DiP : A Parallel Program Development Environment*

Jesus Labarta, Sergi Girona, Vincent Pillet, Toni Cortes, Luis Gregoris

Departament d'Arquitectura de Computadors - CEPBA
Universitat Politècnica de Catalunya
[jesus,sergi,vincent,toni,luisg]@ac.upc.es
URL: *http : //www.ac.upc.es/hpc*

Abstract. This paper describes an environment whose aim is to aid in the development and tuning of message passing applications before actually running them in a real system with a large number of processors. Our objective is not to eliminate tests on real machines but to be able to focus them in a more selective way and thereby minimize their number. The environment presented in this paper consists of three closely integrated tools: an instrumented communication library, a trace driven simulator (Dimemas) and a visualization/analysis tool (Paraver).

1 Introduction

The objective of the DiP environment is to reduce the cost of parallel program development, both in time and hardware requirements. This is achieved by applying a set of tools that allow the use of sequential machines for the development and tuning of parallel applications. In the same way that finite element packages minimize the prototype and experiment requirements, the DiP objective is not to eliminate tests on real machines but to be able to focus them in a more selective way and thus minimize their number. Before actually testing the application on a parallel machine, the developer should be able to optimize it for the target machine and to have an accurate estimate of its performance.

The functions offered by the DiP environment fall in the areas of performance prediction, analysis and visualization. As a prediction tool, DiP estimates the performance that a message passing application would achieve on different types of architectures, ranging from workstation clusters to networks of SMPs and MPPs. As an analysis tool, the DiP environment is of special interest for two reasons: in obtaining detailed quantitative statistics of an application run and in evaluating the effects of different factors in the performance of the application. Two types of factors are available. There are target machine architectural parameters and code blocks or routines duration. Finally, visualization has been conceived as a support mechanism for the analysis and prediction functions, providing flexibility and efficiency.

* This work has been supported by the Spanish Ministry of Education (CICYT) under the TIC-94-537 and TIC-95-0429 contracts.

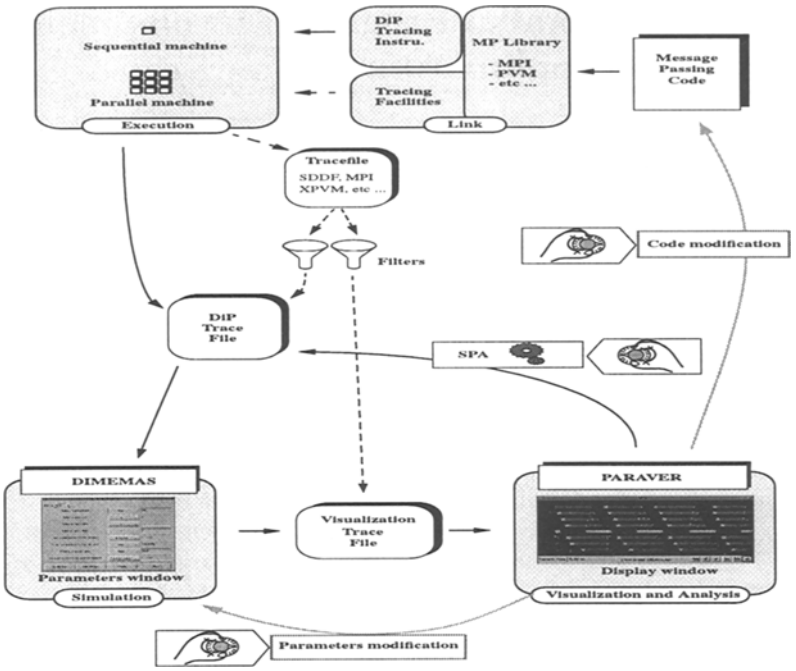

Fig. 1. The DiP Environment Structure

The two main objectives in the design of DiP were: to emphasize a clear division between parts of the tool set, where each module has its own function, and to offer flexible mechanisms to combine those modules in order to construct a very powerful analysis and prediction tool. From these simple concepts, this tool enables complex and large application analyses.

2 The DiP environment

The global structure of the DiP environment is described in Figure 1. Three tools constitute the core of the environment: the DiP instrumented communication library; Dimemas, a distributed memory machine simulator; and Paraver, a visualization and analysis tool. Translators (filters) from several trace formats (SDDF, PICL) to DiP trace format have been included to allow other products to interact with DiP.

