John von Neumann Institute for Computing

Analytical Performance Models of Parallel Programs in Clusters

Diego R. Martínez, Vicente Blanco, Marcos Boullón,
José Carlos Cabaleiro, Tomás F. Pena

published in

Parallel Computing: Architectures, Algorithms and Applications ,
C. Bischof, M. Bücker, P. Gibbon, G.R. Joubert, T. Lippert, B. Mohr,
F. Peters (Eds.),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **38**, ISBN 978-3-9810843-4-4, pp. 99-106, 2007.

http://www.fz-juelich.de/nic-series/volume38

Analytical Performance Models of Parallel Programs in Clusters

Diego R. Martínez[1], **Vicente Blanco**[2], **Marcos Boullón**[1], **José Carlos Cabaleiro**[1], **and Tomás F. Pena**[1]

[1] Dept. of Electronics and Computer Science
University of Santiago de Compostela, Spain
E-mail: {diegorm, marcos, caba, tomas}@dec.usc.es

[2] Dept. of Statistics and Computer Science
La Laguna University, Spain
E-mail: vicente.blanco@ull.es

This paper presents a framework based on an user driven methodology to obtain analytical models on parallel systems and, in particular, clusters. This framework consists of two interconnected stages. In the first one, the analyst instruments the source code and some performance parameters are monitored. In the second one, the monitored data are used to obtain an analytical model using statistical processes. The main functionalities added to the analysis stage include an automatic fit process that provides accurate performance models and the automatic data collection from monitoring. Some examples are used to show the automatic fit process. The accuracy of the models is compared with a complexity study of the selected examples.

1 Introduction

Performance prediction is important in achieving efficient execution of parallel programs. Understanding performance is important not only for improving efficiency of applications, but also for guiding enhancements to parallel architectures and parallel programming environments. As systems become more complex, as in the case of multiprocessor environments or distributed systems, accurate performance estimation becomes a more complex process due to the increased number of factors that affect the execution of an application. In addition to relatively static information, there are many dynamic parameters that must be taken into account in the performance estimation. These dynamic parameters may not be known until run time[10].

Performance models can be grouped into three categories: analytical modelling, simulation modelling, and measurement[18]. Analytical model evaluation takes a little time since it is based on solutions to mathematical equations. However, it has been less successful in practice for predicting detailed quantitative information about program performance due to the assumptions and simplifications built in the model. Simulation modelling constructs a reproduction, not only of the behaviour of the modeled system, but also its structure and organization. Simulation model should be more accurate than analytical models but it is more expensive and time consuming, and can be unaffordable for large systems. Measurement methods permits to identify bottlenecks on a real parallel system. This approach is often expensive because it requires special purpose hardware and software, and in real system they are not always available. Performance measurement can be highly accurate when a correct instrumentation design is carried out for a target machine.

Although analytical approaches are generally less accurate than simulation approaches, they are faster and more efficient since the behaviour is described through mathematical equations. Moreover, analytical modelling provides an abstract view of the hardware and software. Analytical modelling is a common issue in performance evaluation of parallel systems: parallel performance analytical models based on scalar parameters, like BSP[16], LogP[6] or LogGP[1], are widely used to evaluating parallel systems; the information of analytical models can be useful to optimize load balancing strategies[11]; Bosque et. al. propose a heterogeneous parallel computational model based on the LogGP model[4]. Some approaches combine empirical experimentation, analytical modelling and perhaps some light-weight forms of simulation[5,7,15]. The experimentation and simulation are used to characterize the application while the analytical model is used to predict the performance behaviour in terms of the characterization of the algorithm and the knowledge of the platform. This approach is getting relevance in Grid environments[2,9]. Its use is important not only for achieving efficient execution of parallel programs, but also for taking advantage of its qualities. For example, a quick evaluation of the performance behaviour of an application is desirable in a scheduler in order to obtain an efficient use of the computational resources of the grid[8,19].

The proposed framework uses both measurement and analytical modelling and provides an easy to use tool that allows the analyst to obtain analytical models to characterize the performance of parallel applications in clusters. The framework helps the analyst in the process of tuning parallel applications and automatically performs some tedious processes related to performance analysis. Its modular design makes it possible to introduce new tools in the framework, or even to substitute current functionalities, with a minimal impact on the rest of the framework. Analytical models are automatically obtained by a statistical analysis of measurements from real parallel executions. Therefore, the behaviour of an application can be characterized in terms of parameters such as the problem size, the number of processes, the effective network capacity, etc.

