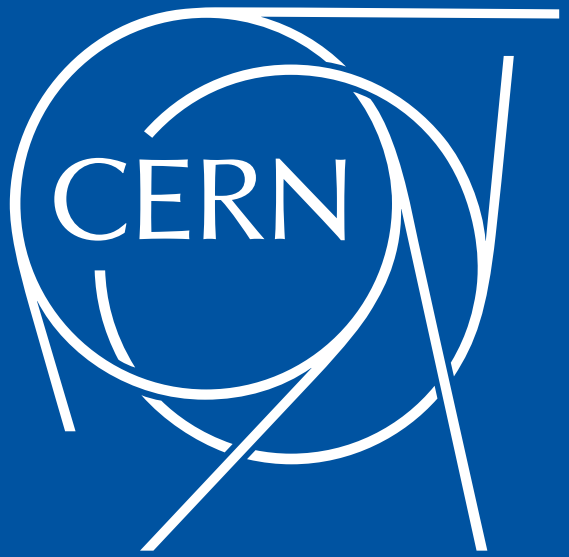
CERN

Introducing SixTrackLib

A Versatile, Hardware-Accelerated Single-Particle Tracking Library

M. Schwinzerl[1], Riccardo De Maria[1], Giovanni Iadarola[1], Adrian Oeftiger[2]

[1]CERN, BE Department, ABP-HSS, [2]GSI/FAIR

PyHEP 2019 :: Abingdon, Uk

Introduction

Usage Examples

Design Principles & Implementation

Performance Analysis

Integration Into Python Programs

Conclusion & Outlook

Introduction :: A $6D$ Single Particle Tracking (Parallel) Library

1. $6D$: Particle motion through phase space $(x; p_x; y; p_y; \ ; \ )$

Introduction :: A $6D$ Single Particle Tracking (Parallel) Library

1. $6D$: Particle motion through phase space $(x; p_x; y; p_y; \ ; \ )$

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

Introduction :: A $6D$ Single Particle Tracking (Parallel) Library

1. $6D$: Particle motion through phase space $(x; p_x; y; p_y; \ ; \ )$

2. Single-Particle: non-interaction particles $p_i; p_k$ with $i \neq k < N_p$
3. Tracking: via symplectic (thin-lens) map f_j for the beam-element at position j in the lattice: $p_i\ (j + 1) \quad f_j\ (p_i\ (j))$

Introduction :: A 6D Single Particle Tracking (Parallel) Library

1. 6D: Particle motion through phase space ($x; p_x; y; p_y; \ ; \ $)

2. Single-Particle: non-interaction particles $p_i; p_k$ with $i \neq k < N_p$
3. Tracking: via symplectic (thin-lens) map f_j for the beam-element at position j in the lattice: $p_i\ (j + 1) \quad f_j\ (p_i\ (j))$
4. Parallel: For $N_p \quad 1$: "embarrassingly" parallel problem

Introduction :: A $6D$ Single Particle Tracking (Parallel) Library

1. $6D$: Particle motion through phase space $(x; p_x; y; p_y; \ ; \ )$

2. Single-Particle: non-interaction particles $p_i; p_k$ with $i \neq k < N_p$
3. Tracking: via symplectic (thin-lens) map f_j for the beam-element at position j in the lattice: $p_i\ (j + 1) \quad f_j\ (p_i\ (j))$
4. Parallel: For $N_p \quad 1$: "embarrassingly" parallel problem
5. Library: independent of application, low barrier of entry, reusable, embedable, extensible (in contrast to established application SixTrack: `https://github.com/SixTrack/SixTrack`)

Introduction :: Usage Scenarios For Such A Library

Building-block for applications (i.e. user-generated simulations or even frameworks) that require tracking (for example PyHEADTAIL)

Introduction :: Usage Scenarios For Such A Library

Building-block for applications (i.e. user-generated simulations or even frameworks) that require tracking (for example PyHEADTAIL)

Usable on PCs and Laptops with limited or no parallel computing capabilities (development & debugging!)

