Sparse Equation Systems in Heterogeneous Clusters of Computers

Fernando G. Tinetti* Walter J. Aróztegui Antonio A. Quijano

Centro de Técnicas Analógico-Digitales
48 y 116 2do. Piso
Facultad de Ingeniería
Universidad Nacional de La Plata
1900 La Plata, Argentina

Instituto de Investigación en Informática - LIDI
50 y 115 1er. Piso
Facultad de Informática
Universidad Nacional de La Plata
1900 La Plata, Argentina

fernando@info.unlp.edu.ar, waroz@graffiti.net, quijano@volta.ing.unlp.edu.ar

Abstract

This paper presents a parallelization strategy in heterogeneous clusters of the Gauss-Seidel's method applied for the solution of sparse equation systems. From the point of view of the numerical solution for matrices of coefficients with low density of non null-elements, the standard lines of thought are followed, that is, only non-null elements are stored and iterative solution-search methods are used.

Two basic guidelines are defined for the parallel algorithm: one-dimensional data distribution and broadcast messages for all data communications. One-dimensional data distribution eases the processing workload balance on heterogeneous clusters. The use of broadcast messages for every data communication is directly oriented to optimize performance on the the most common cluster interconnection: Ethernet. Experimental results obtained in a local network of heterogeneous computers are presented.

1. Introduction

Several branches of Science and Engineering need a considerable computing power when modeling and emulating various physical phenomena. It is enough to quote some problems, such as environmental models in meteorology applications, aerodynamics in the spacecrafts design, electronic circuits, and a wide range of typical scientific applications. Many of these models involve a large number of partial differential equations (PDEs), and the most common way of solving them is by discretization, that is, approaching them by means of equations with a finite number of unknowns. Large linear equation systems are thus produced, with many coefficients equal to zero, which leads to call those systems as linear sparse equation systems.

Since some years ago, advances included in the so-called "domestic" or desktop computers and the more than acceptable relationship between cost and computing power determine the tendency to use parallel computers made up by networks or PC clusters [10]. This paper is focused on heterogeneous clusters connected by Ethernet networks, which can be found in several and daily environments.

In the special case of sparse systems, in which a great part of the element of the system matrix are zeroes, iterative methods are particularly efficient, even more when special storage schemes for sparse matrices are applied and a the system has a low quantity of non- null elements. Iterative or indirect methods give rise to a series of vectors, which ideally converge in the solution of the system by means of the repetition of a simple process. The computation stops before or after a determined number of iterations [4] [7]. Iterative methods are useful given that null coefficients will remain invariable and can be disregarded, saving memory space and processing time, since it is not necessary to neither access nor use null operands.

2. Matrix Storage and Gauss-Seidel's Method

This section initially shows how space can be saved in the storage of matrices with a great amount of null-elements. It then shows how equation systems are solved with the Gauss-Seidel's method and to what extent this method is affected by the sparse matrix storage system.

* Investigador Asistente de la Comisión de Investigaciones Científicas de la Provincia de Buenos Aires

Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA'05)
1550-445X/05 $20.00 © 2005 IEEE

Sparse Matrix Storage. The most simple of these storage schemes is the so-called of coordinated format, which consists of a data structure with three arrays: a real array containing all the non-null values, an integer array containing the row indexes, and a second integer array containing the column indexes. As the elements are usually listed by rows or columns, one of the arrays with the previous indexes can contain redundant information. Taking this into account, the CSR (Compressed Sparse Row) storage format is defined, which is probably the most popular in the context of sparse matrices [8]. In this case, there are also three arrays, though now are:

1) An array containing the non-null elements of the matrix, stored by rows.

2) An integer array with the respective column indexes.

3) An integer array containing the pointers to the beginning of each row.

