"GPU for triggering at Level0 in NA62 experiment"

Software Tech Forum

CERN 27.4.2016

Gianluca Lamanna (INFN)

On behalf of GAP collaboration

Outline

- GPU in future low level trigger
  - Can GPUs be used in real-time selection?
- A physics case: NA62
  - The real life
- Control the Latency
  - Minimize copy back and forth
- High throughput for ring reconstruction
  - The power of the parallel

Next generation trigger

- Next generation experiments will look for tiny effects:
  - The trigger systems become more and more important
- Higher readout band
  - New links to bring data faster on processing nodes
- Accurate online selection
  - High quality selection closer and closer to the detector readout
- Flexibility, Scalability, Upgradability
  - More software less hardware

Different Solutions

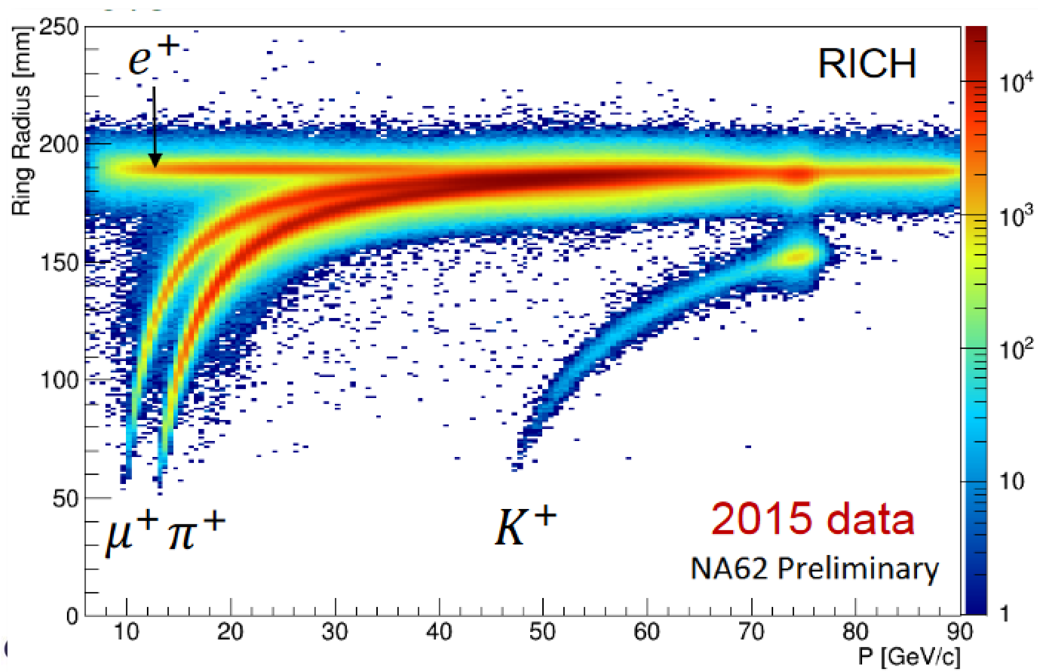
- Brute force: PCs
  - Bring all data on a huge pc farm, using fast (and eventually smart) routers.
  - Pro: easy to program, flexibility; Cons: very expensive, most of resources just to process junk.
- Rock Solid: Custom Hardware
  - Build your own board with dedicated processors and links
  - Pro: power, reliability; Cons: several years of R&D (sometimes to re-rebuild the wheel), limited flexibility

- Elegant: FPGA
  - Use a programmable logic to have a flexible way to apply your trigger conditions.
  - Pro: flexibility and low deterministic latency; Cons: not so easy (up to now) to program, algorithm complexity limited by FPGA clock and logic.
- Off-the-shelf: GPU
  - Try to exploit hardware built for other purposes continuously developed for other reasons
  - Pro: cheap, flexible, scalable, PC based. Cons: Latency

GPU in low level trigger?

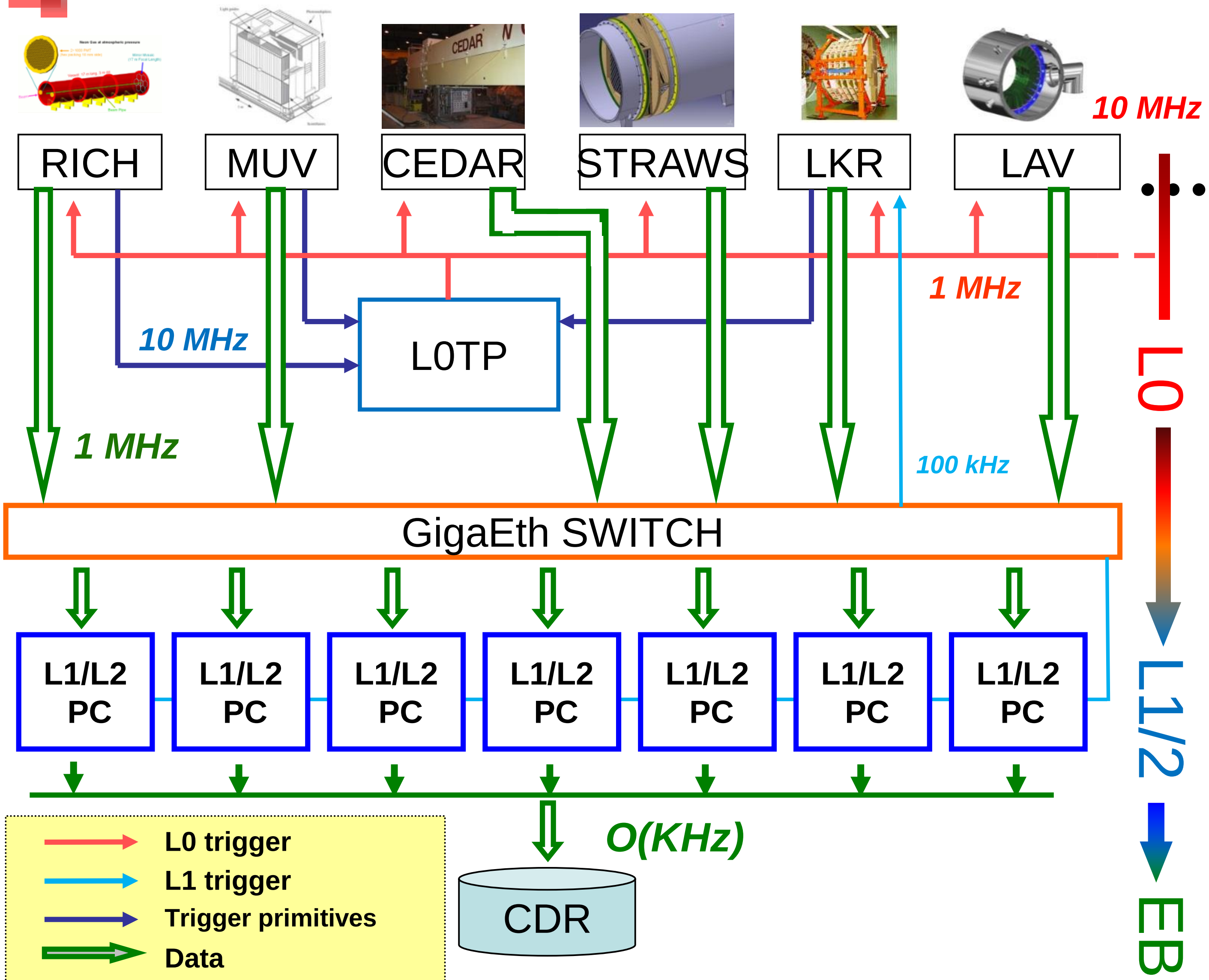
- Latency: Is the GPU latency per event small enough to cope with the tiny latency of a low level trigger system? Is the latency stable enough for usage in synchronous trigger systems?

- Computing power: Is the GPU fast enough to take trigger decision at tens of MHz events rate?



- **RICH:**
  - 17 m long, 3 m in diameter, filled with Ne at 1 atm
  - Reconstruct Cherenkov Rings to distinguish between pions and muons from 15 to 35 GeV
  - 2 spots of 1000 PMs each
  - Time resolution: 70 ps
  - MisID: 5×10^{-3}
  - 10 MHz events: about 20 hits per particle

NA62: Standard Trigger system

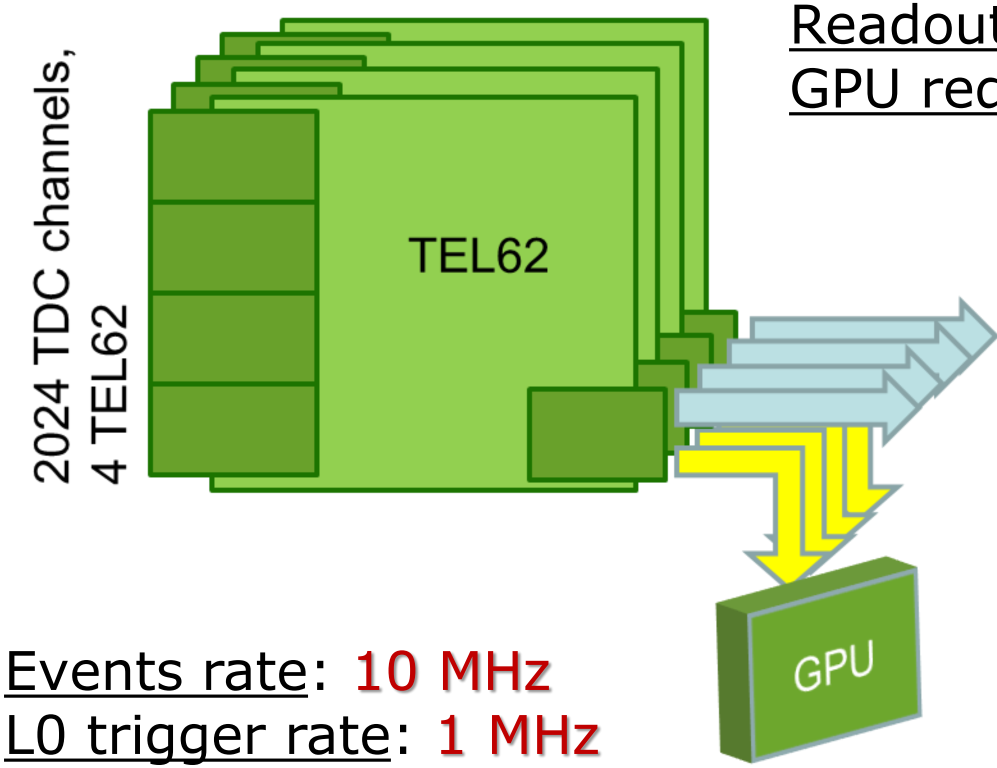


L0: Hardware synchronous level. 10 MHz to 1 MHz. Max latency 1 ms.

L1: Software level. "Single detector". 1 MHz to 100 kHz

L2: Software level. "Complete information level". 100 kHz to few kHz.

