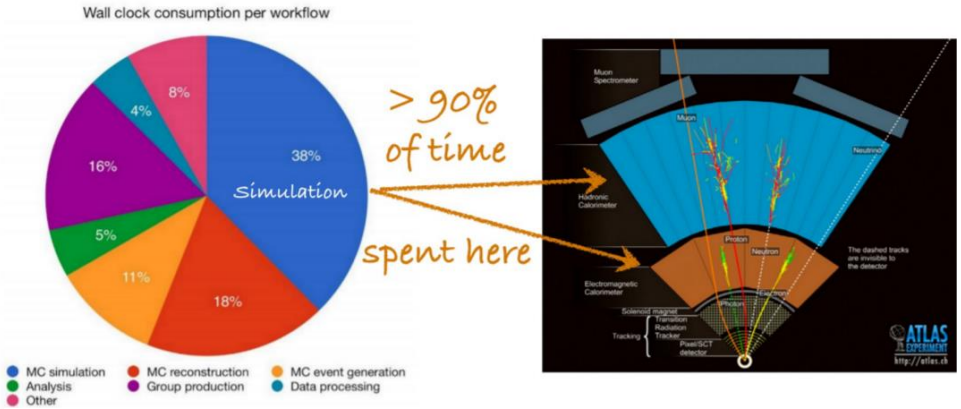
Fast Calorimeter Simulation on GPU

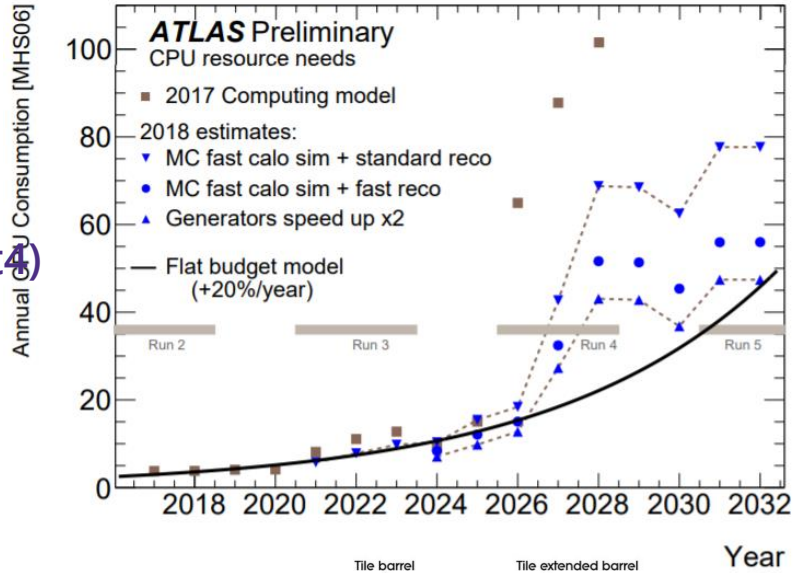
Ke Li , supported by US-ATLAS
University of Washington
03/24/2021
HL-LHC R&D topics

Motivation

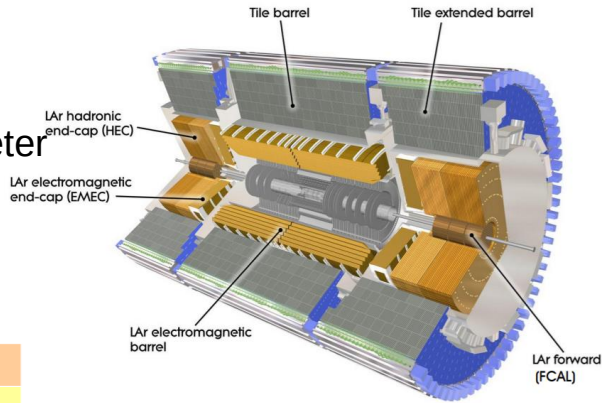
- Challenges from computing resources
- CPU usages is dominant by simulation (Geant4)
- ATLAS needs lots of simulation
- 90% time is spent in calorimeter

