GPGPU Evaluation – Update Napoli

Silvio Pardi

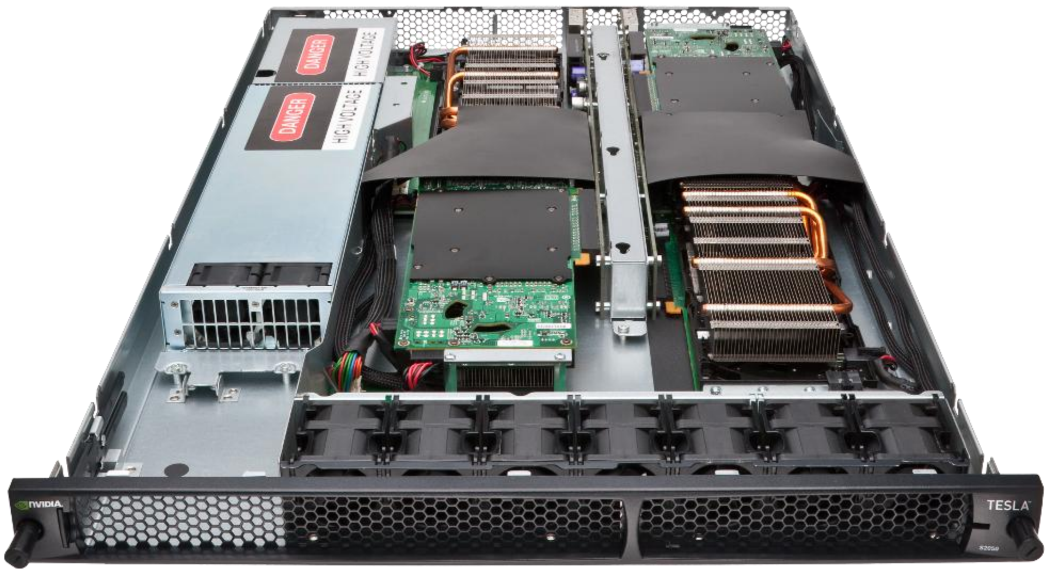
Goal of our preliminary tests

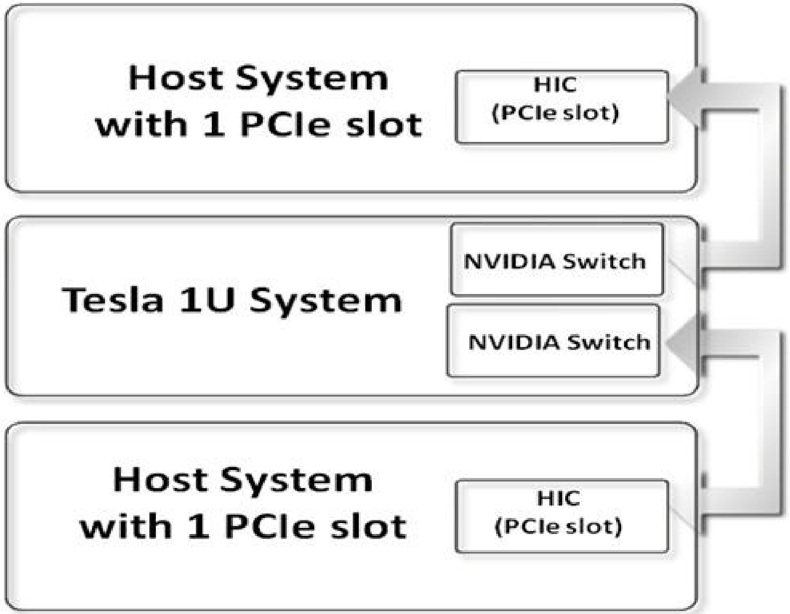
- Achieve know-how on the GPGPU architectures in order to test the versatility and investigate the adoption for some specific tasks interesting for SuperB.
- Create a list of test which we are interesting for.