NA62 GPU trigger system



Readout event: 1.5 kb (1.5 Gb/s)
GPU reduced event: 300 b (3 Gb/s)

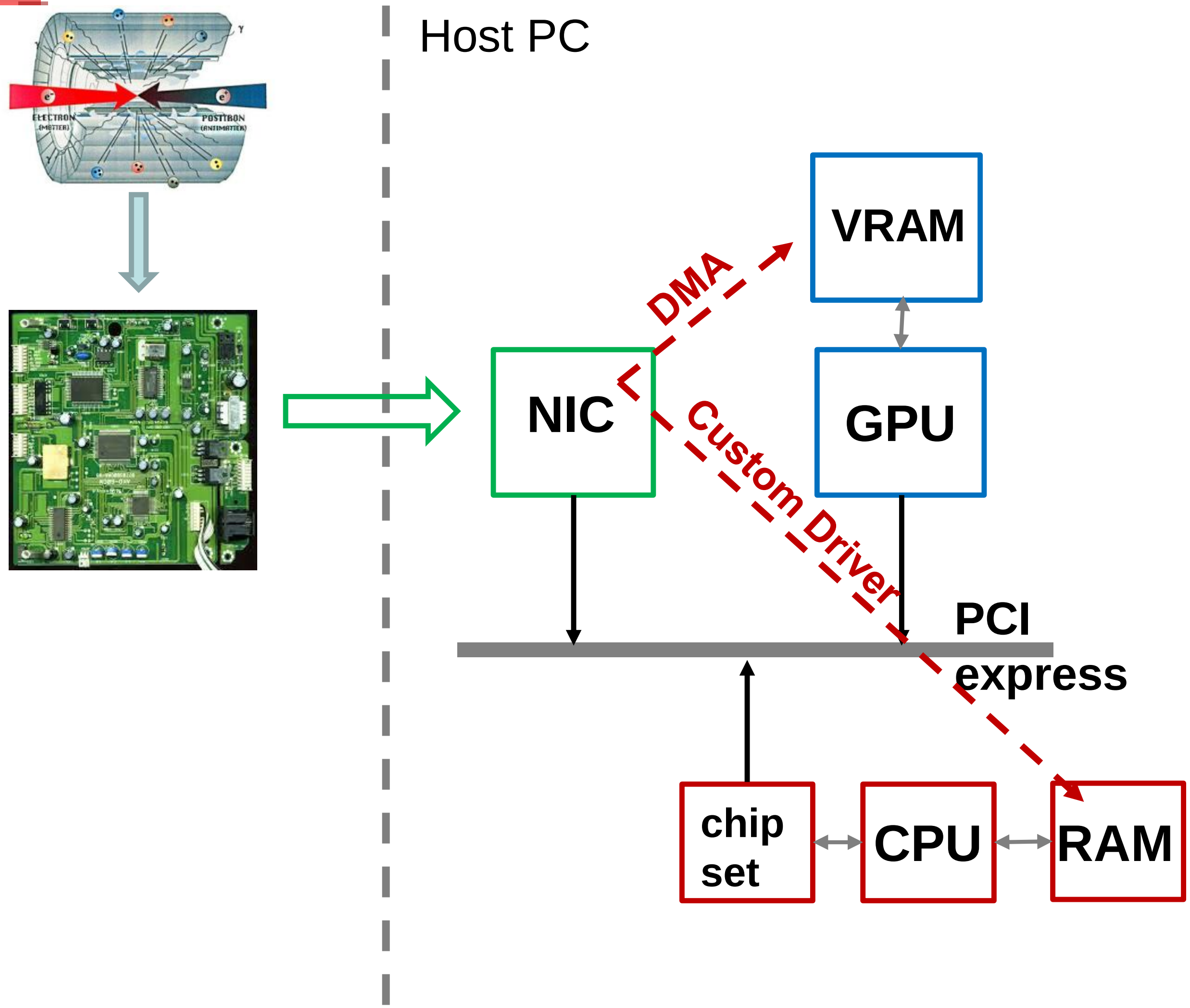
8x1Gb/s links for data readout 4x1Gb/s Standard trigger primitives
4x1Gb/s GPU trigger

GPU NVIDIA K20:

- 2496 cores
- 3.5 Teraflops
- 5GB VRAM
- Bandwidth: 208 GB/s

Events rate: 10 MHz
L0 trigger rate: 1 MHz
Max Latency: 1 ms
Total buffering (per board): 8 GB
Max output bandwidth (per board): 4 Gb/s

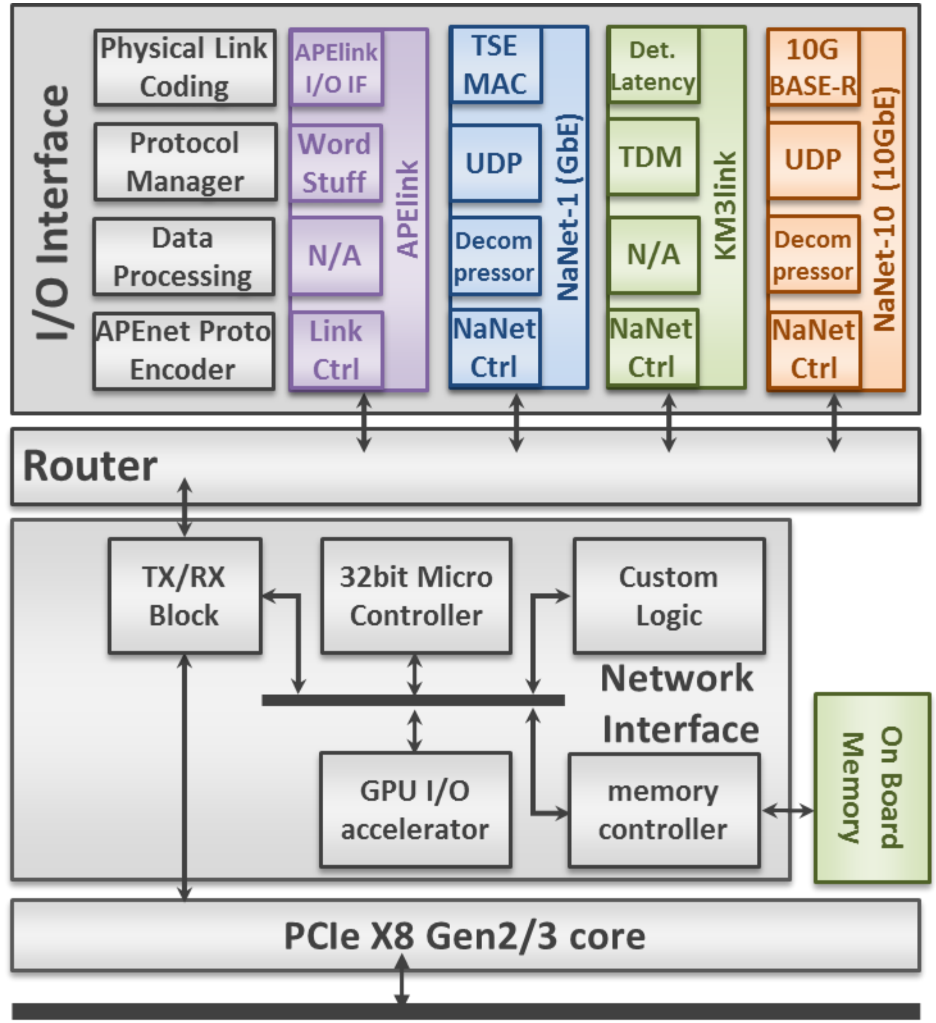
Latency: main problem of GPU computing



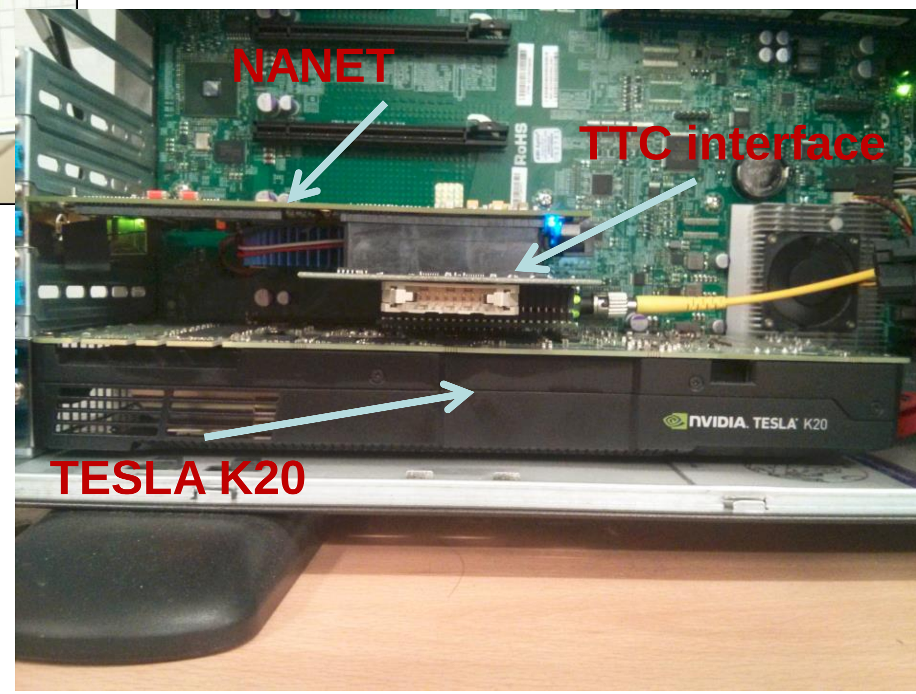
- Total latency dominated by double copy in Host RAM
- Decrease the data transfer time:
  - DMA (Direct Memory Access)
  - Custom manage of NIC buffers
- *"Hide"* some component of the latency optimizing the multi-events computing

