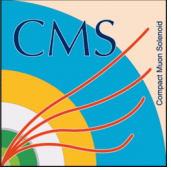


Science and Technology Facilities Council

CMS HLT GPU implementation

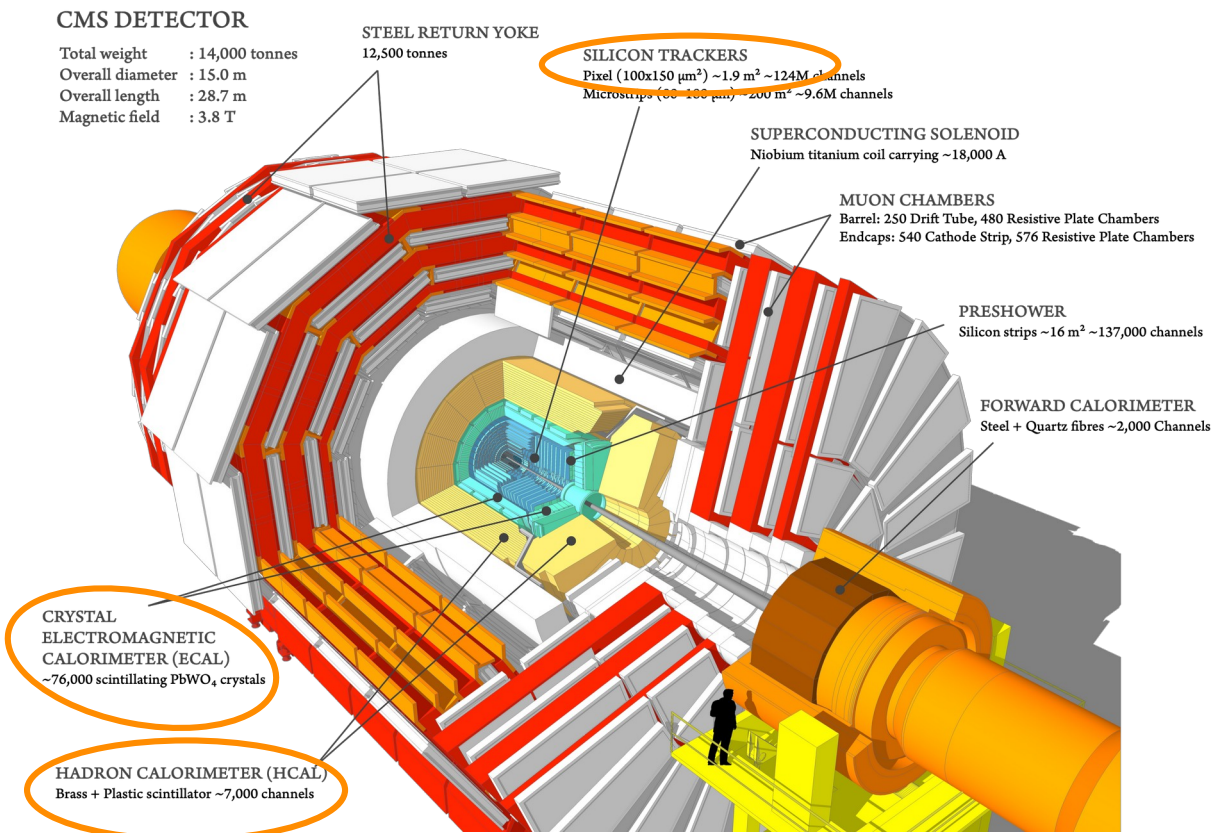
Thomas Reis

Beginner's GPU programming course for UK particle physicists

30th May 2024

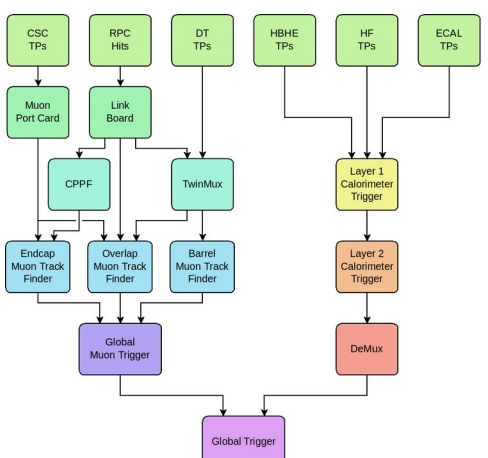
The CMS experiment

- The CMS detector is one of four experiments at the CERN LHC
- Several sub-detectors use GPU code for local reconstruction



Trigger

- CMS uses a two step trigger system to select interesting events for physics analysis
- Level-1 trigger (L1)
  - Uses information from muon system and calorimeters
  - Sees all events
  - Custom hardware boards with large FPGAs
  - Firmware performs the reconstruction and selection depending on physics analysis requirements
  - Trigger decision in 3.8 μs (fixed latency)
  - Reduces event rate from 40 MHz to ~100 kHz

