SYCLOPS

Deliverable 4.4 - SYCL interpreter

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**

D4.4 - SYCL interpreter

Executive Summary: This deliverable presents the work carried out to enable interactive SYCL execution within the ROOT's interpreter Cling, detailing the modernization of the interpreter (LLVM-18 upgrade, new pass manager, plugin infrastructure) and demonstrations of speedup over CPU-only execution. All the contributions are publicly available in ROOT (through corresponding PRs)

WP:	4		
Author(s):	Devajith Valaparambil Sreeramaswamy		
Editor:	Raja Appuswamy		
Leading Partner:	CERN		
Participating Partners:			
Version:	1.0	**Status:**	Draft
Deliverable Type:	Other	**Dissemination Level:**	PU
Official Submission Date:	06-Oct-2025	**Actual Submission Date:**	30-Sep-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	Technical update	CERN
1.0	Final draft	EUR

Authors

Author	Partner
Devajith Valaparambil Sreeramaswamy	CERN

Reviewers

Name	Organisation
Aleksandar Ilic	INESC
Vincent Heuveline	UHEI
Danilo Piparo	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This deliverable contains original unpublished work except where clearly indicated otherwise.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 presents the work carried out to enable interactive SYCL execution within the ROOT's interpreter, Cling, as part of the Task 4.4 in WP4 of the SYCLOPS project. The goal is to make SYCL kernels executable directly in ROOT, both at the command line and within Jupyter notebooks, thereby combining the portability of SYCL with the interactive analysis capabilities of ROOT.

To achieve this, significant modernization of Cling was required. The interpreter has been updated to LLVM 18, migrated to the new pass manager, and extended with plugin infrastructure so that compiler passes from external projects such as AdaptiveCpp can run seamlessly within Cling. These changes not only support SYCL integration but also reduce technical debt, align Cling more closely with upstream clang-repl, and ensure long-term maintainability.

On top of these foundations, the AdaptiveCpp SYCL implementation was integrated with Cling. This enables users to define, compile, and execute SYCL kernels interactively, with support validated on both the OpenMP (CPU) and CUDA (NVIDIA GPU) backends. The same SYCL code runs without modification across these backends, providing a portable and user-friendly workflow. Demonstrations were performed in both in ROOT and Jupyter notebooks, illustrating the interactive, exploratory analysis style central to ROOT.

Preliminary performance evaluations show clear benefits, showing both acceleration and portability across devices without changes to application logic.

All contributions are publicly available in the ROOT and AdaptiveCpp Github repositories through the corresponding pull requests. The current implementation is marked experimental in ROOT, with Linux as the primary supported platform.

This work establishes the technical foundation for SYCL-enabled interactive analysis in ROOT, opening the door to heterogeneous computing workflows for high-energy physics and other data-intensive domains.

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.

