Experience with Development of Performance Monitoring Tools for Parallel Applications

Marian Bubak[1,2], Włodzimierz Funika[1], and Roland Wismüller[3]

[1]Institute of Computer Science, AGH, al. Mickiewicza 30, 30-059 Kraków, Poland
[2]Academic Computer Centre – CYFRONET, Nawojki 11, 30-950 Kraków, Poland
[3]LRR-TUM – Technische Universität München, D-80290 München, Germany
phone: (+48 12) 617 39 64, *fax:* (+48 12) 633 80 54
{bubak,funika}@uci.agh.edu.pl,wismuell@in.tum.de

Abstract. *The paper presents an overview of issues connected with the functionality and design of off-line and on-line tools for performance analysis of parallel applications. The research started from a detailed examination of functionality of the existing tools and introduced a concept of a performance monitoring and analysis environment which evolved from an off-line performance evaluation tool to an on-line performance monitoring system based on a comprehensive monitoring infrastructure.*

1. Introduction. When developing a program for modern parallel architectures, it is often crucial to have at disposal tools supporting design, debugging, testing and evaluation phases of program development. Our research related to parallel tools started with a thorough investigation in functionality of the existing tools. The concept of a new environment evolved through three stages.

The first stage was connected with an off-line performance analysis tool, what ment that performance data would be collected with an monitoring facility, according to observation goals formulated prior to a monitoring session itself. The latter is followed by an analysis session, based on evaluation goals with visualization techniques.

The second stage was aimed at transforming the off-line tool into an on-line one, in order to enable formulation of analysis goals just in the course of an application execution. This included an enhancement of the functionality of the monitoring subsystem (MS), based on use of the On-line Monitoring Interface Specification (OMIS), which introduces the *request/reply* concept into a tool system [1].

The third phase comprised a substitution of the previously developed MS by a comprehensive OMIS-compliant Monitoring System (OCM) and incorporating a part of the existing analysis mechanisms into the PATOP and TATOO tool.

2. Off-line performance analysis tool. Primarily, we focused on defining the goals and model of an off-line performance analysis tool, the range and levels of performance data to be analyzed and implementing the tool. The tool operation (shown in Fig. 1) being based on event tracing of the application execution [2] features *source code instrumentation, establishing global time reference, compensation for perturbations, filtering performance data*, and a reasonably large range of *performance visualisations.*

The basis of performance evaluation in the tool is a *measurement.* Generally, formulating a measurement target implies the specification of the object, location and conditions of a measurement. The object may be a routine call, loop, individual statement and inter-process interaction as well as system resources usage. The location specifies the place in the program where the measurement is to be made. The conditions imply that the measurement action is applied to those objects

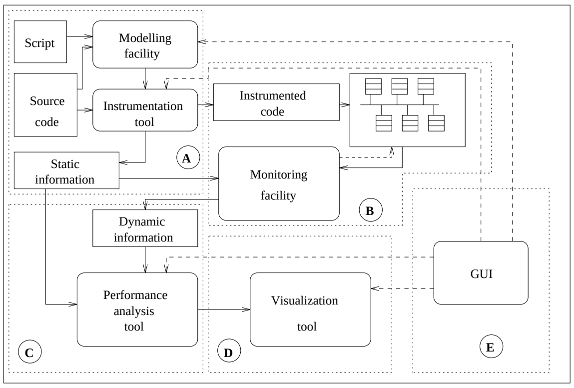

Figure 1: Structure of off-line performance tool

which match the constraints imposed.

The measurement model, which is related to the source code of application, is analysed by a parser and transformed into the instrumentation specification of the program. An instrumentation facility generates the instrumented version of the program code. Event-driven performance tracing is carried out by a monitoring facility. A post-mortem component of the tool enables to compensate the trace data for the sampling intrusion into the execution. Trace data are processed to extract relevant performance information with a special language THL. Performance analysis and visualization phases allow for the computation of statistics on each of the performance data item, the profiling of time usage across nodes, processes and procedures, and dynamic visualization: processor utilization and inter-process relationships. Each of the views can be correlated with the source code of the program to enable focusing on those parts of the code which require a detailed investigation.

3. On-line performance monitoring tool. The ability to analyze performance in off-line mode is sometimes insufficient to understand the pitfalls encountered by the application being investigated. Whereas performance evaluation in off-line mode starts from cumulative performance views and goes to more subtle elements of execution when necessary, on-line performance evaluation is mainly based on instant views, thus leading to quite often sampling of the state of application and other values. This implies that the monitoring system which supports on-line monitoring must feature a well-defined mechanism of interaction with other software components, must be prompt, low-intrusive and able to access various computer system levels.

