Integration of the HIP Portability Layer into the ACTS Parallelization Project

Summer Internship Report

European Organization For Nuclear Research (CERN)

Yasmine Badr Elhouda Larbaoui
ESI Algiers – Higher National School of Computer Science
Github Repository
ky_larbaoui@esi.dz

Joana Niermann
European Organization for Nuclear Research (CERN)

July 2025 - August 2025

Abstract

This report presents the integration of the HIP portability layer into the ACTS parallelization project at CERN. ACTS (A Common Tracking Software) is a high-energy physics toolkit for charged-particle tracking, which includes detray, a modern GPU-friendly tracking detector description and propagation library. The project aimed to extend GPU support beyond NVIDIA's CUDA by enabling HIP (Heterogeneous-computing Interface for Portability), thereby supporting both NVIDIA and AMD accelerators.

The work involved configuring HIP support in the project's CMake-based build system, porting CUDA kernels to HIP while maintaining unified source files, and validating the implementation through unit and integration tests on detray. Benchmarks were executed on an AMD EPYC 7413 CPU, an NVIDIA RTX A5000 GPU, and an AMD Radeon RX 6700 XT GPU, with performance comparisons between CUDA, HIP on NVIDIA, and HIP on AMD platforms.

Preliminary results indicate that HIP on AMD achieves performance comparable to CUDA on NVIDIA hardware, while HIP on nvidia shows same performance as CUDA on NVIDIA. This integration demonstrates HIP's viability as a cross-platform solution and provides a significant step towards performance portability for HEP tracking software.

1 Introduction

Graphics Processing Units (GPUs) have become a cornerstone of modern high-performance computing (HPC). Originally designed for graphics rendering, GPUs are now indispensable accelerators for scientific computing, artificial intelligence, and large-scale simulations due to their massive parallelism and high throughput.

Over the past two decades, the rise of CUDA (Compute Unified Device Architecture) by NVIDIA has significantly shaped the GPU programming landscape. CUDA offers developers a rich set of tools and libraries, enabling optimized execution on NVIDIA hardware. However, this dominance comes at a cost: the lack of portability across hardware vendors. Applications written in CUDA are inherently tied to NVIDIA GPUs, creating a vendor lock-in that is increasingly problematic in heterogeneous computing environments.

In parallel, other hardware vendors, most notably AMD, have developed competitive GPU architectures. With the growing diversity of accelerator hardware in supercomputing facilities, the demand for portable programming models has become critical. This demand

has motivated the creation of HIP (Heterogeneous-Compute Interface for Portability), a C++ runtime API and kernel language designed by AMD. HIP provides a CUDA-like programming model but enables compilation to both NVIDIA and AMD GPUs, offering developers a potential pathway to performance portability without rewriting large codebases.

This report investigates the porting of an existing CUDA implementation to HIP, focusing on performance evaluation across heterogeneous hardware. The target application is a CUDA-based propagation benchmark from the Detray project, a geometry and propagation toolkit for particle detector simulations.

2 Technical Background

2.1 ACTS and Detray

The ACTS (A Common Tracking Software) project is a modular C++ toolkit designed for charged-particle tracking in high-energy physics experiments.

Within ACTS, the detray library provides an efficient geometry and propagation engine. Written in C++20, detray is designed for GPU acceleration. It uses a flat memory layout and data structures optimized for parallelism, suitable for large-scale simulations where millions of track propagations must be computed. Detray supports:

- Propagation of tracks in magnetic fields,
- Covariance transport for error estimation,
- Modeling of material interactions during particle propagation.

2.2 GPU Programming Models

Modern high-performance computing relies heavily on GPUs due to their massively parallel architecture. However, GPU programming models differ significantly depending on the hardware vendor.

CUDA is a proprietary framework developed by NVIDIA, exclusively optimized for NVIDIA GPUs. HIP, on the other hand, is an open-source framework developed by AMD, designed to bridge the gap between CUDA and other GPU architectures.

The Heterogeneous-computing Interface for Portability (HIP) API is a C++ runtime API and kernel language that lets developers create portable applications running in heterogeneous systems, using CPUs and AMD GPUs or NVIDIA GPUs from a single source code. HIP provides a simple marshalling language to access either the AMD ROCM back-end, or NVIDIA CUDA back-end, to build and run application kernels.

