Introduction: The HEP-CCE IOS project

Peter van Gemmeren (ANL)
Saba Sehrish (FNAL)
for the HEP-CCE IOS team

CCE/IOS: Goals

IOS will focus and concentrate effort on:

- **Parallel serialization/de-serialization** of HEP data models
  - both single node and multi-node access patterns
- **Persistable data representations** tuned for HPC storage systems.
  - Connection to PPS exploration of portable parallelization libraries
  - can benefit from Write-Once/Read-Many HEP access models
- **Accessing partial, partitioned or sub-event data blocks**
  - matched to specific algorithm consumption requirement
- **Runtime memory mapping of data**
  - exploit batched, vectorized, and data parallel operations and transforms on columnar data.
  - taking into account CPU-XPU communication

IOS Main Activities

Understanding and achieving efficient and high performance IO when running on HPCs

- IO profiling studies with Darshan
- Root serialization scaling studies
- Alternate IO formats and data models
  - In particular investigating HDF5 as intermediate storage
  - Root serialization scaling studies
- Memory-friendly data (in collaboration with the PPS group)
  - Still starting out

Cast of Characters (snapshot)

- Amit Bashyal (ANL)
- Doug Benjamin (BNL)
- Jakob Blomer (CERN)
- Suren Byna (LBNL)
- Philippe Canal (FNAL)
- Matthieu Dorier (ANL)
- Chris Jones (FNAL)
- Kenneth Herner (FNAL),
- Patrick Gartung (FNAL).
- Rob Latham (ANL)
- **Rob Ross (ANL)**
- Qiao Kang (LBNL)
- Liz Sexton-Kennedy (FNAL)
- Kyle Knoepfel (FNAL)
- Saba Sehrish (FNAL)
- Shane Snyder (ANL)
- **Peter van Gemmeren (ANL)**
- Torre Wenaus (BNL)

IO profiling studies with Darshan

(Material provided by Rob, Shane, Patrick and Ken)

Darshan

From the Darshan website

- Darshan is a scalable HPC I/O characterization tool. Darshan is designed to capture an accurate picture of application I/O behavior, including properties such as patterns of access within files, with minimum overhead.
- Darshan can be used to investigate and tune the I/O behavior of complex HPC applications. In addition, Darshan's lightweight design makes it suitable for full time deployment for workload characterization of large systems.
- Primarily used to characterize the I/O of MPI applications e.g. identify I/O bottlenecks in multiple processes on multiple nodes writing to shared files
- Records all POSIX and STDIO for each process in a darshan file
- Provides tools for generating pdf files with info for each darshan file

Using Darshan

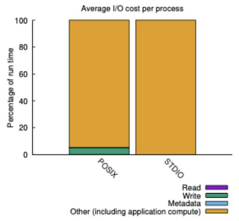
- For non-MPI HEP applications, Darshan is built as dynamic library that is loaded with LD_PRELOAD
- Initial work
    - Integration of Darshan into workflows
    - Darshan logs generated for ATLAS, CMS, DUNE example workflows
    - Identified small accesses; problematic for HPC parallel file systems
- Recent enhancements
    - Make Darshan library fork-safe
    - Add Darshan runtime configuration to control memory usage, files to instrument/ignore, etc

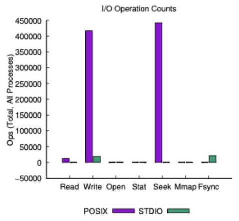
Darshan log example of CMS Reconstruction

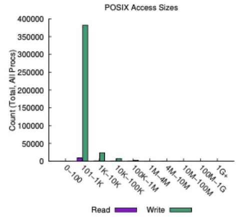
cmsRun (1/28/2021) 1 of 3

jobid: 3107	uid: 72001	nprocs: 1	runtime: 1019 seconds

I/O performance *estimate* (at the POSIX layer): transferred 6966.6 MiB at 128.87 MiB/s
I/O performance *estimate* (at the STDIO layer): transferred 0.2 MiB at 4.39 MiB/s







Most Common Access Sizes
(POSIX or MPI-IO)

	access size	count
POSIX	130	2742
	261	820
	269	599
	286	459

File Count Summary
(estimated by POSIX I/O access offsets)

type	number of files	avg. size	max size
total opened	7	1.3G	6.7G
read-only files	3	2.3G	6.7G
write-only files	2	929M	1.9G
read/write files	0	0	0
created files	2	929M	1.9G

