30

The PS project: development of a simulator of PVM applications for Heterogeneous and Network Computing

R. Aversa[a], A. Mazzeo[a], N. Mazzocca[a] and U. Villano[b]

[a]DIS, Universita' di Napoli, Via Claudio 21, 80125 Napoli (Italy)
[b]IRSIP-CNR, Via Claudio 21, 80125 Napoli (Italy)
e-mail: [aversa,mazzeo,mazzocca,villano]@nadis.dis.unina.it

Abstract

Heterogeneous computing environments require performance evaluation techniques that are more sophisticated and cost-effective than those currently used. This paper briefly describes a project aimed at the development of PS, a simulation environment for the performance analysis of distributed applications executed in Heterogeneous and Network Computing environments through the PVM run-time system.

Keywords

Heterogeneous Computing, Network Computing, Simulation, Performance Evaluation, PVM.

1 INTRODUCTION

Heterogeneous Computing (HC) (Khokhar, 1993), (Mechoso, 1994) and Network Computing (NC) (Anderson, 1995) share as a common denominator the exploitation of heterogeneous computing resources. The increased complexity of heterogeneous environments calls for performance measurement and analysis techniques that are more sophisticated and cost-effective than those commonly used in homogeneous parallel and distributed computing systems. Of great practical interest in particular is to obtain performance data before the software implementation, since this enables the software developer to choose carefully the workload to be assigned to each target machine as early as in the software design stage. This is a particularly thorny problem both in HC (Wang, 1992) and in NC (Mazzeo, 1995).

The above considerations are among the premises of the PS project, started in 1993 as a cooperation between the Department of Informatics of the University of Naples and IRSIP, an institute of the Italian National Research Council (CNR). PS (*PVM Simulator*) is a simulator for the performance analysis of distributed applications based on the Parallel Virtual Machine paradigm, the *de facto* standard for programming HC and NC systems (Geist, 1994). The software simulated by PS can either be a complete PVM program or a *prototype*, i.e., a

I. Jelly et al. (eds.), *Software Engineering for Parallel and Distributed Systems*

partially implemented program design. The simulation of the whole hardware/software system makes it possible to obtain aggregate and analytical indexes related to the heterogeneous system performance (e.g., efficiency, throughput, response time, individual processor utilisation), or traces which can be processed off-line by *Paragraph* to visualise the simulated program execution in a variety of different views.

In our opinion, the role that simulation techniques can play in parallel software engineering has not yet been fully recognised. In a recent paper, we have shown that simulation tools can help managing the complexity of software development for heterogeneous hardware (Aversa, 1995a). In this context, the fundamental advantage of simulation is flexibility. Simulation tools make it possible to compare the behaviour of different algorithms on the same hardware platform, to assess the effect of different problem decompositions, task allocations and load sharing techniques, or even to study the performance of a single algorithm on several existing or hypothetical computing environments. Simulation tools require neither the oversimplifications which are commonly used to deal with complex hardware/software systems through analytical models, nor the availability of fully-developed software and of a real machine, which are necessary for the performance analysis by monitoring/tracing tools. On the minus side, it should be noted that accurate simulations are computationally expensive. PS is characterized by a light simulation overhead, thanks to the adoption of models of program tasks which are at a higher level of abstraction than those adopted by other existing simulators. The modelling of the communication subsystem is instead particularly accurate, and is also capable of considering the effect of external network load. The fairly high accuracy attained in all validation tests of the simulator show that these are perfectly reasonable solutions, at least for the heterogeneous computing platforms which are currently available.

2 PS: A PVM SIMULATOR

PS (*PVM Simulator*) makes it possible to simulate a complete PVM application by combining predefined objects that simulate the inter-task communication subsystem (PVM daemons, TCP/IP protocol, network interfaces, physical interconnection medium) along with objects simulating the tasks making up the user program (Aversa, 1994-1995b). Its simulation kernel has been implemented using the *Ptolemy* environment (Buck, 1994), and can therefore be ported to any computing platform where Ptolemy can be executed, e.g., Sun, DEC and HP workstations, and IBM PC under Linux. A complete Ptolemy application (called a *Universe*) consists of a network of interconnected Blocks. Blocks may be either *Stars* (atomic objects) or *Galaxies* (composite objects, made up of Stars and other Galaxies). PS provides predefined objects (Stars and Galaxies) that model the computing environment. These objects are to be connected using the Ptolemy interactive graphical interface to Galaxies modelling user code, thus setting up a Universe which is representative of a complete PVM application. The effect of network load can be taken into account by means of predefined Stars sharing the same physical network medium, which generate a given statistical network load distribution.

