Reconstruction of Charged Particle Tracks in Realistic Detector Geometry Using a Vectorized and Parallelized Kalman Filter Algorithm

Giuseppe Cerati²,*; Peter Elmer³; Brian Gravelle⁵; Matti Kortelainen²; Vyacheslav Krutelyov⁴; Steven Lantz¹; Mario Masciovecchio⁴; Kevin McDermott¹; Boyana Norris⁵; Allison Reinsvold Hall²; Michael Reid¹; Daniel Riley¹; Matevž Tadel⁴; Peter Wittich¹; Bei Wang³; Frank Würthwein⁴; and Avraham Yagil⁴

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

Abstract. One of the most computationally challenging problems expected for the High-Luminosity Large Hadron Collider (HL-LHC) is finding and fitting particle tracks during event reconstruction. Algorithms used at the LHC today rely on Kalman filtering, which builds physical trajectories incrementally while incorporating material effects and error estimation. Recognizing the need for faster computational throughput, we have adapted Kalman-filter-based methods for highly parallel, many-core SIMD and SIMT architectures that are now prevalent in high-performance hardware. Previously we observed significant parallel speedups, with physics performance comparable to CMS standard tracking, on Intel Xeon, Intel Xeon Phi, and (to a limited extent) NVIDIA GPUs. While early tests were based on artificial events occurring inside an idealized barrel detector, we showed subsequently that our mkFit software builds tracks successfully from complex simulated events (including detector pileup) occurring inside a geometrically accurate representation of the CMS-2017 tracker. Here, we report on advances in both the computational and physics performance of mkFit, as well as progress toward integration with CMS production software. Recently we have improved the overall efficiency of the algorithm by preserving short track candidates at a relatively early stage rather than attempting to extend them over many layers. Moreover, mkFit formerly produced an excess of duplicate tracks; these are now explicitly removed in an additional processing step. We demonstrate that with these enhancements, mkFit becomes a suitable choice for the first iteration of CMS tracking, and eventually for later iterations as well. We plan to test this capability in the CMS High Level Trigger during Run 3 of the LHC, with an ultimate goal of using it in both the CMS HLT and offline reconstruction for the HL-LHC CMS tracker.

*e-mail: cerati@fnal.gov

This manuscript has been authored by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics.
1 Introduction

The reconstruction of charged particle tracks (tracking) is crucial for the physics goals of the Large Hadron Collider (LHC) experiments as it is needed to estimate the particle momenta, identify the particle type, tag the flavor of hadron jets, and improve the resolution of both jet energy and missing transverse momentum. Tracking is typically the most time consuming reconstruction task and its time scales poorly with the detector occupancy; this is a problem as the LHC luminosity increases and leads to a larger and larger number of proton interactions per beam crossing (pile-up, or PU). The challenge will be even greater at the High Luminosity LHC and especially at the High Level Trigger (HLT), an accurate measurement is needed quickly so that the most interesting events are stored for further processing. The time budget constraint needs to be addressed either by reducing the tracking phase space, with clear negative consequences on the experiments’ physics reach, or by speeding up the tracking algorithms.

In the last 10-15 years, the continuous exponential increase in the number of transistors per processor did not lead anymore to an exponential increase in clock frequency; therefore the needed speedup will not come “for free” and will require a substantial rework of the algorithms. For modern processors, most of the overall performance increase (in terms of overall floating point operations per second) comes from parallelizing over a large number of logical cores, while more moderate single-thread performance improvements are due to vector or SIMD (single instruction multiple data) operations. In light of these trends, algorithms should be reworked such that parallelism at both the data and instruction levels can be exploited.

The mission of the mkFit project\(^1\) is to speed up Kalman filter (KF) tracking algorithms [1] using highly parallel architectures. KF is widely used for tracking because the high efficiency of its physics performance is well understood and it includes a robust handling of material effects. The KF is a two-step iterative process: the track state (parameters and their uncertainties) are propagated from layer N-1 to layer N; then, the track state is updated using the position estimate provided by a detector measurement (hit). The two KF-based steps of tracking are track building and track fitting. Track building is a combinatorial search of the hits belonging to a given track; typically it is initiated from a proto-track (seed) defined in the innermost detector layers, and proceeds outwards. Fitting simply applies the KF procedure to all hits identified during building in order to extract the best estimate of the track parameters. Track building takes the largest fraction of time and it is not straightforward to parallelize efficiently due to the branching required by the combinatorial exploration of many possible track candidates, and due to the work imbalance caused by the variability inherent in the data (different number of hits per track and tracks per event). Further computing challenges are posed by the many operations with small-size matrices leading to low arithmetic intensity, and the large number of hits per event (order of 100k at PU=70).

2 Algorithm Overview

Our implementation of the KF tracking algorithm is described in detail elsewhere [3–12] and it has not fundamentally changed, so only its main features will be summarized here.

