FastCaloSim on GPUs

BNL CSI: Zhihua Dong, Kwangmin Yu, Meifeng Lin

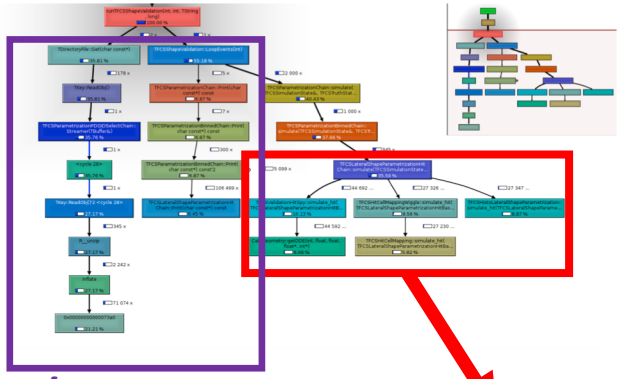
ATLAS: Tadej Novak, Ahmed Hasib, Heather Gray

+ Others

FastCaloSim

- Fast simulation of ATLAS calorimeter system
- Relatively self-contained => initial target for performance analysis and GPU porting exploration
- Original standalone version runs under the ROOT interpreter
  - Difficult to use standard profiling tools
  - Parallelization/GPU porting would also be difficult
  - Developed a compiled version of standalone FCS
  - program → "**runTFCSShapeValidation**"
- Project funded by HEP-CCE
- Collaboration between ATLAS and BNL Computational Science Initiative

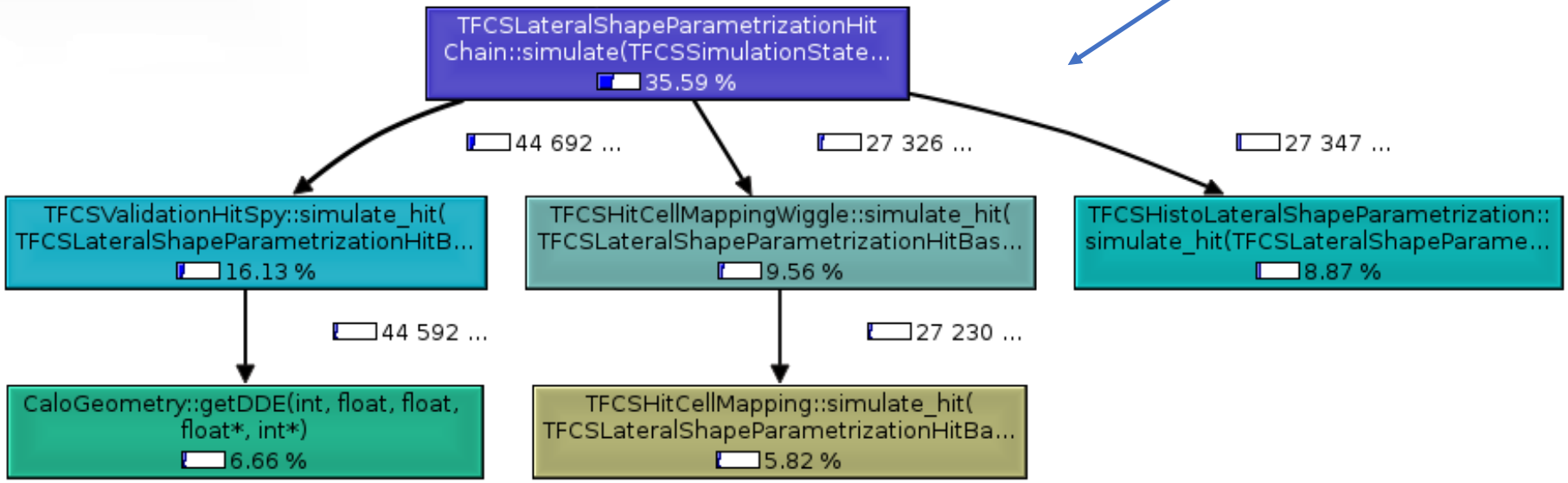
Performance Profile



I/O routines

- **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.

