SYCLOPS

Deliverable 5.3 - SYCL-ROOT Library

GRANT AGREEMENT NUMBER: 101092877

This project has received funding from the European Union's HE research and innovation programme under grant agreement No 101092877

Project acronym:	**SYCLOPS**
Project full title:	**Scaling extreme analYtics with Cross architecture acceLeration based on OPen Standards**
Call identifier:	**HORIZON-CL4-2022-DATA-01-05**
Type of action:	**RIA**
Start date:	**01/01/2023**
End date:	**31/12/2025**
Grant agreement no:	**101092877**

D5.3 - SYCL-ROOT Library

Executive Summary: This deliverable presents SYCL-ROOT library, which introduces SYCL-based components to enable performance-portable HEP data analysis using heterogenous hardware, preparing ROOT for efficient GPU and accelerator execution. This includes GenVectorX, a SYCL reimplementation of ROOT's GenVector library validated on various GPUs with performance gains, and RDataFrame, where prototype GPU support shows speedups in histogramming. Physic case studies (eg, di-muon invariant mass and Folded W boson analysis) demonstrate the benefits of up to 2x speedups. All developments are public on Github and being integrated into ROOT, with results shared through conferences and publications.

WP:	5		
Author(s):	Monica Dessole, Devajith Valaparambil Sreeramaswamy, Danilo Piparo		
Editor:	Raja Appuswamy		
Leading Partner:	CERN		
Participating Partners:			
Version:	1.0	**Status:**	Draft
Deliverable Type:	Other	**Dissemination Level:**	PU
Official Submission Date:	30-Sep-2025	**Actual Submission Date:**	06-Oct-2025

Disclaimer

The SYCLOPS consortium consists of the following partners:

No.	Partner Organisation Name	Partner Organisation Short Name	Country
1	EURECOM	EUR	**FR**
2	INESC ID - INSTITUTO DE ENGENHARIADE SISTEMAS E COMPUTADORES, INVESTIGACAO E DESENVOLVIMENTO EM LISBOA	INESC	**PT**
3	RUPRECHT-KARLS-UNIVERSITAET HEIDELBERG	UHEI	**DE**
4	ORGANISATION EUROPEENNE POUR LA RECHERCHE NUCLEAIRE	CERN	**CH**
5	HIRO MICRODATACENTERS B.V.	HIRO	**NL**
6	ACCELOM	ACC	**FR**
7	CODASIP S R O	CSIP	**CZ**
8	CODEPLAY SOFTWARE LIMITED	CPLAY	**UK**

Document Revision History

Version	Description	Contributions
0.1	Initial draft	**EUR**
0.2	Updated core contribution	**CERN**
1.0	Finalized draft	**EUR**

Authors

Author	Partner
Devajith Valaparambil Sreeramaswamy	**CERN**
Monica Dessole	**CERN**

Reviewers

Name	Organisation
Aleksandar Ilic	INESC
Vincent Heuveline	UHEI
Stefan Roiser	CERN
Nimisha Chaturvedi	ACC
Martin Bozek	CSIP

Statement of Originality

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Table of Contents

Executive Summary

This deliverable reports on the development of the SYCL-ROOT library, a set of SYCL based components which enable performance-portable high-energy physics (HEP) data analysis on heterogenous hardware. The work targets both foundational mathematics (via GenVectorX) and high-level analysis interface (RDataFrame), aiming to prepare ROOT for efficient execution on GPUs and emerging accelerators across multiple vendors.

Within task 5.3, progress was achieved in:

1. **GenVectorX**: A SYCL-based reimplementation of ROOT's GenVector library, which is intended to provide classes and functionalities to represent and manipulate particle events. The library enables these operations to run on GPUs and accelerators with performance portability. Initial validations confirm functional correctness on NVIDIA and AMD GPUs, with benchmarked speedups on NVIDIA GPUs compared to CPU execution.
2. **RDataFrame**: RDataFrame is ROOT's high-level data analysis interface, representing physics data in a columnar format (Dataframe). HEP analysis involves iterating over the events (rows) to compute distributions (histogramming), applying filters and computing new columns. Prototype support has been developed to offload selected RDataFrame actions to GPUs, like histogramming, and increasing computational intensity by offloading define action (define + histogramming), which reduces kernel launch overhead and demonstrates speedups in physics analyses.
3. **Physics Case Studies**: The prototypes have been validated in simplified analyses, like the di-muon invariant mass computation and folded W boson analysis. These studies highlight the benefits (up to ~2X speedups in some cases).
4. **Performance portability**: Benchmarks across CUDA and SYCL implementations (DPC++, AdaptiveCpp) explored memory models (buffers vs USM) and portability trade-offs. While CUDA remains faster on NVIDIA devices, USM-based SYCL approach reach competitive performance with the added benefit of cross-vendor portability.
5. **Profiling**: Performance bottlenecks have been measured using Perf, Nsight, and AdaptivePerf, providing guidance for further optimization.

