Application-level semi-on-line monitoring of VisualMP applications on clusters of workstations

Norbert Podhorszki

A new application-level, software tracing monitor is designed and implemented for the VisualMP graphical parallel programming environment to support semi-on-line monitoring of message-passing programs in heterogeneous environments. We present the design aspects and implementation issues of the monitor.

VisualMP is a graphical programming environment integrating several tools to support the life-cycle of building parallel applications. Its major goal is to provide an easy-to-use, integrated set of programming tools for development of general message-passing applications to be run in heterogeneous computing environments. Its main benefits are the visual interface to define all parallel activities in the application, the syntax independent graphical definition of message passing instructions, full support of compilation and execution on heterogeneous environment and the integrated use of the debugger and performance visualiser. Tools of the VisualMP program development environment are the GRAPNEL graphical parallel programming language, GRED graphical editor to write parallel applications in GRAPNEL, the GRP2C pre-compiler to produce the C code with PVM or MPI function calls from the graphical program, the DIWIDE distributed debugger, the PROVE execution and performance visualisation tool and the GRM distributed monitor. For detailed overview of the tools of VisualMP, see [1] and [2]. PROVE is presented in [4].

We believe that post-mortem analysis of long traces is not a practical way to evaluate and improve an application. When several millions of events are visualised at once no one can effectively use it to focus on performance problems. Moreover, a post-mortem visualiser is not capable of visualising very large trace data. Practical ways of performance evaluation of long-running programs can be on-line visualisation, presentation of statistics for the whole execution and the use of automatic performance analysis tools. Having already a trace visualiser we decided to explore the possibilities of the first two methods, keeping in mind the possible use of automatic tools in the future.

In off-line monitoring, trace events are stored in local or global storages (memory and files) and are processed after execution. In on-line monitoring trace events are sent immediately to a tool that processes (visualises or evaluates) them. Instead of sending each individual trace event to the tool, events can be buffered in local storage and collected into a global trace only when they are needed for processing. We call this method semi-on-line monitoring (and visualisation). One end of this method is off-line monitoring (i.e. buffering is used and traces are collected only when the application is finished). The other end of this method is on-line monitoring (no buffer is used).

In order to support the examination of long running or cyclic applications PROVE is modified for semi-on-line visualisation. For this purpose a new distributed monitor - GRM - was designed and developed that supports trace collection and provides a consistent global time reference at any time during execution. During execution the user can force trace collection at any time and events locally buffered until that time are collected by the monitor and the trace is presented in PROVE. Collection can be started either by the user, the application itself (with a special instrumentation function call), regularly by the visualisation tool or the debugger tool.

The monitoring is event-driven, both trace collection and counting are supported. The collection is fully software tracing and the instrumentation method is direct source code instrumentation. For a classification of monitoring techniques see [3]. The direct source code instrumentation is the easiest way of instrumentation. Since VisualMP keeps in hand the whole cycle of application building and source instrumentation is supported graphically we chose this option. The GRP2C precompiler inserts instrumentation function calls into the source code and the application process generates the event trace. Both trace collection and statistics are supported by the same monitor and the same instrumentation of the application. Trace collection is needed to give data to PROVE for execution and performance visualisation. Statistics have less intrusion to the execution by generating fixed amount of data and they support initial evaluation of long-running programs.

The main goals in the design of a new monitor were strongly related to the VisualMP environment. They were:

1. Support of monitoring and visualisation of programs on GRAPNEL level. 2. It is part of an integrated development environment. 3. Portability. Use on heterogeneous clusters on UNIX operating systems (Irix, Solaris, Linux, DEC-Alpha, etc.) 4. Semi-on-line monitoring and visualisation to

support - evaluation of long-running programs, - debugger in VisualMP with execution visualisation. 5. Dynamic instrumentation to support - evaluation of long-running programs, - automatic performance analysers integrated into VisualMP in the future. 6. Both statistics and event trace collection should be supported. 7. Trace data should not be lost at program abortion. We want to visualise the execution to the point of abortion. 8. The execution of the application and the development environment should be separate. Thus, an application can be developed on a local host while it is executed on a remote cluster (and it is visualised semi-on-line on the local host).

The above goals simplified the design of the monitor in several aspects. The tight integration of GRM into VisualMP enabled us to put several functionalities of a stand-alone monitoring tool into other tools of VisualMP. For example, - instrumentation is naturally done in the graphical editor, - trace events are generated only by instrumentation function calls in the application processes not by the monitor - trace events of different processes are not sorted into time order since the preprocessing phase in PROVE does not need a globally sorted trace, - local monitors are started on the hosts defined by the environment, - the monitor is started and stopped by GRED, - the monitor does no bookeeping of processes, it does not even recognise the finish of the application.

The design goals gave constraints on the monitor, too. The high portability requirement forced us to use only standard UNIX solutions for every problems related to monitoring. Operating system or hardware specific solutions, counters or performance analysis tools must have been avoided. Fortunately, keeping us only on GRAPNEL level we could implement GRM in standard UNIX.

If trace events generated by a process are stored in local memory they can be lost when the process aborts. GRM should use shared-memory segments for trace storage in order to keep trace events accessible to VisualMP to the last point of execution. The use of shared-memory segments makes also possible the dynamic instrumentation of the program without modifying the application program code. However, this solution increases the intrusion since a new race condition is introduced by the use of a shared object.

Three basic problems arise when an application executed in a distributed environment is monitored. They are clock synchronisation, handling large amount of trace data and intrusion. This paper discuss these problems and the answers of GRM for the challenges.

References

[1] P. Kacsuk, G. Dózsa, T. Fadgyas, and R. Lovas: "GRADE: a Graphical Programming Environment for Multicomputers", Journal of Computers and Artificial Intelligence, Slovak Academy of Sciences, Vol. 17, No. 5, 1998, pp. 417-427

[2] P. Kacsuk: "Macrostep-by Macrostep Debugging of Message Passing Parallel Programs", PDCS'98, Los Angeles, 1998, pp. 527-531

[3] J. Chassin de Kergommeaux, E. Maillet and J-M. Vincent: "Monitoring Parallel Programs for Performance Tuning in Cluster Environments", In "Parallel Program Development for Cluster Computing: Methodology, Tools and Integrated Environments" book, P.Kacsuk and J.C.Cunha eds, Chapter 6., will be published by Nova Science in 2000.

[4] P. Kacsuk: "Performance Visualization in the GRADE Parallel Programming Environment", accepted for HPCN Asia, Beijing, China, 2000.