Efficient Broadcasts and Simple Algorithms for Parallel Linear Algebra Computing in Clusters

Fernando G. Tinetti[1, 2]
[1]Universidad Nacional de La Plata
Facultad de Informática, 50 y 115
1900 La Plata, Argentina
fernando@ada.info.unlp.edu.ar

Emilio Luque[2]
[2]Universidad Autónoma de Barcelona
Escuela Técnica Superior de Ingeniería
08193 Barcelona, España
emilio.luque@uab.es

Abstract

This paper presents a natural and efficient implementation for the classical broadcast message passing routine which optimizes performance of Ethernet based clusters. A simple algorithm for parallel matrix multiplication is specifically designed to take advantage of both, parallel computing facilities (CPUs) provided by clusters, and optimized performance of broadcast messages on Ethernet based clusters. Also, this simple parallel algorithm proposed for matrix multiplication takes into account the possibly heterogeneous computing hardware and maintains a balanced workload of computers according to their relative computing power. Performance tests are presented on a heterogeneous cluster as well as on a homogeneous cluster, where it is compared with the parallel matrix multiplication provided by the ScaLAPACK library. Another simple parallel algorithm is proposed for LU matrix factorization (a general method to solve dense systems of equations) following the same guidelines used for the parallel matrix multiplication algorithm. Some performance tests are presented over a homogeneous cluster.

1. Introduction

Computation intensive applications -often referred to as scientific processing- take advantage of the growing processing power of standard computers, along with their low cost and the relatively easy way in which they can be available for parallel processing. Every local area network (LAN) has the two main facilities needed for parallel computing: a) processing power, which is provided by each computer (CPU-memory), and b) interconnection network among CPUs, which is provided by the LAN hardware, usually Ethernet [11], and (at the software lowest levels), by the protocols implemented by the operating system, usually the TCP (Transmission Control Protocol), UDP (User Datagram Protocol), and IP (Internet Protocol) protocol stack.

In this context of cluster (parallel) computing, the message passing model is usually adopted as the programming model, which is based on CSP (Communicating Sequential Processes) [9]. Many software libraries have been proposed and are used to implement the message passing routines necessary to exchange data among parallel processes. Three of the most used at the time of this writing are PVM (Parallel Virtual Machine) [6] being the first *de facto* standard in this field, and free implementations of the *formal* standard MPI [13], such as MPICH [8] and LAM/MPI [3].

Linear algebra operations and methods are considered highly representative in the field of computation intensive applications. A great effort has been made in order to optimize solution methods for serial as well as parallel computing. The LAPACK (Linear Algebra PACKage) [1] and BLAS (Basic Linear Algebra Subroutines) [10] [5] [7] definitions represent the main results of this effort.

Parallel computing in clusters impose very specific and hard constraints for scientific and, more specifically, linear algebra operations:

- High performance processors and low performance interconnection network (high startup time and low data bandwidth). From this point of view, the parallel machine built up with computers in a LAN is unbalanced. For tightly coupled applications this unbalanced processing-communication performance implies a great effort in order to enlarge parallel computing granularity.
- Performance on Ethernet based interconnection network depends heavily on cabling hardware (hubs, switches, mixing of hub-switches). From the point of view of performance, switched Ethernet should be used. However, the hardware switching cost usually does not grow linearly with the number of computers, and it becomes very expensive when the number of computers grows enough. From another point of view, if installed networks of computers are going to be used as parallel machines, it is unlikely to have fully switched networks, or at least it depends on the organization policy for LAN hardware (i.e. it is not

possible to change easily the cabling hardware in order to have a fully switched Ethernet network for parallel processing).

- In heterogeneous clusters (with different computers in the LAN), there are strong difficulties to maintain balanced workload. For example, in [2] it is shown that bidimensional processing workload balancing problem is NP-Complete. It is worth noting that bidimensional data distribution is the most accepted data distribution (and hence processing workload balance) for linear algebra operations solved in parallel.

Thus, parallelization of linear algebra operations is a problem still to be solved in terms of obtaining optimized performance on clusters. The main project on which this work is contained focuses on a methodology to parallelize linear algebra operations and methods on Ethernet based clusters, taking into account the specific constraints explained above. The methodology involves optimizing algorithms in general, and communication patterns in particular, in order to have optimized parallel performance in clusters interconnected by Ethernet, thus having a performance comparable with more expensive and *ad hoc* interconnection networks, such as Myrinet, SCI (Scalable Coherent Interface), ATM (Asynchronous Transfer Mode), and HiPPI (High Performance Parallel Interface).

