EPJ Web of Conferences 337, 01092 (2025) https://doi.org/10.1051/epjconf/202533701092
CHEP 2024

Application of traccc to the CEPC Vertex Detector

Yizhou Zhang[1,2,*], *Weidong* Li[1,2,**], *Xiaocong* Ai[3], *Tao* Lin[1], *Jiaheng* Zou[1], *Teng* Li[4], and *Xingtao* Huang[4]

[1]Institute of High Energy Physics, Chinese Academy of Sciences, Beijing 100049, China
[2]University of Chinese Academy of Sciences, Beijing 100049, China
[3]Zhengzhou University, Zhengzhou, Henan 450001, China
[4]Shandong University, Qingdao, Shandong 266237, China

Abstract. The Circular Electron Positron Collider (CEPC) is a future experiment mainly designed to precisely measure the Higgs boson's properties, study electroweak physics at the Z-boson peak, and search for new physics beyond the Standard Model. In the design of the CEPC detector, the Vertex Detector (VTX) is the innermost tracker playing a dominant role in determining the vertexes of a collision event. The VTX detector is also responsible for providing seeds for the track following algorithms to find tracks in the outer trackers. traccc is one of the R&D lines aiming for developing the demonstrator for a full tracking chain for accelerators within the ACTS project.
This contribution presents the implementation of the track seed-finding algorithm for the VTX detector based on traccc in the CEPC software (CEPCSW) environment. The integration of traccc with the CEPCSW, which is using Gaudi as the underlying framework and employing DD4hep as detector description tool and EDM4hep as the event data model, will be introduced. The CEPC VTX detector has three layers and both sides of each layer are mounted with silicon pixel sensors. To accommodate this specific detector structure, the default track seed-finding algorithm in traccc, which creates three-space-points seeds, has been extended for six space points case. This contribution will also describe the solution of using one common memory for both EDM4hep and VecMem to avoid the overhead from data copy. For all the above work, both physics performance and computing performance are measured and will be presented.

1 Introduction

The Circular Electron Positron Collider (CEPC) is a proposed next-generation collider designed to measure Higgs boson properties precisely, study electroweak physics at the Z-boson peak, and to probe physics beyond the Standard Model [1]. Following the completion of its Conceptual Design Report (CDR) [2] in 2018, CEPC has advanced to the Technical Design Report (TDR) phase.

As detailed in the CDR, the tracking system of the CEPC baseline detector, illustrated in Figure 1, comprises the Vertex Detector (VTX), Silicon Internal Tracker (SIT), Forward

*e-mail: zhangyz@ihep.ac.cn
**e-mail: liwd@ihep.ac.cn

Tracking Detector (FTD), Time Projection Chamber (TPC), and Silicon External Tracker (SET). The VTX, positioned at the innermost region, extends radially from 16 mm to 60 mm and longitudinally from -125 mm to 125 mm. It plays a vital role in precisely determining interaction vertices and generating high-quality track seeds for downstream reconstruction. The VTX consists of three concentric cylindrical layers, each equipped with high-resolution silicon pixel sensors mounted on both sides.

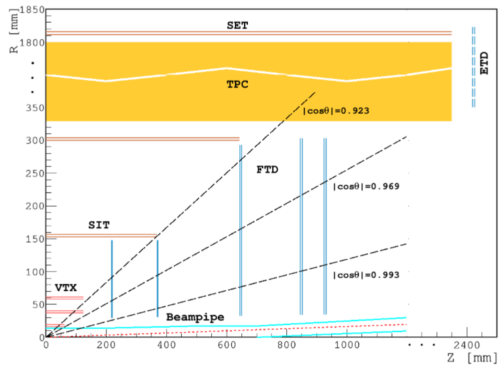

Figure 1. Schematic view of the CEPC baseline tracker system layout. (From CEPC CDR [2])

During Z-pole operation, computing demands are expected to reach the exabyte scale, presenting technical challenges comparable to those faced at the High-Luminosity Large Hadron Collider. To address this challenge, ongoing research is investigating the integration of emerging technologies, such as heterogeneous computing, into existing systems. One such initiative is the application of ACTS [3, 4] to the CEPC tracking system, which provides an experiment-independent toolkit specifically designed for charged particle reconstruction.

