Performance Study of a GPU in Real-Time Applications for HEP Experiments

Stefano Gelain
(University of Padua)

Supervisor Ted Liu - Mentor Silvia Amerio

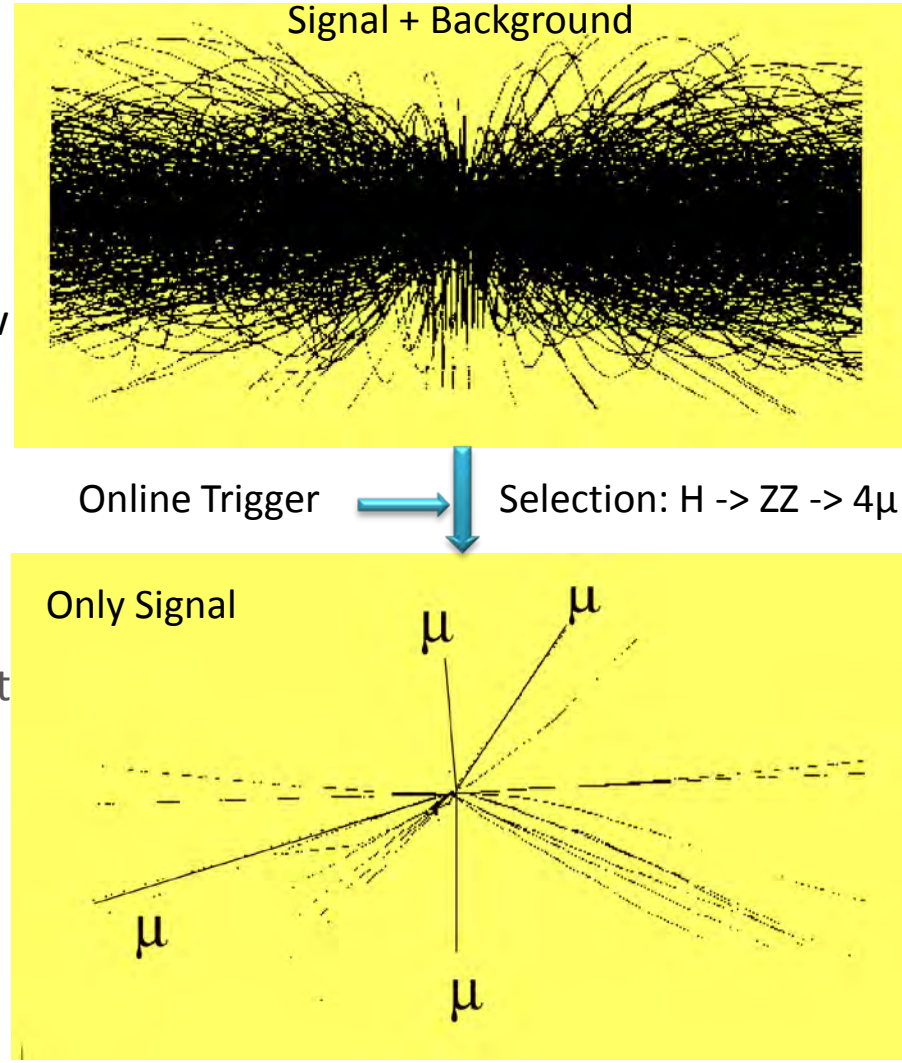
Introduction

Importance of online selection (Trigger) for selection of interesting events in HEP Experiments

Usually Trigger systems especially in low latency environments, are based on dedicated hardware:

- very powerful
- but expensive (custom made)
- requires lot of manpower and expertise for development and support

GOAL of this study:
Can we think of using a commercial device as a GPU to develop online selections in a low-latency environment?



