The Performance of MPI Parallel Jacobi Implementation on Workstation Clusters

M. Acacio, O. Cánovas, J.M. García and P.E. López-de-Teruel

Abstract— **MPI has become the standard for implementing message-based parallel programs in distributed-memory computing environment. The goal of a parallel program is to reduce the execution time regarding the faster sequential program solving the same problem. Clusters of workstations or PCs are being considered as a good alternative to traditional parallel computers, and, currently, there are some MPI implementations for Windows NT, Solaris and Linux environments. This paper presents a study of the possibilities of MPI in these environments using the Jacobi method for solving differential equations. Results of performance tests indicate that speedups of better than $p/2$ are possible with an optimal number p of nodes on a single Ethernet bus.**

Keywords— **MPI implementations, MPI performance, Jacobi method.**

I. Introduction

THE MPI Forum states the main goal of MPI [5] as "to develop a widely used standard for writing message-passing programs. As such the interface should establish a practical, portable, efficient and flexible standard for message passing".

Recently, clusters of workstations or PCs are being used to run parallel-distributed versions of computationally intensive application programs. By running in a parallel mode on a cluster of workstations, it becomes flexible to execute larger cases of the applications that are impractical for single workstation execution. The main problem in this environment is related with the data communication and process synchronization issues.

Using MPI to code parallel programs adds an extra communication overhead. This overhead has widely been evaluated for elementary tests [10], [8], and also for complex programs [11]. Sometimes, this overhead is seen as an important difficult to make an efficient use of clusters of workstations in scientific computation.

This paper shows how MPI implementations can be used successfully with real problems. We will illustrate it using MPI implementations for Windows NT and Linux/Solaris platforms.

As we have already mentioned, portability is one of the main goals of MPI. The tests we have executed have been done using the same program, without any changes. The portability allowed us to write, debug and run our programs under a Windows NT machine and, once tests had been made, they were moved to a parallel environment of clusters of workstations. So, we made the whole work under a visual environment, with more intuitive tools and smaller development times.

Our initial results reported here show that for an optimal number of workstations, p, the speedup is, of course, not p due to communication overhead, but it is a significant value, greater than $p/2$, in many cases. Our investigations suggest various ways to improve the performance, which will make clusters of PCs a competitive approach to parallel-distributed computation for an important class of applications.

The organization of the paper is as follows. Section 2 describes the fundamentals of both the sequential and the parallel versions of the implemented algorithm. In section 3 we show the different issues we have taken into account in our experiments. The results of the evaluation and a performance analysis are presented in section 4. Finally, in section 5 several conclusions and future work are drawn.

II. Jacobi Algorithm

Many of the most widely used numerical computing techniques use large multidimensional arrays as the primary data structure. This presents an opportunity for data parallelism, as different elements of the arrays can be processed in parallel. Many of these algorithms have an iterative nature where, in each iteration, some kind of improvement is performed on the current solution, until an acceptable one is reached.

Applications include the numerical solution of sets of differential equations, which specify the dynamical behavior of large systems such as real and artificial neural networks, iterative methods for solving systems of linear equations, optimization, and other numerical problems [1]. Also, many discrete problems such as network flow, shortest path, artificial intelligence and computer science problems involving constraint satisfaction, production systems and logic programs involve iterative maps.

In this kind of algorithms there is an opportunity for parallelism within each iteration. Each process can work on a different part of the global data. However, each process should communicate, at the end of each iteration, the values produced because the rest of processes need these results for the next iteration.

Jacobi is an algorithm for solving a differential equation called Laplace's equation. An example of the application of this algorithm is to consider a body represented by a two-dimensional array of particles. This body is in contact with a fixed value of temperature on the four boundaries; all four boundaries can have different temperatures, and each particle has an initial value of temperature. To determine the final values at the internal particles of the body, one must

The authors are with the Departamento de Ingeniería y Tecnología de Computadores, Universidad de Murcia, Campus de Espinardo, s/n, 30080 Murcia (Spain). E-mail: {jmgarcia,pedroe}@dif.um.es.

solve Laplace's equation for all the internal points. With the differential equation, the temperature at each particle is the average of the temperatures at its four neighboring particles.

