Experiences on the software performance of LHCb's first level trigger

Arthur Hennequin – arthur.hennequin@cern.ch
On behalf of the RTA team

The LHCb dataflow

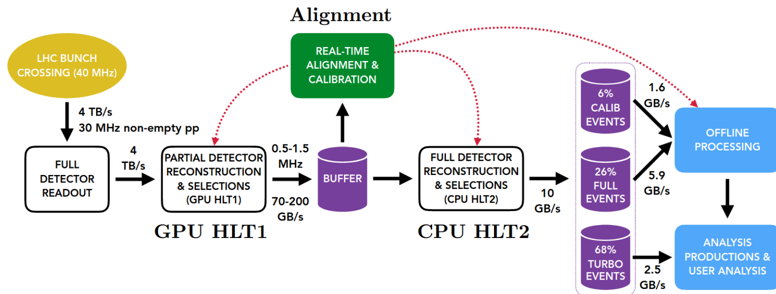


Figure from HLT1 TDR

The event builder / HLT1 farm

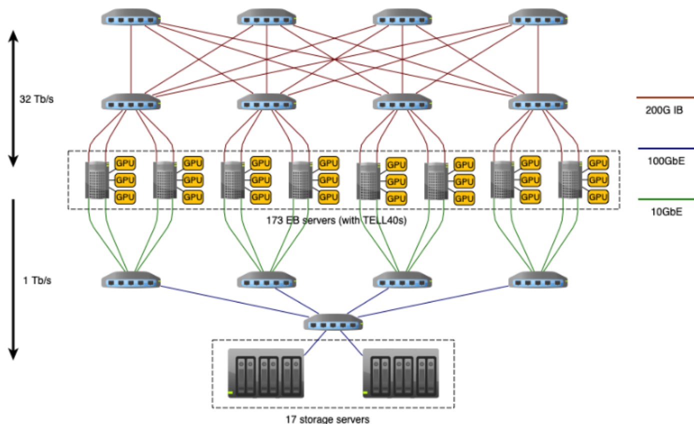
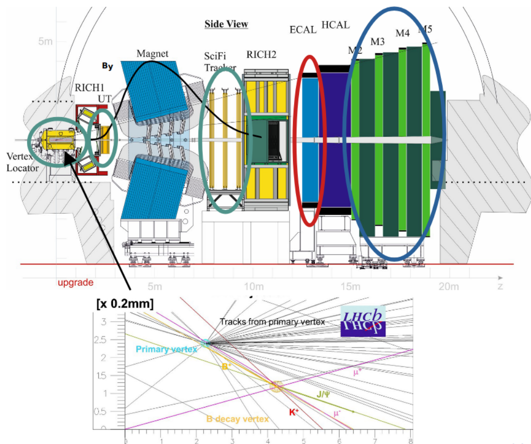


Figure from HLT1 TDR

Real time analysis

HLT1 tasks (as in the TDR):

- Decoding of binary rawbanks from each sub-detector
- Reconstuction of charged particles trajectories
- Identification of electron and muon particles
- Reconstruction of primary and particle decay vertices
- Selection of proton-proton bunch collisions (events) to store in buffer



HLT1 commissioning

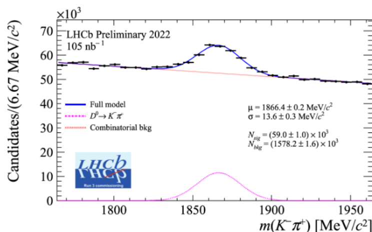
Timeline:

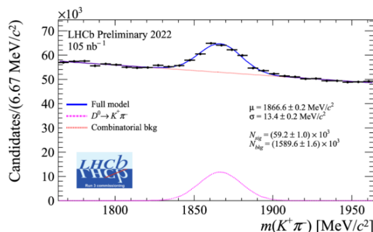
- 2022: little data, mostly dedicated to commissioning
- 2023: vaccum incident in LHCb Velo and UT not yet installed
- 2024: data not yet approved, comming soon