Ex: simultation of tracker data reduction at CMS

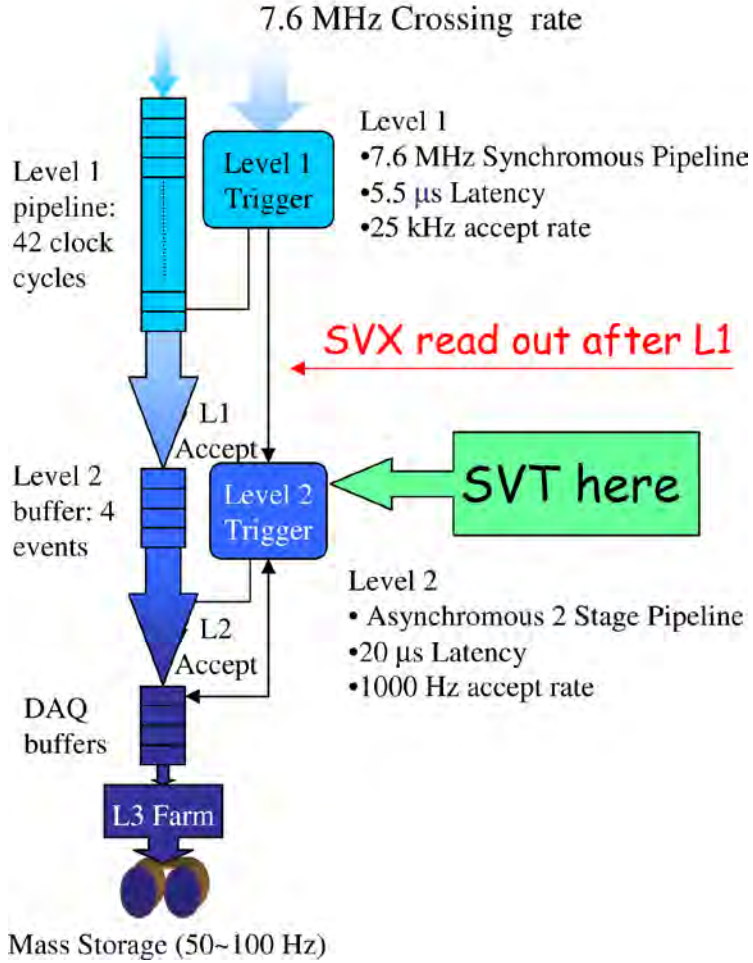
Graphics Processing Unit

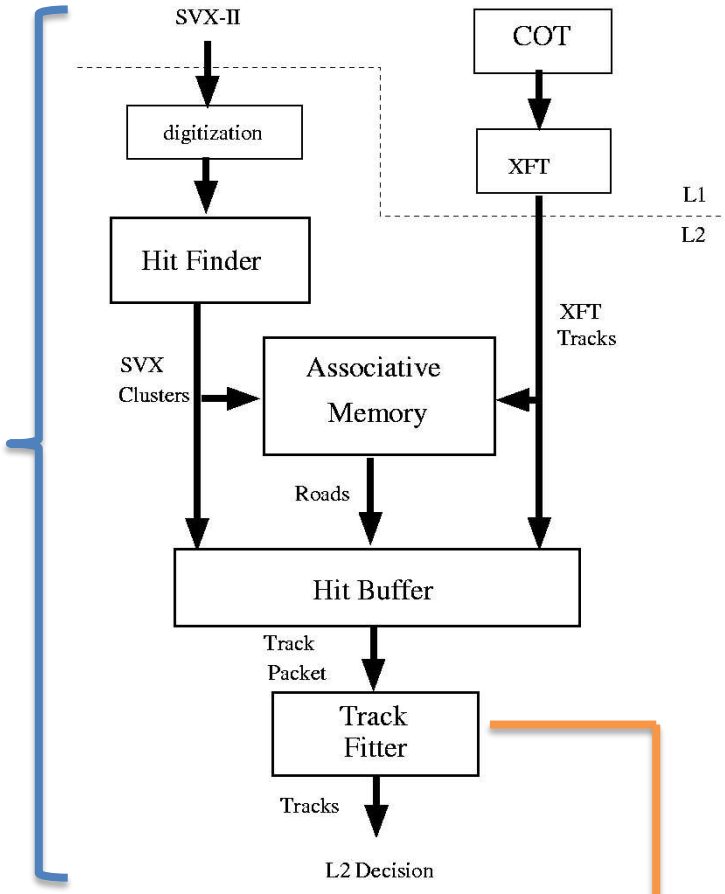
Why GPU?

- **Power of GPUs has increased rapidly due to demands of 3D graphics (and computer games)**
  - **Highly parallelized architecture**
  - **High memory bandwidth**
- **They are**
  - **Commercially available ➔ cheaper than dedicated Hardware**
  - **Application programming interfaces like nVidia's "CUDA C" ease development of software for new applications**

Photograph of GTX 285 GPU courtesy of nVidia

The reference point - Track-fitting in SVT





Our reference is the trackfitting algorithm implemented in GigaFitter dedicate board

The Computation: Linearized Track Fitting

- **Want to run algorithm that would be used in HEP trigger**
  - **Perform track-fitting inside roads using simple scalar product**

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

- $\vec{x}$ — Track coordinates (input hit information)
  Copied from **host** to **device**
- $\vec{f}_i$ — *Known constants*
  *Precalculated and **stored in memory***
- p_i — *Track parameters (output)*
  *Copied from **device** to **host***

The GPU run a simple scalar product to simulate the track-fitting algorithm

Track coordinates and constant are simple random words and numbers

Experimental Setup

GPU has been never used for HEP experiment!