All developments are publicly available, with SYCL-ROOT components hosted on Github, and will be integrated to ROOT (via pull requests). Results are disseminated through presentations at conferences and papers.

1 Introduction

Figure 1 shows the SYCLOPS hardware-software stack consists of three layers: (i) infrastructure layer, (ii) platform layer, and (iii) application libraries and tools layer.

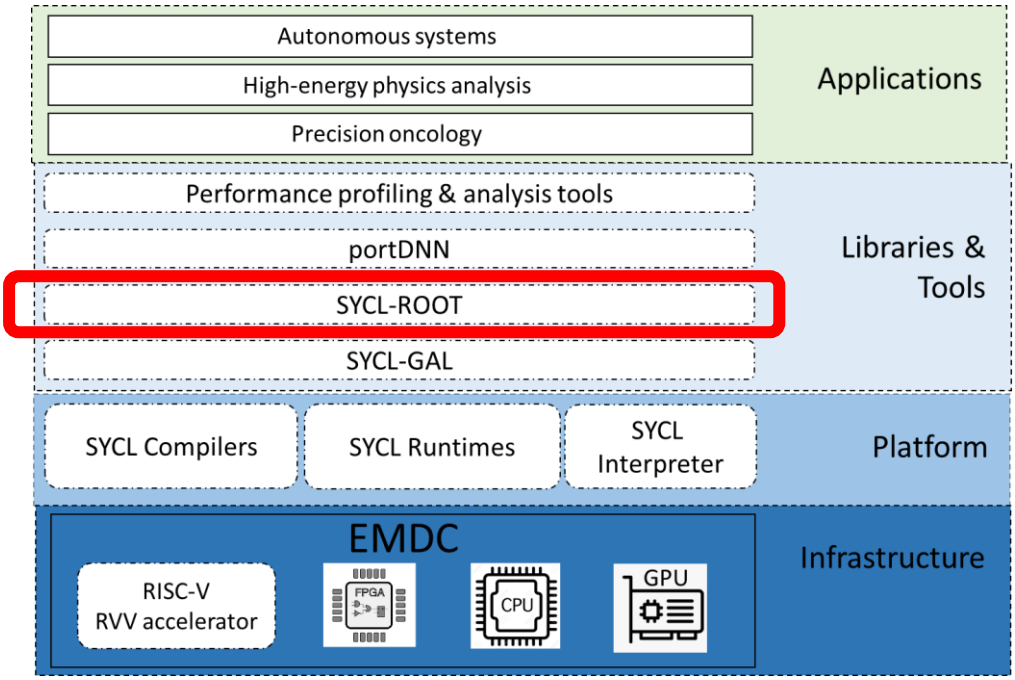


Figure 1. SYCLOPS architecture

Infrastructure layer: The SYCLOPS infrastructure layer is the bottom-most layer of the stack and provides heterogeneous hardware with a wide range of accelerators from several vendors.

Platform layer: The second layer from the bottom, the platform layer, provides the software required to compile, execute, and interpret SYCL applications over processors in the infrastructure layer. SYCLOPS will contain oneAPI DPC++ compiler from CPLAY, and AdaptiveCpp from UHEI. In terms of SYCL interpreters, SYCLOPS will contain Cling from CERN.

Application libraries and tools layer: While the platform layer described above enables direct programming in SYCL, the libraries layer enables API-based programming by providing pre-designed, tuned libraries for various deep learning methods for the PointNet autonomous systems use case (SYCL-DNN), mathematical operators for scalable HEP analysis (SYCL-ROOT), and data parallel algorithms for scalable genomic analysis (SYCL-GAL).

This deliverable covers the **SYCL-ROOT** part of the stack as highlighted in Figure 1. Several developments have been made in the context of “WP5: Task 5.3 SYCL-ROOT Library”. This deliverable is a summary of this work done.

