Track-pattern recognition on GPGPUs in the LHCb experiment

S. Gallorini
on behalf of the GPU@LHCbTrigger team
University & INFN Padova
GPU Computing in HEP - Pisa - September 11th 2014

Introduction

- The use of commercial General Purpose Graphic Processing Units (GPGPUs) and other many-core architectures (MIC, …) opens up possibilities for **new complex triggers**
- GPGPUs can be used for real-time selection, and may offer a solution for reducing the cost of the High Level Trigger (HLT) farm for the 2020 LHCb upgrade
- Track finding algorithms are usually well suited for parallelization
- ALICE already integrated GPU gaming cards for tracking in the HLT ("Cellular Automaton" algorithm) [IEEE Transactions On Nuclear Science, 58, 4, 2011]

Many-core in LHCb

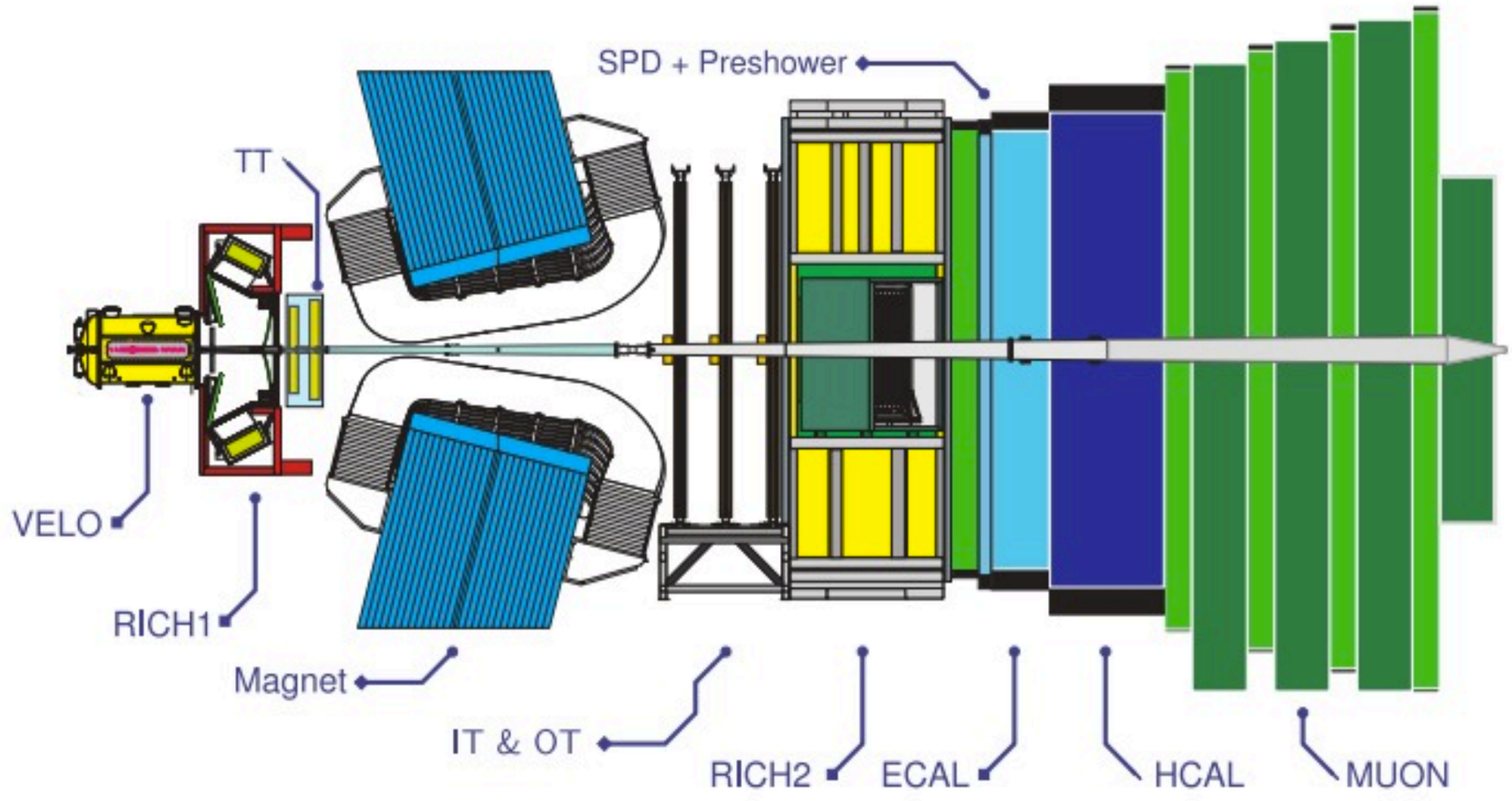
- In LHCb we have a small event size (O(100 kB)) and a relatively short processing time

  ➡ A huge speed-up could be obtained by processing **many events in parallel**

- Several activities started in LHCb aiming to use many-core architectures in the HLT
  (see Daniel's talk "Manycore feasability studies at the LHCb trigger")

