FastCaloSim on GPUs

BNL CSI: Zhihua Dong, Kwangmin Yu, Meifeng Lin

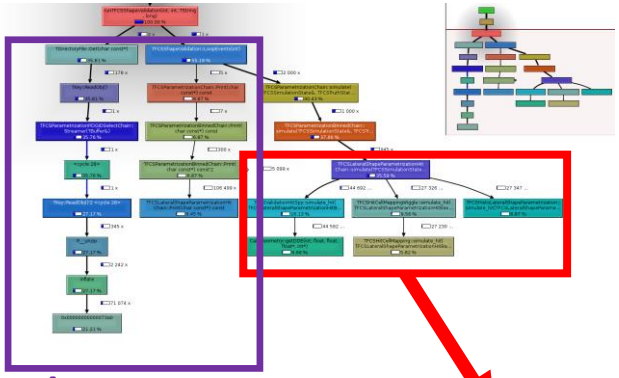
ATLAS: Tadej Novak, Ahmed Hasib, Heather Gray

+ 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
  - Now working on a compiled version of standalone FCS
  - program → "**runTFCSShapeValidation**"
- Project funded by HEP-CCE
- Collaboration between ATLAS and BNL Computational Science Initiative

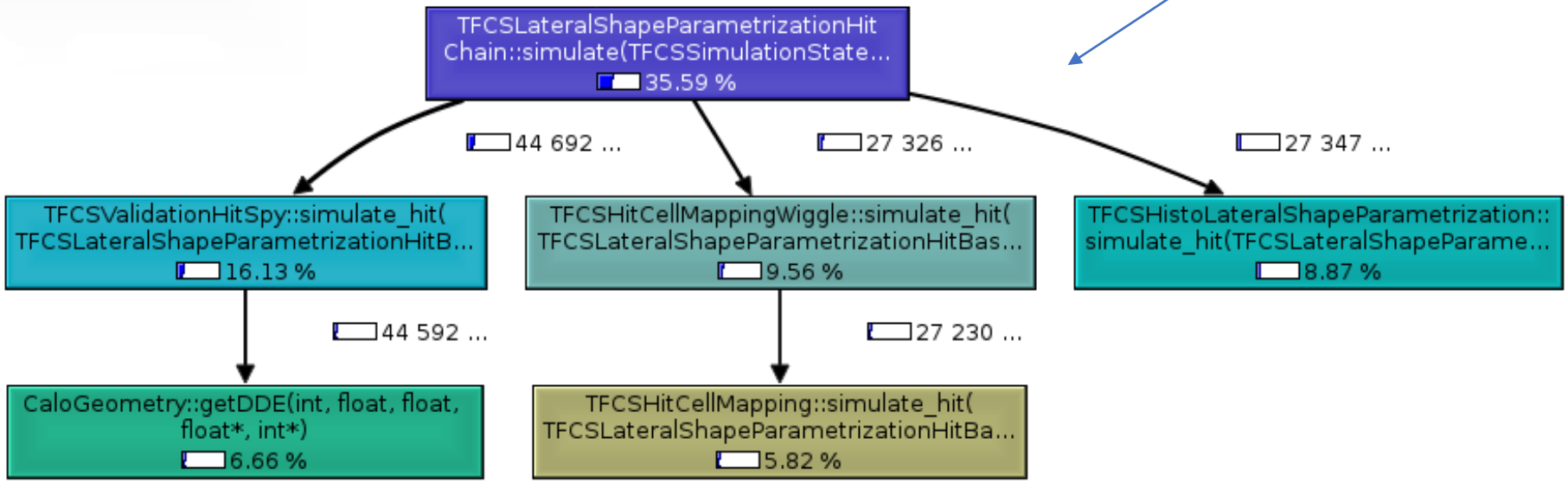
Performance Profile



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.

