Observing Parallel Programs at a Flexible Level of Granularity

Rainer Klar, Andreas Quick, Markus Siegle
Computer Science Department (IMMD VII)
University of Erlangen-Nürnberg
Martensstraße 3, D-8520 Erlangen, Germany
email: siegle@informatik.uni-erlangen.de

Abstract: A new method of carrying out hybrid monitoring for the observation of parallel programs is presented. A comprehensive set of interacting tools, the PEPP-ZM4-SIMPLE environment, is briefly discussed. These tools support modelling, measurement preparation, monitoring, event trace analysis and assist in result interpretation. The emphasis of the paper is on the flexibility of the method with respect to the level of abstraction. It is shown that the level of granularity of observation is user-determined and can be changed with minimal effort using our tools. An application example, in which the dynamic behaviour of a parallel program on SUPRENUM is analyzed, illustrates the benefits of the method.

Keywords: granularity, event-driven hybrid monitoring, parallel programming, stochastic graph model

1. Introduction — The Method

Insight into the dynamic behaviour of parallel computers is a prerequisite for efficiently using their vast resources. The observation of parallel programs — a means through which insight can be obtained — is of great importance to the application programmer, but poses great challenges to developers of parallel programming tools. Some ready-to-use and easy-to-handle techniques, such as profiling, have the capability to provide users quickly with performance data about the execution of their programs. However, this purely statistical information is often found to be insufficient for answering important questions about the reasons for the manifest behaviour, since the inflexible granularity of observation — in this case the procedure level — does not allow to focus attention on desired details. More advanced methods, which can disclose the functional and timing behaviour of the program at a level of detail suitable for the analysis of correctness and performance issues, are often costly and cumbersome. One such example is the collection of extensive memory reference traces [FNAG92], which may cause serious disturbance of the system under study, and which makes it difficult to extract relevant information from the overwhelming flood of trace data. This problem is alleviated by parallel programming environments such as the portable communication library PICL [GHPW90] where only communication activities are instrumented.

Exceeding indiscriminating event trace collection, *event-driven hybrid monitoring* permits the recording of user-defined events bearing problem-oriented identifiers. This makes it possible to

observe the behaviour of a parallel program only at points of interest, i.e. at a flexible level of detail. For instance, selected portions of the code may be observed at the memory access level, while others are observed at the procedure level. This feature also helps to make the method applicable to such a wide variety of systems as message-passing systems and systems with shared-variable communication. In conjunction with the use of a model (see below), the technique allows multiple views on a problem while causing only negligible intrusion. The attention of the user is thereby directed to those sections of the program which are relevant in terms of correctness and performance. Thus, hybrid monitoring supports both debugging and tuning of parallel programs.

Our method is called *model-driven monitoring* and can be briefly described as follows: Based on a functional model of the program under study (e.g. a stochastic graph model or a Petri net model), the user specifies instrumentation points. Model constructs have the same names as their corresponding parts in the program. Therefore the instrumentation of the program can be done automatically. During the following execution of the instrumented program, events are issued when trace statements at instrumentation points are executed. These events are recorded by an external hardware monitor system. In contrast to software monitoring, the use of an external recording device prevents the deprivation of system resources and therefore ensures negligible perturbation. Furthermore, globally valid time stamps which are rarely available inside the system under study, can be provided by the monitor system. The resulting event trace, consisting of well-defined event records which correspond to the instrumentation points in the model, is then validated by checking whether the observed sequence of events is a possible event sequence of the model. Programming errors (inconsistencies between the intended behaviour specified by the model and the observed program behaviour) can be detected during this consistency check. In case of a positive outcome of the validation, the event trace is analyzed in order to obtain both performance indices of the program (time distributions, frequency attributes) and insight into the program's dynamic behaviour. Parts of this evaluation process can be automized through the knowledge about the functional program behaviour included in the model.

During event trace evaluation, there may remain open questions because of an insufficient number of instrumentation points. In this case, another refined view can be easily obtained by repeating the described steps, changing the level of abstraction in the model and thereby changing automatically the granularity of instrumentation. Developing complex parallel application codes, this cycle is typically repeated several times. Every new instrumentation will focus on only a few sections of the program under study, to which the user has been directed by the results of the previous measurement. These are the hot-spots which impair program performance.

