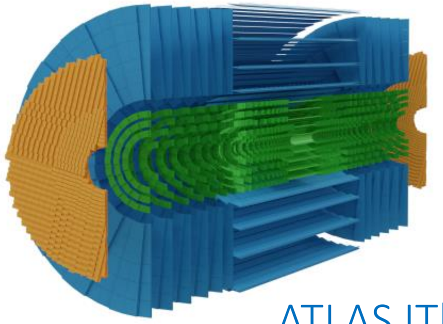
ACTS GPU Track Reconstruction Demonstrator for HEP

Paul Gessinger,[1] Hadrien Grasland,[2] Heather Gray,[3,4]
Sylvain Joube,[2] Konrad Kusiak,[1] Attila Krasznahorkay,[1] Charles Leggett,[4] Georgiana Mania,[5]
Joana Niermann,[1] Andreas Salzburger,[1] Nicholas Styles,[5] Stephen Swatman,[1] Beomki Yeo[3,4]

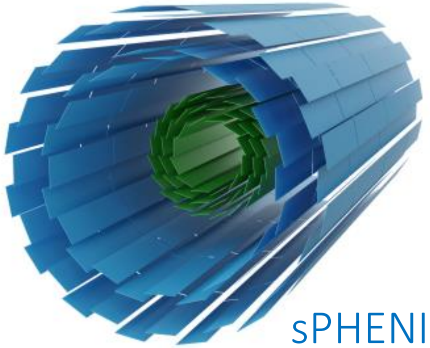
[1]CERN
[2]IJCLab
[3]University of California, Berkeley
[4]Lawrence Berkeley National Laboratory
[5]Deutsches Elektronen Synchrotron

A Common Tracking Software (ACTS)

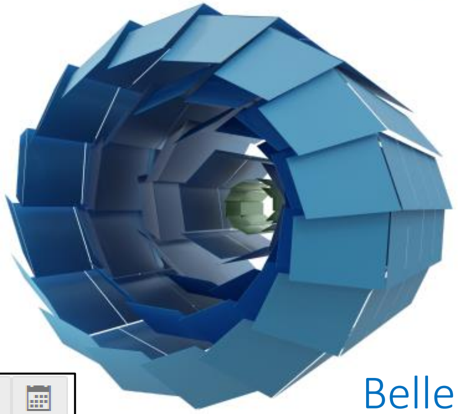
- A modern open-source HEP tracking toolkit
  - Based on C++17
  - Experiment-Independent Design
- An R&D platform to explore new techniques
  - GPU Parallelization
  - Machine learning
- Related Talks in CTD 2022



ATLAS ITk



sPHENIX



Belle II

Implementation of ACTS into LDMX track reconstruction

31 May 2022, 15:30
25m
PCTS Conference room, 4th fl

Speaker

Pierfrancesco Butti (SLAC National Acce

Vecpar - a portable parallelization library

1 Jun 2022, 09:00
25m
PCTS conference room (4th floor) (Jadwin Hall, Prin

Speaker

Georgiana Mania (Deutsches Elektron...

Exploration of different parameter optimization algorithms within the context of ACTS software framework

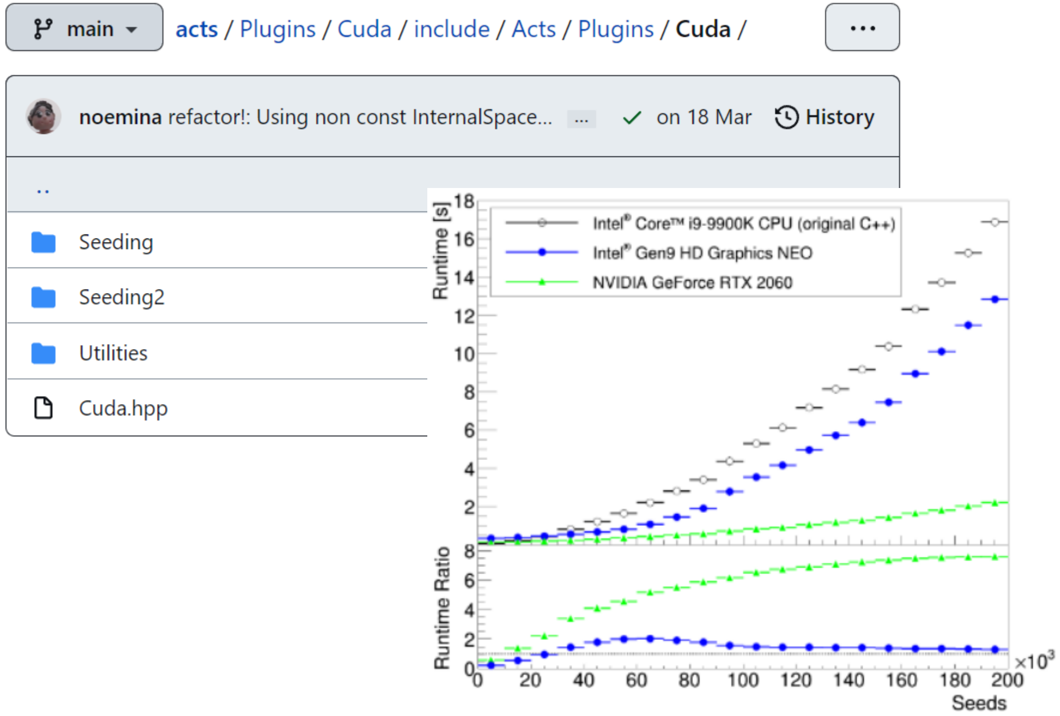
Plenary | YSF Plenary

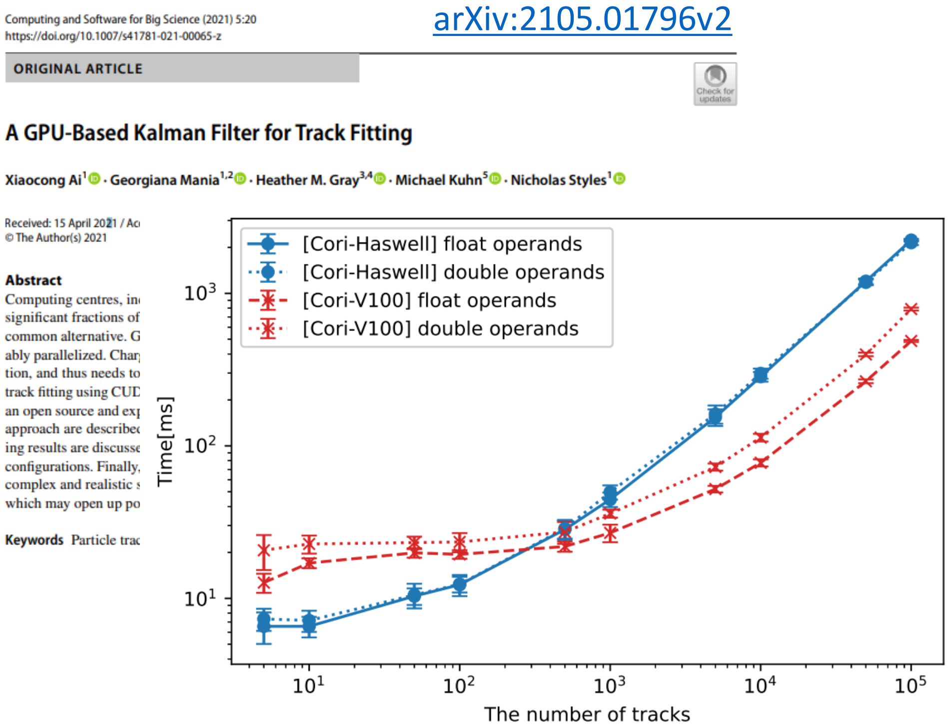
2 Jun 2022, 10:40
15m
PCTS conference room (4th floor) (Jadwin Hall, Princeton University)

Speaker

Rocky Bala Garg (Stanford University ...