This is a preliminary study of:

- **GPU architecture and operational capability**
- **Feasibility for a future Track Fitting system**

Experiment consists in time latency measurements

Two experimental setups with two different GPU:

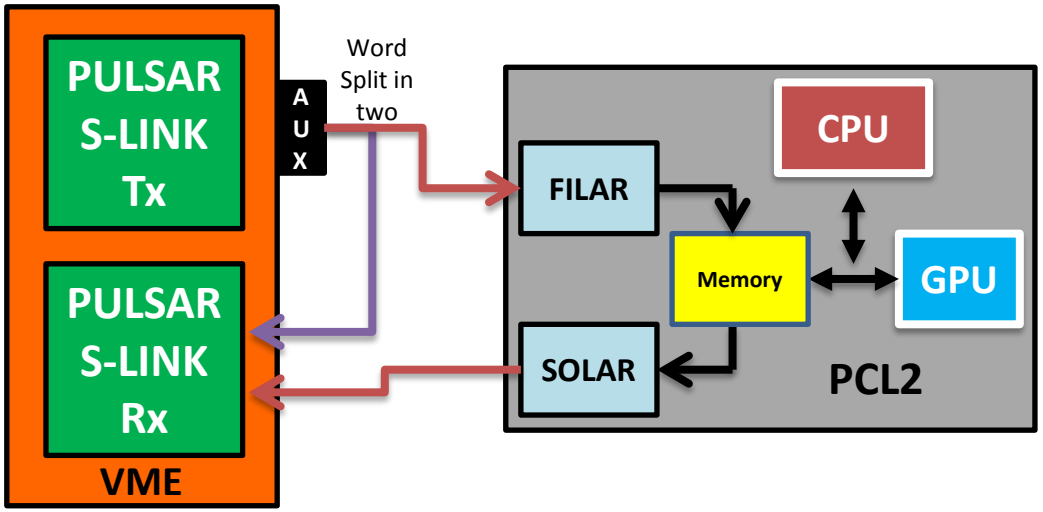
PC Level 2 - Desktop + Crate - GPU GXT 285.

At Fermilab
High precision time detection external setup. Independent to PCL2
Precision ~0,1 μs

CLIO - Desktop - GPU GXT 580 (more powerful)

At Padua (Italy) (remote controlled by ssh)
Not indipendet time detection setup. Latency measurement are obtained using internal GPU function (Time Stamps)
Precison ~0,01ms.

PCL2 Experimental Setup



CRATE VME

- **Transmitter**: PULSAR (PULSer And Recorder) board
- **Receiver**: PULSAR (PULSer And Recorder) board

Desktop PC-L2

- **Filar**: S-Link data received (Four Input Links for Atlas Readout)
- **Solar**: S-Link data send (Single Output Links for Atlas Readout)
- **GPU**: (Graphics Processing Unit)

PCL2 Experimental Setup

DATA FLOW

- Receive input data on FILAR from PULSAR (Transmitter)
- Copy input data (track coord.) from Filar and PC into GPU (device)
- Perform calculations:
  - Retrieve constant set used for evaluating fit parameters
  - Run (linear) track fitting algorithm to calculate fit parameters
- Copy results from GPU
- Store calculated parameters (and internal timing info) to be sent on SOLAR
- Send output to PULSAR (Receiver)

Target: *Determine latency and total time for performing a HEP trigger algorithm from input going into the PC (t1) and the output leaving the PC (t2) and determine latency (t2 – t1)*

Preliminary Studies

Study of the GPU structure to optimize the configuration for the track-fitting and improve performance

