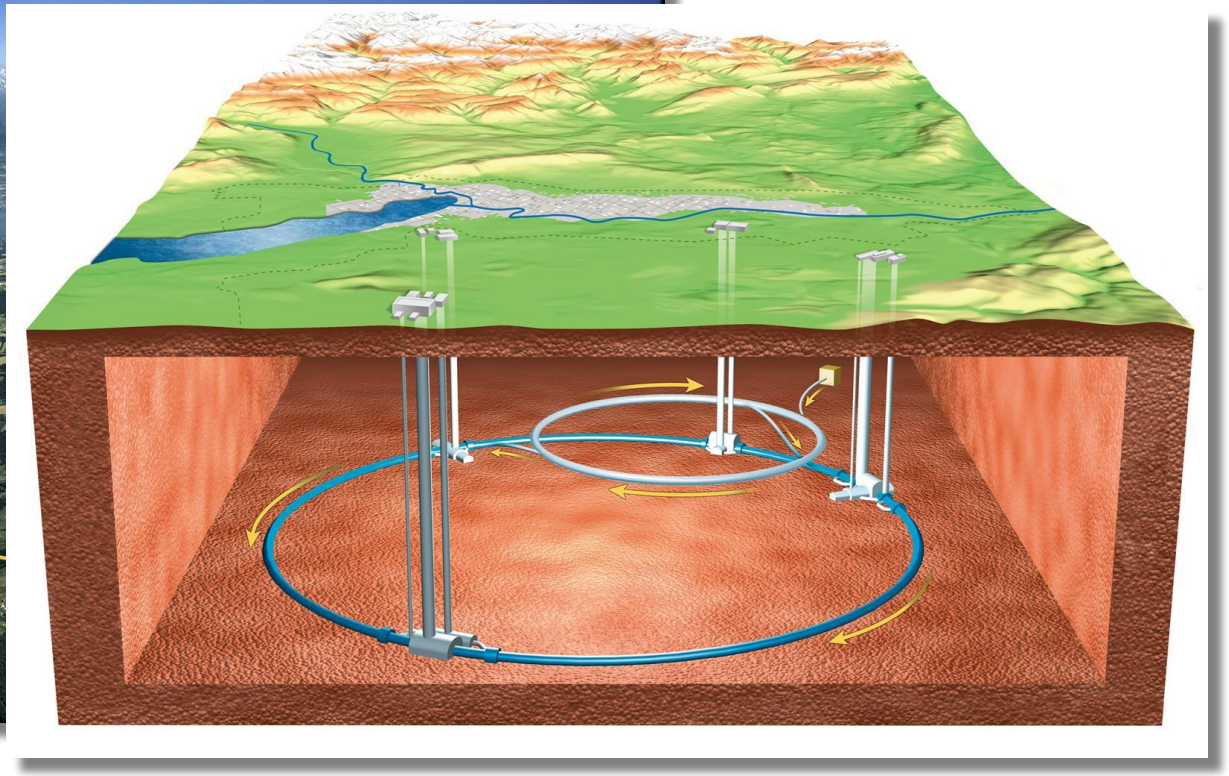
ATLAS Charged Particle Seed Finding with DPC++