Brief History on ACTS GPU Studies





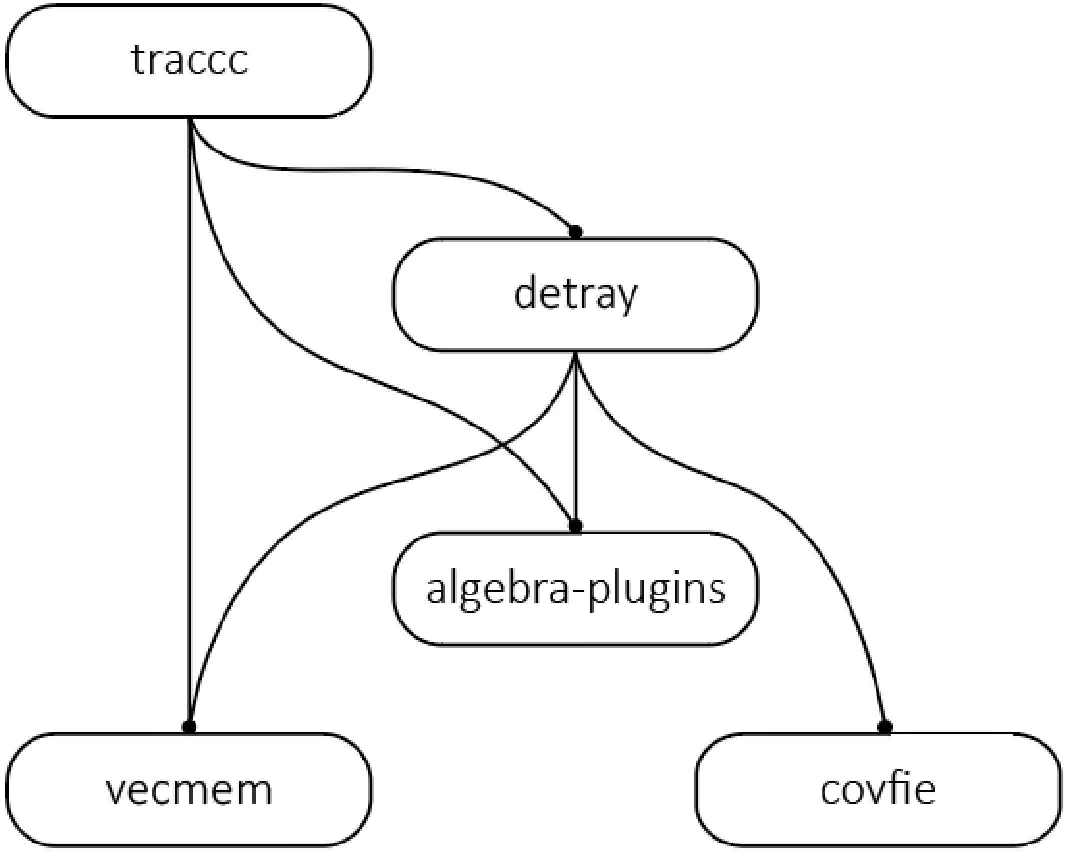
- In ACTS, there have been pilot studies on GPU seeding and Kalman filtering
- However, there is a clear limit when it comes to offloading the full tracking chain
  - Not all C++17 features are supported in GPU
  - Runtime polymorphism in tracking geometry is problematic
- As a result, we decided to launch GPU R&D projects to continue early studies and combine them into single piece

Requirements

- Same physics performance as the existing CPU algorithms
- Experiment-independent design
- Realistic detector setup
    - Tracking geometry and magnetic field
- Event Data Model (EDM) shared by CPU and GPU
- Support for single and double precision
- Primarily focusing on CPU, CUDA and SYCL implementations
    - CUDA has been a standard GPU API working with NVIDIA hardware
    - SYCL is a cross-platform heterogeneous computing API working with NVIDIA, AMD and Intel hardware

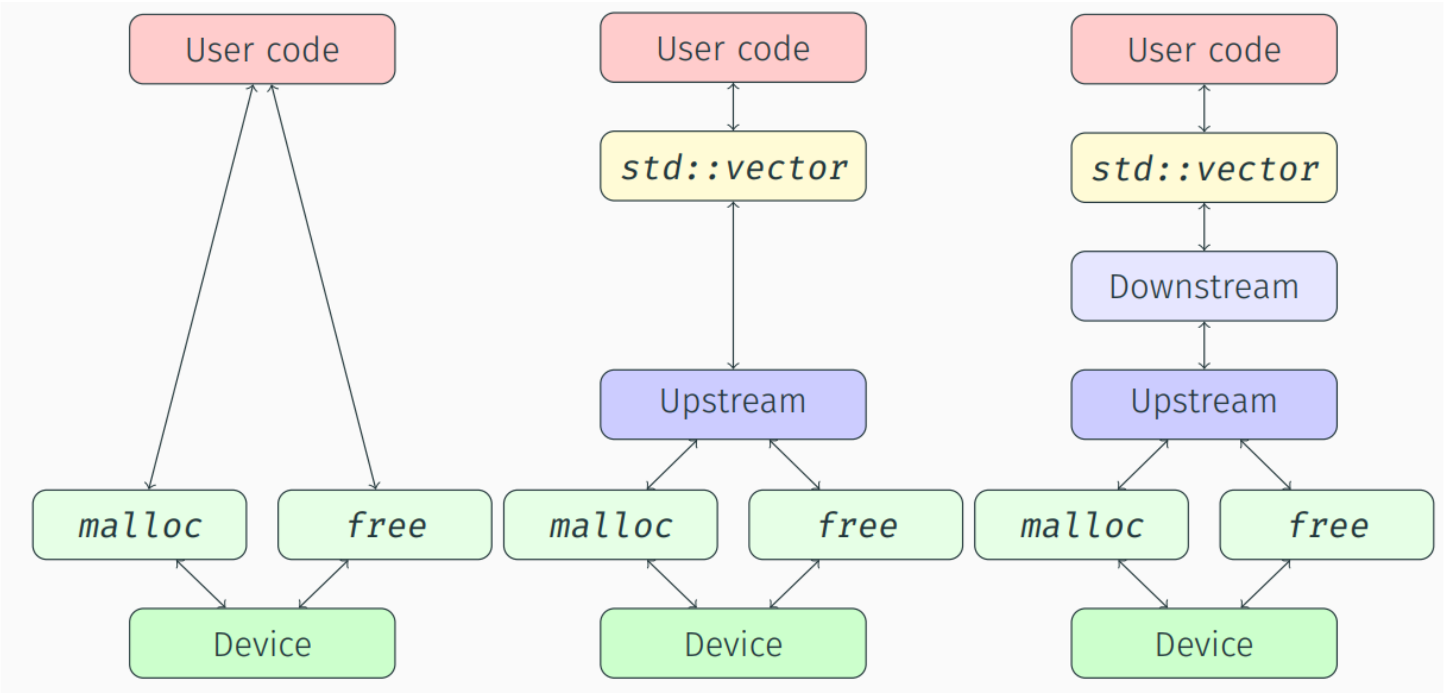
Ecosystem of ACTS GPU R&D

- **ACTS GPU R&D Projects**
  - traccc
    GPU track reconstruction demonstrator
  - detray
    Tracking geometry description without run-time polymorphism
  - covfie
    Compile-time vector field processor (for B field)
  - algebra-plugins
    vector and matrix algebra for multiple plugins
  - vecmem
    GPU memory management tool



