Parallelized and Vectorized Tracking Using Kalman Filters with CMS Detector Geometry and Events

Giuseppe Cerati, Peter Elmer, Brian Gravelle, Matti Kortelainen, Vyacheslav Krutelyov, Steven Lantz, Matthieu Lefebvre, Mario Masciovecchio, Kevin McDermott, Boyana Norris, Allison Reinsvold Hall, Daniel Riley, Matevž Tadel, Peter Wittich, Frank Würthwein, and Avi Yagil

1UC San Diego, La Jolla, CA, USA 92093
2Princeton University, Princeton, NJ, USA 08544
3Cornell University, Ithaca, NY, USA 14853
4Fermilab, Batavia, IL, USA 60510-5011
5University of Oregon, Eugene, OR, USA 97403

Abstract. The High-Luminosity Large Hadron Collider at CERN will be characterized by greater pileup of events and higher occupancy, making the track reconstruction even more computationally demanding. Existing algorithms at the LHC are based on Kalman filter techniques with proven excellent physics performance under a variety of conditions. Starting in 2014, we have been developing Kalman-filter-based methods for track finding and fitting adapted for many-core SIMD processors that are becoming dominant in high-performance systems.

This paper summarizes the latest extensions to our software that allow it to run on the realistic CMS-2017 tracker geometry using CMSSW-generated events, including pileup. The reconstructed tracks can be validated against either the CMSSW simulation that generated the hits, or the CMSSW reconstruction of the tracks. In general, the code’s computational performance has continued to improve while the above capabilities were being added. We demonstrate that the present Kalman filter implementation is able to reconstruct events with comparable physics performance to CMSSW, while providing generally better computational performance. Further plans for advancing the software are discussed.

1 Introduction

Over the past few years, plans for the High-Luminosity Large Hadron Collider upgrade project, and the accompanying tenfold leap in luminosity, have made it clear that a significant research and development effort is required towards the 2020 to 2025 timeframe to meet the increased complexity and computational requirements of the track finding algorithms. The expected increase in event complexity, coupled with the technological changes that continue to drive interest in multi/many-core processors, have motivated the community to explore radically different algorithms and computing architectures to address the anticipated issues. Our approach, however, has been to focus on the traditional, well-known,
and well-understood Kalman Filter (KF) method, to see how far KF-based tracking can be pushed in this new environment. To that end we have been developing the mkFit framework, designed from the ground up for performance, that is better suited to utilize the types of parallelism available in contemporary general-purpose computing hardware.

There were several motivating factors that made us choose this route. First, there was practically no other research going in this direction, and we believed that an honest attempt needed to be made to modernize the algorithms that have been in use for over 30 years and whose physics performance is understood in detail. Second, we felt that while a combinatorial KF-based tracking algorithm is not trivial to parallelize, it is, mathematically, the most efficient algorithm for accomplishing its purposes. Given proper handling of measurement and track parameter errors, the algorithm selects possible trajectories with maximal available precision and thus rejects statistically unlikely combinations at an early stage, a feature which will only become more important in a high occupancy environment. Third, at that time it was unclear which computing architectures would become predominant in scientific computing and therefore it seemed prudent to support mainstream options while aiming also to explore more innovative ones whose future seemed less certain (Xeon Phi products and GPGPUs). It should be pointed out that many factors influence the choice of the hardware that is ultimately deployed and made available to the experiments; other parts of event reconstruction, event simulation, and physics analysis may be more or less amenable to a high degree of parallelization.

All things considered, a fully-integrated, parallelized and vectorized implementation of a KF-based tracking application provides a strong reference point for evaluation of more exotic solutions, both in terms of computational as well as physics performance. Indeed, new solutions will have to demonstrate superior performance in both of these respects for operations in environments with moderate time constrains such as high-level trigger applications and offline reconstruction.

This paper focuses on recent developments required to fully support realistic geometry of the Compact Muon Solenoid (CMS) detector and to process simulated CMS data with up to 70 minimum bias pp collisions superimposed over the signal $t\bar{t}$ events. To facilitate this discussion a brief overview of the project is given in appendix A and an outline of our track finding algorithm is shown in appendix B. Generalized geometry handling is described in section 2 and physics and computational performance are presented in sections 3 and 4.

