Data Analysis with GPU-Accelerated Kernels

ICHEP 2020

28 July, 2020

Irene Dutta[1], Nan Lu[1], Harvey Newman[1], Joosep Pata[1], Maria Spiropulu[1], Jean-Roch Vlimant[1], Christina Reissel[2], Daniele Ruini[2]

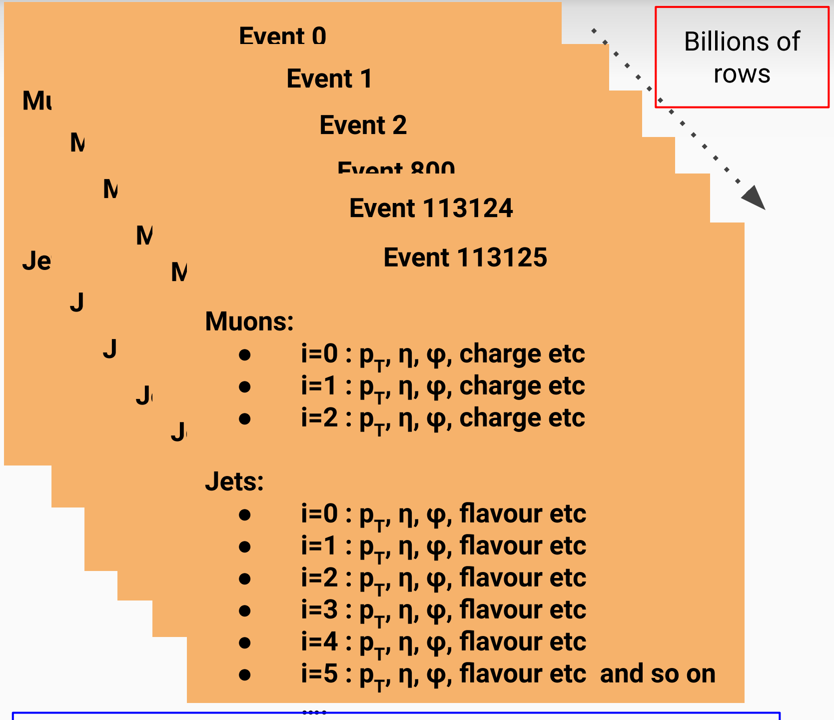
[1]California Institute of Technology
[2]ETH Zurich

Introduction

- A typical data analysis from a collider experiment (CMS or ATLAS) involves running over 10 TBs of data and simulation samples repeatedly over a period of a year or longer.
- Typical **compressed event sizes** for reduced data formats is **few kilobytes per event** (for eg CMS NANOAOD or the final ROOT skimmed ntuples used in any analysis)
- For each iteration of the analysis → few hundreds of batch jobs
- Few hundred iterations over the course of a year → considerable time spent in computation
- **GOAL** : Reduce complexity and increase speed of these workflows→ deliver results from large datasets with faster turn-around times

hepaccelerate: efficient analysis methods

- The standard HEP software framework based on ROOT → dynamically-sized arrays, complete C++ classes with arbitrary structure
- High speed parallel computing with GPUs and FPGAs is increasingly popular these days.
- We developed an array based HEP computational analysis framework that is suitable for such parallel architecture needs: hepaccelerate.
- This is based on the approach first introduced in uproot and awkward-array python libraries



Typical ROOT HEP data format: Stacks of events with variable lengths of particle properties

