Parallelized Kalman-Filter-based Reconstruction of Particle Tracks on Many-Core Architectures with the CMS detector

11 March 2019

G. Cerati\textsuperscript{4}, P. Elmer\textsuperscript{3}, B. Gravelle\textsuperscript{5}, M. Kortelainen\textsuperscript{4}, S. Krutelyov\textsuperscript{1}, S. Lantz\textsuperscript{2}, M. Masciovecchio\textsuperscript{1}, K. McDermott\textsuperscript{2}, B. Norris\textsuperscript{5}, A. Reinsvold Hall\textsuperscript{4}, D. Riley\textsuperscript{2}, M. Tadel\textsuperscript{1}, P. Wittich\textsuperscript{2}, F. Würthwein\textsuperscript{1}, A. Yagil\textsuperscript{1}

Overview

- Motivation

- Introduction of parallelized Kalman Filter (KF) tracking
  - Aka. mkFit

- Performance
  - Physics performance
  - Time performance

- Plans & Summary

Project website: http://trackreco.github.io/
Motivation

• Exponential growth of CPU needs for high pileup at LHC to be addressed, to speed up event reconstruction
  → Review tracking strategy
  → Max utilization of computing resources
     ◦ Many-core SIMD and SIMT arch’s

CMS event display from 2018 high PU (136) run
Kalman Filter

- **Kalman Filter** technique consists of two steps:
  1. Produce an estimate of the current state (prediction)
  2. Update the state with the next measurement

- **Why** use it for track track reconstruction:
  - Robust handling of multiple scattering, energy loss, and other material effects
  - Widely used in HEP field
  - Demonstrated physics performance

- **Our goal:**
  - Exploit parallel and vector architectures
    - Improve computational performance
  - Maintain physics performance
Track building, in a nutshell

- Track building is the primary focus of our project:
  - Start with a seed track ($\leq 4$ measurements)
    - Seed finding is out of our scope
  - **Estimate** track state from seed track
  - **Propagate** track state to next detector layer
  - Find candidate detector response “hits” near projected intersection point(s) of track with layer
  - Evaluate goodness of fit for each hit, wrt. track
  - Select best fit track-hit combinations as **track candidates**
  - **Update** estimated state of all track candidates with new hit
  - **Propagate** all track candidates to next layer
  - **Iterate**
The parallelized KF tracking project

- Parallelized KF tracking project ongoing for 3+ years
  - **Aim:** implementation of traditional **KF-based tracking**, maximizing usage of **vector units** and **multicore** architectures

- R&D started in context of simplified geometry and simulation

- Current focus on realistic geometry & events
  - CMS detector geometry
  - Realistic events from CMS simulation
  - Seed tracks from CMS
  - Integration in CMS software (CMSSW)
    - **Aim:** test online in LHC **Run III @ High Level Trigger** (HLT)
    - Will extend to HL-LHC and Phase-II CMS geometry

- **Note:** will refer to parallelized KF tracking as “**mkFit**”
  - Matriplex Kalman Finder/Fitter
Geometry is implemented as a plugin.

Do not deal with detector modules, only layers (unlike current CMSSW).
- Algorithm is lighter & faster

Track propagation to center of layer, then hit selection.
- Additional propagation step for every compatible hit is required (exploiting vectorization)

Mono/stereo modules are described as separate layers.

Can pick-up only one hit/layer during outward propagation.
- Could pick-up overlap hits during backward fit, or afterwards
The big mkFit picture

1. **Seed finding**
   - For development, use either **CMSSW** seeds or **MC truth** seeding
   - For CMSSW seeds, apply **cleaning** prior to track finding
   - When employed @ CMS HLT, will use available seeds

2. **Track finding**
   - **Primary focus**
     - First milestone: tracking with CMS-2017 geometry
     - 4-hit pixel seeds with beam-spot constraint

3. **Track fitting**
   - Can do track fitting within mkFit
   - However, rely on CMSSW for final fit
     - Most/more precise set of tools

4. **Validation**
   - Physics performance
   - Time performance
Parallelization & vectorization

