A Strategy for Point-to-point Node Communication Performance Prediction in Cluster Environments*

Kuan-Ching Li (李冠憬)
Hsiao-Hsi Wang (王孝熙)

Dept. of Computer Science and Information Management
Providence University
Taiwan ROC
E-mail: {kuancli | hhwang}@pu.edu.tw

Jean-Luc Gaudiot

Dept. of Electrical Engineering and Computer Science
Univ. of California-Irvine
USA
E-mail: gaudiot@uci.edu

Abstract

In this paper, we present and apply a methodology for parallel programming, along with MPI performance measurement and prediction in a class of distributed computing environments, named networks of workstations (NOW). Our approach is based on a two-level model where, at the top, a new parallel version of timing graph representation is used to explicit the parallel communication and code segments of a given parallel program. At the bottom level, analytical models are developed to represent execution behavior of parallel communications and code segments. Obtained execution time, together with problem size and number of nodes are input to the model, which allows us to predict the performance of similar cluster computing systems with a different number of nodes. We validate our analytical model by performing experiments over homogeneous cluster of workstations. Results show that our approach produces accurate predictions, within 6% of actual results.

Keywords. *Communication modeling, analytical modeling, MPI parallel program, cluster computing*

1. Introduction

Advances in networking, high-end computers and middleware capabilities in recent years have resulted in new computing infrastructures called networks of workstations (NOW), or PC-based clusters. The potential of this computing infrastructure has attracted attention from the computing industry, since this technology depends solely on commodity components. Furthermore, they have been widely used to improve the performance of applications with intensive demands for computational power. In merely a few years, computer clusters have become one of the most convenient and cost-effective tools for solving many complex computational problems such as the *Grand Challenges* [14]. These problems are fundamental in science and engineering with broad scientific and economic impact, whose solution can definitely be advanced by high-performance computing. The popularity of NOWs is due to their scalability, their ability to provide significant cost effective computing, to rely on commodity technology, and to efficiently support both single processor interactive processing and large batch parallel processing.

The workstations are typically interconnected through a high-speed network, such as Gigabit Ethernet, SCI, or Myrinet, and they run commodity operating systems, such as Microsoft Windows or Linux. Many software tools have been developed to support distributed computing over a network of workstations, including popular tools such *Parallel Virtual Machine* (PVM) [15] software and *Message Passing Interface* (MPI) [16, 17, 18], and low-level communication mechanisms, such as Active Messages [20] or Fast Messages [19].

In any performance prediction methodology or software tool, a high-level abstraction of an application plays an important role. Based on the distributed programming

* This research is supported by the National Science Foundation/USA under grants no. CSA-0073527 and INT-9815742, National Science Council/Taiwan under grants no. NSC92-2213-E-126-006 and NSC92-2213-E-126-012. Special thanks to SMC Networks/USA and Accton/Taiwan.

paradigm used in MPI, PVM and other programming systems, we defined a new class of timing graphs, which we call Distributed Processing Graph (*DP*Graph*), introduced in [10, 21]. *DP*Graph* is designed based upon previous works in [3, 4, 5, 6, 7, 9, 12, 13]. The objective of this class of timing graphs is to describe the parallel executions as well as the communication and synchronization relationships of the parallel computations. To separately quantify the effects of the program structure and those of the system, the communication and synchronization points are independently identified in the graph of the application.

In this paper, we applied a methodology for performance measurement and prediction of parallel programs to study MPI point-to-point communication primitives. Our proposed methodology provides an integrated interface that binds performance and analysis back to the original source code, allowing users to estimate the execution time of the execution under excellent bounds. Also, this should afford a better understanding and investigation of parallel program structure, performance and behavior.

The remainder of this paper is organized as follows. Some related work is briefly discussed in section 2, followed by the description of the methodology in section 3. In section 4, we present experiments and results of performance measurements and predictions. Finally, we give some remarks and conclusions in section 5.

