Data Handling in Allen at LHCb

Tom Boettcher

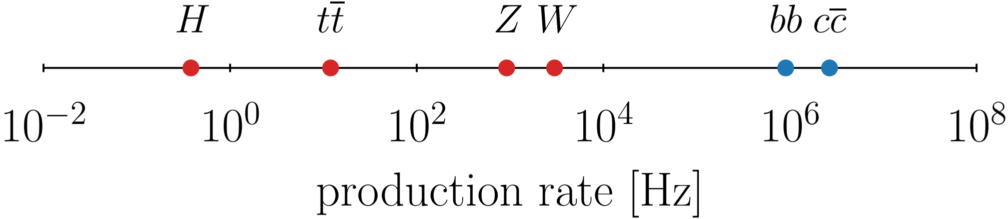
on behalf of the LHCb Real Time Analysis project

Streaming Readout X
May 18, 2022

Triggering on MHz signals

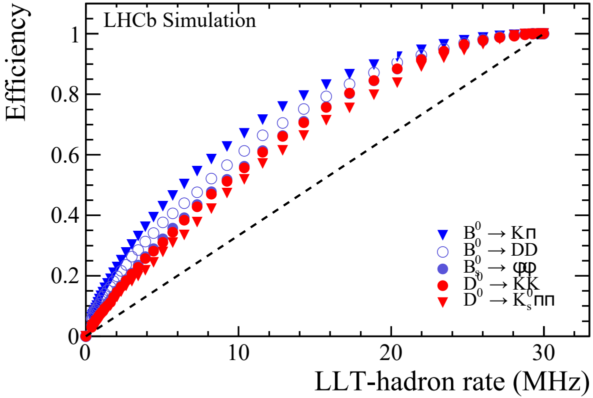
$\mathcal{L} = 2 \times 10^{34}\ \mathrm{cm}^{-2}\mathrm{s}^{-1}$ (ATLAS/CMS) $\qquad \sqrt{s} = 14$ TeV

$\mathcal{L} = 2 \times 10^{33}\ \mathrm{cm}^{-2}\mathrm{s}^{-1}$ (LHCb)



Triggering on heavy flavor decays

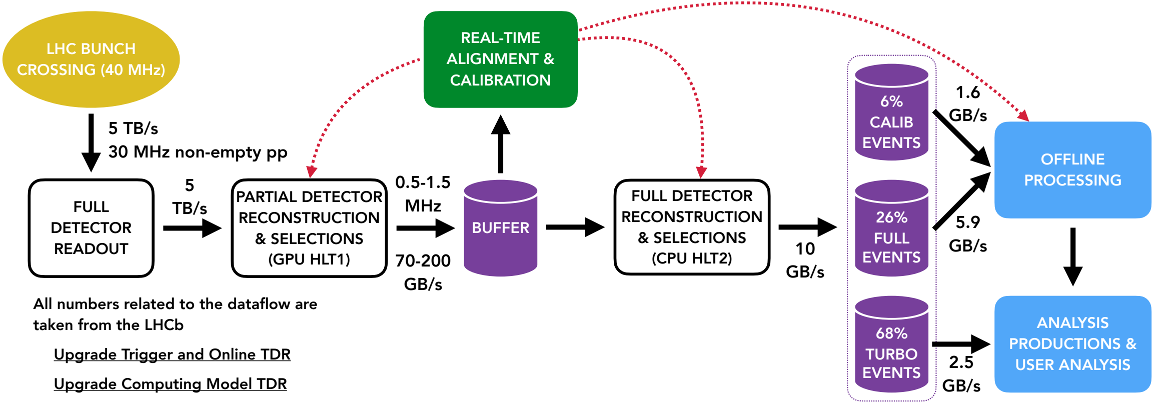
Hardware trigger performance LHCb-TDR-016



- Heavy flavor hadrons are long-lived and decay into low-momentum particles
- Can't effectively trigger on heavy flavor decays using hardware signatures
- Solution: process every event (30 MHz, 5 TB/s) in software

