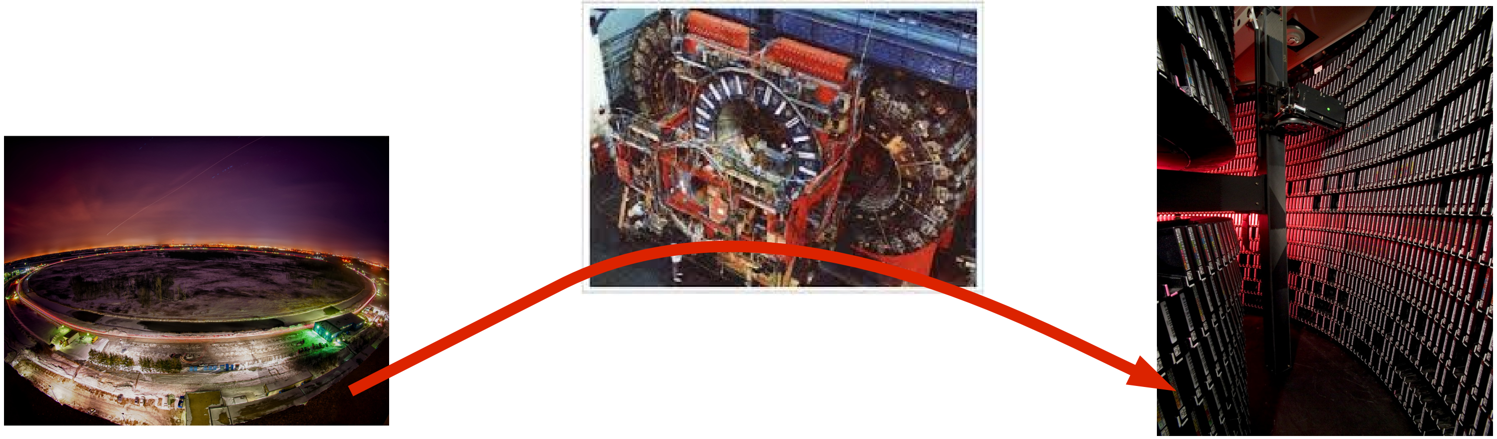
Many-core applications to online track reconstruction in HEP experiments

S.Amerio, D.Bastieri, A.Gianelle, D.Lucchesi (INFN and University of Padova)
M. Corvo (University of Ferrara)
T.Liu, R.Rivera (Fermilab)
W.Ketchum (Los Alamos National Laboratory)
S.Poprocki, P.Wittich (Cornell University)
A.Lonardo, L.Tosoratto, P.Vicini (INFN Roma)

CHEP 2013 – 14-18 October 2013

Real time selections in HEP physics



From collisions - O(10 MHz) - to data saved on tape – O(1 kHz): 10^4 reduction factor!

A real time selection system (trigger) plays a key role in HEP experiments at hadron colliders, *reducing rate and selecting the most signal-like events.*

A trigger system should be...
- fast, not only in **data processing** but also in **data transfer**
- **scalable**, to adapt to changing data taking conditions
- **easy to maintain**
- **not too expensive**, if possible :-)

Parallel architectures and real time selections

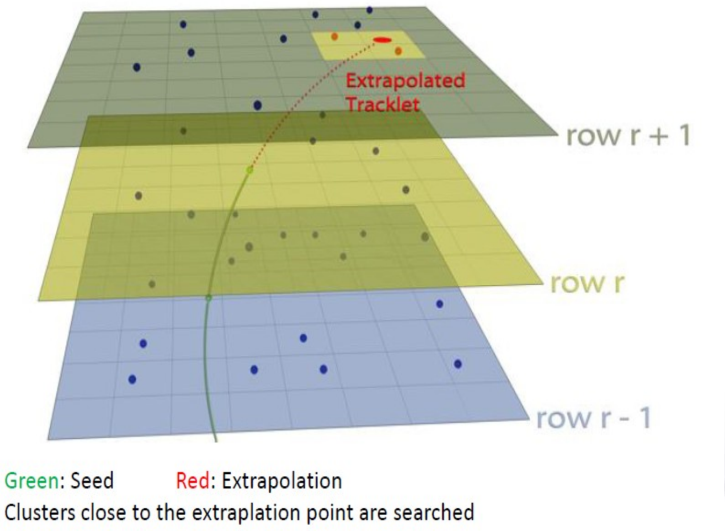
Interest in *parallel architectures* is increasing in scientific computing.

Graphic Processing Units (GPUs) and multi-core CPUs provide

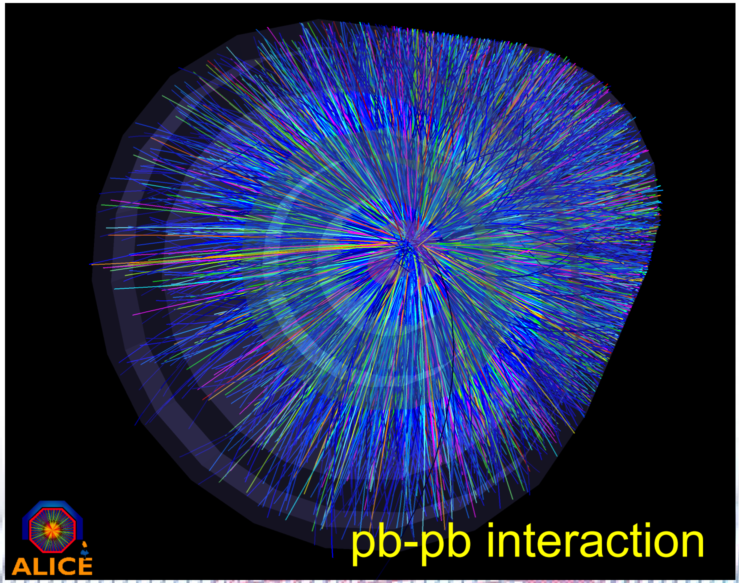
- *A lot of computing power for highly parallelizable tasks;*
- *High level programming languages (C/C++, CUDA, OpenCL);*
- *Continuous improvement* of performance driven by the market.

Real time selections are usually based on algorithms well suited for parallelization, e.g. the reconstruction of trajectories left by charged particles (online tracking).

From hits in the detector.... to tracks



Green: Seed Red: Extrapolation
Clusters close to the extraplation point are searched



In this talk...

- Performance study of parallel architectures in real time selections.
- Use case: online track reconstruction at CDF experiment at Tevatron
  - SVT (Silicon Vertex Trigger) track fitting algorithm ported to parallel architectures
- Measurements of
  - *Data processing latency*, comparing different parallel architectures (GPU, MIC, CPU)
  - *Data transfer latency*, comparing different data transfer protocols (GPU Direct V1.0, Cuda Aware MPI, P2P)

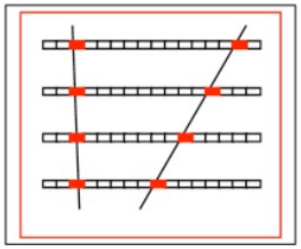
Goal: identify strength and weakness of the different technologies.
Investigate different data taking environments: from thousands to millions of track fits.

Previous studies presented at NSS2012
http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6551422

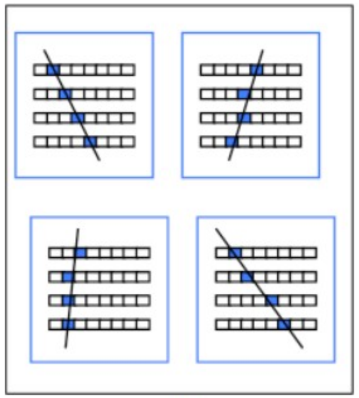
SVT algorithm in a nutshell

SVT – based on custom hardware – reconstructed tracks in time for a Level-2 trigger decision (~ 20 μs) in two steps:

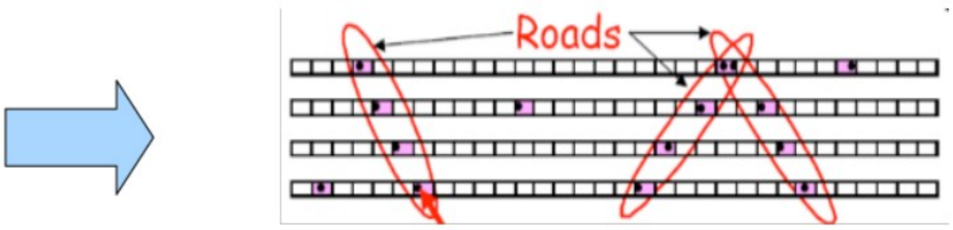
1) Pattern recognition to form hit combinations (roads)