2. The Tools

The method of event-driven hybrid monitoring is supported by a comprehensive set of tools developed at the University of Erlangen-Nürnberg. There is the modelling tool PEPP, the distributed monitor system ZM4 and the event trace analysis package SIMPLE.

2.1. Modelling

The package PEPP [DHK+92] supports the design and analysis of stochastic graph models and has special features which make the tool suitable for model-driven monitoring. The name PEPP stands for *Performance Evaluation of Parallel Programs*. User interaction with PEPP is done

via a comfortable graphical interface which eases model design and editing. Pop-up menus and context-dependent help facilities guide users through the modelling procedure.

Stochastic acyclic graphs [ST87], often also called task dependence graphs or program graphs, are a paradigm well-suited for the modelling of parallel programs [SW90]. Nodes of the graph represent program activities, and arcs are used to specify dependences between nodes (an arc from node i to node j states that processing of node j may only commence after processing of node i has terminated).

PEPP supports five different node types:

- Elementary nodes
- Parallel nodes, representing *n* identical activities running in parallel
- Cyclic nodes, representing the iteration of an activity
- Hierarchical nodes which contain a complete subgraph of arbitrary complexity
- Hierarchical loop nodes which can be used to model loops with a complex loop body

Cyclic nodes are needed to model program loops in the context of strictly acyclic program graphs. With the last two node types, PEPP supports the concept of hierarchical modelling which assists the user in regarding the modelling problem at different levels of granularity. The runtime distribution of PEPP nodes can be deterministic, exponential, branching Erlang, or consisting of one deterministically and one exponentially distributed phase. Furthermore, measured distributions can be employed: Measurement results can be transformed into empirical distribution functions by the tool CAPP (*Calculation And Presentation Package*) and then used in the model. The use of measured distribution functions yields realistic parameters for performance modelling and therefore relevant results.

The great variety of program and model structures requires different analysis techniques. PEPP offers a multitude of evaluation methods, including exact and approximate analysis [Söt90, HM92] of the mean execution time of the graph model. Classical state space analysis becomes prohibitively expensive for large models because of the combinatorial state space explosion. Therefore approximate state space analysis was developed whose computation time is about two orders of magnitude smaller. Further approximation was introduced by bounding methods, permitting efficient computation of lower and upper bounds for the mean execution time of models which cannot be analyzed otherwise. The implementation of bounding methods showed that they lead to results comparable in quality to approximate state space analysis while being much less time-consuming. Series-parallel graph structures can be analyzed using series-parallel reduction. For graph models containing series-parallel subgraphs this technique can be used to simplify the model significantly before performing state space analysis.

The nodes of the graph model are given the same names as their corresponding parts of the modelled program. After the specification of instrumentation points in the model by the user, a command file for the automatic instrumentation tool AICOS (*Automatic Instrumentation of C Software*) is generated by PEPP. AICOS modifies the source code of the program under investigation by inserting monitoring statements at the points which were specified in the model. Instrumentation of program sources with a tool is much less error-prone than manual instrumentation with a text editor. This procedure guarantees a systematic approach to the abstraction of the program under study to events of interest. The use of a model forces the user to clearly understand the structure of the program and to limit the set of instrumentation points to the ones which are necessary for the current considerations.

The program behaviour manifest in a measured event trace can be animated and validated by the model in a stepwise fashion or in quick mode [DKQ92]. In this context, validation refers to a consistency check between the observed program behaviour and the model. It is checked whether the measured event trace represents a possible event sequence of the model. During validation, the nodes of the model are graphically highlighted as “ready-to-execute” or “done”, according to the current position in the trace. The occurrence of premature events is detected as an error in the trace. In addition to the features mentioned already, PEPP allows the automatic generation of description and command files for the tools of the SIMPLE package (see Section 2.3) to be used during trace evaluation.

2.2. Monitoring

For the measurement we use the distributed hardware monitor system ZM4 [Kla92]. ZM4 is already the fourth generation of monitors developed at the University of Erlangen-Nürnberg. This monitor system has been designed for the observation of arbitrary parallel and distributed systems. Therefore, ZM4 itself is a distributed system. It is not dedicated to one particular architecture but can be easily adapted to any conceivable parallel or distributed system, including homogeneous multiprocessors, clusters of workstations and distributed real-time systems.