2.3 Portability in Scientific Software

For long-term sustainability of large research software frameworks like ACTS, avoiding vendor lock-in is essential. Porting to HIP provides a path toward hardware flexibility while maintaining performance.

HIP was selected within the CERN software stack because:

- It enables support for both NVIDIA and AMD hardware,
- It integrates with the ROCm toolchain,
- It allows incremental migration from existing CUDA implementations.

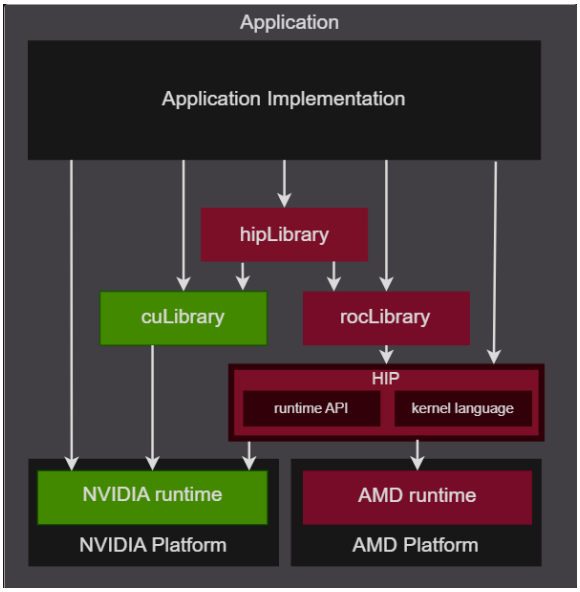

Figure 1: HIP Portability Layer for Heterogeneous GPU Platforms.

The main challenge addressed by HIP is portability without performance loss. This research investigates whether HIP represents a practical compromise between performance and portability for GPU-accelerated scientific applications.

3 Methodology

3.1 HIP Integration into the Build System

HIP support was added to the Detray build system through CMake by specifying compiler options, enabling HIP source compilation, and setting target GPU architectures. Flags for warnings, debugging, and C++20 compliance were configured to ensure correct compilation and performance across AMD and NVIDIA GPUs.

To support HIP as an additional accelerator backend in `detray`, the CMake configuration was extended at two levels: the main `CMakeLists.txt` and a dedicated module `hip-compiler-options.cmake`. The goal was to align HIP with the existing CUDA and SYCL backends while ensuring strict C++20 compliance and portability across NVIDIA and AMD hardware.

HIP was introduced as a first-class language with the following configurations:

- C++20 Standard Enforcement: These settings enforce standard C++20 across HIP code, mirroring CUDA and host configurations.

- Architecture Targeting : This explicitly targets the AMD Navi 22 architecture (Radeon RX 6700 XT).

```
set(CMAKE_HIP_ARCHITECTURES gfx1031)

set(CMAKE_HIP_STANDARD 20 CACHE STRING "The (HIP) C++ standard to use")
set(CMAKE_HIP_EXTENSIONS FALSE CACHE BOOL "Disable (HIP) C++ extensions")
```

Figure 2

- **Build Control Option:**

```
option(DETRAY_BUILD_HIP "Build the HIP sources included in detray" OFF)
```

Figure 3

Provides users explicit control over HIP source compilation.

- **Compatibility Safeguards:**

```
if(${DETRAY_BUILD_HIP} AND ${CMAKE_HIP_STANDARD} LESS 20)
    message(
        SEND_ERROR
        "CMAKE_HIP_STANDARD=${CMAKE_HIP_STANDARD}, but detray requires C++>=20"
    )
endif()
```

Figure 4

Prevents builds with outdated C++ standards.

- **Conditional Build Integration:**

Updated dependent options to include HIP alongside CUDA and SYCL:

```
cmake_dependent_option(
    DETRAY_BUILD_HOST
    "Build the host sources included in detray"
    ON
    "DETRAY_BUILD_CUDA OR DETRAY_BUILD_SYCL OR DETRAY_BUILD_HIP"
    ON
)
```

Figure 5

- **Language Detection:**

Integrated HIP detection via the existing `vecmem-check-language.cmake` mechanism.