GPU Memory Management (vecmem)

- Make use of std::pmr::memory_resource (upstream memory resource) to customize the memory allocation scheme of std::vector
  - CPU, CUDA, SYCL, and HIP
- Caching (downstream memory resource) is supported to reuse the memory allocated in the previous events
- Used in traccc and detray
  - EDM container
  - Container for detector components



S. Swatman, ACAT (2021)

Vector and Matrix Algebras (algebra-plugins)

- Algebra-plugins provides vector and matrix algebras required for track reconstruction
- Users can configure the following at compile-time:
  - Single or double precision
  - Which backends to use:
    - cmath (home-brew)
    - Eigen
    - SMatrix from ROOT

Backend	CPU	CUDA	SYCL
cmath	✅	✅	✅
Eigen	✅	✅	🟡
SMatrix	✅	⚪	⚪

✅ Natively supported
🟡 Natively supported, but not tested
⚪ No support

```
// Define matrix operator with cmath backend and single precision
using matrix_operator = cmath::matrix::actor<float>;

// Column-wise cross product between matrix (m) and vector (v)
__host__ __device__
inline matrix_type<3, 3> cross(const matrix_type<3, 3>& m,
                               const vector3& v) const {
    matrix_type<3, 3> ret;

    auto m_col0 = matrix_operator().template block<3, 1>(m, 0, 0);
    auto m_col1 = matrix_operator().template block<3, 1>(m, 0, 1);
    auto m_col2 = matrix_operator().template block<3, 1>(m, 0, 2);

    matrix_operator().set_block(ret, vector::cross(m_col0, v), 0, 0);
    matrix_operator().set_block(ret, vector::cross(m_col1, v), 0, 1);
    matrix_operator().set_block(ret, vector::cross(m_col2, v), 0, 2);

    return ret;
}
```

An example usage of matrix algebra

Track Reconstruction Chain in traccc

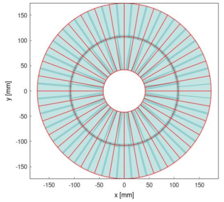
Connected Component Labeling algorithm

Extend triplets to full tracks

Data input → Hit Clusterization → Seeding → Combinatorial Kalman fitting → Kalman fitting

Digitized hit data from ACTS simulation

Search for triplets



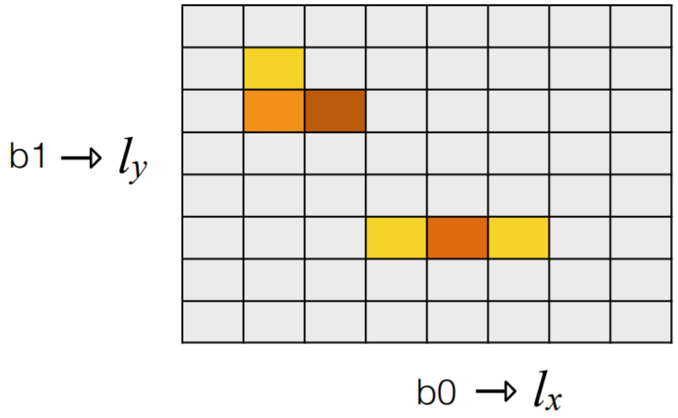
Detray Geometry

Refine the CKF results

Hit Clusterization

- Connected Component Labeling (CCL)
  - SparseCCL algorithm
- Measurement creation
  - Calculate the weighted average of cluster cell positions and covariances
- Spacepoint formation
  - local to global transformation
  - input to seeding algorithm

Connected Component Labeling (CCL)



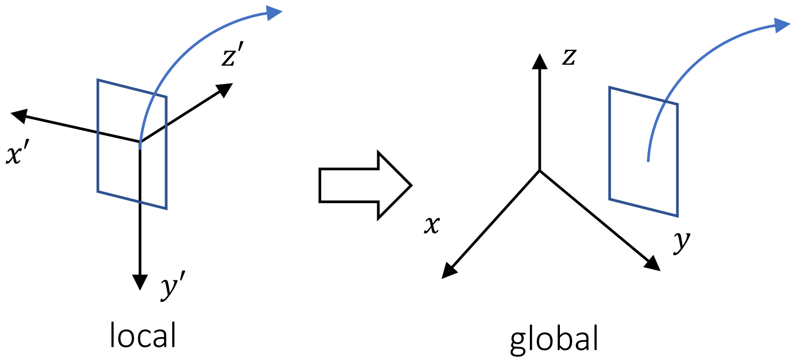
Measurement creation

$$l = \frac{1}{\sum_{(i,j)} w_{(i,j)}} \sum_{(i,j)} w_{(i,j)} l_{(i,j)}$$

S

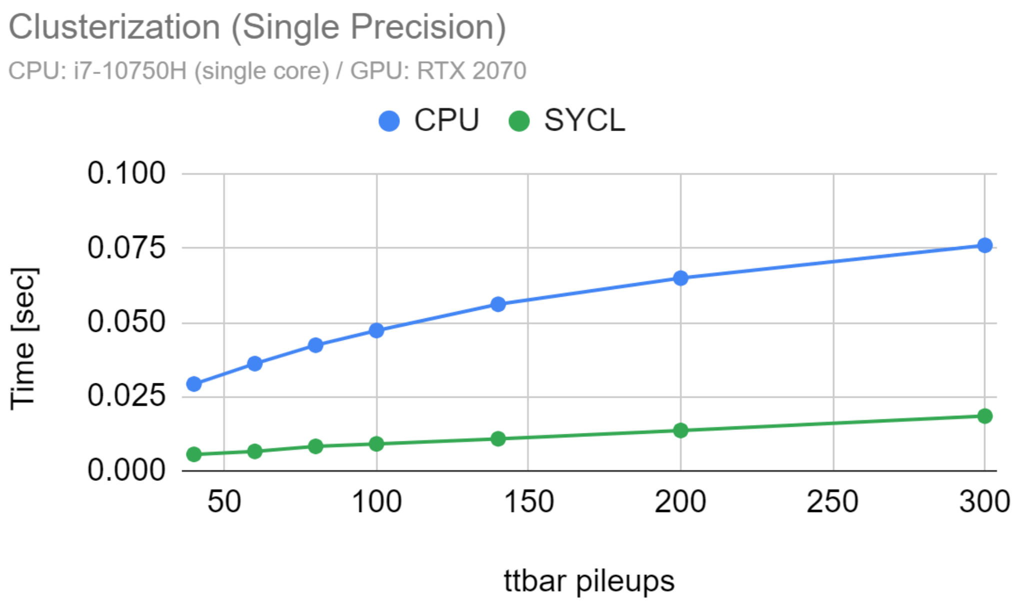
Calibration input:
weight calibration, Lorentz shift
resulting covariances (parametrised)