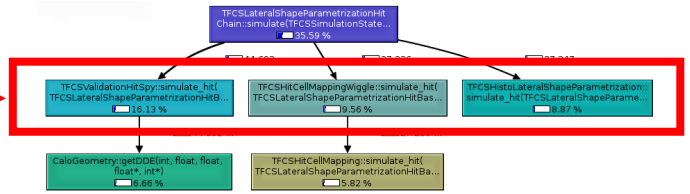
1000 event



Analysis

TFCSLateralShapeParametrizationHitChain::simulate() Structure

```
TFCSLateralShapeParametrizationHitChain::simulate() {
    …
    …
    Loop on nhit
        …
        Loop on hit_simulation_chain
            Call
        End Loop
        …
    End Loop
    …
}
```



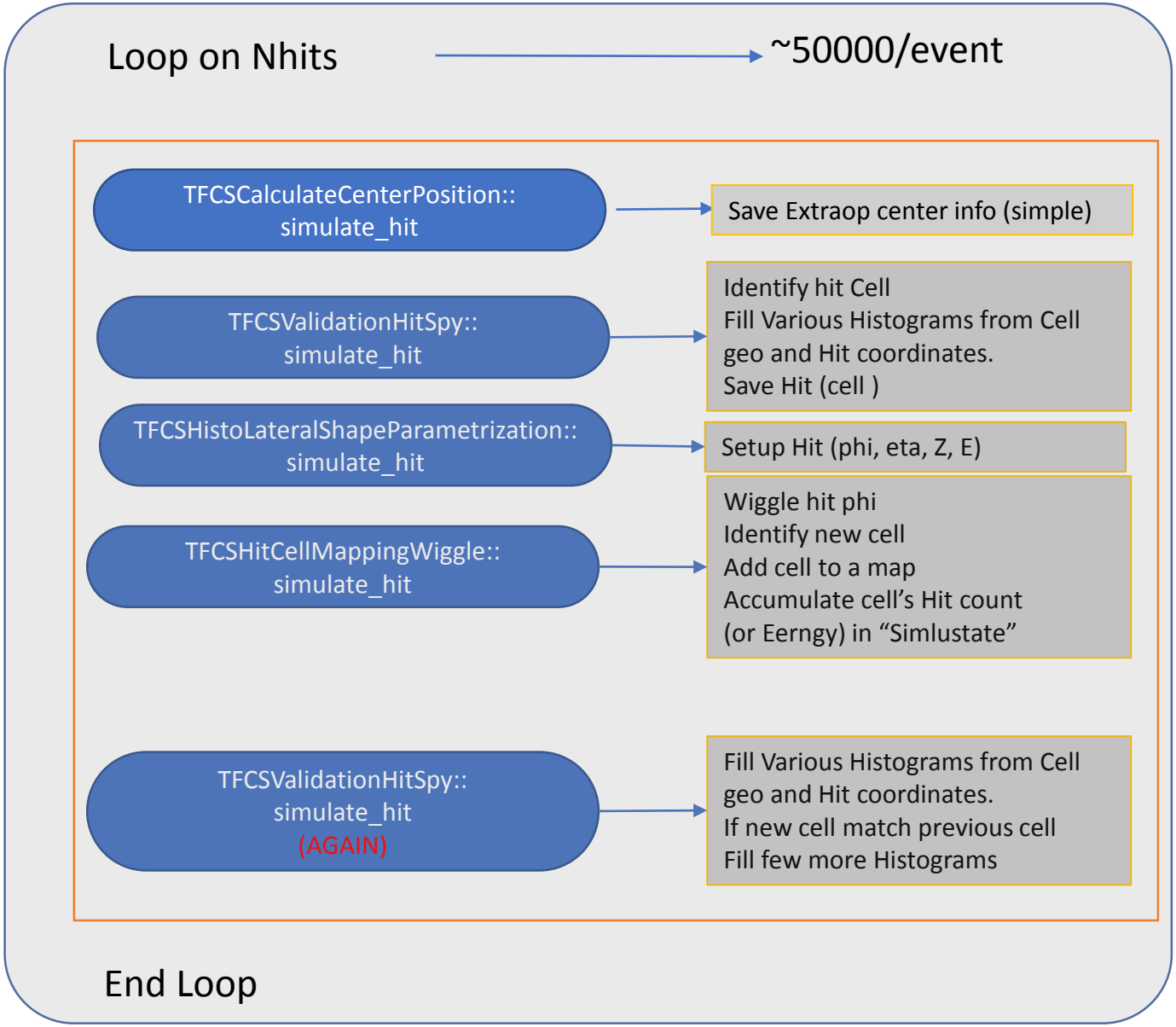
Possible Parallelization

Impossible Parallelization because of data dependency

GPU Porting

- ✓ Dependence on ROOT and C++ nature of FCS make it difficult to use OpenACC
- ✓ Initial path: using CUDA
