CUDA to SYCL

porting traccc seeding algorithm

ACTS

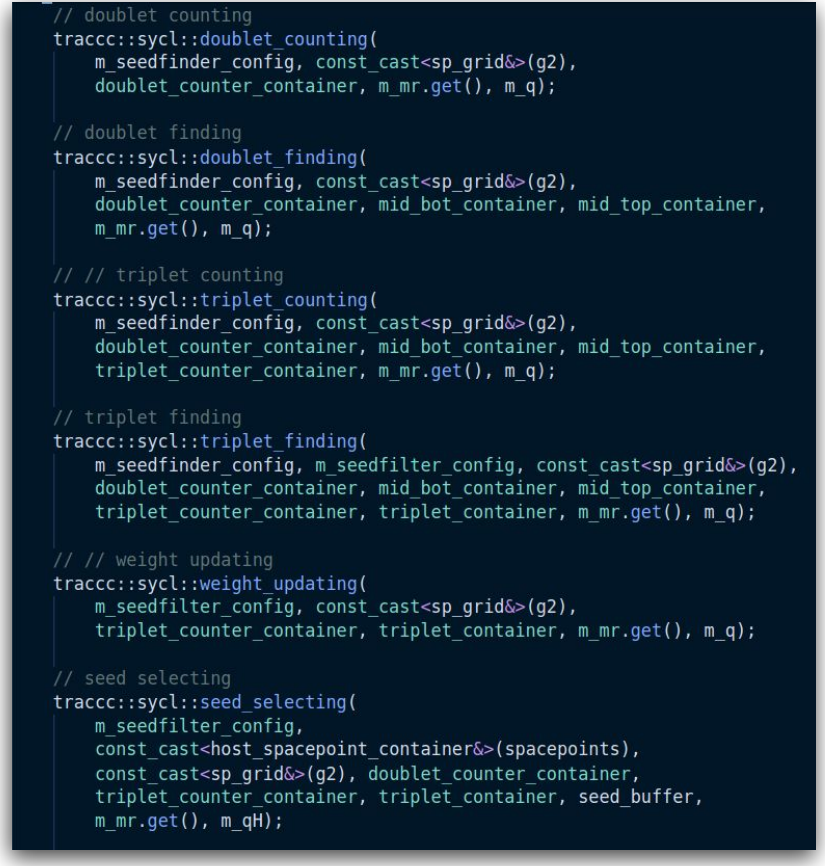
Konrad Kusiak
konrad.aleksander.kusiak@cern.ch
Queen Mary University of London

February 18, 2022

Motivation

- Portability: targets CPUs, GPUs from Nvidia, Intel, AMD and other hardware e.g. FPGAs
- Observed minimal overhead between native CUDA implementations and SYCL using cuda backend - advertised in different talks
- Most CUDA concepts map 1:1 with SYCL nd_range kernel execution model

Seeding algorithm structure



```
// doublet counting
traccc::sycl::doublet_counting(
    m_seedfinder_config, const_cast<sp_grid&>(g2),
    doublet_counter_container, m_mr.get(), m_q);

// doublet finding
traccc::sycl::doublet_finding(
    m_seedfinder_config, const_cast<sp_grid&>(g2),
    doublet_counter_container, mid_bot_container, mid_top_container,
    m_mr.get(), m_q);

// // triplet counting
traccc::sycl::triplet_counting(
    m_seedfinder_config, const_cast<sp_grid&>(g2),
    doublet_counter_container, mid_bot_container, mid_top_container,
    triplet_counter_container, m_mr.get(), m_q);

// triplet finding
traccc::sycl::triplet_finding(
    m_seedfinder_config, m_seedfilter_config, const_cast<sp_grid&>(g2),
    doublet_counter_container, mid_bot_container, mid_top_container,
    triplet_counter_container, triplet_container, m_mr.get(), m_q);

// // weight updating
traccc::sycl::weight_updating(
    m_seedfilter_config, const_cast<sp_grid&>(g2),
    triplet_counter_container, triplet_container, m_mr.get(), m_q);

// seed selecting
traccc::sycl::seed_selecting(
    m_seedfilter_config,
    const_cast<host_spacepoint_container&>(spacepoints),
    const_cast<sp_grid&>(g2), doublet_counter_container,
    triplet_counter_container, triplet_container, seed_buffer,
    m_mr.get(), m_qH);
```

Kernel steps:

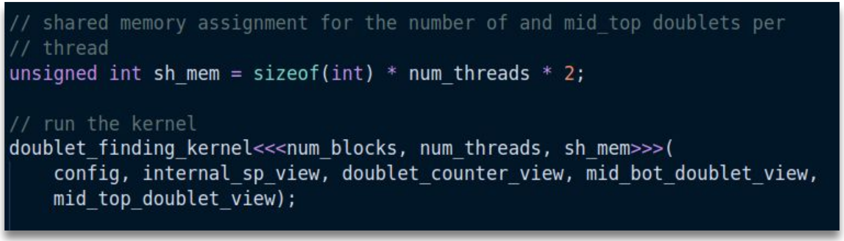
- ➢ Count compatible **bottom and top spacepoints** for every **middle spacepoint**
- ➢ Find the **compatible doublets** and add them in the container, in **sorted order**
- ➢ Count the compatible **triplets** for every **middle-bottom** doublet
- ➢ Find the **compatible triplets** and place them in the container, in **sorted order**
- ➢ For every triplet, **iterate over other triplets** with the same **middle-bottom doublets** to update its weight based on the **number of compatible triplets**
- ➢ **Select seeds** based on the experiment dependent cuts

Mapping to SYCL

Doublet Finding example

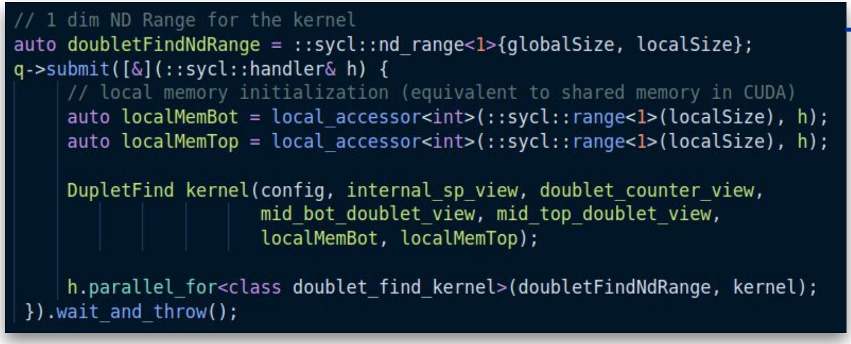
Invoking the kernel

CUDA

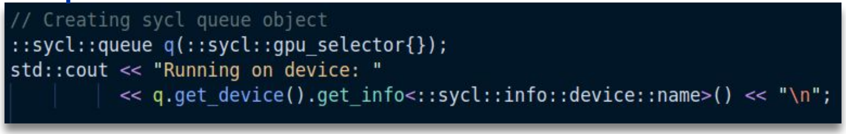


```
// shared memory assignment for the number of and mid_top doublets per
// thread
unsigned int sh_mem = sizeof(int) * num_threads * 2;

// run the kernel
doublet_finding_kernel<<<num_blocks, num_threads, sh_mem>>>(
    config, internal_sp_view, doublet_counter_view, mid_bot_doublet_view,
    mid_top_doublet_view);
```

SYCL



```
// 1 dim ND Range for the kernel
auto doubletFindNdRange = ::sycl::nd_range<1>{globalSize, localSize};
q->submit([&](::sycl::handler& h) {
    // local memory initialization (equivalent to shared memory in CUDA)
    auto localMemBot = local_accessor<int>(::sycl::range<1>(localSize), h);
    auto localMemTop = local_accessor<int>(::sycl::range<1>(localSize), h);

    DupletFind kernel(config, internal_sp_view, doublet_counter_view,
                      mid_bot_doublet_view, mid_top_doublet_view,
                      localMemBot, localMemTop);

    h.parallel_for<class doublet_find_kernel>(doubletFindNdRange, kernel);
}).wait_and_throw();
```

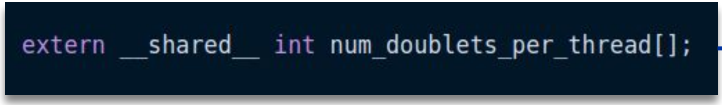


```
// Creating sycl queue object
::sycl::queue q(::sycl::gpu_selector{});
std::cout << "Running on device: "
          << q.get_device().get_info<::sycl::info::device::name>() << "\n";
```