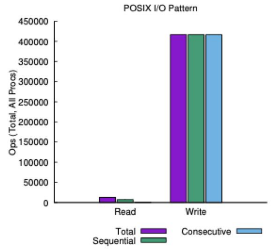
cmsRun step3_RAW2DIGI_L1Reco_RECO_RECOSIM_PU.py

Average I/O per process (POSIX and STDIO)

	Cumulative time spent in I/O functions (seconds)	Amount of I/O (MB)
Independent reads	1.559249	5110.16824245453
Independent writes	51.536927	1856.61143684387
Independent metadata	1.011185	N/A
Shared reads	0	0
Shared writes	0	0
Shared metadata	0	N/A

Data Transfer Per Filesystem (POSIX and STDIO)

File System	Write		Read	
	MiB	Ratio	MiB	Ratio
UNKNOWN	0.21151	0.00011	0.00000	0.00000
/	0.00000	0.00000	0.00003	0.00000
/global/cscratch1	1856.39993	0.99989	5110.16821	1.00000



cmsRun step3_RAW2DIGI_L1Reco_RECO_RECOSIM_PU.py

What is next?

- Characterization and (ideally) tuning of ROOT I/O workloads for use on HPC storage
  - Collaboration with ROOT and experiment experts
  - Realistic experiment configurations
  - Identification of I/O bottlenecks worth investigating on relevant HPC storage technologies
  - Implementing changes to workflows and/or ROOT
- Workflow-aware reports using a set of Darshan logs
  - DAGs of workflow phases and accessed files
  - File and file system access characteristics of different workflow phases
- More trace-based visualizations
  - Write/read access patterns for files produced/consumed by different workflow phases

Test Framework for data formats comparison and root serialization scaling studies

(by Chris Jones)

Storage Format Comparisons

Test storage formats for use on HPC

Compare

- Read & write performance scaling with number of available cores
- Memory usage scaling
- File sizes
- Large scale usage impact on HPC sites

Testing Framework

Developed a simplified HEP multi-threaded framework

- based implementation on CMS's framework

High levels of concurrency supported

- Concurrent read/process/write of events as a whole
- Concurrent read/write of individual data products within an event
- Serial processes are scheduled by framework and never block a thread
  - e.g. if multiple events want to write to the file system their requests are queued and only one happens a time while the other thread can be used to do other work

Testing Framework (2)

Supports any number of storage formats

- Just implement the necessary base class interfaces
- Can read from any supported format and write to any supported format

Can read ATLAS, CMS and DUNE experiment ROOT files

- allows testing using real world data

Write Performance testing

Used CMS’s smallest data format

Minimized time spent in reading

- Read first 100 events and then cache them to memory
- Replayed cached events over and over

Kept number threads == number of concurrent events

- Intra event concurrency helps when serialization happens

Kept all cores of machine busy

- # jobs = (# cores on machine) /(# threads in job)

Write Performance Testing Formats

Standard HEP ROOT format

- Events stored as an element in a TTree
- Each Event Data Product in its own TBranch of the TTree

