EPJ Web of Conferences **337**, 01198 (2025) https://doi.org/10.1051/epjconf/202533701198
CHEP 2024

FPGA Implementation of the General Triplet Track Fit

Kadir Tastepe[1,*], *Sebastian* Dittmeier[1], *Abhirikshma* Nandi[1], *Christof* Sauer[1,2], and *André* Schöning[1]

[1]Physikalisches Institut, Ruprecht-Karls-Universität Heidelberg,
Im Neuenheimer Feld 226, 69120 Heidelberg, Germany
[2]European Organization for Nuclear Research (CERN),
Espl. des Particules 1, 1217 Meyrin, Switzerland

Abstract. The reconstruction of charged particle trajectories is one of the most computationally intensive tasks within current and future filter farms of large High-Energy Physics (HEP) experiments. Due to the increasing number of simultaneous collisions in future high-luminosity colliders, like the HL-LHC and FCC-hh, the challenge of online tracking and event reconstruction becomes even more significant and requires innovative algorithms and appropriate hardware choices for its acceleration.
The General Triplet Track Fit is a novel parallelizable track-fitting algorithm that offers a great potential for speed-up by processing triplets of hits independently and allowing to factorize the track reconstruction chain into a detector-dependent and independent parts. FPGAs, with their inherent parallelism, power efficiency, and reconfigurability, are becoming increasingly attractive as co-processors for large data centres, such as heterogeneous online farms, to meet the challenges of increasing throughput and computational complexity.
A preliminary FPGA implementation of the General Triplet Track Fit has been developed using High-Level Synthesis on AMD FPGAs. Synthesis results indicate that with float precision, a throughput of approximately 10^7 track fits per second can be achieved, with further room for improvement. The method's versatility across diverse detector types and its capability to reject fake triplets hold early promise for robust performance in future high-energy physics experiments.

1 Introduction

High luminosities are crucial for detecting rare processes and exploring physics beyond the Standard Model of particle physics. The challenges of particle tracking in high luminosity experiments lie primarily in the required computational intensity. With the increasing number of simultaneous collisions, the complexity and data volume that needs to be processed grow rapidly. At the High Luminosity(HL) LHC the pile-up rates of up to 200 collisions per bunch crossing will be reached, and for FCC-hh unprecedented 1000 collisions per bunch crossing are foreseen, presenting *significant* challenges for data processing and analysis. Traditional track fitting algorithms, such as the Kálmán filter, are sequential and iterative, which limits their computational efficiency in high-rate environments due to the limited parallelization.

*e-mail: tastepe@physi.uni-heidelberg.de

The General Triplet Track Fit (GTTF) [1] is a novel parallelizable track fitting algorithm that extends the Multiple Scattering(MS)-only fit, introduced in Ref. [2], by including hit uncertainties, making it usable for high-energy collider experiments. The local processing of hit triplets absorbs the detector dependent conditions such that the subsequent global track fit becomes independent of the concret detector layout. This flexibility, combined with the ability to filter out poor-quality triplets early in the reconstruction process, has the potential to reduce the computational load. Parallel algorithms and hardware acceleration can be leveraged to improve online reconstruction, trigger, and data acquisition systems to overcome computational challenges in high-luminosity environments. This work presents an early-stage FPGA implementation of the GTTF algorithm, exploring its potential performance on a sample of tracks simulated in a idealized barrel tracking detector. The effect on performance from varying the number of tracking layers is studied.

2 The General Triplet Track Fit

The General Triplet Track Fit can be factorized into two main steps: the local processing of triplets and the global fit. The local processing step includes calculating triplet parameters and hit gradients, which take into account detector-specific information, such as the magnetic field and material in the tracking detectors. This step is crucial as it allows for filtering bad-quality triplets, ensuring that only reliable triplets are used in the subsequent track fitting process. Additionally, triplets of hits can be processed in parallel using hardware accelerators.