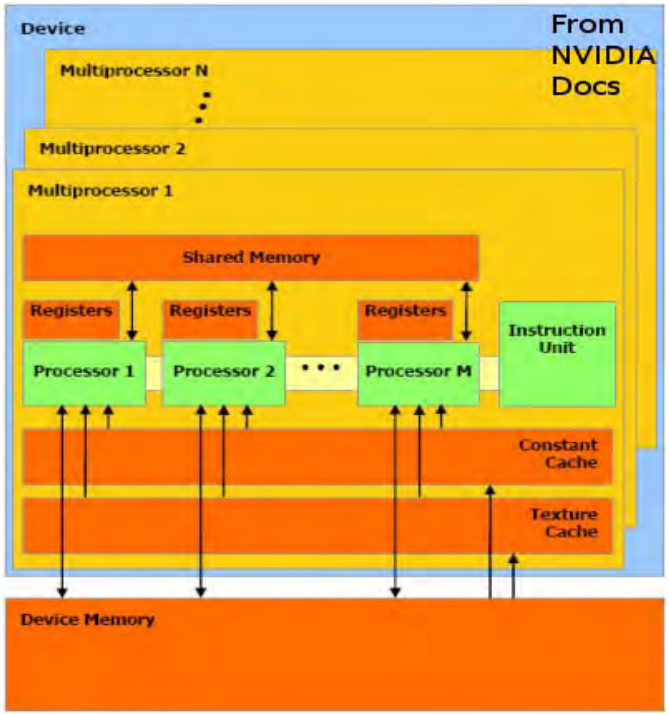
My Analysis :

- Memories to store constants
- Communication HOST – DEVICE (Data access)

  Copy data input on the GPU to perform calc.

  and copy results to the HOST

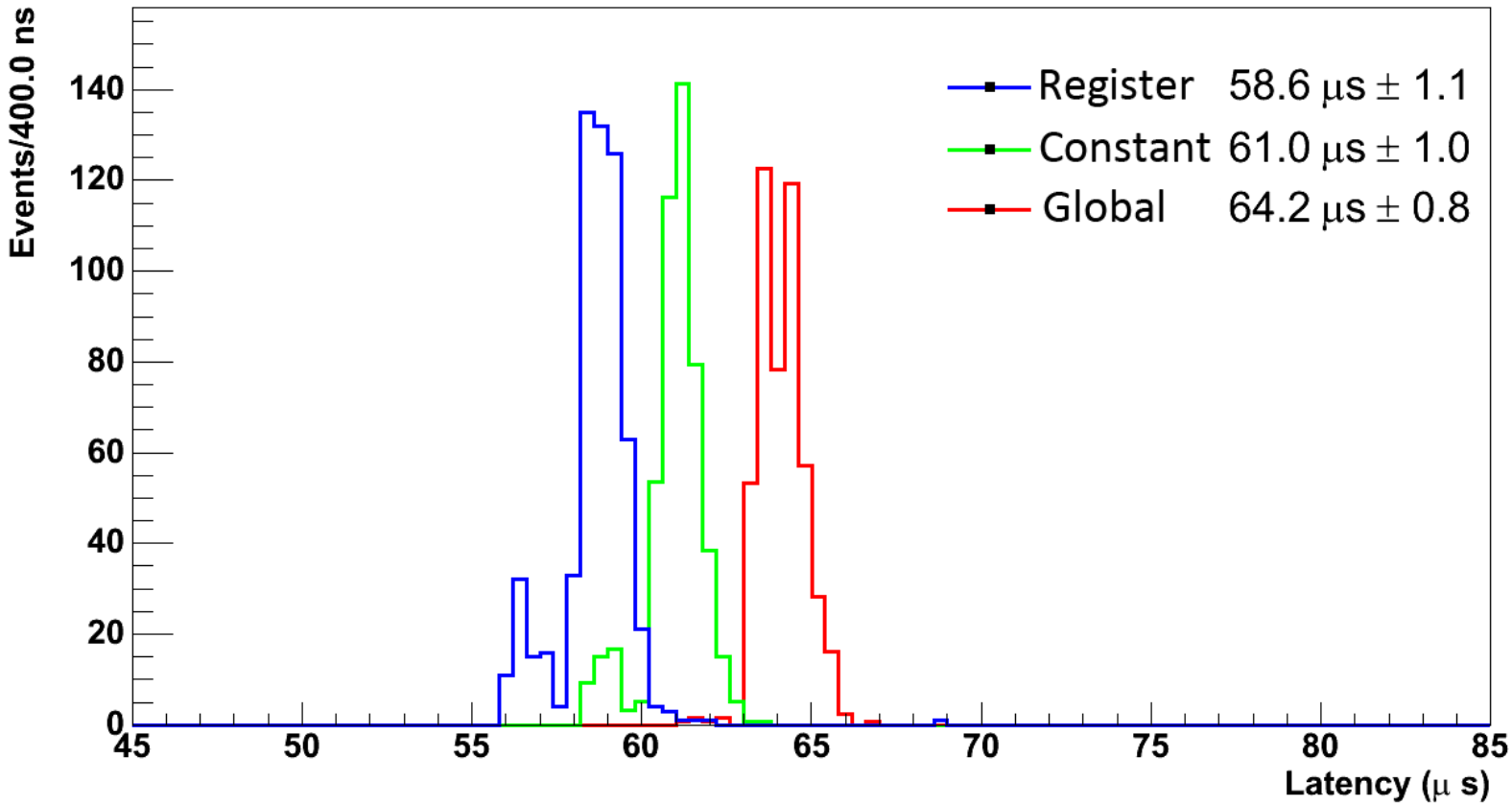
  (higher latency ~30 μs)
- Optimization GPU configuration for calculations
  - configuration Threads/Blocks



