Parallel Matrix Multiplication and LU Factorization on Ethernet-Based Clusters

Fernando G. Tinetti, Mónica Denham, and Armando De Giusti

Laboratorio de Investigación y Desarrolo en Informática (LIDI),
Facultad de Informática,
UNLP,
50 y 115, 1900 La Plata, Argentina,
Tel.: +54 221 4227707, Fax: +54 221 4273235,
{fernando,mdenham,degiusti}@lidi.info.unlp.edu.ar

Abstract. This work presents a simple but effective approach for two *representative* linear algebra operations to be solved in parallel on Ethernet-based clusters: matrix multiplication and LU matrix factorization. The main objectives of this approach are: simplicity and performance optimization. The approach is completed at a lower level by including a broadcast routine based directly on top of UDP to take advantage of the Ethernet physical broadcast facility. The performance of the proposed algorithms implemented on Ethernet-based clusters is compared with the performance obtained with the ScaLAPACK library, which is taken as having highly optimized algorithms for distributed memory parallel computers in general and clusters in particular.

Keywords: Cluster Computing, Parallel Linear Algebra Computing, Communication Performance, Parallel Algorithms, High Performance Computing.

1 Introduction

The field of parallel computing has evolved in several senses, from computing hardware to algorithms and libraries specialized in particular areas. The processing basic hardware used massively in desktop computers has also increased its capacity in orders of magnitude and, at the same time, it has reduced its costs to end users. On the other hand, costs associated to a specific (parallel) computer cannot always be afforded, and remained *almost constant* since many years ago. From this point of view, clusters have the two main facilities needed for parallel computing: 1) Processing power, which is provided by each computer (CPU-memory), 2) Interconnection network among CPUs, which is provided by LAN (Local Area Network) hardware, usually Ethernet [7] at the physical level.

Traditionally, the area of problems arising from linear algebra has taken advantage of the performance offered by available (parallel) computing architectures. LAPACK (Linear Algebra PACKage) [1] and BLAS (Basic Linear Algebra

A. Veidenbaum et al. (Eds.): ISHPC 2003, LNCS 2858, pp. 431–439, 2003.

Subroutines) [3] definitions represent the main results of this effort. Parallel algorithms in the area of linear algebra have been continuously proposed mainly because of their great number of (almost direct) applications and large number of potential users. Two immediate examples are: 1) Matrix multiplication has been used as a low-level benchmark for sequential as well as parallel computers [6] or, at least, as a BLAS benchmark. In some way or another, results are still being reported in relation to matrix multiplication performance in parallel and sequential computers, and 2) LU factorization is used to solve dense systems of linear equations, which are found in applications such as [5] airplane wing designs, radar cross-section studies, supercomputer benchmarking, etc.

In the field of parallel (distributed) linear algebra, the ScaLAPACK (Scalable LAPACK) library has been defined [2]. ScaLAPACK is no more (and no less) than LAPACK implementing high quality (or optimized) algorithms for distributed memory parallel computers. Matrix multiplication and LU factorization will be analyzed in detail for Ethernet-based clusters, and they will also be taken as representatives of the whole LAPACK-ScaLAPACK libraries.

The main project of which this work is part of 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 obtain optimized parallel performance in clusters interconnected by Ethernet.

2 Algorithms for Matrix Multiplication and LU Factorization

Being parallel matrix multiplication and LU factorization implemented in very well known, accepted, and optimized libraries such as ScaLAPACK, is it necessary to define *another* parallel matrix multiplication and LU factorization? Parallel algorithms selected for implementation and inclusion in ScaLAPACK are assumed to have optimized performance in parallel computers with distributed memory and good scalability. However, as the performance is usually strongly dependent on hardware, the algorithms selected are also influenced by the *current* distributed memory parallel computers at the time ScaLAPACK was defined (the '90s). Those computers follow the guidelines of the *traditional* distributed memory parallel (super)computers, i.e.: 1) High-performance CPUs, the best for numerical computing, and sometimes designed *ad hoc* for this task, 2) Memory directly attached to each processor, with message passing necessary for interprocess-interprocessor communication, and 3) Low latency (startup)-High throughput (bandwidth) processor interconnection network. But clusters are not (at least *a priori*) parallel computers.

As regards clusters interconnected by Ethernet networks, the interconnection network implies strong performance penalties when processes running on different computers of the cluster are communicated. From this point of view, parallel algorithms have not been designed *for* the parallel architecture of the clusters,

they have been rather used as designed for distributed memory parallel computers. Furthermore, there has been and there is a strong emphasis on leveraging the interconnection network, so as to approach the interconnection network of the traditional parallel (super)computers.