Event Hits



Compare to Pattern Bank



We implemented this part of the code on parallel architectures

2) Track fitting inside roads using simple scalar product

$$p_i = \vec{f}_i \cdot \vec{x} + q_i$$

Known constants. Precalculated and ***stored in memory***

track parameters (output)

track coordinates (input hit information)

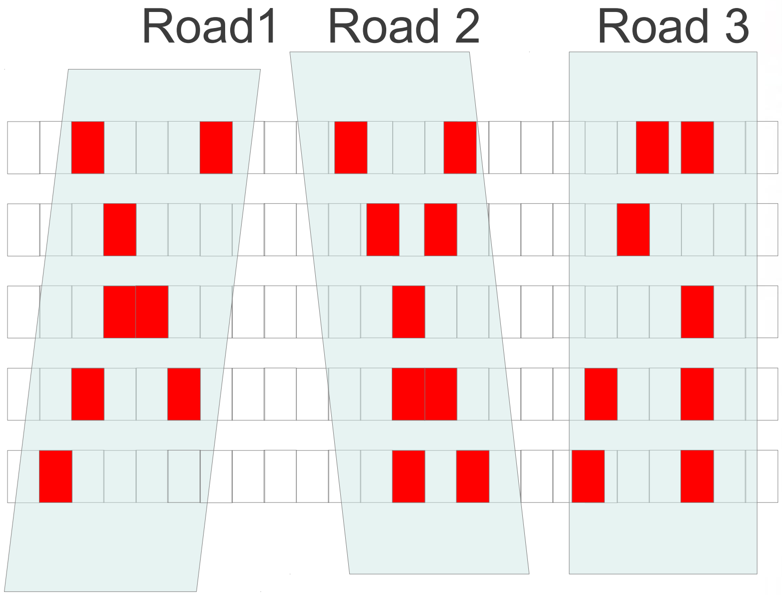
SVT track fitting algorithm step by step

Unpack input data (24-bit words) and fill all the necessary *arrays.*

SVT track fitting algorithm step by step

Unpack input data (24-bit words) and fill all the necessary *arrays*.

For each event and for each road, *calculate all the possible combinations* (each layer can have more than one hit)

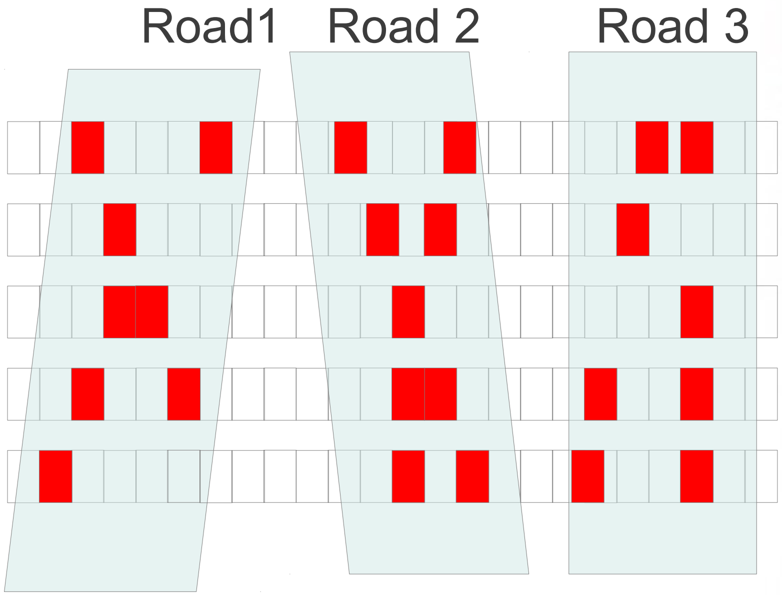


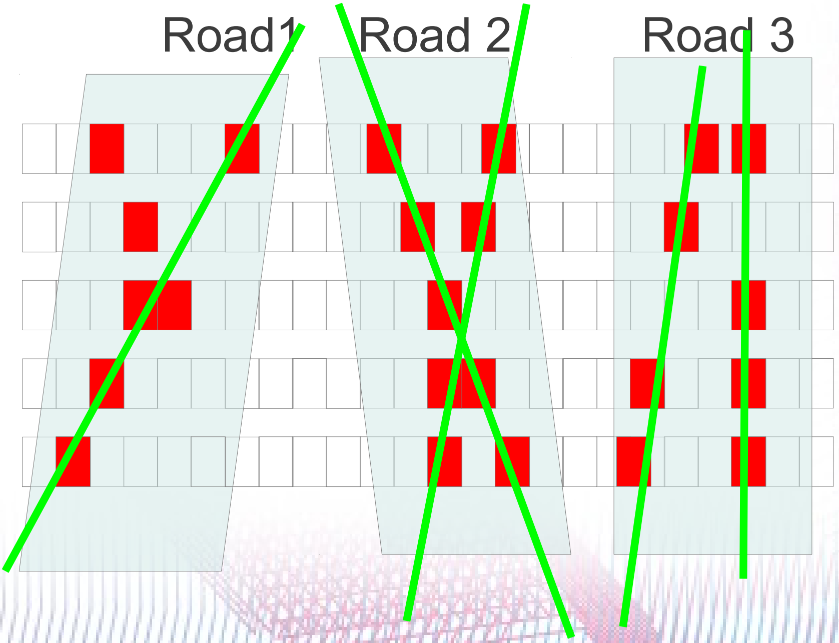
SVT track fitting algorithm step by step

Unpack input data (24-bit words) and fill all the necessary *arrays.*

For each event and for each road, *calculate all the possible combinations* (each layer can have more than one hit)

For each combination:
- *retrieve fit constants from device memory*
- perform scalar product and chi2 cut
- format good tracks for output





Code implementation

- Our approach to the implementation as been as much conservative as possible
- Starting point: SVT simulation code, written in *C language*.
- Strategy: do not re-think the track fitting algorithm, but parallelize whenever possible
  - GPU
    - Write a different kernel for each function (CUDA)
    - parallelize over events, roads, combination --> *each thread processes 1 candidate track*
  - MIC
    - pragma OpenMP parallel for statements (*embarassingly parallel* approach)
    - parallelize over events --> *each core processes 1 event*

Not only performance measurements, but also assess feasibility for porting current serial code to parallel architectures.

Hardware specifications

	Tesla M2050	Tesla K20m	GeForce GTX TITAN	MIC 5110P
Performance (SP, GFlops)	1030	3520	4500	2022
Memory Bandwidth (GB/s)	148	208	288	320
Memory Size (Gb)	3	5	6	8
Number of cores	448	2496	2688	60
Clock speed (GHz)	1.15	0.706	0.837	1.053

	Intel Core i7-3770	Intel Xeon E5630
Memory Bandwidth (GB/s)	25	25
Number of cores	8	4
Clock speed (GHz)	3.40	2.53

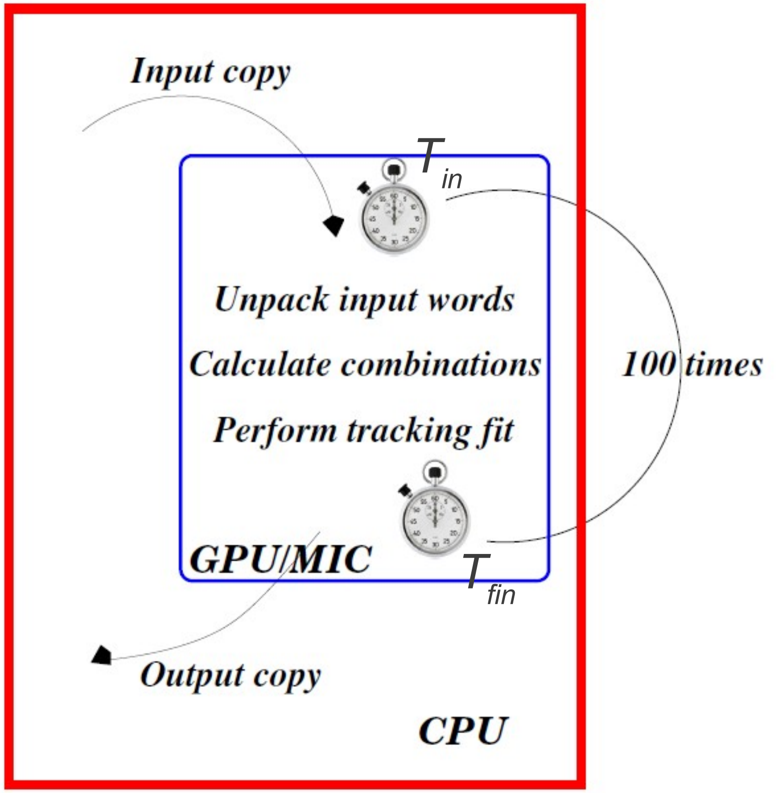
Data processing latency measurements

Input data
Events have fixed number of roads (64) and combinations (32) → *2048 candidate tracks to be fitted in each event.*

Each event is 3 kB.

