Generic Library Interception for Improved Performance Measurement and Insight

Achievement: Develop the extension user library wrapping for the performance analysis framework Score-P that lets users create wrapper libraries for any C/C++ library. Score-P then uses these wrappers to give insight into library usage and performance of an application.

Significance and Impact: Because software and hardware systems become increasingly complicated, and applications aim to achieve more and more, the overall complexity of the software stack rises steadily. To simplify development and improve performance of common routines, functionality is encapsulated into submodules, e.g. libraries in C/C++. These developments necessitate performance optimization, but at the same time make debugging and reasoning about an application’s performance increasingly difficult.

_user library wrapping is one puzzle piece in performance analysis that enables highly improved insight in the use and performance of libraries in C/C++._

Research Details:
- Develop a libclang-based wrapper library generator
- Develop a workflow to guide users through the otherwise tedious and error-prone process of wrapper creation
- Demonstrate the usefulness of our implementation using two scientific applications and the robustness using various difficult to wrap libraries.

Sponsor/Facility: Oak Ridge Leadership Computing Facility

Team: Ronny Brendel¹, Bert Wesarg², Ronny Tschüter², Matthias Weber², Thomas Ilsche² and Sebastian Oeste² (¹ORNL, ²TU Dresden)

Publication: Ronny Brendel, Bert Wesarg, Ronny Tschüter, Matthias Weber, Thomas Ilsche and Sebastian Oeste, “Generic Library Interception for Improved Performance Measurement and Insight”, 6th Workshop on Extreme-Scale Programming Tools, Springer, 2017 (accepted preprint), (presentation), (github/examples)

Overview: Modern performance analysis tools like Arm MAP, Intel VTune Amplifier and HPCToolkit all use fixed library wrappers to intercept calls to important libraries like for example MPI, PThreads and OpenMP. This is needed to gain insight into the use of these libraries, and to e.g. extract information like which rank sends how many bytes to which other rank. Which libraries are intercepted depends on what the specific performance tool supports.

Generic library wrapping is popular for providing bindings for higher-level languages to libraries e.g. via SWIG or CLIF. None of the popular tools provide C/C++ wrappers for C/C++ libraries.

In performance analysis, generic library wrapping has been achieved before, but all of the solutions lack one or multiple desirable properties like robustness, usability, genericity or support for C++.

With the inception of LLVM/Clang, tools development in general became a lot easier. Developers can now use a fully-featured compiler to investigate source code. Before tools, like e.g. SWIG and TAU, relied on custom parsers or commercial ones, both of which have their shortcomings and cannot keep up with new language standards. Parsing C/C++ is very difficult.
Specifically, \textit{libclang} enables us to achieve robust C/C++ library wrapping that supports the newest language standards.

With this feature, which we implemented in the performance analysis framework Score-P, we can now:

- Record all interaction of an application with any C/C++ library.
- Record interaction of a library with itself and other libraries.
- Gain insight into closed-source libraries, since this technique requires only header files and library files (.a/.so) even without debugging information (no source code).

Technically library wrapping is a way of \textit{instrumenting}. The popular alternative to instrumentation is \textit{sampling}. While you can get insight into library usage through sampling, the usual limits apply: Since sampling interrupts and investigates in intervals, it attributes a time slice to the one function it hits, it does not capture all interactions, and cannot count the number of function calls. It also requires debugging information to be present in the application and libraries. Compiler-based instrumentation cannot provide what library wrapping does, since the compiler only instruments only what it compiles, i.e. the application sans libraries. Another upside of library wrapping over compiler-based instrumentation is that it only requires link time changes or preloading via LD\_PRELOAD. Recompiling is not necessary.

To create a wrapper library, you have to mimic the target library. For this you need the list of exported functions, including their names and signatures and all involved data type declarations. Library wrapping uses the linker to achieve interception. That means you can only intercept functions that are present as a symbol in the library file. On top of that some language restrictions as to which data types can be forwarded apply.

Therefore, the process of creating a wrapper is tedious and error-prone. To make creation and usage of the wrappers simple and robust, we conceived a workflow. It contains checks for many pitfalls and e.g. makes sure the list of functions from the header file analysis matches the symbol table of the library file.

We demonstrate the usefulness of the approach using GROMACS and PERMON, wrapping FFTW and PetSC. Furthermore, we provide other examples. To demonstrate the robustness, we wrapped Qt’s QtGui and QtWidgets modules. Qt is a large and the most popular C++ GUI framework. It surprised even us that our implementation is able to wrap it.

\textbf{Figure 1:} Example application without (white background) and with (blue) user library wrapping. Although a substantial amount of time is spent in the GNU Multi-Precision library (orange), traditional performance analysis and visualization does not show it but instead attributes the time to the outer functions in OpenMP and other application functions.