~~Triggering~~ → Real-Time Analysis

The Run 3 LHCb dataflow

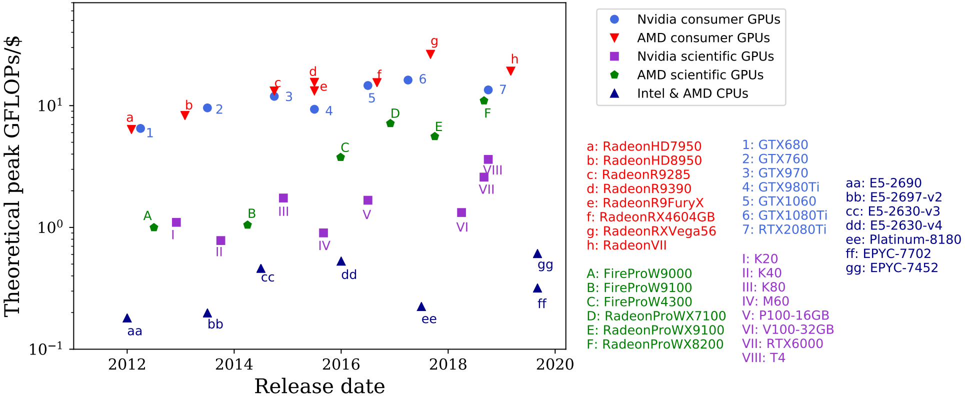


All numbers related to the dataflow are taken from the LHCb

- Upgrade Trigger and Online TDR
- Upgrade Computing Model TDR

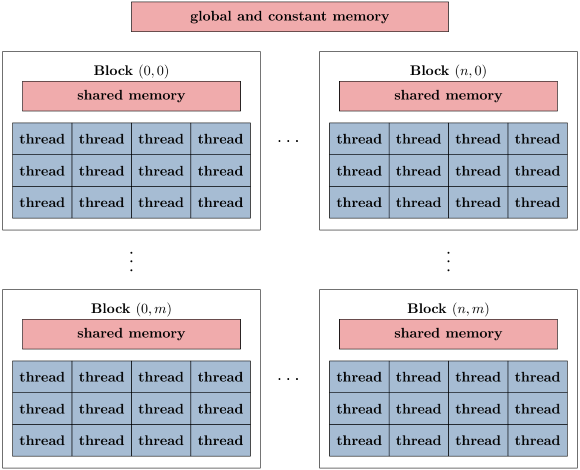
Why consider GPUs?

Affordable computing power in a compact package



Courtesy of Dorothea Vom Bruch, arXiv:2003.11491

GPUs in a nutshell



- Highly parallel processors with thousands of cores
- Relatively little memory
  - $\mathcal{O}(100\,\text{kB})$ “shared”
  - $\mathcal{O}(1\,\text{GB})$ “global”
- **S**ame **I**nstruction **M**ultiple **D**ata computing model
- Many trigger and reconstruction tasks are parallelizable with some work!

Unique challenges at LHCb

	LHCb, **Run 3**	ALICE, **Run 3**	ATLAS/CMS, Run 4
Hardware trigger	**No**	**No**	Yes
Readout rate	40 MHz *pp*	50 kHz PbPb	$\sim$ 1 MHz *pp*
Data rate	5 TB/s	3.5 TB/s	$\sim$ 5 TB/s

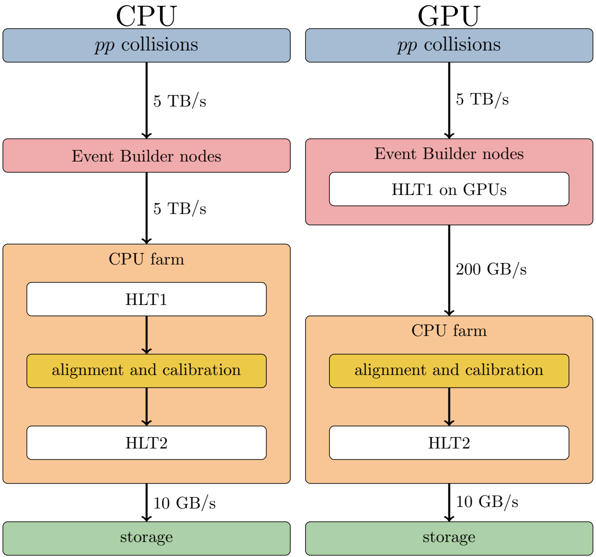
ALICE

- Large event sizes
- Reconstruction time dominated by TPC tracking with $\sim 80\times$ speedup on GPUs
- Clear use case for GPUs