Events are grouped in data samples with different event multiplicity, from 1 to 3000 → *from 2048 to 6.1 millions of fits in each data sample*

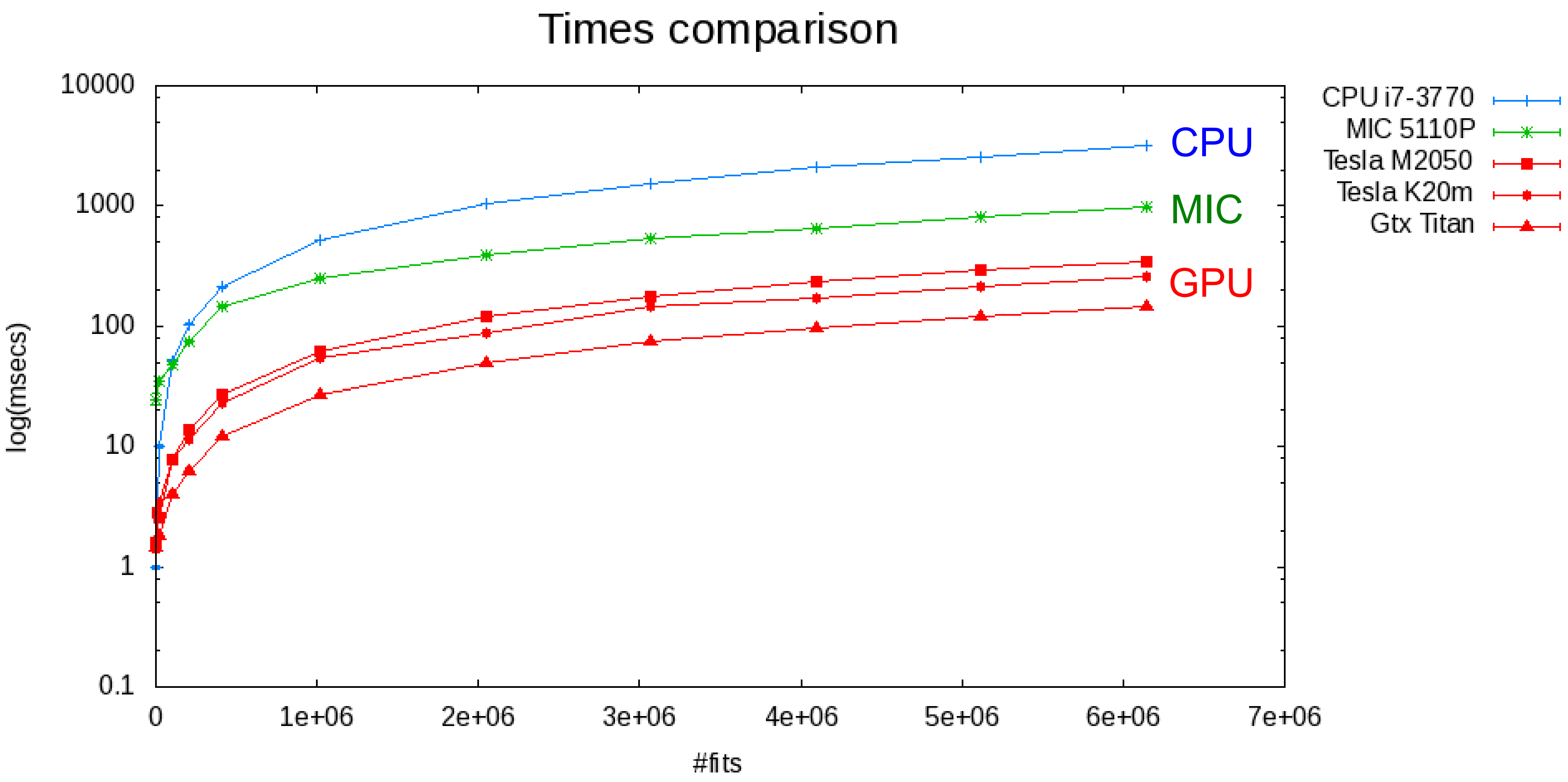
Measurements
Each data sample is processed 100 times
The final latency value is the mean over the 100 measurements.



$\Delta T = T_{fin} - T_{in}$

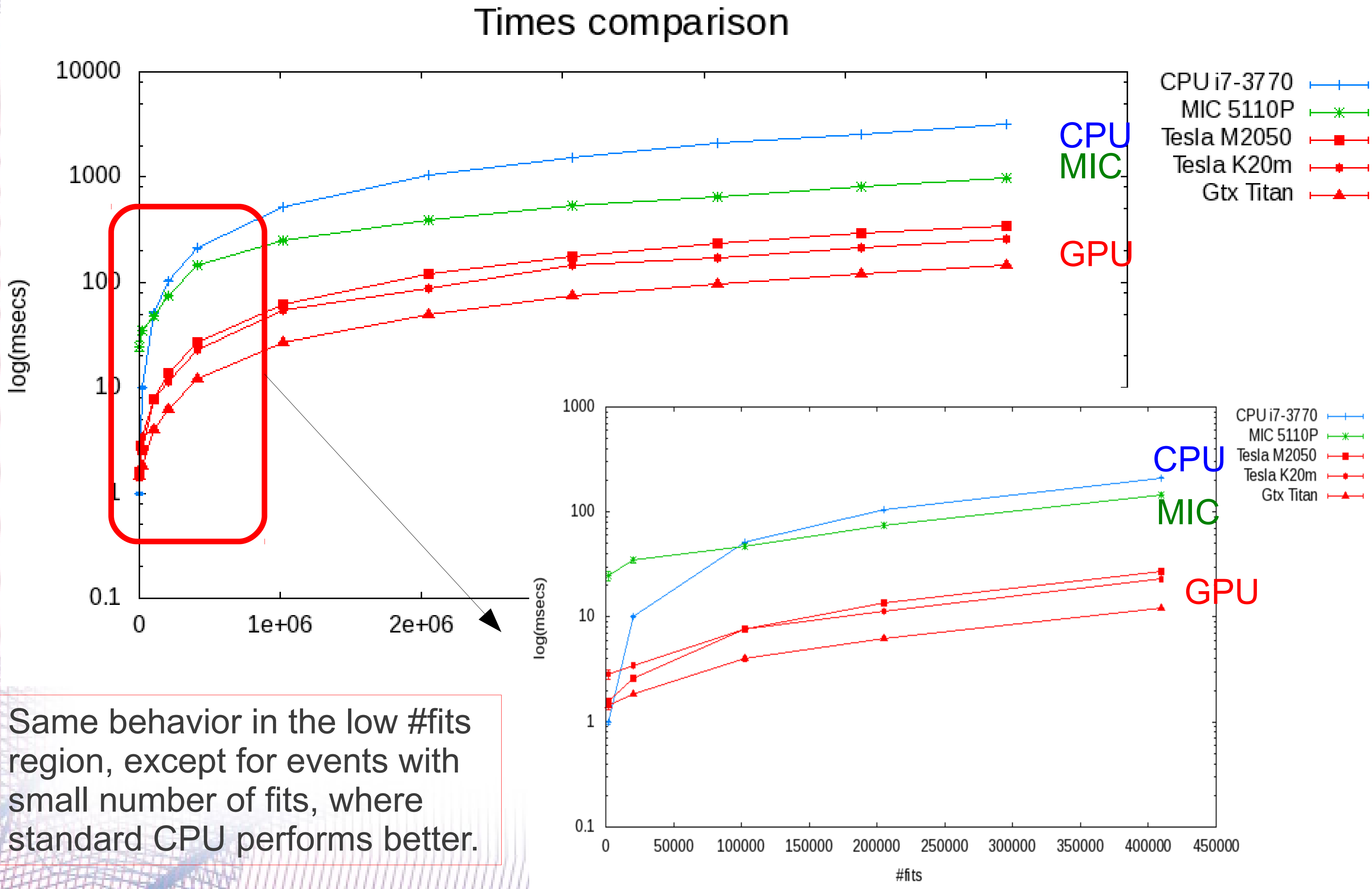
Time measured with standard C libraries (*gettimeofday()* function)