1000 event



Analysis

TFCSLateralShapeParametrizationHitChain::simulate() Structure

TFCSLateralShapeParametrizationHitChain::simulate() {

…

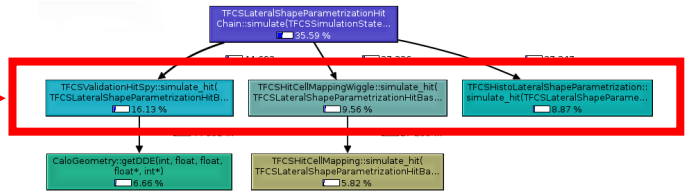
…

Loop on **nhit**

…

Loop on hit_simulation_chain

Call



End Loop

…

End Loop

…

}

Possible Parallelization

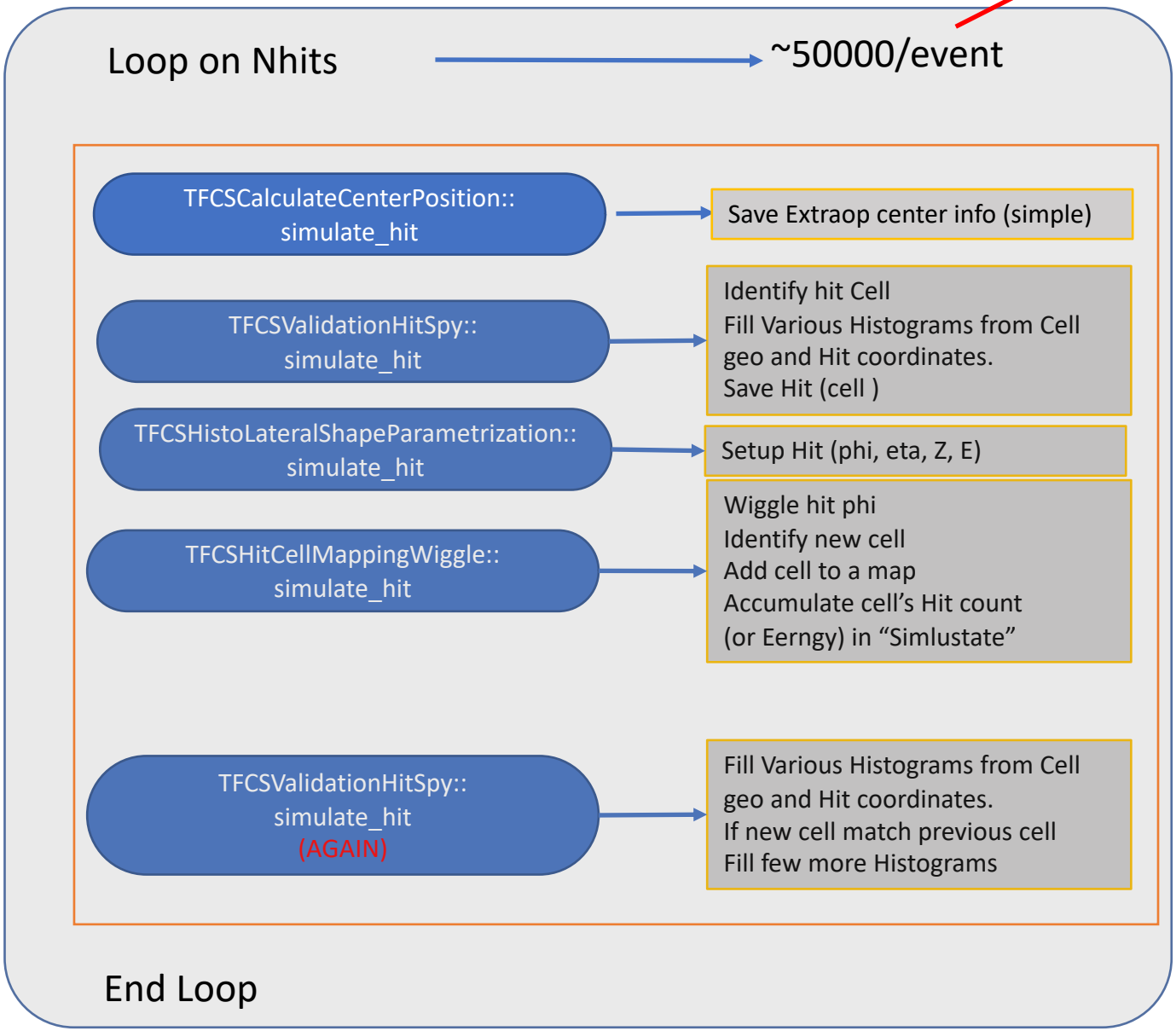
Impossible Parallelization because of data dependency

