Towards Extension of an OCM-based Tool Environment

Marian Bubak[1,2], Włodzimierz Funika[1], Bartosz Baliś[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*
{bubak,funika}@uci.agh.edu.pl, balis@icsr.agh.edu.pl, wismuell@in.tum.de
phone: (+48 12) 617 39 64, fax: (+48 12) 633 80 54

Abstract

There is a high demand for tools supporting parallel programming. These comprise several classes of tools, e.g. debuggers, performance analyzers, visualizers, etc., each class providing a well defined set of services. A well designed tool environment should support monitoring of parallel applications developed using different programming models and offer a wide range of tools capable of cooperating with each other. This paper presents the recent development of the OCM-based environment of on-line tools and outlines perspectives for further research. **Keywords**: parallel programming, monitoring, on-line tools, interoperability, MPI, OMIS.

1 Introduction

Like in the case of sequential programs, tools for parallel programming support are indispensable. On-line tools need a specialized module for observing and possibly manipulating a parallel program's state, which is called a *monitoring system*. Usually, tools are provided with built-in monitoring modules.

Each type of tools for parallel programming support offers a well-defined functionality, therefore, in order to achieve a complex set of services in a tool environment, the *interoperability* of the tools is highly desirable. Interoperability means the capability to run several tools concurrently and to apply them to the same application [12]. Ideally, we would like to enable interoperability between tools coming from different vendors. However, such tools are likely to be incompatible with each other and it might not even be possible to run them concurrently due to low-level conflicts. Moreover, once tools are able to run concurrently, further problems may occur. Different tools may concurrently try to modify the state of a shared object (mutual exclusion problem), a tool may change the state of an object, while another tool may still keep the outdated information about it (consistency problems), etc.

A way to overcome these problems is to create a common monitoring infrastructure which provides the tools with a standardized interface to access the underlying resources.

The goal of the OMIS (*Online Monitoring Interface Specification*) [9] project is to define a common interface between the tools and the monitoring system.

OMIS is designed to support a wide variety of on-line tools and to enable interoperability of tools.

The first monitoring system to implement the OMIS specification is the OCM (OMIS-Compliant Monitor) [16]. Originally, the OCM was intended primarily for clusters of workstations and supported PVM [7] applications. One of the primary platforms for the OCM have been clusters of SGI workstations.

Starting with the OCM, the aim of the work presented in this paper is to provide a uniform monitoring and tool environment for MPI-based message passing programs on SGI parallel systems, like the Origin 2000. The focus is on interoperating debugging and on-line performance analysis tools with explicit support for the MPI programming model. Thus, the tool environment under discussion complements existing development tools for shared memory parallel programs, like SGI's ProDev Workshop [11].

In the following sections, we will first present an overview on the structure of an OCM-based tool environment, outline the evolution of this environment, and present the performance analyzer and the interoperability issues in more detail.

2 OCM-based tool environment

Fig. 1 provides an insight into the layer structure of an OCM-based tool environment. The OCM makes up the layer between the application and tools.

The tools communicate with the OCM via a high-level routine library, ULIBS, which consists of a number of modules, each one providing specific services. ULIBS offers a standardized API for initialization of tools, obtaining information on the parallel target system, running the application, which includes debugging services, facilities for performance analysis tools, which include reading performance data from the monitoring system and translating high level measurement requests to lower level monitoring requests and accordingly transforming the resulting replies, etc. The modules of ULIBS form a set of libraries which are linked to the tools depending on what kind of services they need. Originally, ULIBS was developed as part of the TOPSYS project [1].

To provide effective tracing of communication calls, which is a crucial feature for the performance analysis, the application is linked against an *instrumented* communication library, i.e. a modified version of the standard communication library used by the application that contains parts of the monitoring system [3].

Both ULIBS and a number of tools developed in the framework of the TOPSYS project have been ported to the OCM and become part of The Tool-Set [14]. Among these are VISTOP (program flow visualization), PATOP (performance analysis) [6], DETOP (debugging) [15], CoCheck (checkpointing), LoMan (load balancing) and others. From this set of tools, DETOP and CoCheck are already available for SGI platforms. In this paper, we focus on the debugger DETOP, the performance analyzer PATOP, and their joint use.

