Proceedings of the CTD 2022
PROC-CTD2022-09
March 11, 2023

ACTS GPU Track Reconstruction Demonstrator for HEP

Paul Gessinger[1], Hadrien Grasland[2], Heather Gray[3,4], Sylvain Joube[2], Konrad Kusiak[1], Attila Krasznahorkay[1], Charles Leggett[3], Georgiana Mania[5,6], Joana Niermann[1,7], Andreas Salzburger[1], Nicholas Styles[5], Stephen Nicholas Swatman[1,8], Beomki Yeo[3,4]

[1] *CERN, 1211, Geneva, Switzerland*

[2] *Université Paris-Saclay, CNRS/IN2P3, IJCLab, 91405, Orsay, France*

[3] *Lawrence Berkeley National Laboratory, CA 94720, Berkeley, USA*

[4] *Department of Physics, University of California, CA 94720, Berkeley, USA*

[5] *Deutsches Elektronen Synchrotron, 22607, Hamburg, Germany*

[6] *Department of Informatics, University of Hamburg, 22527, Hamburg, Germany*

[7] *II. Physikalisches Institut, Georg-August-Universität Göttingen, 37077, Göttingen, Germany*

[8] *University of Amsterdam, 1012WX, Amsterdam, The Netherlands*

ABSTRACT

Track reconstruction speed in high-energy physics experiments becomes more important with increasing accelerator luminosity, data volumes, and measurement densities. Heterogeneous computing with GPU is a promising way to accelerate the analysis considering that the track reconstruction is parallelizable. Therefore, the ACTS community initiated R&D projects to develop the GPU track reconstruction demonstrator. In this work, we introduce its recent progress in tracking algorithm implementation and realistic detector setup.

PRESENTED AT

Connecting the Dots Workshop (CTD 2022)
May 31 - June 2, 2022

1 Introduction

A Common Tracking Software (ACTS) [1, 2] is an open source track reconstruction toolkit for high energy physics (HEP) experiments written in the C++ language. The project serves as an analysis toolkit for various experiments, as well as an R&D platform for the applications of GPGPUs [3] in track reconstruction. The massively parallel nature of GPGPUs has great potential to accelerate the data analysis of future HEP experiments with larger data size. With that prospect, pilot studies [4] have been conducted by the ACTS community to offload several track reconstruction algorithms to GPU architectures. Although the first benchmark results are encouraging, embedding GPU algorithms in existing track reconstruction pipelines remains an open problem because run-time polymorphism and dynamic allocation of memory massively used in ACTS is not GPU-friendly.

In an effort to resolve the aforementioned issues, the ACTS community has developed a novel GPU track reconstruction demonstrator for both offline and online [5] analysis. In this work, we present the recent progress in the GPU demonstrator development: Section 2 introduces the R&D libraries of the ACTS GPU demonstrator. Section 3 highlights the currently implemented algorithms and their performance. Section 4 describes a compile-time polymorphic tracking geometry and its track propagation tool. A summary follows in Section 5.

2 R&D Libraries for the GPU demonstrator

The ACTS GPU demonstrator follows an experiment-independent design while aiming to match the physics performance of existing CPU-based algorithms with a realistic detector setup. The demonstrator should also support a variety of heterogeneous programming platforms; CUDA [6] and SYCL [7] are currently the primary targets. We support both the mature but vendor-specific CUDA platform and the more recent and widely applicable SYCL platform. The ACTS R&D ecosystem in which these points are investigated is shown in Figure 1.

2.1 `vecmem`

`vecmem` [8, 9] is a memory management library with a user-friendly interface for a variety of programming platforms that include CUDA and SYCL. In host code, it utilizes polymorphic memory resources [10] to allocate memory through STL-like containers. It also supports STL-like containers on the device to provide a similar user interface between host code and device code. Caching allocation exists as well where the memory allocation time can be saved by reusing the previously allocated memory.

