LHCb's Allen Application and Framework

EIC Software Week 2024

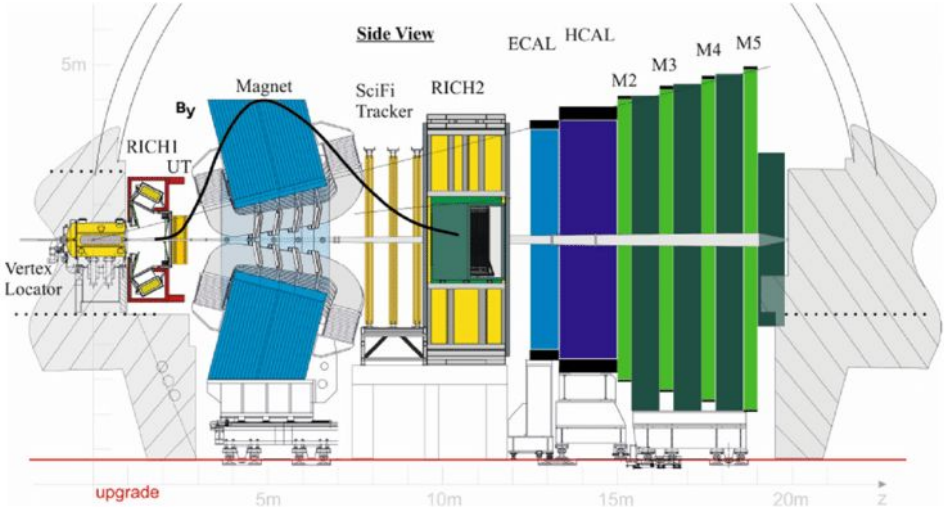
Roel Aaij and Dorothea vom Bruch

April 25th, 2024

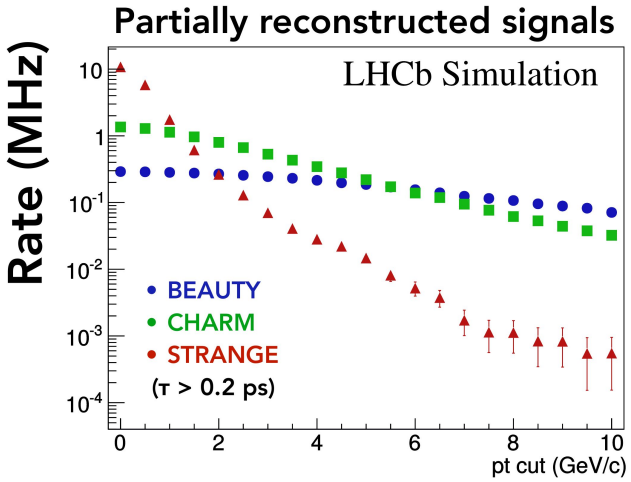
Nikhef

CENTRE DE PHYSIQUE DES PARTICULES DE MARSEILLE
CPPM

LHCb Detector

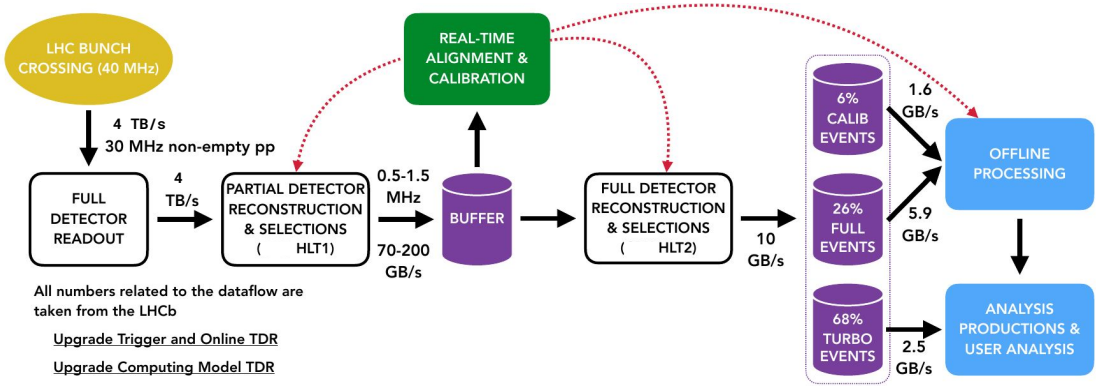


LHCb Upgrade Physics in a Single Slide



30 MHz (4 TB/s) of input contains a MHz of signal, while we can only store 10 GB/s long-term

LHCb Upgrade Dataflow



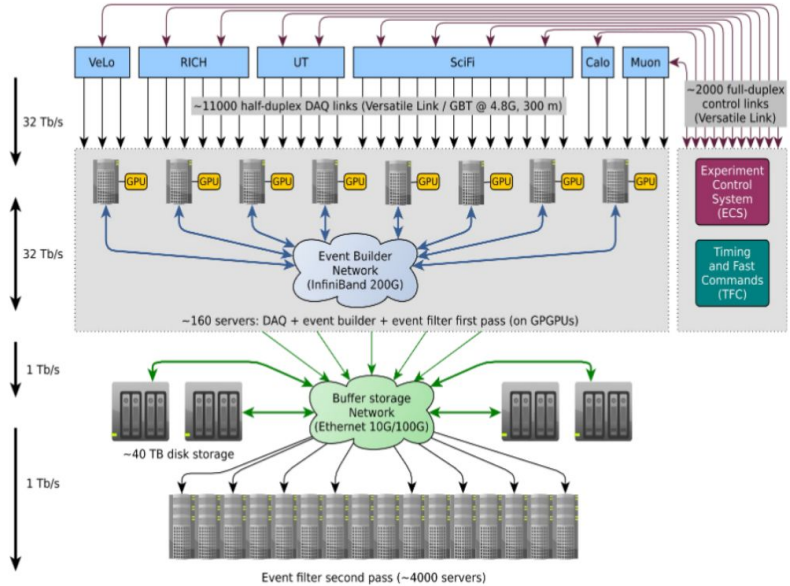
All numbers related to the dataflow are taken from the LHCb

Upgrade Trigger and Online TDR

