The Tool-set for PVM *

Thomas Ludwig, Roland Wismüller
Lehrstuhl für Rechnertechnik und Rechnerorganisation
Institut für Informatik der Technischen Universität München
D-80290 München, Germany
Tel.: +49-89-2105-2042 or -8164 or -8240, Fax: +49-89-2105-8232
E-mail: {ludwig,wismuell,bode}@informatik.tu-muenchen.de

April 8, 1996

Abstract

The Tool-set for PVM will comprise a set of integrated tools which can either be used individually or in concert. Tools will be subdivided into interactive tools for investigation and manipulation of the program state and automatic tools. Both types of tools will use online data, trace data, and checkpoint information to perform their task. The Tool-set will be composed of a debugger, a performance analyzer, a visualizer, a deterministic execution controller, a load balancer including a checkpoint generator, and a parallel file system. All tools will be available under the GNU General Public License Agreement. First versions of the tool environment will be released in spring 1996; some of its components will be available earlier.

1 Introduction

The PVM programming library has become an increasingly popular de-facto standard for writing explicitly parallel programs based on the message passing paradigm. A large variety of parallel applications, e.g. multigrid solvers, image processing or gene sequence analysis has been implemented on top of PVM. PVM is also being used as a platform for developing tools for parallel programming. Although some of them are rather sophisticated, currently only few tools are available and usable for the application programmers' community. One reason for this fact is that many tools are pure research prototypes providing only rudimentary, clumsy interfaces difficult to use for non-experts. A second reason is that existing tools are not integrated and cover only a single aspect of parallel program development. Here the term "integrated" does not primarily mean an integrated implementation of these tools, but the more important aspect of integrated usage, that is whether or not the tools can be used in combination and also provide support for each other. Currently, different tools can usually not be used for the same program, since they require the program to be compiled in different ways or to be linked with instrumented libraries not compatible with each other.

A new project of the Lehrstuhl für Rechnertechnik und Rechnerorganisation, Technische Universität München (LRR-TUM) will change this situation by providing an integrated tool-set for PVM including both run-time support and programming tools. The project is based on the previous work of our group that is performing research in the field of parallel processing tools for more than eight years[1].

*This work is partly funded by the German Science Foundation, Contract: SFB 342, TP A1

[1] If you are interested in the ancestors of The Tool-set, please refer to [1].

It will integrate and adapt our existing tools and experience to form a PVM tool environment supporting parallel I/O [8], load balancing [5], resource management and checkpointing [9], performance analysis [4, 3], debugging [7], deterministic execution [6], and visualization [2] (the citations refer to already existing work). Since we can rely on considerable preparatory work that always aimed towards industrial quality, easy-to-use interfaces, and integration of programming tools, we expect a first version of The Tool-set to be available for the user community in spring 1996.

2 The Tool-set

This section will give an overview over the concepts of The Tool-set. Many of them are already fixed and are waiting just to be implemented, others are still research topics and require more conceptual work. As the complete tool-set has a high internal complexity, not all features will be available at the same time with full functionality. The end of this paper will present a preliminary time schedule and some first deadlines.

The tools can be divided into classes according to the following criteria. First, we will offer interactive tools and automatic tools. Interactive tools (like The Debugger, The Load Balancer) will support implementation and maintenance phases whereas automatic tools (The Load Balancer) mainly concentrate on the production phase of the software. Tools will use different sources of information. A monitoring system will give direct access to the running program on the workstation cluster, thus supporting online debugging and performance measurement. All measured characteristics can be recorded in traces which describe the individual behavior of a single program execution. Traces can be used e.g. for statistical program analysis. In addition, checkpoints will be generated which represent the state of a program at a given point of time. Checkpoint data is mainly used for load balancing purposes. Traces and checkpoints can be entered in a database system for comparison, version management etc. All tools are based upon these few information types, however, none of them uses the complete set. Interactive tools will be integrated considering their graphical user interface. Implementation of the GUIs will be based on OSF/Motif and therefore give the users a standardized look and feel.

