SixTrackLib: Current Status, Future Directions
ABP-CWG Meeting

Martin Schwinzerl, Riccardo De Maria
CERN, Beam Department, ABP
September 19th, 2019

With special thanks to: Adrian Oeftiger(GSI), Giovanni Iadarola, Konstantinos Paraschou, Hannes Bartosik, Julie Lise Aline Malewicz(Summer Student)
Outline

• Introduction & Current Status
• Basic Usage Examples
• Design & Implementation
• Performance
• Usage Scenarios & Examples
• Next Steps
Introduction: A Single Particle Tracking (Parallel) Library

1. Single-Particle: non-interaction particles $p_i, p_k$ with $i \neq k < N_p$

2. Tracking: via symplectic (thin-lens) map $f_j$ for the beam-element at position $j$ in the lattice: $p_i(j + 1) \leftarrow f_j(p_i(j))$

3. Parallel: For $N_p \gg 1$: "Embarrassingly" parallel problem

4. Library: independent of application, low barrier of entry, reusable, embedable, extensible

Use-Cases:

- Building block for applications that require tracking (e.g. PYHEADTAIL)
- Optimal usage (CPU and GPU) of donated computing time via LHC@Home
- Large scale simulations on HPC resources
- Support for additional types: SIMD, Multi-Precision, TPSA, etc. types
SixTrackLib: Current Status

- In development for 18 months+ now
- Supports C99, C++11, and Python 3
- Supports Single threaded CPU (Auto-Vec), OpenCL 1.2 and Cuda
- Pending support for CPU side Multi-Precision\(^1\) and SIMD\(^2\) (C++)
- Available from https://github.com/SixTrack/sixtracklib
- Part of a larger ecosystem of libraries especially targetting Python:
  - cobjects: https://github.com/SixTrack/cobjects
  - pysixtrack: https://github.com/SixTrack/pysixtrack
  - sixtracktools: https://github.com/SixTrack/sixtracktools

- **Usable for Advanced Users** - User API not stable yet

---

\(^1\)https://www.boost.org/doc/libs/1_71_0/libs/multiprecision/doc/html/index.html
\(^2\)https://github.com/QuantStack/xsimd
import sixtracklib as st

pset = st.ParticlesSet().fromfile("./particles.bin")
lattice = st.Elements().fromfile("./lattice.bin")

# Per default, the TrackJob job will use the CPU
job = st.TrackJob(lattice, pset)

# Track all particles until turn 100
job.track_until(100)

# Collect the particle state -> not requ. @ CPU
job.collect_particles()

# pset contains now the updated particles
# ...
Basic Usage Example: C++

```cpp
#include "sixtracklib/sixtracklib.hpp"

int main()
{
    namespace st = sixtrack;
    st::Buffer particle_set( "./particles.bin" );
    st::Buffer lattice( "./lattice.bin" );

    // We have to be explicit about using the CPU for tracking
    st::TrackJobCpu job( particle_set, lattice );

    // Track until turn 100
    job.trackUntil( 100 );

    // Collect the particle state -> would not be needed for the CPU TrackJob
    job.collectParticles();

    // particle_set now contains the tracked data
    // ... Do something with the particles ...

    return 0;
}
```
Basic Usage Example: C99

```c
#include "sixtracklib/sixtracklib.h"

int main()
{
    NS(Buffer)* particle_set = NS(Buffer_new_from_file)( "./particles.bin" );
    NS(Buffer)* lattice = NS(Buffer_new_from_file)( "./lattice.bin" );

    /* We have to be explicit about using the CPU for tracking */
    NS(TrackJobCpu)* job = NS(TrackJobCpu_new)( particle_set, lattice );

    /* Track until turn 100 */
    NS(TrackJob_track_until)( job, 100 );

    /* Collect the particle state ->
     * would not be needed for the CPU TrackJob */
    NS(TrackJob_collect_particles)( job );

    /* particle_set now contains the tracked data */
    /* ... Do something with the particles ... */

    /* Cleaning-Up */
    NS(TrackJob_delete)( job );
    NS(Buffer_delete)( particle_set );
    NS(Buffer_delete)( lattice );

    return 0;
}
```
Slightly More Advanced Examples: Using the GPU

