THE TOOL-SET Project: Towards an Integrated Tool Environment for Parallel Programming*

Roland Wismüller, Thomas Ludwig, Arndt Bode, Rolf Borgeest,
Stefan Lamberts, Michael Oberhuber, Christian Röder, Georg Stellner
Lehrstuhl für Rechnertechnik und Rechnerorganisation (LRR-TUM)
Institut für Informatik, Technische Universität München, D-80290 München, Germany
Tel.: +49-89-2105-8164 or -2042, Fax: +49-89-2105-8232
email: {wismuell,ludwig}@informatik.tu-muenchen.de

Abstract

Writing parallel programs efficiently is still a problematic issue due to the lack of a uniform and integrated tool support. A new project at LRR-TUM works towards changing this situation for the most popular programming environment on workstation clusters: for PVM. THE TOOL-SET for PVM will comprise a set of integrated tools, consisting of a debugger, a performance analyzer, a visualizer, a deterministic execution controller, a load balancer including a checkpoint generator, and a parallel file system. In contrast to previous work, these tools are compatible to each other by using a common monitoring system, thus avoiding special preparation of a parallel program for each single tool. In addition they can be used together to observe and control different aspects of a single program run.

Keywords: Parallel Tool Environments, On-line Tools, PVM, Tool Integration

1 Introduction

Efficient parallel programming requires a huge amount of support by sophisticated tools, which range from special-purpose programming libraries over run-time tools providing insight into the execution behavior of a program to comfortable execution environments. Although in the last years a large variety of this kind of tools has been developed, the situation is still not satisfying for users, since most of these tools can only be used in isolation and cannot work together with other tools. Usually it is not even possible to use different tools with the same version of a parallel program. The reason for this situation lies in the fact that current tools use proprietary monitoring systems for gathering the required information, which are not compatible to other ones, or require the application program to be prepared in a special way, e.g. by a special preprocessing, a special compilation or by linking with special library versions, where each tool needs a preparation different from that of other tools.

What is currently missing is something like an integrated and iteroperable tool environment covering all aspects of programming and development support. By "integrated" we do not mean that there should be one big tool with a single user interface that is able to support everything, but a collection of specialized tools that can work together and offer uniform interfaces. There are three levels of integration that should be achieved in a tool environment:

Level 1: Different tools should be individually usable for the same version of a parallel program.

Level 2: It should be possible to use several tools at the same time for the same program. This allows to have a set of specialized tools, each supporting different aspects of parallel programming, which is beneficial for both tool developers, since modularity is increased, and for users, since they only have to acquaint themselves with the tools they actually need.

Level 3: Ideally, different tools should support and supplement each other.

For the reasons described above, most existing tools do not even meet the first requirement. Levels 2 and 3 are currently far from being reached by existing tools.

*Partly funded by the German Science Foundation, Contract: SFB 342, TP A1

A new project at LRR-TUM aims towards changing 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 LRR-TUM that is performing research in the field of parallel processing tools for more than eight years. It integrates and adapts our existing tools and makes use of our long-standing experience to form a PVM tool environment supporting parallel I/O, load balancing, resource management and checkpointing, performance analysis, debugging, deterministic execution, and visualization. By using a common OMIS-compliant on-line monitoring system that acquires the necessary information on the running program and also performs and coordinates the necessary manipulations of program execution, the integration levels 1 and 2 stated above can be reached. However, level 3 requires more research work to be done and is a more long term goal.

2 Existing PVM Tools

In the last years, several tools were developed to give programmers support during all phases of the parallel software life cycle. Relevant phases cover specification and design, implementation (including debugging), maintenance (including performance debugging), and production (using load balancing and fault tolerance). Due to the limited space of this paper, we cannot give a comprehensive overview, but will just enumerate some tools and present an evaluation. Since in the first phase, THE TOOL-SET project focuses on support for PVM, we restrict this enumeration to PVM tools.

