# Performance Cockpit: An Extensible GUI Platform for Performance Tools*

Tianchao Li and Michael Gerndt

Institut für Informatik, Technische Universität München,
Boltzmannstr. 3, D-85748 Garching bei Muñchen, Germany
{lit,gerndt}@in.tum.de

**Abstract.** Within the EP-Cache project, the Performance Cockpit has been developed to provide a unified GUI for a series of performance tools. This is achieved through the establishment of a general extensible architecture and the application of standardized intermediate representations of program structures. This paper describes the design and implementation of this platform, and discusses the future evolvement into a universal GUI platform for performance tools independent of programming language and programming paradigms.

## 1 Introduction

Performance tools are commonly used in high performance computing in order to understand and correct the performance problems of sequential and parallel codes. Such tools monitor a program's execution and produce performance data that can be analyzed to locate and understand areas of poor performance.

There are a number of performance tools, both research and commercial. Many of these tools are language-dependent and can be applied to high performance programs written in one or more of FORTRAN, C, C++ etc, while some are language-independent. There are also different programming paradigms, typically shared memory (PThread, OpenMP) and message passing (MPI, PVM). The support for those programming paradigms also varies.

The most prevalent approach taken by these tools is to collect performance data during program execution and then provide post-mortem analysis and display of performance information. Some tools do both steps in an integrated manner, while other tools or tool components provide just one of these functions. A few tools also have the capability for run-time analysis, either in addition to or instead of post-mortem analysis.

Typically, each performance tool provides a customized user interface for showing the structure of the application, specifying the target of measurement, controlling the measurement execution and displays the result of measurement. The diversity of those user interfaces demands a lot of time for studying the

* The work presented in this paper is mainly performed in the context of the EP-Cache Project, funded by the German Federal Ministry of Education and Research (BMBF)

J.C. Cunha and P.D. Medeiros (Eds.): Euro-Par 2005, LNCS 3648, pp. 104–113, 2005.

usage of a new performance tool, and makes it hard to integrate and incorporate different performance tools.

While the exact sequence can be different, performance measurement typically includes the common procedures: instrumenting the monitored program (source or binary, static or dynamic), linking the instrumented program with specific runtime libraries, program execution and result retrieval (online or post-mortem), and result display and/or analysis. This provides us the possibility to set up a general infrastructure to support different performance tools. In this infrastructure, a performance tool independent platform serves as the basis that can be extended by individual modules (plug-ins) to support different performance tools.

In the German EP-Cache project, Performance Cockpit has been implemented as an extensible GUI platform for a series of performance tools. Based on the open source tooling platform Eclipse [4], this platform supports both post-mortem (i.e. CPTE) and online monitoring (i.e. EPCM) environments, and is intended to integrate other performance tools. The Performance Cockpit serves as the starting point for the development of a universal GUI that is neutral to both programming languages and programming paradigms.

The remainder of this paper is organized as follows. Section 2 introduces the performance monitoring tools developed in the EP-Cache project, and discusses the need for a common GUI platform. Section 3 discusses the major issues involved in the design and development, including the establishment of a general extensible architecture and the standardization of information representations. Section 4 presents the defined architecture, and section 5 introduces the GUI and its typical scenario of usage. Section 6 discussed activities related with our development, and Section 7 looks forward to the development of a universal platform for performance tools based on the efforts in Performance Cockpit. The paper concludes with a short summary in Section 8.

## 2 The Need for a Common GUI Platform

The EP-Cache (Efficient Parallel Programming of Cache Architectures) project is a three-year research project funded by the German Federal Ministry for Education and Research. The goal of this project is to develop new performance analysis tools and performance tuning techniques with which programs can be improved to efficiently utilize the underlying cache architectures, especially on SMPs. As a fundamental part of the EP-Cache project, existing performance measurement tools are evaluated and new tools either using existing hardware counters and/or based on a novel hardware monitor design [7] that reveals further details on the access behaviors for individual data structures and code regions are developed. These include the Counter-based Profiling and Tracing Environment (CPTE) [2] and the EP-Cache Monitor (EPCM) [5].

CPTE is a performance monitoring tool based on hardware performance counters. It provides profiling, tracing, and sampling for arbitrary program regions. Performance measurement with CPTE is done with the following steps -

instrumentation of the program, specification of measurements, program execution and generation of the measured values, and analysis of resulting performance data. Measurement results are produced in the form of a trace file which may contain measurements for individual instances of a region, and/or summaries of all instances of a region. The results can also be transformed for visualization with KCachegrind [6].