Matrix multiplication and LU factorization are the first problems approached and the PVM broadcast performance impose a strong performance penalty on the whole parallel performance. As MPI implementations are not reliable on broadcast performance, it is developed an Ethernet optimized broadcast. Parallel algorithms performance (and, partially, the parallelization methodology) is fully evaluated by experimentation.

Section 2 describes the simple algorithms proposed to solve a matrix multiplication and a LU factorization in parallel using clusters of computers. Preliminary experimental work is presented in Section 3, where it is made clear an optimized broadcast message is needed for acceptable performance. Section 4 presents the main items taken into account for an optimized broadcast message on Ethernet based clusters. Experimentation with algorithms on three different clusters are explained in section 5. Conclusions and possible enhancements are presented in section 6.

2. Parallel Linear Algebra Operations on Clusters

The parallel algorithms proposed on this paper follow a few but very restrictive guidelines, taking into account the specific characteristics of clusters used as parallel computers mentioned above:

- Coarse granularity, given the performance penalties implied by the unbalanced processing-communication ratio.
- Broadcast based communication between parallel processes, given that it is not always possible to have fully switched Ethernet networks, and Ethernet itself provides broadcast at its lowest level, thus giving some chance of optimized communication performance. Also, parallel algorithms based on broadcasting data result simpler than those focusing on point-to-point messages, where a neighborhood for each processor has to be defined [15].
- Unidimensional data distribution, in order to: a) take advantage of the logical bus defined by the standard Ethernet, for which optimized performance can be obtained, and b) avoid the bidimensional NP-complete processing workload balancing problem.

2.1 Matrix Multiplication

As most of the parallel numerical computing algorithms, the simple parallel multiplication algorithm proposed follows the SPMD (Single Program-Multiple Data) processing model, which can be stated in terms of an initial data distribution plus a common program with explicit sections devoted to data processing and data exchange (communication) between processes. Data distribution takes into account the possibly different speed of each computer in the cluster to maintain a balanced workload. For the P computers in the cluster, ws_0, ..., ws_{P-1}, it is defined its normalized relative computing power pw_i such that $pw_0 + ... + pw_{P-1} = 1$. The computers Mflop/s (millions of floating point operations per second) to solve a single matrix multiplication can be used to define the pw_i.

If square matrices of order n are involved in the matrix multiplication C = A×B, data distribution to computers is defined in terms of row blocks for matrices A and C and column blocks for matrix B:

- ws_i contains $r\mathrm{A}_i = \lfloor n \times pw_i \rfloor$ rows of matrix A,
- ws_i contains $r\mathrm{C}_i = \lfloor n \times pw_i \rfloor$ rows of matrix C ($r\mathrm{C}_i = r\mathrm{A}_i$), and
- ws_i contains $c\mathrm{B}_i = \lfloor n/P \rfloor$ columns of matrix B.

where $\lfloor x \rfloor$ denotes the greatest integer such that $\lfloor x \rfloor \leq x$.

Thus, the number of rows of matrix A (and C) assigned to each ws_i, ($r\mathrm{A}_i$) is proportional to the computer relative processing power. This data distribution is not uniform when the machines are heterogeneous. According to the previous definitions, it is possible that $dr = r\mathrm{A}_0 + ... + r\mathrm{A}_{P-1} < n$. The remaining rows can be uniformly distributed among computers, ws_0, ..., $ws_{(n-dr-1)}$, one row for each computer. Given that the usual case is $P << n$, this reassignment of rows can be considered irrelevant from the point of view of proportional (according to the machines relative processing power) data distribution. The same kind of reassignment can be accomplished with $dc = c\mathrm{B}_0 + ... + c\mathrm{B}_{P-1}$ columns of matrix B.

Fig. 1 shows schematically the data distribution defined above in computer ws_i, which contains:

IEEE COMPUTER SOCIETY

- $A^{(i)}$, the local block of matrix A, $rA_i \times n$ elements,
- $B^{(i)}$, the local block of matrix B, $n \times cB_i$ elements, and
- $C^{(i)}$, the local block of matrix C, $rA_i \times n$ elements ($rC_i = rA_i$).