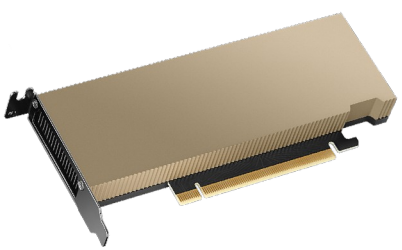


- High Level Trigger (HLT)
  - Uses full detector information read out after L1 accepted an event
  - Large filter farm using GPUs
  - Runs CMSSW framework for reconstruction and filtering
  - ~O(450 ms) processing time per event
  - Reduces L1 event rate to ~ 5 KHz

High Level Trigger

- The High Level Trigger is part of the CMS data acquisition chain
- For L1 accepted events the full detector information is read out from the front end electronics and sent to the HLT filter farm on the surface
- HLT paths reconstruct and select events based on physics analysis requirements
  - All paths together make the HLT “menu”
- Opportunistic use of capacity not required for triggering as T2 grid site

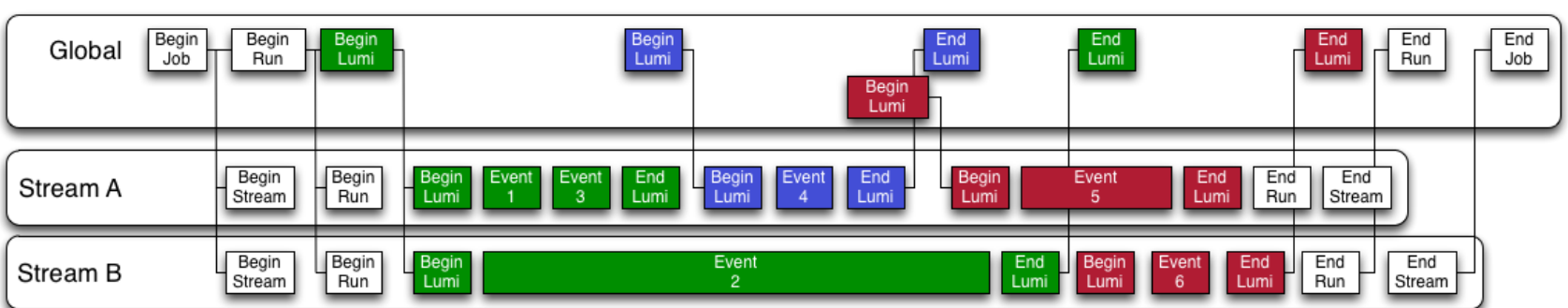
- 2022 configuration:
  - 200 nodes, each with
    - 2 AMD EPYC 7763 “Milan” (64 cores each)
    - 2 NVIDIA Tesla T4 GPUs (2560 CUDA cores, 16 GB memory)
    - 256 GB system memory
  - Total of 25,600 CPU cores and 400 GPUs



- 2024 upgrade:
  - 18 nodes, each with
    - 2 AMD EPYC 9754 “Bergamo” (128 cores each)
    - 3 NVIDIA L4 GPUs (7680 CUDA cores, 24 GB memory)
    - 768 GB system memory
  - Addition of 4608 CPU cores and 54 GPUs (+20%)

CMS software

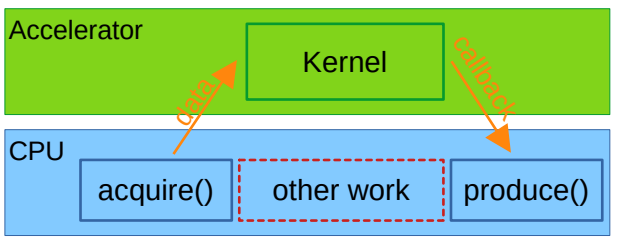
- The CMS software framework (CMSSW) is used at the HLT, for offline reconstruction, and simulation
  - Open source on github with > 1.1k contributors
- Written in C++ with one executable (cmsRun)
  - Plugin modules selected and configured by python scripts
- Multi-threaded processing
  - Concurrent events, processed in streams
  - Concurrent luminosity sections (~23 s chunks of data)
  - Concurrent runs
- Full support for asynchronous work on external accelerators