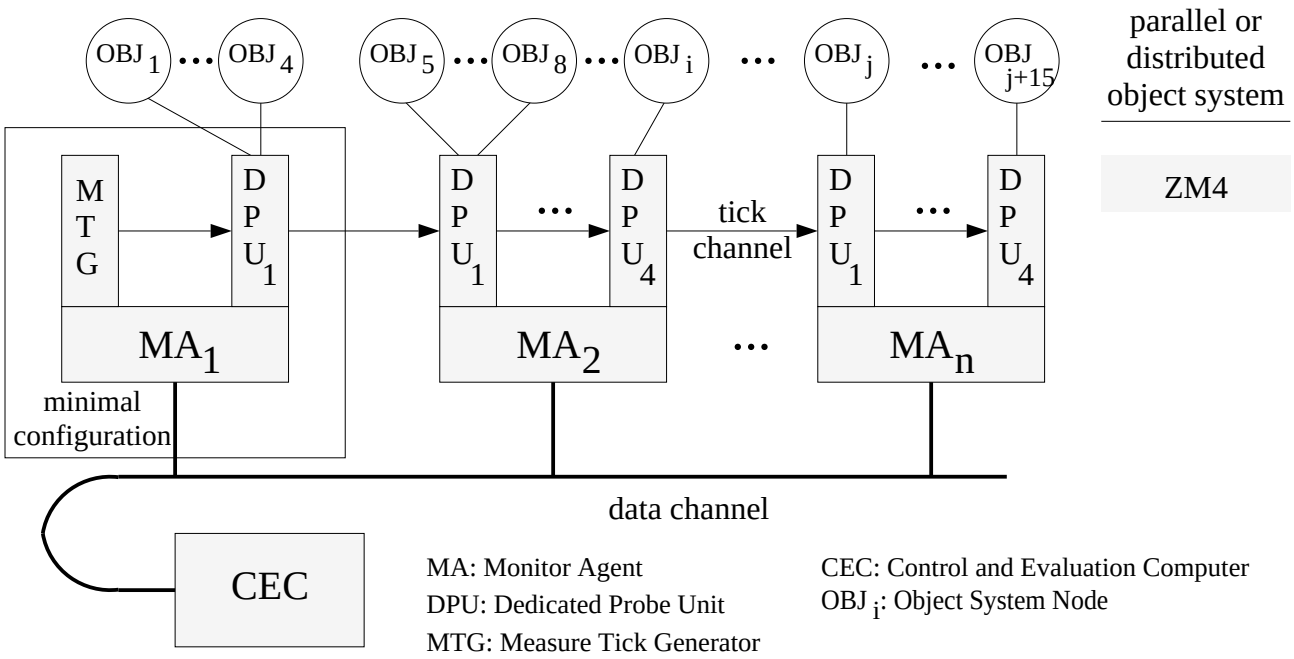

Figure 1: General Configuration of the ZM4 Hardware Monitor System

Figure 1 shows the general configuration of ZM4, consisting of one control and evaluation computer (CEC) and a scalable number of monitor agents (MAs) interconnected by the data channel (Ethernet with TCP/IP). The CEC is used for preparing and controlling the measurement and for the analysis of the measured data.

The monitor agents are standard PCs enhanced with special-purpose hardware. Each monitor agent is equipped with up to four dedicated probe units (DPUs) whose essential tasks are the observation of the nodes of the object system (OBJ_i), time-stamping and high-speed buffering of incoming events.

The measure tick generator (MTG) constitutes the master part of ZM4's global clock. It is connected to the local high-precision clocks of the DPUs by the tick channel, allowing a synchronization of these clocks under a sophisticated fault-tolerant protocol [Hof93]. The capability of providing globally valid high-precision time stamps in a distributed environment distinguishes ZM4 from other monitors (e.g. [BLT90, Mal89, MCNR90]). It makes the system most suitable for parallel program analysis, during which it is of great importance to compare the acquisition time of events on different nodes. This is needed in order to relate those events to each other and thereby obtain a global view of the program's dynamic behaviour, including the interprocessor communication.

