LHCb software modernisation needs

M. Clemencic *on behalf of LHCb*

March 23, 2017

CERN - LHCb

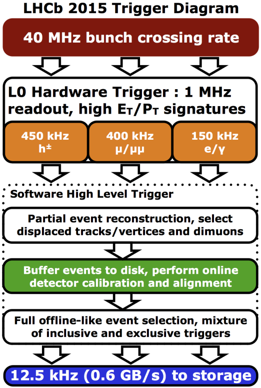
Introduction

- LHCb Software designed $\sim$15 years ago
- It copes with the current workload
- We will need something better for the future

LHCb Upgrade

Status

Run 2 (2015-2018)



LHCb 2015 Trigger Diagram

40 MHz bunch crossing rate

L0 Hardware Trigger : 1 MHz readout, high E_T/P_T signatures

450 kHz $h^\pm$

400 kHz $\mu/\mu\mu$

150 kHz e/γ

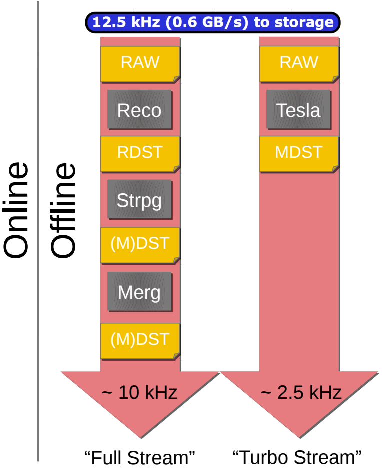
Software High Level Trigger

Partial event reconstruction, select displaced tracks/vertices and dimuons

Buffer events to disk, perform online detector calibration and alignment

Full offline-like event selection, mixture of inclusive and exclusive triggers

12.5 kHz (0.6 GB/s) to storage



Online

Offline

12.5 kHz (0.6 GB/s) to storage

RAW

Reco

RDST

Strpg

(M)DST

Merg

(M)DST

~ 10 kHz

“Full Stream”

RAW

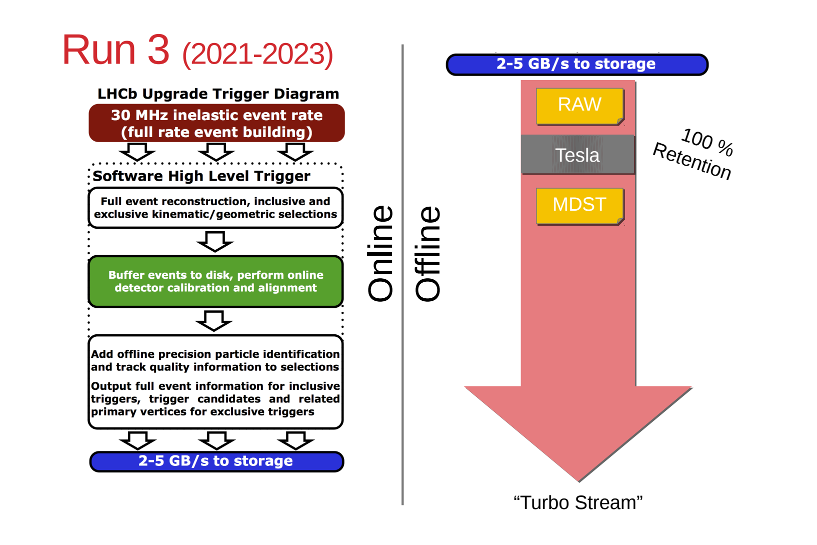
Tesla

MDST

~ 2.5 kHz

“Turbo Stream”

Upgrade



Run 3 (2021-2023)

LHCb Upgrade Trigger Diagram

30 MHz inelastic event rate (full rate event building)

Software High Level Trigger

Full event reconstruction, inclusive and exclusive kinematic/geometric selections

Buffer events to disk, perform online detector calibration and alignment

Add offline precision particle identification and track quality information to selections

Output full event information for inclusive triggers, trigger candidates and related primary vertices for exclusive triggers

2-5 GB/s to storage

Online

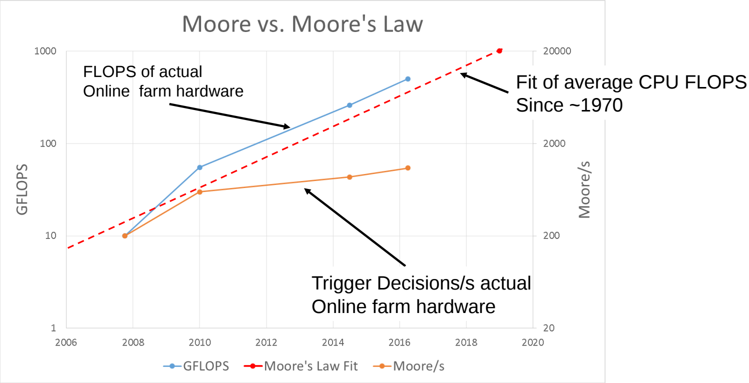
Offline

2-5 GB/s to storage

"Turbo Stream"

Getting Faster

Scaling the Code



FLOPS of actual Online farm hardware

Fit of average CPU FLOPS Since ~1970

Trigger Decisions/s actual Online farm hardware

- Tests done with 2014 Farm tender benchmark
- Dates are release dates of CPU

Areas of Study

We need to improve our software:

- Multithreading
- Vectorization
- Accelerators

Multithreading

- Multithreading in the software framework Gaudi
  - intra/inter-event concurrency
  - task based (TBB)
  - easy thread safety via a *functional* model
  - scheduling optimizations
  - asynchronous I/O
- Adapting LHCb applications
  - migration of reconstruction in progress

Vectorization

- Investigating several possibilities
  - horizontal
  - vertical
  - automatic
  - explicit (libraries)
- Not easy for non experts
  - learning best practices the hard way
  - looking for ways to help developers

Accelerators

- Working on specific tasks
    - demonstrate feasibility
- Not easy integration with the framework
    - how to solve the latency problem?

Getting Better

Improving the Code

- Main code base developed on C++98 standard
- Modern C++ ($\geq$ C++11) helps developers
    - easier to write and understand
    - faster and more correct
- Slowly migrating to *new* best practices
    - follow C++ Core Guidelines

Quality Assurance

- We need code we can trust
    - unit tests
    - integration tests
    - continuous integration
    - static analysis
- Developers need help
    - how to write testable code
    - testing frameworks

Summary

Summary

- Addressing modernization of the code under various angles
- Some aspects require experts' knowledge
    - could it be made easier?
- Distributed computing (Grid) is still important
    - can we optimize and be portable?