LHCb

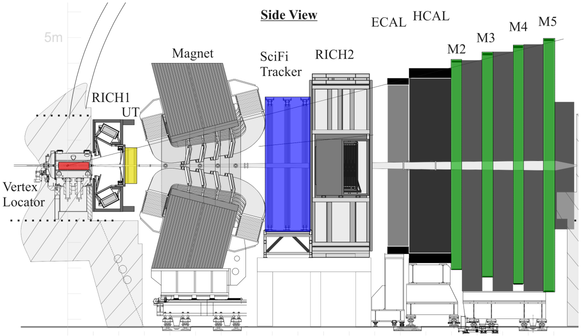
- Extremely high rate of small events
- No step dominates reconstruction time
- Advantages of GPUs compete with challenges

The GPU technology decision (CSBS 6 (2022) 1, 1)



- Fully CPU- or GPU-based HLT1?
- GPU solution leads to cost savings on processors and networking
- Enough throughput headroom for additional features
- **The final verdict:** A GPU-based software trigger will allow LHCb to expand its physics reach in Run 3 and beyond

Allen is LHCb's GPU-based first level software trigger (HLT1)

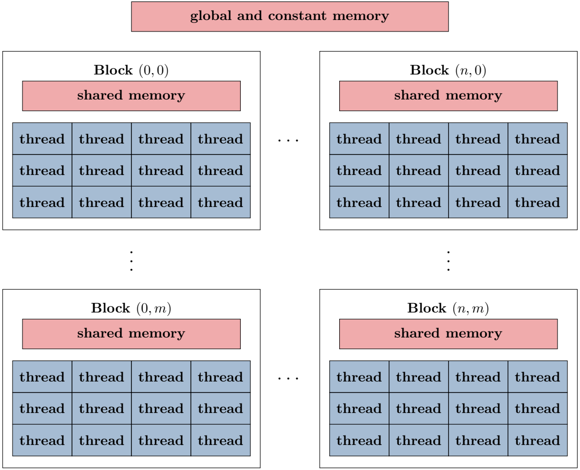


- Decode data from the **VELO**, **UT**, **SciFi**, and **Muon** systems
- Cluster detector data into "hits"
- Build tracks (**VELO**, **UT**, and **SciFi**)
- Find primary vertices (PVs) (**VELO**)
- Match tracks to **Muon** hits

Work as a standalone application or as part of LHCb's software stack

Can be compiled for CPU and GPU with CUDA or HIP

Adapting to GPUs



Parallelization strategy

- Each block processes 1 event
- Use threads for intra-event parallelism

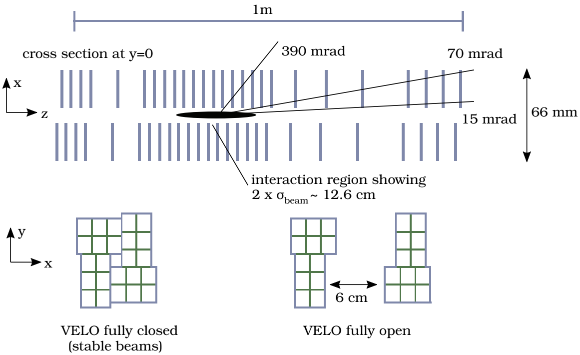
Take advantage of GPU computing power

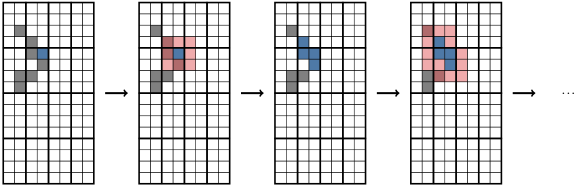
- Some reconstruction tasks work very well as parallel algorithms
- Can be rewritten to expose parallelism

Deal with the GPU limitations

- Not all HLT1 tasks parallelize nicely
- Need to be rewritten to deal with memory limitations, optimize memory access patterns, etc.