The `hip-compiler-options.cmake` file centralizes HIP compiler flags for consistency across platforms:

- **Toolchain Location:**
`find_package(HIP REQUIRED)`

Ensures ROCm or HIP-enabled NVCC is available.

- **AMD Platform Configuration:Strict warning flags for code quality:**

```
if(
    ("${CMAKE_HIP_PLATFORM}" STREQUAL "hcc")
    OR ("${CMAKE_HIP_PLATFORM}" STREQUAL "amd")
)
    detray_add_flag( CMAKE_HIP_FLAGS "-Wall" )
    detray_add_flag( CMAKE_HIP_FLAGS "-Wextra" )
    detray_add_flag( CMAKE_HIP_FLAGS "-Wshadow" )
    detray_add_flag( CMAKE_HIP_FLAGS "-Wunused-local-typedefs" )
    detray_add_flag( CMAKE_HIP_FLAGS "-pedantic" )
endif()
```

Figure 6

- **NVIDIA Platform Configuration:** Debug symbol generation

```
if(
    ("${CMAKE_HIP_PLATFORM}" STREQUAL "nvcc")
    OR ("${CMAKE_HIP_PLATFORM}" STREQUAL "nvidia")
)
    detray_add_flag( CMAKE_HIP_FLAGS_DEBUG "-G" )
    detray_add_flag( CMAKE_HIP_FLAGS "--expt-relaxed-constexpr" )
endif()
```

Figure 7

This structured approach ensures HIP builds behave consistently while adapting to different backends (AMD ROCm vs. NVIDIA CUDA), providing a robust foundation for cross-platform GPU acceleration in `detray`.

3.2 Porting CUDA Kernels to HIP

To enable GPU acceleration beyond CUDA-only environments, selected CUDA source files were ported

to HIP. This ensures performance portability across heterogeneous architectures, particularly AMD and NVIDIA GPUs. The HIP adaptation preserves the same parallel execution model and memory hierarchy while introducing compatibility with the ROCm ecosystem.

3.3 HIP Porting and Test Workflow

The HIP integration into Detray followed a structured technical workflow to ensure GPU acceleration, portability, and correctness of particle tracking simulations.

Build System Integration:

- Configure CMake to enable HIP support.
- Set compiler options and specify target GPU architectures.

```
message(STATUS "Building detray HIP unit tests")
cmake_minimum_required(VERSION 3.21) # HIP langauge support requires minimum 3.21

find_package(HIPToolkit)

# Enable HIP as a language.
enable_language(HIP)

include(detray-compiler-options-hip)
```

Figure 8

- Link required libraries (`vecmem::hip`, GTest, ACTS).
- Define unit and integration test targets for automated verification.

```
detray_add_unit_test(hip_${CMAKE_HIP_PLATFORM}_${algebra}
    "detector_hip_kernel.hpp"
    "detector_hip.cpp"
    "detector_hip_kernel.hip"

    LINK_LIBRARIES GTest::gtest_main vecmem::hip  detray::core
    detray::algebra_${algebra} detray::test_common detray::test_utils HIP::hiprt
)

# Make hipcc interpret .cpp files as .hip files to make linking work
# (only needed for the amd backend)
if(
    ("${CMAKE_HIP_PLATFORM}" STREQUAL "hcc")
    OR ("${CMAKE_HIP_PLATFORM}" STREQUAL "amd")
)
    set_source_files_properties(detector_hip.cpp PROPERTIES LANGUAGE HIP)
endif()
```

Figure 9

This setup provides a consistent GPU build environment and ensures correct kernel functionality.

Kernel Porting:

- Adapt CUDA kernels to HIP using a single-source approach.
- Implement detector kernels to transfer volumes, surfaces, transforms, and masks to HIP-managed memory.
- Ported propagator kernels to handle tracks, steps, and magnetic field interactions.
- Preserve Runge-Kutta stepping and propagation constraints.
- These adaptations maintain correct particle propagation physics on GPU.