2.3. Event Trace Analysis

SIMPLE (*Source-related and Integrated Multiprocessor and -computer Performance evaluation, modeLing and visualization Environment* [Moh92a, Moh92b]) is a package of software tools supporting the analysis of event traces. Traces originating from ZM4 or any other monitor or simulation tool may be analyzed with SIMPLE. Access to arbitrarily formatted traces is possible through the use of a trace description language (TDL) and a problem-oriented access interface (POET).

A description of the trace format and trace semantics in TDL can be either written manually or generated from the model automatically by the tool PEPP, as described in Section 2.1. The latter is possible because instrumentation points and event semantics are known from the instrumentation. With the help of the TDL description, the tools of the SIMPLE package can access the trace through the standardized POET interface.

The package comprises tools for statistical trace analysis as well as dynamic analysis in terms of visualization and animation of the program behaviour exhibited in the trace. It is our experience that Gantt diagrams (time-state diagrams) are most helpful for visualizing causal and temporal properties of a program's execution. During the trace analysis, information is presented referring to the identifiers of program activities familiar to the user. It is the same set of identifiers which had also been used previously during modelling. This makes the numerical and graphical results easier to understand and greatly eases interpretation. The actions of the SIMPLE tools are controlled by command files which can be created manually or supplied by PEPP, using the information included in the model. For example, a command file for the tool GANTT can be derived automatically from the PEPP model.

2.4. Integration of Modelling and Monitoring Tools

The tools PEPP, ZM4 and SIMPLE have been successfully applied to real-world analysis problems, both at the University of Erlangen-Nürnberg and at numerous partner institutions. Originally they had been developed independently of each other but with the intention of integrating them later on. Therefore powerful interfaces had been provided.

The integration of modelling and monitoring, i.e. further development, extension and integration of the modelling tool PEPP and the monitoring tools ZM4 and SIMPLE, is a new approach. It is based on the abstraction of a parallel program's dynamic behaviour to a set of events, which is used in both monitoring and modelling. An integrated procedure has many advantages over ad hoc approaches to monitoring. In particular, the event specification problem is solved in a systematic way. The flexibility of observation is also greatly enhanced by the integration.

It is easy to change the level of abstraction of the program under study by modifying the model. Different levels of granularity can be achieved through modelling and automatic re-instrumentation. Event-trace analysis at a new level of granularity is inexpensive, because it is for the most part automated.

3. The Application

The concept of multiple views will now be demonstrated by an example series of measurements. We monitored the behaviour of a parallel ray tracing program on SUPRENUM [Tro88, SH92], a MIMD system with distributed memory in which nodes communicate via message passing.

The parallel ray tracing application consists of one master process and an arbitrary number of servant processes. The master is responsible for high-level control of the algorithm and for the writing of the image file, while the servants are performing the geometrical intersection computations of the ray tracing algorithm. The activities *Distribute Jobs*, *Send Jobs*, *Wait for Results*, *Receive Results* and *Write Pixels* are carried out by the master in a cyclic fashion (see Figure 2), but the activity *Write Pixels* is not part of every cycle. The servants' ray tracing activity is denoted by *Work*. Load balancing in this parallel application is achieved through a decentralized dynamic load distribution scheme and a credit mechanism which ensures that each servant always has some work in its input buffer.

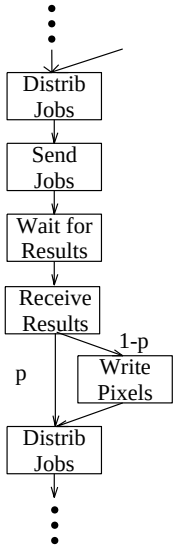

Figure 2: The Activities of the Master Process

The master communicates with each of the servants for distributing work and receiving results, but there is no inter-servant communication in this application. For the internode communication between the master and the servants, SUPRENUM's mailbox mechanism was used. This decision was made in order to achieve asynchronous communication, thereby decoupling the activities of the sending and receiving node. However, local measurements of the servant processes in an *n*-processor scenario revealed that node idle times were extremely high, which led us to analyze the communication behaviour in more detail.