GPU structure

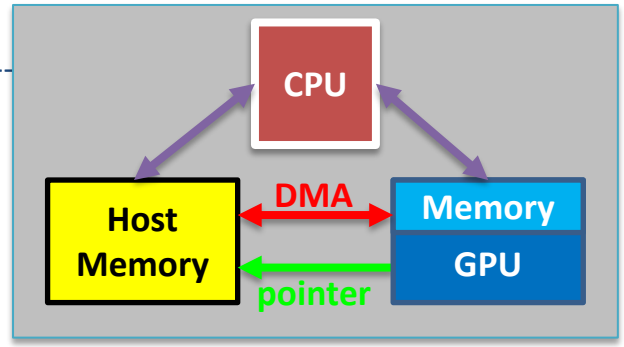
Latency test for Costants

Comparison between total times using three different memory allocations for costants, with same conditions



Register memory comes out to be the fastest way to allocate constants

Memories to Data Access



PAGEABLE MEMORY

- *High storage*
- *Easy and stable to use*
- *Most common*
- *Slow access to data from device* → GPU must talk to CPU

PAGE-LOCKED (Pinned) MEMORY

- *Very small*
- *Handle with care*
- *Fast access to data from device* → GPU copy directly from/to host using DMA without talking to CPU

ZERO COPY (Mapped P.L.) MEMORY

- *Very small*
- *Handle with care*
- *The fastest* → GPU operate directly on host mem, without cpy on device memory

Memories to Data Access – Test on CLIO

GXT 285 does not support ZeroCopy

Comparison between total times for the three different data access

Number of words in input	Pageable			Page Locked			Zero Copy		
	Mean (ms)	Err (ms)	Err %	Mean (ms)	Err (ms)	Err %	Mean (ms)	Err (ms)	Err %
512	0,23	0,02	8,97	0,16	0,01	6,97	0,06	0,01	12,75
1024	0,23	0,02	6,88	0,16	0,01	7,35	0,07	0,01	13,32
2048	0,26	0,02	6,22	0,17	0,01	7,13	0,07	0,01	11,23
4096	0,32	0,02	5,09	0,19	0,01	7,39	0,09	0,01	9,01
10240	0,49	0,02	4,43	0,25	0,01	5,73	0,14	0,01	7,21
76800	2,28	0,05	2,21	0,88	0,04	4,07	0,96	0,04	4,28
102400	2,97	0,09	3,00	1,11	0,06	5,80	1,27	0,08	6,25
512000	10,13	0,48	4,72	5,01	0,34	6,77	6,23	0,42	6,79

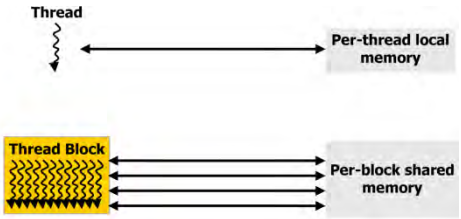
Zero copy is the fastest for low number of data input

For high number of data input **Page-Locked** is faster

GPU COMPUTATIONS STRUCTURE

GPU Hardware

STREAM MULTIPROCESSOR: *15 (16) for GXT 285 (580)*
PROCESSOR (CORE): *240 (512) for GXT 285 (580)*

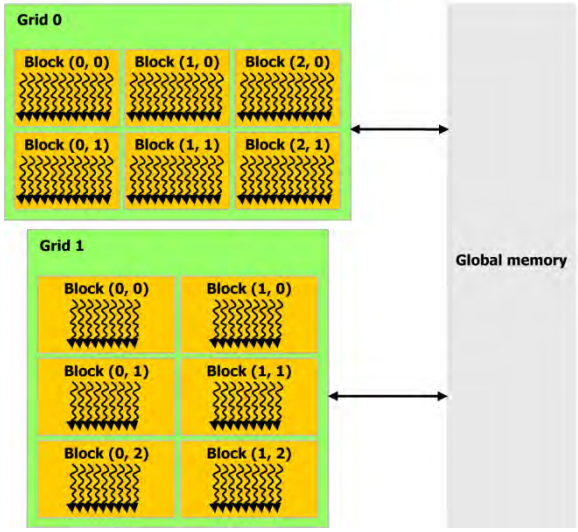


GPU Computation Structure

KERNELS: *Functions that, when called, are executed N times by N different THREADS.*

