GPGPU Accelerated Beam Dynamics
Bridging PyHEADTAIL with SixTrackLib

Meghana Madhyastha
Supervised by: Adrian Oeftiger and Haroon Rafique

August 16, 2018

Outline

Agenda

- Speeding up wakefields
- Interfacing pyheadtail with sixtracklib

Wakefields

- Port wakefields to the GPU
- Machine configurations
  - Machine: Proton Synchrotron (PS)
  - Machine model: linear betatron tracking + nonlinear longitudinal tracking + wakefield
  - Number of Macro-particles $= 10^6$
  - Number of Slices $= 10^4$
  - Number of Segments $= 1$
  - Number of Turns $= 10^4$

Simulation Cases: Profiling and Analyses

- Run different simulation cases
- profiling tools: cProfile and nvprof
- find bottlenecks
- port wakefields to GPU

Simulation Cases

1. Run master branch (recent as of Jan 26)master branch as is
2. Simulation Without Wakefield
3. Wakefield without convolution
4. Wakefield on GPU I: FFT and Inverse FFT
5. Wakefield on GPU II: Regular Convolution on the GPU + one time slicing

cProfile Results

- Running the master branch

ncalls(sec)	cumulative time(sec)	filename:lineno(function)
10000	99.203	wakes.py:119(track)
2070033	25.309	gpuarray.py:162(__init__)
20000	6.518	gpu_wrap.py:780(convolve)

