Application of new technologies to the LHCb trigger

GAP meeting - Pisa

M.Corvo
S.Gallorini, A.Gianelle, A.Rugliancich

UniFe and INFN Padova

13 Gennaio 2014

The LHCb trigger

L0 hardware trigger

- High p_T muon ($1.4 GeV$) or di-muon
- High p_T local cluster in HCAL ($3.5 GeV$)
- ECAL ($2.5 GeV$)

rate reduction: crossings w/ interactions → 1MHz

HLT1 software trigger

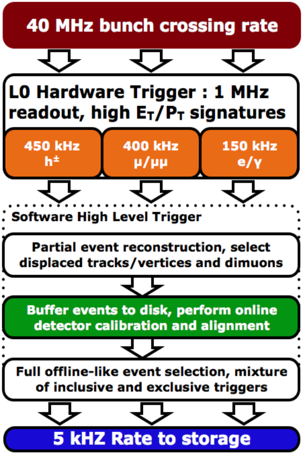
- VELO tracks and primary vertices
- at least one track matching p, p_T, impact parameter

rate reduction: 1MHz → 50kHz

HLT2 software trigger

- Full track reconstruction, w/o particle id.
- 25% of the events are deferred: temporarily stored on disk and processed during the inter-fills

rate reduction: 50kHz → 5kHz



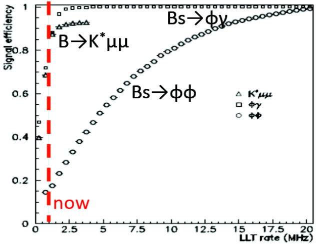
Current LHCb trigger schema

The bottleneck

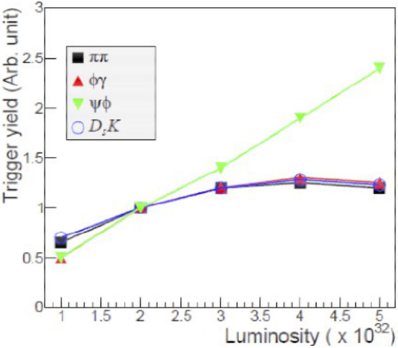
In 2018 LHCb will run at $L = 2 \cdot 10^{33} cm^{-2} s^{-1}$
1 MHz detector readout is a bottleneck, particularly for fully hadronic modes.

Table 3.1: Signal efficiencies for three LLT-accept rates.

LLT-rate (MHz)	1	5	10
$B_s \to \phi\phi$	0.12	0.51	0.82
$B^0 \to K^*\mu\mu$	0.36	0.89	0.97
$B_s \to \phi\gamma$	0.39	0.92	1.00



Trigger efficiency



Trigger Yield

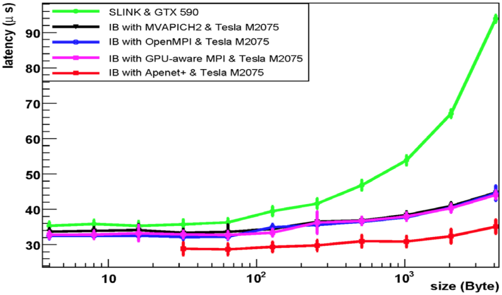
The LHCb trigger upgrade

- DAQ
  - major upgrade, read out each sub-detector @ 40 MHz
- Low Level Trigger (LLT)
  - reduce the rate to a manageable level according to size of online farm
- High Level Trigger (HLT)
  - full event reconstruction
  - offline quality level track reconstruction
  - CPU farm currently, but usage of accelerators (GPU) proposed and under evaluation

The starting point

Our work started using a C simulation of the SVT (the Silicon Vertex Tracker of CDF) hardware behaviour. The aim was to measure

- data transfer latency using different I/O techniques (different transfer protocols w/ and w/o direct access to GPU memory)
- data processing latency on a GPU using a simplified version of SVT



Data transfer latencies

From CDF to LHC

CDF - SVT ($L_{max} = 3 \cdot 10^{32} cm^{-2} s^{-1} @ 2\,TeV$)	Atlas - FTK simulation ($L_{max} = 3 \cdot 10^{34} cm^{-2} s^{-1} @ 14\,TeV$)
Max 768 roads/event	4k-15k roads/event
Max 1500 fits/event	22k-600k fits/event
Max 15 tracks per event	$\sim$ 50 tracks per event