The requirements mentioned gave impetus to emergence of the idea of separation of the pure tool functionality and the monitoring system functionality, which is defined by the OMIS specification [1] with the concepts of the user-formulated *request* and the *reply* generated by the monitoring system. This suggests that the centre of activity in the tool moves into the interactive component of the monitoring system which we call *request controlling system* (RCS), intermediate between the monitoring system and the user steered tool. Such a monitoring system should provide a large number of services common in different systems and to enable extension of its functionality by new services specific to particular environments. According to OMIS there should be defined what objects are to be monitored and what operations could be performed on these objects. The range of objects selectable for monitoring includes: *node*, *process*, *thread*, *function*, *loop* while the set of operations comprises *control* (e.g. suspend a process), *informational* (e.g. get a node usage) , and *metrical* (e.g. compute the minimum message length sent from process A to process B) ones. Thus, a list of services available to tools is defined.

When designing the interaction mechanism of the tool and the monitoring system (MS) the existing mechanism of interaction between an application and the monitoring system was used to build an intermediate layer to enable the operation of the tool with regard to MS in the *client–server* mode, based on OMIS [3]. In line with the above assumptions the structure of the tool was transformed as shown in Fig. 2.

Processing *Omis_request* requires special consideration when implementing RCS. A *request* is a string-typed specification of various OMIS services to be executed, for ex-

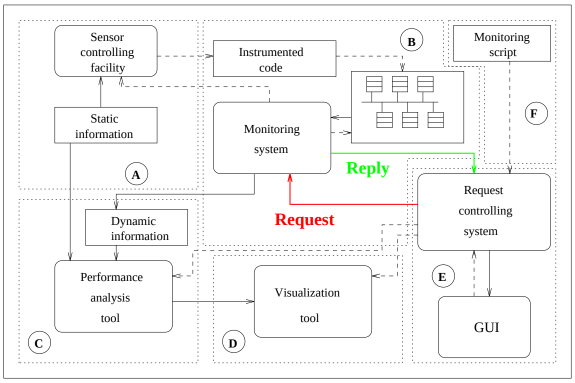

Figure 2: Structure of on-line performance tool

ample `"thread_has_started_lib_call([], "pvm_bcast"): print ($time, $proc)"`. This is a conditional request, executed whenever any application thread calls `pvm_bcast`. When such a request is defined, RCS needs to parse it first. When RCS finds out that the request is conditional, it makes sure that the specified event service is supported and that all its parameters are specified correctly. A similar check applies to all the actions in the action list. Enabling conditional requests involves expanding the thread list and sending a suitable request for reports to the monitoring system, using bidirectional asynchronous RPC. On arrival of the request, MS redistributes it to all the application processes specified. Each relevant process of the application receives the request and initiates actions to report calls to `pvm_bcast`, if any.

The current implementation of the tool and its underlying layers – the monitoring system and the request control system – allows for obtaining such performance views as current state of application process and node, inter-process communication relations, computer resources usage vs. process, or event trace of application execution.

4. OCM based performance tools. In order to fully use the functionality offered by the monitoring system interface and to allow for concurrent running of multiple tools operating on the same instance of parallel application, we decided to port and enhance the functionality of two existing performance measurement tools, `PATOP` and `TATOO`, using the `OCM` (OMIS Compliant Monitoring system) for low-level monitoring [4]. *PATOP* and *TATOO*, tools intended for on-line and off-line performance analysis respectively, have originally been developed at LRR-TUM for the analysis of applications running under the PARIX environment on Parsytec systems [5]. They support measurement and visualisation at various levels of detail: the whole system, individual nodes, particular processes or user functions. The *OCM* is a reference implementation of OMIS, which provides support for monitoring message passing applications. The OCM is a distributed monitoring environment, with a central Node Distribution Unit and local monitoring processes (one per node), communicating with the monitored processes via shared memory.