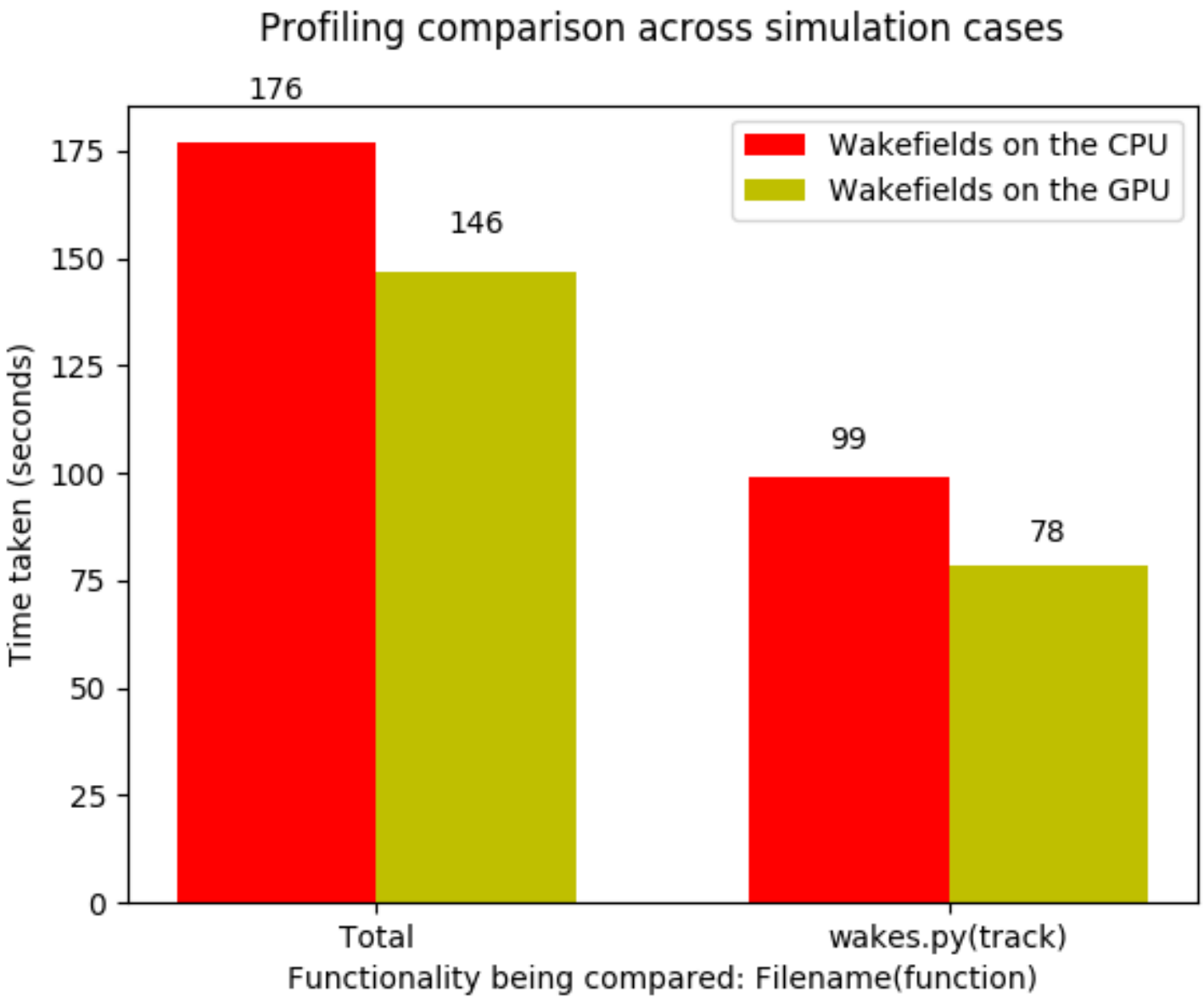
Table: cProfile Log

- Wakefield on GPU II: Regular Convolution on the GPU + one time slicing

ncalls(sec)	cumulative time(sec)	filename:lineno(function)
10000	78.637	wakes.py:119(track)
2070033	20.504	gpuarray.py:162(__init__)
20000	1.150	gpu_wrap.py:780(convolve)

Table: cProfile Log

Comparison at a glance



Bridging sixtracklib with pyheadtail

- Recap: Speed up wakefields by porting to the GPU
- Next: Non-linear longitudinal tracking (sixTrackLib) + collective effects (PyHEADTAIL)

Overall framework

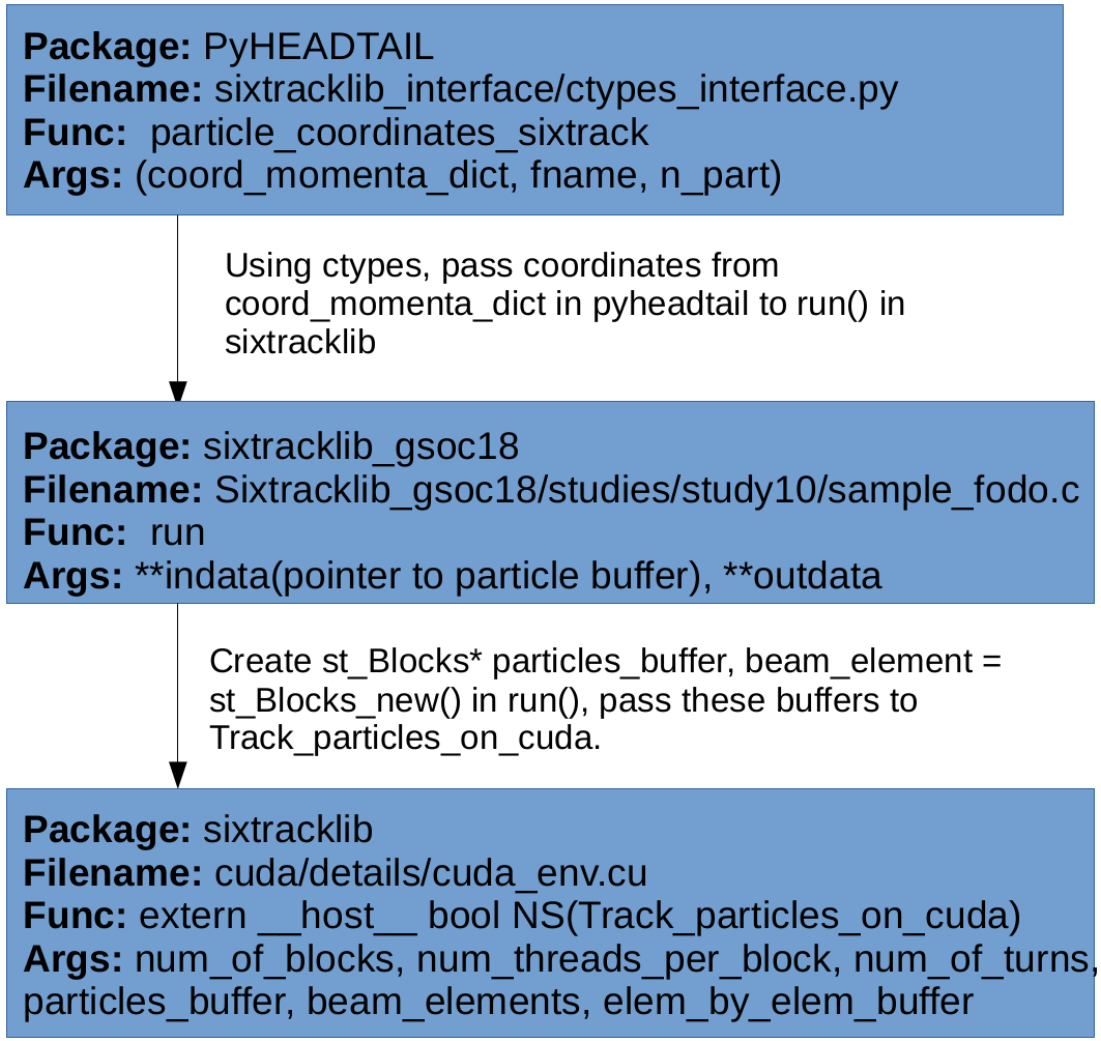


Figure: PyHEADTAIL-SixTrackLib bridge

Cases to consider

- **CASE 1: PyHEADTAIL particle on the CPU**
  - Steps on CPU
  - Steps on GPU
- **CASE 2: PyHEADTAIL particle on the GPU**
  - Steps on CPU
  - Steps on GPU