Nanet-1 board

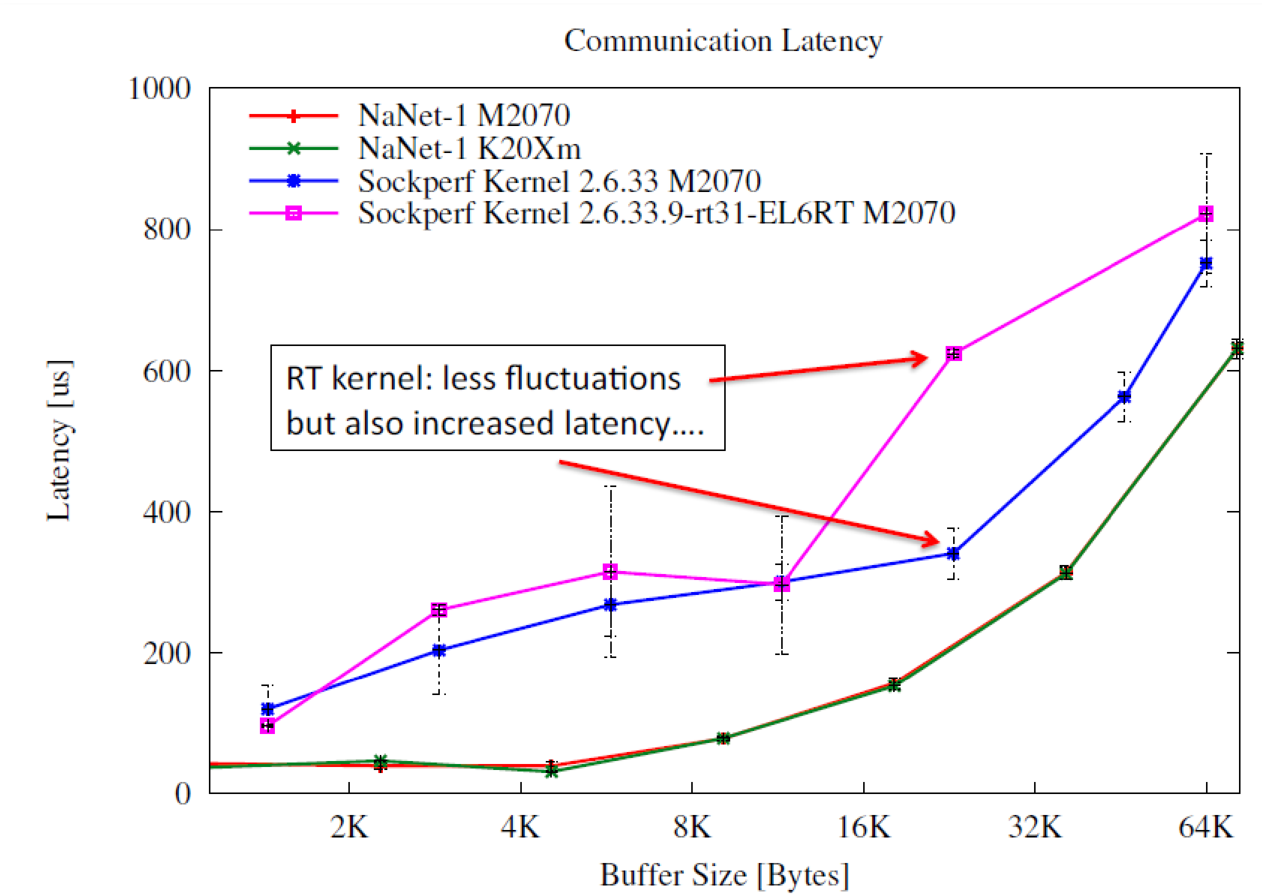
- **Nanet-1:** board based on the ApeNet+ card logic
  - PCIe interface with GPU Direct P2P/RDMA capability
  - Offloading of network protocol
  - Multiple 1Gb/s link support
  - Use FPGA resources to perform on-the-fly data preparation



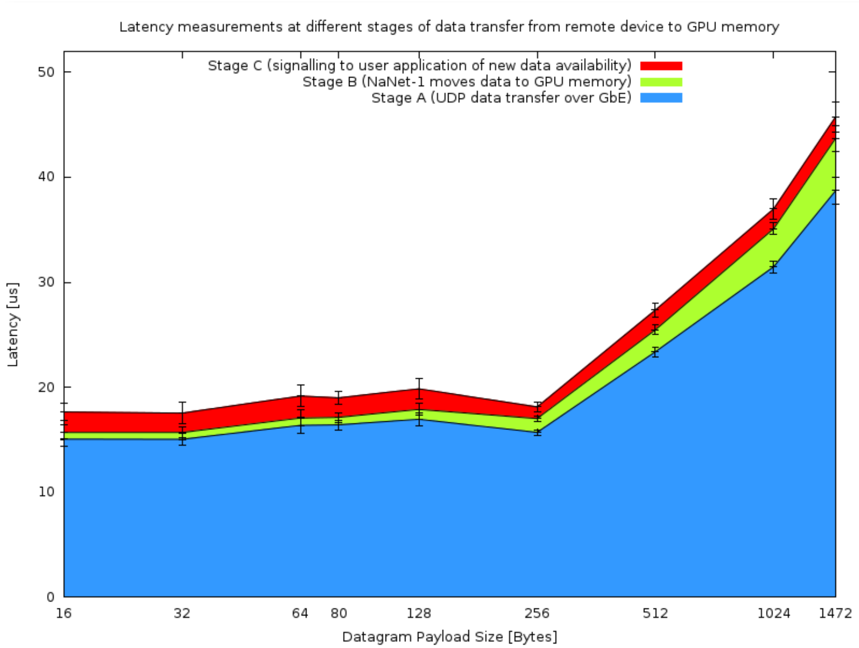
Nanet-1 in NA62

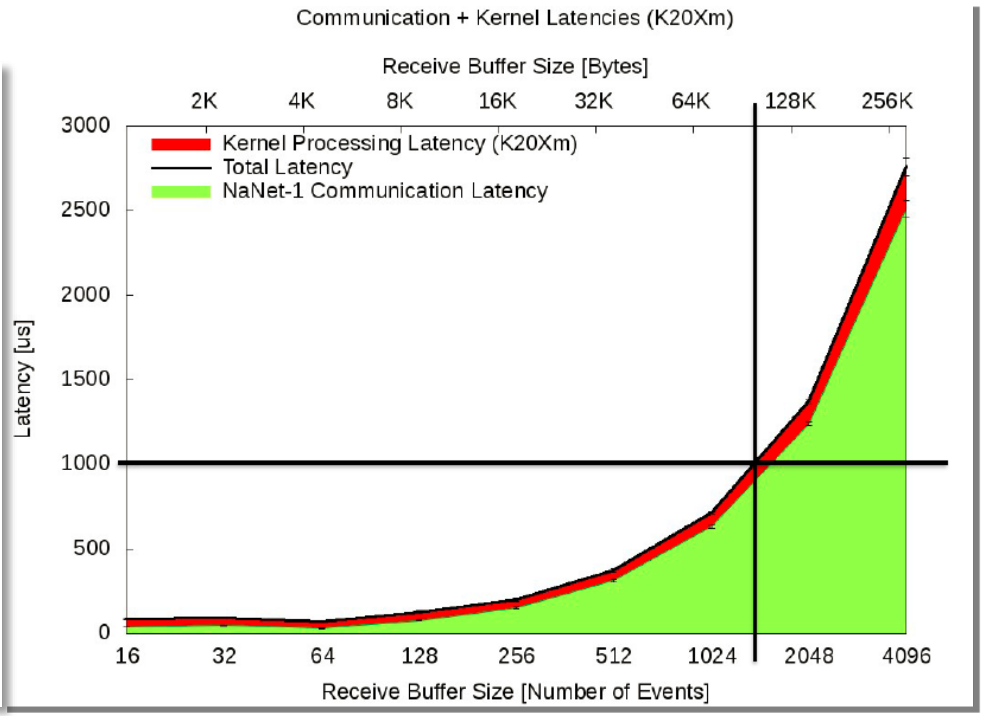


Nanet-1: Performances



Nanet-1: Performances

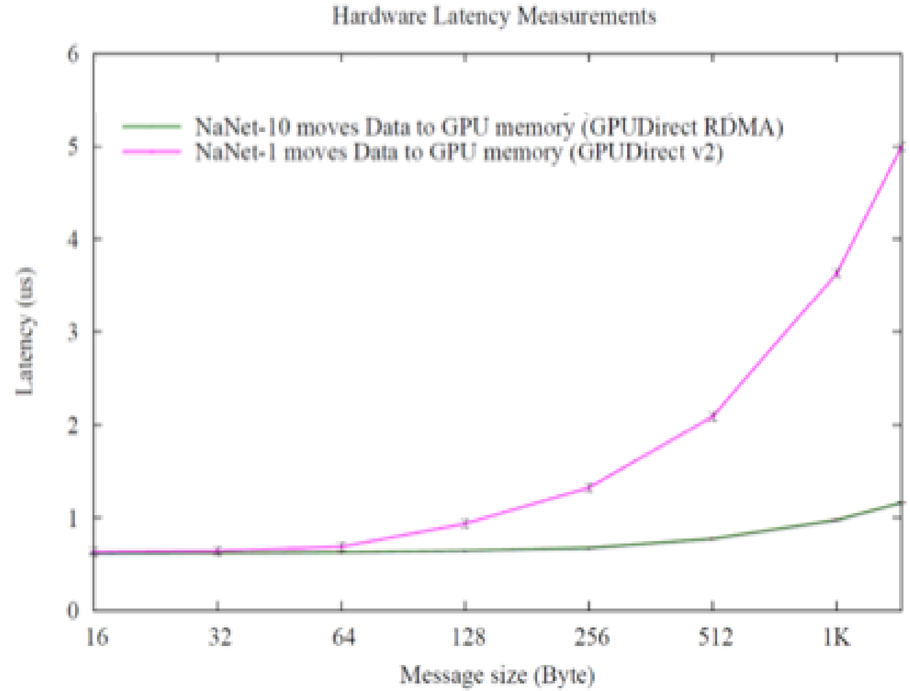


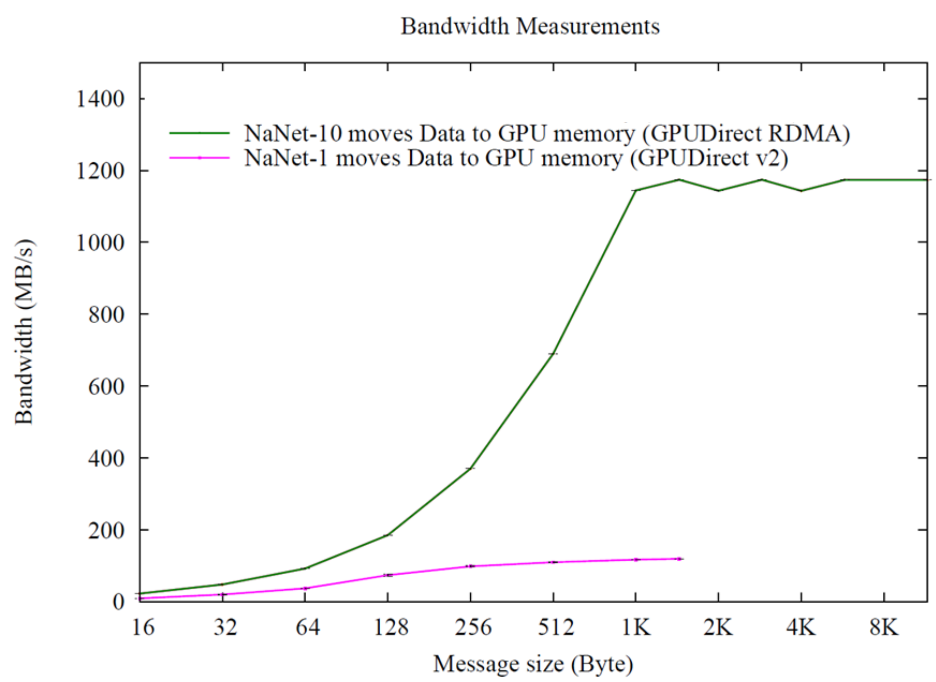


After NANET latency if fully dominated by GbE transmission.

