Data Analysis with GPU-Accelerated Kernels

J. Pata,[a] I. Dutta,[a,*] N. Lu,[a] J. R. Vlimant,[a] H. Newman,[a] M. Spiropulu,[a] C. Reissel[b] and D. Ruini[b]

[a] *California Institute of Technology,*
1200 E California Blvd, Pasadena, CA, USA 91125

[b] *ETH Zurich*
Rämistrasse 101, 8092 Zürich, Switzerland

E-mail: idutta@caltech.edu, jpata@caltech.edu, reielc@ethz.ch

At HEP experiments, processing billions of records of structured numerical data can be a bottleneck in the analysis pipeline. This step is typically more complex than current query languages allow, such that numerical codes are used. As highly parallel computing architectures are increasingly important in the computing ecosystem, it may be useful to consider how accelerators such as GPUs can be used for data analysis. Using CMS and ATLAS Open Data, we implement a benchmark physics analysis with GPU acceleration directly in Python based on efficient computational kernels using Numba/LLVM, resulting in an order of magnitude throughput increase over a pure CPU-based approach. We discuss the implementation and performance benchmarks of the physics kernels on CPU and GPU targets. We demonstrate how these kernels are combined to a modern ML-intensive workflow to enable efficient data analysis on high-performance servers and remark on possible operational considerations.

40th International Conference on High Energy physics - ICHEP 2020
July 28 - August 6, 2020
Prague, Czech Republic (virtual meeting)

*Speaker

1. Introduction

At the general-purpose experiments of the Large Hadron Collider such as CMS or ATLAS, the data formats typically consist of columns of physics related variables or features for the recorded particles such as electrons, muons, jets and photons for each event in billions of rows. In addition to the columns of purely kinematic information, each particle carries other reconstruction details, for example, the number of tracker layers with an associated hit for a muon. A typical event for such reduced data formats are on the order of a few kilobytes per event.

A typical physics analysis at the LHC requires tens to hundreds of iterations over such datasets using batch jobs. By reducing the complexity and increasing the speed of these workflows, HEP experiments could deliver results from large datasets with faster turn-around times.

Recently, heterogeneous and highly parallel computing architectures beyond consumer x86 processors such as GPUs, TPUs and FPGAs have become increasingly prevalent in scientific computing. We investigate the use of array-based computational kernels that are well-suited for parallel architectures for carrying out HEP data analysis using the example kernels in the hepaccelerate [1] library.

2. Data structure

We can represent HEP collider data as 2D matrices, where the rows correspond to events and columns correspond to features in the event. The standard HEP software framework based on ROOT represents this as dynamically-sized arrays and complete C++ classes with arbitrary structure [2].

Based on the approach first introduced in the `uproot` [3] and `awkward-array` [4] python libraries, HEP data files with a varying number of particles per event can be represented and efficiently loaded as sparse arrays with an underlying one-dimensional array for a single feature. Event boundaries are encoded in an offset array that records the particle count per event. Therefore, the full structure of N events, M particles in total, can be represented by a contiguous offset array of size $N + 1$ and a contiguous data array of size M for each particle feature.

3. Computational kernels

In the context of this report, a kernel is a function that is evaluated on the elements of an array to transform the underlying data, for example, computing the square root of all values in an array. When the individual kernel calls across the data are independent of each other, they can be evaluated in parallel over the data using single-instruction, multiple-data (SIMD) processors.

We note that the columnar data analysis approach based on single-threaded kernels is already recognized in HEP [5]. Our contribution is to further extend the computational efficiency of the kernels to parallel hardware such as multi-threaded CPUs and propose a GPU implementation.

A prototypical HEP-specific kernel would be to find the scalar sum $H_T = \sum_{i \in \text{event}} p_T^i$ of all particles passing some quality criteria in an event. We show the Python implementation for this on Listing 1 (for multi-threading on CPUs) and on Listing 2 (for GPUs).

Other generic kernels that turn out to be useful are related to finding the minimum or maximum value within the event offsets (for example, finding the jet with the highest p_T in an event) or

```
 1 def sum_ht(
 2   pt_data, offsets,
 3   mask_rows, mask_content,
 4   out):
 5 
 6   N = len(offsets) - 1
 7   M = len(pt_data)
 8 
 9   #loop over events in parallel
10   for iev in prange(N):
11     if not mask_rows[iev]:
12       continue
13 
14     #indices of the particles in this event
15     i0 = offsets[iev]
16     i1 = offsets[iev + 1]
17 
18     #loop over particles in this event
19     for ielem in range(i0, i1):
20       if mask_content[ielem]:
21         out[iev] += pt_data[ielem]
```

Listing 1: Python code for the kernel computing the scalar sum of selected particle momenta H_T. The inputs are `pt_data`, an M-element array of p_T data for all the particles, the $N + 1$-element `offsets` array with the indices between the events in the particle collections, as well masks for events and particles that should be considered. On line 10, the kernel is executed in parallel over the events using the Numba `prange` iterator, which creates multi-threaded code across the loop iterations. On line 19, the particles in the event are iterated over sequentially.

retrieving or setting the m-th value of an array within the event offsets. Several such kernels can already be found in [1].