2 Handling of CMS geometry and events

2.1 Geometry & Detector description

Geometry in mkFit is described as a vector of LayerInfo structures that contain the physical dimensions of a layer ($r_{\text{in}}$, $r_{\text{out}}$, $z_{\text{min}}$ and $z_{\text{max}}$ are sufficient for both barrel and endcap layers) and parameters and flags relevant for track finding. This includes information about layer type, stereo/mono layers, hit search windows, and an optional hole in detector coverage as needed for the CMS endcap detectors (this could be extended for even more general acceptance handling).

For track finding, steering parameters need to be defined for every tracking region. So far, it has been sufficient to consider only $\eta$ regions (barrel, $+z/-z$ transition, and $+z/-z$ endcap) but the concept could be used also to separate regions by $p_T$ or by tracking iteration. The steering parameters contain, most importantly, a vector of LayerControl structures that hold layer indices (mapping into the LayerInfo vector) that need to be traversed during track finding. Additionally, it contains layer parameters and flags that are specific for this tracking region, such as tagging layers as possible seeding layers or as layers to be only considered.
during backward fitting. This allows the track finding algorithm to be completely agnostic of the detector structure: it simply follows the layer propagation plan in the steering parameters and executes operations in accordance with the control flags in LayerControl and LayerInfo structures.

Geometry and steering parameter setup is implemented as a plugin that populates the in-memory data structures with the required information. With this functionality, we are able to support both a simple geometry used for development and CMS-2017 geometry with all detector-specific information existing only in the plugin code. For the CMS-2017 geometry, we include the effects of multiple scattering and energy loss by defining two-dimensional arrays for the radiation length and material composition that is quickly indexed in $r-z$. These constants are taken from CMS simulation for the amount of material a particle would traverse propagating from module to module. mkFit supports usage of constant and parametrized magnetic field; type of field can be selected each time propagation is required in the code, e.g., for inter/intra layer propagation, forward/backward fit.

### 2.2 Handling and processing of CMS events

When processing CMS events mkFit relies on hit and seed data to be provided externally. In the standalone case, mkFit reads these data from a binary file created by a converter application. Additionally, the binary file can also contain vectors of simulated tracks and reconstructed tracks as found by standard CMS tracking used in the validation of mkFit’s performance.

Before passing seeds to mkFit for track finding, the seed collection is “cleaned” by removing multiple instances of seeds that are most likely based on hits belonging to the same outgoing particle. The cleaning algorithm uses the identity of hits and fitted seed parameters $p_T$, $\eta$, and $\varphi$ to eliminate duplicate seeds and is tuned so as to not cause any drop in track finding efficiency for high pile-up events. The duplicate seeds arise due to detector module overlaps that are rather significant, especially in the endcaps, where the modules on the same blade of a disk overlap considerably, and the modules on the two rings of the same disk can be spaced over almost 10 cm. Low $p_T$ tracks (below 2 GeV/c) are more affected due to bending in the $r-\varphi$ plane during their flight through the detector wheel. Multiplicity of seeds from a single particle frequently reaches 8 and can go as high as several tens.

In principle, seed cleaning could be performed as a final step in the seed finding algorithm; however, due to the way standard track finding works in CMSSW, this was not deemed necessary. CMSSW processes seeds one by one and when a track candidate is found, its hits are tagged as used. A seed is rejected if all its hits have already been used by a track candidate found earlier. The technique relies on CMSSW backward propagation to find additional hits on the seeding layers, and reduces the duplicate seeds in the first step of their consideration to a negligible level. This is not possible in mkFit where we process up to 32 seeds in parallel and, as we try to processes together seeds that are close in $\eta$ and $\varphi$ to maximize memory cache reuse of hit data, this could lead to significant waste of processing slots.