Calorimeter-dominated





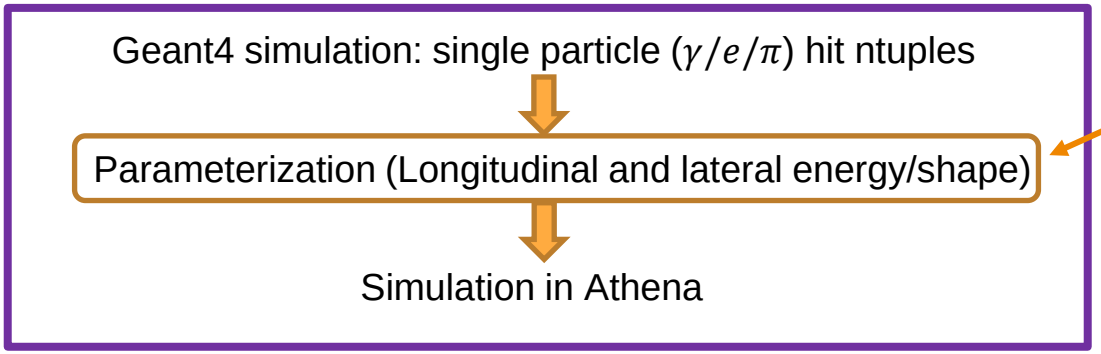
ATLAS Calorimeter



System	EM Barrel	EM EC	Hadronic EC	FCAL	Tile
#Channels	110k	64k	5.6k	3.5k	9.8k

FastCalorimeterSimulation (FCS)

- **A very large fraction of the simulation's computational budget is spent by the LAr Calorimeter**
  - **Parametrized simulation can speed things up enormously: FastCaloSim**
- **Based on the Geant4 simulation of single particles derived in a fine grid of particle energies and directions, simulated on the calorimeter surface.**



FCSParameterization
- self contained
- few dependencies

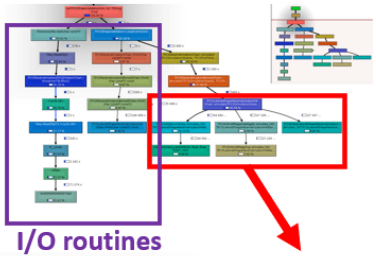
More details in cds

~10 times faster than full simulation

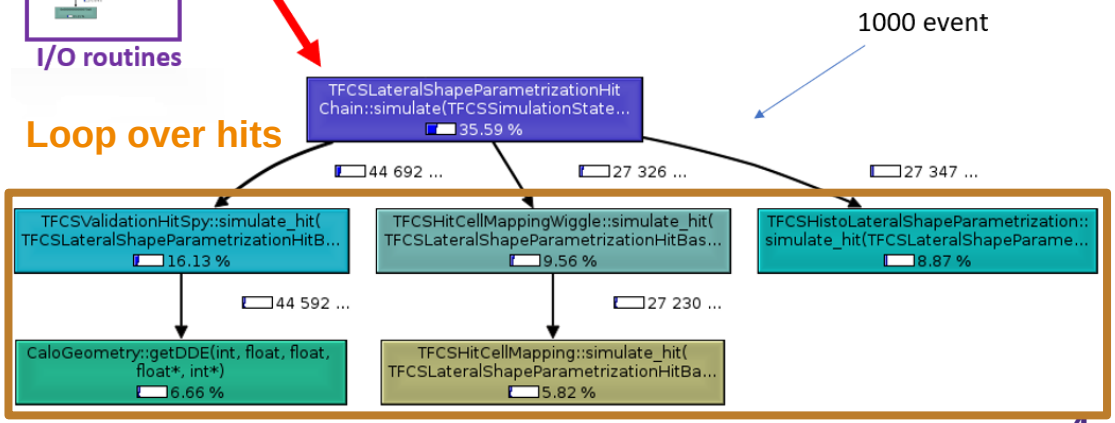
Goal: further acceleration with GPU

- LAr Calorimeter has massive inherent parallelism
  - lots of independent cells and associated tasks.
- Profiling studies identified likely hotspots that are paralellizable
- Cuda kernels created to run thes parts on the GPU
  - modified data structures
  - reimplement Geometry and parametrization tables for GPU – no STL allowed
  - 3 kernels:
    - reinit memory
    - main simulation
    - reduction

Performance Profile



- TFCSLateralShapeParametrizationHitChain::simulate() is the most significant routine except I/O parts.
- TFCSLateralShapeParametrizationHitChain::simulate() The running time scale with the number of events .
- TFCSLateralShapeParametrizationHitChain::simulate() is our target to parallelize.