These kernels have been implemented in Python and just-in-time compiled to either multi-threaded CPU code using the Numba package [6] or using CUDA [7] for GPUs. We have chosen Python and Numba to implement the kernels in the spirit of quickly prototyping this idea, but this approach is not restricted to a particular programming environment.

4. Analysis benchmark

CMS Open Data

We benchmark this approach in a prototypical top quark pair analysis using CMS Open Data from 2012 [8]. The datasets are processed from the experiment-specific format to a columnar format using a publicly-available tool [9] and correspond to about 60GB of simulation and 40GB of data with the single muon trigger . Our objective here is to replicate the computing conditions that are encountered in HEP data analyses and not to reproduce published physics results.

The benchmark analysis implements the following features in a single end-to-end pass:

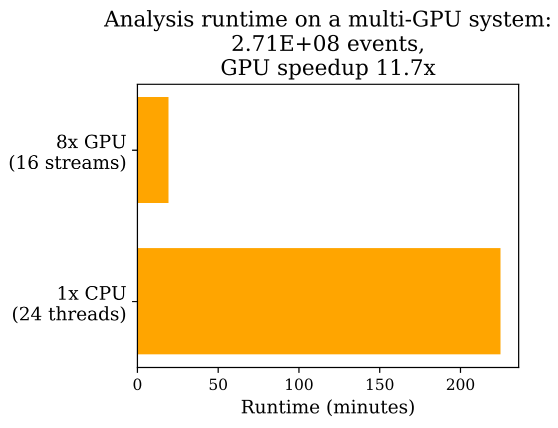
- event selection based on event variables: trigger bit, missing transverse energy selection

```
 1  @cuda.jit
 2  def sum_ht_cudakernel(
 3    pt_data, offsets,
 4    mask_rows, mask_content,
 5    out):
 6
 7      xi = cuda.grid(1)
 8      xstride = cuda.gridsize(1)
 9      for iev in range(xi, offsets.shape[0]-1, xstride):
10          if mask_rows[iev]:
11              start = np.uint64(offsets[iev])
12              end = np.uint64(offsets[iev + 1])
13
14              #loop over particles in this event
15              for ielem in range(start, end):
16                if mask_content[ielem]:
17                  out[iev] += pt_data[ielem]
```

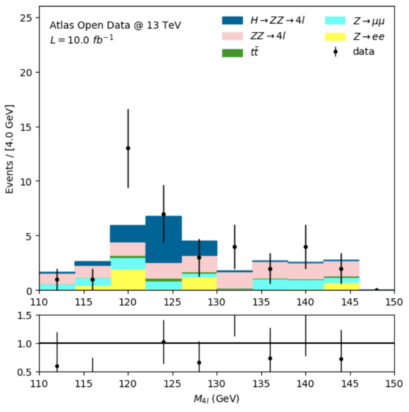
Listing 2: Python code for the kernel computing the scalar sum of selected particle momenta H_T. The inputs are `pt_data`, an array of p_T data for all the particles, the `offsets` array with the indices between the events in the particle collections, as well masks for events and particles that should be considered. On line 9, the kernel is executed in a GPU compatible style which is parallel over the events using CUDA. On line 15, the particles in the event are iterated over sequentially.

- object selections : jet, lepton selection based on p_T, η
- matching of pairs of objects: jet-lepton ΔR matching, jet to generator jet matching
- event weight computation based on histogram look-ups: pileup, lepton efficiency corrections
- jet energy corrections based on histogram look-ups: *in-situ* jet energy systematics, increasing the computational complexity of the analysis by about 40x
- high-level variable reconstruction: top quark candidate from jet triplet with invariant mass closest to $M = 172$ GeV
- evaluation of $\simeq 40$ typical high-level inputs to a deep neural network (DNN)
- Multi-layer, feed-forward DNN evaluation using `tensorflow`
- saving all DNN inputs and outputs, along with systematic variations, to histograms ($\simeq 1000$ individual histograms)

We perform two benchmark scenarios of this analysis: one with a partial set of systematic uncertainties to emulate a simpler IO-limited analysis, and one with the full set of systematic uncertainties to test a compute-bound workload. The timing results from the benchmark are reported on Fig. 1a. In the former case, we observe the GPU-accelerated version performing about 12x faster than a single multi-threaded CPU. For the latter case, where the main workload is repeated around 40x, a single GPU is about 15x faster than 1 multi-threaded CPU.