Results on data processing



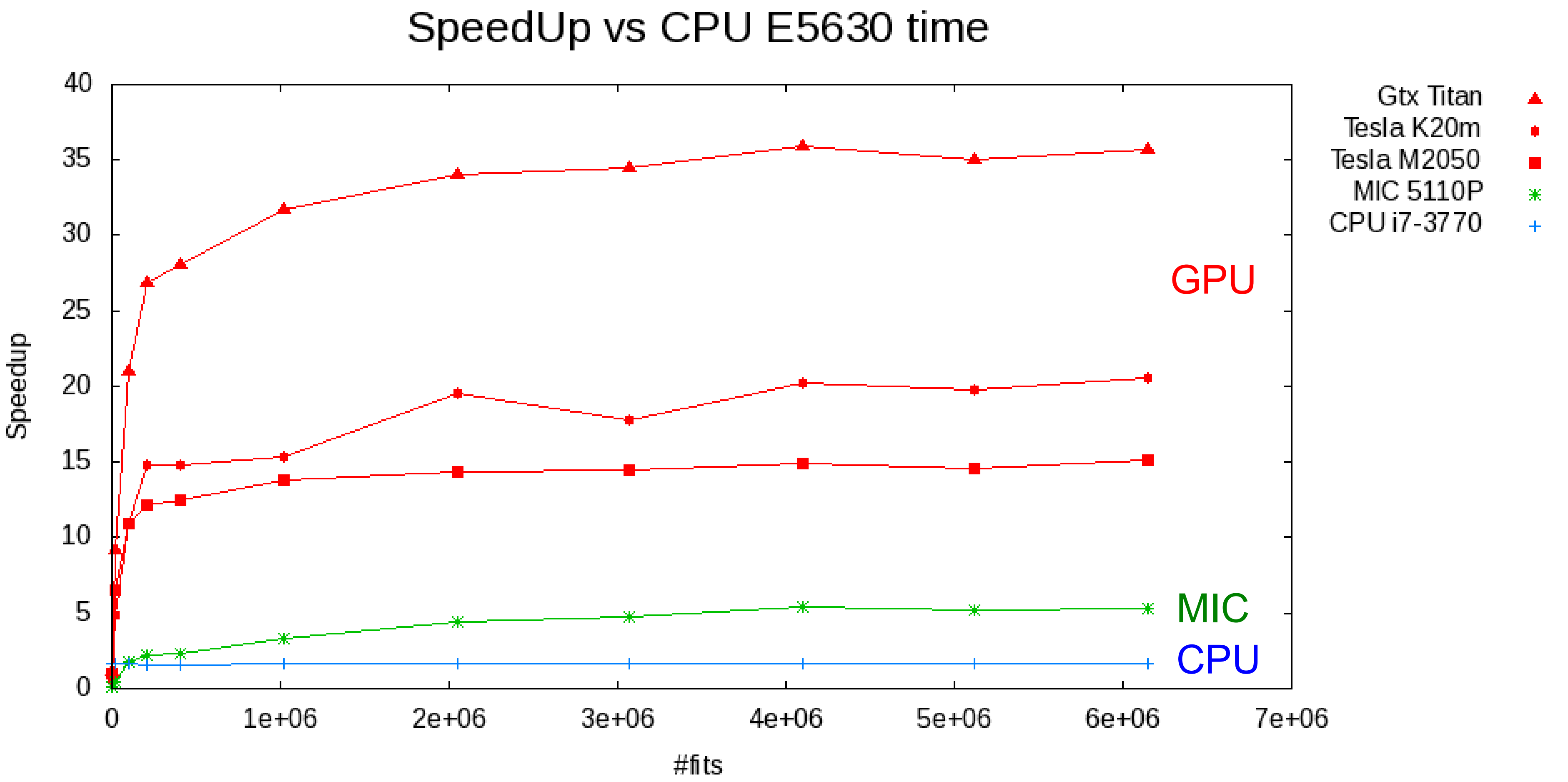
Better performance of GPUs thanks to the parallelization of the 3 nested loops. (Events, Roads, Combinations)

Results on data processing



Same behavior in the low #fits region, except for events with small number of fits, where standard CPU performs better.

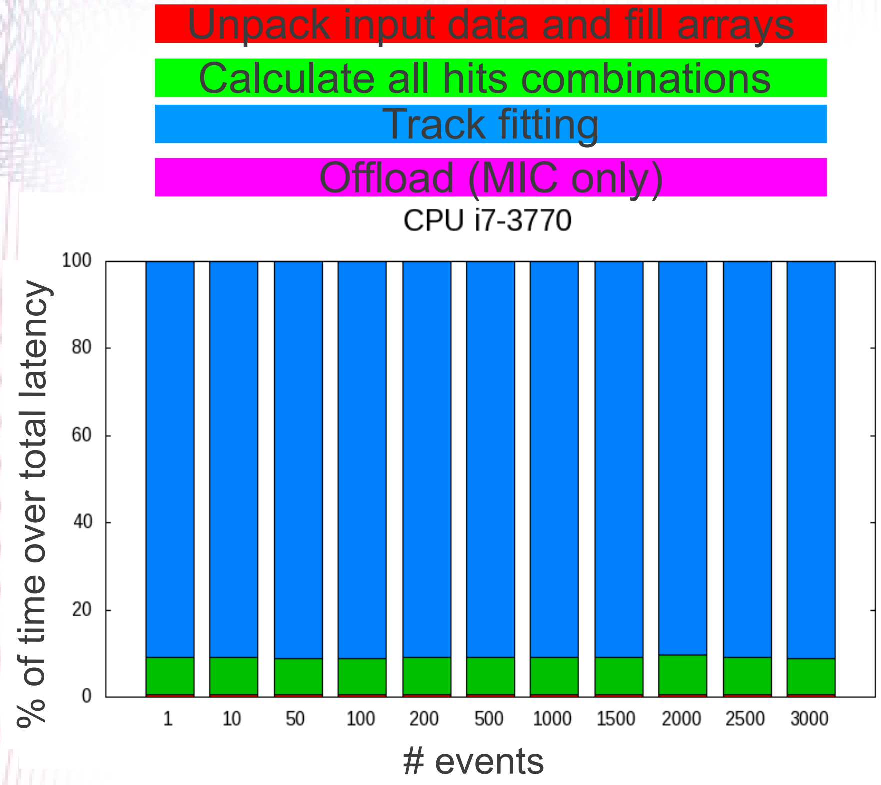
Speedup



Speedup over a standard CPU (E5630).
Maximum gain obtained with > 1000 events (or > 2 M fits) processed in parallel
To fully exploit parallel devices, we need to perform a lot of calculations.

Breakdown of computing time

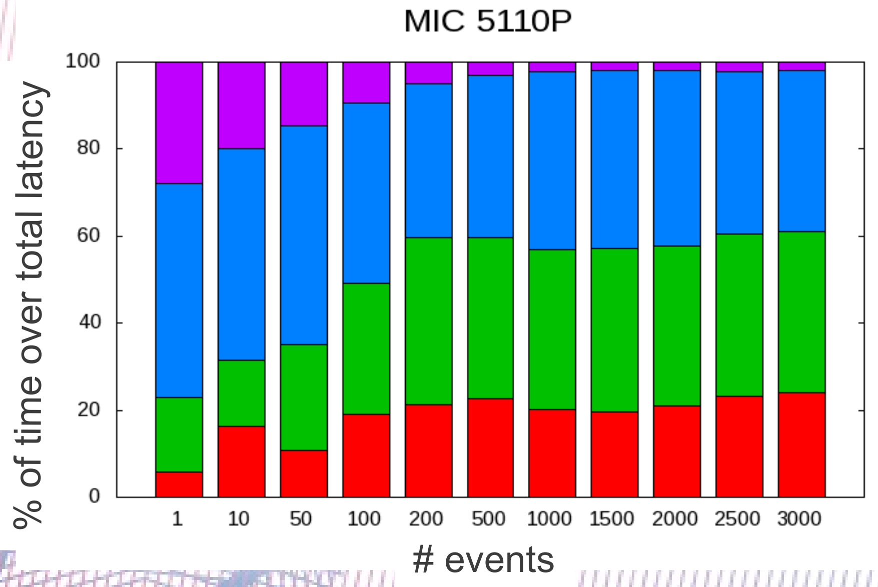
Where is most of the time spent on each device?