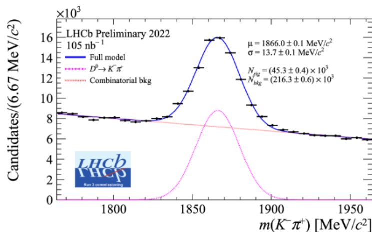
HLT1 adapted successfully to each new issue:

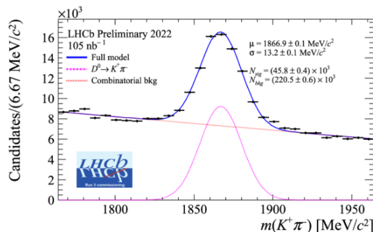
- developped a new sequence capable of reconstructing long tracks without the UT
- re-tuned the selections with a VELO not fully closed

$D^0 \to K^-\pi^+$ HLT1 mass plots (2022)









LHCb-FIGURE-2023-009

HLT1 on GPUs: The Allen framework

Quick tour:

- Multi-stream, multi-event batches
- Static sequence scheduler
- Dynamic stream-ordered memory allocator with statically known lifetime
- Minimal impact monitoring
- Fully deterministic algorithms (ie. non-deterministic operations have no impact on output)
- Hand optimized kernels
- Emulation through compiler macros for CPU targets

Multi-level parallelism

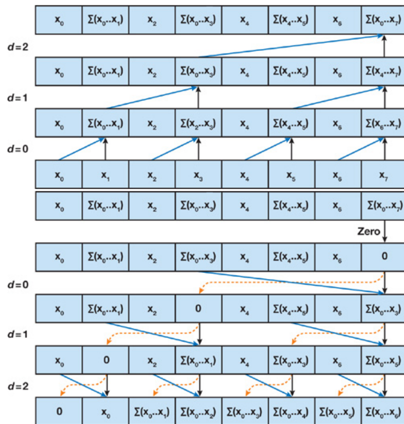
- Each node of HLT1 contains multiple GPUs
- Each GPU is attributed to one Allen instance
- An Allen instance runs multiple streams (configurable, typically 16)
  - A stream has one CPU thread taking care of the scheduling, dynamic memory allocations and host algorithms.
  - All kernels from an Allen Stream runs in the same CUDA stream (sequentially) $\Rightarrow$ multiple kernels from different streams runs concurently on the GPU to ensure full resource utilisations
- Each stream process a slice of events (configurable, typically 500)
- Each algorithm is free to choose how to parallelise over the slice of events. A common pattern is to assign one cuda block to one event and each threads to the objects (hits, tracks, ...) that are processed.

Static scheduling and dynamic allocations

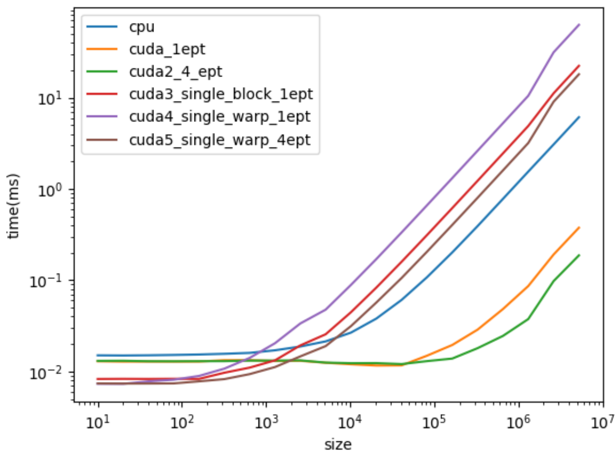
- The dataflow graph representing each sequence is linearized into a sequence that respect the data dependencies
- The lifetime of each buffer is pre-computed and stored for the allocations
- Allocations can be dynamic but only between kernels (we don't do any allocations inside a kernel)
- A common pattern is to calculate or estimate the buffer size in a first kernel, then run a prefix sum on the array of sizes to get the offsets of each event, allocate the buffer and then run the kernel that fill the buffer. $\Rightarrow$ since prefix sums are widely used, special care was taken to optimise the algorithm, allowing up to 8% throughput gain in some sequences.

