Broadcast-Based Parallel LU Factorization

Fernando G. Tinetti* and Armando E. De Giusti

III-LIDI, Facultad de informática
Universidad Nacional de La Plata
50 y 115, 1er. Piso
1900 La Plata, Argentina

Abstract. This paper presents a parallel LU factorization algorithm designed to take advantage of physical broadcast communication facilities as well as overlapping of communication and computing. Physical broadcast is directly available on Ethernet networks hardware, one of the most used interconnection networks in current clusters installed for parallel computing. Overlapped communication is a well-known strategy for hiding communication latency, which is one of the most common source of parallel performance penalization. Performance analysis and experimentation of the proposed parallel LU factorization algorithm are presented. Also, the performance of the proposed algorithm is compared with that of the algorithm used in ScaLAPACK (Scalable LAPACK), which is commonly accepted as having optimized performance.

1 Introduction

Parallel computing on low-cost clusters is now a common approach in many scientific areas [3] [4]. Problems of linear algebra in general, and systems of equations in particular, have usually taken advantage of such parallel computing platforms to reduce the time needed to obtain a solution. In this context of linear algebra, there are some libraries avilable for parallel computing, such as ScaLAPACK [6] and PLAPACK [22]. Libraries specifically designed for linear algebra computing are used from many years ago. LAPACK (Linear Algebra PACKage) [2] is considered the *de facto* standard for the whole area of linear algebra applications, and the BLAS (Basic Linear Algebra Subroutines) library is well-suited for performance optimization, including parallelization [9].

Solving dense systems of equations is one of the most important tasks in the field of linear algebra. Furthermore, this problem is used as a benchmark for supercomputers, and the list of the 500 fastest computers is basically made by measuring the time to solve large systems of equations [12]. From the point of view of numerical processing, LU factorization is made following the well-known block processing approach, which is adopted for most (if not all) of the computing subroutines/algorithms in the field of linear algebra [16] [13]. For the parallel approach, bidimensional block cyclic decomposition is used in most of the approaches, included that of ScaLAPACK [10] [6]. Currently, the ScaLAPACK

* Investigador Asistente CICPBA

J.C. Cunha and P.D. Medeiros (Eds.): Euro-Par 2005, LNCS 3648, pp. 867–876, 2005.

approach is considered good enough to be well established and used, and research is being conducted for workload balance in heterogeneous and/or dynamically changing environments [5] [7] [8]. A parallel LU factorization algorithm based on broadcast messages is proposed taking into account that:

1) Many of the current parallel computing platforms are low-cost clusters of computers interconnected by Ethernet [15]. On these clusters, physical broadcast is available even when switching (with Ethernet switches) is used to allow multiple point-to-point simultaneous communications. Furthermore, the standard Ethernet is being upgraded for better performance [20] [1] maintaining backward compatibility in general, and the broadcast address in particular. From the theoretical point of view, physical broadcasts are very interesting because of their natural scalability. In practical terms, however, synchronizations and/or acknoledgements are necessary and the time taken for these tasks is usually dependent on the number of computers involved in broadcasts. Also, some other interconnection networks support physical broadcasts or multicasts, in top of which a broadcast routine can be designed [17] [14].

2) The block processing LU factorization defines a very clear pattern of data dependence, since it is defined in terms of iterations with [2] [16]: a) single *current* block processing, on a small submatrix of the matrix being factorized, and b) *trailing* matrix update with the processed *current* block. This implies that a *current* block is needed for most of the processing in all the iterations and, if the matrix is distributed amongst two or more computers, the *current* block should be broadcasted to the computers with elements to be updated.

3) The LU factorization is similar enough to other factorizations such as QR, Cholesky, etc. to expect that the same approach could be applied at least in those factorizations with a similar processing pattern. Taking the QR factorization as an example, the processing pattern is almost the same as that of the LU factorization, even when the processing on a single block and the trailing matrix update are not the same for the two factorizations [10].

2 Broadcast-Based Parallel LU Factorization Algorithm

Some issues should be addressed for a broadcast-based parallel LU factorization:

1) High message latency (or startup) times penalizing parallel performance. Message latency on clusters are mainly due to a combination of the message passing interface/library/operating system overhead plus the latency of the interconnection hardware involving interface cards and, usually, switches or hubs.

