Performance Analysis of a Hardware Solution for Collective Communication Operations on Clusters of PCs

Martha Torres[1], Sergio Takeo Kofuji[1]

[1] Laboratório de Sistemas Integravéis, Universidade de São Paulo
CEP 05508-900, São Paulo, S.P., Brasil
{mxtd, kofuji@lsi.usp.br}

Abstract—
This paper presents a performance evaluation of a hardware solution for some collective communication operations on a cluster of 8 PCs based on Myrinet and FastEthernet switches. It was implemented a modified version of TTL_PAPERS [DIE 95] for 8 nodes and the hardware collective operations were included in MPICH version 1.1 [GRO 96], which is an implementation of MPI message passing standard. The performance evaluation was done measuring the execution time of the collective operations and evaluating their impact on the total execution time of several application programs. We concluded the special hardware for collective communication operations is a suitable option to improve the performance on a parallel machine based on cluster of PCs.

Keywords— **Collective communication operations, MPI, performance evaluation, cluster of PCs**

I. Introduction

In high performance computing, the communication operations may be either point-to-point, which involve a single source and a single destination, or collectives, in which more than two processes participate. The collective operations can be used to exchange data (broadcast, gather, allgather, scatter), for global calculation (reduce, allreduce, scan) and process control (barrier synchronization). Collective operations are found in numerous parallel algorithms as sorting, graph, and search algorithms. Regarding their importance, the syntax and semantic of collective operations were included in MPI passage message standard [SNI 96].

There are a few publications about the influence of efficient solutions for collective operations on parallel machine performance. In this work, it is analyzed the performance behavior of a cluster of 8 Dual Pentium Pro using an independent hardware for some collective operations.

The platform adopted is a cluster of eight Dual Pentium Pro 200 MHz processors running Red Hat Linux 5.3 kernel 2.0.35 based on FastEthernet 100 Mb/s and Myrinet (0.64 + 0.64) Mb/s networks. The I/O interface is done by PCI bus. For Myrinet, the network switch and interface are M2F-SW8 and M2F-PCI32B, respectively. For FastEthernet, the network switch is a BAY Network 350 of 8 ports and network interface employs TULIP DC21140 chips.

The objective of this work is to quantitatively evaluate the influence of a hardware solution for collective operations on the total execution time of some parallel applications. In order to accomplish this research, the MPICH version 1.1 [GRO 96] library was used, which is an efficient implementation of MPI message passing standard. Moreover, we implemented a modified version of TTL_PAPERS [DIE 95], a hardware solution for some collective operations.

It was used two versions of the MPICH library, the first one contains efficient implementations of collective communication operations in software, and the second one includes collective operations in hardware plus those modifications.

The remainder of the paper is organized as follows. The hardware solution is described in Section 2; the measurement techniques are specified in Section 3; the application descriptions are exposed in Section 4; the performance analysis is presented in Section 5; and finally, conclusions and future researches are summarized in Section 6.

II. Description of the Hardware Solution

The built prototype is based on TTL_PAPERS [DIE 95]; PAPERS (Purdue's Adapter for Parallel Execution and Rapid Synchronization) is a hardware projected to provide barrier synchronization and other collective operations as *broadcast*, *allgather* e *allreduce*. TTL_PAPERS is a version TTL of PAPERS interconnecting 4 nodes through the parallel port.

Our prototype is an expanded version of TTL_PAPERS for 8 processors, it is done on an Altera® FPGA and the original implementation is modified with a register for collective operations. Figure 1 presents the block diagram of our prototype.