Introduction :: Usage Scenarios For Such A Library

Building-block for applications (i.e. user-generated simulations or even frameworks) that require tracking (for example `PyHEADTAIL`)

Usable on PCs and Laptops with limited or no parallel computing capabilities (development & debugging!)

Large-scale simulations (i.e. many particles, many turns) on dedicated HPC infrastructure

Introduction :: Usage Scenarios For Such A Library

Building-block for applications (i.e. user-generated simulations or even frameworks) that require tracking (for example `PyHEADTAIL`)

Usable on PCs and Laptops with limited or no parallel computing capabilities (development & debugging!)

Large-scale simulations (i.e. many particles, many turns) on dedicated HPC infrastructure

Optimal usage of donated computing time (GPU and CPU) via LHC@Home volunteer project `http://lhcathome.web.cern.ch/projects/sixtrack`

Introduction :: Usage Scenarios For Such A Library

Building-block for applications (i.e. user-generated simulations or even frameworks) that require tracking (for example `PyHEADTAIL`)

Usable on PCs and Laptops with limited or no parallel computing capabilities (development & debugging!)

Large-scale simulations (i.e. many particles, many turns) on dedicated HPC infrastructure

Optimal usage of donated computing time (GPU and CPU) via LHC@Home volunteer project
`http://lhcathome.web.cern.ch/projects/sixtrack`

Across these: same API/syntax
(i.e. without having to rewrite any user-code)

Regular users should not need any GPU/HPC knowledge
(but allow advanced users to tweak things)

Introduction :: Current SixTrackLib Status

Available fromhttps://github.com/SixTrack/sixtracklib

Early stage, intended for advanced users & selected studies

In development for 18 months+ now

Introduction :: Current SixTrackLib Status

Available fromhttps://github.com/SixTrack/sixtracklib

Early stage, intended for advanced users & selected studies

In development for 18 months+ now

Supports C99, C++11, and Python 3 consistent API

Introduction :: Current SixTrackLib Status

Available fromhttps://github.com/SixTrack/sixtracklib

Early stage, intended for advanced users & selected studies

In development for 18 months+ now

Supports C99, C++11, and Python 3 consistent API

Supports Single threaded CPU (Auto-Vec), OpenCL 1.2 and Cuda

Introduction :: Current SixTrackLib Status

Available fromhttps://github.com/SixTrack/sixtracklib

Early stage, intended for advanced users & selected studies

In development for 18 months+ now

Supports C99, C++11, and Python 3 consistent API

Supports Single threaded CPU (Auto-Vec), OpenCL 1.2 and Cuda

Part of a larger ecosystem of libraries especially targeting Python:

Introduction :: Current SixTrackLib Status

Available fromhttps://github.com/SixTrack/sixtracklib

Early stage, intended for advanced users & selected studies

In development for 18 months+ now

Supports C99, C++11, and Python 3 consistent API

Supports Single threaded CPU (Auto-Vec), OpenCL 1.2 and Cuda

Part of a larger ecosystem of libraries especially targeting Python:

cobjects: Specialised binary serialisation bu er/protocol
https://github.com/SixTrack/cobjects

Introduction :: Current SixTrackLib Status

Available fromhttps://github.com/SixTrack/sixtracklib

Early stage, intended for advanced users & selected studies

In development for 18 months+ now

Supports C99, C++11, and Python 3 consistent API

Supports Single threaded CPU (Auto-Vec), OpenCL 1.2 and Cuda

Part of a larger ecosystem of libraries especially targeting Python:

cobjects: Specialised binary serialisation bu er/protocol
https://github.com/SixTrack/cobjects
pysixtrack: Rapid prototyping Python-only implementation
https://github.com/SixTrack/pysixtrack