- **multi-core Technology**
- High speed and complex processing unit
- General Purpose

- **many-core technology**
- Hundred of simple Processing Units
- Designed to macht the SIMD paradigm (Single Instruction Multiple Data)

The Hardware Available in Napoli





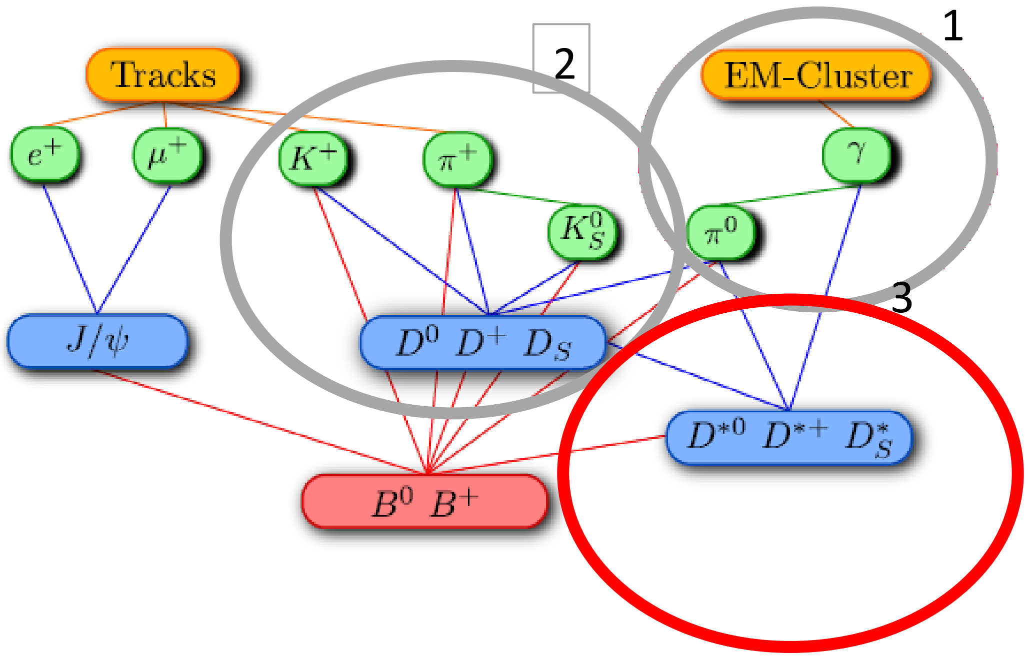
1U rack NVIDIA Tesla S2050

- 4 GPU Fermi
- Memory for GPU: 3.0 GB
- Core for GPU: 448
- Processor core clock: 1.15 GHz

2U rack Dell PowerEdge R510

- Intel Xeon E5506 eight-core @ 2.13 GHz
- 32.0 GB DDR3
- 8 hard disk SATA (7200 rpm), 500 GB

B-meson reconstruction algorithm



stage	particelle
1	tracks, K_S, γ, π^0
2	$D^{\pm}_{(s)}$, D^0, e J/ψ
3	$D^{*\pm}_{(s)}$ e D^{*0}
4	$B^{\pm}$ e B^0