• **Task scheduling** is handled via **TBB** library, by Intel

• **Parallelization** at multiple levels
  - parallel-for: **N** events in flight
    - parallel-for: **5** regions in $\eta$ in each event
      - parallel-for: seed-driven batching, 16 or 32 seeds per batch
        → **Vectorized** processing of candidates, where possible

• **Architectures:**
  - KNL (64 physical cores, 256 logical cores)
    - Intel® Xeon Phi™ CPU 7210 @ 1.3 GHz, AVX512 support
  - SNB (12 physical cores, 24 logical cores)
    - Intel® Xeon® CPU E5-2620 0 @ 2 GHz, AVX2 (256) support
  - SKL (32 physical cores, 64 logical cores)
    - Intel® Xeon® Gold 6130 CPU @ 2.1 GHz, AVX512 support
  - Nvidia / CUDA (GPU) – to a limited extent
• mkFit algorithm can be used in two different setups:

1. **Standalone code**
   - Input: simple data-format, from memory dump of data structures
     - Hits, seeds, simulated and reconstructed tracks
   - Useful for development and validation of computing performance

2. **Integrated within CMSSW**
   - Input: data are pulled from CMSSW event record
     - Format into mkFit data structures
     - After building, mkFit tracks are re-formatted into CMSSW tracks
   - mkFit is deployed as external package + CMSSW module

• For both setups, test using CMSSW samples
  - 10-μ samples – mainly for development
  - \( t\bar{t} \) (PU=0, 50, 70) samples
Physics performance
Physics performance: validations

Mario Masciovecchio (UCSD), 11 March 2019

• Different validation suites are used for the two runtime options
  o Different choices and definitions to achieve different goals (next slide)

1. mkFit validation: algorithm-level efficiency
   o Used for standalone configuration
   o Goal: validate physics performance on long (≥10 hits) tracks, wrt. CMSSW
   o Starting point to evaluate mkFit physics performance

2. Multi-Track Validation (MTV): absolute efficiency
   o Used for mkFit integrated into CMSSW
   o Goal: evaluate absolute performance of tracking algorithm
   o Including seed building efficiency

• Efficiency = fraction of reference tracks matched to a reconstructed track
• Duplicate rate = fraction of reference tracks matched to >1 reconstructed track
• Fake rate = fraction of reconstructed tracks not matched to any reference track
# Validation definitions

<table>
<thead>
<tr>
<th></th>
<th>mkFit validation</th>
<th>MTV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference tracks</td>
<td>• SIM or CMSSW tracks with ≥ 12 layers (including 4 seed layers)</td>
<td>SIM tracks satisfying</td>
</tr>
<tr>
<td></td>
<td>• SIM tracks must be matched to a seed</td>
<td>• $p_T &gt; 0.9$ GeV</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• $</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• $</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No seed matching requirement</td>
</tr>
<tr>
<td>To-be-validated</td>
<td>• Reco. tracks with ≥ 10 hits</td>
<td>No additional requirements</td>
</tr>
<tr>
<td>reconstructed tracks</td>
<td>• For mkFit tracks, 4 of the hits are required from the seed</td>
<td></td>
</tr>
<tr>
<td>Matching criteria</td>
<td>Considered matched if ≥ 50% of the hits are shared, excluding the seed</td>
<td>Considered matched if ≥ 75% of the clusters of the reco track</td>
</tr>
<tr>
<td>between ref. and reco.</td>
<td></td>
<td>contain charge induced by the reference track</td>
</tr>
<tr>
<td>tracks</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
mkFit efficiency: mkFit validation

- $t\bar{t}$ (PU=70)
  - Algorithm-level efficiency, for long tracks
    -> mkFit is at least as efficient as CMSSW
mkFit efficiency: MTV

• $t\bar{t}$ (PU=50)
  o Absolute efficiency of full tracking algorithm (including seeding)

→ mkFit is as efficient as CMSSW standard tracking for tracks with $\geq 12$ layers
  o As observed in previous slide

→ Inefficiency at shorter track lengths has been understood. Work in progress.
  o Run a test to confirm hypothesis about inefficiency for shorter tracks
    ▪ “Feature” of ranking of candidates
  o Test confirms hypothesis is correct
  o Working on permanent solution
Time performance

Mario Masciovecchio (UCSD), 11 March 2019
Time performance: integrated into CMSSW

