Allen: A software framework for the GPU High Level Trigger 1 of LHCb

Daniel Hugo Cámpora Pérez, on behalf of the LHCb Collaboration
dcampora@cern.ch

CHEP, November 4th, 2019

Nikhef
University of Maastricht

The LHCb Upgrade

The LHCb detector and Data Acquisition system will be upgraded to prepare for a new data taking period in 2021. The design of the upcoming trigger[1] will be challenging due to two factors:

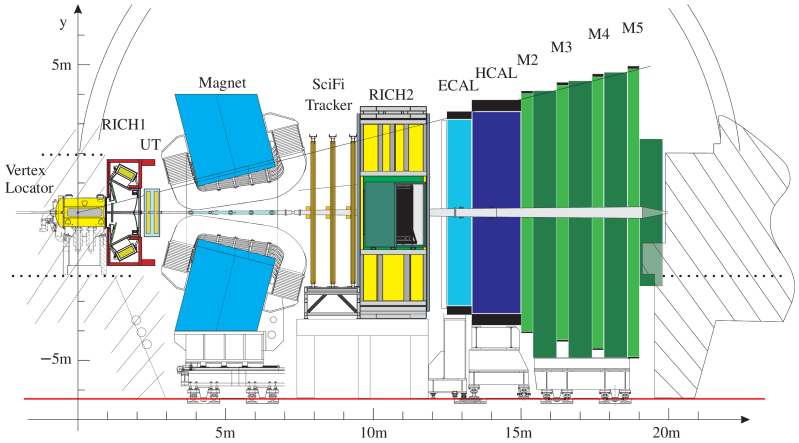
- LHCb will remove its hardware level trigger, turning to a full-software trigger
- Luminosity will increase to $2 \times 10^{33} \mathrm{cm}^{-2} \mathrm{s}^{-1}$

LHCb will filter data at a rate of 40 Tbit/s in a software trigger

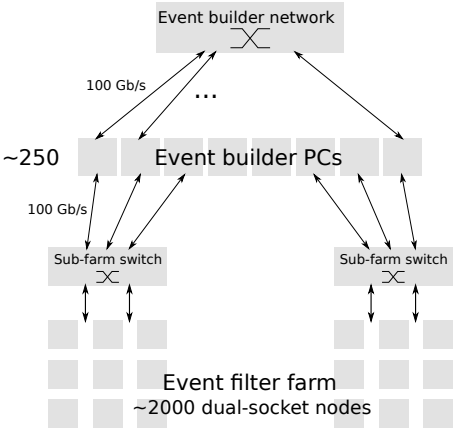
[1] *LHCb Trigger and Online Upgrade Technical Design Report*, https://cds.cern.ch/record/1701361

A refresher of the LHCb detector layout

LHCb is upgrading most of its components for Run 3. Most subdetectors will be upgraded.



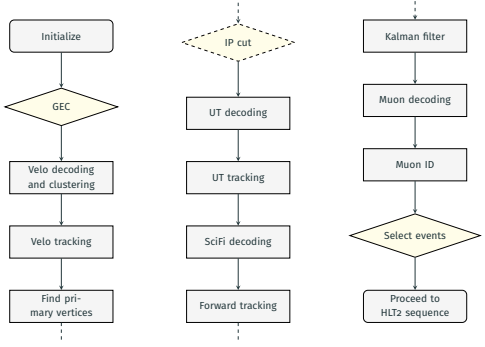
Slice of LHCb Online System at Run 3



High Level Trigger 1 at LHCb

The first stage of software trigger, also known as *High Level Trigger 1*, is a critical stage of the software reconstruction. It must make a decision in near-time over all input data, at the collision rate.

The entire HLT1 involves the decoding, clustering and track reconstruction of all tracking detectors at LHCb, as well as the Kalman filter, PV finder and trigger decision algorithms.



A GPU HLT1 for the LHCb Upgrade

We are proposing to run the entire LHCb HLT1 on GPUs. Our proposal features:

- *Full software HLT1 sequence* — The baseline HLT1 programme has been fully implemented.
- *Scalability and versatility* — Growing codebase encompassed by framework.
- *Compatibility* – The codebase compiles across architectures.
- *Compact solution* — Strategic placement in the Event Builders allows for cost savings.

Allen

The Allen framework

The *Allen* framework is a compact, scalable and modular framework, built for running the LHCb HLT1 on GPUs.