2. Related work

A number of performance evaluation and prediction research projects are known. These include algorithms, techniques and projects, and can be recognized as iterative algorithms [13], analytical approaches [3, 4, 7], trace transformation, symbolic performance modeling [5, 6], or adaptive sampling statistics techniques [9, 12]. However, these techniques and algorithmic approaches are not well suited for general studies of interactions between PC-based cluster systems performance and parallel programming with MPI.

H. W. Cain, B. P. Miller and B. J. Wylie [2] have introduced strategies for performance diagnosis, G. Karypis and V. Kumar [8] introduced analysis techniques for multilevel graph partitioning, while P. Puschner and A. Schedl [11] introduced an analytical technique to analyze program execution times. Timing graphs are used to describe the sequence of execution of a program code. The computation of MAXTs (maximum execution time) is mapped to a graph problem, a generalization of maximum cost circulation calculus of a directed graph.

3. Methodology

The methodology introduced by Li in [10] eases performance analysis and prediction of parallel programs implemented with message passing interface, executed in a homogeneous network of workstations environment (Figure 1). The methodology basically entails the definition of an extension for *T-Graphs* (timing graphs), which we name ***DP*Graphs***, a class of graphs from which we can represent not only sequential programs, but also parallel programs instrumented with communication and synchronization. Moreover, new strategies are defined for the performance measurement and prediction of parallel applications described by a parallel programming language, for this class of distributed computing system [10]. Analytical models are developed using experimental execution times, allowing us to accomplish performance analysis and to predict execution times.

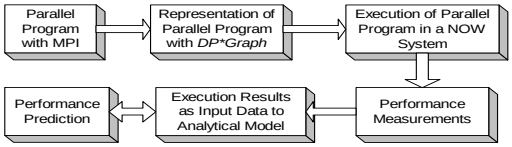


Figure 1. Methodology scheme.

Once we obtained the graph representation of the parallel program and its analytical model, it is possible to proceed with experimental evaluations and studies of the performance prediction, based on the experimental data obtained previously.

3.1. Program representation with DP*Graph

Using DP*Graph elements (Figure 2), we can represent correctly the synchronization and communication stages in a message passing program. Also, it's possible to study the execution flow and analyze the structure of the parallel program. The representation of parallel programs with MPI (or any other message passing interface) can be worked out as show in Figure 3.

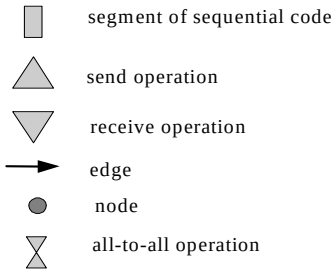


Figure 2. Graph elements for representation of parallel programs.

3.2. Execution time calculus

To obtain the execution time of a parallel application, it is needed to evaluate the execution time of all running processes in each processing node of a parallel computer system. The process of calculating the total execution time of a parallel program with MPI is introduced in Figure 3.

The execution time of a parallel program is taken as the maximum execution time among all processing nodes, each of them obtained through the sum of the partial execution times t_i.

$$T_{exec} = \max (\sum t^1_i, \sum t^2_j, \sum t^3_k, \ldots, \sum t^n_t)$$

where $\sum t^n_t$ stands for: sum of all partial times t_t of the n-th processing node.

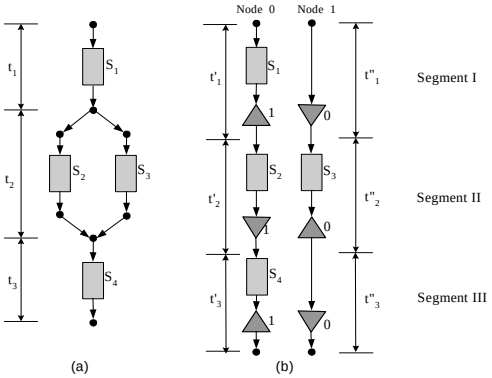


Figure 3. Example of execution time calculation process.

