Study on the performance of the ATLAS TopoCluster algorithm using GPGPU Acceleration

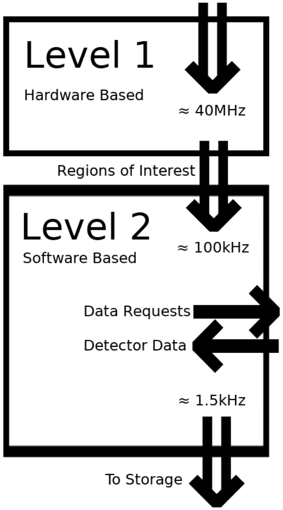
Supervisor: Dra. P. Conde Muíño

Eduardo Ferreira

LIP · IST

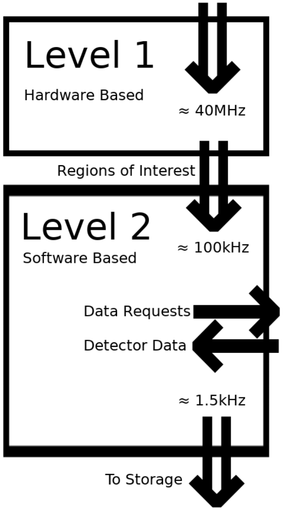
05 Setember 2018

ATLAS Trigger and Data Acquisition System



- The LHC @ CERN provides a bunch crossing rate of $\approx$ 40Mhz and 22 collisions per bunch crossing.

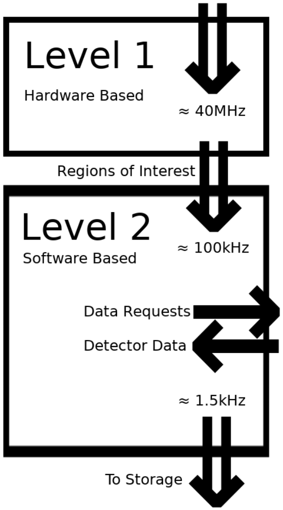
ATLAS Trigger and Data Acquisition System



- The LHC @ CERN provides a bunch crossing rate of $\approx$ 40Mhz and 22 collisions per bunch crossing.

1. 1st Level - Hardware Based - Reduces the 40MHz event rate to $\approx$ 100kHz

ATLAS Trigger and Data Acquisition System



- The LHC @ CERN provides a bunch crossing rate of ≈ 40Mhz and 22 collisions per bunch crossing.

1. 1st Level - Hardware Based - Reduces the 40MHz event rate to ≈ 100kHz
2. 2nd Level - Software Based - Algorithms run on the previously selected events - Reduces to a rate of ≈ 1.5kHz to be stored for further processing

Motivation

- LHC will undergo a two phase update to increase its luminosity:
  1. 1st Phase (2019-20)
  2. 2nd Phase (2024-26)

  This will dramatrically increase the number of collisions per bunch crossing.

Motivation

- LHC will undergo a two phase update to increase its luminosity:
  1. 1st Phase (2019-20)
  2. 2nd Phase (2024-26)

  This will dramatrically increase the number of collisions per bunch crossing.
- Hi-Lumi LHC will further increase the amount of data collected per run.

Motivation

- LHC will undergo a two phase update to increase its luminosity:
  1. 1st Phase (2019-20)
  2. 2nd Phase (2024-26)

  This will dramatrically increase the number of collisions per bunch crossing.
- Hi-Lumi LHC will further increase the amount of data collected per run.
- Computational power is constrained (by power, area, heat dissipation...) - Need to find new solutions to increase computational throughput.