Transitions seen during event processing

CMSSW modules

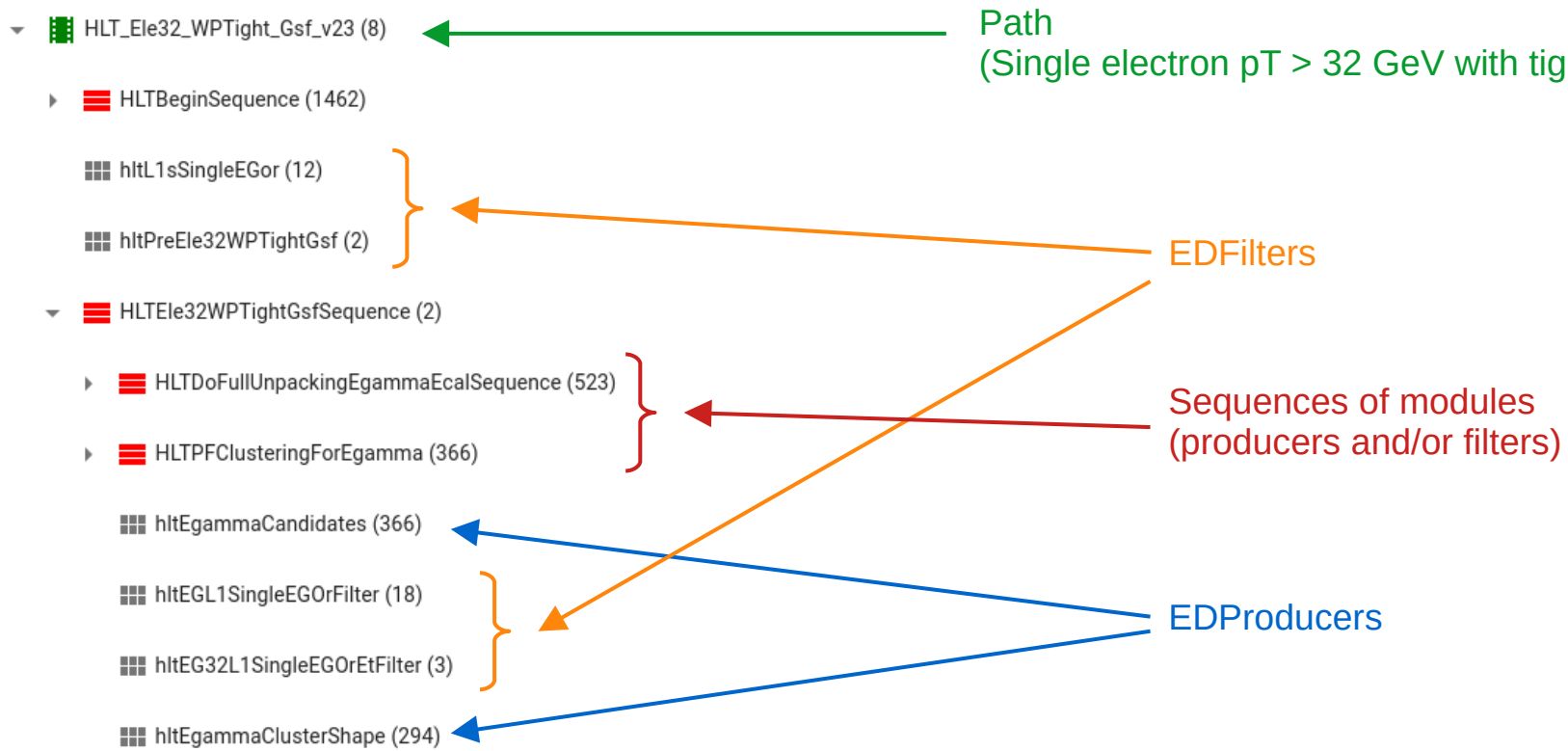
- CMSSW has different module types that are loaded dynamically
- EDProducers
  - Take collections of objects as input and use them to produce derived collections
    - E.g. Take digitised samples and perform an amplitude reconstruction to output amplitude values
  - A special version exists for asynchronous work
    - acquire() function launches the asynchronous work
    - produce() function is called once the asynchronous work has finished



- EDFilters
  - Based on input collections make decisions if an event should be processed further
    - E.g. filter events with at least two muons with pT above a threshold
    - Particularly important for efficient processing of events at the HLT
- EDAnalyzers
  - Analyse event data and e.g. produce histograms
- ESProducers
  - Producing Event Setup conditions like calibrations or configurations

CMSSW at the HLT

- HLT paths perform real time event reconstruction and selection
- One path consists of several EDProducers and EDFilters to gradually reconstruct more complex information from the raw data
- Processing of events not passing a filter is stopped to save resources