The Jacobi method [3] is based on the following idea. First, we have to establish some initial values for the temperature at each particle on the grid. Then, the main iteration is performed repeatedly on the data: recompute the value at each particle on the grid as the average of its four neighboring points. The values of all internal particles will change on each iteration to gradually converge to a more accurate solution in each iteration. The algorithm stops when the desired accuracy is reached.

This algorithm is amenable to parallelize because the recomputation at each point is independent. The only drawback is that each process needs the values calculated in the previous iteration. So, it is not possible to parallelize the iterations. This section describes the main particularities of both the sequential and the MPI parallel version.

A. Sequential Implementation

The analysis of this implementation is focused in the iterative loop, which concentrates most of the calculus. There are two fundamental loops in this phase. In the first loop, the new body values are calculated and stored on an intermediate matrix. In the second one, the accuracy test is performed for all points of the body. Each of these loops iterates on N^2 elements, thus, complexity could be expressed as $O(KN^2)$, where K represents the number of iterations.

B. Parallel Implementation

A row distribution has been chosen for implementing the parallel version used in this paper. Each process has M rows of the original matrix. In this approach, each process works with less elements, but some communications are needed to interchange the values of the boundaries between processes. This reduction on the number of elements assigned to every process makes the execution phase complexity $O(K(NM))$.

In the communication phase, two types of data transmission between processes are needed in order to solve the problem:

- *Neighboring rows.* The evaluation of each value is performed by using the four adjacent values in the previous iteration. Some of these values have been calculated by neighboring processes. For this communication we use the blocking primitives `MPI_Send()` and `MPI_Recv()`.
- *Accuracy test.* The convergence criterion is reached when every partition of the matrix passes this test. We perform it using MPI collective communications, to be exact, `MPI_Allreduce()`.

In the table I we can see the information about the number of bytes transmitted by each process.

TABLE I
SIZE OF COMMUNICATIONS.

Explanation	Transmitted information	Transmitted elements	Bytes per element
Neighbouring rows	Upper and lower rows	$2 * N$	(float) 4 bytes
Accuracy test	Boolean value with local result	1	(int) 4 bytes

III. CONSIDERED ISSUES

In order to analyze the utility of parallel programming in different kinds of environments using MPI, it is necessary to take into account some issues. Our analysis focuses on the next four items:

- *Communications network.* The choice of the network technology will concern the results obtained with parallel programs. We have proved both Ethernet 10 Mbps and Fast Ethernet 100 Mbps in our measures. Such networks are based in a shared medium without switching devices.
- *Operating system.* This is the basis on which the program will be executed. It provides communication primitives and transmission protocols for the used MPI implementations. Our measures have been performed on Linux, Solaris and Windows NT.
- *Message-Passing implementation.* We have used W32MPI v.0.8b [4] and MPICH v.1.0.13 [6].
- *Portability.* It is one of the MPI standard goals. Its aim is to be able to compile the same program on several platforms with another MPI implementation, without changes in the source code.

IV. EVALUATION

A. Testing Environments

We have carried out our tests on several clusters of workstations. These environments are the following ones:

- *(C1)* Cluster of Intel Pentium 200 MHz processor, 32 MB main memory, 256 KB cache memory and Fast Ethernet 3Com 905-network adapter. The operating system used is Linux 2.0.32 and the MPI implementation is MPICH v.1.0.13.
- *(C2)* Cluster of Intel Pentium 166 MHz processor, 32 MB main memory, 256 KB cache memory and Ethernet shared medium. The operating system used is Windows NT Workstation v4.0 and the MPI implementation is W32MPI v.0.8b.
- *(C3)* Cluster of Intel 486 DX4 133 MHz, 24 MB main memory, 128 KB cache memory and Ethernet shared medium. The operating system used is Linux 2.0.32 and the MPI implementation is MPICH v.1.0.13.
- *(C4)* Cluster of Sun UltraSparc 145 MHz, 32 MB main memory, 256 KB cache memory and Ethernet shared medium. The operating system used is Solaris SunOS 5.5.1 and the MPI implementation is MPICH v.1.0.13.