• Time performance of mkFit when integrated into CMSSW, vs. CMSSW
  o For corresponding tracking step
  o $N(\text{threads}) = 1; \ N(\text{streams}) = 1$
  o Using $t\bar{t}$ (PU=50)
  o Test on SKL-SP Gold
  o mkFit compiled with AVX512

→ **Track building** is **4.4x faster** (mkFit)
  o mkFit time currently accounts for data-format conversions
    ▪ ~40% ⇒ Actually ≥ 7x faster
  o mkFit gets faster with # threads

➢ Can only improve from here
Track building for mkFit is faster than track fitting (unlike CMSSW)

- Fake and duplicate rates are higher
- Larger amount of tracks to be fitted
- Will need dedicated “final” cleaning
  - Work in progress.
• \( \bar{t}t \) (PU=70), using standalone configuration on SKL-SP Gold
  o Speed-up vs. # threads for track building
  → Excellent scaling at low # threads
Plans

Mario Masciovecchio (UCSD), 11 March 2019

- Continue working on integration into CMSSW
  - Optimize conversion between data formats
- Implement final track cleaning & quality selection
  - To minimize duplicate & fake rates in mkFit
  - Without it, fit on mkFit tracks takes longer than CMSSW
    - Due to larger amount of duplicate and fake tracks
- Explore GPU-based implementation (not covered today)
- **Goal:** deploy into CMSSW and test (online, @ HLT) in LHC Run III
- **Longer term:** extend to Phase-II CMS geometry, for HL-LHC
Summary

• Status of parallelized KF tracking (aka. mkFit) is well advanced
  o As well as its integration into CMSSW

• First round of physics performance optimization in mkFit resulted in equivalent or better efficiency than CMSSW for long tracks
  o Current work on improving performance for shorter tracks

• Already observe ~4.4x speed-up wrt. CMSSW, when running within CMSSW (without optimization of data-format conversion)

• Further development is on-going to deliver “final” tracks

• Performance expected to be mostly useful @ HLT, or even offline (possibly already during LHC Run III)
Data structure: Matriplex

- “Matrix-major” matrix representation designed to fill vector unit with \( n \) small matrices operated on in synchronization
Track building: challenges

- Good physics performance (efficiency) requires consideration of multiple track hypotheses
  - In a dense detector geometry, many tracks will find hit candidates that are the best local fit, but lead to a globally poor fit
    - Consider many track hypotheses for every seed, depending on occupancy

- Track building involves multiple branch points
  - Select candidate hits at each layer
  - Evaluate a variable number of track candidate-hit candidate combinations
  - Select best combinations for propagation to next layer
  - A number of seeds “die” out after few layers

→ Lead to irregular work loads and memory access patterns
## Key differences: mkFit vs. CMSSW

<table>
<thead>
<tr>
<th>Seed Cleaning</th>
<th>CMSSW</th>
<th>MkFit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build tracks sequentially and reject seeds that have already been included in a track candidate</td>
<td>Everything is done in parallel. Apply seed cleaning before trying to build any tracks. After track building we can specifically try to remove duplicates (not done yet)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hit Position</th>
<th>CMSSW</th>
<th>MkFit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reevaluate the hit position using the track direction</td>
<td>Hit position is taken from local reconstruction and not updated</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Inactive modules</th>
<th>CMSSW</th>
<th>MkFit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Able to access the detector status DB to make sure modules were active</td>
<td>Cannot access DB so no knowledge of inactive modules</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Geometry</th>
<th>CMSSW</th>
<th>MkFit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Retains information about the detailed CMS geometry</td>
<td>Knows only about layers, not detector modules</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Magnetic Field</th>
<th>CMSSW</th>
<th>MkFit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameterized magnetic field</td>
<td>Currently using flat field. Will eventually use parameterized field</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Module Overlaps</th>
<th>CMSSW</th>
<th>MkFit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Can pick up multiple hits while track building</td>
<td>MkFit can only pick up one hit. We could pick up overlap hits during backward fit. Not implemented yet.</td>
<td></td>
</tr>
</tbody>
</table>
mkFit duplicate rate: mkFit validation

- $t\bar{t}$ (PU=70)
mkFit fake rate: mkFit validation

- $t\bar{t}$ (PU=70)