KF operations are implemented using the Matriplex library, a matrix-major representation that allows for SIMD processing of track candidates; Matriplex auto-generates vectorized code that is aware of the matrix sparsity. The algorithm is multithreaded using TBB tasks, where independent tasks are created at multiple levels: for different events, different detector regions, and for different seeds (grouped in bunches). Due to the limited bandwidth and cache

\(^1\)http://trackreco.github.io/
size of parallel architectures, we implemented a lightweight description of detector in terms of geometry, material, magnetic field; for instance, the geometry representation collapses barrel (forward) layers at average r (z), and has no knowledge of detector modules so it relies on the 3D position of hits. Within the combinatorial part of the algorithm we minimize memory operations both in their number and in their size: this requires compact data formats and a bookkeeping of the explored track candidates, so that only the best ranking ones are cloned at each layer (with a per-seed cap).

The standalone implementation of our algorithm has been tested on a 32-core system with dual Intel Xeon Gold 6130 processors (Skylake, SKL), see Fig. 1. Hyperthreading is enabled, but Turbo Boost is disabled to emphasize the scaling behavior of the code over that of the hardware. For Gold 6130, this means the clock frequency is fixed at 2.1 GHz for all cores, though it drops by 10% if more than 11 threads are active per processor. The core of the track building algorithm achieves nearly 3x speedup from vectorization when using a vector size of 16 floats. Ahmdal’s law implies that 60-70% of core algorithm code is effectively vectorized.

We note that the full track building application includes data preparation and cleaning steps for each event, and these steps are not vectorizable. Using multithreading, we achieve further speedups of more than a factor of 30 compared to the single-threaded, vectorized execution of the full application. The multithreaded scaling is close to ideal when the number of threads does not exceed the number of available physical cores (32), and all threads are dedicated to different events (which is equivalent to a multi-process execution over multiple events).

3 **Integration in CMSSW**

The deployment of our code in the framework of the CMS experiment, CMSSW, has recently been the main focus of the group. mkFit is currently integrated in CMSSW as an external package, pulled from a public repository, compiled, and distributed along with CMSSW releases. This is a major achievement that makes our work available to the whole collaboration. Two aspects are not ideal in this first integration: first, within a central CMSSW release mkFit is compiled with gcc using the core2 instruction set (limiting vectors to 128 bits), and second, dedicated steps are used to convert CMSSW data formats to and from the mkFit ones.
The CMS tracking [2] is structured in more than ten iterations, where each iteration performs the same steps: track seeding, building, fitting, and selection. Hits used in tracks reconstructed with high quality are masked to reduce the combinatorics of the next iteration. As a first milestone, our focus has been the track building for the initial iteration, based on seeds from the four innermost pixel layers and targeting prompt tracks. However, there is nothing specific to the first iteration, so the application of mkFit could be extended to other iterations as well.

The integration in CMSSW gave access to the central validation tools, which revealed a subpopulation in the initial iteration where the mkFit efficiency was suffering: short tracks. After updating the logic to count the number of missing hits in a track in a consistent way, and after re-tuning the score used to decide which candidates are the best ones, we recovered efficiency for short tracks so that our efficiency is on par with CMSSW across the board (Fig. 2). Although the mkFit performance in terms of fake and duplicate rate is currently at an acceptable level, some more work is still needed to bring them down to the CMSSW level. Another feature that we plan to implement in mkFit is the recovery of hits in overlapping detector modules, so that more than one hit per layer can be assigned to tracks.

The initial iteration time for a single-threaded execution of the CMSSW tracking on SKL has been evaluated using the CMSSW default track building as well as mkFit (Fig. 3); the sample used consists of $\bar{\nu}\nu$ events with an average PU of 50. We observe a speedup of a factor 6.2 when using mkFit. It is interesting to note that when mkFit is used the track building step becomes faster than the track fit. Data format conversions between CMSSW and mkFit account for about 25% of mkFit time, so a larger speedup is possible in case the data formats are harmonized in a way that allows for the conversion step to be reduced or entirely removed. In this test mkFit is compiled with the Intel compiler icc 19.0.4.243 enabling the AVX-512 instruction set; if we were to use the default CMSSW compilation settings (gcc 7.3.1, core2) the speedup would reduce by more than a factor of two.

4 Towards an HLT implementation

While the work so far mostly focused on applying mkFit to CMS tracking with the offline configuration, the most natural environment for a first deployment of mkFit is the HLT. The
CMS HLT configuration has different challenges with respect to offline: first, in many HLT paths, tracking is run only in regions of interest around the direction of leptons or jets, and second, the silicon strip hit reconstruction is performed on-demand within the track building. On-demand hit reconstruction means that the raw data are not processed upfront for all modules in the silicon strip detector; instead, when a track candidate first searches for hits on a given module, the module raw data are processed and then cached for subsequent access. The on-demand option is currently not supported within \texttt{mkFit}, so all hits need to be available upfront. With the tools currently available in \texttt{CMSSW}, the global strip hit reconstruction chain is costly, so we need to investigate a faster implementation if we want to profit from the \texttt{mkFit} speedups at HLT. Our goal is to start from raw strip detector data and output hits in the \texttt{mkFit} data format; we also require that the new implementation be compatible with GPU execution. The steps to be performed in the strip processing are: raw data unpacking and remapping, strip data calibration, strip data clustering, and evaluation of the 3D hit position. Here we report on a preliminary implementation of the clustering algorithm.

