CMSSW Framework Status and Roadmap

Matti Kortelainen

CSAID Roadmap meeting

9 March 2023

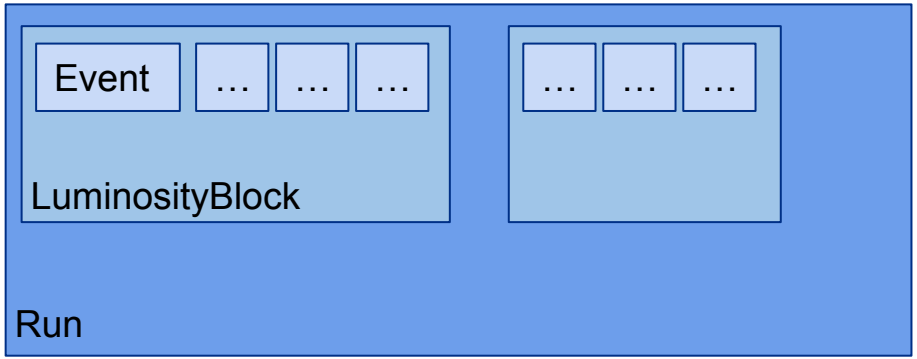
Current team

- David Dagenhart (FNAL)
- Patrick Gartung (FNAL; code profiling, build support)
- Chris Jones (FNAL)
- Matti Kortelainen (FNAL)
- Dan Riley (Cornell)

Two subsystems for data

Event system for collision data

- Hierarchy
  - Event: smallest atomic unit of data
  - LuminosityBlock
    - Collection of Events
    - Corresponds to ~23.3 s of data
    - Smallest atomic unit for calibration Intervals of Validity
  - Run
    - Collection of LuminosityBlocks
    - Corresponds to several hours of data
    - E.g. trigger menu does not change

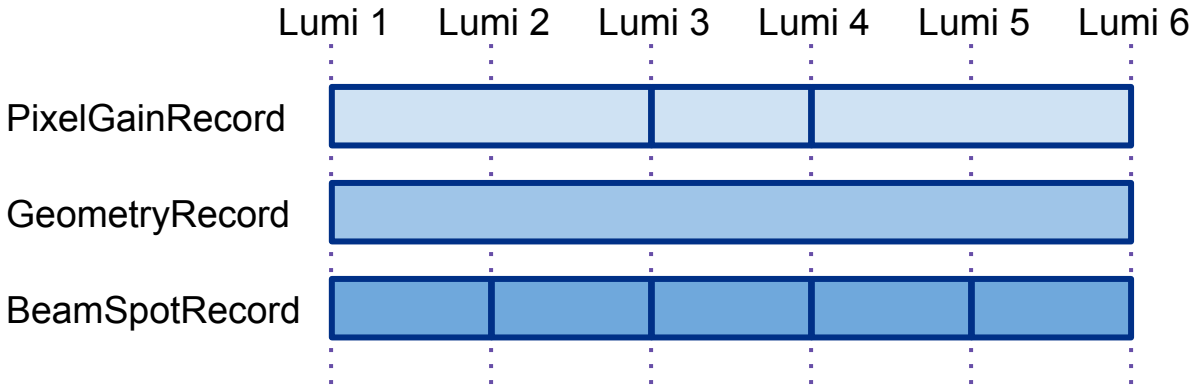


- Orthogonally: ProcessBlock
  - Process-level data storage

Two subsystems for data

EventSetup system for calibration data etc.

- Arbitrary number of Records
  - Each Record has its own Interval of Validity (IOV)
    - From one Run:Lumi to another Run:Lumi
  - Each Record can contain arbitrary number of data products
    - New IOV triggers production of new data products in the Record



