539–551, 1988.

[8] V. Kann. Strong lower bounds on the approximability of some NPO PB-complete maximization problems. In *Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science*, pages 227–236. Springer-Verlag, 1995. Lecture Notes in Computer Science 969.

[9] J. Lenstra, D. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, 46:259–271, 1990.

[10] 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.

[11] 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.

[12] Jack J. Dongarra and Clint R. Whaley. Automatically tuned linear algebra software (ATLAS). In *Proceedings of SC'98 Conference*. IEEE, 1998.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] E.-J. Im. *Automatic optimization of sparse matrix-vector multiplication*. PhD thesis, University of California, Berkeley, California, 2000.

[24] Ali Pinar and Michael T. Heath. Improving performance of sparse matrix-vector multiplication. In *Proceddings of SC'99*, 1999.

[25] Sivan Toledo. Improving the memory-system performance of sparse matrix-vector multiplication. *IBM Journal of Research and Development*, 41(6), November 1997.

[26] A. Bik and H. Wijshoff. Advanced compiler optimizations for sparse computations. *Journal of Parallel and Distributing Computing*, 31:14–24, 1995.

[27] Jakob Ostergaard. *OptimQR – A software package to create near-optimal solvers for sparse systems of linear equations*. `http://ostenfeld.dk/~jakob/OptimQR/`.

[28] 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.

[29] Jack J. Dongarra, Piotr Luszczek, and Antoine Petitet. The LINPACK benchmark: Past, present, and future. *Concurrency and Computation: Practice and Experience*, 15:1–18, 2003.

[30] M. Frigo. A fast Fourier transform compiler. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, Atlanta, Georgia, USA, 1999.

[31] 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.

[32] 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.

[33] Ian Foster and Carl Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, 1999.

[34] F. Berman et al. The GrADS project: Software support for high level grid application development. *International Journal of High Performance Computing Applications*, 15:327–344, 2001.

[35] Antoine Petitet et al. Numerical libraries and the grid. *International Journal of High Performance Computing Applications*, 15:359–374, 2001.

[36] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. *The International Journal of Supercomputer Applications and High Performance Computing*, 8, 1994.

[37] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard (version 1.1), 1995. Available at: `http://www.mpi-forum.org/`.

[38] 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`.

[39] MPICH. `http://www.mcs.anl.gov/mpi/mpich/`.

[40] LAM/MPI parallel computing. `http://www.mpi.nd.edu/lam/`.

[41] Thomas Sterling. *Beowulf Cluster Computing with Linux (Scientific and Engineering Computation)*. MIT Press, October 2001.

[42] Jack J. Dongarra, Iain S. Duff, Danny C. Sorensen, and Henk A. van der Vorst. *Numerical Linear Algebra for High-Performance Computers*. Society for Industrial and Applied Mathematics, Philadelphia, 1998.

[43] Erik Elmroth and Fred G. Gustavson. New serial and parallel recursive QR factorization algorithms for SMP systems. In *Proceedings of PARA 1998*, 1998.

[44] Fred G. Gustavson. Recursion leads to automatic variable blocking for dense linear-algebra algorithms. *IBM Journal of Research and Development*, 41(6):737–755, November 1997.

[45] Sivan Toledo. Locality of reference in LU decomposition with partial pivoting. *SIAM J. Matrix Anal. Appl.*, 18(4):1065–1081, October 1997.

[46] Dror Irony and Sivan Toledo. Communication-efficient parallel dense LU using a 3-dimensional approach. In *Proceedings of the 10th SIAM Conference on Parallel Processing for Scientific Computing*, Norfolk, Virginia, USA, March 2001.

[47] L. Suzan Blackford, J. Choi, Andy Cleary, Eduardo F. D'Azevedo, James W. Demmel, Inderjit S. Dhillon, Jack J. Dongarra, Sven Hammarling, Greg Henry, Antoine Petitet, Ken Stanley, David W. Walker, and R. Clint Whaley. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, 1997.

[48] J. Choi, Jack J. Dongarra, Susan Ostrouchov, Antoine Petitet, David W. Walker, and R. Clint Whaley. The design and implementation of the ScaLAPACK LU, QR, and Cholesky factorization routines. *Scientific Programming*, 5:173–184, 1996.

[49] Rich Wolski et al. The network weather service: A distributed resource performance forecasting service

for metacomputing. *Future Generation Computing Systems*, 14, 1998.

[50] Antoine Petitet. *Algorithmic Redistribution Methods for Block Cyclic Decompositions*. Computer Science Department, University of Tennessee, Knoxville, Tennessee, December 1996. PhD dissertation.

[51] B. Kågström, P. Ling, and Charles Van Loan. Portable high performance GEMM-based Level 3 BLAS. In *Proceedings of the 6th SIAM Conference on Parallel Processing for Scientific Computing*, pages 339–346, Philadelphia, 1993.

[52] Hans W. Meuer, Erich 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`).