Fermilab U.S. DEPARTMENT OF ENERGY Office of Science

DUNE Data Serialization/Low-Level Processing and Production

Tom Junk

LArSoft Multithreading/Acceleration Workshop

March 3, 2023

Goals of Multithreading/Acceleration

- Improved throughput
- Better utilization of CPU and accelerator resources
- Current OSG nodes have ~2 GB RAM/CPU
  - Our usage is accounted in CPU hours but it's really GB hours since we run 6 GB/job (PDSP reco) mostly single-threaded.
- Our processing is embarrassingly parallel (event level for sure, and parts of events too are independent).
- Goal is to reduce memory usage per CPU. Multithreading addresses part of this.
- Multithreading would let us share "fixed" memory: code, geometry, service caches
- Loading up ROOT libraries and JIT compiler takes ~500 MB all by itself
- Rest of this talk is about per-event memory usage and how to best spread it out.
- If data size dominates, iterating over it is a natural solution. SNB data will have to be iterated over anyway.

Data Sizes -- TPC

A raw FD normal trigger record for one horizontal-drift module has size 6000 ticks * 14 bits * 150 APAs * 2560 channels: **4.0 GB**.

Possibly fewer ticks per trigger record. 4500 ticks covers the drift time at full volts. **3.0 GB**

ProtoDUNE-SP needed a longer readout window to cover the copious out-of-time cosmic rays (not to mention the 1 ms AC coupling time constant of the electronics)

“Protocol overhead” was ~20% for ProtoDUNE-SP. WIB frame headers, extra data to cover the 6000-tick windows that would be trimmed to size offline (feature of the RCE 1024-tick blocks)

Data Sizes: ProtoDUNE-SP TPC

- ProtoDUNE-SP data sizes: 6 APAs x 6000 ticks x 12 bits x 2560 channels = **138 MBytes** (without "protocol overhead")
- Compression achieved: Factor of ~4
  - RCEs performed compression before data sent to artdaq boardreader.
    - Needed for throughput – network was the bottleneck
    - Required significant expert development
    - Some fragility when nticks ended up being nonstandard for some channels
  - FELIX compression performed with Intel Quick-Assist Technology cards (I think.. could have been done also in software).
  - Compression was custom – not ROOT based. Best to turn off ROOT compression if data are already compressed.
- DUNE FD data will *not* be compressed, as far as I know.

Data Size: Far Detector SNB Trigger Records

- One FD-HD SNB Trigger: 100 sec * 2M samples/sec * 14 bits * 150 APAs * 2560 channels = **134 TBytes/module**
- Just under 1 TByte/APA. Try to keep protocol overhead low for this, but WIB frame headers will be needed.
- Compression not planned for DAQ. 100 seconds isn't enough time to compress 134 TBytes on the fly.
- Transfer time to FNAL: 4+ hours (10?). Plenty of time to compress data while it is waiting to stream over the network.
- Normal trigger records come once per 20 seconds. Plenty of time to compress those, too.
- Copying to dCache and enstore, as well as analysis readback will benefit from compression of data
- Lossless compression factors depend on noise and gain. We can always choose to write noisy bits...

Processed Data

- recob::Wire – has ROI features built into it.
- recob::Hit, clusters, tracks, showers, etc – all small
- Reconstruction data products scale with interaction size and not detector size
- Cosmic-ray air showers and high-energy showers will make big tails on these.

Non-TPC Data

- Photon-detector data: Much smaller than the TPC data
  - 10 PDS channels per APA vs 2560 wires.
  - Data are collected in snippets, already zero-suppressed. Single-PE signals are above threshold so there is no point in not zero-supressing them.
  - With the TPC, single electrons induce very small signals and induction-plane signals involve long-range effects and cancellation, so there is no lower bound of useful signals strengths, hence the desire not to zero-suppress the TPC data.
  - PDS snippet data scales with interaction size, not detector size – most of the detector is empty on nearly all events.
- Trigger primitives, CRT (ProtoDUNE has one), beam instrumentation are all small data loads

Monte Carlo Specific Data Products

- ProtoDUNE-SP had a lot of space used by
  - MCParticles – contain trajectory information – optimizable
    - EM shower daughters
    - Trajectory density
  - Sim::EnergyDeposits
  - Sim::SimChannels
- All of these scale with interaction size and not detector size
  - FD MC should mostly be much smaller than ProtoDUNE-SP for these data products
  - Some tiny number of interesting cosmic-ray air showers or single high-energy cosmic rays showering in the detector can create a long tail on the size here.

Near Detector Annual Volumes and Spill Sizes

Table 7.6: Annual DUNE near detector data volume estimates. No compression is assumed.

