Chapter 5: Comparison with ScaLAPACK

This chapter presents two important aspects as regards the validity and use of the contributions of this thesis: 1) Application of parallelization principles in homogeneous environments dedicated to parallel computing for two specific cases: matrix multiplication and matrix LU factorization; and 2) Comparison of the results obtained by experimentation in terms of the ScaLAPACK library performance. This library is specifically dedicated to homogeneous parallel computing platforms with distributed memory, and is generally accepted as that implementing the best existing parallel algorithms in terms of accessibility and performance optimization.

The use of the ScaLAPACK library is, until now, oriented only to homogeneous hardware and, thus, the homogeneous network which we have worked with is initially taken into account. However, for this comparison, we have had the possibility of counting with a second homogeneous network, which is also described (at least, the necessary) in this chapter. In a certain way, several characteristics of the experimentation change in this chapter for two reasons: 1) The experimentation is classic and simple as regards tests. For instance, there is no analysis of the problem sizes off the limits of each machine's memory since what we are aiming at is the direct comparison of the algorithms; 2) We only take into account homogeneous computer networks, since otherwise comparison with the algorithms implemented in ScaLAPACK will not be possible or would be "biased".

This chapter also adds the problem of matrix LU factorization, to which the same parallelization concepts as those of matrix multiplication are applied. In this case, the objective has not been the exhaustive examination of algorithms, implementations, etc. - as in the case of matrix multiplication-, but the direct application of parallelization principles to this specific problem. For this reason, the description of the problem in itself and of parallel algorithms will not be exhaustive.

5.1 ScaLAPACK General Characteristics

It is really interesting to highlight that algorithm analysis and algorithm proposals for solving parallel linear algebra problems are closely related to the traditional parallel machines (such as those belonging to the "Ad Hoc" and "Commercial" parallel computer types of Chapter 1, Figure 1.2). In consequence, the analysis carried out for the case of matrix multiplication is applicable to the rest of the operations included in ScaLAPACK and/or used to solve them.

From the point of view of ScaLAPACK implementation, we can identify several software "layers" [21], which are shown in Figure 5.1 from the point of view of the software to be used in each computer.

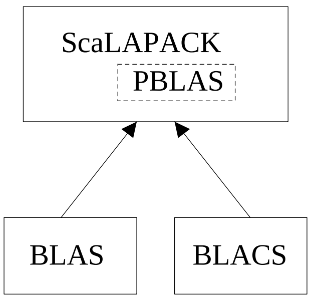


Figure 5.1: A Layered Vision of ScaLAPACK.

As an *integrating* part of ScaLAPACK, we make special emphasis on the basic operations (like in the LAPACK context), which are called PBLAS (Parallel Basic Linear Algebra Subroutines). All what is related to local computing in terms of calculations or operations of matrix computing remains directly related to BLAS (Basic Linear Algebra Subroutines), and, in this way, we take advantage of all the efforts and developments of several years of research. On the other hand, we should take into account the need of communication among parallel computer processes, in general, and distributed memory parallel computers, in particular, and we thus add BLACS (Basic Linear Algebra Communication Subroutines).

The addition of BLACS is interesting from two points of view:

- As previously mentioned, we are aware of the need of communicating among processes in a distributed memory environment. In this sense, we tend to discard the possibility of shared memory (let be it physically distributed or not), and we tend to adopt the message-passing programming model of distributed parallel architectures. This decision is, in turn, directly related to the obtaining the optimized performance in terms of computing and communications.
- None of the available message passing libraries is directly used: nor those of free use (PVM, and MPI implementations) or those provided by commercial companies of parallel computers which are specifically optimized for their interconnection networks (IBM, for instance, for their SP machines). In fact, there exist free-use BLACS implementations which make direct use of PVM or MPICH (one of MPI implementations).

Parallel algorithms implemented in ScaLAPACK are strongly influenced by those proposed in the area of multicomputers with processor bi-dimensional interconnection or organization or with even more complex interconnection networks, such as those of hypercube topology [2] [3] [21]. This decision is oriented to using and taking advantage of all the investigations and results obtained to the present. However, this type of algorithms tends to count with a high performance penalization in the cluster context, since with clusters based on Ethernet interconnection networks and with general purpose operating systems (Linux, AIX, IRIX, Solaris, etc.), the cost-communication relations is several orders of magnitude worse (for parallel application performance) than in traditional multicomputers. The most important reason for this to happen is that neither Ethernet networks nor the traditional operating systems are thought for parallel computing, alike multicomputers, which are built for parallel computing.

Consequently, it is easy to understand why clusters, over which libraries like ScaLAPACK are used and where parallel computing tasks are generally solved, are restricted to the completely "switched" wiring. With this type of wiring it is possible to carry out multiple point-to-point communications, and this makes the hardware similar to that of multicomputers. However, we shall see that the cost of completely "switched" networks increases much more than linearly in function of the quantity of interconnected computers.

5.2 LU Factorization Parallelization

The matrix factorization method called LU is widely known and accepted, both in terms of its use and its numerical stability properties (specifically, when pivoting is, at least, partially used), as computing and storage requirements. The *initial* definition of the method is oriented to the solution of equation systems, and is directly based on what is known as Gaussian elimination [27]. Initially, the block LU factorization sequential algorithm is described, and then the proposed parallel algorithm for computer clusters is presented, following the same principles used in matrix multiplications.

5.2.1 Block LU Factorization Sequential Algorithm

Given a square matrix A of order *n*, two matrices usually called L and U, also square and of order *n*, are searched for such that

$$A = L\times U \tag{5.1}$$

where L is lower triangular and U is upper triangular.

If A is not singular, it can be proved that L and U exist [59]. The LU factorization method does nothing but successively apply Gaussian elimination steps so that the elements of L and U matrices are computed iteratively [84]. Generally, and with the objective of stabilizing the calculations from the numerical point of view (to basically delimit the error), the partial pivoting technique is incorporated within the LU method.

From the point of view of memory requirements, no requirement other than the storage of matrix A being factorized is added, and thus is kept in the $O(n^2)$. From the point of view of computing requirements, the number of floating point operations needed for LU computing is $O(n^3)$. In this way, we get to the same basic relation as that obtained for level 3 BLAS operations, i.e. the number of floating point operations is $O(n^3)$ with $O(n^2)$ data being processed.

With the objective of taking advantage of the underlying computing architecture in most of the computers, and specifically of the memory hierarchy with the inclusion of at least a cache memory level, most of the linear algebra operations have been defined in terms of data blocks (or submatrices). Specifically within the context of the LU method, a partition of matrix A is determined taking as basis a block or submatrix of A, A_{00}, of $b \times b$ data, such as Figure 5.2 shows.

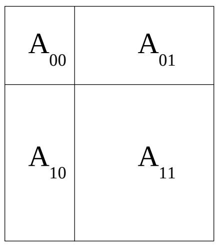


Figure 5.2: Division of a Matrix in Blocks.

