SixTrack project
status, runtime environment and new developments

Present coordinator: R. De Maria
Main author and former coordinator: F. Schmidt.
LHC@Home IT support: I. Zacharov, L. Field, P. Jones, N. Oimyr.
Legal support: N. Ziogas, M. Ayass.
Special thanks to LHC@Home volunteers
Features Highlights

- SixTrack is **fast single particle tracking code** used to simulate charged particle trajectories in synchrotrons for many turns or many particles or both.
- It contains **symplectic** models for drift, thick dipole and quadrupoles, thin multipoles and solenoids, accelerating cavities, (frozen) beam-beam interactions, linear and non-linear deflecting cavities, wire, hollow e-lens, **scattering** models for collimators, beam gas interaction.
- It can be **interfaced** with Fluka, Geant4, ROOT.
- It also computes:
  - Phase space observables from tracking data: Linear and non linear invariants, Lyapunov analysis, tunes;
  - Optics functions using 4D, 5D, 6D Mais-Ripken formalisms;
  - High order transfer maps, normal forms;
- It supports Linux, Windows, MacOs, FreeBSD, NetBSD, OpenBSD, GNU hurd, on x86, amd64, ARM, ARM, PPC using gfortran, intel, nagfor compilers in a **numerically portable** way for all combinations.
- It originates from RaceTrack and it has been developed at CERN in the last few decades mainly by F. Schmidt and E. Mcintosh.
- It is **open source** and developed also outside CERN.
What is used for at CERN

SixTrack is used to:

• Evaluate impact of magnetic field imperfection in the LHC, HL-LHC, HE-LHC, FCC and specify target for field quality.
• Evaluate the impact of weak-strong beam-beam effect in the LHC, HL-LHC, FCC with or without machine imperfections.
• Simulate losses and collimation efficiencies and background in the SPS, LHC, HL-LHC. FCC
• Simulate failure scenarios (e.g. crab cavities in the SPS and HL-LHC).

SixTrack is used whenever:

• the speed of MadX is not sufficient
• the flexibility and accuracy of MadX/PTC is not needed

SixTrack main value is:

• in the integration in the CERN BATCH and LHC@Home environment (using the SixDesk runtime environment) and in the toolchains of many different LHC studies.
LHC Dynamic Aperture Studies (DA) pipeline

- Optics model
- Field imperfection
- Non-linear corrections

- MADX
- SixDesk
- SixTrack Tasks
  - SixTrack on BATCH or LHC@Home

- Particle Distribution

- SixTrack Lattice
- Tracking Results
- SixDesk, SixDB

- DA Results
Example of usage

Example of an LHC simulation:
- 30k initial conditions;
- $10^7$ turns;
- 20k beam line elements
- 4k high order multipoles
- 200 beam-beam interactions

Code speed:
- average 100 ns per particle per beam element
- 250-400 $\mu$s turn-particle on single core (depending on the hardware
- Memory footprint 100 MB

Survival time in number of turns as a function of decoupled actions.
Example of usage

5D, $E = 6.5$ TeV, $I_{\text{set}} = 510$ A, $\text{Beam} - \text{beam ON}$, $\varepsilon_n = 2.5 \mu \text{m}$, $\beta^* = 40 \text{ cm}$, $q = 15$

$(Q_x, Q_y) = (62.31, 60.32)$, $V_{\text{RF OFF}}$, $\delta p = 27 \times 10^{-5}$, 99 angles, $0.1 - 6.1 \sigma$, sliding NAF$^2$ $f_r = 550$ Hz, $\Lambda_r = 10^{-7}$ at MQXA.1, MQXA.3, MQXB.A2, MQXB.B2 of IP1, IP5

Analysis using Laskar’s tune analysis post processing method from LifeTrack.

HL-LHC simulation with the combination of beam-beam effects, Landau octupoles and power converter ripple in triplets.
Programming languages, style

SixTrack is made of:

- 70K lines ported to Fortran 2008 from Fortran 77/90 code blocks.
- It uses an external (but embedded) C library to generate portable special functions (crlibm) and perform frequency analysis (naff-cpp).
- It supports several compilers and operating systems.
- Can be linked with BOINC libraries for the LHC@Home project.