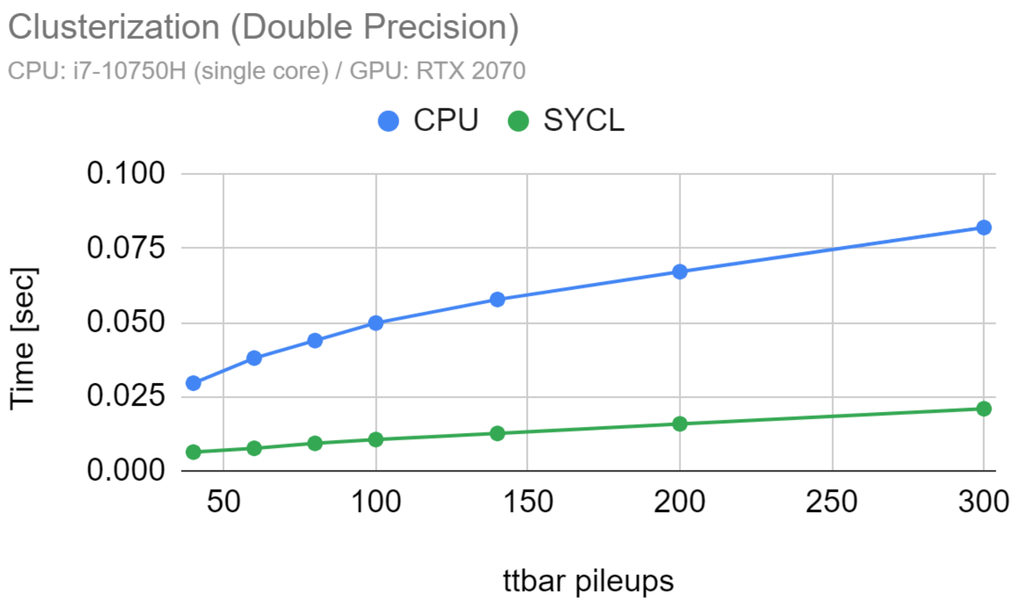
Spacepoint formation



GPU Implementation and Performance

- The GPU algorithm is divided into the following steps:
  - The CCL indices (to which cluster cells belong) are recorded in the vector
  - The number of clusters is counted from the CCL indices to initialize the vector of clusters with the proper size
    - Counting is required because the GPU does not allow the dynamic allocation in the kernels
  - The vector of clusters is filled with cells by looking up the CCL indices again
  - The measurement creation (Averaged local position and variance) and spacepoint formation (local to global transformation) are straightforward thanks to one-to-one correspondence

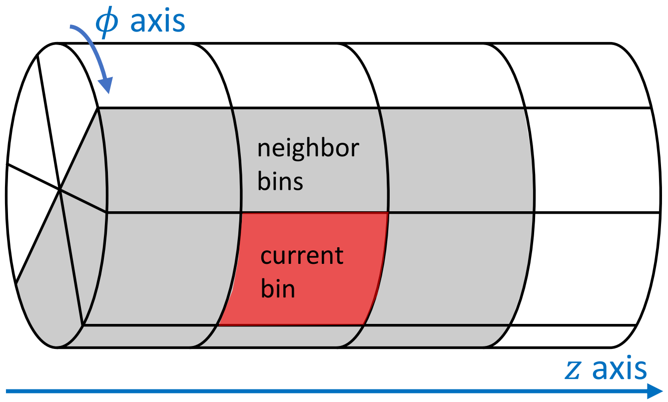




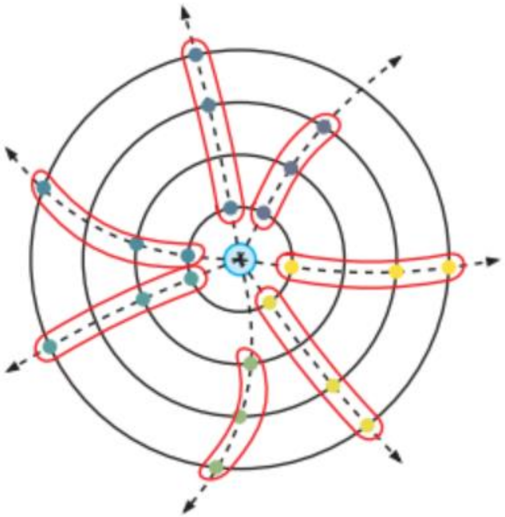
Seeding Algorithm

- Spacepoints are grouped based on their azimuthal angle and longitudinal position
- Doublets are obtained by iterating the spacepoints in the neighborhood bins
- Triplets are the combination of two doublets which satisfy the physical criteria (impact parameter, curvature, etc.)
- Track parameter estimation
  - global to local transformation
  - input to track fitting

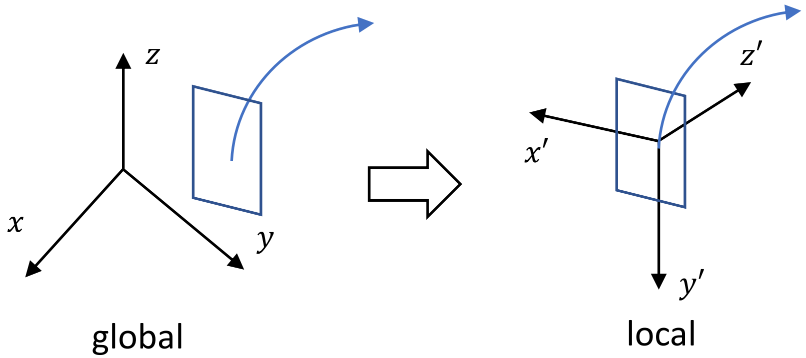
Spacepoint binning



Triplet finding



Track parameter estimation



GPU Implementation

- As for the clusterization algorithm, the size of doublet and triplet containers should be known before filling them

- The sub-algorithms of triplet finding (except the last filtering process) is divided into counting and finding
  - Counting: Counts the number of objects to be filled and initialize the vector containers with the proper size
  - Finding: Populates the objects into the vector containers

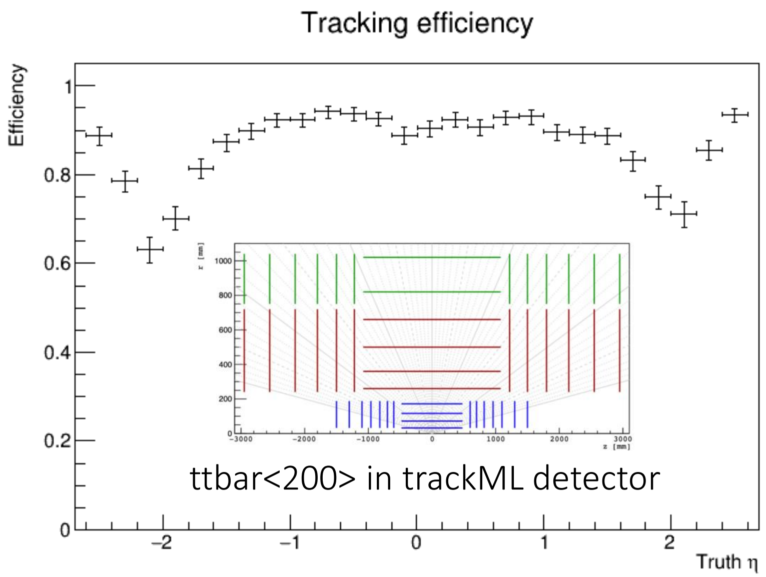
- Track parameter estimation (global to local) is as straightforward as the spacepoint formation (local to global)