- ✓ Port the parallelization possible part in TFCSLateral~~~HitChain::simulate()
  - ➢ Currently implementing CUDA kernel & device functions
- ✓ Data structure relocation from CPU to GPU
  - ➢ Will need to minimize data transfer between CPU and GPU during runtime

FCS GPU acceleration



CUDA

GPU version of the 4 functions

1st stage
Nhits Threads

Tasks for to Port 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

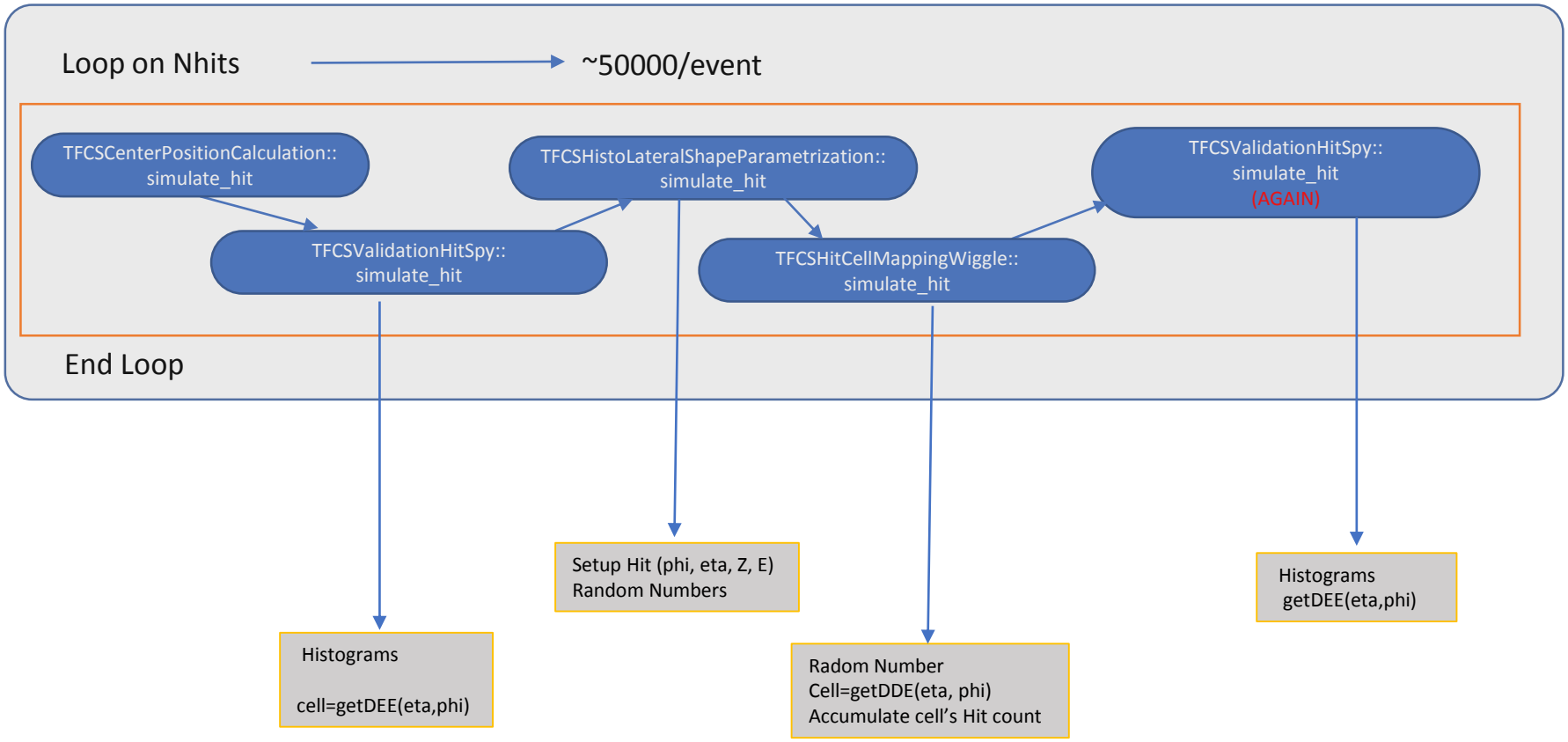
  - Re-implenment GPU CaloGeomory structure and supporting Classes
  - Simpler, no ROOT Dependence only needed methods