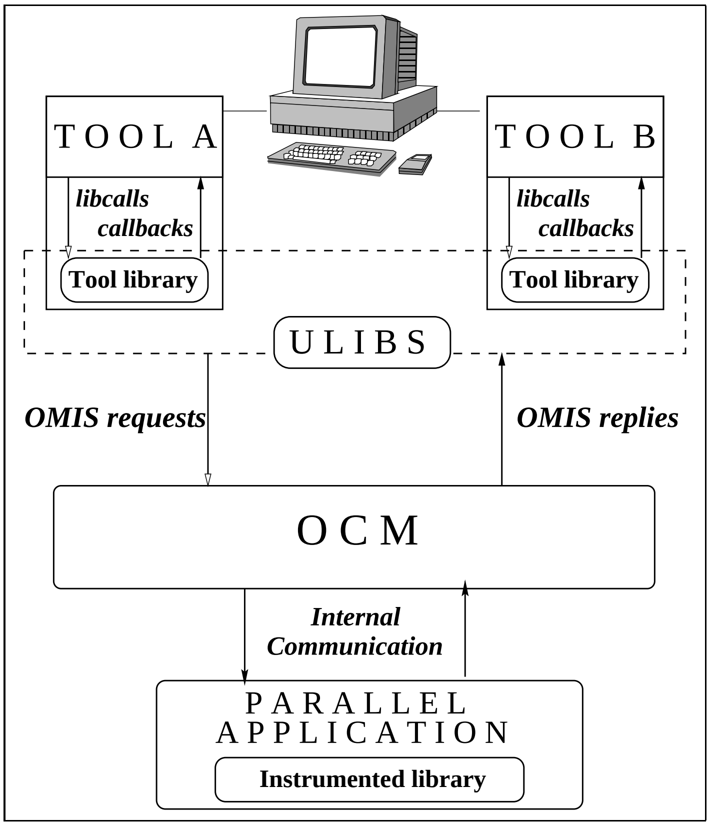

Fig. 1: Structure of the OCM-based tool environment.

3 Environment evolution

The OCM-based tool environment is continuously being enhanced and extended.

- The **OCM** was extended with support for performance analysis. New types of objects - *integrators* and *counters* - meant for effective storing of performance data were introduced. A new extension to the OCM has been developed to support PATOP's advanced capabilities [6].
  The OCM was also enhanced to support monitoring of MPI applications [5]. Unlike PVM, MPI is provided in many, vendor-specific, implementations. Moreover, the MPI standard provides no means to obtain information on target parallel system configuration or detailed information on an application's processes. Therefore, the OCM's port to MPI depends on the specific MPI implementation. Currently, the OCM supports `mpich`, a free MPI implementation [8].
- **PATOP** is an on-line performance analysis tool which provides a wide

range of performance measurements and visualization [6]. The measurements comprise the utilization of computational resources, time spent in a send/receive call and the volume of messages transferred between processes. PATOP works on the system, node, process and even source code level. Originally, PATOP offered such performance displays as bargraphs, multicurves, color bars and distrigrams. When porting PATOP to the OCM, PATOP was extended with new displays - scatter plots and piecharts [6].
- **DETOP** supports the debugging of parallel programs [15]. It offers a common set of debugging capabilities, like controlling the application execution, setting breakpoints and inspection of a program's data space. DETOP has been ported to the OCM and originally supported debugging of PVM applications. The port of the OCM to MPI has made possible the debugging of MPI applications, too. Currently, DETOP underlies an advanced symbolic debugger for HPF programs, SPiDER [2].

Succeeding enhancements concerned the performance analysis of MPI applications and interoperability of tools. These topics are described in the following sections.

4 Performance analysis of MPI applications

MPI is a widely accepted standard of message-passing. It provides a much richer set of communication subroutines than PVM. To enable performance analysis of MPI applications with PATOP, a number of enhancements were necessary [3]. Their very first stage dealt with the instrumentation of the MPI library and the definition of PATOP's measurements in ULIBS, which are expressed in terms of the OCM requests and rely on the semantics of the particular library communication calls.

After this had been accomplished, it was found that the current measurements in PATOP does not allow to monitor MPI collective communication. Whereas the current communication measurements always apply to a particular communication participant (e.g. *measure the delay of send calls* or *measure volume of received messages*), in case of MPI collective calls like `MPI_Bcast` it is either impossible or not necessary to determine whether the particular call is a send or a receive one. Therefore, two new measurements were added to PATOP, i.e. *measure the delay of a collective call* and *measure the volume of messages transferred within a collective call*. Moreover, since a collective call always applies to a collection of processes, a so called *communicator*, to provide convenience in monitoring MPI collective calls, communicators were introduced to PATOP as new objects to which a measurements can be applied. Fig. 2 shows the PATOP's *New Measurement* dialog window enhanced with new items.