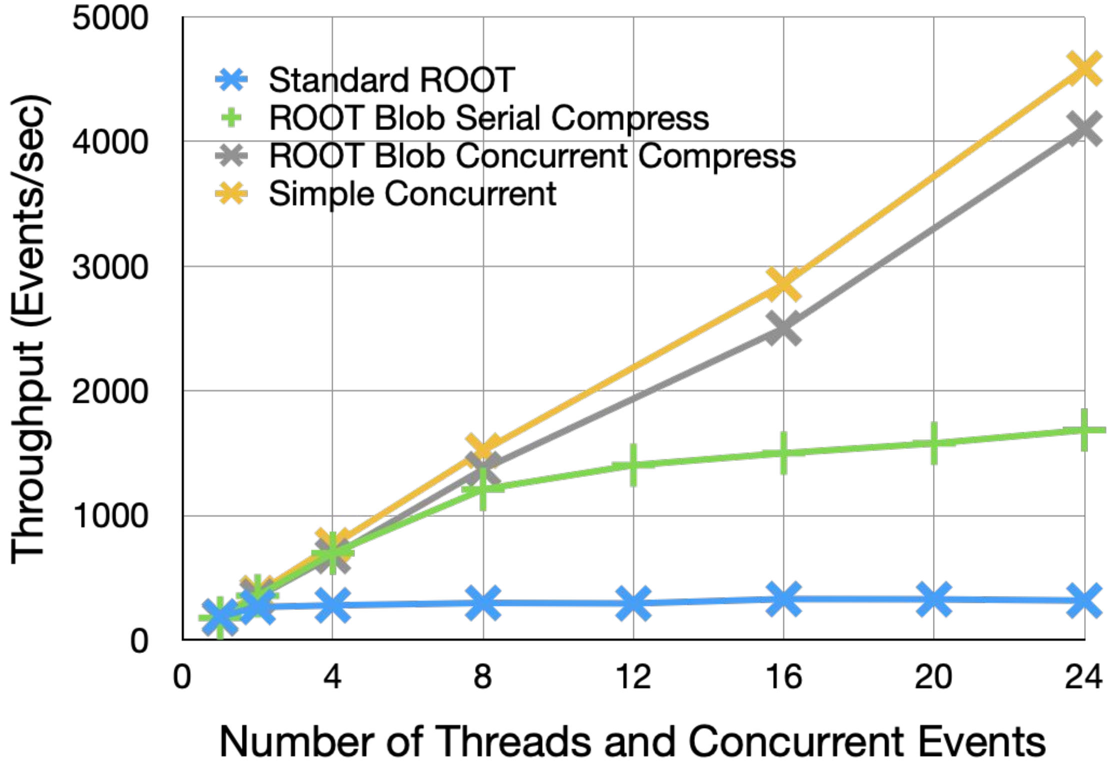
Simple per Event storage format

- Designed to support concurrency

ROOT with one blob per Event

- Explicitly do object serialization concurrent
- Store all Data Products for an Event into one blob
- Compress either external (concurrently) or internal (serially) to ROOT

Preliminary LZ4 Compression Write Results



HDF5 and Parallel IO investigations

(contributed by Amit, Qiao, Saba)

HDF5

- HDF5 (Hierarchical Data Format) is a portable, self-describing file format designed to store large amounts of data
  - It is maintained by the HDF Group [https://www.hdfgroup.org]
  - It is widely available at HPC centers, and easily installable on laptops
  - It supports parallel IO using MPI, and has special drivers tuned for parallel file systems at HPC centers
- A few key abstractions are:
  - datasets, which are multidimensional arrays of homogeneous types,
  - groups, which are containers of datasets and other groups, and
  - attributes, which are small metadata objects to describe groups and datasets
- Allows efficient columnar data access for the “required” data products

Design assumptions

- The input data (intermediate output) is already serialized with ROOT.
  - Complex objects but presented as byte stream to HDF5
- We are not working “directly” with C++ classes that represent data products
- We are not working with analysis-ready data either
- We are interested in designing an experiment-independent approach, so there is no experiment specific assumptions in the implementation.
  - Will need experiment-specific set up to run it to make sure ROOT dictionaries are available

Design details HDFOutputer

- Using 1D dataset of chars for each data product, and a corresponding 1D dataset for size/offset per event.
- Write happens in batches; a batch corresponds to number of events that are aggregated before a write happens
- There is one dataset create call per data product, and n write calls covering all the data products, where n is total number of events divided by batch size.
- A call to resize data set happens upon every write request
- **Issues:** Two data sets per data product generates a lot of metadata especially when there are thousands of data products in an event

Improvements and alternate layouts for HDFOutputer

- Reduce the number of I/O calls on meta-data related data-sets.
  - Collect the metadata for all datasets for each event and save once.
  - Once tested more thoroughly, port to the HDFOutputer.
- Implemented another HDF5 outputer by storing events as a blob in a single event dataset
  - Number of datasets is reduced to 3
  - Tuning and performance evaluation underway
- Both implementations are general and rather straightforward to adopt for incorporating parallel design

Design details HDFSource

- Read in the HDF5 data; only supports data product based design approach
- Read one event at a time
    - Locate start index of a data product and calculate end index using EventID dataset and offsets datasets
- Able to read events randomly since we can index into datasets as needed
- Next steps:
    - Performance evaluations
    - HDFSource for reading event blob format

Running tests on HPC

- Non-trivial to set up and run any experiment code on HPC machines
- Haven't even tried running with ATLAS or DUNE data files on Cori
    - But have locally used CMS, ATLAS and DUNE data during development
    - ATLAS is using this HDF5 design with their framework (but not in production)
- We can run the test framework with CMS reco files, and are evaluating performance of currently available IO modes