- **GPU Histograms**

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

- **GPU Hit Cell Counting**

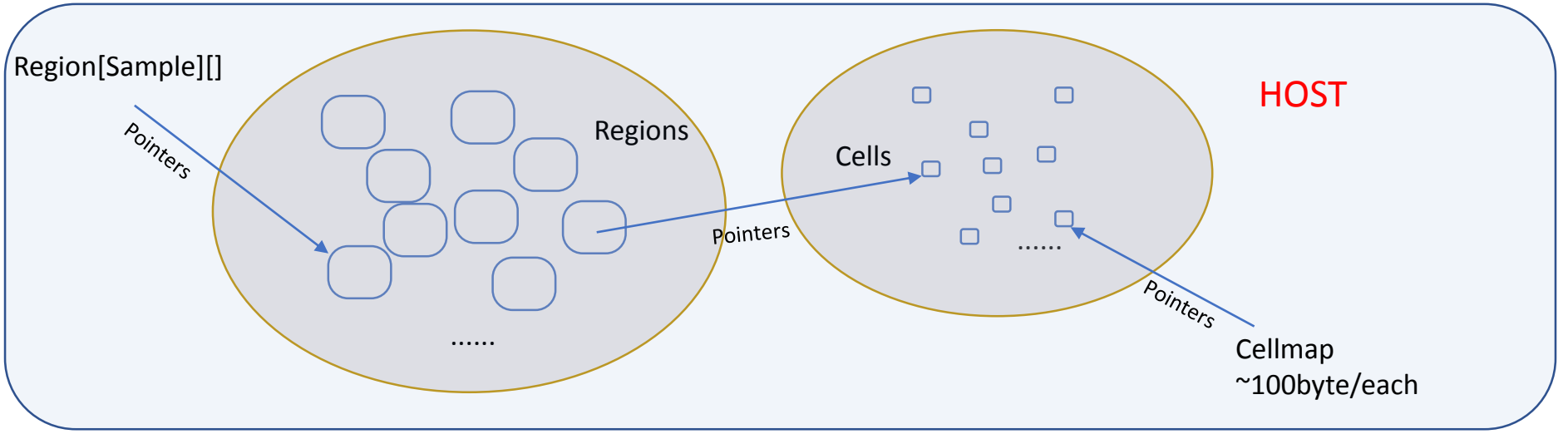
  ~50000 hits endup in <200 (?) cells (out of ~20,000)
  Multi Stage CUDA kernels
  Extra step to narrow down hit cells
  before standard GPU Histogram(count).