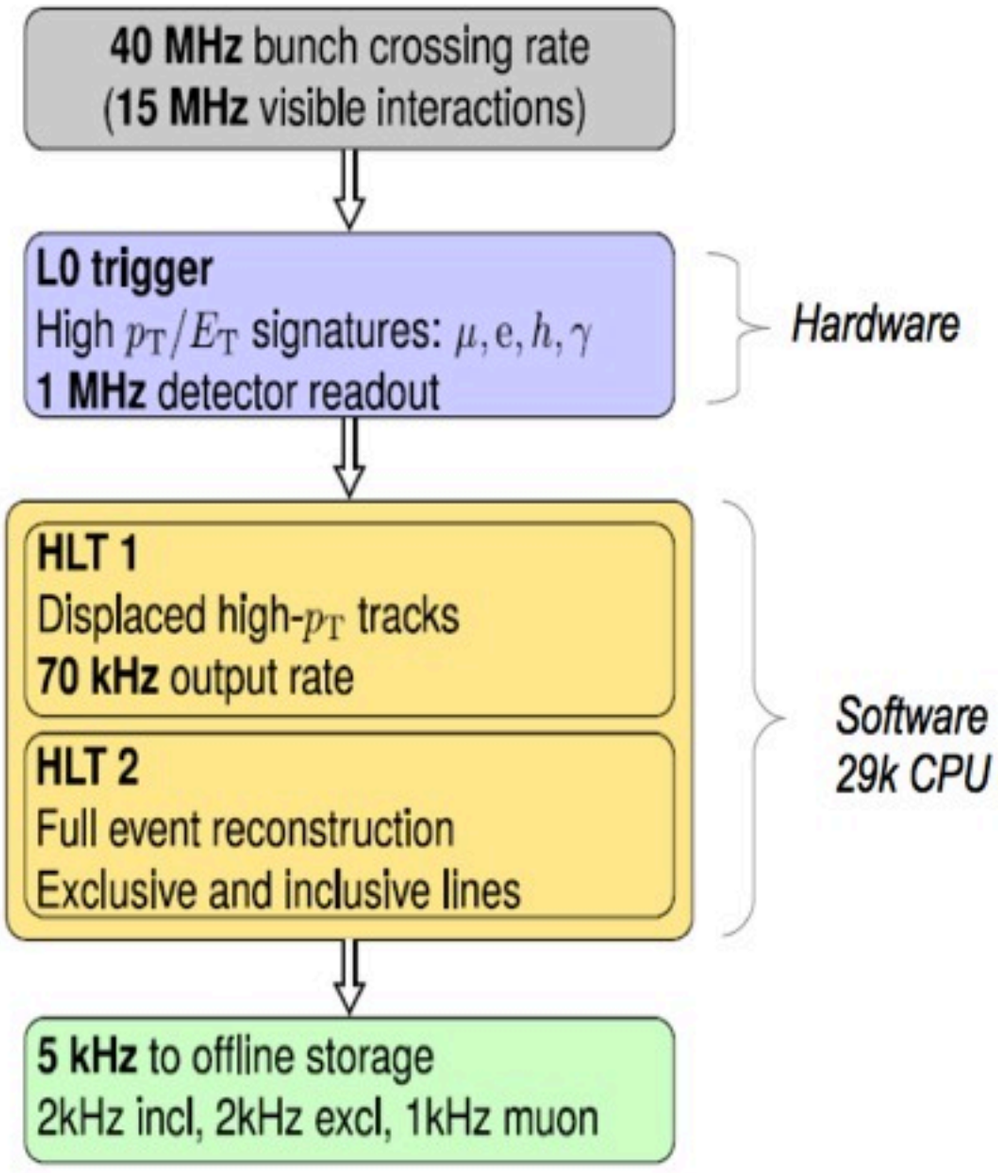
- In this talk, I will focus on the study of tracking algorithms on GPU for the current VErtex LOcator detector (VELO)

The LHCb detector



- LHCb is a single-arm forward spectrometer at the LHC aiming at precision beauty and charm physics:
  - CP violation, rare decays, heavy flavour production
- An highly efficient trigger is required for selecting pure data samples for rare decays