EPCM is a data-structure centric performance monitoring tool. It is based on a novel hardware monitor [7] designed to be integrated into cache controllers which provides counters that can be configured to measure events for certain address ranges, and record the accesses in the form of event counts and access histograms. As the hardware monitor is not available, EPCM is actually implemented on top of a simulator that provides runtime instrumentation of application binary, on-the-fly simulation of the cache access behavior and performance monitoring for multi-processor shared memory systems [14]. EPCM provides Monitoring Request Interface (MRI, ref. [8]), through which performance analysis tools can specify monitoring requests and retrieve monitoring data in online fashion. EPCM also generates trace records compatible with VAMPIR [16] that is extended with OpenMP, data structure and histogram support.

CPTE and EPCM share some similarities in that both environments are targeted to Fortran 95 OpenMP programs and extendable for other programming languages and programming paradigms provided that the specific instrumenters are available. Both require selective code-region instrumentation in user specified source files and region types. The differences between these two environments are even more evident. The post-mortem data analysis in CPTE and online monitoring and analysis in EPCM requires different procedures in the measurement. EPCM's support for code-regions involves additional code region instrumentation and different specifications of measurement targets. The measurement results are also different for CPTE and EPCM in both content and format, and are to be visualized with different visualization and analysis tools.

In order to ease the usage, graphical user interfaces (GUIs) are demanded for both CPTE and EPCM. Taken into consideration of the vast differences between those two environments, it might seem a natural choice to develop separate GUI for each of those platforms. However, this leaves many problems like duplicate work for the common features, low maintainability, inconsistent in the user interface and low inter-operability. Instead, we have chosen another approach - to implement a common GUI to support both these monitoring environments as well as other existing and future monitoring environments through the establishment of a common extensible infrastructure, namely the Performance Cockpit.

## 3 Key Issues in Design and Implementation

In the design and implementation of such an extensible GUI platform as the Performance Cockpit, the major issues to be considered include the establishment of a general extensible architecture and the standardization of information representations.

### 3.1 Define General Extensible Architecture

A general architecture should be constructed for integrating different performance tools through extension. Generality and extensibility are the major considerations of the defined architecture. While the powerful extension mechanism from Eclipse provides extensibility, the GUI elements required by different tools are to be studied and organized with respect to their nature for generality.

The generic GUI elements and the underlying supporting mechanisms form the basic platform, and the tool-specific elements are to be grouped into individual extension modules, i.e. plug-ins. Each plug-in extends the basic platform through properly defined interfaces, i.e. extension points. The interface between the basic platform and the extensions should be defined generic enough to allow possible situations of extensions.

For more details of the established architecture, please refer to Section 4.

### 3.2 Define Standard Representation for Relevant Information

For the interaction between Performance Cockpit and the different performance tools that are integrated, standardized representation should be defined for all relevant information. The information includes program code region structure, program instrumentation targets and/or monitoring requests, as well as the measured performance data.

For the program code region structure, we have participated in the development of Standardized Intermediate Representation (SIR) [13], a standardized abstract representation of program structure for Fortran 95, Java, C and C++ programs defined in the APART working group [1]. SIR is defined in the format of XML document; each SIR is a XML document following the DTD or XML schema definition for SIR. SIR is intended to be used by performance tools and contains only high-level information about positions and types of statements and directives (e.g. OpenMP) that represent the coarse structure of programs, as opposed to more complicated intermediate languages like WHIRL used in the Open64 compiler suite [9]. This simplicity helps keep SIR compact and applicable for both procedural and object-oriented programs of various languages.

For the information of program instrumentation and monitoring, common formats that are general enough for the performance tools of EP-Cache project are also defined.

## 4 The General Extensible Architecture

A general architecture for integrating different performance tools has been constructed (see Figure 1). This architecture follows a layered design and is based on the extension mechanism provided by Eclipse.

In this architecture, the generic functions including the management of monitoring projects (new project or example project), configuration of common preference and project properties, and the management of platform extensions forms

the performance platform. The support for code instrumentation is provided with separate instrumentation plug-in, each for a different instrumenter. And the concrete support for different underlying monitoring platforms, either based on hardware counters (the CPTE platform), or software simulators (the EPCM platform) are implemented as separate plug-ins.

