BE-ABP: Beam Dynamics on the GPU

SixTrackLib + PyHEADTAIL:

Summary of Day 1

Riccardo de Maria, Lotta Mether, Adrian Oeftiger, Martin Schwinzerl

CERN – Openlab E4 Hackathon

15-17 April 2019

Goal

Starting point:

- **two existing codes:**
  `SixTrackLib` (templated C) and `PyHEADTAIL` (Python + PyCUDA)

⟶ **merge functionality:** single-particle tracking + multi-particle dynamics

- abstracted PyHEADTAIL in `CuPy`: jupyter notebook ↗ on github ↗

Goals:

- extend this python script based on PyHEADTAIL to
  - done: prepare the accelerator optics in SixTrackLib in chunks (not SixTrackLib's usual one-turn behaviour)
  - done: share the macro-particle coordinates / memory from PyHEADTAIL with SixTrackLib
  - o.t.w.: transferring jupyter notebook into `PyCUDA` (some kernels missing)
  - todo: call to SixTrackLib to track through chunks of the optics lattice before returning to a PyHEADTAIL multi-particle interaction module
- optimisation of performance and architecture support (*single* implementation for both multi-core CPU + GPU)

What Did we Do?

Steps to unite codes:

1. take memory pointer (and array length) and construct PyCUDA memory to communicate between PyHEADTAIL and SixTrackLib based on the SixTrackLib memory structure
   - ⟶ did not manage to do the same `CuPy`
   - ⟶ on the way to implement PyHEADTAIL notebook in `PyCUDA` (so far exists only in CuPy)
2. initiate SixTrackLib trackjobs with only parts of optics lattice (changed track function of SixTrackLib) ⇝ still need to provide python functions for this

Open questions for optimisation focus on the way:

- improvement of embedding strategy in high-level language (Python): PyCUDA vs. CuPy vs. arrayfire vs. numba
- code redundancy vs. multi-hardware support (multi-core CPU, GPU)
- code structuring (kernel size)
- low-level optimisation: register pressure etc.

PyHEADTAIL's Context Management

usual script code:

```
bunch = (...)
one_turn_map = (...)


for turn in range(n_turns):
    for m in one_turn_map:
        m.track(bunch)
```

PyHEADTAIL's Context Management

extended script code:

```
import pycuda.autoinit
from PyHEADTAIL.general.contextmanager import GPU

bunch = (...)
one_turn_map = (...)

with GPU(bunch):
    for turn in range(n_turns):
        for m in one_turn_map:
            m.track(bunch)
```

$\longrightarrow$ wrap "`with GPU(bunch) as cmg:`" around simulation code

$\Longrightarrow$ PyHEADTAIL takes care of managing CPU RAM and GPU RAM