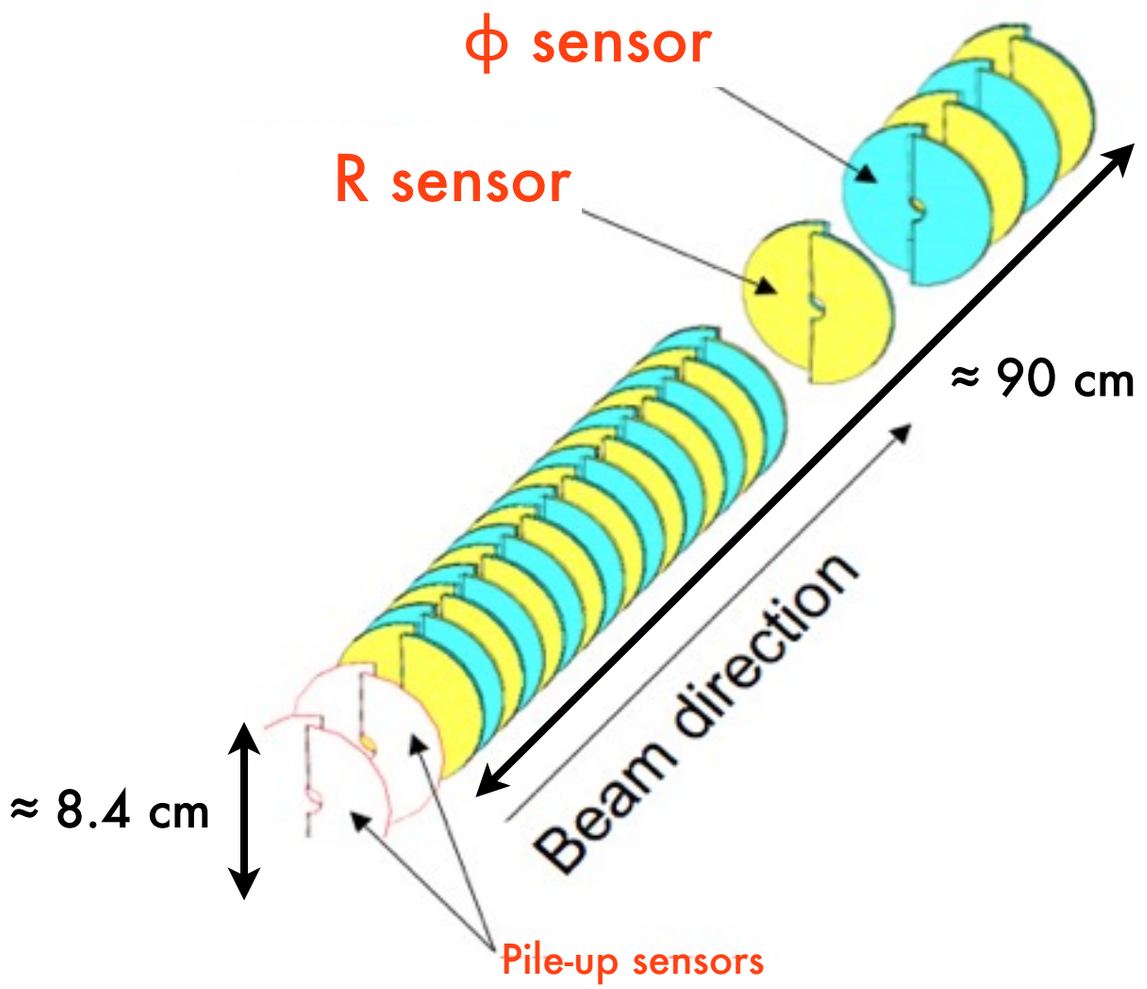
The LHCb trigger system

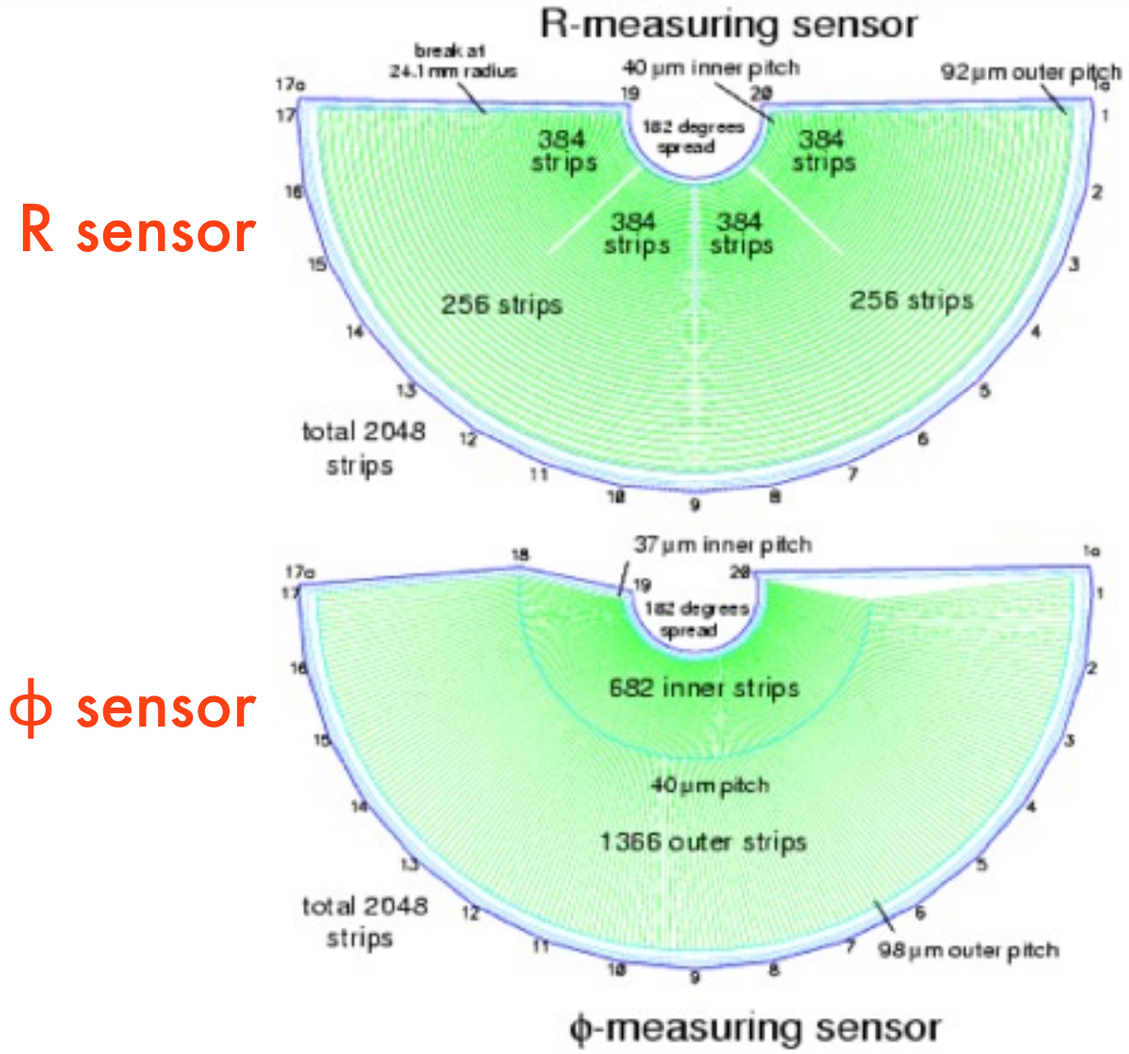


- **L0** reduces the rate to below 1.1 MHz:
  - Input from the calorimeter and muon systems
  - Read-out decision in 4μs
- **HLT1** reconstructs:
  - Tracks in the vertex detector (VELO)
  - Primary vertices
  - Forward tracks to tracking detectors downstream the magnet
- **HLT2** fully reconstructs the event:
  - Performance close to offline reconstruction

- The available resources in the today Event Filter Farm limited the time per event in the HLT to ≈30 ms

The VELO detector





- The VELO detector is a silicon micro-strip detector situated close to the interaction region
- R-ϕ layout, 21 stations with 2R and 2ϕ sensors each (+ 4 pile-up sensors)
- It provides precise and fast tracking information which was employed in the HLT during Run1 (2011-2012)

VELO pattern recognition

- "FastVelo" is the algorithm developed by LHCb for pattern recognition in the VELO

- It ran online in the HLT farm during Run1:
  - Written to be fast and highly efficient to cope with high rate and hit occupancy
  - Several conditions and checks introduced throughout the code to speed up the execution

- The VELO track reconstruction is done in two steps:

1. **RZ tracking:** only hits on R sensors are used. Find tracks in the R-Z plane.
2. **Space tracking:** add the information of ϕ sensors to each to build the full tracks

FastVelo on GPU (1)

- The goal of this work is to:
  - Evaluate the performances of FastVelo (running in HLT1) on GPU wrt the original code (**optimized** for CPU)
    - Timing and tracking efficiencies (e.g. clone and ghost rates, efficiency for long tracks)
  - Test GPU tracking algorithms in a parasitic mode in the HLT during the Run2 starting in 2015

Useful definitions:

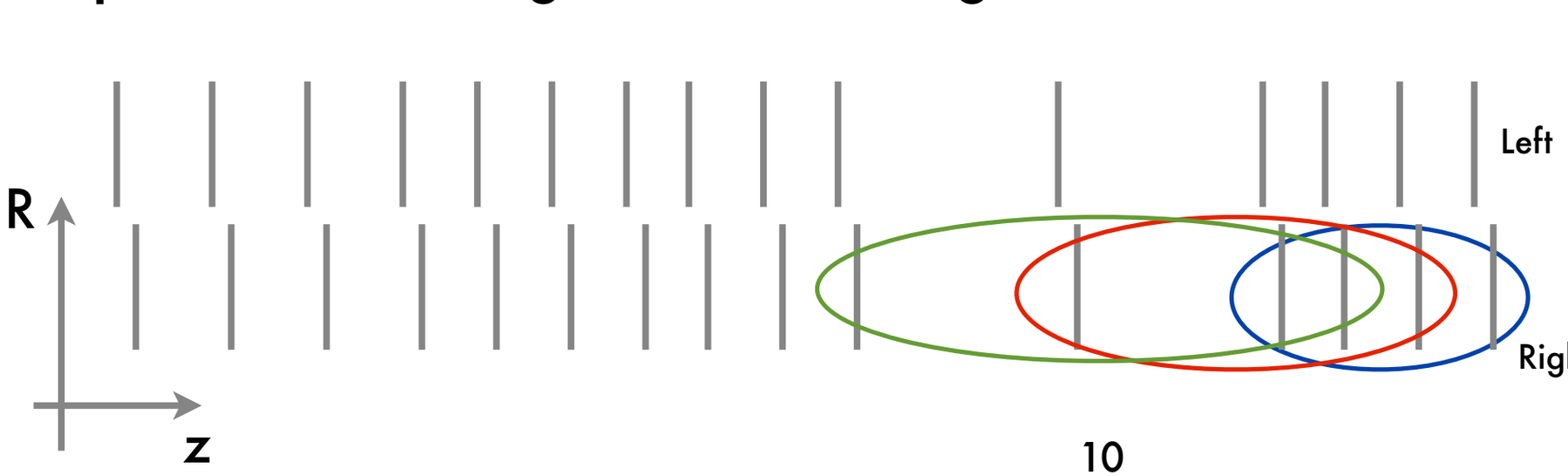
- efficiency: $\frac{N_{reconstructed\ \&\ reconstructible\ particles\ \&\ no\ electrons}}{N_{reconstructible\ particles\ \&\ no\ electron}}$
- ghost track: reconstructed track not matched to any true particle
- clone tracks: tracks associated to the same true particle
- long track: track reconstructed in VELO and in tracking stations ("T-stations")