In the Jacobi problem, we have considered the following data:

- Boundary temperatures: left 30^oC, up 50^oC, right 30^oC and down 50^oC.
- Body temperature: 15^oC.
- Accuracy factor: 0.01^oC.

• Body dimensions: 400x400 cm, 800x800 cm and 1600x1600 cm.

The processes-processors mapping is 1:1.

B. Performance Measurements

Figures 1, 2 and 3 illustrate the results obtained in the clusters mentioned above. As we can observe, the cluster *C4* has only five machines, so there are no results for the execution with 6 computers. Furthermore, the sequential execution with a size of the body of 1600x1600 elements has only been able to be accomplished in two of the clusters. Therefore the speedups corresponding to the others two clusters have been omitted.

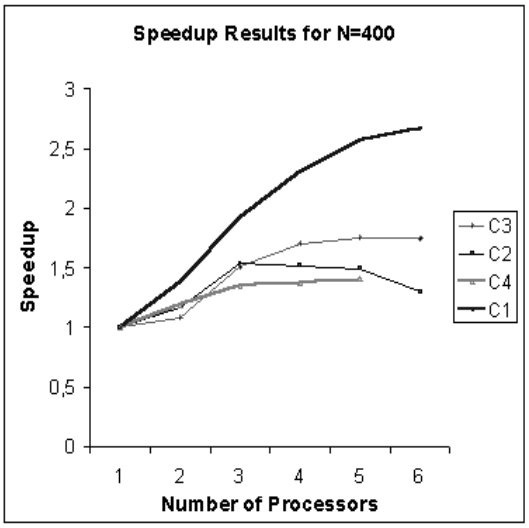

Fig. 1. Speedup reached with 400x400 size.

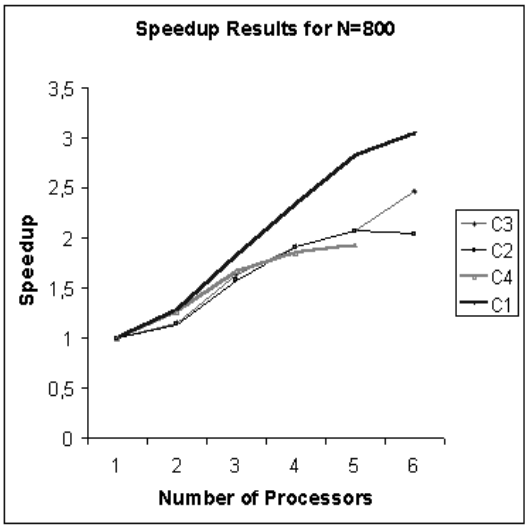

Fig. 2. Speedup reached with 800x800 size.

Figure 3 does not reflect the speedups reached with *C2* and *C3* clusters for a body size of 1600x1600 elements. This

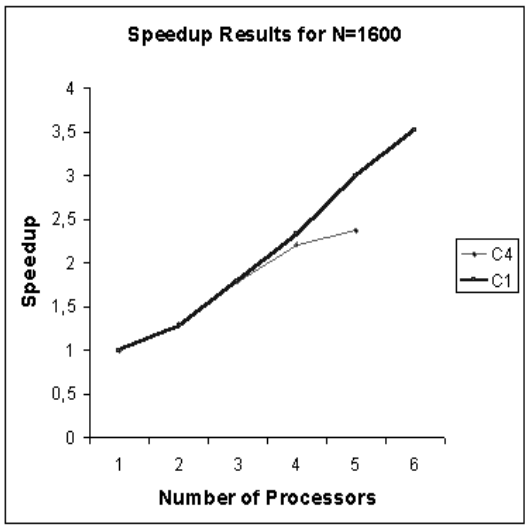

Fig. 3. Speedup reached with 1600x1600 size.

is because execution of sequential program has no enough RAM to accomplish the calculations, so it causes swapping. The needed time to solve the problem in this case is almost 48 hours, while the execution time using two machines can be accomplished in less than 30 minutes.

