

EZTrace: a generic framework for performance analysis

François Trahay
Yutaka Ishikawa
Riken, University of Tokyo
{trahay, ishikawa}@il.is.s.u-tokyo.ac.jp

François Rue
Raymond Namyst
INRIA Bordeaux – Sud-Ouest
LaBRI, University of Bordeaux
{rue,namyst}@inria.fr

Mathieu Faverge
Jack Dongarra
University of Tennessee
{mfaverge,dongarra}@eecs.utk.edu

Abstract—Modern supercomputers with multi-core nodes enhanced by accelerators, as well as hybrid programming models introduce more complexity in modern applications. Exploiting efficiently all the resources requires a complex analysis of the performance of applications in order to detect time-consuming sections. We present EZTRACE, a generic trace generation framework that aims at providing a simple way to analyze applications. EZTRACE is based on plugins that allow it to trace different programming models such as MPI, pthread or OpenMP as well as user-defined libraries or applications. EZTRACE uses two steps: one to collect the basic information during execution and one *post-mortem* analysis. This permits tracing the execution of applications with low overhead while allowing to refine the analysis after the execution. We also present a script language for EZTRACE that gives the user the opportunity to easily define the functions to instrument without modifying the source code of the application.

Keywords—performance analysis; hybrid programming model; trace generation;

I. INTRODUCTION

The increasing need for computing power have led to designing massively parallel computers that now reach petaflops. For exploiting efficiently such supercomputers, application developers have to go through a crucial optimisation phase. However, this phase requires to understand precisely the behavior and the performance of the application. The complexity of supercomputer hardware as well as the use of various programming models like MPI, OpenMP, MPI+threads or MPI+GPUs makes it more and more difficult to understand the performance of an application.

The use of convenient analysis tools such as VAMPIR-TRACE [1] or SCALATRACE [2] is a great help for understanding the performance of an application. However, the variety of scientific libraries and programming models makes it mandatory for such tools to be generic. Instrumenting an application with these tools can be tedious since it requires to modify the source and to recompile the program. Allowing easy instrumentation of any kind of library or application is crucial in order to work on most modern platforms and to meet the requirements of emerging programming models.

This paper describes the design of EZTRACE, a generic

framework for performance analysis. EZTRACE uses a two phase mechanism based on plugins for tracing applications. This permits easy specification of the functions to analyze as well as the way they should be represented. Moreover, EZTRACE provides an easy to use script language that allows the user to instrument functions without modifying its source code.

II. EZTRACE: A GENERIC FRAMEWORK FOR PERFORMANCE ANALYSIS

EZTRACE has been designed for providing a generic way to analyze an application without modifying it nor impacting its execution. The use of *plugins* permits to select easily the type of application to analyze or the point to focus on. EZTRACE provides pre-defined *plugins* for analyzing applications that use MPI libraries, GNU OpenMP or Pthreads. However, user-defined *plugins* can also be loaded in order to analyze application functions or custom libraries.

EZTRACE uses a two phase mechanism for analyzing applications. During the first phase that occurs while the application is executed, functions are intercepted and events are recorded using the FxT library [3]. After the execution, the *post-mortem* analysis phase is in charge of interpreting the recorded events. During this phase, EZTRACE can generate a trace file viewable with visualization tools such as Vampir [4] or ViTE. It can also compute statistics in order to analyze the overall characteristics of the application (communication scheme, OpenMP parallel regions, etc.)

This two phase mechanism permits the library to separate the recording of a function call from its interpretation. This offers a global view of the application when analyzing events and avoids being restricted by reactivity issues. Separating the recording of the execution of an application from its analysis also permits to run the application once and to work on the collected traces later. It is thus possible to refine the analysis of a set of traces. For example, the execution of an application mixing MPI and OpenMP can be recorded using EZTRACE. The analysis of this execution can then focus on either MPI, OpenMP or both paradigms depending on the user interest.

```

NAME foo
DESC "Plugin for the foo library"
LANGUAGE C
TYPE LIBRARY

int foo(int arg1, int arg2)
BEGIN
    RECORD_STATE("doing function foo")
END

void bar(int arg1)
BEGIN
    EVENT("function bar called")
END

```

Figure 1. Example plugin.

