PERFORMANCE EVALUATION OF DIFFERENT COMMUNICATION SCHEMES ON A PARALLEL IMPLEMENTATION OF THE CONJUGATE GRADIENT METHOD

A. Passaro
R. Y. Tanaka
O. F. Lima
A. Muraro Jr.
N. M. Abe
angelo@ieav.cta.br
roberto@ieav.cta.br
Laboratório de Engenharia Virtual, Divisão de Física Aplicada, Instituto de Estudos Avançados (LEV/EFA/IEAv)
Rodovia dos Tamoios, km 5,5, 12228-001, São José dos Campos – SP – Brazil

A. J. Preto
S. Stephany
Laboratório Associado de Computação e Matemática Aplicada - LAC/INPE
Av. dos Astronautas, 1758, 12227-010, São José dos Campos – SP – Brazil
stephan@lac.inpe.br
airam@lac.inpe.br

Abstract. This paper presents a performance evaluation of different communication schemes applied to the parallelization of the Conjugate Gradient (CG) method. The parallel implementations employ the MPI (Message Passing Interface) communication library and are based on C++ object-oriented sequential version, previously developed at the Laboratório de Engenharia Virtual of the Instituto de Estudos Avançados (LEV/IEAv). The analyzed schemes are based either on point-to-point or collective communication. Pre-conditioning is not considered in this work in order to compare the parallel implementation with the procedures implemented in other two software packages used as standards to analyze the results presented.

Keywords: Conjugate Gradient Method, Parallel programming, Communication strategies, Performance evaluation
1. INTRODUCTION

Several engineering problems can be reduced to the solution of linear equations. Parallel versions of direct and iterative methods for the solution of large sparse systems of linear equations have been developed, mostly, to be executed on massive parallel computers with distributed memory (Murthy et al., 2001), (Yang&Brent, 2002), (Zhang&Maple, 2002), (Yang&Brent, 2003). Several options, including computation-communication overlap, were considered in order to maximize the performance on different parallel architectures. Recently, some works dealing with the parallelism of iterative methods in Beowulf clusters (set of computers interconnect by Fast Ethernet, Myrinet, or more recently, Gigabit Ethernet networks) were presented (Kumar&Kumar, 2005).

The increasing use of clusters for parallel processing requires careful cost analysis of the different communication strategies employed in the solution methods, since it represents the performance bottleneck for this kind of architecture.

The MPI (Message Passing Interface) communication library is commonly used in solver implementations. In some cases, optimized MPI implementations are developed in order to reduce communication overhead, but in general, the main solution to reduce the effect of the communication overhead is the adoption of computation-communication overlap or combining messages for several inner products (Kumar&Kumar, 2005), (Sturler&van der Vorst, 1995).

Despite the MPI library allows the use of different communication mechanisms in order to transfer data among the cluster nodes we have not found an explicit study of the impact of the communication schema on the solver performance.

This work compares the performance of different communication mechanisms used in a parallel implementation of the Conjugate Gradient (CG) method. These implementations use the MPI communication library and are derived from a C++ object-oriented sequential version developed in a previous work. The analyzed communication mechanisms are based on point-to-point communication (send/receive) and collective communication (broadcast, allgather and reduction).

2. PARALLEL CONJUGATE GRADIENT ALGORITHM

The CG method (CGM) can be seen as an iterative solution method to solve linear systems of equations,

\[ A x = b , \]  

where \( A \) is a nonsingular symmetric positive definite matrix, by minimizing quadratic functionals. The minimization takes place over certain vector spaces called Krylov spaces.

This method, as well as other CG-based methods, has widespread use because it presents high rate of convergence, is parameter free. Also, the short recurrence relation makes the execution time per iteration and the memory requirements acceptable, the roundoff errors are acceptable, and the convergence rate can be improved by using preconditioning techniques (Axelsson, 1996).

The CGM uses operations that are inherently parallel such as products between vectors and
matrices. Usually, matrix-matrix and matrix-vector products are implemented using block-algorithms (Golub & Van Loan, 1996).

The parallel algorithm adopted in this work is presented in Fig. 1 (Barret et al. 1994).