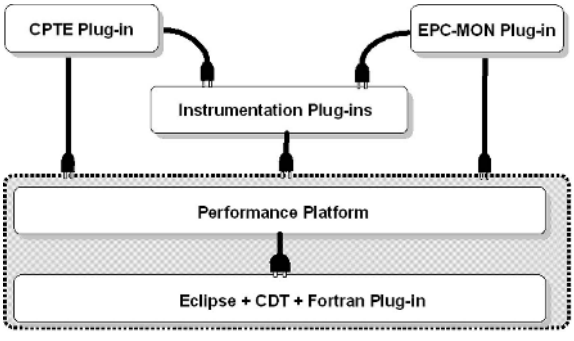


**Fig. 1.** The General Extensible Architecture for Performance Cockpit

### 4.1 Eclipse, CDT and Fortran Plug-In

Eclipse [4] is a kind of universal tooling platform - an open-source extensible IDE for the integration of various software development tools. Eclipse represents a component-based approach for software development, which promotes a view of software development in which applications are composed out of reusable, relatively large-grained, and mostly pre-existing components.

The C/C++ Development Tools (CDT) [3] provides a full functional C and C++ IDE for the Eclipse platform. It provides support for C/C++ edit, build, launch, and debug. For project building, CDT incorporates a *standard make* feature (a term used by Eclipse to represent a group of tightly related plug-ins) that support standard makefiles.

By the time of implementation, Fortran support for Eclipse is not available, and a self-developed plug-in has been developed for simple Fortran support. It implements a Fortran 95 editor with simple syntax highlighting, and supports the building of Fortran applications by reusing the incrementally build functionality provided by the makefile support of CDT. This part can be replaced once an advanced Fortran plug-in such as Photran [11] becomes mature.

### 4.2 The Performance Platform

The performance platform provides the basis for integration and extension for different performance tools. It is based on Eclipse, Eclipse CDT and the self-developed Fortran plug-in. The major function of this platform includes the management of the performance projects, the management of common properties

and preferences, and the management of performance tools extensions. For the management of performance tools extensions, custom interfaces (i.e. extension points), are defined by the performance platform, through which the individual performance tools can be discovered and integrated.

### 4.3 Instrumentation Plug-Ins

Code instrumentation is a separate process from performance measurement, and is often shared among tools. For each specific code instrumenter, a separate instrumentation plug-in is to be implemented. Each of these plug-ins provides GUI elements for the instrumentation of the whole project or selected files of a certain type (e.g. Fortran programs), and directs the underlying instrumenter upon user control. The instrumenter also generates information about the source code structure, in the format of SIR, which will later be read by the GUI and the individual tools plug-ins.

### 4.4 Tools Plug-Ins

Each performance tool requires the development of an individual plug-in to be integrated into the performance platform. The responsibility of each plug-in includes the translation of standard-based data representation to tool-specific data formats, providing custom GUI elements, and the interaction with the underlying tools.

The plug-in for each performance tool must implement certain interfaces to be managed by the performance platform. User interactions with the common GUI elements are processed by the performance platform and translated into appropriate function calls as defined in the interfaces. User interactions with the custom GUI elements are directly handled by the tool plug-in.

## 5 The Performance Cockpit GUI

### 5.1 GUI Elements

In terms of Eclipse, the common GUI elements for performance monitoring provided by the Performance Cockpit include:

**Monitoring Perspective:** This perspective organizes all relevant components into a role-oriented GUI to the user of the monitoring environment.

**Project Creation Wizards:** These wizards help create projects that are either empty or containing example programs. Projects created with these wizards are marked with *Monitoring Nature*, which is identified later by other components of the Performance Cockpit.

**Monitoring Resource Explorer:** As a resource explorer customized for our measurement environment, it provides standard project and file manipulation functions; however all unnecessary details are hidden from the user, such as the

