Clustering algo for the NA62 GTK

GAP Meeting - Ferrara

M.Corvo

INFN and University of Ferrara

March 23^{rd} 2016

1. Introduction
2. GigaTracker Reconstruction algorithms
    - Clustering
    - K-means
3. Implementation
    - Implementation on GPU
4. Results
    - Performances
    - Drawback
5. Conclusions
    - Improvements

Motivations and objectives

Motivations

Speed up the algorithms for the reconstruction of tracks in the GigaTracker

Means

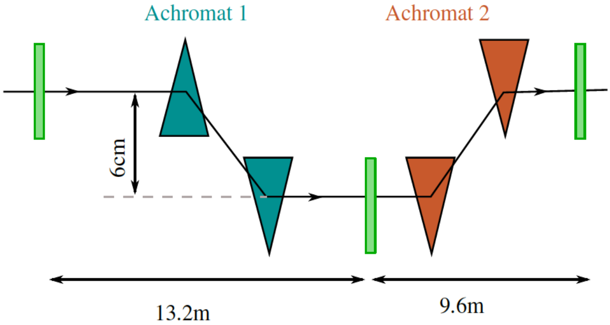
Exploit the execution of algorithms in hardware accelerators (GPGPUs, Intel Phi, multi core CPUs)

Goals

Prove that we can gain a (possibly) significant speedup in algo execution

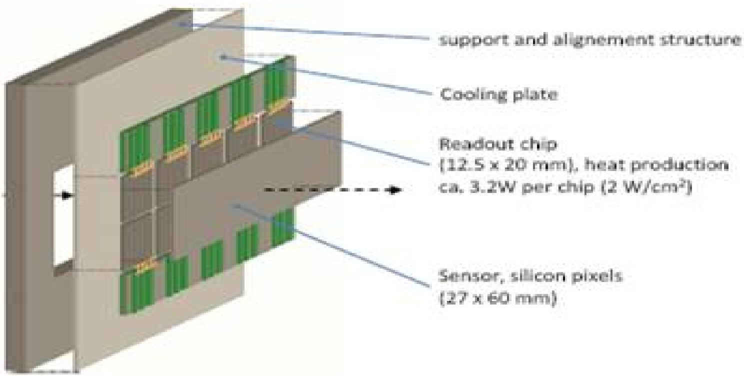
The detector

The GigaTracker detector is made of three silicon pixel stations and its aim is to measure time, direction, and momentum of all the beam tracks ($\sim 10^9 sec^{-1}$)



GigaTracker

The detector II



Particular of one station

Clustering

- Generally speaking we define clustering as a mechanism by which we group points in a data set in a way such that points within each cluster are similar to each other
- In detectors that record spatial points, the similarity can be defined via a (Euclidean) distance measure

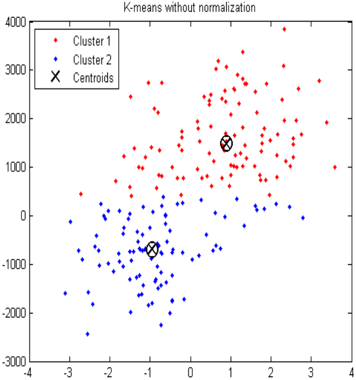
Currently the reconstruction code of the GTK starts by creating clusters in each station by means of two nested for loops, which implies a (worst case) computational complexity of n^2, where n is the number of hits in a given event.

Idea

Delegate n threads to perform the calculation for each hit.

Clustering Algo

Consider this image



Spatial points

To aggregate the spatial points we can exploit an algorithm called **k-means** The idea behind this algorithm is:

1. provide an initial number of possible clusters
2. assign the coordinates of the centroid to each of the initial cluster
3. calculate the distance between each spatial point and the cluster centroids
4. assign to each cluster those spatial points whose distance from the centroid is less than a given threshold
5. recalculate the centroid taking into account the contributions from all the points that now belong to the cluster
6. goto 3 unless there are no more changes either in the membership of hits or in the cumulative difference in distance

K-means algo adaptation

- The **k-means** is an iterative algorithm which needs the (supposed) number of clusters around which all the spatial points will 'aggregate'
- This is clearly a limitation, as we cannot forsee how many clusters we'll have
- This is why, in my implementation, every hit in each GTK station is considered a potential cluster

Changes in the code and GPU implementation

Some changes were needed to implement the GPU version of k-means:

- For best cache performance, two arrays for x and y coordinates were added in class TRecoVEvent
    - Some methods of the class had to be modified accordingly
    - Plan is to find a 'cleverer' way to manipulate CUDA data structures avoiding adding new methods by menas of inheritance
- One new simpler class *Cluster* containing the cluster centroids, hits and hits id

Setup

Tests have been performed on a GTX Titan with

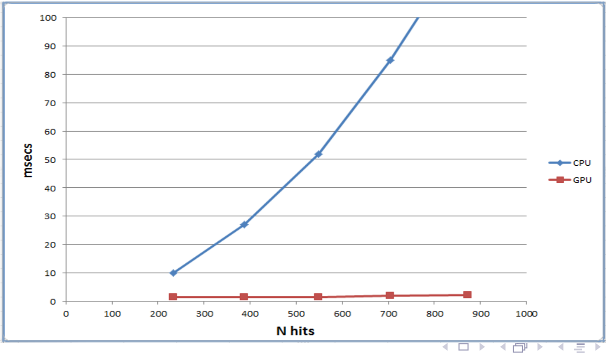
1. 14 MP
2. 875 MHz
3. 6 GB ram memory
4. 1024 maximum thread per block

mounted on a Intel®CPU

1. 4 cores (8 with HyperThreading)
2. 1.6 GHz
3. 32 GB ram memory

Performance

Hits	Memory setup (GPU in ms)	Clusterization time (CPU in ms)	Clusterization time (GPU in ms)
233	6.52	10.	1.69
388	0.11	27.	1.47
548	0.14	52.	1.47
705	0.12	85.	1.46
873	0.42	127.	1.38



Results

- The speedup is evident by the execution time of the two algorithms
- This is only one part of the story though
- The algorithm doesn't make any distinctions between clusters which are 'clones' of each other
    - Clones must be killed
- Clone killing is performed using the *Thrust* libraries

Clone killing

- **Thrust** is a CUDA library similar to the C++ standard one
- It consists of both containers (like std :: vector, std :: tuple, . . .) and algorithms (like std :: sort, std :: find, . . .)

- The easiest way to detect and delete clones is to std :: sort the list of clusters and then std :: unique them
- The drawback of this further computation is a degradation of performances

Performance with clone killing procedure

Hits	Memory setup (GPU in ms)	Clusterization time (CPU in ms)	Clusterization time (GPU in ms)	Clone killing time (ms)
233	6.52	10.	1.69	4.25
388	0.11	27.	1.47	4.42
548	0.14	52.	1.47	5.39
705	0.12	85.	1.46	6.08
873	0.42	127.	1.38	7.45

Improvements

Room for improvements is pretty wide. On GPU side:

- Use of constant memory for hits
- Use of streams to hide memory latency

On code side

- Port of track fitting to complete the tracking sequence

Conclusions

- Use of GPU to speedup serial code is well know
- Though common approach is 'brute force' fashion
    - This means 'Give me as many threads as you can and perform a task in each thread'
- This can be affordable, but has some drawbacks, as seen with clone killing procedure which must be fired after clusterization
- Speedup is always wellcome, but beware of Amdahl's law

$$\frac{1}{(1-F)+\frac{F}{N}} \tag{1}$$