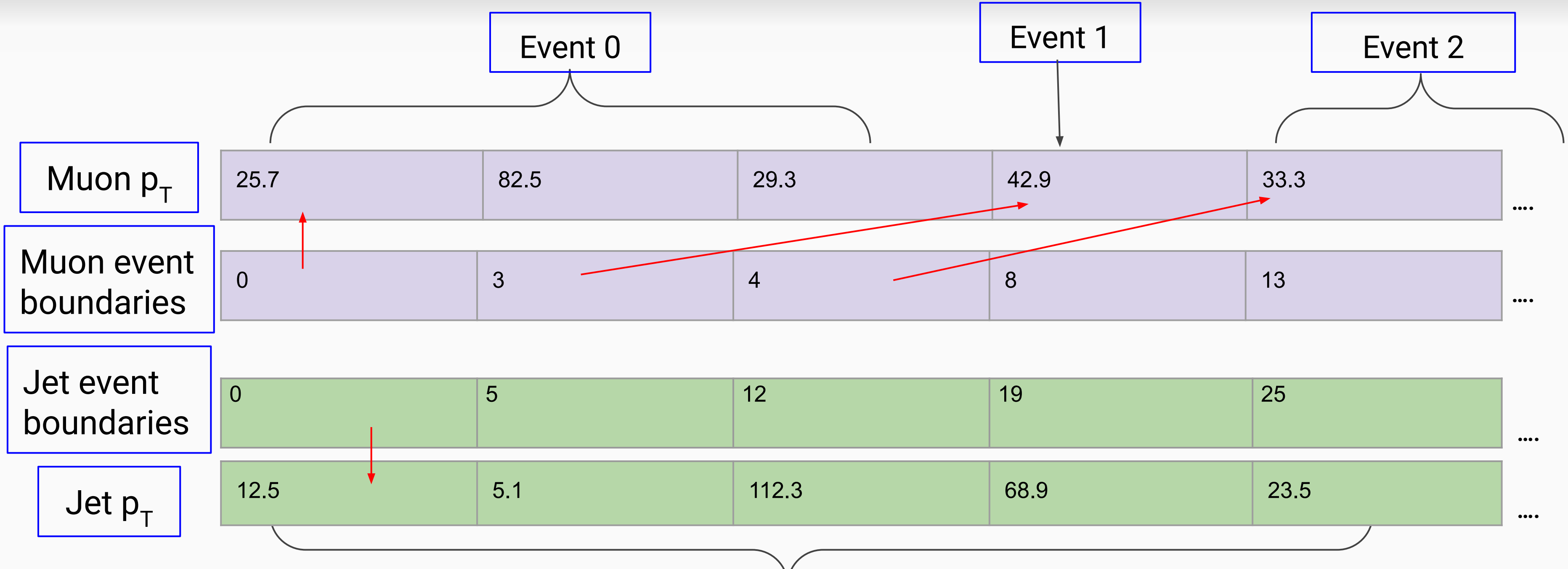
hepaccelerate: efficient analysis methods

We follow these simple steps to carry out an analysis with our new library:

1. Transform the data from a compressed events format to linear arrays of particle properties (using awkward-arrays)
2. Perform parallel computations on linear arrays using kernels
3. Save results

Disclaimer : The emphasis here is to show the computational performance and not reproduce already public physics results !

Transforming the data structure



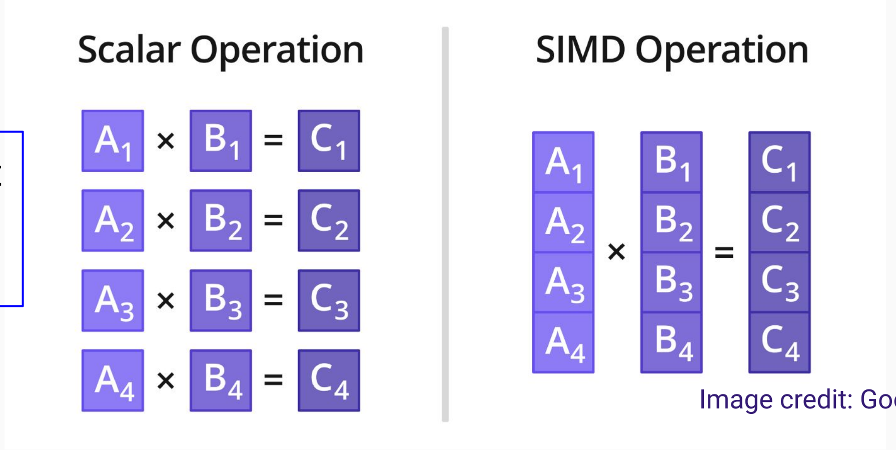
Varying number of particles per event →loaded as sparse arrays with an underlying one-dimensional array for a single feature.

Linear arrays of particle properties with an additional array to mark event boundaries - introduced by awkward arrays

Computational kernels

- **Kernel:** a function that is evaluated on all elements of an array. For eg. compute the square root of all the values in an array
- If individual kernel calls across the data are independent of each other → evaluate in parallel using single-instruction, multiple-data (SIMD) processors.