PyHEADTAIL Particles on the CPU

- steps on CPU (C interface)
- steps on GPU (in SixTrackLib)

PyHEADTAIL Particle on the CPU
STEPS on CPU

- Steps 1-4 occur in rdemaria/sixtracklib_gsoc18/tree/master/studies/study10

1. Create st_Blocks* particles_buffer
2. st_Particles* particles = st_Blocks_add_particles(Set particles$\rightarrow x$, particles$\rightarrow px$,etc)
3. call st_Blocks_serialize.
4. Call "st_Track_particles_on_cuda" passing particles_buffer which is on the CPU.

```
1 bool __host__ NS(Track_particles_on_cuda)(
2     int const num_of_blocks,
3     int const num_threads_per_block,
4     SIXTRL_UINT64_T const num_of_turns, NS(Blocks)* SIXTRL_RESTRICT
        particles_buffer,
5     NS(Blocks)* SIXTRL_RESTRICT beam_elements,
6     NS(Blocks)* SIXTRL_RESTRICT elem_by_elem_buffer )
```

PyHEADTAIL Particle on the CPU STEPS on GPU

- These steps occur in SixTrackLib

1. Transfer contents from structured particles_buffer to the host buffer.

```
host_particles_data_buffer = NS(Blocks_get_data_begin)(  particles_buffer
    )
```

2. The cuda buffer is created on the GPU and the contents of the host buffer are transferred to this cuda buffer.

```
cudaMalloc( ( void** )&cuda_particles_data_buffer, particles_buffer_size )
    ;
cudaMemcpy( cuda_particles_data_buffer, host_particles_data_buffer,
    particles_buffer_size, cudaMemcpyHostToDevice );
```

3. Memory is remapped to make a structure out of the device buffer.

```
Track_remap_serialized_blocks_buffer<<< num_of_blocks,
    num_threads_per_block >>>
```

PyHEADTAIL Particle on the CPU STEPS on GPU(continued)

4. The drift is computed.

```
Track_particles_kernel_cuda<<< num_of_blocks, num_threads_per_block >>>(
    num_of_turns, cuda_particles_data_buffer,
    cuda_beam_elements_data_buffer, cuda_elem_by_elem_data_buffer,
    cuda_success_flag );
```

5. The buffer contents now on the cuda buffer are copied back into the host buffer.

```
cudaMemcpy( host_particles_data_buffer, cuda_particles_data_buffer,
    particles_buffer_size, cudaMemcpyDeviceToHost );
```

6. The contents of this buffer are unserialized on the GPU.

```
NS(Blocks_unserialize)(particles_buffer, host_particles_data_buffer )
```

PyHEADTAIL Particles on the GPU

- steps on CPU (C interface)
- steps on GPU (in SixTrackLib)

PyHEADTAIL Particle on the GPU
STEPS on CPU

1. Create st_Blocks* particles_buffer
2. st_Particles* particles = st_Blocks_add_particles(Set particles→ x , particles→ px,etc)
3. Call st_Blocks_serialize.
4. Call "st_Track_particles_on_cuda" passing particles_buffer which is on the CPU. We note line 7 is the extra parameter (which is different from the CPU particle case)

```
1 bool __host__ NS(Track_particles_on_cuda)(
2     int const num_of_blocks,
3     int const num_threads_per_block,
4     SIXTRL_UINT64_T const num_of_turns,     NS(Blocks)* SIXTRL_RESTRICT
        particles_buffer,
5     NS(Blocks)* SIXTRL_RESTRICT beam_elements,
6     NS(Blocks)* SIXTRL_RESTRICT elem_by_elem_buffer ),
7     double* pyheadtail_ptr[]
```

PyHEADTAIL Particle on the GPU
STEPS on GPU

1. Transfer contents from structured particles_buffer to the host buffer.

```
host_particles_data_buffer = NS(Blocks_get_data_begin)(  particles_buffer
    )
```