Prefix sum on GPU

- We have a lot of prefix sums of various size to do in a typical sequence:
  - $O(10^3)$ events per slice
  - $O(10^5)$ tracks per slice
  - $O(10^6)$ hits per slice
- Different algorithms based on the data size.
- Based on Blelloch's scan
- Exploit GPU memory hierarchy



Prefix sum - throughput



(Algorithms evaluated on synthetic data)

Running selections

Selections are very different from reconstruction algorithms:

- Reconstruction algorithms:
    - Large kernels
    - Very sequential (one kernel run after the other)
    - Written by GPU experts
- Selection algorithms:
    - Very small kernels (launch cost become significant)
    - Embarasingly parallel ($O(80)$ different and independent lines)
    - Written by anyone

We mitigate selection kernel launch cost using CUDA dynamic parallelism to run all selections concurrently (we launch kernels from a parent kernel). First attempt was using template metaprogramming to run every lines in the same kernel, using dynamic parallelism provided a 10% throughput improvement.

Allen Monitoring

Monitoring is a crucial part of the trigger. Since we throw away 29/30 of the data, the monitoring plots are needed to understand system performance in real-time.

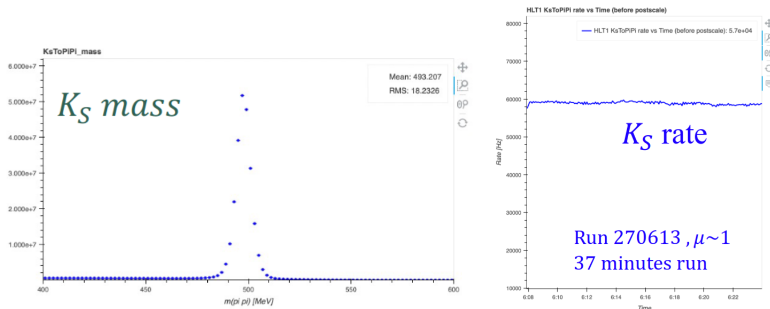


Figure from LHCb status report at LHCC open session

Allen Monitoring, first version (2022-2023)

- Simple implementation: each algorithm responsible for allocating histogram buffers, filling them using atomics and transfering back to host where they are merged with a Gaudi Accumulator (one instance per stream)
- Gaudi Monitoring take care of merging accumulators from different streams periodically
- Worked well! ... Until we had a closer look at throughput in data taking conditions. We found that monitoring cost $O(10\%)$ of our throughput.
- Why? Transfering many small buffers (one per histogram / counter) for every event slice, for every stream was creating a lot of pressure on the DMAs.
- The solution was to rewrite the whole monitoring system from scratch.

Allen Monitoring (v2)

- Interfaced with Gaudi Monitoring (which takes care of the aggregation of different nodes and the presenting)
- Supports counters and ND-histograms (design based on Gaudi Accumulators and adapted to GPU ⇒ familiar API)
- Aggregation is done on device: buffers are shared between streams and written to using atomics
- Buffers are persisted on device for multiple slices and periodically transfered to host, using double-buffering so the streams are never interrupted
- All monitoring allocations are static and done once for host and device, at configuration time.
- Now the throughput cost is negligeable.

The importance of optimisations

- We are on a budget ! currently 330 x A5000 GPUs installed in the EB farm. Room for a bit more but not much.
- Target throughput per card: 90 kEvts/s (to reach 30MHz, actual event rate closer to 25-26MHz)
- The baseline system described in the TDR worked very well, allowing to add a lot of new features (calorimeter reconstruction, removing the global event cut, low momentum track reconstruction, secondary vertices with 3 daughter particles, to cite a few..)
- Given theses additions, we are tight on throughput that we continue to improve to fit even more.
- Every throughput % and byte of memory counts, we must use every trick in the GPU programming book, and invent new ones...

Conclusion

- We had a successful data-taking with an all-software trigger system in 2022-2024 and are still improving it.
- Writting real time analysis software that runs at 30 MHz is challenging.
- Entry barrier for GPU programming is relatively low, but getting every bit of performance requires a deep understanding of the architecture, that is only acquired through years of experimentation.
- Good developpement tools are essential: profilers, monitoring counters, continuous integration, framework..
- There is no one-to-all solutions.

Thank you!