Doublet Finding example

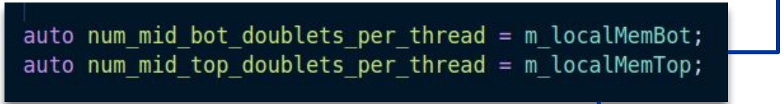
Shared Memory vs Local Memory

CUDA



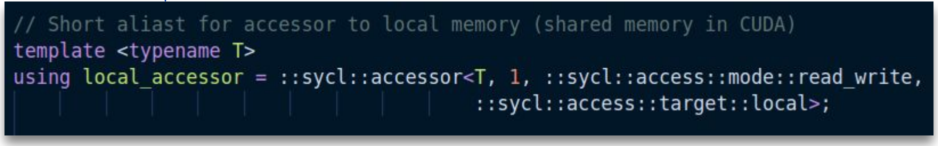
```
extern __shared__ int num_doublets_per_thread[];
```

SYCL



```
auto num_mid_bot_doublets_per_thread = m_localMemBot;
auto num_mid_top_doublets_per_thread = m_localMemTop;
```

Inside the doublet finding kernel

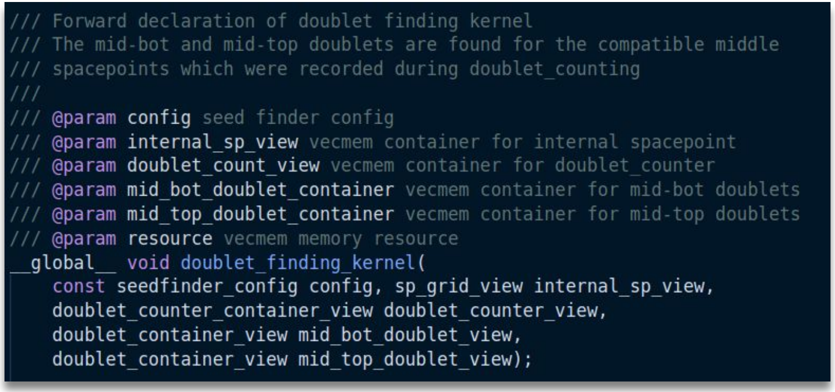


```
// Short aliast for accessor to local memory (shared memory in CUDA)
template <typename T>
using local_accessor = ::sycl::accessor<T, 1, ::sycl::access::mode::read_write,
                                         ::sycl::access::target::local>;
```

Doublet Finding example

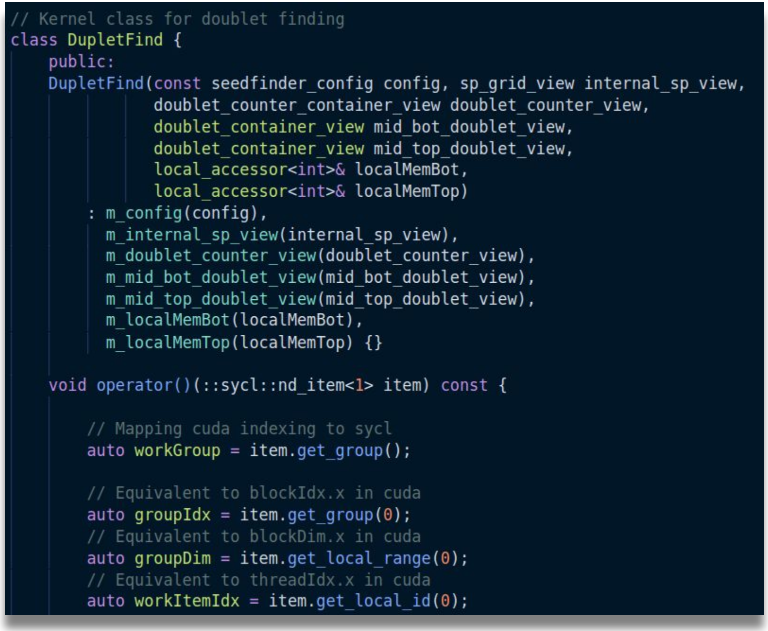
Kernel definition

CUDA



```
/// Forward declaration of doublet finding kernel
/// The mid-bot and mid-top doublets are found for the compatible middle
/// spacepoints which were recorded during doublet_counting
///
/// @param config seed finder config
/// @param internal_sp_view vecmem container for internal spacepoint
/// @param doublet_count_view vecmem container for doublet_counter
/// @param mid_bot_doublet_container vecmem container for mid-bot doublets
/// @param mid_top_doublet_container vecmem container for mid-top doublets
/// @param resource vecmem memory resource
__global__ void doublet_finding_kernel(
    const seedfinder_config config, sp_grid_view internal_sp_view,
    doublet_counter_container_view doublet_counter_view,
    doublet_container_view mid_bot_doublet_view,
    doublet_container_view mid_top_doublet_view);
```