Combinatorial problem

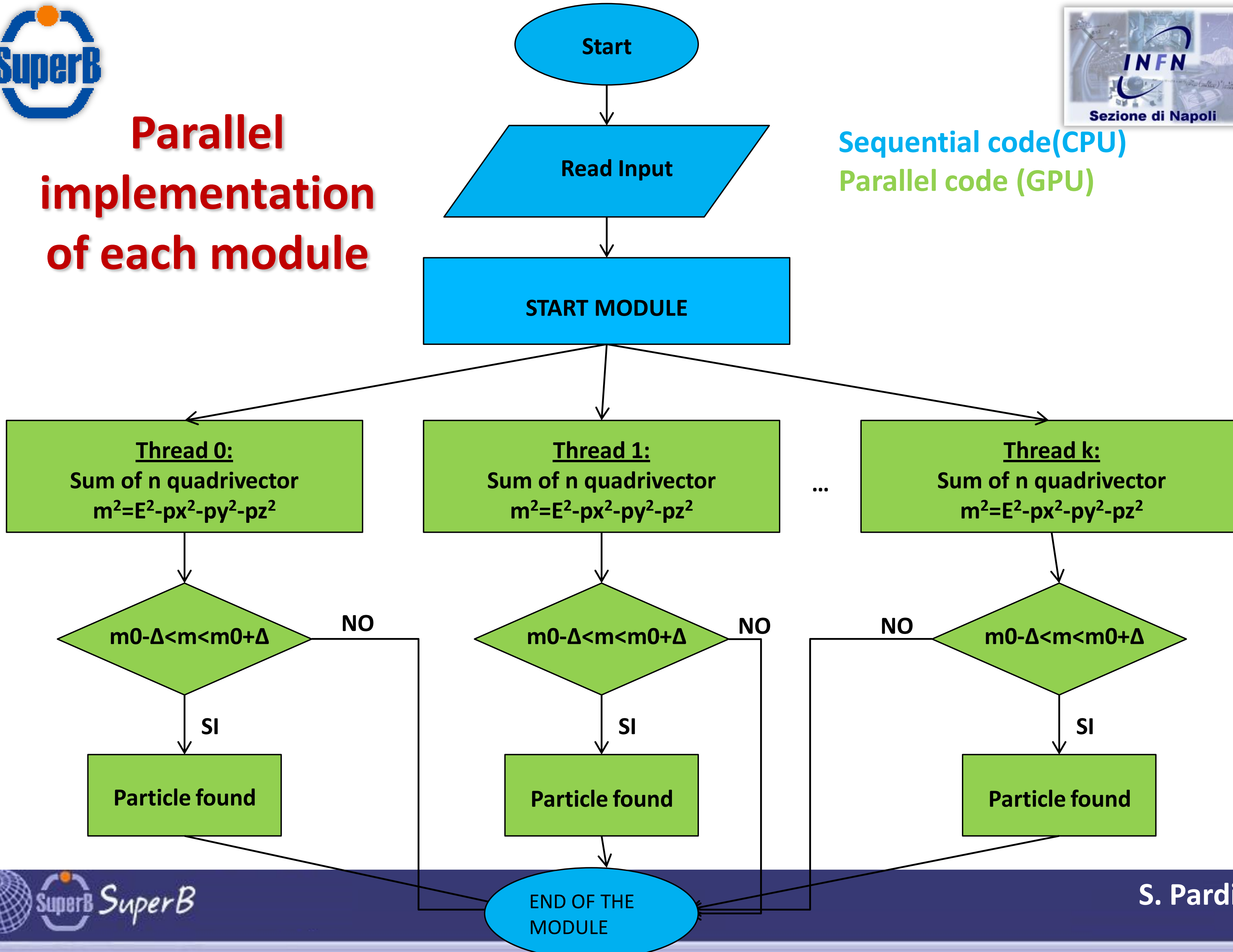
Problem Modellization: given N quadrivector (spatial component and energy), combine all the couple without Repetition. Then calculate the mass of the new particle and check if the mass is in a range given by input.

GOAL: Understand the impact, benefits and limits of using the GPGPU architecture for this use case, through the help of a toy-model, in order to isolate part of the computation.

A new stage of the algorithm (2 in figure) has been implemented by Master's Students.

Parallel implementation of each module

Sequential code(CPU)
Parallel code (GPU)



Methodology used in the test

- Memory in input
- CudaMalloc input
- CudaMalloc output
- Load input array
- cudaMemcpy() hostTOdevice
- cudaMemset()
- Kernel
- cudaMemcpy() deviceTOhost
- cudaFree()
- free()
- Total time

A sequential version of the toy algorithm has be implemented to compare the parallel one.

We increment the input in order to found the minimal data set in which the GPU algorithm overcome the sequential code.

