HK Reconstruction

Mathieu Guigue

HK Software, October 17th 2024

Performance tracker

First implementation of a performance tracker inside fiTQun
Main goal: count the number of iterations and calls of functions

Infrastructure:

- Class PerfTracker which holds a Yaml::Node with various counters (global and event-by-event)
- Global instance accessible in any class with statement like:
```
PerfTracker& perftracker = PerfTracker::getInstance();
```
- Add a quantity to the tracker or increment value using:
```
PerfTracker::getInstance().AddCurrEvtInfo("a_key", a_value);
PerfTracker::getInstance().IncrCurrEvtInfo(“a_key”);
PerfTracker::getInstance().IncrGlobalInfo("a_key");
```
- Dump counter at the end of runfiTQun and save it into a file (PerfTracker.log)
- Can be compared with a reference as part of Continuous Integration

```
09:26:10  INFO PerfTracker.cc:86: globals: ~
09:26:10  INFO PerfTracker.cc:86: events:
09:26:10  INFO PerfTracker.cc:86:   - event_num: 0
09:26:10  INFO PerfTracker.cc:86:     prefit_iter: 62149
09:26:10  INFO PerfTracker.cc:86:   - event_num: 1
09:26:10  INFO PerfTracker.cc:86:     prefit_iter: 62699
```

Convenience implementation

Cmake compilation
- feature enabled by default
- can be disabled using the cmake option `-DfiTQun_PerfTracker_ENABLED=OFF`

Convenience macros

```
#define PT_OUTFILENAME(name) PerfTracker::getInstance().SetFilename(name);
#define PT_ADD_GLOBAL_INFO(key,value) PerfTracker::getInstance().IncrGlobalInfo(key);
#define PT_ADD_CURREVT_INFO(key,value) PerfTracker::getInstance().AddCurrEvtInfo(key,value);
#define PT_INCR_GLOBAL_INFO(key) PerfTracker::getInstance().IncrGlobalInfo(key);
#define PT_INCR_CURREVT_INFO(key) PerfTracker::getInstance().IncrCurrEvtInfo(key);
#define PT_NEXTEVT PerfTracker::getInstance().NextEvent();
#define PT_DUMP PerfTracker::getInstance().DumpInFile();
#define PT_PRINT PerfTracker::getInstance().Print();
```

Continuous Integration
- Run a test bash script for a given particle gun type
- Produces a log file and store it as artefact of the build e.g.
  https://github.com/hyperk/fiTQun/actions/runs/11456485010/artifacts/2086869469

Next steps

~~- Add convenience preprocessor macros (could be used to compile code without this feature)~~
~~- Cosmetic changes (set output filename, a bit more documentation, etc)~~
- Release into the master branch (currently under github.com/hyperk/fitqun:feature/perf_tracker)
- Setup the CI check
    ~~- Current plan: storing the file as part of CI artefacts and compare with artefact in master branch~~
    - Could be stored externally for long-term evolution tracking
    - Compare with the master branch version in every PR (and replace once merged)
- Add important counters throughout the code