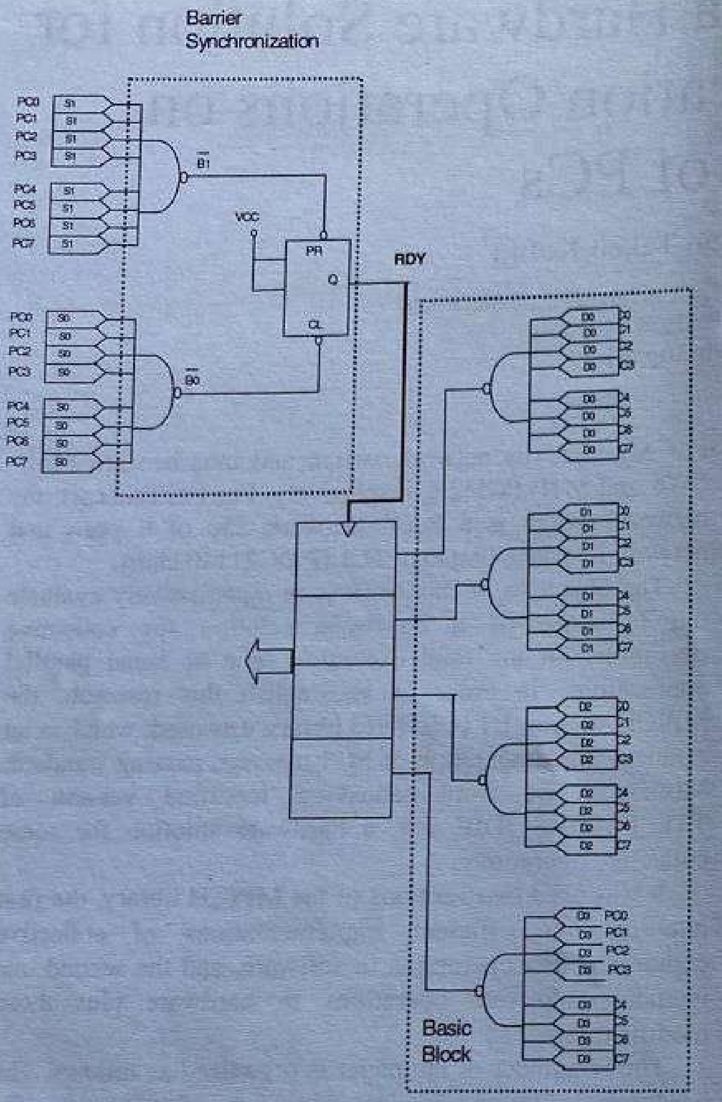


Fig.1 Block diagram of the special hardware

We added a register in the basic block that implements the collective operations for data movement. The register function is to stabilize the circuit behavior because it stores 4 bits allowing the NAND gates to be correctly read by the processing nodes.

Figure 1 also exhibits the barrier synchronization block composed by two NAND gates and a register. This configuration allows each node initializing and finishing a barrier operation in an independent form [DIE 95].

In this figure, the basic block constitutes the fundamental structure to carry out the collective operations that demand manipulation of data. The communication is accomplished by the standard parallel port, limiting the basic data size to 4 bits.

In the original implementation [DIE 95], the collective operations were executed jointly with the barrier synchronization block, so the coordination was done by software. In our implementation the coordination is done by hardware through the added register, as explained above.

It allows the execution of broadcast, allgather, allreduce and the barrier synchronization, and supports any number of nodes, whenever this number is less than or equal to 8 nodes. The figure 2 shows a photograph the prototype. The prototype was built together with the Elebra Company for the Hipersistemas Integráveis project, financed by FINEP.

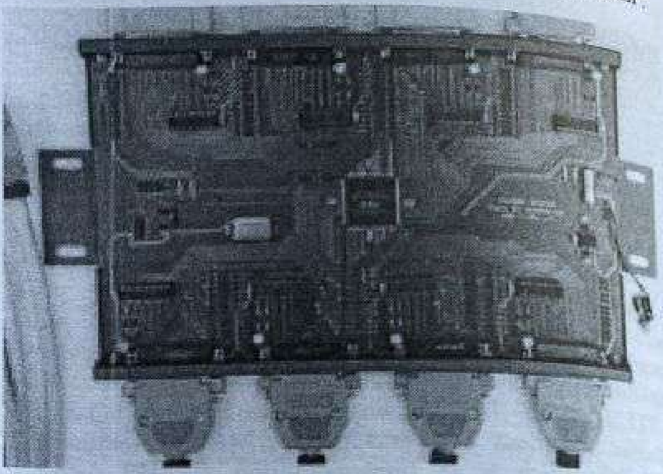


Fig.2 Special hardware prototype

The NAND gates of the basic block are responsible by the execution of *put* and *get* basic operations. These basic operations "get" and "put" data on the parallel port. The high level collective operations are based on these two operations. For example, a 4 bits broadcast operation requests the execution of a *put* basic operation by the root node and a *get* basic operation by the other nodes. A 8 bits broadcast operation requests the execution of two *put* basic operations by the root node and two *get* basic operations by the other nodes. Therefore the increment of the number of bits increases the number of basic operations.