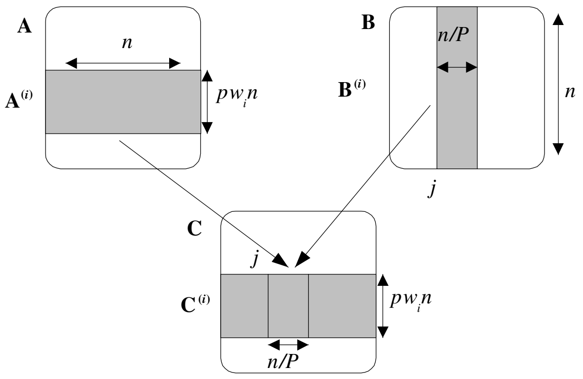
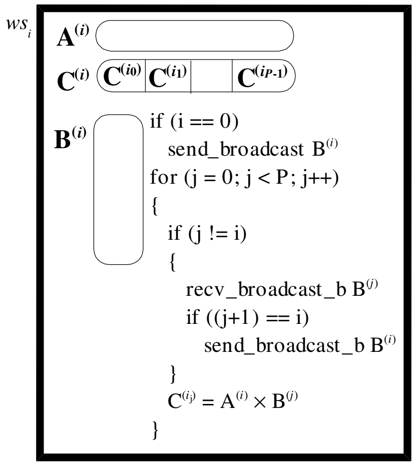


Figure 1: Data Distribution for Matrix Multiplication.

Also, Fig. 1 shows that ws_i has local data only for a fraction of the local submatrix $C^{(i)}$, $C^{(ii)}$. The complete $C^{(i)}$ will be calculated as a sequence of matrix multiplications $C^{(ij)}$ between $A^{(i)}$ and $B^{(j)}$ blocks with $j = 0, ..., P-1$. There are several ways of arranging communication and computing steps in order to have the complete submatrix $C^{(i)}$ calculated on each ws_i. Fig. 2 shows the pseudocode of the local processing in ws_i arranged to take advantage of overlapped communication in the computers where it is available.



```
if (i == 0)
  send_broadcast $B^{(i)}$
for (j = 0; j < P; j++)
{
  if (j != i)
  {
    recv_broadcast_b $B^{(j)}$
    if ((j+1) == i)
      send_broadcast_b $B^{(i)}$
  }
  $C^{(ij)} = A^{(i)} \times B^{(j)}$
}
```

Figure 2: Simple Parallel Matrix Multiplication Algorithm: Local Computing in ws_i.

The operations (in Fig. 2) send_broadcast_b and recv_broadcast_b are used to send and receive respectively broadcast data in "background", i.e. overlapped with other processing in the computer. While libraries such as PVM, MPICH and LAM/MPI can be used with this non-blocking (or locally blocking) feature, overlapped (in *background*) communication with computing depends on many factors such as the NIC (Network Interface Card) hardware. Heterogeneous machines in a LAN do not necessarily have this facility though it will be used where available.

2.2 LU Matrix Decomposition

The simple parallel algorithm proposed and tested for LU matrix decomposition assumes homogeneous computers, at least from the point of view of relative computing power. The arrangement for balanced workload on heterogeneous computers is relatively simple, but it has not been fully tested yet. Details of the simple parallel LU factorization algorithm are given for data distribution and computers local processing-communication steps. Data distribution for LU decomposition is made taking into account the evolution in data processing for Gaussian elimination. Fig. 3 shows the data distribution of a matrix A among four computers, ws_0,..., ws_3, often referred to as row block cyclic partitioning [12], where the block size is usually 32 or 64 [2].

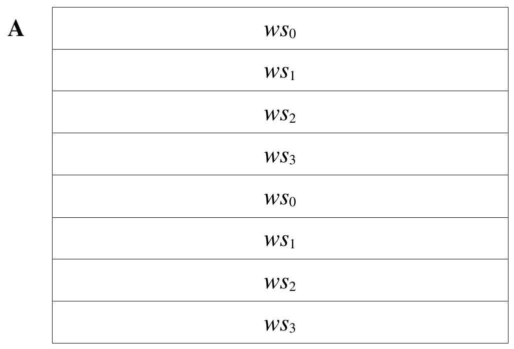


Figure 3: Data Distribution for LU Factorization.

The pseudocode of the local processing in ws_i to obtain matrices L and U from matrix A is shown in Fig. 4.

```
ws_i
for (j = 0; j < blq#; j++)
{
  if (i == (j mod P))
  {
    LU on block j
    Broadcast_send
  }
  else
    Broadcast_receive
  Apply pivots
  Update L
  Update trailing matrix
}
```

Figure 4: Simple Parallel LU Factorization Algorithm: Local Computing in ws_i.

The simple parallel LU factorization algorithm is directly based on the classical LU block algorithm [7], where:

- Data blocks are numbered from 0 to *blq*#-1.
- LU factorization is made on a complete block, and partial pivoting is applied in order to maintain numerical stability
- On each communication step (Broadcast_send and Broadcast_receive), the complete factored block along with its corresponding pivots are communicated.

