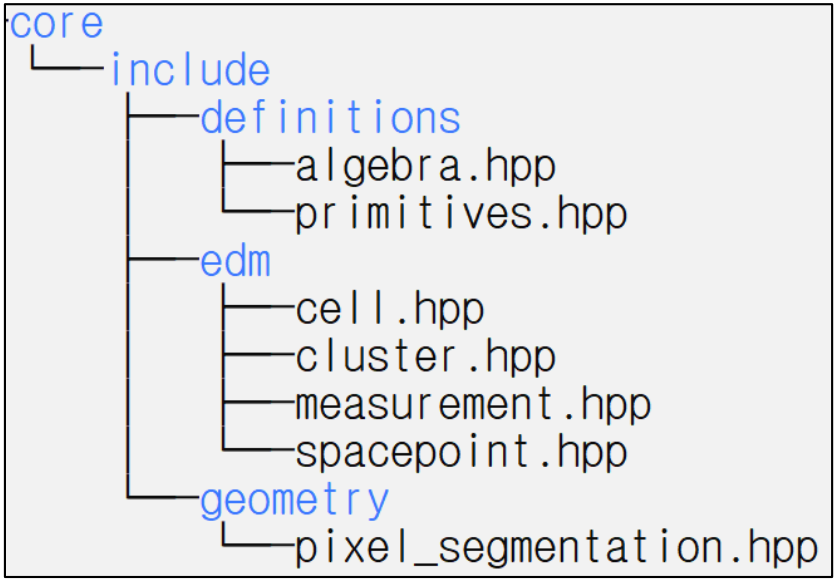
Traccc cuda implementation

Beomki Yeo

General strategy

- The project should be able to be compiled only with c++ compiler for users who do not have gpu
- Cuda algorithms should minimize the interaction with cpu
- cpu and cuda algorithms share the same source codes (e.g.)edm, geometry) as much as possible

Traccc core (common) directory



- Defines things commonly used by cpu/gpu
  - EDM for single object
  - Geometry
  - Useful algebras and data type

Traccc cpu & cuda directory

```
cpu
└── include
    ├── algorithms
    │   ├── component_connection.hpp
    │   ├── detail
    │   │   └── sparse_ccl.hpp
    │   ├── measurement_creation.hpp
    │   └── spacepoint_formation.hpp
    └── edm
        ├── cell_container.hpp
        ├── measurement_container.hpp
        └── spacepoint_container.hpp
```

```
cuda
└── include
    ├── algorithms
    │   └── spacepoint_formation
    │       ├── component_connection_kernels.cuh
    │       └── spacepoint_formation_kernels.cuh
    ├── edm
    │   ├── cell_container.hpp
    │   ├── label_container.hpp
    │   ├── measurement_container.hpp
    │   └── spacepoint_container.hpp
    └── utils
        └── cuda_error_check.hpp
```

- Event size EDM and functions are defined separately based on the single object EDM

EDM example

core/include/edm/cell.hpp

```
namespace traccc {

    using channel_id = unsigned int;

    /// A cell definition:
    ///
    /// maximum two channel identifiers
    /// and one activiation value, such as a time stamp
    struct cell {
        channel_id channel0 = 0;
        channel_id channel1 = 0;
        scalar activation = 0.;
        scalar time = 0.;
    };

    /// A module definition
    struct module_config{
        geometry_id module = 0;
        transform3 placement = transform3{};
        std::array<channel_id,2> range0 = {std::numeric_limits<channel_id>::max(), 0};
        std::array<channel_id,2> range1 = {std::numeric_limits<channel_id>::max(), 0};
    };
}
```

- cell.hpp defines single cell object

cpu/include/edm/cell_ontainer.hpp

```
namespace traccc {
    /// A cell collection:
    ///
    /// it remembers the moduleentifier and also
    /// keeps track of the cell ranges for chosing optimal
    /// algorithm.
    struct cell_collection {
        event_id event = 0;
        module_config modcfg;
        std::vector< cell > items;
    };
    using cell_container = std::vector<cell_collection>;
}
```