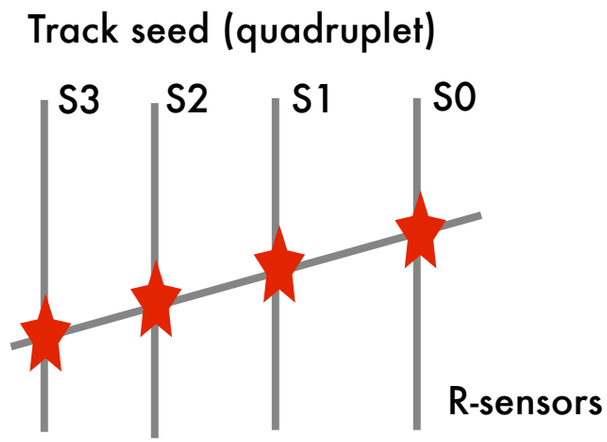
FastVelo on GPU (2)

- **Strategy:**
- Parallelize on the events (obvious…)
- Parallelize the algorithm:
  - Process each RZ track concurrently:
    - In the original algorithm hits already used in a track are marked and not further considered in the following iterations (“hit tagging”)
    - To avoid race-conditions, **hit tagging** must be **removed** in the GPU algorithm (clones and ghosts tracks diverge!)
- For the rest… try to keep the GPU version as closest as possible to the original one (code written in CUDA)

FastVelo on GPU (4)

- **RZ tracking:**
- Only R-sensors are used
- The algorithm looks for quadruplets of hits in four contiguous R-sensors (seed) on both halves.
  - Each thread works on a set of four contiguous R-sensors and find all quadruplets.
- Then each quadruplet is extended in parallel as much as possible adding the remaining R-sensors





FastVelo on GPU (5)

- **Space tracking:**

- Add hits on ϕ-sensors

- Each RZ track is processed concurrently by assigning a space-tracking algorithm to each thread:

  - Search for a triplet of hits: for each hit in the first two ϕ-sensors, the candidate hit in the third sensor is the one most compatible with predicted position (best χ^2)

  - The track is extended and its parameters are found by minimizing $\Sigma_{points}\ \chi^2$ (linear system solved by substitution)

  - This part is almost a re-writing in CUDA of the original space-tracking code

FastVelo on GPU (6)

- **Main issue:**

- Without tagging on used hits, we end up with a large amount of **clones and ghosts**

- **Solution:**

- **"Clone killer"** algorithms are needed throughout the GPU code to reduce clones and the number of tracks

- All pairs of tracks are checked in parallel:

  - Each thread of the clone killer algorithm takes a track and computes the number of hits in common with the others; if two tracks have more than 70% of hits in common, the one with worst χ^2 is discarded

Performance evaluation

- **GPU:** NVidia Titan (14 SM, 192 CUDA cores/SM, 6GB of memory)

- **CPU:** a **single core** (Intel i7, 3.40 GHz) in the same PC hosting the GPU & a **multicore CPU** (Intel Xeon E5-2600, 24 cores w/ Hyper Threading, @CNAF)

- Used data samples:
  - $B_s \rightarrow \phi\phi$ MC events and MinBias data
  - b-inclusive MC events (simulated with 2015 data taking conditions)

- We use standard LHCb tools to get track efficiencies and resolutions

- **Tracking time** only! (data transfer **not included!**)

Results (1)

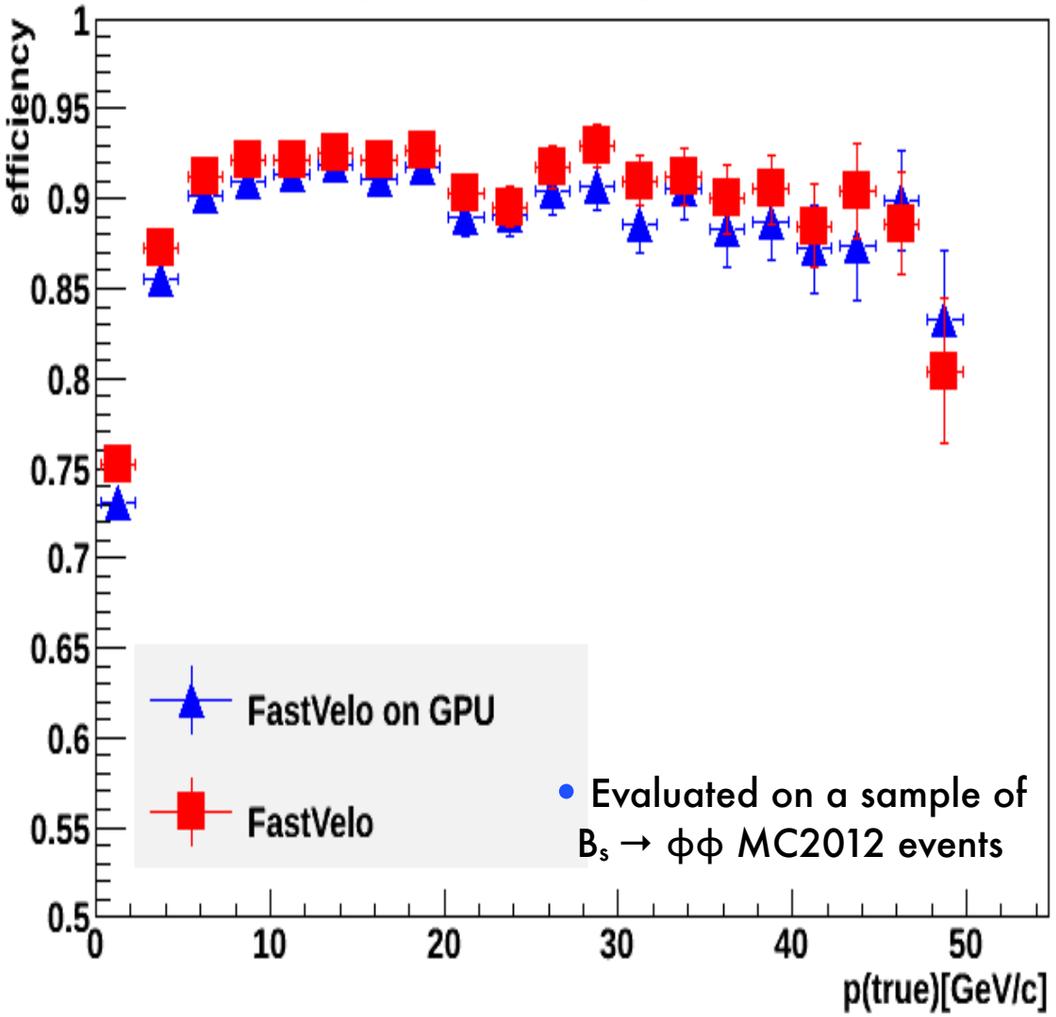
- Tracking efficiencies comparison:

- FastVelo running in HLT1
- Evaluated on a sample of $B_s \rightarrow \phi\phi$ MC events

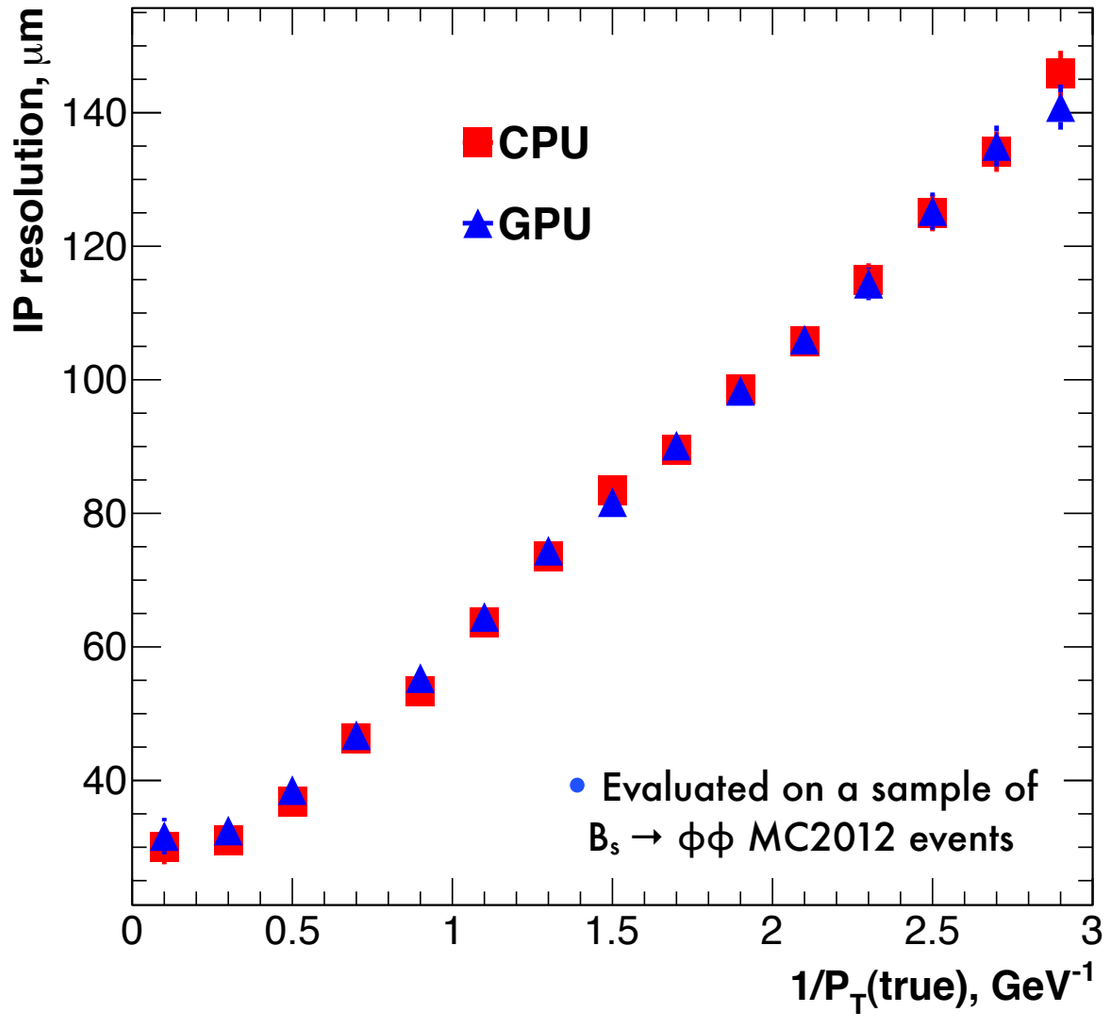
Track category	FastVelo on GPU		FastVelo on CPU	
	Efficiency	Clones	Efficiency	Clones
VELO, all long	86.6%	0.2%	88.8%	0.5%
VELO, long, p > 5 GeV	89.5%	0.1%	91.5%	0.4%
VELO, all long B daughters	87.2%	0.1%	89.4%	0.7%
VELO, long B daughters, p > 5 GeV	89.3%	0.1%	91.8%	0.6%
VELO, ghosts	7.8%		7.3%	

Results (2)

Tracking efficiency vs P(true)



Impact parameter resolution vs $1/P_T$(true)