Set $r^{(0)} = b$, $p^{(0)} = (r^{(0)}, r^{(0)})$, $\beta = 0$, $p^{(-1)} = 0$, $\alpha = 0$, $x^{(-1)} = x^{(0)}$

Do

\begin{align*}
\text{itc} &= \text{itc} + 1 \\
p^{(i)} &= r^{(i)} + \beta_{i-1} p^{(i-1)} \\
q^{(i)} &= A p^{(i)} \\
\gamma &= (p^{(i)}, q^{(i)}) \\
x^{(i)} &= x^{(i-1)} + \alpha_{i-1} p^{(i-1)} \\
\alpha &= \frac{p^{(i)}}{\gamma} \\
r^{(i+1)} &= r^{(i)} - \alpha q^{(i)} \\
\rho_{i+1} &= (r^{(i+1)}, r^{(i+1)}) \\
\text{if } \rho_{i+1} < \varepsilon \text{ then} \\
x^{(i+1)} &= x^{(i)} + \alpha p^{(i)} \\
\text{quit} \\
\beta &= \rho_{i+1} / \rho_i \\
\text{while } (\text{itc} < \text{itcmax})
\end{align*}

Figure 1 - CG algorithm implemented in the parallel version.

where $(z,y)$ correspond to the inner product in $\mathbb{R}^n$, $r^k$ is the $k$th residual, $p^k$ is the $k$th search vector, which is orthogonal to the search vector of the previous iterations, $\text{itcmax}$ is the prescribed maximum number of iterations and $\varepsilon$ is the prescribed precision.

In this work, only sparse matrices generated by discretizing an elliptic partial differential equation are considered.

The matrix $A$ is partitioned in $k$ blocks of lines, where $k$ is the number of available processors. If $n$ is not multiple of $k$, the number of lines that corresponds to the remainder of the integer division $n/k$ is assigned to the last processor. Vectors are distributed among processors in the same way. In the case of the inner products, each processor performs a sequential inner product for its local blocks and the global inner product is obtained by a reduction operation using the $\text{MPI} \_\text{Allreduce}$ function.

In the case of the matrix-vector product $Ap^{(i)}$, each processor must have a copy of the entire vector $p^{(i)}$ to compute the local product on its block of $n/k$ lines.

In order to evaluate the effect of some communication strategies on the CG performance with sparse matrices, six different communication schemes were used to update the vector $p^{(i)}$ of all processors. The strategies discussed here were evaluated in a distributed memory parallel architecture composed of multiple computers connected by a network switch, a PC cluster.

Scheme 1: the point-to-point communication functions $\text{MPI} \_\text{Send}$ and $\text{MPI} \_\text{Recv}$ are applied. All processors send the data obtained from the local computations to a central processor that sum up these results in order to assemble the global vector and broadcast it using the $\text{MPI} \_\text{Bcast}$ function.

Scheme 2: an increasing ring is implemented using the $\text{MPI} \_\text{Send}$ and $\text{MPI} \_\text{Recv}$ functions.
In the logical ring, each processor has a predecessor and a successor for the communication of blocks of elements of vector $p$. In the first cycle, a processor of rank $m$ sends its block of $p$ to its successor (rank $m+1$) and receives another block from its predecessor (rank $m-1$). In the following $k-1$ cycles, each processor forwards and receives the remaining blocks that were exchanged in the previous cycles. Figure 2 illustrates the communication among the processing nodes.

![Figure 2 - Increasing ring implemented with the MPI_Send and MPI_Recv functions.](image)

**Scheme 3**: The first process executes the `MPI_Send` and `MPI_Recv`, in this order. The other processes execute these operations in the inverse order: `MPI_Recv – MPI_Send`. This scheme was named master-slave and is illustrated in Fig. 3.

![Figure 3 – Two cycles of the master-slave communication scheme.](image)
Scheme 4: the MPI_Allgatherv concatenation function is used. Each processor collects the local blocks of the other processors. In MPICH versions older than 1.2.5, this function implements a logical ring scheme, while recursive doubling is employed in the latter versions (Benson et al., 2003).

Scheme 5: A variation of schemes 2 and 3, denoted even-odd ring, employs two steps in each cycle. In the first step of the first cycle, all even-ranked processors send its block to the odd-ranked successors. In a second step, odd-ranked processors send block to the even-ranked successors. In the next cycles, these two steps are performed in a similar way for the received blocks. Figure 4 shows the communication among the processors in this scheme.

Scheme 6: In this scheme, a variation of the increasing ring (scheme 2) was implemented employing the MPI_Sendrecv function, a blocking communication which guarantees a deadlock-free exchange procedure combining both send and receive operations and determining which two processes may send/receive first (an ordering mechanism).

In all schemes presented the master computer only distributes tasks and do not participate on the computation.