Attila Krasznahorkay

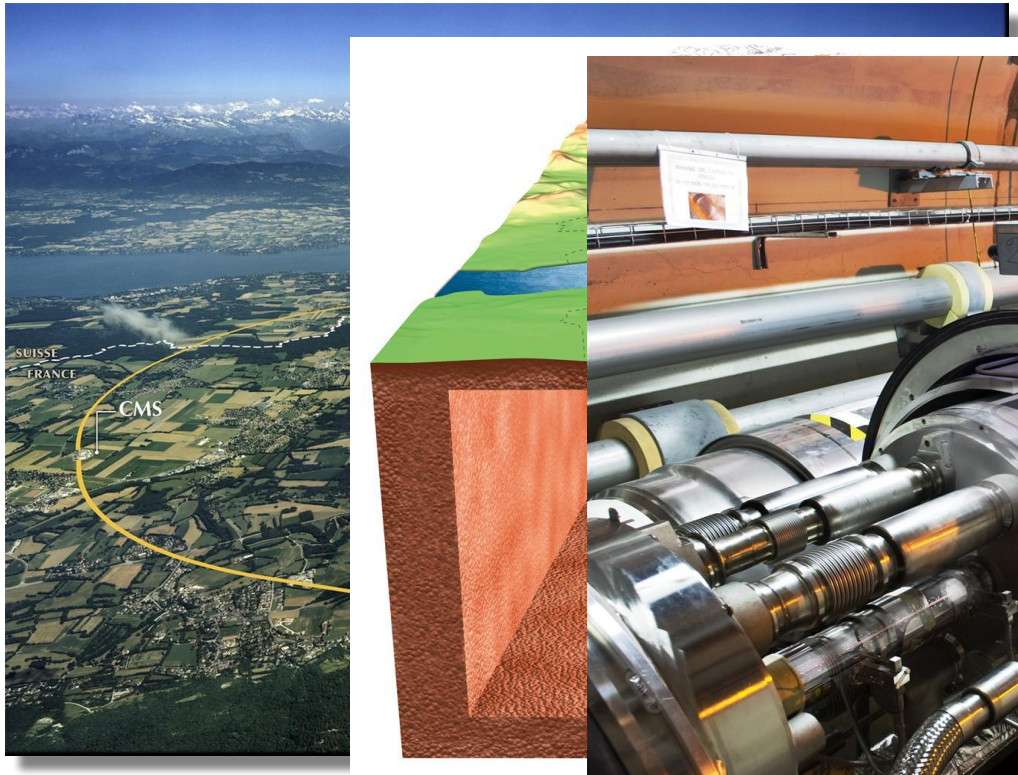
*with a **lot** of work from Angéla Czirkós*

