Hydra.Python

Python bindings for Hydra

Deepanshu Thakur

Presented at DIANA Meeting, CERN, September 11, 2017

Google Summer of Code 2017 was a learning experience for me under the supervision of Dr. Antonio Augusto Alves Jr. and Dr. Eduardo Rodrigues. The GSoC 2017 started with the goal to bind the header-only C++ Hydra library.

- Efficient Python routines for analysis on massively multi-threaded platforms / Python bindings for the Hydra C++ library
- More than 100 commits to the newly created repository.

Content

- Overview
- Main features of Hydra.Python
- Examples
- Summary

Hydra is a header-only templated C++11 compliant library designed to perform common HEP data analyses on massively parallel platforms. The goal of Hydra.Python is to provide bindings to Hydra's containers and algorithms, keeping semantics and syntax as close as it is possible to the native C++ equivalent code.

- Hydra is implemented on top of C++11 Standard Library and a variadic version of the Thrust library.
- The Hydra.Python containers are able to describe multi-dimensional data structures allocated in different memory spaces.
- The basic library used for binding is `pybind11` with some complementary `ctypes`.
- Hydra.Python also deploys some template and macro tricks and workarounds, to ease the binding work, make containers opaque, etc.

PhaseSpace class

The Hydra C++ library provides many algorithms and following the mentors advice I focused on the PhaseSpace Monte Carlo generation.

- The Hydra's implementation of the PhaseSpace MC generation class can have any number of particles in the final state.
- The Python version can have up to 10 particles in the final state. (easily extendable to more)
- The Python PhaseSpace class stores the generated decays in the Events container.

PhaseSpace example (device)

This is a basic example of showing the generation of a phase-space sample corresponding to the decay of particle B0 -> K pi J/psi in this case.

```
1  import HydraPython as hypy
2  nentries = 1000000
3  B0_mass = 5.27955  # All the masses are in GeV/c^2
4  Jpsi_mass = 3.0969
5  K_mass = 0.493677
6  pi_mass = 0.13957061
7  B0 = hypy.Vector4R(B0_mass, 0.0, 0.0, 0.0)
8  masses = [Jpsi_mass, K_mass, pi_mass]
9  phsp = hypy.PhaseSpace3(masses)  # Here 3 stands for number of final-state particles
10
11 events_d = hypy.device_events_3(nentries)
12 phsp.GenerateOndevice(B0, events_d)
```

Example: hydra::PhaseSpace (device)

```
for i in range(2): print(events_d[i])

(0.003434478211972907,
        (3.326536152819228, -0.7376241292510032, 0.9527533342879685, 0.15239715864543849),
        (0.532378104738375, -0.12361475568728784, 0.154537771441584, 0.023386651108379133),
        (1.4206357424423983, 0.8612388849382909, -1.1072911057295525, -0.1757838097538176)
)
(0.20687943370357853,
        (3.3327060111834546, -0.44741166640978447, 1.012640505284964, -0.5390007001803998),
        (0.6174301429964726, 0.2751401741060331, 0.22469710003517362, 0.10635835171875056),
        (1.3294138458200735, 0.17227149230375138, -1.237337605320138, 0.43264234846164934)
)
```

In the tuple the first element of the tuple describe the event weight and the rest of the 3 tuples describes the four-vector of the final state particles.

PhaseSpace example (host)

```
1  import HydraPython as hypy
2  nentries = 1000000
3  B0_mass = 5.27955  # All the masses are in GeV/c^2
4  Jpsi_mass = 3.0969
5  K_mass = 0.493677
6  pi_mass = 0.13957061
7  B0 = hypy.Vector4R(B0_mass, 0.0, 0.0, 0.0)
8  masses = [Jpsi_mass, K_mass, pi_mass]
9  phsp = hypy.PhaseSpace3(masses)  # Here 3 stands for number of final-state particles
10
11 events_h = hypy.host_events_3(nentries)
12 phsp.GenerateOnhost(B0, events_h)
13 # On line number 11 and 12 I just changed the word from *device* to *host*
```

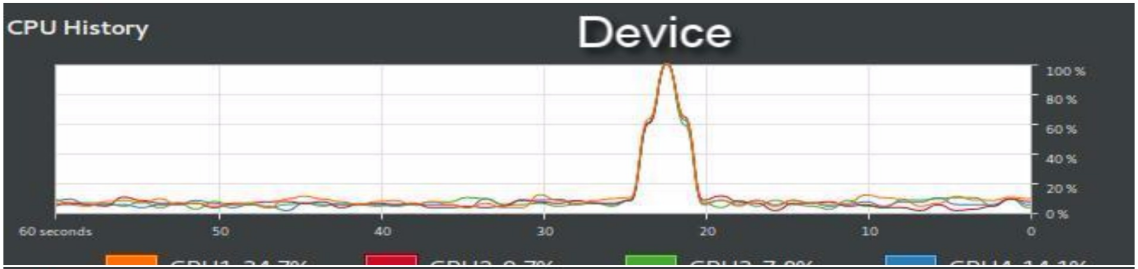
Example: hydra::PhaseSpace (host)