In order to perform the evaluation of the communication schemes, no preconditioning technique was used in this work. An example of recent efforts on parallelizing preconditioners is presented in Kumar&Kumar (2005A). Douglas et al. (2003) presents algorithms for parallel Incomplete LU (ILU), and parallel Incomplete UL (IUL) factorization methods and preconditioned CG.
3. THE OBJECT-ORIENTED CG IMPLEMENTATION

The simplified class diagrams of the solver methods and preconditioners used in this work are presented in Figs. 5 and 6, respectively. A sparse matrix class library was used to store the coefficient matrix and to provide matrix-matrix and matrix-vector operations that fully exploits the sparsity of the matrices. These classes have been developed at LEV and compound the class library used for the development of the computer code LEVSOFT (Abe et al., 2002).

The non stationary iterative methods (CG, BiCG, CGS, and BiCGSTAB) are derived from an abstract class which defines the public interface of the derived classes and allows the application of polymorphism in the software application. The preconditioner classes follow the same scheme. The CNon_Sta_Ite_Method aggregates a preconditioner class. Notice the definition of the Null Preconditioner (CInc_Fac_Preconditioner_Null). This class allows solving the sparse equation system without the use of preconditioners and avoids the use of conditional clauses in the code. This option was adopted for the tests executed in this work.

![Figure 5 - Class diagram of the numerical methods.](image)

![Figure 6 - Class diagram of the preconditioner methods.](image)

The class library allows the storage of sparse matrices either in the CRS (compressed row storage) or in the CCS (compressed column storage) format and provides functions for the partitioning of the matrix. The solver parallelization is implemented in the scope of the classes derived from CNon_Sta_Ite_Method.
4. RESULTS

In this work, the sparse linear system of equations was generated by applying the Finite Element Method (FEM) to the solution of a heat transfer problem. A steady-state heat conduction in a square region of side 1 m and material thermal conductivity $k = 10 \text{ W/(m °C)}$ was assumed. The left, right and bottom surfaces were maintained at the temperature $T = 100 \text{ °C}$ and the upper surface at $T = 500 \text{ °C}$.

The matrix elements different from zero were stored in CRS format. No preconditioners were used in the parallel performance evaluation. However, all numerical solutions obtained with the parallel versions evaluated agree with the ones obtained with the sequential version using preconditioners – the Incomplete Cholesky (IC) and the Incomplete LU (ILU) factorizations. Non blocking communication was also implemented. As expected, the performance was similar to the obtained with blocking communication for this CG parallel algorithm.

The results and speedup presented in this work were compared to the ones obtained with two software packages extensively used by the scientific community. These two libraries are adopted as standard packages and used for performance comparison purpose.

Systems of various orders were solved in two different clusters identified as Cluster1 and Cluster2 in this work. Cluster1 was originally specified for coarse grain applications. The basic characteristics of these clusters are presented in Table 1.

Table 1 – Characteristics of the two clusters used in the experimental performance evaluation.

<table>
<thead>
<tr>
<th>Identification</th>
<th># processors</th>
<th>Processor</th>
<th>Memory</th>
<th>Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster1</td>
<td>12</td>
<td>AMD Athlon Barton XP 2500+</td>
<td>1 GB</td>
<td>Fast Ethernet (100 Mbps)</td>
</tr>
<tr>
<td>Cluster2</td>
<td>5</td>
<td>Pentium III 1.26 GHz processor</td>
<td>1 GB</td>
<td>Gigabit Ethernet (1000 Mbps)</td>
</tr>
</tbody>
</table>

In the tests performed in this work the processor used in Cluster1 was 1.6 times faster than the one of Cluster2. All evaluations were carried out under the MPICH implementation.

4.1. The standard packages

Two software packages, SuperLU_DIST (Li&Demmel, 2003) and PETSc (Balay et al., 2004), (Balay et al., 1997) were adopted as standards in order to compare the performance of the five communication schemes presented in Section 2.

The SUPERLU package allows the resolution of sparse linear systems using LU decomposition. This package is built on the BLAS package (Blackford et al., 2002), which implement efficient basic linear algebra operations, such as vector-vector operations (BLAS – LEVEL 1), matrix-vector operations (BLAS – LEVEL 2), and matrix-matrix operations (BLAS – LEVEL 3).