Finally everything is grouped around PVM version 3.3.x. We expect The Tool-set to be easily adaptable to future versions of PVM as only the monitoring system is closely interconnected with internal PVM mechanisms. However, any change of programming model (e.g. new communication types) would also require an adaption of those interactive tools which should be able to handle the new constructs.

Figure 1 shows the module structure of The Tool-set. Due to the limited space of this paper we can only give a list of some selected highlights and a short description of two of the tools: of The Debugger and of The Performance Analyzer.

- The Debugger can use checkpoint information to resume execution of a program starting from a specific point during the program run. Debugging cycles will become shorter as the program has not to be restarted from the beginning.
- The Performance Analyzer will have extended functionality to try to automatically detect a focus of interest, e.g. workstations with high idle-time or communication frequency. It will also offer statistical data and comparison of different program runs by using traces.
- The Visualizer will show the behavior of PVM programs at the level of tasks and communications thus unveiling e.g. deadlock situations.

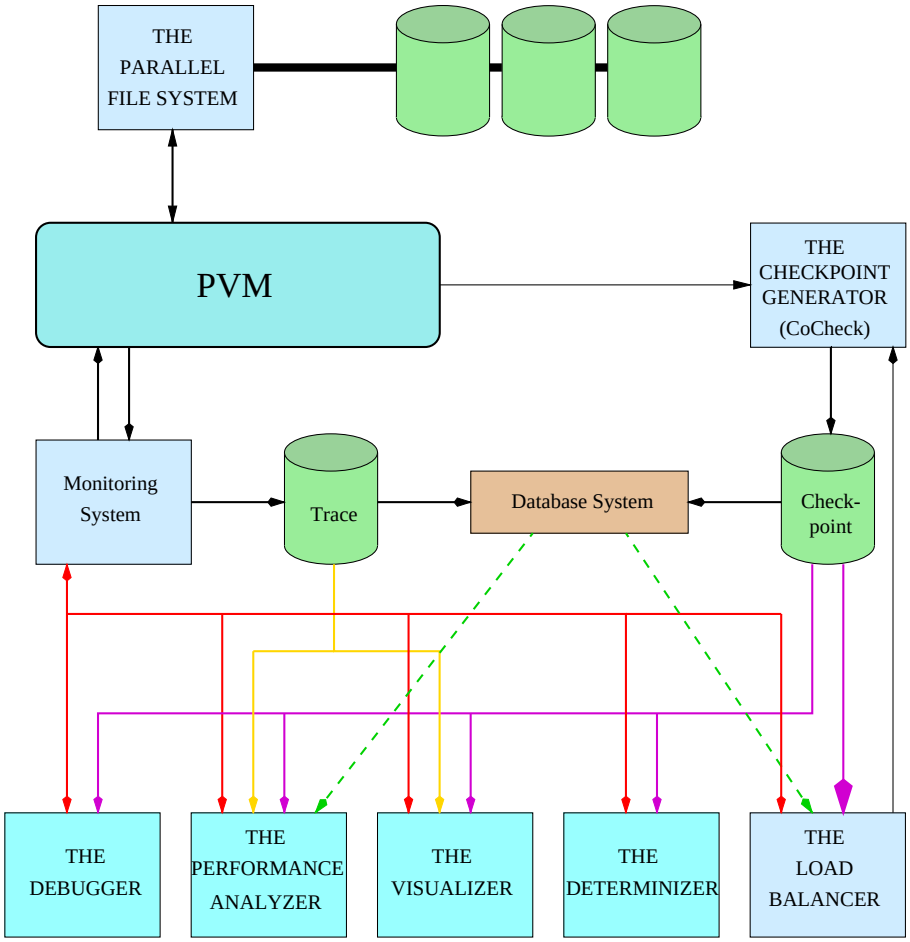

Figure 1: Module structure of the complete tool environment

- The Deterministic Execution Controller (also called The Determinizer) supports deterministic runtime behavior of a parallel program thus ensuring reproducible program runs and results during test phases.
- The Load Balancer can migrate running processes from loaded workstations (e.g. owner wants to work at his console) to free workstations, using the mechanisms of the checkpoint generator CoCheck. Furthermore, the heuristics for controlling migration decisions can be improved by learning from trace material from previous program runs.
- The Parallel File System will support various modes for accessing data files. Moreover, all activities can be monitored and evaluated by The Performance Analyzer.