How to use the GPU? Create the TrackJob instance differently:

```python
# Python; "opencl:0.1" -> platform 0, device
job = pystlib.TrackJob(elements, particles, device="opencl:0.1")
```

```cpp
// C++, 0.1 -> platform 0, device 1
st::TrackJobCl job("0.1", particle_set, lattice);
```

```c99
/* C99; "0.1" = platform 0, device 1 */
NS(TrackJobCl)* job = NS(TrackJobCl_new)("0.1", particle_set, lattice);
```
Slightly More Advanced Examples: Working with a MAD-X Sequence

```python
# from examples/python/test_sis18/test_timings.py

from cpymad.madx import Madx
import sixtracklib as st
import pysixtrack

import time
from scipy.constants import e, m_p, c

import numpy as np

p0c = 6 * 1e9  # in eV
Etot = np.sqrt(p0c**2 + (m_p / e)**2 * c**4) * 1e-9  # in GeV

mad = Madx()
mad.call(file="fodo.madx")
mad.command.beam(particle='proton', energy=str(Etot))
mad.use(sequence="FODO")

# ....

sis18 = mad.sequence.FODO

nturns = 1
ps_line, _ = pysixtrack.Line.from_madx_sequence(sis18)
elements = st.Elements()
elements.append_line(ps_line)

npart = 1000
particles = st.Particles.from_ref(npart, p0c=p0c)
particles.x += np.linspace(0, 1e-6, npart)

job = pystlib.TrackJob(elements, particles, device="opencl:0.0")
job.track_until(1000)
job.collect_particles()
# ....
```
The Physics

The Boring Parts: Business Logic & Architecture Specific Code
Particles and Particle - Sets

We currently keep 21 attributes per particle:

- Six degrees of freedom for tracking: \( x, p_x, y, p_y, \zeta, \delta \)
- Four status variables to keep track of the particle (\texttt{particle\_id, at\_element\_id, at\_turn, state})
- 11 attributes that are not strictly needed:
  - Five attributes to describe the reference particle \( q_0, m_0, \beta_0, \gamma_0, (P_0 \cdot c) \)
  - Four attributes to describe properties of the particle relative to the reference particle \( r_{pp} = P_0 / P, r_{vv} = \beta / \beta_0, \text{charge\_ratio} = (q / q_0), \chi = (q / q_0) / (m_0 / m) \)
  - \( s \): linear distance from start
  - \( p_{\sigma} \): conjugate to \( \sigma \)
- For double and int64_t, we need 168 Bytes / particle

**Note:** All state encoded in particles - beam-elements could be read-only\(^3\)!
Supported BeamElements

In General: Similar to SixTrack

- Drift, DriftExact
- Multipole: including Dipoles, Quadrupoles, Sextupoles, etc.
- Cavity
- XYShift: transversal shift
- SRotation: rotation in the transversal plane
- BeamMonitor: programmable dump of particle state
- BeamBeam4D, BeamBeam6D
- SpaceChargeBunched, SpaceChargeCoasting\(^4\)
- DipoleEdge\(^4\)
- LimitRect, LimitEllipse: aperture checks\(^4\)
- RFMultipole\(^5\)