Upgrade Computing Model TDR

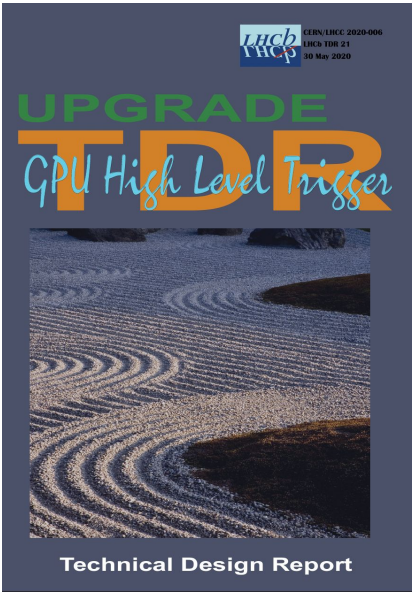
HLT1 challenge: reduce ~4 TB/s to 100 GB/s in real-time with high physics efficiency

LHCb Upgrade Trigger and DAQ



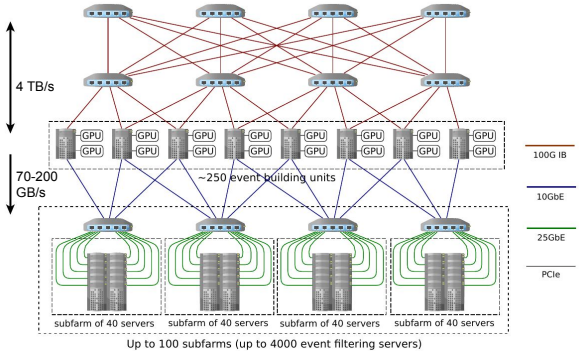
LHCb HLT1

- Inclusive trigger
- Reduce rate from 30 MHz to 1 MHz
- Need to reconstruct:
  - Velo tracks
  - Primary Vertices
  - “Long” tracks (Velo->UT->SciFi)
  - Muon ID
- Optional ingredients
  - ECal reconstruction
  - Electron ID
  - Photons
  - “T-Tracks” (SciFi)
  - “Downstream” tracks (UT-SciFi)
  - RICH PID
- Avoid global event cuts if possible