2.1 The Debugger

The Debugger will provide support for the efficient source level debugging of distributed programs written in C or Fortran 77. In contrast to other projects it is not a simple interface built around a sequential debugger, but a real parallel debugger with special support for PVM. The Debugger offers a comfortable graphical interface which is based on a debugger window showing the source code and command output. In addition, visualizers for regular and also irregular data structures will be provided. The debugger window can be associated with an arbitrary set

of tasks that is displayed in a list. All debugger commands, e.g. single-step, set breakpoint or print, are applied to all these tasks or a selectable subset. So data parallel programs where all tasks execute roughly the same code can be examined using a single window. Multiple windows can be used to debug groups of tasks executing different codes.

Besides the flexible user interface, The Debugger provides a variety of other features essential for parallel programming. We will discuss only the most important ones here. First of all, PVM is fully supported. The Debugger can be used for heterogeneous environments and allows to inspect PVM objects like a task's incoming message queue or a barrier's task queue. Tasks can be identified not only by their ID, but also by additional information, such as host, file and group names or parent task. Based on patterns using this information, dynamically spawned tasks can be automatically stopped at the beginning and may be added to any debugger window.

Second, The Debugger uses the event-action paradigm instead of a simple breakpoint scheme. Events which may be parameterized, represent special conditions in the program, e.g. 'task reaches a source line' or 'message is received'. Each event defined can be associated with a list of actions that are executed when the event occurs. Actions include stopping any set of tasks, tracing the event, printing variables or defining new events and actions. Since actions are evaluated autonomously by the distributed monitoring system without interaction with the debugger's front end or the user, intrusion is kept very low. Furthermore, actions may also be triggered by events on remote hosts, so global halting and distributed breakpoints are possible. We will also provide special breakpoints allowing to follow message transfers: when a task sends a message, the breakpoint will stop the receiver immediately after it has received that message, so data processing can be watched across task boundaries.

Finally, The Debugger will be integrated with The Determinizer and CoCheck, making cyclic debugging practical. Parallel programs usually run for a very long time, so re-running them becomes a tedious job. In addition, program behavior may not be reproducible due to race conditions. Therefore, a form of backtracking will be provided by generating a checkpoint and saving the debugger's configuration upon user request. The user may then return relatively quickly to that point and re-execute the last part of the program. The Determinizer will then either ensure reproducible behavior or may enforce a different message ordering, so the effects of race conditions can be examined.

2.2 The Performance Analyzer

The Performance Analyzer will we able to handle two different kinds of measurements types: online analysis and trace evaluation. Both types offer slightly different functionality and are distinguished by the fact that a trace reflects a fixed program run of the past which includes only a limited amount of measured data. Measures which have not been evaluated during runtime are lost for trace analysis.

Let us have a closer look at the functionality of The Performance Analyzer. It provides analysis on four levels of abstraction: at the level of the virtual machine, the node level, the task level, and the function level. Three types of load characteristics can be evaluated: the amount of computation (CPU-time etc.), communication characteristics like number of bytes sent/received, and I/O characteristics (also volume and time based). An investigation of these measures should give the programmer some idea on possibly inefficient constructs in his program. However, with an increasing number of tasks it becomes difficult to detect the nodes of interest. Therefore, The Performance Analyzer offers special attributed measurements. The nodes which shall contribute to a value are specified via a predicate. If the

predicate is not true, their current value is disregarded. Currently, specification of predicates has to be done manually. In future, we will think about methods of how to add some automatism, e.g. by stepwise strengthening user-defined predicates in order to minimize the set of interesting nodes. By having identified nodes with unusual behavior it would be possible to automatically start a refinement of performance analysis.

