Performance Prediction of Distributed Applications Running on Network of Workstations

Julien Bourgeois, François Spies
Laboratoire d'Informatique de Besançon
IUT Belfort-Montbéliard
BP 527 Rue Engel Gros
90016 Belfort Cedex, France

Michel Tréhel
Laboratoire d'Informatique de Besançon
16, route de Gray
25030 Besançon Cedex
France

Abstract *The Networks of Workstations (NoW) are becoming real distributed execution platforms for scientific applications. Nevertheless, the heterogeneity of these platforms makes complex the design and the optimization of distributed applications. To overcome this problem, we have developed a performance prediction tool called ChronosMix, which can predict the execution time of a distributed algorithm on parallel or distributed architecture. The objective of ChronosMix is to compare distributed algorithms and to classify them according to their performance on different architecture. C/MPI programs can be modeled by static or a postmortem methods.*

Keywords: Performance prediction, parallel applications, distributed applications, simulation

1 Introduction

Usually scientific applications are intended to run only on dedicated multiprocessor machines. With the continual increase in workstation computing powers and especially the explosion of communication speed, the networks of workstations (NoW) became possible distributed platforms of execution and inexpensive for scientific applications. Its main problem lies in the heterogeneity of NoW compared to the homogeneity of the multiprocessor machines. In a NoW, it is difficult to allocate the entire work in an optimal manner, it is difficult to know the exact benefit or if there is any or simply to know which best algorithm to solve a specific problem is. Therefore, the optimization of the distributed application is a hard task to achieve.

The abstraction of the target architecture and the mapping of the application is not possible, during the development of the parallel application because the efficiency depends on it. This fact is very important, because sequential algorithm complexity is independent of the architecture. When analyzing performance evaluation of a parallel application, it is necessary to take into account the execution context which is the target architecture.

Thanks to performance prediction, it is possible to subject the algorithm to a set of situations in order to define its behavior in extreme conditions. Indeed, the algorithm may be executed on non-available or future architecture to collect information about foreseeable performance. It is always interesting to examine an algorithm which runs on one thousand workstations or with shortly available processor speed or communication bandwidth. Finally, the algorithm may be executed in a real context where resources are shared and used at various thresholds, to control the algorithm behavior subjected by an important workload better.

This paper describes ChronosMix, our performance prediction tool applied to distributed systems. Section 2 shows our model and mechanisms inside ChronosMix.

2 Description of the tool

ChronosMix was developed to compare the performance of various algorithms solving the same problem and simulating them on a given distributed architecture. The distributed architecture used for the execution is entirely flexible. It can exist, or be a combination of existing elements or be prospective. Thus, it is possible to know the effects on one or more applications by changing one of the components.

The performance prediction of distributed applications passes through a modeling phase of the application and the target architecture. This modeling is used to put these two elements in an easy-to-handle mathematical form.

To predict the performance of a distributed application, ChronosMix will try to approach its execution time according to the performance of the execution platform. The time of a distributed application is known when three parameters can be extracted: the number of basic instructions inside the C/MPI program [1], the number of executions of each one of these instructions and the time taken by each basic instruction according to the workstation where the execution takes place. An execution time estimate is obtained in the form of a mathematical expression by combining these three parameters according to the path of the program execution.

To conclude this function of performance prediction, ChronosMix has three major components, the MTC (*Machine Time Characterizer*), the CTC (*Communication Time Characterizer*) and the PIC (*Program Instruction Characterizer*). These components are described in next sections.

2.1 Modeling of distributed system architecture

Distributed system architecture is composed of two resources: local and interconnected. Local resources represent the computing power. This computing power depends on three factors which are the workstation, the operating system and the compiler used. It is possible to model a local resource in two different ways. The first one consists in modeling the three elements of the local resource like in SimOS project [2]. The second way of modeling a local resource consists in considering its three components as a whole which acts on performance. As the modeling is global, and not any more divided into three layers, it is simple and therefore it can be automated. Our approach consists in defining a set of elementary operations called instruction set and in evaluating these instructions using the micro-benchmark technique described in [3]. Thus, each time that the local resource changes, the set of micro-benchmarks has only to be started again to get the new characteristics of this resource. The choice of the instruction set is crucial in order to extract the parameters which have the greatest effect on the system performance. Our experience acquired during our first work on modeling distributed systems in the EDPEPPS project [4], has helped us to define a relevant set of instructions.