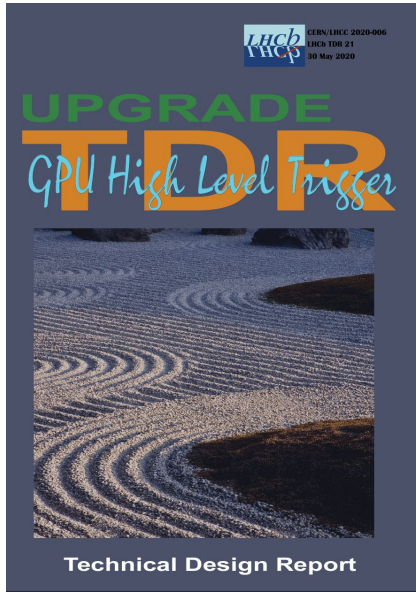


GPU HLT1 TDR

HLT1 on GPUs: Allen

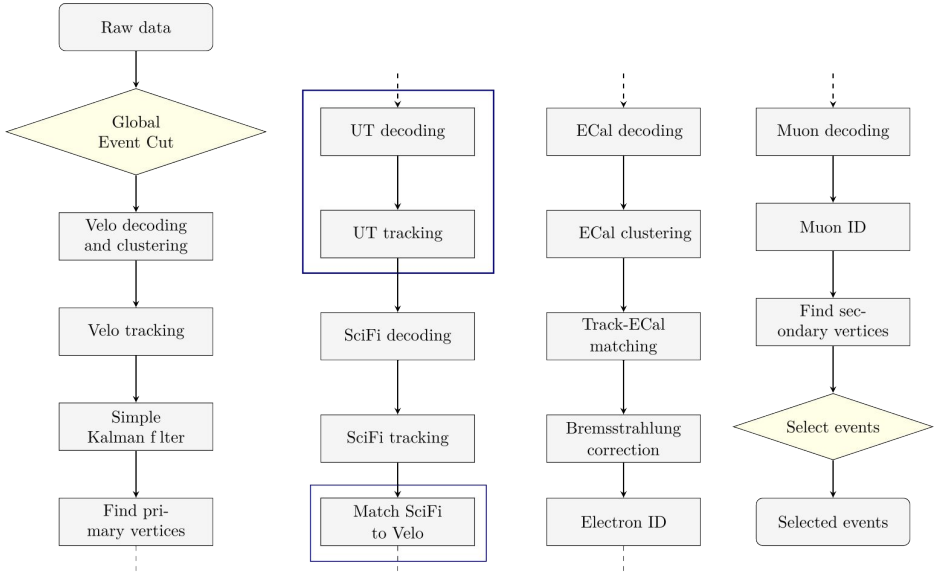


Allen implements HLT1 as a GPU application; currently 2 GPUs installed in each event builder server



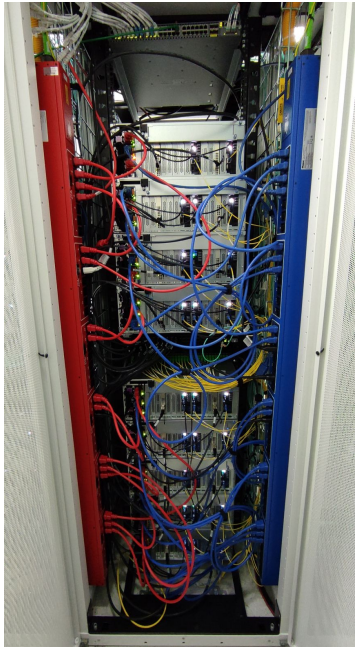
GPU HLT1 TDR

Allen Kernels



DAQ with GPUs

- 400 GPUs installed in Event Builder servers
- Input data copied to GPUs in EB format: ~1000 multi-fragment-packets in multi-event-packets of 30000 events
- 20-25 GB/s per server
- Event data memory layout “transposed” with respect to event-by-event
- Input data directly from shared memory
- Output in custom binary format to DAQ
- Experiment Control System steers HLT1
- Obtain geometry and conditions from LHCb software on-the-fly