There are a few design tools for PVM (HeNCE [3], TRAPPER [19]) that support both a graphical design of parallel programs and the analysis of run-time behavior. In addition, there is a variety of run-time tools allowing to analyze the execution behavior of PVM programs, ranging from simple to rather sophisticated ones: Xab [2] is a simple tracing tool offering a textual display of events; XPVM [8] includes more sophisticated displays of traced events and a graphical PVM console. DBPVM [17] is a graphical interface for controlling multiple sequential debuggers, while P2D2 [6] and TotalView [1] are native parallel debuggers supporting PVM. Performance debugging is supported by the ParaDyn [14] tool, which offers an automated guidance to performance bottlenecks. There are also several tools for PVM that are not used interactively, but are part of the run-time environment, supporting e.g. parallel file I/O (PIOUS [15]) and automatic load balancing (DynamicPVM [7]; MPVM/UPVM [5]).

The overview shows that there is already nearly a dozen of more or less powerful tools for PVM. However, there is nothing like an integrated tool environment covering several of the software life-cycle phases. HeNCE and TRAPPER are a kind of exception, since they cover design, configuration and analysis. However, without additional work these tools cannot be used to analyze programs generated manually or by other tools. They also cannot be used in combination with other tools. This is true for all the tools presented above. The main reason for this is that they either require special changes of the source code or use monitoring systems and specially instrumented versions of PVM that are not compatible with implementation concepts of other tools.

Another reason why these tools cannot be used together is simply that no one ever tried to make them compatible to each other. Since these tools have been developed by different research groups, this would either require a large joint project or a team that has detailed knowledge on the implementation of all of the tools. Obviously, this is not feasible. Furthermore, the tools presented above are designed very differently, rely on different user interface libraries and use very different techniques to acquire data.

On the other hand, LRR-TUM is in the situation of having developed a rather complete set of tools within the same research group. Although also these tools are currently not integrated, we have good prerequisites for an integration, since the implementation knowledge is focused, the tools are designed in a similar fashion and use identical techniques for data acquisition. Thus we have started a project called THE TOOL-SET to overcome the drawbacks of existing PVM tools by providing an integrated approach for different kinds of tools. THE TOOL-SET will support implementation, maintenance, and production phases of the parallel software life-cycle and thus give the programmer a consistent concept for developing efficient parallel programs.

In the following section, we will give an overview on the structure of THE TOOL-SET, the already existing parts, and the additional work necessary to put these parts together to a uniform tool environment.

3 The Tool-set

3.1 The Structure of The Tool-set

The Tool-set includes different kinds of tools that can be divided into interactive ones and automatic ones. Interactive tools (like e.g. The Debugger[1]) support implementation and maintenance phases whereas automatic tools (e.g. The Load Balancer) mainly concentrate on the production phase of the software. A common monitoring system gives the tools direct access to the running parallel program, thus supporting e.g. on-line debugging and performance measurement. All acquired data can be recorded in traces which describe one individual behavior of a program run. 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 needed for load balancing and fault tolerance purposes. We also plan to use a database where traces and checkpoints can be entered for comparison, version management etc.