MTC has been created to obtain the values of the instruction set for a specified computer. It has to evaluate the execution time of each instruction. A local resource will only be evaluated once by MTC, that is to say that the local resource model can be used to simulate all programs modeled by the PIC.

In the interconnection resources, as in local resources, several elements act together on communications and define performance. Our approach to model interconnection resources is the same as for local resources. As we have a parameter-based model, the most appropriate method to model interconnection resources seems to be a benchmark. SkaMPI [5] is one of the most suitable MPI communication benchmarking tool for our needs. This project is developed at the University of Karlsruhe by Ralf H. Reussner. The main advantage of SkaMPI over the other benchmarking tools, is that it has a lot of parameters to tune its behavior. CTC is based on SkaMPI and benchmarks the different communication point-to-point functions.

The MTC and the CTC model all the ele-

ments which define a distributed system architecture.

2.2 Modeling of parallel applications

PIC received two files: the first one is a C/MPI program and the second is given by MTC and CTC and contains the execution time of each element of the instruction set. Thanks to these two files, PIC will estimate the execution time of the C/MPI program.

PIC uses Sage++ [6], an object-oriented toolkit for building and restructuring parsing trees of C, C++, Fortran 77 and Fortran 90 programs.

To make the program analysis even easier, programs are split into blocks which enable a hierarchy to be defined within the program. For example, a loop block contains the body of the loop. Thus, it is easy to transmit information, like iteration number or iteration variable, only to the blocks located within the loop. The other advantage of this split is to compact information. Actually, after the instruction gathering within a block, the information which should have appeared on each line of the program, is saved. But, the greatest benefit concerns the postmortem analysis. To realize this analysis, the application is instrumented. Trace points are inserted in particular parts of the program to determine their number of executions. Instead of inserting a trace point after each line of the program, only one trace point per block of instructions is inserted. Thus, the execution time concerning the traced program is saved and the data trace file is compacted.

The PIC analysis starts with the attribution of execution numbers to each block. This attribution can be done in two different ways: statically, i.e. without proceeding to its execution or in a postmortem way with a trace. The aim is to be able to predict the execution time of a program as quickly as possible, i.e. with a minimum slowdown. The static method, when it can be applied, gives an almost instantaneous prediction. This method is therefore preferred to the postmortem one as much as possible.

2.2.1 Static evaluation

Static analysis of a C/MPI program will try to determine the execution numbers of each block without proceeding to the real execution of the program. The block execution numbers are obtained by analyzing the control structures, i.e. loops and conditional instructions.

The conditional instructions are handled by a private function called *PIC_proba(x)* where x represents the probability that the condition is realised. x can take any forms mentioned above. The user can insert this function into the program to estimate the result of a conditional expression.

Concerning a loop with integer bounds and an integer linear increment, PIC will produce the iteration numbers of the loop under a numerical form. For example, PIC will estimate the iteration number of the following loop at 100 :

$$for(i = 0; i < 100; i++)$$

When one or several bounds of the loop are defined by a constant variable or when the increment is a constant variable, a literal expression gives the iteration numbers. For example, PIC will estimate the iteration number of the following loop at n :

$$for(i = 0; i < n; i++)$$

When one or several bounds of the loop are expressed with linearly modified variables e.g. in the case of interdependent nested loops, this method cannot be applied :

$$for(i = 0; i < n; i++)$$
$$for(j = 0; j < i; j++)$$

In this case, the solution is to integrate to PIC a library developed at ICPS which allows to count the number of integer solutions in a system of rational and parametric equations and inequations [7]. The nested loops must be put under the form of one or two matrices which define a polyhedron. The index values of the loop are then the coordinates of the points that are included within the polyhedron, each of those integer points represent an iteration. So as to

determine the iteration numbers of the structure, the number of points inside the polyhedron are counted. Matrices are transmitted to the computing functions that give a result under the Ehrhart polynomial form.