The Large Hadron Collider

The Large Hadron Collider

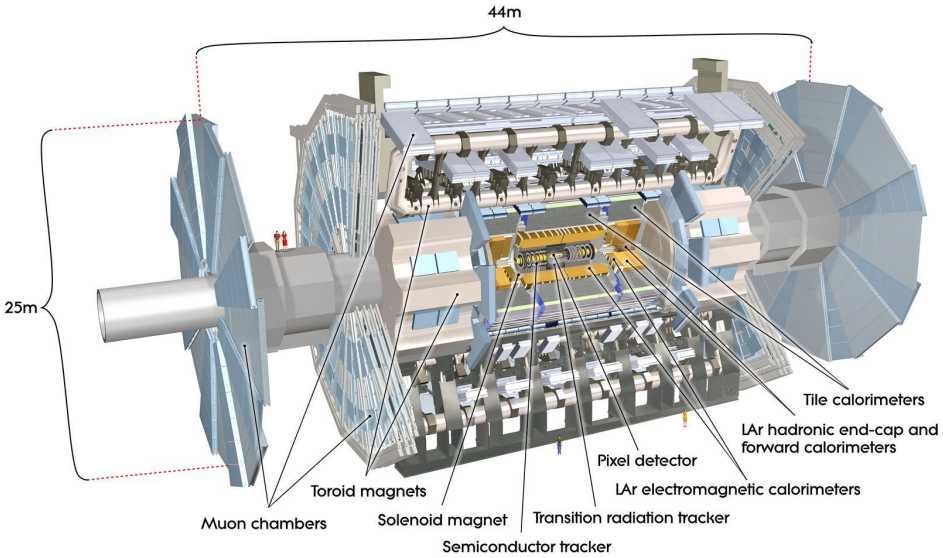


The Large Hadron Collider



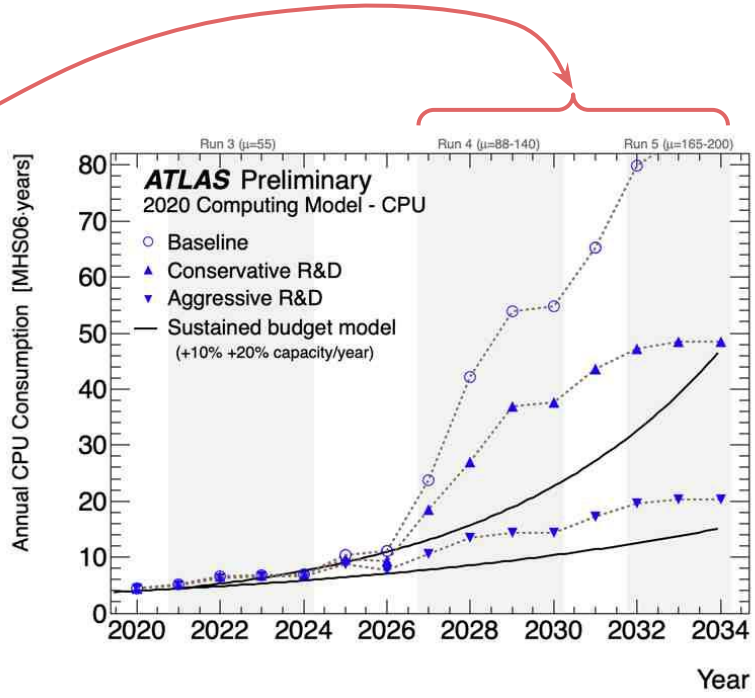
The ATLAS Experiment

- ATLAS is:
  - One of the general purpose experiments at the Large Hadron Collider
  - Collecting ~1.6 MB proton-proton (and sometimes Pb-Pb, Pb-p) data events with O(1) kHz rate, which our offline software has to process/analyse
  - Using hundreds of thousands of CPUs all over the world to process O(100) PB of data 24/7
  - Undergoing major hardware and software updates for LHC's Run-3/4