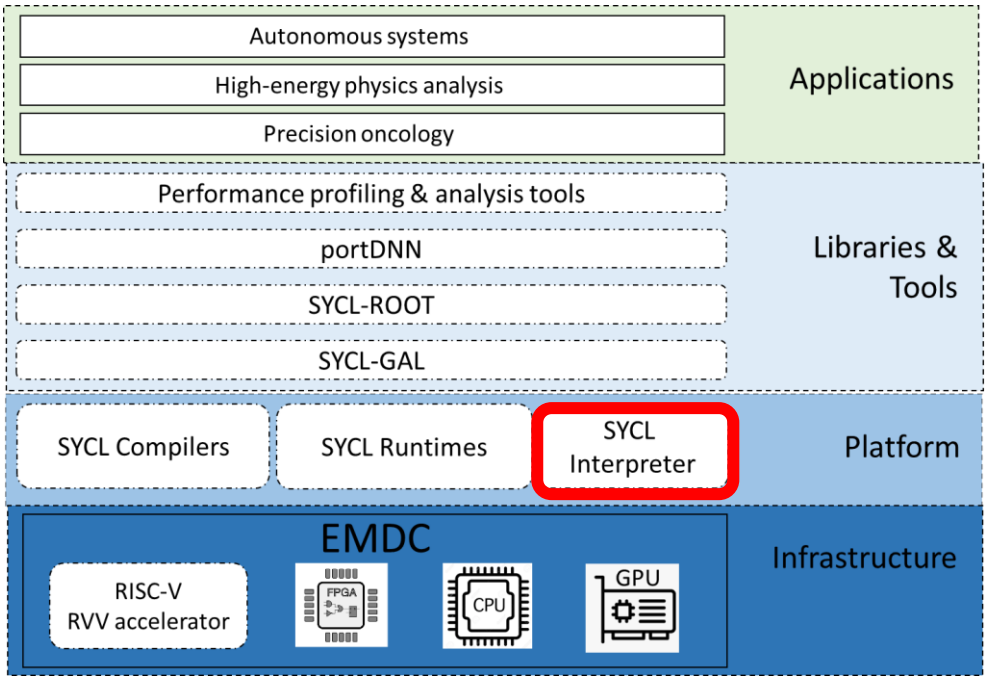

Figure 1. SYCLOPS architecture

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 presents the work carried out to enable **SYCL Interpreter** as highlighted in Figure 1 in the context of “Task 4.4 SYCL interpreter” of the SYCLOPS project. The objective is to extend Cling - ROOT’s C++ interpreter, with the ability to compile and execute SYCL kernels interactively, thereby combining ROOT’s interactive analysis capabilities with SYCL’s portable heterogeneous programming model. The deliverable situates this work within the broader SYCLOPS platform. ROOT and Cling represent the interpreter layer of the SYCLOPS software stack, enabling end-users to interactively develop, prototype, and validate algorithms. By integrating AdaptiveCpp into Cling, SYCLOPS provides a new interpreted SYCL execution environment that directly supports exploratory analysis workflows on CPUs and GPUs.

2 Background

The Large Hadron Collider (LHC) at CERN has collected more than 2 Exabytes of data since 2010. With the upcoming High-luminosity LHC, the data-rate is expected to increase by an order of magnitude. By extending Cling, ROOT's C++ interpreter to support a SYCL execution environment will provide a fast, flexible and interactive tool to explore this data.

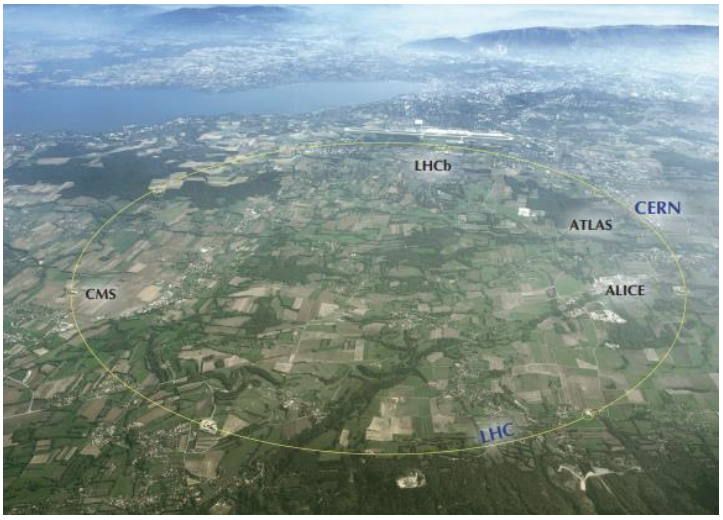

Figure 2. Position of the LHC tunnel

2.1 ROOT and Cling at CERN

ROOT is an opensource-data analysis framework developed at CERN and is widely used among the HEP community. It provides functionalities like data analysis, I/O, visualization and much more and written in C++ for high performance.