Figure 3 shows the graph representations of a parallel application; Figure 3(a) is the graph representation of a parallel application, in execution process view, while figure 3(b) shows in detail the execution of the parallel application, considering the execution of each process in each processing node. The total execution time for this parallel application can be given as:

$$T_{exec} = \max (t'_1 + t'_2 + t'_3, t''_1 + t''_2 + t''_3)$$

3.3. Communication operation modeling

The time spent in communication is an important factor to be considered in the study and analysis of parallel applications designed to run on clusters of workstations. Considering a message with n elements, this time can be decomposed into the following components [10]:

- $t_e(n)$ = time spent to transfer the message from memory to network buffer;
- $t_t(n)$ = time spent to transfer the message by the network between two nodes;
- $t_r(n)$ = time spent to receive the message from the network buffer.

These components can be best viewed in the conceptual model presented in Figure 4.

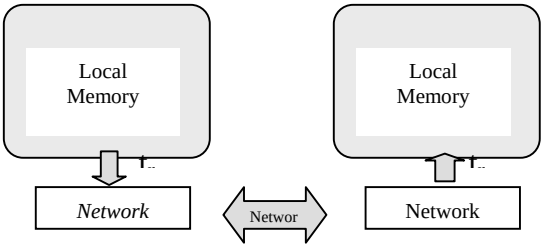


Figure 4. The communication time components involved in a message transfer.

In order to construct a proper and consistent model, it is important to identify the factors that may influence the communication performance. Among these many factors that may influence the time spent in the communication, this work considers two of them represented by the following constants [10]:

- k_l, the network latency;
- k_b, its bandwidth.

In order to construct a proper and consistent model, it is important to identify the factors that may influence the communication performance. Among these many factors that may influence the time spent in the communication, this work considers two of them represented by the following constants [10]:

- k_l, the network latency;
- k_b, its bandwidth.

Thus, the components of communication time viewed above can be represented by these equations:

$t_e(n) = n * c1$

$t_t(n) = k_b * n + k_l$

$t_r(n) = n * c2$, where c1 and c2 are constants.

From these considerations, the communication time of a message with n elements is given by:

$t_c(n) = t_e(n) + t_t(n) + t_r(n)$

$t_c(n) = n*c1 + k_b*n + k_l + n*c2$

$t_c(n) = (c1+ k_b+c2)*n + k_l$

$t_c(n) = c*n + k_l$, where c is a constant

4. Experiments

The example program used to perform the tests was implemented in C with some communication primitives of the message-passing interface. This program was designed to run with exactly two processes: a sender and a receiver. Each of these processes runs on one node of the clusters described in section 4.1.

The first process runs a sequential code, sends a message to the second process, runs another segment of sequential code and then terminates. The second process, named receiver,

is similar to the first process, but it receives a message from the sender instead of sending a message. In this parallel MPI program, a message is an array of elements of a given type, and the type used in our experiments was MPI_INT. Figure 5 shows a graph representation of the program, according to the DP*Graph representation introduced in previous section.

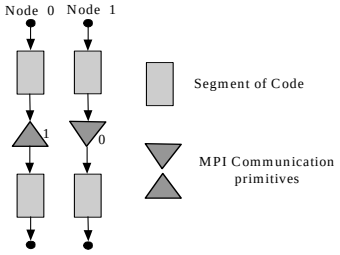


Figure 5. Representation of the program used in this experiment.

4.1. Experimental setup

To validate our methodology, tests have been done on two clusters of workstations with the follow characteristics:

	Cluster 1	**Cluster 2**
# Nodes	16	8
Processor	Celeron 433 MHz	Pentium4 1.6 GHz
Memory	128 MB	384 MB
Network	Fast-Ethernet	Fast-Ethernet
OS	Linux Red Hat 6.2	Linux Red Hat 7.2
MPI version	LAM 6.4	MPICH 1.2.1

Table 1. Clusters used in experiments

4.2. Experimental results

This section presents the results of the tests performed on two homogeneous PC-based clusters described previously. The tests with the four send modes (standard, Buffered, Ready and Synchronous) were performed, considering the following message sizes (# of integers): 100,000, 250,000, 400,000, 550,000, 700,000, 850,000 and 1,000,000. The average times (in seconds) of these experiments are presented in Table 2, where t_{sender} and $t_{receiver}$ represent the time spent respectively by the process sender and the process receiver.