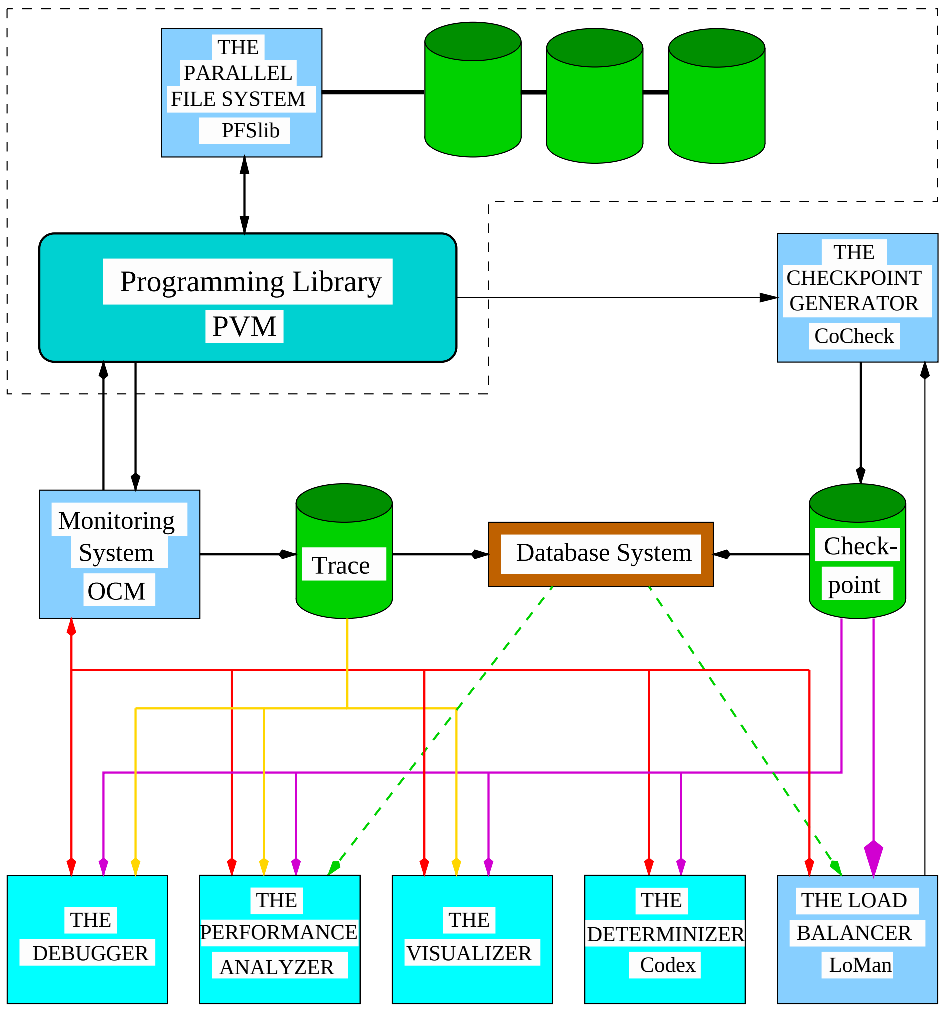

Figure 1: Module structure of the complete tool environment

Figure 1 shows the module structure of The Tool-set. It consists of three automatic tools that can be viewed as being part of an application's operating environment: a parallel file system, a checkpoint generator, and a load balancer. Interactive tools include a source level debugger, a performance analysis system, a program flow visualizer, and a tool enforcing deterministic execution. They provide graphical user interfaces based on OSF/Motif and therefore give the users a standardized look and feel. All these tools use a common OMIS compliant monitoring system (OCM) for acquiring data on the application's execution, but also to manipulate the execution, where necessary. The whole tool-set is currently grouped around PVM version 3.3. We expect it to be easily adaptable to

[1] We don't have decided yet what the names of all of our tools will be once they are integrated into The Tool-set. Therefore we refer to some of them as The Debugger, The Load Balancer etc.

future versions of PVM as only the monitoring system is closely interconnected with internal PVM mechanisms.

In the following subsection, we will briefly present the individual tools in THE TOOL-SET and their current implementation status.

3.2 The Building Blocks of THE TOOL-SET

3.2.1 Parallel File System

Most programming environments lack sophisticated I/O facilities, restricting programmers in writing applications that operate on large amounts of data. Therefore, THE TOOL-SET includes a parallel file system that is implemented by the PFSLib library [10][2] and offers the functionality of the parallel file system PFS used with the Intel Paragon computers. The design and implementation of PFSLib is independent from any message passing library, thus providing a universal I/O interface for different environments, including PVM and MPI.

3.2.2 Checkpointer and Load Balancer

On networks of workstations, we do not only need load balancing for improving the performance of parallel applications, but also process migration facilities allowing to remove parts of a parallel application from a workstation when an interactive session is started. In addition, long running programs require fault tolerance. All of these requirements are supported by a transparent checkpointing mechanism allowing to migrate processes and to save checkpoints on disk for a later restart. CoCheck [20] implements such a mechanism[3]. The basic idea for providing consistent checkpoints is to guarantee that the communication network is stateless, i.e. there are no longer any messages in transit. After that each process of the application can create a local checkpoint file or can be migrated to another computing host.