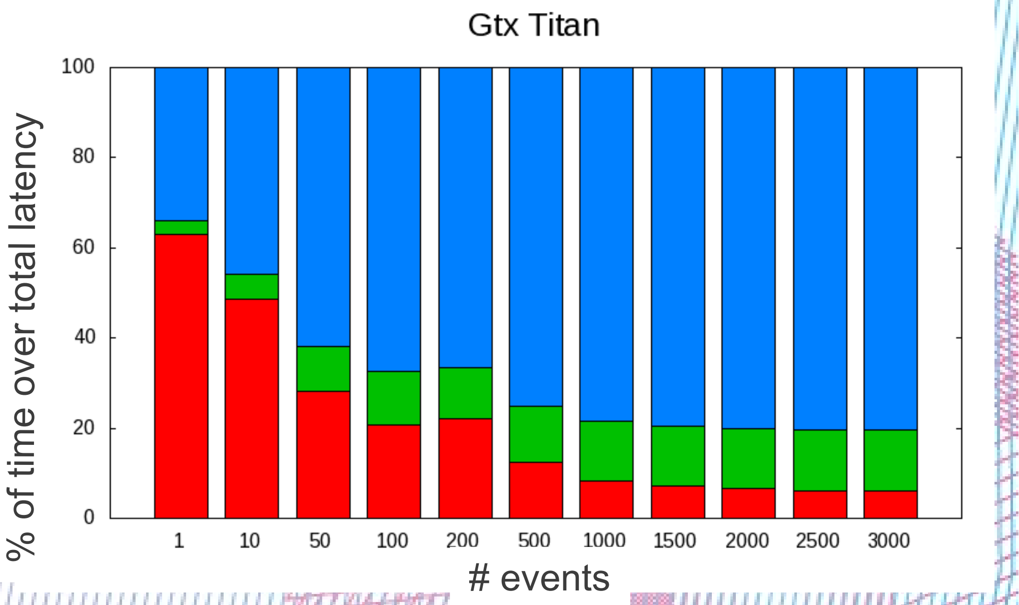


Standard CPU: most of the time spent in the fitting part; code completely serial → percentage of time flat as # evts increases

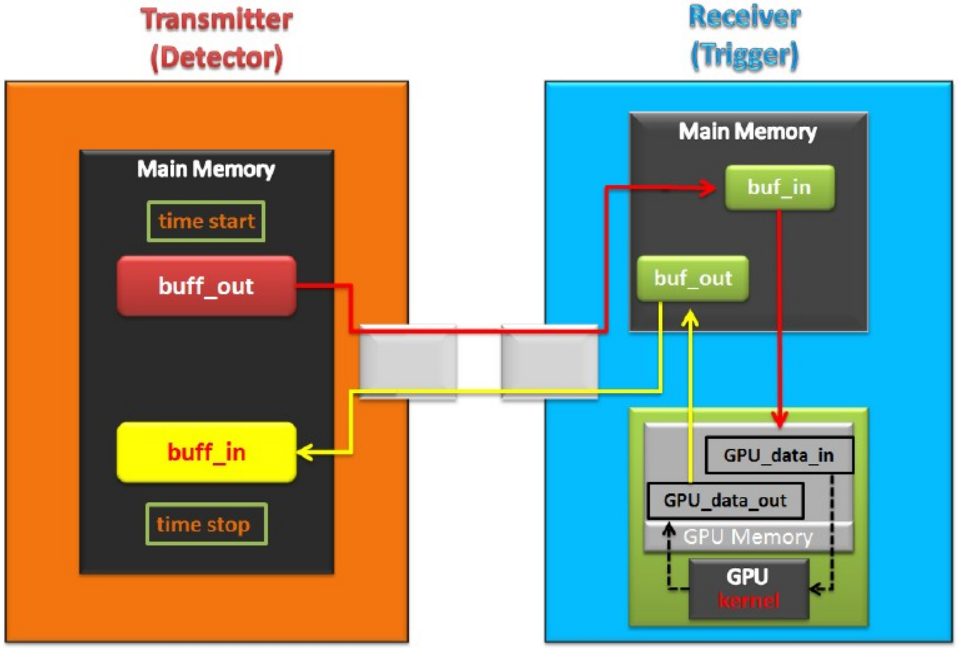
MIC:combinations and fitting part take the same time for high number of events.

GPU: the fitting stage dominates for high multiplicity of tracks.





Data transfer: experimental setup



To mimic the data transfer between Detector and Trigger system we used two PCs connected by Infiniband links.

Transmitter = Detector
Receiver = Trigger

Measurements

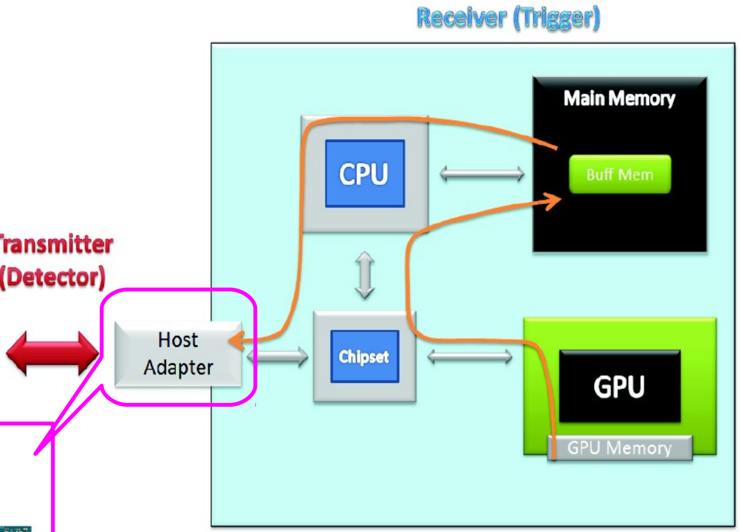
ΔT = time_stop – time_start
10k loops
Time measured in the Transmitter via standard C libraries (*gettimeofday()* function).

Data transfer: experimental setup

We consider different data transfer strategies to the GPU

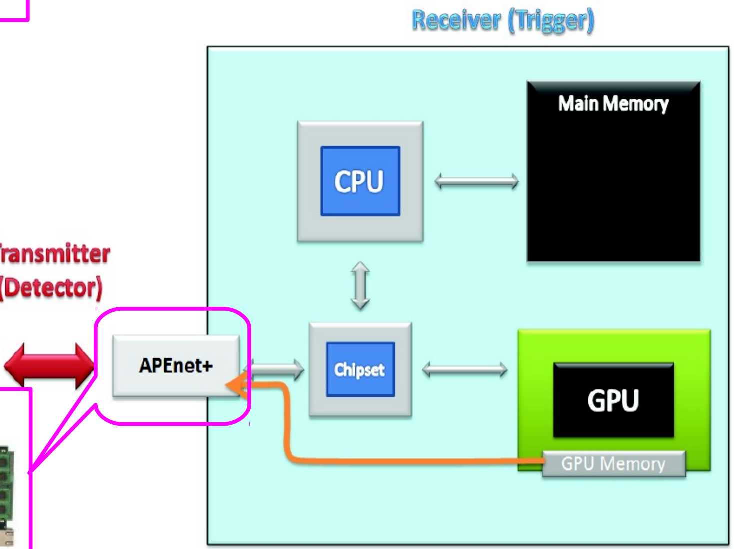
Data are transferred using *Direct Access Memory to the system memory (GPUDirect).* Two different version tested:

- **GPUDirect v1.0**, where GPU buffers need to be staged through the system memory;
- **Cuda-Aware MPI**, where GPU buffers can be directly passed to MPI functions.



