

GPU Optimizations for HEP Analysis in ROOT

Kevin Nobel

Supervisors: Monica Dessolet and Jolly Chen



Introduction

Batch Histogramming

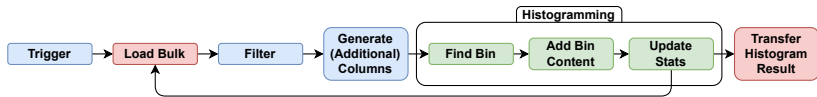
Offloading `.Define()` to GPUs

Future Work

Conclusion



Introduction



A typical HEP analysis:

- Load data
- Filter data
- Define additional columns
- Fill a histogram

Batch Histogramming

Goal and Motivation

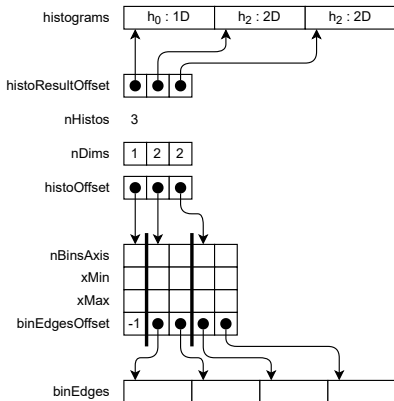
- Motivation: Future increase in data to be processed requires faster histogramming
- Goal: fill **multiple histograms** in parallel on GPUs
- Especially interesting if there is **overlap** in the input data
 - Better locality and a reduction in data transfers

Batch Histogramming Implementation

Features:

- Multiple histograms
- Mixed dimensions
- Different axis sizes
- Fixed and variable bins

GPU Datastructure →



Batch Histogramming Results

- CPU: Ryzen 7 5700g, GPU: RTX 3060
- 1D + 2D + 2D histogram, 100 Million rows
- Speedup: $5.9\times$ over single-threaded CPU impl.
- GPU Fill is much faster ($417\times$ speedup)
- Spend 98.6% of the (GPU) runtime on transferring data...

Offloading `.Define()` to GPUs

- Multiple histograms on the same columns are rare
- Generating new data based on the same columns is more common
- What if we compute new columns on the GPU?
 - Define may be computationally heavy
 - Potentially less data transferred

Poster Presentation

- Presented the idea at the Summer Student Poster Presentation^a

^aindico.cern.ch/event/1435014/

Eliminating Coffee Breaks at CERN: ROOT Histogram Performance Improvements using GPUs and Batching

Kevin Nobel (University of Amsterdam)
CERN Summer Student in EP-SFT

Supervisors: Monica Dessole (CERN) and Jolly Chen (University of Twente)



Figure 1: Typical HEP analysis workflow in ROOT.

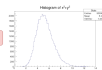


Figure 2: Example of a 1D ROOT histogram.

Introduction

ROOT [1] is an open source C++ data analysis framework widely used in High Energy Physics (HEP) to efficiently analyze petabytes of data. A typical HEP analysis (Figure 1) generally consists of filtering data, producing new columns from existing data and computing histograms (Figure 2).

Motivation

With the upcoming High Luminosity LHC upgrade, the intensity is expected to increase by a factor of 30 [2], leading to a similar increase in data to be analyzed. In the future, more computing power is needed to meet the demands of HEP experiments.

Accelerators like GPUs are increasingly common in modern computing infrastructures. Making use of the full capabilities of heterogeneous systems is an implicit requirement for the ROOT framework. Current stable versions of ROOT only support CPU parallelism to compute histograms.

The goal of this project is to develop efficient batch histogramming implementations in ROOT that make use of GPUs.

ROOT Code Example

The ROOT code in Listing 1 shows how multiple 1D histograms can be generated from a single `NDatavec`. In this example, the data is filtered and 4 virtual columns are defined. Based on the newly defined columns, 4 histograms are generated.

```
ROOT -> MakeFrame -fHDataVecVec_4columns.cxx
***
void HDataVecVec::HDataVecVec(int nEvents) {
  mDataVec.Resize(nEvents);
  mDataVec.Fill(1);
  mDataVec.Fill(2);
  mDataVec.Fill(3);
  mDataVec.Fill(4);
  mDataVec.Fill(5);
  mDataVec.Fill(6);
  mDataVec.Fill(7);
  mDataVec.Fill(8);
  mDataVec.Fill(9);
  mDataVec.Fill(10);
  mDataVec.Fill(11);
  mDataVec.Fill(12);
  mDataVec.Fill(13);
  mDataVec.Fill(14);
  mDataVec.Fill(15);
  mDataVec.Fill(16);
  mDataVec.Fill(17);
  mDataVec.Fill(18);
  mDataVec.Fill(19);
  mDataVec.Fill(20);
  mDataVec.Fill(21);
  mDataVec.Fill(22);
  mDataVec.Fill(23);
  mDataVec.Fill(24);
  mDataVec.Fill(25);
  mDataVec.Fill(26);
  mDataVec.Fill(27);
  mDataVec.Fill(28);
  mDataVec.Fill(29);
  mDataVec.Fill(30);
  mDataVec.Fill(31);
  mDataVec.Fill(32);
  mDataVec.Fill(33);
  mDataVec.Fill(34);
  mDataVec.Fill(35);
  mDataVec.Fill(36);
  mDataVec.Fill(37);
  mDataVec.Fill(38);
  mDataVec.Fill(39);
  mDataVec.Fill(40);
  mDataVec.Fill(41);
  mDataVec.Fill(42);
  mDataVec.Fill(43);
  mDataVec.Fill(44);
  mDataVec.Fill(45);
  mDataVec.Fill(46);
  mDataVec.Fill(47);
  mDataVec.Fill(48);
  mDataVec.Fill(49);
  mDataVec.Fill(50);
  mDataVec.Fill(51);
  mDataVec.Fill(52);
  mDataVec.Fill(53);
  mDataVec.Fill(54);
  mDataVec.Fill(55);
  mDataVec.Fill(56);
  mDataVec.Fill(57);
  mDataVec.Fill(58);
  mDataVec.Fill(59);
  mDataVec.Fill(60);
  mDataVec.Fill(61);
  mDataVec.Fill(62);
  mDataVec.Fill(63);
  mDataVec.Fill(64);
  mDataVec.Fill(65);
  mDataVec.Fill(66);
  mDataVec.Fill(67);
  mDataVec.Fill(68);
  mDataVec.Fill(69);
  mDataVec.Fill(70);
  mDataVec.Fill(71);
  mDataVec.Fill(72);
  mDataVec.Fill(73);
  mDataVec.Fill(74);
  mDataVec.Fill(75);
  mDataVec.Fill(76);
  mDataVec.Fill(77);
  mDataVec.Fill(78);
  mDataVec.Fill(79);
  mDataVec.Fill(80);
  mDataVec.Fill(81);
  mDataVec.Fill(82);
  mDataVec.Fill(83);
  mDataVec.Fill(84);
  mDataVec.Fill(85);
  mDataVec.Fill(86);
  mDataVec.Fill(87);
  mDataVec.Fill(88);
  mDataVec.Fill(89);
  mDataVec.Fill(90);
  mDataVec.Fill(91);
  mDataVec.Fill(92);
  mDataVec.Fill(93);
  mDataVec.Fill(94);
  mDataVec.Fill(95);
  mDataVec.Fill(96);
  mDataVec.Fill(97);
  mDataVec.Fill(98);
  mDataVec.Fill(99);
  mDataVec.Fill(100);
}
```