The LU factorization of A is sought, which expressed in function of the previous block division will be such that

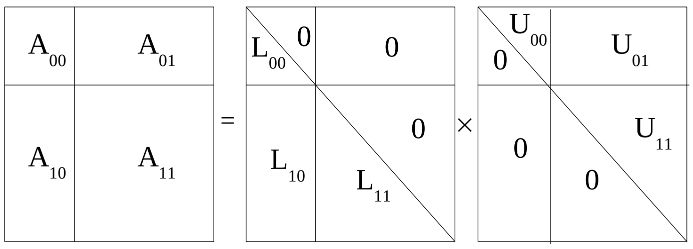


Figure 5.3: LU Factorization in Terms of Blocks.

where the following equations should be fulfilled, taking into account the submatrices of A, A_{00}, A_{01}, A_{10}, and A_{11}, the submatrices of L different from zero, L_{00}, L_{10} y L_{11}, and the submatrices of U also different from zero U_{00}, U_{01} y U_{11}.

$$A_{00} = L_{00} U_{00} \tag{5.2}$$
$$A_{01} = L_{00} U_{01} \tag{5.3}$$
$$A_{10} = L_{10} U_{00} \tag{5.4}$$
$$A_{11} = L_{10} U_{01} + L_{11} U_{11} \tag{5.5}$$

and matrices L_{ij} are lower triangulars and U_{kl} ($0 \leq i, j, k, l \leq 1$) are upper triangular.

If LU factorization is directly (in the *clasical* way) applied to the block A_{00} of A, the triangular matrices L_{00} and U_{00} are obtained such that Eq. (5.2) is directly verified (due to the application of LU factorization method). Using Equation (5.3), $L_{00}\times U_{01} = A_{01}$, and as L_{00} is lower triangular, thus the triangular equation system solution method with multiple results (multiple right hand sides) can be directly applied in order to obtain each of the columns of U_{01}. Likewise, or similarly, Eq. (5.4) is used to obtain L_{10}, since $L_{10}\times U_{00} = A_{10}$ and, in this case, U_{00} is upper triangular and the triangular equation system solution method with multiple results (multiple right hand sides) can be also directly used.

It would only remain the computing of L_{11} and U_{11} in order to obtain the whole factorization of matrix A. In this case, by Eq. (5.5), $L_{11}\times U_{11} = A_{11} - L_{10}\times U_{01}$, i.e. finding matrices L_{11} and U_{11} implies applying the same LU method by blocks to the (sub)matrix resulting from $A_{11} - L_{10}\times U_{01}$. What has remained explicitly unsaid is the necessary processing related to the partial pivoting, which basically implies selecting the pivot and exchanging the corresponding rows or columns.

With the previously explained method by blocks, all the operations can be defined in terms of submatrices of the original matrix A, and, also, two processing characteristics that can be successfully used in terms of code optimization are defined:

1. Most of the floating point operations are run to solve matrix $A_{11} - L_{10}\times U_{01}$ update, which is basically a matrix multiplication.
2. All the LU factorization method can be solved using two BLAS-defined routines, which are that of triangular equation system resolution and matrix multiplication (_trsm and _gemm, respectively in terms of BLAS C interface) and that of LAPACK (_getrf, in terms of LAPACK C interface).

5.2.2 LU Factorization Parallel Algorithm *for* Multicomputers

The previously described method for LU factorization is almost directly implemented in sequential computers, though it has some performance drawbacks in distributed memory parallel computers. Data distribution has a decisive relevance in terms of performance from two points of view: a) load balance, and b) scalability.

It is relatively easy to identify that, as the algorithm iterations advance, the left upper corner of the matrix is computed, and this computed part is each time (each iteration) greater. This directly implies that these data do not need to be updated and they not even take part in the following computations. Taking into account that data are distributed in different computers, all the computers which have these data assigned to will not use them again neither for updating them nor computing others in function of them. In order to avoid the load unbalance produced as the iterations advance, libraries such as ScaLAPACK and PLAPACK use the distribution called two-dimensional block cyclic decomposition, such is mentioned in [26] [21] [45] and detailed at the end of Chapter 3. The main ideas on which this data distribution is based are:

- Having much more data blocks than processors, and in this way, the distribution makes the same processor have blocks updated almost in every iteration. In consequence, all the processors tend to have blocks which are updated in every iteration and they all process in all the iterations.
- It is assumed that two-dimensional distributions are scalable enough at least for linear algebra applications.
- The experimental work is generally carried out in multicomputers or in homogeneous clusters with completely "switched" and high performance (in a way, *ad hoc* for parallel processing) interconnection networks.

In consequence, the parallel algorithm used for LU factorization is directly based on the description given for the LU computation in function of blocks, but with two-dimensional and block-cyclic distribution of the matrix data.

5.2.3 LU Factorization Parallel Algorithm *for* Clusters

The guidelines for matrix LU factorization parallelization are basically the same as those used for matrix multiplication:

- Message-passing programming model: processes locally computed and communicated through messages with the remaining.
- SPMD (Single Program, Multiple Data) computing model: a same program run by every computer using different data.
- One-dimensional data: the complete matrix is divided by rows or columns (not in both ways, all of which would lead to two-dimensional distributions).
- Communication among processes only for broadcast messages: most of the data transferences among processes (or all of them) are carried out via broadcast messages in order to use to the maximum the characteristics of the Ethernet networks in terms of transference data rates and scalability.

Data Distribution. Since all communications tend to be of broadcast type, *one-dimensional* data distributions are favored over the two-dimensional or those which take into account the interconnection in hypercubes, for instance. The communication (or data transfer) among computers that is prone to be used is the very definition of the Ethernet standard as regards the computers' logic interconnection with the only bus. In this sense, data distribution is one-dimensional, though computers interconnection is not that of a ring, which is considered as *typical* in the context of one-dimensional process organization [82].

Once data distribution is restricted to the one-dimensional ones, there are only two similar alternatives: by rows or by columns. Even though the initial proposal can be the division of the complete matrix in as many parts as available processors (computers), this implies a direct loss of processing load balance. If, for instance, there are four processors ws_0, ws_1, ws_2, and ws_3, and the matrix is divided in four row blocks, such as Figure 5.4 shows, when the rows assigned to processor ws_0 are all processed, processor ws_0 no longer has a processing task. This means that, from this moment on, all the matrix factorization task is carried out without the computing power of ws_0. Something similar happens when the rows assigned to ws_1 are then processed, from which all subsequent processing is carried out only in processors ws_2 and ws_3, and this clearly implies that the processing is not balanced among the four processors.

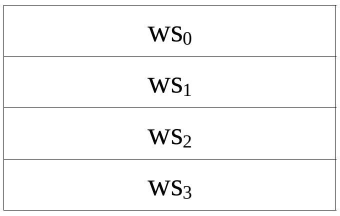


Figure 5.4: Partition and Designation of a Matrix by Row Blocks.