A typical PS simulation session consists of the following phases (Figure 1):

1) The Galaxies modelling user code are built. User code can be supplied either as a complete PVM program or as a *prototype*. Prototypes are skeletons of code containing PVM calls, where some of the computations have been replaced by calls to delay procedures taking into account the time spent in the actual code.

2) The user sets up a Universe that is representative of the whole hardware/software system by constructing a graph where the nodes are icons corresponding to Galaxies or Stars, and the (directed) arcs represent interactions between them.
3) A suitable number of monitoring Stars, which record the events occurring in a specific point of the Universe, are inserted into the Universe diagram.
4) The modules of simulation code corresponding to the icons contained in the Universe are compiled and linked to the event scheduler.
5) The simulation is performed by executing the program produced at point 4.
6) The information on system performance produced during the simulation is examined. As Stars which produce a trace of program execution in a format compatible with *Paragraph* (Heath, 1991) are provided, it is also possible to collect more in-depth information about program behaviour and system performance off-line. If the measured performance is not satisfactory, new hardware configurations or allocation strategies have to be tried. This entails repeating all the previous steps several times.

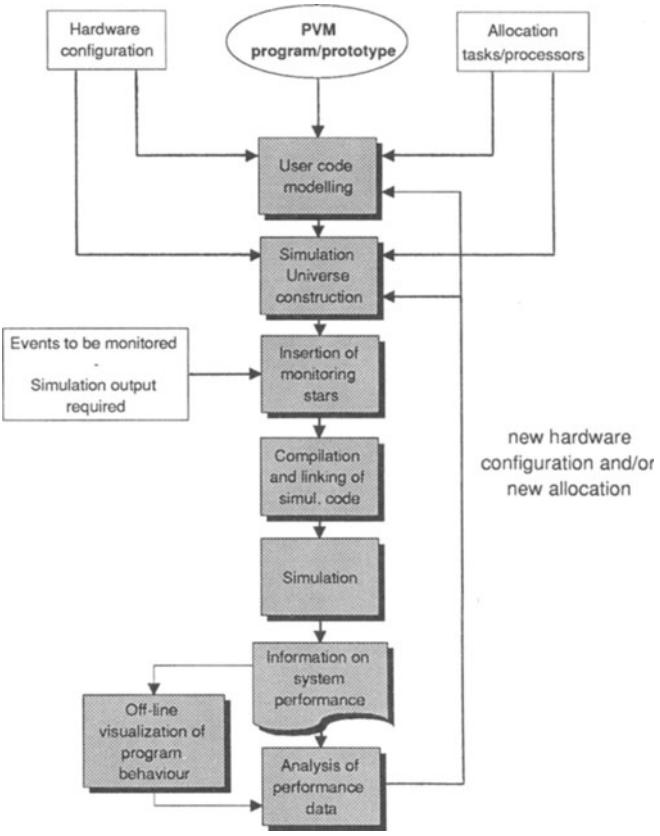

Figure 1 Phases of a PS simulation session.

A point that is worth discussing in more detail is how the program to be simulated is modelled in PS. The PVM application behaviour is modelled at user-task level by a set of *UT* (User Task) Galaxies, which model the user tasks running on every processor. For the sake of simulation accuracy, these objects should reproduce as closely as possible the sequence and the timing of the run-time requests of the actual program to the PVM daemon. The PS environment provides two different methods for the construction of *UT* Galaxies. The first possibility is to adopt a *delay-PVM call* model, building a graph where Stars invoking PVM primitives are interleaved by delay Stars corresponding to the time spent executing the code contained between them (the computation delays are estimates found by direct measurement under suitable test conditions or through statistical and/or analytical models). Another possible approach relies on the examination of the source code by a static program analysis tool, which analyses the performance behaviour of each task on the particular sequential or parallel computer where it will be actually executed. A tool for the static analysis of PVM code within the PS framework is currently under testing, and will be released in the near future.