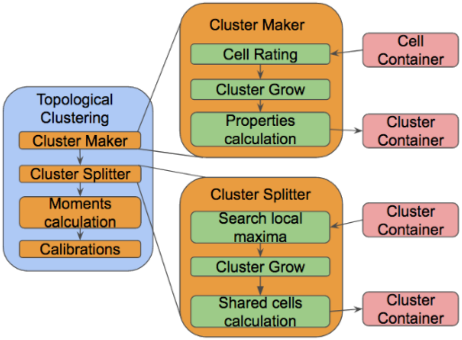
Motivation

- LHC will undergo a two phase update to increase its luminosity:
  1. 1st Phase (2019-20)
  2. 2nd Phase (2024-26)

  This will dramatrically increase the number of collisions per bunch crossing.
- Hi-Lumi LHC will further increase the amount of data collected per run.
- Computational power is constrained (by power, area, heat dissipation...) - Need to find new solutions to increase computational throughput.
- One particular alternative is being considered: Parallel processing using Nvidia GPGPU's using the Nvidia CUDA Techonology

Topological Clustering

- Our main focus: Accelerate the performance of the Topological Clustering Portion of the HLT
    - This handles the reconstruction of particle jets by grouping calorimeter cells into structures named Clusters



GPU Cluster Growing

1. Sorts cells by their $\left(\frac{\text{signal}}{\text{noise}}\right)^2$ ratio (only cells with S/N 4 are selected as seed cells);

GPU Cluster Growing

1. Sorts cells by their $\left(\frac{\text{signal}}{\text{noise}}\right)^2$ ratio (only cells with S/N 4 are selected as seed cells);
2. Assigns seed cells a tag based on their position in the ordering;

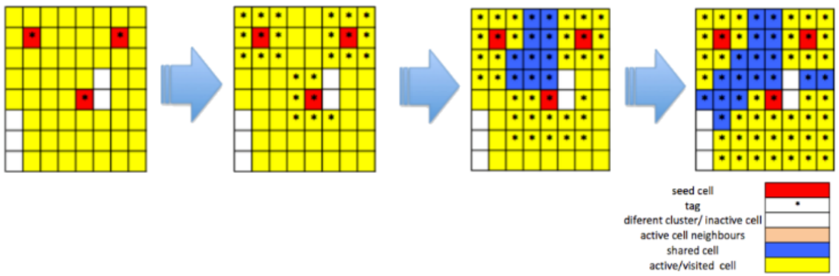
GPU Cluster Growing

1. Sorts cells by their $\left(\frac{\text{signal}}{\text{noise}}\right)^2$ ratio (only cells with S/N 4 are selected as seed cells);
2. Assigns seed cells a tag based on their position in the ordering;
3. Assigns a thread to a pair of neighbour cells;

GPU Cluster Growing

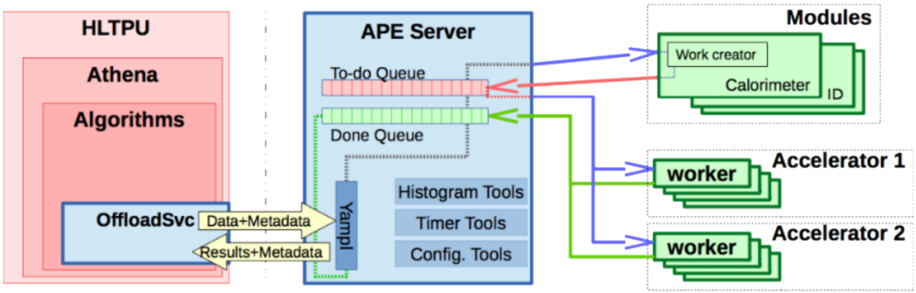
1. Sorts cells by their $\left(\frac{\text{signal}}{\text{noise}}\right)^2$ ratio (only cells with S/N 4 are selected as seed cells);
2. Assigns seed cells a tag based on their position in the ordering;
3. Assigns a thread to a pair of neighbour cells;
4. Each thread determines the pair's tag:
    - Higher Tags get propagated;
    - Cluster Growing stops when meets a cell with (S/N $\leq$ 1).

GPU Cluster Splitter



- The splitter takes as input the previously produced cluster and outputs new smaller clusters.

Architecture



- The Trigger will now use an client-server architecture:
  - Athena (ATLAS Trigger Software), running in a CPU, will interface with another machine, APE.
  - APE will receive Athena's requests and execute them using the several accelerator resources available.
  - APE will then return the processed data back to Athena
- This allows Athena to be independent of the specific accelerator details.