(a)



(b)

Figure 1: (a) Benchmarks of the full analysis with 270M events, 100GB of numerical data on a multi-GPU system. We find that by using 8x nVidia GTX 1080 GPUs, 2 compute streams per device, we can reduce the analysis runtime by a factor of 12x, compared to using multiple threads on the CPU alone. (b) The 4 lepton invariant mass from the benchmark analysis of $H \longrightarrow ZZ \longrightarrow 4l$ implemented using ATLAS Open Data.

We have also found that it is beneficial to ensure that the kernels are called on sufficiently large datasets to reduce the overhead of kernel scheduling with respect to the time spent in the computation.

It is promising to see that most of the physics analysis methods can be implemented with relative ease on a GPU, such that with further optimizations, a small number of multi-GPU machines might be a viable option in the future (depending upon the availability and pricing of resources).

ATLAS Open Data

To demonstrate the capability of [1] to run on data formats from different collider experiments, we tried to reproduce the classic $H \longrightarrow ZZ \longrightarrow 4l$ analysis with the 13 TeV ATLAS Open Data [10]. The benchmark analysis implements the following features:

- object selection based on lepton variables: p_T, η, charge
- event weight computation based on lepton efficiency corrections, trigger weights
- high-level variable reconstruction: 4 lepton invariant mass closest to $M = 125$ GeV

The invariant 4 lepton mass is shown in Fig. 1b.

5. Summary

We demonstrate that it is possible to carry out prototypical end-to-end high-energy physics data analysis using a relatively simple code in a HPC context. This is achieved with efficient input data preparation, array processing approaches and a small number of specialized kernels for jagged arrays implemented in Python using the Numba package. It is also possible to offload parts of these array computations to accelerators such as GPUs, which are highly efficient at parallel processing.

We demonstrate that such kernels can run an order of magnitude faster on a multi-GPU machine as compared to using a multi-threaded CPU approach.

Acknowledgments

The availability of the excellent open-source Python libraries `uproot`, `awkward`, `coffea`, `numba`, `cupy` and `numpy` as well as the CERN Open Data was imperative for this project. We are grateful to the developers, maintainers and funders of these open-source projects. This work was partly conducted at "*iBanks*", the AI GPU cluster at Caltech. We acknowledge NVIDIA, SuperMicro and the Kavli Foundation for their support of "*iBanks*". This work is supported by the the Prime NSF Tier2 award 1624356 and the Caltech HEP DOE award DE-SC0011925.

References

[1] J. Pata *et al.* DOI: 10.5281/zenodo.3491423

[2] I. Antcheva *et al.*, Comput. Phys. Commun. **180**, 2499 (2009) doi:10.1016/j.cpc.2009.08.005 [arXiv:1508.07749 [physics.data-an]].

[3] J. Pivarski *et al.* `https://github.com/scikit-hep/uproot`

[4] J. Pivarski *et al.* `https://github.com/scikit-hep/awkward-array`

[5] The Coffea Team, coffea v0.6.8, 10.5281/zenodo.3358981 (2019)

[6] Lam, S.K., Pitrou, A. and Seibert, S., 2015, November. Numba: A llvm-based python jit compiler. In Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC (p. 7). ACM.

[7] Nvidia CUDA Home page `https://developer.nvidia.com/cuda-zone`

[8] CMS collaboration (2017). Simulated datasets in AODSIM format for 2012 collision data. CERN Open Data Portal. DOIs:

1. 10.7483/OPENDATA.CMS.42GY.2VJI
2. 10.7483/OPENDATA.CMS.2DSE.HYDF
3. 10.7483/OPENDATA.CMS.7RZ3.0BXP
4. 10.7483/OPENDATA.CMS.ARKO.6NV3
5. 10.7483/OPENDATA.CMS.REHM.JKUH
6. 10.7483/OPENDATA.CMS.DELK.2V7R
7. 10.7483/OPENDATA.CMS.HHCJ.TVXH
8. 10.7483/OPENDATA.CMS.DELK.2V7R
9. 10.7483/OPENDATA.CMS.KAYE.XLAH

[9] `https://github.com/cms-opendata-analyses/AOD2NanoAODOutreachTool`

[10] ATLAS collaboration (2020)- 13 TeV samples collection with at least 4 leptons (electrons or muons) `http://opendata.cern.ch/record/15005`