Example IOV structure

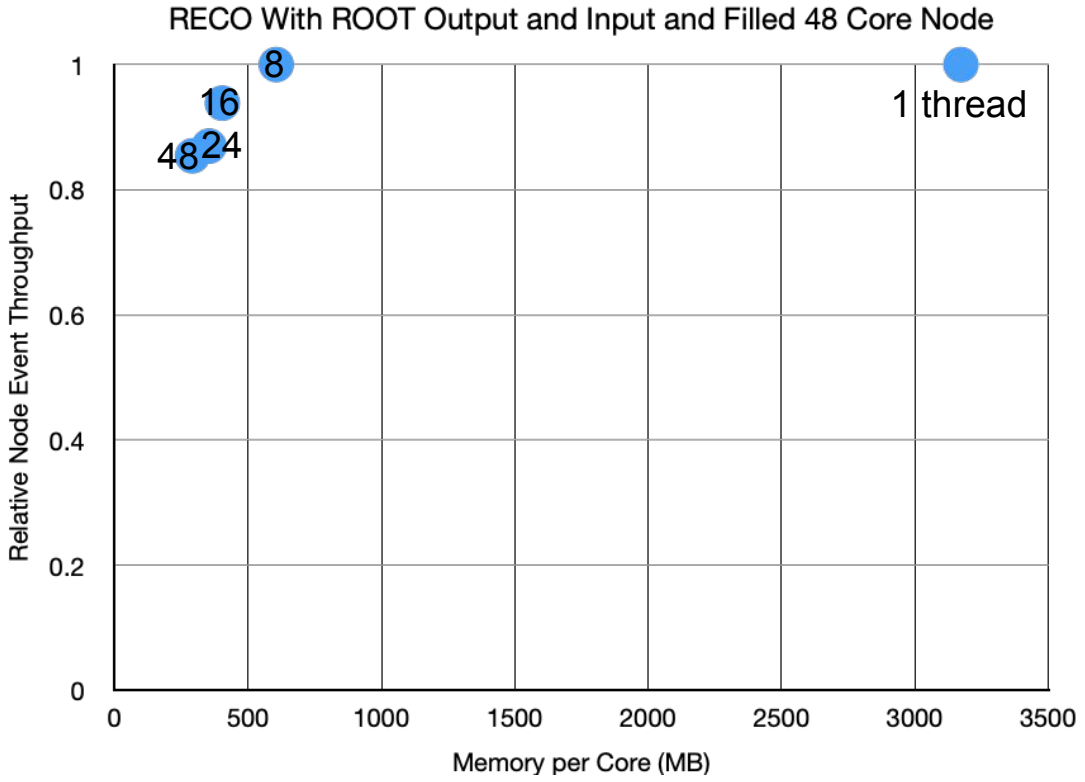
Available concurrency levels in CMSSW

- Multiple Events are processed concurrently (2014)
- Independent modules processing the same Event are run concurrently
  - Auto-scheduled modules run concurrently (2014)
  - All trigger Paths are run concurrently (2017)
  - All modules between two Filters in a Path are run concurrently (2020)
- Process Events from multiple LuminosityBlocks concurrently (2018)
- Process Events from multiple IOVs concurrently (2019)
- Process multiple EventSetup modules concurrently (2020)
- Process Events from multiple Runs concurrently (2022)
  - Not enabled by default yet

- Powered by a task execution engine based on asynchronous execution concepts and implemented with Intel oneTBB

Sneak peek on scalability

- Doing a full-fledged scalability study for CHEP23
  - Latest CMSSW, ROOT 6.26
- Preliminary results on 48-core virtual machine for reconstruction application
  - 8 threads fully efficient
  - 48 threads ~85 % efficient