Within ACTS, traccc [5] focuses on optimizing algorithms for modern computing architectures, particularly Graphics Processing Units (GPUs). The algorithms are developed using both CUDA and SYCL. The SYCL programming model extends support beyond NVIDIA GPUs to include AMD and Intel CPUs and GPUs. Another innovation in traccc is its utilization of VecMem [6], a specialized library facilitating optimized data transfer through vectorized data structures that minimize transfer overhead between CPU and GPU memory while providing architecture-appropriate memory access patterns.

This contribution presents the implementation of track seed finding in the CEPC VTX based on traccc. In Subsection 2.2, we describe the integration of traccc with the CEPC software, followed by the preparation of the geometry required by ACTS in Subsection 2.1. Subsection 2.3 introduces the implementation of track seed finding, while Subsection 2.4 focuses on optimizing CPU-GPU data communication. Additionally, Subsection 2.5 covers compilation and library building. Finally, in Section 3, we evaluate the implementation's physics performance and computational efficiency.

2 Implementation of track seed-finding algorithm based on traccc

The CEPC software, CEPCSW, is built upon several foundational HEP software packages, including the Gaudi framework for event processing, EDM4hep for the event data model, and DD4hep for detector description, among others. Its overall structure consists of three layers. The Application Layer interacts with event data, enabling physicists to perform tasks

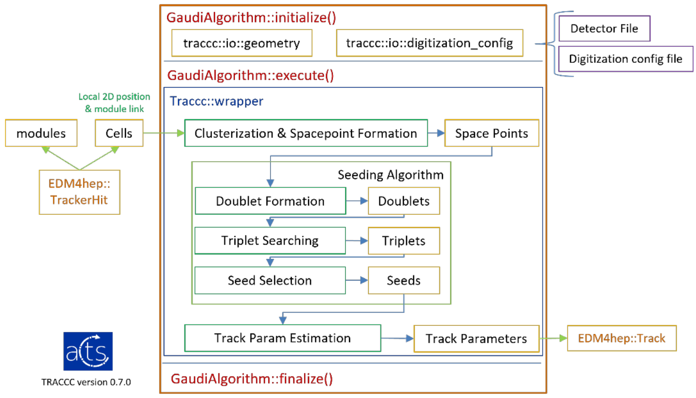

Figure 2. Schematic representation of traccc reconstruction implementation within CEPCSW Gaudi algorithms, showing data flow and processing stages

such as physics generation, simulation, event reconstruction, and data analysis using both the framework and external libraries. The Core Software Layer provides essential infrastructure and tools for building applications, while the External Library Layer integrates third-party packages and libraries into CEPCSW, ensuring full functionality.

2.1 Geometry conversion

CEPCSW employs DD4hep to describe the complete detector geometry for Geant4 simulation. However, reconstruction tasks require only a subset of this detailed geometry. Specifically, the track seed-finding algorithm necessitates only the geometric information of sensitive layers and their corresponding pixel size information. Since traccc utilizes ACTS as its geometry description tool, these geometric elements must be converted into ACTS-compatible format.

The conversion process leverages ACTS's capability to extract geometric objects from TGeo geometry files. The workflow begins with converting CEPCSW's DD4hep format geometry to TGeo format using DD4hep's `geoConverter` tool. Following this conversion, a configuration file is created to specify the sensitive layers by name and r-z coordinates. ACTS's TGeo conversion tools then transform the volume-based TGeoNode descriptions into `Acts::Surface` objects, generating a CSV file that contains the sensitive surface descriptions. Additionally, the pixel size information for each module is manually configured according to ACTS's digitization-configuration JSON file format.