III. ANALYZING USER-DEFINED FUNCTIONS

Although EZTRACE provides pre-defined plugins – for instance for MPI or OpenMP – users can create modules for their own applications or libraries. In order to ease the development of such plugins, EZTRACE provides a convenient way to create plugins using a script language. This language allows for easily describing the functions to instrument as well as their interpretation. User-defined scripts are then converted into plugins using a source-to-source compiler

Due to the two phase mechanism used in EZTRACE, writing a plugin requires specifying the way functions are instrumented and the interpretation of these function calls in the output trace. However, providing EZTRACE with the interpretation is often sufficient for guessing how to instrument a function. For example, changing the *state* of a thread during the execution of a function implies recording an event at the beginning and at the end of the corresponding function, whereas notifying the occurrence of a function with an *event* only requires recording an event at the beginning of the function. The script language thus focuses on the interpretation of these functions, leaving the instrumentation to EZTRACE. As illustrated in Figure 1, creating a plugin boils down to describing the plugin and interpreting each function that has to be analyzed. The source-to-source compiler then generates the source code that intercepts the function calls, records events and interprets them.

This script language provides a convenient way to create EZTRACE plugins: the small set of interpretation keywords permits acting on all the *trace modifiers* provided by the trace formats. Thus, the simplicity of the script language does not restrict the possibilities of event interpretation. Moreover, since the source-to-source compiler generates C files, it is possible to use them as a basis and to modify them in order to control more precisely the interpretation of a function. Creating a plugin for EZTRACE can thus be achieved easily by providing a set of functions to instrument. The interpretation of function calls can be controlled precisely. As a result, the script language is powerful enough for generating plugins for most users, while leaving the

possibility for the advanced users to analyze more precisely an application execution.

IV. CONCLUSION AND FUTURE WORKS

The complexification of hardware and hybrid programming models make it more and more difficult to efficiently exploit current supercomputers. Application developers thus need convenient tools for understanding the performance of their programs and to detect the phases to improve. Even if several tools offer the capability to trace MPI or OpenMP events, tracing a set of user-defined functions can be tedious and it requires to modify the source code of the application.

We propose a generic framework for performance analysis based on two steps. During the execution of the program, a set of functions specified by plugins are instrumented and trace files are generated. After the application run, a *post-mortem* analysis interprets the traces. This permits generating execution traces with a low impact on the performance of the application while allowing refinement of the analysis after the program run. The script language developed with EZTRACE provides a simple way for the user to describe which functions he/she wants to follow and how to analyze the data collected.

EZTRACE is available as an open source project online¹. Our future work is to integrate hardware counters collected by libraries such as PAPI [5] into our trace generation in order to follow the evolution of these counters during execution. We also plan to improve the scalability of EZTRACE as well as the *post-mortem* analysis capabilities.

REFERENCES

- [1] M. Muller, A. Knupfer, M. Jurenz, M. Lieber, H. Brunst, H. Mix, and W. Nagel, "Developing scalable applications with Vampir, VampirServer and VampirTrace," *Minisymposium on Scalability and Usability of HPC Programming Tools at PARCO*, 2007.
- [2] M. Noeth, P. Ratn, F. Mueller, M. Schulz, and B. de Supinski, "ScalaTrace: Scalable compression and replay of communication traces for high-performance computing," *Journal of Parallel and Distributed Computing*, vol. 69, no. 8, 2009.
- [3] V. Danjean, R. Namyst, and P. Wacrenier, "An efficient multi-level trace toolkit for multi-threaded applications," *Euro-Par 2005 Parallel Processing*, 2005.
- [4] A. Knüpfer, H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M. Müller, and W. Nagel, "The Vampir Performance Analysis Tool-Set," *Tools for High Performance Computing*, 2008.
- [5] S. Browne, J. Dongarra, N. Garner, G. Ho, and P. Mucci, "A portable programming interface for performance evaluation on modern processors," *International Journal of High Performance Computing Applications*, vol. 14, no. 3, 2000.

¹<http://eztrace.gforge.inria.fr/>