As already mentioned, we have recently started the process of including mkFit in standard CMS software distribution. mkFit is used as an external software package with a dedicated CMS processing module running within the CMS framework. This module packages the input data (seeds and hits) in format expected by mkFit, and provides high-level configuration and steering of mkFit execution. When an event is processed, it copies resulting tracks back into CMS format. This mode of inclusion allows mkFit code to remain independent of CMS particularities and overhead as well as allows us to perform development and testing in a more lightweight environment.
3 Physics performance

This section presents current basic physics performance plots for mkFit running on CMS-2017 geometry and CMSSW generated sample of 500 $t\bar{t}$ events superimposed with a mean of 70 minimum-bias $pp$ collisions per event. Constant magnetic field of 3.8 T has been used. We are showing results for processing of CMS initial tracking iteration where seeds are required to have 4 hits all coming from distinct inner pixel layers and be compatible with the beam spot constraint. Results shown for CMSSW are using the same set of input seeds.

While these results show the actual performance of mkFit, they are preliminary in the sense that we know further work is necessary to make a fair comparison between CMSSW and mkFit initial iteration tracking:

- mkFit’s hit selection windows, candidate scoring criteria, and final track quality criteria have not yet been tuned for optimal performance.
- Cleaning and merging of the final track collection have not yet been implemented in mkFit. This includes removal of duplicate tracks due to multiple seeds per particle.
- To ensure a fair comparison of efficiency, the same final track selection criteria and post-processing need to be applied for both algorithms. CMSSW intentionally uses stronger requirements in initial iteration, relying on later iterations to pick up less likely track candidates.

Track finding efficiency versus $p_T$ and $\eta$ for mkFit and CMSSW are shown in figure 1. mkFit’s performance is essentially equivalent to that of CMSSW for $p_T > 1$ GeV/c. Below that, mkFit’s inefficiency is largest in the transition region and noticeable in the endcaps. We believe that tuning of hit selection windows and candidate scoring criteria can help us achieve efficiencies comparable to CMSSW for all tracking regions, down to $p_T = 450$ MeV/c.

Figure 1. Efficiency versus $p_T$ (left) and efficiency versus $\eta$ for tracks with $p_t > 0.9$ GeV/c, our target $p_T$ limit for CMS HLT operation (right).

Figure 2 shows mkFit’s duplicate track rates versus $p_T$ and $\eta$. CMSSW’s duplicate rates are 0. Duplicate rate is significant for all values of $p_T$. In $\eta$, it is below 5% level in the barrel and rises sharply when tracks start entering the endcap disks. The duplicate rate distribution is exactly the same when using 10 muon events the duplicate rate and can be explained entirely by duplicate seeds and the absence of a duplicate removal procedure in mkFit (see section 2.2).

As already mentioned, further work is required to make more detailed assessment of mkFit’s performance. However, with mkFit being available within the CMSSW, all quality assurance and validation tools developed for CMS tracking are available for more detailed studies and debugging.
4 Computational performance

Computational benchmarks are shown for our main development platforms:

- SNB – Sandy Bridge – 2 sockets x 6 cores: Intel Xeon CPU E5-2620 @ 2.00GHz
- KNL – Knights Landing – 64 cores: Intel Xeon Phi CPU 7210 @ 1.30GHz
- SKL-SP – Skylake Gold – 2 sockets x 16 cores: Intel Xeon Gold 6130 CPU @ 2.10GHz

   SKL-SP processor cores feature different frequency characteristics depending on which vector instruction set is being used on them. The base frequency can change, as can the enhanced “turbo” frequency provided by the Turbo Boost feature. The latter also depends on thermal state of the die and the number of cores being actively used. We have only recently started working with Skylake and while we understand the issue and its consequences, we have not yet attempted to clear up the confusion that arises in benchmark results due to these features, partially due to poor support of Hardware-Controlled Performance States controls on Linux. On our SNB and KNL machines the Turbo Boost feature is turned off. KNL can also vary frequency to a lesser extent when AVX-512 code is being executed. This will be further discussed in individual subsections and results for SKL-SP will be presented separately.

   Results presented in this section were obtained using a subset of the events with the configuration described in the introduction to section 3. Intel icc compiler was used to compile the code. mkFit uses Intel Thread Building Blocks (TBB) for multi-threading.