1-Slide Framework

- gitlab: gitlab.cern.ch/lhcb/Allen
- C++17 (soon 20), CUDA (12.X), HIP (5.X)
- Built with CMake and runs on CPU and GPU (NVIDIA and AMD)
- Standalone builds and “stack” builds

- Single precision throughout
- Batches of ~1000 events (~100 kb/event)
- GPUs have their own memory; framework provides functions to copy data
- ~10 batches in parallel using CUDA/HIP streams (1 CPU thread per stream)
- No dynamic allocations
- Configurable (in Python) sequence of algorithms
- Asynchronous event loop
- All algorithms written from scratch for good performance on GPUs
- documentation: https://allen-doc.docs.cern.ch/

Philosophy

- Must not interfere with event building
- Do everything on the GPU: raw data in, decisions and candidates out
- Maximise (GPU) algorithm performance
- Start with barebones framework and write kernels in CUDA
- Implement performant reconstruction algorithms,
  i.e. significantly faster/$ than on CPU
- Batches of ~1000 events with control flow
- Event model **will** evolve so keep it simple
  - little to no dynamic memory allocation
  - SOA containing (small structs of) PODs
  - Count first, write later
- Minimise serialization of event data
- Opportunistic use of the CPU
  - Prefix sums
  - Monitoring
  - Low IPC algorithms that require little data

Portability

- No portability frameworks, just write CUDA
- `#ifdef` and a tiny middleware (1400 LoC) to allow running on CPU (x86, ppc64le, ARM) and AMD GPUs
- Port to Intel GPUs nearly ready
- Allow dispatching to architecture-specific functions for extra performance
- No performance penalty due to portability

Configuration

- Database of algorithms, inputs, outputs and properties built using code parsing with `libclang`
- Allow configuration of the sequence of algorithms/kernels
- Allow properties of algorithms to be set
- Multiple instances of an algorithm with separate inputs and outputs
- Configuration in Python using LHCb's PyConf package

Memory Management

- Memory allocations on the GPU are very slow
- Allocate memory for event data up front
- Chunk of memory allocated per stream
  ~1 MB per event

- Each algorithm proceeds in two steps:
  - Request memory for outputs
  - Run kernel
- Strong preference for “Count First, Write Later”
- Sequence uses data dependencies to track lifetime
- Device Memory is released as soon as possible
- Failure to reserve memory aborts the batch ->
  Split in two and try again
- Host memory done analogously, but not released until after data is output from the application

Event Loop

- 150 kHz of events per server
- 20-25 GB/s per server
- Low overhead
- Batches of ~1000 events

- Based on ZeroMQ
- Initial use case was a benchmark for performance measurement
- Asynchronous event processing added later
- Support four data flow models:
  - Benchmark with single batch of events
  - Process all input in a set of files (simulation, development)
  - Externally controlled, i.e. wait for data and process whatever arrives and stop/exit when told
  - Benchmark with multiple (preloaded) batches
- All of this is currently mixed together in the same code
- It works, but is not very pretty and needs refactoring

Integration with LHCb stack

- Need geometry and conditions data
- LHCb conditions change slowly (a 10 minute runs is considered short)
- All required geometry and conditions data are converted to blobs that can be memcpy'd to the device and are fast to use
- Use parameterisations when possible,
  e.g. currently no magnetic field map on the device
- Setup a Gaudi/LHCb application with a fake event loop "on the side"
- When data with a new run number arrives:
  - Finish processing data of previous run
  - Trigger a single event in the fake event loop to update blobs
  - Copy new blobs to the device
  - Restart processing
- A derived Gaudi::Application handles interaction with control system
- Input and output to the DAQ are Gaudi Services with an extra ABC