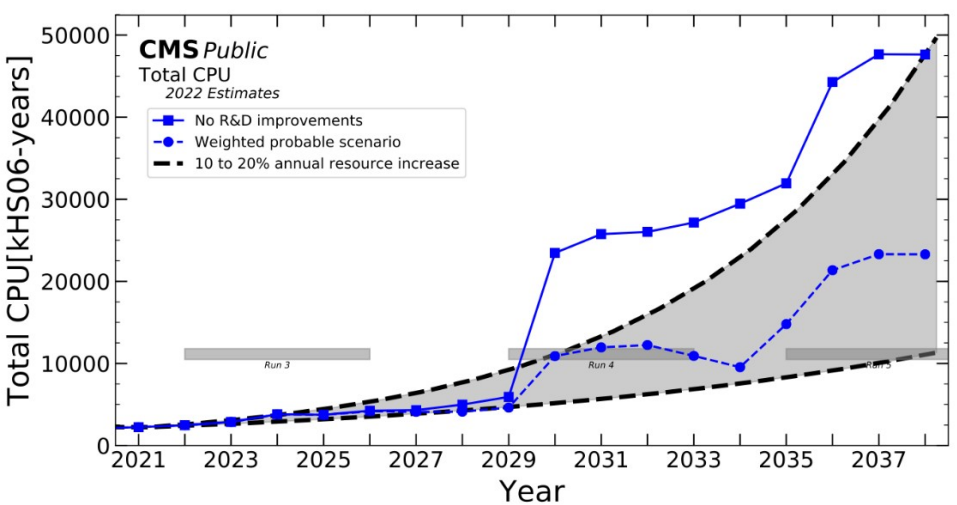


History of GPUs in CMS

- 2016: Start of investigations to offload reconstruction to GPUs
- 2017: CUDA code on pixel local reconstruction
- 2018 – 2020: R&D on algorithms and data structures, memory and caching
  - CUDA algorithms for ECAL and HCAL local reconstruction
  - Automatic offloading when GPUs are available
- 2021: Integration at the HLT
- 2022: Deployment of GPU reconstruction at the HLT
- 2023: Migration to Alpaka
  - Framework support for Alpaka and new SoA format
  - Migration of existing CUDA algorithms
- 2024: Deployment of Alpaka based code for some algorithms

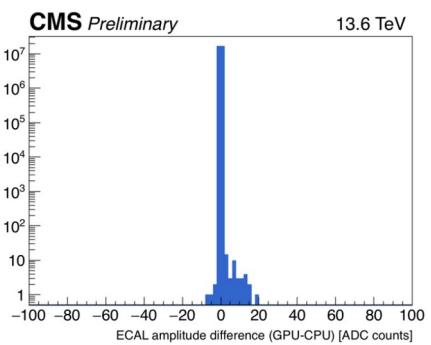
Why are we doing this?

- Because we can
  - Many tasks in event reconstruction are independent of each other
    - E.g. unpacking and reconstruction of subdetector data
    - Independent reconstruction of channels
  - Events are independent and can be processed concurrently
  - Framework takes care of scheduling
- Because we have to
  - High luminosity LHC updates will bring new highly granular detectors and a 2.5 fold increase in instantaneous luminosity
  - This translates to 30x increase in required computing resources for similar physics goals
  - Not achievable with given budget by relying on CPU performance increase alone
  - Parallelism and algorithmic improvements are needed
  - Lower power consumption for same performance



Current setup

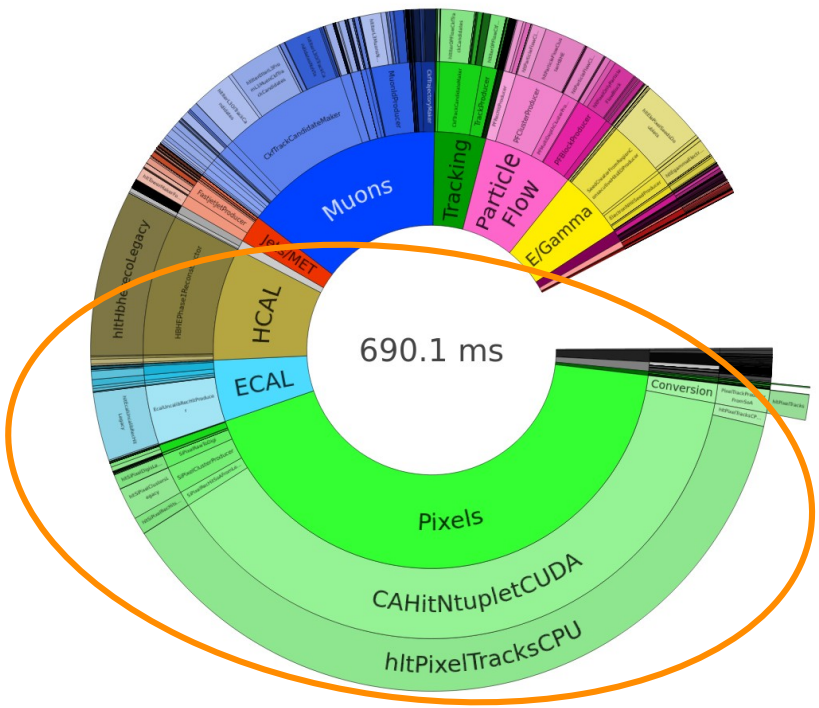
- Offloading to GPUs is fully integrated in the CMSSW framework
- Part of the reconstruction algorithms rewritten to offload to GPUs
  - Initially using CUDA for NVIDIA GPUs
  - Partially ported to Alpaka
- Deployed at the HLT since the beginning of LHC Run 3 (2022)
  - Algorithms: Pixel local reconstruction, Pixel tracking, ECAL and HCAL local reconstruction
  - More algorithms being ported
- Offloading up to 40% of runtime to GPUs
- Validation and monitoring using the CMS Data Quality Monitoring system
  - CPU vs. GPU validation of fraction of events from HLT
- Code can also be used for offline reconstruction