The global fit step is designed to be *detector-independent*. It uses the calculated parameters and hit gradients of the individual triplets to fit the entire track. Particle tracks are calculated by minimizing the following χ^2:

$$\chi^2 = \sum_{\text{Triplets k}} \left\{ \frac{\Delta\Theta^2}{\sigma_\Theta^2} + \frac{\Delta\Phi^2}{\sigma_\Phi^2} \right\}_{\text{MS, k}} + \sum_{\text{Hits m}} \left\{ \delta\vec{x}^{\mathrm{T}} V^{-1} \delta\vec{x} \right\}_{\text{m}}, \tag{1}$$

which takes into account both multiple scattering and hit uncertainties, and involves dependence of kink angles on the track curvature (momentum), which is linearized around the circle solution [1]. Using spherical coordinates, for each triplet, the kink due to multiple scattering can be projected onto the polar ($\Delta\Theta$) and azimuthal ($\Delta\Phi$) angles, which are divided by their respective expected errors (σ_Θ and σ_Φ). Hit positions are represented as shifts relative to measured positions, denoted by $\delta\vec{x} = \vec{x}_{\text{fit}} - \vec{x}_{\text{meas}}$. For each hit, the hit position uncertainty is described by a hit covariance matrix V.

To fit tracks, a minimum of three detector layers is required. Increasing the number of detector layers leads to more hits and triplets to process, thereby increasing the computational workload.

3 FPGA Implementation

The GTTF algorithm was first developed and validated on a CPU platform. The AMD Vitis HLS Toolflow [3] is used to translate the higher-level algorithmic code written in C++, for running on FPGAs. The target deployment involves using the GTTF on an accelerator, where the CPU (Host) is connected to FPGAs (Device) via PCIe on the same motherboard. The algorithm is implemented on an AMD Alveo U280 accelerator card that is specifically designed for high-performance applications and consists of three Super Logic Regions (SLRs), as shown in Figure 1.

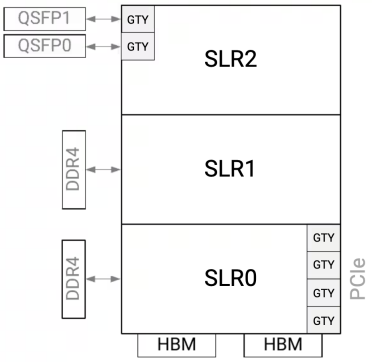

Figure 1: Floorplan of the XCU280 device. It supports PCIe 4.0, 8 GB High Bandwidth Memory (HBM2), two 16 GB DDR4 RDIMMs, and two QSFP28 Ethernet ports (100 Gb/s each). The HBM is co-located on the device and connects directly to SLR0, delivering a total bandwidth of 460 GB/s. [4]

To enhance performance and resource utilization, a multi-kernel design approach is used for the implementation. As shown in Figure 2, two distinct kernels have been developed: the Local Processing Kernel (LPK) and the Global Fit Kernel (GFK). Each kernel is strategically assigned to separate SLRs. This design choice not only simplifies the implementation but also allows for parallel processing, enabling each kernel to operate independently.

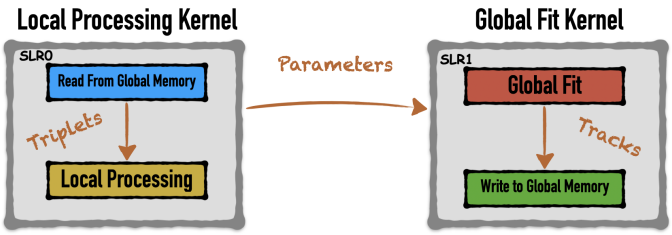

Figure 2: The implemented design consists of two FPGA kernels that communicate via streams for efficient data transfer.

The dataset, which includes hit triplets obtained from a toy detector simulation, is prepared on the host side. The triplet data consists of the hit positions and their uncertainties, the value of the homogeneous magnetic field, and the material thickness per detector layer. Using XRT Native APIs [5] the dataset is sent to the FPGA for processing. The LPK reads triplet data from the FPGA's global memory, calculates the triplet parameters and hit gradients, and then streams this information to the GFK, which calculates the track parameters and subsequently sends results back to the host.

3.1 Functional Verification

The functional correctness of the translated HLS C++ code is verified first using software emulation. The design is tested in hardware emulation to simulate the FPGA behavior and validate the interaction between kernels, data streams, and resource usage.

The verification step is crucial to validate that the HLS C++ implementation will perform as expected when synthesized into hardware. Figure 3 shows that the FPGA implementation results are consistent with the CPU-based implementation. After successfully passing emulations, the design is implemented for hardware and tested on the FPGA.

