Chaining Multiple Tools and Libraries Using Gotcha

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

Abstract

- Users often want to run applications linked with multiple tools.
  - Tools here refers primarily to performance tools or user-level libraries that work by intercepting functions.
  - Currently, applications cannot run with more than one tool that intercept 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 profiles.
- Our work makes it possible to chain tools together, making profiles more comparable, and saving time for runs with different tools.

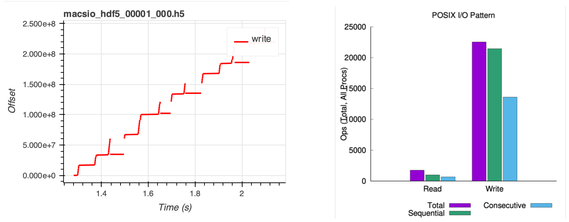

Fig 1. Recorder and Darshan are two performance tools that record I/O data at different granularities.

Motivation

- Most tools define wrappers that override functions to be intercepted.
- Runtime finds wrappers through the Global Offset Table[4]. When a function is needed, the pointer 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.

Global Offset Table (GOT)
fwrite@wrapper1.so
fwrite@wrapper2.so
fwrite@wrapperN.so
fwrite@libc.so.6

Methodology

- To solve this problem, we utilize a library called Gotcha, which lets users explicitly define wrapper functions and bind them to 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: constructors 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.

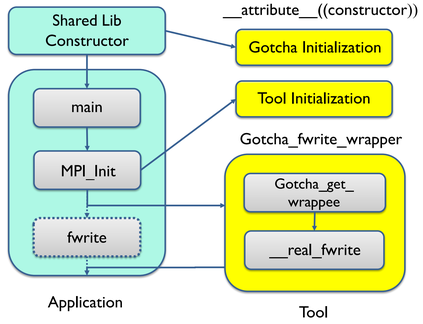

Fig 2. Modified control flow of an end-user application when using a Gotcha enabled tool with it.

- Runtime environment calls the constructor, which initializes Gotcha so that functions can be wrapped.
- Then the actual application starts to run. `MPI_Init` is redirected to tools' `MPI_Init` wrappers (chained), where each tool initializes. Similarly, when other functions are called, each tool intercepts them one by one.
- For each tool that needs to be chained, we modify its source code to define Gotcha wrappers and wrappee handles for all functions it intercepts.
- Then define a constructor, which uses Gotcha to bind wrappers with functions to be intercepted, and set the priority for it to control the ordering of tools.

Results

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

Benefit 1: Ability to chain tools and libraries together.

- A program linked with Recorder, Darshan and UnifyFS (a user-level file system)
- Fig 3 shows Recorder is able to catch Darshan's work and I/O to a new directory mounted by UnifyFS.
- It provides a simple way to check overheads of a tool (Darshan here)

Fig 3. Performance data captured by Recorder for UnifyFS[5] and Darshan when the tools are chained together.

Benefit 2: Make 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 repetitively to use these tools separately.
- System noise makes profiles from tools for the same job not able to complement each other.
- Tested with two benchmarks, MACSio and IOR.
- We run them with Recorder and Darshan loaded separately and then together.
- Pick some common metrics and check the difference between Darshan and Recorder reports.
- The difference when running them together is much less than that when running them separately.

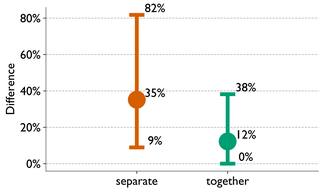

Fig 4. Difference between common metrics ("time of longest write on rank 0") reported for the same application when the tools are used separately versus together in the same run.

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 n times shorter than running them separately with n tools.

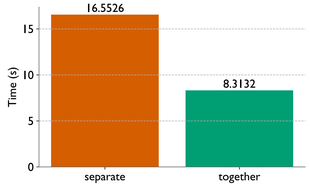

Fig 5. Time spent in getting both Recorder & Darshan profiles when the tools are used separately versus used together in the same run.

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 measure 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

[1] C. Wang, J. Sun, M. Snir, K. Mohror and E. Gonsiorowski, "Recorder 2.0: Efficient Parallel I/O Tracing and Analysis," *IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2020.

[2] Darshan. https://www.mcs.anl.gov/research/projects/darshan/

[3] Gotcha. https://gotcha.readthedocs.io/en/latest/

[4] Linux Foundation. https://refspecs.linuxfoundation.org/

[5] UnifyFS. https://unifyfs.readthedocs.io/en/latest/

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-POST-825920.