2.2 `algebra-plugins`

`algebra-plugins` [11] is a library that provides different implementations of vector and matrix algebra operations necessary for track reconstruction algorithms. It supports various backends such as `cmath` (custom implementation), `Eigen` [12] and `SMatrix` [13]. The development of custom implementations of cmath backend is motivated by the need of having native support for CUDA and SYCL. Users can determine which backend and floating-point precision to use at compile-time.

2.3 `covfie`

`covfie` [14] is a library for general vector field calculation, which supports a wide range of possible vector fields, including the sampled inhomogeneous magnetic fields found in HEP experiments. Similar to the design of `algebra-plugins`, the GPU APIs, vector field dimension, and interpolation algorithm can be determined at compile-time.

2.4 detray

detray [15, 16] is a tracking geometry library with a GPU-friendly design that relies on static compile-time polymorphism. It also supports the track propagation functionality required for track reconstruction algorithms. Components of the tracking geometry are stored in vecmem containers which are compatible with both host-side execution and device-side execution. covfie is integrated into detray for the track propagation with an inhomogeneous magnetic field.

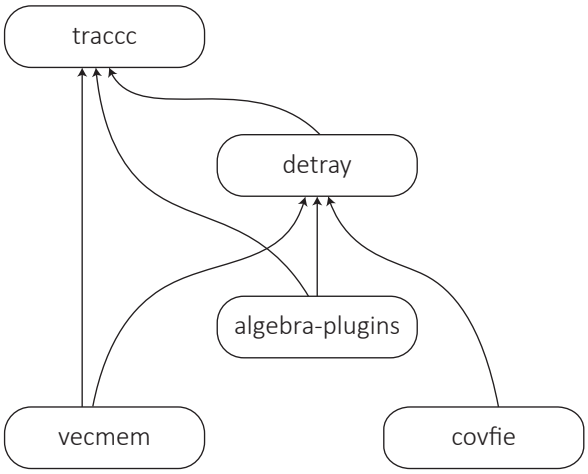


Figure 1: ACTS GPU R&D projects. The dependencies between the projects are represented by arrows.

3 Tracking Algorithm Implementations

traccc [17] makes use of all other libraries introduced in Section 2 to perform the track reconstruction on GPU. It covers track reconstruction algorithms from the raw data processing to the precise fitting of charged particle trajectories as explained in Ref [1] and illustrated in Figure 2. The rest of this section will describe the implementation and performance of the cell clusterization and seeding on GPU, while Section 4 is dedicated to detray tracking geometry which is an essential step towards the implementation of a parallel (Combinatorial) Kalman Filtering [18, 19, 20, 21] on GPU.

3.1 GPU Implementation

One of the challenges in implementing GPU algorithms is avoiding the dynamic allocation of memory. This means that the size of output vector containers must be known before executing the kernel. An intuitive solution for this would be allocating the vector size as large as a theoretical limit. For example, if there are N hits in an event, the number of clusters should be less than or equal to N, where N is the theoretical limit. Clearly, this approach will waste a significant amount of memory, therefore, it was not considered a plausible option. As an alternative, a vector size can be estimated by investigating the number of output objects as a function of the number of the input objects. However, it is not a convenient solution because the hard coded sizes need to be updated whenever there is a change in the detector setup. The adopted solution is to run two kernels instead of one: a counting kernel and a populating kernel. The counting kernel estimates the number of output objects after which a suitably large output vector can be allocated on the host, and the populating function fills the output vector. Even though this comes at the cost of running some functions twice and the counter variables need to be reset to zero after the populating kernel, this method was selected for the purpose of code maintainability and to optimize memory operations.