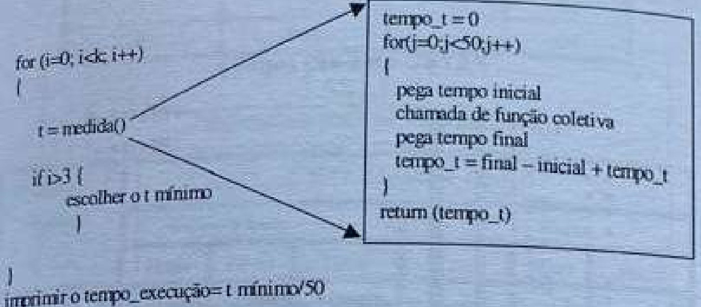
III. PERFORMANCE MEASURE TECHNIQUES DESCRIPTION

Unfortunately there are not patterns for measuring the total execution time of collective operations, each author generally adopts its own methodology.

They were analyzed in detail different methodologies to find the common points and to establish a procedure that allows measuring in an exact form the collective operations performance.

The collective operations evaluated in this paper are two types: all-to-all and one-to-all. The number of participant nodes changes in the beginning and/or the end of the execution of each collective operation class, influencing their execution time. Therefore, in this work we adopted dissimilar methods for measuring the execution time of all-to-all and one-to-all operations.

The time delay of an all-to-all operation (barrier synchronization, allreduce, allgather) is measured by the procedure showed in Figure 3. The *medida()* function consists in accumulating 50 times the operation execution time in each process involved. This accumulated time is measured 30 times ($k = 30$) and it is chosen the minimum time (thus reducing the effects of other jobs), so the minimal time is collected from all processes. To interpret the results we focus on the maximal time because it reflects all processes involved in the machine have finished the operation. This method is used for many authors for such type of collective operation [GRO 96].



```
for (i=0; i<k; i++)
{
    t = medida()

    if i>3 {
        escolher o t mínimo
        }
}
imprimir o tempo_execução= t mínimo/50
```

```
tempo_t = 0
for(j=0;j<50;j++)
{
  pega tempo inicial
  chamada de função coletiva
  pega tempo final
  tempo_t = final – inicial + tempo_t
}
return (tempo_t)
```

Fig.3 Procedure to measure the execution time of collective operations "all-to-all"

The time delay of an one-to-all operation (broadcast) is measured by the adopted procedure in [NUP 95]. The author shows to utilize traditional methodologies in this type of operations produces erroneous results on the performance. The procedure consists of centralizing the measures in the root process. This process begins the collective operation. After receiving the corresponding data, a non-root process sends a 0-byte message to the root process. The root process receives the message and marks the time of that specific process. The operation is accomplished with the all non-root processes, one each time. The execution time of the collective operation is the maximum time of the root process measures.

In all measurements, the timing starts on the third iteration in order to ensure that the connection setup time is not included.

The application programs were compiled with C GNU compiler using the optimization option –O. The MPICH library was configured for using ch_p4 device and the –nodevdebug option for excluding the overhead of debugging time. In order to collect the execution time was used the MPI_Wtime primitive, which is based on the UNIX ***gettimeofday*** system call.

IV. Description of Application Programs

The application programs employed are ScaLAPACK [DON 95] and PETSc [BAL 95].

ScaLAPACK is a library of high-performance linear algebra routines for distributed-memory message-passing MIMD computers and networks of workstations supporting PVM and/or MPI. Like LAPACK, the ScaLAPACK routines are based on block-partitioned algorithms in order to minimize the frequency of data movement between different levels of the memory hierarchy.

The library provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. The associated matrix factorizations (LU, Cholesky, QR, SVD, Schur, generalized Schur) are also provided as are related computations such as reordering of the Schur factorizations and estimating condition numbers. We used the ScaLAPACK library version 1.5.

They were chosen some example programs of ScaLAPACK for performance analysis; ***xdscaex*** is one of them. This program solves systems of linear equations for double precision numbers.

The figure 4 shows the ***xdscaex*** behavior using a 3x2 matrix for 6 nodes. The charts indicate the number of collective and point-to-point operations in the program. Also they reveal the message size of these operations by mean of the "count" and "type" arguments of MPI primitives. The "count" argument informs the number of data in the communication operations and the "type" argument informs the class of data (float, integer, byte, etc). The first chart shows the message size of the collective operations specified by the "count" and "type" arguments of MPI primitives. The second graphic shows how many collective communication operations were utilized and the number of participating nodes. The last chart shows the message size of point-to-point operations which are not part of collective operations.