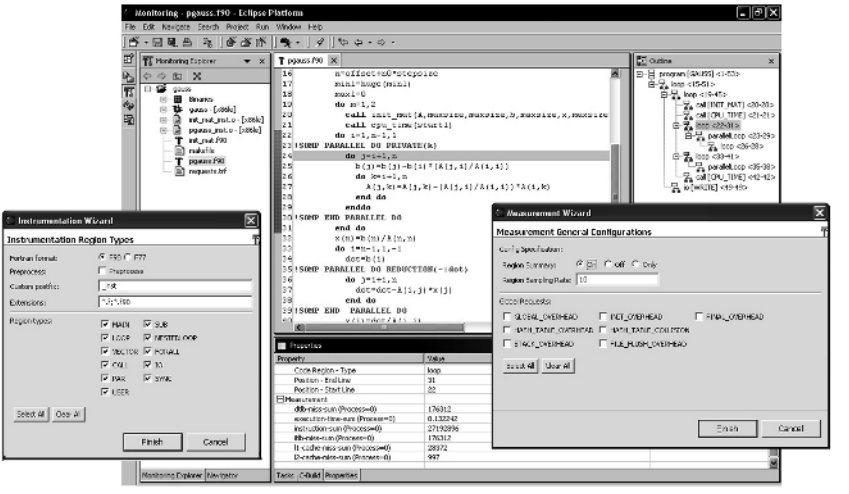


**Fig. 2.** Extensible GUI Platform for Performance Tools

files created during the process of instrumentation and internal configuration files.

**Monitoring Environment Preferences:** This enables required configurations for the monitoring environment, such as the path of instrumenter executable, the path of result format converter etc.

**Instrumentation Wizards:** These wizards guide the user through the process of instrumentation either for selected files or for the whole project.

**Code Regions Outline View:** This view provides an outline of code regions for the active editor, according to the result of instrumentation. Context menu items of this view also allow users to add/remove certain code region(s) into/from target of measurement.

**Code Region Properties View:** This view displays available properties and measurement results of individual code regions, in response to user selections in the Code Regions Outline View.

In the current implementation of the Performance Cockpit, the following performance tool specific GUI elements are defined by the plug-ins for CPTE and EPCM environment:

**Measurement Wizards:** The measurement wizards guide the user through the process of measurement. A separate wizard is defined for each of the performance tools. For example, the measurement wizard for CPTE directs the user to specify parameters and general requests for the measurement, generates configuration file, and launches the program measurement.

**Measurement Result Views:** These views display available properties and measurement results. For example, the EPCM displays the result of each measurement request as a single count or a histogram.

**Visualization Wizards:** These wizards guide the user to choose and invoke appropriate external visualizers to display the measurement results.

The above GUI elements, either provided by the common performance platform or contributed by the individual performance tools plug-ins, are seamlessly integrated with the Eclipse platform. From the user's perspective, those two types of GUI elements are not distinguishable (see Figure 2).

All components described above are grouped into a *Monitoring Feature*, which allows the whole platform to be installed and updated in a way that coexists with other Eclipse based systems.

### 5.2 Usage Scenario

The process of measurement using Performance Cockpit can be summarized as follows. The user creates a project with one of the Project Creation Wizards, and then the user can manipulate the content of the project with the Monitoring Resource Explorer. To do instrumentation, the user can right click on the whole project or selected files and start Project Instrumentation Wizard or Files Instrumentation Wizard from the context menu. After specifying the required parameters like region types to be instrumented, the instrumenter is invoked by the wizard. The user can then open the instrumented files in the Fortran Source Editor, examine the code regions in the Code Regions Outline View, and specify local measurement requests for specific code regions. After building the program, the user invokes the Measurement Wizard for measurement that will guide the user through out the measurement process, which varies from tool to tool. In any case, the wizard will launch the program for execution. Once the execution finishes, the user can choose appropriate Measurement Result Views to examine the individual measurement, or invoke visualization tools that are integrated in the platform through the help of Visualization Wizards.

## 6 Related Works

Previous attempts to construct general interfaces for instrumentation and visualization also exist in other parallel tool groups. The Pablo project [10] at the University of Illinois has implemented svPablo, a graphical interface for instrumenting source code and browsing runtime performance data. The Tool Gear project [15] at LLNL is a GUI tool and database for dynamic instrumentation and display of the instrumentation results. However, the extensibility and flexibility of such tools are not comparable to our Performance Cockpit. In fact, taken the vast differences between instrumentation and measurement tools (e.g. consider just profiling vs. tracing tools), opportunities for integration can be only guaranteed by a general extensible platform like the Performance Cockpit.

Existing Eclipse-based GUIs for performance tools include the Eclipse OProfile plug-in as a CDT contribution and the Intel VTune Performance Analyzer for Linux [17]. Both are specific to the underlying tool (OProfile and VTune), and none of them address the extensibility and coexistence with other performance tools. The tight dependence with Eclipse CDT also makes Eclipse OProfile plug-in restricted to C/C++. Intel VTune Performance Analyzer for Linux supports multiple languages, including Intel Visual Fortran, Java and languages supported by the Linux GNU Compiler Collection (GCC); however the proprietary nature of this product and its closed internal data models makes integration with other tools impractical.