Adding MPI support to Root serialization

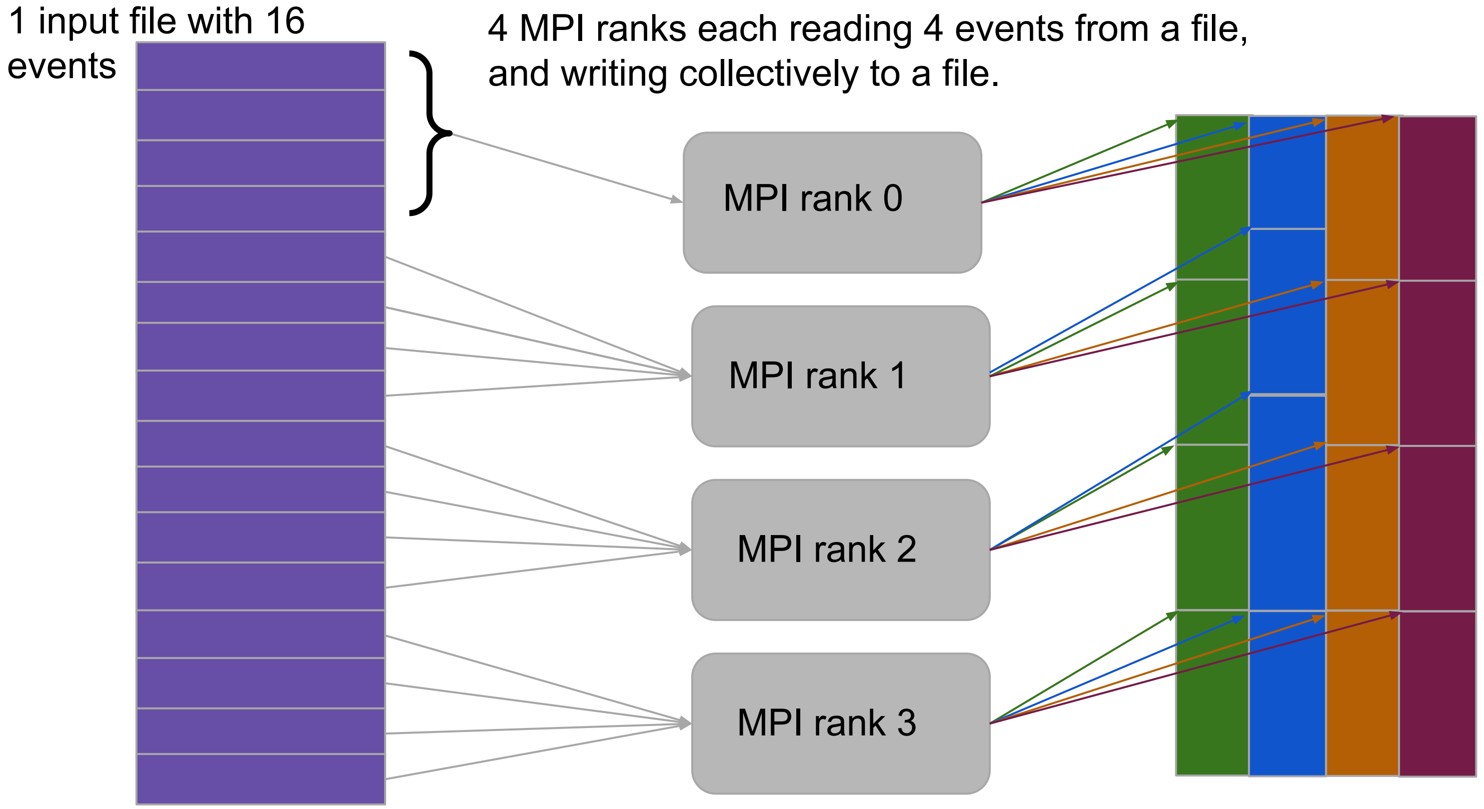
- Created an MPI-based version of root serialization application
- Goal is to be able to evaluate multi-node performance as well as pave the way for parallel IO using HDF5
- Current supported modes:
  a. N MPI ranks able to read N input files and write N output files, trivial, running multiple processes of root serialization all through MPI (any input/output mode combination is supported)
  b. N MPI ranks reading 1 input file and write N output files (any input/output mode combination should work logically)
  c. N MPI ranks reading N files and writing 1 output file (will only be supported for HDF5) → Not done yet

Parallel HDF5 approach

- N number of MPI ranks participate in the reading of file(s) and writing into 1 single output collectively.
- Writing a file collectively has the advantage that final file might not need merging or less resource dedicated to merge the output files.
- Since the multiple ranks are writing on the same output file, need to figure out how to assign the position of the data within a data-set, indexing of events etc.
- Use of MPI Functionalities to communicate with various processes to exchange relevant information for collective IO.
- Use of Parallel-HDF5 to do the reading and writing by various MPI nodes collectively.
  - Test improvements from serial HDF5 design to do collective I/O on writing into the HDF File.