Requirements

- A C++17 compliant compiler, boost, zeromq
- CUDA v10.0

Features

- Configurable static sequences.
- Pipelined stream sequence.
- Custom memory manager, no dynamic allocations, SOA datatypes.
- Built-in validation with Monte Carlo.
- Optional compilation with ROOT for generation of graphs.
- Integration with Gaudi build system.
- Cross-architecture compatibility.

Framework scalability and modularity

As the codebase of Allen grows, the underlying framework remains scalable and accessible, requiring as little framework-specific knowledge as possible, and using common practices where possible.

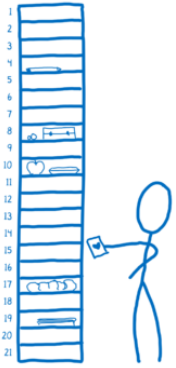
Sequence configuration is done at compile time in Allen. Adding / removing an algorithm is as easy as modifying one line in a sequence configuration file.

```
SEQUENCE_T(
  velo_estimate_input_size_t,
  prefix_sum_velo_clusters_t,
  velo_masked_clustering_t,
  velo_calculate_phi_and_sort_t,
  velo_fill_candidates_t,
  velo_search_by_triplet_t,
  velo_weak_tracks_adder_t)
```

Memory management

In Allen, we allocate memory at the **startup** of the application. A custom memory manager assigns memory segments on demand.

- All data dependencies and memory assignments are resolved at compile-time.
- There are no dynamic memory allocations.
- GPU memory is plenty for the Allen use case.



CPU support

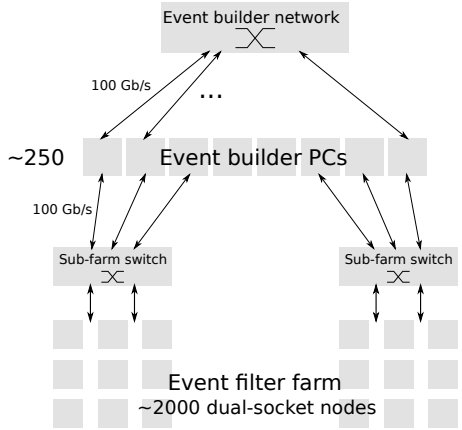
Allen is compatible with CPU architectures.

- *No extra maintenance* – Implemented as a compilation of the Allen codebase.
- Multi-threaded.
- Cross-architecture (x86_64, Power, ARM).

Differences in results are very small, stemming from diverse architectures / compilers. Similar differences are observed within same processor by changing vectorization target configuration.

Integration

Baseline scenario without GPUs



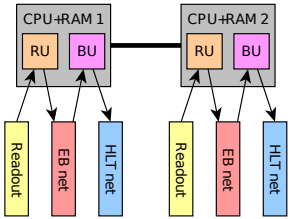
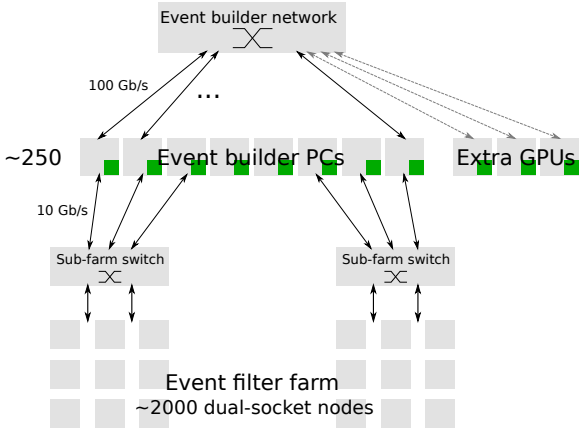


Figure 1: Event builder PC.

Event builders with GPUs



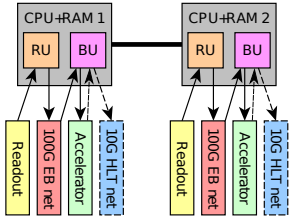
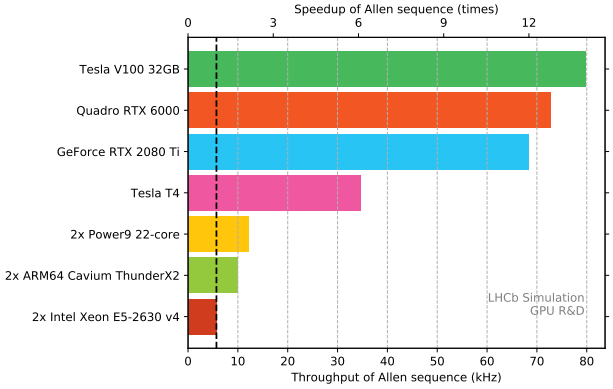


Figure 2: GPU-equipped event builder PC.