Primary analysis indicates that all processes present a linear increase in their communication time. For instance, considering the synchronous mode executed on cluster 1, the average time for the sender process presented an increase by approximately 0.062 as each new message length introduced. In fact, this behavior can be best observed in Figure 6.

However both clusters described in section 4.1 use the same network type (Fast-Ethernet), the time spent by all processes in each send mode on cluster 2 was smaller than on cluster 1 (as we can see in Table 2). This fact may be explained by the greater influence of overhead on communication performance than network latency. Cluster 2 has better configuration than cluster 1, considering issues as processor and memory, so it presented smaller overhead and faster communication times.

As noted in section 3, the communication time can be expressed by $t_c(n) = c*n + k_1$. From the average time of the tests presented in Table 2, some equations were developed to represent the behavior of the send modes analyzed. An interpolation method was used to construct these equations, which form the analytical model to estimate the communication time (Table 3).

Figure 6 shows the communication time to cluster 1 in relation to the message length, considering each send mode. The lines t_{pre_sender} and $t_{pre\_receiver}$ represent an interpolation of the points which correspond to the sender and receiver communication times respectively.

Despite the almost linear behavior presented by t_{pre_sender} and $t_{pre\_receiver}$, the results point the performance of the send modes considering message lengths. When other lengths, much larger or smaller than these values, are used, factors such as operating system overhead or bandwidth may highly influence the performance communication. It may therefore lead to erratic communication times.

The results from cluster 2 have presented a similar behavior, as we can see analyzing the values presented in Table 2.

Thousands of integers		100	250	400	550	700	850	1000
St.	T_{sender}	0.0385	0.0990	0.1611	0.2223	0.2839	0.3463	0.4076
	$T_{receiver}$	0.0484	0.1178	0.1881	0.2588	0.3294	0.4007	0.4723
B.	T_{sender}	0.0067	0.0168	0.0266	0.0375	0.0485	0.0578	0.0690
	$T_{receiver}$	0.0484	0.1178	0.1876	0.2592	0.3308	0.3989	0.4707
R.	T_{sender}	0.0401	0.1025	0.1660	0.2288	0.2918	0.3551	0.4174
	$T_{receiver}$	0.0499	0.1213	0.1926	0.2639	0.3351	0.4062	0.4777
Sy.	T_{sender}	0.0397	0.1017	0.1649	0.2272	0.2898	0.3529	0.4148
	$T_{receiver}$	0.0503	0.1234	0.1964	0.2689	0.3415	0.4148	0.4890

(a) Cluster 1

Thousands of integers		100	250	400	550	700	850	1000
St.	T_{sender}	0.0336	0.0846	0.1355	0.1866	0.2376	0.2886	0.3394
	$T_{receiver}$	0.0367	0.0911	0.1457	0.2001	0.2549	0.3091	0.3639
B.	T_{sender}	0.0019	0.0045	0.0070	0.0099	0.0125	0.0153	0.0179
	$T_{receiver}$	0.0385	0.0956	0.1526	0.2101	0.2674	0.3250	0.3812
R.	T_{sender}	0.0338	0.0846	0.1355	0.1865	0.2376	0.2885	0.3396
	$T_{receiver}$	0.0367	0.0910	0.1456	0.2003	0.2548	0.3088	0.3633
Sy.	T_{sender}	0.0338	0.0846	0.1355	0.1866	0.2375	0.2886	0.3395
	$T_{receiver}$	0.0367	0.0911	0.1456	0.2001	0.2546	0.3091	0.3636

(b) Cluster 2

Table 2. Experimental results for cluster 1 and cluster 2 introduced.