SYCL

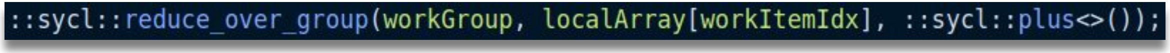


```
// Kernel class for doublet finding
class DupletFind {
    public:
    DupletFind(const seedfinder_config config, sp_grid_view internal_sp_view,
               doublet_counter_container_view doublet_counter_view,
               doublet_container_view mid_bot_doublet_view,
               doublet_container_view mid_top_doublet_view,
               local_accessor<int>& localMemBot,
               local_accessor<int>& localMemTop)
        : m_config(config),
          m_internal_sp_view(internal_sp_view),
          m_doublet_counter_view(doublet_counter_view),
          m_mid_bot_doublet_view(mid_bot_doublet_view),
          m_mid_top_doublet_view(mid_top_doublet_view),
          m_localMemBot(localMemBot),
          m_localMemTop(localMemTop) {}

    void operator()(::sycl::nd_item<1> item) const {

        // Mapping cuda indexing to sycl
        auto workGroup = item.get_group();

        // Equivalent to blockIdx.x in cuda
        auto groupIdx = item.get_group(0);
        // Equivalent to blockDim.x in cuda
        auto groupDim = item.get_local_range(0);
        // Equivalent to threadIdx.x in cuda
        auto workItemIdx = item.get_local_id(0);
```

Key changes

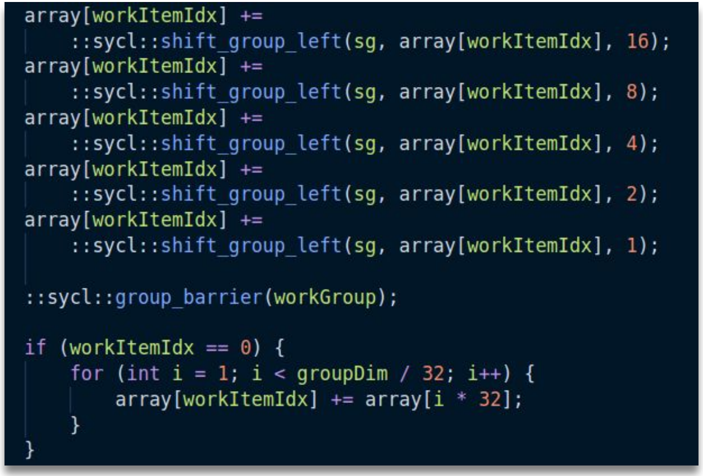
Work group reductions in the local memory (3 approaches):

1. SYCL 2020 reduce_over_group algorithm



```
::sycl::reduce_over_group(workGroup, localArray[workItemIdx], ::sycl::plus<>());
```

2. Using inter-thread operations i.e shift_group_left()

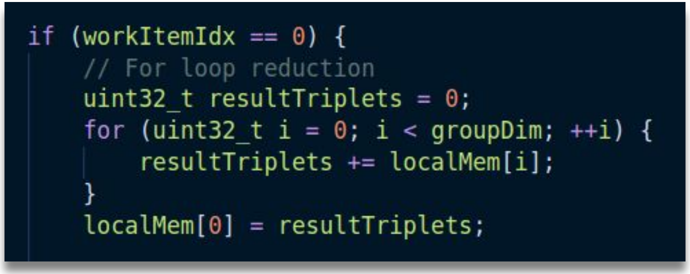


```
array[workItemIdx] +=
    ::sycl::shift_group_left(sg, array[workItemIdx], 16);
array[workItemIdx] +=
    ::sycl::shift_group_left(sg, array[workItemIdx], 8);
array[workItemIdx] +=
    ::sycl::shift_group_left(sg, array[workItemIdx], 4);
array[workItemIdx] +=
    ::sycl::shift_group_left(sg, array[workItemIdx], 2);
array[workItemIdx] +=
    ::sycl::shift_group_left(sg, array[workItemIdx], 1);

::sycl::group_barrier(workGroup);

if (workItemIdx == 0) {
    for (int i = 1; i < groupDim / 32; i++) {
        array[workItemIdx] += array[i * 32];
    }
}
```