In the *Type of Measurement* window, two new items were added: *Delayed in Global* and *Global Volume*. In the window, where the scope of a measurement is to be chosen, beside the *Node Level* and *Process Level* lists, a new *Group Level* list containing the identifiers of existing processes collections (communicators)

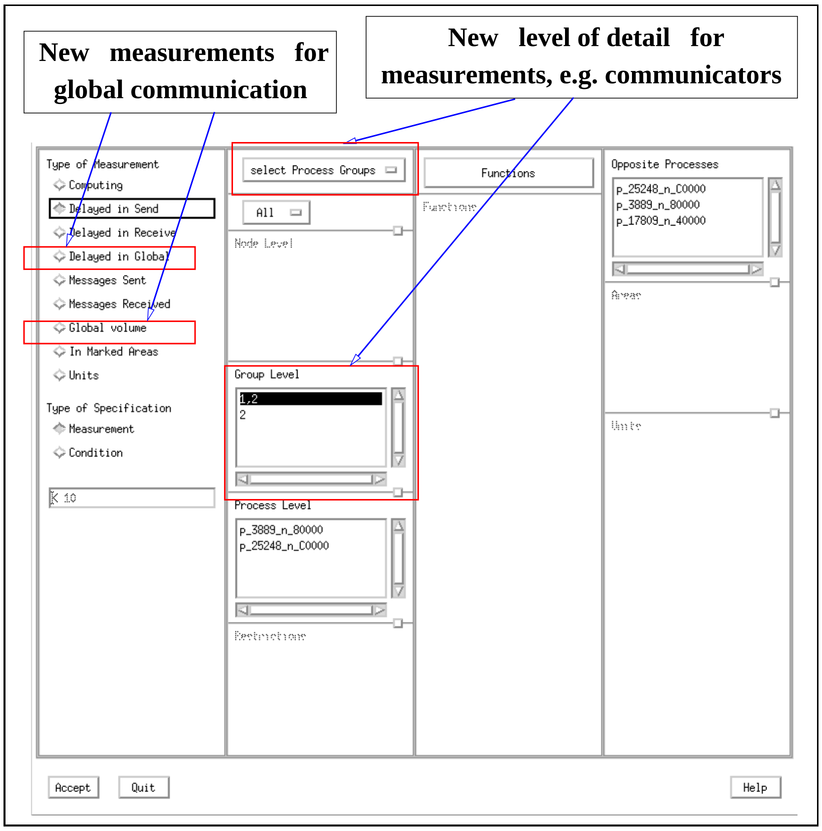

Fig. 2: Changes in the *New Measurement* dialog in PATOP

was added.

5 Interoperability in the OCM

Problems with the interoperability of tools appear at various levels [12]. First of all, the tools must be able to run concurrently. This is assured by the OCM's structure [13]. However, further problems may occur at higher levels and cause an incorrect behaviour. These problems have to be addressed. In this section the interoperability of DETOP and PATOP will be examined [4].

The cooperation between DETOP and PATOP is desirable, since PATOP provides high-level performance visualization of an application, whereas DETOP provides a source code level insight into the application's behavior. Thus, for example, whenever PATOP reveals an incorrect behaviour of an application observed via its displays, DETOP can be used to examine the source code position, values of proper variables, etc. In addition the concurrent use of the tools allows to start/stop performance measurements at clearly specified points in the application's execution. DETOP can be used to set breakpoints, where then measurements can be defined or deleted. Thus, this would enable an exact measurement of single loops. The same technique also allows to define measurements on dynamically created entities like communicators or – in the case of MPI-2 – processes.

However, the cooperation of these tools revealed some *consistency problems*. The incorrect behaviour occurred in two cases. First, when the application is started with PATOP, it starts reading the performance data from the OCM and updates performance displays to visualize the execution. However, when the application is suspended by DETOP, PATOP proceeds reading data and updating displays, while the expected behaviour is that PATOP stops monitoring while the application is being suspended. This is only possible if PATOP is notified of a thread suspension so that it can react properly. Second, once the application is started or resumed with DETOP, PATOP does not start monitoring, as it is not aware that a process has been continued. Since to resolve these problems a tool must be notified of actions performed by another tool, we sometimes say that *direct interactions* between the tools are needed.