Thus, there are two arrays of length equal to the number of non-null elements, and another of the length $n + 1$ with n equal to the number of rows (equations), being its last element the one that indicates the total number of non-null elements plus one. Other ways of storing sparse matrices are: CSC (Compressed Sparse Column) [5], MSR (Modified Sparse Row), [8], by diagonals [8] [2], and Ellpack-Itpack [8].

Gauss-Seidel's Iterative Method. Assuming that the equation system coefficients are denoted as a_{ij}, and b_i is the result of each equation, a general expression for the Gauss Seidel's procedure is:

$$x_i^{(k)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_i^{(k)} - \sum_{j>i} a_{ij} x_i^{(k-1)} \right) \qquad (1)$$

which at least requires that $a_{ii} \neq 0$, and $x_i^{(k)}$ are the unknown values on iteration k. The error computation often uses the differences among successive values of variables:

$$\Delta x_i = |x_i^{(k)} - x_i^{(k-1)}| \qquad (2)$$

The Gauss-Seidel's method is not altered by the way in which the coefficient matrix is stored. It is clear that the access to data, in the case of CSR storage for sparse matrices, should be carried out using the previously mentioned vectors. CSR storage does not change the algorithm's numerical characteristics but the access pattern to data and the number of operations carried out since null-values are not used. This paper presents the parallelization of the Gauss-Seidel's method for several reasons:

1) It is one of the most representatives among the simplest iterative algorithms.

2) It is an improvement of the Jacobi's method, with better convergence time, and can be considered as a particular case of another well-known procedure, the SOR (Successive Overrelaxation).

3) Data dependency in Eq. (1) is strongly enough not to be in the best of the cases for parallel computing requirements, as in the case of the Jacobi's method. In the Jacobi's method, once a solution is obtained, the computations for the following solution are all independent and their parallelization is thus immediate.

3. Parallel Gauss-Seidel's Method

As previously mentioned, parallelization of the Gauss-Seidel's method is not immediate due to the existing data dependency when computing the values of each iteration. In general, for sparse matrices, efforts are focused not as much on parallelization as on matrices' pre-ordering in order to turn them into forms that can be efficiently implemented in parallel. Apart from using storage specifically oriented to sparse equation systems, the proposed parallelization of this paper follows the principles outlined in [9], which can be stated as follows:

1) Message-passing programming model, which is considered the most appropriate for parallel processing in clusters.

2) SPMD (Single Program-Multiple Data) processing model, which is simple and scalable. On the other hand, most of the parallel algorithms for lineal algebra problems follow this processing model.

3) One-dimensional data distribution, i.e. all the processes/processors are considered as interconnected in a linear array or by a communication bus. There exists a tendency to think in a bus following the very definition of the Ethernet standard of computers' interconnection in a local network.

4) Communication among processes via broadcast messages when possible, i.e. all data interchange is thought in function of a process sending data and the remaining processes receiving the data sent. There also exists the tendency to optimize the implementation of this communication primitive over the hardware provided by the Ethernet standard [3].

The partition or distribution of data to a processor among processes or processors is carried out by row blocks. The row block size is defined directly proportional to the computer's relative computing power, pwi, defined as

$$pw_i = \frac{\max_{j=0}^{p-1} t_j}{t_i} \qquad (3)$$

where t_k, $0 \leq k \leq p-1$, can be found by running the same (small enough to be stored in memory) sparse system on each computer. Figure 3 schematically shows the partition of an equation system among three processors P_0, P_1 and P_2, where P_0 is the slowest computer, P_1 is two times

faster than P_0, and P_2 is three times faster than P_0. Recall that, on the one

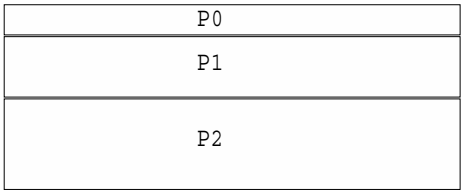

Figure 1. Row Block Distribution.

