

An empirical performance-portability evaluation for Lorentz Vectors computations via SYCL

Monica Dessolet

ROOT team,
EP SFT, CERN

13th March 2024

- ① Background
- ② Migrating from CUDA to SYCL
- ③ Numerical Experiments
- ④ Conclusions and future work

Heterogeneous computing and portability

Current hardware accelerators in modern (pre)-exascale supercomputers

Machine	GPU vendors	language
Frontier	AMD	ROCm
Aurora	Intel	oneAPI
LUMI	AMD	ROCm
Polaris	NVIDIA	CUDA
Perlmutter	NVIDIA	CUDA

Two Types of Portability:

Functional Portability

The ability for a single code to run anywhere.

Performance Portability

The ability for a single code to run well anywhere.

Performance portability frameworks

Figure: Hardware support of portability layers¹

	CUDA	HIP	OpenMP Offload	Kokkos	dpc++ / SYCL	alpaka	std::par
NVidia GPU					<i>codeplay and intel/llvm</i>		<i>nvc++</i>
AMD GPU				<i>feature complete for select GPUs</i>	<i>via openSYCL and intel/llvm</i>	<i>hip 4.0.1 / clang</i>	
Intel GPU		<i>CHIP-SPV early prototype</i>		<i>native and via OpenMP target offload</i>		<i>prototype</i>	<i>oneAPI::dpl</i>
multicore CPU							<i>g++ & tbb</i>
FPGA						<i>via SYCL</i>	

¹Mohammad Atif et al. "Evaluating Portable Parallelization Strategies for Heterogeneous Architectures in High Energy Physics". In: *arXiv preprint (2023)*.

GenVector package

GenVector is a large package (~ 11k lines) within ROOT enabling fundamental operations for HEP analysis

- 2, 3 and 4 dimensional physical vectors
- coordinate systems: Cartesian, Polar, and so forth
- user can specify the underlying scalar type, say single or double precision floats
- more advanced operations: rotations, Lorentz and Poincare transformations

GenVectorX

Extend GenVector to **parallel execution** on NVIDIA GPUs via CUDA and other backends via SYCL, while retaining **performance** execution and **user friendly API**.

SYCL computational model

SYCL is a cross-platform abstraction layer that for a “single-source” code using modern ISO C++. Two notable implementations:

- Intel(R) oneAPI Toolkit
- AdaptiveCPP² (before openSYCL/hipSYCL)

Memory management

- buffers and accessors (BUF), handled entirely by the SYCL runtime
- unified shared memory pointers (PTR), explicitly defined by the user

Kernel definition

- lambda functions, optionally named
- named function objects, same functionality as any C++ function object

²Aksel Alpay et al. “Exploring the Possibility of a HipSYCL-Based Implementation of OneAPI”. In: *International Workshop on OpenCL. IWOC'L'22. ACM, 2022.*

From CUDA to SYCL³

Figure: Index Hierarchy

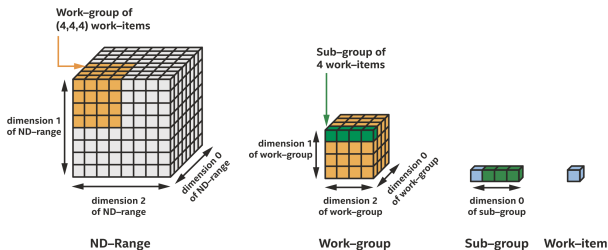


Table: Execution model

CUDA	SYCL
Thread	work-item
Warp	sub-group
Block	work-group
Grid	ND-range

³James Reinders et al. *Data Parallel C++: Mastering DPC++ for Programming of Heterogeneous Systems using C++ and SYCL*. 2021.

Example: Invariant Masses

Figure: SYCL function object

```
template <class Scalar, class Vector>
class InvariantMassesKernel
{
public:
    InvariantMassesKernel
        (LVector *v1, LVector *v2, Scalar *m, size_t
         n)
        : d.v1(v1), d.v2(v2), d.m(m), N(n) {}

    void operator()(sycl::nd_item<1> item) const
    {
        size_t id = item.get_global_id().get(0);
        if (id < N)
        {
            LVector w = d.v1[id] + d.v2[id];
            d.m[id] = w.mass();
        }
    }

private:
    LVector d.v1;
    LVector d.v2;
    Scalar d.m;
    size_t N;
};
```

Figure: CUDA Kernel

```
template <class Scalar, class LVector>
__global__ void InvariantMassesKernel
(LVector *v1, LVector *v2, Scalar *m, size_t N)
{
    int id = blockDim.x * blockIdx.x + threadIdx
        .x;
    if (id < N)
    {
        LVector w = v1[id] + v2[id];
        m[id] = w.mass();
    }
}
```


Code Similarity

Given a problem and a set of platforms (here CUDA, SYCL and CPU), code similarity is a **measure of similarity between code bases** that takes value in $[0, 1]$. It's based on the **union and intersection of the sets of code lines** for each platform.