Figure 4 indicates there are many point-to-point operations that are not part of collective operations, incrementing the network traffic. Also, it is observed there are more barrier synchronization operations than other collective operations. Besides, it is observed the number of participant nodes in collective operations (allreduce, broadcast and reduce) depends on the matrix dimensions.

Since the special hardware has limitations of supported messages type and size, only 35 of 90 collective operations of ***xdscaex*** program are executed there.

The Portable, Extensible Toolkit for Scientific Computation (PETSc) is a suite of data structures and routines that provide the building blocks for the implementation of large-scale application codes on parallel (and serial) computers. PETSc 2.0 uses the MPI standard for all message-passing communication.

PETSc consists of a variety of components similar to classes in C++. Each component manipulates a particular family of objects (for instance, vectors) and the operations one would like to perform on the objects. Some of the PETSc modules deal with index sets, including permutations; vectors; matrices (both sparse and dense); distributed arrays (useful for parallelizing regular grid-based problems); Krylov subspace methods;

preconditioners; nonlinear solvers; unconstrained minimization; timesteppers for solving time-dependent (nonlinear) PDEs; and graphic devices. The PETSc was developed by Argonne National Laboratory and the version adopted was 2.0.21. Each component of PETSc has example programs which are used in our performance analysis.

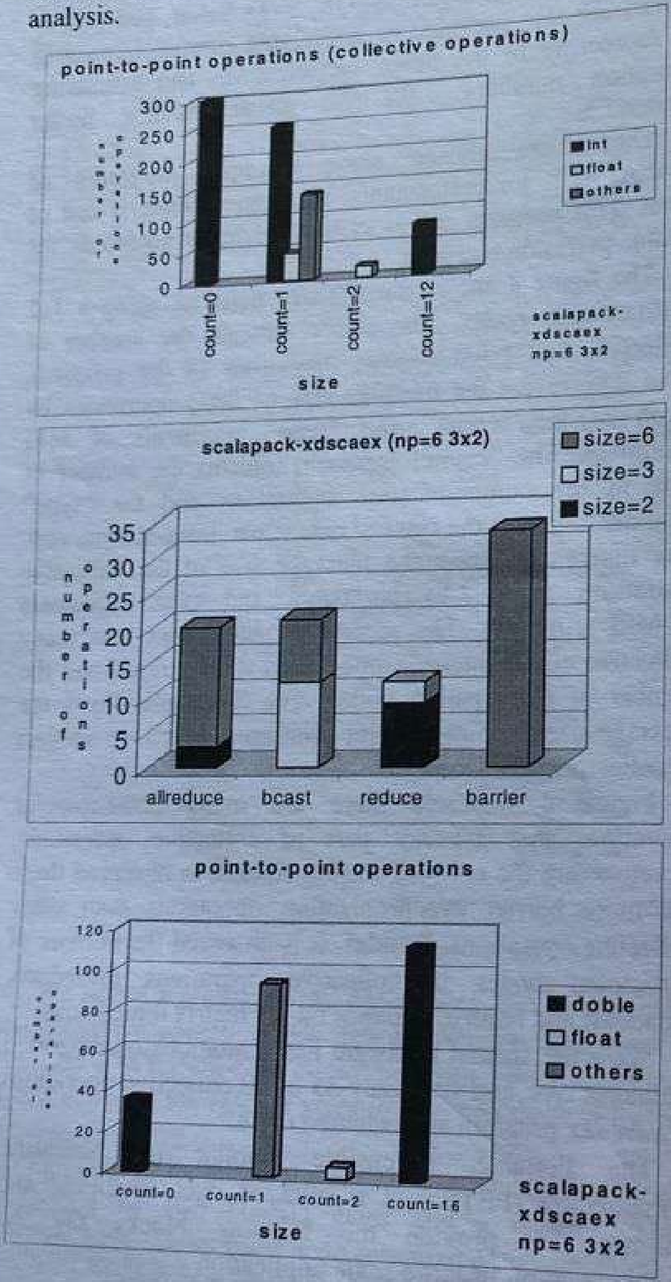


Fig.4 Characteristics of ***xdscaex*** program

Figure 5 illustrates the behavior of ***sles-ex3*** example program of PETSc. ***sles-ex3*** is part of ***sles*** component. ***sles*** is a Linear Equations Solvers and it is intended for solving nonsingular systems of the form $Ax = b$, where A denotes the matrix representation of a linear operator, b is the right-hand-side vector, and x is the solution vector.