Figure 3 shows part of a stochastic graph model of the parallel program for a 2-processor scenario. The program was instrumented at a level of detail corresponding to the model. Measurement disclosed the communication behaviour of the two nodes (master and servant). This is illustrated in the Gantt diagram shown in Figure 4 (top). A Gantt diagram visualizes the activities on different nodes of the system under study over a common time axis. Providing globally valid time stamps by the monitor system used for the measurement of multiple nodes of a parallel or distributed system is a prerequisite for this kind of depiction.

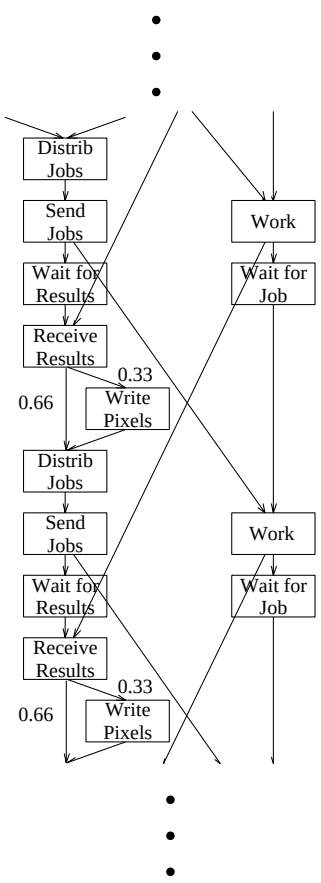

Figure 3: Stochastic Graph Model (Cut-out)

