Darshan for HEP applications

Douglas Benjamin\textsuperscript{2}, Patrick Gartung\textsuperscript{3}, Kenneth Herner\textsuperscript{3}, Shane Snyder\textsuperscript{1}, \textit{Rui Wang}\textsuperscript{1}, Zhihua Dong\textsuperscript{2}

1. Argonne National Laboratory
2. Brookhaven National Laboratory
3. Fermi National Accelerator Laboratory

Thursday, 11 May, 2023
Modern HEP workflows are increasingly scaled and complex
- Running on big computing farms or world-wide grid

HPC facilities may be employed to help to meet the growing data processing needs of these workflows and to reduce the time required to make new scientific insights

Ability to instrument the I/O behavior of the HEP workflows could be critical to characterize and understand their I/O patterns and underlying bottlenecks to be able to meet the performance expectations of the HPC systems
Darshan

- **Darshan** is a lightweight I/O characterization tool that captures concise views and entire traces (DXT) of applications’ I/O behavior

- **Widely available** – Deployed (and commonly enabled by default) at many HPC facilities
  - LCFs, NERSC, etc. and CVMFS

- Has become a popular tool for HPC users to better understand their I/O workloads
  - *Easy to use* – no code changes required
  - *Modular* – straightforward to add new instrumentation sources

https://www.mcs.anl.gov/research/projects/darshan/
Darshan enhancements for HEP use case

- Originally designed specifically for message passing interface (MPI) applications, but recently we have modified Darshan to also work in non-MPI contexts
  - HEP workflows are traditionally not been based on MPI
  - In recent Darshan versions (3.2+), any dynamically-linked executable can be instrumented

- Ability to instrument the forked processes
  - AthenaMP (multi-process offline software of ATLAS) creates parallel workers which are forked from the main process
Case study: ATLAS workflow

Broadwell on LCRC@ANL
GPFS

50 events, 8 threads
7267 seconds
~90 MB/s

3600 events, 8 processes
~3800/1908 seconds
~53.1/326.8 MB/s

Worker proc: ~1300MB ~0.32MB
Write proc: ~3.13MB ~485MB
~314MB ~380MB

Simulation
Generation ➞ ROOT ➞ Simulation ➞ ROOT ➞ Reconstruction ➞ ROOT ➞ Filtering ➞ ROOT ➞ Analysis

Data
RawData ➞ ROOT

AthenaMT (multi-thread Athena)
❖ Gaudi task scheduler maps tasks to kernel threads
❖ Shared single pool of heap memory

AthenaMP+SharedWriter
❖ Utilizing the Copy on Write principal to share memory across workers
❖ A shared writer executed alongside the other workers which retrieves the output data objects from the workers and merges them on the fly
Case study: ATLAS workflow

Broadwell on LCRC@ANL
GPFS
SDCC@BNL
Lustre

50 events, 8 threads
7267 seconds
~90 MB/s

Worker proc: ~1300MB
Write proc: ~3.13MB

~314MB
~380MB

~3800/1908 seconds
~53.1/326.8 MB/s

~6709 MB
~106MB

405K events, serial
1319 seconds
~84.5 MB/s

xAOD analysis
First look on analysis stage
Case study: CMS workflow

Haswell on Cori @Nersc
SSD + Lustre

100 events, 16 threads

Simulation

Data

~0.007MB  ~193MB

~188MB  ~6831MB

~188MB  ~6831MB

~5110MB  ~1857MB

1019 seconds  ~128.9 MB/s

1648 seconds  ~266.6 MB/s

383 seconds  ~321.4 MB/s

~431MB  ~25MB

~431MB  ~25MB

1680 seconds  ~49 MB/s

~5110MB  ~1857MB

~6831MB
Case study: I/O operations

Equal number of writes/seeks
- Generation & Simulation & Reconstruction & SharedWriter process in Filtering stage at ATLAS (marked)

Equal sequential & consecutive I/O
- Sequential – next access came somewhere after the last one in the file
- Consecutive – next access starts with the byte immediately following the last access
Case study: I/O operations

- **Seeks > reads**
  - Filtering stage (worker process at ATLAS)

- **Sequential > consecutive I/O**
  - Sequential – next access came somewhere after the last one in the file
  - Consecutive – next access starts with the byte immediately following the last access
Case study: Access size

Broadwell on LCRC@ANL
GPFS

Simulation

Generation → ROOT

Simulation → ROOT

Reconstruction → ROOT

Filtering → ROOT

Analysis

RawData

Data

Haswell on Cori @Nersc
SSD + Lustre
100 events, 16 threads

Small reads/writes at O(1KB)