In FCS-GPU (cuda)

- Based on a standalone version of FCS parameterization (git) loop on Event
  - Loop on Particle
    - Loop on Hits

parallelize / port to GPU

```
TFCSLateralShapeParametrizationHitChain::simulate(
                                TFCSSimulationState& simulstate
                                const TFCSTruthState* truth,
                                const TFCSExtrapolationState* extrapol ) {
    int nhit = get_number_of_hits( simulstate, truth, extrapol );
    float Ehit = simulstate.E( calosample() ) / nhit;

    for ( int i = 0; i < nhit; ++i ) {  //  hit simulation loop
      TFCSLateralShapeParametrizationHitBase::Hit hit;
      hit.E() = Ehit;
      for ( TFCSLateralShapeParametrizationHitBase* hitsim : m_chain ) {
        hitsim->simulate_hit( hit, simulstate, truth, extrapol );
      }
    }
}
```

In GPU

FCS GPU Acceleration

Loop on Nhits → ~50000/event

TFCSCalculateCenterPosition:: simulate_hit	Save Extraop center info
TFCSValidationHitSpy:: simulate_hit	Identify hit Cell Fill Various Histograms from Cell geo and Hit coordinates. Save Hit (cell)
TFCSHistoLateralShapeParametrization:: simulate_hit	Setup Hit (phi, eta, Z, E)
TFCSHitCellMappingWiggle:: simulate_hit	Wiggle hit phi Identify new cell Add cell to a map Accumulate cell's Hit count (or Eerngy) in "Simlustate"
TFCSValidationHitSpy:: simulate_hit (AGAIN)	Fill Various Histograms from Cell geo and Hit coordinates. If new cell match previous cell Fill few more Histograms

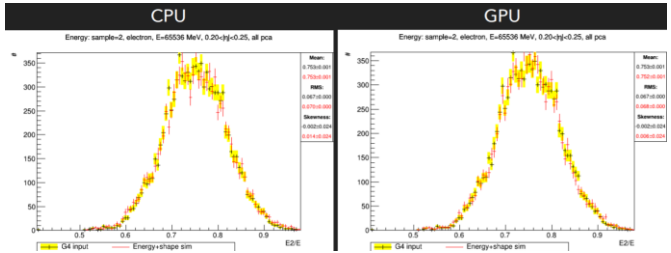
End Loop

Geometry Construction → CUDA

> **Random number:**

- FCS needs lots of random numbers
- 3 per hit x ~5k hits per event
- Cuda has a very good random generator (cuRAND), much faster than CPU