3. Initial Experimental Evaluation

Several installed local area networks were used to evaluate the algorithms by experimentation. Two of the clusters (LIDI-Duron, and LIDI-PIII) are homogenous with eight identical computers each, whose characteristics are summarized in Table 1 and 2 respectively; both of them are interconnected by 100 Mb/s switched Ethernet.

Name	CPU	Clock	Mem	Mflop/s
lidipar{70..77}	AMD Duron	850 MHz	256 Mb	1200

Table 1: LIDI-Duron Cluster.

Name	CPU	Clock	Mem	Mflop/s
lidipar{14, 13, 12, 9, 8, 7, 6, 5}	Pentium III	700 MHz	64 Mb	579

Table 2: LIDI-PIII Cluster.

Computers on the third cluster (LQT) are heterogeneous, and their details are shown in Table 3; they are interconnected by 10 Mb/s Ethernet with a single hub. Mflop/s is obtained with fully optimized sequential code (used also for local computing in the parallel algorithms).

Name	CPU	Clock	Mem	Mflop/s
lqt_07	Pentium III	1 GHz	512 Mb	625
lqt_06	Pentium III	1 GHz	512 Mb	625
lqt_02	Celeron	700 MHz	512 Mb	479
lqt_01	Pentium III	550 MHz	512 Mb	466
lqt_03	Pentium II	400 MHz	512 Mb	338
lqt_04	Pentium II	400 MHz	512 Mb	338

Table 3: LQT Cluster

3.1 Matrix Multiplication and LU Factorization

The matrix multiplication algorithm was first implemented using PVM library calls for every data communication. Performance results in terms of speedup values are shown in Fig. 5 for a matrix multiplication in the LQT cluster with matrices of order 9000 elements in single precision floating point numbers. The reference value of the sequential algorithm was taken in the fastest computer (lqt_07), as expected in heterogeneous networks [16]. Computers are included (indicated by a "+" prefixed to the name in the x axis of the graph) to the "parallel virtual machine" according to its relative processing power following the usual better-to-worse approach. The speedup values shown as "Opt" are the optimal expected for each set of machines. The obtained performance is clearly unacceptable.

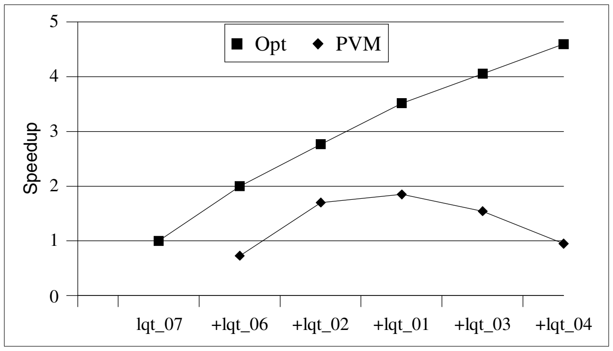


Figure 5: Matrix Multiplication Performance with PVM, LQT Cluster.

Experiments were carried out to take local execution times for computing as well as communication steps. As an example, local execution times on each computer taken when the six machines are used, corresponding to the last speedup value shown in Fig. 5:

1. Most of the running time each computer is waiting for (*executing*) a message, the average time used for communications is approximately 1700s.
2. The average computing time is approximately 630s.

The same kind of behavior in performance (local running times) is found for every combination of computers and algorithms. Evidently, the performance loss is due to communications. As the PVM broadcast is the only one communication routine used by the algorithm, the PVM broadcast is the origin of the performance penalty imposed to the algorithm. The same communication performance penalty was found in every computing platform (LQT cluster shown in Fig. 5, LIDI-PIII and LIDI-Duron), so the diagnosis -at least with PVM- is confirmed.

The simple parallel LU factorization was implemented using the PVM broadcast, Fig. 6 shows performance values on the LIDI-PIII network depending on the number of computers used for a matrix of order 3500 elements, and a row blocking size of 64 rows. According to Fig. 6, the best obtained performance (speedup) is about 2 when 3 and 4 computers are used in parallel. Performance degradation begins with 5 machines and it is worse when more computers are used. The same kind of local timings were taken as for the matrix multiplication, and the resulting values were identical: communication (broadcast) take much more time than expected on an Ethernet network.