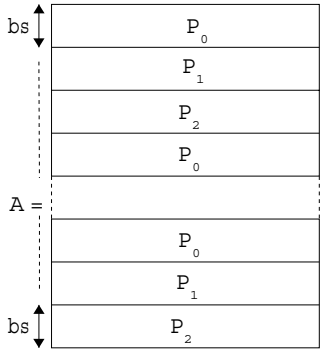
2) The broadcast message routine implemented in message passing libraries does not usually take advantage of the physical broadcast facilities provided by interconnection networks (e.g. Ethernet). Message passing libraries such as the freely available implementations of MPI [19] are not focused on optimizing the broadcast message routine by using physical broadcast or multicast facilities. Instead, message passing libraries implementations usually optimize point-to-point routine/s and implement broadcast in terms of spanning trees [17] or just multiple messages from the broadcast sender (the broadcast *root*).

3) The parallel algorithm itself, since most (if not all) current parallel algorithms are based on bidimensional data distributions and simultaneous broadcasts on the two dimensions or broadcasts and point-to-point communications expected to be carried out simultaneously, such as in ScaLAPACK [6]. Given that physical broadcasts are going to be used to optimize broadcast messages, more than one message at a given time should not be scheduled, since it would imply performance penalization by sequential communications. It is clear that a single computer cannot receive a physical broadcast and another frame/packet at the same time.

Each one of the above three issues have been addressed fairly straightforward, and decisions can be evaluated either analytically or by experimentation. The strategy to avoid the *high message latency* penalization has been overlapping local computing with communication, which is very useful on parallel computing. This overlapping should be introduced in the algorithm, thus becoming a constraint/guideline for designing the algorithm. A *broadcast message routine* has been designed and implemented on top of the User Datagram Protocol (UDP) [18], whose broadcast facilities are implemented by most of the operating systems (e.g. Linux, Solaris, AIX) with physical broadcast on Ethernet networks. The parallel algorithm is designed with one-dimensional data distribution, which in the specicic case of LU factorization is useful also to avoid communications for selecting the pivot/s on each iteration. More specifically, a *row block cyclic partitioning* is made, with the block size (or bs, number of consecutive rows in a block) defined for performance tuning, as in ScaLAPACK. In either case, the block size is small enough to have many more blocks than computers, and blocks are cyclically distributed. The *row block cyclic partitioning* used for LU matrix factorization amongst computers P_0, P_1, and P_2 is shown in Fig. 1-a). A simplified pseudocode of the process running on computer P_i is shown in Fig. 1-b), where `nblocks` $= n/bs$, `Factorize` implies LU factorization with pivoting on a single block, `send_b` and `recv_b` are the routines to send and receive a broadcast message respectively, and `Update` -on a single block or on local blocks- implies several tasks: a) applying pivots, b) a triangular system solve, and c) a matrix multiplication. In fact, numerical processing (i.e. factorization and update) is the same as in the sequential case [2] except that every computer modifies only local data. More details on the sequential as well as parallel LU factorization algorithm can ba found in [21].

The idealized case in which computing and communication are overlapped and there is no overhead due to broadcast messages (except for the communication of the first block) is shown in Fig. 2. Most of the local computing time in each computer (shown as Proc_i on Fig. 2) is mainly due the trailing matrix update which includes a matrix multiplication.

Time Required by Floating Point Operations. Processing requirements of the trailing matrix update depend on the iteration. Given a matrix of $n \times n$ elements, the block size bs, and starting iterations with $i = 1$, the matrix update in the i_th iteration is made on a submatrix of $(n - i * bs) \times (n - i * bs)$ elements. Furthermore, the trailing matrix update is defined as




```
if (i == 0)
  Factorize and send_b block 0
for (j = 0; j < nblocks; j++)
{
  if (block j is not local)
    recv_b factorized block j
  if (block j+1 is local)
  {
    Update and Factorize block j+1
    send_b factorized block j+1
  }
  Update local blocks /* block j+1
       has been already updated */
}
```

a) Row Block Cyclic Distribution. b) Parallel LU Factorization Pseudocode.

Fig. 1. Row Block Cyclic Partitioning.