In its current version the algorithm mimics the \texttt{CMSSW} implementation and works as follows: first, seed strips with ADC values greater than 3 times the expected noise are identified; second, clusters are formed adding to the seed all strips with ADC values above twice the expected noise that are either consecutive or have a small gap (the size of the allowed gap depends on the presence of bad strips in the module); finally, the cluster is required to have a quadrature sum of the strip ADCs larger than 5 times the quadrature sum of the expected noise, and to have a minimum total charge. We developed a first implementation both for CPU (C++ code parallelized using \texttt{OpenMP}) and for GPU (using \texttt{CUDA}). Preliminary tests on single events show that overheads currently account for the dominant fraction (>90%) of the time for the GPU execution, where overheads include data transfer and memory allocation. Work is ongoing towards an improved implementation that reduces the overheads by processing multiple events concurrently, thus allowing for asynchronous memory transfer, and by using a memory pool to pay the allocation overhead only at the beginning and end of job.
5 Studies Towards a Portable Implementation

Given that the processor market, HPC centers, and the experiments’ online farms are all trending towards an increase in the utilization of GPUs, the exploration of GPU-compatible versions of our algorithm is becoming a priority. A portable solution would guarantee that the code is maintainable, with minimal differences between CPU and GPU versions, even if such portability may require trade-offs in terms of performance.

We started a collaboration with the RAPIDS\textsuperscript{2} group at Oak Ridge National Laboratory to explore usage of compiler directives for portability between CPU and GPU. First developments in this respect include a version of \texttt{mkFit} where threads are managed with \texttt{OpenMP} (working only on CPU for now), and testing \texttt{OpenACC} for a single function. There is not a dominant function, taking most of the compute time in our code, so the choice of which function to study was somewhat arbitrary. We chose the "PropagationToZ" function as it is one of the top contributors but it is simple enough to be an ideal candidate for exploration tests. We compared a serial version with various parallel versions, and found an \texttt{OpenMP} CPU version to be 23.9 times faster than the serial, and an \texttt{OpenACC} version to be 207.1 (375.1) times faster than the serial CPU version on a single GPU when accounting for (excluding) the data transfer. Tests were performed on a Summit supercomputer node (dual IBM POWER9 CPUs, each with 22 physical cores, plus 6 NVIDIA Volta V100 GPUs). While these results are encouraging, there are significant challenges ahead, including the management of data transfers, and the interface with \texttt{CMSSW}.

6 Conclusions

In summary, the \texttt{mkFit} code has been integrated in \texttt{CMSSW} as an external library, and shows physics performance in terms of efficiency that is on par with the current \texttt{CMSSW} tracking algorithm, while running more than 6 times faster when \texttt{mkFit} is compiled with \texttt{icc} and AVX-512. The utilization of GPUs is being explored at different stages, including strip hit reconstruction for the HLT application and elements of a portable implementation of the track building algorithm.

7 Acknowledgements

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 and Office of High Energy Physics, Scientific Discovery through Advanced Computing (SciDAC) program.

References

[1] Rudolf Fruhwirt, Nucl. Instrum. Meth. A262, 440–450 (1987)
[2] S. Chatrchyan et al. [CMS Collaboration], JINST 9, no. 10, P10009 (2014)
[3] Giuseppe Cerati et al., J. Phys.: Conf. Ser. 608, 012057 (2015)
[4] Giuseppe Cerati et al., J. Phys.: Conf. Ser. 664, 072008 (2015)
[5] Giuseppe Cerati et al., doi:10.1109/NSSMIC.2015.7581932
[6] Giuseppe Cerati et al., EPJ Web of Conferences 127, 00010 (2016)

\textsuperscript{2}SciDAC Institute for Resource and Application Productivity through computation, Information, and Data Science. SciDAC is the Scientific Discovery through Advanced Computing program.
[7] Giuseppe Cerati et al., J. Phys.: Conf. Ser. 898, 042051 (2017)
[8] Giuseppe Cerati et al., EPJ Web of Conferences 150, 00006 (2017)
[9] Giuseppe Cerati et al., J. Phys.: Conf. Ser. 1085, 042016 (2018)
[10] Giuseppe Cerati et al., EPJ Web Conf. 214, 02002 (2019)
[11] Giuseppe Cerati et al., arXiv:1906.02253 [physics.ins-det]
[12] Giuseppe Cerati et al., arXiv:1906.11744 [physics.ins-det]