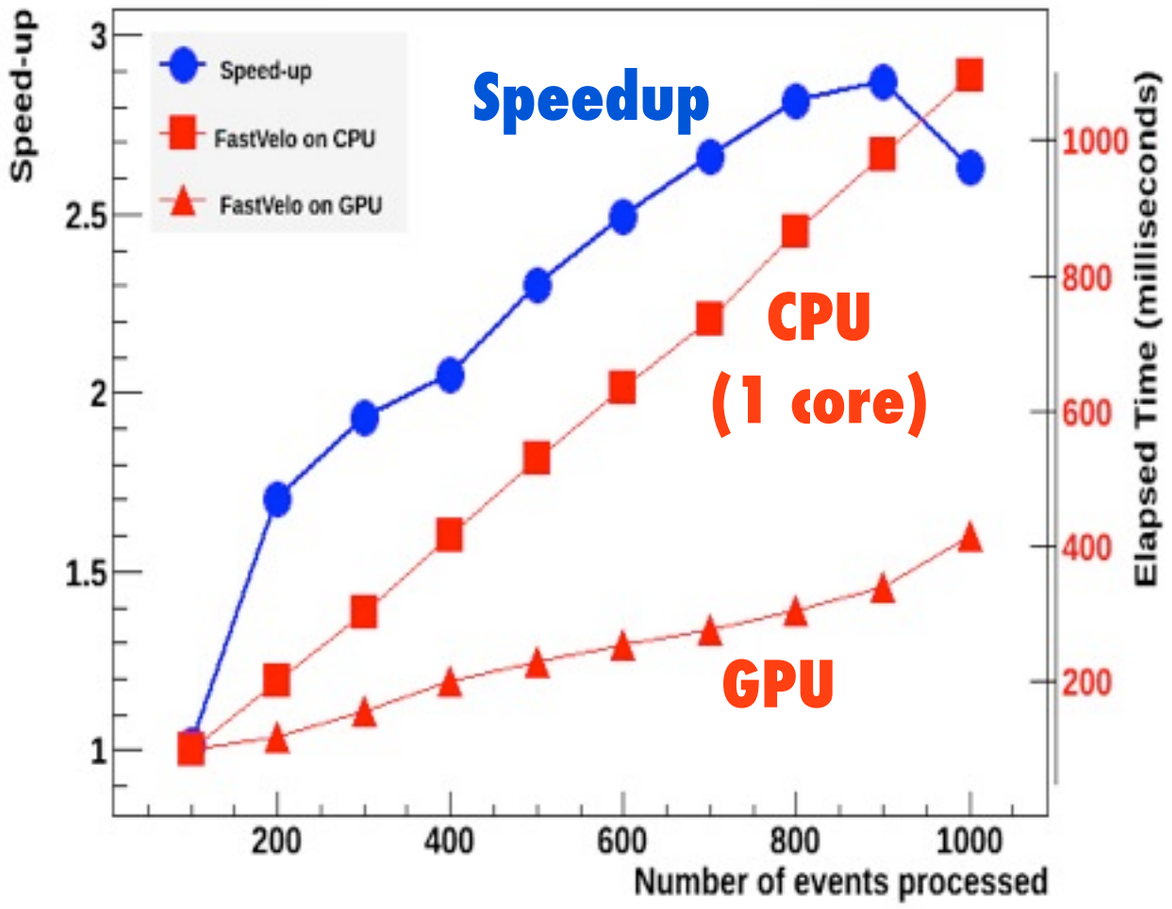


Tracking performances close to the optimized CPU code

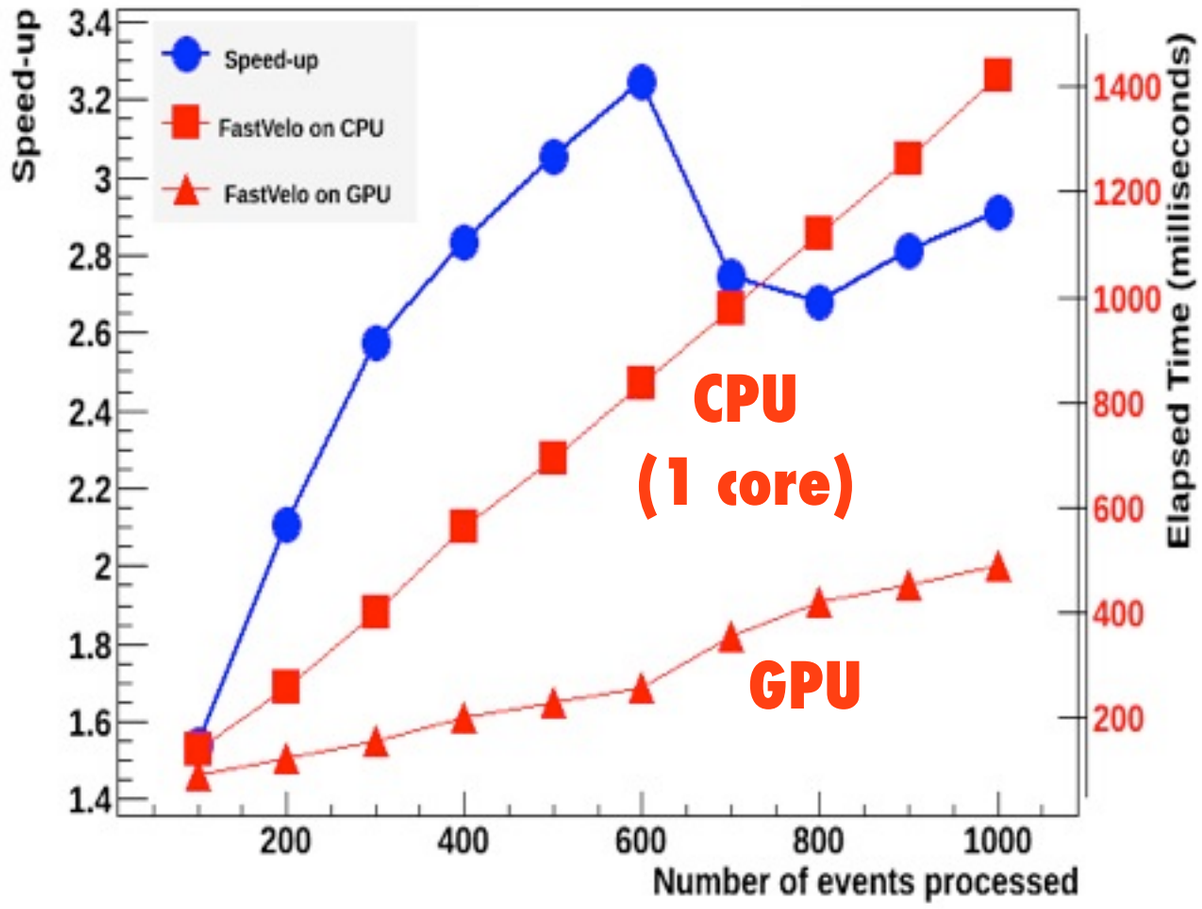
Results (3)