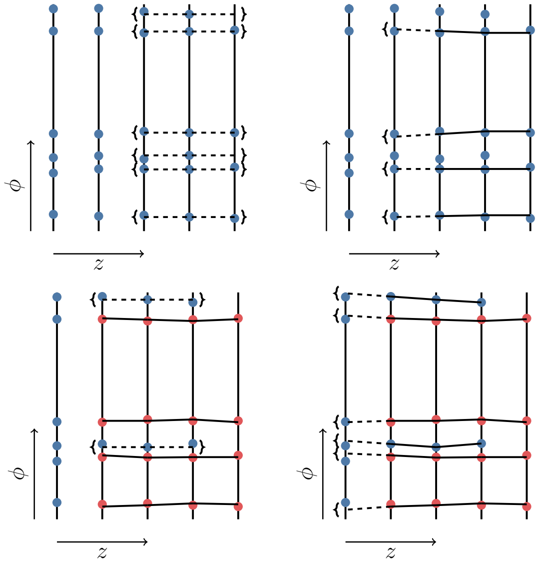
Example: VELO clustering





- 26 layers of silicon pixel detectors
- Identify seed candidates
- Cluster in constant time using bit masks

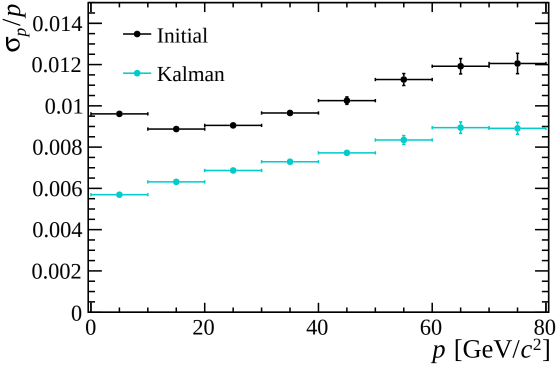
Example: VELO reconstruction



- Sort hits by azimuthal angle ϕ
- Create velo tracks
  - Create triplets of hits at similar ϕ
  - Extrapolate track candidates to the next layer and look for hits
  - Repeat starting at the next layer

D. Campora, N. Neufeld, A. Riscos Núñez: "A fast local algorithm for track reconstruction on parallel architectures", IPDPSW 2019

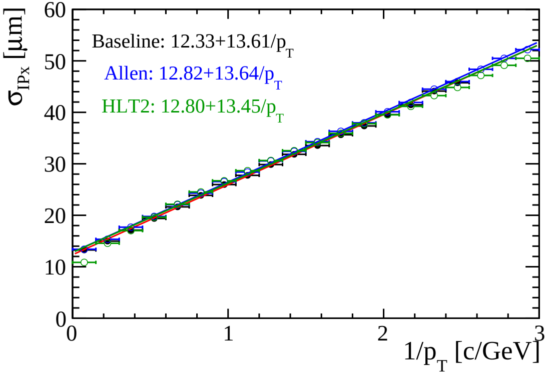
Example: Redesigning the Allen Kalman filter



- Achieve the best possible estimate of track parameters at the state closest to the beamline
- Improve momentum resolution
- Best possible estimate of track impact parameter
- Uses detailed magnetic field map and detector description to propagate tracks
- Requires many expensive calculations

Can we design a Kalman filter to work with GPU constraints without sacrificing physics?

Example: Redesigning the Allen Kalman filter



Parameterized Kalman filter CPC 265 (2021) 108026

- Parameterize track trajectories and multiple-scattering noise
- Calculations still expensive

Simplified Kalman filter

- Only fit the VELO segment
- Two independent fits in x and y
- Produces close-to-optimal IP estimate
- No improvement in σ_p but this doesn't really matter in HLT1

Simplified fit contributes negligibly to Allen's execution time and memory footprint and accomplishes the physics goals of HLT1.

Making data GPU-friendly

Structure of Arrays (SOA) vs. Array of Structures (AOS)

- AOS: $x_0, y_0, z_0, x_1, y_1, z_1, x_2, y_2, z_2$
- SOA: $x_0, x_1, x_2, y_0, y_1, y_2, z_0, z_1, z_2$

No dynamic memory allocation during kernel execution

- No `push_back` or `emplace_back`!
- Need to allocate global memory before kernel execution

GPU memory accesses are expensive

- Copying an entire structure from an AOS may take many global memory accesses
- Can often coalesce memory accesses using SOA formats

Allen data is generally stored in simple `C` arrays with corresponding arrays of offsets that provide structure.