One of the core components of ROOT is Cling, a C++ interpreter built on top of LLVM/Clang. Before Cling, the role of interpreter was served by CINT, developed in the 1970s for data access and evaluating simple expressions. Over time, CINT evolved to support function evaluation and eventually code development. This was highly convenient, as it was fast (requiring no linking) and enabled rapid prototyping of algorithms, serving a path for interactive, exploration-driven development.

As CINT began to reach its limitations and C++ evolved rapidly, CERN and Fermilab developed Cling: a C++ interpreter based on just-in-time (JIT) compilation using/extending the functionality provided by LLVM/Clang, mapping the concept of an interpreter to a compiler (JIT). Cling can run interactively both in a Jupyter Notebook and in the command line as a REPL (read-eval-print loop), as shown in Figure 3, and it includes error recovery mechanisms to handle compile/runtime errors.

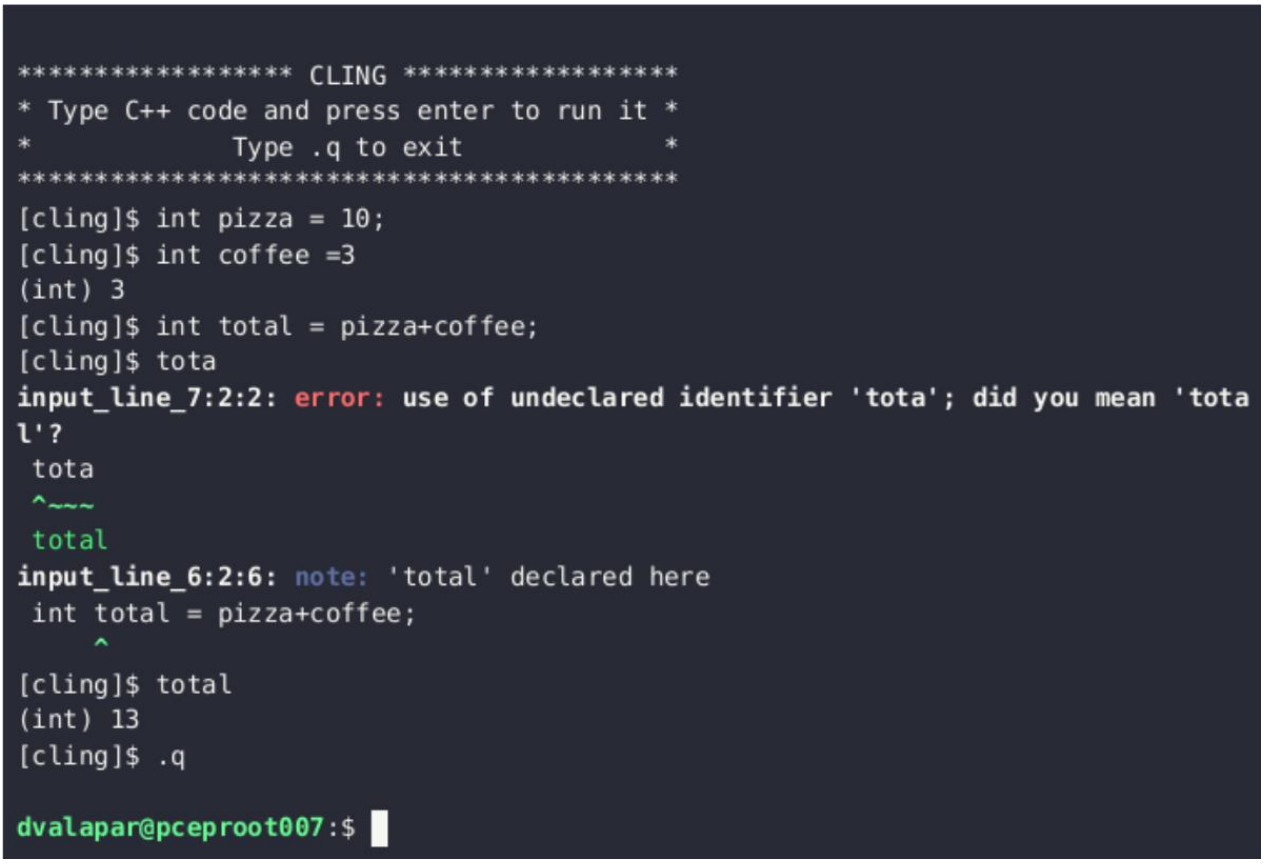

Figure 3. CLING (in a REPL environment)

2.2 Why SYCL in Cling?

By embedding SYCL execution capabilities directly in the interpreter, ROOT users gain:

1. The ability to prototype heterogenous algorithms on CPUs and GPUs.
2. Portable execution across multiple backend (OpenMP, CUDA, HIP, OpenCL) without changing application logic.
3. Integration with ROOT's analysis ecosystem and notebook-based workflows.

Within SYCLOPS, the platform layer includes two SYCL compiler toolchains: oneAPI DPC++ and AdaptiveCpp (formerly hipSYCL). For the purposes of this work, AdaptiveCpp was selected because its Clang/LLVM plugin model made it easier to integrate with Cling's infrastructure.

3 Foundations for Interactive SYCL in ROOT

Enabling SYCL execution within Cling required some pre-integration and modernization work in the interpreter. Cling, developed by CERN on top of LLVM/Clang has been in production for quite a while and carries the tests from over a decade ago. To host complex LLVM passes required by AdaptiveCpp and to support the requirements of AdaptiveCpp, some technical foundations had to be put in place.