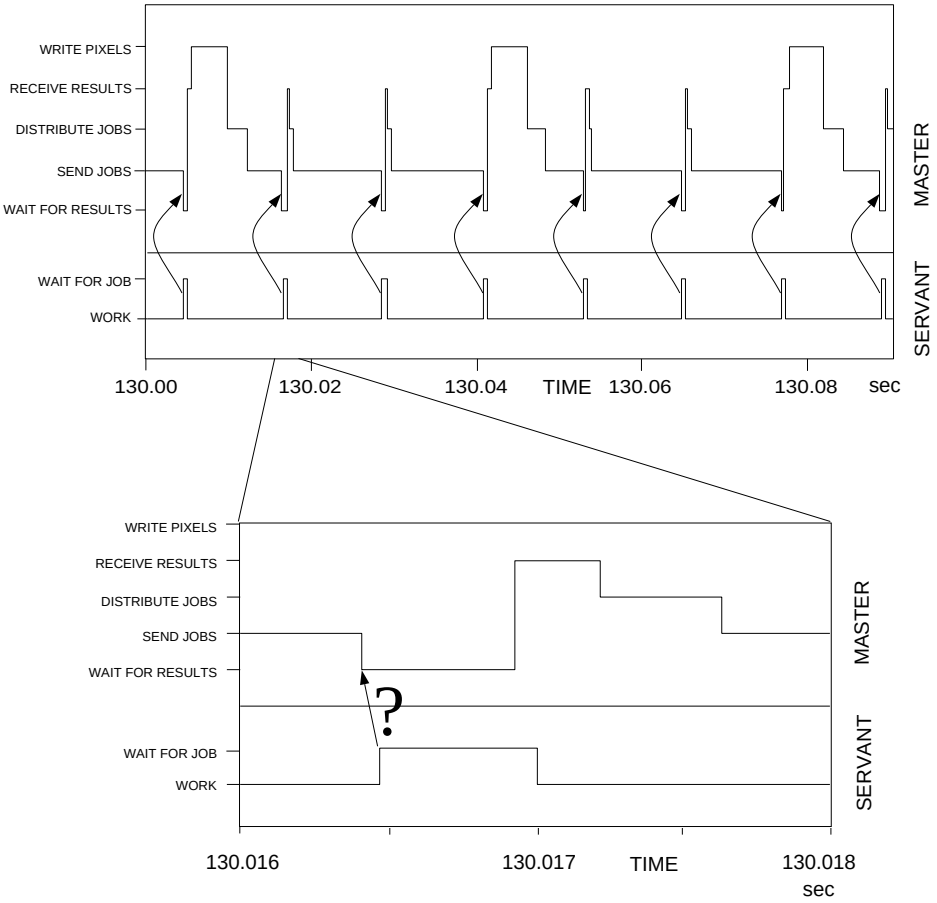

Figure 4: SUPRENUM Mailbox Communication

The diagram in Figure 4 suggests that SUPRENUM's mailbox mechanism behaves like synchronous communication. The sender of a message (activity *Send Jobs* on the master node) is blocked until the receiver initiates communication (activity *Wait for Job* on the servant node). However, a closer look at the communication revealed the insufficiency of the instrumentation (Figure 4, bottom). The master becomes unblocked before the transition from *Work* to *Wait for Job* on the servant node (see "?" in the figure). The reason for this observation must be an uninstrumented activity at the end of the servant's *Work* activity which causes the unblocking of the master. Indeed, the servant sends the results to the master at the end of each *Work* activity. This led to a refinement of the model, splitting up the servant's activity *Work* into the two activities *Work'* and *Send Results*, see Figure 5. Figure 6 shows the observed behaviour of SUPRENUM's mailbox mechanism after re-instrumentation at this finer level of granularity. Now the unblocking of the master's *Send Jobs* by a communication activity on the receiving servant node becomes clear. This example illustrates the benefits of observing parallel programs at a flexible level of detail. Using our monitoring methodology it is easily possible to change the granularity of the observation. Re-instrumentation permits a quick change of the level of abstractions and helps to focus the user's attentions to the most important details.

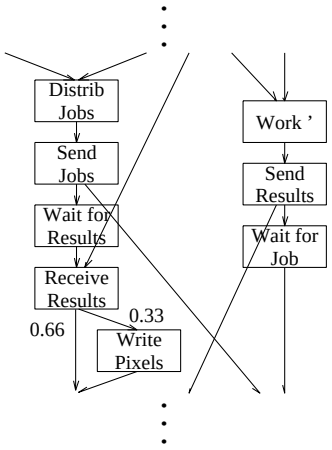

Figure 5: Refined Model

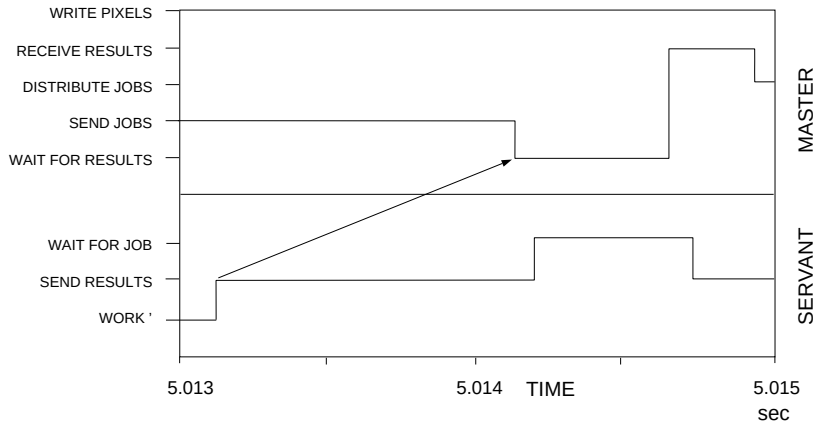

Figure 6: SUPRENUM Mailbox Communication, Detail after Re-instrumentation

4. Conclusion

We have described the method of event-driven hybrid monitoring and its integration with modelling. A set of tools for modelling, monitoring and event trace analysis has been described in Section 2. The case study in Section 3 has demonstrated how these tools can be used to analyze the dynamic behaviour of parallel programs. The method of model-driven monitoring and the tool support enable users of parallel environments to analyze functional and performance aspects of their parallel programs very efficiently. Questions of arbitrary level of detail can be answered easily, by instrumenting all portions of the code at their desired level of granularity. Since instrumentation, measurement and analysis are carried out with the help of sophisticated tools, the technique of model-driven monitoring provides users quickly with the needed information.

References

[BLT90] T. Bemmerl, R. Lindhof, and T. Treml. The Distributed Monitor System of TOPSYS. In H. Burkhart, editor, *CONPAR 90–VAPP IV, Joint International Conference on Vector and Parallel Processing. Proceedings*, pages 756–764, Zürich, Switzerland, September 1990. Springer, Berlin, LNCS 457.