THREADS:
Launching a kernel creates a grid of threads that all execute the kernel function.
Each threads is processed by a core (hide to the programmer)

BLOCKS: *Threads are grouped in Blocks.*
Each block is processed by a single multiprocessor (hide to the programmer)



GPU organization and memory hierarchy

GRIDS: *Blocks are grouped in Grids. Each grid executes only one kernel at a time*
Grid is executed by different multiprocessor (hide to the programmer)

Rules for Threads/Blocks Configuration

- Number of Blocks > Number of Streaming Multiprocessor
  - Each SM has at least one Block to execute
- Number of Blocks / Number of SM > 2
  - Many Blocks can run concurrently on a SM
  - Work on another Block if one Block is waiting on barrier
- The number of Threads per Block should be a multiple of 32 (warp size)
- Want as many warps running as possible to hide latencies
  - Maximum number of Threads in a Block (for GXT 285 < 512)
  - Maximum number of Blocks in a Grid (< 65535)

Using the better configuration latency can be reduced of 40% both on CLIO and on PCL2

Conclusions

GPUs are suitable for Real Time computing!

- **are optimized to perform highly parallel computation**
- **well fit for different purposes (very flexible)**

For low latency environments they have some limitations:

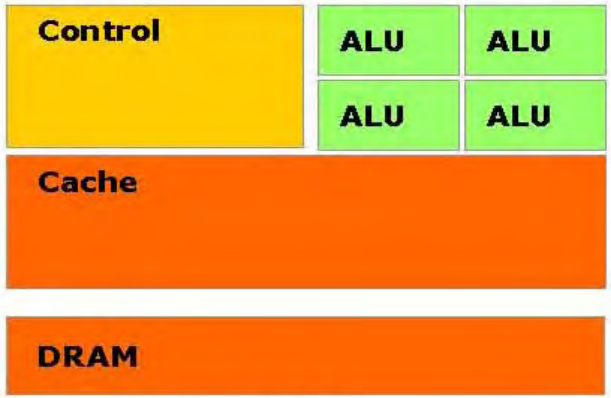
- **Slow latency for host↔device communication**
  - Computational gain must be worth what you pay to move data host ↔ device.
- **Sensitivity to memory access requires full comprehension and careful optimization**

We don't know yet if GPUs are suitable for track fitting (low latency environment)
However there is margin to increase performance!

CPU Computation vs GPU

CPU (Intel Core i7-930)

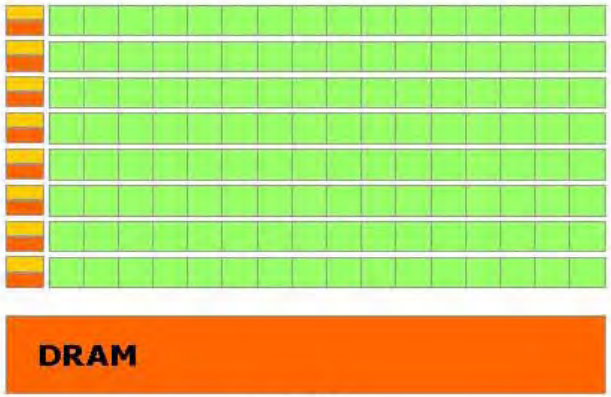
- Limited number of simultaneous calculations possible
  - 1 microprocessor
  - 4 cores
  - 8 threads
- Large cache size
  - 8 MB
- Sits directly on motherboard
  - Latency scale set by number/ speed of operations



CPU

GPU (nVidia GeForce GTX 285)

- Designed for running many instances of same routine simultaneously
  - 30 microprocessors
  - 240 cores
  - 1024 x 30 threads (max)
- Small cache size
  - 8 kB / microprocessor
- Talks with CPU through PCIe bus
  - Latency scale set by communication host (CPU) ↔ device (GPU)



GPU