HDF5 as intermediate storage for HPC processing

Saba Sehrish for the CCE IOS team

HEP CCE All-hands Meeting Meeting Spring 2023

Goal

- Evaluate if moving intermediate data (i.e. data between different processing steps) of an HEP workflow to a parallel file format (such as HDF5) could be beneficial for HEP data processing on HPC?
  - To support complex data models the strategy is to use ROOT serialization and store BLOBs in HDF5

Recap from Fall 2022 - Next steps and plan for FY23

- Complete performance evaluation studies on Cori
  - Understand IO behavior, untimed calls that are actually constituting 50% of the IO time, use Darshan logs
- Move evaluation studies to Perlmutter soon
- Write a paper/report
- HDF5 optimizations

Current status overall

Major Activities	Status
Multithreaded test framework development	Complete
Serial output and input modules development and evaluation	Complete
MPI extension and Parallel HDF5 output modules development and evaluation	Evaluation
Report	In progress

In the last all hands meeting in October, we presented the parallel design for HDF5 approach and some initial studies.

Since then we are focusing on gathering more results on Cori (later on Perlmutter if needed) and analyzing them and working on our final report.

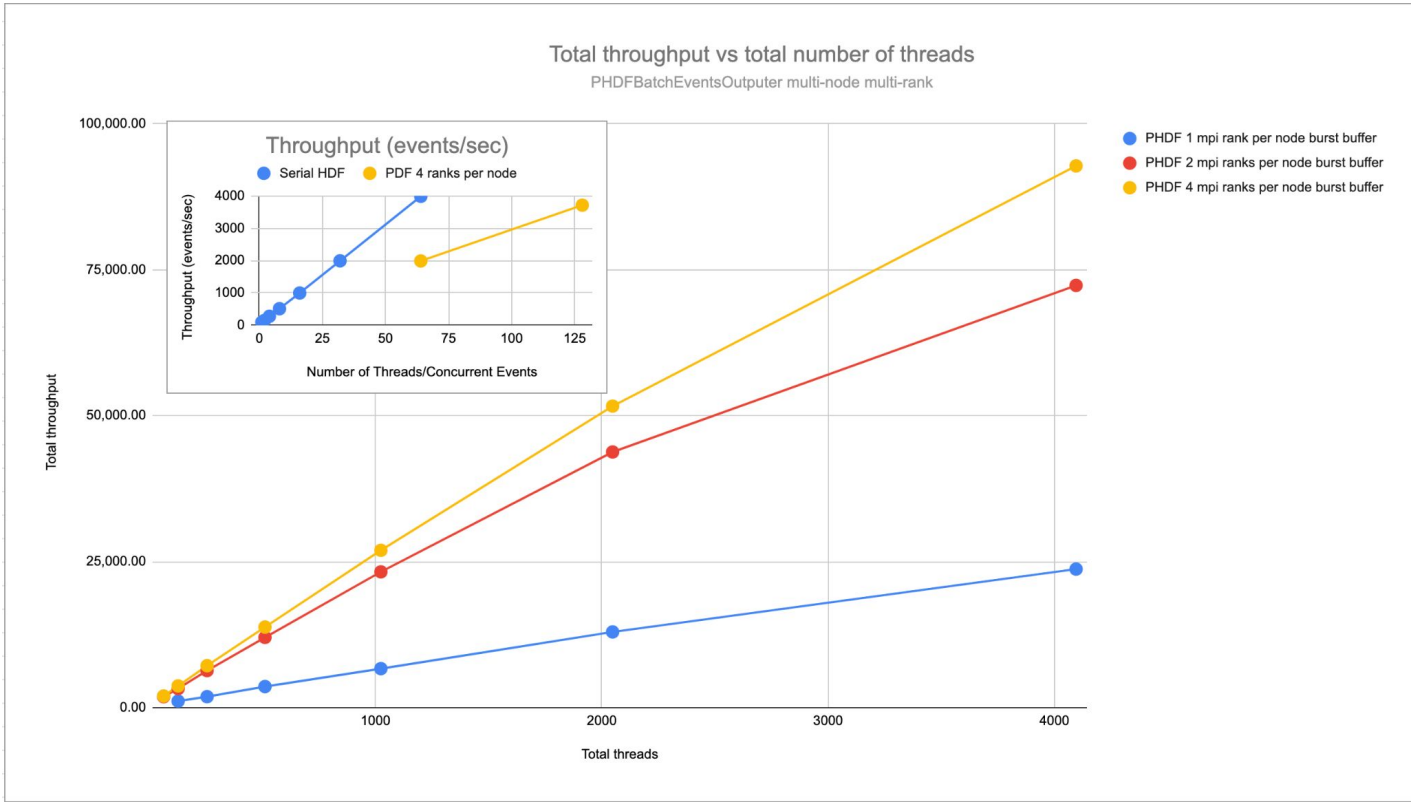
Some observations and takeaways

- Snowmass 2021 paper based on some of the initial work
- The test framework with an experiment independent design provided a means for studying
  - Root serialization and thread scaling behavior beyond what can be done now (more than 8 threads …)
  - Investigate various input/output modes
- There is no scaling limitation imposed by the test framework, anything we observe in IO scaling is due to the IO libraries in question
- Several findings regarding root serialization scaling resulted in improvements and fixes in ROOT and were directly applied to CMSSW
  - More in the CMS experiment talk

Some observations and takeaways

- Serial HDF5 writing performance comparable with ROOT
  - A much simpler format PDS performed the best for the BLOB storage of intermediate data
- Parallel HDF5 implementation shows performance scaling with the number of nodes
  - Overhead of using MPI as compared with serial HDF5 on a single node -- if running on a single node use serial mode
    - Can make use of expert guidance on current implementation
  - Multiple MPI ranks per node (4 per Haswell node on Cori) performs the best
  - Studies include both Lustre file system and burst buffers
  - Darshan logs show a mix of different sized writes (~40% 4-10MB, and rest smaller) - still need to be investigated.

From serial HDF5 to PHDF5



Some observations and takeaways

- CMSSW uses multithreading and there is no apparent use of multiprocess supported parallel IO
  - Primitive multithreaded IO capability in HDF5 was never explored
- Athena can use multiprocessing or multithreading and did implement a prototype for serial HDF5 writing using the CCE work
- DUNE uses HDF5 (in their DAQ system) and have no “immediate” use of parallel IO

Status of the report

- In progress, goal is to complete by CHEP 2023.
- The current draft is on overleaf, and includes
  - Details about the test framework
  - Technical details for serial and parallel HDF5 design
  - Analysis and plots for serial output modes comparison
  - Analysis and plots for parallel output mode (scaling)

More on future directions

- There are several HDF5-related technical tasks that we can carry out such as
  - Potential work on more direct HDF5 mapping for subset of experiments data (e.g. nano-AOD, PHYSLITE, DUNE DAQ) that have data models with limited complexity.
  - Investigate HDF5's capability for offloading data to GPU.
  - Study HDF5 as data format for AI/ML.
  - Apply lessons learned with HDF5 to other I/O backends (e.g. Parquet).
- Looking forward to learn from experiments on their IO needs and interests

Parallel HDF5 approach - recap

- N number of MPI ranks participate in the reading of file(s) and write to a single HDF5 file collectively (i.e. parallel write across a single dataset).
  - Writing a file collectively has the advantage that the final file might not need merging.
- The key feature in the parallel design is event distribution approach among MPI ranks and the use of MPI functions to exchange relevant information such as data sizes and offsets to coordinate for collective IO.
  - Store events as blobs in a single HDF5 dataset, and aggregate events before writing