Infiniband Mellanox card (Connect-X2)

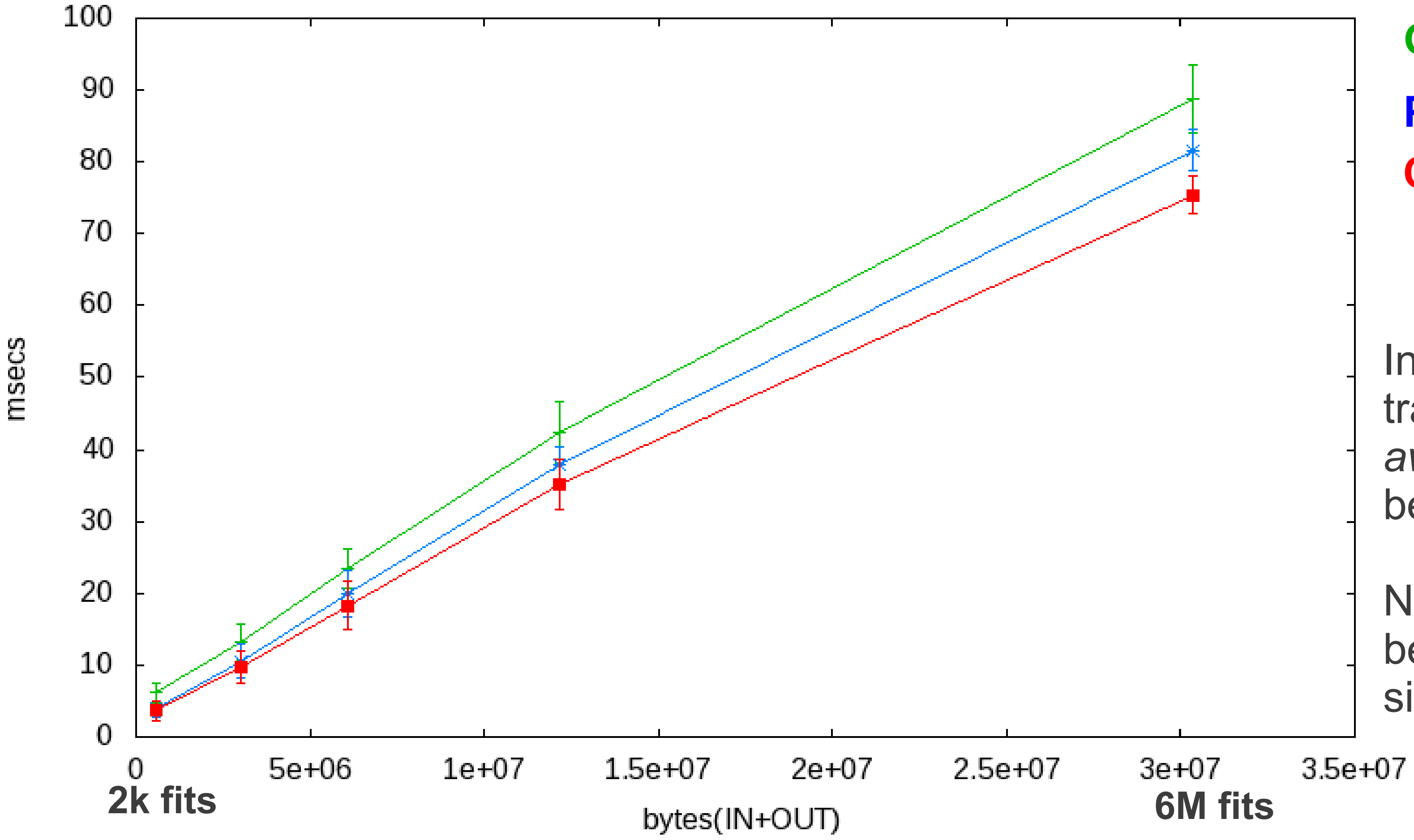
Data are transferred *directly to the GPU, avoiding any copy to the system memory* (**PeerToPeer**).



Apenet+ card (StratixIV-based PCIe board supporting P2P communication with Tesla and Kepler GPUs)

Data transfer: results (I)

Total latency (data transfer + copy of data to GPU + data processing on GPU) vs data transferred



GPU Direct v1

P2P

CUDA-aware MPI

In this range of data transfer sizes *CUDA-aware MPI* gives the best performance.

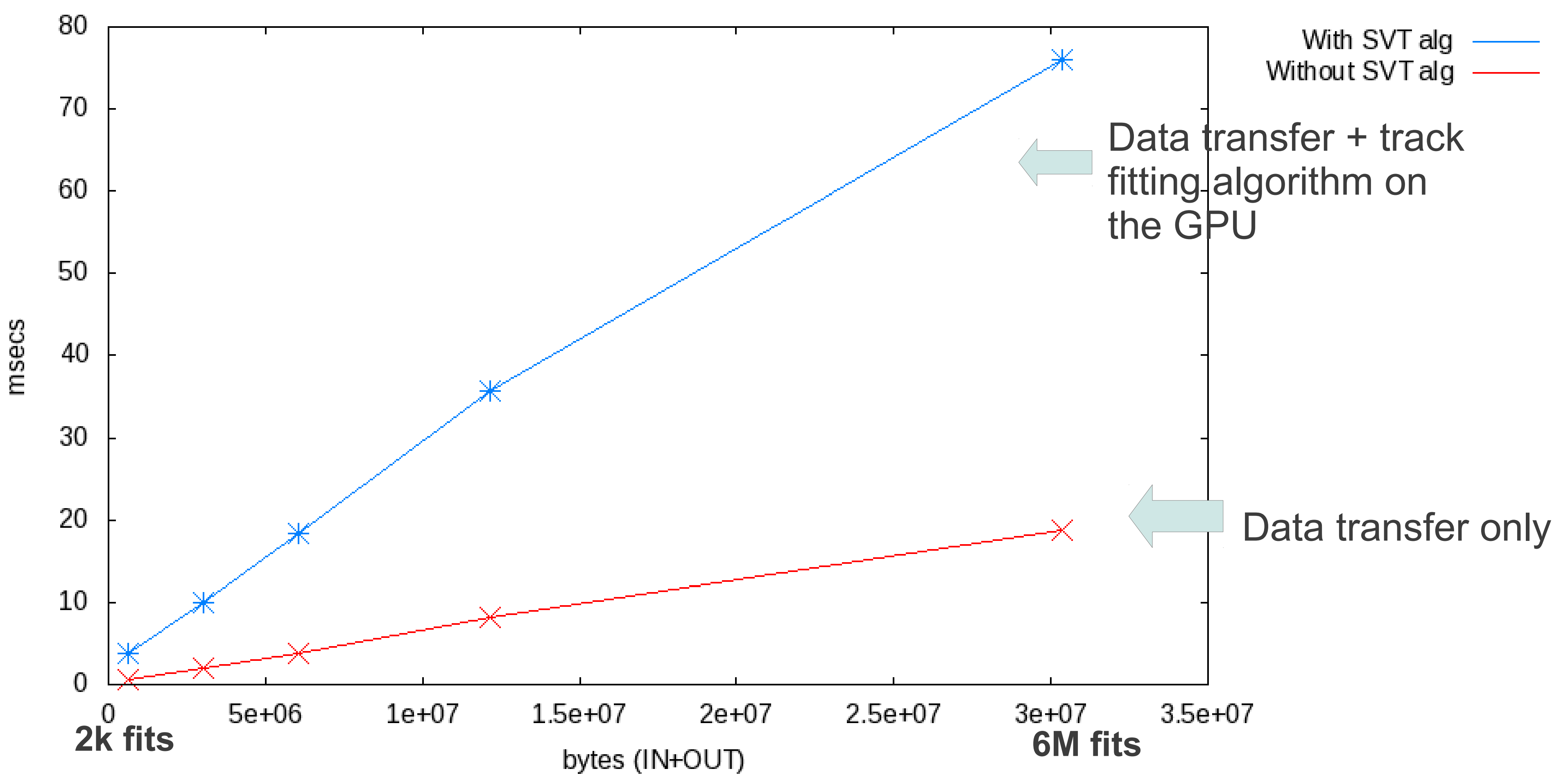
NB: P2P significantly better for small data sizes.

Each event has 3 kB in input and 57 kB in output → ~ 60 kB per event
Each point is the mean over 10k TX → RX → TX loops.

Data transfer: results (II)

How much of the total latency is due to data transfer?

Latency with and without SVT algorithm, for CUDA-Aware MPI



2k fits

6M fits

Data transfer + track fitting algorithm on the GPU

Data transfer only

Latency due to data transfer to/from GPU is about 20-25% of total.

Conclusions

There are different parallel architectures on the market; which performance can we obtain on parallel devices for a typical (parallel) real time selection algorithm? What can we get with current technology without re-thinking our codes from scratch?

Lessons learned

Porting

- With a native parallel code porting to CUDA not so easy but doable without help of experts. Easier to MIC using the *embarassingly parallel* approach

Data processing and transfer

- The maximum gain wrt CPU is obtained with millions of tracks fitted in parallel.
- Data transfer is a significant part of total latency

Data structures

- Limited memory on devices
- We worked with simple data structures: fixed size arrays (easier to handle on GPUs) and no empty events (no unoccupied threads) → careful organization of input data in a real application, where events have different sizes!

Next steps

This work is a starting point for future developments of accelerator-based trigger algorithms.