C. Performance Analysis

Six computers have been using as reference on calculations, because it was the maximum number of available machines for any cluster (except in the cluster *C4*). Also, the results have been represented with respect to the speedup obtained to normalize the different execution times reached in these different computers. This speedup is calculated according to the better sequential algorithm, without taking into account the initialization and completion phases of processes.

In the results obtained for the different operating systems, we can appreciate that the best performance is obtained when Linux is used, as the speedups reached with the clusters *C1* and *C3* indicate. We can observe that the cluster *C3* based on Linux behaves better than the cluster *C2*, based on Windows NT, both using an Ethernet network. Sun workstations cluster obtains the worse results.

From the point of view of the interconnection network, we can appreciate that the cluster based on Fast Ethernet always behaves better than the others do. Furthermore, this fact becomes more evident for small sizes of the problem, mainly in the 400x400 case.

As we can observe, for a given size of the problem, there exist a number of processes from which the speedup diminishes, for all the types of networks. The reason is the following: the transmitted data raise linearly with the number of processes involved, while the number of instructions that each processor executes is reduced in a $O(1/p)$ order. Anyway, the results show that for an adequate number of workstations, p, the speedup is not p due to communication

overhead, but it is a significant value, greater than $p/2$, in many cases.

V. Conclusions and Future Work

Parallel computing on clusters of workstations and PCs has very high potential, since it takes advantage of existing hardware and software. Performance tests of the implementations show that they are superior to many existing parallel programming environments for some application problems. So, clusters of workstations can be considered as a good alternative to parallel computers.

This paper illustrates how parallel programming with MPI reaches good results in some clusters available nowadays. Here, we have used the Jacobi iteration algorithm to solve the Laplace differential equation. The results show that for an optimal number of workstations p the speedup is greater than $p/2$. Our investigations suggest various ways to improve the performance, which will make clusters of PCs a competitive approach to parallel-distributed computation for an important class of applications.

Portability has been displayed as a fundamental property when we must to work with different platforms. It is one of the MPI standard goals and it has allowed that our Jacobi code could be compiled without any changes on different platforms.

Considering the performance differences between both Fast Ethernet and Ethernet networks, it is sure that Gigabit Ethernet employment will increase considerably the speedups obtained.

The realization of above tests on Mirynet and Gigabit networks, and the extension of MPI functionality to visual programming environments, such as Delphi, could be considered as future work.

References

[1] D.P. Bertsekas and J.N. Tsitsikils, *Parallel and Distributed Computation: Numerical Methods*, Prentice Hall, New York, 1989.

[2] W. Gropp et al., *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, MIT Press, 1994.

[3] B. Lester, *The art of parallel programming*, Englewood Cliffs, New Jersey, Prentice Hall, 1993.

[4] J. Meireles Marinho. *WMPI Home Page*, http://dsg.dei.uc.pt/-~fafe/w32mpi/.

[5] Message Passing Interface Forum. *MPI: A Message Passing Interface Standard*, International Journal of Supercomputer Applications and High Performance Computing, 8 (3/4), 1994.

[6] *Message Passing Interface (MPI) standard.* MPI Home Page, http://www.mcs.anl.gov/mpi.

[7] *MPICH-A Portable Implementation of MPI*, http://www.mcs.-anl.gov/Projects/mpi/mpich.

[8] N. Nupatiroj and L.M. Ni, *Performance Evaluation of Some MPI Implementations on Workstations Clusters*, Proceedings of the 1994 Scalable Parallel Libraries Conference.

[9] S. Otto et al., *MPI: The Complete Reference*, MIT Press, 1996.

[10] J. Piernas, A. Flores and J. M. García, *Analyzing the Performance of MPI in a Cluster of Workstations Based on Fast Ethernet*, 4th European PVM/MPI Users' Group Meeting, volume 1332 of Lecture Notes in Computer Science, pp 17-24, 1997.

[11] X. Wang and E.K. Blum, *Parallel Execution of Iterative Computations on Workstations Clusters*, Journal of Parallel and Distributed Computing, 34, pp 218-226, 1996.