This section describes the preparatory work that made SYCL integration easier, focusing on three main areas: (i) migration to the new LLVM pass manager, (ii) introduction of the plugin infrastructure, and the (iii) broader LLVM 18 upgrade. It also highlights ongoing efforts to align Cling more closely with the upstream clang-repl, removing technical debt and simplifying the interpreter architecture.

3.1 Migration to the new LLVM Pass Manager

Cling relied on LLVM's legacy pass manager. AdaptiveCpp, however implements its SYCL compilation passes using the new pass manager and is now standard across the ecosystem. Without this migration, AdaptiveCpp passes could not be scheduled along Cling's own transformations. A dedicated pull request (#14267) completed this transition, ensuring that Cling could host modern LLVM passes in its JIT pipeline. This also reduced the legacy APIs and infrastructure that we might need to maintain as these are no longer evolving upstream. Importantly, it enabled the co-existence of ROOT's JIT needs and AdaptiveCpp's SYCL passes, which was a prerequisite for task 4.4

3.2 Plugin support in Cling

To allow external projects (such as AdaptiveCpp) to extend Cling, it is necessary for Cling to be able to load clang plugins. With this feature, AdaptiveCpp can be integrated into Cling as a plugin. This also makes Cling extensible beyond SYCL. Other compiler passes/plugins can now be integrated easily.

3.3 LLVM 18 upgrade

Cling and ROOT have lagged behind LLVM's tip-of-tree due to complexity of integration and the behemoth testing infrastructure (>2000 tests). For SYCL support, this means bringing Cling to a more recent baseline. The LLVM 18 rebase (#15696) was a significant undertaking during this period. This required rebasing and adapting LLVM patches across ROOT and Cling, debugging many failing tests and addressing issues one-by-one. Around 50 patches were adapted on top of Clang that are required for ROOT/Cling.

The result is a modernized Cling interpreter, capable of leveraging LLVM's current features and APIs, providing a stable foundation for future upgrades. Several bugs/issues by the LLVM 18 upgrade were fixed, improving interpreter robustness beyond the scope of SYCL.