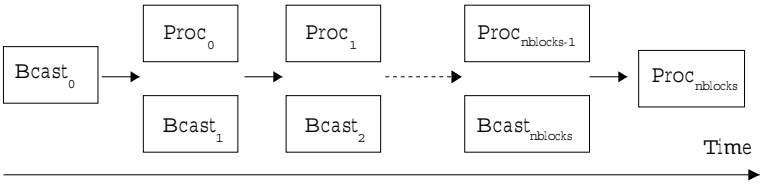


Fig. 2. Overlapped Computing and Communication.

$$tA - tL \times tU; \quad tA \in I\!R^{(n-i*bs)\times(n-i*bs)},\ tL \in I\!R^{(n-i*bs)\times bs},\ tU \in I\!R^{bs\times(n-i*bs)}$$

Thus, the number of floating point operations required for the trailing matrix update in iteration i, $FlopUpd(i)$, is given by

$$FlopUpd(i) = 2 * bs * (n - i * bs)^2 \tag{1}$$

because the matrix multiplication $tL \times tU$ requires $(n - i * bs)^2 * (2 * bs - 1)$ floating point operations and the matrix subtraction requires $(n - i * bs)^2$ floating point operations. Taking into account that the processing workload is evenly distributed amongst p computers, the time required for floating point operations on each computer in iteration i is given by

$$t(FlopUpd(i), p) = \frac{tf * FlopUpd(i)}{p} \tag{2}$$

where tf is the time required for a single floating point operation.

Time Required by Broadcast Communications. The timing model of point-to-point communications can be taken as a starting point:

$$t(m) = \alpha + \beta * m \tag{3}$$

where m is the amount of data to transfer, α is the communication latency or startup cost, and $1/\beta$ is the network communication bandwidth. Timing models for broadcast communication usually depend on the implementation selected for the broadcast routine in a specific implementation. In the specific case of the research presented in this paper, the broadcast message implementation is such that: a) data is physically broadcasted using UDP, and b) acknowledgements are received at the broadcast root from all the receivers to provide a reliable broadcast message. These details are hidden to the user (pertain to the broadcast implementation). Given that data is sent as in a point-to-point operation and there is a very low rate of message loss, the time required for data transmission through the network can be modeled as in the poit-to-point messages, i.e. $(\beta * m)$ in Eq (3), with $m = bs * (n - i * bs)$ on iteration i. However, ackowledgements sent from receivers to the root attempt against scalability, because these messages cannot be received simultaneously at the broadcast root. Even when there are multiple ways of avoiding such a performance drawback, it is still possible to analyze the time required by broadcast messages. Summarizing, the timing model for the broadcast operation in the i_th iteration is

$$t(bcast(i,p)) = \alpha_b + lpp * (p-1) + \beta * bs * (n - i * bs) \tag{4}$$

where α_b is the latency of the broadcast implementation independently of the number of computers involved, lpp is the latency per processor of the broadcast implementation, and $p-1$ is the number of receivers in a broadcast operation. Summarizing, Eq. (4) is the timing model of the broadcast implementation made in the context of the research related with this paper.

Time Required by the Algorithm. Taking into account the pseudocode of Fig. 1-b) and the algorithm behavior described in Fig. 2, the time required to complete the parallel algorithm on p processors is given by

$$t(parLU,p) = \sum_{i=1}^{n/bs} max(t(FlopUpd(i),p), t(bcast(i,p))) \tag{5}$$

It is expected that the numerical computing time in the first iterations is greater than the time required by broadcast communications. Also, given that a) the trailing matrix is made smaller as more iterations are completed, and b) broadcast message latency is constant from the point of view of trailing matrix size,

$$\begin{aligned} t(FlopUpd(i),p) &\geq t(bcast(i,p)); \quad i \leq k \\ t(FlopUpd(i),p) &< t(bcast(i,p)); \quad i > k \end{aligned}$$

and Eq. (5) *becomes*

$$t(parLU,p) = \sum_{i=1}^{k} t(FlopUpd(i),p) + \sum_{i=k+1}^{n/bs} t(bcast(i,p)) \tag{6}$$

3 Comparison with ScaLAPACK: Expected Time and Experimentation

The expected time for the Scalapack LU factorization algorithm is well known:

$$t(ScaLU, p) = \frac{2 * n^3}{3 * p} tf + \frac{(3 + log_2(p)/4) * n^2}{\sqrt{p}} \beta + (6 + log_2(p)) * n * \alpha_{ptp} \quad (7)$$

where α_{ptp} is the message latency for a point-to-point message [7] [8] [6], and the rest of parameters/coefficients have already been explained and used.