In this paper the methodology used to obtain analytical models from instrumentation data is described in detail, and some examples of its use are illustrated. Section 2 introduces the proposed framework based on two stages. The second stage where analytical models are obtained is described in detail in Section 3, and selected examples are used in Section 4 to show its use. Finally, Section 5 presents the main conclusions of this work and future developments.

2 The Modelling Framework

A methodology to obtain analytical performance models of MPI applications is introduced in this section. Fig. 1 shows a scheme of the whole framework that consists of two stages. The first stage is devoted to instrumentation, where information about the performance of the execution of the parallel application is monitored. The second stage uses this information to obtain an analytical model by means of statistical analysis to characterize the performance of the parallel code. This stage will be explained detailed in Section 3.

The instrumentation stage is based on CALL[3], which is a profiling tool for interacting with the code in an easy, simple and direct way. It controls external monitoring tools through the so called CALL drivers, that implement an application performance interface to those external tools. Therefore, CALL is a modular tool and new drivers can be developed

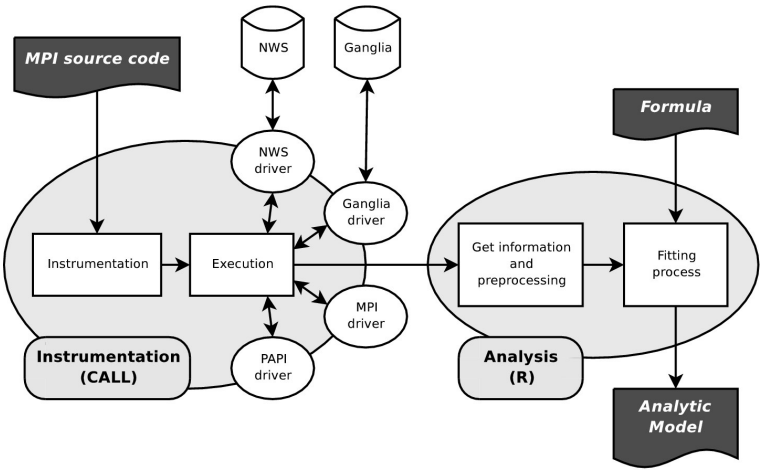


Figure 1. The two stages of the framework.

as new monitoring tools are available. CALL is based on pragmas that allow the user to define areas of interest in the source code. In these parts of the code, some selected parameters are monitored, so specific kernels or functions may be modeled instead of the whole application.

The measurements of the monitored parameters during the execution of the instrumented code are gathered and stored in XML files, one file for each CALL driver used to monitor the performance parameters. Some CALL drivers were specifically designed for cluster environments, like the NWS CALL driver and the Ganglia CALL driver[12]. The NWS CALL driver uses NWS[17] to monitor the effective latency and bandwidth of a parallel environment. Through a configuration file, the analyst can establish which nodes must be monitored during the execution of a NWS CALL experiment. The Ganglia CALL driver provides several performance metrics, both static and dynamic, of the nodes in a parallel system[13]. These two drivers play an important role in this stage, and their data are used in the analysis stage.

3 Analysis Stage

The second stage is the analysis one, that is based on R[14], a language and environment for statistical analysis. In early versions of CALL, the statistical analysis was integrated with the instrumented stage so that information about the behaviour of the CALL experiments had to be incorporated before the instrumented code was actually executed. The analysis stage was redesigned to get both stages fully uncoupled. Therefore, specific R functions were developed to process the data from multiple executions of a CALL experiment and to automatically perform analytical models of CALL experiments by means of an iterative fitting process. These functions are grouped into modules with well defined interfaces, so any change in a module produces a minimal impact on the others, and the capabilities of the analysis environment can be easily extended. Fig. 2 shows a scheme of the analysis

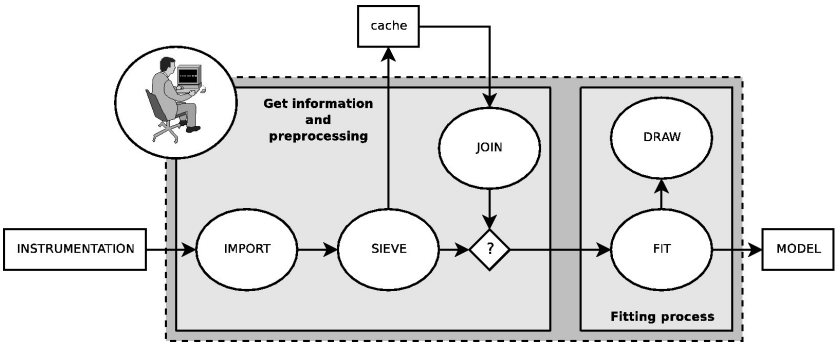


Figure 2. The modules of the analysis stage based on R.

stage and the relationship among its five modules.