cuda/include/edm/cell_container.hpp

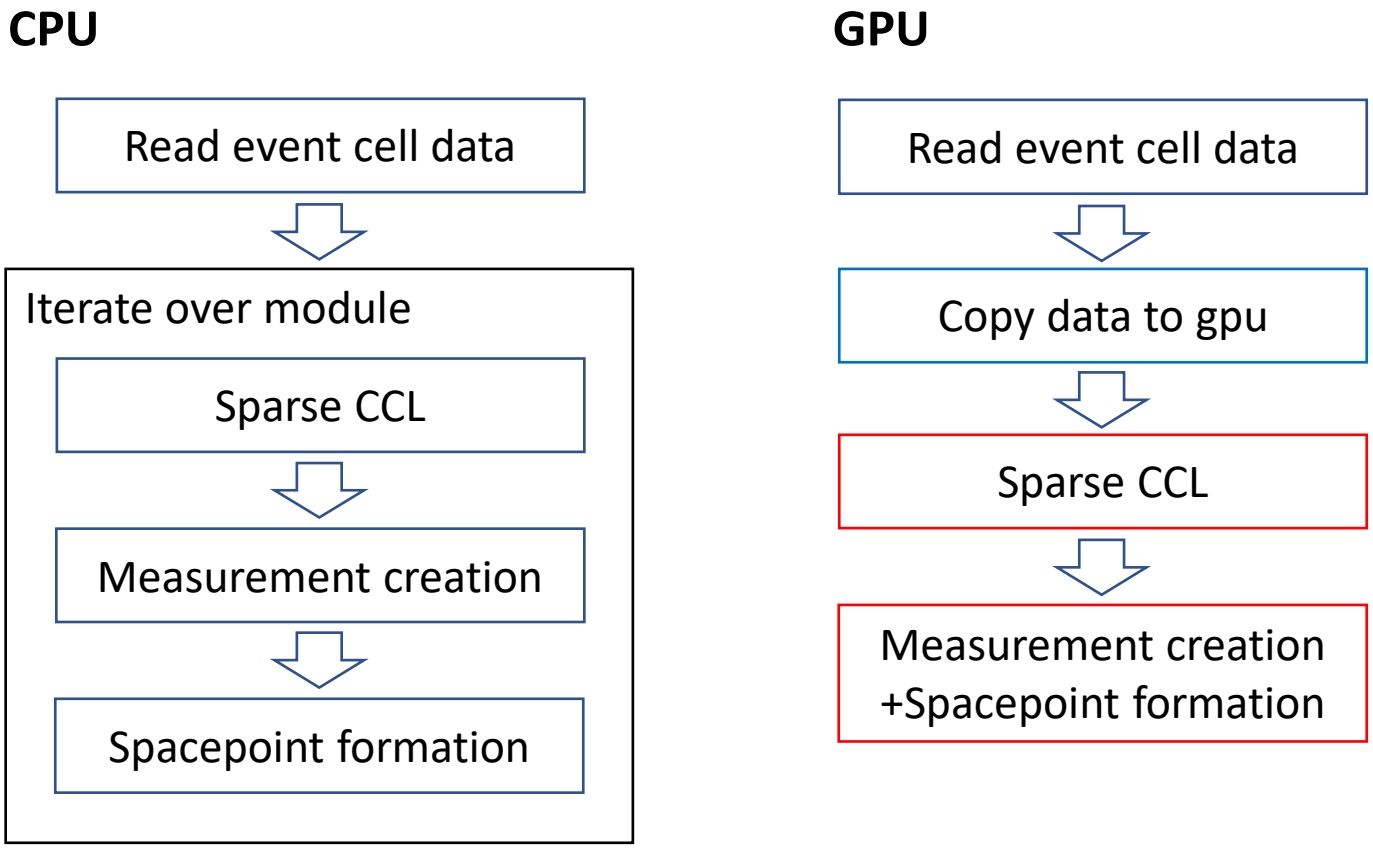
```
namespace traccc {

    struct cell_container_cuda {
        event_id event = 0;
        vecmem::cuda::managed_memory_resource m_mem;
        vecmem::vector <module_config> modcfg;
        vecmem::jagged_vector< cell > items;

        cell_container_cuda():
            modcfg(vecmem::vector<module_config>(&m_mem)),
            items(vecmem::jagged_vector<cell>(&m_mem))
        {
        }
    };
}
```

- cell_container contains the cell objects of an event

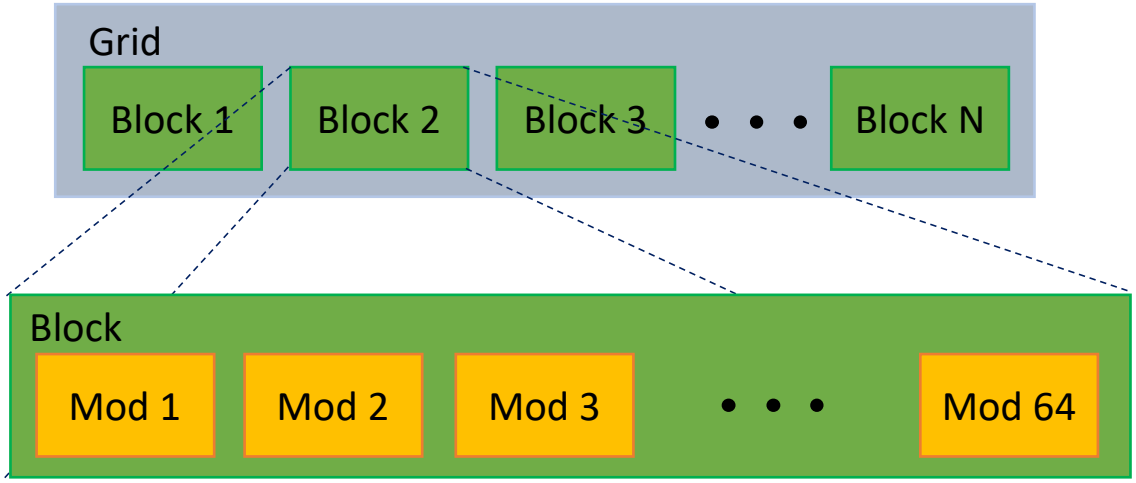
Algorithm flowchart



- CPU algorithm: Iterates over module
- GPU algorithm: One kernel launch

Parallelization scheme

- One thread handles one module
- $\text{Number of threads per block} = 2 \times \text{Warp Size}\ (32) = 64$
- $\text{Number of blocks} = \text{Number of modules}/64 + 1$



Elapsed time for an event

- TrackML data was used
- Statistics:
  - 199547 cells with 3859 modules
  - 36907 clusters from sparseCCL
  - 36907 measurements
  - 36907 spacepoints
- Fast math mode of cuda was disabled to get the same results with cpu

	CPU (Intel i7-10750H)	**GPU (RTX 2070)**
Read event cell data	0.77 sec	
Copy cell data to gpu	-	0.17 sec
Algorithm	0.074 sec	0.077 sec
Total	0.85 sec	1.03 sec

Summary

- EDM and algorithm for cuda were introduced (any feedback is welcome)
- The cuda and cpu algorithms showed similar computing speed