In this moment, the idea of processing by blocks is directly used, as well as the so-called block-cyclic distribution. A block size relatively small in relation to the total size of the matrix is established, and the distribution is carried out by blocks of this chosen size. In the case of one-dimensional distributions, this block size is the number of rows or columns distributed as a unit. If, for instance, there are eight blocks and four processors, the block-cyclic distribution is carried out as Figure 5.5 shows, where block *i* is assigned to processor *i* mod *P*, where *P* is the total number of computers.

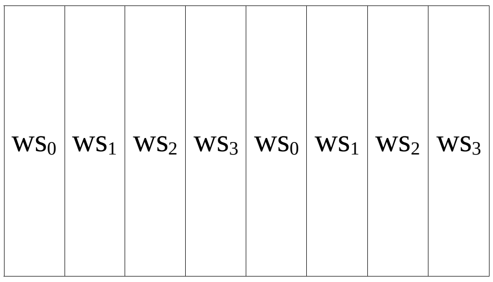


Figure 5.5: Block-Cyclic Column Distribution.

The greater is the number of blocks, the greater will also be the resulting load balance. Since, normally, the number of rows and columns of the matrices to be processed is much greater than the number of computers, the load balance implemented in this way does not present any inconveniences. On the one hand, data distribution is simple, and with very few parameters to be defined (only the block size) and, on the other, the load balance necessary to obtain acceptable performance in the LU factorization processing is obtained.

Processing. Like most of the numerical applications belonging to the area of linear algebra, the processing model is SPMD; all the computers in the cluster run the same program. An initial proposal, with computing and communication periods run sequentially in each of the computers of the cluster, is shown in Figure 5.6 in pseudo-code for machine ws_i with $0 \leq i \leq P-1$ [127]. As shown in the pseudo-code, all communications are of broadcast type and, thus, if this type of data transfer is optimized among processes of a parallel application (using the Ethernet network physical broadcast facility, for instance), the *complete* parallel processing is optimized almost directly for the LU factorization.

ws_i

```
for (j = 0; j < nblocks; j++)
{
  if (i == (j mod P))      /* Current block is local */
  {
    Factorize block j
    broadcast_send (factorized block j and pivots)
  }
  else
    broadcast_receive (factorized block j and pivots)
  Apply pivots                 /*                       */
  Update L                     /*  Update local blocks  */
  Update trailing matrix       /*                       */
}
```

Figure 5.6: Pseudo-code of a Parallel Algorithm for LU Factorization.

It should be noticed in the pseudo-code of Figure 5.6 that everything related to the handling of pivots is explicitly incorporated because now the matrix is distributed among processors and, thus, the interchanges produced by pivots should be explicitly distributed from the computer factorizing a block. On the other hand, assuming, for example, that the matrix is divided in row blocks, when a row block is factorized, the blocks which in Figure 5.2 appear as L_{00}, U_{00} and U_{01} are already computed, and consequently, what remains to be computed are the blocks corresponding to L_{10} (L in the Figure 5.6) and to $A_{11}-L_{10}\times U_{01}$ ("trailing matrix" in Figure 5.6).

As in the case of matrix multiplication, the pseudo-code of Figure 5.6 imposes a very well defined and strict sequence of steps of local computing and communications with the rest of the processes/processors. Also, as in the case of matrix multiplication, the computing can be organized so as to carry out communications overlapped with local computing (whenever possible) and, in this way, attempt to reduce the performance penalization imposed by communications in the clusters. This proposal is shown in Figure 5.7 in pseudo-code.

The algorithm of Figure 5.7 adds the idea of the "next block" to the classical algorithm of Figure 5.6. Since LU factorization of a block and its corresponding send_broadcast_b communication operation impose a waiting time to all the remaining processors, the next block is factorized and sent "in background" so that it is available in the next iteration in order to update the rest of the matrix.

```
                                                      ws_i
if (i == 0)
  Factorize and broadcast_send block 0
for (j = 0; j < nblocks; j++)
{
   if (i == (j mod P))  /* Current block is local */
      Update local blocks
   else if (i == ((j+1) mod P))  /* Next block is local */
   {
      broadcast_recv_b (factorized block j)
      Update and Factorize block j+1
      broadcast_send_b (factorized block j+1)
      Update local blocks (block j+1 already updated)
   }
   else /* ws_i does not hold block j nor block j+1 */
   {
      broadcast_recv_b (factorized block j)
      Update local blocks (block j+1 already updated)
   }
}
```

Figure 5.7: Parallel LU Factorization with Overlapped Computing and Communication.

5.3 Experimentation

Initially, we should define the parameters with which the experiments were carried out; more specifically, in the case of the ScaLAPACK it is necessary to define:

- Communication Supporting Library (PVM, MPI implementation, etc.).
- Processors grid (two-dimensional processors array).
- Block Size.

Since ScaLAPACK is used in homogeneous environments, it is no longer necessary to make reference to the relative computing power of processors for the load balance. In addition, the speedup can be used directly as the performance index in order to compare the parallel algorithms implemented in ScaLAPACK with others, such as those proposed in this chapter and in chapter 3.

In order to avoid confusions, and since the comparison made in this chapter tends to quantify the differences between the algorithms implemented in ScaLAPACK with those proposed in this chapter and in chapter 3, the algorithms used are those tending to use the overlapped communication and computing facility among computers. In this way, there is a single proposed algorithm for solving each of the tasks to be carried out in parallel: matrix multiplication and matrix LU factorization.

5.3.1 Set of Experiments

Hardware. The first homogeneous network that will be used is the only one that has been used under these conditions: the LIDI local network. However, there has been access to another local network that could be used and interconnected to the LIDI local network as well. This second network is made up of 8 PCs with Duron processors with higher computing and storing capacity than LIDI's PCs, with 10/100 MB/s Ethernet network interconnection cards. In addition, since we could use other two Ethernet switches (apart from that of the LIDI local network) of 8 ports 100 MB/s, experiments can be carried out with two more "parallel machines": one with the eight PCs with Duron processors completely interconnected by a switch, and the other with a combination of the two networks with three switches. Finally, in the last of these experimentation sections, the results obtained in the last of the homogeneous clusters will be shown, in which experiments more specifically oriented to comparing the performance of the algorithms proposed in this thesis and those implemented by ScaLAPACK were carried out.

The network with the eight PCs with Duron processors and a single switch (which will be called hereinafter LIDI-D), is a classical homogeneous cluster, like the very LIDI network, though it has an important difference with the LIDI network: the computing-communication ratio. Since computers of the LIDI-D network have a higher computing and storing power and the same interconnection network as those of the LIDI network, the difference in the computing-communication relation between them can be quantified by using directly the computing power differences among PCs. Table 5.1 briefly shows the characteristics of the PCs of the LIDI-D cluster, together with their computing capacity expressed in terms of Mflop/s.

Processors	Clock Freq.	Memory	Mflop/s
AMD Duron	850 MHz	256 MB	1200

Table 5.1: Characteristics of the Computers of the LIDI-D Cluster.