Very next steps

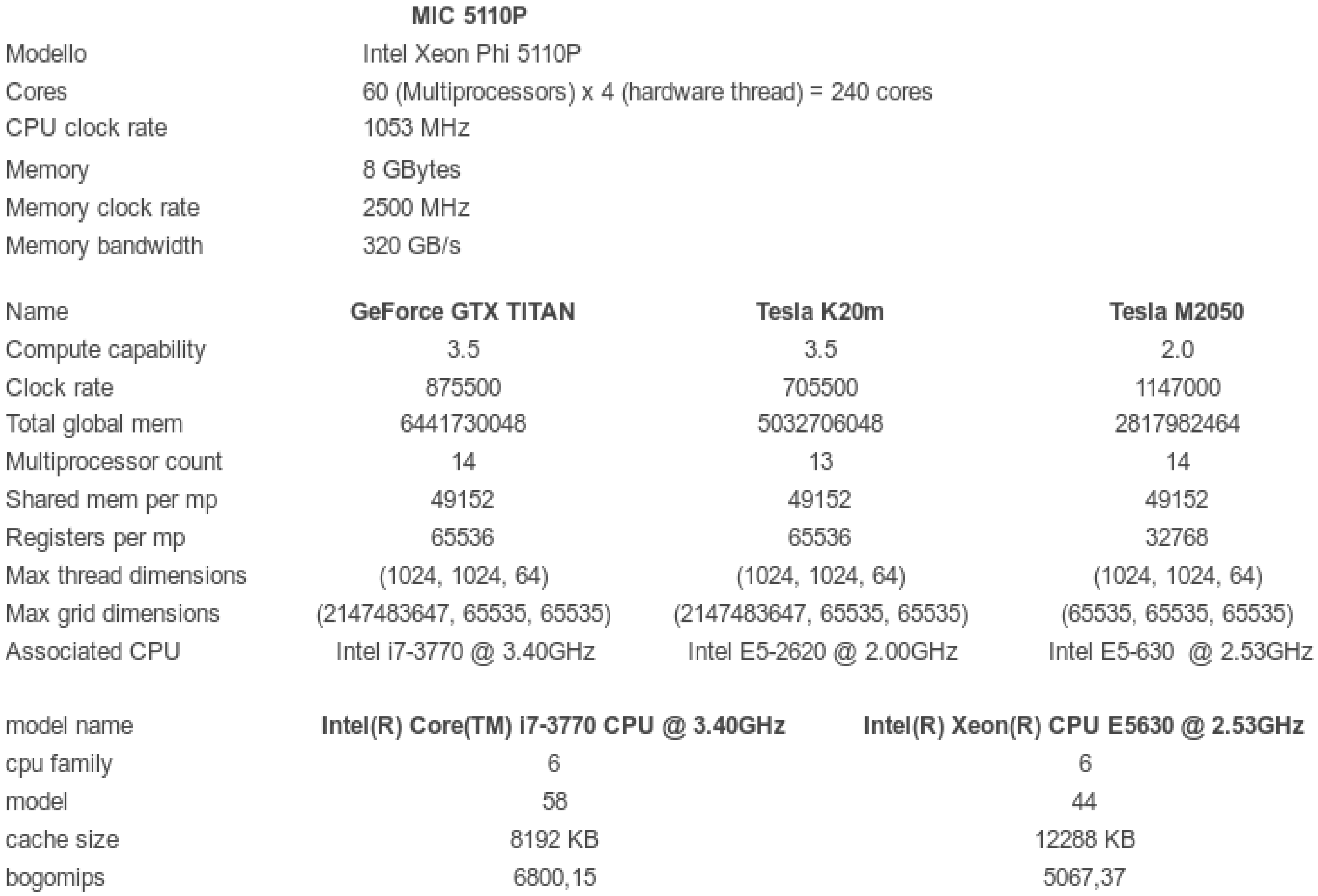
- *Implementation in OpenCL, comparison vs CUDA on Nvidia and AMD GPUs;*
- *Performance on Nvidia GPU paired to ARM processor*
- *Results will be shown at next NSS*

Possible future applications:

- LHCb and CMS high level trigger
- MicroBoone (calibrations and hit finding, 3D track fitting)

BACKUP

Hardware specifications

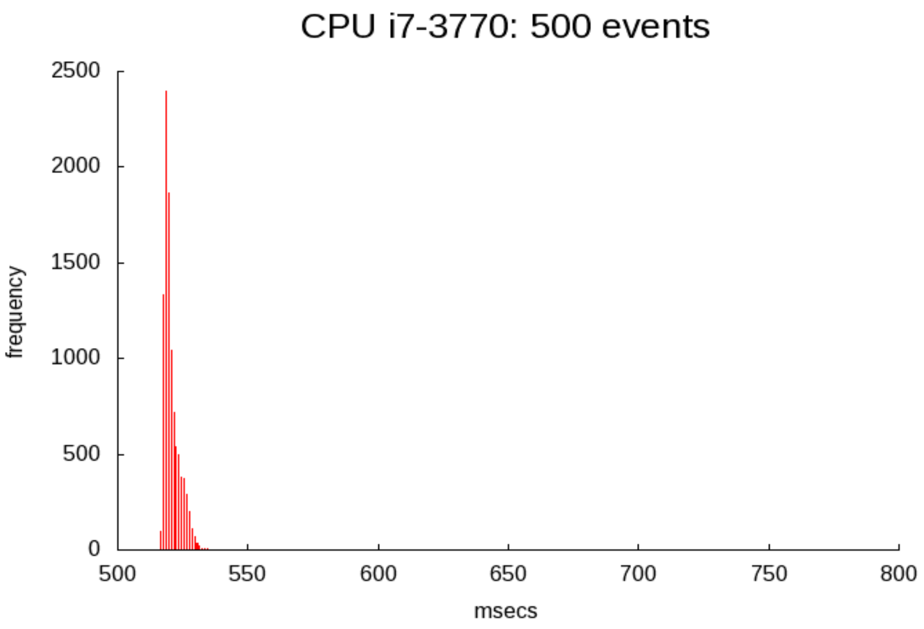


	MIC 5110P
Modello	Intel Xeon Phi 5110P
Cores	60 (Multiprocessors) x 4 (hardware thread) = 240 cores
CPU clock rate	1053 MHz
Memory	8 GBytes
Memory clock rate	2500 MHz
Memory bandwidth	320 GB/s

Name	GeForce GTX TITAN	Tesla K20m	Tesla M2050
Compute capability	3.5	3.5	2.0
Clock rate	875500	705500	1147000
Total global mem	6441730048	5032706048	2817982464
Multiprocessor count	14	13	14
Shared mem per mp	49152	49152	49152
Registers per mp	65536	65536	32768
Max thread dimensions	(1024, 1024, 64)	(1024, 1024, 64)	(1024, 1024, 64)
Max grid dimensions	(2147483647, 65535, 65535)	(2147483647, 65535, 65535)	(65535, 65535, 65535)
Associated CPU	Intel i7-3770 @ 3.40GHz	Intel E5-2620 @ 2.00GHz	Intel E5-630 @ 2.53GHz

model name	Intel(R) Core(TM) i7-3770 CPU @ 3.40GHz	Intel(R) Xeon(R) CPU E5630 @ 2.53GHz
cpu family	6	6
model	58	44
cache size	8192 KB	12288 KB
bogomips	6800,15	5067,37