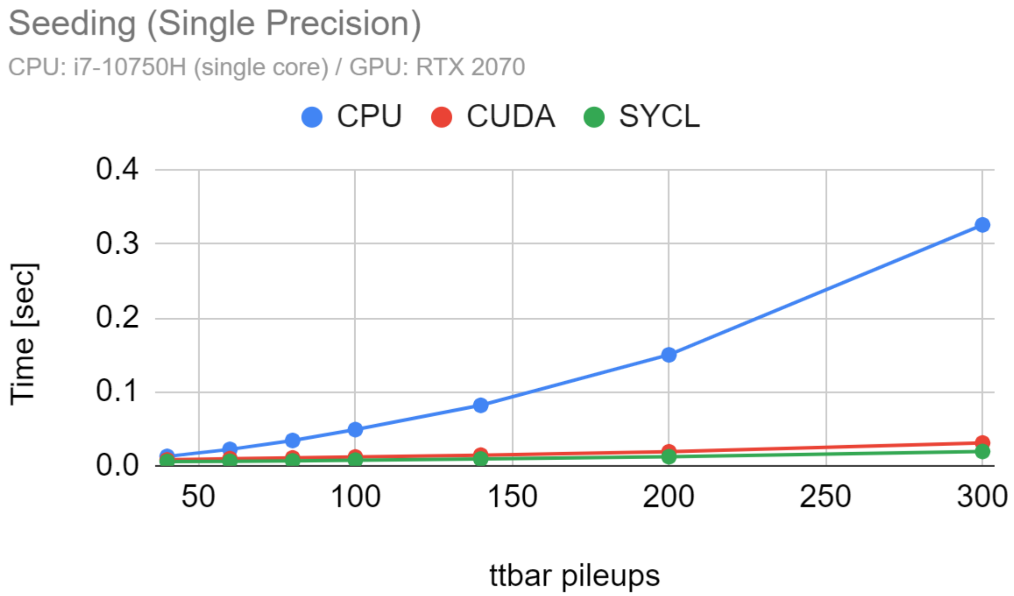
Seeding kernel executions per event

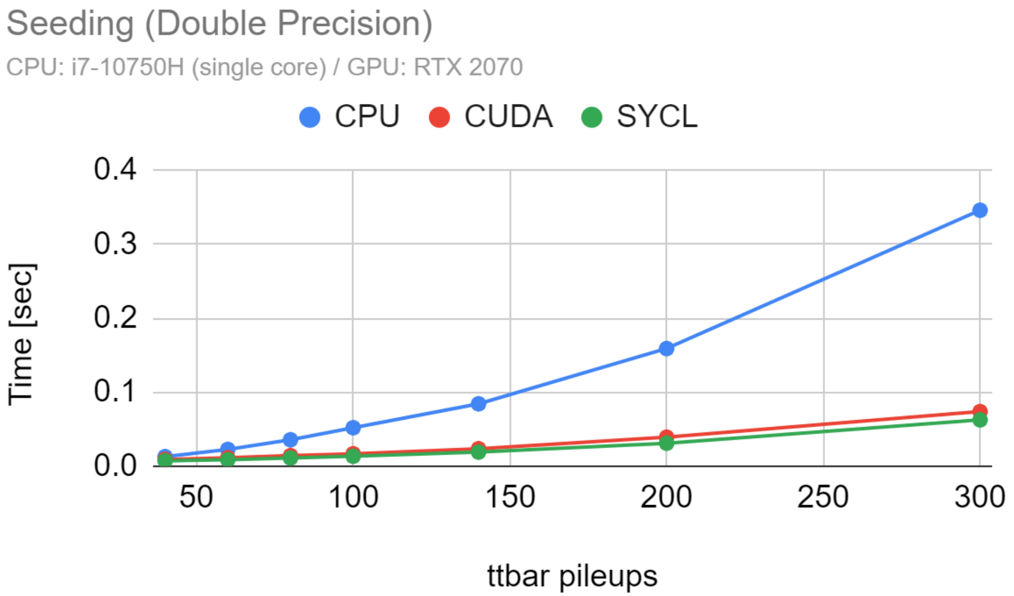
Seeding Performance

- For ttbar <200> pileup events in trackML detector, one order of magnitude of speedup (CUDA vs. single CPU core) improvement is achieved with the single precision

- SYCL is slightly faster than CUDA because the spacepoint EDM is already located in GPU device from clusterization algorithm







Tracking Geometry without Run-time Polymorphism (detray)

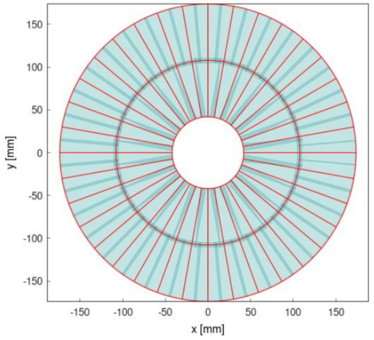
- Run-time polymorphism with pointers, which is widely used for detector building, is not very GPU-friendly
- In detray, run-time polymorphism is removed, and detector configurations are determined at compile-time.

Compile-time

- Detector-specific metedata to configure detector components
- initialize tuple containers for components

Run-time

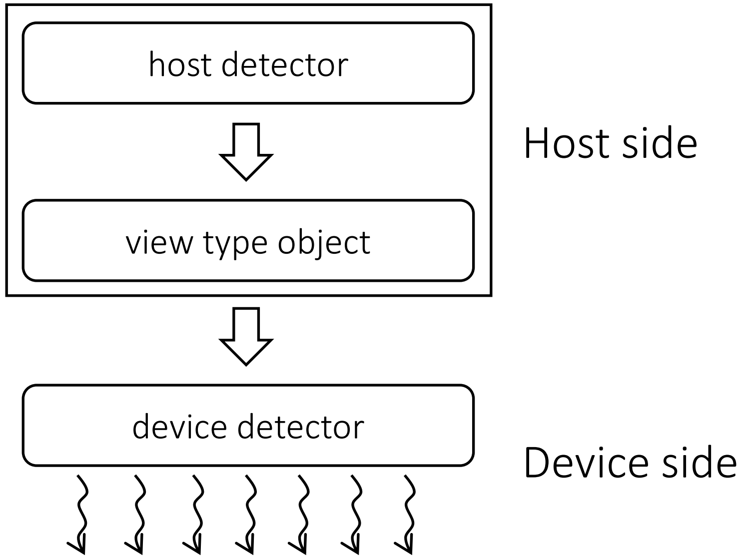
- Fill the components inside tuples



Detector GPU Offloading

- The host/device trait of the detector depends on the vecmem container type
  - host detector with vecmem::vector
  - device detector with vecmem::device_vector