hand, this partition is non-dependent on the CSR storage of a sparse matrix and, on the other, given a row block, the set of variables to be calculated by the process is determined automatically: each process compute the value of the variables which are associated to the main diagonal of coefficients matrix (sparse), such as can be derived from Eq. (1). In general, each computer will make pw_i/PW_p of the total work to be done, where PW_p is

$$PW_p = \sum_{i=0}^{p-1} pw_i \tag{4}$$

From now on, the iterative process begins, which can be separated into several stages, and it should be described from the first iteration, assuming an initial value for the unknowns $(x_1^{(0)}, \ldots, x_n^{(0)})$:

1.- The process with the first row block begins the computation of the associated variables. That is, process P_0 may determine the values of the variables $(x_1^{(1)}, \ldots, x_{bs_0}^{(1)})$, where bs_0 is the row block size corresponding to Process P_0. The remaining processes can only carry out the intermediate results corresponding to the second sum in Eq. (1), i.e. those dependent on initial values.

2.- Once the first bs_0 values of the first iteration are computed, the remaining processes can use these values to compute the first sum in Eq. (1). Since process P_0 is the one that has the values of these variables and the remaining processes need them, it is natural to think in a broadcast from process P_0 to the rest. In fact, in this second step, all the computations necessary for the values $(x_{bs_0+1}^{(1)}, \ldots, x_{bs_0+bs_1}^{(1)})$ are completed in process P_1. These *new* bs_1 values can be sent to the remaining processes for the computation depending on them according Eq.(1). Notice that these values can be also used in process P_0, though this time for the computation of the variables of the following Gauss-Seidel iteration.

3.- The two previous steps can be continued taking as reference processes P_i up to the last process with the last row/variable block, where: all the values for variables of a Gauss-Seidel's iteration are computed, and the total error can be obtained, and thus it is possible to determine whether to go on or not, considering the value of this error.

4. Experimental Results

In terms of hardware, experiments were carried out in a network of five PCs interconnected with switched 100 Mb/s Ethernet. Table 1 summarizes the relative computing powers of the five computers according with Eq. (3). Values are ordered from the fastest, PC1, to the slowest, PC5. All

Comp.	PC1	PC2	PC3	PC4	PC5
Pwi	6.50	3.45	1.45	1.45	1.00

Table 1. Relative Computer Speeds.

the PCs are installed with the Linux operating system and LAM/MPI, the LAM implementation (Local Area Multicomputer) [1] of the MPI standard library (Message Passing Interface) [6] for message passing among processes. Only the broadcast function was used from the MPI library, since in fact the algorithm was conceived (for this) to be possible. As the size of the systems increases, the communication cost increases as well, and one of the purposes of the experimentation is precisely to determine the communication relative cost with respect to the computation for this algorithm. Different number of computers and different sizes of equation systems were used with a dense percentage (non-null elements) of the coefficient matrix, and the performance is determined by the obtained speedup. Summarizing the experimentation:

a) Between two and five computers in parallel (which is the maximum quantity of available PCs used for the research work) were used. The best computers were always chosen for experiments, i.e. for experiments with three computers; PCs used were PC1, PC2, and PC3.

b) The sizes of the equation systems were: 2400 and 4800 equations with a density of 20%. The maximum number of equations is related to the maximum amount of data that can be stored in a computer's main memory and also with experiments running time (so as not to be excessive).

c) The speedup is computed in relation to the sequential processing without making use of any message passing primitive on the slowest computer of those used, e.g., i.e. for experiments with three computers, PCs used were PC1, PC2, and PC3 and the speedup is computed relative to PC3.

Figure 4 shows the results obtained in the experimentation for all the values previously described. In the x axis the number of computers is shown, and the speedup values obtained in the experiments is shown on the y axis. Three se-

ries of data are shown: those corresponding to sparse system sizes, 2400 and 4800, and the optimum speedup values for each number of computers.