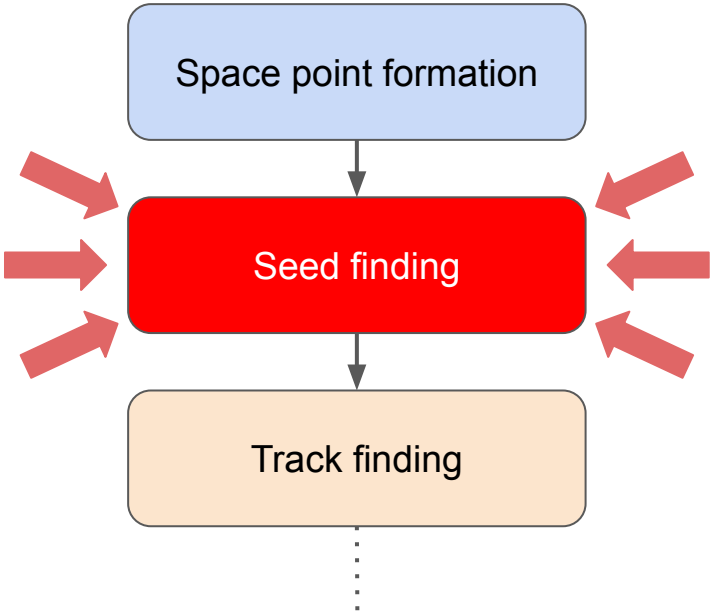


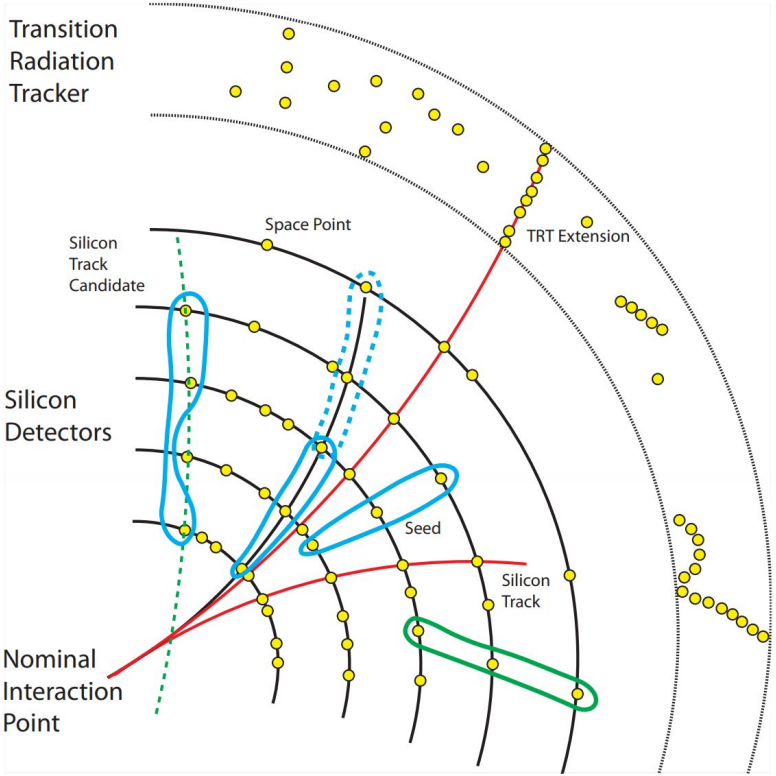
ATLAS's Computing Challenge

- The LHC will provide a much larger amount of data in what we call the "High Luminosity LHC" period
- Our current software will not be able to cope with that data with the budget that we will have
  - So we are looking into novel data processing techniques, including using accelerators
  - SYCL/oneAPI is one of the possibilities being looked at for integrating accelerated algorithms into ATLAS's codebase



Charged Particle Reconstruction





Acts Seed Finding

- Acts is an experiment agnostic track reconstruction toolset, based on ATLAS's track reconstruction code
  - https://acts.readthedocs.io
  - https://github.com/acts-project/acts
- During "seed finding" the code needs to find space point triplets that are compatible with coming from a helix track left by a charged particle
  - In general it's an N^3 problem. But with strict/clever selection techniques (in the existing C++ implementation), it behaves more like an N^2 problem.
- The algorithm is split into the following steps:
  - Spacepoint duplet finding, only keeping duplets passing certain selection criteria
  - Spacepoint triplet formation, only keeping triplets passing certain selection criteria
  - Spacepoint triplet filtering, keeping only triplets that are the "best" in varying groups of triplets

Acts Seed Finding

- Acts is an experiment agnostic track reconstruction toolset, based on ATLAS's track reconstruction code
  - https://acts.readthedocs.io
  - https://github.com/acts-project/acts
- During "seed finding" the code needs to find space point triplets that are compatible with coming from a helix track left by a charged particle
  - In general it's an N^3 problem. But with strict/clever selection techniques (in the existing C++ implementation), it behaves more like an N^2 problem.
- The algorithm is split into the following steps:
  - Spacepoint duplet finding, only keeping duplets passing certain selection criteria
  - Spacepoint triplet formation, only keeping triplets passing certain selection criteria

  "Easily" parallelisable
  - Spacepoint triplet filtering, keeping only triplets that are the "best" in varying groups of triplets

Acts Seed Finding

- Acts is an experiment agnostic track reconstruction toolset, based on ATLAS's track reconstruction code
  - https://acts.readthedocs.io
  - https://github.com/acts-project/acts
- During "seed finding" the code needs to find space point triplets that are compatible with coming from a helix track left by a charged particle
  - In general it's an N^3 problem. But with strict/clever selection techniques (in the existing C++ implementation), it behaves more like an N^2 problem.
- The algorithm is split into the following steps:
  - Spacepoint duplet finding, only keeping duplets passing certain selection criteria
  - Spacepoint triplet formation, only keeping triplets passing certain selection criteria

  "Easily" parallelisable
  - Spacepoint triplet filtering, keeping only triplets that are the "best" in varying groups of triplets