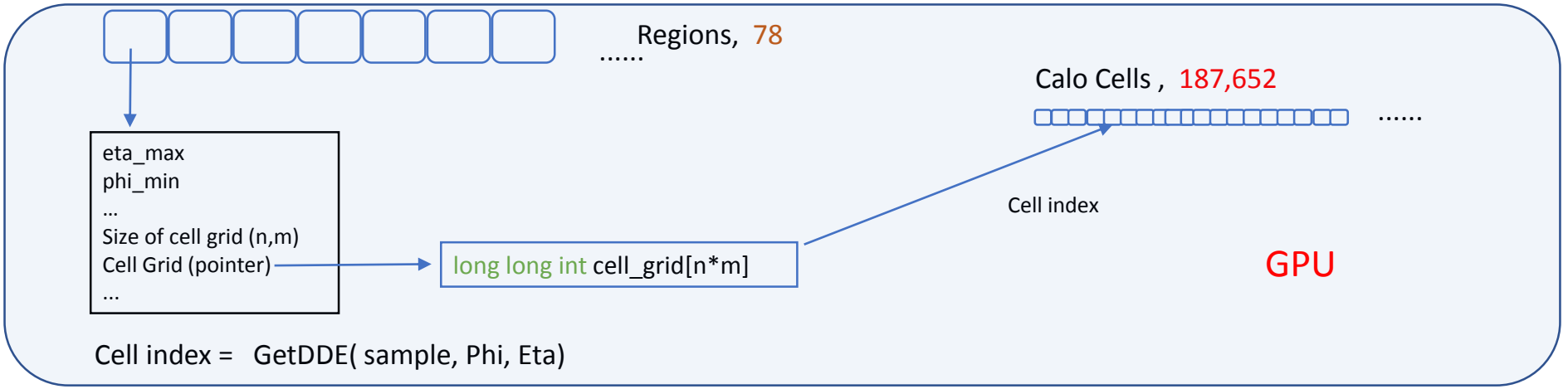


Tasks:

Device function: getDDE(sample, eta, phi)
Histogram
Hit count on large number of cells

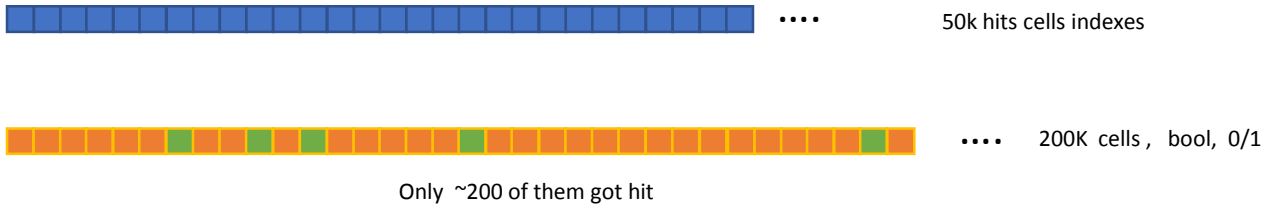
CaloGeometry



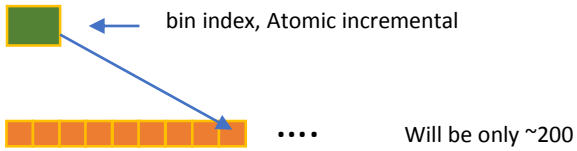


Hit Count on Large number of cells

Step 1: 50k Threads (less with loop)



Step 2: 200k Threads (less with loop)

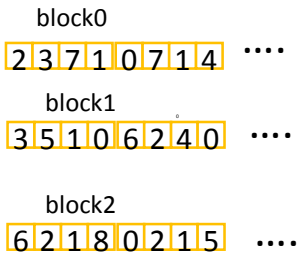


Step 3: 2 phase histogram

CUDA blocks

0	1	2	3

Local histogram(count)
using atomics
and shared memory



Merge(reduction)

Final counts

Further planning

- Multi GPU
- CUDA Stream for parallel on events

Summary