The arrows in Figure 1 describe the possible combinations of the different tools. Part of the power and usefulness of the environment comes from the flexibility it provides in sequencing the execution of the individual tools.

The path shown in Figure 2a matches the well known parallel programs debugging and visualization process [9]. This approach is applied on many widely used tools (XPVM [6], Paragraph [7], HeNCE [3] and Xab [2]). In this path,

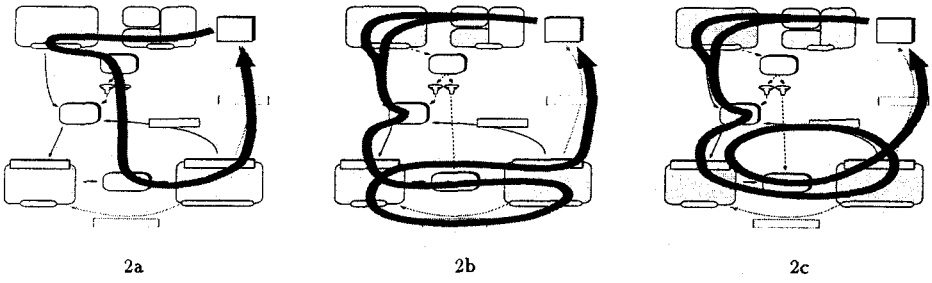

Fig. 2. Typical path (2a) versus DiP processes (2b,2c)

when the application is run on a parallel machine, the profiling facilities generate a trace file with absolute times. The user can visualize this trace file and, based on these observations, modify the code and repeat the whole process.

Figure 2b shows the typical path used in DiP to carry out studies on the influence of the simulator parameters. In this case, the application can be run with the DiP instrumented library on a single workstation or in a parallel system. The instrumented library generates a trace file with relative times, which characterizes the application. From this trace file, Dimemas rebuilds the behavior of the application on a parameterized target machine. A new trace file, with absolute times, is generated for visualization with Paraver. The simulation and visualization phase can be repeated several times in order to predict and analyze the performance of the application on different target architectures. Based on visualization and quantitative measurements, the user can modify the application source code. DiP allows users to analyze and predict the performance of parallel applications using a sequential machine.

The repetition of the simulation with several architectural parameters provides significant information on the application's behavior, that should be used before recompiling the application or obtaining new traces. For example: statistics on processor utilization can be used to analyze the application load balancing, instantaneous communication parameters can be used to evaluate the effect of common communication dependences, changing then communication latency and keeping and instantaneous transfer rate allows the analysis of the communication granularity,...

The third possible path (shown in Figure 2c) involves the SPA (Synthetic Perturbation Analysis). Paraver is able to generate a perturbed trace in which the duration of selected routines has been modified by a certain amount. Several simulations will provide an indication of the routine's effect in the execution of the application. Special functions have been included in the library to provide the possibility of marking the entry and exit of routines or code blocks.

3 Communication Library Instrumentation

The objective of the instrumented library is to capture, in a trace file, some states and event records that characterize the application. The idea is to extract information inherent to the application and to avoid the effects of the platform used for this purpose on the application. This can be done by running the instrumented application on a parallel machine or by running all the processes on a single processor in time shared mode. DiP includes instrumentation for PVM and MPI.

The key to the application characterization lies in two types of records: state and event. State records represent actual resource demands and their duration, CPU bursts for example. For a precise application rebuild, it is important to achieve precision and accuracy in the measurement of the duration of the CPU bursts. Event records represent punctual occurrences between state records and can be either user defined or communication endpoints. Although the communication records capture the relationship between different tasks, no absolute times are included in the trace.

Some functions have been defined in order to introduce user defined event records in the trace which can be used to tag specific "events" in the code. These user events are used to mark the entry and exit of routines or code blocks.

DiP supports multi-threaded applications, represented by the following three level process model: the application itself, the different tasks that constitute it and the possibility of having multiple threads within a task. This model is general enough to include most of the current process models. The library has been extended for semaphore support for synchronization between different threads within a single task.