Nanet-10

- ALTERA Stratix V dev board (TERASIC DE5-Net board)
  - PCIe x8 Gen3 (8 GB/s)
  - 4 SFP+ ports (Link speed up to 10Gb/s)
- GPUDirect /RDMA capability
- UDP offloads supports
- FPGA preprocessing (merging, decompression, …)



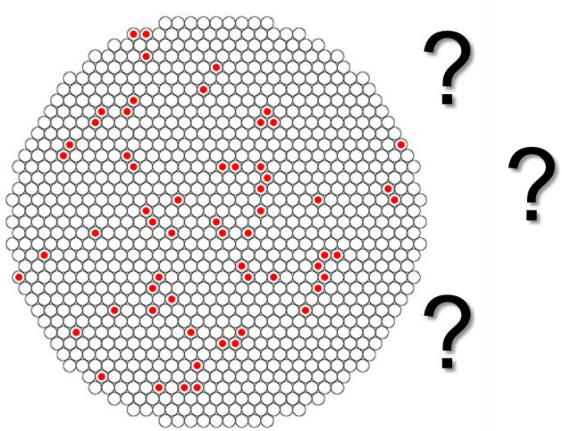


DMA in Heterogeneous computing

- Is it the DMA important for standard (not real-time) GPGPU?
  - The DMA enables the CPU to keep on working concurrently on other task while long lasting memory operations take place; considerably boosting overall system performance. This is considerably true in applications where I/O is relevant (Reconstruction, HLT, …)
  - The DMA allows to imagine a smart networking based on PCs instead of switches (Data Acquisition, Clusters,…)

Ring fitting

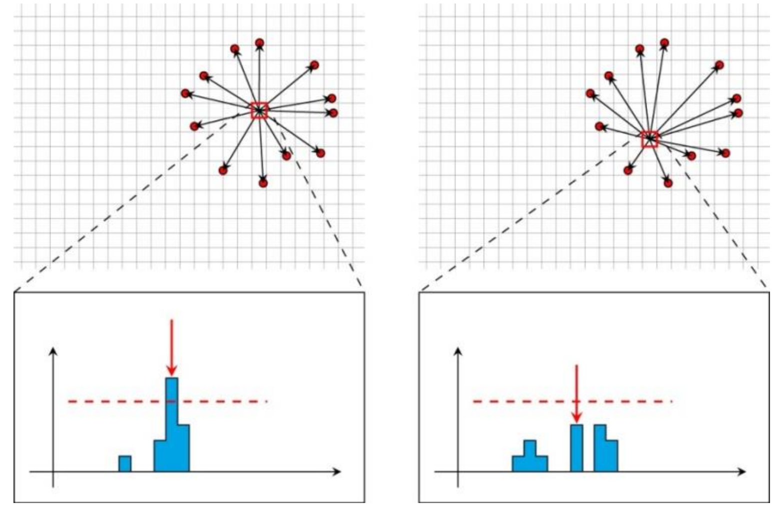
- Trackless
  - no information from the tracker
  - Difficult to merge information from many detectors at L0
- Fast
  - Not iterative procedure
  - Events rate at levels of tens of MHz
- Low latency
  - Online (synchronous) trigger
- Accurate
  - Offline resolution required

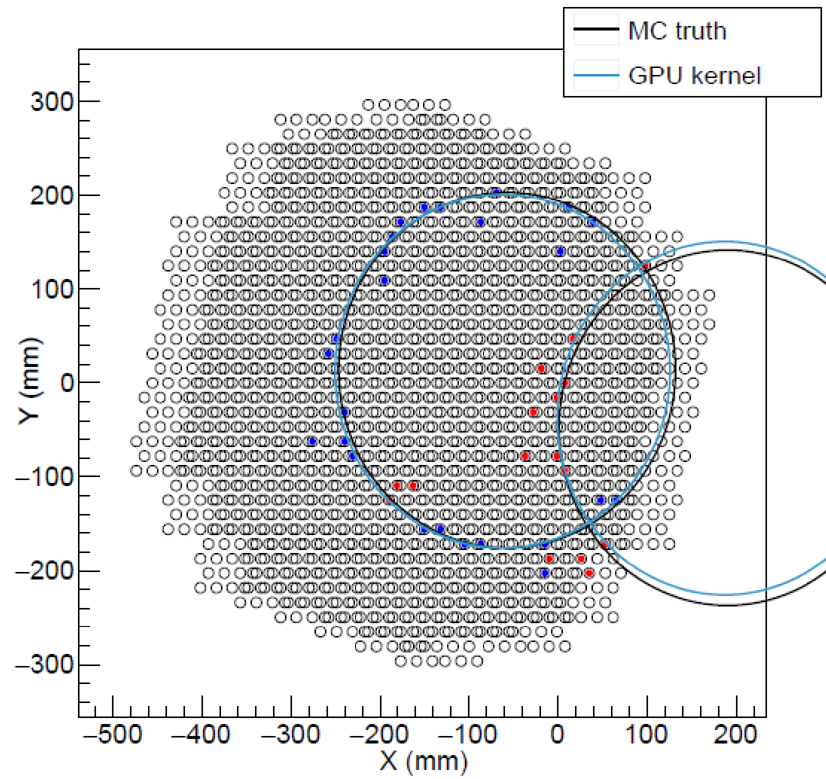


- Multi rings on the market:
  - With seeds: Likelihood, Constrained Hough, …
  - Trackless: fiTQun, APFit, possibilistic clustering, Metropolis-Hastings, Hough transform, …

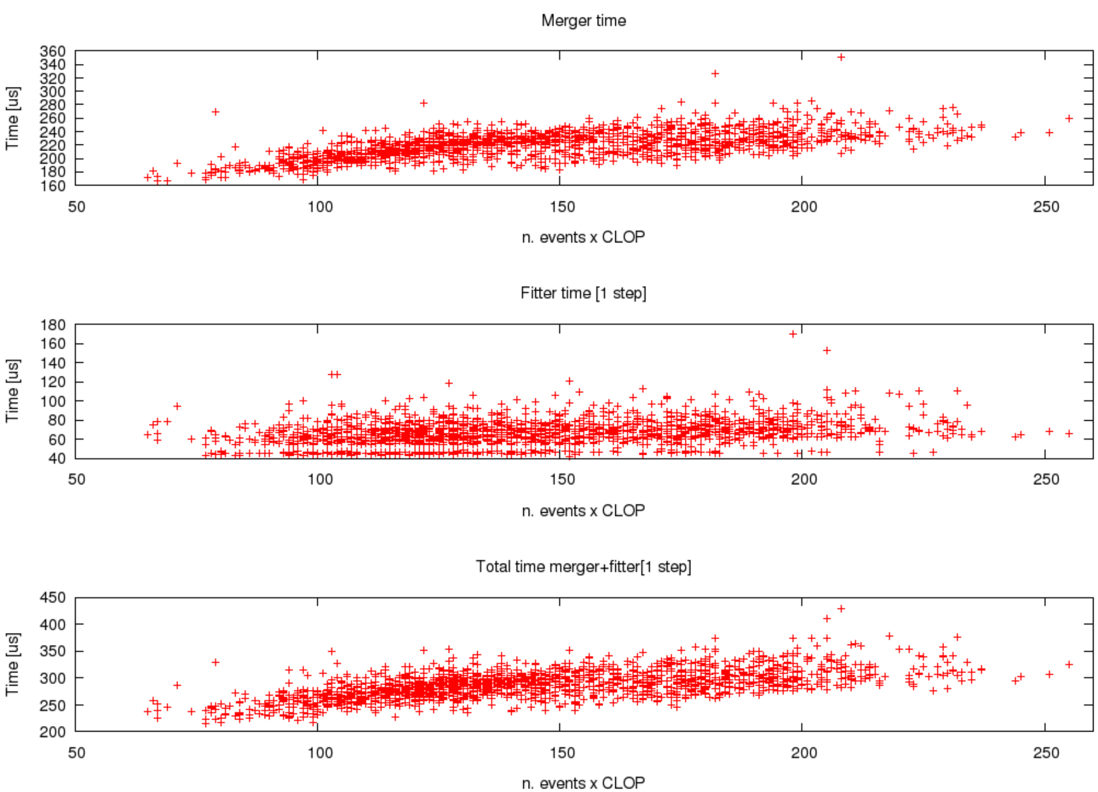
Histogram algorithm

- XY plane divided into a grid
- An histogram is created with distances from the grid points and hits of the physics event
- Rings are identified looking at distance bins whose contents exceed a threshold value





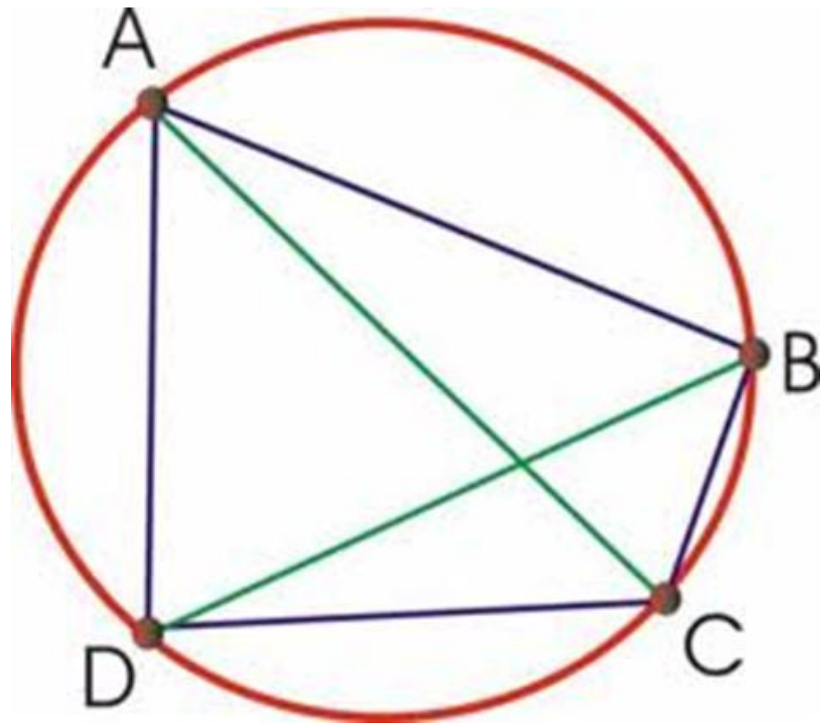
Results



- Sending real data from NA62 2015 RUN
  - NaNet-1 board
  - GPU NVidia K20
- Merging events in GPU from two different sources
- FPGA merger will be implemented soon
- Kernel histogram
- 33×10^6 protons per pulse
  - >10 MHz
  - Max 1ms latency allowed