This deliverable is structured as follows. Section 1 of this deliverable provides a high-level overview of the overall SYCLOPS architecture and positions this deliverable with respect to both components in the SYCLOPS stack and WP/tasks in the work plan. Section 2 gives the background on ROOT and the relevant libraries. Section 3 describes the work done on the SYCL-ROOT library.

2 Background

The discovery of Higgs boson by the ATLAS and CMS collaborations in 2012 (Figure 2) marked a milestone in particle physics and highlighted the critical role of large-scale data analysis frameworks such as ROOT. ROOT provides the tools needed to store, transform and analyse the petabyte-scale datasets produced by the LHC. With the upcoming high-luminosity LHC, the data rate is expected to increase by an order of magnitude. As data volumes and analysis complexity increase, ROOT must evolve to exploit heterogenous hardware platforms such as GPUs and emerging accelerators while maintaining its broad adoption in the physics community.

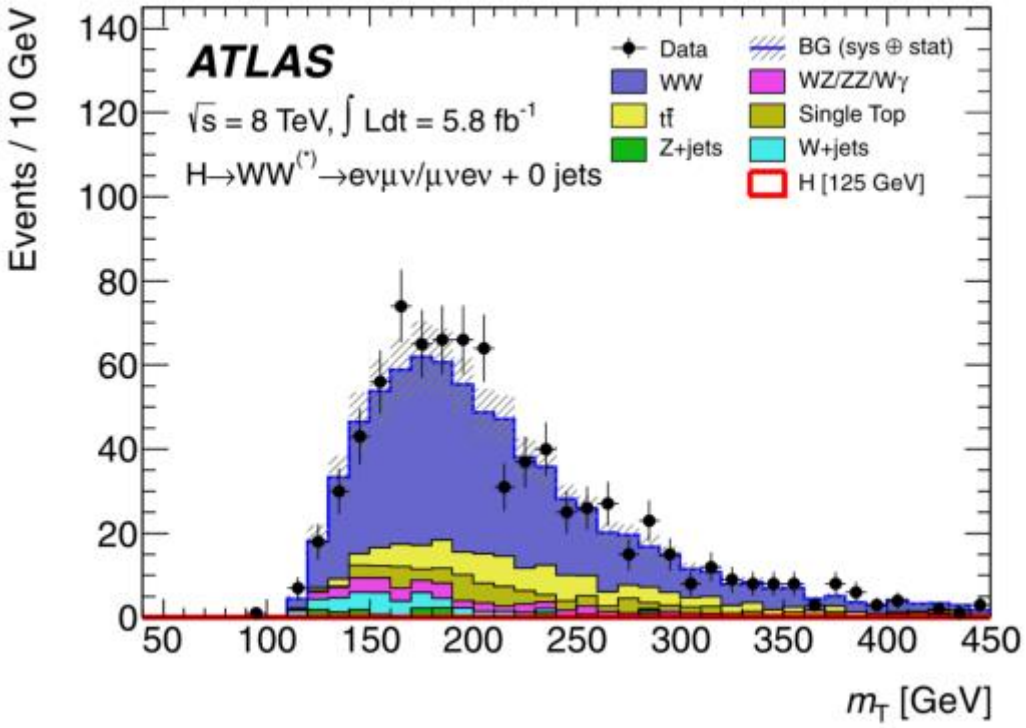


Figure 2: The ATLAS Collaboration. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC

Physics Letters B. 2012 Sep 17;716(1):1-29, Figure 5a, DOI: https://doi.org/10.1016/j.physletb.2012.08.020

This section introduces the ROOT components relevant to Task 5.3, reviews the limitations of vendor-specific GPU implementations, and motivates the use of SYCL as a sustainable approach to performance portability.

2.1 ROOT, RDataFrame, Vector Algebra in HEP

ROOT offers a rich ecosystem of libraries that support diverse HEP analysis tasks. Among these, the two components are especially relevant for GPU acceleration:

- RDataFrame (RDF): ROOT's high-level interface for analysis of columnar data: event rows with property columns. It has a declarative interface that allows users to build analysis pipelines through high-level transformations and aggregations.
- GenVector: A math package that provides 2, 3 and 4 dimensional physical vectors in different coordinate systems, enabling fundamental operations for HEP analysis. This include more advanced operations like rotations, Lorentz and Poincare transformations.