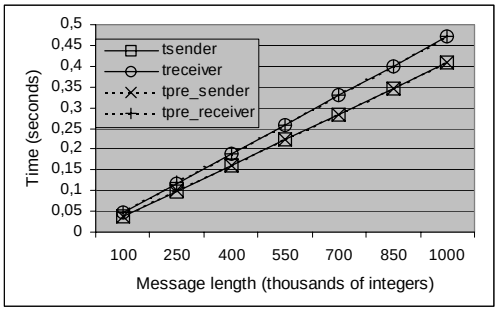
	Cluster 1	Cluster 2
$t_{ss}(n)$	4.10604E-07*n - 0.0031578	0,000339883*n - 0,0003657
$t_r(n)$	4.7111E-07*n + 0.0002422	0,000363441*n + 0,0002894

(a) Standard Send

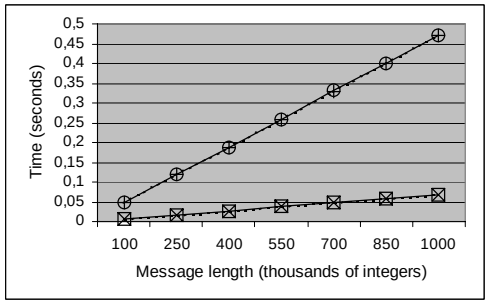
	Cluster 1	Cluster 2
$t_{bs}(n)$	6.92293E-08*n - 0.0005128	0,0000179555*n - 0,0000086
$t_r(n)$	4.69539E-07*n + 0.0008028	0,000381352*n + 0,0003167

(b) Buffered Send

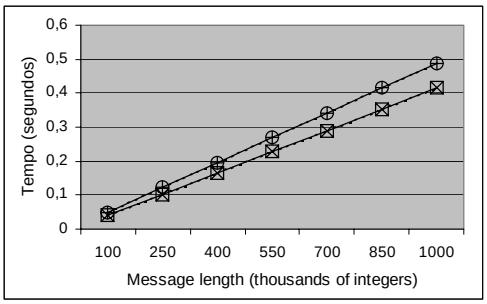
Table 3. Analytical models for *Send* modes.



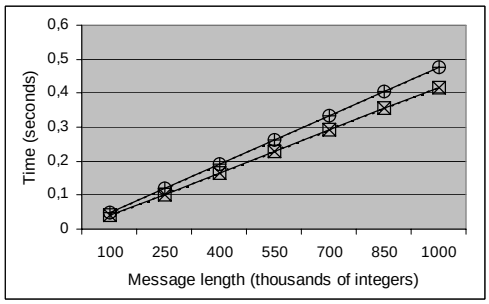
(a) Standard Send



(b) Buffered Send



(c) Synchronous Send



(d) Ready Send

Figure 6. Communication time variation related to message length - cluster 1.

where: $t_{ss}(n)$ is the time spent to send a message with n elements with a standard send;
$t_{bs}(n)$ is the time spent to send a message with n elements with a buffered send;
$t_{ys}(n)$ is the time spent to send a message with n elements with a synchronous send;
$t_{rs}(n)$ is the time spent to send a message with n elements with a ready send;
$t_r(n)$ is the time to receive a message with n elements in one of the send modes.

4.3. Performance prediction

Let us test the analytical models described in the previous section on the program examples described in section 4.2. The tests were re-executed with new message lengths and the obtained results to each send modes are presented in Table 4. The results show that the methodology produced accurate models: the error of the prediction studies change from 0.06% to 16%.

After these initial tests, we applied later our methodology on a more complex example, the benchmark program IS (Integer Sort) / NPB (NASA Parallel Benchmarks). This benchmark can be described as its main goal to sort a given set of numbers in parallel. All experiments were done using problem sizes A and B. More information about this and other NAS benchmark programs can be found at [1].