The information stored in XML files from a particular CALL experiment is loaded into the environment, and it is stored in suitable structures for the statistical processing (IMPORT module in Fig. 2). At this point, some preprocessing may be necessary to obtain derived parameters. For example, NWS CALL driver provides information about the network state of some point-to-point communications. In general, this information consists of an array of values, where the impact of network fluctuations is present, and it could be necessary to obtain the mean of these values in order to associate only one value of latency and bandwidth to one execution of a CALL experiment.

The resulting object of the IMPORT module is an array of registers. Each register contains the monitored information related to a specific execution of the CALL experiment. In parallel applications, each parallel task has its own register. The number of fields of the register depends on the number of monitored parameters during the execution of the instrumented code. This object contains a huge amount of useful information, so the relevant data for a particular performance analysis must be extracted (SIEVE module in Fig. 2). For example, in a parallel program, the elapsed time of the slowest task is the elapsed time of the whole parallel program, so the elapsed time of the other tasks may be unnecessary; although this information can be useful if the user is interested in load balancing studies, for example. Therefore, a selection of useful information for the later analysis is performed guided by the user. In this step, the statistical outliers are checked, so executions with a elapsed time too far from the mean for executions under the same experimental conditions are not considered.

At this point, the framework has been designed to save this information in a file with a specific statistical format (cache in Fig. 2). Therefore, gathering information from CALL experiments becomes an independent process from the generation of a performance model. All the files concerning a specific CALL experiment can be stored, so that the volume of data increases as new executions of a instrumented code are performed. These files can be joined in the same data structure for the next fit process (JOIN module in Fig. 2). Note that only the common parameters in all instances of a CALL experiment are taken into account.

Once the information from all considered sources is ready, these data are fitted to an analytical expression which accurately describes the behaviour of the elapsed time of the CALL experiment as a function of the monitored parameters (FIT module in Fig. 2). This

is the main concern of the analysis stage. It starts from an initial attempt function as a first approximation of the elapsed time and continues with a set of iterations. In each iteration, the accuracy of the fit is established based on the standard error of the coefficients of the function. If the relative error of a coefficient is greater than a certain threshold (in our experiments, we consider 10%), it is assumed that such term can be neglected or that the model does not fit properly. Then, this term is removed and a new iteration is performed. This iterative process ends when all coefficients have an error smaller than the threshold. This process can start with a set of functions so that the process is applied to each individual function and the best fit is selected as the final result. The time consumption of the fitting process depends both on the number of terms of the initial function and on the size of data.

The user can provide the initial function or the initial set of functions of this process. Otherwise, a set of predefined and generic functions is used. These predefined functions implement a simple model based on the combination of a computational term (related to the number of processors) and a communication term (related to the latency and bandwith of the network). Each term is defined as a sum of different terms, which can be logarithms and powers of the monitored parameters, or a product of logarithmic and power terms, etc. Anyway, only expressions which have a theoretical base are considered.

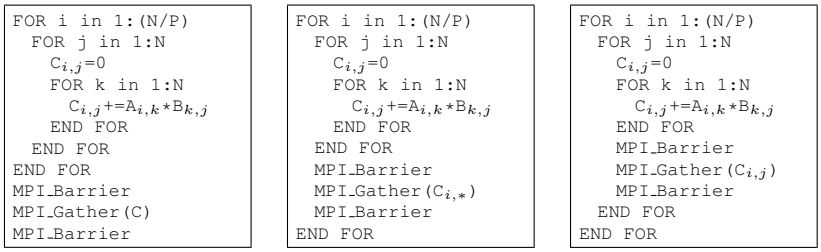
All the information about the resulting model of this process is shown to the user including coefficient of determination, standard error of coefficients, and some other useful statistical information. Besides, a comparison study between experimental measurements and the obtained model is performed through a specific visual environment (DRAW module in Fig. 2). In parallel applications, this environment shows both the experimental and predicted elapsed time versus the number of processes during the execution of the instrumented code. These graphics show the estimated times according to the model and the experimental values for all considered experimental conditions. The residuals of the model can be shown through a box-and-whisker plot.