Almagest: multi-ring identification

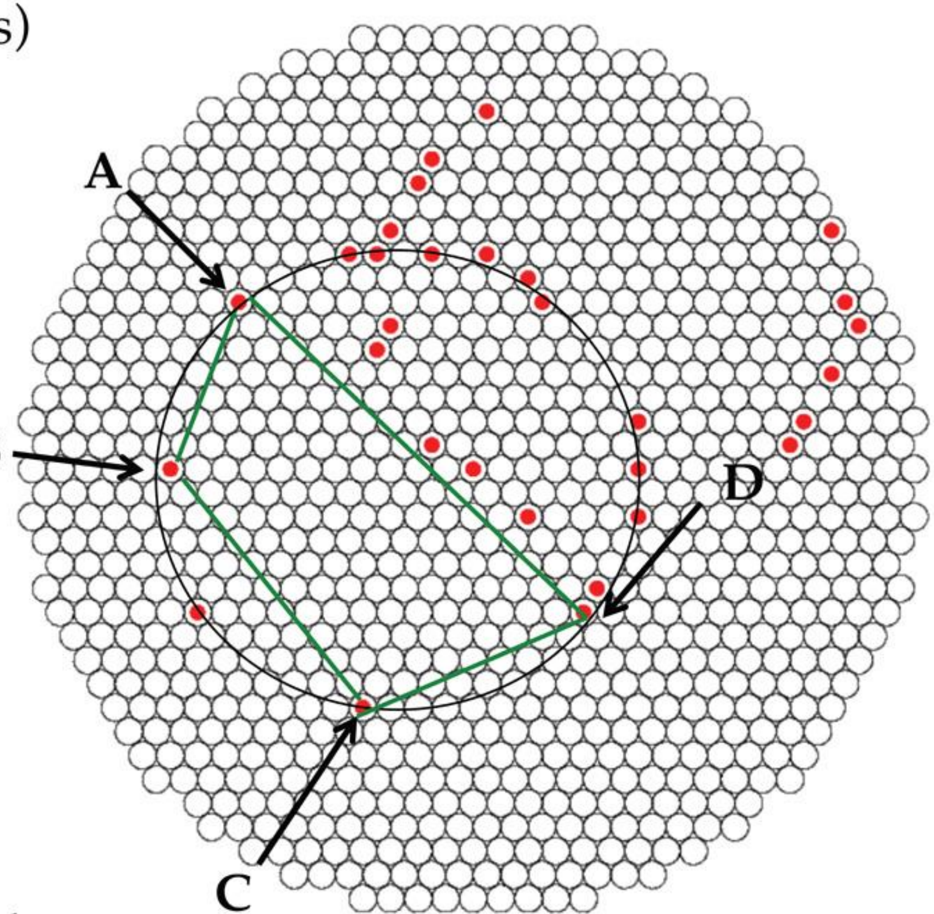
- New algorithm (Almagest) based on Ptolemy's theorem: *"A quadrilateral is cyclic (the vertex lie on a circle) if and only if is valid the relation: AD*BC+AB*DC=AC*BD* "
- Select a triplet and check if all the other points lie on the same ring by checking the Ptolemy's theorem
- Design a procedure for parallel implementation



Almagest: multi-ring identification

i) Select a *triplet* (3 starting points)

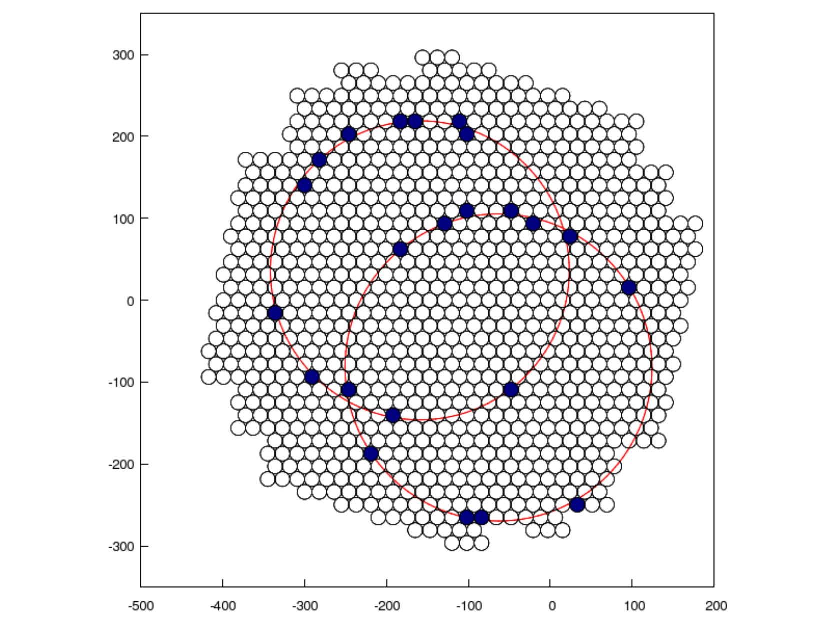
ii) Loop on the remaining points: if the next point does not satisfy the Ptolemy's condition then reject it



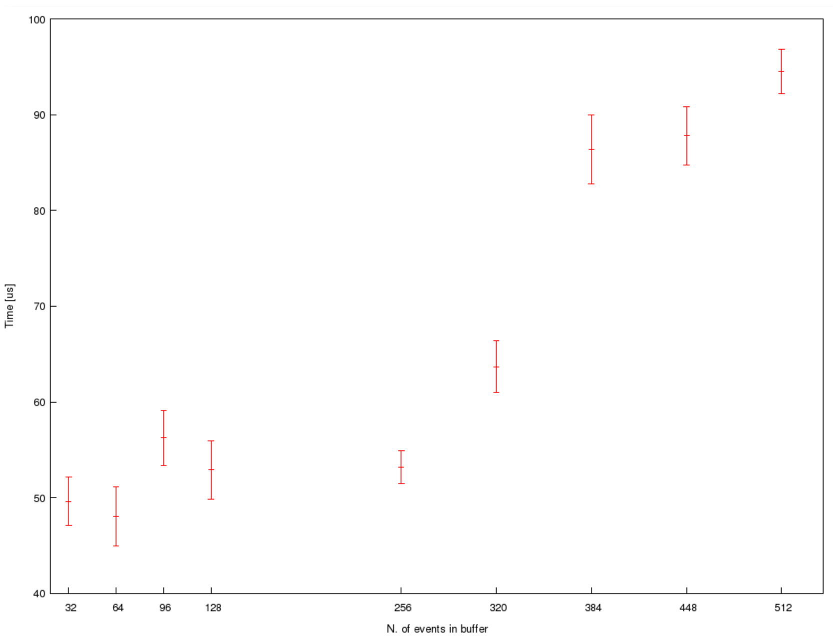
iii) If the point satisfy the Ptolemy's condition then consider it for the fit

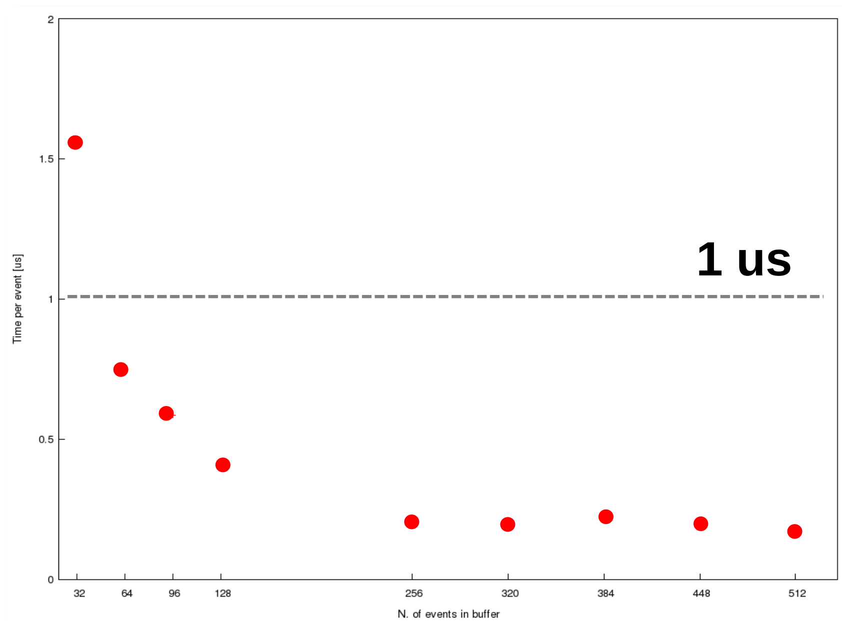
iv) …again…

Almagest results



- Tesla K20
- Only computing time presented
- <0.5 us per event (multi-rings) for large buffers





Conclusions (1)

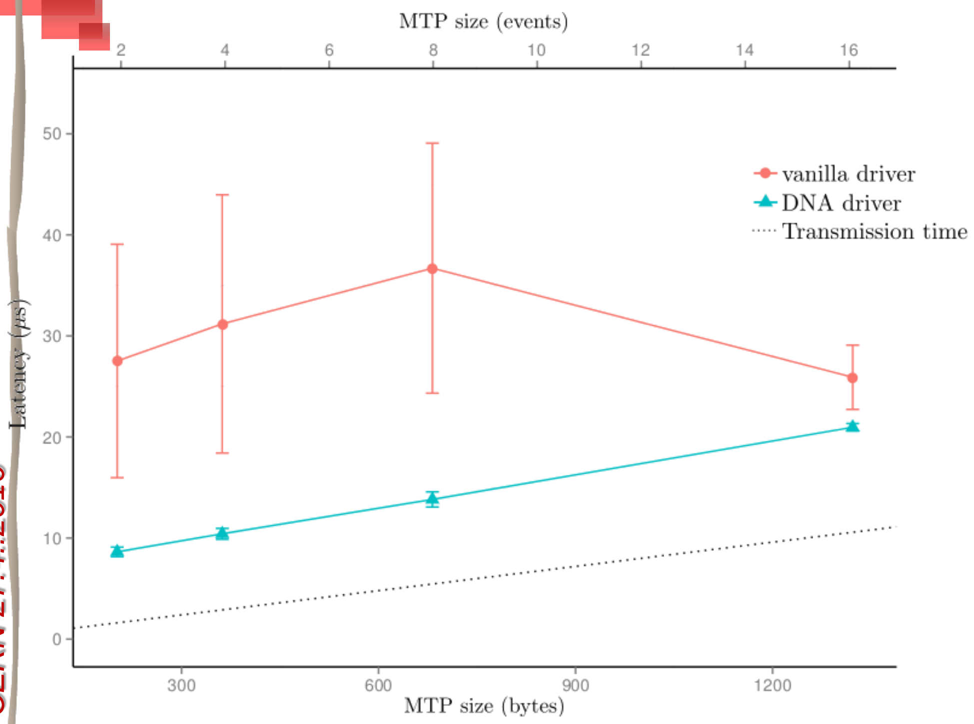
- To match the required latency in Low Level triggers, it is mandatory that data coming from the network must be copied to GPU memory avoiding bouncing buffers on host.
- A working solution with the NaNet-1 board has been realized and tested on the NA62 RICH detector.
- The GPU-based L0 trigger with the new board NaNet-10 will be implemented during the next NA62 Run started yesterday.
- Multi-ring algorithms such as Almagest and Histogram are implemented on GPU.