Target processing rate

In order to be able to perform the HLT1 filter inside the event builder with GPUs, the full throughput of collisions must be processed in near-time.



Integration test

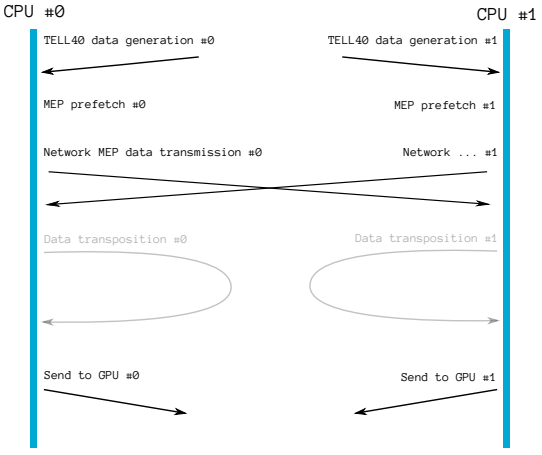
It is important to understand the feasibility of such a system, taking into account as many factors as possible to replicate a prospective production setup.

To name a few: CPU consumption, memory consumption and throughput, airflow, thermal stability, GPU performance stability, network throughput...

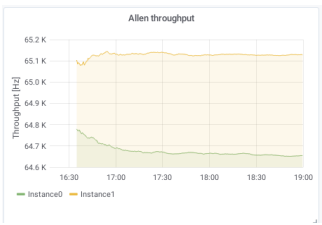
We setup a single server to test these. Configuration on each socket:

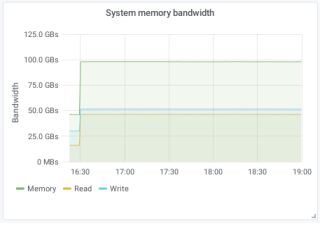
- Readout card (TELL40)
- Network card (Infiniband EDR card 100 Gbps)
- Graphics card (Gigabyte GeForce RTX 2080 Ti)

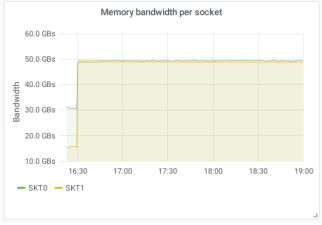
Integration test dataflow

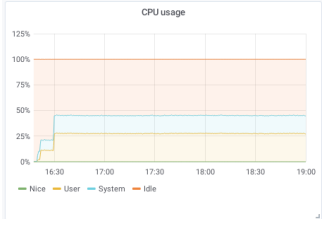


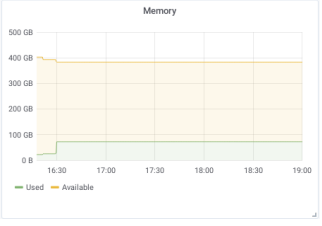
Selected integration test results

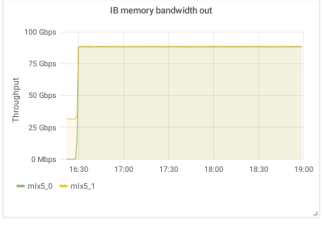












Conclusions

A GPU HLT1 framework: Allen

- Allen, a GPU HLT1 framework, has been presented.
- A compact GPU HLT1 system can run with O(500) cards.
- Check out D. vom Bruch's plenary talk on Wednesday for physics details.

- The framework permits defining a static sequence of algorithms.
- Data is handled by custom memory manager, one instance per thread.
- Cross-architecture compatibility.
- The framework is modular and scalable.

- Integration with the system is in the works.
- Allen compiles with the LHCb Gaudi build system.
- A first integration test was positive.

Future work

A more comprehensive integration test is in preparation:

- A group of servers will be tested.
- Monitoring of the node will be added.
- Allen will be run without doing data transposition.

Other features are in the works:

- Configuration of sequences and algorithms at configuration time.
- Monitoring of Allen.
- Improvements to GPU reconstruction algorithms.

Thanks!

Thanks a lot to all people involved in the development of Allen!