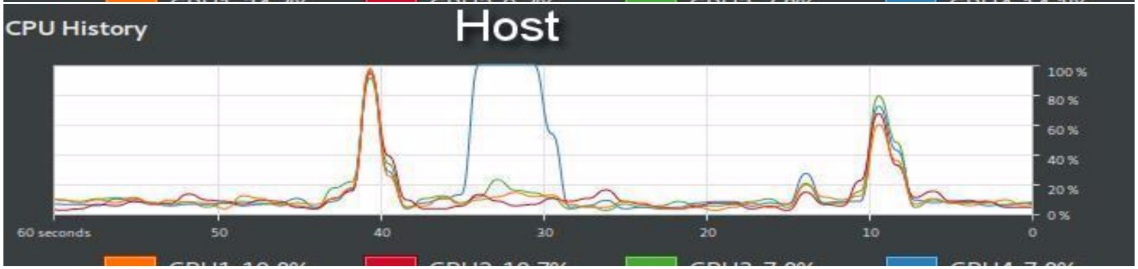
```
for i in range(2): print(events_h[i])

(0.003434478211972907,
(3.326536152819228, -0.7376241292510032, 0.9527533342879685, 0.15239715864543849),
(0.532378104738375, -0.12361475568728784, 0.154537771441584, 0.023386651108379133),
(1.4206357424423983, 0.8612388849382909, -1.1072911057295525, -0.1757838097538176)
)
(0.20687943370357853,
(3.3327060111834546, -0.44741166640978447, 1.012640505284964, -0.5390007001803998),
(0.6174301429964726, 0.2751401741060331, 0.22469710003517362, 0.10635835171875056),
(1.3294138458200735, 0.17227149230375138, -1.237337605320138, 0.43264234846164934)
)
```

Above output is exactly similar to the output of device.

Device vs Host





Device vs Host

```
1  nentries = 10000000  # 10 million
2  events_d = hypy.device_events_3(nentries)
3  events_h = hypy.host_events_3(nentries)
4
5  %timeit phsp.GenerateOndevice(B0, events_d)
6  # 1.9 s +- 23.3 ms per loop (mean +- std. dev. of 7 runs, 1 loop each)
7
8  %timeit phsp.GenerateOnhost(B0, events_h)
9  # 4.23 s +- 42 ms per loop (mean +- std. dev. of 7 runs, 1 loop each)
```

Benchmarking is done on Intel® Core™ i5-5200U CPU @ 2.20GHz x 4

Example: hydra::Random

Sampling a 3D Gaussian distribution:

```
In [1]: import HydraPython as hypy
In [2]: import math
...: def Gauss3D(*args):
...:     g = 1.0
...:     mean = 0
...:     sigma = 1.0
...:     for i in range(3):
...:         m2, s2 = ((args[i] - mean) * (args[i] - mean)), (sigma * sigma)
...:         g *= math.e ** ((-m2/(2.0 * s2 ))/( math.sqrt(2.0*s2*math.pi)))
...:     return g

In [3]: container = hypy.host_vector_float3(200)
In [4]: RNGen = hypy.Random()
In [5]: sample = RNGen.Sample(container, [-5, -5, -5], [5, 5, 5], Gauss3D)
```

Example: hydra::Random (continue...)

```
In [6]: for s in sample: print(s)
(0.4219701517577934, -1.0787015977117247, -0.004686508332087391)
(1.104623969741747, 1.3037309621495485, 2.3973570041333856)
(-1.9187522418040337, -0.08930860561254306, -1.0155871287133422)
(-1.5125352763687587, -1.5901792110783786, 0.4594501671003648)
(-1.5901792110783786, 0.4594501671003648, -1.8808818393208848)
(0.4594501671003648, -1.8808818393208848, -2.047111193693347)
(0.593116419010904, 0.3227611308198064, 0.0026509352053061264)
(0.3227611308198064, 0.0026509352053061264, -2.0365943778647058)
(0.0026509352053061264, -2.0365943778647058, 1.5529442593017073)
(0.8387782432500068, -1.3353265461840913, 2.4524315516021398)
(0.9237460195308045, 0.14420039499569715, 0.6973759836492832)
(1.4777260334023512, 1.3134375366507445, 0.9434145325267818)
(1.3134375366507445, 0.9434145325267818, -0.4370092185558838)
(0.22710920332661755, 2.788461062860174, 1.8040150327645377)
(0.3127681327171328, -2.369360255421475, -1.3887238049774653)
```

- In this example the host back-end is CPP.

Summary

- The project concluded with the first official release, 0.1.0, of Hydra.Python.
- Test cases, documentation and examples have been prepared together with the development of the bindings.
- I have decided to continue with the project outside the scope of GSoC.
- You all are invited to contribute to the project: /MultithreadCorner/Hydra.Python
- Check out the documentation
- Join our gitter channel
- The GSoC 2017 project report on