Size (thousands of integers)		40	60	1600	1700
Standard	Measure	0.0124	0.0224	0.6521	0.6927
	Predict	0.0133	0.0215	0.6538	0.6949
	error(%)	6.846	-4.0179	0.2684	0.3129
Buffered	Measure	0.0027	0.0042	0.109	0.1162
	Predict	0.0023	0.0036	0.1103	0.1172
	error(%)	-16.009	-14.286	1.1494	0.8741
Synchronous	Measure	0.0129	0.0224	0.6659	0.7074
	Predict	0.0145	0.0215	0.6654	0.7072
	error(%)	11.704	-4.0179	-0.0631	-0.0341
Ready	Measure	0.0131	0.0235	0.6693	0.711
	Predict	0.0147	0.0231	0.6696	0.7116
	error(%)	12.135	-1.7021	0.0496	0.0831

(a) Cluster 1.

Size (thousands of integers)		80	1200	1400	2000
Standard	Measure	0.0267	0.4089	0.4762	0.6799
	Predict	0.0268	0.4075	0.4755	0.6794
	error(%)	0.3007	-0.3356	-0.1538	-0.0774
Buffered	Measure	0.0015	0.0215	0.0252	0.0361
	Predict	0.0014	0.0215	0.0251	0.0359
	error(%)	-6.9756	0.3246	-0.4698	-0.4567
Synchronous	Measure	0.0267	0.4075	0.4761	0.6809
	Predict	0.0269	0.4074	0.4754	0.6792
	error(%)	0.5843	-0.0163	-0.1582	-0.2498
Ready	Measure	0.0266	0.4074	0.4761	0.6808
	Predict	0.0268	0.4075	0.4755	0.6794
	error(%)	0.7208	0.0136	-0.1404	-0.208

(b) Cluster 2.

Figure 7. Predicted versus measured results for IS/NPB benchmark program.

Figure 7 shows execution results of the program IS. The curve named `Real_Exec_Time` brings us results from experimental executions of the benchmark program executed in the cluster 1.

A second curve, named `Pred_Exec_Time`, shows results obtained from the models elaborated with our methodology applied to IS parallel program. After analyzing these results, the largest difference among the data checked for same points were less than 6%.

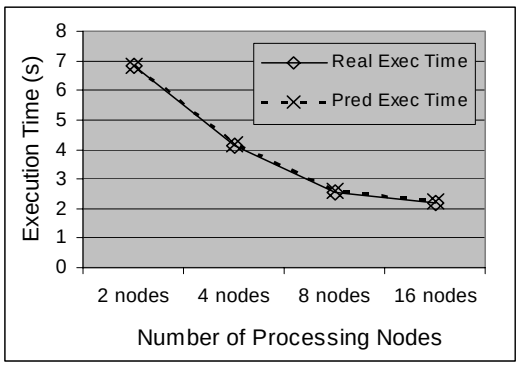


Figure 7. Predicted versus measured results for IS/NPB benchmark program.

5. Conclusion

In this paper, a methodology for the performance analysis and prediction of parallel programs is applied. A new graph representation for parallel programs was defined, mainly representation issues regarding on communication operations. Concurrently, analytical models are constructed to represent the behavior of the communication operations. The accuracy of the methodology introduced was confirmed by experimental tests realized on two different clusters with the verification of the predicted and measured results.

As a next step in this research, new studies about performance analysis and prediction about factors that may contribute to improve communication overhead will be done. Also, studies will be done on other interconnection networks, such as Gigabit Ethernet, SCI, Myrinet and ATM.

Heterogeneous cluster systems are more popular today than ever, since it is easy to connect a computer system into an existing cluster computing system. In our research investigation, once a graph representation of a parallel application is mapped, load balancing can be applied for task distribution, to minimize the total execution time.

There is a high demand for parallel program analysis tools and, at the same time, a need for tools to study and analyze those applications that demand high performance. Nowadays, from a cost/benefit point of view, PC-based cluster systems are an excellent way to access supercomputing.

Acknowledgements

Special thanks to SMC Networks / USA and Accton / Taiwan, who contributed with a generous equipment donation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation/USA or National Science Council/Taiwan.