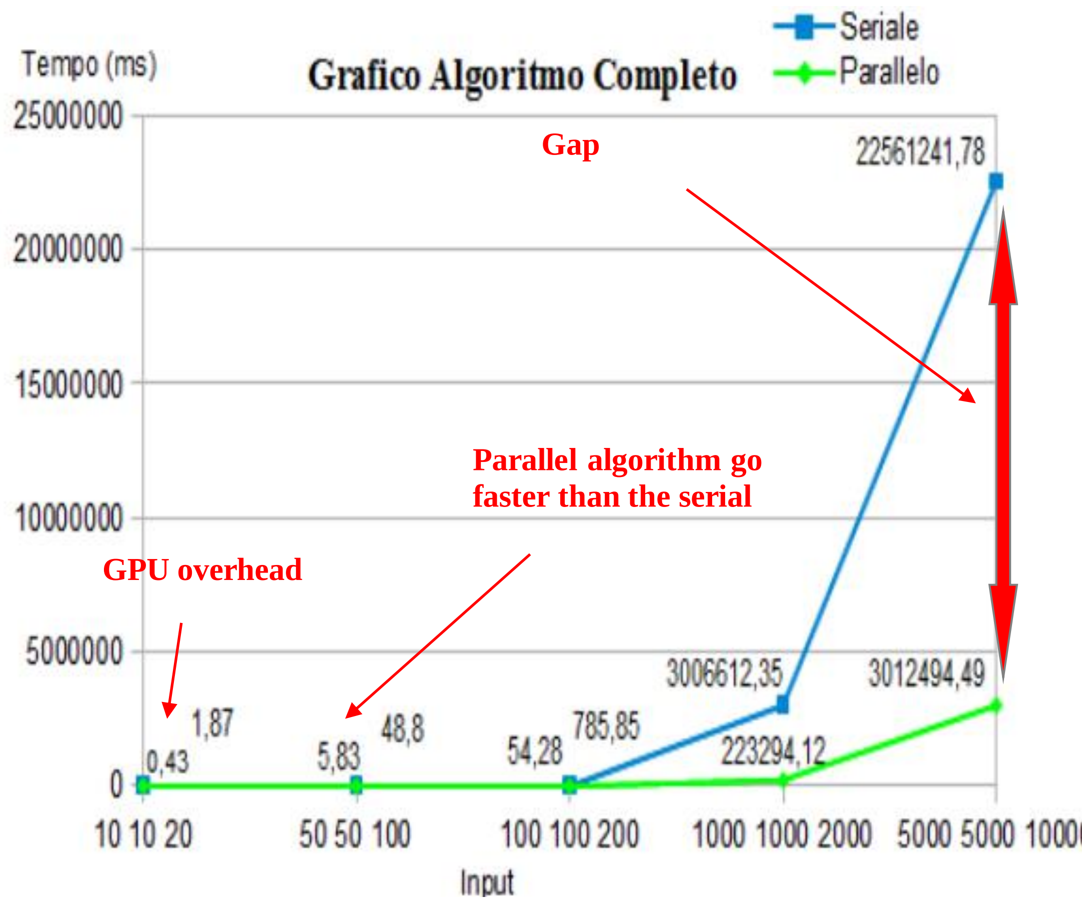
In testing the CUDA algorithm we measuring the time spent in any single activities in order to evaluate the overhead.

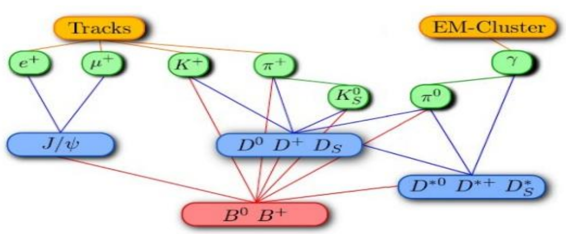
On-going Tests

- Code performance in term of time execution
- We start the code validation in term of reconstruction capability