Type	Volume/year
ND-LAr	
In-spill data	144 TB
Out-of-spill cosmics	16 TB
Calibration	16 TB
Total	176 TB
ND-GAr	
In-spill data	52 TB
Out-of-spill cosmics	10 TB
Calibration	6 TB
Total	68 TB
System for on-Axis Neutrino Detection (SAND)	
In-spill data	40 TB
Out-of-spill cosmics	8 TB
Calibration	1 TB
Total	49 TB
Total ND	**293 TB**

Volume/spill

9.6 MB

3.4 MB

2.6 MB

~300 sec CPU/spill

TMS: Small data size

Data/spill assumes 1.5 x 10^7 spills/year

Fermilab

Natural Granularity

- TPC data naturally divide into APAs.
- Usually, it doesn't make sense to divide data any finer (FEMBs? Not really).
- SNB data also need to be divided by time. ~200 μs extra on the ends is required for deconvolution.

- "All other data" – PDS, trigger etc can be handled together unless it is a problem. SNB PDS data can be divided in time but not as convenient for per-APA.

- SNB low-level processing job – does it need both TPC and PDS data? Hopefully we can factorize that. Divide jobs up by APA may need duplication of PDS data then...

Use Cases: Online Monitoring and DQM

- Prefer these applications to be lightweight –
  - start up quickly at the beginning of a run
  - low latency is a big plus
- Not all data need to be processed.
- ProtoDUNE-SP online monitor was custom software, DQM was running LArSoft. Some code was shared but not much (dune_raw_data)
- DQM can have longer latency than Online Monitor

Use Case: Raw Digit Event Display

- Critical for understanding detector/electronics features
- Need ability to plot waveforms and 2D tick vs wire data
- Also useful for hand-scanning events.
- Deconvolved event display can mask issues with the data
- Random access to any APA's worth of data is required – need large-RAM machines to host the EVD?
- Random-access EVD does not have to run on a batch worker, though. Interactive vs. web server?
  running the raw EVD on the same computer as the display was much more comfortable than over a network.
- Online monitor/control room EVD needs this.
- Batch-mode EVDs can be made serially (dataprep makes these now).
- We do not need it for every trigger record, but we need it for any trigger record. Need it also for a sample of MC.

Use Case: Front-End Data Processing

- Unpacking
- Channel Map
- Trigger primitive/activity/candidate recalculation
- ADC mitigation
- Correlated Noise Removal
  - Currently by FEMB and plane, but some noise sources don't respect these boundaries
- Frequency-based Noise Filtering, which is part of:
- 2D Deconvolution
- recob::Wire, recob::Hit, 3D charge image outputs
- Optional plots/visualization of output of each step

Front-End Data Processing

- Work already done to reduce memory consumption through dataprep
  - Delayed reader – both for ROOT and HDF5 data. Don't read the whole TPC data in for a trigger record, just one APA at a time
  - Do not put raw::RawDigits into the *art* event – keep them in temporary storage and delete them when done.
  - *art* has a RemoveCachedProduct method but it throws an exception if you try to remove a produced product. Just products from files can be removed. raw::RawDigits are produced because raw data are not in raw::RawDigit format but in DAQ formats.
  - Trigger record consistency check – do all FEMBs read out the same number of ticks? Requires some inter-thread communication.

- Memory bottleneck – transferring recob::Wire to WireCellToolkit from DataPrep. Mentioned by Brett and Haiwang.

ProtoDUNE-SP Data Reco Processing Times

The Biggies:

EmTrackMichelID
(a CNN inference step,
this run was CPU-based)

DataPrep (incl. file i/o
and unpacking/
decompression)

WireCellToolkit

DataPrep and WireCellToolkit
scale with detector size,
not interaction complexity
(x25 for a FD module)
15 sec of DataPrep was
disk I/O, decompression, reformatting

Module Label	time/event (sec)
RootInput(read)	0.14
beamevent:BeamEvent	1.6
caldata:DataPrepByApaModule	83.0
wclsdatasp:WireCellToolkit	79.9
gaushit:GausHitFinder	1.6
reco3d:SpacePointSolver	9.4
hitpdune:DisambigFromSpacePoints	1.5
pandora:StandardPandora	39.9
pandoraTrack:LArPandoraTrackCreation	4.6
pandoraShower:LArPandoraShowerCreation	3.7
pandoracalo:Calorimetry	2.1
pandoracalonosce:Calorimetry	1.9
pandoraShowercalo:ShowerCalorimetry	3.3
pandoraShowercalonosce:ShowerCalorimetry	3.2
emtrkmichelid:EmTrackMichelId	233.8
canodepiercerst0:T0RecoAnodePiercers	1.1
pandora2Track:LArPandoraTrackCreation	11.6
pandora2calo:Calorimetry	4.9
pandora2calonosce:Calorimetry	4.5
pandora2Shower:LArPandoraShowerCreation	4.2
pandora2Showercalo:ShowerCalorimetry	4.2
pandora2Showercalonosce:ShowerCalorimetry	3.8
RootOutput(write)	2.8
Total:	**507.9**