GPU Porting

- ✓ Dependence on ROOT and C++ nature of FCS would make it difficult to use OpenACC/OpenMP
- ✓ **Initial path: using CUDA**
- ✓ Port the parallelization possible part in TFCSLateral~~~HitChain::simulate()
  - ➢ Implemented CUDA kernel & device functions
- ✓ Data structure relocation from CPU to GPU
  - ➢ Geometry data can be loaded once and be reused

FCS GPU acceleration

For validation against GEANT4



CUDA

GPU version of the 4 functions

1st stage
Nhits Threads

Tasks for Porting to GPU

- **GPU Function to identify cell.** **getDDE(sample, eta, phi)**

  Load Geometry Info to GPU. (Deep Copy)

  - ~200,000 Calo Cells in 24 Layers (samples). → 20+MB
    (code setup run on Layer 2 has around 20,000 cells)
  - Various regions' Geo info and cell pointers

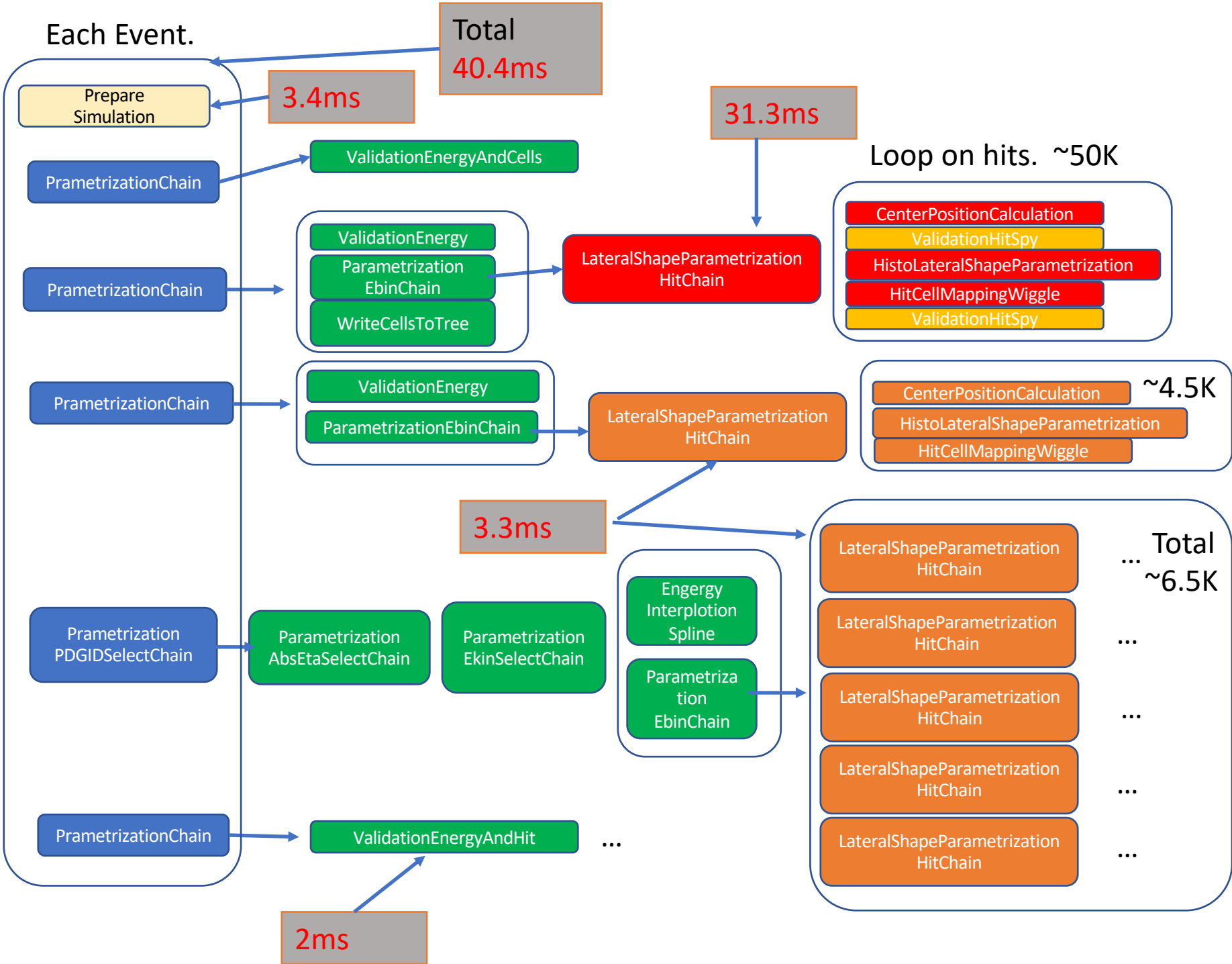
  - Re-implement GPU CaloGeomory structure and supporting Classes
  - Simpler, no ROOT Dependence, only needed methods