3.4 Alignment with clang-repl

Another major part of modernization is the ongoing alignment of Cling with clang-repl, LLVM's upstream REPL for C++. Cling was upstreamed to LLVM (as clang-repl), but the two diverged. Effort was put into closing this gap (by reducing the number of patches on top of clang), which is a significant undertaking, but ensures long-term sustainability.

4 Integrating AdaptiveCpp in the Interpreter

With the modernization of Cling in place, the next step was to integrate AdaptiveCpp, the open-source SYCL implementation developed at Heidelberg University in the context of SYCLOPS into the interpreter. AdaptiveCpp provides a single-source, single-pass SYCL compiler (SSCP) built on top of Clang/LLVM, and supports multiple heterogenous backends (OpenMp, CUDA, ROCm, OpenCL).

The main challenge was to adapt a toolchain originally designed as a clang plugin into Cling's interactive environment. The section explains how the integration was done and support and interactive SYCL execution both via REPL and Jupyter notebooks.

4.1 Plugin-based integration

AdaptiveCpp can be built as a clang plugin. The first approach was therefore, to extend Cling's new plugin infrastructure to dynamically load AdaptiveCpp:

- AdaptiveCpp pass plugin was compiled and loaded alongside Cling
- Cling could then invoke AdaptiveCpp passes during JIT compilation, similar to how Clad automatic differentiation plugin integrates with ROOT.
- This approach required building AdaptiveCpp against ROOT's LLVM toolchain to prevent runtime conflicts between LLVM builds and also required ROOT to be built with dynamic LLVM.

4.2 Static integration

Even though runtime plugin-based integration with dynamic LLVM was a straightforward path to SYCL support in Cling. ROOT has had problems with dynamic LLVM (#12156). LLVM option collisions occurred when both Cling and AdaptiveCpp attempted to register options at runtime. -DLLVM_LINK_LLVM_DYLIB=ON settings incompatible with ROOT's default build. To address the above issue, adaptiveCpp was integrated statically to ROOT. All the required passes were registered and executed via Cling's BackendPasses pipeline. This mirrors how Cling integrates its own JIT passes, providing a more stable runtime.

4.3 AdaptiveCpp challenges

A key difference between compiling full SYCL programs and interpreting SYCL code interactively is kernel discovery. During traditional compilation, AdaptiveCpp discovers all kernels in the translation unit at compile time. In Cling, code is submitted incrementally, and kernels may only be visible after multiple interpreter inputs. This required some modifications to AdaptiveCpp. Static initializers used for kernel cache registration were unreliable in a JIT context. Patches were introduced so that Cling could notify AdaptiveCpp whenever a new SYCL kernel was encountered, triggering cache updates dynamically. This ensured kernels defined interactively could be executed even if they are across notebook cells.

4.4 Backends and Platforms

Initial demonstrations of SYCL in Cling focused on OpenMP (CPU) and CUDA (NVIDIA GPU) backends. This was mostly because these backends were readily available for testing. These were validated in both REPL and Jupyter notebooks.

```
dvalapar@pceproot007:$ bin/root -l
root [0] .L test.cpp
root [1] test();
Running on NVIDIA GeForce RTX 3060 Ti
'+ptx86' is not a recognized feature for this target (ignoring feature)
'+ptx86' is not a recognized feature for this target (ignoring feature)
'+ptx86' is not a recognized feature for this target (ignoring feature)
1: Optimization Level::00
[AdaptiveCpp Warning] kernel_cache: This application run has resulted in new bin
aries being JIT-compiled. This indicates that the runtime optimization process h
as not yet reached peak performance. You may want to run the application again u
ntil this warning no longer appears to achieve optimal performance.
  A { 1, 2, 3, 4 }
+ B { 4, 3, 2, 1 }
-----------------
= C { 5, 5, 5, 5 }
root [2] test();
Running on NVIDIA GeForce RTX 3060 Ti
  A { 1, 2, 3, 4 }
+ B { 4, 3, 2, 1 }
-----------------
= C { 5, 5, 5, 5 }
root [3]
```

Figure 4. SYCL code running in a REPL environment

Figure 5. SYCL code running in a Jupyter notebook

4.5 Evaluation and testing of SYCL-enabled ROOT/Cling

Integration of AdaptiveCpp into Cling has been validated both functionally (kernels execute interactively in the interpreter) and quantitatively (measurable speedups over CPU-only execution)

Three types of validation were performed:

1. **Functional correctness:** Execution of SYCL kernels and verification across backends (OpenMP, CUDA) and results matched expected outputs.