\(^4\) available, not fully integrated
\(^5\) Coming Soon\(^{TM}\)
Example Tracking Map (C99)

```c
SIXTRL_INLINE NS(track_status_t) NS(Track_particle_drift)(
    SIXTRL_PARTICLE_ARGPTR_DEC NS(Particles)* SIXTRL_RESTRICT p,
    NS(particle_num_elements_t) const ii,
    SIXTRL_BE_ARGPTR_DEC const NS(Dec) *const SIXTRL_RESTRICT drift )
{
    typedef NS(particle_real_t) real_t;

    real_t const rpp = NS(Particles_get_rpp_value)( p, ii );
    real_t const xp = NS(Particles_get_px_value)( p, ii ) * rpp;
    real_t const yp = NS(Particles_get_py_value)( p, ii ) * rpp;
    real_t const length = NS(Dec_get_length)( drift );
    real_t const dzeta = NS(Particles_get_rvv_value)( p, ii ) -
                         ( ( real_t )1 + ( xp*xp + yp*yp ) / ( real_t )2 );

    NS(Particles_add_to_x_value)( p, ii, xp * length );
    NS(Particles_add_to_y_value)( p, ii, yp * length );

    SIXTRL_ASSERT( NS(Particles_get_beta0_value)( p, ii ) > ( real_t )0 );

    NS(Particles_add_to_s_value)( p, ii, length );
    NS(Particles_add_to_zeta_value)( p, ii, length * dzeta );

    return SIXTRL_TRACK_SUCCESS;
}
```
namespace sixtrack
{

    template< class PData, class BeData >
    typename track_result_t< PData, BeData, OBJECT_TYPE_DRIFT >::type
    Drift_track(
        TrackParticleInterface< PData >& SIXTRL_RESTRICT_REF particle,
        DriftInterface< BeData > const& SIXTRL_RESTRICT_REF drift )
    {

        typedef typename TrackParticleInterface< PData >::real_t real_t;

        real_t const xp = particle.px * particle.rpp;
        real_t const yp = particle.py * particle.rpp;
        real_t const dzeta = particle.rvv -
            ( ( xp * xp + yp * yp ) / ( real_t )2.0 + ( real_t )1.0 );

        particle.x += xp * drift.length;
        particle.y += yp * drift.length;
        particle.s += drift.length;
        particle.zeta += dzeta * drift.length;

        return TRACK_SUCCESS;
    }
}
SixTrackLib: Design & Implementation

1. The Physics
2. The Boring Parts: Business Logic & Architecture Specific Code
Central Object: TrackJob

- Manages Sets of particles and beam elements
- Manages space to dump particle states during tracking (i.e. "output")
- Provides a common interface regardless of architecture
- Persists book-keeping variables and state between tracking operations
- Three modes of tracking:

  - Implemented in terms of Buffers, Controllers, Arguments
Buffers: Data Exchange and Storage

How to store (serialize) and exchange particle state, lattices, ... ?
Especially

- For different architectures (i.e. OpenCL and Cuda)
- Between ”Hosts” and ”Devices”
- In environments with segmented memory and strict alignment requirements
- Across different computing languages (incl. header-only C)
- Allowing for structured data with arrays, nested structures
- Allowing for zero-copy data retrieval
- With minimal overhead (i.e. retrieve data inside tight kernel loops)

⇒ cobjects: https://github.com/SixTrack/cobjects
Buffers: cobjects

- Python 3 implementation available (pip install cobjects)
- C/C++ implementation part of SixTrackLib
- Retrieval: Find start-address of stored object → ”cast” to proper data type
- Storage: Objects describe their own memory layout

```python
#!/usr/bin/env python
#
from cobjects import CBuffer, CObject, CField

class Multipole(CObject):
    _typeid = 4
    order = CField(0, 'int64', default=0, const=True, alignment=8)
    length = CField(1, 'real', default=0.0, alignment=8)
    hxl = CField(2, 'real', default=0.0, alignment=8)
    hyl = CField(3, 'real', default=0.0, alignment=8)
    bal = CField(4, 'real', default=0.0, alignment=8, pointer=True,
                 length='2 * order + 2')

# ....
```
Performance Benchmark