The style is monolithic, procedural with very few functions/procedures, dynamically allocated shared state.

Very steep learning curve, but the resulting executable is very fast.

The code uses vectorization as a form of parallelization.

SixDesk and SixDB are used to prepare, submit, manage, collect and process jobs for LHC and FCC studies starting from MadX input and a parameter definition file.

- About 70k lines shell scripting, Python.
- Jobs management and physics intermixed.
Each task is 60 particles for $10^5$ turns in the LHC (~1/2 hours), 25M task per year. LHC@Home more resources than what are we able to use.
SixTrack GPU effort

Main strategy:
• rewrite CPU intensive loop in subset of C such that it can be compiled for CPU and GPU using both OpenCL (1.2) to use the largest set of available hardware.
• Be compatible with CUDA for specific application if needed

Status:
• first proofs of principles and benchmark with realistic simulations done;
• still furthers tests, exploration of optimization strategies and feature coverage needed to go in production.

LHC: 8k drifts, 4.6k >11th order multipoles, 20k particles

<table>
<thead>
<tr>
<th></th>
<th>Year</th>
<th>Cores</th>
<th>Clock [MHz]</th>
<th>GFlops</th>
<th>Speed [us/(part*turn)]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel i7 920</td>
<td>2009</td>
<td>1</td>
<td>2670</td>
<td>5.2</td>
<td>545</td>
</tr>
<tr>
<td>Intel Xeon E5-2630</td>
<td>2016</td>
<td>1</td>
<td>2200</td>
<td>17</td>
<td>364</td>
</tr>
<tr>
<td>Intel 2x Xeon E5-2630</td>
<td>2016</td>
<td>2x10</td>
<td>2200</td>
<td>2x340</td>
<td>16</td>
</tr>
<tr>
<td>Nvidia GTX 1080</td>
<td>2016</td>
<td>2560/32</td>
<td>1700</td>
<td>288</td>
<td>12.8</td>
</tr>
<tr>
<td>Nvidia K20x</td>
<td>2015</td>
<td>2688/2</td>
<td>732</td>
<td>1312</td>
<td>10.8</td>
</tr>
<tr>
<td>AMD R9 280x</td>
<td>2013</td>
<td>2048/4</td>
<td>1000</td>
<td>1024</td>
<td>4.3</td>
</tr>
<tr>
<td>AMD W8100</td>
<td>2014</td>
<td>2560/2</td>
<td>824</td>
<td>2110</td>
<td>4.0</td>
</tr>
<tr>
<td>Nvidia P100</td>
<td>2016</td>
<td>3584/2</td>
<td>1480</td>
<td>5300</td>
<td>1.8</td>
</tr>
<tr>
<td>Nvidia V100</td>
<td>2017</td>
<td>5120/2</td>
<td>1370</td>
<td>7014</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Main bottleneck for GPU: # of register and FP64 Gflops rate

The aim is to produce an independent and portable C library to incorporate it SixTrack and in any other code that needs a fast tracking engine.
SixTrackLib: scaling

Scaling using **monolithic** kernel:
- Whole simulation without leaving GPU
- Big switch-case statement to choose the element.

Two examples:
- **minimal kernel**: just the elements needed
- **BB**: additional beam-beam lens in the code, but not used!

**Optimal scaling**: 20k particles

**Optimization work**: kernel complexity, number of kernels
Resources and future plans

Website (cern.ch/sixtrack) is the single point of information:
• Access to code: GitHub repository for SixTrack
• Documentation:
  • User manual: stable but under review
  • Physics manual: draft in progress but almost complete
  • Developer Wiki: informal live wiki document
• Contacts: Support email, Mailing List.
• SixTrack is licensed with LGPLv2.0

Future plans:
• Continue the development and support of the main code:
  • Adding new physics for LHC/HLLHC/FCC studies
  • Continuous effort in refactoring and documentation
• Develop SixTrackLib:
  • C Library implementing SixTrack Physics that can be embedded in other applications
  • Support GPU