statistically equivalent



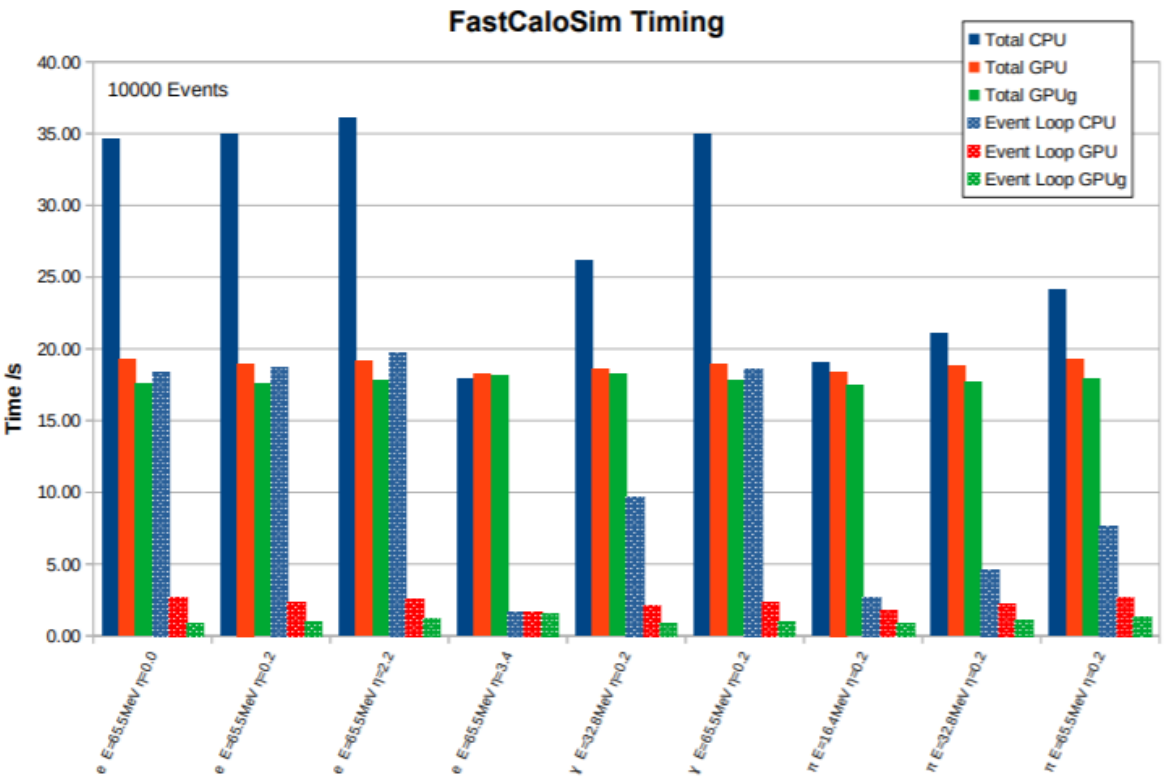
Performance

- Significant improved with GPU
- GPU is less efficient for small numbers of hits (lower energy, higher η)
- I/O to read/unpack parametrization files is expensive: ~15s
- GPU kernels very short - launch latency limited
- Better performance if group work (GPUg) between multiple events to give more work to GPU

			Total Time /s			Event Loop Time /s		
particle	E (MeV)	η_{min}	CPU	GPU	GPU_g	CPU	GPU	GPU_g
Electron	65536	0.00	34.62	19.26	17.54	18.30	2.71	0.90
Electron	65536	0.20	34.97	18.98	17.61	18.67	2.37	0.97
Electron	65536	2.20	36.09	19.13	17.83	19.69	2.56	1.19
Electron	65536	3.40	17.95	18.22	18.14	1.60	1.64	1.54
Photon	32768	0.20	26.15	18.60	18.29	9.71	2.06	0.80
Photon	65536	0.20	34.94	18.91	17.83	18.60	2.31	0.97
Pion	16384	0.20	19.02	18.41	17.46	2.67	1.76	0.89
Pion	32768	0.20	21.06	18.83	17.67	4.57	2.17	1.08
Pion	65536	0.20	24.13	19.28	17.90	7.68	2.70	1.30

Performance

- Sigr
- GPU
- I/O
- GPU
- Bett
  GPU



give more work to

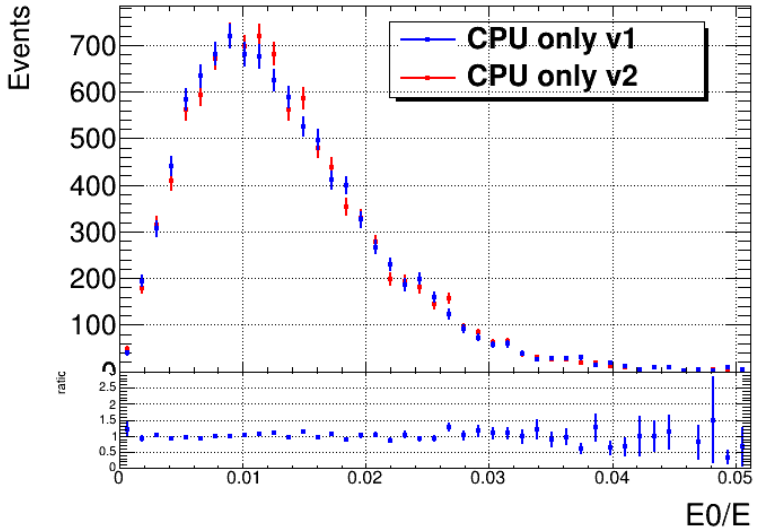
parti	e /s GPU$_g$
Elect	0.90
Elect	0.97
Elect	1.19
Elect	1.54
Pho	0.80
Pho	0.97
F	0.89
F	1.08
F	1.30