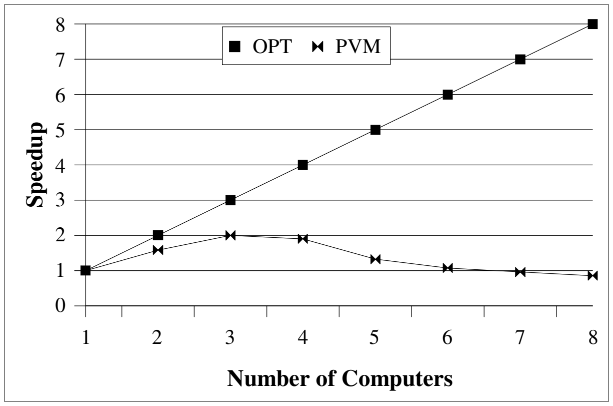


Figure 7: LU Factorization Performance with PVM, LIDI-PIII Cluster.

As with the matrix multiplication algorithm, the same communication problem was found in every computing platform (LIDI-PIII cluster shown in Fig. 7, LIDI-Duron, and LQT), so the diagnosis -at least with PVM- is confirmed.

It is worth noting that the main performance penalty is not due to message startup cost (latency) but on the low MB/s (throughput) obtained with PVM. Taking into account the number of computers and matrix sizes, the algorithms impose a very little number of messages of length in $O(10^5)$ and $O(10^6)$ bytes. In this sense, the algorithms tend to have coarse granularity and, thus, the communication performance depends heavily on data bandwidth effectively obtained from the interconnection network.

4. UDP Broadcasts

Although PVM, LAM/MPI, and MPICH are the most widely used libraries for parallel processing in computer clusters, they do not seem to be specifically designed and/or implemented to take advantage of Ethernet broadcast to optimize communication performance. There are some issues concerning broadcast performance in message passing libraries:

- Point-to-point communication routines are usually optimized in terms of protocol handling and data transmission.
- There are a large number of communication routines (more than 100 on the MPI standard) and, thus, it becomes impractical to optimize all of them, including broadcasts.

From another point of view (the one taken in this paper), having defined the parallel algorithms in terms of broadcast communication and given that broadcast could obtain optimized performance in Ethernet based clusters, it is likely to obtain acceptable parallel computing performance if broadcast messages are implemented taking advantage of Ethernet characteristics (basically, the standard Ethernet logical bus).

A broadcast message was implemented directly on top of the UDP protocol, which makes use of the Ethernet broadcasting facility on a wide range of operating systems and computer hardware (it was tested on Linux-PCs, BSD-Sun, Solaris-Sun, Solaris-PCs, and AIX-IBM). Some consequences of this approach are:

- Portability, since the UDP protocol is available on almost every computer.
- User level communication routine (it is not necessary to modify the operating system kernel or to impose any special or root level priority).
- If Ethernet is not found at the hardware level for communications, the UDP protocol tends to optimize the hardware capabilities. On ATM networks for example, it is likely that the UDP broadcasting facility will be better than every user-designed broadcast message. One of the main reasons for this assumption is that the UDP protocol is specifically oriented to obtain the best available performance.
- Optimized performance and scalability. Performance is optimized because UDP itself and its implementation is explicitly focused on communication performance. Scalability is optimized because the UDP protocol implementations exploit directly the Ethernet broadcast facility, thus communication time tends to be constant and independent from the number of processes involved in the communication routine.

The TCP protocol is used for control messages of the implementation, and the number of control messages is minimized to avoid many point-to-point data exchange for a single broadcast message. Control messages are used for two main reasons concerning the UDP protocol, since in the UDP protocol:

- It is not guaranteed to avoid losing data, i.e. a (local) successful sent data do not necessarily reach its destination
- It is not guaranteed data ordering, i.e. if data d_a is sent before data d_b, it is not guaranteed d_a will reach destination before d_b. Relationship between control data and user data is about 1/50000, i.e. approximately one *control* byte is sent every 50000 bytes of user data (pertaining to a broadcast message).

The layers of the resulting broadcast message are shown in Fig. 8, where UDP and TCP layers are relatively simple, just a few calls to the operating system socket interface. The Broadcast API (Application Programmer Interface) hides every implementation detail and follows the common syntax and semantics of PVM and MPI.

The broadcast message implemented is maintained independent from PVM and MPI implementations to explicitly set aside this operation focused on optimization over Ethernet broadcast from those provided by the libraries.

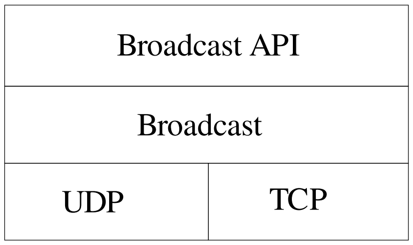


Figure 8: Broadcast Layers.