Since PCs of the LIDI-D have a performance of approximately of 1200 Mflop/s, slightly more than the double of LIDI network computers, it can be asserted that the computing-communication ratio of the LIDI-D network is approximately two times worse for parallel computing than that of the LIDI network. In other words, at the same time interval:
- LIDI-D computers can perform the double of the computations than those of the LIDI network..
- LIDI-D computers can transmit the same amount of data than those of the LIDI network..

The amount of RAM installed in the computers of the LIDI-D network is of 256 MB, with which bigger than those tested in the LIDI network can be tested.

In the case of the combination of both networks with three switches of eight ports, the situation is not so common for two reasons: a) there does not exist homogeneity in the sixteen PCs; b) there is not complete "switching" capacity, i.e. it is not possible to combine all the simultaneous communications of both PCs at the same time. Figure 5.8 shows how to interconnect the sixteen PCs ($PIII_0$, …, $PIII_7$ of the LIDI network, and D_0, …, D_7 of the

LIDI-D network) with three switches of eight ports, where we can clearly observe how the complete "switching" capacity is lost among the eighteen PCs. From now on, this local network with sixteen PCs will be called LIDI-16.

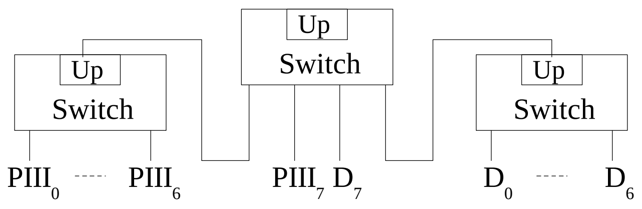


Figure 5.8: LIDI and LIDI-D Networks Interconnected with Three Switches.

In the LIDI-16 network, and from the performance point of view, it will be taken into account that all the computers have the computing power of the LIDI network's computers, i.e. they are considered as homogeneous. The ScaLAPACK performance and that of the algorithms proposed will be considered taking into account the power of the LIDI network computers. However, there does not exist such a simple solution for the case of communications, since ScaLAPACK tends to assume that the two-dimensional processors arrays behave as a static network with direct links, and this is not necessarily possible with the organization of interconnection with three switches shown by Figure 5.8. All the same, this will be employed at least as an idea of what may happen with a network of sixteen computers, trying to identify the ScaLAPACK and the proposed algorithms' scalability characteristics. Summing up, tests will be carried out in three networks of *homogeneous* PCs: LIDI, LIDI-D and LIDI-16, interconnected with 100 Mb/S Ethernet switches.

Basic Software. In all the cases of local computing performed in each PC, it is completely optimized. In the case of ScaLAPACK, there are several alternatives for the data communication routines (BLACS possible implementations): PVM and current MPI implementations. In order to avoid an *a priori* comparison of which the best of them is, PVM and MPICH have been used. The selected implementation of MPI is MPICH, since in the documentation and installation of ScaLAPACK it tends to use this library as a *reference* of MPI. In the case of the proposed algorithms for the multiplication and LU factorization of matrices, the broadcast message explicitly implemented for the utilization of Ethernet networks will be used. In consequence, there are two implementation alternatives of ScaLAPACK: with PVM and with MPICH, and only one base software alternative for the proposed algorithms.

Running Parameters. In the case of ScaLAPACK, the processor grids and the block sizes should be defined. In the case of the 8-processor networks, grids of 8x1, 1x8, 2x4 and 4x2 processors have been defined. In the case of the 16-processor network, grids of 8x2, 2x8 and 4x4 processors have been defined. As regards the block sizes, the idea is to experiment with small and large sizes of data blocks, plus some intermediate considered as "classic" within the context of ScaLAPACK. In consequence, the sizes with which the experimentation was carried out were (always square blocks, square submatrices of the matrices to be processed): 16, 32, 64, 126, 256, 512, and 1024 elements.

One last decision refers to the matrix sizes; in all of the cases the maximum sizes of matrices were used, which depend on the total quantity of the computer memory and on the very problem. In the case of matrix multiplication, three matrices have to be stored and, in the case of the LU factorization, only one. When the LIDI-16 network is used, it is considered that **all** the machines count with 64 MB of RAM. The specific sizes for each problem and each computer network are shown in Table 5.2.

Cluster	Matrix Multiplication	LU Matrix Factorization
LIDI	5000	9000
LIDI-D	10000	20000
LIDI-16	8000	13000

Table 5.2: Sizes of Problems in each Local Network.

Summary of the Experiments. In the case of ScaLAPACK, we have for each problem (multiplication and LU factorization of matrices) two sets of results in each PC network, depending on the communication library: PVM and MPICH. In the case of the algorithms proposed, there is only one possibility, since both the algorithm and the broadcast communication routine are unique. Table 5.3 summarizes the experiments carried out, where:

- ScaMM denotes the matrix multiplication implemented in ScaLAPACK, more specifically, in PBLAS
- PropMM is the algorithm proposed to multiply matrices with overlapped computing and communications.
- ScaLU is the algorithm implemented by ScaLAPACK to make the LU factorization in parallel.
- PropLU is the algorithm proposed to make the LU factorization of matrices in parallel.
- Bcast-UDP is the broadcast communication routine specifically optimized to take advantage of the broadcast characteristics of Ethernet networks.

Comm.	**ScaMM**	**PropMM**	**ScaLU**	**PropLU**
PVM	#blq, Grilla		#blq, Grilla	
MPICH	#blq, Grilla		#blq, Grilla	
Bcast-UDP		1		#blq

Table 5.3: Experiments with ScaLAPACK and with the Proposed Algorithms.

The empty entries in Table 5.3 correspond to meaningless experiments or those which could not be carried out:

- ScaLAPACK matrix multiplication and factorization cannot be immediately carried out with the routine implemented to take advantage of the Ethernet networks broadcast characteristics because ScaLAPACK needs the complete BLACS library, and not just a single broadcast routine among processes.
- It has already been proven in the previous chapter that the performance obtained with the matrix multiplication with the proposed algorithm and the PVM library is far from being acceptable. Thus, we have chosen to experiment with the proposed algorithms discarding the PVM library and all the MPI implementations (including MPICH) which,

a priori, do not assure the optimized implementation of broadcast messages.

The entries in Table 5.3 containing "#blq" indicate that the performance depends at least on the data block size we are working with. If, in addition, the entry has "Grid", it means that it is necessary to define a two-dimensional array of the processors used. There is only one entry in Table 5.3 which contains a "1" and this shows that there does not exist any parameter to be defined for this alternative. In other words, the algorithm proposed to multiply matrices does not depend on anything but on the computers to be used. In the next subsections, the results are shown in function of ScaLAPACK and compared directly to the results obtained by the proposed algorithms, since this entails the objective of the whole experimentation in this chapter.

