Validation of Dimemas communication model for MPI collective operations*

Sergi Girona[1], Jesús Labarta[1], and Rosa M. Badia[1]

[1] CEPBA-UPC, Mòdul D6 – Campus Nord, c/ Jordi Girona 1 - 3,
08034 Barcelona, Spain
{sergi, jesus}@cepba.upc.es, rosab@ac.upc.es

Abstract. This paper presents an extension of Dimemas to enable accurate performance prediction of message passing applications with collective communication primitives. The main contribution is a simple model for collective communication operations that can be user-parameterized. The experiments performed with a set of MPI benchmarks demonstrate the utility of the model.

1 Introduction

Dimemas [5] has been previously used for performance prediction of message passing programs. In applications were communications are mainly point-to-point it has been demonstrated that is a valuable tool [4]. The next step is to prove its utility for collective operations. It is necessary to develop a collective communication model, as the point-to-point model based on the latency and bandwidth is insufficient.

The second goal is to prove the validity of the tool for point-to-point and collective communications when using communication intensive benchmarks. The results obtained in communication intensive benchmarks will demonstrate the correctness of the models, as they stress the communication.

The paper is organized as follows: section 2 reviews related work. Section 3 presents Dimemas simulator and its point-to-point communication model. The collective operation model is presented in subsection 3.1. Section 4 reports the experiments and results obtained. Finally, in section 5, some conclusions are presented.

2 Related Work

In [2] the LogP, a model of a distributed memory multiprocessor in which processors communicate by point-to-point messages, is presented. The model specifies the performance characteristics of the interconnection network but does not describe its structure. The model is based on the following parameters: L, latency or delay to

* This work has been supported by the Ministry of Education of Spain under contract CICYT TIC 98-0511 and by the European Commission under Esprit Project 26276, SEP-Tools

transmit a message that contains a word; o, overhead, length of time that a processor is engaged in the transmission or reception of a message; g, gap, minimum interval between two messages; and P, number of processors. These parameters are not equally important in all situations, and it is possible to ignore one or more parameters depending on the application.

In [8] the authors present the APACHE system, a performance prediction tool for PVM programs. The performance model they use makes difference between computation time and communication time. All nodes are considered to be homogeneous. Their approach is divided in three phases. In the first phase, the compiler constructs a call graph of the PVM program and creates an instrumented version. In the dynamic analysis phase, the instrumented PVM program is executed. It generates a set of equations. With this information and some parameters the prediction phase evaluates the equations and obtains a performance time prediction for the program.

In [10] the authors present a comparison between the performance of collective communication primitives in different systems. The results of these experiments are stored as a database, and are used for performance evaluation. The response time of parallel programs is decomposed in local computation part (LP) and communication part (CP). LP time is predicted by running a program that consists of the local computation part of the program being studied. CP time is derived from the performance database of the communication primitives.

In [7] a parallel simulator for performance evaluation of MPI programs is presented. This simulator uses direct execution to obtain computation time of programs. One of the drawbacks of this system is that host and target processors should be similar to obtain accurate results. Communication and I/O times are obtained by simulation. MPI calls to the MPI library are changed by calls to a library of the simulator, MPI-SIM. Presented results show prediction errors between 5 and 20%.

In [1] the authors present an approach similar to Dimemas. As Dimemas, a trace file obtained from traced execution of the parallel program on a platform different from the one to be evaluated is used as input to a simulator. Previous to this trace execution, a static analysis step is performed. As a result of this step, only one iteration of communication patterns present on the loops appears on the trace. The simulator is oriented to heterogeneous computing environments, and is obtained less accuracy if is used for performance prediction of Massively Parallel Processor systems.

3 Dimemas

Dimemas [5] is a performance prediction tool for message passing programs. It is a trace driven simulator that rebuilds the behavior of a parallel program from a trace file and some parameters of the target architecture. The input trace file characterizes the application. Initially it was developed with the aim of studying the effects of time sharing message-passing programs among several applications [3].