Development

- Standalone build has helped developers
  - Easy to build anywhere
  - Fast turnover
- Getting into CUDA is not harder than C++
  - Parallel programming requires a different mindset
  - Good docs and examples are very important
  - Making it performant is hard
- A good and fast CI pipeline is a huge asset
  - Allen CI runs CPU and GPU builds and a representative set of tests in about 10 minutes on every push to every MR
  - More extensive pipeline takes another 20 minutes
  - Fast feedback for developers
- Very agile development in the first years
  - Helped test new ideas
  - Great motivation for contributors

People

- Project grew quite fast from the start
- Good core team with complementary skills
  - GPU performance and parallel programming
  - Knowledge of the experiment and existing stack
  - Project management
  - Modern C++/CUDA
  - Mixed CS/HEP background
- Political support
- Support from DAQ/Online

- Most contracts are short, so bringing new people and skills on board has to happen all the time
- Knowledge transfer is hard
- Skilled people are few.
  - They are highly motivated, but career prospects matter
  - Allen has already contributed to at least 4 people getting a big grant or a permanent position.

Commissioning 1/2

- Running in production since 2022
- Overall good performance, Allen has rarely been a bottleneck
- Many moving targets
  - DetDesc -> DD4hep
  - Quickly changing detector conditions
  - Different sets of detectors participating in data taking
  - Evolving sub-detector geometries and data formats
- Additional features requested and implemented with very fast turnover
  - Output of short trains of bunch crossings
- Minimal monitoring was a serious issue
  - Slowed-down iteration between detector experts and reconstruction experts

Commissioning 2/2

- Performance has been good, 2nd GPU available since 2023
- Running at target pile-up
- Currently about 100 kHz of events per GPU
  - “matching” tracking sequence
  - ECal reconstruction
  - Fast MLPs for ghost rejection
  - electron ID, improved muon ID
  - nearly 90 selections
  - fixed-target experiment in parallel (SMOG)
  - luminosity measurement
- Monitoring improved for 2024
  - easy histograms on the device
  - double buffering
- Configuration provenance implemented for 2023
  - each configuration has unique identifier
  - stored in each event

Framework Issues

- Too little protection against memory errors
  - Pool allocations make it worse
  - Move to span instead of raw pointers
  - Have tools to assist with debugging
- Two memory layouts of raw data supported
- Event loop code is very messy
- Complex application, many threads
- Too tight coupling between input handling, output handling, sequence and overarching data-flow mode
- No Service equivalent, e.g. detector data store is a `struct`
- (Physics) Monitoring came quite late
  - Very important during commissioning
  - Device-side monitoring more important than we thought
- Glue-like interface to LHCb stack is not very elegant
- Need more tests

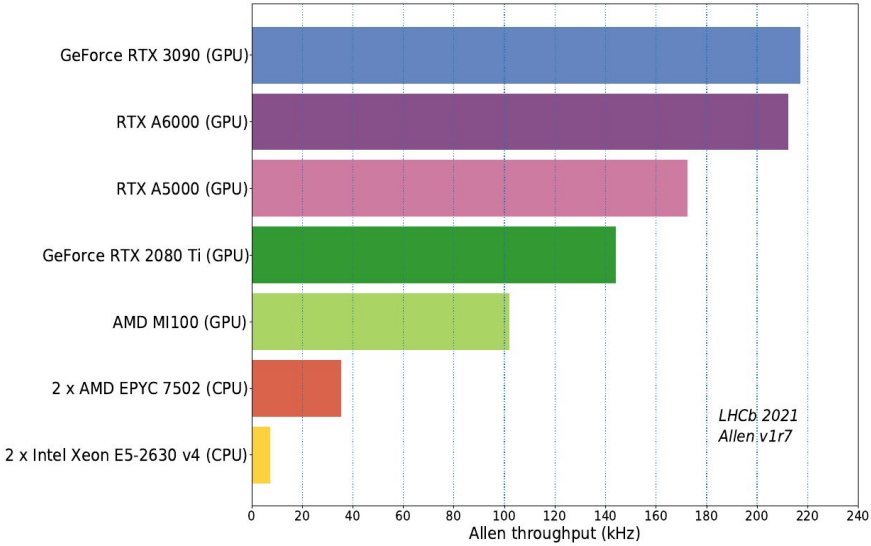
Future (IOHO)

