I/O Traces of HPC Applications

Chen Wang[1]
chenw5@illinois.edu

Kathryn Mohror[2]
kathryn@llnl.gov

Marc Snir[1]
snir@illinois.edu

I. Introduction

Understanding HPC application I/O behavior is an important task for improving performance. To aid in improving this understanding, we have created a publicly available dataset of I/O traces of 14 HPC applications that includes records of the layered I/O including HDF5, MPI-IO and POSIX. We have made our dataset public so that it can be used repeatedly by researchers to perform different analysis tasks with the goal of optimizing I/O performance or designing more efficient I/O libraries and file systems. In this work, we give background information about our I/O trace dataset along with some example analysis. Our traces are available at https://doi.org/10.6075/J0Z899X4.

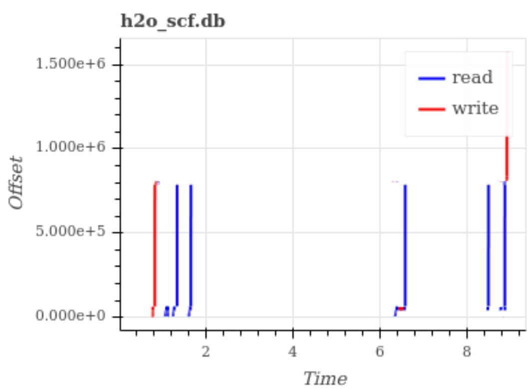

Fig. 1. Overlapping Accesses of NWChem

II. Dataset

The first release of our dataset includes I/O traces from 14 HPC applications spanning a variety of domains. Those applications perform I/O using POSIX, MPI-I/O and other higher level libraries such as HDF5, NetCDF, Silo and ADIOS. We utilized the multi-level I/O tracing tool Recorder [1] to generate the traces from those applications. The trace records include entry/exit time stamps, function name, and all function parameters, except the data buffer. The detailed traces enable I/O researchers to perform useful analysis such as identifying access patterns, detecting conflicting accesses, etc.

For example, Figure 1 shows the accesses of an internal database file in NWChem throughout the computation. The accesses exhibited both read-after-write and write-after-write patterns, which suggest that local caches maybe helpful. Figure 2 shows the count of unique write sizes observed in FLASH with independent I/O. As can been seen, there are a large number of small writes (e.g., 512 bytes) that could potentially hurt the performance. Using collective I/O significantly reduces the number of small writes (figure not shown due to space limit) but also introduces additional communication cost as suggested in Figure 3. In the figure, `MPI_File_write_at_all()` and `MPI_File_write_at()` spend in total of 30 seconds whereas `write()` takes only about 7 seconds.

[1]University of Illinois at Urbana-Champaign
[2]Lawrence Livermore National Laboratory

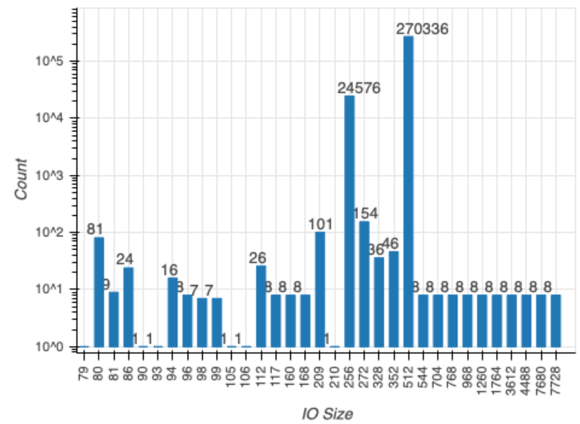

Fig. 2. Write sizes of FLASH using independent I/O

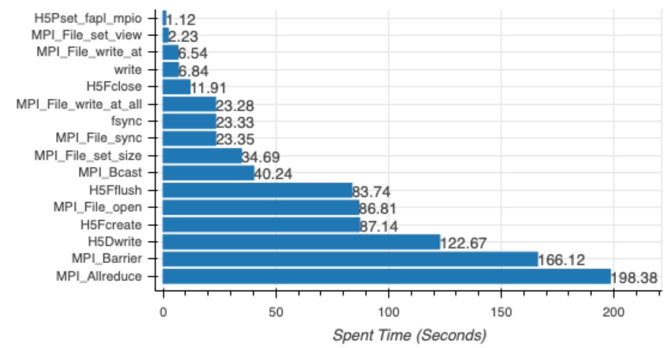

Fig. 3. Most expensive functions of FLASH using collective I/O.

III. Future Work

The current traces of each application were generated from one or a few configuration runs. We plan to include more configurations (especially those used in real scenarios) and also more applications in our future work.

References

[1] C. Wang, J. Sun, M. Snir, K. Mohror, and E. Gonsiorowski, “Recorder 2.0: Efficient parallel I/O tracing and analysis,” in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 1–8, IEEE, 2020.

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-ABS-815154.