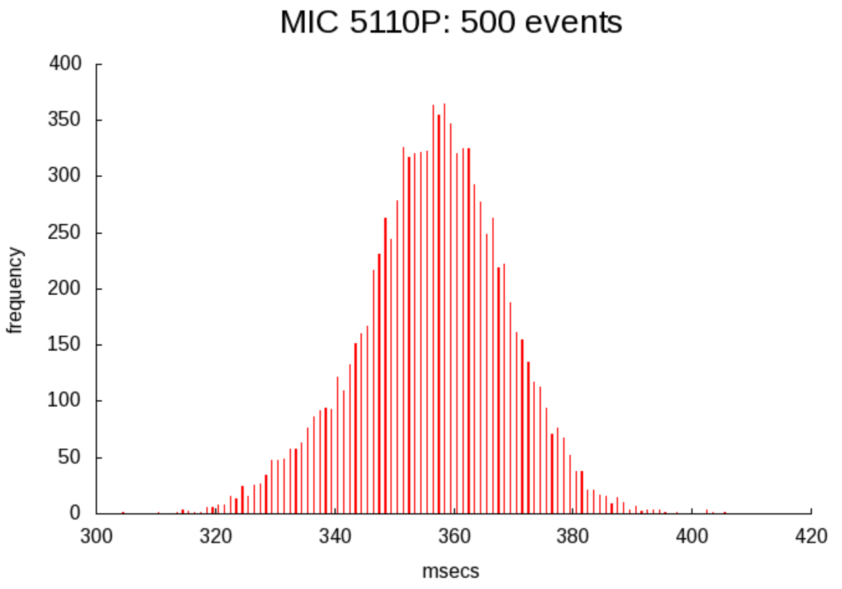
Latency distributions

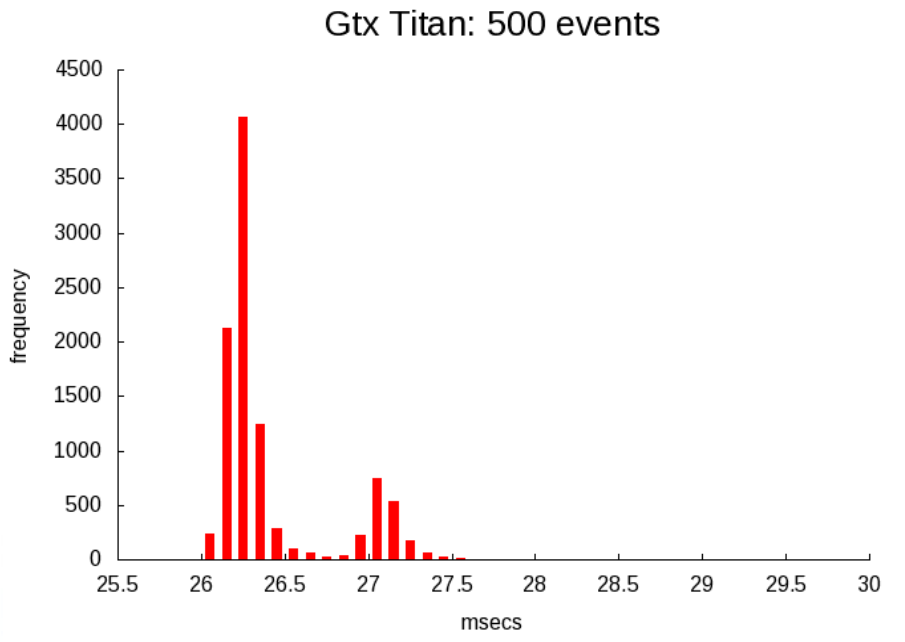


To study the stability over time we increased the number of *loops from 100 to 10k.*

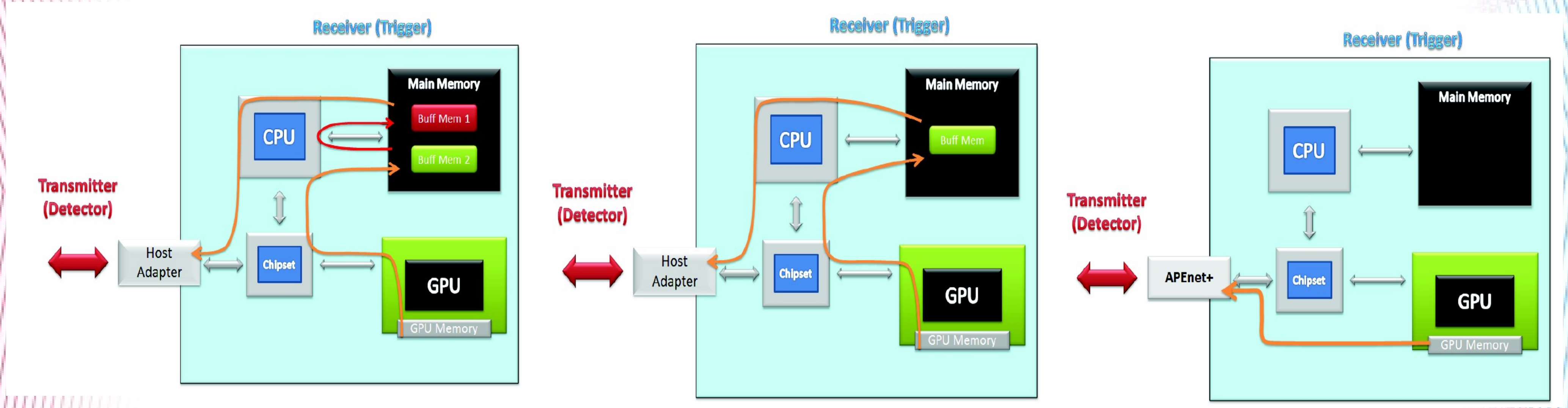
The distribution of all latencies is very narrow on the CPU, while it shows multiple peaks and long tails on GPU/MIC.

Probably due to CPU-GPU/MIC communication.

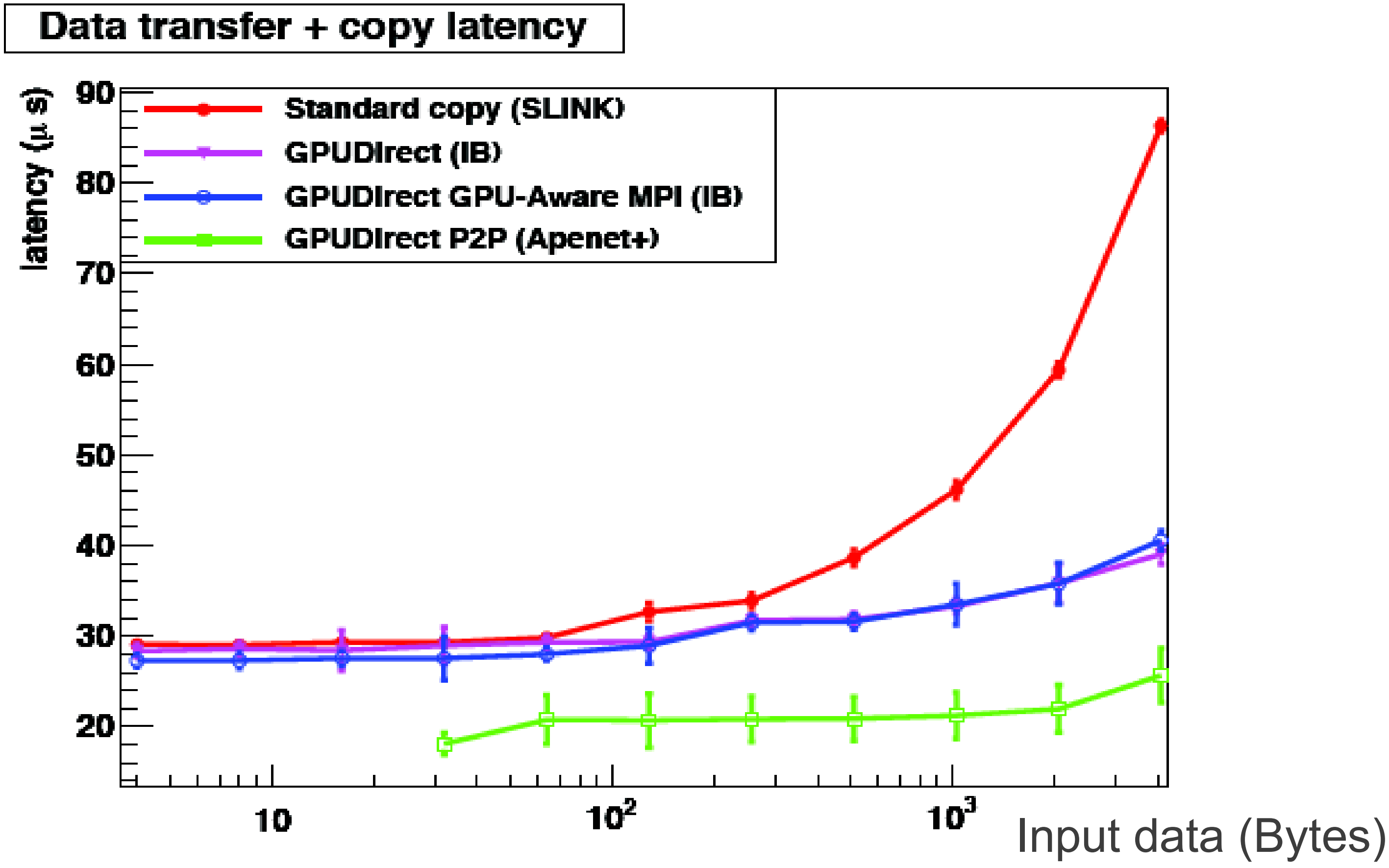




Data copy mechanisms



Data transfer latency for small data sizes


Data transfer + copy latency
latency (μ s)
Standard copy (SLINK)
GPUDirect (IB)
GPUDirect GPU-Aware MPI (IB)
GPUDirect P2P (Apenet+)
10^2
10^3
Input data (Bytes)