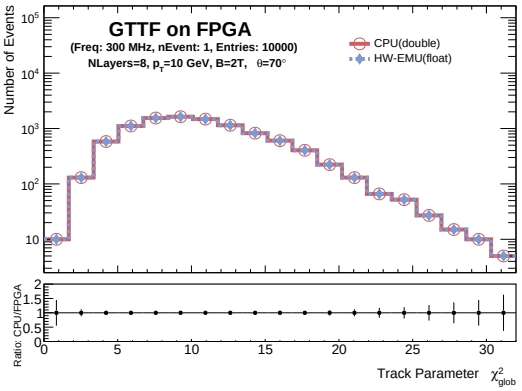

Figure 3: To compare the results of the CPU-based implementation (red line) and HLS (blue dashed line), the track parameter χ^2 is given as an example. Using a toy detector simulation with particles having $p_T = 10$ GeV and $\theta = 70°$, a detector with a 2T magnetic field and 8 tracking layers is simulated to produce a triplet dataset. Developed kernels are HLS-synthesized, and each event is sent to the FPGA kernels one at a time, where it is processed in 10^4 iterations to compute tracks. The same dataset of events is also calculated on the CPU.

3.2 Optimization for High Throughput

The design is optimized by simplifying calculations and breaking them into smaller, reusable steps as illustrated in Figure 4. This reduces logic depth and improves throughput. Each variable is defined by a single operation involving two operands. Bit shift manipulations are used for multiplication and division. Loop optimizations (pipelining, unrolling, flattening, etc.) are applied to enhance the performance and maximize the throughput.

```
distance = sqrt(pow(x,2)+pow(y,2));
```

⟶

```
a=x*x;
b=y*y;
c=a+b;
d=sqrtf(c);
```

Figure 4: This figure illustrates an example of the simplification of expressions for calculating the Euclidean distance between two points. The code on the left represents sequential execution, while the simplified code on the right benefits from partially parallel execution by removing dependencies among individual tasks.

The input of the LPK and the output of the GFK are assigned to memory ports to achieve parallel data access. The dataflow directive is used to enable task-level pipelining in both

kernels. The implemented design operates at a clock frequency of 300 MHz, with the initiation interval defined as the number of clock cycles between the start of consecutive tasks. This interval determines how frequently new data can be processed in a pipelined design, as illustrated in Figure 5.

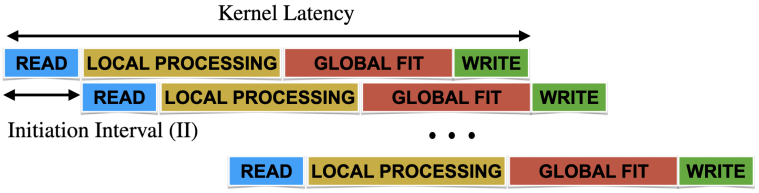

Figure 5: This diagram shows the pipelined design. To process an event, triplets must go through these stages. Paralellism is leveraged by sending the next triplet before the previous one is finished. By reducing the initiation interval, the pipeline achieves higher throughput.

Further optimizations are possible depending on the tracking regime and detector geometry. Examples are the small bending approximations and the fast MS-only fits [2].

3.3 Resource Usage and Throughput

The throughput of the implemented algorithm can be estimated from the length of kernels' initiation intervals and the clock frequency. This, along with the resource utilization of the algorithm, is studied and optimized as a function of the number of tracking layers. The optimization strategy is to minimize the kernels' initiation intervals while maintaining efficient hardware usage.

The minimum number of initiation intervals is shown in Table 1 as a function of the number of tracking layers. The numbers are given for the LPK and the GFK, and consider the available logic and memory resources in the FPGA. For a larger number of tracking layers, the initiation intervals need to be increased in order to fulfill the resource constraints. Increased initiation intervals, however, also lead to a decrease in throughput.

Number of Layers	3	4	5	6	7	8	9
Local Processing Kernel	6	11	16	22	28	34	40
Global Fit Kernel	1	2	3	4	5	6	7

Table 1: Initiation intervals of the two algorithm kernels (in clock cycles) with a different number of tracking layers.