4 Case of Study

Three different MPI versions of the parallel matrix product of two dense $N \times N$ matrix of single-precision floating-point numbers have been used as a case of study. Fig. 3 shows the pseudocode of these versions, that cover a broad spectrum from communication intensive to computation intensive applications. It is supposed that the distribution of the matrices has been performed in a previous step. The goal of the experiment is to compare both a theoretical model and the automatic model obtained from the proposed analysis stage using a set of predefined functions in a simple and well known parallel code.

The instrumented programs were executed in a homogeneous cluster of six 3 GHz Pentium 4 connected by a Gigabit Ethernet switch. The driver of the network interface card of the nodes allows the user to force the speed of the network. Therefore, different network states (10, 100, and 1000 Mbps) were obtained by using different input parameters of this driver. The NWS CALL driver was used to monitor the network state just before the execution of the programs. Each program was executed four times using three matrix sizes (N=300, 400 and 500) for each number of processors (from 2 to 6) and each network state.

A model of the three cases was automatically obtained using the developed analysis stage. As the structure of the code is known in the three cases, the parameters that may



a) CASE1 b) CASE2 c) CASE3

Figure 3. Pseudocode of the three versions of parallel matrix product (C=A×B), where the size of the matrices is $N \times N$, and P is the number of processes.

have influence in the application performance were identified. In this case, the following parameters were selected: N, P, L, and BW, where N is the size of the matrices, P the number of processors, L the mean latency of the network and BW is the mean bandwith of the network. The proposed initial set of functions for the iterative fitting process (described in Section 3) were identical in all cases. These functions cover all possible combination of the following expression:

$$t = A + BN^iP^j + CN^iP^j\lceil \log_2 P \rceil L + DN^iP^jBW^{-1}$$

where $i = 0, 1, 2, 3$, $j = 0, -1$ and the coefficients B, C and D can be set to 0.

In order to evaluate the accuracy of the proposed automatic fit process, the data from the instrumentation stage were also fitted to a theoretical expression based on the complexity analysis of the three codes. To build this theoretical model, a tree-base behaviour was supposed in global MPI communication. Besides, the effect of the communication-computation overlapping was supposed to be negligible.

Table 1 shows the final expressions obtained using the automatic approach of our analysis stage and the theoretical functions for each case. In these expressions, t is the elapsed time. The third column shows the coefficient of determination (R^2) for both the automatic and the theoretical fits from the complexity analysis. Note that the theoretical expressions presents accurate results, even for the third case in which the number of communications is huge.

5 Conclusions and Ongoing Work

A framework based on a methodology to obtain analytical models of MPI applications on multiprocessor environments is introduced in this paper. The framework consists of an instrumentation stage followed by an analysis stage, which are based on the CALL instrumentation tool and in the R language, respectively. A number of functionalities were developed in the analysis stage to help the performace analyst to study analytical models in parallel environments. One of the most relevant features is an automatic fit process to obtain an analytical model of parallel programs. This process provides accurate models as good as those based on a theoretical complexity analysis of source codes.

We are currently developing an efficient and flexible process to automatically generate and fit all analytical attempt functions. The process will permit the user to introduce in-

	Models	$\mathbf{R}^2$
CASE1	$t_{\text{Automatic}} = b\frac{N^3}{P} + d\frac{N^2}{BW}$	0.9945
	$t_{\text{Theoretical}} = A + B\frac{N^3}{P} + C\lceil\log_2 P\rceil L + D\frac{N^2}{BW}$	0.9946
CASE2	$t_{\text{Automatic}} = b\frac{N^3}{P} + cN\lceil\log_2 P\rceil L + d\frac{N^2}{BW}$	0.9966
	$t_{\text{Theoretical}} = A + B\frac{N^3}{P} + C\frac{N}{P}\lceil\log_2 P\rceil L + D\frac{N^2}{P}\frac{1}{BW}$	0.9904
CASE3	$t_{\text{Automatic}} = bN^2 + d\frac{N^2}{BW}$	0.980
	$t_{\text{Theoretical}} = A + B\frac{N^3}{P} + C\frac{N^2}{P}\lceil\log_2 P\rceil L + D\frac{N^2}{P}\frac{1}{BW}$	0.956

Table 1. The automatically obtained functions and the theoretical functions in each case. The third column is the corresponding coefficient of determination.

formation about the behaviour of the code, so that the precision of the obtained model will depend on the amount of the provided information. If no information is introduced, the monitored parameters will be used to build the initial functions. This process will provide a full search on all possible attempt analytical functions with physical meaning.

