OneAPI for LHCb High Level Trigger

Apostolos Karvelas

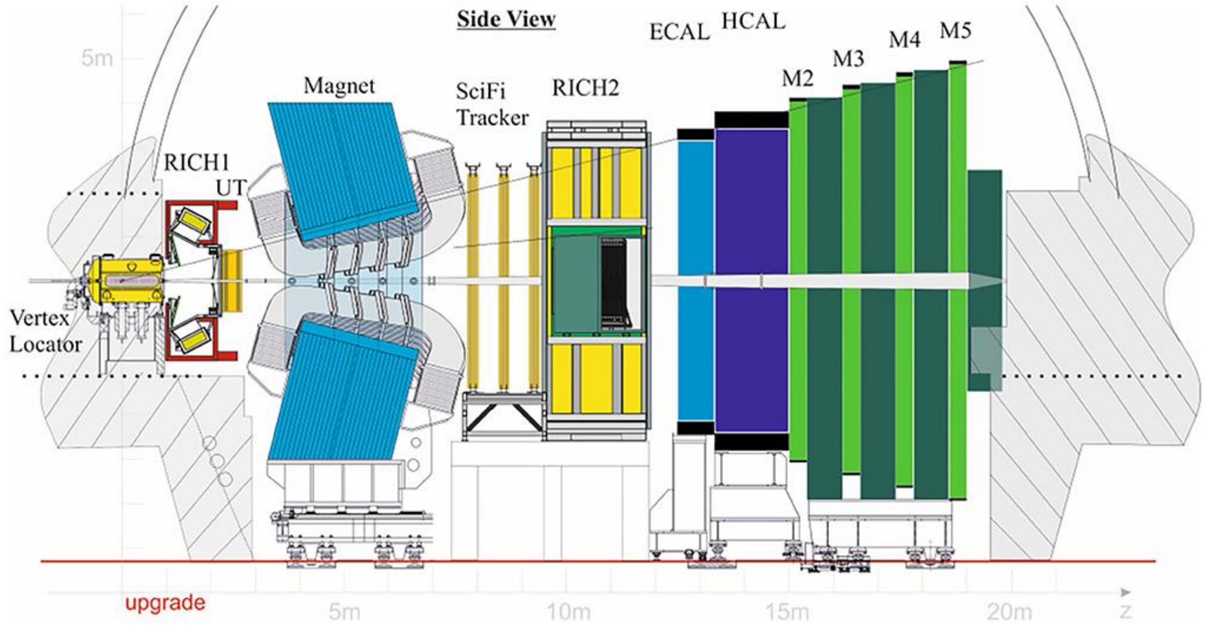
Supervisors: Niko Neufeld & Daniel Hugo Campora Perez

01.07.2022 – 31.12.2022

LHCb ALLEN

Allen is a fully GPU-based implementation of the first level trigger.

It has the capability to handle 40 Tbit/s of data in real-time and can execute a broad range of pattern recognition operations.



LHCb ALLEN

The Allen framework is a modular, scalable and flexible framework for LHCb physics reconstruction on accelerators with:

- Multi-threaded, pipelined, configurable framework.
- Multi-event scheduler, event batches support.
- Custom memory manager, flexible datatypes.
- Built-in validation. Generation of graphs with ROOT.

LHCb ALLEN

Allen has 89 kernel functions and more than 50000 lines of code.

500 of those lines produce the majority of the CUDA to SYCL migration.

Allen compiles natively in:

- CUDA
- HIP
- CPU
- SYCL

oneAPI

•**Open**, **cross-industry** framework.

•**Provides a unified and standards-based** programming model.

•**Tools, libraries**, and **APIs**.

SYCL & oneAPI

•**oneAPI** includes **support** for **SYCL**.

•**High performance, productivity, and portability.**

•Able to run on **various hardware architectures**.

SYCL Implementation

There is a complete SYCL implementation.

The entirety of Allen has been migrated:

- All sequences are operational.
- Yielding similar result.
- Up-to-date and supports the latest version.

Achievements

Key technical aspects that facilitated the migration process:

- Library types and functions not yet supported by SYCL has been developed.
- 3-dimensional kernel invocation has been developed, similar to the one found in CUDA.
- A shared memory model has been created, mimicking the functionality of CUDA's shared memory system.
- Warp-Level Primitives have been developed specifically for SYCL.

Functions

Integer, casting and half precision functions.

- **Complete C++** code from scratch.
- **Combination of C++** code and **already implemented SYCL** functions.

```
inline uint16_t __float2half(const float f)
{
    const half_t temp = static_cast<const half_t>(f);
    const uint16_t* temp_u16_p = reinterpret_cast<const uint16_t*>(&temp);
    return *temp_u16_p;
}

inline int __popcll(const int64_t n)
{
    const int32_t low_n = static_cast<int32_t>(n);
    const int32_t high_n = static_cast<int32_t>(n >> 32);
    return sycl::popcount(high_n) + sycl::popcount(low_n);
}
```

Data types and structures

Data structures: span

- **Lightweight** contiguous sequence of values.
- **Non-owning** type.