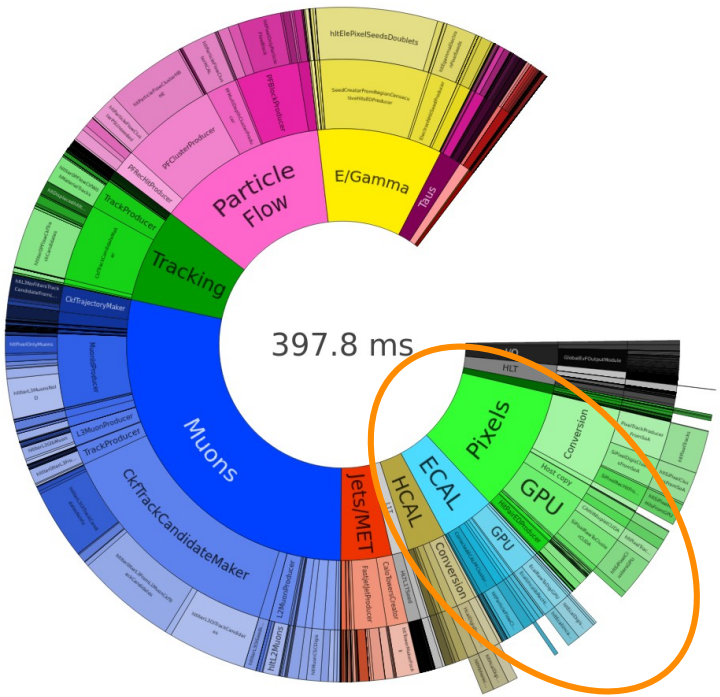
Performance

- HLT with GPU offloading (2022 numbers)
  - 40% less time per event
  - 50% better performance per W
  - 20% better performance per initial cost

https://cds.cern.ch/record/2851656/files/DP2023_004.pdf



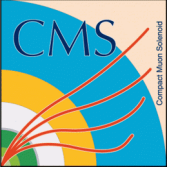
No GPU, 32 threads, 24 streams



GPU, no MPS, 32 threads, 24 streams

Alpaka

- CMS has chosen Alpaka as performance portability library
- Alpaka provides portability of code for various backends by adding an abstraction layer for the backends parallelism
- All backends of interest for CMS are supported
  - Serial (CPU)
  - NVIDIA and AMD GPUs
  - Intel GPUs and FPGAs (experimental)
- Performance close to native code
- Open source
  - Some improvements contributed already by CMS people
- Alpaka support added to CMSSW
  - Documentation
  - Examples
- Some CUDA algorithms ported to Alpaka and in production
  - More currently being ported
  - Some newly developed directly with alpaka