5.3.2 Results: ScaLAPACK-PVM

ScaLAPACK installation *over* PVM is rather simple, and the different running (parametrical) alternatives can be tested. The axis **y** of the following figures shows the performance in terms of the Speedup obtained by each algorithm, and the axis **x** of the same figure shows the algorithms and parameters used for its running. The results obtained with ScaLAPACK are shown in the light bars, identifying each bar with the three parameters with which they were obtained: block size - quantity of grid processor rows, and quantity of grid processor columns. The results obtained with the algorithms proposed are shown in the dark bars and identified as "Prop".

Figure 5.9 shows the best five performance results with different combinations of parameters with ScaLAPACK-PVM, and the performance of the algorithm proposed in Chapter 3 (overlapped computing and communications) implemented with broadcast messages optimized in LIDI network. The best result of the performance obtained with ScaLAPACK-PVM in LIDI network (last bar in light gray from left to right of Figure 5.9, identified with 32-4-2) corresponds to the running with data blocks of 32x32 elements and the processor grid of 4x2. The proposed algorithm, with the optimized communications, is identified as "Prop" and its corresponding bar in the figure is the darkest.

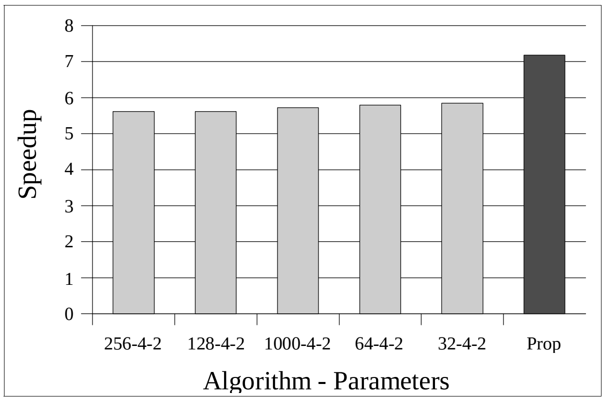


Figure 5.9: Matrix multiplication in LIDI, ScaLAPACK-PVM.

The best result in terms of Speedup obtained by ScaLAPACK-PVM is of slightly less than

5.8, and the performance obtained by the algorithm proposed is about 7.2; the percentage improvement with respect to ScaLAPACK-PVM is of approximately 25%. That is, with the same hardware, the proposed algorithm has almost a 25% better performance than that of ScaLAPACK-PVM. The optimal speedup is 8.

Figure 5.10 shows the best five performance results with the different parameter combinations with ScaLAPACK-PVM, and the performance of the algorithm proposed in Chapter 3 (overlapped computing with communications) plus the broadcast messages optimized in the LIDI-D network.

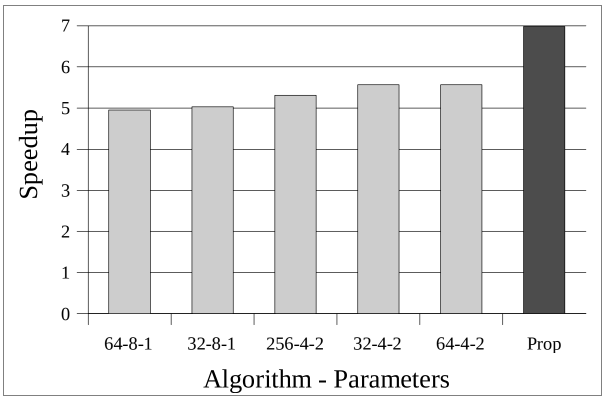


Figure 5.10: Matrix multiplication in LIDI-D, ScaLAPACK-PVM.

The best performance result obtained with ScaLAPACK-PVM in the LIDI network (last gray bar from left to right of Figure 5.10, identified with 64-4-2) corresponds to the running with data blocks of 64x64 elements and a 4x2 processor grid. The algorithm proposed, with optimized communications, is shown and identified as "Prop" and its corresponding bar is the darkest in the figure. The best performance results in terms of Speedup obtained by ScaLAPACK-PVM is of slightly less than 5.6 and the performance obtained by the proposed algorithm is of almost 7; the percentage improvement in relation to ScaLAPACK-PVM is of slightly more than 25%.

The best result in terms of Speedup obtained by ScaLAPACK-PVM is of slightly less than 5.6, and the performance obtained by the algorithm proposed is of almost 7; the percentage improvement with respect to ScaLAPACK-PVM is of approximately 25%. That is, with the same hardware, the proposed algorithm has almost a 25% better performance than that of ScaLAPACK-PVM. The optimal speedup is 8.

Figure 5.11 shows the best five performance results with the different parameter combinations with ScaLAPACK-PVM, and the performance of the algorithm proposed in Chapter 3 (overlapped computing with communications) plus the broadcast messages optimized in the LIDI-16 network.

The best performance result obtained with ScaLAPACK-PVM in LIDI-16 network (last light gray bar from left to right, identified with 256-4-4) corresponds to the running with 256x256-element data blocks and the 4x4 processors grid. The proposed algorithm, with the optimized communications, is shown and identified as "Prop", and its corresponding bar is the darkest in the figure.

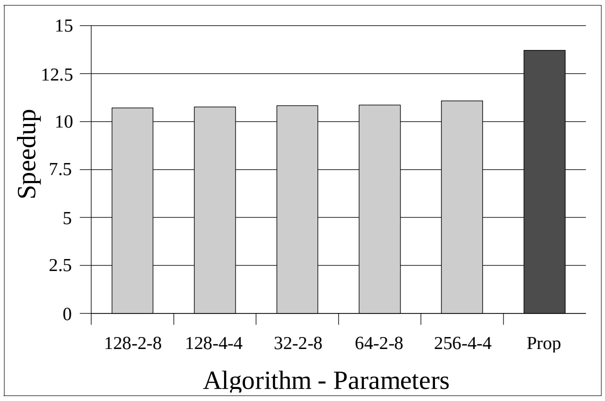


Figure 5.11: Matrix multiplication in LIDI-16, ScaLAPACK-PVM.

The best performance result in terms of Speedup obtained by ScaLAPACK-PVM is of slightly more than 11 and the performance obtained with the proposed algorithm is of almost 13.8; the percentage improvement with respect to ScaLAPACK-PVM is of almost 24%. That is, with the same hardware, the proposed algorithm has almost a 24% better speedup than that of ScaLAPACK-PVM. The optimal speedup in this case is 16.

Since for LU factorization of matrices there are different block sizes for the proposed algorithm, the results shown next have the following characteristics:

- The different block sizes are shown in each bar of the following figures, which correspond to the proposed algorithm of LU factorization, identified now with "Prop-block size".
- The best performance results for ScaLAPACK and for the proposed algorithm are shown, with each figure showing three light gray bars and three dark gray bars.

Figure 5.12 shows the three best performance results with the different parameter combinations with ScaLAPACK-PVM and the three best results performance results of the algorithm proposed in this chapter (computing overlapped with communications, different block sizes) implemented with broadcast messages optimized in LIDI network.