Also, the authors wuld like to thank the

anonymous reviewers for suggestions that led to improvements in the presentation of this paper.

References

1. D. H. Bailey, J. T. Barton, T. A. Lasinski and H. D. Simon. The NAS parallel benchmarks. Tech. Report NASA memorandum 103863, NASA Ames Research Center, July 1993.
2. H.W. Cain, B.P. Miller and B.J.Wylie. "A callgraph-based search strategy for automated performance diagnosis". In: Proceedings of the Euro-Par 2000, Munich, Germany, 2000.
3. M. E. Crovella. Performance Prediction and Tuning of Parallel Programs. Ph.D. thesis, Department of Computer Science, University of Rochester, 1994.
4. T. Fahringer. Automatic performance prediction for parallel programs on massively parallel computers. PhD thesis, Tehnischen Universität Wien, Vienna, 1993.
5. A.J.C. van Gemund. Performance modeling of parallel systems. PhD thesis, Delft University of Technology, Delft University Press, ISBN 90-407-1326-X, 1996.
6. A.J.C. van Gemund. Compile-time performance prediction of parallel systems. In: Computer Performance Evaluation: Modeling Techniques and Tools (Tools´95), LNCS 977, 1995, pp. 299-313.
7. P.G. Harrison, N.M. Patel. Performance modeling of communication networks and computer architectures. Addison-Wesley, 1993.
8. G. Karypis, V. Kumar. Analysis of multilevel graph partitioning. Technical report 98-037, University of Minnesota, 1998.
9. J. Landrum, J. Hardwick and Q.F. Stout. "Predicting algorithm performance". Computing Science and Statistics, 30, 1998, pp. 309-314.
10. K.C. Li. Performance measurement and prediction of parallel programs on network of workstations. Ph.D. thesis, Department of Computer Engineering and Digital Systems, University of São Paulo, Brazil, 2001.
11. P. Puschner, A. Schedl. "Computing Maximum Task Execution Times – a graph-Based Approach", Journal of Real-Time Systems, vol. 13, no.1, 1997, pp. 67-91.
12. E. Strohmaier. Statistical performance modeling: case study of the NPB 2.1 results. Technical report UTK-CS-97-354, Dept. of CS, University of Tennessee, Knoxville, 1997.
13. D.F.Vrsalovic, D.P. Siewiorek, Z.Z. Segall and E.F. Gehringer. "Performance prediction and calibration for a class of multiprocessors". IEEE Transactions on Computers, v. 37, n. 11, 1988, pp. 1353-1364.
14. A. Beguelin and J. Dongarra, Solving computational Grand Challenges using a network of heterogeneous supercomputers, Proceedings of 5th SIAM Conference on Parallel Processing, 1991.
15. G. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam, PVM: Parallel Virtual Machine – a user's guide and tutorial for networked parallel computing. MIT press, 1994.
16. Argonne National Laboratory. MPICH: a portable implementation of MPI, 2003. http://www-unix.mcs.anl.gov/mpi/mpich.
17. W. Gropp, E. Lusk and A. Skjellum, Using MPI: portable parallel programming with the Message Passing Interface, MPI Press, Cambridge, MA, 1994.
18. M. Snir and W. Gropp, MPI: The complete reference, MIT press, 2nd edition, 1998.
19. M. Lauria and A. Chien, MPI-FM: high performance MPI on workstation clusters, Journal of Parallel and Distributed Computing, 1997.
20. S. S. Lumetta, A. M. Mainwaring and D. E. Culler, Multi-Protocol active messages on a cluster of SMP's, Proceedings of SC'97, San Jose, 1997.
21. K.C. Li, J-L. Gaudiot and L.M. Sato, Performance measurement and prediction of parallel programs for NOW environments using P3MP, in NPDPA´2002 IASTED International Conference on Networks, Parallel and Distributed Processing, and Applications, Tsukuba, Japan, 2002. (Published by IEEE Computer Society)