  Currently only partially parallelised

Acts::Sycl::Seedfinder (1)

```
namespace Acts::Sycl {

template <typename external_spacepoint_t>
class Seedfinder {
 public:
  Seedfinder(
      Acts::SeedfinderConfig<external_spacepoint_t> config,
      const Acts::Sycl::DeviceExperimentCuts& cuts,
      Acts::Sycl::QueueWrapper wrappedQueue = Acts::Sycl::QueueWrapper());

  ~Seedfinder() = default;
  Seedfinder() = delete;
  Seedfinder(const Seedfinder<external_spacepoint_t>&) = delete;
  Seedfinder<external_spacepoint_t>& operator=(
      const Seedfinder<external_spacepoint_t>&) = delete;

  /// Create all seeds from the space points in the three iterators.
  /// Can be used to parallelize the seed creation
  /// @param bottom group of space points to be used as innermost SP in a seed.
  /// @param middle group of space points to be used as middle SP in a seed.
  /// @param top group of space points to be used as outermost SP in a seed.
  /// Ranges must return pointers.
  /// Ranges must be separate objects for each parallel call.
  /// @return vector in which all found seeds for this group are stored.
  template <typename sp_range_t>
  std::vector<Seed<external_spacepoint_t>> createSeedsForGroup(
      sp_range_t bottomSPs, sp_range_t middleSPs, sp_range_t topSPs) const;

 private:
  Acts::SeedfinderConfig<external_spacepoint_t> m_config;

  /// Experiment specific cuts
  Acts::Sycl::DeviceExperimentCuts m_deviceCuts;

  /// Configuration object for the device side.
  Acts::Sycl::detail::DeviceSeedfinderConfig m_deviceConfig;

  /// Wrapper around a SYCL queue object.
  QueueWrapper m_wrappedQueue;
};

}  // namespace Acts::Sycl
```

- The primary user interface to the SYCL/oneAPI based R&D seedfinder implementation is templated
  - Necessary to make it possible to use the code with an experiment specific “event data model”
  - Provides some challenge in hiding the SYCL/oneAPI code from the interface

Acts::Sycl::Seedfinder (2)

- It helped development a lot to be able to implement kernels “inside the host code” in lambdas
  - However I found that long-term readability and “profile-ability” can suffer from writing the code in this way. I’m in the process of separating device code from host code a bit more formally in the implementation at the moment…
- What we have no solution for yet is to allow users to provide their own “filtering code” for the triplets that the main algorithm could use

```
//*********************************************//
// ********** DUPLET SEARCH - BEGIN ********** //
//*********************************************//

{
  sycl::buffer<uint32_t> countBotBuf(countBotDuplets.data(), M);
  sycl::buffer<uint32_t> countTopBuf(countTopDuplets.data(), M);
  q->submit([&](cl::sycl::handler& h) {
    auto countBotDupletsAcc =
        sycl::ONEAPI::atomic_accessor<uint32_t, 1,
                                      sycl::ONEAPI::memory_order::relaxed,
                                      sycl::ONEAPI::memory_scope::device>(
            countBotBuf, h);
    h.parallel_for<duplet_search_bottom_kernel>(
        bottomDupletNDRange, [=](cl::sycl::nd_item<2> item) {
          const auto mid = item.get_global_id(0);
          const auto bot = item.get_global_id(1);
          // We check whether this thread actually makes sense (within
          // bounds).
          // The number of threads is usually a factor of 2, or 3*2^k (k
          // \in N), etc. Without this check we may index out of arrays.
          if (mid < M && bot < B) {
            const auto midSP = deviceMiddleSPs[mid];
            const auto botSP = deviceBottomSPs[bot];

            const auto deltaR = midSP.r - botSP.r;
            const auto cotTheta = (midSP.z - botSP.z) / deltaR;
            const auto zOrigin = midSP.z - midSP.r * cotTheta;

            if (!(deltaR < seedfinderConfig.deltaRMin) &&
                !(deltaR > seedfinderConfig.deltaRMax) &&
                !(cl::sycl::abs(cotTheta) > seedfinderConfig.cotThetaMax) &&
                !(zOrigin < seedfinderConfig.collisionRegionMin) &&
                !(zOrigin > seedfinderConfig.collisionRegionMax)) {
              // We keep counting duplets with atomic access.
              const auto ind = countBotDupletsAcc[mid].fetch_add(1);
              deviceTmpIndBot[mid * B + ind] = bot;
            }
          }
        });
  });
```