The second option is to use *execution-driven* simulation (Convington, 1988), which is surely the recommended option when program execution time has not simple dependence on input data. In execution-driven mode, every PVM program task is modelled by a *UT* Star containing a version of the user code where PVM calls have been replaced with calls for service to the objects simulating the Parallel Virtual Machine level. During the simulation, the PVM program is *actually executed* (i.e., it is not simulated) in quasi-concurrence on the workstation hosting the PS environment. Whereas, the behaviour of the PVM run-time support, of the communication protocols and of the interconnection network is still simulated. Unlike other simulators based on the execution-driven approach (Davis, 1991), (Brewer, 1991), PS uses a simplified method to estimate the execution time of the block of code between two successive interactions with the simulation engine (interactions occur at every PVM call). This method introduces a much lower simulation overhead and leads to no significant accuracy loss, due to the coarse granularity of PVM tasks.

Figure 2 shows a screen dump taken during one of the simulation sessions described in (Aversa, 1995a). The picture shows the host display during the simulation of one of the algorithms considered for performing a matrix multiplication on a network of four workstations. The large window in the middle of the picture is the graphical representation of the simulation Universe. The four icons on the left of the Universe are the *UT* Galaxies simulating the behaviour of the application code executed on every node of the network. The Parallel Virtual Machine level, which models a computing node along with its run-time support, consists of the column of four icons in the middle of the window. The Network level is made up of the single icon on the right, which represents the shared transmission medium. The window in the lower left corner is an insight view of a *UT* Galaxy, showing in some detail its interface to the Galaxy simulating the PVM daemon. Figure 2 also shows samples of the Paragraph performance summaries of the simulated program execution, namely a Spacetime diagram and a Utilization Gantt chart.

3 CURRENT STATUS OF THE PROJECT - FUTURE DIRECTIONS

Over the last year and half PS has evolved from a largely-incomplete prototype to a fully working version. The results of all our validation tests, some of which are reported in (Aversa,

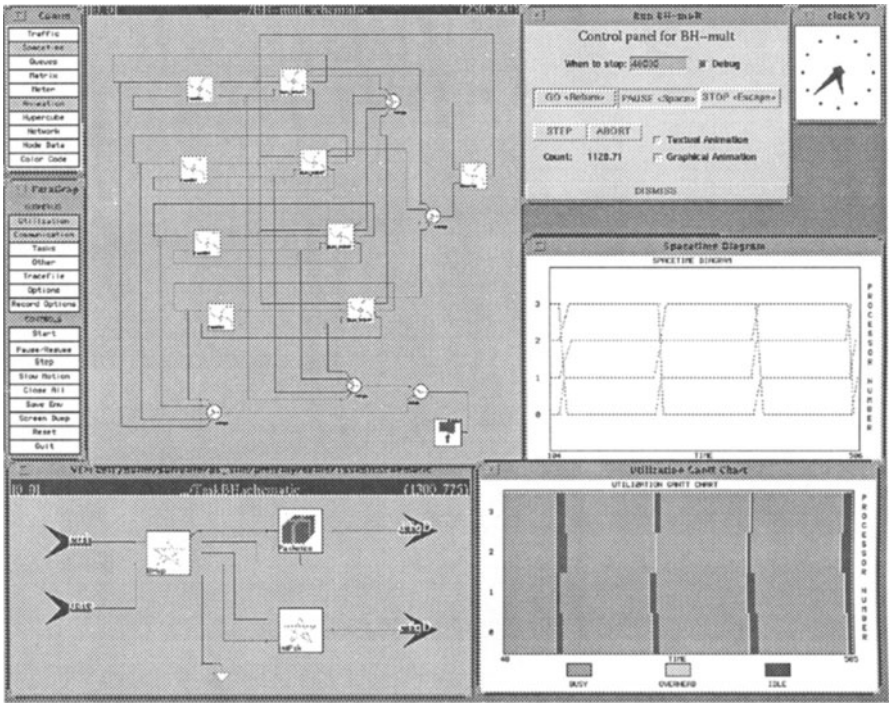

Figure 2 Host display during a PS simulation session.