4.1 Single event performance of core track finding

To assess performance of the track finding algorithm alone, we run a dedicated benchmark measuring the time of central track finding loop for processing 20 events, without measuring the time needed to pre-process the hits and seeds, and to post-process the track candidates. This allows us to focus on the most relevant part of our code and to sideline the more administrative tasks that might, in a production system, be performed outside of mkFit itself.

First, we show the speedup as function of the Matriplex width which effectively controls how many slots in the vector registers are used. The results are shown in figure 3. For SNB the obtained AVX vectorization speed up is 2.4. On KNL, with AVX-512 and usage of auto-generated intrinsics code for Matriplex operations, we observe a speedup of 3.3 (3.1 for icc auto vectorization). Assuming effective vectorization speedup of 3 and applying Amdahl’s law one finds that about 72% of our code gets executed as vector instructions.

Notice that the speedup when shifting from Matriplex width of 1 to width of 2 is consistently smaller than for larger widths (and even nonexistent for KNL). This is due to scalar
instruction set being used for width 1. On KNL an additional drop is observed due to usage of AVX-512 intrinsics and the related drop in frequency.

![Figure 3](image1.png)  
**Figure 3.** Vectorization speedup as a function of used vector width for Sandy Bridge (left) and Knights Landing (right) processors. Open circle for Matriplex width of 16 for KNL is the result when Matriplex auto-generated intrinsics code is used instead of compiler assisted vectorization.

Figure 4 shows speedup as a function of number of threads TBB is configured to use. Note that events are processed sequentially and all parallelism happens within processing of seeds belonging to the same event. SNB shows good scaling up to the number of physical cores (12) and a reduced slope after that. For the large number of available cores on KNL the standard work chunk of 16 or 32 seeds needs to be reduced, leading to increased TBB overhead and poorer scaling.

![Figure 4](image2.png)  
**Figure 4.** Multi-threading speedup as a function of used number of threads for Sandy Bridge (left) and Knights Landing (right) processors.

Vectorization and multi-threading speedups for SKL-SP are shown in figure 5. Vectorization using AVX-512 vector instruction set gives a speedup of 2.75 (compared with 3.3 on KNL). This can be explained with the reduced AVX-512 base frequency on this platform. For multi-threading, the effect of turbo boost masks the real scaling behavior.

4.2 Full processing with multiple concurrent events in flight

To assess the scaling behavior of the full event processing chain as it would run in CMSSW which can process several events concurrently, we implemented support for multiple concurrent events in flight in mkFit as well. This balances out the tail effects present in event-by-event processing and allows the tasks themselves to be larger, thus reducing the overhead of
TBB. The number of events processed for each test was 20 times the number of events in flight.

Scaling behavior for multiple events in flight is shown in figure 6. Many of the administrative tasks related to pre-processing of hits and seeds have not yet been fully optimized for multi-threaded operation or to use vectorization. One can see the effect of those by comparing results for one event in flight with corresponding result in the previous section, figure 5. SNB shows good scaling with 8 events in flight. For KNL, 16 events in flight offer the best performance up to 64 threads; above that, 32 events in flight are required, but having more than 32 events in flight is not helpful, possibly due to the fact that in KNL a given memory reference can only be “owned” by 1 of 32 tiles in the layout of cores. KNL shows no gain in using more than 128 threads, i.e., hyperthreading does not yield any additional speedup. For SKL-SP (not shown) the scaling is again masked by the effects of turbo boost; 16 events in flight are sufficient to fill up the processing threads up to the maximum of 64, with some benefit observed from hyperthreading.

![Figure 5. Preliminary vectorization (left) and multi-threading (right) speedups for the Skylake Gold processor.](image)

![Figure 6. Multi-threading speedup for different number of concurrent events in flight as a function of used number of threads for Sandy Bridge (left), Knights Landing (right).](image)

### 4.3 Estimated performance of mkFit at CMS HLT

For CMS HLT Run 3 running full tracking is planned for selected events. Full tracking is also being considered to be run for all events (100 kHz rate) as this would allow better event selection, cleaner physics data sets and better utilization of available computing resources after data is already taken. Measured mkFit full-node event processing rates for the expected
LHC Run 3 pileup of 70 are 115 events/s for KNL and 250 events/s for SKL-SP. Thus, to process events at CMS HLT at the expected 100 kHz rate, one would need an equivalent of 400 32-core Skylake machines for track reconstruction alone. Note that this is below the current size of the CMS HLT cluster. This is an upper estimate with the current version of code which we believe can be further optimized. A significant increase in speed could also be achieved by not processing the seeds with estimated $p_T$ below certain threshold.