Some different points of view prevent a direct comparison among Eq. (7) and Eq. (6) above. The first term of Eq. (7) reflects the number of floating point operations in ScaLAPACK's timing model: $2/3 * n^3$. This is the *traditional* number of operations for the sequential LU factorization as given in the literature [16]. The timing model given for the proposed parallel algorithm takes into account that most of the computing time is needed for the trailing matrix update whose number of operations is given in Eq. (1) for the i_th iteration. However, both algorithms are directly based on the blocked LU factorization, so the number of floating point operations should be the same and it is not necessary a deeper comparison analysis to determine which one -Eq. (7) or Eq.(6)- is more accurate.

The ScaLAPACK timing model for communication is reflected in the second and third terms of Eq. (7). ScaLAPACK's communication costs are taken into account for every block/element of the matrix. On the other hand, for the approach proposed in this paper, Eq. (5) and Eq. (6) directly reflect that a broadcast communication adds time to the total expected algorithm time only when it is greater than the corresponding trailing matrix update time. Even when the numerical computing time is greater than the broadcast time in only a few iterations -e.g. $k < 20$ or $k < 30$ in Eq. (6)- the communication time (in those iterations) does not add time to the total processing time. However, the broadcast timing model of Eq. (4) is far from optimal and implies at least that the latency grows linearly with the number of processors. On the other hand, ScaLAPACK relies on spanning trees and, thus, the timing model implies a logarithmic growth depending on the number of processors.

Table 1. Cluster Characteristics.

Clock	Mem	Mflop/s (DGETRF)
2.4 GHz	1 GB	$\cong$ 2500

Some simple experimentation will clarify the comparison on a real environment. Computers (PCs) used for experimentation have the characteristics summarized in Table 1, and the interconnection network is 100 Mb/s Ethernet with complete switching. Performance in Table 1 is given in Mflop/s using DGETRF, the sequential LU matrix factorization with double precision floating point number representation. The total number of available computers is 20, and experiments were made with 2, 4, 8, 16, and 20 computers. Matrix sizes

are scaled up according to the number of computers and memory available. Local/sequential computing is made by using fully optimized ATLAS BLAS (Automatically Tuned Linear Algebra Software BLAS) [23]. ScaLAPACK communication is made as usual: BLACS (Basic Linear Algebra Communication Subroutines) implemented on top of MPICH implementation of MPI. Every possible bidimensional processors grid $P \times Q$ was considered for ScaLAPACK routines, e.g. for 16 processors, the experimental grids were: 1×16, 16×1, 2×8, 8×2, and 4×4. Also, square block sizes were used for ScaLAPACK routines: 16, 32, 64, 100 and 128. The proposed algorithm does not need to define a bidimensional processors grid, and the block values used for experimentation are the same as those used for ScaLAPACK routines.

Figure 3 shows the parallel perfomance measured as efficiency for LU matrix factorization on different number of computers from 2 to 20. The matrix order (size) for each number of computers is shown in parenthesis on the x axis. Bars show the best efficiency value obtained by the algorithms for each number of computers. Light gray bars labeled as "Sca" correspond to values obtained by ScaLAPACK's PDGETRF. Dark gray bars labeled as "Prop" correspond to values obtained by the proposed parallel LU matrix factorization algorithm. The proposed algorithm performance is better than that implemented in ScaLAPACK from the point of view of "raw" efficiency and performance degradation from 2 to 20 computers. It is worth to mention the similarity among the ScaLAPACK's results shown in Fig. 3 with those in [8], where ScaLAPACK is used for LU matrix decomposition and linear equation system solving. In Fig. 3 as well as in [8] the efficiency is about 0.5 (or 50% of the total available computing power). It is possible now to analyze the specific results regarding, for example, the advantage of broadcast overlapping of the proposed algorithm. For 20 computers, for example, the specific experimentation values are: $n = 45000$, $bs = 64$, the total number of blocks and iterations is 704 and

$$t(FlopUpd(i), p) \geq t(bcast(i, p)); \quad i = 1, \ldots, 363$$
$$t(FlopUpd(i), p) < t(bcast(i, p)); \quad i = 364, \ldots, 704$$

i.e. the first 363 broadcast messages do not add any time to the total elapsed time of the parallel algorithm on 20 computers. Thus, more than 51% of the broadcasts are completely made in background and this explains the very good parallel performance values shown in Fig. 3.