FTK is the Atlas evolution of SVT
We expect that LHCb case is more similar to CDF

Implementation

- Approach to the implementation has been as much conservative as possible
- Optimization and eventually code re-engineering is a further step
  - Step 0 is to evaluate the feasibility of the project
- No black magic with the devices, just a straightforward adaptation of the code
  - For GPUs simple CUDA kernels and for *Intel*® Phi pragma statements to unroll nested `for` loops

Evolution and perspectives

SVT has been a good gym but we need to work now on the real thing. Activities include

- 1:1 porting of FastVelo, the current Velo algorithm, on GPU
  - Algo is mainly divided into two parts: Find Quadruplets (search for track seeds) and Make Space Tracks (complete 3D tracks adding Phi sensors)
  - 15% of computational time is absorbed by Find Quadruplets, 68% by Space Tracks
  - → Try to parallelize these functions
- Use (fast) Hough transform for track finding/fitting
  - Basic algo is straight forward, need to address the problem to a real LHCb event and see how the algo performs

Backup

BACKUP

Why Hough transform

- Hough transform is the simplest, and consequently largely used, algorithm to recognize straight lines
- It's based on a simple principle: every line in the space corresponds to a point in the parameter space, that is the point (a, b) where a is the angular parameter and b the intercept, and viceversa

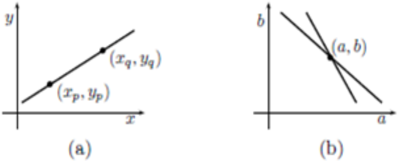


Fig. 1. (a) A line through two points in an image. (b) Corresponding two lines in Hough space.

Basic concept behind Hough transform

Find tracks

- If any two points in the space "share" the same line, the corresponding lines in parameter space will share a point
  - The more points lie on a line, the more the point corresponding to that line "scores" in parameter space
- If we count the scores (local maxima) in the parameter space, we find the tracks in the real space
  - In case of poor resolution, one can make a fit of the values around the alleged maximum

Algo

For simplicity suppose we have a bunch of points (hits) that belong to a single, well defined track:

- A point in the space (x_a, y_a) corresponds to the line with equation

$$b = -x_a \cdot a + y_a \tag{1}$$

  in parameter space
- We fix the boundary and resolution of our angular parameter a and compute a set of b
  - This is equivalent to draw the bundle of lines passing through (x_a, y_a)
- At the very same time we score one point for every couple (a, b)
- Repeat this procedure for every "hit" in our space, the couple (a, b) that has the highest score is our track

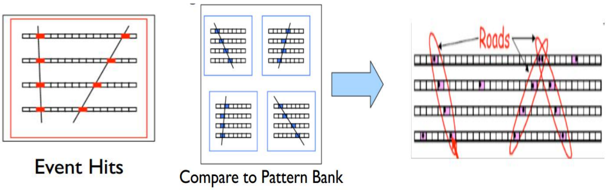
Is it worth it?

- Many, if not all, LHC experiments investigated the usage of Hough transform as a track finding/fitting algorithm so also LHCb is walking the same way.
- The algorithm is not the best candidate for parallelization, as it requires many additions to update the score of every point in parameter space
- Some implementation though proved to be fast enough, wrt CPUs, to make them useful
  - Our current work is based on the implementation by a group of engineers from the Netherlands
- Last, but not least, Hough transform is the baseline to implement the retina algo proposed by people from Pisa.

SVT code

SVT was based on custom hardware and finds displaced vertices in time for a Level-2 decision in two steps

1. Pattern recognition to form hit combinations (called *roads*)
2. Track fitting for every *combination* inside *roads* using simple scalar product $p_i = \vec{f_i} \cdot \vec{x} + q_i$



Our case study focuses on point 2.

SVT code - Track fitting algo

The input is a file containing 24-bit words with hits and associated *roads* of all SVT layers in a specific sector, called wedge. Code performs:

1. Word unpacking to fill data arrays
2. Calculation of all possible *combinations* of hits per *road* (CDF ∼ 64 roads per sector and 12 sectors resulting in max 768 roads/event)
3. Calculation of the fit for each *combination* via $p_i = \vec{f}_i \cdot \vec{x} + q_i$ where $\{\vec{f}_i, q_i\}$ are retrieved from memory (CDF ∼ 2 *combinations* per *road* → max 1500 combinations/event
4. Apply χ^2 cut and format track parameters to be sent to level 2 for trigger decision