These components are widely used. For example, a typical analysis workflow involves reading and decompressing event data, applying filters, defining additional columns, and aggregating results through

by filling histograms. Figure 3 shows an example RDF code to compute invariant masses for a set of 4 dimensional particles expressed as GenVector objects. Vector algebra operations occur repeatedly in these pipelines making them natural candidates for acceleration.

```
1  ROOT::RDataFrame df(dataset);
2  auto InvariantMass = [](double x, double y, double z,
       double e) {
3       PxPyPzE p(x,y,z,e);
4       return e.M();
5     };
6  auto df1 = df.Filter("x != 0")
7                .Define("m", InvariantMass,
                     {"x","y","z","e"});
8  auto h1 = df1.Histo1D("m");
9  h1.Draw();
```

Figure 3: Basic RDataFrame workflow

3 SYCL-ROOT Acceleration Library

SYCL-ROOT acceleration library is the central outcome of Task 5.3. It provides a set of SYCL-based implementations of performance-critical ROOT components.

All SYCL-ROOT components are developed in public repositories to ensure transparency and reproducibility:

- GenVectorX: https://github.com/root-project/genvectorx
- ROOT integration branch (RDataFrame): https://github.com/mdessole/root/tree/genvectorx_gpu_histogram_bulk-2.0

3.1 GenVectorX

The library GenVector was extended to GenVectorX, which enables parallel execution on NVIDIA GPUs via CUDA and other backends via SYCL, while retaining performance, minimizing code duplication and maximizing code reuse. This promotes code sustainability and portability. The below figure states the code similarity (a value ranging between 0 and 1, where 1 means that two code bases are equal) between GenvectorX and GenVector. i.e. the GenVector original code, and the other platforms taken into account. For a detailed description of how code similarity is evaluated, please refer to reference (3).

Similarity	Platform	Problem
0.9694	CUDA	Invariant Masses
0.9715	SYCL	Invariant Masses

Figure 4: Code similarities against pure C++ code

Tests were carried out on different NVIDIA GPUs (NVIDIA GeForce RTX 3060 using CUDA 12.2, NVIDIA L4 using CUDA 12.3), focusing on the performance gap between native CUDA and SYCL code execution. For an invariant mass computation problem, we study scaling and demonstrate that our implementation reaches performance portability, for almost all sizes of inputs. Figure 5 shows kernel execution time (excluding memory transfers) against input size (number of particles). CUDA and SYCL implementations are compared, the latter being compiled with both AdaptiveCpp and oneAPI DPC++ and using two memory paradigms, namely buffers+accessors (BUF) and USM device pointers (PTR). Figure 6 shows total execution time breakdown for several input sizes. Three categories are highlighted: memory operations, kernels and CUDA API calls. We do not observe a significant difference between the two memory management strategies, BUF and PTR. However, oneAPI DPC++ version performs better and is closer to native CUDA.