Methodology

- Our work consisted in running the TopoCluster Algorithm using a Standalone Client to simulate Athena's requests.

Methodology

- Our work consisted in running the TopoCluster Algorithm using a Standalone Client to simulate Athena's requests.
- Tests were run for the Growing Portion and the Splitter Portion separately.
- Finally the two were combined.

Methodology

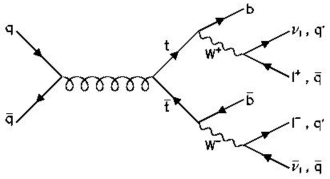
- Our work consisted in running the TopoCluster Algorithm using a Standalone Client to simulate Athena's requests.
- Tests were run for the Growing Portion and the Splitter Portion separately.
- Finally the two were combined.
- Input data was a collection of sample pre processed Athena jets and $t\bar{t}$ events.

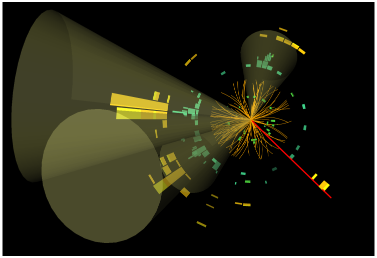
Methodology

- Our work consisted in running the TopoCluster Algorithm using a Standalone Client to simulate Athena's requests.
- Tests were run for the Growing Portion and the Splitter Portion separately.
- Finally the two were combined.
- Input data was a collection of sample pre processed Athena jets and $t\bar{t}$ events.
- Test Bed:
  - CPU: AMD FX-8320 8-core @ 3.5GHz
  - GPU: Nvidia GeForce GTX 650 (2048MB of VRAM)
  - RAM: 8GB