These 3 forms able to determine statically the iteration numbers of a wide range of loops. However, in the case of non linearly interdependent nested loops or loops with bounds modified within the loop, the static method cannot be applied. A postmortem analysis is then required.

2.2.2 Postmortem evaluation

The main problem of trace analysis is the slowdown added to the program execution. Actually, trace generation concerning a performance prediction generally requires a wide range of parameters (number of memory accesses, operation types, etc.) as a result, the trace file is voluminous and therefore it takes a long time to generate and treat it. PIC uses trace only to determine the number and the execution order of each block. Thus, the generated trace is very simple, it only contains an integer per block.

This performance prediction with a trace does not need the execution architecture to be able to predict the application performance: this represents a great benefit. As the execution is only necessary to count the execution number and order of each block, it can be run on any architecture. It is even possible to execute it with a unique mono-processor computer. Moreover, the execution slowdown of the application is insignificant as the information added to the program is not numerous (one per block) and does not take a long execution time.

2.2.3 Estimation of the execution time

The module of the estimation of the execution time gathers the data supplied by PIC, MTC and CTC. This phase of time estimation is completely independent of the characterization phase of the physical resources and of the C/MPI program. i.e. once determined the execution number and order of the blocks, it is possible to forecast the execution time of the program concerning any execution platform already benchmarked, and vice versa.

To estimate the execution time of the distributed program, PIC must be able to estimate the execution time of the sequential blocks. Then, it must take into account the distributed aspect so as to deduce the total execution time.

Let $T_{m,t_{seq}}$, be the execution time of a sequential part of a task t_{seq} executed on a given workstation m. $T_{m,t_{seq}}$ is obtained by the following formula:

$$T_{m,t_{seq}} = \sum_{b=1}^{Nb(t_{seq})} NE_b . \sum_{i=1}^{n} t_m(S_i).NI_b(S_i) \quad (1)$$

Where $Nb(t_{seq})$ represents the number of blocks contained in t_{seq}, n the instruction number of the instruction set, S the instruction set, $t_m(S_i)$ the execution time of the S_i instruction on computer m, $NI_b(S_i)$ the number of S_i instructions contained in block b and NE_b the execution number of block b.

To estimate the execution time of a distributed program, PIC creates as many "tasks" objects as executed MPI processes. Each task object has a list of messages, an array containing the time of the instruction set supplied by MTC for the target computer, an identifier equivalent to MPI_RANK and the time spent since the beginning of the simulation. The aim of this method is to estimate the possible waiting times due to synchronous programming (MPI_Barrier, MPI_Recv, etc.). Each task object, one after the other, goes through its blocks according to its execution order. When the task object meets an MPI instruction, it sends a request to the message list. Moreover, if the MPI instruction is synchronous, the task object adds to its execution time the idle time associated with the synchronization. As soon as every task has gone through their blocks, PIC calls a message handler which synthesizes every request of message sending and receipt as well as every request of synchronization between tasks. It solves un-