From the point of view of data integrity, computers in clusters have a highly uniform data representation, where:

- In general, PCs as well as workstations use IEEE floating point standards.
- PCs have exactly the same data representation regardless of microprocessors used.
- The only difference found in data representation when using PCs and workstations in a cluster is usually about byte ordering (little-big endian), which is easily solved at each side, according to sender-receiver computer.

5. Full Experimental Evaluation

New experiments were carried out to evaluate this new communication (broadcast) routine and the algorithms proposed. The matrix multiplication results are shown in heterogeneous as well as homogeneous clusters. Later, the LU factorization performance values are given according to those obtained in a homogenous cluster. Matrix multiplication as well as LU factorization algorithms were tested on each cluster described in Section 3. The most representative set of results combining clusters and algorithms were selected to show the effectiveness of the algorithms along with the broadcast message implemented taking advantage of the Ethernet broadcast facility.

5.1 Matrix Multiplication

Matrix multiplication is implemented now replacing PVM broadcasts by the proposed UDP based broadcast message. Fig. 9 shows the performance results obtained in the LQT network including the UDP based broadcasts (which appears as UDP), for different numbers of computers. From Fig. 9, it is possible to note that

- performance is greatly improved when the broadcast messages take advantage of the underlying (Ethernet) network, thus avoiding the performance penalties imposed by the PVM broadcast implementation.
- Even on a very low performance interconnection network, such as the Ethernet 10 Mb/s (which is usually discarded for parallel computing [14]), the performance values are very close to the optimal ones, as shown in Fig. 9.
- Performance grows when the number of computers grows, thus showing the good scalability of the algorithm even using high performance computers and a low performance interconnection network.

The same kind of optimized (UDP compared with PVM) results is obtained in the three clusters (LQT cluster shown in Fig. 9, and LIDI-PIII and LIDI-Duron clusters).

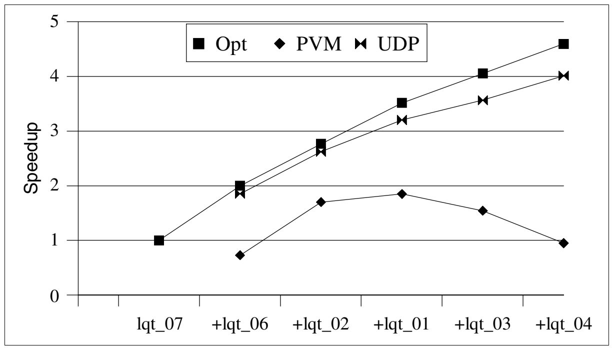


Figure 9: Matrix Multiplication Performance with UDP, LQT Cluster.

On the homogeneous clusters, it is possible to compare the proposed matrix multiplication algorithm with the one provided by ScaLAPACK [4] (more specifically, PBLAS), which is assumed to be a high quality parallel code on switched networks, given that:

- It has been designed to take advantage of simultaneous point-to-point messages, also taking advantage of optimized point-to-point performance of the message passing libraries.
- The processing block size as well as the bidimensional (grid) interconnection of computers can be tuned for optimized parallel performance.

ScaLAPACK is oriented *a priori* to homogenous parallel computers, thus preventing its use in clusters like LQT, since computing power of processors are among 338 Mflop/s and 625 Mflop/s. Fig. 10 shows the results obtained in the eight computers of the LIDI-Duron cluster (with matrices of order 10000 elements), where it is possible to compare performance of the simple parallel algorithm proposed (which appears as UDP) with the one provided by ScaLAPACK with different processor grids (i.e. processors row and processors columns) and square block sizes sorted according to the performance obtained.

Each bar of Fig. 10 is labeled with values for the three parameters (processors row, processors column, and block size), except the last one, which shows the performance obtained with the simple matrix multiplication algorithm. Different performance values show the ScaLAPACK dependence on blocking size and processor grid settings. On the other hand, the simple matrix multiplication algorithm does not have such a dependence and also obtains better performance (given that the broadcast message takes advantage of the Ethernet broadcast facility). The best ScaLAPACK performance value was obtained for a block size of 64 elements, 4 processors per row of processors, and 2 processors per column of processors in the processors grid. The optimum performance value is 8, the best ScaLAPACK value is about 5.5 and the one obtained with the simple parallel matrix multiplication algorithm is about 7, which is shown as UDP in Fig. 10. Results obtained in the LIDI-PIII clusters are similar.