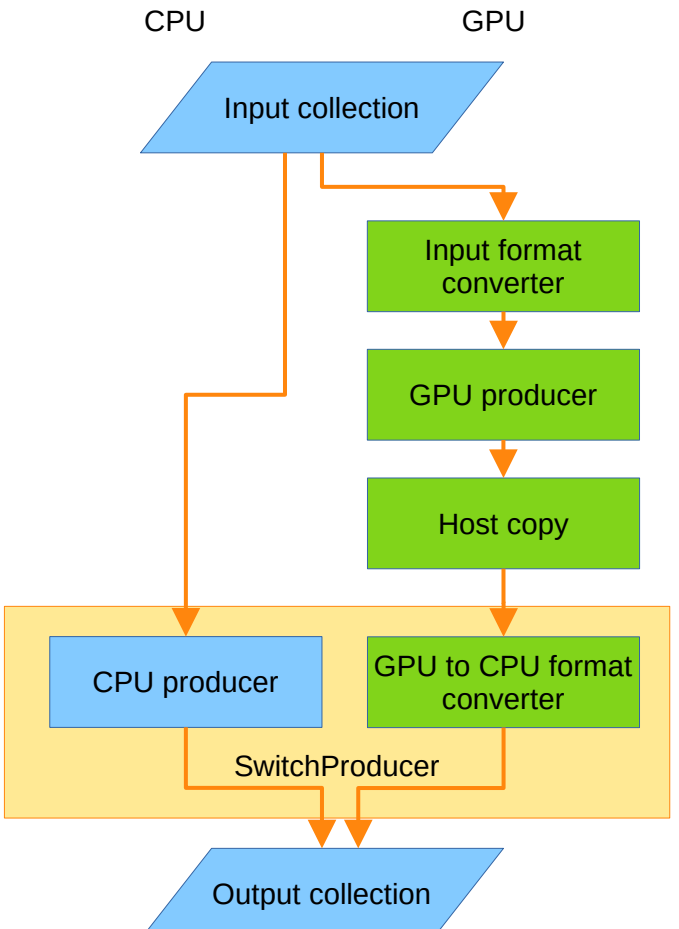


Science and Technology Facilities Council

Framework implementation details

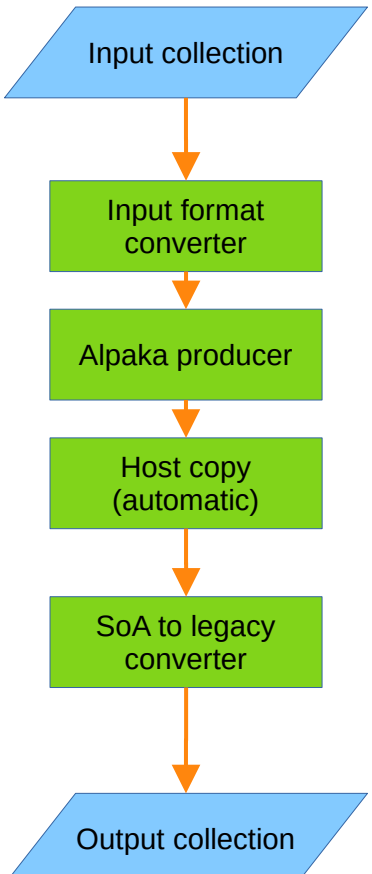
SwitchProducerCUDA

- Separate CPU and CUDA versions of the algorithms
- The framework needs to run the right module based on the availability of a GPU
- Solved by SwitchProducerCUDA module
  - A special module in the python configuration for cmsRun pointing to a CPU producer module or a GPU producer module
  - Output collection type of the two branches must be the same
- In practice the GPU branch module is a module that converts the GPU data format to the legacy CPU format
  - The actual GPU algorithm module is automatically scheduled by the framework to run before the conversion module



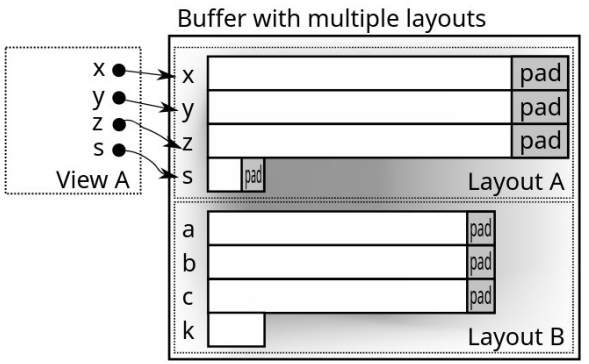
CMSSW and Alpaka

- Alpaka eliminates the need for a SwitchProducer module
  - Backends are chosen automatically depending on availability
- Still requires a conversion module to legacy data formats for downstream CPU modules
- The framework automatically takes care of the host copy if needed
  - In case a CPU module requires a data product produced on an accelerator
- Automatic device copies are provided for conditions consumed on a device
- Provides various functions to help with looping over elements
- The Alpaka dependent code is compiled for each backend
- Code for accelerators is placed in a special namespace whose values depend on the backend the code is compiled for



Data formats

- In order to work efficiently parallel algorithms need data formatted as structures-of-arrays (SoA)
  - Minimise memory access operations
  - Legacy data formats in CMSSW are arrays-of structures (AoS)
  - AoS “feels” more intuitive to may developers
- New generic SoA data format developed
  - Usable with CUDA and Alpaka code
  - Layout divides a buffer and views to access the data
  - Supports columns (vectors), scalars, and Eigen like matrices
- Portable collections (host and device side) wrap SoA format layout and views
  - Manages buffer allocations, interface to memory transfers, and serialisation to ROOT files
  - Access syntax close to AoS




```
GENERATE_SOA_LAYOUT(TestSoALayout2,
                    // columns: one value per element
                    SOA_COLUMN(double, x2),
                    SOA_COLUMN(double, y2),
                    SOA_COLUMN(double, z2),
                    SOA_COLUMN(int32_t, id2),
                    // scalars: one value for the whole structure
                    SOA_SCALAR(double, r2),
                    // Eigen columns
                    // the typedef is needed because commas confuse macros
                    SOA_EIGEN_COLUMN(Matrix, m2))

using TestSoA2 = TestSoALayout2<>;
```