Each example program is represented by two charts. The first one shows how many collective communication operations are executed and the number of participating nodes. The second chart shows the message length specified by the "count" and "type" arguments of MPI primitives. The PETSc example programs have only collective communication operations; they do not have point to point operations.

Figure 5 indicates Allreduce is the collective operation more frequently used. Also, it illustrates a large amount of messages are small and integers.

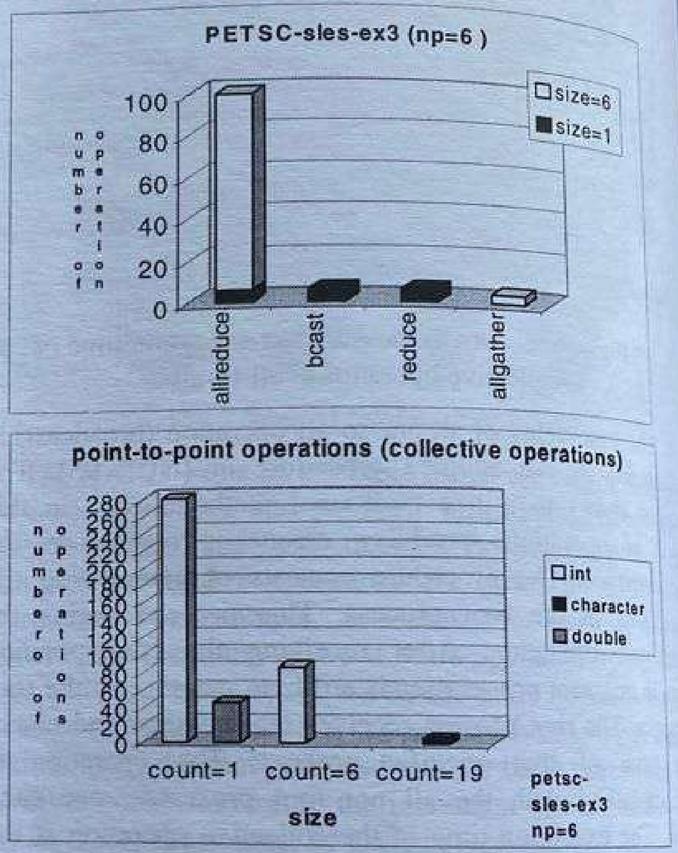


Fig.5 Characteristics of ***sles-ex3*** program

V. PERFORMANCE RESULTS

The performance evaluation was done measuring the execution time of collective operations and evaluating their impact on the total execution time of several application programs. Unfortunately, the Myrinet switch has only available up to six ports, therefore the Myrinet measures are limited to 6 nodes. The tables I to IV present a comparison of execution time of collective operations in hardware and efficient software. Fast, Myri and Hard refer to FastEthernet, Myrinet e Hardware, respectively.

The results allow concluding the hardware solution has some restrictions. The special hardware uses a parallel port as interface with date size of 4 bits and a basic collective operation based on NAND logic; these facts limit the execution time and number of date types supported. It presents a low performance when the message length is increased, then causing a poor bandwidth.

TABLE I
Execution time of Barrier Synchronization

Number of Nodes	MPI_Barrier (μs)		
	FastEthernet	Myrinet	Hardware
4	598	811	24
5	903	1086	24
6	964	1308	24
7	1086		24
8	981		24

TABLE II
Execution time of *Allrreduce*

Number of Nodes	MPI_Allreduce (ms)					
	1 byte			16 bytes		
	Fast	Myri	Hard	Fast	Myri	Hard
4	0.59	0.7	0.067	0.6	0.8	0.8
5	0.96	1.2	0.065	0.98	1.2	0.8
6	1.13	1.4	0.1	1.16	1.6	1.35
7	1.14		0.12	1.16		1.63
8	1.08		0.14	1.1		1.91

TABLE III
Execution time of *Allgather*

Number of Nodes	MPI_Allgather (ms)					
	1 byte			16 bytes		
	Fast	Myri	Hard	Fast	Myri	Hard
4	0.81	1.21	0.095	0.84	1.2	0.82
5	0.99	1.41	0.11	1.02	1.5	1.07
6	1.0	1.55	0.13	1.05	1.5	1.38
7	1.19		0.15	1.22		1.66
8	1.01		0.16	1.04		2.04