The recently proposed PTP (Parallel Tools Platform) project [12] aims to extend the Eclipse framework to support a rich set of parallel programming languages and paradigms, and provide a core infrastructure for the integration of a wide variety of parallel tools. Although the PTP is still in the initial status of proposition, it casts new light on the construction of a generic platform including performance monitoring. We have expressed our interest in this project and will actively participate in the discussions to influence the design so that the work in PTP and our work can be seamlessly integrated.

## 7 Towards a Universal Platform for Performance Tools

The Performance Cockpit provides the basis for the future development of a universal integration platform for performance tools. Such a platform will be beneficial to users and developers of all performance tools in that it provides a consistent user experience and gentle learn curve, enables interoperability among performance tools, reduces redundant work by reusing common functions, etc. It is intended to be programming language neutral, programming paradigm neutral, and performance tool neutral.

While the extension architecture and the standardized representation of information defined in its development generally enables the step-forward towards universal platform, further efforts are required. In order to be performance tool neutral, the currently defined architecture should be refined, and standardized representation of more types of information (e.g. the trace record) should be defined. This requires of course the examination of a large amount of existing performance tools and identify the commonalities and specialties. This will also involve a lot of compromise between generality and functionality.

For Eclipse, supports to programming languages and programming paradigms are usually provided by extensions from different parties. The integration of Performance Cockpit with those diverse programming extensions constitutes another challenge, and will foreseenably result into changes to the general architecture and implementation. For example, the PTP described above that provides support for parallel programming will be integrated as part of the underlying platform.

## 8 Conclusions

Performance is a very important factor that drives the development of computing. Code optimization with the help of performance tools is one of the major measures to achieve better performance. However, the existing performance tools usually have different graphical user interfaces and results into difficulty in the usage and poor interoperability.

In the EP-Cache project, the Performance Cockpit, a GUI platform that provides a unified user interface for a series of performance tools, has been developed. Compared to other GUIs for performance tools, the Performance Cockpit excels in its easy learning and usage, its extensibility and interoperability. The general extensible architecture and standard representations for related information that are defined in the development of Performance Cockpit provide the basis for the future development of a universal platform for performance tools. The integration of performance tools with the Eclipse environment would also allow programmers of high-performance systems to exploit the general advantages of the integrated interactive development environment.

## References

1. APART Working Group. http://www.fz-juelich.de/apart/
2. M. Gerndt, T. Li: *Automated Analysis of Memory Access Behavior*, Proceedings of HIPS-HPGC 2005, Denver Colorado, April, 2005
3. Eclipse C/C++ Development Tools. http://www.eclipse.org/cdt/
4. Eclipse. http://www.eclipse.org
5. E. Kereku, T. Li, M. Gerndt, and J. Weidendorfer: *A Selective Data Structure Monitoring Environment for Fortran OpenMP Programs*, Proceedings of Euro-Par 2004, Pisa, Italy, Aug. 31th - Sept. 3rd, 2004
6. KCachegrind. http://kcachegrind.sourceforge.net
7. M. Schulz, J. Tao, J. Jeitner, W. Karl: *A Proposal for a New Hardware Cache Monitoring Architecture*, Proceedings of MSP 2002, Berlin, Germany. June 2002
8. M. Gerndt, E. Kereku: *Monitoring Request Interface Version 1.0*, http://wwwbode.in.tum.de/~kereku/projects/epcache/pub/MRI.pdf
9. Open64 Compiler Tools. http://open64.sourceforge.net
10. Pablo Research Group. http://www.renci.unc.edu
11. Photran. http://www.photran.org
12. Eclipse Parallel Tools Platform. http://www.eclipse.org/ptp/
13. C. Seragiotto et. al.: *Standardized Interfaces for Representing, Instrumenting, and Monitoring Fortran, Java, C and C++ Programs*, Concurrency and Computation: Practice and Experience, submitted.
14. SMART: A Simulation Tool for Monitoring Cache Access Behavior on SMPs, http://wwwbode.cs.tum.edu/~lit/smart/
15. Tool Gear. http://www.llnl.gov/CASC/tool_gear/
16. VAMPIR. http://www.pallas.com/pages/vampir.htm, www.tu-dresden.de/zhr/
17. VTune Performance Analyzer for Linux. http://www.intel.com/software/products/vtune/vlin/index.htm