ActsUnitTestSeedfinderSycl

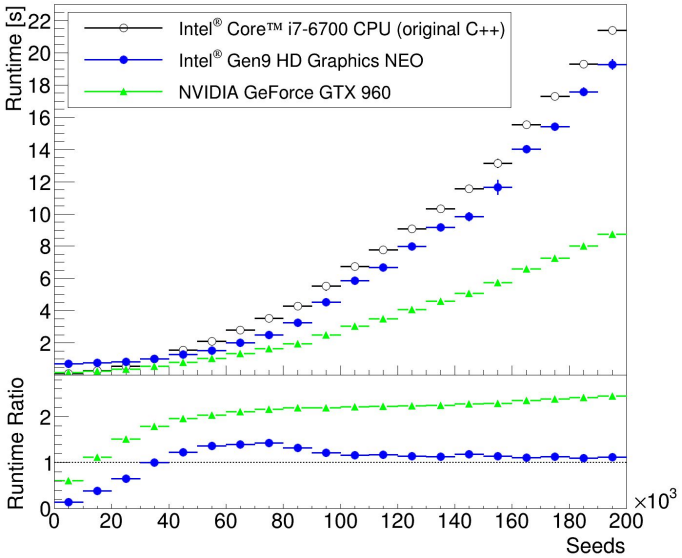
- The R&D code is exercised by a very simple executable for now
- Which we are using with inputs coming from ATLAS's detector simulation
  - Setting up a slew of inputs with differing event complexities, to test the algorithm across a reasonably wide phase space

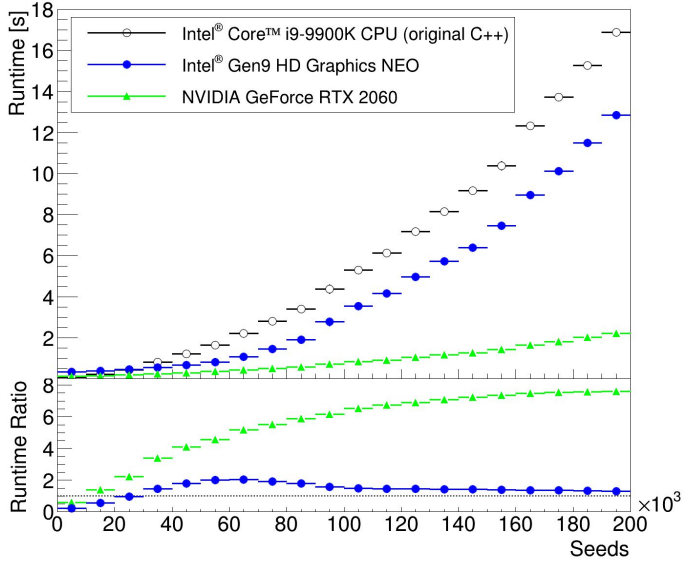
```
// ------------------------------------ //
// -------- EXECUTE ON GPU - SYCL ------- //
// ------------------------------------ //

auto start_sycl = std::chrono::system_clock::now();

group_count = 0;
std::vector<std::vector<Acts::Seed<SpacePoint>>> seedVector_sycl;

for (auto groupIt = spGroup.begin(); !(groupIt == spGroup.end()); ++groupIt) {
  seedVector_sycl.push_back(syclSeedfinder.createSeedsForGroup(
      groupIt.bottom(), groupIt.middle(), groupIt.top()));
  group_count++;
  if (!cmdlTool.allgroup && group_count >= cmdlTool.groups) {
    break;
  }
}
auto end_sycl = std::chrono::system_clock::now();
```

```
[bash][atspot01]:acts > ActsUnitTestSeedfinderSycl -f /atlas/acts_data/atlas_seeds/pu100/evt23.txt -d "GeForce" -m
10:17:12    Sycl::QueueW   INFO      Running on: GeForce RTX 2060
read 187172 SP from file /atlas/acts_data/atlas_seeds/pu100/evt23.txt
Preparation time: 0.423617
Analyzed 260 groups for CPU
Analyzed 260 groups for SYCL

------------------------ Time Metric -------------------------
            Device:        CPU       SYCL  Speedup/ Agreement
         Time (s):   2.449449   0.452237           5.416298
       Seeds found:      67527      67527          99.989632
--------------------------------------------------------------

[bash][atspot01]:acts >
```