- **GPU Histograms**

  Multi-Stage CUDA kernels
  - Block-wise atomic update with shared memory
  - Reduction of results from all blocks

- **GPU Hit Cell Counting**

  ~50000 hits end up in < ~200 cells (out of ~20,000)
  Multi Stage CUDA kernels
  Extra step to narrow down hit cells
  before standard GPU Histogram(count).


Each Event.
Total
40.4ms
Prepare Simulation
3.4ms
31.3ms
Loop on hits. ~50K
PrametrizationChain
ValidationEnergyAndCells
PrametrizationChain
ValidationEnergy
Parametrization EbinChain
WriteCellsToTree
LateralShapeParametrization HitChain
CenterPositionCalculation
ValidationHitSpy
HistoLateralShapeParametrization
HitCellMappingWiggle
ValidationHitSpy
PrametrizationChain
ValidationEnergy
ParametrizationEbinChain
LateralShapeParametrization HitChain
CenterPositionCalculation
HistoLateralShapeParametrization
HitCellMappingWiggle
~4.5K
3.3ms
Prametrization PDGIDSelectChain
Parametrization AbsEtaSelectChain
Parametrization EkinSelectChain
Engergy Interplotion Spline
Parametriza tion EbinChain
LateralShapeParametrization HitChain
LateralShapeParametrization HitChain
LateralShapeParametrization HitChain
LateralShapeParametrization HitChain
LateralShapeParametrization HitChain
...
Total
~6.5K
PrametrizationChain
ValidationEnergyAndHit
...
2ms

Code Strucuture

G++

```
Load_geometry() ;
…
for (ievent=0; ievent<nevent  ievent++) {
     rand_gen_init();
     Init_gpu () ;
     Prepare_simulation();

    for( auto ichain  : m_chains) {
        ichain-> simulate()
    }


     finish_gpu()
    rand_gen_finish()
}
```

```
TFCSLateralParametrizationChain::Simulate(){

…
If(nhits>2000 && my_chain_type ) {
   Load_2Dfuction() ;
   Load_wiggle1Dfunction();
  args=prepare();
  Simulate_Hit_gpu(args)
} else {
  for(ihit=0;ihit<nhit; nhit++)
    for(auto ichain : m_chain)
          ichain->SimulateHit_cpu();
}
….
}
```

nvcc (host code)

```
Simulate_Hit_gpu(args){
…
cuMalloc(…);
malloc(…) ;

rand_gen();

blocksize=512;
nblock=args.nhits/blocksize
Kernel_A<<<nblock, blocksize>>>(args);

blocksize=64 ;
nblock=ncells/blocksize ;
kernel_B<<<nblock, blocksize>>>(args) ;

cudaMemcpy( hitcells,…) ;
cudaMemcpy(num_hitcells…) ;
…
Kernel_C<<<….>>>(args);
…
Kernel_D<<<….>>>(args)
…
cudaMemcpy( hitcells_counts,…) ;
HitSpy_stage2<<<…>>>(args)
HitSpy_stage3<<<…>>>(args)

cuFree(…)
Free(…)

}
```