Memory Management:

- Allocate device and managed memory usine **`vecmem::hip::device_memory_resource`** and **`vecmem::hip::managed_memory_resource`**.
- Transfer data between host and device efficiently.
- Synchronize kernel launches using **`hipDeviceSynchronize`** and check for errors.
- This ensures consistent particle states and correct detector interactions.

Kernel Execution and Validation:

- Launch detector and propagator kernels using **`hipLaunchKernelGGL`** with optimized thread and block configurations.

```
    // run the test kernel
    hipLaunchKernelGGL(detector_test_kernel , dim3(block_dim), dim3(thread_dim) , 0, 0 ,
        det_data, volumes_data, surfaces_data, transforms_data, rectangles_data,
        discs_data, cylinders_data);
    // HIP error check
    DETRAY_HIP_ERROR_CHECK(hipGetLastError());

    DETRAY_HIP_ERROR_CHECK(hipDeviceSynchronize());
```

Figure 10

```
[ylarbaou@pcadp04 real_detray]$ ./detray_build/bin/detray_unit_test_hip_amd_array
Running main() from /mnt/ssd1/ylarbaou/real_detray/detray_build/_deps/googletest-src/googletest/src/gtest_main.cc
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from detector_hip
[ RUN      ] detector_hip.detector
INFO: Checking detector consistency...
WARNING: No entries in volume finder
INFO: Consistency check: OK
[       OK ] detector_hip.detector (617 ms)
[----------] 1 test from detector_hip (617 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (617 ms total)
[  PASSED  ] 1 test.
```

Figure 11

This workflow integrates HIP into Detray efficiently, combining GPU porting, memory management, and automated testing while preserving ACTS/Detray physics requirements for particle tracking and detector geometry handling.

4 Benchmarks

To evaluate HIP performance, dedicated benchmark executables were added to the `detray` build system.

4.1 HIP Propagation Benchmark Kernel

A dedicated propagation benchmark kernel was implemented in HIP (`propagation_benchmark.hip`). The kernel evaluates track propagation within `detray` geometries by instantiating propagators on the GPU and comparing synchronous vs. asynchronous propagation:

Kernel launch: `hipLaunchKernelGGL` with optimized block size (256 threads).

Detector setup: Detector views, magnetic field views, and actor states are copied to HIP-managed memory.

Propagation logic:

1. Initialize propagator state per track.
2. Update particle hypothesis (momentum, charge),
3. Run propagation with or without synchronization,
4. Collect results for validation and benchmarking.

Error handling: the `DETRAY_HIP_ERROR_CHECK` macro ensures correctness at runtime.

4.2 Benchmark Workflow

The benchmark workflow follows three stages:

Setup

- Allocate hip-memory memory : **`vecmem::hip::device_memory_resource managed_memory_resource`**.
- Copy actor states from host to device (`hipMemcpy`).
- Configure geometry, material, and grid files.

Execution

- Launch benchmark kernel with a configurable number of tracks (`n_samples`).
- Track propagation under magnetic fields is executed on GPU.

Validation

- Compare HIP outputs against CPU reference results.
- Measure throughput and runtime using `Google Benchmark`.
- Dump results in JSON format for plotting and analysis (`-benchmark_out`).

4.3 Benchmark Results

Benchmarks were executed on three different hardware setups:

- **CPU**: AMD EPYC 7413 .
- **NVIDIA GPU**: RTX A5000 (CUDA & HIP backends).
- **AMD GPU**: Radeon RX 6700 XT (HIP backend).

Benchmark Configuration and Data Generation

the benchmarks runs generate detailed JSON output files containing the measured propagation throughput for the given detector geometry, magnetic field, and physics configuration. The JSON files are subsequently used to create plots that visualize performance across different backends and algebra plugins.