Performance

- The code can be compiled for all the available backends of DPC++
  - Here I show results of running the code with the OpenCL and NVPTX backends, on two different hosts
- The “reference result” is coming from the CPU optimised code running in a single thread, not using SYCL/oneAPI at all



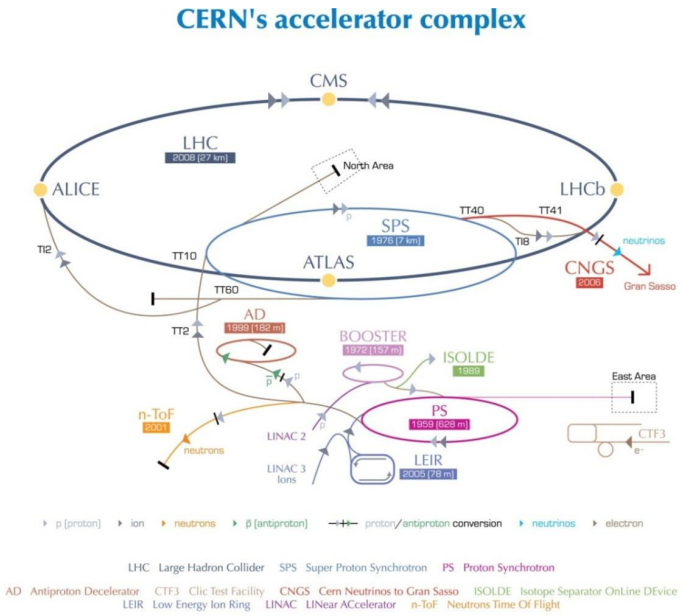


Summary

- We have a working prototype for “seed finding” in Acts, using SYCL/oneAPI
  - The tests show a healthy speedup from the code when using a dedicated GPU, but only a modest one when using an integrated one
- Other steps of the charged particle reconstruction in Acts are also under development to be ported to accelerators
- Ease of “code prototyping” is a big plus during algorithm development
  - However our initial idea of using the same algorithm on the host when no accelerator is available, is likely not going to work out. The CPU code just needs to be optimised differently than the GPU one. ☹
- ATLAS will likely need to put code like this into production in LHC’s Run 3/4 to be able to cope with the LHC’s increased performance
  - We will be continuing with our developments for years to come…