SuperLU_DIST is a scalable distributed direct solver for large sets of sparse linear system of equations. The solver is based on sparse Gaussian elimination with a static pivoting strategy that allows a priori determination of data structures and communication patterns and the exploitation of techniques used in parallel sparse Cholesky algorithms to better parallelize both
LU decomposition and triangular solution on large scale distributed machines.

The second package, named PETSc, is an object oriented package extensively used for computational fluid dynamics. It provides several numerical methods for the resolution of sparse linear systems, including the CG method.

Figure 7 shows the speedup obtained by using the standard packages for the solution of two sparse linear systems in Cluster1. The BLAS package installed in this cluster, which determines the performance of the SuperLU package, was optimised by using ATLAS (Whaley et al., 2000), (Whaley & Dongarra, 1999).

![Figure 7 - “Speed up” of the two standard packages for the solution of sparse linear systems on Cluster1.](image)

Figure 8 presents the estimated error in the solution obtained by applying the sequential version of CG method implemented in PETSc with and without preconditioning.

![Figure 8 - Evaluation of the solution precision with and without preconditioning. Data obtained from computations performed with the PETSc package.](image)

The precision of the solution is evaluated from a metric based on the residue of the solution:

$$\sigma = \frac{\|Ax - b\|_w}{\|A\|_w \|x\|_w \varepsilon N},$$

(2)
where: \( \| \|_{\infty} \) represents the infinite norm, \( \varepsilon \) is the machine precision, and \( N \) is the order of the matrix. The preconditioning process not only accelerates the convergence process but also reduces the numerical error associated to the solution. However, for the purpose of this work, no pre-conditioning process was adopted. The estimated errors were considered acceptable.

4.2. Performance obtained for the CG

Figures 9 and 10 show the performance results of the CG solver for each communication scheme described in Section 2 in Cluster1 and Cluster2, respectively. The figures shown the total number of processors involved, which includes the master. But processor 1, the master, does not participate in the computation.

![Figure 9 - Performance of CGM x communication scheme for the Cluster1.](image)

![Figure 10 - CGM Performance x communication scheme for the Cluster2.](image)

It can be noticed the impact that the communication network has on the performance. Modest speedups were only possible to be achieved in Cluster2. In this case, some speed up close to 2 was obtained using schemes 4 to 6. However, Figs. 7 and 9 show that the speedup obtained for the various communication schemes evaluated are comparable with the ones observed solving the system of linear equations by applying the standard packages. It can be noticed that the communication scheme adopted to spread the computation in each node of the cluster can result in considerable gain, and it has to be carefully chosen in order to achieve a better performance for applications of medium grain in this class of computer. Additionally, the
MPI_Allgatherv and the MPI_Sendrecv communication schemes present comparable performance in each cluster. The even-odd scheme presents best results for an even number of processors.

Scheme 2 resulted in deadlocks for matrices greater than 71000 and less than five processors. Typically, a system buffer area is reserved to hold data in transit which is opaque to the programmer and managed entirely by the MPI library. This buffer is a finite resource able to exist on the sending side, the receiving side, or both. However it can be easy to exhaust, resulting in deadlocks. MPI also provides for a user managed send buffer, but this option was not exercised in this work. The partial results are not presented in this work.

Figures 11 and 12 show the results obtained in Cluster2 considering communication with different block sizes: full data, blocks of 12500 elements, and blocks of 6000 elements. The curves were normalized to the best performance obtained in each case, which corresponds to blocks of 6000 elements. In Cluster1 the block size had negligible effect on the performance.
5. CONCLUSIONS

In this work, the performance of a parallel implementation of the Conjugate Gradient method was evaluated for different communication schemes. Only blocking communication functions were used. The results presented illustrate the effect of these schemes on the performance of CGM for the solution of sparse linear system of equations in two different clusters. No preconditioning was used but in all cases the results obtained agree with the ones obtained from the solutions using the sequential version of the CGM with IC and ILU preconditioning. Modest speedups were only possible to be achieved in Cluster2.

The best performances were obtained with the scheme 4 (MPI_Allgatherv), the even-odd scheme and the scheme 6 (MPI_Sendrecv). The choice of the best communication method clearly depends on the internal network of the cluster. For Cluster2 these three methods are almost equivalent, but for Cluster1 the even-odd scheme obtained a better result particularly if an even number of processors is used for the solution.

We also studied the effect of the size of data blocks used in the communication. Despite the small gain observed in the performance of CGM in Cluster2 for block size of 6000 elements (24000 bytes), this gain is small if compared with the effect of the adopted communication scheme.

REFERENCES