Besides summarized performance data, Dimemas can generate trace files that can be viewed with Vampir [11] and Paraver [5]. Combining a trace driven simulator

such as Dimemas with a visualization tool helps understanding the summarized statistics. The user can analyze sensitivity of its program to architectural parameters without modifying the source code and run it again. In a similar way, the effect in global application behavior of a potential improvement in a routine can be observed.

Other significant target in the design of Dimemas is that it should be possible to obtain trace files in a "normal/typical" development environment. By "normal/typical" development environment we understand a single workstation or a time-shared, throughput oriented, parallel machine. Dimemas allows obtaining trace files for performance analysis of message passing programs in one of such environments, without needing a dedicated parallel platform. From this point of view, Dimemas is a tool that avoids the nasty effects of time sharing in trace-based visualization of parallel programs behavior [11]. Dimemas input trace files for MPI programs are generated by VAMPIRtrace, an instrumented MPI library and API [11].

Dimemas models the target architecture (the simulated machine) as a network of nodes. Each node is an SMP connected to the network with a set of links and buses. Every node is composed of one or more processors and local memory.

The model of the target architecture is defined by several parameters: number of nodes, number of processors per node, network bandwidth, communication latency, number of inputs and output links, number of buses, etc. On rebuilding the parallel program execution, Dimemas differentiates between point to point communications and collective communications. Point to point communication time is modeled as:

$$T = L + \frac{S}{B} \quad (\mathbf{1})$$

where L is the latency, S the size of the message and B the bandwidth. This formula can be applied in a network without contention, with an unlimited number of resources (buses and links). To model the bisection bandwidth of the system, a maximum number of available buses (defined by the user) are considered by Dimemas. Also, to model the injection mechanism, a number of input and output links between the nodes and the network can be defined. Half-duplex link can also be specified.

3.1 A communication model for collective MPI operations

Many collective operations have two phases: a first one, where some information is collected (fan in) and a second one, where the result is distributed (fan out). Thus, for each collective operation, communication time can be evaluated as:

$$T = FAN\_IN + FAN\_OUT \quad (\mathbf{2})$$

FAN_IN time is calculated as follows:

$$FAN\_IN = \left(L + \frac{SIZE\_IN}{B} \right) \times MODEL\_IN\_FACTOR \quad (\mathbf{3})$$

Depending on the scalability model of the fan in phase, the parameter MODEL_IN_FACTOR can take the following values:

Table 1. MODEL_IN_FACTOR possible values

MODEL_IN	MODEL_IN_FACTOR	
0	0	Non existent phase
CT	1	Constant time phase
LIN	P	Linear time phase, P = number of processors
LOG	Nsteps	Logarithmic time phase

In case of a logarithmic model, MODEL_IN_FACTOR is evaluated as the *Nsteps* parameter. *Nsteps* is evaluated as follows: initially, to model a logarithmic behavior, we will have $\lceil log_2\ P \rceil$ phases. Also, the model wants to take into account network contention. In a tree-structured communication, several communications are performed in parallel in each phase. If there are more parallel communications than available buses, several steps will be required in the phase. For example, if in one phase 8 communications are going to take place and only 5 buses are available, we will need $\lceil 8/5 \rceil$ steps. In general we will need $\lceil C/B \rceil$ steps for each phase, being C the number of simultaneous communications in the phase and B the number of available buses. Thus, if $steps_i$ is the number of steps needed in phase *i*, Nsteps can be evaluated as follows:

$$Nsteps = \sum_{i=1}^{\lceil \log_2 P \rceil} steps_i \qquad (4)$$

For FAN_OUT phases, the same formulas are applied, changing SIZE_IN by SIZE_OUT. SIZE_IN and SIZE_OUT can be:

Table 2. Options for SIZE_IN and SIZE_OUT

MAX	Maximum of the message sizes sent/received by root
MIN	Minimum of the message sizes sent/received by root
MEAN	Average of the message sizes sent and received by root
2*MAX	Twice the maximum of the message sizes sent/received by root
S+R	Sum of the size sent and received root

4 Model Validation

To validate the communication model presented in previous section, several experiments were performed. The experiments were done in 64 processors SGI Origin from CEPBA-UPC with a set of micro-benchmarks that intensively stress some of the MPI communication primitives [6]. Each benchmark was run with dedicated resources and the dedicated elapsed time (DET) was measured. Also, an input trace

file for Dimemas for each benchmark was obtained by running them in a loaded system.