2. The cuda buffer is created on the GPU and the contents of the host buffer are transferred to this cuda buffer.

```
cudaMalloc( ( void** )&cuda_particles_data_buffer, particles_buffer_size )
    ;
cudaMemcpy( cuda_particles_data_buffer, host_particles_data_buffer,
    particles_buffer_size, cudaMemcpyHostToDevice );
```

3. Memory is remapped to make a structure out of the device buffer.

```
Track_remap_serialized_blocks_buffer<<< num_of_blocks,
    num_threads_per_block >>>
```

PyHEADTAIL Particle on the CPU
STEPS on GPU(continued)

4. The particle buffer on the gpu is copied form from pyheadtail to sixtracklib buffer

```
1 Copy_buffer_pyheadtail_sixtracklib<<<num_of_blocks, num_threads_per_block
      >>>(cuda_particles_data_buffer, pyheadtail_ptr, cuda_success_flag )
```

5. The drift is computed.

```
1 Track_particles_kernel_cuda<<< num_of_blocks, num_threads_per_block >>>(
      num_of_turns, cuda_particles_data_buffer,
      cuda_beam_elements_data_buffer, cuda_elem_by_elem_data_buffer,
      cuda_success_flag );
```

6. The particle buffer is copied from sixtracklib buffer on gpu to pyheadtail buffer on gpu

```
1 Copy_buffer_sixtracklib_pyheadtail<<<num_of_blocks, num_threads_per_block
      >>>(cuda_particles_data_buffer, pyheadtail_ptr, cuda_success_flag )
```

Copy Kernels

- Copy_buffer_pyheadtail_sixtracklib

```
1  __global__ void Copy_buffer_pyheadtail_sixtracklib(
2      unsigned char* __restrict__ particles_data_buffer,
3      double*  __restrict__ py_particles_buffer[],
4      int64_t*       __restrict__ ptr_success_flag
5  ){
6          NS(Blocks) particles_buffer;
7          NS(Blocks_preset)( &particles_buffer );
8          NS(Blocks_unserialize_without_remapping)( &particles_buffer,
       particles_data_buffer );
9
10         NS(BlockInfo)* ptr_info  = NS(Blocks_get_block_infos_begin)( &
       particles_buffer );
11         NS(Particles)* particles = NS(Blocks_get_particles)( ptr_info );
12         size_t num_of_particles  = NS(Particles_get_num_particles)(
       particles );
13
14         memcpy( NS(Particles_get_x)( particles ),
15               py_particles_buffer[0],
16               num_of_particles * sizeof( double )
17               );
18         memcpy( NS(Particles_get_px)( particles ),
19               py_particles_buffer[1],
20               num_of_particles * sizeof( double ));
21 }
```

Copy Kernels(continued)

- Copy_buffer_sixtracklib_pyheadtail

```
1  __global__ void Copy_buffer_sixtracklib_pyheadtail(
2      unsigned char* __restrict__ particles_data_buffer,
3      double*  __restrict__ py_particles_buffer[],
4      int64_t*       __restrict__ ptr_success_flag
5  ){
6          NS(Blocks) particles_buffer;
7          NS(Blocks_preset)( &particles_buffer );
8          NS(Blocks_unserialize_without_remapping)( &particles_buffer,
       particles_data_buffer );
9
10         NS(BlockInfo)* ptr_info  = NS(Blocks_get_block_infos_begin)( &
       particles_buffer );
11         NS(Particles)* particles = NS(Blocks_get_particles)( ptr_info );
12         size_t num_of_particles  = NS(Particles_get_num_particles)(
       particles );
13         memcpy( py_particles_buffer[0], NS(Particles_get_const_x)(
       particles ),
14                 num_of_particles * sizeof( double )
15                 );
16         memcpy( py_particles_buffer[1], NS(Particles_get_const_px)(
       particles ),
17                 num_of_particles * sizeof( double )
```

PyHEADTAIL particle structure

- instead of passing a pointer to a double* array containing particle information from pyheadtail, create a "pyheadtail_particle" structure and pass a pointer to it.
- sixtracklib/common/pyheadtail_particles.h

```
1            struct ParticleData;
2            typedef struct ParticleData{
3                int npart;
4                double *x;
5                double *xp;
6                double *y;
7                double *yp;
8            } ParticleData;
9
```

- The python ctypes equivalent (to be passed to sixtracklib)

```
1    class ParticleDataCtypes(ctypes.Structure):
2    _fields_ = [("npart", c_int),
3                ("x", POINTER(c_double)),
4                ("xp", POINTER(c_double)),
5                ("y", POINTER(c_double)),
6                ("yp", POINTER(c_double))
7                ]
8
9
```

Conclusion

- Port Wakefields to GPU; speedup of 25%.
- sixTrackLib's nonlinear tracking from PyHEADTAIL