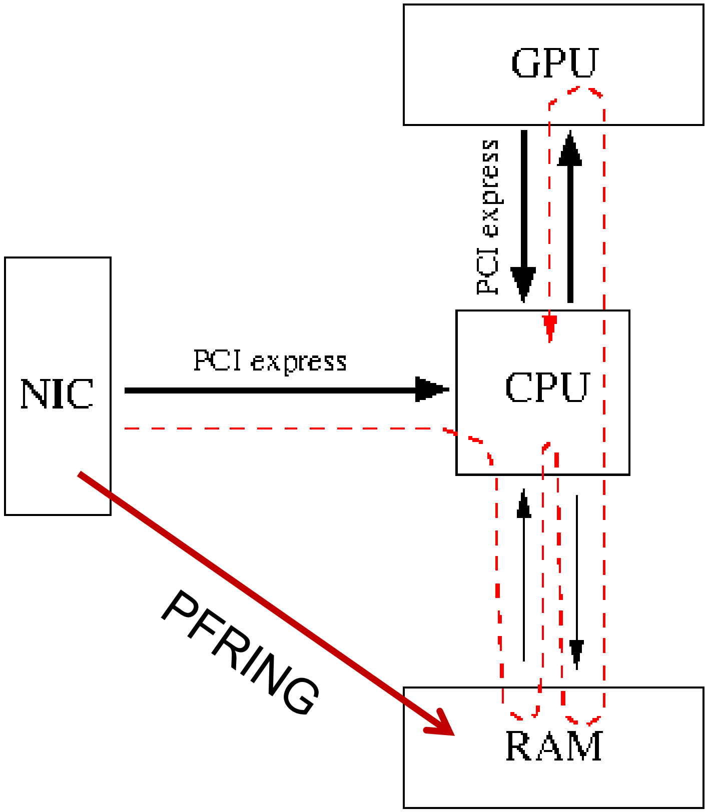
Conclusions (2)

- The GPU in the trigger could give several advantages, but the processing performances should be carefully studied (IO, Latency, Throughput)

- GPUs are flexibles, scalable, powerful, ready to use, cheap and take advantage of continuous development for other purposes: they are a viable alternative to other expensive and less powerful solution.

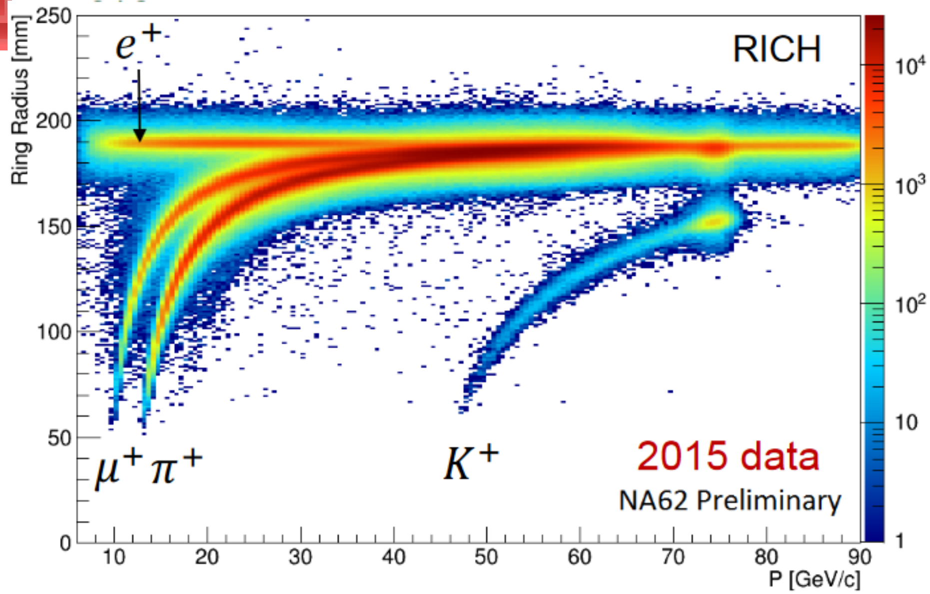
SPARES





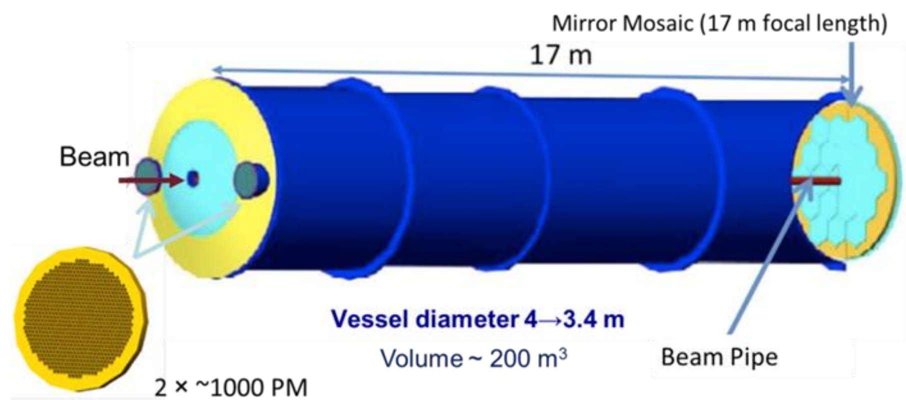
- Special driver for direct access to NIC buffer
- Data are directly available in userland
- Double copy avoided
- Pros: No extra HW needed; Cons: Pre-processing on CPU

Low Level Trigger: NA62 Test bench

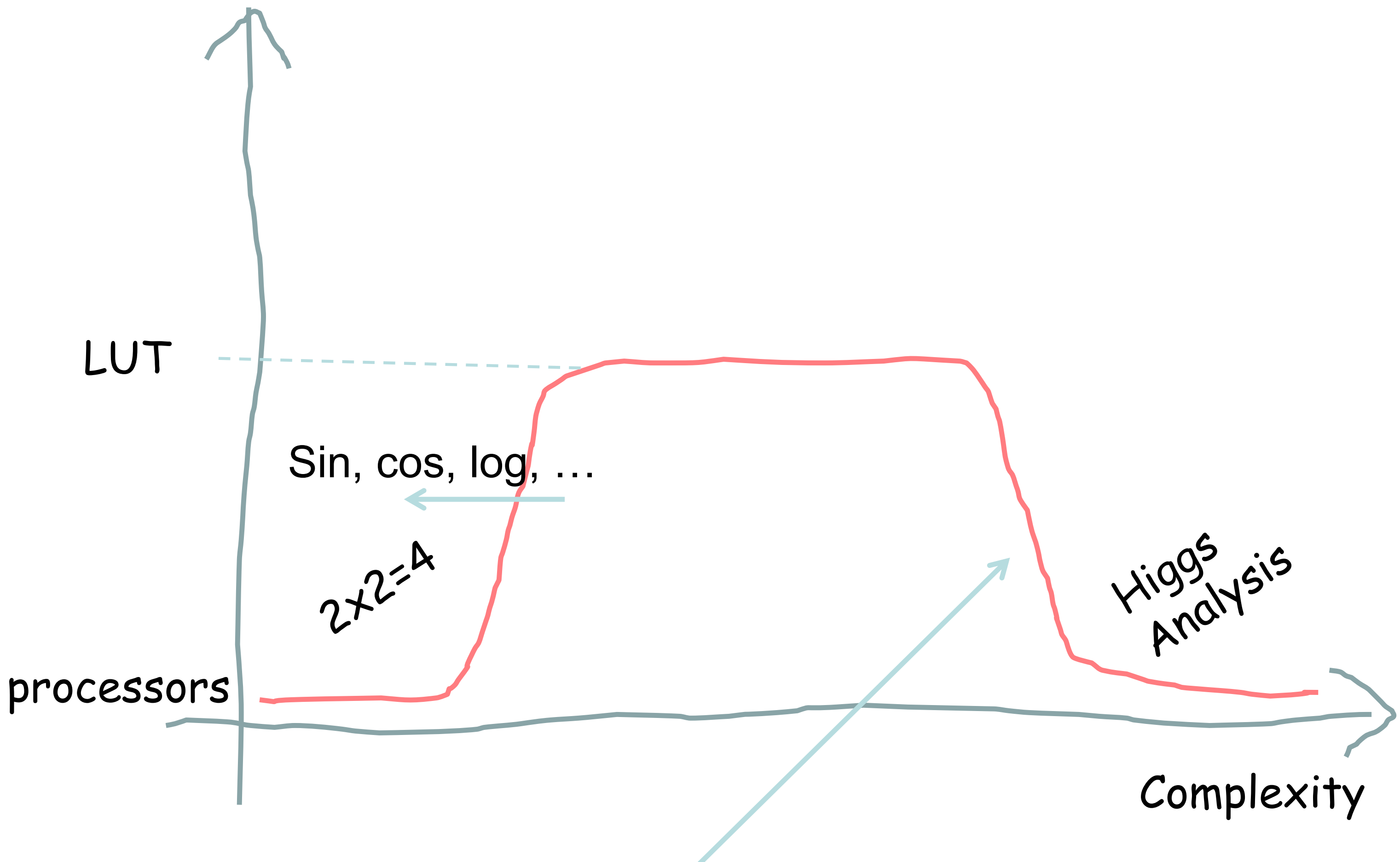


- Kaon decays in flight
  - High intensity unseparated hadron beam (6% kaons).
  - Event by event K momentum measurement.
- Huge background from kaon decays
  - ~10^8 background wrt signal
  - Good kinematics reconstruction.
  - Efficient veto and PID system for not kinematically constrained background.