Based on the migration mechanism provided by CoCheck, we are developing an integrated load management facility (LoMan) for parallel programs on heterogeneous clusters of workstations [18]. LoMan dynamically distributes application tasks to a possibly changing set of execution nodes. Reconfiguration is viable in case of load changes, failure/recovery and activation/finishing of interactive workstation usage. In contrast to other approaches LoMan covers both aspects of heterogeneity: different performance indices of target workstations and application requirements are matched during the load modeling phase, different hardware configurations and operating systems of target workstations are considered during the redistribution phase.

3.2.3 Debugger

THE DEBUGGER is based on a parallel debugging tool called DETOP [22] that has been developed by LRR-TUM within a special research grant (SFB 342) and a cooperation with Parsytec, a German vendor of supercomputers and parallel embedded systems[4]. In contrast to other tools it is not a simple interface built around a sequential debugger, but a real parallel debugger with special support for parallel programming models. The debugger offers a graphical interface, allowing efficient control and observation of multiple tasks within a single debugger window. It is able to handle different executable files at the same time. Thus, support for non-SPMD applications and also for heterogeneous systems is provided. In order to ease debugging of programs that dynamically create new tasks, tasks can be identified not only by their numerical ID, but also by additional information, e.g. file name and parameters. Based on patterns using this information, dynamically spawned tasks can be automatically stopped at the beginning of their execution and added to a debugger window.

Although the current implementation is adapted to Parsytec computers, porting to PVM will be easy once the monitoring system for PVM is available, since the debugger is already split into an interface part that is independent of the target system and a monitoring system encapsulating all platform dependencies.

3.2.4 Performance Analyzer

THE PERFORMANCE ANALYZER will we able to handle two different kinds of measurements types: online analysis and trace evaluation. It thus unifies two tools of our current performance analysis environment: PATOP and TATOO. PATOP [22] has been developed within the special research grant

[2] PFSLib is available under the terms of the GNU license from ftp://ftpbode.informatik.tu-muenchen.de/PFSlib

[3] CoCheck is available under the terms of the GNU license via http://wwwbode.informatik.tu-muenchen.de/CoCheck

SFB 342 and an industry cooperation with Parsytec[4], TATOO [4] is part of the European Esprit project HAMLET. Both tools use the same graphical user interface and largely identical routines to process performance data. The essential difference is the method of data acquisition. While PATOP uses an on-line monitoring system which is the same as for our debugger, TATOO is based on event traces that are processed off-line. The performance analysis system provides analysis on three levels of abstraction: at the level of the virtual machine, the node level, and the task or thread level. Measurements can be related to either the complete application program, individual functions, or user defined regions in the source code. Three types of load characteristics can be evaluated: the amount of computation (CPU-time etc.), communication characteristics like number of bytes sent/received and delay in waiting queues, and I/O characteristics (also volume and time based).

Since the performance analysis environment is also divided into an interface part and the monitoring system, but is much less dependent on the target system than the debugger, it can be ported even more easily.

3.2.5 Program Flow Visualizer

THE VISUALIZER is an enhanced version of a visualization tool called VISTOP developed at LRR-TUM [21]. The basic idea of the visualizer is to provide a global, high level view of an application's parallel execution at an abstract level. The main features compared to other approaches are its on-line mode of operation, the internal modeling of consistent global states of the monitored application, and the use of a formal time model based on causal relationships and constraints between events and the modeled global state to enforce a consistent visualization. The system visualizes a program's execution as a sequence of snapshots where each snapshot shows different views of the program's global state. Currently, the visualizer provides three main views of program state: the *concurrency view* showing tasks states, communication and synchronization, the *object creation view* displaying parent-child relationships between tasks and the *system view* showing the mapping from tasks to hosts. For visualizing groups, there is an additional view, called *group manager*.

The current implementation of THE VISUALIZER is fully operational in an off-line mode based on traces generated by XPVM, but since the on-line monitoring system is not yet finished, the on-line features are not available. These features include the ability to start data acquisition and visualization in the middle of a program execution, which requires the current global state to be read from the application.

3.2.6 Deterministic Execution Controller

Since parallel programs include an inherent nondeterminism due to the possibility of message races, a tool environment supporting debugging must include a means to enforce deterministic program execution. This is essential for reproducing bugs in the program runs observed by a parallel debugger. The usual approach to this problem is to use replay techniques that record the communication behavior of a program execution and force subsequent executions to exhibit the same behavior. However, these techniques have two severe disadvantages:

1. Since the recording process influences the program's execution due to the probe effect, it may be impossible to reproduce an erroneous behavior with recording turned on.