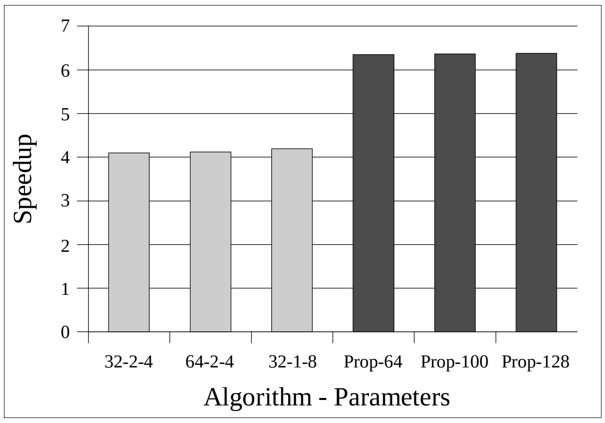
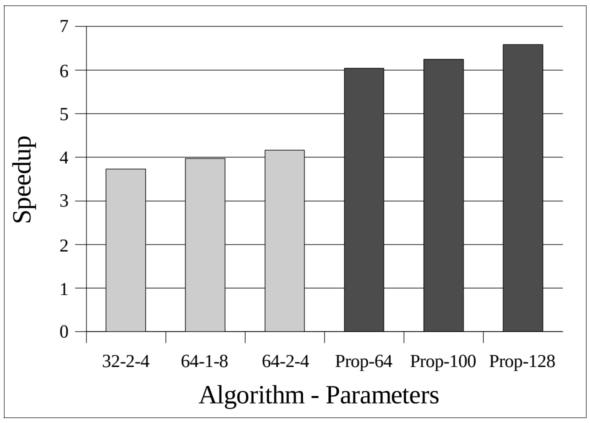


Figure 5.12: LU Matrix Factorization in LIDI, ScaLAPACK-PVM.

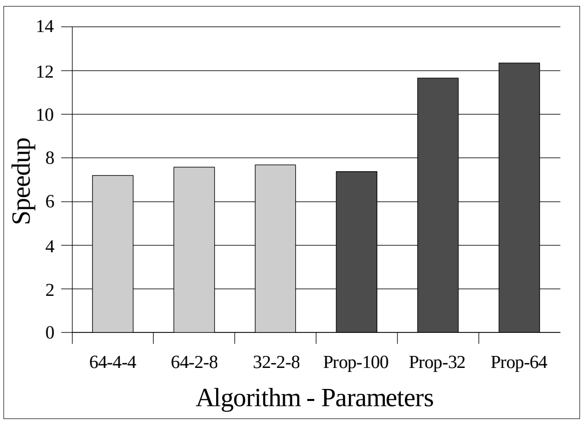
The best performance result obtained with ScaLAPACK-PVM in LIDI network (last light gray bar from left to right of the figure, identified with 32-1-8) corresponds to the running with block sizes of 32x32 elements and the 1x8 processor grid. The best performance of the proposed algorithm corresponds to the block size 128 and is indicated as "Prop-128" in the last bar from left to right of the figure.

The best performance result in terms of speedup obtained by ScaLAPACK-PVM is of approximately 4.2 and the performance obtained with the proposed algorithm is of almost 6.4; the percentage improvement with respect to ScaLAPACK-PVM is of almost 52%. That is, with the same hardware, the proposed algorithm has an approximately 52% better speedup than that of ScaLAPACK-PVM. The optimal speedup is 8.

In Figure 5.13, the results of the speedup obtained in LIDI-D network and LIDI-16 are shown. In all the cases, the improvement with the proposed algorithm is among the 50% and 60% in relation to the performance obtained by ScaLAPACK-PVM.



a) LIDI-D



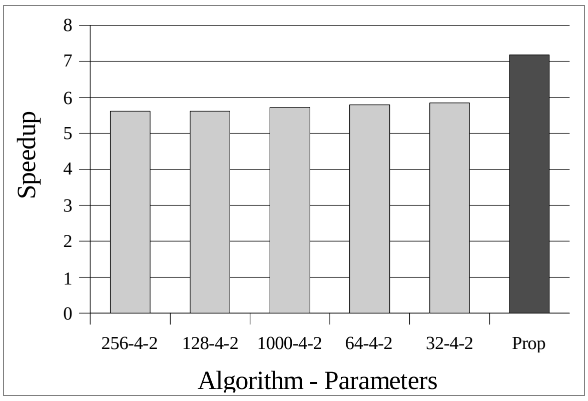
b) LIDI-16

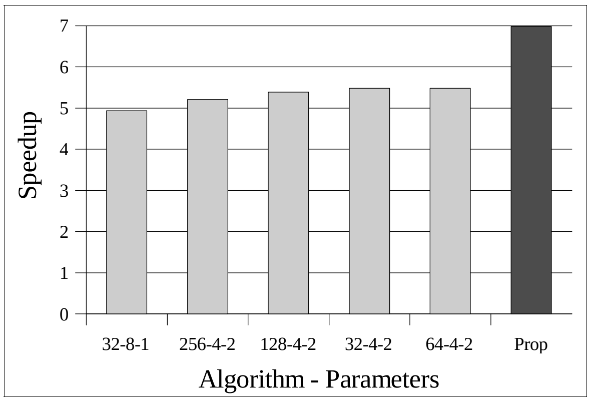
Figure 5.13: LU Factorization of Matrices in LIDI-D and LIDI-16, ScaLAPACK-PVM.

5.3.3 Results: ScaLAPACK-MPICH

ScaLAPACK installation *over* MPICH is also really simple, and so are the different running alternatives (which are parametric). Since the format of the figures showing the results obtained is the same, the results will be presented not as in detail as in the previous cases. In all the cases, the performance obtained with the algorithms proposed in this thesis is repeated in order to make a more immediate visual comparison. Figure 5.14 shows the results of the experiments in the two networks with eight PCs: LIDI and LIDI-D.

In the LIDI cluster, the algorithm proposed to multiply matrices has an almost 23% better performance than the ScaLAPACK library (in terms of Speedup values) and in the LIDI-D cluster it outperforms more than the 27%.





a) LIDI b) LIDI-D

Figure 5.14: Matrix Multiplication in LIDI and LIDI-D, ScaLAPACK-MPICH.

In order to complete the data, the Figure 5.15 shows the results of the performance obtained in the LIDI-16 network.