Acknowledgements

This work was supported by the Spanish Ministry of Education and Science through TIN2004-07797-C02 and TIN2005-09037-C02-01 projects and the FPI programme.

References

1. A. Alexandrov, M. F. Ionescu, K. E. Schauser and C. Scheiman, *LogGP: Incorporating Long Messages into the LogP Model for Parallel Computation*, J. Parallel and Distributed Computing, **44**, 71–79 (1997).
2. R. M. Badía, J. Labarta, J. Giménez and F. Escale, *DIMEMAS: Predicting MPI applications behavior in Grid environments*, in: Workshop on Grid Applications and Programming Tools (GGF8), (2003).
3. V. Blanco, J.A. González, C. León, C. Rodríguez, G. Rodríguez and M. Printista, *Predicting the performance of parallel programs*, Parallel Computing, **30**, 337–356, (2004).
4. J. L. Bosque and L. Pastor, *A Parallel Computational Model for Heterogeneous Clusters*, IEEE Trans. Parallel Distrib. Syst., **17**, 1390–1400, (2006).
5. M. E. Crovella and Th. J. LeBlanc, *Parallel Performance Prediction Using Lost Cycles Analysis*, in: Proc. 1994 ACM/IEEE Conference on Supercomputing, (1994).
6. D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, and T. von Eicken, *LogP: Towards a realistic model of parallel computation*, in: 4th

ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, (1993).
7. Th. Fahringer, M. Gerndt, G. D. Riley and J. Larsson Träff, *Specification of Performance Problems in MPI Programs with ASL*, in: International Conference on Parallel Processing (ICPP'00), (2000).
8. I. Foster and C. Kesselman, *The Grid2: Blueprint for a New Computing Infrastructure*, Elsevier, Inc., (2004).
9. S. Jarvis, D. P. Spooner, H. N. Lim Choi Keung, J. Cao, S. Saini, and G. R. Nudd. *Performance prediction and its use in parallel and distributed computing systems*, Future Generation Computer Systems: Special Issue on System Performance Analysis and Evaluation, **2**, 745–754, (2006).
10. N. H. Kapadia, J. A. B. Fortes and C. E. Brodley. *Predictive Application-Performance Modeling in a Computational Grid Environment*, in: 8th IEEE International Symposium on High Performance Distributed Computing (HPDC '99), (1999).
11. D. R. Martínez, J. L. Albín, J. C. Cabaleiro, T. F. Pena and F. F. Rivera, *A load balance methodology for highly compute-intensive applications on grids based on computational modeling*, in: Proc. Grid Computing and its Application to Data Analysis (GADA'05) - OTM Federated Conferences and Workshops, (2005).
12. D. R. Martínez, V. Blanco, M. Boullón, J. C. Cabaleiro, C .Rodríguez and F. F. Rivera, *Software tools for performance modeling of parallel programs*, in: Proceedings of the IEEE International Parallel & Distributed Processing Symposium, (2007).
13. M. L. Massie, B. N. Chun and D. E. Culler, *The ganglia distributed monitoring system: design, implementation, and experience*, Parallel Computing, **30**, 817–840 (2004).
14. R Development Core Team (2006), *R: A language and environment for statistical computing*, R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.
15. A. Snavely, L. Carrington, N. Wolter, J. Labarta, R. Badia, and A. Purkayastha. *A framework for performance modeling and prediction*, in: Proc. ACM/IEEE conference on Supercomputing, (2002).
16. L. G. Valiant, *A Bridging Model for Parallel Computation*, Commun. ACM, **33**, 103–111, (1990).
17. R. Wolski, N. Spring and J. Hayes, *The Network Weather Service: A distributed resource performance forecasting service for metacomputing*, J. Future Generation Computing Systems, **15**, 757–768, (1999).
18. X. Wu, *Performance Evaluation, Prediction and Visualization of Parallel Systems*, (Kluwer Academic Publishers, 1999).
19. Y. Zhang, C. Koelbel and K. Kennedy, *Relative Performance of Scheduling Algorithms in Grid Environments*, in: Seventh IEEE International Symposium on Cluster Computing and the Grid – CCGrid (2007).