[DHK+92] P. Dauphin, F. Hartleb, M. Kienow, V. Mertsiotakis, and A. Quick. PEPP: Performance Evaluation of Parallel Programs — User's Guide – Version 3.1. Technical Report 5/92, Universität Erlangen–Nürnberg, IMMD VII, April 1992.

[DKQ92] P. Dauphin, M. Kienow, and A. Quick. Model-driven Validation of Parallel Programs Based on Event Traces. In *Proceedings of the "Working Conference on Programming Environments for Parallel Computing", Edinburgh 6–8 April*, 1992.

[FNAG92] J.K. Flanagan, B.E. Nelson, J.K. Archibald, and K. Grimsrud. BACH: BYU Address Collection Hardware, The Collection of Complete Traces. In R. Pooley and J. Hillston, editors, *Sixth International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 51–65, Edinburgh, September 1992.

[GHPW90] G.A. Geist, M.T. Heath, B.W. Peyton, and P.H. Worley. PICL: A Portable Instrumented Communication Library. Technical Report ORNL/TM-11130, Oak Ridge National Laboratory, Tennessee, July 1990.

[HM92] F. Hartleb and V. Mertsiotakis. Bounds for the Mean Runtime of Parallel Programs. In R. Pooley and J. Hillston, editors, *Sixth International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 197–210, Edinburgh, 1992.

[Hof93] R. Hofmann. *Gesicherte Zeitbezüge für die Leistungsanalyse in parallelen und verteilten Systemen*. Dissertation, Universität Erlangen-Nürnberg, 1993.

[Kla92] R. Klar. Event-Driven Monitoring of Parallel Systems. In G. Kotsis and G. Haring, editors, *Proceedings of Workshop on Monitoring and Visualization of Parallel Systems*, Moravany, Oct. 1992. Austrian Center for Parallel Computation and Slovak Academy of Sciences, to be published by Elsevier.

[Mal89] A.D. Malony. Multiprocessor Instrumentation: Approaches for CEDAR. In M. Simmons, R. Koskela, and I. Bucher, editors, *Instrumentation for Future Parallel Computing Systems*, chapter 1, pages 1–33. ACM Press, Frontier Series, Addison–Wesley Publishing Company, New York, 1989.

[MCNR90] A. Mink, R. Carpenter, G. Nacht, and J. Roberts. Multiprocessor Performance–Measurement Instrumentation. *Computer*, 23(9):63–75, September 1990.

[Moh92a] B. Mohr. SIMPLE — User's Guide Version 5.3.
Part A: TDL Reference Guide
Part B: POET Reference Manual
Part C: Tools Reference Manual
Part D: FDL / VARUS Reference Guide. Technical Report 3/92, Universität Erlangen–Nürnberg, IMMD VII, März 1992.

[Moh92b] B. Mohr. Standardization of Event Traces Considered Harmful or Is an Implementation of Object-Independent Event Trace Monitoring and Analysis Systems Possible? In *Proc. of the Workshop Environments and Tools For Parallel Scientific Computing*, Saint Hilaire du Touvet, September 1992. CNRS-NSF. to be published in the "Parallel computing series" of Elsevier Science Publishers.

[SH92] M. Siegle and R. Hofmann. Monitoring Program Behaviour on SUPRENUM. *Computer Architecture News*, 20(2):332–341, May 1992.

[Söt90] F. Sötz. A Method for Performance Prediction of Parallel Programs. In H. Burkhart, editor, *CONPAR 90–VAPP IV, Joint International Conference on Vector and Parallel Processing. Proceedings*, pages 98–107, Zürich, Switzerland, September 1990. Springer–Verlag, Berlin, LNCS 457.

[ST87] R. Sahner and K. Trivedi. Performance Analysis and Reliability Analysis Using Directed Acyclic Graphs. *IEEE Transactions on Software Engineering*, SE-13(10), October 1987.

[SW90] F. Sötz and G. Werner. Lastmodellierung mit stochastischen Graphen zur Verbesserung paralleler Programme auf Multiprozessoren mit Fallstudie. In *11. ITG/GI–Fachtagung Architektur von Rechensystemen*, pages 231–243. vde–Verlag, Berlin und Offenbach, Januar 1990.

[Tro88] U. Trottenberg. (ed). Special Issue on the 2nd International SUPRENUM Colloquium. *Parallel Computing*, 7, 1988.