In DiP, the probe effect problem has been reduced to the cache pollution caused by other processes sharing the system, specifically in the case of traces which are generated from a single workstation. In [10] the effect of this problem and the quality of the prediction are shown. The common I/O generated probe effect problem of interaction between the instrumentation and application behavior is not present because the measurements are based on the duration of the CPU bursts, and absolute time is not used at all in the instrumentation.

4 Dimemas

Dimemas is a simulator that reconstructs the behavior of a parallel program from a set of traces that characterize the application as described in the previous section. New traces with absolute times can be obtained using the architectural model of a target machine. Dimemas also gives global statistics on the performance of the application. Other features supported by Dimemas include the ability to work with multi-threaded applications and multiprogrammed workloads.

The architecture model is simple and flexible. It is composed of a set of SMP (Shared Memory Processors) with a communication network. This communication network can model a full connectivity network, (to model MPP clusters)

or a single bus network, (to model networks of workstations). Shared memory communications is also possible when several tasks share the same SMP node.

Dimemas uses a linear function (latency and bandwidth) to model the communication time. The influence of the distance between processors is considered irrelevant in accordance with networks state of the art. Network conflicts are modeled with two different possibilities: conflict in the network usage (for the bus networks), limit on network connectivity and conflict in the usage of communication resources (links into the network for each individual node).

Dimemas also models several scheduling policies (FIFO, RR, Unix like,...) to allow processor sharing among several threads. The model of different processors is reduced to a single parameter called relative processor speed, representing the ratio between the processor speed where the trace is obtained and the processor speed on the target machine. An accurate value for this parameter can be obtained by running the sequential application on both processors, but public information about processors speed is an alternative that may also lead to useful results. In Section 6, prediction accuracy results are presented.

A file system module [5] is currently being added to Dimemas in order to study caching policies in parallel environments and their interaction with the scheduling policies. A communication module has also been included for modeling communication using ATM networks.

5 Paraver

Paraver is the visualization and analysis part of the tool-set. Paraver offers a single type of view flexible enough to represent a large amount of information. It consists of a Gantt diagram with one line for each object to be represented (the type of object can be selected for each window among processors, application, tasks or threads). The information for each line may be represented with an encoded color (each color represents a possible state for different objects) or as a time function. User events and/or communication between several objects can be included in the same view. Users can select the information to be displayed in a very flexible way based on the tool structure presented in Figure 3. Two basic processes can be applied using Paraver: the visualization and static analysis process and the Synthetic perturbation analysis process.

The visualization process structure is represented by the left side of Figure 3 and is applied to every display window controlled with DiP. The functions of the visualization process are managed from different modules, each of them filtering information on to the next. The filter module $\boxed{f}$ selects which events (user and communication events) must be passed on to the next modules, e.g. display only those messages with a specific tag. The semantic module $\boxed{\int}$ computes the values to be represented, using the information associated to each window. The visualization module $\boxed{v}$ is responsible for the representation of events on the screen, for the pointing device management and for the window creation (through the zoom facility or a user command). The static analysis module $\boxed{A}$