TABLE IV
Execution time of *Broadcast*

Number of Nodes	MPI_Broadcast (ms)					
	1 byte			100 bytes		
	Fast	Myri	Hard	Fast	Myri	Hard
4	0.58	0.79	0.035	0.66	0.83	0.99
5	1.23	1.39	0.037	1.0	1.4	1.09
6	1.2	1.6	0.032	1.21	1.66	1.07
7	1.38		0.037	1.18		1.08
8	0.98		0.035	0.97		1.32

A broadcast operation in hardware is efficient for smaller messages 90 bytes. Allreduce and Allgather operations are efficient for smaller messages 9 bytes. The hardware solution supports only traditional data types, data like MPI_PACKED are not supported. The performance characteristics of a hardware solution can be improved using another type of interface as a PCI bus, since it will increase the bandwidth.

Next, the impact of hardware solution on total execution time of some parallel applications is shown. In Figures 6 and 7, Ótimo refers to MPICH implementation with efficient solutions in software for *scatter*, *gather*, reduce, *broadcast*, *allreduce*, *allgather* and barrier synchronization, whereas, Ótimo+HB refers to preceding implementation including the special hardware for *broadcast*, *allgather*, *allreduce* and barrier synchronization. The time is normalized as the ratio of the program execution time to the best execution time.

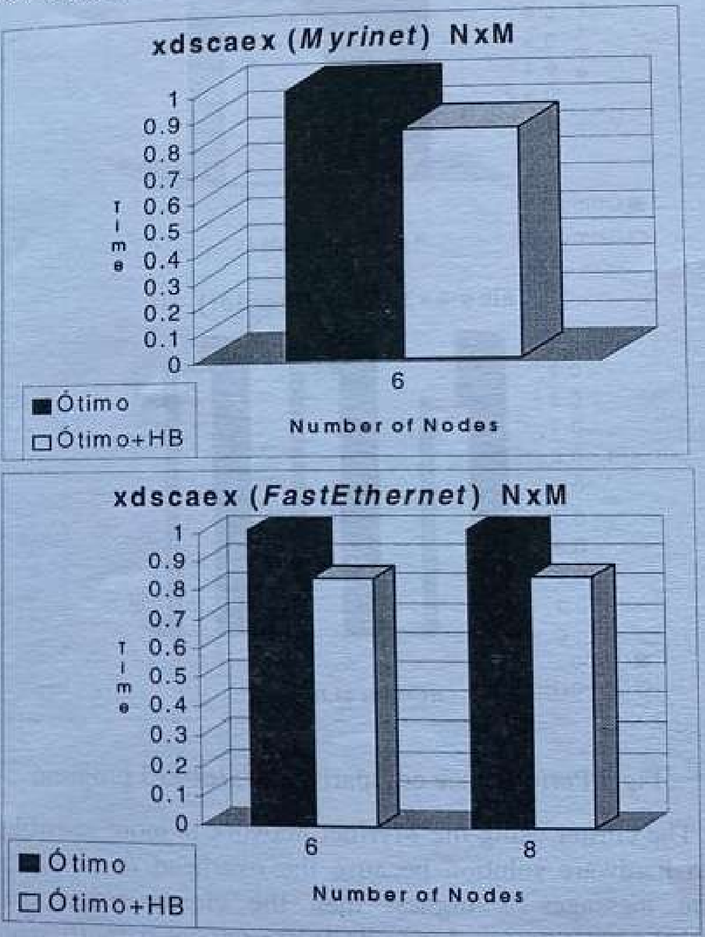


Fig.6 Performance comparison of ***xdscaex*** program

Figure 6 exhibits there is a performance gain for the hardware solution. In both of the cases, Myrinet and FastEthernet switches, the program execution time decreases approximately 20% with the special hardware. Although the special hardware does not execute all collective operations of example programs, as mentioned in section IV: no more than 35 of 90 collective operations are realized by hardware special.

Figure 7 presents the performance results of ***slex-ex3*** program. The program execution time decreases by 45% with the hardware solution for a 6 nodes cluster based on Myrinet switch. In a cluster based on FastEthernet switch, the program execution time decreases by 25%.