LU matrix factorization is specially penalized in ScaLAPACK's two dimensional matrix distribution due to the partial pivoting needed for numerical stability. Partial pivoting implies a collective communication in a row or a column of processors (for pivot selection) which implies a group communication penalization in an algorithm defined mainly for point-to-point communications. Given that the proposed parallel LU matrix factorization distributes data by column block or row block, this penalization is not found. Finally, the proposed parallel LU matrix factorization algorithm efficiency for 20 computers is about 7% worse than the efficiency for 2 computers, while SaLAPACK efficiency for 20 computers is about 23% worse than the efficiency for 2 computers.

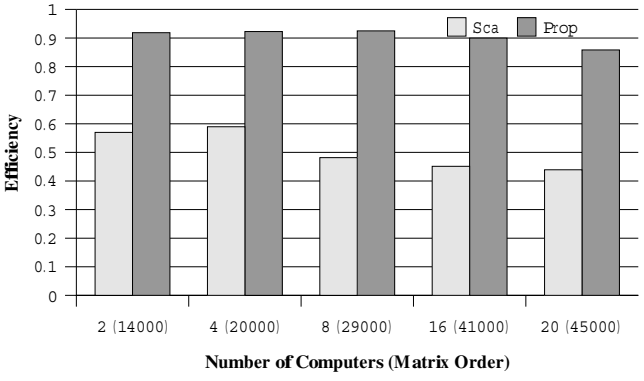


Fig. 3. LU Matrix Factorization Efficiency.

4 Conclusions and Further Work

A parallel LU factorization algorithm based on broadcast messages seems to be a good idea to optimize performance at least on clusters interconnected by networks with broadcast/multicast facilities, such as Ethernet and InfiniBand. Furthermore, the parallel LU matrix factorization algorithm presented in this paper is very simple and, thus, easy to understand and implement on clusters. A specific broadcast message routine has been implemented focusing Ethernet networks, and this routine has been successfully used in the parallel algorithm.

The performance analysis has been presented regarding the computing time of the proposed algorithm, which is dependent on the broadcast message implementation. Very good performance values are obtained in experiments by taking advantage of physical broadcast as well as overlapping communication with computing in the proposed parallel algorithm. These performance values are shown to be better than those obtained by the ScaLAPACK LU factorization algorithm, which is currently assumed to be optimized for distributed memory parallel computers. Furthermore, the proposed algorithm not only obtains better performance than that implemented in ScaLAPACK but also have better performance scalability as the number of computers is increased.

Other factorization algorithms from the field of linear algebra seem to be well suited for parallelization beased on broadcast messages. Factorization methods such as QR and Cholesky are among the inmediate candidates given their similarity in the numerical computing pattern, even when the individual operations are different from those in the LU factorization.

From the point of view of parallel hardware and clusters, it would be highly beneficial to experiment on more powerful clusters. The cluster computing power could be increased by having more computers as well as with computers with more processing power. Also, interconnection networks with performance better than Ethernet 100 Mb/s should be used. Immediate candidates in this sense are Ethernet 1 and 10 Gb/s.

Acknowledgements

Experimentation presented in this paper has been carried out at the Universidad Nacional de La Plata's JavaLab, with 20 IBM NetVista 8305-HRY, donated by IBM, as part of the Academic Cooperation Agreement signed between the University and IBM Argentina.