- Execution time as a function of the number of events

$B_s \rightarrow \phi\phi$ MC events



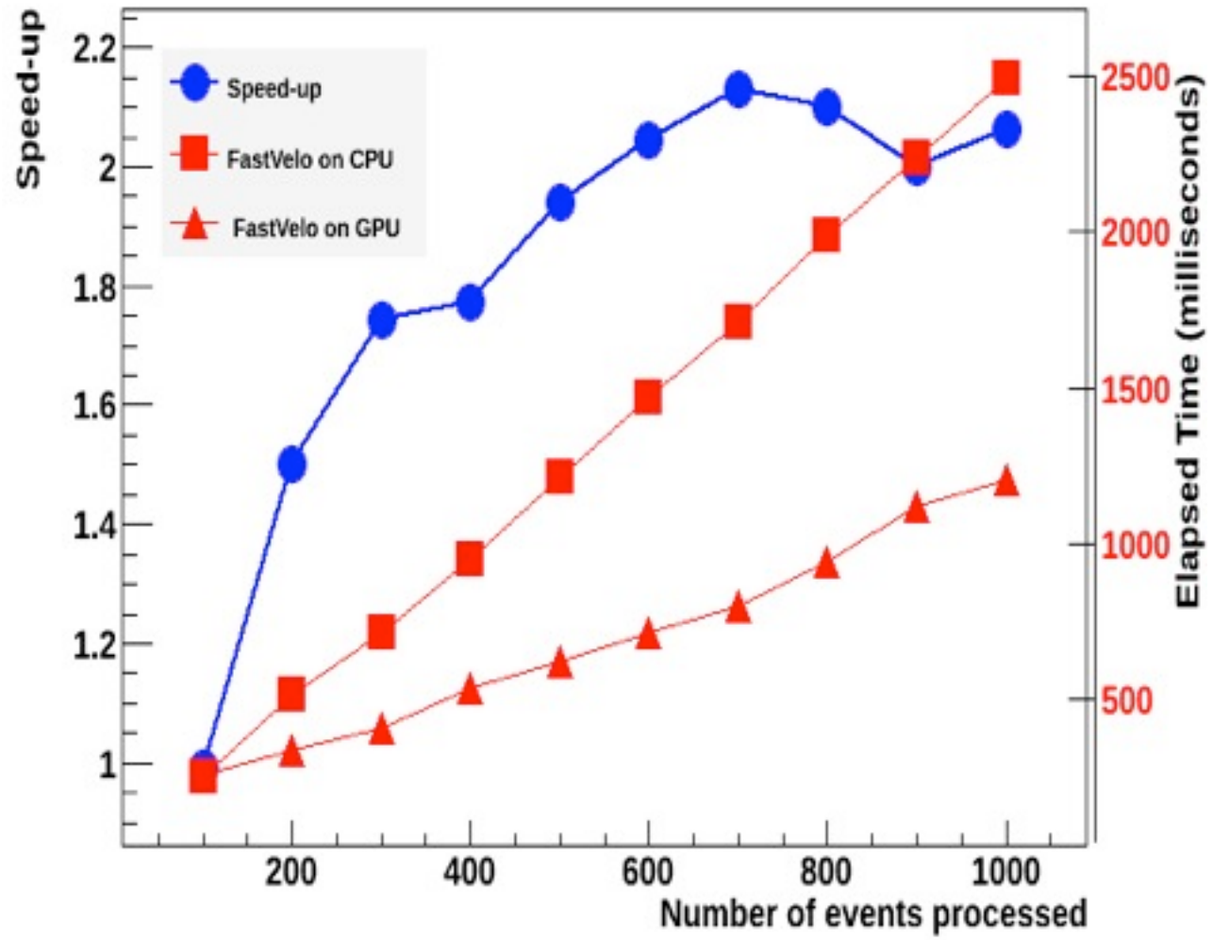
NoBias data



- GPU performances **increases** with the number of events: GPU resources are more efficiently used as the number of events increases (more threads running at the same time)

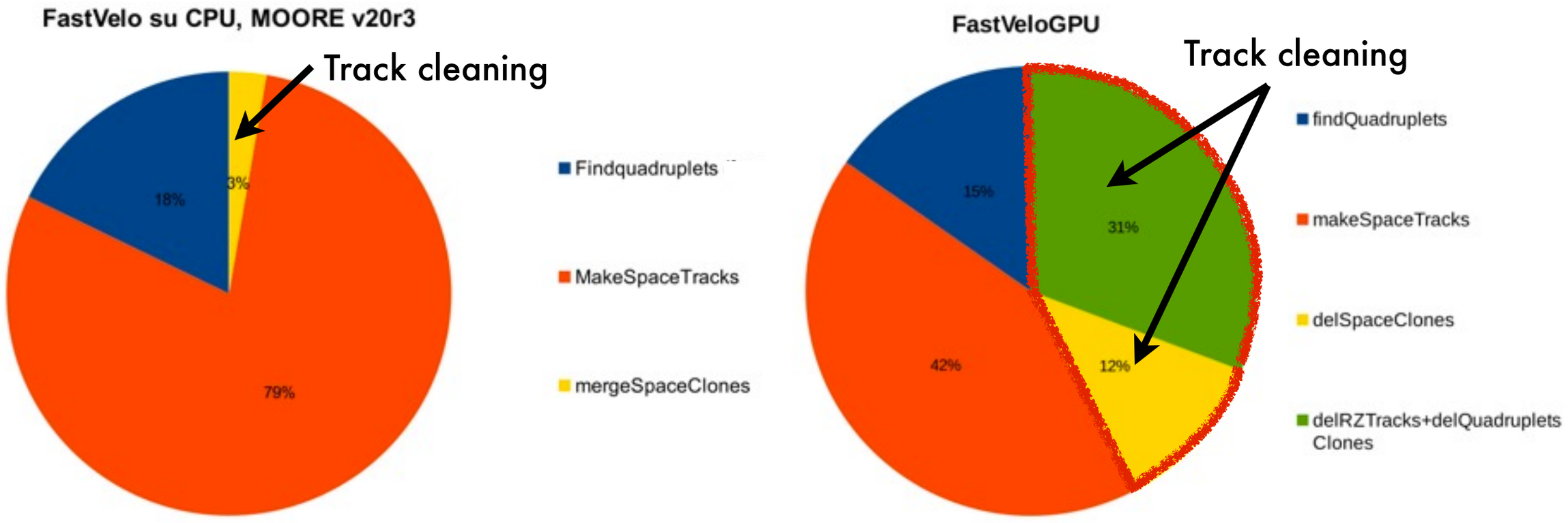
Results (4)

b-inclusive MC events simulated with expected 2015 conditions



Results (5)