Attributed measurements not only make it easier to find bottlenecks but also improve the scalability of the tool. Especially with a higher number of nodes or tasks this feature can reduce the amount of information that has to be displayed on the screen.

Apart from offering current values, mean values, and a graphical representation of values changing during runtime The Performance Analyzer will support enhanced statistical functionality like minimum and maximum values, statistical distributions of values over the time-axis, and may-be user definable combinations of primitive performance measures. Finally, The Performance Analyzer will support the program dynamics of PVM. Not only will it be possible to handle a dynamically changing set of active workstations, but also to evaluate the positive or negative influence of load balancing. This will be achieved by a monitoring system which informs the tool about every change of the current process graph and its mapping.

3 Project Status and Availability

Currently we are implementing The Visualizer, The Parallel File System, and CoCheck. The latter will be available in 9/95. As the monitoring system is not yet implemented, The Visualizer will preliminaryly be based upon traces generated by PVM and XPVM. We have not yet decided whether we will also provide a version of The Performance Analyzer for this trace type.

The next important step to be made is the implementation of the monitoring system which is a prerequisite to all interactive tools and to The Load Balancer. With the monitor being finished we will adapt our existing tools to the instrumented PVM environment. First products should be available for the users in spring 1996. They will be fully functional but will not yet have integrated enhanced features which are current research topics of our group.

References

[1] H. Beier, T. Bemmerl, A. Bode, et al. TOPSYS - Tools for Parallel Systems. Research report SFB 342/9/90 A, Technische Universität München, Jan. 1990.

[2] A. Bode and P. Braun. Monitoring and Visualization in TOPSYS. In G. Kotsis and G. Haring, editors, *Proc. Workshop on Monitoring and Visualization of Parallel Processing Systems*, pages 97 – 118, Moravany nad Váhom, CSFR, Oct. 1992. Elsevier, Amsterdam, 1993.

[3] R. Borgeest. Cyclic Performance Debugging. Submitted for the 2nd European PVM Users' Group Meeting, Lyon, France, Sept. 1995.

[4] O. Hansen. A Tool for Optimizing Programs on Massively Parallel Computer Architectures. In *High-Performance Computing and Networking, Volume II*, volume 797 of *Lecture Notes in Computer Science*, pages 350 – 356, München, Apr. 1994. Springer, Berlin.

[5] T. Ludwig. Aspects of Load Management on Parallel Computers. In V. Malyshkin, editor, *Proc. International Conference Parallel Computing Technologies, PaCT-93, Obninsk, Russia*, pages 301–313, Vol. II, Moscow, Russia, Sept. 1993. Recursive Super Computers (ReSCo).

[6] M. Oberhuber. Elimination of Nondeterminacy for Testing and Debugging Parallel Programs. In *Proc. of the 2nd Int. Workshop on Automated and Algorithmic Debugging (AADEBUG'95)*, St. Malo, France, May 1995.

[7] M. Oberhuber and R. Wismüller. DETOP - An Interactive Debugger for PowerPC Based Multicomputers. In P. Fritzson and L. Finmo, editors, *Parallel Programming and Applications*, pages 170–183. IOS Press, May 1995.

[8] C. Röder, S. Lamberts, and T. Ludwig. PFSLib - An I/O Interface for Parallel Programming Environments on Coupled Workstations. Submitted for the 2nd European PVM Users' Group Meeting, Lyon, France, Sept. 1995.

[9] G. Stellner and J. Pruyne. Resource Management and Checkpointing for PVM. Submitted for the 2nd European PVM Users' Group Meeting, Lyon, France, Sept. 1995.