GPU R&D Integration

Some Conversation Points

Attila Krasznahorkay

Timeline

	Q4/2023	Q1/2024	Q2/2024	Q3/2024
Open Data Detector Demonstrator	finish demonstrator	Performance, optimisation traccc paper writing	traccc paper writing & finishing	Traccc standalone wrap-up.
ITk Demonstrator	start with ITk geometry integration	finish demonstrator	Performance studies ATLAS documentation	
ACTS Integration		traccc as ACTS/thirdparty	ACTS -> traccc offloading (Partly, full chain)	ACTS reintegration discussion
ATLAS/ATHENA Integration		Review components for direct integration in ATHENA	Traccc deomonstrator In ATHENA	

- I like Andi's table on this…

Code Placement

- At first, we could just teach Acts to build traccc as an external, with a CMake recipe under thirdparty/
  - As long as the traccc EDM is in a nice enough shape by then, copying input data into and out from the traccc EDM containers should be fairly efficient as-is
  - Could possibly provide some adaptor/helper functions in Acts at this point for creating/digesting the traccc data more easily
- Would then move the EDM and algorithms into the acts repository itself
  - How exactly we may want to do this, is the main topic of this discussion (in my mind)

EDM

- There is still discussion / R&D around this, but the input and output containers of the GPU algorithms will definitely need to be SoA
  - ATLAS, and because of that ACTS, use SoA everywhere. So for efficient data ingestion/extraction the GPU algorithms must do the same.
  - Internal data could still be AoS or some hybrid format. 🤔
- Right now I envision a continued life for vecmem here
  - I don't quite think that ATLAS xAODs will ever be a full solution for an EDM in GPU algorithms. xAODs just need to be too many things to too many people…

```
// Project include(s).
#include "traccc/definitions/common.hpp"
#include "traccc/definitions/primitives.hpp"

// VecMem include(s).
#include "vecmem/edm/accessor.hpp"
#include "vecmem/edm/container.hpp"

namespace traccc::edm {

/// Pixel cell (SoA) container
struct pixel_cell_container
    : vecmem::edm::container<vecmem::edm::schema<
          vecmem::edm::type::vector<channel_id>,
          vecmem::edm::type::vector<channel_id>,
          vecmem::edm::type::vector<scalar>,
          vecmem::edm::type::vector<scalar>,
          vecmem::edm::type::vector<unsigned int> > > {

    /// @name Accessors for the individual container variables
    /// @{

    /// First (x) channel identifier of the cell
    using channel0 = vecmem::edm::accessor<0, schema>;
    /// Second (y) channel identifier of the cell
    using channel1 = vecmem::edm::accessor<1, schema>;
    /// Activation / signal strength of the cell
    using activation = vecmem::edm::accessor<2, schema>;
    /// Time of the cell signal
    using time = vecmem::edm::accessor<3, schema>;
    /// Index of the pixel module that the pixel cell belongs to
    using module_index = vecmem::edm::accessor<4, schema>;

    /// @}

};  // struct cell_container

}  // namespace traccc::edm
```

Algorithms → Tools (1)

```
namespace traccc::cuda {

/// Main algorithm for performing the track seeding on an NVIDIA GPU
///
/// This algorithm returns a buffer which is not necessarily filled yet. A
/// synchronisation statement is required before destroying this buffer.
///
class seeding_algorithm : public algorithm<seed_collection_types::buffer(
                          const spacepoint_collection_types::const_view&)> {

    public:
    /// Constructor for the seed finding algorithm
    ///
    /// @param mr The memory resource to use
    /// @param mr The memory resource(s) to use in the algorithm
    /// @param copy The copy object to use for copying data between device
    ///             and host memory blocks
    /// @param str The CUDA stream to perform the operations in
    ///
    seeding_algorithm(const seedfinder_config& finder_config,
                      const spacepoint_grid_config& grid_config,
                      const seedfilter_config& filter_config,
                      const traccc::memory_resource& mr, vecmem::copy& copy,
                      stream& str);

    /// Operator executing the algorithm.
    ///
    /// @param spacepoints_view is a view of all spacepoints in the event
    /// @return the buffer of track seeds reconstructed from the spacepoints
    ///
    output_type operator()(const spacepoint_collection_types::const_view&
                           spacepoints_view) const override;

    private:
    /// Sub-algorithm performing the spacepoint binning
    spacepoint_binning m_spacepoint_binning;
    /// Sub-algorithm performing the seed finding
    seed_finding m_seed_finding;

};  // class seeding_algorithm

}  // namespace traccc::cuda
```

- This is one area where I don't quite see all the angles yet. ☹
  - While we call them "algorithms", the things in traccc are more "tools" than "algorithms". (There's no framework that would schedule them, we just call them explicitly in our test applications.)
  - This is by design, as I knew that eventually they would need to become tools anyway.

Algorithms → Tools (2)

- Input/output data would either need to be handled through an Acts specific EDM, or through a mechanism similar to the one used for the TrackContainer
  - We anyway have to make a copy of the data for the GPU algorithms. (Experiment specific EDM will at most put data into shared/managed memory. Which is not good enough for performance.)
  - So we may as well require the GPU tool users to put data into a vecmem based EDM with some helper code. And then provide them with helper code for extracting the data from a vecmem based output into their own EDM.
    - However a “TrackContainer-like approach” would allow us to put the same API on the CPU and GPU implementations of the tools, while putting the burden of data movement to/from the GPU on our code. 🤔

Main Points (For Me…)

- GPU based tools can at maximum have a thin templated layer as an API
  - It would be a bad design to leave the compilation of GPU code to the client software. So the Acts API must hide all the GPU code from the interfaces.
- This means that EDM questions are even more important than for the current Acts tools
  - The fully templated API of Acts::SeedFinder for instance does not fly for a GPU implementation
  - If we want to make it easy for clients to switch between the CPU and GPU implementations, the CPU implementation's API will have to change as well
  - Luckily the effort with TrackContainer is a good direction for the GPU EDM as well as far as I can see

http://home.cern