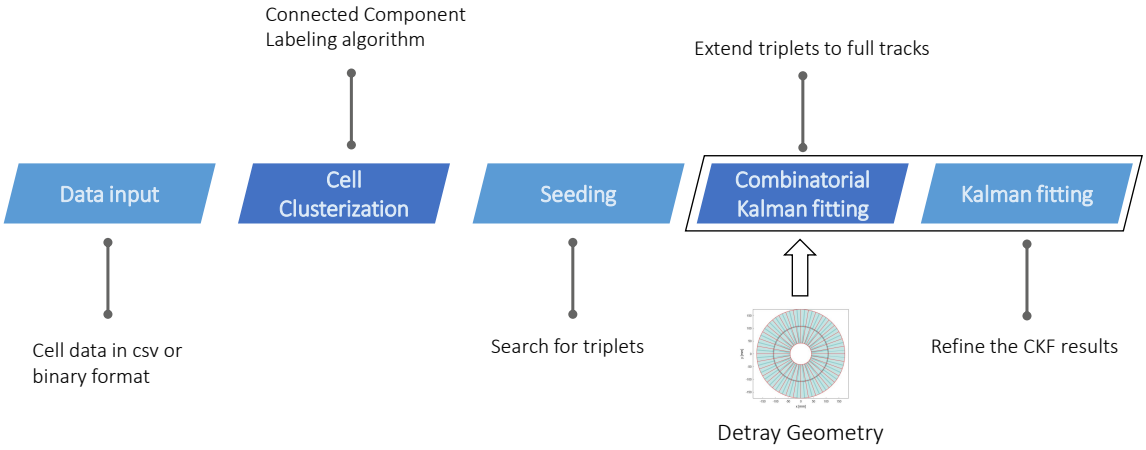
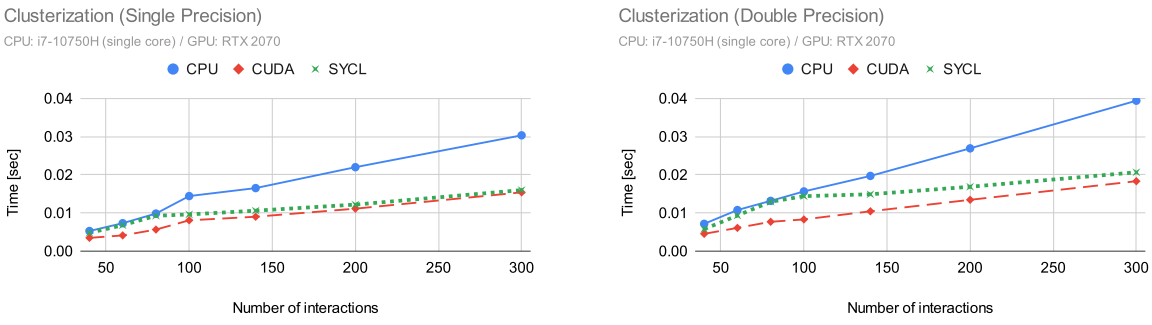


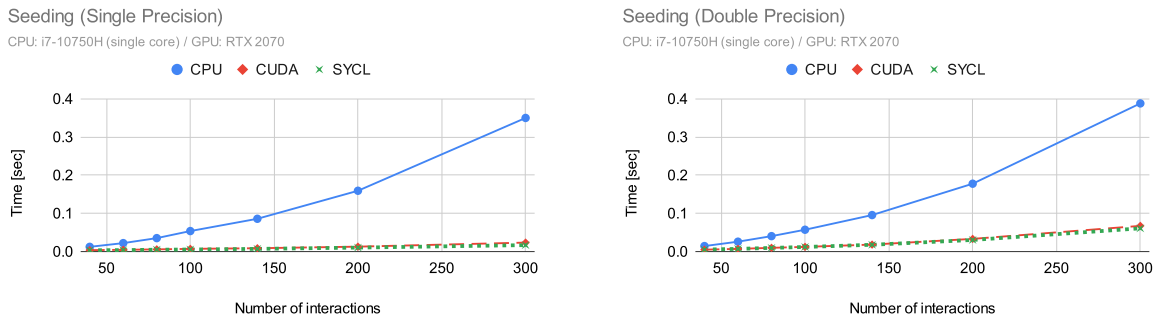
Figure 2: The track reconstruction chain of `traccc`.