Figure: RTX3060 Figure: L4

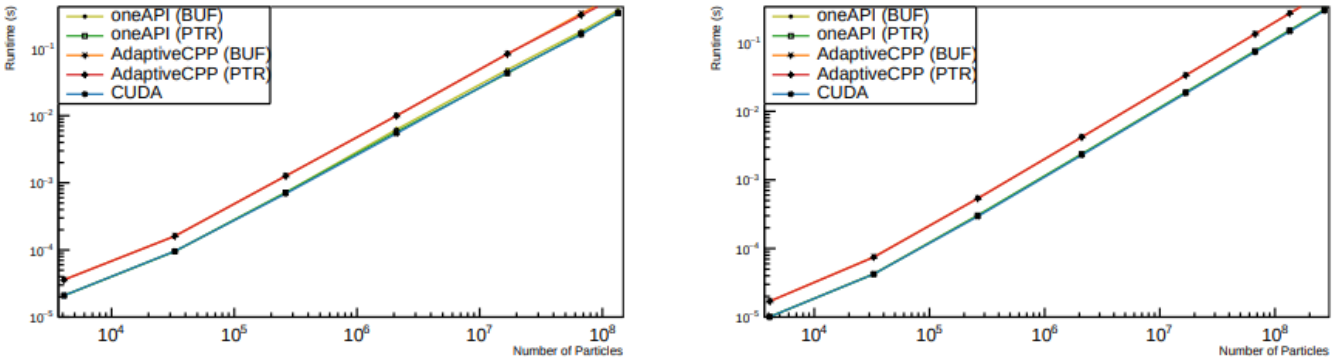


Figure 5: Scaling - Kernel Execution Time

Figure: RTX3060 Figure: L4

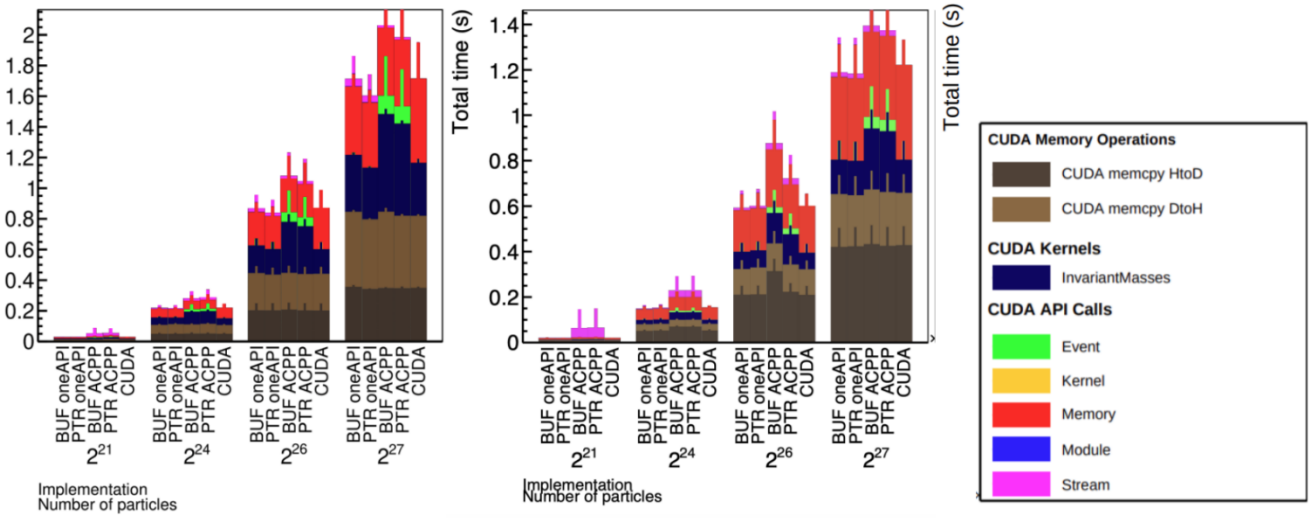


Figure 6: Total Execution Time Breakdown

3.2 Histogramming

RDF has been extended with SYCL offloading for histogramming, enabling these actions to run on GPUs and accelerators. This work demonstrates how we can benefit from heterogenous execution.

The histogram action in ROOT normally involves 3 steps:

- Determining which coordinate to fill based on the input coordinate
- Incrementing the bin with a given weight
- Updating the sum variables for histogram statistics (e.g., mean, standard deviation)

In the CUDA version, two base kernels compute the three histogramming steps:

- Histogram kernel: Each thread block first fills its own local histogram in shared memory (reducing contention), then these results are merged into the final global histogram.
- Reduction kernel: Computes statistics like mean and standard deviation.

Figure 7 shows the computational workflow for the histogramming action. The dataset is broken down in bulks of events which can fit into GPU memory and are sequentially processed.