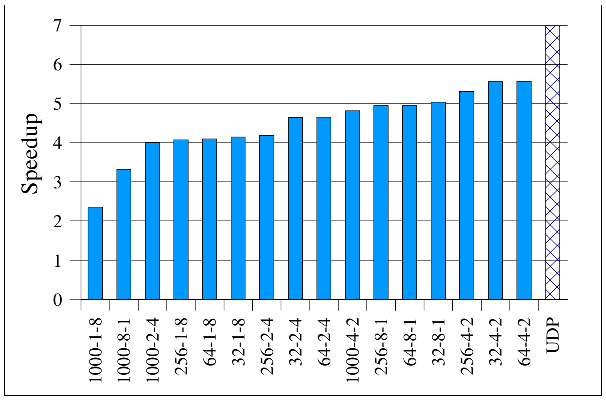


Figure 10: LU Factorization Performance with UDP, LIDI-Duron Cluster.

5.2 LU Factorization

The simple parallel LU factorization was implemented using the PVM broadcast and the UDP based broadcast. Fig. 11 shows performance values for each implementation on the LIDI-PIII network depending on the number of computers used for a matrix of order 3500 elements, and a row blocking size of 64 rows.

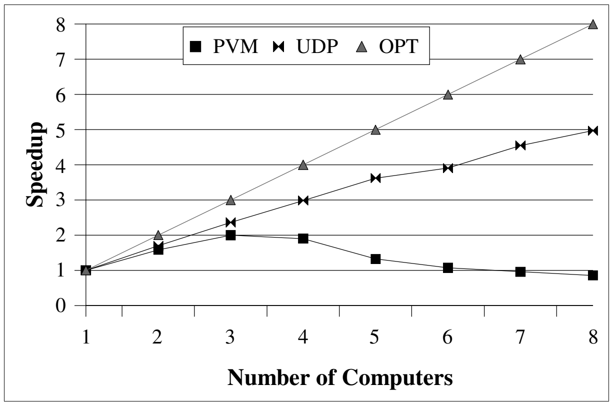


Figure 11: LU Factorization Performance, LIDI-PIII Cluster.

From Fig. 11 it should be noted that UDP based values are better than those obtained with PVM. However, performance values are not as close to the optimal ones as it could be expected taking into account the previous matrix multiplication performance (speedup, more specifically) values. It should be also noted that:

- LIDI-PIII computers have much less memory than computers in LIDI-Duron and LQT clusters. Reducing the matrix size also reduces the room for taking advantage of the $O(n^3)$ processing requirements over $O(n^2)$ memory requirements (thus resulting in a finer granularity).
- LU factorization (as most of the matrix factorizations) impose higher data dependency than matrix multiplication, thus preventing large time intervals of asynchronous computing.
- It was not tested any possible overlapping of computing-communication.

As explained in section 2.2, the simple parallel LU factorization algorithm assumes homogenous computers, so it should be adapted to heterogeneous clusters before its utilization in the LQT cluster. Some possibilities are being tested, and the final results (and algorithm to be used) are not obtained yet. Results obtained in the LIDI-Duron cluster are similar in tendencies to those shown in Fig. 11, but since LIDI-Duron computers have more memory, the UDP implementation have better values (closer to the optimal ones).

6. Conclusions and Further Work

Linear algebra operations defined in terms of the LAPACK and ScaLAPACK libraries are very good candidates for optimization as shown in the literature. Specifically, BLAS level 3 operations are provided usually optimized at a hardcoded level by each microprocessor designer. In this sense, large number of applications and the reduced number of LAPACK and BLAS level 3 operations provide good reasons for a one-by-one optimization. This approach is taken specifically for parallel computing algorithms proposed for Ethernet based clusters of computers.

Two parallel simple algorithms and one optimized broadcast message are presented specifically designed for cluster computing. The matrix multiplication algorithm can be used on heterogeneous as well as on homogeneous clusters, where it is shown by experimentation that obtains better performance that the high-quality parallel matrix multiplication provided by ScaLAPACK. Also, the broadcast message provided by PVM is shown highly penalized in clusters, and implementations of MPI do not provide *a priori* any confidence about performance. The LU factorization algorithm presented is considered preliminary and subject to optimizations, such as the arrangement of communication overlapped with computing.