Comparing CMSSW and mkFit single-threaded performance of initial iteration tracking on KNL one finds that mkFit runs about 10-times faster than CMSSW tracking. While this indicates mkFit brings significant improvements in processing rates, this result should be deemed preliminary, especially since track post-processing has not yet been implemented in mkFit. Further, effects of usage of icc compiler and various optimization options related to vectorization and fast mathematics also need to be understood, both for CMSSW and mkFit.

5 Conclusion

Following developments required to support complex, realistic detector geometries, mkFit is now in position to demonstrate its potential for use in real-world reconstruction scenarios. Preliminary results show that mkFit exhibits physics performance on par with existing, traditional KF tracking algorithms while retaining a significant boost in computational performance. It also shows the potential to make efficient use of many-core architectures with few concurrent processes.

Ongoing work is focusing on finishing the tuning of track finding algorithm parameters and implementing the missing final post-processing of tracks. Integration with CMSSW is proceeding in parallel with the goal of early participation in the CMS HLT test-bed system for Run 3 of the LHC.

6 Acknowledgments

This work is supported by the U.S. National Science Foundation, under the grants PHY-1520969, PHY-1521042, PHY-1520942 and PHY-1624356, and by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program.

References

[1] See, for example, the presentations and proceedings of the “Connecting the Dots 2018” workshop, https://indico.cern.ch/event/658267/overview
[2] Giuseppe Cerati et al., J. Phys.: Conf. Ser. 608, 012057 (2015).
[3] Giuseppe Cerati et al., J. Phys.: Conf. Ser. 664, 072008 (2015).
[4] Giuseppe Cerati et al., EPJ Web of Conferences 127, 00010 (2016).
[5] Giuseppe Cerati et al., J. Phys.: Conf. Ser. 898, 042051 (2017).
[6] Giuseppe Cerati et al., J. Phys.: Conf. Ser. 1085, 042016 (2018).
[7] Giuseppe Cerati et al., EPJ Web of Conferences 150, 00006 (2017).
A Brief project overview & history

The project was started in 2014 with detailed investigation of the performance of vectorized code on Xeon and Xeon Phi processors\(^1\), and with the development of a matrix operation library, named Matriplex, optimized for simultaneous vectorized processing of sets of small matrices. From this basis, the initial implementation of vectorized KF fitting was demonstrated on a simplified barrel-only detector\(^2\). Summarizing the initial results, we observed a vectorization speedup of 8 on an Intel KNC co-processor when using MIC 512-bit intrinsics code for KF operations with compiler-assisted vectorization for track propagation, all in single-precision floating point arithmetic. Amdahl’s law indicates that 93% of the code was fully vectorized, so this provided an encouraging starting point for further development. At this point the code-name for the project was chosen to be mkFit– Matriplex Kalman Fitter.

The next stage was the implementation of track finding using the above technology, demonstrated on the same simplified geometry \(^3\). Two tracking algorithms were implemented, the first one simply picking the best matching hit on each layer, and the second one considering up to \(N_{\text{max}}\) track candidates for every seed, chosen by their number of hits and their \(\chi^2\) score. Simple multi-threading was implemented through segmentation of tracks in up to 21 \(\eta\) bins and usage of OpenMP parallel pragma. Physics performance was adequate (95% effic and correct \(\chi^2\) distribution of tracks and pulls of the track parameters), but the achieved parallelization speedups were a bit disappointing (x2 for vectorization and x10 for multi-threading on Intel KNC), indicating the need to decrease the fraction of non-vectorizable code and implement a better work partitioning scheme.

From that point onward, development proceeded in several directions concurrently.