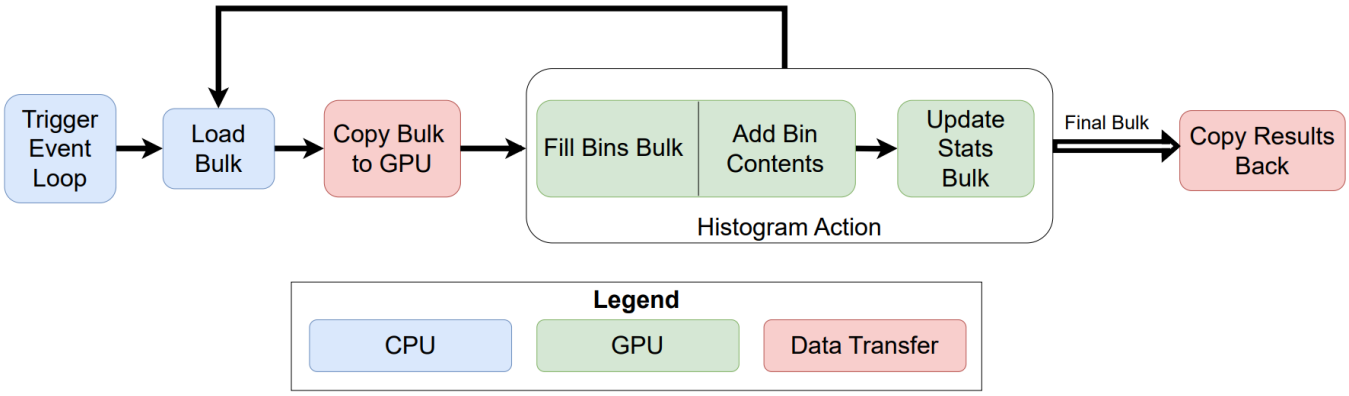


Figure 7: The GPU Histogram Action

Jolly Chen, Monica Dessole, and Ana-Lucia Varbanescu. "Migrating CUDA to SYCL: A HEP Case Study with ROOT RDataFrame". In: Proceedings of the 12th International Workshop on OpenCL and SYCL. IWOCL '24. Chicago, IL, USA: Association for Computing Machinery, 2024. doi: 10.1145/3648115.3648122

For the SYCL version, we ported the CUDA kernels to a named function object with identical behaviour (with equivalent SYCL calls). We achieve a speedup (against 24 threaded CPU) of around 1.9x with DPC++ and 1.4x with AdaptiveCpp at one billion event. Large amount of time is spent on memory operations.

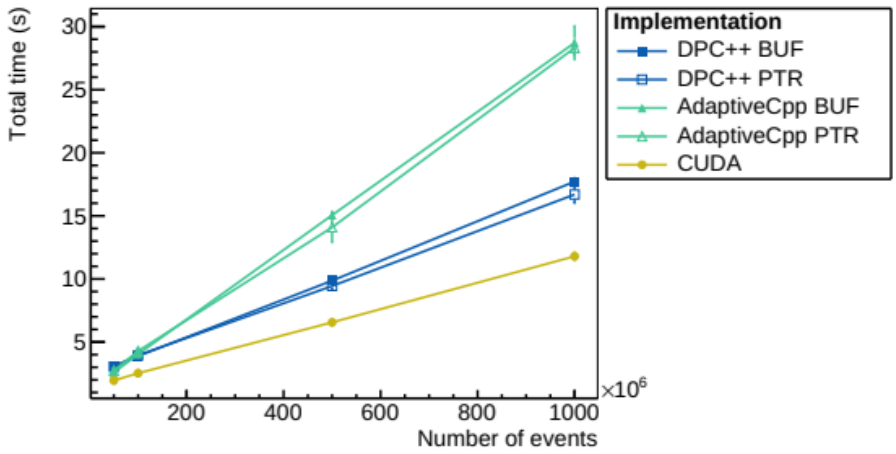


Figure 8: Total runtime of RDataFrame histogramming

Jolly Chen, Monica Dessole, and Ana-Lucia Varbanescu. "Migrating CUDA to SYCL: A HEP Case Study with ROOT RDataFrame". In: In: Proc. 12th Int. Workshop on OpenCL and SYCL (IWOCL '24). ACM, 2024. Figure 1. DOI: 10.1145/3648115.364812

