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)

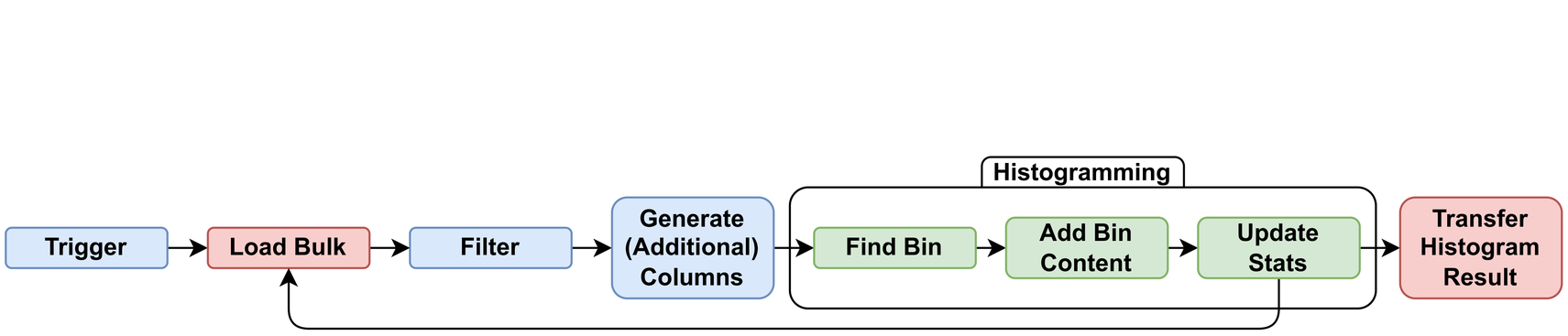


Figure 1. Typical HEP analysis workflow in ROOT.

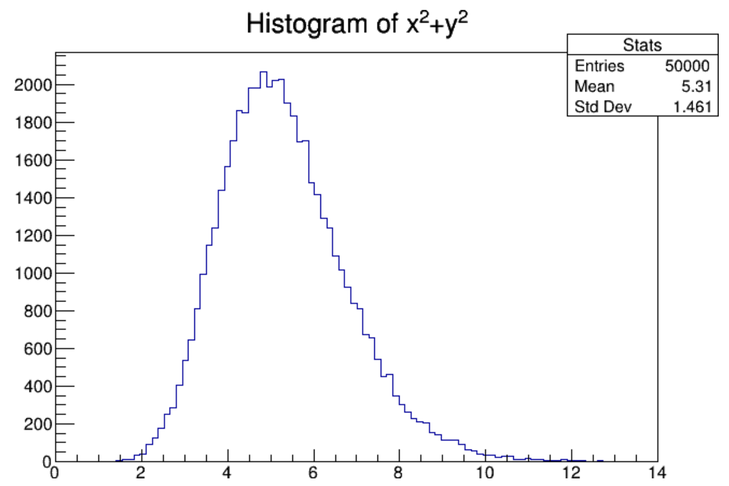


Figure 2. Example of a 1D ROOT histogram.

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 upcomming High Luminostiy 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 infrastructure. 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.

