PyHEADTAIL Space Charge Module

... an Overview!

Adrian Oeftiger

CERN – PyHEADTAIL Meeting #27

12 June 2019

Space Charge Models

●: on GPU (based on PyPIClib's solver) ●: on CPU ●: on CPU (optimisable)

- ●● **frozen 3D fieldmap** with interpolation (e.g. from beam blueprint)
  - ⟶ supports adaptive mode (auto-updating from beam distribution)
- ●● **frozen transverse fieldmap** with longitudinal self-consistent line charge density
  - ⟶ pre-computed Bassetti-Erskine interpolated on large grid → faster than exact version (equivalent to pre-computed table in SixTrack)
- ●● **exact Bassetti-Erskine** with slice-based actual rms sizes
- ● **self-consistent 3D PIC**
- ● **self-consistent 2.5D PIC**, recentring of transverse grids along slices
  - ⟹ also implemented adaptive aperture fixed to recentring grids!
- ● **self-consistent 2D PIC**

To be implemented next:

⟶ frozen **exact Bassetti-Erskine** (fixed r.m.s. sizes and line density)

PyPIClib

The particle-in-cell part of the `PyHEADTAIL` space charge module depends on `PyPIClib`[1]:

- mainly implemented on GPU:
  - free space (open boundary) FFT with integrated Green's function

  ⟶ 2D, 3D

  ⟶ 2.5D: parallel transverse 2D grids along longitudinal plane (slices)
  - linear algebra sparse matrix solver with Dirichlet boundary condition

- `PyHEADTAIL.spacecharge.pypic_spacecharge.SpaceChargePIC` calls:

  ⟶ `pypic.poissonsolver.is_25D` (if `True`, also `pypic.mesh.dz`)

  ⟶ `pypic.mesh.dimension` (assert 3D)

  ⟶ `pypic.pic_solve`

[1] integrated under `GPU` directory in `PyPIC`, `https://github.com/PyCOMPLETE/PyPIC/`
stand-alone: `https://github.com/aoeftiger/PyPIClib/`

Performance of 3D PIC

Timing of a single 3D open boundary FFT-based PIC kick for

- 1×10^6 macro-particles
- mesh of $256 \times 256 \times 100$

Table: Full Timing for Space Charge Node

hardware	**cores**	**time [ms]**
NVIDIA GPU Tesla P100	3584	53
NVIDIA GPU Tesla C2075	448	694
CPU Intel Xeon E5	1	1349

CPU-based kick implemented in a test bed:

`https://github.com/aoeftiger/PIC_testground/blob/master/PIConCPU.ipynb` ↗

$\Longrightarrow$ requires some whetting (`p2m` has a numerical bug somewhere), close to being ready!

GPU Performance of 3D PIC

Relative timing on NVIDIA P100 GPU based on `line-profiler`:

1. particle-to-mesh interpolation: 1.7%
2. Poisson solve: 90.9%
3. electric field gradient: 3.7%
4. mesh-to-particle interpolation: 3.3%

(total: 99.6%, rest: 0.4%)

$\Longrightarrow$ Poisson solve takes most time, inside the `poisson_solve` python function most time is spent inside `cufft` (GPU FFT) library: 98.6%

Conclusions

$\longrightarrow$ python overhead in SC kick is negligible!

$\longrightarrow$ recent high-end GPUs (since NVIDIA Maxwell architecture) profit from double precision acceleration of `atomic` operations (meshing)

$\longrightarrow$ GPU is the natural parallelisation approach!

Thank you for your attention!