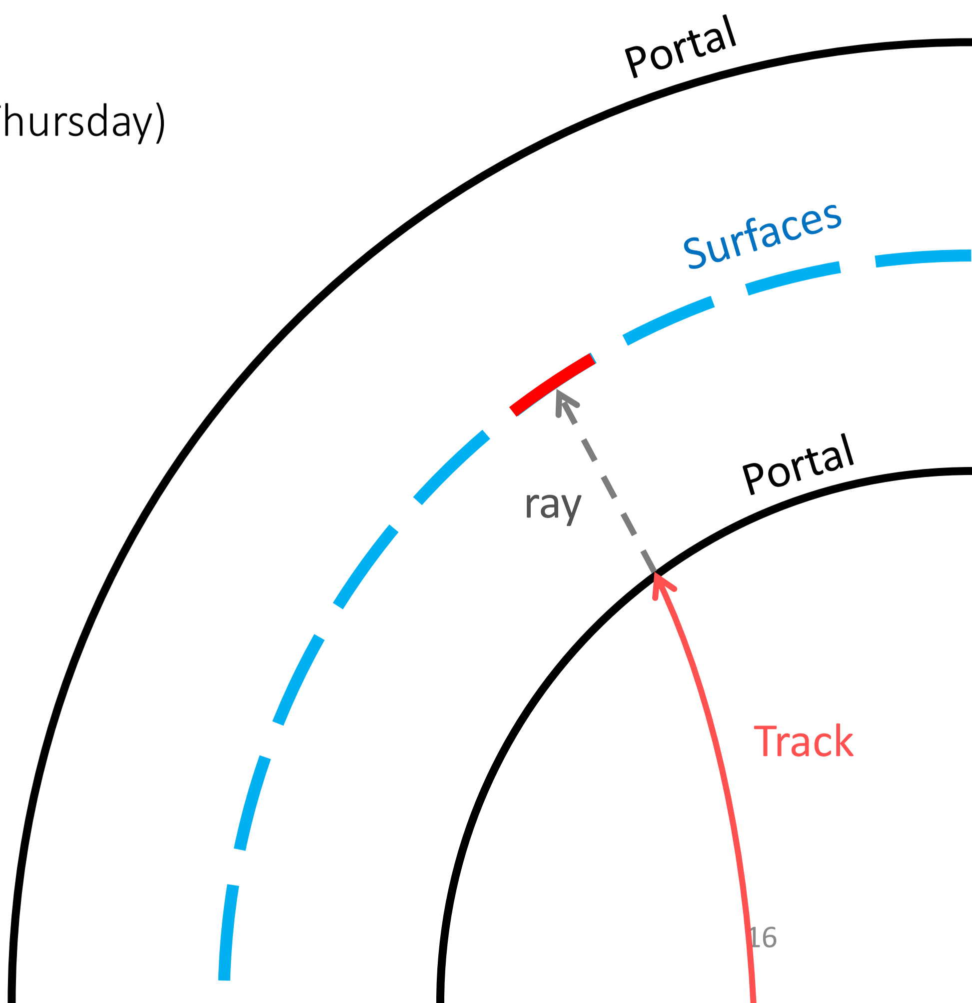
- The host detector is passed to the device code via view type object




```
// cuda kernel function declaration
__global__ void test_kernel(detector_data data);

int main(){

  // cuda unified shared memory resource
  vecmem::cuda::managed_memory_resource resource;

  // host detector with a metadata and vecmem::vector
  detector<metadata, vecmem::vector> host_detector(resource);

  // ... Fill detector components in runtime ...

  // detector view type object
  detector_data data(host_detector);

  // run cuda kernel
  test_kernel<<<1, 1>>>(data);
}

// cuda kernel function implementation
__global__ void test_kernel(detector_data data){

  // device detector with a metadata and vecmem::device_vector
  detector<metadata, vecmem::device_vector> device_detector(data);

  // ... do something with parallelization
}
```

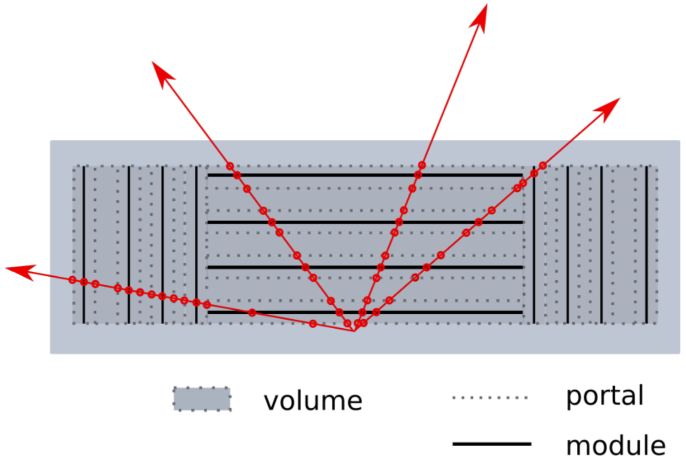
Propagation Tools

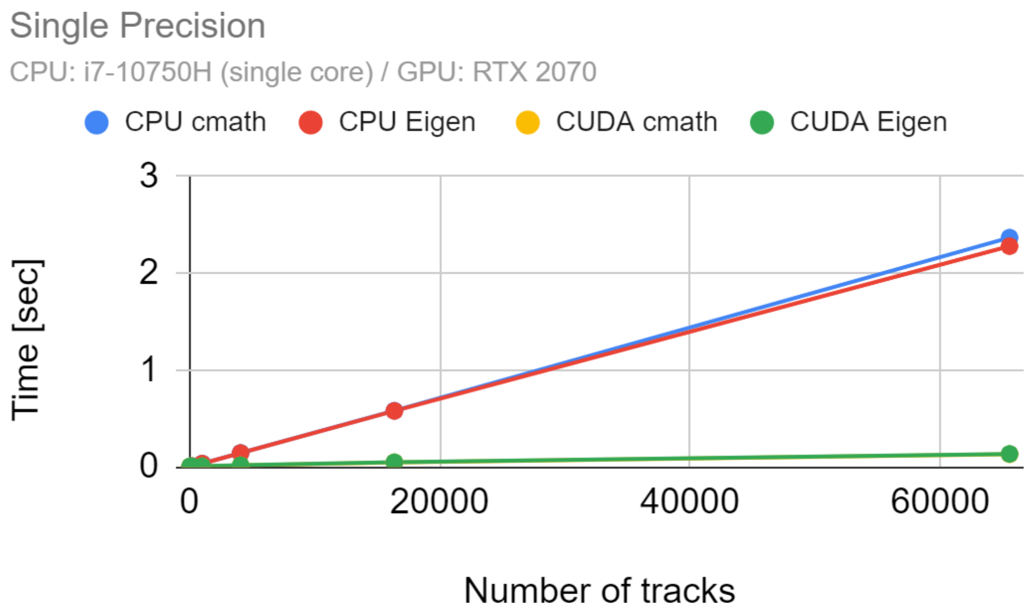
- Stepper
  - Advances the track state through geometry
  - Adaptive Runge-Kutta-Nyström method (See [G. Mania's talk] on Thursday)
- Navigator
  - Provides the next candidate surface and its distance
  - Candidate surface search is based on ray-tracing
- Propagator
  - Steers the workflow between stepper and navigator

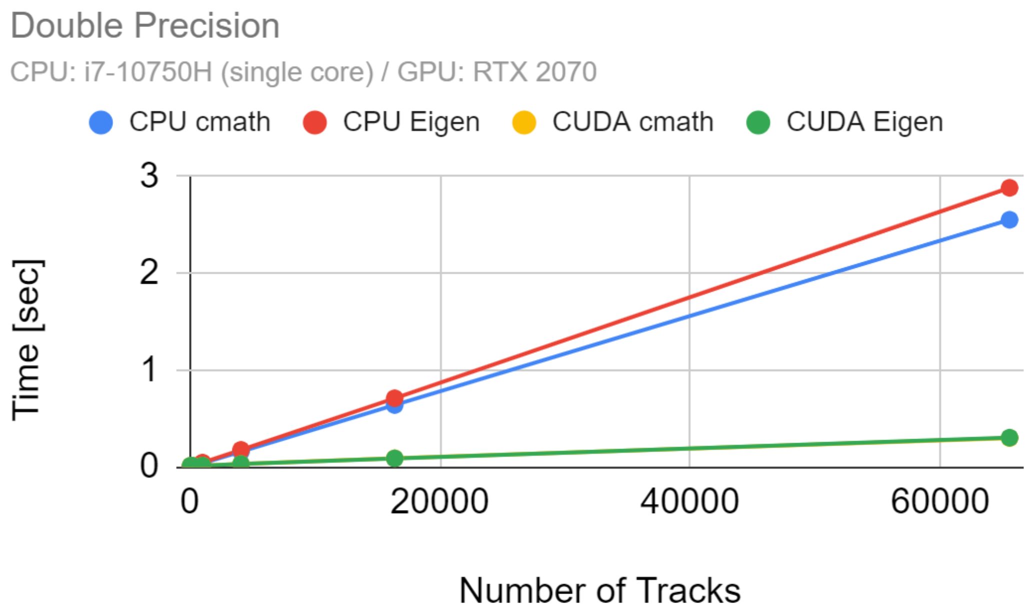


Propagation Speed Benchmark

- CUDA propagation was benchmarked with the pixel part of trackML detector
  - Runge-Kutta-Nyström stepper with constant 2 T
  - One order of magnitude of speedup with $\mathrm{O}(10^4)$ tracks