DUNE Software and Computing CDR

Use Case: Very long readout window & Calib data

- In ProtoDUNE-SP and ICEBERG, some data were taken with very long readout windows.
- 30-second ICEBERG data. 1280 channels @ 2 MHz. No event builder, however, so files were divided up by electronics boundaries and we had to read several in at a time.
- ProtoDUNE-SP data – just two FEMBs read out by FELIX for of order 1 second.
- Purpose was to study very low-frequency noise.
- Bespoke workflows needed for these. Not going to run regular reco on them but that's not the point.
- ICEBERG data may be practice for SNB readout. 128 GB of data in 30 seconds

Use Cases: Monte Carlo

- Detsim Stage1 takes 1169 seconds per ProtoDUNE-SP event running WireCell's raw digit simulation – a lot of random numbers thrown for noise simulation. Good opportunity for multithreading/acceleration!

- DUNE does not need raw digits persisted for most FD MC
- Strategies:
  - Drop the raw digits as early as possible
  - Use smaller geometries
  - Zero suppress the MC

  BUT:
- Firmware developers have expressed interest in DAQ-formatted MC – full geometry, no ZS.
- Would like "eager writing" – flush raw::RawDigits to output file and free up memory
- This needs to be incorporated into the WireCellToolkit's output methods.

Existing Tools – MC DAQ-format writers

- Standard *art* analyzer modules – read raw::RawDigits from the event memory, write HDF5 files
- Uses pre-Nov. 2022 HDF5 file format (and data format)
- fdhddaqwriteexample_nozs.fcl runs a simulation with a particle gun, drops raw::RawDigits from the artROOT file (so as not to hit the 1 GB limit), and writes an HDF5 file with simulated WIB frames.
- hdcolbox_hdf5daqwrite.fcl – the same, but for the HD coldbox

- Need channel maps to look up hardware locations (crate, slot, fiber, chan) from offline channel numbers. PD-HD map evolved a bit as firmware was developed, and FD-HD map is a bit of a guess at the moment. But there is one!

ROOT's 1 GB limit

- Attempting to write more than 1 GB to a TBranch on one entry results in ROOT throwing an exception.
- I am told this limit can be adjusted at compile time
  - We would have to enforce usage of special DUNE-compiled versions of ROOT in order to process raw digits in this way
- There does not appear to be a limit on TTree entry sizes, just TBranch entry sizes.

Memory Consumption Writing 4.5 GB to a Single TTree Entry

- Toy data: five 900 MB blocks of random bits (designed not to compress well)
- Write each block as a vector of ULong64_t's in a TBranch and call TTree::Fill() to write the output file. Usual way to run ROOT.
- Default ROOT settings: high-water mark of memory usage: 14 GB
  - 4.5 GB for user-space memory
  - ROOT output buffer
  - ROOT compression
- Use TBranch::Fill() instead of TTree::Fill() and de-allocate user-space memory after it is done ("eager writing"): 9 GB memory usage
- Turn off ROOT's compression: 4.4 GB memory usage (similar to output file size).

Breaking the Relationship Between ROOT TTree Entry number and Run:Subrun:Event ID

- *Art* assumes a relationship between these
- We can write our own input source that forms *art* events
- ND-LAr's Module-0 data already does not have a concept of a trigger – just a big conveyer belt of ata.
- Module-0 input source and Python workflows have to define what an "event" is.

- DUNE-DAQ has added a "sequence ID"
  - always zero so far
  - Intended for SNB data where one trigger has one Run::Subrun::Event and many sequence IDs.

FD-VD Data and CPU Needs from Computing CDR

Table 7.4: Useful quantities for computing estimates for VD readout based on the DAQ requirements document of January 2022. CPU times are scaled from ProtoDUNE-SP assuming all detectors are used in hit finding but interactions are confined to a subsection of the detector not much larger than ProtoDUNE-SP.

Quantity	Value	Explanation
Far Detector Vertical Drift		
CRPs per module	160	DAQ spec.
TPC channels	491,520	DAQ spec.
TPC channel count per CRP	3,072	DAQ spec.
TPC ADC sampling time	512 ns	DAQ spec.
TPC ADC dynamic range	14 bits	DAQ spec.
VD module trigger record window	4.25 ms	DAQ spec.
Extended FD module trigger record window	100 s	DAQ spec.
Size of uncompressed trigger record	8 GB	DAQ spec.
Size of uncompressed extended trigger record	180 TB	DAQ spec.
Compression factor	TBD	
Beam rep. rate	0.83 Hz	Untriggered
Hit finding CPU time	6,000 sec	from MC/ProtoDUNE
Pattern recognition CPU time pre event	1,500 sec	from MC/ProtoDUNE
Simulation CPU time per event	2,700 sec	from MC/ProtoDUNE
Memory footprint/CRP	0.5-1GB	ProtoDUNE experience