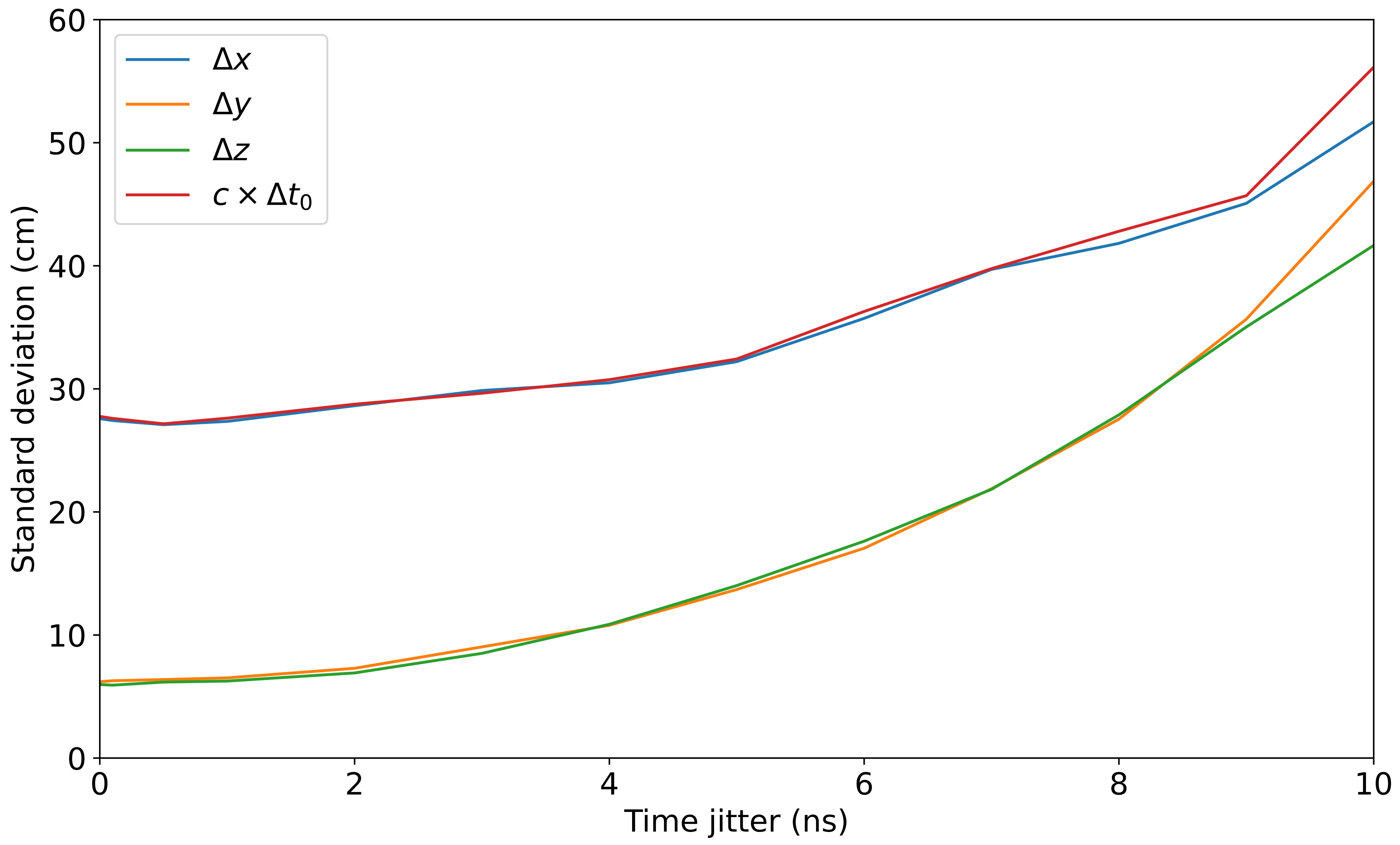
Timing effect on reconstruction

HK Electronics requires 100ps time jitter of the time synchronisation link
- another way to see this is that random error on hit time should be smaller than 100 ps (roughly…)

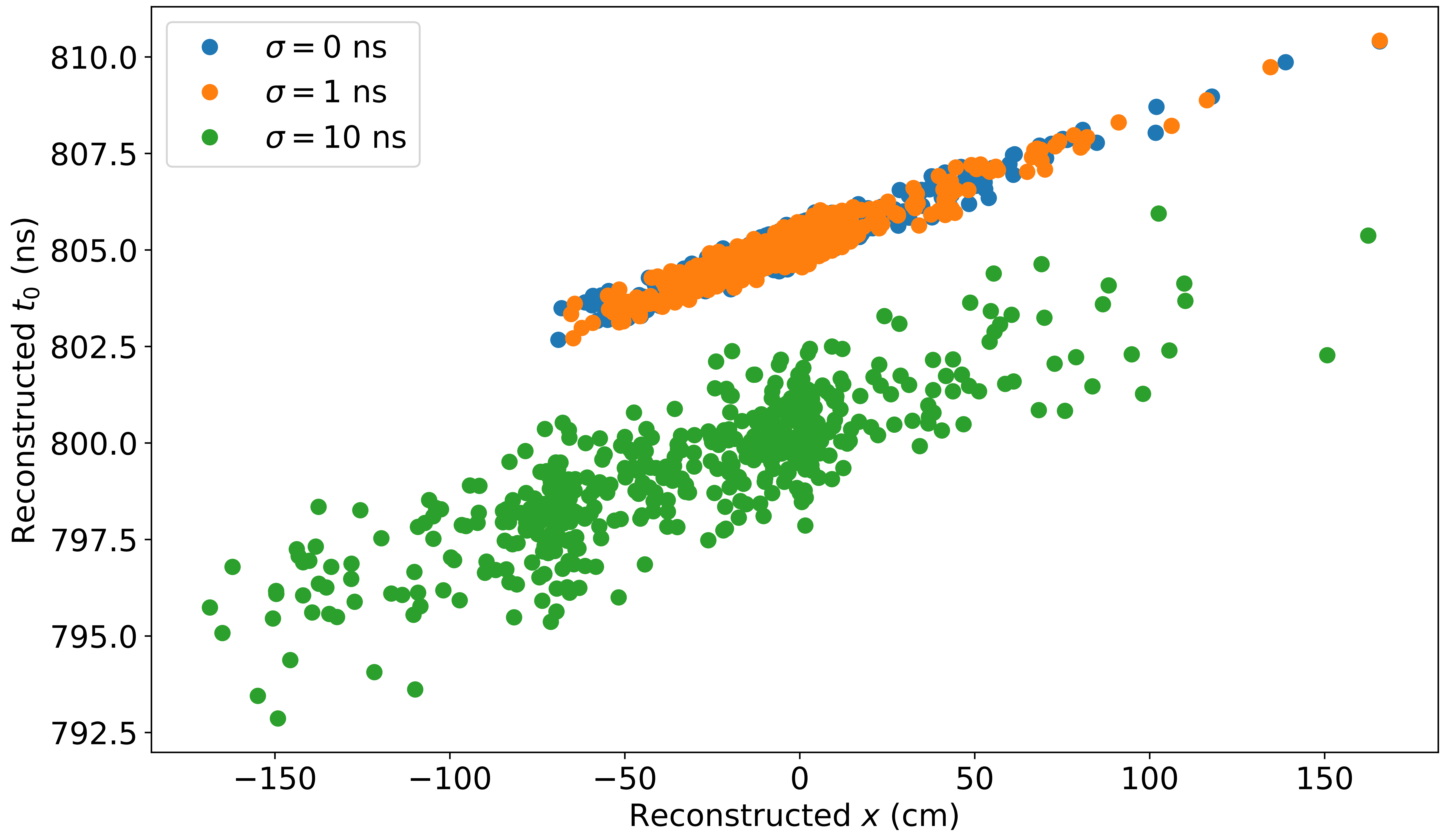
Simulation of 500 electrons with 300 MeV energy at the center of detector toward barrel (x direction)
- Intrinsic transit time spreading (TTS) of 5 ns: fluctuations of electrons transit time in dynode
- WCSim 1.12.17