Data types: dim3 and float3

- Easily **migrate CUDA** functions

Kernel Invocation

Kernel function invocation:

- Queue in FIFO order
- Running the same kernel function multiple times in parallel using 3-dimensional nd_range.

```
void invoke_device_function(
    Fn&& function,
    const dim3& grid_dim,
    const dim3& block_dim,
    const Allen::Context& context,
    const Tuple& invoke_arguments,
    std::index_sequence<I...>){
  context.queue().submit([&](sycl::handler& h) {
      h.parallel_for(
          sycl::nd_range<3>(
              sycl::range<3>(grid_dim.x * block_dim.x, grid_dim.y * block_dim.y, grid_dim.z * block_dim.z), sycl::range<3>(block_dim.x, block_dim.y, block_dim.z)),
          [invoke_arguments, function](sycl::nd_item<3> item) {
              auto invoke_arguments_copy = invoke_arguments;
              if constexpr (ConfigurationExists) {
                std::get<IndexOfArgumentRefManager>(invoke_arguments_copy).config.m_item = &item;
              }
              function(std::get<I>(invoke_arguments_copy)...);
      });
  });
}
```

Device Shared Memory

- A **buffer** is generated in the **device memory**, which can be accessed by all work-items in the nd_range kernel.
- **Dividing the buffer** and **assigning each section to every work-item** in the kernel, creating CUDA __shared array within the kernel.

```
define SHARED(_name, _size, _span) \
   decltype(_span)::value_type* _name = _span.data() + \
    _size * (parameters.config.blockIdx<2>() * parameters.config.gridDim<1>() * parameters.config.gridDim<0>() + \
   parameters.config.blockIdx<1>() * parameters.config.gridDim<0>() + parameters.config.blockIdx<0>());
```

Warps & Sub-groups

Most of the warp-level primitives and atomics of CUDA are **implemented in SYCL.**

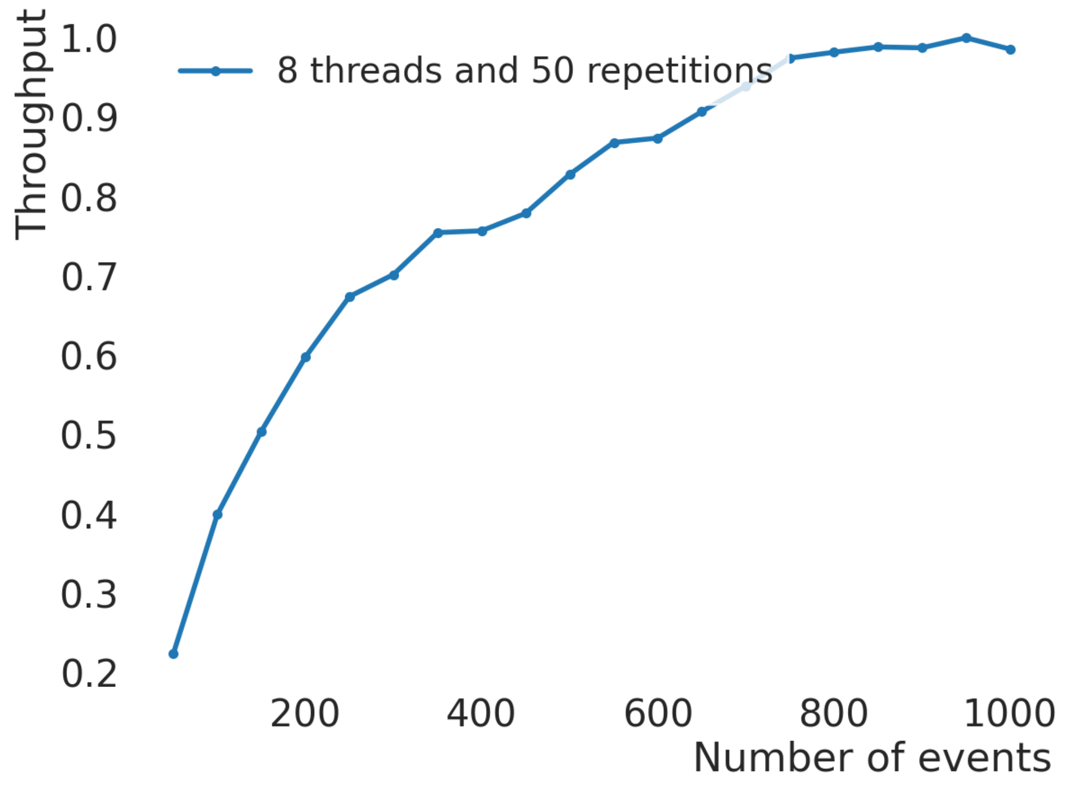
```
__syncwarp -> item->get_sub_group().barrier

__shfl_sync -> item.get_sub_group().sub_group::shuffle

__ballot_sync ->
uint32_t ballot(const sycl::sub_group& sg, int predicate) {
    size_t id = sg.get_local_linear_id();
    uint32_t local_val = (predicate ? 1u : 0u) << id;
    return sycl::reduce_over_group(sg, local_val, sycl::plus<>());
}

atomicAdd ->
template<class T, class S>
T atomicAdd(T* address, S val)
{
    cl::sycl::atomic<T> atomic {cl::sycl::global_ptr<T> {address}};
    return atomic.fetch_add(val);
}
```

Results



Maximum throughout: 8820 events/s

Plateau could be due to:

- Memory bandwidth limitations.
- Thread contention.

What’s Next

- Benchmarking Allen on FPGAs.
- Fine-tuning for improved efficiency results.
- Performing throughput optimizations targeting Intel’s GPUs.

CERN

https://home.cern