And the one that actually works… →

Key changes

3. Simple for loop



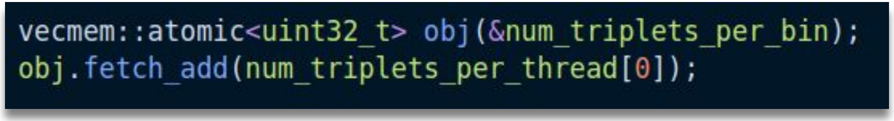
```
if (workItemIdx == 0) {
    // For loop reduction
    uint32_t resultTriplets = 0;
    for (uint32_t i = 0; i < groupDim; ++i) {
        resultTriplets += localMem[i];
    }
    localMem[0] = resultTriplets;
```

Doesn’t seem to decrease the performance in any significant way (on the Cuda backend)

Key changes

Atomic operations

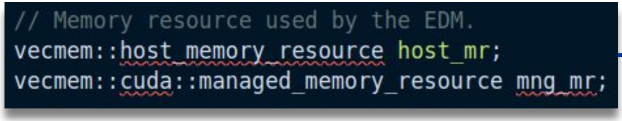
- Handled by vecmem::atomic object



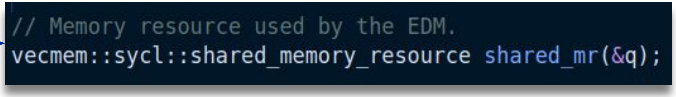
```
vecmem::atomic<uint32_t> obj(&num_triplets_per_bin);
obj.fetch_add(num_triplets_per_thread[0]);
```

Memory management

- Vecmem made it very easy to switch to SYCL…



```
// Memory resource used by the EDM.
vecmem::host_memory_resource host_mr;
vecmem::cuda::managed_memory_resource mng_mr;
```



```
// Memory resource used by the EDM.
vecmem::sycl::shared_memory_resource shared_mr(&q);
```

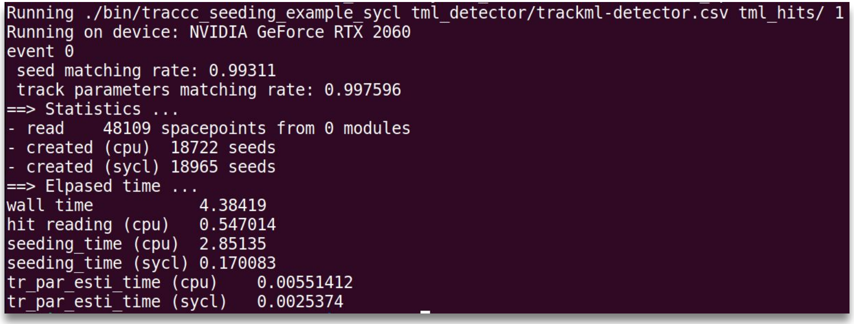
Results…

…and Issues

- The SYCL seeding algorithm successfully runs the tests on NVIDIA cards when compiled for CUDA backend with the Intel’s open source llvm compiler.

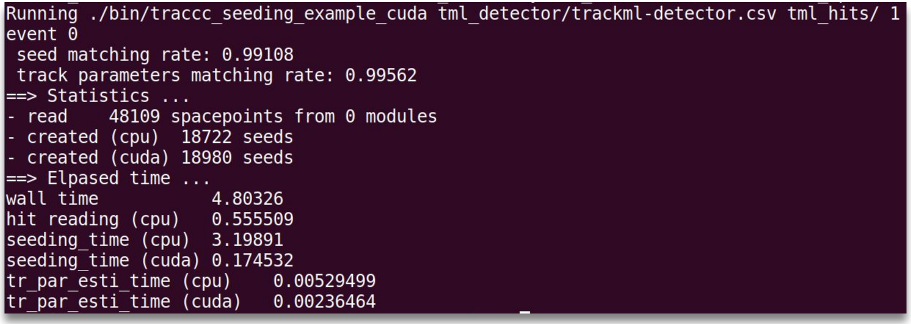
Running seeding_example test, on NVIDIA GeForce RTX 2060

SYCL



```
Running ./bin/traccc_seeding_example_sycl tml_detector/trackml-detector.csv tml_hits/ 1
Running on device: NVIDIA GeForce RTX 2060
event 0
 seed matching rate: 0.99311
 track parameters matching rate: 0.997596
==> Statistics ...
- read    48109 spacepoints from 0 modules
- created (cpu)  18722 seeds
- created (sycl) 18965 seeds
==> Elapsed time ...
wall time           4.38419
hit reading (cpu)   0.547014
seeding_time (cpu)  2.85135
seeding_time (sycl) 0.170083
tr_par_esti_time (cpu)    0.00551412
tr_par_esti_time (sycl)   0.0025374
```