"Hit finding" Includes signal processing

Fermilab

HDF5 vs. ROOT

	ROOT	HDF5
Automatic Serialization of C++ class data	Yes	No (flat arrays). Custom structs possible but you have do do it yourself.
Support for schema evolution of data products	Yes	No
Hierarchical grouping of data	Yes	Yes
Tools for adding Metadata	Yes	Yes
No-Code dumpable	Yes	Yes
No-Code browsable	Yes	Yes
No-Code statistical analysis	Yes (TBrowser, TTree:Draw())	No (There are browsers, but they assume data formats)
Improperly closed files readable	No	No (?)

HDF5 vs. ROOT

	ROOT	HDF5
Streamable via XRootD	Yes/Static* Interface	Yes/Dynamic* Interface
Automatic Compression	Yes	Yes
Multiple buffering of data on input/output	Rather heavyweight	Seems to be more lightweight
Jagged Arrays	Yes	Yes, but some functionality doesn't work for them
Data Selection Tools	Yes, by entry, scriptable	Yes, by dataset/shaped clipping of datasets
Community Support	HEP. Some use outside of HEP (Finance?)	Scientific community, seems to have large support in image-processing fields

Multithreading and HDF5

- As far as I can tell, compiling HDF5 as thread-safe adds a global mutex on methods that access common data structures and I/O
- Often, we are limited by disk I/O anyway, so this is not an additional burden on grid nodes.
- New HDF5 features – subfile for example, may allow this not to be a bottleneck on striped RAID systems.
- We use ROOT I/O in a single-theaded way – accumulate, compress and flush. Flushing TBranches works but I do not know if it bottlenecks if you do a lot of them in parallel, even if the disks can handle it.

Streaming Data with XRootD

- ROOT natively supports XRootD I/O automatically
  - detects XRootD URLs and sends I/O operations to XRootD static interface methods
- HDF5 does not natively support XRootD I/O, but the XRootD Posix library has a dynamic interface
  - LD_PRELOAD can be set to point to the XRootD Posix I/O library which intercepts fopen, fread, fwrite, etc.
  - Shown to work with *art* jobs and h5dump-shared. Streaming works!

Embedding HDF5 files in ROOT files and vice versa

- If we want to write out an HDF5 file and a ROOT file from a MC job, we want to keep these together
- I have a ROOT macro that adds a TTree with one branch with one leaf with one byte that can ingest any arbitrary file one byte at a time and store it in the ROOT file.
- And a macro that reconstitutes all such packed files.
- Not hard to write something similar that adds a big blob of data in an HDF5 dataset.

- Somewhat inconvenient – requires a pre-job program to run to split the input file apart.
- If you want to add raw digits to a ROOTfile from MC, no real benefit to using HDF5.

New DUNE Memory Usage Task Force

- Andrew Norman
- Barnali Chowdhury
- Jake Calcutt
- Paul Laycock
- Thomas Junk

Some To-Do Items

- Eager-Writer for WireCell to produce DAQ-formatted HDF5 output from WireCell MC
- Haiwang showed an example tarfile output from WireCell at the January DUNE Collab meeting
- Break the wall between dataprep and WireCell so that it can iterate over APAs. Or run them in parallel.
- Write an ICEBERG DAQ writer – mostly just need a new channel map as the WIB firmware has changed.
- Read in Trigger primitives, Trigger Activities and Trigger Candidate data from the HDF5 files.

- Simulate a SNB readout! Stretch our capabilities. Maybe just 0.5 seconds worth however, and one FD module.

A Couple of HDF5 To-Do Items

- Write a Virtual File Layer (VFL) plug-in for HDF5 that uses the static XRootD interface like ROOT does. Or a Virtual Object Connector (VOL). There is an example POSIX VFL plugin in the HDF5 library one can use as an example.

- Investigate HPC optimizations for HDF5
  - I'm guessing that if each job reads a separate file there is no advantage over ROOT or other formats, but if many reads overlap in time, we may need to look at how the data are laid out and read in.

Summary

- Work has been invested in streaming data into jobs in small pieces (smaller than a trigger record), necessary to keep the memory usage down.
- Delayed readers are important. Allows longitudinal parallelism in an *art* job but not using event store
- Not all steps of the DUNE workflow have been divided into small pieces yet.
- Chunks can be processed serially or in parallel
- Data streaming at the file I/O level (XRootD) is necessary and we will work to keep supporting/using it.
- Evolution of our data products, DAQ, the computing environment and multithreading/acceleration means that ongoing attention needs to be paid to data serialization and readin.