2. Replay does not offer any support for testing, i.e. if a program contains an unintended race leading to an error in some program runs, there is no way to enforce a program execution unveiling the error. Programmers have to wait until the error occurs by chance.

Therefore, we follow another approach that enables the user to eliminate nondeterminism by specifying access patterns for common objects (e.g. the receive queue of a task). At runtime, these patterns are used to enforce a total ordering of interprocess communication and thus a deterministic execution [16]. Currently, the deterministic execution controller (called Codex) is in the state of a research prototype.

3.2.7 On-line Monitoring System

From the above, it is evident that the monitoring system is a key part of the presented tool environment. It not only acquires the run-time data needed by the tools and manipulates the program's execution where necessary, but is also the common part of all of the tools. Thus, designing the monitor's interface in such a way that it meets the requirements imposed by the different tools and is able

[4] DETOP and PATOP are commercially marketed by Parsytec

to serve several tools at the same time is a key issue for the implementation of an integrated tool environment. Because of its importance for The Tool-set and also for other tool developments in the field, the specification of the monitoring interface has developed itself into an independent research project, called OMIS (On-line Monitoring Interface Specification) [13, 12]. The goal of OMIS is to define a standardized, flexible, and extendible on-line monitoring interface usable for all tools that need on-line data and control of a running parallel application. Version 1.0 of the specification is available since Feb. 1996[5]; an updated version 2.0 is in the state of a final draft and will be available in Summer 1997.

We are currently implementing an OMIS compliant monitoring system (OCM) for the PVM programming library on networks of workstations [11]. At a coarse grained view, the monitoring system consists of a local monitor process per processing node and a central process responsible for the management and distribution of monitoring requests and the proper assembly of replies (see Fig. 2). For efficiency reasons, there are also parts of the monitor linked into the application processes. These parts are responsible for detecting relevant events in the application process and for performing simple actions, e.g. incrementing counters or starting and stopping timers. The necessary communication with the monitor process is handled via a shared memory segment. A first version of the OCM will be available in Fall 1997.

Additionally, we are also implementing a Node Status Reporter (NSR) as a strongly interrelated but decoupled part of the OCM. The NSR enables the on-line observation of all available workstations even if some of the workstations are currently not involved in the parallel computation [9]. For example, a load management system might use the information provided by the NSR to decide for dynamic reconfiguration of the execution nodes.

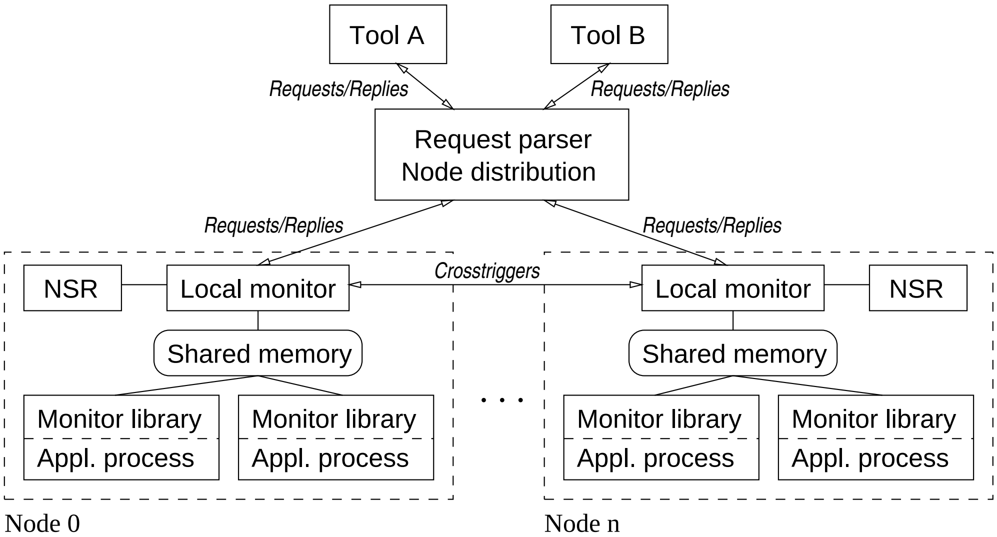

Figure 2: Structure of the OMIS compliant monitoring system

3.3 Putting the Parts Together

When the implementation of the monitoring system has been finished, the integration levels 1 and 2 defined in Section 1 are already obtained. The third level of integration is a research task where we will investigate the possible advantageous (and also disadvantageous) interactions between the different tools. We will especially focus on two scenarios:

1. A combination of The Visualizer, The Debugger, CoCheck and Codex can greatly simplify and accelerate the task of debugging parallel programs. The Visualizer currently provides a good means to view high-level program execution and to detect the presence of e.g. communication errors. However, localizing the reason of an error in the source code is not possible with that tool alone. What is needed for this purpose is to start an interactive debugger on the program when it is in the state displayed by the visualizer. Since this is a past state, the program has to be reset in some manner. This can be achieved in an efficient way by using both periodic checkpointing and deterministic re-execution.

[5] URL: http://wwwbode.informatik.tu-muenchen.de/~omis

2. Workstation clusters for parallel computing are more and more controlled by resource management systems which can also incorporate process migration and load balancing. For the use of on-line tools process migration creates several problems, since the monitoring activities also have to migrate in this case. On the other hand, tools like e.g. performance analyzers can provide valuable insights into the effectiveness of a load balancer based on process migration. The load balancer might even use this performance information to automatically adapt its heuristics.

Other kinds of interactions we will investigate within the framework of THE TOOL-SET include:

- THE PERFORMANCE ANALYZER and THE VISUALIZER can be used to watch the operation of PFSlib for optimizing file I/O.
- PFSlib can be used by the monitoring system or CoCheck to efficiently write trace or checkpoint data.
- THE PERFORMANCE ANALYZER and THE DEBUGGER can be used in combination to perform efficient performance analysis of selected phases in a program's execution or for quick experimentation and steering (e.g. by on-line modification of parameters in an application)

4 Project Status and Availability

Obviously, an ambitious tool environment cannot be designed and implemented within a few months as there are many aspects and interdependencies that have to be checked carefully. PFSlib, CoCheck and THE VISUALIZER are already fully operational for PVM. As the monitoring system is not yet fully implemented, THE VISUALIZER preliminaryly is based upon traces generated by XPVM.

The next important step to be made is the implementation of the OMIS compliant monitoring system and the adaption of our existing tools to the monitored PVM environment. Since the monitoring system encapsulates all of the platform dependent parts of THE TOOL-SET, adapting the debugger and the performance analyzer does not require a big effort, so first versions will be available soon after the monitoring system in Fall 1997.

With all tools being available, we will use THE TOOL-SET for both parallel application programming as well as performing research on interactive and automatic tools and their integration. All components of THE TOOL-SET will finally be available to the public under the GNU license conditions[6].

References

[1] BBN Systems and Technologies. *TotalView User's Guide, Version 3.4*, Mar. 1995.

[2] A. Beguelin. Xab: A tool for monitoring PVM programs. In *Workshop on Heterogeneous Processing*, pages 92–97, Los Alamitos, California, Apr. 1993.

[3] A. Beguelin, J. Dongarra, A. Geist, R. Manchek, and K. Moore. HeNCE: A Heterogeneous Network Computing Environment. Technical report UT-CS-93-205, Univ. of Tennessee, Computer Science Department, Aug. 1993.

[4] R. Borgeest, B. Dimke, and O. Hansen. A trace based performance evaluation tool for parallel real time systems. *Parallel Computing*, 21(4):551 – 564, April 1995.

[5] J. Casas, R. Konuru, S. W. Otto, R. Prouty, and J. Walpole. Adaptive Load Migration Systems for PVM. In *Proc. Supercomputing'94*, pages 390–399, Washington, D.C., Nov. 1994. IEEE Comp. Soc. Press.

[6] D. Cheng and R. Hood. A Portable Debugger for Parallel and Distributed Programs. In *Proc. Supercomputing'94*, pages 723–732, Washington, D.C., Nov. 1994. IEEE Comp. Soc. Press.