Figure 3 and 4 show the time measurement [22] for the cell clusterization with the SparseCCL algorithm [23, 24] and seeding as a function of the number of $t\bar{t}$ interactions per event, respectively. The computing environment was an Intel i7-10750H (2.6 GHz base frequency) CPU and an NVIDIA RTX 2070 GPU. The algorithms were benchmarked in both single-precision and double-precision floating point mode with the `cmath` backend of `algebra-plugins`. The input data [25] was prepared by ACTS simulation with $t\bar{t}$ interactions (14 TeV center-of-mass energy collision) in the trackML detector [26] which conceptually follows the upgraded design of ATLAS and CMS detectors [27, 28]. There was no notable speedup gain from the GPU over the single-core CPU implementation for the cell clusterization algorithm which we believe to be due to imbalance in the workloads between threads, which could be further optimized. In case of the seeding algorithm, both the CUDA and SYCL implementations achieved a speedup of roughly one order of magnitude over the single-core CPU implementation for single precision while the speedup dropped to a factor five for double precision. The speedup drop in seeding with double precision is expected as the GPU hardware has fewer double-precision arithmetic units. The same effect was not seen in the cell clusterization as it has much less floating point arithmetic compared to the seeding.



(a) (b)

Figure 3: (Preliminary) Cell clusterization time averaged over 10 events as a function of the number of $t\bar{t}$ interactions per event with 14 TeV center-of-mass energy collision. (a) The left figure is measured for single precision, and (b) the right one is measured for double precision. SYCL uses the CUDA backend for GPU measurement.



(a) (b)

Figure 4: (Preliminary) Seeding time averaged over 10 events as a function of the number of $t\bar{t}$ interactions per event with 14 TeV center-of-mass energy collision. (a) The left figure is measured for single precision, and (b) the right one is measured for double precision. SYCL uses the CUDA backend for GPU measurement.

4 Towards a Parallel CKF: Tracking Geometry

A tracking geometry is necessary for the implementation of the CKF method because the algorithm needs to look up the measurements of a detector surface encountered during the track propagation. In this section, we explain a design of such a tracking geometry and its track propagation functionality with an inhomogeneous magnetic field interpolation.

4.1 Detector Description

The `detray` tracking geometry consists of several component types which are inter-linked through index variables. A surface stores the indices of masks, which describe the shape and dimensions, and an affine transform matrix which carries the rotation and translation information. A volume represents a section of the detector as an aggregation of surfaces, including its own boundary (virtual) surfaces called portals and physical surfaces.

To facilitate host-server communication, the tracking geometry is implemented using static polymorphism. Masks of different types are stored in a tuple, and the required mask types are declared at compile-time in detector specific metadata. Since it is not possible to access tuple elements with run-time index variables, the tuple container of masks is unrolled using variadic templates to retrieve the specific mask that is referenced by a particular surface in the tracking geometry. With the shape information being pushed entirely to the lightweight mask objects, the rest of the data, i.e. the transform matrices, surfaces and volumes can be stored in standard vector containers due to their fixed types.

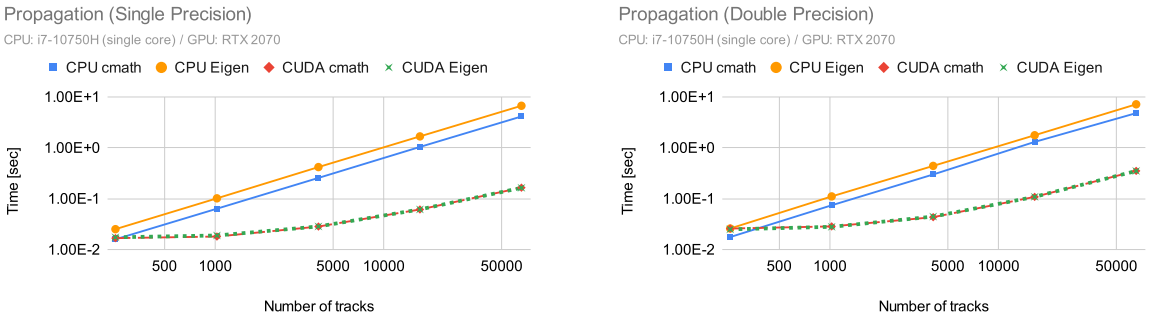
The tracking geometry is built inside the host code and passed over to the device if required: the host geometry transferred via an intermediate view type object constructs the corresponding device geometry inside the kernel function.