Taking into account the characteristics of clusters with Ethernet network interconnection, some guidelines have been devised in order to design parallel algorithms for linear algebra operations and methods:

- SPMD (Single Program, Multiple Data) parallel execution model, which is not new in the field of linear algebra numerical computing. Also, the SPMD model is very useful for simplifying the programming and debugging task.
- The Ethernet hardware defined by the IEEE standard 802.3 [7] *almost directly* leads to: 1)Unidimensional data distribution, given that every interconnected computer is connected at the same "logical bus" defined by the standard, and 2) Broadcast based intercommunication among processes, given that the Ethernet logical bus makes broadcast one of the most *natural* messages to be used in parallel applications. Also, every cabling hardware is IEEE 802.3 compliant and, thus, broadcast has to be physically implemented.

2.1 Matrix Multiplication

The basic parallel matrix multiplication algorithm ($C = A \times B$) proposed is derived from the algorithm already published in [10] for heterogeneous clusters, and can be considered as a *direct* parallel matrix multiplication algorithm [11]. It is defined in terms of data distribution plus the program that every computer has to carry out for the parallel task. The (one-dimensional) matrix distribution is defined so that: 1) Matrices A and C are divided into as many row blocks as computers in the cluster, and every computer holds one row block (row block partitioning [8]); computer P_i has row blocks $A^{(i)}$ and $C^{(i)}$, 2) Matrix B is divided into as many column blocks as computers in the cluster, and every computer holds one column block (column block partitioning [8]); computer P_i has column block $B^{(i)}$, 3) Each computer P_i has only a fraction of the data needed to calculate its portion of the resulting matrix $C^{(i)}$, i.e. $C^{(i_i)} = A^{(i)} \times B^{(i)}$.

Fig. 1 shows the pseudocode of the program on each computer, which has only local computing plus broadcast communications for processor P_i, where 1) `C^(i_j)` stands for $C^{(i_j)}$, `A^(i)` for $A^{(i)}$, and `B^(j)` for $B^{(j)}$, and 2) The operations `send_broadcast_o` and `recv_broadcast_o` are used to send and receive broadcast data in "background", i.e. overlapped with local processing in the computer, where available.

2.2 LU Factorization

Currently used parallel LU factorization algorithm is derived from the combination of [5]: a) the traditional right-looking LU factorization algorithm, b)

```
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^(i_j) = A^(i) * B^(j)
}
```

Fig. 1. Matrix Multiplication with Overlapped Communications.

data processing made by blocks (block processing), and c) two dimensional matrix data partitioning. Following the guidelines stated previously, the selected matrix partitioning is chosen following the so-called row block matrix distribution [8]. To avoid the unacceptable performance penalty of unbalanced workload maintaining the one-dimensional data distribution among computers, the matrix is divided in many more blocks than computers and assigned cyclically to processors. In this way: 1) One dimensional data partitioning is made, there is no defined neighborhood for each processor according to its position in a processor mesh, and 2) When the number of blocks is large enough ($bs \ll p$), every processor will have data to work with in most of the iterations.

2.3 Broadcasting Data Using UDP

The algorithms explained are *strongly* dependent on the broadcast performance. In particular, the broadcast performance of PVM [4] is not acceptable, since it is implemented as multiple point-to-point messages from the sender to each receiver when processes are running on different computers. In general, MPI [9] implementations do not necessarily have to optimize broadcast performance and/or optimize the availability of the Ethernet logical bus since the MPI standard itself does not impose any restriction and/or characteristic on any of the routines it defines.

When the previously mentioned broadcast-based guideline is used for parallelization, the design and implementation of an optimized version of the broadcast communication between processes of parallel programs is highly encouraged. Ethernet networks are based on a logical bus which can be reached from any attached workstation [7], where the method to access the bus is known as multiple random access. Basically, in every data communication, the sender floods the bus, and only the destination/s will take the information from the channel. The destination/s might be a group of computers and, in this case, the sender uses a multicast/broadcast address, sending data only once. All the referenced computers will take the data from the channel at the same time.

3 Experimental Work

The proposed parallel algorithms (for matrix multiplication and LU factorization) plus the new broadcast routine optimized for Ethernet interconnected clusters are used in two homogeneous clusters of eight PCs each: the PIII cluster and the Duron Cluster. The characteristics of PCs in each cluster are summarized in Table 1. Mflop/s was taken by running the best matrix multiplication sequential algorithm on single precision floating point matrices. The interconnection network on each cluster is a fully switched 100 Mb/s Ethernet. Algorithms performance is compared with that obtained by the ScaLAPACK library, which is installed on both clusters on top of MPICH. A *third* cluster is considered, which will be called 16-PC: both previous clusters interconnected and used as a cluster with sixteen *homogeneous* PCs of 580 Mflop/s with 64 MB each.