Magnetic Field Interpolation (covfie)

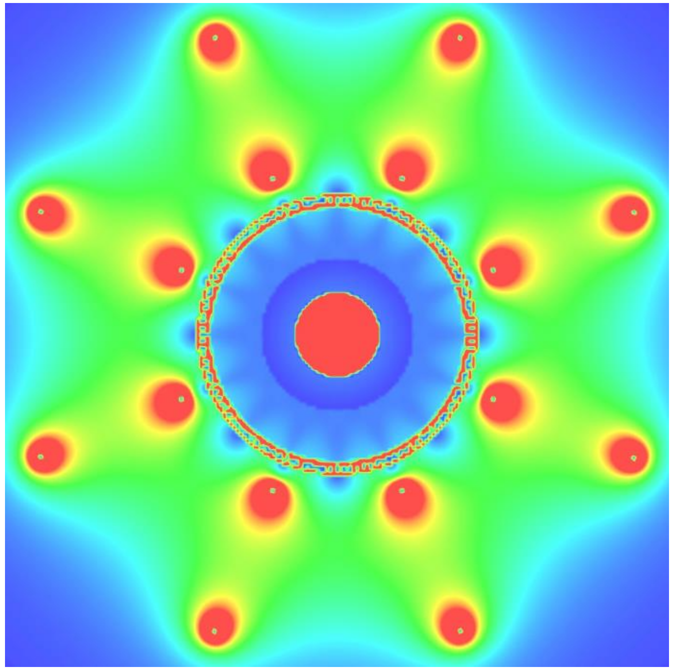
- **Co**-processor **v**ector **fie**ld library (covfie) is designed not only for magnetic field interpolation but for anything that uses a vector field

- Everything is known at compile-time: GPU APIs, dimension of vectors, interpolation algorithm, etc.

```
using cuda_field_t = covfie::field<covfie::backend::transformer::
    affine<
    covfie::backend::transformer::interpolator::linear<
        covfie::backend::cuda_array<3, covfie::backend::datatype::
    float3>>>>;

cuda_field_t cuda_field(cpu_field);
```

- Benchmark result is quite promising

	8192 X 8192 lookup time [ms]
CPU (Intel i5-7300U)	191719.2
GPU (GTX 1660 Ti)	90.4
GPU w/ texture memory	17.1



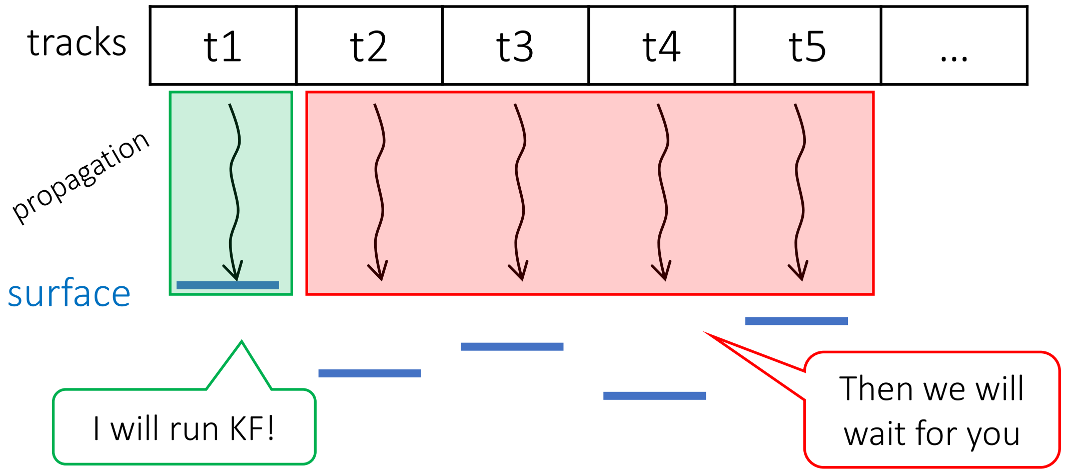
ATLAS magnetic field rendered at z = 0mm

Prospect for Kalman Filtering

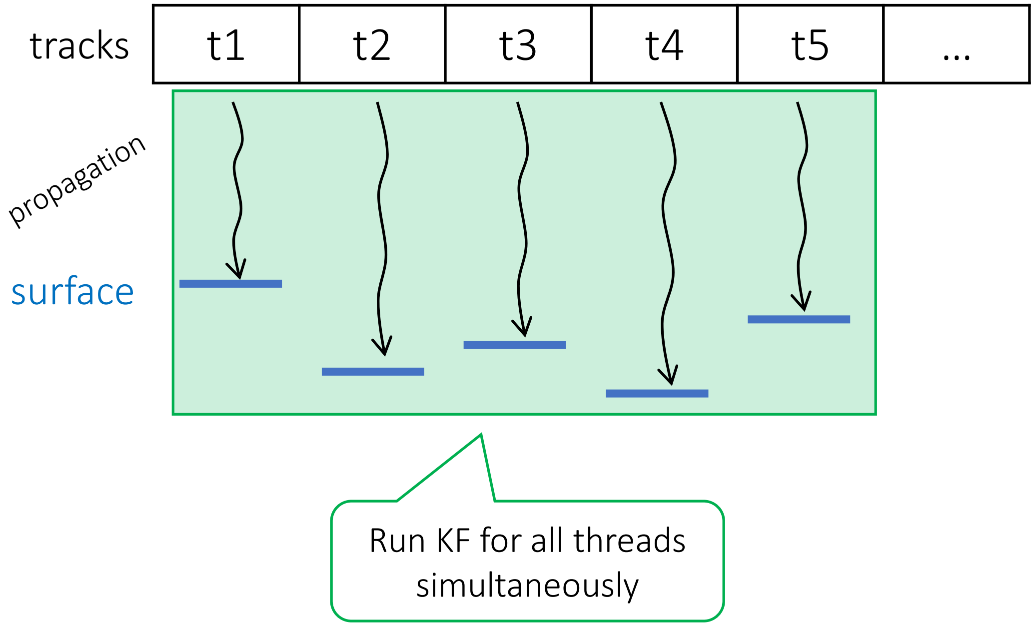
- Branch divergence would be quite problematic in Kalman fitting implementation
  - In SIMT architecture, the threads always execute the same instruction
  - This means that when a track on a thread comes across the surface and runs Kalman Fitting (or any other operations), all other threads will get idle

- Mitigating the branching divergence with a clever thread synchronization will be an interesting task