CUDA

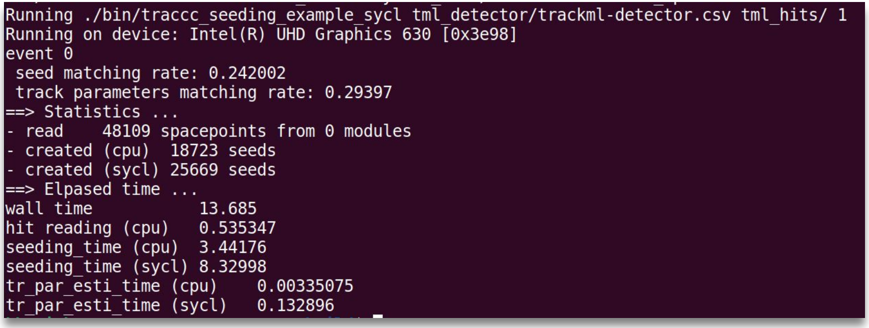


```
Running ./bin/traccc_seeding_example_cuda tml_detector/trackml-detector.csv tml_hits/ 1
event 0
 seed matching rate: 0.99108
 track parameters matching rate: 0.99562
==> Statistics ...
- read    48109 spacepoints from 0 modules
- created (cpu)  18722 seeds
- created (cuda) 18980 seeds
==> Elapsed time ...
wall time           4.80326
hit reading (cpu)   0.555509
seeding_time (cpu)  3.19891
seeding_time (cuda) 0.174532
tr_par_esti_time (cpu)    0.00529499
tr_par_esti_time (cuda)   0.00236464
```

and Issues…

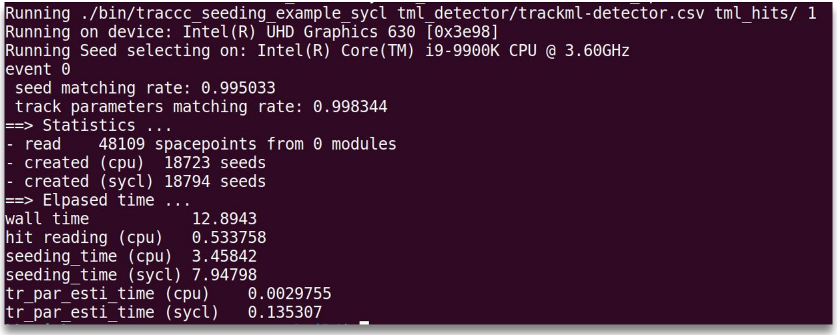
- The code doesn’t give 100% correct results when run on an Intel platform - we narrowed the Issue down to the **seed selecting** kernel

All kernels run on the Integrated GPU



```
Running ./bin/traccc_seeding_example_sycl tml_detector/trackml-detector.csv tml_hits/ 1
Running on device: Intel(R) UHD Graphics 630 [0x3e98]
event 0
 seed matching rate: 0.242002
 track parameters matching rate: 0.29397
==> Statistics ...
- read    48109 spacepoints from 0 modules
- created (cpu)  18723 seeds
- created (sycl) 25669 seeds
==> Elapsed time ...
wall time           13.685
hit reading (cpu)   0.535347
seeding_time (cpu)  3.44176
seeding_time (sycl) 8.32998
tr_par_esti_time (cpu)    0.00335075
tr_par_esti_time (sycl)   0.132896
```

Seed selecting being run on the CPU



```
Running ./bin/traccc_seeding_example_sycl tml_detector/trackml-detector.csv tml_hits/ 1
Running on device: Intel(R) UHD Graphics 630 [0x3e98]
Running Seed selecting on: Intel(R) Core(TM) i9-9900K CPU @ 3.60GHz
event 0
 seed matching rate: 0.995033
 track parameters matching rate: 0.998344
==> Statistics ...
- read    48109 spacepoints from 0 modules
- created (cpu)  18723 seeds
- created (sycl) 18794 seeds
==> Elapsed time ...
wall time           12.8943
hit reading (cpu)   0.533758
seeding_time (cpu)  3.45842
seeding_time (sycl) 7.94798
tr_par_esti_time (cpu)    0.0029755
tr_par_esti_time (sycl)   0.135307
```

Next steps

- Identify the problem with running the Seed selecting kernel on the Intel devices
- Make the code more portable - choosing the sizes of kernel grid based on the device’s capabilities
- Try running the code on an AMD gpu
- Search for bottlenecks in the code when run on other platforms