The main advantages of the parallelization methodology as well as the specific parallel algorithms proposed for matrix multiplication and LU factorization along with the proposed broadcast can be summarized as:

- Parallel performance depends just on local computing and the broadcast message. Local computing is made with optimized BLAS libraries and it has been shown that the broadcast message can be optimized for Ethernet based clusters. Thus, parallel performance is completely optimized.
- The unbalanced workload produced by cluster heterogeneity can be solved easily by using unidimensional data distributions, as it was shown by the matrix multiplication algorithm.
- Simple algorithms can be considered optimized for clusters, since simple algorithms are able to obtain better performance than, for example, algorithms provided by the ScaLAPACK library. Also, simple algorithms do not depend on parameters such as the configuration of processors grid arrangement.

For the currently installed clusters as well as local area networks which can be used for parallel computing, the algorithms as well as the broadcast message are scalable and tend to optimize the available resources for linear algebra parallel computing, at least for the two algorithms presented. Simple algorithms should be proposed for other linear algebra operations and/or methods, such as those included in LAPACK and ScaLAPACK libraries.

Acknowledgments

Prof. Alicia Jubert and Prof. Reinaldo Pis Diez, from the Laboratorio de Química Teórica, CEQUINOR, Departamento de Química, Facultad de Ciencias Exactas, Universidad Nacional de La Plata, provided the LQT cluster. Prof. Armando De Giusti, from the Laboratorio de Investigación y Desarrollo en Informática, Facultad de Informática, Universidad Nacional de La Plata provided the LIDI clusters as well as numerous suggestions. Prof. Dolores Rexachs has also offered continuos support in many ways to this work.

References

[1] Anderson E., Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, LAPACK: A Portable Linear Algebra Library for High-Performance Computers, Proceedings of Supercomputing '90, pages 1-10, IEEE Press, 1990.

[2] Beaumont O., V. Boudet, A. Petitet, F. Rastello, Y. Robert, “A Proposal for a Heterogeneous Cluster ScaLAPACK (Dense Linear Solvers)”, IEEE Trans. on Computers 50, 10, pp.1052-1070, Oct. 2001.

[3] Burns G., R. Daoud, and J. Vaigl, LAM: An Open Cluster Environment for MPI. Ohio Supercomputer Center, May 1994. LAM/MPI is available at University of Notre Dame (http://www.mpi.nd.edu/lam) - 1998-2001.

[4] Choi J., J. Dongarra, R. Pozo, D. Walker, “ScaLAPACK: A Scalable Linear Algebra Library for Distributed Memory Concurrent Computers”, Proc. 4th Symposium on the Frontiers of Massively Parallel Computation, Ieee Computer Society Press, pp. 120-127, 1992.

[5] Dongarra J., J. Du Croz, S. Hammarling, R. Hanson, “An extended Set of Fortran Basic Linear Subroutines”, ACM Trans. Math. Soft., 14 (1), pp. 1-17, 1988.

[6] Dongarra J., A. Geist, R. Manchek, V. Sunderam, Integrated pvm framework supports heterogeneous network computing, Computers in Physics, (7)2, pp. 166-175, April 1993.

[7] 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.

[8] Gropp W., E. Lusk, N. Doss, A. Skjellum, “A high-performance, portable implementation of the MPI message passing interface standard”, Parallel Computing, Vol. 22, No.6, pp 789-828, Sep, 1996.

[9] Hoare C., Communicating Sequential Processes, Englewood Cliffs, Prentice-Hall, 1986.

[10] Lawson C., R. Hanson, D. Kincaid, F. Krogh, “Basic Linear Algebra Subprograms for Fortran Usage”, ACM Transactions on Mathematical Software 5, pp. 308-323, 1979.

[11] 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.

[12] Kumar V, Grama A, Gupta A, Karypis G, Introduction to Parallel Computing. Design and Analysis of Algorithms, The Benjamin/Cummings Publishing Company, Inc., 1994.

[13] Message Passing Interface Forum, MPI: A Message Passing Interface standard, International Journal of Supercomputer Applications, Volume 8 (3/4), 1994.

[14] Nagendra B., L. Rzymianowicz, “High Speed Networks”, in in R. Buyya Ed., High Performance Cluster Computing: Architectures and Systems, Vol. 1, Prentice-Hall, Upper Saddle River, NJ, USA, pp. 204-245, 1999.

[15] Wilkinson B., Allen M., Parallel Programming: Techniques and Applications Using Networking Workstations, Prentice-Hall, Inc., 1999.

[16] Zhang X., Y. Yan, “Modeling and characterizing parallel computing performance on heterogeneous NOW”, Proceedings of the Seventh IEEE Symposium on Parallel and Distributed Processing, *(SPDP'95)*, IEEE Computer Society Press, San Antonio, Texas, October 1995, pp. 25-34.

IEEE COMPUTER SOCIETY