4 MPI ranks each reading 4 events from a file, and writing collectively to a file.

HEP-CCE



The colorful blocks show one HDF5 file. Each color represents a different dataset corresponding to a data product. There are four data products. Each product per event may have different sizes. What is not shown for simplicity is offsets and Event ID datasets here. All the green arrows represent parallel write to the first dataset, then all the blue arrows represent the second parallel write that happens after the first write.

With event blob approach, this design becomes much simpler.

GPU friendly Data Model

- Work has just begun on this and right now a very rough draft.
- Need of serializer tool to persistify the complex C++ object. Maybe not possible in the GPUs.
- Investigation of the data structures that do not need secondary tools to persistify.

Summary

- Darshan has been a useful tool to guide us about HEP workflows IO patterns and usage on HPC. This work has resulted in enhancements to Darshan as well.
- Root serialization scaling studies have uncovered opportunities to improvements in ROOT and therefore impacting experiment frameworks using them.
- The test framework provides us with a valuable tool that can be used to do performance comparison among different IO strategies and formats in an experiment-independent code
- HDF5 with parallel IO is currently being explored and evaluated against other IO formats
    - Serial (within framework) and parallel (standalone) versions exist and being evaluated

What is next for HDF5?

- Event aggregation
  - Coalesce I/O requests for the same HDF5 dataset.
  - Reduce the number of H5D calls.
- HDF5 grouping
  - Groups contains all data product
  - One event per group
  - Enable better parallel I/O performance
- Multi-threaded HDF5
  - There is a feature branch available with simple read/write patterns, which we have in our use case, that we should look into
- Explore direct storage access from GPUs

What is next for evaluation and testing?

- Do read performance tests
- Do write performance tests on an HPC node (Cori/Theta)
  - Test at higher thread count
- Test HDF ability to write to one file from multiple processes (Parallel IO capabilities)
- Large scale testing on HPC site
- Use many nodes concurrently running the test framework
- Do fast write/read tests using different formats

What is HEP-CCE?

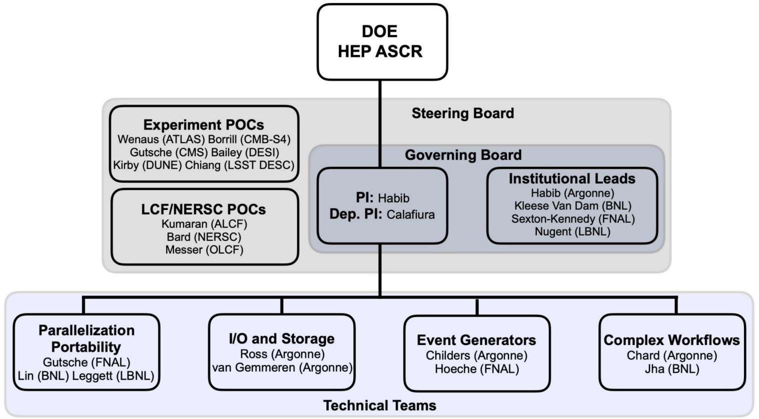
HEP-CCE

Three-year (2020-2023) **pilot** project

four US labs, six experiments, ~12 FTE, ~30 collaborators

https://www.anl.gov/hep-cce

1. Portable parallelization strategies
   - exploit massive concurrency
   - portability requirements
2. Fine-grained I/O and related storage issues
   - new data models (zero-copying, SOA,...)
   - event batching (XPU offloading)
3. Optimizing event generators
4. Running complex workflows on HPCs
   - main use case: cosmology surveys



Open collaboration

https://indico.fnal.gov/category/1053/

U.S. DEPARTMENT OF ENERGY | Office of Science   Argonne NATIONAL LABORATORY   BROOKHAVEN NATIONAL LABORATORY   Fermilab   BERKELEY LAB Bringing Science Solutions to the World