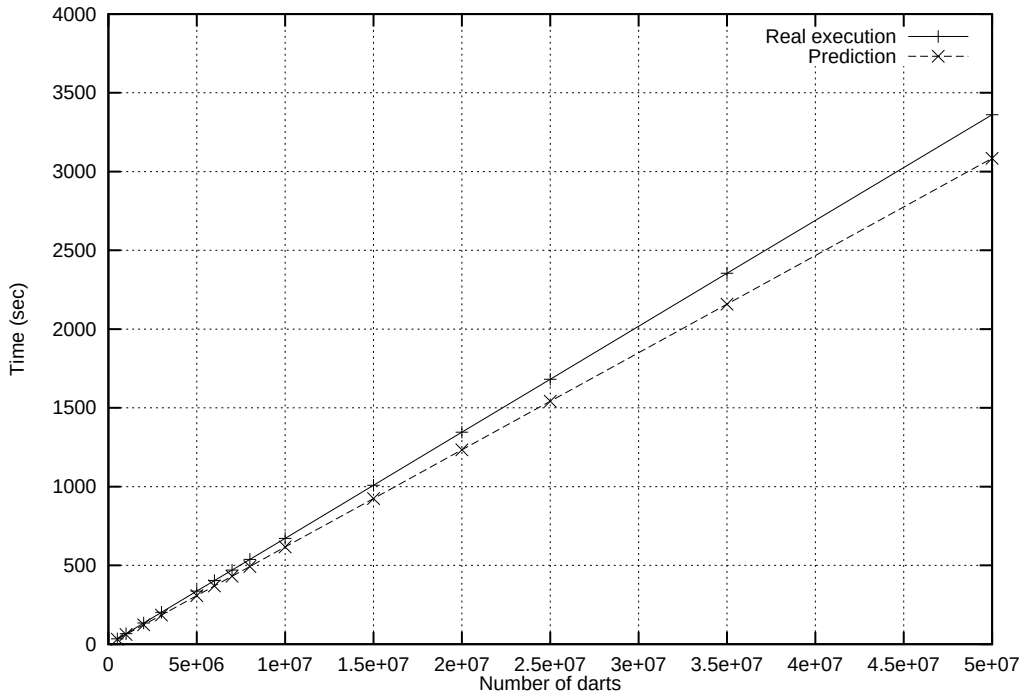

Figure 1: Computation of Π real execution vs. prediction

known idle times using the matching principles stated in the MPI standard. When every idle time is computed, formula 1 is applied to each task. The greatest time represents the total execution time of the application.

3 Case Study

This section discusses an experiment to check the accuracy of our model. We have chosen a classical C/MPI program of Π calculation. The method used is the dartboard algorithm. This program which can be found on the web (*http://www.mhpcc.edu/training/workshop/html /samples/index.html*), has been implemented in C by Roslyn Leibensperger and parallelized with MPI by George L. Gusciora. The programming paradigm is a master-worker. Each worker computes an approximation of Π and sends it to the master. The master also computes a value of Π, gathers the approximations and works out the average. Each approximation of Π is computed by throwing a given number of darts onto a board. This number varies from 5e+05 to 5e+07 throws. The execution platform is composed of 4 PC (Pentium II 350), interconnected by a switched Fast-Ethernet network (100 Mbit/s). Figure 1 shows the measured and the predicted execution time. The peak error between predicted and measured time is located at 5e+05 and represents a difference of 8.8%. The average error of 8% is a good result.

Figure 2 shows the graphical interface in Java for ChronosMix. This interface is a beta-version so it does not contain all the functionalities needed by the developer. So, it allows the developer to have an insight into the C/MPI program. The window on the left shows the program source code. The user can click on the different part of the code to select a block. The block is highlighted and its predicted time for each machine is displayed on the bottom right-hand window. On the top right-hand window, the predicted time of each function is printed for each machine. By clicking on the instruction mode button on the toolbar, it is possible