❑ Naïve KF implementation



❑ Thread synchronization to run KF simultaneously



Summary

- Acts R&D projects are being developed to offload tracking algorithms to GPUs
- **vecmem** is the core library for defining detector geometry and event data model
- **algebra-plugins** provides vector and matrix algebras to detray and traccc
- **detray** constructs the tracking geometry for (combinatorial) Kalman filtering
  - Benchmark study on propagation in trackML detector is promising
  - **covfie** library will be used to interpolate inhomogeneous magnetic field in GPU devices
- **traccc** is the downstream project for GPU tracking demonstration
  - Demonstrated clusterization and seeding algorithm on CUDA and SYCL
  - Lots of works needs to be done to get the event throughput
    - Kalman filtering implementation using the detray tracking geometry
    - Utilization of **vecmem** downstream resource to reuse previously allocated memory
    - Multithreading benchmark for apple-to-apple comparison between CPU and GPU

BACKUPS

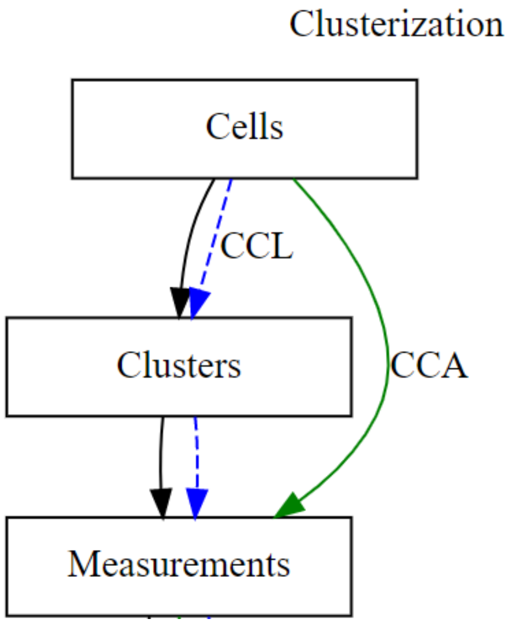
Software Support Chart

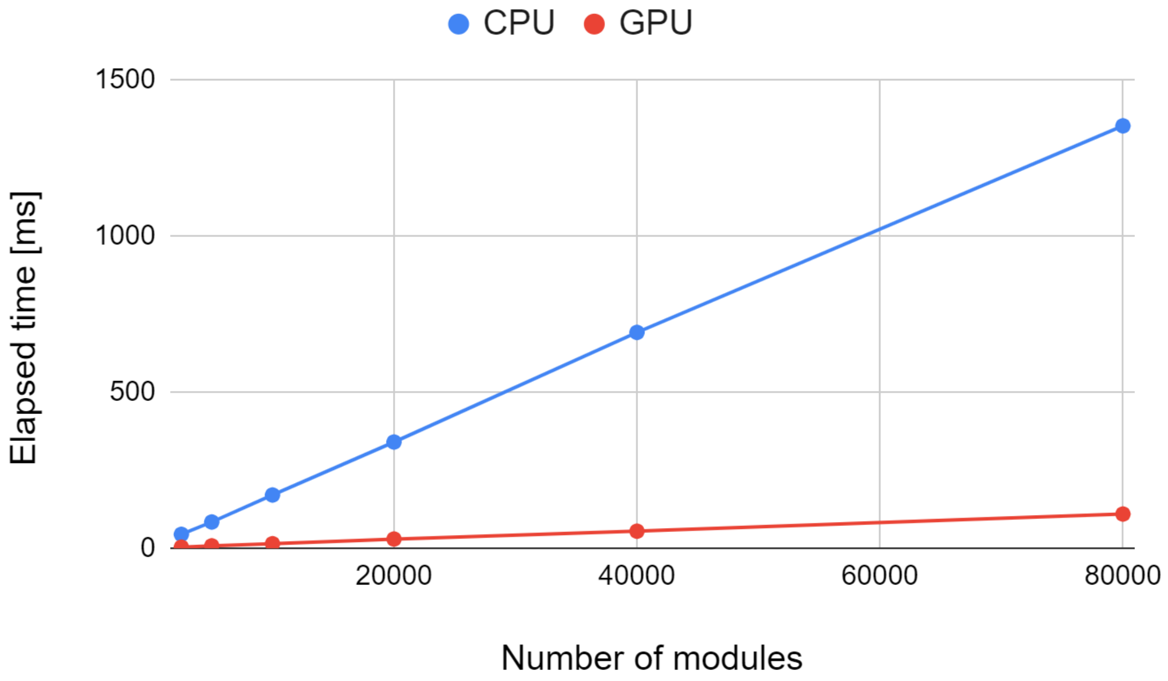
C. Leggett

	OpenMP Offload	Kokkos	dpc++ / SYCL	HIP	CUDA	Alpaka
NVIDIA GPU			*codeplay and intel/llvm*			
AMD GPU		*experimental (feature complete)*	*via hipSYCL and intel/llvm*			
Intel GPU		*prototype*		HIPLZ: *very early development*		*prototype*
CPU						
Fortran						
FPGA						*possibly via SYCL*

CCL on GPU: FastSV algorithm

- To skip the explicit cluster EDM outputs, Connected Component Analysis (CCA) has been studied by composing CCL and measurement creation
- FastSV algorithm for CCA showed promising results with CUDA

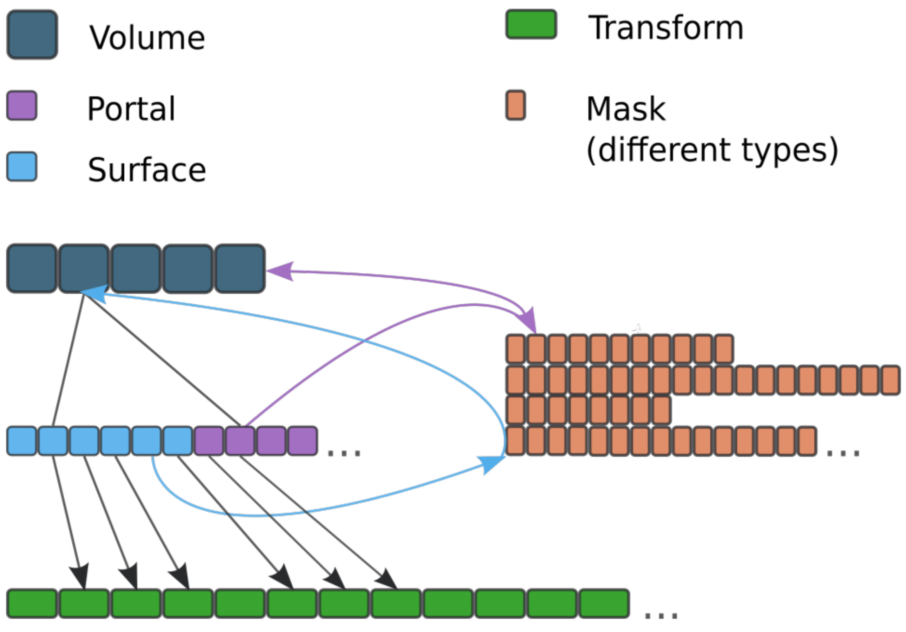




Detray Detector Model

- General concept
  - Detector components are serialized in vecmem::vector
  - Each components are linked with an index

- Detector components:
  - **Volume** keeps the indices to surfaces and portals
  - **Surface/portal** keeps the indices to mask and transform
  - **portal** is a surface that connects two volumes
  - **Transform** contains matrix for local↔global transformation
  - **Mask** is a shape of a surface (rectangle, disk, etc.) linked to each surface
  - **Material** provides the detailed description of material mapping on masks
  - **Surface grid** provides a neighborhood lookup for fast volume navigation



J. Niermann, ACAT (2021)