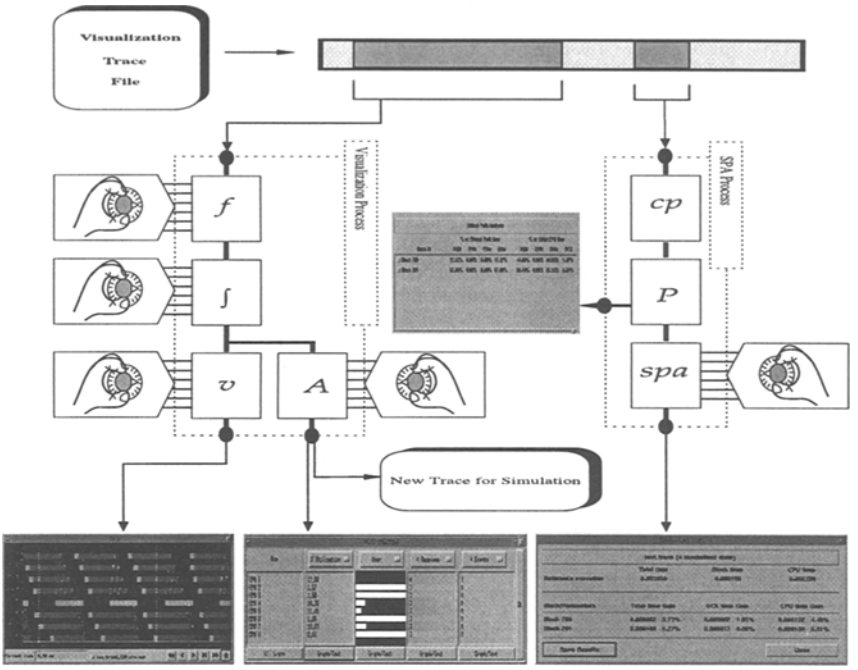

Fig. 3. Paraver Internal Structure

computes statistics (processor utilization, number of communications,...) on all the trace files or only in a user selected area. User specific functions can be developed and linked using a procedural interface.

DiP offers a single window to display information, but all the application's behavior can be displayed on this one. The semantic module allows the user to select what kind of information will be displayed. For example, the user can select to display the information as contained in the trace file, only displaying the current object state (communication overhead, waiting for processor, useful computation time, waiting for resources,...). However, if the object is a task, with several threads, the user may want to display whether some thread in the current task is running or to select the number of threads within the task that are currently waiting for resources. The user can extend the semantic module capabilities by linking user functions. Another example is to display applications and select to draw the number of threads currently running; the information displayed is the instantaneous parallelism profile of an application.

The Synthetic Perturbation Process, on the right side of Figure 3, has been developed to help the user to find execution bottlenecks. It shows how to modify applications to enhance performance and highlight hardware critical parameters for the application. We have adapted [11] the Synthetic Perturbation Analysis (SPA) technique based on a factorial design experiments [8].

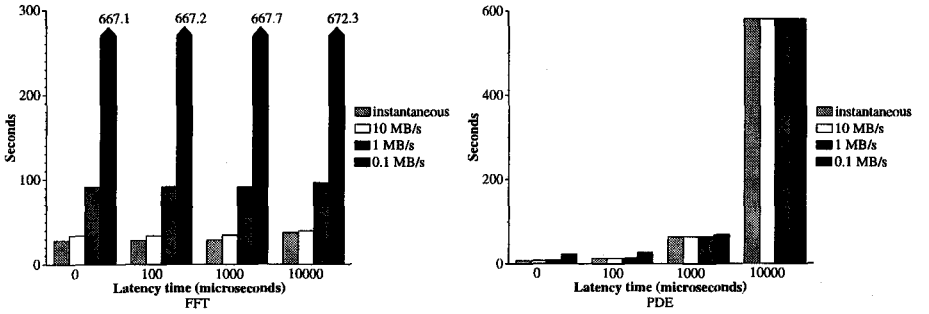

Fig. 4. FFT and PDE application time

The application code must be instrumented, manually or using compilation techniques, to mark those functions and code blocks which are going to be analyzed. This process is also separated into different modules: the Critical Path [*cp*] module finds the *critical path* of the application, the Profile module [*P*] displays the percentage of time spent by each function (code block) in the critical path with reference to the global application time, and the Synthetic Perturbation Analysis module [*spa*] prepares a set of modified trace files where some of the records (those corresponding to the blocks selected by the user) are modified by a certain amount. Statistical analysis is presented with the information obtained with these modified trace files and by rerunning Dimemas several times.

The SPA module can help the user to answer questions such as: " *What will the total time gain be in the application execution if the execution time of one function is improved by 5%?*", " *What is the influence of the communication latency on the total execution time?*" and " *What is the interaction effect of two functions?*".

6 Examples

This section includes some practical examples on how the DiP environment can be used for several purposes in the areas of: the evaluation of the influence of communication speed and performance prediction.

6.1 Influence of Network

The influence of communication latency on overall application time for FFT and PDE applications is analyzed in Figure 4. This figure shows how latency has a strong influence on PDE because this application has a lot of communications. On the other hand, on FFT the bandwidth is the important issue, because it does not communicate very often and uses large messages.

Figure 5 shows the influence of network conflicts in the behavior of the application. As expected, the influence is higher as the communication bandwidth decreases, simply because the network is busy for more time.