The method we follow for the validation of the model is based on the execution of the simulator with different parameters. For all experiments we used ST-ORM, a tool for stochastic optimization, to help us in the specification, execution and analysis of the different experiments [9].

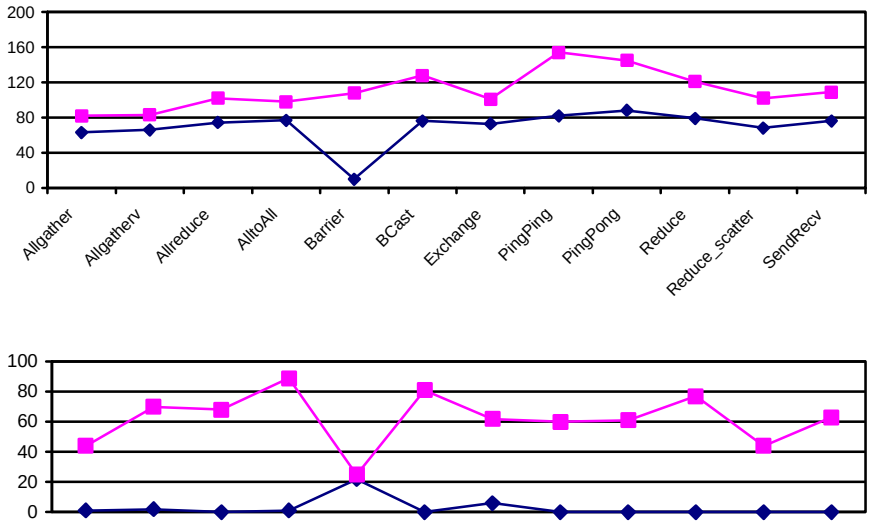

Fig. 1. Up: range of bandwidths (Mbytes/s) that lead to a predicted time between the 10% of error of the dedicated elapsed time; down: Range of latencies (μs) that lead to a predicted time between the 10% of error of the dedicated elapsed time

4.1 General parameters

A configuration file is used to model the behavior of collective MPI primitives. As we do not know implementation details of the used MPI library, a first set of experiments was performed to fix this file. We found out that a reasonable model would be:

Table 3. Parameters used to model collective operations

Id_op	*MODEL_IN*	*SIZE_IN*	*MODEL_OUT*	*SIZE_OUT*	*Colletive operation*
0	LIN	MAX	LIN	MAX	/* MPI_Barrier */
1	LOG	MAX	0	MAX	/* MPI_Bcast */
2	LOG	MEAN	0	MAX	/* MPI_Gather */
3	LOG	MEAN	0	MAX	/* MPI_Gatherv */
4	0	MAX	LOG	MEAN	/* MPI_Scatter */
5	0	MAX	LOG	MEAN	/* MPI_Scatterv */
6	LOG	MEAN	LOG	MEAN	/* MPI_Allgather */
7	LOG	MEAN	LOG	MEAN	/* MPI_Allgatherv */

8	LOG	MEAN	LOG	MAX	/* MPI_Alltoall */
9	LOG	MEAN	LOG	MAX	/* MPI_Alltoallv */
10	LOG	2MAX	0	MAX	/* MPI_Reduce */
11	LOG	2MAX	LOG	MAX	/* MPI_Allreduce */
12	LOG	2MAX	LOG	MIN	/* MPI_Reduce_Scatter */
13	LOG	MAX	LOG	MAX	/* MPI_Scan */

From all collective communication primitives, only MPI_Barrier shows linear behavior. This benchmark was executed changing the number of processors. The results obtained show how the execution time grows linearly with the number of processors. This can be explained by the fact that being the used MPI implementation based in shared memory all processors have to update sequentially a fixed memory position.

