CUDA CKF Implementation in TRACCC

(tracc#352)

Beomki Yeo

Algorithm Overview

CKF algorithm (One seed case)

1. Spawn a track from a seed, which is a bound track parameter
2. Apply material interaction and Kalman update for the measurements on the surface
3. Pick measurements that satisfy (chi2 < chi2_max) and record their indices in a link container
4. Spawn kalman-updated tracks for the good measurements
5. Propagate the tracks to the next surface
6. Repeat 2 - 5 until all tracks are exhausted
7. Build full tracks from the link container

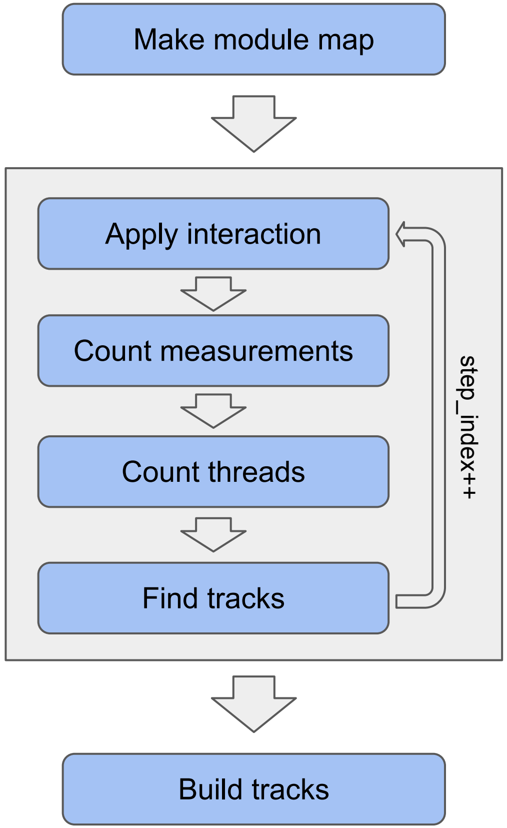
GPU Implementation

- The main idea is launching the kernels for every step. A step covers a set of kernels called between two surfaces.
  - This enables us to evaluate the proper size of link container and grid-block dimension based on the number of spawned tracks
  - A downside of this approach is that the number of kernel launches will be [the number of steps X the number of kernels per step]

- All algorithms run on device side except for some device-to-host transfer of a global counter object

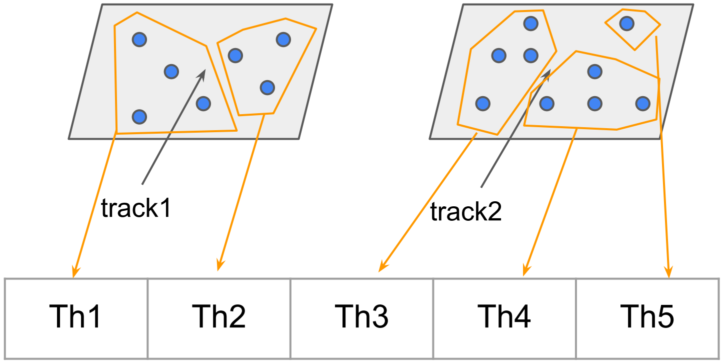
GPU Implementation

- 6 kernels in total:
  - *make_module_map*
    - To convert module ID to measurement container index
  - *apply_interaction*
    - To apply material interaction to tracks
  - *count_measurements & count_threads*
    - To calculate the number of measurements to iterate per GPU thread, and reassign the block dimensions for *find_tracks*
  - *find_tracks*
    - To apply Kalman Update to measurements and record the the index of good measurements in the link container
    - Propagate the updated tracks to the next surface
  - *build_tracks*
    - Build full tracks from the link container



find_tracks kernel

- We want to make each thread iterate over *N* (or less than *N*) measurements equally
  - *count_threads* provides the information on how many threads is required for every track