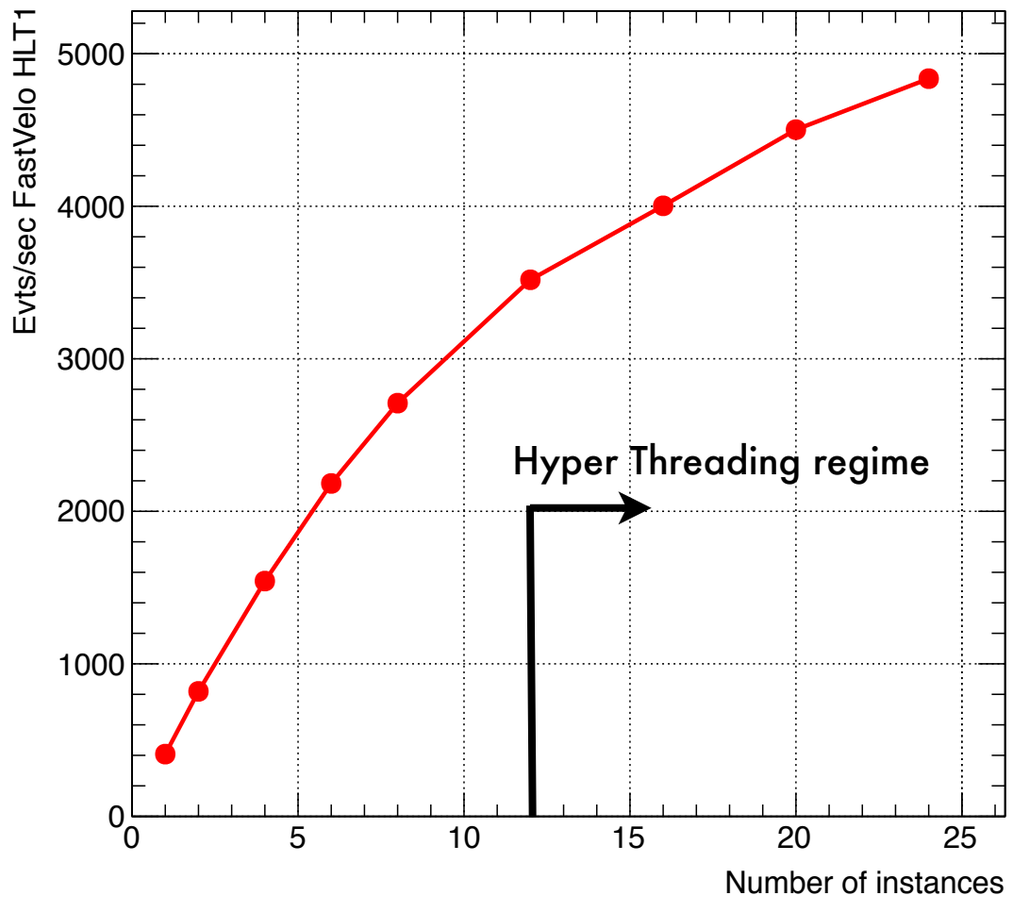
- Execution times



- The time spent by FastVeloGPU to kill clones and ghosts is **≈40%** of whole algorithm (wrt **≈3%** of the original code)

Results (6)

- GPU performances compared to a **multicore CPU** (24 cores w/ HT)
  - An instance of FastVelo sent to each core at the same time, with each job processing the same number of events (1000 events/job)



- **The throughput** of a single core **decreases** if more instances are running in parallel

- Rate of processed events:

≈5000 evts/sec (CPU) vs **≈2600** evts/sec (GPU)

Conclusions

- Preliminary results on VELO tracking on GPU have been shown
- Tracking performances of the GPU version close to the original CPU code
- A better performance estimator is the rate of processed events normalized to the cost of the hardware!
  - The GPU gaming-card cost less than a CPU used in a node of the HLT farm, so also a moderate speed-up (e.g. 2x) compared to e.g. Intel Xeon could bring a real saving to the experiment

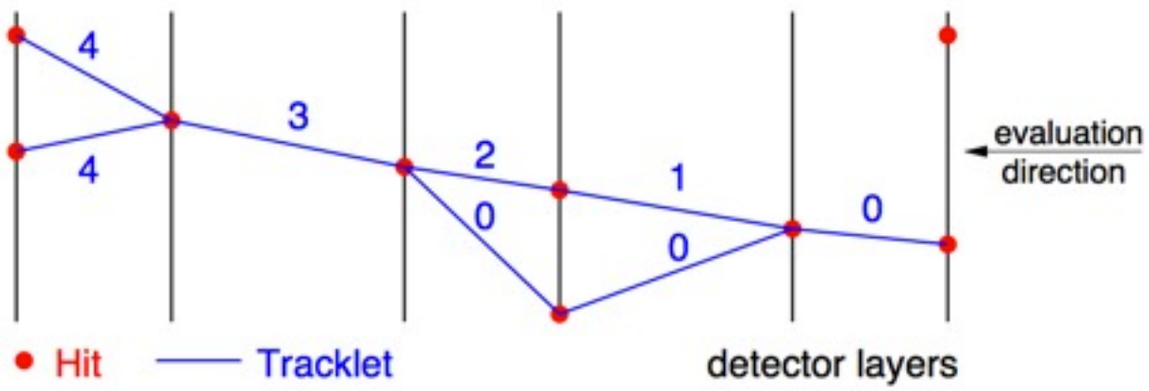
Outlook (1)

- Improve VELO tracking performances (improve memory coalescing, try new algorithms, …)

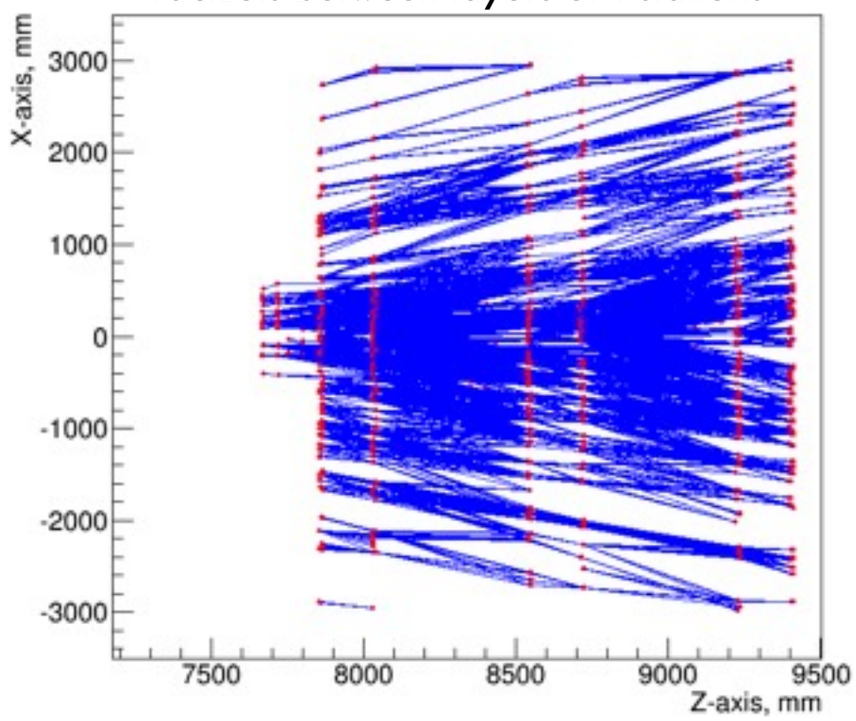
- Add hits on the T-stations. Two approaches are on-going:
  - Porting of the original algorithm which extends VELO tracks to the forward direction by adding T-hits (“Forward tracking”)
  - Find tracks on T-stations using Cellular Automata and matching with VELO tracks
  - Other algorithms (Kalman filter, …)

Outlook (2)

Cellular Automata



Tracklets between layers of T-stations



- Test HLT1 (HLT2?) tracking algorithms on GPUs in a parasitic mode during the Run2 starting in 2015
  - Setup one or more PC with GPUs in the HLT farm
  - Setup SW framework (GPUManager)
  - Measure latency
  - …

Backup slides