While this process extracts the required geometry from CEPCSW, an issue arises from the different geometry ID definitions between ACTS and CEPCSW. To ensure accurate module identification for CEPCSW simulated hits, the geometry ID conversion between the two needs to be completed. During input hit processing, DD4hep's decoder tool extracts the Layer and Module indices from their geometry ID, which are then converted to corresponding Layer and Sensitive indices in the traccc geometry through numerical calculations. These converted indices are used to create traccc geometry identifiers.

It is worth noting that the current geometry conversion approach represents an older methodology. The traccc development team has since transitioned to detray, a compile-time polymorphic geometry description tool. Our development roadmap prioritizes completing the migration to detray's enhanced geometry framework as the next implementation phase.

2.2 Integration with the CEPCSW

As illustrated in Figure 2, complete traccc's track seed-finding algorithm is integrated into the CEPCSW framework by implementing a Gaudi algorithm, which invokes Clusterization, Spacepoint Formation, Seeding, and Track Parameter Estimation. Among these, the Seeding stage itself encompasses multiple operations, including Doublet formation, Triplet searching, and Seed selection.

The Gaudi algorithm provides three fundamental methods: `Initialize()`, `Execute()`, and `Finalize()`. In our implementation of the Gaudi algorithm, the `Initialize()` loads the detector geometry and creates the ACTS geometry in memory. During `Execute()`, the program reads the EDM4hep tracker hit objects of the current event, converts them into traccc's event data, and associates them with their respective modules through the geometry ID conversion described in Section 2.1. Subsequently, the Gaudi algorithm executes the complete track seed-finding algorithm sequence. Finally, it translates the resulting track seeds back into EDM4hep objects for downstream processing. The `Finalize()` ensures proper resource management by releasing heap-allocated memory.

In the current implementation, a single algorithm manages all track seed-finding steps, which offers the advantage of maintaining all operations within the same C++ domain, thereby facilitating seamless data exchange between different stages. As part of our future development roadmap, we plan to separate seed finding and seed parameter estimation into distinct algorithms, enabling parallel execution and potentially improving overall performance.

2.3 Track seed finding

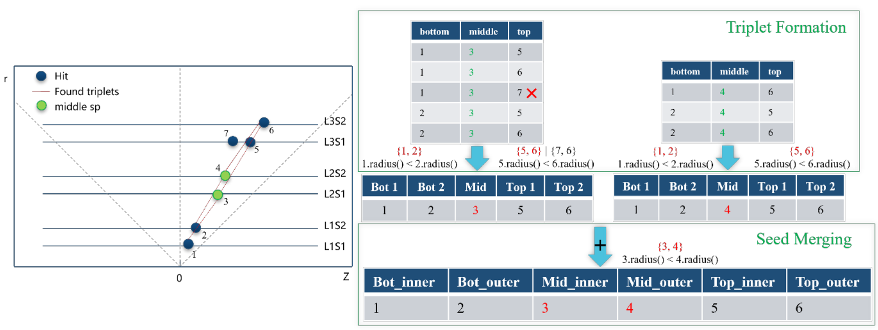

Figure 3. Schematic diagram of the extended seeding algorithm for the CEPC VTX detector. The algorithm processes six space points per particle trajectory through a two-stage approach: triplet formation followed by seed merging.

The VTX detector consists of three layers, each with two sides, typically providing six space points for a charged particle as it traverses the detector. Track seed finding begins with the identification of triplets based on a fixed middle space point, where each triplet consists of three space points with distinct coordinates. Given the minimal thickness of a VTX layer, the radial difference between two space points on opposite sides of a layer is negligible. To ensure high-quality triplets, their formation is restricted to space points from different layers. In the next stage, all identified triplets sharing the same middle space point are grouped into five-point seed candidates based on triplet parameter similarity. Finally, in the last stage, five-point seed candidates derived from different middle space points are merged to form final track seed candidates containing six space points. At any stage of this process, the search for track seeds may terminate if further progression is not possible, meaning a track seed can contain 3, 4, 5, or 6 space points.