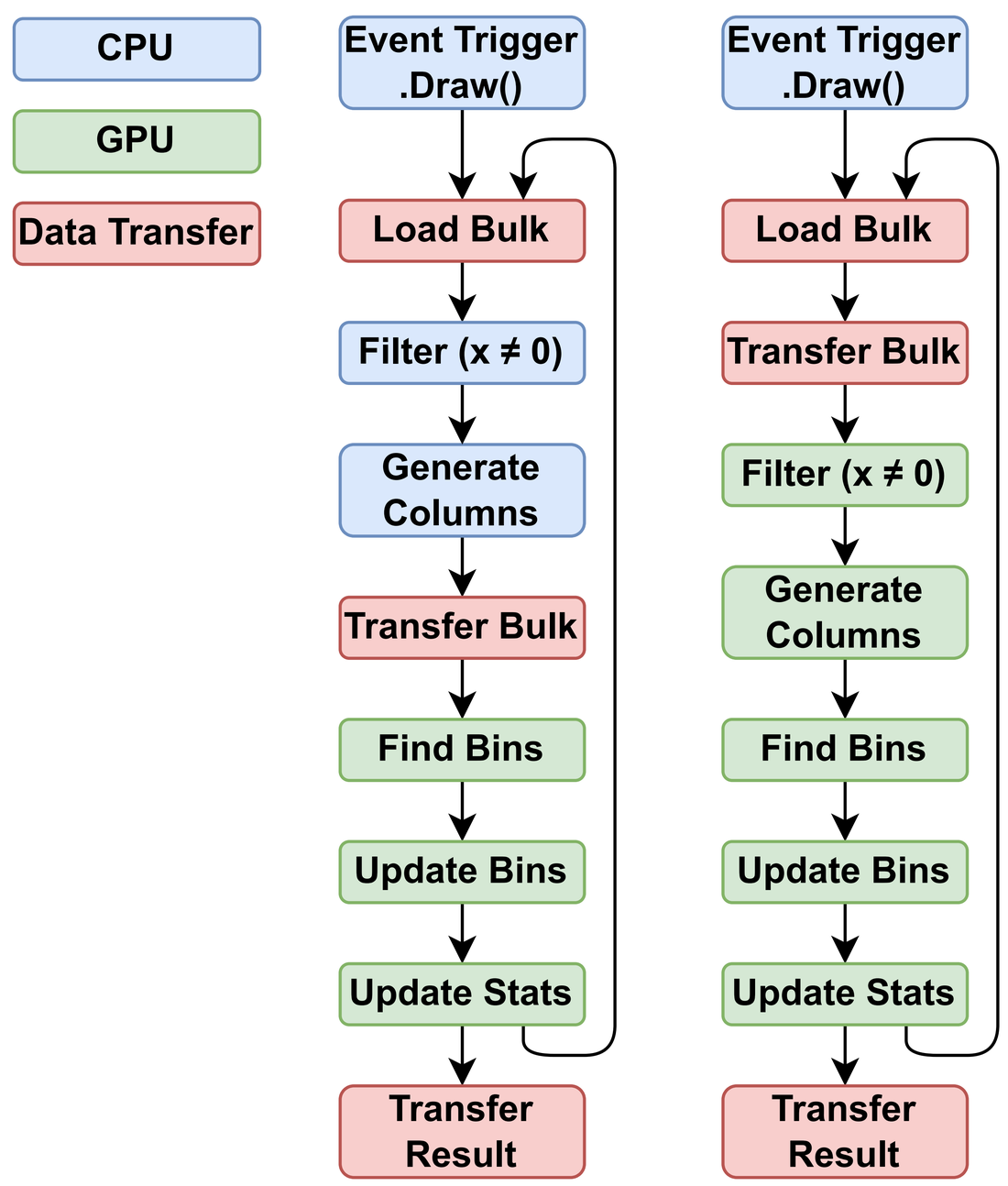
ROOT Code Example

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

```
ROOT::RDataFrame df(dataSource, bulkSize);

auto dfFiltered = df.Filter("x != 0")
                    .Define("v0", "x + y")
                    .Define("v1", "x*x + y")
                    .Define("v2", "x + y*y")
                    .Define("v3", "x*x + y*y");

auto h0 = dfFiltered.Histo1D<double>("v0");
auto h1 = dfFiltered.Histo1D<double>("v1");
auto h2 = dfFiltered.Histo1D<double>("v2");
auto h3 = dfFiltered.Histo1D<double>("v3");

h0->Draw(); h1->Draw(); h2->Draw(); h3->Draw();
```

Listing 1. Example ROOT code to generate 4 histograms.

Optimizations



(a) Only histogramming on GPU.

(b) Most of the workflow on GPU.

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

Implementation A (fig 3a): Only implements a parallel histogramming kernel on the GPU:

- Filter and column generation still performed on the CPU.
- **Four columns** must be transferred from CPU to GPU (v_0, v_1, v_2 and v_3).

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 (x and y).
- Increased **computational intensity** (data transfer vs computation) on the GPU.

References

[1] R. Brun, F. Rademakers, P. Canal, *et al.*, *ROOT*, version v6-18-02, Jun. 2020. DOI: 10.5281/zenodo.3895860. [Online]. Available: https://doi.org/10.5281/zenodo.3895860.

[2] HEP Software Foundation, J. Albrecht, A. A. Alves, *et al.*, "A roadmap for HEP software and computing R&D for the 2020s," *Computing and software for big science*, vol. 3, pp. 1–49, 2019.