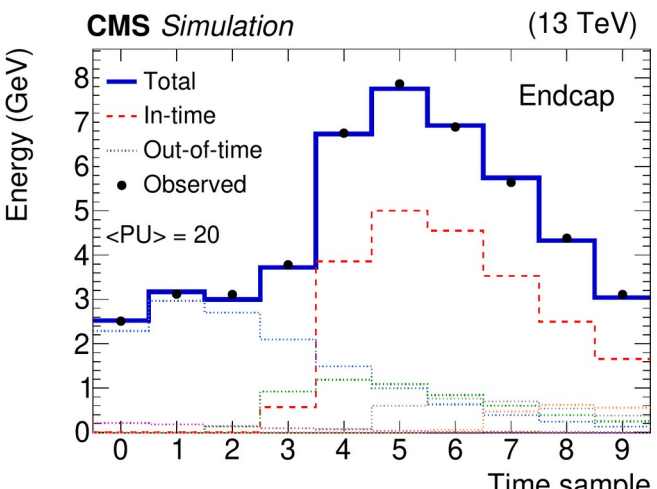
ECAL local reconstruction

ECAL local reconstruction

Processing step	Data collection
• Raw data unpacking	→ EcalDigis

• Amplitude reconstruction
  ▸ Multifit algorithm
• Time reconstruction
  ▸ Ratio method
  ▸ Future: Cross correlation method
	→ EcalUncalibRecHits

• Energy reconstruction	→ EcalRecHits

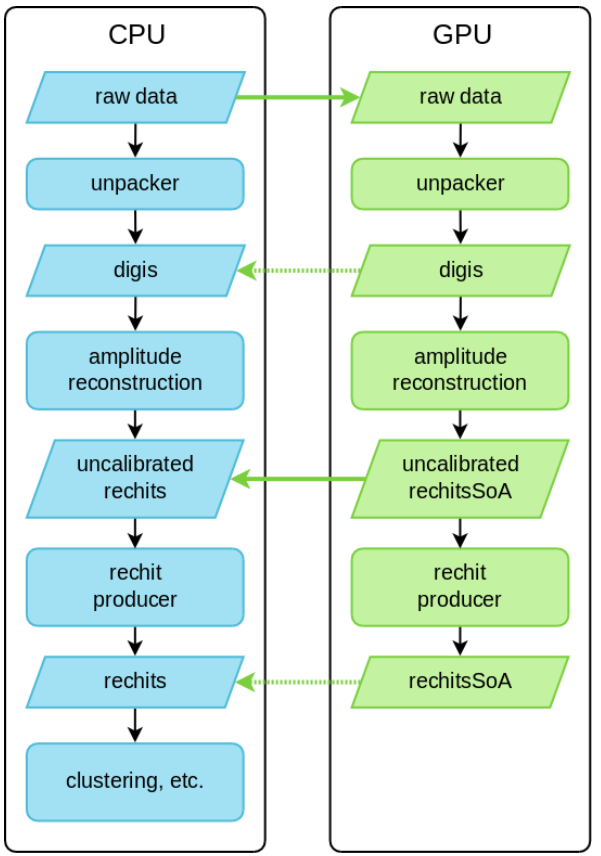


ECAL multifit reconstruction

ECAL local reconstruction on GPU

- Running at the HLT during Run 3
- Mostly migrated to Alpaka from original CUDA version

- Modules
  - Unpacker for digis
  - Multifit amplitude and time reconstruction
  - Energy reconstruction
    - Not all CPU features are available in GPU version
      E.g. Energy recovery
    - Not running at the HLT
  - Various conversion modules between legacy and portable data formats



ECAL reconstruction chain with GPUs

ECAL portable data formats

- Data formats
  - EcalDigiSoA
  - EcalDigiPhase2SoA (for HL-LHC upgrade)
  - EcalUncalibratedRecHitSoA
- Only digi data formats differ between Run 3 and Phase 2
- Explicit size scalar part of data format
  - Because of zero suppression in raw data the number of digis is only known at runtime

```
GENERATE_SOA_LAYOUT(EcalUncalibratedRecHitSoALayout,
                    SOA_COLUMN(uint32_t, id),
                    SOA_SCALAR(uint32_t, size),
                    SOA_COLUMN(float, amplitude),
                    SOA_COLUMN(float, amplitudeError),
                    SOA_COLUMN(float, pedestal),
                    SOA_COLUMN(float, jitter),
                    SOA_COLUMN(float, jitterError),
                    SOA_COLUMN(float, chi2),
                    SOA_COLUMN(float, OOTchi2),
                    SOA_COLUMN(uint32_t, flags),
                    SOA_COLUMN(uint32_t, aux),
                    SOA_COLUMN(EcalOotAmpArray, outOfTimeAmplitudes))

using EcalUncalibratedRecHitSoA = EcalUncalibratedRecHitSoALayout<>;
```

ECALUncalibratedRecHitSoA

- Conditions data formats (conditions for all channels also stored as SoA)
  - EcalElectronicsMappingSoA
  - EcalMultifitConditionsSoA
  - EcalMultifitParametersSoA