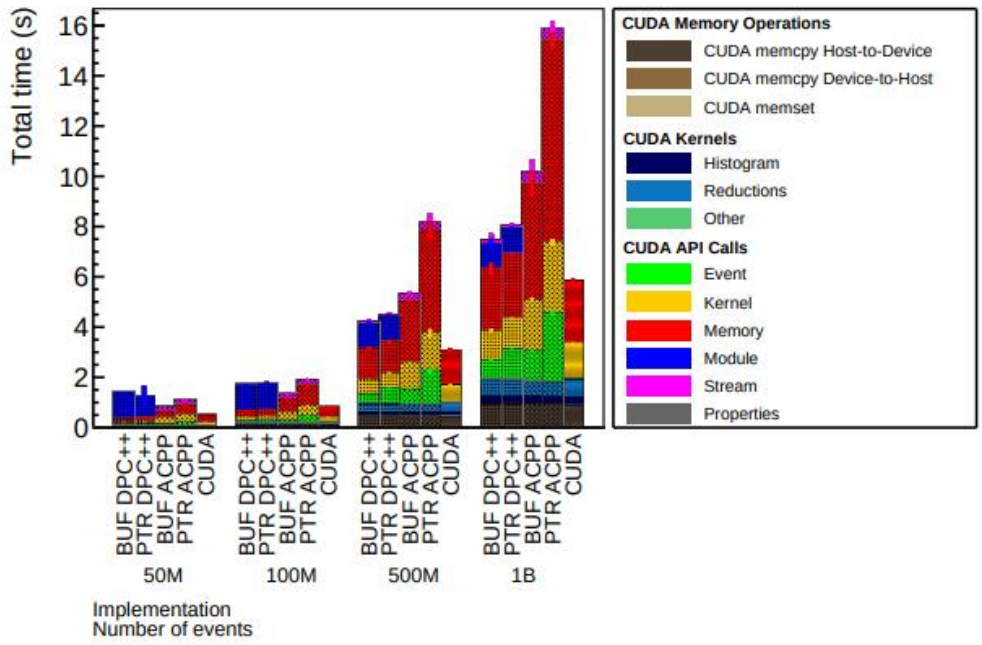


Figure 9: GPU activity profiled using NSight Systems

Jolly Chen, Monica Dessole, and Ana-Lucia Varbanescu. “Migrating CUDA to SYCL: A HEP Case Study with ROOT RDataFrame”. In: In: Proc. 12th Int. Workshop on OpenCL and SYCL (IWOCL ’24). ACM, 2024. Figure 2. DOI: 10.1145/3648115.364812

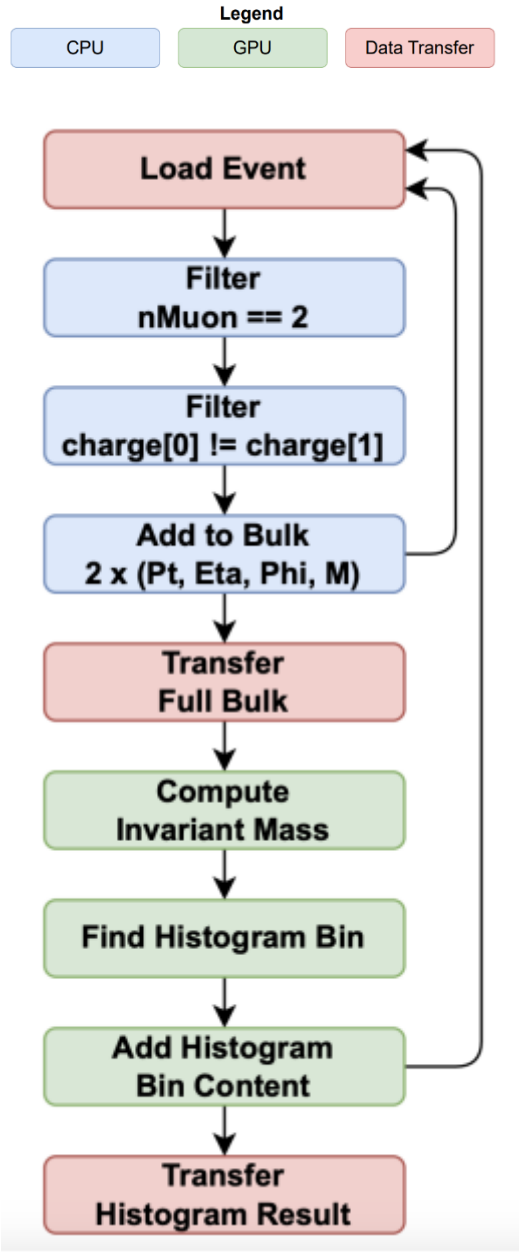
Figures 8 and 9 illustrate the results of the benchmark. We evaluate our SYCL implementation with buffers+accessors (BUF) and USM device pointers (PTR) for transferring bulks of events. We do not observe a significant difference between the two memory management strategies, BUF and PTR. However, DPC++ version performs better and is closer to native CUDA. The performance gap between the SYCL and CUDA implementations widens (i.e., SYCL performance gets relatively worse) with more events.

For the second part of the investigation, we consider a real-world use case; before histogramming, many analyses also require computing new quantities from the raw event data. Example usages include adding a column that contains the invariant mass of a particle, or a selecting a subset of elements of an array (e.g. only the pts of "good" muons).