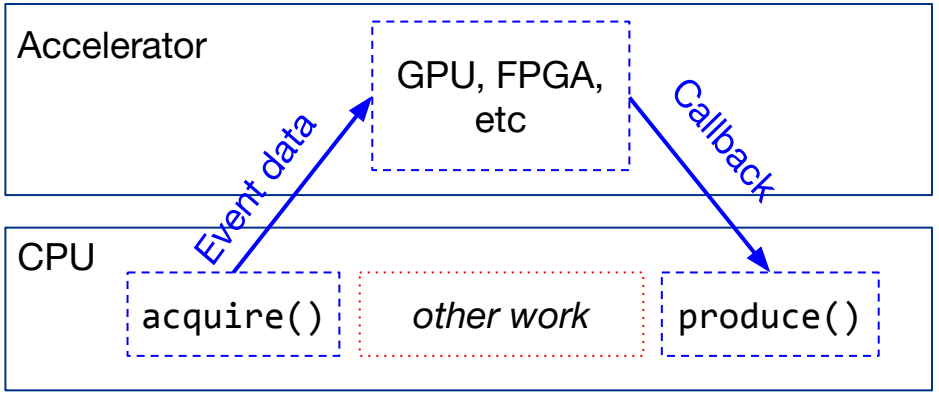
Fermilab

Improving CPU efficiency on high thread count

- I/O is the main bottleneck on high thread count
  - Here “I/O” means mostly ROOT serializing and compressing the data products across many Events into a columnar format
    - Storing data in per-Event blobs would scale ~perfectly (studied in HEP-CCE)
    - Columnar format compresses much better, and want to to minimize the size of archived data
  - Want to work on simplifying I/O for temporary files in grid jobs running CMSSW chains
    - Need some support from Workflow Management, being worked on
  - Also looking in lossy compression
  - On HL-LHC timescale we are working with the ROOT team to make their new RNTuple storage format usable in CMSSW
- Some parts of the applications do not scale well
  - In the recent past the event generators have been the main source of issues
  - Increasing the job thread count in production may reveal new areas that will need attention

Compute accelerators

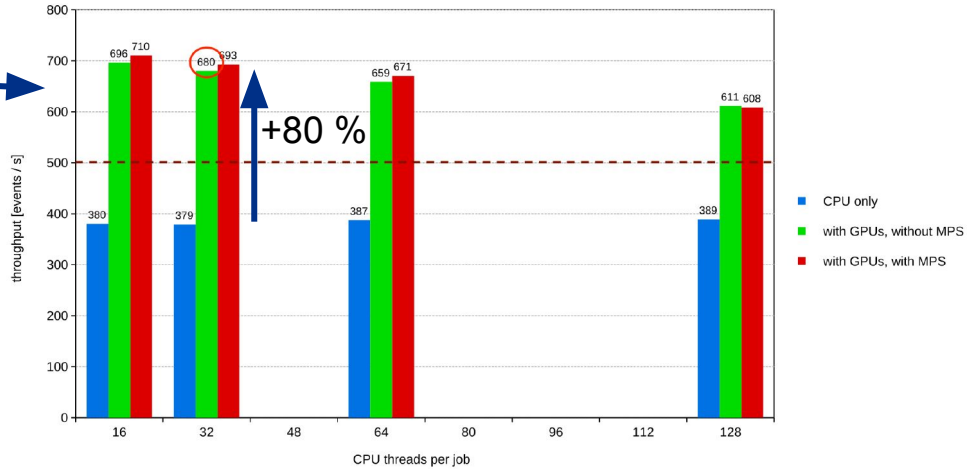
- CMS' general approach has been to
  - Implement general mechanisms, that are agnostic of accelerator specifics, in the framework
  - Implement accelerator specific code as a layer between the framework and user code
  - Allow gradual adoption, keep rest of codebase unchanged
- Generic mechanism for "outside of CMSSW" work called "external worker"
  - Allows CPU thread to do other work
    - Once external work finishes, new task added to TBB
  - Used for
    - Direct GPU usage via CUDA/Alpaka
    - SONIC (ML inference as a service)
    - GeantV integration exercise



CHEP19 doi:10.1051/epjconf/202024505009

Direct accelerator usage with CUDA/Alpaka

- Main use case has been CMS' High Level Trigger
- Some defining characteristics
  - Chains of modules that keep the data in GPU memory, minimizing synchronization
  - Ability to run a configuration on "any hardware" ("portable configuration")
- CUDA support added in 2020, used at HLT in 2022 data taking
- CUDA code being migrated to Alpaka now, to be validated and used at HLT later this year
  - AMD GPU support being worked on
- Module interfaces were improved and simplified for Alpaka
- Expect to retire direct CUDA within a year



ACAT22

2x 64 core / 128 thread AMD Milan
2x NVIDIA Tesla T4

Fermilab

Future work on accelerator support

- Main theme is efficient use of accelerators
- Ability for asynchronous execution in EventSetup modules
  - First use case: copy calibration data from host to device memory asynchronously
- Record information on worker node hardware in the data files
  - Stored module provenance information is not good enough to describe the past behavior anymore
- Want to automate the ability of deleting temporary Event data products after they are no longer needed
  - Expect to be useful especially for low-memory GPUs
- Develop better mechanisms to deal with “memory spaces” of data products
  - Want to be able to handle both discrete memory and unified memory cases
- Want to investigate batching of data from multiple Events to reduce overheads