- LHCb Upgrade 2 baseline is all of HLT1 and all HLT2 reconstruction on GPUs; particle combinatorics better done on CPUs
- Amount of preprocessing and reconstruction on FPGAs to be decided
- Gaudi has already solved many of the issues that Allen has
- Maintaining two frameworks makes little sense
- Impedance mismatch is actually rather small

- Put batched input data on the event data whiteboard
- `DeviceAlgorithm` that implements the two-step approach of Allen
- `DeviceDataHandles` to interact with the memory pool manager and handle copying of data between host and device
- `DeviceConditionAccessor` to manage device geometry data using derived conditions
- `DeviceBatchContext` to propagate the GPU stream
- No need for an additional portability framework
- Use LHCb's CPU scheduler to schedule batches
- Static balancing of GPU/CPU load, i.e. as different sequences/applications

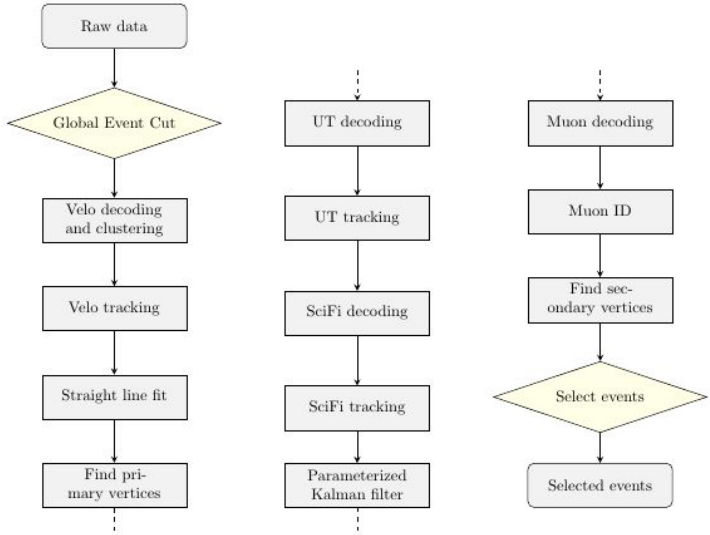
BACKUP

Throughput



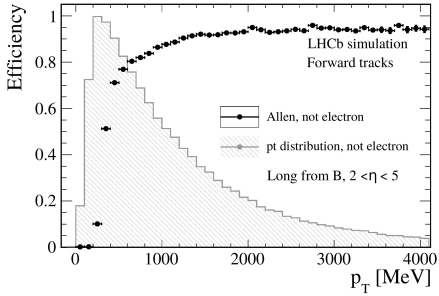
LHCb-FIGURE-2020-014

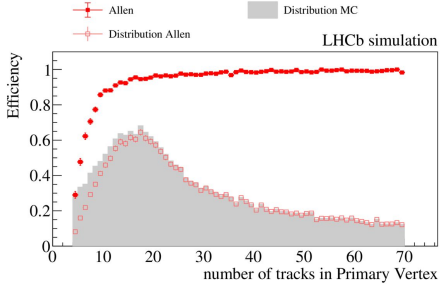
Reconstruction Sequence

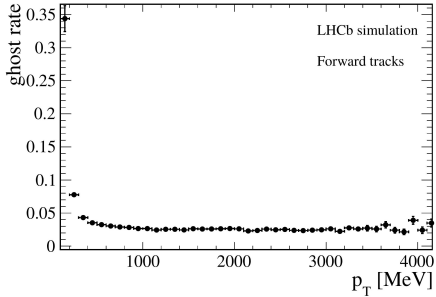


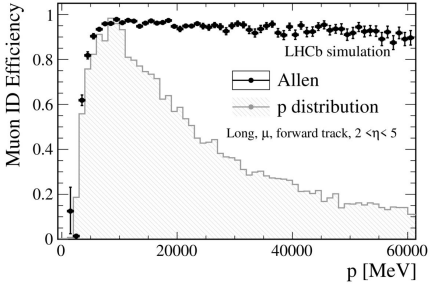
GPU HLT1 TDR

Reconstruction Performance









LHCb-FIGURE-2020-014