This presented another challenge, as computational intensity of Define or Histogram actions alone is not fully sufficient to exploit GPUs. Therefore, we designed a fused Define+Histo action, where the histogramming class is templated over the Define operator which can be provided by the user. Preliminary investigation was conducted on following two real world cases:

3.2.1 DiMuon analysis

- Calculate invariant mass of all events with exactly 2 muons with opposite charge
- Discard irrelevant events on CPU
- Transfer 8 doubles (two 4-dimensional particles) to fill a bin in a single histogram
- Calculate invariant masses on GPU
- Fill histogram on GPU



3.2.2 Folded W analysis

- Calculate forward folding and invariant mass of all good events
- Discard irrelevant events on CPU
- Forward folding depends on 2 variables, scale and resolution, each of each might take 100 values
- Transfer 10 doubles (2 4-dimensional particles and additional value) to fill a bin in a one of the 10 000 histograms
- Calculate forward foldings and invariant masses on GPU
- Fill histograms on GPU

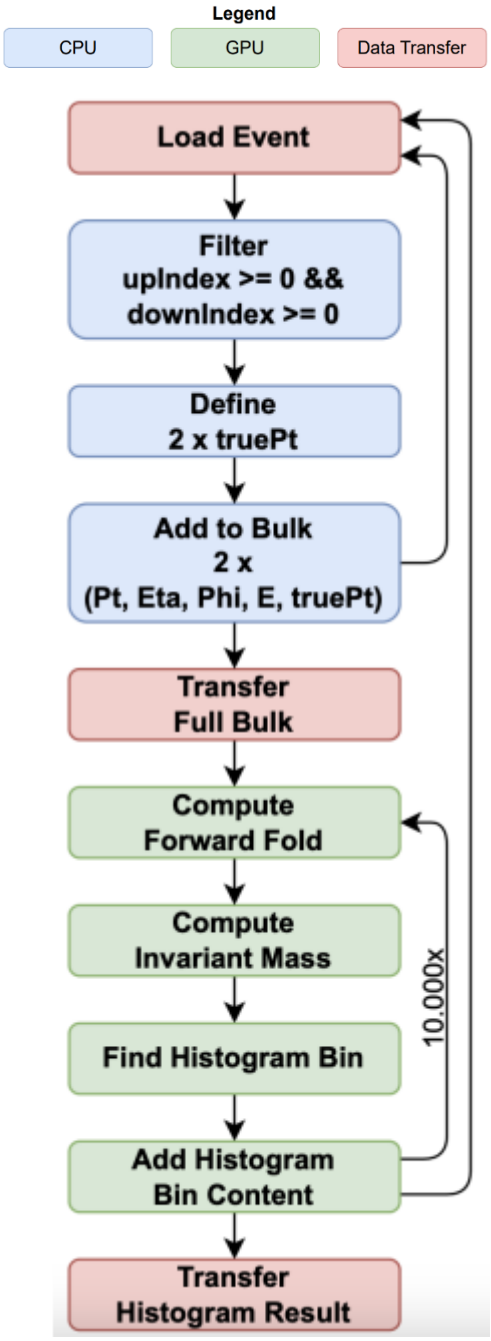


Figure 10 shows the performance results for these two test cases. Performance is heavily test-case dependent, and results do not take into account the overhead related to RDF Bulk API.

System: AMD Ryzen 7 5700G 16-cores CPU, NVIDIA RTX 3060 GPU

	DiMuon	Folded W
Number of events	24 067 843	100 000
Speedup over 16 threaded CPU	2.6x	95×
Time for data transfers	57.5%	0.1%

Figure 10: Speedup over CPU only execution

4 Conclusion

Task 5.3 has demonstrated that GPU acceleration of ROOT workflows using SYCL is both feasible and beneficial. The SYCL-ROOT library provides the first concrete integration of performance-portable SYCL components into ROOT, lowering the barrier for heterogenous execution in high-energy physics analyses.

Key achievements include:

1. We detailed the migration to both SYCL and CUDA of a large, complex, C++ code base, i.e. GenVector and Histograms
2. We evaluated code divergence of GenVectorX, to estimate the benefits in maintaining a single source code without specializing regions of code for specific targets
3. We empirically showed performance portability of the migrated SYCL code on different platforms and architectures
4. We provided evidence of performance gain for suitable test cases when combining Define and Histogram actions

What needs to be done:

1. Integration within ROOT is work in progress. Integration with RDF and SYCL-enabled CLING is only partially available.
2. Carry out performance evaluation on other platforms and architectures, namely Intel GPUs and RISC-V.
3. Extend GenVectorX to tackle even more complex HEP analyses.

References

(1) https://arxiv.org/abs/2401.13310

(2) https://dl.acm.org/doi/10.1145/3648115.3648122

(3) https://arxiv.org/abs/2312.02756