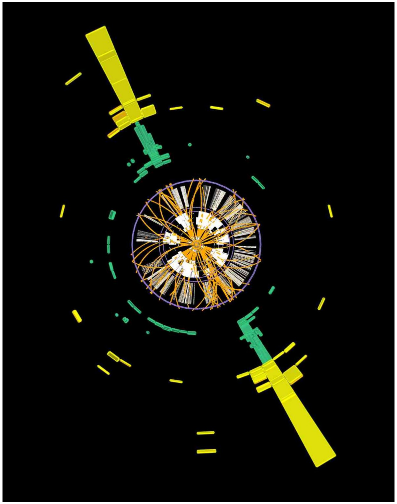
$t\bar{t}$ vs jets

$t\bar{t}$

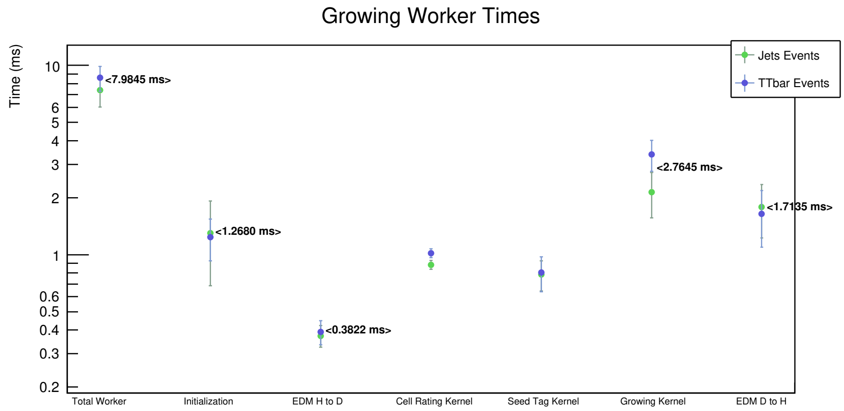




Jets

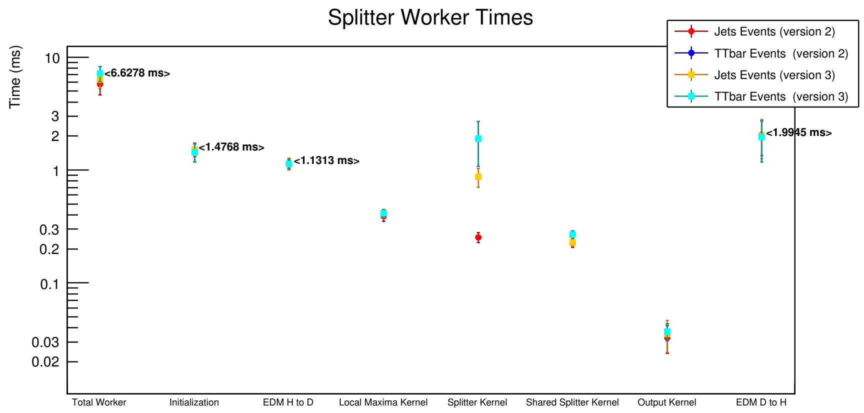


Growing Condensed Results



- Initialization and copying are significant
- Input: Cell Energy - Output: Clusters containing all cell information
- $t\bar{t}$ is more demanding than jets.

Splitter Condensed Results



- Initialization and copying very significant
- Most kernels take less than 1ms.
- Two different versions

Why two versions?

- Properties of GPGPU's: tasks are no longer run sequentially. A large number of threads run the same instructions concurrently.

Why two versions?

- Properties of GPGPU's: tasks are no longer run sequentially. A large number of threads run the same instructions concurrently.
- Double edged sword: Threads write and read to the same memory addresses at the same time, in no particular order.

Why two versions?

- Properties of GPGPU's: tasks are no longer run sequentially. A large number of threads run the same instructions concurrently.
- Double edged sword: Threads write and read to the same memory addresses at the same time, in no particular order.
- This can create race conditions and synchronization problems.

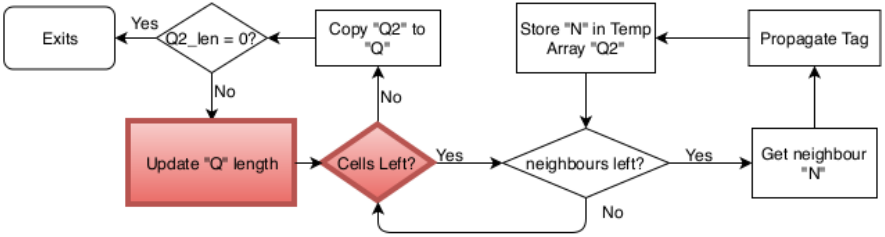
Why two versions?

- Properties of GPGPU's: tasks are no longer run sequentially. A large number of threads run the same instructions concurrently.
- Double edged sword: Threads write and read to the same memory addresses at the same time, in no particular order.
- This can create race conditions and synchronization problems.
- Global variables need special care: we must ensure writes and reads are in the order we desire.

Why two versions?

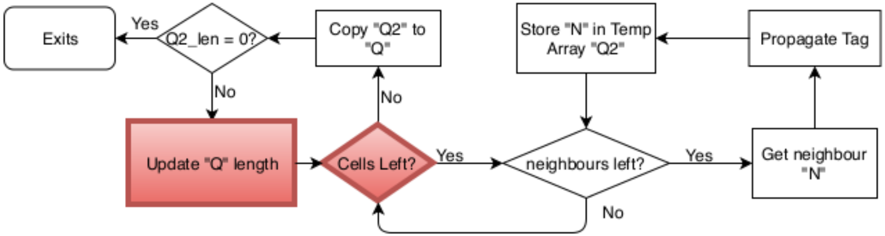
- Properties of GPGPU's: tasks are no longer run sequentially. A large number of threads run the same instructions concurrently.
- Double edged sword: Threads write and read to the same memory addresses at the same time, in no particular order.
- This can create race conditions and synchronization problems.
- Global variables need special care: we must ensure writes and reads are in the order we desire.
- CUDA defines a special function to do this: `__syncthreads();`