4.2 Track Propagation

In `detray` track propagation, track parameters of charged particles are advanced using a fourth order Runge-Kutta-Nyström (RKN) method [29]. The step size of the RKN method is dynamically adjusted by the local error expected from the gradient of the magnetic field while it is limited by the distance to the closest surface. A straight-line ray is shot to find candidate surfaces, which the track might intersect, and the corresponding distances to the surfaces. The distance to the closest surface is updated for every step, and the list of candidates is renewed when the track enters a new volume through a portal.

For the GPU implementation, one track was mapped to each thread of a kernel function because the tracks can be propagated independently from each other during the propagation. Figure 5 shows the time

measurement [30] of propagation in the trackML detector in a constant 2 T magnetic field. Tracks of 10 GeV were generated at the origin of the detector and shot in isotropic direction. Similarly to the seeding algorithm, the CUDA implementation showed an order of magnitude of speedup over a single CPU core, while the `cmath` and `Eigen` backends did not show significant difference in performance one from each other, regardless of the hardware.



(a) (b)

Figure 5: (Preliminary) Propagation time in the pixel section of the trackML detector as a function of the number of tracks in log-log scale, estimated for (a) single and (b) double precision, respectively. Tracks of 10 GeV were generated in isotropic direction in a homogeneous magnetic field of 2 T.

4.3 Magnetic Fieldmap Interpolation

In most HEP experiments, the magnetic field is inhomogeneous throughout the detector. Therefore, it is common for such magnetic fields to be described by a set of field vectors at discrete, uniformly sampled points. In lieu of a continuous field map, values must be interpolated from nearby sample points. `covfie` supports a trilinear interpolation method [31], implemented both in software as well as in hardware through the use of GPU texture-fetching units. The integration of these magnetic field accessors into the `detray` track propagation is still under development.

5 Summary

The ACTS community has initiated R&D projects to run tracking algorithms in GPUs. `vecmem` is a memory management tool developed in order to provide a convenient interface to the GPU APIs. `algebra-plugins` provides vector and matrix operations necessary for track reconstruction. In order to allow for a realistic detector descriptions, `detray` and `covfie` have been set up to provide track propagation through tracking geometries and interpolation of inhomogeneous magnetic fields, respectively. `traccc` integrates these libraries for track reconstruction and we have demonstrated clusterization and seeding algorithms using CUDA and SYCL. While the first benchmark results on track reconstruction and propagation are promising, the overall performance needs to be optimized with the `vecmem` caching allocation and multi threading method for apple to apple comparison between CPU and GPU. Ongoing developments include the implementation of (Combinatorial) Kalman filtering using the `detray` track propagation.

ACKNOWLEDGEMENTS

This work was supported in part by the National Science Foundation under Cooperative Agreement OAC-1836650. This research was supported in part by the U.S. Department of Energy's Office of Science, Office of

High Energy Physics, of the US Department of Energy under Contract No. KA2102021. We acknowledge the support by DASHH (Data Science in Hamburg - HELMHOLTZ Graduate School for the Structure of Matter) with the Grant-No. HIDSS-0002. This work has been sponsored by the Wolfgang Gentner Programme of the German Federal Ministry of Education and Research (grant no. 13E18CHA).

References

[1] X. Ai, et al., A Common Tracking Software Project, Comput Softw Big Sci **6**, 8 (2022).

[2] ACTS developers, ACTS, GitHub repository (2022), https://github.com/acts-project/acts.

[3] D. Luebke, et al., GPGPU: general-purpose computation on graphics hardware, Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, 208–es (2006).

[4] X. Ai, G. Mania, H. M. Gray, N. Styles, A GPU-Based Kalman Filter for Track Fitting, Comput Softw Big Sci **5**, 20 (2021).