The process of track seed finding is illustrated in Figure 3, where $L_m S_n$ denotes a space point located on the m_{th} layer (with $m \in \{1, 2, 3\}$) and the n_{th} side (with $n \in \{1, 2\}$). In the figure, space points are numbered; for example, space point 1 is located on $L_1 S_1$. During the triplet-finding stage, six triplets are identified based on the middle space point 3: (1,3,5), (1,3,6), (1,3,7), (2,3,5), (2,3,6), and (2,3,7). In the triplet assembly stage, these triplets are merged into a five-point seed candidate: (1, 2, 3, 5, 6). Similarly, based on the middle space point 4, another five-point seed candidate (1, 2, 4, 5, 6) is identified. Finally, in the combing stage, the two five-point seed candidates are combined into a final track seed candidate: (1, 2, 3, 4, 5, 6).

2.4 Optimization of CPU-GPU data communication

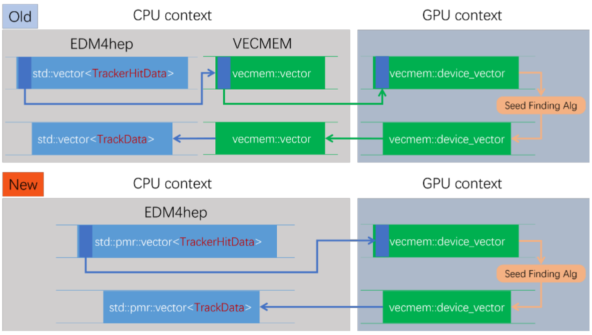

Figure 4. Modified data transfer process enabling direct memory access between EDM4hep and traccc, eliminating unnecessary data conversion steps.

EDM4hep is a standardized event data model that provides a set of predefined C++ classes for use in reconstruction software. These classes follow the Plain-Old-Data (POD) style and are generated using the PODIO code generator. PODIO employs `std::vector`<T>, the standard dynamic array provided by the C++ Standard Library, for memory management.

However, traccc utilizes vecmem to handle event data in heterogeneous computing environments. vecmem relies on `std::pmr::vector`<T>, which supports the Polymorphic Memory Resource (PMR) model. Due to the differences in memory management between PODIO and vecmem, data copying is typically required when converting EDM4hep data objects into vecmem-compatible formats.

To address this, we implemented a unified memory management approach in which both EDM4hep and VecMem (traccc's vectorized data model) share the same `std::pmr::vector` storage format, as illustrated in Figure 4. This integration allows traccc to directly access and manipulate hit data from EDM4hep without requiring redundant data copying operations, thereby eliminating the overhead associated with converting between `std::vector` and `std::pmr::vector`.

By modifying PODIO's DataContainer (which underlies EDM4hep), we added interfaces that provide direct access to the underlying `std::pmr::vector`. This optimization reduces the time overhead related to data conversion by approximately 5% of the total data transfer time between GPU and CPU memory — a significant improvement for the computationally intensive processing of high-multiplicity events.

2.5 Compilation and library building

The track seed-finding algorithm is developed based on SYCL. However, The GCC compiler, which is used by CEPCSW, does not natively support SYCL. To enable integration with a Gaudi algorithm, we package the SYCL-related components as an external library, exposing all necessary interfaces without introducing any direct dependencies on SYCL. A pure virutal C++ class is defined, providing essential interfaces such as loading geometry and event data. A derived class is then implemented to handle all the traccc and SYCL specific details. Additionally, a factory method is introduced to create a concrete object without explicitly invoking the constructor of the derived class, so the implementation is hidden to the Gaudi algorithm. During initialization, the Gaudi algorithm calls the factory method to obtain a pointer to the pure virtual class. Then, the Gaudi algorithm in CEPCSW uses this pointer without knowing the underlying implementation. Finally, the external library is linked to the Gaudi module during the building stage.