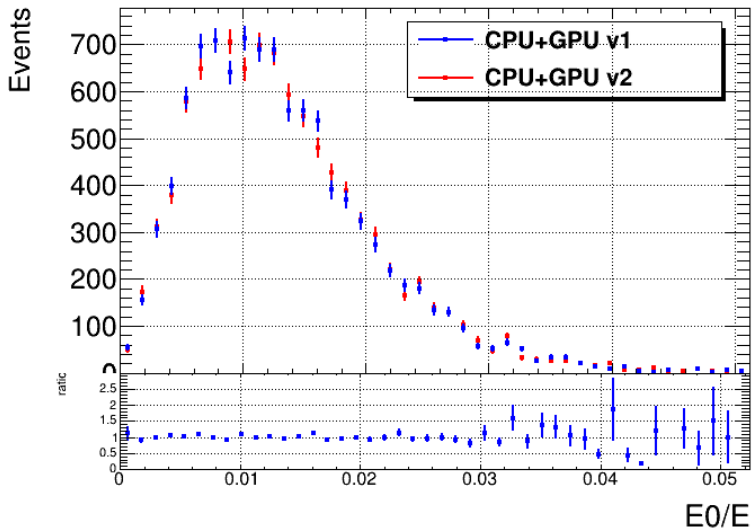
Recent activities

- Update to FCS v2
- Compiler: C++14 -> C++17
- ROOT: 6.14.08 – 6.22.00
- Updated the port to cuda
- Integration to Athena (ATLAS offline software framework)

Validation

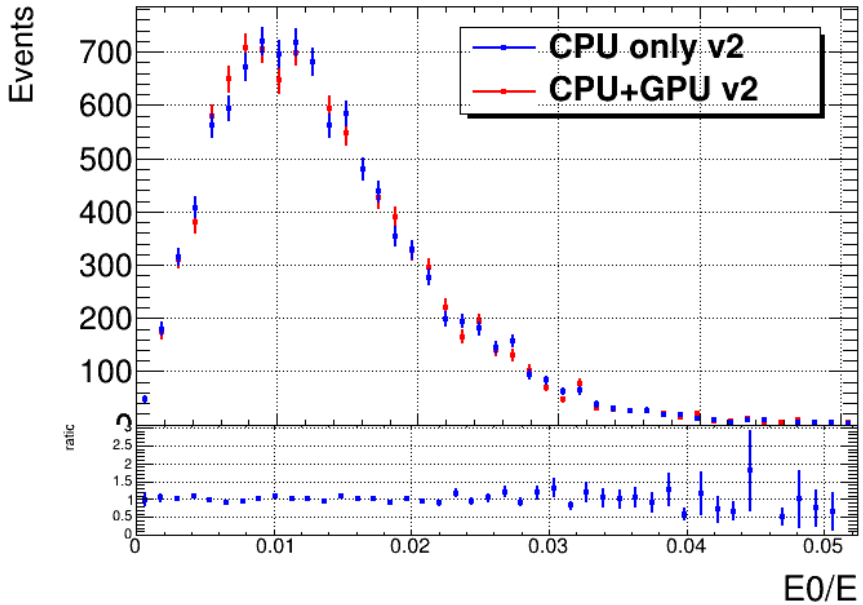
> Consist deposit energy fraction distributions for FCS-cuda v1 and v2





Validation

- Deposit energy fraction distributions are consist with CPU only and CPU+GPU with FCS-cuda v2



Updated performance

- Tested in Cori
- One CPU core + one GPU core (NVIDIA V100)

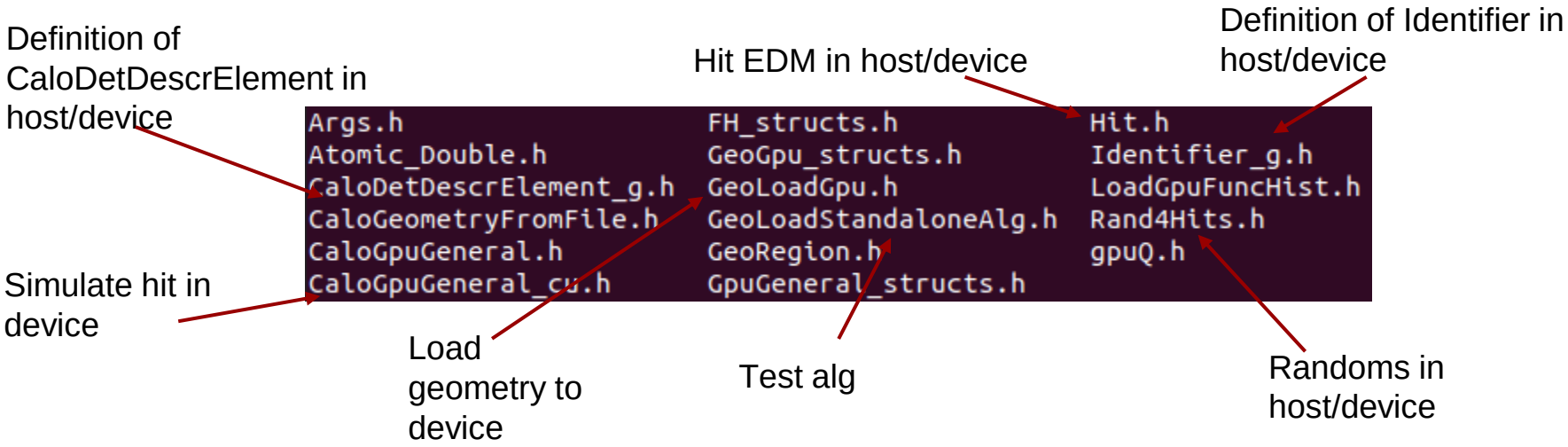
	CPU only (s)	CPU+GPU (s)
v1	18.30	2.71
New test in Cori (v1)	19.34	3.97
v2	-	2.63