Kernels

- In the current implementation all kernels use the same queue

- Unpacking kernel
  - 1D kernel with 32 channels per block

- Amplitude kernels
  - 1D and 2D kernels prepare the input data for the minimisation
  - Minimisation kernel uses the same core function than HCAL algorithm
- Timing kernels
  - 6 separate kernels for initialisation, correction for the pre-amplifier slew rate, calculation of the null hypothesis, ratio method calculation, chi2 and amplitude, and final time corrections

Experience during Run 3

- Significant improvement in reconstruction time (~50% faster)

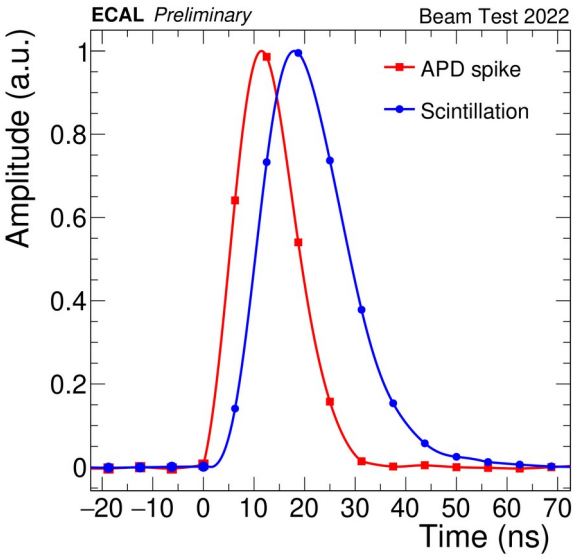
- Initial memory consumption issues mitigated
  - Optimised unpacker memory requirements
- Unpacking of corrupted raw data can lead to problems
  - Not as many sanity checks as in CPU unpacker initially implemented
- GPU unpacker does not unpack auxiliary collections
  - Initially not planned to use them at HLT
  - Now the features are used in dead channel recovery and particle flow clustering
  - Temporary workaround: Run CPU and GPU unpacker and use aux. collections from CPU unpacker
  - Work ongoing to upgrade unpacker to unpack aux. collections

Alpaka migration of ECAL local reconstruction

- New generic SoA formats and portable collections simplify algorithm code enormously
- Automatic device to host copies of data and host to device copies of conditions allow to remove several dedicated framework modules
- Much of the kernel code did not need to be changed
  - Mostly changes to kernel interfaces and due to use of new portable collections
  - Core fit function could stay the same (also used by HCAL local reconstruction)
  - Some loops needed to be split in outer and inner loops order to use shared memory
    - On GPU there are multiple threads each working on one element (no inner loop was needed in CUDA version)
    - On CPU one thread works on multiple elements and inner loops over the elements are needed, separated by synchronisations
- Migration of unpacker and amplitude and time reconstruction modules still took more than a year
  - ~5k lines of code spread over ~100 files
  - Adapt to framework changes while migrating
  - Due to the Alpaka design the compilation errors can be very hard to understand which makes debugging slow
- Reconstruction time practically unchanged from CUDA version
- Improved CPU vs. GPU agreement
  - Yes, it is the same code but compiled differently and small discrepancies can still exist

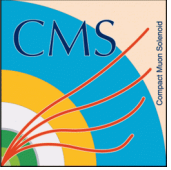
HL-LHC ECAL local reconstruction

- Differences wrt. Run 3 reconstruction
  - No ECAL endcaps
  - Shorter pulses
  - 16 samples per digi instead of 10
  - 160 MHz sampling frequency instead of 40 MHz
  -
- Similarities wrt. Run 3 reconstruction
  - ECAL barrel remains
  - Same or similar data formats
  - Same or similar reconstruction modules
    - Likely to use multifit algorithm for amplitude reconstruction
    - May reconstruct fewer Out-of-time PU pulses since pulses are shorter



ECAL HL-LHC upgrade pulses

- A weights based amplitude and time reconstruction was developed as baseline
- CUDA version of the weights algorithm developed by Southampton master student
- Alpaka migration by a second master student almost finished

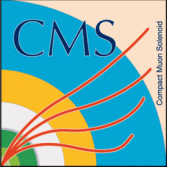


Science and Technology Facilities Council

Ongoing and future developments

Future improvements

- Work ongoing to migrate more modules to Alpaka
  - HCAL code migration almost finished
- Currently testing with non-NVIDIA GPUs (AMD)
- Add missing links to chain algorithms together on device
  - Avoid expensive copies of data back to host memory
- New developments for accelerators
  - Particle Flow clustering, Strip unpacking and clustering, Primary vertex reconstruction, Electron seeding algorithm
- Use portable code also for offline reconstruction
- Further in the future: remove legacy code



Science and Technology Facilities Council

Questions?