References

1. 10 Gigabit Ethernet Alliance (10GEA), 10 Gigabit Ethernet Technology Overview White Paper, May 2002. Available at http://www.10gea.org/10GEA_White_Paper_ Rev1_May2001.pdf.
2. Anderson E., Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, D. Sorensen, LAPACK Users' Guide (Second Edition), SIAM Philadelphia, 1995.
3. Anderson T., D. Culler, D. Patterson, and the NOW Team, "A Case for Networks of Workstations: NOW", IEEE Micro, Feb. 1995.
4. Baker M., R. Buyya, "Cluster Computing at a Glance", in R. Buyya Ed., High Performance Cluster Computing: Architectures and Systems, Vol. 1, Prentice-Hall, Upper Saddle River, NJ, USA, pp. 3-47, 1999.
5. Beaumont O., V. Boudet, A. Petitet, F. Rastello, Y. Robert, "A Proposal for a Heterogeneous Cluster ScaLAPACK (Dense Linear Solvers). IEEE Transactions on Computers, 50(10):1052-1070, 2001.
6. Blackford L., J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, R. Whaley, ScaLAPACK Users' Guide, SIAM, Philadelphia, 1997.
7. Chen Z., J. Dongarra, P. Luszczek, K. Roche, "Self adapting software for numerical linear algebra and LAPACK for clusters", Parallel Computing 29, pp. 1723-1743, Elsevier B.V., 2003.
8. Chen Z., J. Dongarra, P. Luszczek, K. Roche, "The LAPACK for Clusters Project: an Example of Self Adapting Numerical Software", Proceedings of the 37th Hawaii International Conference on System Sciences, pp. 1-10, 0-7695-2056-1/04, IEEE, 2004.
9. Choi J., J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, R. Whaley, "A proposal for a set of parallel basic linear algebra subprograms", Technical Report CS-95-292, University of Tennessee Knoxville, LAPACK Working Note 100, May 1995.
10. Choi J., J. Dongarra, L. Ostrouchov, A. Petitet, D. Walker, R. Whaley, "The Design and Implementation of the ScaLAPACK LU, QR, and Cholesky Factorization Routines", Report ORNL/TM-12470, Sep. 1994.
11. K. Dackland and E. Elmroth. "Design and Performance Modeling of Parallel Block Matrix Factorizations for Distributed Memory Multicomputers", Proceedings of the Industrial Mathematics Week, pp 102-116, Trondheim, 1992.
12. Dongarra J., "Performance of Various Computers Using Standard Linear Equations Software", University of Tennessee, Knoxville TN, 37996, Computer Science Technical Report Number CS - 89 - 85, January 2005, http://www.netlib.org/benchmark/performance.ps.
13. Dongarra J., D. Walker, "Libraries for Linear Algebra", in Sabot G. W. (Ed.), High Performance Computing: Problem Solving with Parallel and Vector Architectures, Addison-Wesley Publishing Company, Inc., pp. 93-134, 1995.

14. InfiniBand Trade Association, InfiniBand Architecture Specification, Release 1.0, October 24 2000.
15. Institute of Electrical and Electronics Engineers, Local Area Network - CSMA/CD Access Method and Physical Layer Specifications ANSI/IEEE 802.3 - IEEE Computer Society, 1985.
16. Golub G., C. Van Loan, Matrix Computations, 2nd Edition, The John Hopkins University Press, 1989.
17. Liu J., A. R, Mamidala, D. K. Panda, "Fast and Scalable MPI-Level Broadcast Using InfiniBand's Hardware Multicast Support", Proc. 18th International Parallel and Distributed Processing Symposium (IPDPS'04), p. 10b, Santa Fe, New Mexico, USA, April 2004.
18. Postel J., "User Datagram Protocol", RFC 768, USC/Information Sciences Institute, Aug. 1980.
19. Snir M., S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra., MPI: The Complete Reference, Volume 1 - The MPI-1 Core, 2nd edition. The MIT Press, 1998.
20. Spurgeon C. E., Ethernet: The Definitive Guide, O'Reilly and Associates, ISBN 1565926609, 2000.
21. Tinetti F. G., "LU Factorization: Number of Floating Point Operations and Parallel Processing in Clusters", Technical Report LIDI PLA-001-2003, May 2003, available at https://lidi.info.unlp.edu.ar/∼fernando/publis/LUops.pdf
22. van de Geijn R., Using PLAPACK: Parallel Linear Algebra Package, The MIT Press, 1997.
23. Whaley R. C., A. Petitet, J. J. Dongarra, Automated Empirical Optimization of Software and the ATLAS Project. Available at http://www.netlib.org/lapack/lawns/lawn147.ps