Introduction :: Current SixTrackLib Status

Available fromhttps://github.com/SixTrack/sixtracklib

Early stage, intended for advanced users & selected studies

In development for 18 months+ now

Supports C99, C++11, and Python 3 consistent API

Supports Single threaded CPU (Auto-Vec), OpenCL 1.2 and Cuda

Part of a larger ecosystem of libraries especially targeting Python:

cobjects: Specialised binary serialisation bu er/protocol
https://github.com/SixTrack/cobjects
pysixtrack: Rapid prototyping Python-only implementation
https://github.com/SixTrack/pysixtrack
sixtracktools: Access SixTrack IO les from Python
https://github.com/SixTrack/sixtracktools

Examples :: Track All Particles Until Turn On CPU

Examples :: Track All Particles Until Turn On GPU (OpenCL)

Examples :: Lattice from MAD-X Sequence (via pysixtrack)

Design Principles :: Overview

Design Goal: separate the technical details (CPU, OpenCL, Cuda, parallel computing, HPC, etc.) from the physics

Design Principles :: Overview

Design Goal: separate the technical details (CPU, OpenCL, Cuda, parallel computing, HPC, etc.) from the physics

) Keep SixTrackLib extensible wrt. physics (even by end-users)

) Have a single implementation of the physics models (header only, C99, heavily abstracted & limited DSL)

) Keep SixTrackLib extensible wrt. supported architectures

Design Principles :: Overview

Design Goal: separate the technical details (CPU, OpenCL, Cuda, parallel computing, HPC, etc.) from the physics

) Keep SixTrackLib extensible wrt. physics (even by end-users)

) Have a single implementation of the physics models (header only, C99, heavily abstracted & limited DSL)

) Keep SixTrackLib extensible wrt. supported architectures

Design Goal: consistent API across languages & architectures

Design Principles :: Overview

Design Goal: separate the technical details (CPU, OpenCL, Cuda, parallel computing, HPC, etc.) from the physics

- Keep SixTrackLib extensible wrt. physics (even by end-users)
- Have a single implementation of the physics models (header only, C99, heavily abstracted & limited DSL)
- Keep SixTrackLib extensible wrt. supported architectures

Design Goal: consistent API across languages & architectures

Challenge: limit externally facing API surface, stability promises

- Focus SixTrackLib on tracking
- Move auxiliary components out of the library

Implementation :: External Library :: pysixtrack

Idea: minimal, pythonic (Python-only!) tracking implementation

Implementation :: External Library :: pysixtrack

Idea: minimal, pythonic (Python-only!) tracking implementation

Place for prototyping & implementing new physics models

Collect also I/O helper routines (cf. MAD-X example above)

Implementation :: External Library :: pysixtrack

Idea: minimal, pythonic (Python-only!) tracking implementation

Place for prototyping & implementing new physics models

Collect also I/O helper routines (cf. MAD-X example above)

Strong focus on numerical precision and correctness (math.sin!)

Slightly reduced focus on performance and scalability

Implementation :: External Library: pysixtrack

Idea: minimal, pythonic (Python-only!) tracking implementation

Place for prototyping & implementing new physics models

Collect also I/O helper routines (cf. MAD-X example above)

Strong focus on numerical precision and correctness (math.sin)

Slightly reduced focus on performance and scalability

Implementation :: External Library objects

Particles, beam-elements, output

! need to be serialized, stored/restored, exchanged (host <-> device)

Implementation :: External Library: cobjects

Particles, beam-elements, output

! need to be serialized, stored/restored, exchanged (host ↔ device)

cobjects : binary protocol & buffer (Python3, numpy)

C/C++ implementation available as part of SixTrackLib

Implementation :: External Library: cobjects

Particles, beam-elements, output

! need to be serialized, stored/restored, exchanged (host $\leftrightarrow$ device)

cobjects : binary protocol & buffer (Python3, numpy)

C/C++ implementation available as part of SixTrackLib

Allows objects to have nested structured members and vectors

Allows for user-contributed (structured) data-types

Implementation :: External Library: cobjects

Particles, beam-elements, output

! need to be serialized, stored/restored, exchanged (Host $\leftrightarrow$ Device)

cobjects : binary protocol & buffer (Python3, numpy)

C/C++ implementation available as part of SixTrackLib

Allows objects to have nested structured members and vectors

Allows for user-contributed (structured) data-types

Implementation :: Externally Library objects (cont.)