Loop over each element of type A and type B to produce type C.



Matrix multiplication of column A with column B to produce column C.

Image credit: Google images

Computational kernels

- Columnar data analysis approach based on single-threaded kernels is already recognized in HEP using the Coffea tool.
- hepaccelerate extends the computational efficiency and scalability of the kernels to parallel hardware such as multi-threaded CPUs and propose a GPU implementation.
- Idea :
  - No looping over events to calculate variables per event ❌
  - Use linear arrays to perform parallel computation of physics variables across all events → save time on expensive `for` loops ✔️

Example code : sum p_T of jets i.e. H_T

```
def sum_ht(
  pt_data, offsets,
  mask_rows, mask_content,
  out):

  N = len(offsets) - 1
  M = len(pt_data)

  #loop over events in parallel
  for iev in prange(N):
    if not mask_rows[iev]:
      continue

    #indices of the particles in this event
    i0 = offsets[iev]
    i1 = offsets[iev + 1]

    #loop over particles in this event
    for ielem in range(i0, i1):
      if mask_content[ielem]:
        out[iev] += pt_data[ielem]
```

CPU multi-threading enabled with Numba package; For GPUs, use CUDA (example in backup)

If event mask is 0, skip event

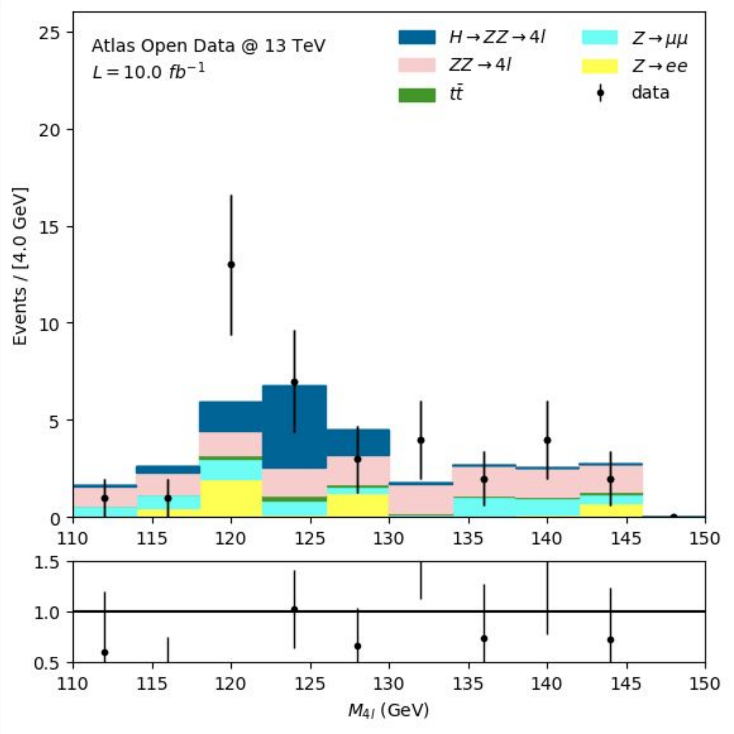
If jet mask is 1, add jet p_T to the sum

- `offsets`: 1D array marking event boundaries (length: N_events+1)
- `pt_data` : 1D array of jet p_T
- `mask_rows` : boolean mask of events (stores information of events passing selections; length: N_events)
- `mask_content`: boolean mask of jets (stores information of jets passing selections)
- `out`: Value of H_T (length: N_events)