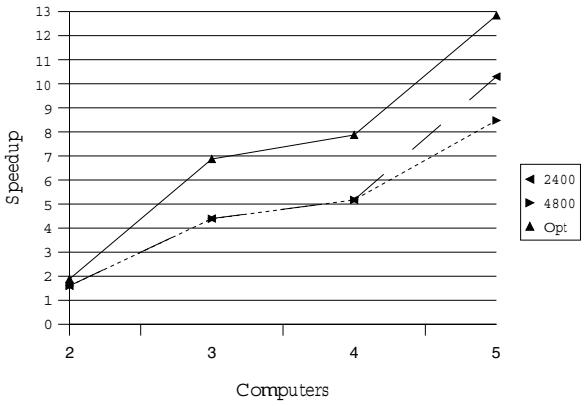

Figure 2. Performance for 20% Density.

The gap between the optimum speedup values and the obtained in the experiments finally shows the weight of the communication time/subsystem in the overall performance. Furthermore:

1) Communication time never becomes relatively greater than computing time even when the number of computers increases which, in turn, increases communication overhead. This is shown by the growth of the obtained performance as the number of computers grows.

2) Performance gain is almost the same for sparse systems of 2400 and 4800 computers, with the only exception found when using five computers.

3) It is expected to find better parallel performance if broadcasts are implemented taking advantage of the Ethernet definition, which has physical broadcast.

5. Conclusions and Further Work

A parallel algorithm of the Gauss-Seidel's method for sparse equation systems over a local area network has been presented, using the advantages of a compressed storage in order to deal with this type of systems of equations. The experimentation shows that the implementation obtains good results in parallel using up to five machines. The heterogenous cluster used in the experiments is a network of PCs of daily use, with a network of relatively low performance (Ethernet 100Mb/s).

Even when a few simple parallelization guidelines are used for design and implementation of the parallel Gauss-Seidel's method, the performance increases as the number of computers increases. Furthermore, performance is not strongly affected by the fact that broadcast messages are not optimized in the MPI implementation used. The LAM/MPI implementation does not take advantage of Ethernet broadcast to optimize broadcast messages amont processes. It is expected that for a greater number of computers, the broadcast imposes a strong performance penalization. This penalization can be avoided with specific optimization for using Ethernet and/or physical broadcast. A boradcast message specifically implemented in top of Ethernet is one of the first candidates for further work on the parallel algorithm implemented. From the algorithmical point of view, using *another* broadcast message implementation is not a major change.

The algorithm performance with higher number of computers is still to be analyzed. In this case, the scope in terms of scalability should be identified with more precision. It is clear that, in this case, experimentation is strongly dependent on the number of available computers.

Also, how the presented algorithm behaves when working with several row blocks per each processor is also still to be analyzed. On the one hand, this would increase the number of communications, but on the other, it would decrease the weight for each of them and, also, there could be a concurrence of the total number of computers.

References

[1] 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.

[2] Golub G., C. Van Loan, Matrix Computations, 2nd Edition, The John Hopkins University Press, 1989.

[3] 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.

[4] Kincaid D., W. Cheney, Numerical Analysis: Mathematics of Scientific Computing, 3rd Edition. Brooks/Cole, Pacific Grove, CA, 2002, 817 pp. ISBN 0-534-38905-8.

[5] MatLab 6.0 release 12, Users' Guide, 2000.

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

[7] Nakamura S., Applied Numerical Methods With Software, ISBN: 0130410470, Prentice Hall, 1991.

[8] Saad Y., Iterative Methods for Sparse Linear Systems 2nd edition, Society for Industrial and Applied Mathematic, 2003

[9] Tinetti F. G., Parallel Computing in Local Area Networks, PhD Thesis, Universidad Autónoma de Barcelona, Barcelona, Spain, March 2004 (in Spanish). Available at https://lidi.info.unlp.edu.ar/~fernando/publis/publi-eng.html

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