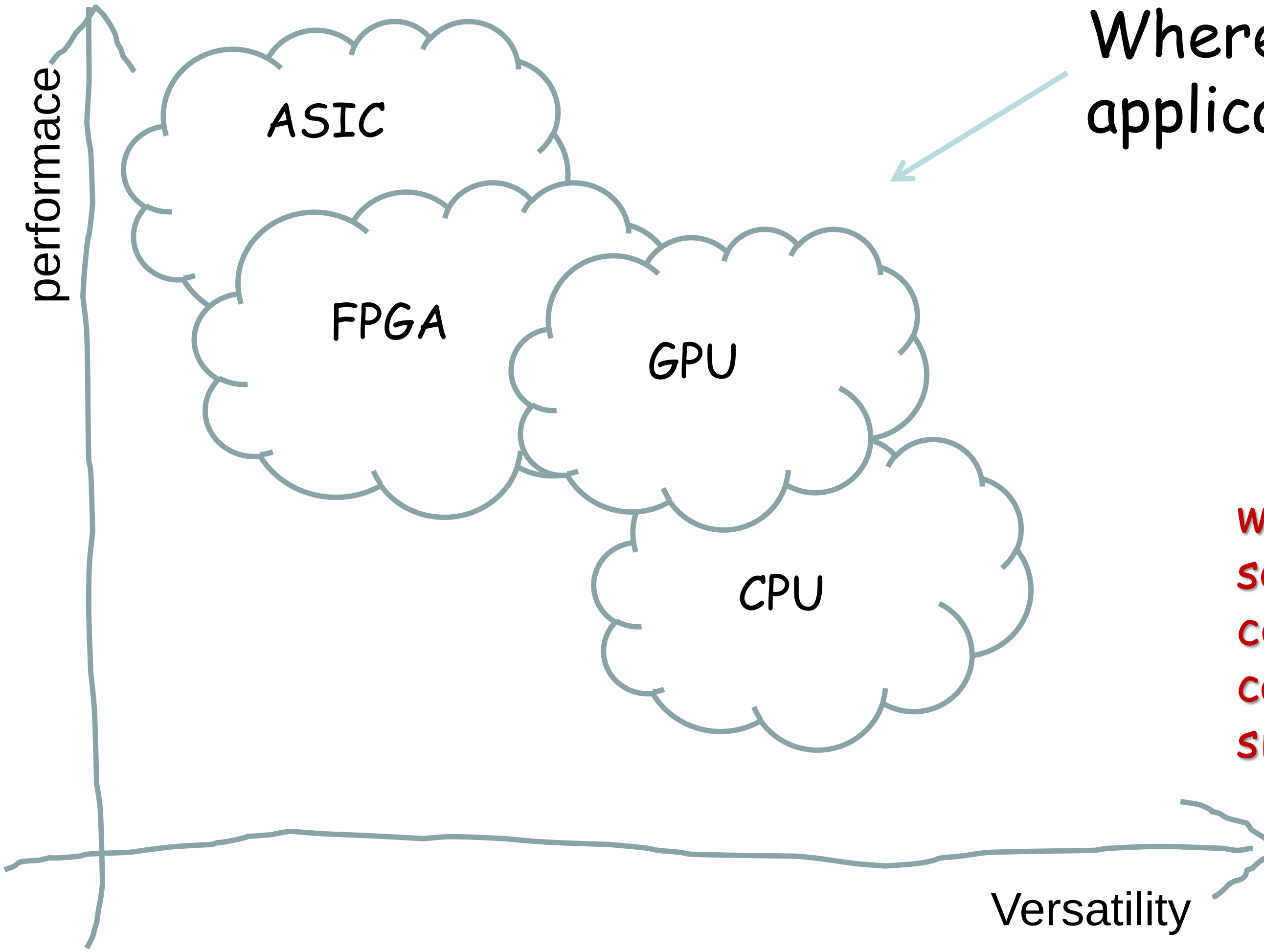
- RICH:
  - 17 m long, 3 m in diameter, filled with Ne at 1 atm
  - Distinguish between pions and muons from 15 to 35 GeV
    - 2 spots of 1000 PMs each
    - Time resolution: 70 ps
    - MisID: 5×10^{-3}
    - 10 MHz events: about 20 hits per particle



Computing vs LUT in FPGA



Where is this limit?
It depends …
In any case the GPUs
aim to shrink this space



Where is your application?

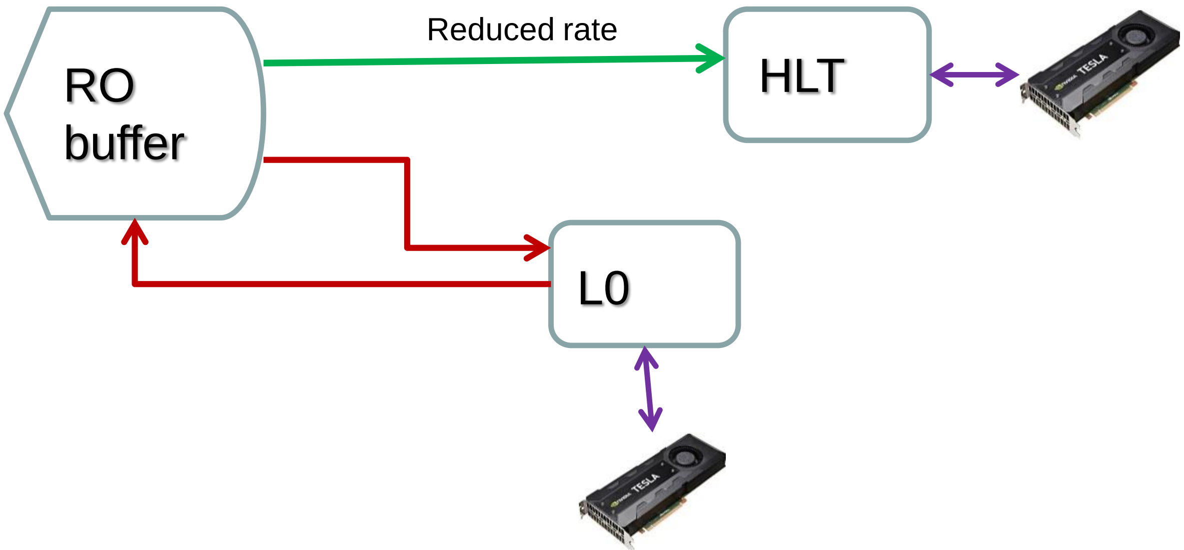
why would I do something in such a complicated way if I can just make it simple?

General purpose or dedicated hardware???

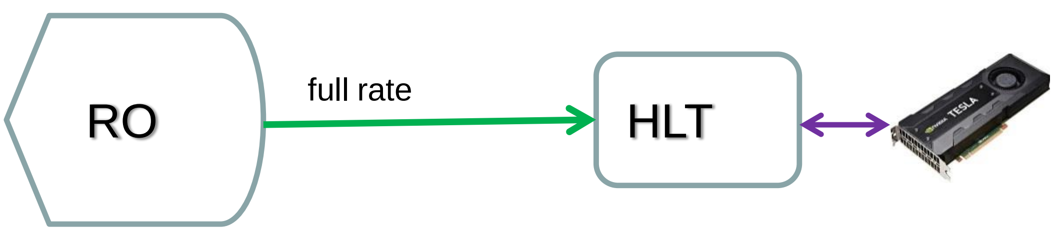
It depends on the application→ i.e. memory speed vs processor speed

GPUs are a good "compromise" …fill the GAP

GPU: where?

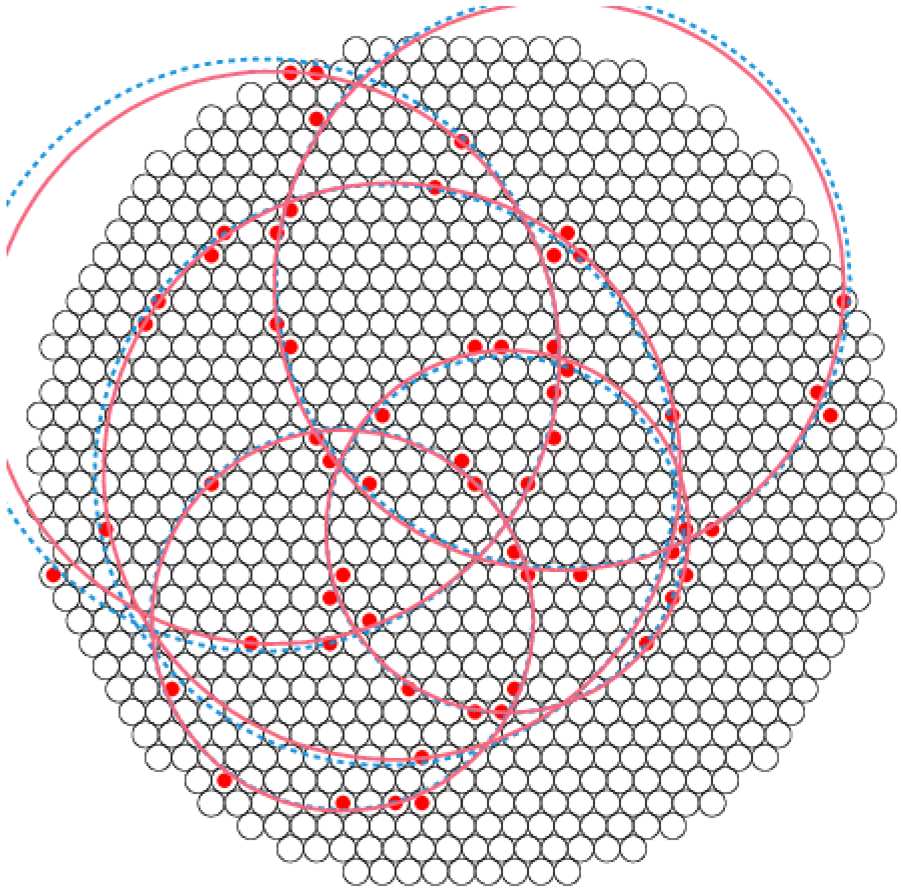


«classical trigger»

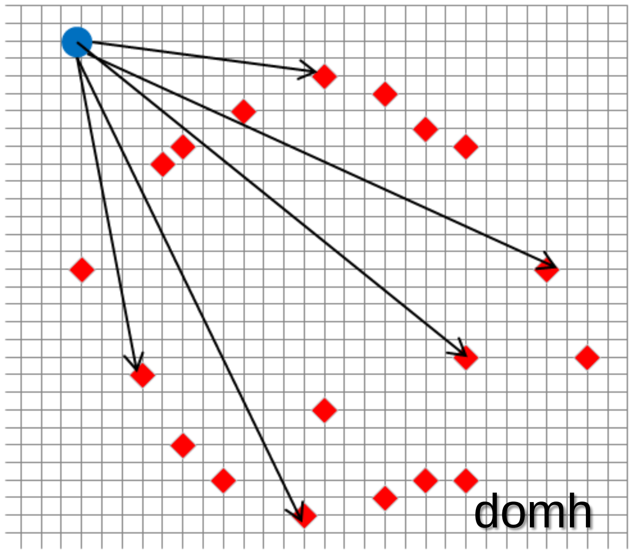


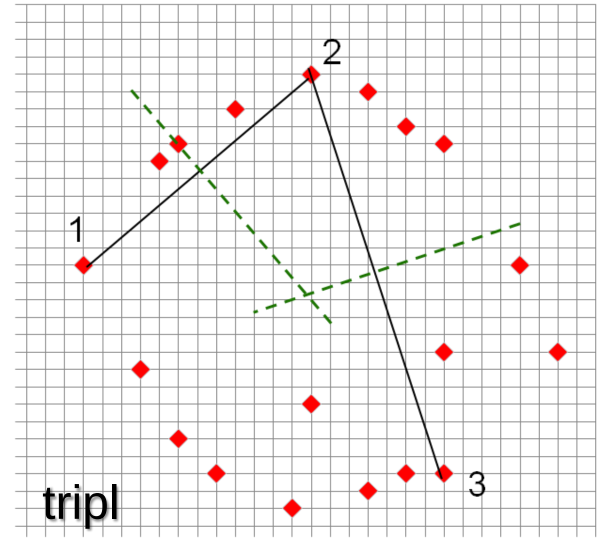
«triggerless»