Splitter Kernel Version 3 - Main Loop



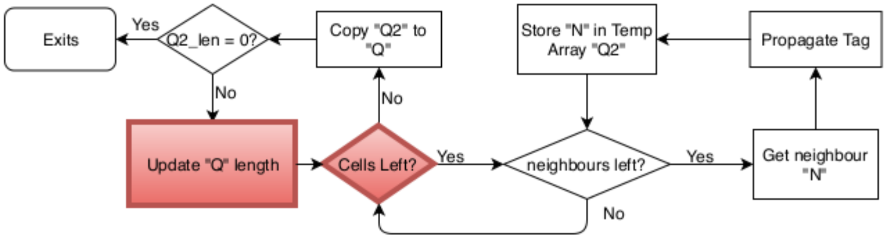
- In the first iteration `Q` is the list of local maxima

Splitter Kernel Version 3 - Main Loop



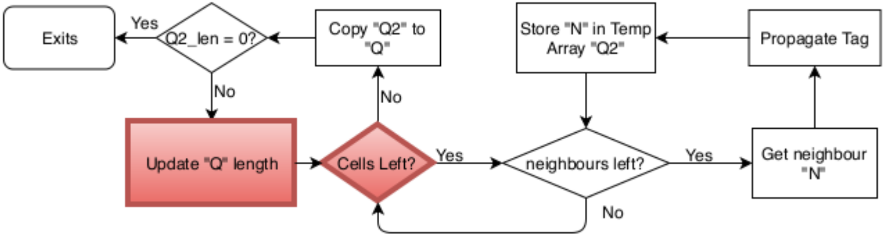
- In the first iteration `Q` is the list of local maxima
- Sometimes threads execute the loop over Q, with the wrong length causing the whole process to hang.

Splitter Kernel Version 3 - Main Loop



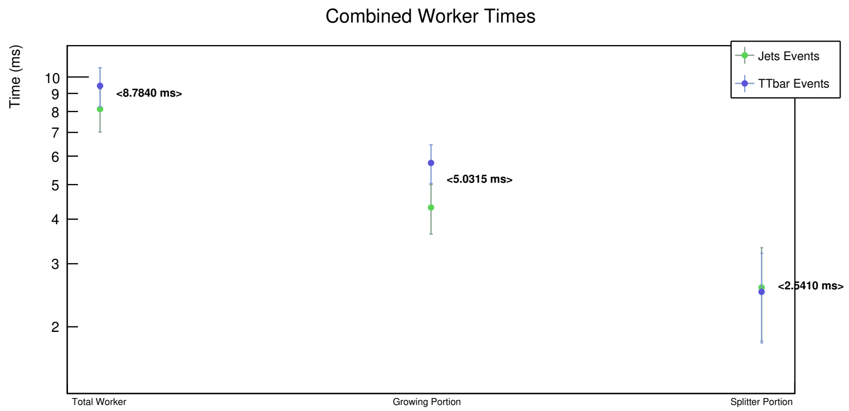
- In the first iteration `Q` is the list of local maxima
- Sometimes threads execute the loop over Q, with the wrong length causing the whole process to hang.
- Solution: `__syncthreads()` before the loop

Splitter Kernel Version 3 - Main Loop



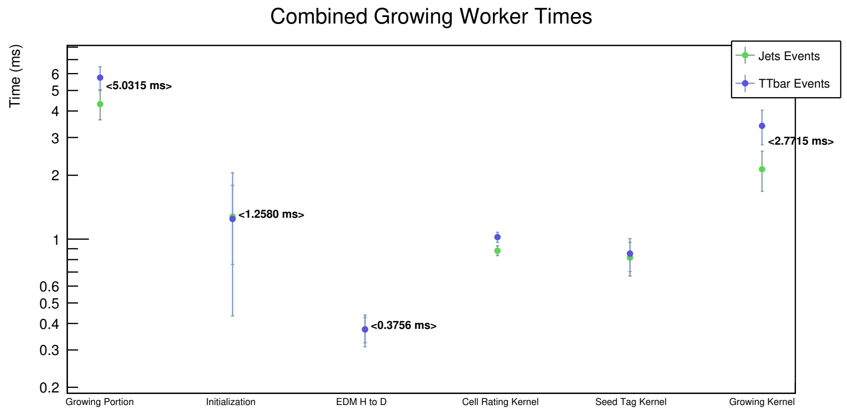
- In the first iteration `Q` is the list of local maxima
- Sometimes threads execute the loop over Q, with the wrong length causing the whole process to hang.
- Solution: `__syncthreads()` before the loop
- This made code slower as we now have to wait for all threads.