Table 1. Clusters Characteristics.

Cluster	CPU	Clock	Mem	Mflop/s
PIII	Pentium III	700 MHz	64 MB	580
Duron	AMD Duron	850 MHz	256 MB	1200

3.1 Matrix Multiplication Performance

The algorithm proposed for parallel matrix multiplication on Ethernet-based clusters (Section 2.1 above) was implemented and compared with the one provided by the ScaLAPACK library. Fig. 2 shows the best five ScaLAPACK performance values in terms of speedup on the PIII cluster and the speedup obtained for the parallel matrix multiplication proposed in this. Two matrices of 5000x5000 single precision floating point numbers are multiplied. Given that ScaLAPACK performance depends on some parameters values, a range of values was used for testing ScaLAPACK matrix multiplication performance: a) Square blocks: 32x32, 64x64, 128x128, 256x256, and 1000x1000 elements, and b) Processors meshes: 1x8, 8x1, 2x4, and 4x2 processors on the clusters with 8 PCs,

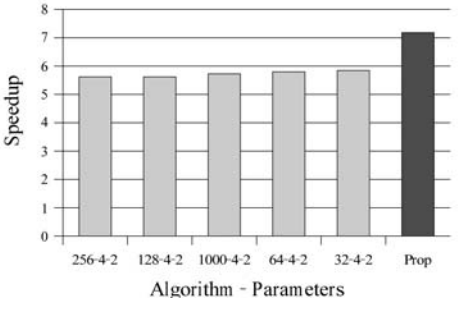
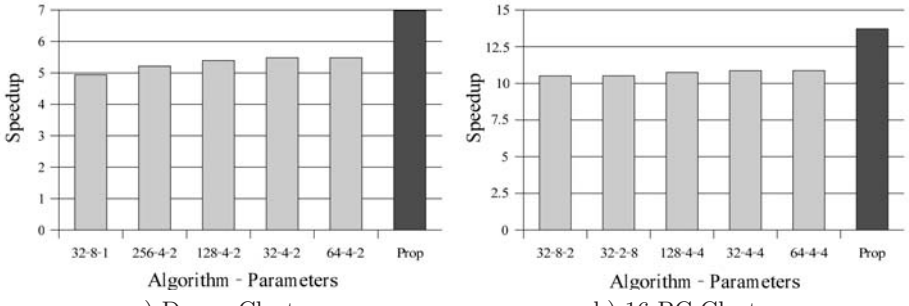


Fig. 2. Matrix Multiplication on PIII Cluster.

and 2x8, 8x2, and 4x4 processors on the cluster with 16 PCs. Each bar of Fig. 2 is labeled according to the algorithm and parameters used (when required). ScaLAPACK matrix multiplication is identified by the values for the three parameters on which it depends upon: processors per row, processors per column, and block size. Performance of the parallel matrix multiplication proposed is labeled as "Prop".



a) Duron Cluster b) 16-PC Cluster

Fig. 3. Matrix Multiplication on the Duron and 16-PC Cluster.

Fig. 3 a) shows the best five ScaLAPACK performance values in terms of speedup on the Duron cluster and the speedup obtained for the parallel matrix multiplication proposed in this work, and matrices of 10000x10000 elements. Also, Fig. 3 b) shows the best five ScaLAPACK performance values in terms of speedup on the 16-PC cluster as well as the speedup obtained for the parallel matrix multiplication proposed in this work. Matrices are of 8000x8000 elements. Table 2 summarizes performance results on the three clusters.

Table 2. Matrix Multiplication Performance Summary.

Cluster	Sc-Best	Prop-Best	% Better Perf.	Opt. Perf.
PIII	5.9	7.2	23%	8
Duron	5.5	7.0	27%	8
16-PC	10.9	13.7	26%	16

3.2 LU Matrix Factorization Performance

Similar tests for parallel LU factorization were carried out on the same clusters. In this case, the proposed parallel algorithm is also defined in terms of a (row) blocking size according to Section 2.2 above. The blocking sizes for ScaLAPACK as well as for the proposed algorithm were: 32, 64, 128, 256, and 1000. The proposed algorithm was tested with other intermediate blocking sizes (as blocks of 100 rows) in order to verify its dependence on row block size.