These problems can be addressed by the mechanisms of **events** as defined by the OMIS specification. A tool can issue a *conditional request* to the monitoring system to trigger a specified list of actions whenever an event specified within a request occurs. Thus, the mentioned problems of DETOP and PATOP interoperability can be addressed if PATOP issues two following conditional requests:

```
thread_has_been_stopped([]):  print($proc)
thread_has_been_continued([]):  print($proc)
```

The semantics of these requests is as follows: *whenever a process to which PATOP is attached*[1] *has been stopped (continued), the process identifier of the stopped (continued) process is returned*. The events are handled by means of a callback mechanism. The process' identifier is actually passed to the appropriate callback function, which is invoked on each occurrence of the event. This callback function performs actions to stop (or resume) the measurements.

To make possible the interoperability of tools and to address the consistency problems, the tool environment has been extended with new elements:

- A new top-level tool OCTET, developed to manage the start-up of the tool environment (i.e. start the application and the tools).
- A new module, added to ULIBS for interoperability purposes.

The OCTET tool is a console application which provides a simple interface to set up the environment (which tools are to run, what type of application is to be executed, how many processes to spawn in case of MPI applications, etc.). Having been provided with the information, OCTET starts the monitoring system, tools, and application automatically.

The interoperability module in ULIBS forms a set of libraries specific for a particular tool. The libraries contain the appropriate code to handle specific

[1] Tools have to explicitly attach to the nodes and processes in the target parallel system on which they are intended to perform some actions. Not attached objects are invisible. The '[]' parameter means *all processes attached*.

scenarios of interoperability, e.g. in case of PATOP, this code issues the two requests mentioned above.

Since some actions are to be performed only if a particular scenario takes place, i.e. if particular tools are running concurrently, the tools must be in some way informed about the other tools running in the environment. The only component that possesses this information is the OCTET tool, which passes it to all tools during the startup.

6 SMP implications to the tool environment

The ever growing interest in shared memory programming is going to underly a succeeding phase in the extension of the tool environment. This will be reflected in the environment elements we describe in the down-top manner.

As the principal entity in SMP is *thread*, the OCM extension is to be focussed on monitoring *loops* and *threads synchronisation* within a program. A special attention in this context is paid to the elaboration of a well-defined naming policy in the OCM, related to loops and barriers in the SMP programming model. In order to be able to track the occurences of these constructs, we need to use information extracted from the symbol table generated by a compiler. This information is intended to be handled within ULIBS and the OCM. The information on the memory an application placed in is also extracted from the symbol table. One of the interesting aspects of multi-threading monitoring concerns tracking the variable collection of threads, created or deleted as the application's execution proceeds. Another important monitoring aspects to be investigated are *nested parallelism* and *cache performance*.

The tool part of the environment is also subject to enhancement. This concerns the user interface of tools, which is going to enable the user to operate on threads. Another changes will deal with the specific aspects of SMP like loops and synchronisation in terms of barriers. Adaptation to multi-threading is needed as well in the internal mechanisms of the tools. To enable effective operation on the source code of application and its calls tree , it is necessary to extend the environment with a new tool, we call *static visualization tool*, intended to support performance analysis provided by PATOP. The work on changing and enhancing the user interface of tools will be accompanied by the work on extending the interoperability of the existing components of the environment to enable the multi-threading related co-operation of the tools. The scalability aspects of the environment operation will also be an object of the on-going research as for a large number of threads.

7 Conclusion

The OCM-based tool environment is successively developing. Currently, it is best suited to work on clusters of workstations and supports only message passing programming models. However, the OCM has already been tested on an SGI Origin 2000 running IRIX 6.4 and performed well. Further research will be focused on adapting the OCM to other programming models and architectures. Since the monitoring interface OMIS already allows for threads, we plan to also extend the OCM implementation with a support for multi-threaded applications. In the long term, this will allow to build tools for applications using mixed parallelism, like MPI and OpenMP, which run on clusters of shared memory parallel computers.