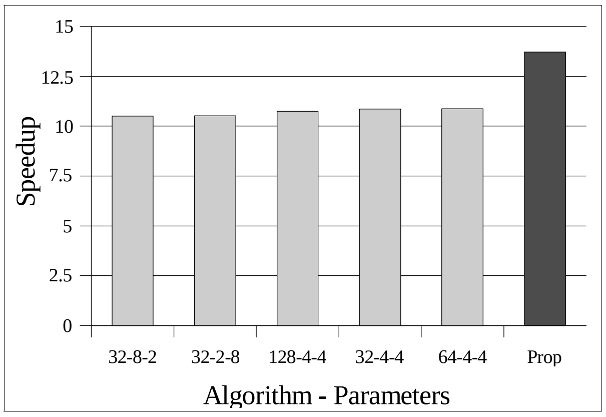
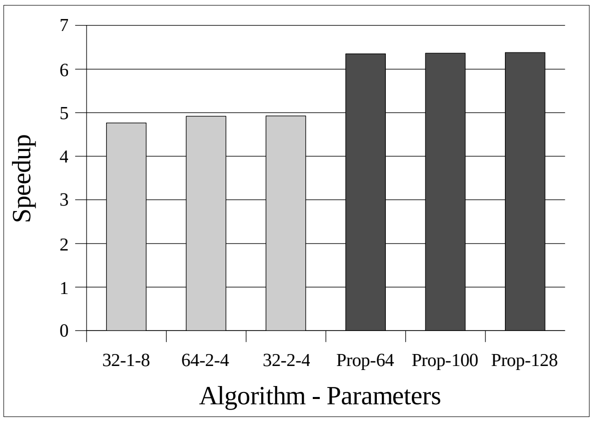


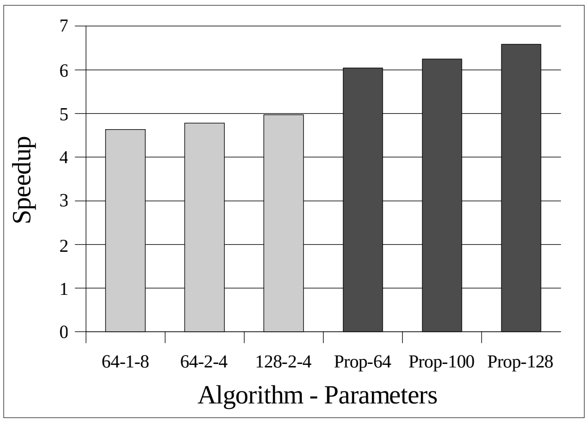
Figure 5.15: Matrix Multiplication in LIDI-16, ScaLAPACK-MPICH.

In this case, the algorithm proposed in this thesis to multiply matrices in clusters outperforms the ScaLAPACK matrix multiplication in more than the 26%.

Summarizing, the results obtained by ScaLAPACK-MPICH are very similar to those obtained with ScaLAPACK-PVM and, thus, the comparison of the performance results with the proposed matrix multiplication algorithm is similar as well.

The situation rather changes when the LU factorization is considered for analysis. The Figure 5.16 shows the speedup values obtained with ScaLAPACK-MPICH and repeats those obtained with the algorithm proposed in this chapter for the two networks of eight PCs.





a) LIDI b) LIDI-D

Figure 5.16: LU Matrix Factorization in LIDI and LIDI-D, ScaLAPACK-MPICH.

Comparing the Speedup values of ScaLAPACK-MPICH appearing in Figure 5.16 with those obtained with ScaLAPACK-PVM in Figure 5.12 and Figure 5.13-a), it is possible to note that the ScaLAPACK library has a better performance of the LU factorization when MPICH is in charge of the data transport. In fact, the proposed algorithm gain in relation to ScaLAPACK-PVM ranges from 50% to 60%, but in relation to ScaLAPACK-MPICH ranges from 30% to 32%. Two of the immediate conclusions from this information are:

- PVM has, in some cases, a strong communication performance penalization, which becomes a penalization of the total performance.
- The proposed parallel algorithms have an "almost constant" gain in relation to the algorithms implemented by ScaLAPACK. This gain, in percentage terms, is of approximately 25% in the case of matrix multiplication, and 30% in the case of LU matrix factorization. This is really interesting because they consist of two problems which are parallelized using the same principles and the gain is higher for both, i.e. parallelization principles for local networks are "proved" as the best, at least for these two cases: multiplication and LU factorization of matrices.

Figure 5.17, apart from completing the data of the matrix factorization in parallel for the 16-PCs network, adds valuable information in relation to ScaLAPACK performance when the network is not completely "switched", as in the case of this particular network.

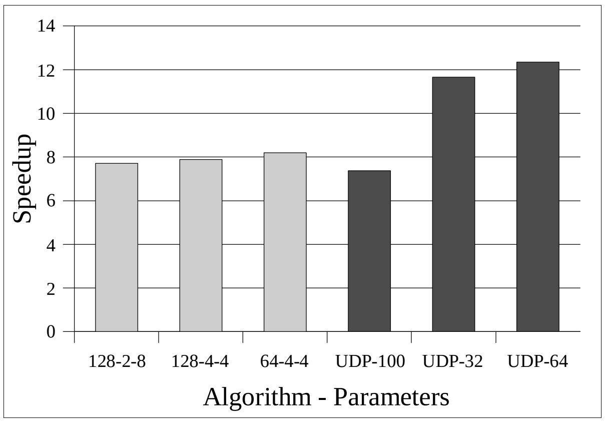


Figure 5.17: LU Matrix Factorization in LIDI-16, ScaLAPACK-MPICH.

As it can be noticed, the performance of ScaLAPACK-MPICH is quite similar to that of ScaLAPACK-PVM (and, thus, the difference with the matrix factorization algorithm proposed in this chapter is again quite big). There are at least two reasons for which ScaLAPACK-MPICH "gets worse":

- ScaLAPACK matrix factorization algorithm is not scalable and, thus, when more computers are used, the performance gets worse.
- The completely "switched" network generates an extra performance penalization not included in the completely "switched" networks and, consequently, the total parallel performance remarkably gets worse.

In this point, it should be taken into account that the Speedup obtained in the two interconnected networks with a single switch is of approximately 63% of the optimum, while the obtained by the LIDI-16 network (which does not have a single switch, but three "cascade" switches) is of approximately 50% of the optimum. In consequence, since

- the performance of ScaLAPACK is highly influenced by the communication performance,
- the relative performance obtained in the two networks interconnected by a single switch is quite higher than that obtained in a network which does not count with that interconnection characteristic,

ScaLAPACK's lack of scalability can be discarded, at least in principle.

5.3.4 ScaLAPACK-MPICH and Scalability

This last subsection shows the results obtained in most of the clusters over which the experimentation could be carried out in order to compare the algorithms proposed in this thesis with those implemented in ScaLAPACK to carry out the same task. This cluster is composed by 20 PCs interconnected by a single 100 Mb/s Ethernet switch, i.e. there exists a complete switching with the 20 machines interconnected by the 100 Mb/s Ethernet. Table 5.4 briefly shows the PCs characteristics of the cluster, which shall be called from now on CI-20, together with its computing capacity expressed in Mflop/s terms.

Processors	Clock Freq.	Memory	Mflop/s
Intel P4	2.4 GHz	1 GB	5000

Table 5.4: Characteristics of the Computers of the Cluster CI-20.

On the other hand, Table 5.5 shows the matrix sizes used in the problems of multiplication and LU factorization, respectively.

Multiplicación de Matrices	Factorización LU de Matrices
38000	65000

Table 5.5: Sizes of Problems in Cluster CI-20.