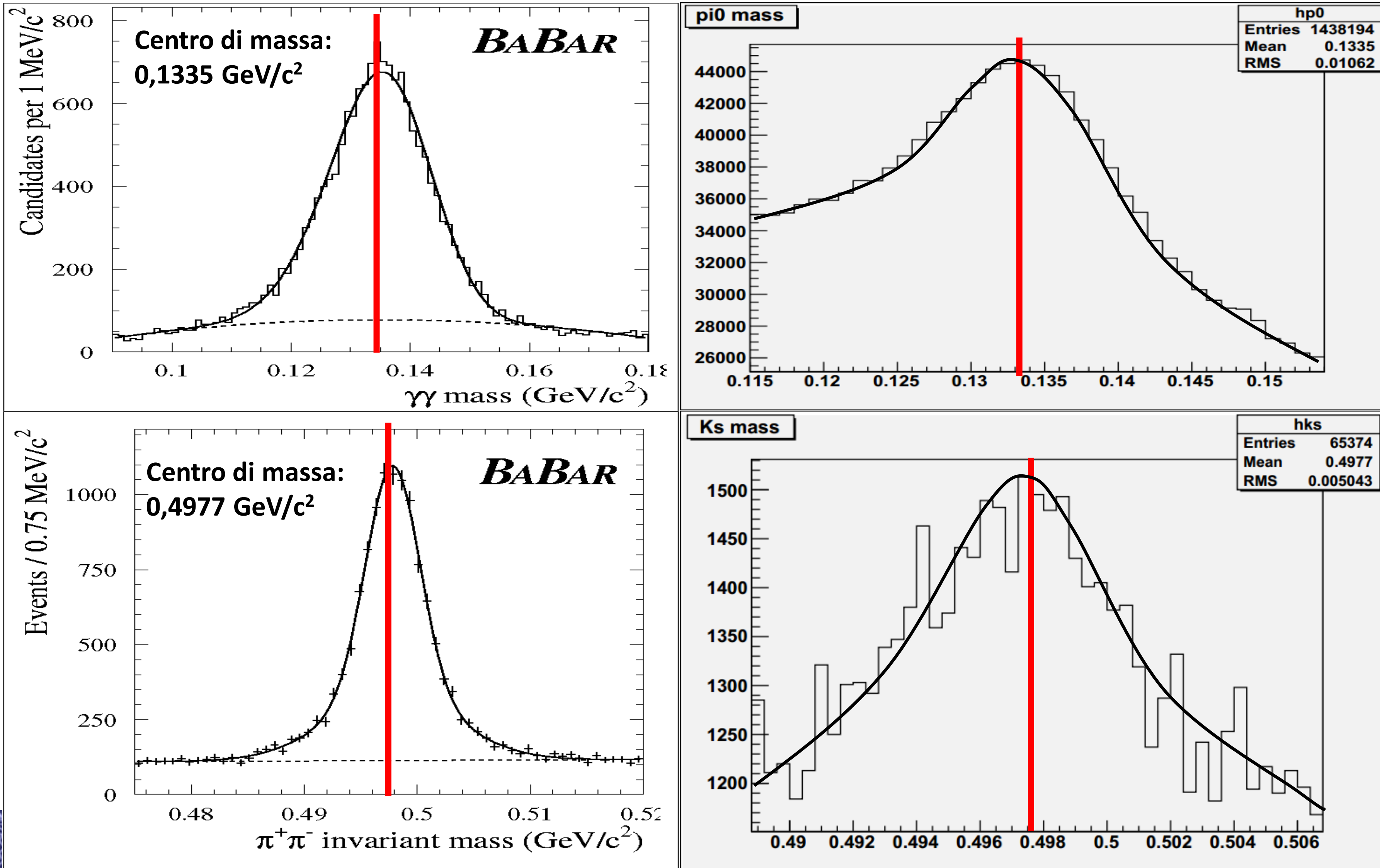
Algorithm Evaluation

Π+	Π-	Gamma	Seriale	Parallelo
10	10	20	0,43	1,87
50	50	100	48,8	5,83
100	100	200	785,85	54,28
1000	1000	2000	3006612,35	223294,12
5000	5000	10000	22561241,78	3012494,49





First Evaluation test on a set of real data (100.000 events)



Conclusion and future work

The GPU benefit is evident when we have a large input, this suggest the opportunity to process more event at the same time.

We must measure the GPU overhead with multiple jobs that work at the same time in the node that host the GPU.

We just started to evaluate the code reconstruction capability with real data. At this stage we used only a little number of particles we are confident that we can improve the showed graph.