1994), (Aversa, 1995b), have been particularly encouraging. PS never led to figures more than 5% far from those obtained by running the actual program on the real computing environment. Using the delay-PVM call model for user task, the PS simulation speed is too high to be compared to the execution rate of the actual program on a single processor. Whereas, if execution-driven mode is chosen, the execution of a PVM program is emulated a factor of only 6 to 9 times slower than the execution of the same program in quasi-concurrence on the simulation host. Unfortunately, we are not equally satisfied of the ease of use and friendliness of PS. The construction of the delay-PVM call Galaxies modelling user code is not easy and requires a sufficient degree of familiarity with the simulator itself. Furthermore, setting up the simulation Universe using the Ptolemy graphical interface is a time-consuming task, and one that does not lend itself to be automated by any software tool. This is the reason why we decided not to release the first version of PS outside our research group, and to redesign the simulator user interface in order to produce a new version amenable to be publicly distributed.

At the state of the art, the core of the version 2.0 of PS has been already implemented and is currently under testing. Besides the execution-driven mode (which is still supported, since it seems the best option for expert users), the new version of PS also makes it possible to launch a simulation session without using the Ptolemy graphical interface. In this case, the simulator is fed with a trace previously obtained by executing the PVM program in quasi-concurrence on a

single workstation or in real concurrence on a scaled-down distributed environment, and tracing the calls to the run-time support by means of a PVM-tracing library. A further issue that is being addressed in the development of the second version of PS is the possibility to study the effect of load due to tasks not belonging to the program to be analyzed. This will be obtained in much the same way as the effect of external network load is taken into account, i.e., by the addition of statistical interference with the computations on-going in each node.

4 REFERENCES

Anderson, T.E., Culler, D.E. and Patterson, D.A. (Feb. 1995) A Case for NOW (Networks of Workstations). *IEEE Micro*, **15**, 54-64.

Aversa, R., Mazzocca, N. and Villano, U. (1994) PS: a Simulator for Heterogeneous Computing Environments, in *Massively Parallel Processing Applications and Development* (eds. L. Dekker, W. Smit and J. C. Zuidervaart), Elsevier, 335-343.

Aversa, R., Mazzeo, A., Mazzocca, N. and Villano, U. (1995) The Use of Simulation for Software Development in Heterogeneous Computing Environments. *Proc. Int. Conf. on Par. and Distr. Processing Techniques and Applications*, Athens, GA, 581-590.

Aversa, R., Mazzocca, N. and Villano, U. (1995) Design of a Simulator of Heterogeneous Computing Environments. to be published in *Simulation Practice and Theory*.

Brewer, E.A., Dellarocas, C.N., Colbrook, A. and Weihl, W.E. (1991) PROTEUS: a High-performance Parallel-architecture Simulator. *Tech. Rep. MIT/LCS/TR-516*, Cambridge, MA.

Buck, J.T., Ha, S., Lee, E.A. and Messerschmitt, D.G., (1994) Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems. *Int. Journal of Computer Simulation*, **4**, 155-182.

Convington, R.C., Madala, S., Mehta, V., Jump, J.R. and Sinclair, J.B. (1988) The Rice Parallel Processing Testbed. *Proc. 1988 ACM SIGMETRICS Conf. on Measurement and Modeling of Computer Systems*, 4-11.

Davis, H., Goldschmidt, S.R. and Hennessy, J. (1991) Multiprocessor Simulation and Tracing using Tango. *Proc. 1991 Int. Conf. on Parallel Processing*, II99-II107.

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. and Sunderam, V. (1994) *PVM: Parallel Virtual Machine*. MIT Press, Cambridge, MA.

Heath, M.T. and Etheridge, J.A. (Sept. 1991) Visualizing the Performance of Parallel Programs. *IEEE Software*, **8**, 29-39.

Khokhar, A.A., Prasanna, V.K., Shaaban, M.E. and Wang, C. (June 1993) Heterogeneous Computing: Challenges and Opportunities. *IEEE Computer*, **26**, 18-27.

Mazzeo, A., Mazzocca, N. and Villano, U. (1995) Efficiency Measurements in Heterogeneous Distributed Computing Systems: from Theory to Practice. submitted to *Concurrency: Practice and Experience*.

Mechoso, C.R., Farrara, J.D. and Spahr, J.A., (Summer 1994) Achieving Superlinear Speedup on a Heterogeneous, Distributed System. *IEEE Par. and Distr. Technology*, **2**, 57-61.

Wang, M., Kim, S., Nichols, M.A., Freund, R.F., Siegel, R.F. and Nation, W.G. (1992) Augmenting the Optimal Selection Theory for Superconcurrency. *Proc. Workshop on Heterogeneous Processing*, IEEE Computer Society Press, 13-21.