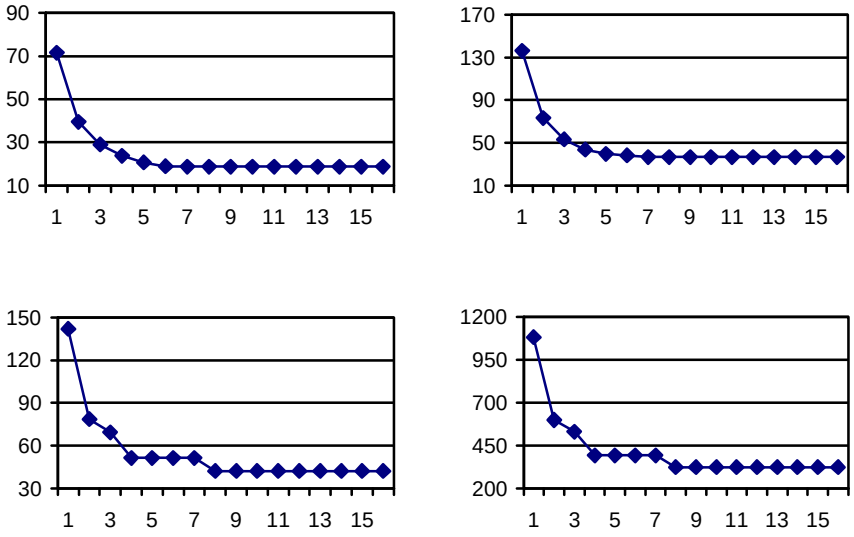

Fig. 2. Influence of the number of buses on the predicted time (in seconds) for the SendRecv (upper-left), Exchange (upper-right), Reduce_scatter (down-left) and Allgather (down-right) benchmarks (BW=87.5, L=25, 1 link HD)

Also this preliminary part of the experiments have shown that setting the number of links of each node to one half-duplex link models better the system than if one full-duplex link is considered. This is also due to the shared memory MPI implementation, where each processor can only be involved in one transfer at a time.

4.2 System parameters optimization

A first set of experiments was performed to evaluate the influence of the latency and bandwidth. For each benchmark a set of simulations was performed (between 70 and 110) varying the latency and bandwidth parameters.

For these simulations, the number of buses was set to 10. This number approximates 0.6·P, being P the number of processors (P=16 for our case). The value 0.6·P is an approximation to the maximum bandwidth of a crossbar network. The number of links was set to one half-duplex link. For each simulation a predicted (PT) time was obtained. Figure 1 shows the range of bandwidths (in Mbytes/second) such that PT has less than 10% error respect DET and the range of latencies (in µseconds) that complies the same previous condition. From these results we can be concluded that, for example, a bandwidth of 80 Mbytes/s and a latency of 25µs can be used for performance prediction.

A second set of experiments was performed to evaluate the effects of the number of buses. In this case, we set the bandwidth to 87.5Mbytes/s, the latency to 25µs and modeled the connection of the nodes to the network with one half-duplex link, while the number of buses is the parameter that varies. The results obtained for PingPing and PingPong show that these benchmarks are not influenced by the network contention. Predicted time does not change significantly with the number of buses.

Figure 2 (up) shows the results for this experiment for the Exchange and SendRecv benchmarks. We can see that in those cases the predicted time suffers great variation depending on the number of defined buses. As the measured DET for SendRecv is 18,2 secs and for Exchange is 37,4 secs we can conclude that any value between 7 and 16 for the number of buses will model correctly the network contention.