[5] ATLAS Collaboration, Technical Design Report for the Phase-II Upgrade of the ATLAS Trigger and Data Acquisition System - Event Filter Tracking Amendment, CERN-LHCC-2022-004

[6] J. Nickolls, I. Buck, M. Garland, K. Skadron, Scalable Parallel Programming with CUDA, 2008 IEEE Hot Chips 20 Symposium, 1-2 (2008)

[7] R. Reyes, V. Lomüller, SYCL: Single-source C++ accelerator programming, Parallel Computing: On the Road to Exascale, IOS Press (2016).

[8] ACTS developers, vecmem, GitHub repository (2022), https://github.com/acts-project/vecmem.

[9] A. Krasznahorkay, P. Gessinger, S. N. Swatman, Managing Heterogeneous Device Memory using C++17 Memory Resources [Conference presentation], ACAT 2021.

[10] P. Halpern, Polymorphic memory resources, C++Standards Committee Working Group ISOCPP (2022).

[11] ACTS developers, algebra-plugins GitHub repository (2022), https://github.com/acts-project/algebra-plugins.

[12] G. Guennebaud, et al., Eigen v3 (2010), http://eigen.tuxfamily.org.

[13] T. Glebe, SMatrix - A high performance library for Vector/Matrix calculation and Vertexing, HERA-B Software Note, 01-134 (2003).

[14] ACTS developers, covfie, GitHub repository (2022), https://github.com/acts-project/covfie.

[15] ACTS developers, detray, GitHub repository (2022), https://github.com/acts-project/detray.

[16] A. Salzburger, B. Yeo, J. Niermann, A. Krasznahorkay, detray - A compile-time polymorphic tracking geometry description [Conference presentation], ACAT 2021.

[17] ACTS developers, traccc, GitHub repository (2022), https://github.com/acts-project/traccc.

[18] R. E. Kálmán, A New Approach to Linear Filtering and Prediction Problems, Journal of Fluids Engineering **82**, 35–45 (1960).

[19] R.Frühwirth, Application of Kalman filtering to track and vertex fitting, Nucl. Instrum. Methods. Phys. Res. A **262**, 444-450 (1987).

[20] P. Billoir, Progressive track recognition with a Kalman-like fitting procedure, Comput. Phys. Commun. **57**, 390–394 (1989).

[21] P. Billoir, S. Qian, Simultaneous pattern recognition and track fitting by the Kalman filtering method, Nucl. Instrum. Methods. Phys. Res. A **294**, 219–228 (1990).

[22] ACTS developers, traccc v0.1.0, GitHub repository (2022), https://github.com/acts-project/traccc/tree/v0.1.0.

[23] L. He, et al., The connected-component labeling problem: A review of state-of-the-art algorithms, Pattern Recognition **70**, 25-43 (2017).

[24] A. Hennequin, B. Couturier, V. V. Gligorov, L. Lacassagne, SparseCCL: Connected Components Labeling and Analysis for sparse images, 2019 Conference on Design and Architectures for Signal and Image Processing, 65-70 (2019).

[25] ACTS developers, traccc-data, GitLab repository (2022), https://gitlab.cern.ch/acts/traccc-data.

[26] S. Amrouche, et al., The Tracking Machine Learning Challenge: Accuracy Phase, The Springer Series on Challenges in Machine Learning (2019).

[27] ATLAS Collaboration, Technical Design Report for the ATLAS Inner Tracker Pixel Detector, Tech. Rep. ATLAS-TDR-030 (2017).

[28] CMS Collaboration, The Phase-2 Upgrade of the CMS Tracker, Tech. Rep. CMS-TDR-014 (2017).

[29] E. Lund, L. Bugge, I. Gavrilenko, A. Strandlie, Track parameter propagation through the application of a new adaptive Runge-Kutta-Nyström method in the ATLAS experiment, J. Instrum. **4**, P04001–P04001 (2009)

[30] ACTS developers, detray v0.10.0, GitHub repository (2022), https://github.com/acts-project/detray/tree/v0.10.0.

[31] D. Lin, L. Seiler, C. Yuksel, Hardware Adaptive High-Order Interpolation for Real-Time Graphics, Computer Graphics Forum **4**, 1-16 (2021)