[7] L. Dikken, F. van der Linden, J. Vesseur, and P. Sloot. Dynamic PVM - Dynamic load balancing on parallel systems. In *High-Performance Computing and Networking, Volume II*, pages 273–277, München, Apr. 1994. Springer. LNCS 797.

[6] Information will be available on our Web page `http://wwwbode.informatik.tu-muenchen.de/Par/tools`

[8] A. Geist, J. Kohl, and P. Papadopoulos. Visualization, Debugging, and Performance in PVM. In *Proc. Visualization and Debugging Workshop*, Oct. 1994.

[9] M. Hilbig. Design and Implementation of a Node Status Reporter in Networks of Heterogeneous Workstations. Master's thesis, LRR-TUM, Technische Universität München, Dec. 1996.

[10] T. Ludwig and S. Lamberts. PFSLib — A Parallel File System for Workstation Clusters. In V. Malyshkin, editor, *Proc. 3rd International Conference on Parallel Computing Technologies, PaCT-95, St. Petersburg, Russia*, pages 246–251, Berlin, Sept. 1995. Springer. LNCS 964.

[11] T. Ludwig, R. Wismüller, and M. Oberhuber. OCM – An OMIS Compliant Monitoring System. In A. Bode, J. Dongarra, T. Ludwig, and V. Sunderam, editors, *Parallel Virtual Machine – EuroPVM'96*, pages 81–90, Munich, Germany, Oct. 1996. Springer. LNCS 1156.

[12] T. Ludwig, R. Wismüller, M. Oberhuber, and A. Bode. An Open Interface for the On-Line Monitoring of Parallel and Distributed Programs. *Intl. Journal of Supercomputer Applications and High Performance Computing*, 11(2), Summer 1997. to appear.

[13] T. Ludwig, R. Wismüller, V. Sunderam, and A. Bode. OMIS — On-line Monitoring Interface Specification (Version 1.0). Technical Report TUM-I9609, SFB-Bericht 342/05/96 A, Technische Universität München, Munich, Germany, February 1996.

[14] B. P. Miller, J. K. Hollingsworth, and M. D. Callaghan. The Paradyn Parallel Performance Tools and PVM. Technical report, Department of Computer Sciences, University of Wisconsin, 1994.

[15] S. A. Moyer and V. S. Sunderam. PIOUS: An Architecture for Parallel I/O in Distributed Computing Environments. In *Workshop on Cluster Computing*, Tallahassee, FL, Dec. 1993.

[16] M. Oberhuber. Elimination of nondeterminacy for testing and debugging parallel programs. In M. Ducassee, editor, *Proc. 2nd Int. Workshop on Automated and Algorithmic Debugging*, May 1995.

[17] K. Perumalla. DBPVM: A Motif-based Graphical Interface to PVM. Report from Georgia Tech's College of Computing, Atlanta, Mar. 1995.

[18] C. Röder and G. Stellner. Design of Load Management for Parallel Applications in Networks of Heterogeneous Workstations. Technical Report SFB 342/18/97 A, SFB 342, Technische Universität München, 80290 München, Germany, May 1997.

[19] C. Scheidler and L. Schäfers. TRAPPER: A Graphical Programming Environment for Industrial High-Performance Applications. In A. Bode, M. Reeve, and G. Wolf, editors, *Proc. PARLE'93, Parallel Architectures and Languages Europe*, pages 403–413, München, June 1993. Springer, Berlin. LNCS 694.

[20] G. Stellner. CoCheck: Checkpointing and Process Migration for MPI. In *Proceedings of the International Parallel Processing Symposium*, pages 526–531, Honolulu, HI, Apr. 1996. IEEE Computer Society Press.

[21] R. Wismüller. State Based Visualization of PVM Applications. In A. Bode, J. Dongarra, T. Ludwig, and V. Sunderam, editors, *Parallel Virtual Machine – EuroPVM'96*, pages 91–99, Munich, Germany, Oct. 1996. Springer. LNCS 1156.

[22] R. Wismüller, M. Oberhuber, J. Krammer, and O. Hansen. Interactive debugging and performance analysis of massively parallel applications. *Parallel Computing*, 22(3):415–442, Mar. 1996.