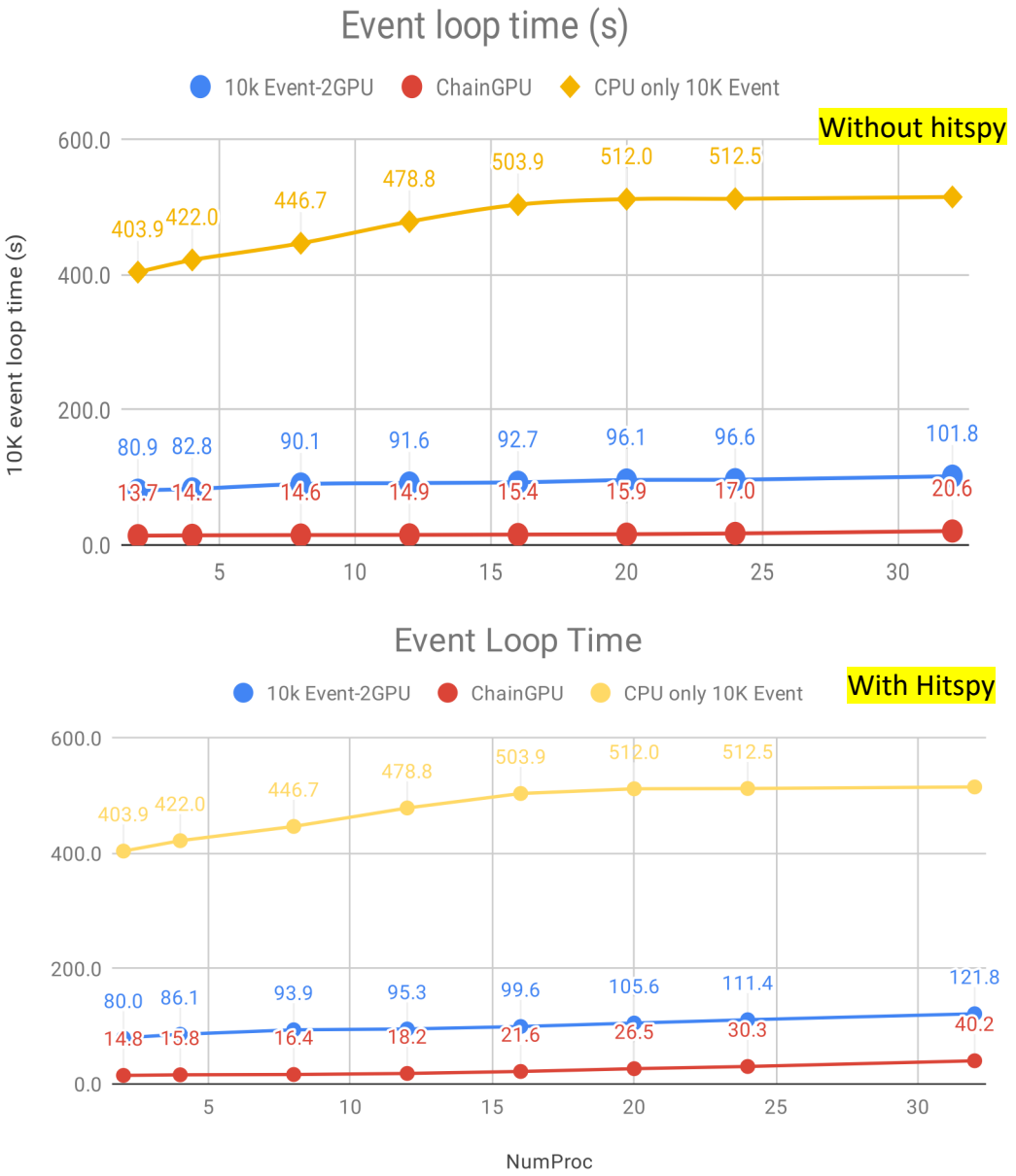
Device Code

```
__global__
kernel_A(args){
Hit hit;
CenterPositionCal(hit&, args);
If(spy) HitSpy(args, hit&);
HistoLateralHit(args,hit&);
CellMapWiggle(args,hit&);
If(spy) HitSpy(args, hit& );
}
```

…..

Tests with multiple instances running

- Validation against GEANT4 most time consuming (~50K hits)
- Run sample program on the same node with up to 32 instances
- Use CUDA-MPS to share 2 P100 GPUs on BNL Institutional Cluster
- ~5X gain with 50K hits compared to CPU only runs (32 parallel processes).



Progress at Hackathon this week

- Trying to optimize the code for production-like environments
  - More particles and energies studied
  - Fewer hits needed than Validation against Geant4

Particle	Energy	Min Eta	CPU (s) / 10K event	GPU (s) / 10K event
Electron	65536	0	18.3	8.3
Electron	65536	0.2	18.8	8.0
Electron	65536	1	19.2	7.9
Electron	65536	2.2	19.8	8.5
Photon	65536	0.2	18.7	6.9
Pion	65536	0.2	7.7	5.7
Pion	32768	0.2	4.5	4.4

Max nhits ~4000-5000

Max nhits ~2000-2500

Only some events use GPU

TODO

- Further optimize the CUDA code
  - Use shared memory for the simulation
  - Multiple particles/energies per event => potentially increase GPU utilization
  - Parallelize hits at different layers (right now only one layer at a time)
  - Kernel fusion to reduce overhead
- Try out portable solutions: Kokkos, RAJA, OpenMP/OpenACC, SyCL