Making data user-friendly

```
unsigned offset = offsets[event_number];
unsigned n_tracks = offsets[event_number + 1] - offset;
for (unsigned i_track = 0; i_track < n_tracks; i_track++) {
  float px = track_data[offset + i_track];
  float py = track_data[offset + n_tracks + i_track];
  float pz = track_data[offset + 2*n_tracks + i_track];
  ...
}
```

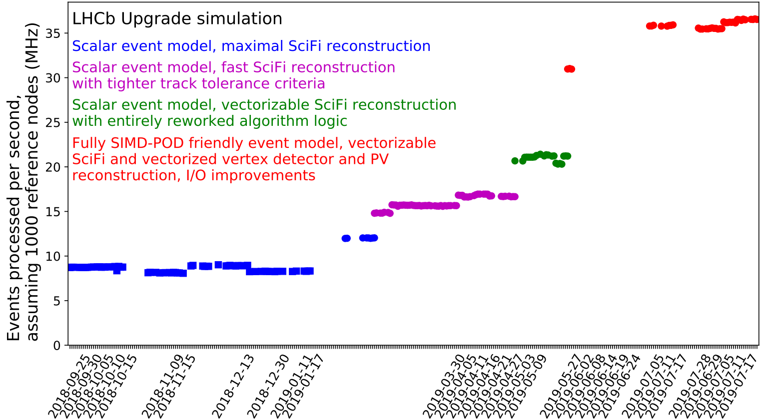
Vs.

```
for (auto track : tracks) {
  float px = track.px();
  float py = track.py();
  float pz = track.pz();
  ...
}
```

- Keeping track of arrays and offsets becomes difficult for complex data structures
- Changes to underlying data structures require changes to downstream code
- Solution: users access slices of arrays using **views**
- Handles indices and offsets behind the scenes

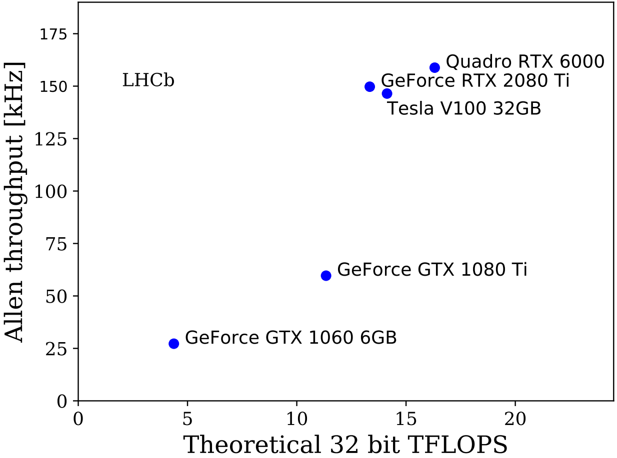
Not just GPUs...

CPU HLT1 throughput



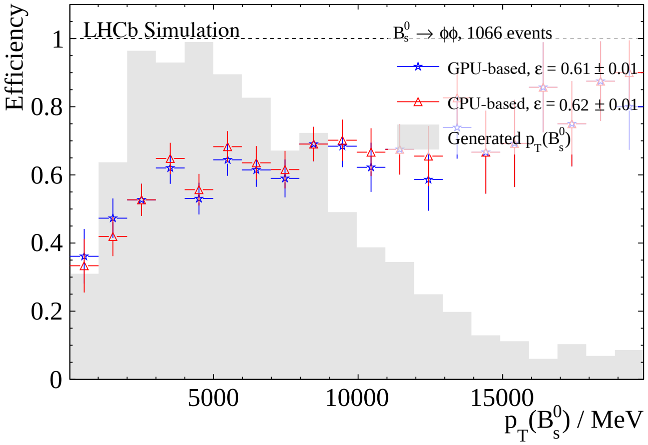
Designing algorithms and data structures to take advantage of parallelization is going to become more and more important regardless of hardware choice!

Computing performance



- Can handle the full LHC event rate with ~ 200 GPUs
- Throughput scales well with theoretical computing power

Physics performance



- Huge improvement over hardware level triggers
- Accomplishes the physics goals defined in the Upgrade Trigger TDR
- Provides throughput headroom for additional algorithms, like ECAL decoding and electron ID

Final thoughts

- Using GPUs will allow LHCb to expand its physics program during Run 3 and beyond
- Taking advantage of GPUs requires redesigning algorithms and data structures
- These design considerations will benefit both GPU- and CPU-based systems

Thank You!