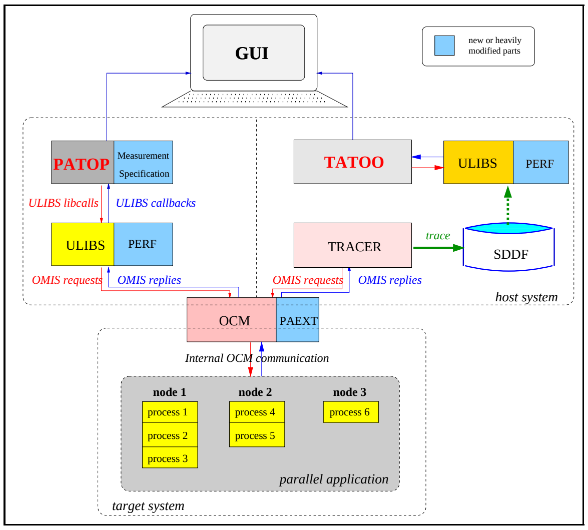

Figure 3: Operation of PATOP and TATOO on top of the OCM

Porting PATOP to the OCM involves an additional layer, ULIBS to ease interfacing PATOP to the OCM [4]. The task of ULIBS is to transform high-level measurement specifications to event/action relations accepted by the OCM and to transform back the results. In Fig. 3 the structure of the on-line monitoring environment is presented with monitoring a sample parallel application. The structure of the TATOO tool is shown on the right side of Fig. 3. Performance data are stored in local trace buffers which are read by the OCM when the application terminates, or a trace buffer is almost full. The TATOO tool reads the trace file saved by TRACER, via a special, off-line version of ULIBS. Adapting PATOP and TATOO to PVM and MPI required to define a number of new services for performance analysis and a modification of the ULIBS functionality.

For the new environment three ways of gathering performance data are designed: *direct communication* — immediate reporting on an event; *tracing* — a delayed reporting on events stored in a buffer; using *counters* and *integrators* — the most important information on each event is summarized in counters (for `integer` values) and integrators (for `floating` values) and read by the OCM on request. For each measurement, we use the *least intrusive* approach that still meets our goals.

ULIBS, a library of high-level performance measurement and debugging functions was primarily developed for the PARIX parallel environment, and has been adapted to provide *asynchronous* requests and callbacks, *starting/stopping* processes, *debugger-oriented* mechanisms.

5. Concluding remarks. A three-stage approach to development of performance tools has been presented. The first stage involves the design and implementation of an off-line tool on top of a monitoring facility, which features establishment of globally consistent time-stamps across workstations and a large number of dynamic and static views. The second stage consisting in the design of the online performance monitoring tool using OMIS enabled to make use of the monitoring facility used previously for the needs of an off-line tool for building an on-line monitoring tool on top of an independently operating monitoring system.

The third stage comprises the substitution of the centralised monitoring system by a comprehensive distributed monitoring infrastructure OCM that enables to provide a larger functionality, portability and interoperability. The performance analysis functionality designed for the first two stages has been transferred into the PATOP performance analyser. This adds to the functions and services available to the user a class of *events services*. The transfer of the monitoring system functionality to the OCM enables to gain a wide portability of tools and to proceed with investigation in the interoperability of OMIS-oriented tools.

Acknowledgements. This work has been carried out within the Polish-German collaboration and supported, in part, by KBN under grant 8 T11C 006 15.

References

[1] Ludwig, T., Wismüller, R., Sunderam, V., and Bode, A., *OMIS – On-line Monitoring Interface Specification (Version 2.0)*, Shaker Verlag, Aachen, Vol. 9, LRR-TUM Research Report Series, (1997)

[2] Bubak M., Funika, W., Mościński J.: Evaluation of Parallel Application's Behavior in Message Passing Environment, in: *Marian Bubak, Jack Dongarra, Jerzy Wasniewski (eds.) Recent Advances in Parallel Virtual Machine and Message Passing Interface. Proceedings of 4th European PVM/MPI Users' Group Meeting, Cracow, Poland, 3-5 November 1997*, Lecture Notes in Computer Science **1332**, Springer, (1997), 234-241

[3] Bubak, M., Funika, W., Iskra, K., Maruszewski, R., and Mościński, J.: Off-line and On-line Performance Monitoring Tools. In: *HiPer'98 Conference – High Performance Computing on Hewlett-Packard Systems*, ETH Zurich, Switzerland, October 14 - 16, (1998), 200-206

[4] Bubak, M., Funika, W., Iskra, K., and Maruszewski, R.: OCM-Based Tools for Performance Monitoring of Message Passing Applications. In: Sloot, P., Bubak, M., Hoekstra, A., Hertzberger, B., (eds.): *Proc. Int. Conf. High Performance Computing and Networking*, Amsterdam, April 12-14, 1999, Lecture Notes in Computer Science **1593**, Springer, (1999), 1270-1273

[5] Wismüller, R., Oberhuber, M., Krammer, J. and Hansen, O.: Interactive Debugging and Performance Analysis of Massively Parallel Applications, *Parallel Computing*, **22**, (1996), 415-442