Listing 1: Example ROOT code to generate 4 histograms.

References

- [1] R. Brun, F. Rademakers, *J. Phys.: Conf. Ser.* **160**, 012001 (2008).
- [2] CERN, *Compact Linear Collider (CLIC) Study*, CERN-2006-004, CERN, Geneva (2006).
- [3] HEP Software Foundation, J. Allison, A. A. Abue, et al., "A roadmap for HEP software and computing 2020 to the 2030s", *Computing and Software for Big Science*, vol. 3, pp. 1-40, 2019.

Optimizations

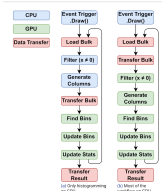


Figure 3: High-level overview of two implementations for heterogeneous architectures.

Implementation A (Fig. 3a) Only implements a parallel histogramming based on the GPU.

- Filter and column generation still performed on the CPU.
- Four columns must be transferred from CPU to GPU (n_1, n_2, n_3 , and n_4).

Implementation B (Fig. 3b) Moves the filtering and column generation to the GPU.

- No computation performed on the CPU.
- Only two columns must be transferred from CPU to GPU (n_1 and n_2).
- Increased computational intensity (data transfer vs computation) on the GPU.

✉ kevin.nobel@cern.ch

CERN Summer Student Poster Session

25 July 2024

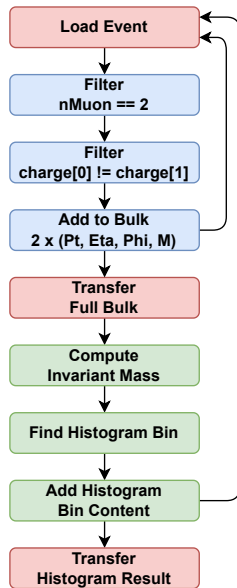
DiMuon

- Porting DiMuon analysis¹ from the ROOT tutorials
- Calculate invariant mass of all events with exactly 2 muons with opposite charge
- Not necessarily a good use-case
 - Just one histogram
 - Transfer 8 doubles to fill a single bin
 - Start with something simple

¹root.cern/doc/master/df102...NanoAODDimuonAnalysis_8C.html

DiMuon Approach

- Discard irrelevant events on CPU
- Calculate invariant mass on GPU
- Fill histogram on GPU



DiMuon

Results

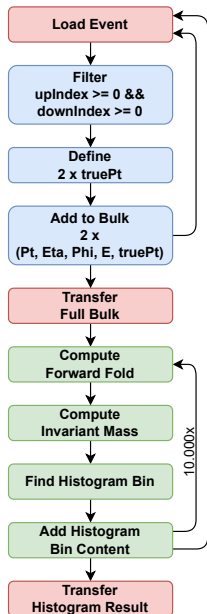
- CPU: Ryzen 7 5700g, GPU: RTX 3060
- 24.067.843 events
- Speedup: $2.6\times$ over 16 threads CPU impl.
- Data transfer: 57.5%

Folded W Mass

- 10.000 defines in a for loop
- All on just a few columns
- Varying scale + resolution for each define
 - 100 scales x 100 resolutions

Folded W Mass Approach

- Discard irrelevant events on CPU
- Calculate truePt values on CPU
- Apply forward folding on GPU
- Calculate invariant mass on GPU
- Fill histogram on GPU



Folded W Mass Results

- CPU: Ryzen 7 5700g, GPU: RTX 3060
- Tested with 100.000 events (1 billion bins filled)
- Speedup: $95\times$ over 16 threads CPU impl.
- Data transfer: 0.1%

Future Work

- Develop a generic solution to execute `.Define()` and `.Filter()` on GPUs
- Integrate GPU histogramming into ROOT

Conclusion

- Implemented a generic batch histogramming GPU kernel
- Explored how to potentially optimize `.Define()` on GPUs
- Presented my work during the Summer Student Poster Presentation

- Code is published on GitHub²

²github.com/tweska/cern-ssp/