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. Bemmerl, T., Bode, A., Braum, P., Hansen, O., Treml, T., and Wismüller, R.: The Design and Implementation of TOPSYS. Technical Report TUM-INFO-07-41-440, Technische Universität München, July 1991.
2. Brezany, P., Grabner, S., Sowa, K., Wismüller, R.: DeHiFo – An Advanced HPF Debugging System. In: 7th Euromicro Workshop on Parallel and Distributed Processing, PDP'99, Madeira, Portugal, February 1999, 226-232, IEEE CS Press.
3. Bubak, M., Funika, W., Baliś, B., and Wismüller, R.: Performance Measurement Support for MPI Applications with PATOP. Workshop on Applied Parallel Computing, Bergen, Norway, June 18 - 21, 2000. Lecture Notes in Computer Science, Springer Verlag, 2000. To appear.
4. Bubak, M., Funika, W., Baliś, B., and Wismüller, R.: Interoperability of OCM-based on-line Tools. EuroPVM/MPI'2000 Conference Balatonfüred, Lake Balaton Hungary, September 10th-13th, 2000. In: Dongarra, J., Kacsuk, P., Podhorszki, N. (eds.): Recent Advances in PVM and MPI. Proceedings of the 7th European PVM/MPI User's Group Meeting, Springer LNCS. To appear.
5. Bubak, M., Funika, W., Gembarowski, R., and Wismüller, R.: OMIS–Compliant Monitoring System for MPI Applications. In: R. Wyrzykowski, B. Mochnacki, H. Piech, J. Szopa (eds.): PPAM'99 - The 3th International Conference on Parallel Processing and Applied Mathematics, Kazimierz Dolny, Poland, 14 - 17 September 1999, pp. 378-386, IMiI Czestochowa (1999).
6. Bubak, M., Funika, W., Iskra, K., Maruszewski, R., and Wismüller, R.: Enhancing the Functionality of Performance Measurement Tools for Message Passing Applications. In: Dongarra, J., Luque, E., Margalef, T., (eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface. Proceedings of 6th European PVM/MPI Users' Group Meeting, Barcelona, Spain, September 1999, Lecture Notes in Computer Science 1697, Springer-Verlag Berlin-Heidelberg, pp. 67-74 (1999)
7. Geist, A., et al.: PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing. MIT Press, Cambridge, Massachusetts (1994)
8. Gropp, W., Lusk, E.: User's Guide for `mpich`, a Portable Implementation of MPI. ANL/MCS-TM-ANL-96/6, 1996.
9. 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)
`http://wwwbode.in.tum.de/~omis/OMIS/Version-2.0/version-2.0.ps.gz`

10. MPI: A Message Passing Interface Standard. In: Int. Journal of Supercomputer Applications, **8** (1994); Message Passing Interface Forum: MPI-2: Extensions to the Message Passing Interface, July 12, (1997)
`http://www.mpi-forum.org/docs/`
11. Silicon Graphics, Inc.: *Developer Central: ProDev Workshop.* WWW page.
`http://www-europe.sgi.com/developers/devtools/tools/prodev.html`
12. Trinitis, J., Sunderam, V., Ludwig, T., and Wismüller, R.: Interoperability Support in Distributed On-line Monitoring Systems. In: M. Bubak, H. Afsarmanesh, R. Williams, and B. Hertzberger (eds.): High Performance Computing and Networking, 8th International Conference, HPCN Europe 2000, volume 1823 of Lecture Notes in Computer Science, Amsterdam, The Netherlands, May 2000. Springer.
13. R. Wismüller. Interoperability Support in the Distributed Monitoring System OCM. In: R. Wyrzykowski et al. (eds.): Proc. 3rd International Conference on Parallel Processing and Applied Mathematics - PPAM'99, pages 77-91, Kazimierz Dolny, Poland, September 1999, Technical University of Czestochowa, Poland. Invited Paper.
14. Wismüller, R., and Ludwig, T.: The TOOLSET — An Integrated Tool Environment for PVM. In: Liddell, H., Colbrook, A., Hertzberger, B., Sloot, P., (eds.): Proceedings of the Int. Conference on High Performance Computing and Networking, Brussels, Belgium, April 1996. Lecture Notes in Computer Science **1067**, Springer, 1029 – 1030, 1996.
`http://wwwbode.informatik.tu-muenchen.de/Par/tools/`
15. Wismüller, R., Oberhuber, M., Krammer, J. and Hansen, O.: Interactive Debugging and Performance Analysis of Massively Parallel Applications. *Parallel Computing*, **22**, (1996), 415-442
16. Wismüller, R., Trinitis, J., and Ludwig T.: OCM - A Monitoring System for Interoperable Tools. In: Proceedings of the 2nd SIGMETRICS Symposium on Parallel and Distributed Tools SPDT'98, Welches, OR, USA, August 1998.