3 Performance evaluation

3.1 Physics performance

The data sample used for the physics performance study comprises single electrons, muons, pions, kaons, and protons. These particles are generated at the origin using a single-particle gun, uniformly distributed in $\cos\theta$ within the coverage of the vertex detector. The generated Monte Carlo (MC) particles are then transported through the detector using Geant4, with the hit efficiency of all sensitive detector elements set to 100% during this process.

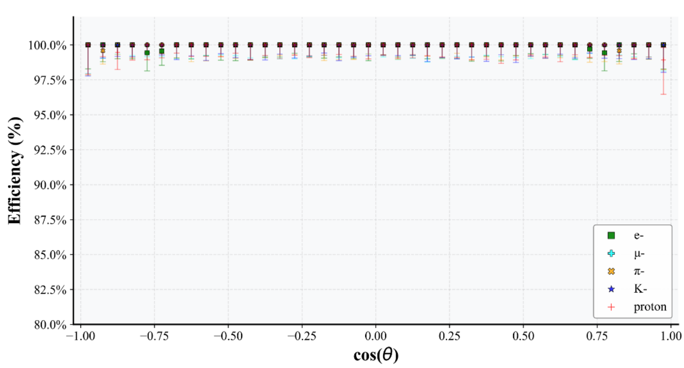

Figure 5. Track seed-finding efficiency as a function of $\cos\theta$ for various charged particle types — electrons, muons, pions, kaons, and protons — each with a momentum of 10 GeV. The efficiency error bars in the figure are calculated using the Clopper-Pearson method at a 1σ confidence level (~ 0.683).

Track seed finding efficiency versus $\cos\theta$ is shown in Figure 5 for various types of charged particles — electrons, muons, pions, kaons, and protons — each with a momentum of 10 GeV. The efficiency of track seed finding is defined as the ratio of the number of "good" seeds to the number of generated particles. A "good" seed is one in which at least half of its space points are generated by the associated MC particle. For all types of charged particles, track seed-finding efficiency exceeds 99.5% across all θ regions, except where $|\cos\theta|$ approaches 1. For electrons, the efficiency drops near $|\cos\theta| = 0.75$, primarily due to increased multiple scattering from additional material. The lower efficiency, accompanied by larger uncertainty, occurs as $|\cos\theta|$ approaches 1, corresponding to the edge of the vertex detector.

The vertex detector is mounted just outside the beam pipe, making it sensitive to beam-induced backgrounds that can affect track seed-finding performance. In the absence of a fully realistic background simulation, the impact of beam-related backgrounds is estimated by adding random hits around the simulated particle trajectories. A random noise level of 0.3% is used, corresponding to the expected background level during the Higgs run. The same track seed-finding algorithm is applied to simulated digits mixed with background hits. Results show that while the seed-finding efficiency remains largely unchanged, the total number of found seeds increases by a factor of 4.3. Although the efficiency is preserved, the increased number of fake seeds imposes a significant computational burden on the downstream track-finding process. As a next step, the criteria for triplet finding, triplet assembly, and final track seed formation should be optimized to reduce the number of fake seeds.

3.2 Computational performance

For computational performance evaluation, data samples containing muons with a momentum of 10 GeV are used. In Higgs-mode collisions, the average number of tracks per event is typically in the tens. To analyze performance trends by increasing computational complexity, events with varying numbers of tracks — up to a maximum of 10,000 — are generated, even though such conditions do not occur in real life.

For track seed finding, the most computationally intensive tasks — Clusterization, Doublet formation, Triplet searching, and Seed parameter estimation — can be executed in a data-parallel manner. However, data input is predominantly serial processes. These steps are integrated for computational performance evaluation.