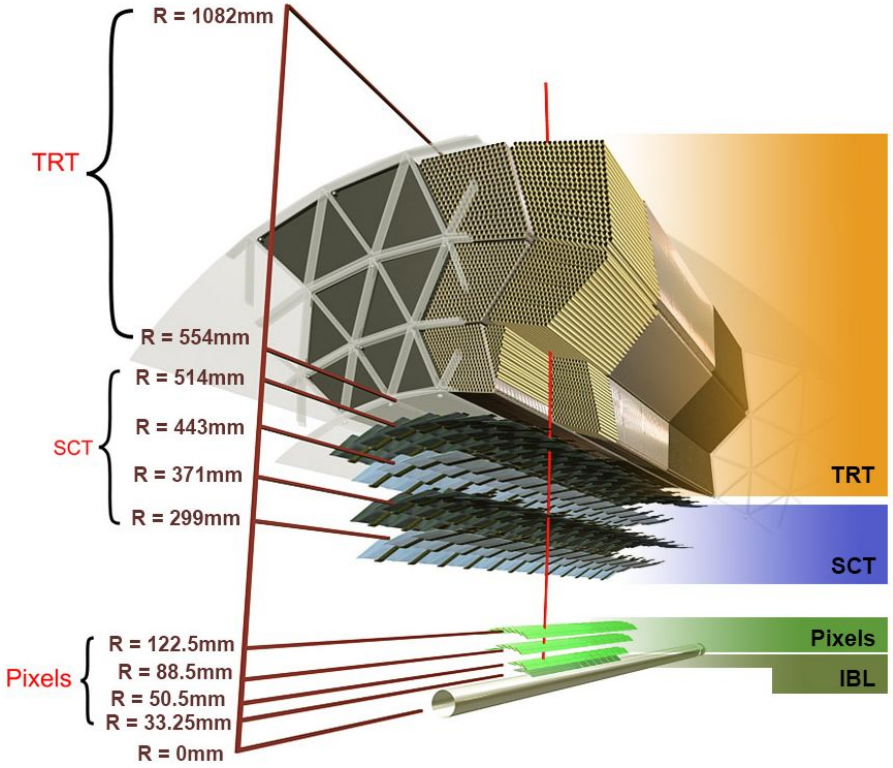
Backup

The Large Hadron Collider



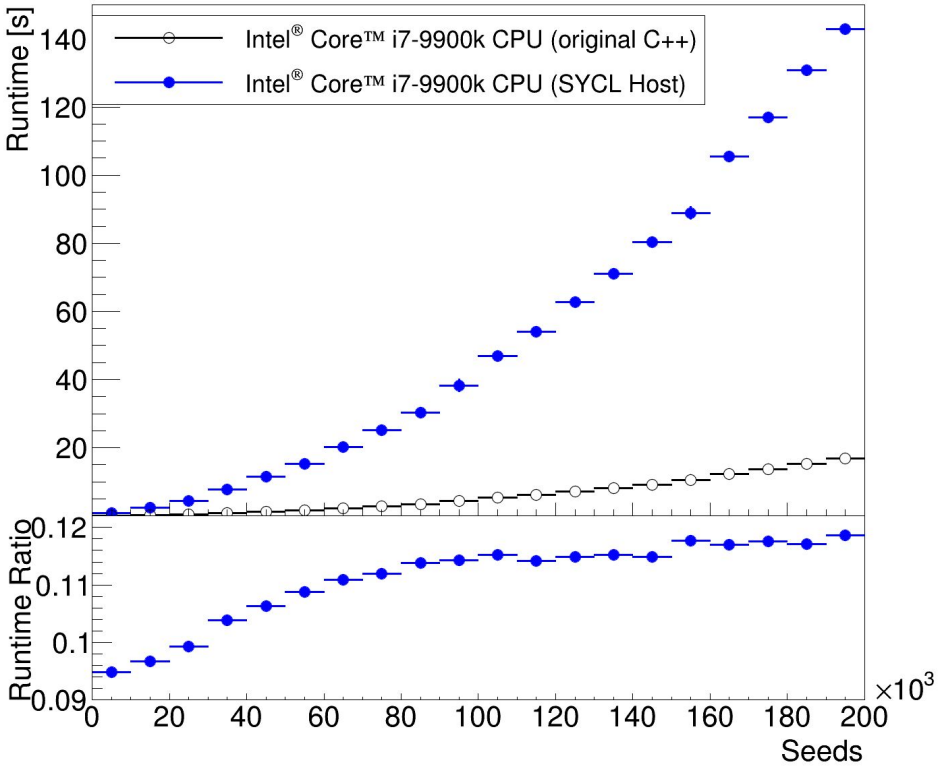
- Searching for new phenomena and making precision measurements at the energy frontier requires a **huge** machine
  - The LHC with all of its experiments is the largest and most complex physics experiment ever constructed
- Protons are accelerated in multiple steps to 6.5 TeV energy, and brought into collision at 4 points around the LHC ring
  - Producing 13 TeV p-p collisions for the 4 main LHC experiments

Charged Particle Detection



- Different detector types are placed in a solenoid magnetic field, which detect (in different ways) when a charged particle goes through them
- Charged particles travel along a helix in the solenoid magnetic field
  - Reconstructing the properties of this helix allow us to determine the physical properties of the particle

Single Threaded SYCL Performance



http://home.cern