Coming Soon to a master branch near you: Scripted Benchmarking

```python
# Example configuration for a benchmark run
name = "pcbe16820_lhc_no_bb_opencl_rx560"
git_hash = "9d9a68"
git_branch = "benchmark"
output_path = "."

# Target configuration - configure the node to run the benchmark on
[target]
  arch = "opencl"
  node_id = "0.0"
  config_str = ""
  optimized = 1
  x_stddev = 0.6  # All std deviations are relative, i.e.
  y_stddev = 0.0  # sigma_x = <x> * x_stddev
  px_stddev = 0.0  # sigma_y = <y> * y_stddev, etc.
  py_stddev = 0.0  # stddev == 0.0 will be ignored and does not cause any
  zeta_stddev = 0.6  # overhead
  delta_stddev = 0.0

# Tracking configuration - configure the track items and the data sources
[track]
  path_particle_dump = "../tests/testdata/lhc_no_bb/particles_dump.bin"
  path_lattice_dump = "../tests/testdata/lhc_no_bb/beam_elements.bin"

num_particles = [1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 20000, 40000, 65536, 100000, 200000, 400000, 1000000, 2000000, 4000000, 8000000, 10000000, 20000000, 40000000,]

num_turns = [100, 100, 100, 100, 100, 50, 50, 20, 20, 20, 10, 10, 5, 5, 5, 5, 5, 5, 2, 2, 2, 2, 2, 2,]

num_repetitions = [10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 5, 5, 5, 5, 5, 5, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,]

# end: pcbe16820_lhc_no_bb_opencl_rx560.config
```
Benchmark: LHC lattice, no Beam-Beam interaction

LHC machine description without Beam-Beam elements (18657 elements)
Comparison of benchmarked results across configurations

- NVidia Titan V
- Intel OpenCL AMD CPU 1950X
- AMD FirePro W8100
- Single Thread AMD CPU 1950X

Time / particle / turn [s]

Number of particles
Benchmark: LHC lattice with Beam-Beam

LHC machine description with Beam-Beam elements (10866 elements)
Comparison of benchmarked results across configurations

Time / particle / turn [s]

NVidia Titan V
Intel OpenCL AMD CPU 1950X
AMD FirePro W8100
NVidia GTX1030
Single Thread AMD CPU 1950X

Number of particles

10^0  10^1  10^2  10^3  10^4  10^5  10^6
Selected Usage Examples

1. Julie Lise Aline Malewicz: "Simulation of Beam Losses at the Large Hadron Colider"
   Beam-Beam and Luminosity Studies Meeting, BE-ABP, CERN, 16. September 2019
   https://indico.cern.ch/event/847766/

2. Adrian Oeftiger: "SixTrackLib + PyHEADTAIL"
   35th ABP-CWG Meeting, 25. Juli 2019
   https://indico.cern.ch/event/836341/
Motivations

- Intensity loss observed in the LHC cannot be fully explained by standard luminosity decay
  - Instead, we realize that non-linear effects, such as beam-beam interactions, cannot be neglected and play a major part

- With a new tracking software such as Sixtracklib, the purpose will be to find how can we most effectively model, compute, visualize and finally draw conclusions about the observed luminosity losses in the LHC?

- **Goal:** Simulate 30 minutes of beam (or 20 million turns), and compare results to actual observed losses of about 0.1%
Tools

- **GPUs**
  - (4) in CNAF Bologna (shared among 3 people)
  - (12) in HTCondor (that other people are also in competition for)
  - 10,000 particles need 4.5 days on 1 GPU for 20,000,000 turns in the full LHC lattice

- **Sixtracklib**
  - simple Python interface
  - supports GPU parallelization

- To use the barebones Sixtracklib software, we still had to reinvent some wheels, such as normalizing the fully linearly coupled motion.
- We also had to work out how to define the beam distribution in the longitudinal plane that accounts for non-linear synchrotron motion.
Total losses

We expected a rapid decrease of the loss, yet instead, we found an increase. Nevertheless, both plots seem to converge to very similar numbers.
Conclusions

- Defined mathematical procedure to evaluate beam losses from tracking simulations
- Made use of GPUs with Sixtracklib
- Managed to simulate some losses for 30 min time of LHC beam with the full lattice and beam-beam effects
- Learned about Monte Carlo integration, variance-reduction, and how to use them at our advantage to make computation of losses more efficient

Suggestions for further improvement:

- Using stratified sampling on the calculation of losses did not show any improvement, but it can be used in combination with importance sampling
- Study how effective extending the bounds of the initial distribution would be (ex: up to $5\sigma$ instead of 4)
- Simulate a longer timescale, to see if losses decrease at some point
Selected Usage Examples