Since the ScaLAPACK library obtains its best performance results when using the MPICH message passing library, all the experiments carried out with ScaLAPACK makes use of MPICH for message passing. In a certain way, the experiments presented next have two characteristics that differentiate them from the previous:

1. They use the greatest quantity of machines and, in a certain way, show quite limitedly (up to 20 computers) the proposed algorithms scalability.
2. They use the computers with highest computing power, keeping the interconnection network in 100 Mb/s, with which we have the worst relation between the performance of local computing and that of the communications.

Since there are 20 computers, and ScaLAPACK recommendations for obtaining optimized performance is to keep the grid PxQ with P as similar as possible to Q [21], the grids used in the experiments were of 4x5 and 5x4 processors.

Figure 5.18 shows the performance values obtained with ScaLAPACK-MPICH and with the algorithm proposed in this thesis (computing overlapped with communications) for the multiplication of 38000x38000 element matrices. The best value obtained by ScaLAPACK -MPICH is obtained with 32-block size and with the machines interconnected as in a 5x4 processor grid. The absolute Speedup value obtained by ScaLAPACK-MPICH is, in this case, of approximately 10. In the case of the algorithm proposed, we obtain a Speedup of slightly more than 16, which represents an improvement of approximately a 62% with respect to ScaLAPACK-MPICH. In this case, the optimum Speedup value is 20, the number of computers used.

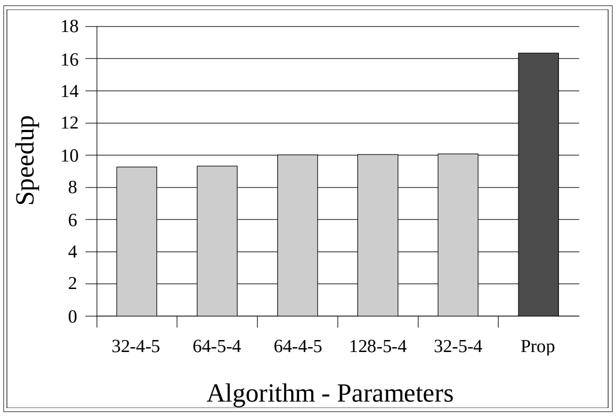


Figure 5.18: Matrix Multiplication in CI-20, ScaLAPACK-MPICH.

Figure 5.19 shows the performance values obtained with ScaLAPACK-MPICH and with the algorithm proposed in this thesis (computing overlapped with communications) for the LU factorization of 65000x65000 element matrices. The best value obtained by ScaLAPACK-MPICH is obtained with 64-block size and with the machines interconnected as in a 4x5 processor grid. The absolute Speedup value obtained by ScaLAPACK-MPICH is, in this case, slightly more than 10. In the case of the algorithm proposed, the obtained Speedup is of more than 18, which represents an improvement of approximately an 80% with respect to ScaLAPACK-MPICH. Also in this case, the optimum Speedup value is 20, the number of computers used.

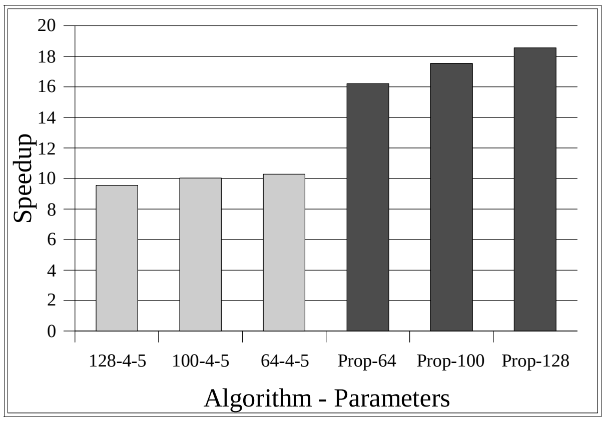


Figure 5.19: Matrix LU Factorization in CI-20, ScaLAPACK-MPICH.

It is very interesting to notice that, beyond the comparison with ScaLAPACK, the obtained Speedup is really close to the absolute maximum. In the case of matrices multiplication, almost the 82% of the optimum performance is obtained, and in the case of the LU factorization almost the 93% of the optimum is obtained, which is really satisfactory, taking into account that, for instance, there is an interconnection network of only 100Mb/s and with high latency.

5.4 Summary of the Comparison with ScaLAPACK

Table 5.6 briefly shows the comparison of the performance obtained by the algorithms proposed in this thesis with those implemented by/in ScaLAPACK. For each of the clusters used (LIDI, LIDI-D, LIDI-16, and Cl-20)

- ScaLAPACK's performance in terms of Speedup (**Sca** column)
- The proposed algorithms' performance in terms of Speedup (**Prop** column)
- The percentage of the improvement in Speedup obtained with the use of the algorithms proposed in this thesis (**%Prop** column).

	LIDI 8 PIII - 100 Mb/s			LIDI-D 8 D - 100 Mb/s			LIDI-16 16 PCs - 100 Mb/s*			Cl-20 20 PCs - 100 Mb/s		
	Sca	**Prop**	**%Prop**	**Sca**	**Prop**	**%Prop**	**Sca**	**Prop**	**%Prop**	**Sca**	**Prop**	**%Prop**
MM	5.84	7.18	+23%	5.48	6.98	+27%	10.87	13.72	+26%	10.08	16.35	+62%
LU	4.93	6.38	+30%	4.97	6.59	+33%	8.2	12.35	+51%	10.28	18.56	+81%

*The only Ethernet network with cascade switches combination; the remaining has complete switching.

Table 5.6: Summary of the Comparison with ScaLAPACK.

Even though the gain in terms of Speedup is noteworthy, it is even more important the fact that in clusters with more computers and with a worse performance relation between local computing and communications, the gain tends to be higher. Even more, independently of the comparison with ScaLAPACK, Table 5.7 shows the absolute Speedup values obtained by the algorithms proposed in this thesis (column **Prop**) together with the optimal percentage that those values represent (column **%Op**).

	LIDI 8 PIII - 100 Mb/s		LIDI-D 8 D - 100 Mb/s		LIDI-16 16 PCs - 100 Mb/s*		Cl-20 20 PCs - 100 Mb/s	
	Prop	**%Op**	**Prop**	**%Op**	**Prop**	**%Op**	**Prop**	**%Op**
MM	7.18	90%	6.98	87%	13.72	86%	16.35	82%
LU	6.38	80%	6.59	82%	12.35	77%	18.56	93%

*The only Ethernet network with cascade switches combination; the remaining has complete switching.

Table 5.7: Relation of the Proposed Algorithms with the Absolute Optimum.

All the values, except for LU factorization in LICI-16, outperforms the 80 % of the absolute optimum, which is highly satisfactory, and even more when recalling that a really low-cost interconnection network, which does not depend on the use of switches for interconnection, is being used. Even when the exception of LU in LIDI-16 could be used as an indicator that the algorithm is not scalable enough, this assumption could be discarded basing on the values obtained for more quantity of computers and of higher CI-20 cluster computing power.