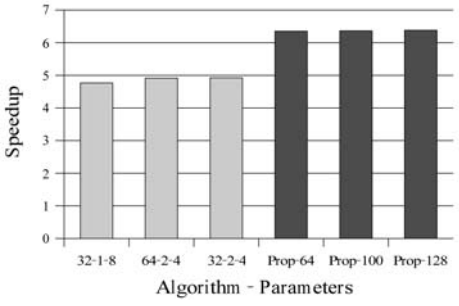
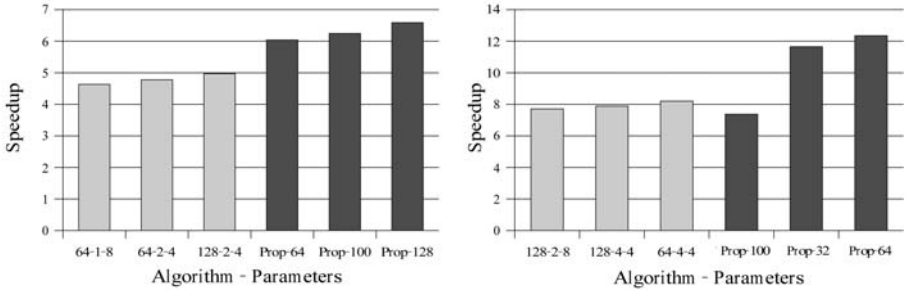


Fig. 4. LU Factorization on PIII Cluster.



a) Duron Cluster b) 16-PC Cluster

Fig. 5. LU Matrix Factorization on the Duron and 16-PC Cluster.

Table 3. LU Matrix Factorization Performance Summary.

Cluster	Sc-Best	Prop-Best	% Better Perf.	Opt. Perf.
PIII	5.0	6.4	30%	8
Duron	5.0	6.6	32%	8
16-PC	8.2	12.3	50%	16

Fig. 4 shows the best three performance (speedup) values for each of the algorithms used on the PIII cluster. The proposed parallel LU matrix factorization algorithm is labeled "Prop-*rbs*" where *rbs* is the row blocking size used in the experiment. On each experiment, a matrix of 9000x9000 single precision elements is factorized into L and U. Fig. 5 a) shows the performance (speedup) values on the Duron cluster. On each experiment, a matrix of 20000x20000 single precision elements is factorized into L and U. Also, Fig. 5 b) shows the performance (speedup) values on the PC-16 cluster where a matrix of 13000x13000 single precision elements is factorized into L and U on each experiment. Table 3 summarizes performance results on the three clusters.

4 Conclusions and Further Work

The main conclusion derived from performance results shown in the previous section is that: there is enough room for optimization of linear algebra algorithms on Ethernet-based clusters. Optimizations are focused to optimize resource usage on Ethernet-based clusters. Proposals on this work can be summarized as: 1) A few guidelines to parallelize linear algebra applications, with two main objectives: simple parallel algorithms, and optimized performance on Ethernet-based clusters. 2) Two parallel algorithms have been successfully designed and implemented: matrix multiplication and LU matrix factorization. 3) It has been shown by experimentation that it is possible to obtain better performance than that obtained by the algorithms implemented in the highly optimized ScaLAPACK library. 4) It is possible to have optimized performance on Ethernet-based clusters without imposing fully switched interconnections. In this way, the whole cost of the interconnection network is greatly reduced for clusters with large number of computers, where the switching cost increases more than linearly when the number of computers grows.

Even when broadcast messages based on UDP are expected to have very good scalability, it is important to experiment on clusters with many computers to verify this assumption. In this sense, experiments should be carried out on cluster with tens and hundreds of computers interconnected by Ethernet. Also, it should be highly beneficial to experiment on the recently proposed Gb/s Ethernet in order to analyze scalability and performance gains. From the point of view of linear algebra applications, it is necessary to continue with optimization of other methods and/or operations as those included in LAPACK-ScaLAPACK.

References

1. Anderson E. *et al.*, "LAPACK: A Portable Linear Algebra Library for High-Performance Computers", Proceedings of Supercomputing '90, pages 1-10, IEEE Press, 1990.
2. Blackford L. *et al.* ScaLAPACK Users' Guide, SIAM, Philadelphia, 1997.
3. 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.
4. 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.
5. 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.
6. Hockney R., M. Berry (eds.), "Public International Benchmarks for Parallel Computers", Scientific Programming 3(2), pp. 101-146, 1994.
7. 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.
8. Kumar V., A. Grama, A. Gupta, G. Karypis, Introduction to Parallel Computing. Design and Analysis of Algorithms, The Benjamin/Cummings Publishing Company, Inc., 1994.

9. Message Passing Interface Forum, "MPI: A Message Passing Interface standard", International Journal of Supercomputer Applications, Volume 8 (3/4), 1994.
10. Tinetti F., A. Quijano, A. De Giusti, E. Luque, "Heterogeneous Networks of Workstations and the Parallel Matrix Multiplication", Y. Cotronis and J. Dongarra (Eds.): EuroPVM/MPI 2001, LNCS 2131, pp. 296-303, Springer-Verlag, 2001.
11. Wilkinson B., Allen M., Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers, Prentice-Hall, Inc., 1999.