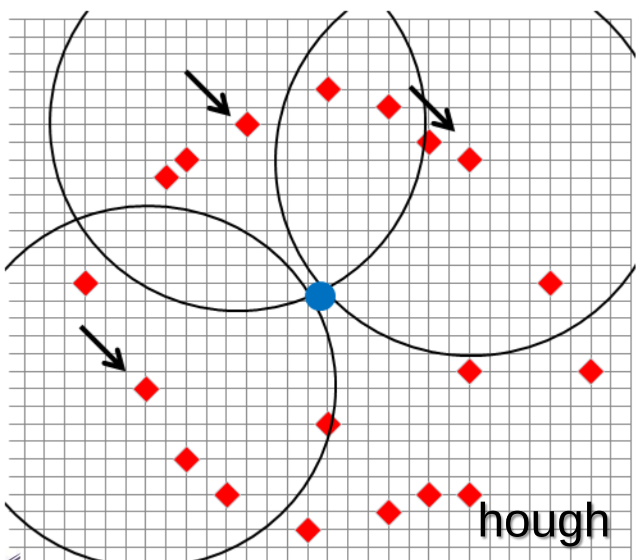
A more “complicated” example

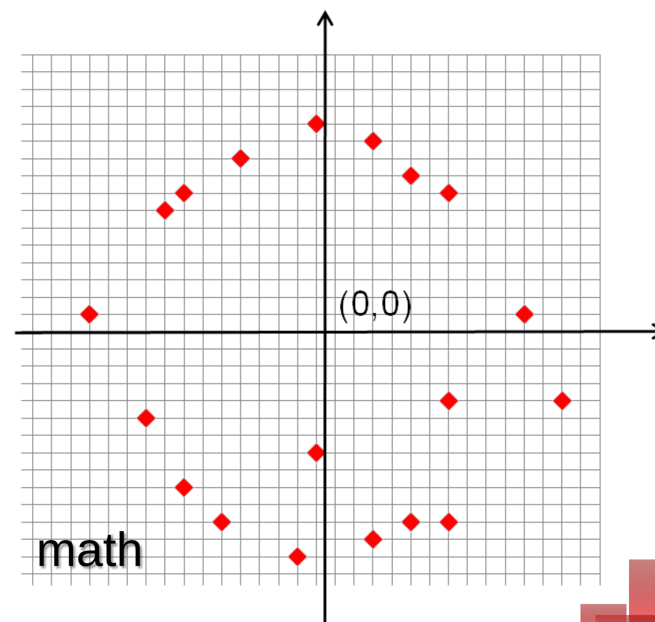


Single ring



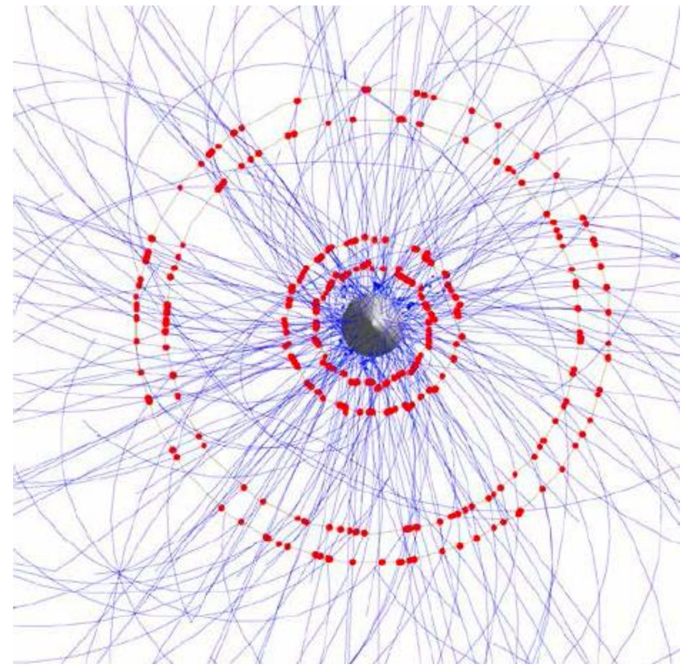


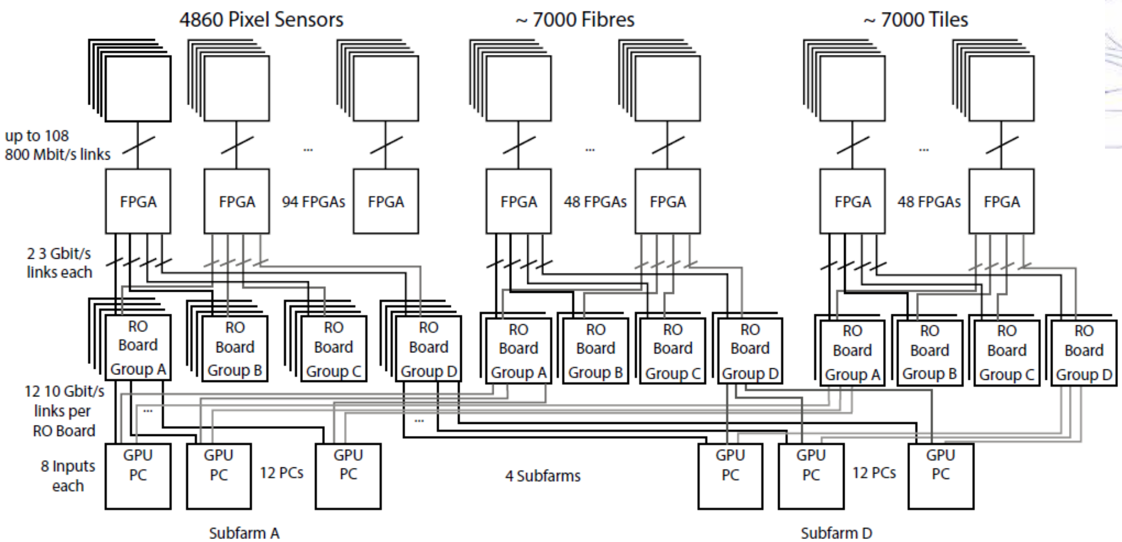


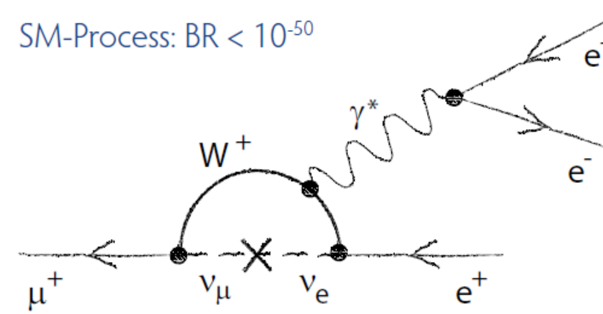


Mu3e

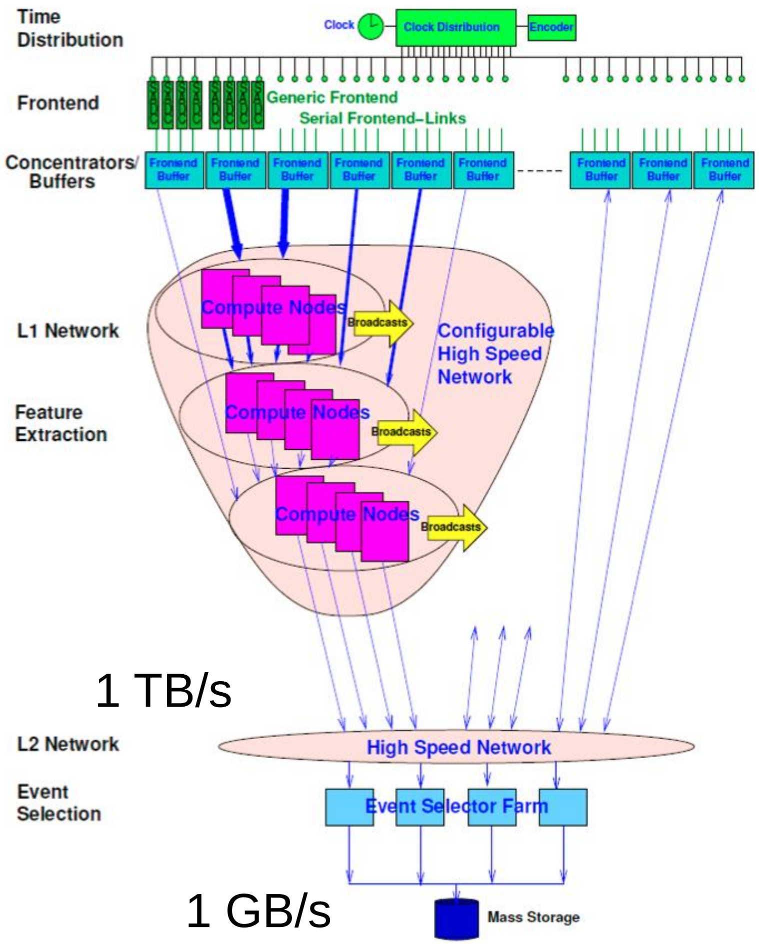
- Possibly a "trigger-less" approach
- High rate: $2\text{x}10^9$ tracks/s
- >100 GB/s data rate
- Data taking will start >2016







PANDA



- 10^7 events/s
- Full reconstruction for online selection: assuming 1-10 ms → 10000 – 100000 CPU cores
- Tracking, EMC, PID,…
- First exercice: online tracking
- Comparison between the same code on FPGA and on GPU: the GPUs are 30% faster for this application (a factor 200 with respect to CPU)

	CPU (ms)	GPU (ms)	Improvement	Occupancy	Notes
total runtime (without Z-Analysis)	117138	590	199		
startUp()	0.25	0.0122	20	2%	runs (num_points) times
setOrigin()	0.25	0.0119	21	25%	runs (num_points) times
clear Hough and Peaks (memset on GPU)	3	0.0463	65	100%	runs (num_points) times
conformalAndHough()	73	0.8363	87	25%	runs (num_points) times
findPeaksInHoughSpace()	51	0.497	103	100%	runs (num_points) times
findDoublePointPeaksInHoughSpace()	4	0.0645	62	100%	runs (num_points) times
collectPeaks()	4	0.066	61	100%	runs (num_points) times
sortPeaks()	0.25	0.0368	7	2%	runs (num_points) times
resetOrigin()	0.25	0.0121	21	25%	runs (num_points) times
countPointsCloseToTrackAndTrackParams()	22444	0.9581	23426	33%	runs once
collectSimilarTracks()	4	2.3506	2	67%	runs once
collectSimilarTracks2()				2%	runs once
getPointsOnTrack()	0.25	0.0187	13	33%	runs (num_tracks) times
nullifyPointsOfThisTrack()	0.25	0.0106	24	33%	runs (num_tracks) times
clear Hough space (memset on GPU)	2	0.0024	833	100%	runs (num_tracks) times
secondHough()	0.25	0.0734	3	4%	runs (num_tracks) times
findPeaksInHoughSpaceAgain()	290	0.2373	1222	66%	runs (num_tracks) times
collectTracks()	0.25	0.0368	7	2%	runs (num_tracks) times