Allows light-weight, zero-copy, in-place access to stored objects

Implementation :: Externally Library objects (cont.)

Allows light-weight, zero-copy, in-place access to stored objects

Implementation :: Externally Library objects (cont.)

Allows light-weight, zero-copy, in-place access to stored objects

Challenge: if a buffer of objects is moved, all stored pointers have to be "remapped"

Implementation :: Externally Library objects (cont.)

Allows light-weight, zero-copy, in-place access to stored objects

Challenge: if a buffer of objects is moved, all stored pointers have to be "remapped"

- objects buffers are intended to be used "sequentially"

Implementation ::cobjects Storage Memory Layout

Performance Analysis :: Full LHC lattice, OpenCL Backend

Integrating SixTrackLib Into Programs :: Overview

SixTrackLib supports several di erent integration strategies. Ranging from "easily accessible" to "complex & invasive":

Use track_until() & collect_particles() via TrackJob

Integrating SixTrackLib Into Programs :: Overview

SixTrackLib supports several di erent integration strategies. Ranging from "easily accessible" to "complex & invasive":

Use track_until() & collect_particles() via TrackJob

Directly use the C99 header-only subset of SixTrackLib

Integrating SixTrackLib Into Programs :: Overview

SixTrackLib supports several di erent integration strategies. Ranging from "easily accessible" to "complex & invasive":

Usetrack _until() & collect _particles() via TrackJob

Integrate required functionality into SixTrackLib (C99,C++)

Directly use the C99 header-only subset of SixTrackLib

Integrating SixTrackLib Into Programs :: Overview

SixTrackLib supports several di erent integration strategies. Ranging from "easily accessible" to "complex & invasive":

Usetrack _until() & collect _particles() via TrackJob

Run-time compile and execute custom kernel function written in the header-only subset (currently only OpenCL, requires C99 interface)

Integrate required functionality into SixTrackLib (C99,C++)

Directly use the C99 header-only subset of SixTrackLib

Integrating SixTrackLib Into Programs :: Overview

SixTrackLib supports several di erent integration strategies. Ranging from "easily accessible" to "complex & invasive":

Use track_until() & collect_particles() via TrackJob

Use track_line() & manipulate particle state in-place

Run-time compile and execute custom kernel function written in the header-only subset (currently only OpenCL, requires C99 interface)

Integrate required functionality into SixTrackLib (C99,C++)

Directly use the C99 header-only subset of SixTrackLib

Example: Integration of SixTrackLib With PyCUDA

```
In [1]: import sixtracklib as st
        import numpy as np
        import pycuda
        from pycuda import gpuarray, driver, autoinit

        particles = st.ParticlesSet.fromfile("./particles.bin")
        lattice = st.Elements.fromfile("./lattice.bin")
        job = st.CudaTrackJob(lattice, particles)

        # Initialize the particles to some values on the host
        job.particles_buffer.get_object(0).x[:] = np.array( [-1.0, -2.0 ])

        # push particle state to the device
        job.push_particles()

In [2]: job.fetch_particle_addresses()
        ptr_particles_addr = job.get_particle_addresses(0) #0 .. only one particle set
        particles_addr = ptr_particles_addr.contents # particles_addr contains addr *on the device*

        print("num_particles  = {0:8d}".format(particles_addr.num_particles))
        print("x     begin at = {0:16x}".format(particles_addr.x))

num_particles  =        2
x     begin at =     7f6ca5000190
```

Example: Integration of SixTrackLib With PyCUDA (cont.)

Allows in-place particle manipulation between calls to `track_line()`

Similar implementation also available with `CuPY`

Example: Integration of SixTrackLib With PyCUDA (cont.)