Table: Code Similarity of GenvectorX vs GenVector

Similarity	Union	Intersection	Platform	Problem
0.9694	10365	10048	CUDA	Invariant Masses
0.9715	10276	9983	SYCL	Invariant Masses
0.9693	10358	10041	CUDA	Boosting
0.9716	10335	10042	SYCL	Boosting

Experimental setting

Four computing environments:

- 1 NVIDIA GeForce RTX 3060 using CUDA 12.2
- 2 NVIDIA L4 using CUDA 12.3
- 3 NVIDIA A100 40GB PCIe using CUDA 12.2
- 4 AMD MI250X using ROCm 5.3.3

The main features of the GPUs that are relevant in our case are summarized here:

Table: GPUs specification

	RTX3060	L4	A100	MI250X
Vendor	NVIDIA	NVIDIA	NVIDIA	AMD
Architecture	Ampere	Ada Lovelace	Ampere	CDNA2
CUDA Capability	86	89	80	-
Global Memory (GB)	12	24	40	128
Peak FP32 (TFLOPs)	12.74	30.3	19.5	95.7
Peak Bandwidth (GB/s)	360.0	300.0	1555.0	3200.0

NVIDIA GPUs: Scaling

Figure: RTX3060

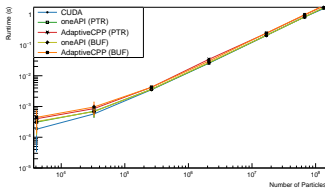


Figure: L4

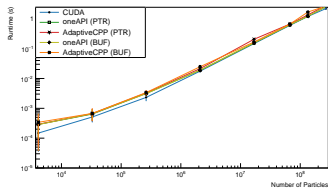
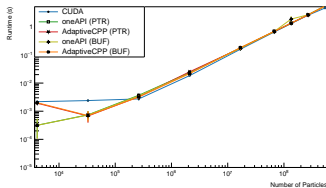


Figure: A100



NVIDIA GPUs: Time analysis

Figure: RTX3060

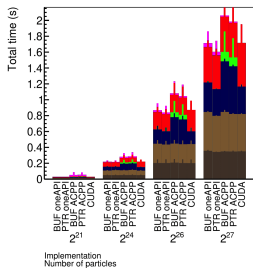


Figure: L4

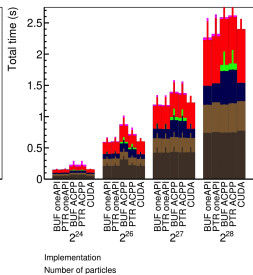
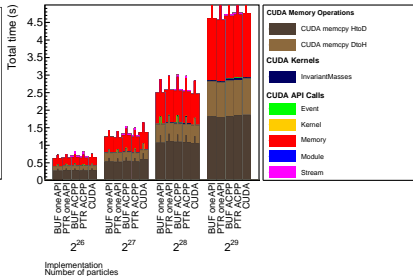


Figure: A100



AMD MI250X GPU

Figure: Time analysis

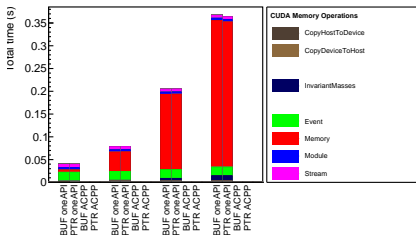
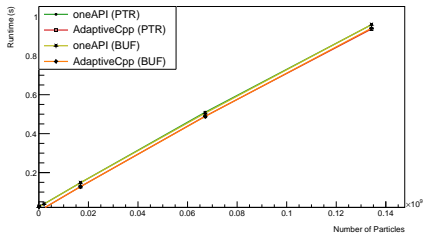


Figure: Scaling



What we did so far




- We detailed the **migration to both SYCL and CUDA** of a large, complex, C++ code base, providing guidance and insights regarding the analogies and differences between the two frameworks for other developers interested in migrating their own codes.
- We evaluated **code divergence** of GenVectorX, to estimate the benefits in maintaining a single source code without specializing regions of code for specific targets.
- We provided a detailed **performance analysis** of the migrated SYCL code on different platforms and architectures.

What needs to be done

- Integration within ROOT is work in progress.
- Performance evaluation on other platforms and architectures, e.g. Intel GPUs, Risc-V.
- Extension to jagged arrays management: real High energy physics analysis deal with independent events, such as particle collisions, decays, or cosmic rays, which can each be fully analyzed in isolation, though the analysis of each event may be very complex.
- Dealing with filtering and skimming: in real data, among all the collisions in an event we may have very few which are significant and need to be kept for further computations.

Thank you for listening!

References I

-  **Alpay, Aksel et al.** “Exploring the Possibility of a HipSYCL-Based Implementation of OneAPI”. In: *International Workshop on OpenCL. IWOCCL'22*. ACM, 2022.
-  **Atif, Mohammad et al.** “Evaluating Portable Parallelization Strategies for Heterogeneous Architectures in High Energy Physics”. In: *arXiv preprint (2023)*.
-  **Reinders, James et al.** *Data Parallel C++: Mastering DPC++ for Programming of Heterogeneous Systems using C++ and SYCL*. 2021.

Code Similarity: Evaluation

Code similarity is calculated as:

$$1 - CD(a, p, H), \quad CD(a, p, H) = \binom{|H|}{2}^{-1} \sum_{(i,j) \in H \times H} d_{i,j}(p, a) \quad (1)$$

where $d_{i,j}(p, a)$ represents the distance between the source code required to solve problem p using application a on platforms i and j (from platform set H). We adopt the Jaccard distance defined as $d_{i,j}(p, a) = 1 - s_{i,j}(p, a)$, where

$$s_{i,j}(p, a) = \left| \frac{c_i(a, p) \cap c_j(a, p)}{c_i(a, p) \cup c_j(a, p)} \right|. \quad (2)$$

Here, c_i and c_j represent the set of source lines required to compile application a and execute problem p for platforms i and j , respectively.