2. **Preliminary performance evaluations:** Used GenVectorX library developed in SYCLOPS project, and more specifically ChangeCoord workload within GenVectorX, as a representative benchmark, to measure speedup over CPU-only execution. After initial JIT compilation for large sizes, interactive performance is consistently better than CPU as shown below.

N (elements)	CPU Time Avg (s)	SYCL Time Avg (s)	Speedup
2^{12}	0.000151	0.000539	0.28×
2^{14}	0.000563	0.000648	0.87×
2^{16}	0.007161	0.001983	3.61×
2^{18}	0.014682	0.004178	3.51×
2^{20}	0.036901	0.010681	3.45×
2^{22}	0.156627	0.045938	3.41×
2^{24}	0.585460	0.157825	3.71×
2^{26}	2.338510	0.564709	4.14×

Figure 6. SYCL-enabled Cling: SYCL vs CPU Performance Benchmark
Using the SYCL interpreter in CLING/ROOT for ChangeCoord example (part of GenVectorX tests)

3. Portability: Same SYCL kernels can run on OpenMP and CUDA backends.

All the benchmarks/validations were run on Linux platforms, using ROOT/Cling built with LLVM 18 and AdaptiveCpp integrated statically.

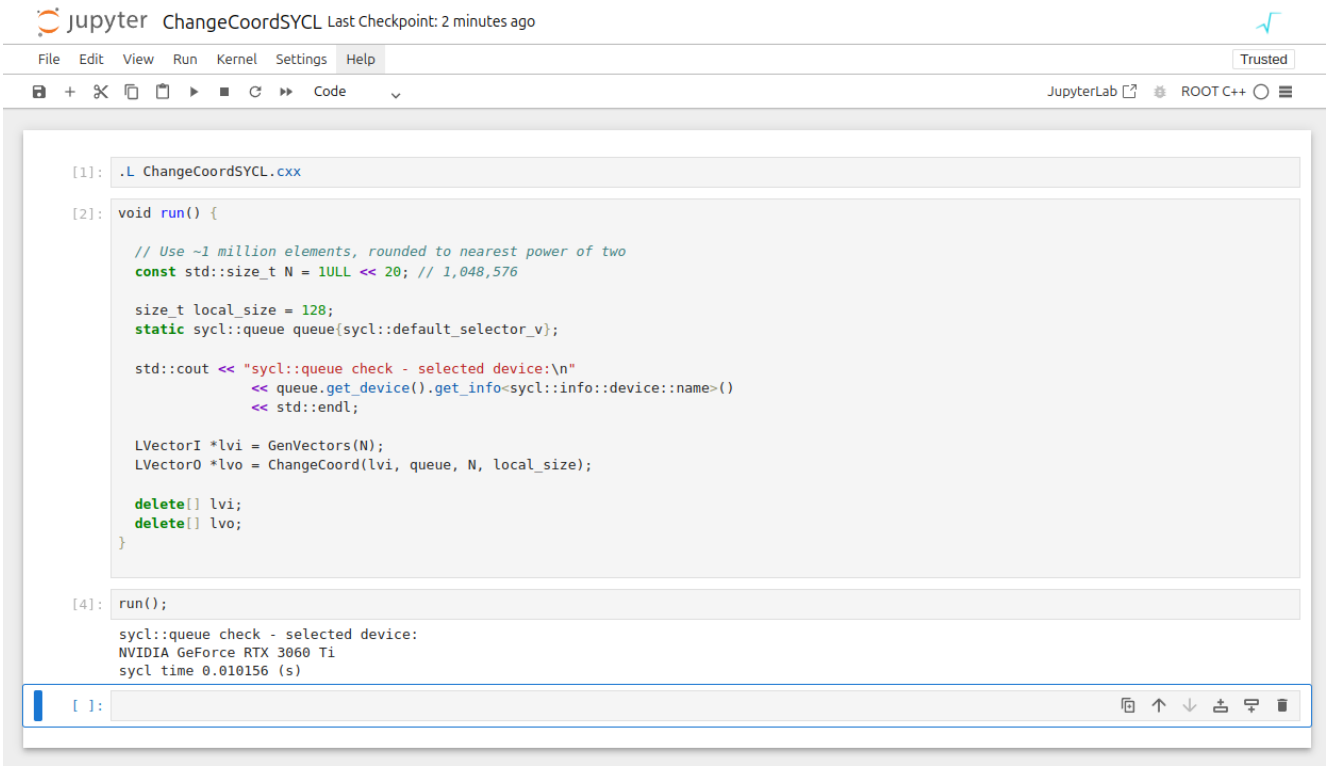

Figure 7. SYCL-enabled Cling: Performance Benchmark running on a Jupyter Notebook

4.6 Repository and PR Summary

Most of the work/PRs that were opened in ROOT (mostly Cling) can be found here. A substantial amount of work was done to modernize and fix issues/bugs. This section lists relevant PRs that were opened and merged in ROOT:

4. Main PR adding SYCL support in ROOT. The feature is exposed initially with an experimental_adaptivecpp flag to enable SYCL support (PR #17209)

5. Update of ROOT and Cling to LLVM 18, with numerous follow-up fixes to reduces failing tests from >200 to 0 (PR #15696)

6. Legacy to new LLVM pass manager, enabling Cling to run compiler passes from AdaptiveCpp (PR #14267)
7. Plugin support in Cling, enabling adaptiveCpp passes to be loaded dynamically as a plugin (PR #15169)