Combined Condensed Results



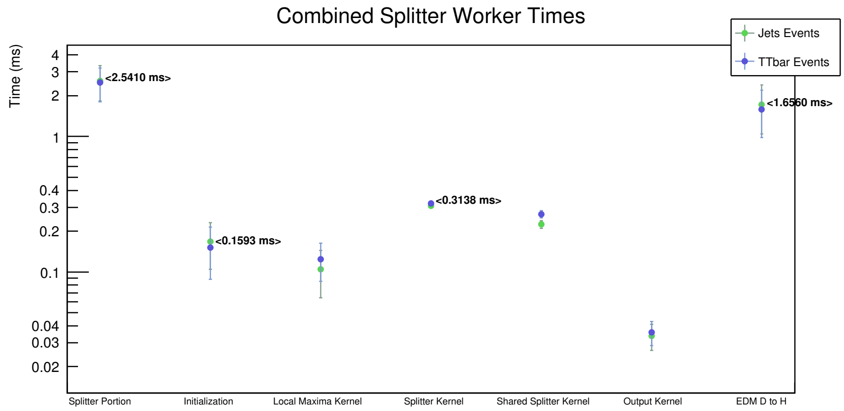
- Splitter V2 used.
- Total time is much lower than the sum of the separate growing and splitting stage

Combined Growing Condensed Results



- Total time decreased
- Note the absence of data copies from the device

Combined Splitter Condensed Results



- Total time decreased about 50%
- Again, no copies to the device, data is already there.
- Most of the time is spent copying data rather than processing

Conclusions

- Growing algorithm achieves an average of $\approx$ 7-8ms per event.

Conclusions

- Growing algorithm achieves an average of $\approx$ 7-8ms per event.
- Splitter algorithm achieves an average of $\approx$ 6-7ms per event.

Conclusions

- Growing algorithm achieves an average of $\approx$ 7-8ms per event.
- Splitter algorithm achieves an average of $\approx$ 6-7ms per event.
- As seen, combining the two steps results in a combined average of $\approx$ 8-9ms per event.

Conclusions

- Growing algorithm achieves an average of $\approx$ 7-8ms per event.
- Splitter algorithm achieves an average of $\approx$ 6-7ms per event.
- As seen, combining the two steps results in a combined average of $\approx$ 8-9ms per event.
- Comparing times for CPU only execution, around 10 fold improvement is achieved.

Conclusions

- Growing algorithm achieves an average of $\approx$ 7-8ms per event.
- Splitter algorithm achieves an average of $\approx$ 6-7ms per event.
- As seen, combining the two steps results in a combined average of $\approx$ 8-9ms per event.
- Comparing times for CPU only execution, around 10 fold improvement is achieved.
- There's a problem: data conversion to and from the GPU as well as data transfer accounts for a significant portion of the time spent.

Conclusions

- Growing algorithm achieves an average of $\approx$ 7-8ms per event.
- Splitter algorithm achieves an average of $\approx$ 6-7ms per event.
- As seen, combining the two steps results in a combined average of $\approx$ 8-9ms per event.
- Comparing times for CPU only execution, around 10 fold improvement is achieved.
- There's a problem: data conversion to and from the GPU as well as data transfer accounts for a significant portion of the time spent.
- GPGPU's show a promising improve in terms of accelerating processing tasks, while providing less power consumption and physical footprint.