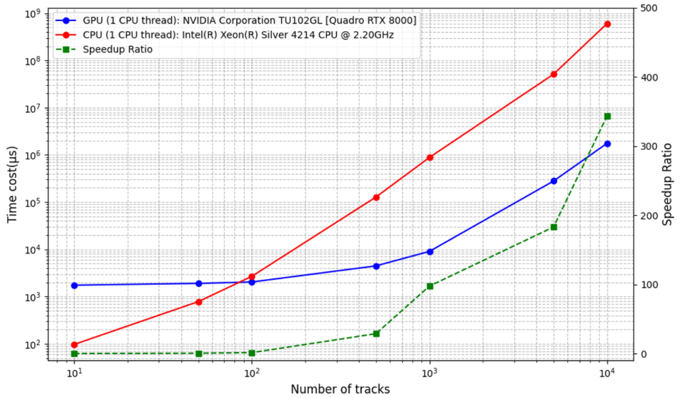

Figure 6. Processing time comparison between single-threaded GPU and CPU implementations across different amount of tracks. And the GPU-to-CPU speedup ratio is shown with the right y-axis. All tests used simulated muons with 10 GeV.

The performance comparison between the GPU and CPU for varying numbers of tracks is presented in Figure 6. At lower track counts, the CPU outperforms the GPU; however, beyond the crossover point of approximately 100 tracks, GPU efficiency increases significantly.

As the number of tracks grows, the workload benefits more from the GPU's architecture, resulting in a substantial performance advantage.

The GPU-to-CPU speedup ratio increases as the number of tracks grows. This behavior can be explained by Amdahl's law: as the track count increases, the proportion of exponentially complex tasks (Doublet formation and Triplet searching) rises, while the time spent on non-accelerated tasks (such as data input) decreases. According to Amdahl's law for fixed-load speedup:

$$S = \frac{1}{(1-p) + \frac{p}{n}}$$

where p represents the proportion of parallelizable work and n denotes the number of GPU threads. As the proportion of complex parallel tasks increases, the overall speedup also improves.

4 Conclusion

This paper presents the integration of traccc with the Gaudi-based CEPC software to enable track seed finding for the Vertex Detector. The seed-finding process has been developed by extending the existing Triplet searching algorithm to accommodate the double-sided layer structure of the Vertex Detector. Additionally, the event data model, EDM4hep, has been modified to utilize the same memory as traccc's VecMem, thereby eliminating the need for data copying between EDM4hep and VecMem.

The geometry required by traccc has been converted from CEPCSW's XML-based format using tools provided by ACTS/traccc. Additionally, detector identifiers have been adjusted to accommodate the different numbering schemes used by CEPCSW and ACTS. By removing CEPCSW's dependency on traccc, both CEPCSW and traccc can now be compiled using GCC and LLVM, respectively. The resulting executable can successfully be run on both CPU and GPU.

The physics performance of the track seed-finding algorithm has been measured using Geant4-simulated data. High track seed-finding efficiencies have been achieved for various particles — electrons, muons, pions, kaons, and protons — with a momentum of 10 GeV. The computational performance has been validated by comparing the execution speeds of Count&Find Doublets and Triplets — a key component of track seed finding — on GPU and CPU, demonstrating the acceleration achieved with GPU usage.

Acknowledgements: This work was supported by the NSFC Basic Science Center Program for "Joint Research on High Energy Frontier Particle Physics" (Grant No. 12188102), and Youth Innovation Promotion Association, CAS (Grant No. 2022011).

References

[1] CEPC Study Group et al., Cepc technical design report: Accelerator, arXiv preprint arXiv:2312.14363 (2023).

[2] CEPC Study Group et al., Cepc conceptual design report: physics & detector, arXiv preprint arXiv:1811.10545 (2018).

[3] Acts project on github., https://github.com/acts-project/acts

[4] X. Ai et al., A Common Tracking Software Project, Comput. Softw. Big Sci. **6**, 8 (2022), 2106.13593. 10.1007/s41781-021-00078-8

[5] Traccc project on github., https://github.com/acts-project/traccc

[6] Vecmem project on github., https://github.com/acts-project/vecmem