Some other such generic kernels are also already available in the library

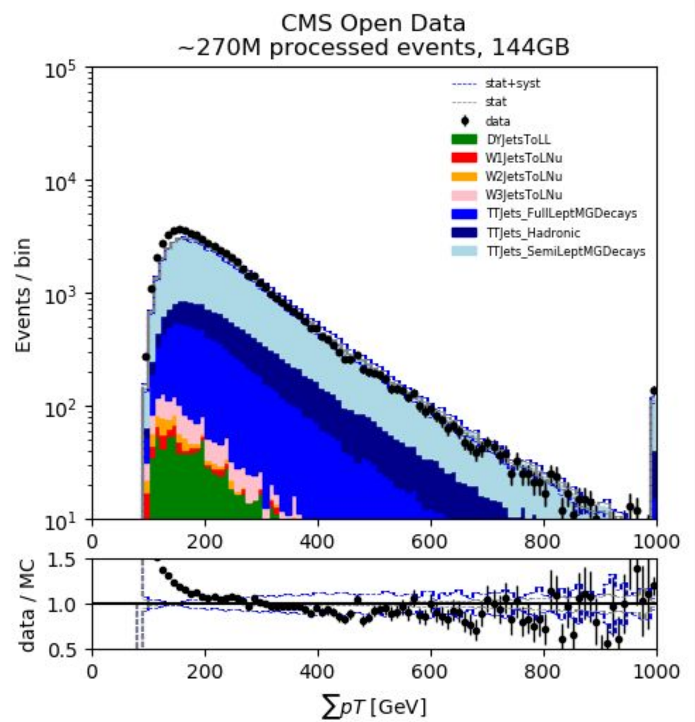
Benchmarking hepaccelerate with CERN Open Data

H→ZZ→4l with the 13 TeV [Atlas Open Data]



Works well for data formats from different experiments

Top quark pair analysis using 8TeV [CMS Open Data]from 2012.



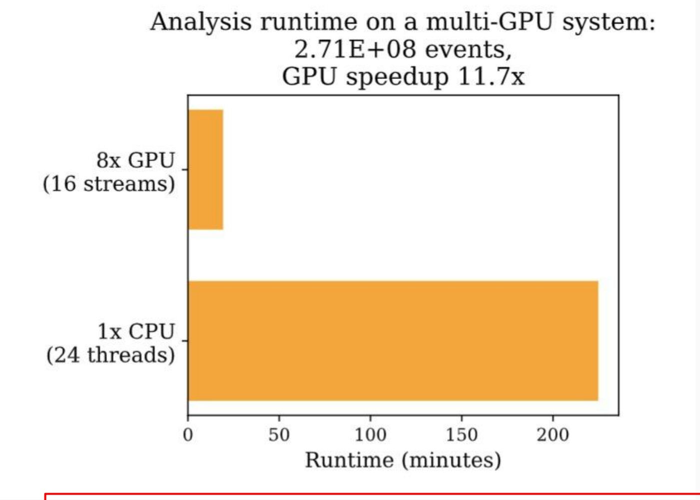
Reproduce our CMS Open Data results from [here]

Benchmarking hepaccelerate with CMS Open Data

- Top quark pair analysis using [CMS Open Data]from 2012.
- Results on 144 GB of CMS Open data.
- **Goal** : Study computational performance
- The benchmark analysis implements the following features:
  - event selections and object selections : trigger bit, missing transverse energy selection, jet/lepton selections based on p_T, η etc
  - event weight computation based on histogram lookups: pileup re-weighting, lepton efficiency corrections
  - jet energy correction systematics based on histogram lookups (computational complexity ~40x higher)
  - high-level variable reconstruction: top quark candidate from jet triplet with invariant mass closest to 173 GeV
  - Multilayer, feedforward DNN evaluation using tensorflow with ~40 typical high-level inputs
  - saving all DNN inputs and outputs, along with systematic variations to ~ 1000 histograms

Benchmarking performance with CMS Open Data

- We observe the following things:
  - GPU-accelerated version performs ~12x faster than a single multi-threaded CPU.
  - complex analysis where the main workload is repeated around 40x (for eg. applying full set of jet energy correction systematics)→ 15x faster on a GPU-version than on a CPU.
- Important to balance overhead of kernel scheduling with the time spent in the computation → run on large datasets .
- Encouraging to see physics analysis methods can be implemented easily on GPUs
- A small number of multi-GPU machines can be viable for the future → choice driven by availability and pricing of resources.



Use 8 Nvidia GTX 1080 GPUs, 2 compute streams per device → reduce the analysis runtime by a factor of 12x compared to using multiple threaded CPU

Summary and Outlook

- We demonstrate the possibility of carrying out high-energy physics data analysis with
  - Efficient input data preparation using linear arrays
  - Using specialized kernels for parallel computation on arrays (implemented in Python using the Numba package for multi-threaded CPU)
- Also possible to do these array computations using GPUs, which are highly efficient at parallel processing .
- This library is generic and can be used on data formats from different collider experiments.
- We show that it's possible to run an order of magnitude faster on a multi-GPU machine as compared to using a single multi-threaded CPU.

