Chaining Multiple Tools and Libraries Using Gotcha

Yiheng Xu†, Kathryn Mohror*, Hariharan Devarajan*, Cameron Stanavige*, Abhinav Bhatel†
†University of Maryland, College Park
‡Lawrence Livermore National Laboratory

Abstract

• Users often want to run applications linked with multiple tools.
• Tools have been primarily to performance tools or user-level libraries that work by intercepting functions.
• Currently, applications cannot run with more than one tool that intercepts a common set of functions in a single run.
• Using multiple tools in the same run can avoid having to run the application more than once. It can thus save time and produce more temporally aligned data at different granularities.

Motivation

• Most tools define wrappers that override functions to be intercepted.
• Runtime finds wrappers through the Global Offset Table (GOT). When a function is needed, the wrapper on the top will be used.
• However, using multiple tools that intercept a common set of functions in the same run can lead to erroneous results.
• When multiple tools define wrappers for the same function, the system decides their order in GOT and which one to call in a non-deterministic way.

Methodology

• To solve this problem, we utilize a library called Gotcha, which lets users explicitly define wrapper functions and bind them in functions to be intercepted.
• Gotcha provides APIs for users to create bindings, set priorities, and get the real (or next) function that is wrapped.
• Issue: Most tools intercept MPI_Init and have their initialization in it. To use Gotcha, each tool has to initialize Gotcha as well. But we cannot put Gotcha initialization in MPI_Init, otherwise it won’t be wrapped by Gotcha.
• Solution: contractors of shared libraries.
• The constructor runs when the shared library is loaded, typically during program startup. We defined a constructor for each tool and put Gotcha initialization in it.

The constructor handles for all functions it intercepts.

Application

• To use Gotcha, we added Gotcha support to two I/O performance tools, Recorder and Darshan.
• We present three benefits our work brings to HPC tools:

Results

Benefit 1: Ability to chain tools and libraries together.
• A program linked with Recorder, Darshan and UnifyFS (a user-level file system) can run simultaneously.
• Gotcha wrappers and runtime environment calls the constructor, which initializes Gotcha so that functions can be wrapped.

Results (continued)

Benefit 2: Profiles for a same run more comparable by eliminating system noises across runs.
• Different tools are often used to profile the same application.
• Each run can only use one tool at one time, which means people have to run repeatedly to use these tools separately.
• System noise makes profiles from tools for the same job not able to complement each other.

Results (continued)

Benefit 3: Saves time and effort for running experiments with multiple tools.
• The time and resources wasted on doing duplicated work can be significant.
• It’s straightforward that time needed for using tools together in a same run is about it times shorter than running them separately with a tool.

Conclusion

• Utilized Gotcha to chain tools together. We have successfully made Recorder and Darshan Gotcha enabled.
• Demonstrated the benefit of work when using multiple tools being able to compare tools’ performance, reducing the effect of system noise for the same run with different tools, and saving time.
• Future: Formulate a standard way to make tools Gotcha enabled. See how performance overheads change with more tools chained together.

References