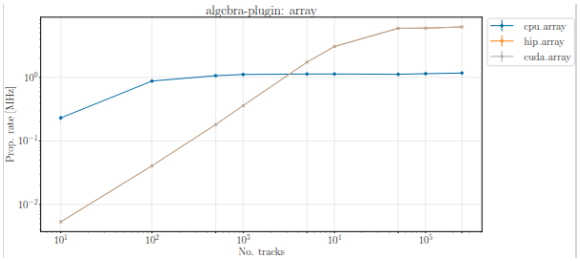

Figure 12: Thrughput Comparison of HIP & CUDA & CPU performance on NVIDIA Backends

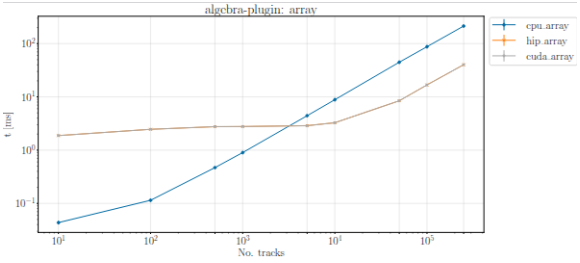

Figure 13: Latency Comparison of HIP & CUDA & CPU performance on NVIDIA Backends

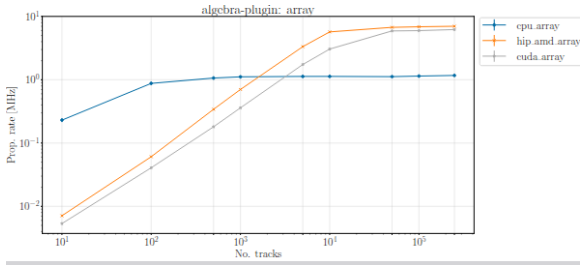

Figure 14: Throughput Comparison of HIP (AMD) & CUDA (NVIDIA) & CPU

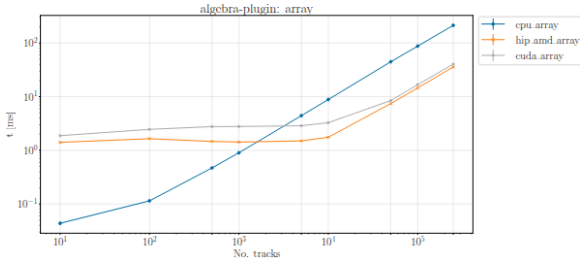

Figure 15: Latency Comparison of HIP (AMD) & CUDA (NVIDIA) & CPU

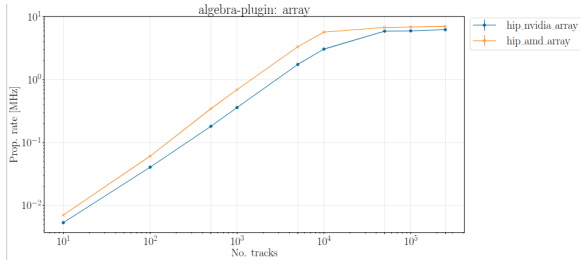

Figure 16: Throughput Comparison of HIP (AMD) & HIP (NVIDIA)

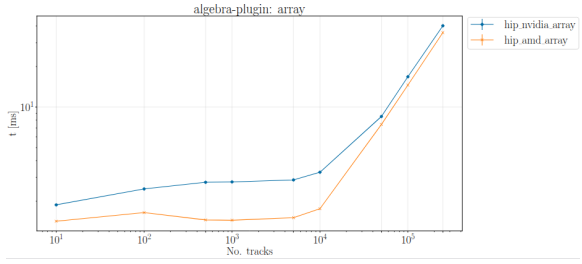

Figure 17: Latency Comparison of HIP (AMD) & HIP (NVIDIA)

Option	Description
–geometry_file	JSON description of the detector geometry.
–grid_file	Precomputed navigation grids for spatial acceleration.
–material_file	Material description (used for multiple scattering, energy loss, covariance transport).
–sort_tracks	Orders tracks for improved memory access and reproducibility.
–randomize_charge	Randomizes track charge to avoid systematic bias.
–eta_range -3 3	Pseudorapidity range of generated test tracks.
–pT_range 0.5 100	Transverse momentum range [GeV].
–search_window 3 3	Lookup window size for navigation in acceleration grids.
–covariance_transport	Enables propagation of track covariance matrices with material effects.
–bknd_name <backend>	Label identifying the benchmarked backend (e.g. cuda_array, hip_amd_array).
–benchmark_repetitions=5	Number of repeated runs for averaging.
–benchmark_out	Output file in JSON for later analysis.