- All stages (marked) except ATLAS Analysis which is at O(100KB)
- Related to ROOT TTreeCache vector I/O support on certain FSes
- Potential bottleneck
- ROOT has a data sieving concept (overread) that might be taken advantage of
Next steps for Darshan

❖ Instrumentation of Intel DAOS I/O libraries
  ➢ Upcoming exascale system at Argonne, Aurora, will feature a new-to-HPC object-based storage system
  ➢ Appealing performance characteristics for I/O middleware (e.g., HDF5 and ROOT) that can effectively leverage storage model
  ➢ File-based module complete, native object-based module underway

❖ Darshan analysis tools for workflows
  ➢ Refactor PyDarshan code to more easily allow aggregation and visualization of Darshan data across multiple logs
    ■ Multiple logs generated by the steps of an HEP workflow
Conclusion

❖ Darshan is a tool developed that could help to improve HEP workflows
   ➢ Characterize I/O activities of various workflow stages at scale
     ■ Amount of data movement in various phases
     ■ Patterns and sizes of access
     ■ Guide performance optimization in response to mismatch of behavior with HPC best practice
   ➢ I/O behavior are mostly as expected for ATLAS and CMS workflow
   ➢ Dune workflow has also been looked into

❖ Guide the further tuning of the I/O patterns to better inform storage capabilities requirements at facilities
   ➢ ROOT
   ➢ HDF5 (DUNE will write Raw data in HDF5)

❖ Uncover the I/O bottlenecks in current workflows when deployed at scale
   ➢ CPU & GPU

❖ Provide recommendations for data format and access patterns for future HEP workloads
Acknowledgments

❖ This work was supported by the U.S. Department of Energy, Office of Science, Office of High Energy Physics, High Energy Physics Center for Computational Excellence (HEP-CCE).

❖ This work is in part supported by the Director, Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-06CH11357; in part supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy’s Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation’s exascale computing imperative; and in part supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program.

❖ This research used resources at Argonne Leadership Computing Facility (ALCF), Argonne Laboratory Computing Resource Center (LCRC), NERSC and BNL Scientific Data and Computing Center (SDCC).
Backups
Darshan runtime library

Detailed runtime library configuration

- HEP Python frameworks access tons of files, many irrelevant for I/O analysis (shared libraries, headers, compiled Python byte code, etc.)
- Darshan users need more control over memory limits and instrumentation scope
- Comprehensive runtime library configuration integrated into Darshan
  - Total and per-module memory limits
  - File name patterns to ignore
  - Application name patterns to ignore

```bash
# allocate 4096 file records for POSIX and MPI-IO modules
# (darshan only allocates 1024 per-module by default)
MAX_RECORDS     5000      POSIX

# the '*' specifier can be used to apply settings for all modules
# in this case, we want all modules to ignore record names
# prefixed with "~/home" (i.e., stored in our home directory),
# with a superseding inclusion for files with a "*.out" suffix)
NAME_EXCLUDE    .pyc$,,^/cvmfs,,^/lib64,,^/lib,,^/blues/gpfs/home/software   *
NAME_INCLUDE     .pool.root.*   *

# bump up Darshan's default memory usage to 8 MiB
MODMEM  8

# avoid generating logs for git and ls binaries
APP_EXCLUDE     git,ls,sh,hostname,sed,g++,date,cclplus,cat,which,tar,ld
```
ATLAS offline software – Athena

Serial Athena

Multi-Process
- **AthenaMP+standalone merging**
  - Independent parallel workers are forked from main process with shared memory allocation
  - Each worker produces its own outputs and merged later via a post-processing merge process

- **AthenaMP+SharedWriter**
  - A shared writer process does all the output writes
  - Reduce time on single thread merging process

- **AthenaMP+sharedWriter (parallelCompression)**
  - Using parallel compression to reduce the time increment when moving to higher No. of process

Multi-thread
- **AthenaMT**
  - Gaudi task scheduler maps task to kernel threads
  - Shared single pool of heap memory

https://twiki.cern.ch/twiki/bin/view/AtlasPublic/ComputingandSoftwarePublicResults
CMS workflow – different hardware

Local skylake CPU
HDD
200 events, 16 threads

Haswell on Cori @Nersc
SSD + Lustre
100 events, 16 threads

1019 seconds
~128.9 MB/s
~10287MB ~4458MB

~5110MB ~1856MB

4104 seconds
~685 MB/s