Reconstruction using fiTQun (HK 2020 tuning)
- Add a time smearing when reading the simulations
- See impact on reconstructed quantities

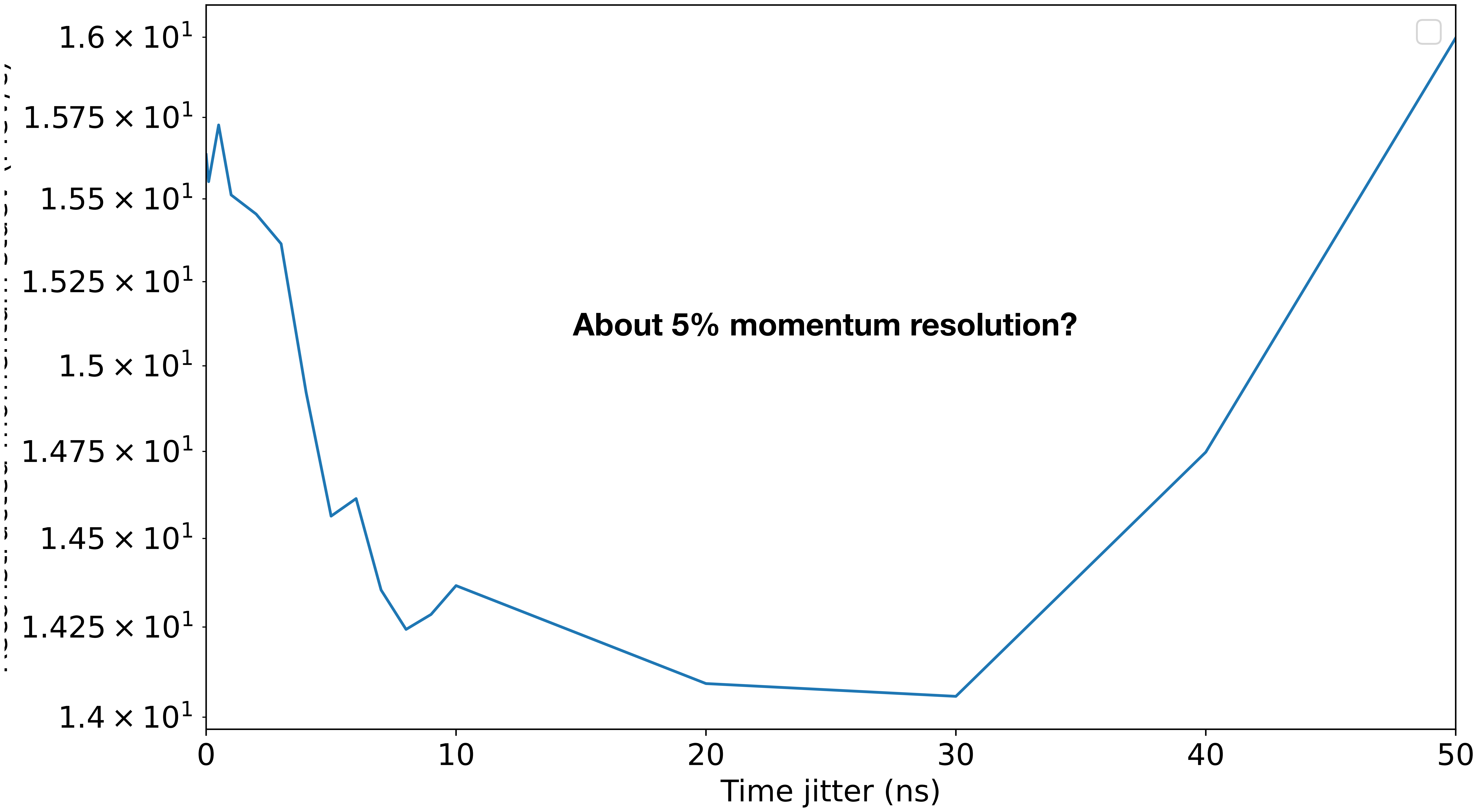
Jitter impact on spatial resolution



Correlation x position and time



Momentum resolution



About 5% momentum resolution?