Minutes of the ABP Computing Working Group meeting

16 Aug 2018

Participants: H. Bartosik, R. De Maria, G. Iadarola, K. Li, M. Madhyastha, L. Mether, E. Metral, N. Mounet, K. Paraschou, H. Rafique, G. Rumolo, M. Schwinzerl.

An introduction to SixTrackLib

R. De Maria presented an introduction to the new SixTrackLib tracking library.

- The library is presently under development. A stable and benchmarked version is expected in October.
- The main design features of the library are:
  - SixTrackLib uses a set of particle coordinates carefully chosen to contain canonical variables in SI units and extra dependent quantities needed to stabilize numerical errors and avoid costly recomputations for typical synchrotrons.
  - SixTrackLib implements a set of symplectic maps that can be composed to obtain symplectic integrators for typical beam line elements.
  - The tracking loop uses a flat list of the symplectic maps and does not have a notion of the original beam line elements. The tracking can be done with single pass on a subset of elements or for many turns.
  - Particles are modified in-place and extra memory is used to extract particles state element-by-element, turn-by-turn or in specific locations if needed.
- Concerning the code structure, the library contains:
  - An header only part containing the physics, usable from C-like languages (C, C++, cuda, opencl) without any assumption on memory layout and data structure. This part can be adapted to concrete framework providing a small set of C macros.
  - A concrete optimized and portable framework, using the part above, for single particle simulations based on vectorized C, OpenCL, Cuda (and maybe others in the future). This framework can be used standalone in C code and will be used inside SixTrack to offload tracking to GPUs.
  - Python library (pysixtracklib, under development) to build memory structure for sixtracklib and control simulations.
- The library lives in an ecosystem including other tools:
  - **pysixtrack** is a reference implementation of the sixtracklib physics in pure Python which duplicates the physics. The implementation is made compatible with any numeric type available in Python.
  - **sixtracktools** is used to read SixTrack input and output files. It can generate a list of maps for sixtracklib and pysixtrack, reproducing the physics models.
  - **pyoptics** is a large collection of utilities to work with MadX data. It can also read MadX sequence files and export a list of maps for sixtracklib and pysixtrack, reproducing the physics models. Only the subset of MadX maps used by SixTrack will be supported.
- Two work-flows are possible to setup a tracking simulation:

  - From the MAD-X description, use ”makethin” to obtain the thin model, export to sixtrack and use sixtracktools to generate a lattice for sixtracklib or pysixtrack.
  - From the MAD-X description, use ”makethin” to obtain the thin model, and use pyoptics to prepare lattice for sixtracklib or pysixtrack.

OpenLab project on GPGPU-accelerated Beam Dynamics

M. Madhyastha presented her development work carried out in the OpenLab framework.

- A first part of the work consisted in porting the wakefield calculation on GPU.
  - Profiling tools (cprofile and nvprof) were used to identify and attack performance bottlenecks.
  - A case-study was used to quantify the impact on performance, consisting in a simulation for the CERN PS including linear betatron tracking, nonlinear longitudinal tracking and wakefields (provided using a wake table). The simulation is performed with 10^6 macroparticles, 10^4 slices for 10^4 turns.
  - For the considered case a reduction of the computing time of about 25% is observed when moving the wakefield calculation to the GPU.
- A second part of the work consisted in interfacing PyHEADTAIL with SixTrackLib to perform instability simulations combined with advanced non-linear tracking on GPU. Two cases needed to be tackled, according to where the heat load data is stored:
  - **PyHEADTAIL particle data stored on the CPU:** in this case a buffer needs to be created on the GPU for the particle data, tracking is performed and the code is transferred back to the CPU memory. The code is explained step-by-step in Megnana’s slides.
  - **PyHEADTAIL particle data stored on the GPU:** in this case the particle buffer on the gpu is copied form from pyheadtail to sixtracklib buffer. The tracking is performed and the particle buffer is copied from sixtracklib buffer on gpu to pyheadtail buffer on the GPU. Copy kernels have been written and they are reported in Meghana’s slides.