Table 1: Benchmark configuration options and their descriptions

4.4 Benchmark Metrics :

To evaluate the performance of the propagation backends, we use the *throughput* as the main metric. Throughput measures how many tracks can be propagated per unit time, and is defined as

$$\text{Throughput} = \frac{N_{\text{tracks}}}{t_{\text{exec}}}, \tag{1}$$

where N_{tracks} is the number of propagated tracks and t_{exec} s is the execution time. In the benchmark plots, throughput is expressed in millions of tracks per second (MHz). The x-axis shows the number of tracks being propagated, while the y-axis shows the resulting throughput. This allows

us to study how different backends scale with increasing workload sizes. For comparison, latency (the time required to propagate a single track) is the inverse of throughput.

5 Results and Discussion

- For a small number of tracks, the CPU is faster, because GPUs need some setup time.

- When the number of tracks grows (thousands and more), GPUs become much faster than CPU.

- HIP CUDA on NVIDIA for this workload once past the tiny-N regime. That's a strong validation of the portability layer

- CUDA and HIP on NVIDIA give almost the same performance, which shows that HIP works well on NVIDIA without slowing things down

Regarding HIP AMD & NVIDIA :

- Small tracks: NVIDIA is a bit faster.

- Medium tracks: AMD actually performs a little better than NVIDIA.

- Large tracks: both become equal and reach the same maximum performance.

The integration of HIP into the ACTS/detray project was successful and allowed running the same code on both NVIDIA and AMD GPUs. The benchmark results show that HIP achieves almost identical performance to CUDA on NVIDIA, confirming that there is no real loss when using HIP instead of CUDA. On AMD GPUs, HIP provides competitive performance, sometimes slightly behind and sometimes even slightly better than NVIDIA, depending on the workload size. For very large numbers of tracks, both AMD and NVIDIA reach almost the same maximum throughput, proving that HIP ensures real performance portability.

This work demonstrates that HIP is a viable cross-platform solution that reduces vendor lock-in while keeping high performance. It lays the foundation for future optimizations on AMD hardware and opens the way to extend GPU support in ACTS to multiple architectures.

6 Conclusion and Future Work

The integration of HIP into the ACTS/detray project has been successfully completed, validated with comprehensive unit and integration tests. This work demonstrates that performance portability can be achieved across different GPU architectures while maintaining the computational efficiency required for high-energy physics simulations.

Key accomplishments include:

- Successful integration of HIP support into the Detray build system
- Porting of CUDA kernels to HIP with maintained performance
- Demonstration of cross-platform compatibility (NVIDIA + AMD GPU support)

For future work, several directions are promising:

- Optimize HIP kernels specifically for AMD hardware characteristics
- Extend the integratin of HIP into all ACTS Parts
- Compare HIP performance with other portability solutions like SYCL
- Investigate performance tuning for specific detector geometries and magnetic field configurations

This work provides a foundation for truly portable high-performance computing in high-energy physics, reducing vendor lock-in while maintaining the computational performance required for large-scale simulations.

7 Acknowledgements

I would like to express my sincere gratitude to my supervisor Joana Niermann at CERN for her exceptional guidance and support throughout this project. I also extend my thanks to the entire ACTS development team for their technical assistance and valuable insights. This internship has been an invaluable learning experience, both professionally and personally.

References

[1] ACTS Collaboration. (2023). *A Common Tracking Software (ACTS).* [Software]. https://github.com/acts-project/acts

[2] Detray Development Team. (2023). *Detray: A GPU-friendly tracking detector description and propagation library.* [Software]. https://github.com/acts-project/detray

[3] AMD. (2023). *ROCm Open Software Platform.* [Software]. https://github.com/ROCm

[4] AMD. (2023). *HIP: Heterogeneous-Compute Interface for Portability.* [Software]. https://github.com/ROCm/HIP

[5] NVIDIA. (2023). *CUDA Toolkit Documentation.* https://docs.nvidia.com/cuda/

[6] Pennycook, S. J., *et al.* (2021). *Performance Portability.* ACM Computing Surveys, 54(4), 1-37.