The resource utilization for the LPK and the GFK is shown in Figure 6 as a function of the number of tracking layers. For the LPK, the resource usage depends only weakly on the number of tracking layers, as the same resources are reused for the processing of different triplets. It should be noted that the current implementation uses many trigonometric functions, which offer a big potential for further optimizations. For the GFK, a strong dependence on the number of tracking layers is observed. This behavior is expected since the global fitting involves an inversion of a matrix (see Eq. (1)) whose size scales with the number of tracking layers.

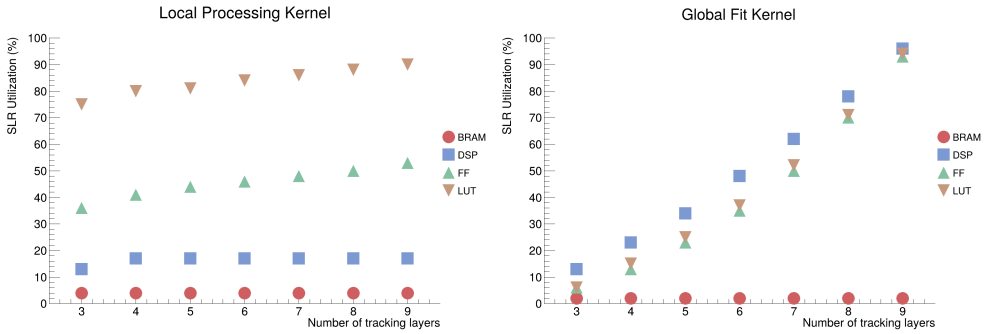

Figure 6: The figures shown above are derived from hardware emulation results using Vitis HLS. The main types of hardware resources are block random access memory (BRAM), digital signal processing element (DSP), flip-flop (FF), and look-up table (LUT). Resource utilization of the implemented kernels, corresponding to a *single* SLR of the FPGA, the LPK (left), and the GFK (right) versus a number of tracking layers.

Figure 7 shows the estimated individual performance for the implemented kernels. In the current implementation, the overall latency will be dominated by the slowest kernel in the design. For implementations with up to six tracking layers, the GFK has higher throughput than the LPK. The GFK must wait for the LPK to deliver the necessary triplet parameters and hit gradients, which is a bottleneck in the global fit. For more than six tracking layers, the increase of the triplet covariance matrix result in a slower global fit.

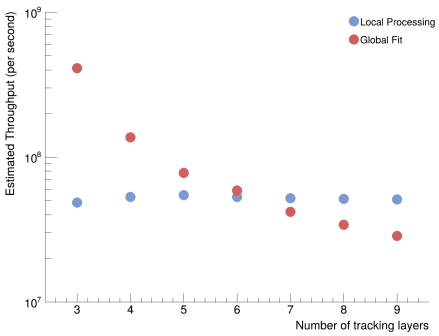

Figure 7: Estimated throughput of the LPK (triplet/s) and GFK (global track fit/s) as a function of the number of tracking layers.

4 Conclusion & Outlook

The General Triplet Track Fit has been implemented on an FPGA using AMD Vitis HLS, and preliminary results from a first kernel design are presented. The synthesis results indicate that, processing rates of approximately 5×10^7 local processes of hit triplets and 4×10^7 global track fits per second for eight tracking layers using floating precision can be achieved.

Different kernel models were tested to see if they can increase throughput. To further improve the performance (throughput), it is possible to divide the fitting algorithm into smaller components and to use several parallel kernels. It is also possible to perform application-specific optimization to reduce the required hardware resources. Depending on the outcome of further studies, one could also consider to include a track finder kernel, which is able to resolve ambiguities between different track candidates.

References

[1] André Schöning, "A General Track Fit based on Triplets," *arXiv*, **2406.05240** (2024). https://doi.org/10.48550/arXiv.2406.05240

[2] N. Berger, A. Kozlinskiy, M. Kiehn, and A. Schöning, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **844**, 135-140 (2017). https://doi.org/10.48550/arXiv.1606.04990

[3] AMD. (2023.1). Vitis Application Development Flow User Guide (UG1393). https://docs.amd.com/r/2023.1-English/ug1393-vitis-application-acceleration

[4] Alveo U280 Data Center Accelerator Card Data Sheet. https://www.mouser.com/datasheet/2/903/ds963_u280-1595425.pdf

[5] XRT Native APIs. https://xilinx.github.io/XRT/master/html/xrt_native_apis.html