For the *slex-ex3* program, as shown in Figure5, the number of allgather and allreduce operations is greater than other collective operations, therefore the efficient execution of these operations reduces the total execution time. The special hardware executes 90% of *allreduce*, *allgather* e *broadcast* operations and consequently it presents a good performance.

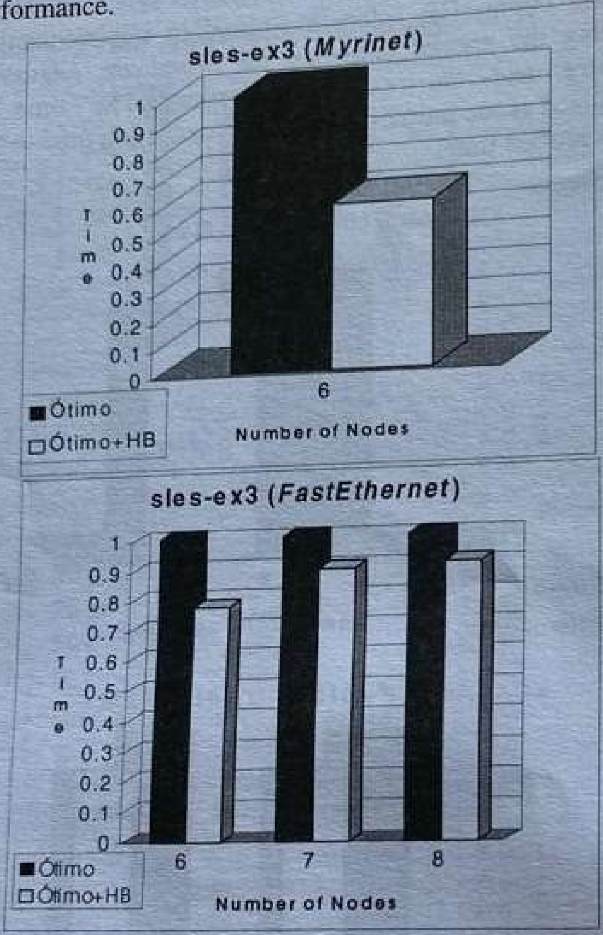


Fig.7 Performance comparison of *sles-ex3* program

The cluster using the Myrinet network is more sensible to a hardware solution because the overhead of point to point messages is highest than the cluster using the FastEthernet network due to TCP/IP protocol is inefficient on Myrinet interface.

VI. CONCLUSIONS

This work quantitatively evaluates the influence of a hardware solution for collective operations on the total execution time of some parallel applications.

In this article shows is important to define a measurement technique for each collective operation to obtain a correct execution time. Moreover, it is necessary to study the application behavior for performance analysis.

Though the implemented hardware solution present some limitations, mainly by its bandwidth, the performance gain obtained for the application programs shows hardware solution decreases the total execution time.

In coming performance analysis will be considered the two processors of the DUAL Pentium node.

In the future, it will be implemented a hardware solution that supports all collective operations and date types of MPI standard with a minimum execution time. It will be possible employing a different interface instead of the parallel port.

ACKNOWLEDGMENTS

The authors thank CNPq and FINEP for infrastructure and financial support.

REFERENCES

[BAL 95] BALAY, S. et al. PETSc 2.0 User Manual. Mathematics and Computer Science Division, Argonne National Laboratory, 1995 (http://www.mcs.anl.gov/petsc/petsc.html).

[DIE 95] DIETZ, H. G. et al. Purdue's Adapter for Parallel Execution and Rapid Synchronization: the TTL_PAPERS Design. West Lafayette, School of Electrical Engineering Purdue University, Jan. 1995 (Technical Report). (http://garage.ecn.purdue.edu/~papers)

[DON 95] DONGARRA, J.J; WALKER, D.W. Software Libraries for Linear Algebra Computations on High Performance Computers. *SIAM Review*. v. 37, n. 2, p. 151-80, June 1995.

[GRO 96] GROPP, W. et. al. A high-performance implementation of the MPI message passing interface standard. *Parallel Computing*. v. 22, p. 789-828, 1996.

[NUP 95] NUPAIROJ, N.; NI, L.M. Benchmarking of multicast communication services. East Lansing, Department of Computer Science Michingan State University, Apr. 1995. (Technical Report MSU-CPS_ACS-103).

[SNI 96] SNIR, M. et al. *MPI: the complete reference*. London: The MIT Press, 1996.