Integration to Athena

- Currently working on a standalone package
- The cuda implementation looks good and we should move on
  - Cuda is supported in Athena, thanks many people
- Two main parts for integration
  - Geometry construction (in GPU)
    - Partly completed
  - Clients update (LateralShapeParametrization ...)
    - Will start later

Package layout

- Build a new package AthFastCaloGpu (following the implementation in the standalone FCS-cuda v2)



Build geometry in device

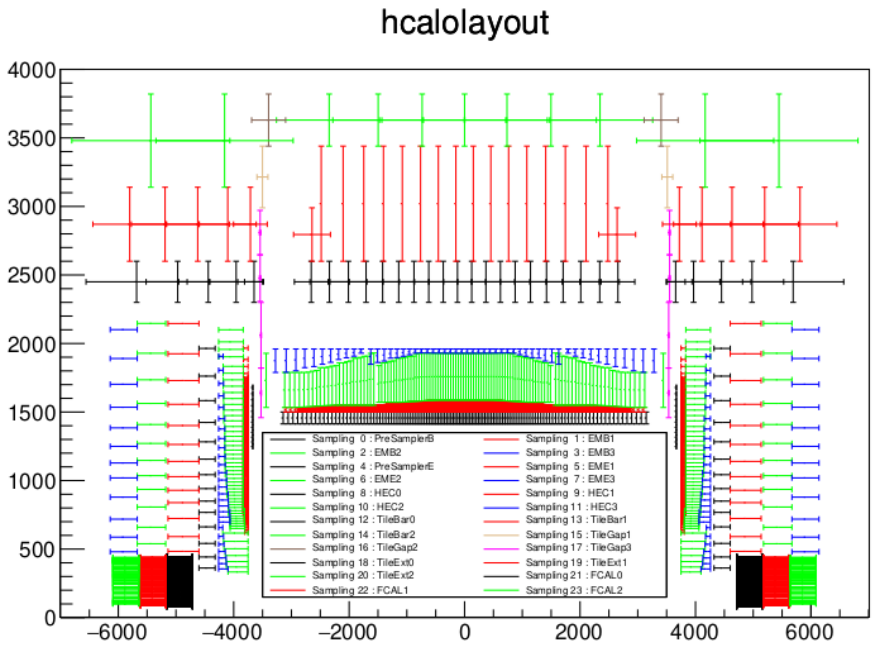
- Following the example in AthExCUDA/LinearTransformStandaloneExampleAlg
- Read geometry from a file
- Copy the geometry to device
- Build a kernel to get the geometry in device and compare with that from host
- WIP:
  - Read geometry from detector manager
  - Use Athena tools/wrappers to manage the tasks

Calorimeter layout

A simple test

- Read geometry from a file
- Build geometry in device
- Compare the geometries
- Plot the layout from host

Load successfully in GPU



Summary

- Significant improvement (reduce total time by half) is obtained with using GPU (cuda) for fast calorimeter simulation
  - GPU resources are not well used
    - kernels are very short, work size is small, need further optimization
- FCS is updated to what Athena (master branch) is currently using
- Integration to Athena is started
  - Many updates/developments are needed

Acknowledgements:
Thanks all the people for the help and the contribution to this project:
Doug Benjamin, John Chapman, Zhihua Dong, Ahmed Hasib, Charles Leggett, Meifeng Lin, Tadej Novak, Vincent Pascuzzi, Douglas Schaefer, Kwangmin Yu
…