1. Processing of track candidates on each layer was optimized to reduce the number of instantiations of Track objects by selecting the best hits based on their \(\chi^2\) score before doing the final Kalman updates \(^4\).

2. OpenMP was replaced by Intel Thread Building Blocks (TBB) to increase flexibility as well as to be in compliance with the CMS code base \(^5\). Further, to avoid imbalances in \(\eta\) regions and to provide more workload tasks for the many available cores, support for processing of multiple events in parallel was added. This allowed the individual tasks to remain relatively large while still being able to fill up all available hardware threads. As a result, mkFit was able to scale much better with the number of available cores with typical number of concurrent events ranging from 8 (12-core Xeon SNB) to 32 (KNC) \(^6\).

3. Significant effort has been put into porting of mkFit to run on GPGPUs using CUDA \(^7\). Fitting and track finding (best hit and optimized combinatorial versions) have been ported for the barrel-only simplified detector. Performance results for track finding were disappointing with mkFit only being able to use about 4% of the available GPU processing power. Nevertheless, Matriplex is observed to outperform standard small-matrix multiplication packages for GPUs. We are currently in the process of quantifying performance plateaus reachable for KF-like operations as a function of problem size, problem segmentation, and arithmetic intensity with the intention of identifying architectural limitations to running KF-based track finding on GPGPUs.

4. Beginning in 2015, mkFit was extended incrementally to handle realistic detector geometries with barrel and endcap sections. This required implementation of the KF and propagation equations for the endcap case, as well as a consolidated steering code that was able to handle both barrel and endcap cases. Finally, a general detector description mechanism was implemented to support arbitrary detector geometries.

\(^1\)Machine Torture: [https://github.com/osschar/mtorture](https://github.com/osschar/mtorture)
5. Early on in the project a simple triplet-based seed finding algorithm was developed. However, when we started using CMS events as input, the CMS cellular automaton seeding algorithm had already been implemented, so we decided to discontinue this development within mkFit.

6. Support for new architectures was added as they became available. We started with Intel Sandy Bridge (AVX) and Intel Knights Corner (KNC) (MIC 512-bit vector instructions). Later, we also added support for Intel Knights Landing (AVX-512) which only needed to be modified slightly to also support the Intel Skylake Scalable Performance processors. We have recently dropped support for KNC as it became clear mkFit will never actually run on this architecture in production.

   We are using recent versions of icc and gcc to build our software. C++14 language support is required to build mkFit.

7. A validation and benchmarking suite has been developed to monitor and improve physics and computational performance and to identify, helping to spot issues that require further attention. The validation and benchmarking suite is run for every code change and results are stored for later reference.

8. Recently, work has started on integration of mkFit into CMS software framework for easy testing and integration with other software expected to run at the CMS HLT during the Run 3 of the LHC.

   Currently, mkFit is able to run on CMS-2017 geometry with reasonable physics and computational performance. Ongoing work is focusing on improving the physics performance through fine-tuning of hit and track selection algorithms. Post-processing of found tracks and duplicate track removal still needs to be implemented or may be delegated to algorithms in CMS software.
B Overview of mkFit track finding algorithm

1. Hit preprocessing: for each layer sort them by $\varphi$ into $\eta$ bins compatible with module dimension along $z$ for barrel and $r$ in endcap.

2. Seed preprocessing: sort seeds by $\eta$, determine tracking region. Fit seeds if needed.

3. parallel_for over $\eta$ regions:
   - parallel_for over groups of seeds (16 or 32):
     - for layer in list of layers for this tracking region:
       Contents of this loop are vectorized using Matriplex (KF operations) and compiler auto-vectorization (propagation).
       i. Propagate candidates to layer centroid.
       ii. Select hits to be considered by each candidate. At the same time check if the candidate is within the sensitive region or close to its edge; if so, mark it accordingly to properly account for missed / skipped layers.
       iii. Calculate $\chi^2$ for every candidate-hit pair; candidate needs to be propagated to $r$ or $z$ coordinate of the hit.
       iv. Select the best $N_{max}$ candidates for every seed for further processing.
       v. Perform Kalman update on selected candidate-hit pairs.

4. Optionally perform backward fit to the first hit or the point of closest approach.