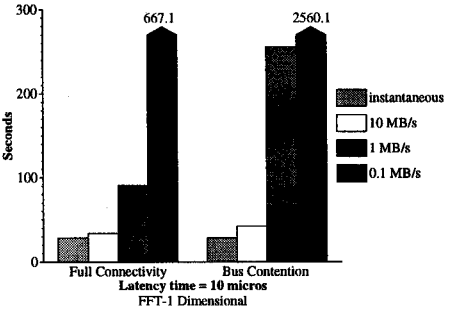

Fig. 5. Full connectivity network versus Bus Contention

	7	8	9
	12.426351	28.480770	90.619751
4	14.995254	31.460633	93.013042
	17.13 %	9.47 %	2.57 %

Table 1. PDE-2 prediction results

	15	16	17
	32.740425	67.560176	142.729947
4	32.644082	67.744027	140.541845
	0.29 %	0.27 %	1.53 %

FFT-1 prediction results

6.2 Prediction

In this section we present some results obtained using the prediction capabilities of the tools. The objective was to predict the performance of the FFT and PDE applications on an SP2 when only a SGI workstation is available. The trace files were obtained by running the application on the SGI workstation, on a time shared environment. The instrumented communication library used was a PVM public domain version.

The parameter for the relative processor speed between one node of an available SP2 and our SGI, was obtained by running the sequential application on both processors. Table 1 contains the seconds spent for the real execution, the seconds predicted by the simulator and the error average. The columns represent the dimension of the problem (2^n) and the rows reflect the number of processors employed by the application.

The results show that the accuracy of the prediction is fairly good. Even if the traces are obtained on a single multiuser workstation and the prediction is for a dedicated SP2, our approach shows up as very significant and falls within the range of accuracy that would be sufficient before carrying out very detailed tunings on a given target machine.

6.3 SPA Example

The code used in this example is a Triangular System of Equations which uses a block cyclic data distribution and a broadcast communication model of solution blocks. The result of the SPA analysis is the % of time gain from reference (initial) execution times. The first part of the analysis module gives the routines

	Profile Values								SPA values		
	% of CP time				% of total CPU time				% of time gain		
routines	RUN	OVH	COM	SUM	RUN	OVH	SUM	BLK	Total	BLK	RUN
main	73.67	0.00	0.00	73.67	90.95	0.00	90.95	0.00	9.21	8.78	9.09
receive	2.65	0.37	18.50	21.52	5.27	0.75	6.02	23.43	0.36	-0.34	0.53
forward	2.73	1.62	0.00	4.35	1.25	0.75	2.00	0.00	0.15	0.67	0.13

Table 2. (RUN=Running, OVH=Overhead, COM=Communicating, BLK=Blocked)

profile (Table 2 Profile values). Only the three most consuming routines are shown in this table.

It is obvious that the function *main* is the most CPU consuming and also the most influent on the critical path although its contribution to it is not as important as to the total CPU time. We can also see that the *receive* routine spent 18.5% of the total execution time waiting for messages. One possible analysis is *" What will be the total execution time gain of my application if I reduce one of these functions by 10% ?"*

Table 2 (SPA values) shows that a 10% optimization of the routine *main* (by looking in more detail at the code and cache locality issues for example) would result in a 9.21% improvement in the total execution time. These results are a bit better than what its contribution to the critical path had suggested (7.3%). The reduction of the computation time within the *receive* and *forward* routines would minimally influence the performance. Relatively the *receive* routine would improve more than the *forward*, but the static analysis indicates that it has a slightly higher contribution to the critical path.

7 Related Work

Many other tools are currently available, ranging from the programming environment at the source code level, as HeNCE [3], to the post-mortem visualization tools like Paragraph [7]. Unlike the DiP environments, most of them like XPVM [6] assume that all programming resources are available. The DiP approach is also to provide a set of tools usually found separately thus providing more efficiency. These tools are : an instrumented library as PICL [7], Pablo [15] (SDDF [1]) , or AIMS [16] but instead of generating wall time events trace files we propose a relative times approach, a trace driven simulator as Proteus [4], a visualization tool as Paragraph [7], Xab [2], Paradyn [12], XPVM [6] or PARvis [14] but based on a single type of representation providing powerful features not available on all tools, profiling and statistics functionalities as AIMS [16] or Pablo [15] and prediction possibilities as Speedy [13] or AIMS [16].