1. Julie Lise Aline Malewicz: "Simulation of Beam Losses at the Large Hadron Colider"
   Beam-Beam and Luminosity Studies Meeting, BE-ABP, CERN, 16. September 2019
   https://indico.cern.ch/event/847766/

2. Adrian Oeftiger: "SixTrackLib + PyHEADTAIL"
   35th ABP-CWG Meeting, 25. Juli 2019
   https://indico.cern.ch/event/836341/
Overview

Timeline of SixTrackLib + PyHEADTAIL:

- Summer 2018: Summer student Meghana Madhyastha implemented with Martin Schwinzerl first script calling both SixTrackLib and PyHEADTAIL and passing coordinates (GPU-GPU via CPU)
- Jan 2019: Kick-off meeting CERN openlab E4/NVIDIA project
- Apr 2019: NVIDIA hackathon, first full GPU prototype of STL+PyHT
- Jun 2019: Non-linear resonance studies in SIS-100 without space charge (factor 2000 faster than MAD-X thin-lens tracking for $1 \times 10^6$ macro-particles)
- Jul 2019: Space charge driven resonant halo dynamics in FODO cell with self-consistent 3D PIC from PyHEADTAIL

In my SixTrackLib + PyHEADTAIL playground repository you find all of these joint code simulations and examples
**MAD-X vs. SixTrackLib Tracking**

**Figure:** Benchmark element-by-element MAD-X vs. STL: \( \epsilon = \mathcal{O}(10^{-10}) \)

(a) on-coupling \( Q_x = Q_y = 18.88 \)

(b) off-coupling \( Q_x = 18.84, \ Q_y = 18.73 \)

(c) different integers \( Q_x = 19.84, \ Q_y = 17.73 \)
**SixTrackLib + PyHEADTAIL (Oeftiger)**

**MAD-X vs. SixTrackLib Tracking**

**Figure:** Benchmark 10000 turns MAD-X vs. STL: transverse $\epsilon_{x,y} = O(10^{-7})$ and longitudinal $\epsilon_z = O(10^{-9})$

(a) horizontal plane  
(b) vertical plane  
(c) longitudinal plane

$\longrightarrow$ single-particle physics in MAD-X and SixTrackLib are equivalent (up to numerical errors)
SixTrackLib + PyHEADTAIL brings a lot of flexibility – ingredients of the FODO study comprised:

- removing the non-linear RF cavity from SixTrackLib lattice and replace it by linear map from PyHEADTAIL
  - other dynamic elements (feedback systems, noise, ripple etc) easily injectable

- using PyHEADTAIL's statistics monitors on the bunch distribution, as well as extending the monitoring of data to emittance quantiles
  - running standard case (256 × 256 × 64 with 1 × 10^6 macro-particles) on GPU takes ≈ 20 min
**Conclusion & Outlook 1**

- SixTrackLib is complete enough to allow early studies
- → Continue to use these studies as test-cases to improve and further complete SixTrackLib (i.e. additional beam-elements, access patterns, usage scenarios, capabilities)
- → Evaluate the design decisions and implementations based on these real-world use cases
- **Main Challenge:** Number of Lines of Code (> 100k LOC in current master) → huge potential for code-generation
  - Physics Code: auto-generate from a description (pysixtrack)
  - Business Logic & Architecture specific code: introspection with libtooling / libclang, auto generation of C and Python bindings, tests, etc.
Outlook II

- Use the C++ template interface to SixTrackLib to
  - perform studies to improve the numerical stability and reproducibility (boost::multiprecision)
  - provide a SIMD option for CPU targets to improve over the current situation (i.e. relying on auto-vectorization)
  - look into allowing a TPSA type for tracking
- Use scripted benchmarking suite to study optimizations for different architectures
  - Reduce register pressure for heavy kernels (Beam-Beam, Space-Charge)
  - Study impact of different memory access patterns
  - Use code-generation to optimize tracking maps (i.e. branch-less versions for SIMD and GPU targets)
- Make SixTrackLib available as a tracking back-end to SixTrack
Thank you for your Attention!