Allows in-place particle manipulation between calls to `track_line()`

Similar implementation also available with `CuPY`

Example: Integration of SixTrackLib With PyCUDA (cont.)

Allows in-place particle manipulation between calls to `track_line()`

Similar implementation also available with `CuPY`

This way of integration is an additional motivation to support both OpenCL and CUDA!

But: corresponding integration with PyOpenCL tbd/wip

What Could Possibly Go Wrong? (Cuda Edition)

Cuda High-Level API context management is implicit ! sharing contexts relies on conventions (Alternative: Driver API)

What Could Possibly Go Wrong? (Cuda Edition)

Cuda High-Level API context management is implicit *!* sharing contexts relies on conventions (Alternative: Driver API)

PyCUDA, CuPY: context initialized & destroyed via Python

SixTrackLib objects created and destroyed in-between

What Could Possibly Go Wrong? (Cuda Edition)

Cuda High-Level API context management is implicit ! sharing contexts relies on conventions (Alternative: Driver API)

PyCUDA, CuPY: context initialized & destroyed via Python

SixTrackLib objects created and destroyed in-between

- Inputs (particles, lattices): used by SixTrackLib
- Device bu ers, Output bu er: managed by SixTrackLib

What Could Possibly Go Wrong? (Cuda Edition)

Cuda High-Level API context management is implicit ! sharing contexts relies on conventions (Alternative: Driver API)

PyCUDA, CuPY: context initialized & destroyed via Python

SixTrackLib objects created and destroyed in-between

- Inputs (particles, lattices): used by SixTrackLib
- Device bu ers, Output bu er: managed by SixTrackLib

Coordinated device selection PyCUDA / CuPY $ SixTrackLib

If selected devices mismatch, PyCUDA may try to be "helpful" ! very slow device $ device mem cpy

What Could Possibly Go Wrong? (Cuda Edition)

Cuda High-Level API context management is implicit ! sharing contexts relies on conventions (Alternative: Driver API)

PyCUDA, CuPY: context initialized & destroyed via Python

SixTrackLib objects created and destroyed in-between

- Inputs (particles, lattices): used by SixTrackLib
- Device bu ers, Output bu er: managed by SixTrackLib

Coordinated device selection PyCUDA / CuPY $ SixTrackLib

If selected devices mismatch, PyCUDA may try to be "helpful" ! very slow device $ device mem cpy

SixTrackLib currently only exposes default stream
everything else: explicit device synchronisation

What Could Possibly Go Wrong? (Cuda Edition)

Cuda High-Level API context management is implicit ! sharing contexts relies on conventions (Alternative: Driver API)

PyCUDA, CuPY: context initialized & destroyed via Python

SixTrackLib objects created and destroyed in-between

- Inputs (particles, lattices): used by SixTrackLib
- Device bu ers, Output bu er: managed by SixTrackLib

Coordinated device selection PyCUDA / CuPY $ SixTrackLib

If selected devices mismatch, PyCUDA may try to be "helpful" ! very slow device $ device mem cpy

SixTrackLib currently only exposes default stream
everything else: explicit device synchronisation

...

Beyond entry-level usage ! Platform-level integration!

Conclusion & Outlook

Writing a cross-platform, multi-language library that behaves like a proper Python module is challenging

Approach chosen for SixTrackLib is feasible & performance numbers and feedback from users are encouraging

Programming to least-common denominator & heavily relying on abstractions has its limitations *!* Look into re ection & automatic code-generation from `pysixtrack`

Goal: Using SixTrackLib as tracking backend to `SixTrack` and within the context of LHC@Home

Goal: Simplify installation and deployment for Python users

Goal: Optimise run-time performance and scalability

Goal: Continue integration e orts with `PyHEADTAIL` et al

Thank You For Your Attention!

Backup Slides

Example :: Tracking Map (Drift)

```
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(Drift) *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(Drift_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;
}
```

Example :: C++ API Example

```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;
}
```