Backup

Some generic kernels

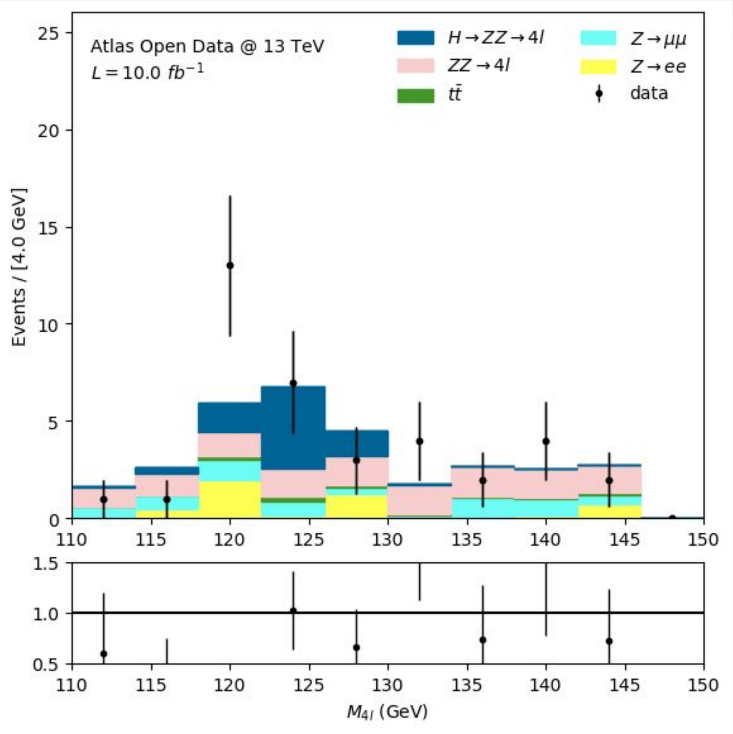
Some general purpose kernels already available in the library :

- `sum_in_offsets`: given jagged data with offsets, calculates the sum of the values within the rows. For eg. compute observables such as H_T.
- `fill_histogram`: given a data array, a weight array, histogram bin edges and contents, fills the weighted data to the histogram. This is used to create 1-dimensional histograms that are common in HEP. Extension to multidimensional histograms is straightforward.
- `get_bin_contents`: given a data array and a lookup histogram, retrieves the bin contents corresponding to each data array element.

And so on ....

Benchmarking with Atlas Open Data

- Reproduce the H→ZZ→4l with the [Atlas Open Data]
- **Goal**: Show reproducibility with different data formats
- The benchmark analysis implements the following features:
  - object selections : lepton selections based on pT, η, charge etc
  - event weight computation
  - high-level variable reconstruction: Invariant mass of 4 leptons



Example code : sum p_T of jets (i.e. H_T) using GPUs

```
@cuda.jit
def sum_ht_cudakernel(
  pt_data, offsets,
  mask_rows, mask_content,
  out):

    xi = cuda.grid(1)
    xstride = cuda.gridsize(1)
    for iev in range(xi, offsets.shape[0]-1, xstride):
        if mask_rows[iev]:
            start = np.uint64(offsets[iev])
            end = np.uint64(offsets[iev + 1])

            #loop over particles in this event
            for ielem in range(start, end):
              if mask_content[ielem]:
                out[iev] += pt_data[ielem]
```

Run in parallel over GPUs using CUDA

If event mask is 0, skip event

If jet mask is 1, add jet p_T to the sum

- `offsets`: 1D array marking event boundaries (length: N_events+1)
- `pt_data` : 1D array of jet p_T
- `mask_rows` : boolean mask of events (stores information of events passing selections; length: N_events)
- `mask_content`: boolean mask of jets (stores information of jets passing selections)
- `out`: Value of H_T (length: N_events)

Minimal changes to code to run over GPU !

Benchmarking with CMS Open Data

job type	partial systematics	full systematics
processing speed (kHz)		
1 thread	50	1.4
4 threads	119	4.0
GPU	440	20
walltime to process a billion events (hours)		
1 thread	5.5	200
4 threads	2.3	70
GPU	0.6	13