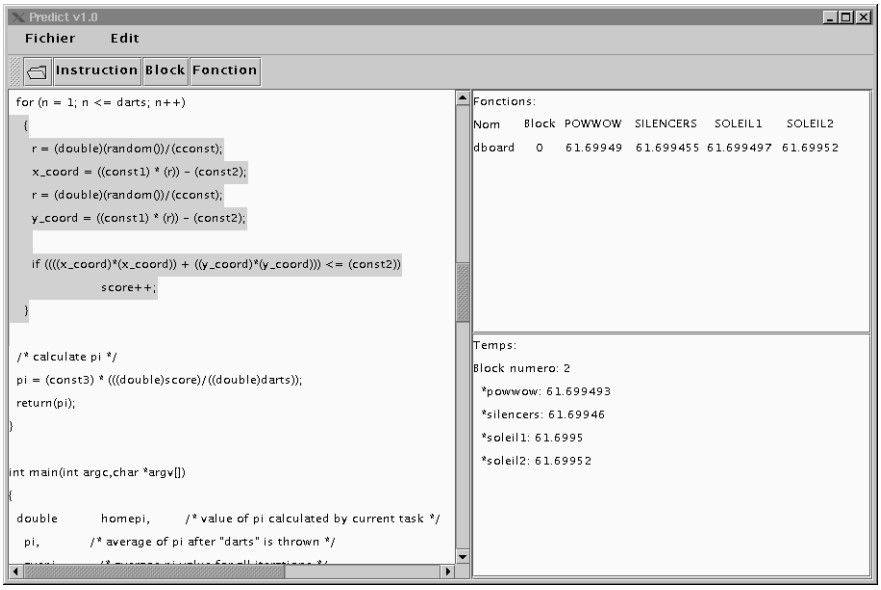

Figure 2: Beta version of the interface tool

to see in more detail how the predicted time is divided between the different operations. For example, for the selected loop of the Figure 2, the random calls take 51% percent of the loop time, the divisions take 32% of the time, and so on.

4 Conclusion

Static performance prediction brings swiftness which is not possible with other performance prediction methods. The word "slowdown" is no more appropriate to characterize this performance prediction, but there is a real speedup to extract results.

The performance of ChronosMix is validated statistically comparing real execution times with predicted times, from a standard example. Computational and communication times have been studied separately in order to near real times precisely.

The ability to model distributed system architecture with a set of micro-benchmarks allows ChronosMix to take heterogeneous architecture completely into account. Indeed, in a sense, modeling is automatic, because simulation parameters are assigned by the MTC execution to the local resources and by the CTC execution between workstations. The distributed architecture is simply and rapidly modeled, which allows to follow the processor evolution, but also to adapt our tool to a wide range of architecture.

It is possible to build target architecture from existing one by extending the distributed architecture, e.g. a set of one thousand workstations. It is also possible to modify all the parameters of the modeled architecture, e.g. to stretch the network bandwidth or to increase the floating-point unit power four-fold. A distributed application is exceptionally the exclusive consumer of some resources, only on specific parallel machines. It usually needs to share resources with other applications. This external load decreases performance application and must be included in the model. Finally, the CTC only takes into account a subset of MPI functions. Extensions will be conducted to include more MPI functions and especially group communication functions.

References

[1] Message Passing Interface Forum. Mpi: A message passing interface standard. Technical report, University of Tennesee, June 1995.

[2] Mendel Rosenblum, A. Herrod, Stephen, Emmett Witchel, and Anoop Gupta. Complete computer simulation: The simos approach. *IEEE Parallel and Distributed Technology*, 1995.

[3] R.H. Saavedra-Barrera, E. Miya, and A.J. Smith. Machine characterization based on an abstract high-level language machine. *IEEE Transactions on computers*, vol. 38(12):1659–1679, December 1989.

[4] T. Delaitre, M.J. Zemerly, P. Vekariya, G.R. Justo, J. Bourgeois, F. Schinkmann, F. Spies, S. Randoux, and S. Winter. A toolset for the design and performance evaluation of parallel applications. In *Proc. 4th Int. Euro-Par'98 Conf., Southampon, UK*, 1998.

[5] R. Reussner, P. Sanders, L. Prechelt, and Mller M. Skampi: A detailed, accurate mpi benchmark. In *5th European PVM/MPI Users' Group Meeting*, Liverpool, UK, September 1998. Springer Lecture Notes in Computer Science.

[6] F. Bodin, P. Beckman, D. Gannon, J. Gotwals, S. Narayana, S. Srinivas, and B. Winnicka. Sage++: An object-oriented toolkit and class library for building fortran and c++ restructuring tools. In *2nd Annual Object-Oriented Numerics Conf.*, 1994.

[7] Ph. Clauss and V. Loechner. Parametric analysis of polyhedral iteration space. *Journal of VLSI Signal Processing, Kluwer Academic Pub.*, 19(2):179–194, July 1998.