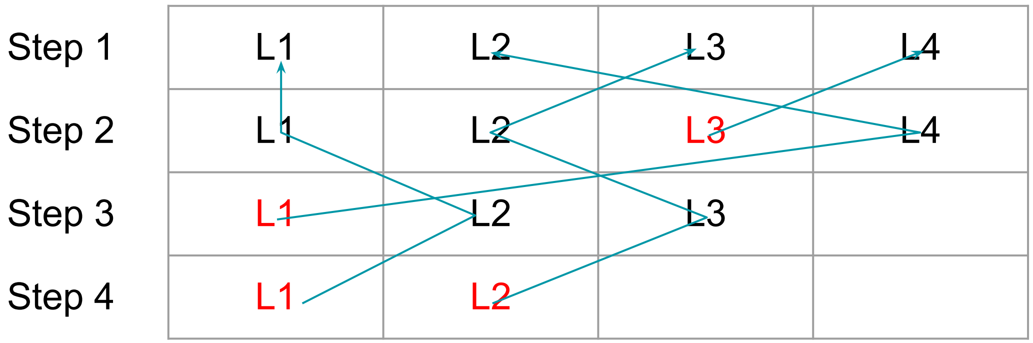


- If the chi2 of measurement < chi2_max, add its measurement index and an index of the link from the previous step to the link container

- Propagate the kalman-updated tracks to the next surface
  - If a track reaches a surface with measurements, its bound track parameter will be used for the next step
  - Otherwise, the link is added to the *tip link container* as well so that we can know which links in the link container are the final measurements of the full tracks

build_tracks kernel

- Every link holds an index to the corresponding measurement and an index to the link from the previous step
- A full track is built by starting from the *tip* link and connecting it to the link of previous steps, iteratively

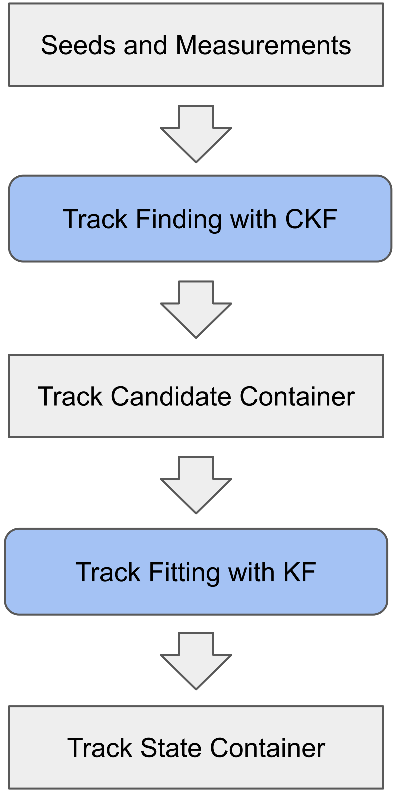


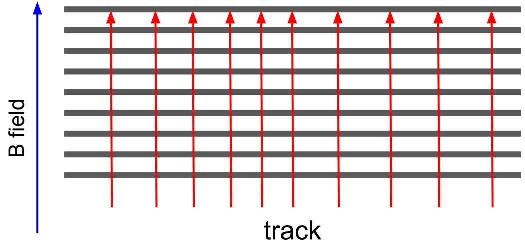
Building track from link container

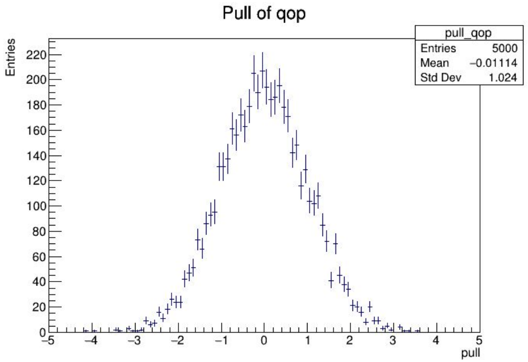
Unit test for Sparse Tracks

Unit test was written for sparse tracks which does not make any combinatorics:

- The number of found tracks should be equal to the number of truth tracks
- Track fitting is applied to the found tracks to run the pull value test
- ACTS' default chi2_max (15) misses few tracks, e.g. ~ 1 out of 1000 tracks. So changed to 30







Concerns

- The implementation is not complete yet but the PR itself is ready
    - Some features is hard to test with sparse tracks. Full implementation will come with unit test with dense tracks and CPU version

- Implementing SYCL version might be easy if thrust functions are compatible with SYCL

- Haven't adapted to the flat EDM of measurement, but it could be done later
    - Measurements should be sorted by module ID and delivered with a vector of boundary indices

Summary

- CUDA CKF is implemented and it seems working well without serious bugs

- Still many things to work on:
  - Implemente CPU version
  - Write unit test for dense tracks
  - Check the physics performance
  - Check the speedup