`https://gitlab.cern.ch/lhcb-parallelization/allen`

Backup

CPU support: CUDA code

Consider the following CUDA code:

```
1  constexpr int N = 32;
2  __global__ void saxpy_plus(float* x, float* y, const float a) {
3    y[threadIdx.x] = x[threadIdx.x] * a + y[threadIdx.x];
4    __syncthreads();
5    if (threadIdx.x < 10) {
6      y[i] += 1;
7    }
8    if (threadIdx.x == 11) {
9      y[threadIdx.x] += 20;
10   }
11 }
12 ...
13 saxpy_plus<<<1/*blocks*/, M, /*threads*/ N>>>(x, y, a);
```

- The number of threads is set statically to N=32.
- The statement in line 3 makes assumptions of the number of threads.
- The two `if` statements also make assumptions of the number of threads (they require at least 11 threads).

CPU support: Flexible code

In contrast, consider this code:

```
constexpr int N = 32;
__global__ void saxpy_plus(float* x, float* y, const float a) {
  for (int i=threadIdx.x; i<N; i+=blockDim.x) {
    y[i] = x[i] * a + y[i];
  }
  __syncthreads();
  for (int i=threadIdx.x; i<10; i+=blockDim.x) {
    y[i] += 1;
  }
  if (threadIdx.x == 0) {
    y[11] += 20;
  }
}
...
saxpy_plus<<<1/*blocks*/, M, /*threads*/ 1>>>(x, y, a);
```

- A call to `saxpy_plus` with any number of threads will produce the same result.

CPU support: A CPU version

If the code has *block-dimension strided for loops*, and all `if` statements for a single thread refer to threads of index 0, then with some macros and function definitions it is possible to compile the code for CPUs:

```
// Definitions
#define __global__
#define __syncthreads()
struct GridDimensions { uint x, y, z; };
struct BlockIndices { uint x, y, z; };
struct BlockDimensions { uint x=1, y=1, z=1; };
struct ThreadIndices { uint x=0, y=0, z=0; };
thread_local GridDimensions gridDim;
thread_local BlockIndices blockIdx;
thread_local BlockDimensions blockDim;
thread_local ThreadIndices threadIdx;

...

// Kernel call excerpt
gridDim = {num_blocks.x, num_blocks.y, num_blocks.z};
for (unsigned int i = 0; i < num_blocks.x; ++i) {
  for (unsigned int j = 0; j < num_blocks.y; ++j) {
    for (unsigned int k = 0; k < num_blocks.z; ++k) {
      blockIdx = {i, j, k};
      function(std::get<I>(invoke_arguments)...);
    }
  }
}
```

Integration test setup (1)

For our first test, we setup a single server with:

- Supermicro server
- 2 $\times$ Intel Xeon Silver 4114
- 376 GB of memory
- Differences wrt. candidate server: Cascade Lake (better PCIe performance), different chassis (better thermals)

It has three PCIe Gen3 16x slots per socket. Two of those are double width. Configuration on each socket:

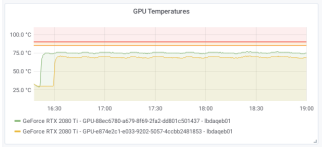
- Infiniband EDR card (100 Gbps)
- TELL40
- Gigabyte GeForce RTX 2080 Ti

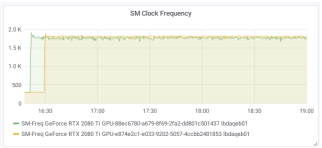
Integration test setup (2)

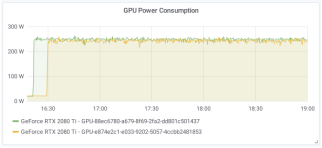
Notes:

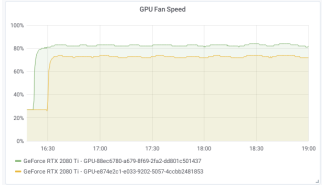
- The TELL40 can generate data into the server memory on each socket.
- Both network cards are connected back to back. A flow can then be simulated as if coming from the event building application.
- Each GPU can process data independently from each other. Two GPU applications are run, each one attached to a different GPU.

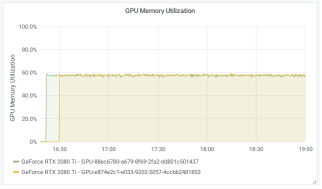
More integration test results (1)

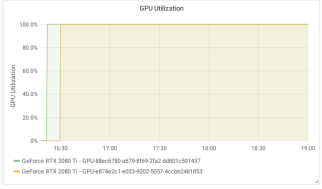












More integration test results (2)