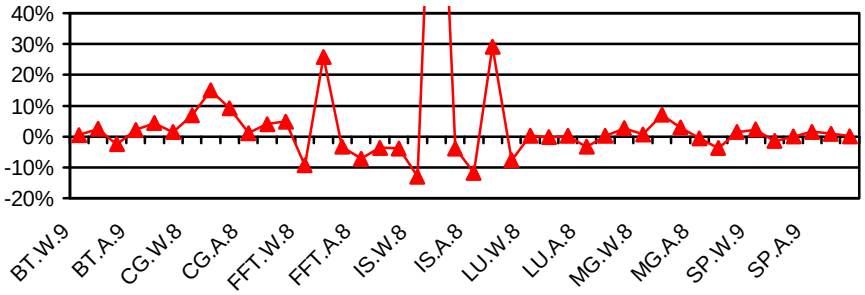

Fig. 3. Error (%) in prediction when simulating with BW=87,5, L=25, 1 HD link, 16 buses

For all collective operations, even MPI_Barrier, the results of the experiment were similar to those obtained for PingPing and PingPong examples, with gaps in most of cases when the number of buses is a power of 2. In figure 2 (down) we can see the results obtained for Allgather and Reduce_scatter benchmarks. Also, the defined number of buses influences the predicted time. As the DET for the Allgather benchmark is 385.1 secs, and for Reduce_scatter is 44.0 secs we can conclude that any value between 8 and 16 can be used to model bus contention for these cases (the same result was obtained for the remaining collective operations).

Given that the bandwidth of the machine we used is wide enough, we can model bus contention with high values. Probably, if a computer with a lower bandwidth had been used, we would have to model bus contention with lower values.

Finally, to validate the correctness of the parameters obtained in the previous experiments, a third set of experiments was performed. For this series of experiments, we run Dimemas for all NAS benchmarks (classes A and B, number of tasks 8-9, 16, 25-32). Figure 3 shows the percentage of error obtained. Most of the benchmarks are predicted with less than a 10% of error. The point out of the graphic takes the value 150% error. This error and those over 10% refer to really short executions (less than five seconds). Thus the real difference between execution and prediction is negligible.

5 Conclusions

In this paper we have presented an approach that takes into account the difference between collective and non-collective message passing primitives. A simple but accurate formulation for the prediction of communication time invested by collective operations has been defined. This formulation has been included in Dimemas.

The experiments developed by using an MPI implementation and communication intensive benchmarks show the validity of Dimemas for performance prediction of message passing programs.

References

1. R. Aversa, B. Di Martino, and N. Mazzocca, “Reducing Parallel Program Simulation Complexity by Static Analysis”, Proc. of PDPTA´99, Las Vegas, USA, June 1999.
2. D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, and T. von Eicken, “LogP: Towards a Realistic Model of Parallel Computation”, in Proc. of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming”, May 1993.
3. S. Girona, T. Cortes and J. Labarta , “Analyzing scheduling policies using DIMEMAS”, Environments and Tools for Parallel Scientific Computing III, Faverges de la Tour, France, August 1996
4. S. Girona and J. Labarta, “Sensitivity of Performance Prediction of Message Passing Programs”, Proc. of PDPTA´99, Las Vegas (USA), June 1999.
5. J. Labarta, S. Girona, V. Pillet, T. Cortes and L. Gregoris , “DiP: A Parallel Program Development Environment”, Euro-Par'96, Lyon, France, August 1996
6. Pallas MPI Benchmark – PMB, Part MPI – 1. Revision 2.1, Pallas, 1998.
7. S. Prakash, and R. L. Bagrodia, “MPI-SIM: Using Parallel Simulation to Evaluate MPI Programs”, Proc. of the 1998 Winter Simulation Conference, pp. 467-474, 1998.
8. M.R. Steed, and M.J. Clement, “Performance Prediction of PVM Programs”, in Proc. of IPPS´96, pp. 803-807, 1996.
9. ST-ORM web site, http://www.cepba.upc.es/ST-ORM/

10. Y. Tanaka, K. Kubota, M. Matsuda, M. Sato, and S. Sekiguchi, “A Comparison of Data Parallel Collective Communication Performance and its application”, Proc. of the HPC Asia 97, pp. 137 144, 1997.
“Vampir 2.0 Visualization and Analysis of MPI Programs”, http://www.pallas.