8 Conclusions

In this paper we have presented an environment developed at CEPBA (European Center for Parallelism in Barcelona) to reduce costs in parallel program

development and tuning. The environment is based on a trace driven simulator and a tool for visualizing and analyzing those traces. Its objective is to predict the performance of a message passing application on machines not readily available while doing most of the development on a workstation. The Synthetic Perturbation Analysis makes it possible to determine the relevance of software and hardware parameters.

The design of the tools is modular with the objective of enabling the study of other factors of parallel program performance such as locality and cache utilization, file systems,...

The environment is being used in several industrial projects with great success. Dimemas, with the Synthetic Perturbation Analysis capabilities, is commercially available from PALLAS GmbH (Germany) and Paraver will be available as a Public Domain product.

References

1. R.A. Aydt. "SDDF: The Pablo Self Describing Data Format". University of Illinois at Urbana-Champaign Technical Report, Mars 1992.
2. A. Beguelin. "Xab: A Tool for Monitoring PVM Programs". Workshop on Heterogeneous Processing, Los Alamitos, California, pp. 92-97, April 1993
3. A. Beguelin, J. Dongarra, A. Geist, R. Manchek, K. Moore and V. Sunderam, "PVM and HeNCE: Tools for Heterogeneous Network Programming", Environments and Tools for Parallel Scientific Computing, pp 139-153, Eds: J.J. Dongarra et al., Elsevier Science Publishers, 1993.
4. E.A. Brewer et al., "Proteus: A High-Performance Parallel-Architecture Simulator", Massachusetts Institute of Technology, Technical Report MIT/LCS/TR-516, September 1991.
5. T. Cortes, S. Girona and J. Labarta, "PACA: a Cooperative File System Cache for Parallel Machines", Euro-Par'96, Lyon, August 1996.
6. G.A. Geist, J. Kohl and P. Papadopoulos, "Visualization, Debugging and Performance in PVM", Processings of Visualization and Debugging Workshop October 1994.
7. M. Heath and J.Etheridge. "Visualizing the Performance of Parallel Programs". IEEE Software, pp. 29–39, Sept. 1991
8. R. Jaim, "The Art of Computer Systems Performance Analysis", John Wiley and Sons, New York, 1991.
9. A. Hondroudakis, "Performance Analysis Tools for Parallel Programs", Edinburgh Parallel Computing Centre, Technical Report 1995
10. J. Labarta, S. Girona, V. Pillet T. Cortes and L. Gregoris, "DiP: A Parallel Program Development Environment", UPC-DAC Tech. Report RR-UPC-DAC-1996-04
11. G. Lyon, R. Snelick and R. Kacker, "Synthetic-Perturbation Tuning of MIMD Programs", The Journal of Supercomputing, Vol. 8, pp 5-28, 1994.
    Fifth Brazilian Symposium on Computer Architecture, Florianopolis, September 1993.
12. B.P. Miller, J.M. Cargille, R.B. Irvin, K. Kunchithapadam, M.D. Callaghan, J.K. Hollingsworth, K.L. Karavanic and T. Newhall, "The Paradyn Parallel Performance Measurement Tools"
13. B.W. Mohr, A.D. Malony and K. Shanmugam, "Speedy: An Integrated Performance Extrapolation Tool for pC++ Programs", University of Oregon, 1995.
14. W.E. Nagel and A. Arnold, "Performance Visualization of Parallel Programs - The PARvis Environment", Research Centre Julich (KFA), Central Institute for Applied Mathematics (ZAM) Germany 1994.
15. D.A. Reed, R.A. Aydt, R.J. Noe, P.C. Roth, K.A. Shields, B.W. Schwartz and L.F. Tavera, "Scalable Performance Analysis: The Pablo Performance Analysis Environment", Scalable Parallel Libraries Conference, IEEE Computer Society, 1993.
16. J.C.Yan, "Performance Tuning with AIMS- An Automated Instrumentation and Monitoring System for Multicomputers", Proceedings of 27th Hawaii International Conference on System Science, Wailea, Hawaii, Vol II, pp 625-33, January 1994.