8. Refactoring to move Cling close to upstream without breaking existing functionality (PR #15374)
9. Fixed test failures on macOS cause by the update to the new pass manager (PR #14622)
10. And many more: addressing JIT lifetime, symbol resolution, module map issues etc. that might not be directly related but still relevant.

Some contributions were also made upstream to AdaptiveCpp to ensure compatibility with Cling and ROOT:

1. Enabling AdaptiveCpp to build and run in Cling without assertions (PR #1816)
2. Further patches upstreamed as part of integration effort (PR #1817, PR #1678)

A few upstream contributions to LLVM (PR #110092, PR #150215) were also made.

4.7 Ongoing work

Ongoing work includes LLVM 20 rebase (PR #17865), reviewing clang patches and trying to get ROOT/Cling with AdaptiveCpp run on RISC-V machine as part of integration efforts.

5 Conclusion

This deliverable concludes the work done in “Task 4.4: SYCL Interpreter” of WP4 in the SYCLOPS project. We presented the work carried out to extend ROOT’s C++ interpreter, Cling, to run SYCL kernels natively. The project achieved its primary goal enabling SYCL kernels to be compiled and executed interactively in ROOT’s interpreter and within Jupyter notebooks.

ROOT users can now experiment with SYCL kernels directly in their familiar analysis environments, combining portability of SYCL and the interactivity of Cling.

All contributions have been made publicly available through PRs in ROOT/Cling and AdaptiveCpp repositories. ROOT/Cling with adaptiveCpp support is available in root master when compiled with the flag “-Dexperimental_adaptivecpp”.

References

(1) https://root.cern/Cling/

(2) https://clang.llvm.org/docs/ClangRepl.html

(3) https://dl.acm.org/doi/abs/10.1145/3585341.3585351