Acts Parallelization R&D

Beomki Yeo (UC Berkeley and LBNL)
On behalf of ACTS Project
Motivation for GPU Track Reconstruction

- The track reconstruction of HL-LHC will face into a huge computation
  - Conservative budget model for CPU computing is pessimistic

- Major software development is necessary for the acceleration of online and offline computing
  - GPU may play a key role with higher speed and lower power consumption beyond conventional CPUs

- Track reconstruction problem is inherently parallelizable, thus, a good fit to GPU offloading
Acts as a GPU Demonstrator

- Acts is a general track reconstruction toolkit for HEP experiments
  - open source
  - Thread-safe design
  - Adopts Modern C++17 concept

- There are already some core algorithms ported for CUDA and SYCL but there is a clear limit when it comes to full offloading
  - C++17 features is supported with a restriction in device code
  - Some event data model and geometry are not GPU-friendly

- Instead, Acts community has decided to work on GPU demonstrator by launching several R&D projects
R&D Projects for Acts Parallelization

- **traccc**
  - Demonstrator for tracking algorithms in GPU

- **detray**
  - GPU geometry builder

- **algebra-plugin**
  - Vector and matrix algebra for multiple plugins

- **vecmem**
  - GPU memory management tool for other R&D projects
vecmem: GPU Memory Management Tool

- Make use of `std::pmr::memory_resource` to customize the allocation scheme in the host side
  - Supports CPU, CUDA, SYCL, and HIP

- Provides STL-like containers for host side:
  - `vecmem::vector`
  - `vecmem::jagged_vector` (vector of vector)
  - `vecmem::array`

- There are also containers for device code:
  - `vecmem::device_vector`
  - `vecmem::jagged_device_vector`
  - `vecmem::device_array`

- Extensively used for detray and traccc to describe detector and event data model

```cpp
namespace vecmem {
    template <typename T>
    using vector = std::vector<T, vecmem::polymorphic_allocator<T>>;
}

namespace vecmem::cuda {
    void *host_memory_resource::do_allocate(std::size_t bytes, std::size_t) {
        // Allocate the memory.
        void *res = nullptr;
        VECMEM_CUDA_ERROR_CHECK(cudaMallocHost(&res, bytes));
        return res;
    }

    void host_memory_resource::do_deallocate(void *, std::size_t, std::size_t) {
        // Free the memory.
        VECMEM_CUDA_ERROR_CHECK(cudaFreeHost(p));
    }
}
```
vecmem User Interface

- The vecmem interface is pretty simple compared to vanilla usage of GPU APIs

```cpp
__global__ do_something(vecmem::data::vector_view<int> input);
int main() {
  // The managed memory resource.
  vecmem::cuda::managed_memory_resource managed_resource;

  // Create an input in managed memory.
  vecmem::vector<int> inputvec({1, 2, 3, 4, 5, 6, 7, 8, 9, 10}, &managed_resource);

  // Run CUDA kernel on input vector
  do_something<<<1, 1>>>(vecmem::get_data(inputvec));
}

__global__ do_something(vecmem::data::vector_view<int> input){
  // Get device vector from input
  vecmem::device_vector<int> vec(input);

  // ... do something ...
}
```
algebra-plugin: Vector & Matrix Algebras

- Development of fast algebra for both CPU and GPU, which includes:
  - home-brew array plugin
  - Eigen3
  - SMatrix
  - Vc

- Currently focusing on the development of matrix operation per-thread but also planning to add operations per-block to benefit from the GPU architecture
detray: GPU Geometry Builder

- Motivation of the project:
  - Current Acts geometry highly relies on runtime polymorphism, which is not so GPU friendly

- Goals of the project:
  - Build the vecmem-based tracking geometry
  - Capable of translating Acts geometry into detray one

- Deliverables:
  - Classes for detector and its sub-detector components
  - Magnetic fields
  - Tools for geometry navigation with stepping algorithms
Detector Model in detray

- General concept
  - Detector subcomponents are serialized in vecmem-based container
  - They are inter-linked with an index rather than a pointer to avoid run-time polymorphism

- Detector subcomponents:
  - **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
  - **Surface grid** provides a neighborhood lookup for volume local navigation
The host and device trait of detector fully depends on the vecmem container type
- host detector with `vecmem::vector`
- device detector with `vecmem::device_vector`

```cpp
// 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 container with vecmem::vector
    detector<vecmem::vector> host_detector(resource);

    // ... build detector ...

    // obtain data container
    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 container with vecmem::device_vector
    detector<vecmem::device_vector> device_detector(data);

    // ... do something with parallelization
}
```
- NVIDIA GPU texture memory has been studied for magnetic field lookup and interpolation.

- Trilinear interpolation using the GPU’s built-in hardware, thus, no software overhead.
  - For 8192 × 8192 image, the CUDA kernel time for rendering is ~20 ms.

ATLAS magnetic field rendered at z = 0mm
Tools for Propagation

- Stepper
  - Track state changes based on the stepping algorithm
  - Linear and adaptive Runge-Kutta-Nyström 4th order

- Navigator
  - Take a detector as an input argument
  - Finds candidate surfaces in a volume by shooting a ray
  - Update linear distances between track and candidates
  - Pass a step size limit to the stepper

- Propagator
  - Steers the workflow of stepper and navigator
CUDA speed was benchmarked with the pixel part of trackML detector

- Runge-Kutta stepper with constant 2 T
- One order of magnitude of speedup with $O(10^4)$ tracks

![Graph showing time vs. number of tracks for CPU and GPU benchmarks with different precision levels.](image-url)
traccc: GPU Demonstrator for Tracking Algorithms

- traccc aims for demonstrating tracking algorithms on GPU
- The event data model (EDM) with vecmem-based container
  - Currently focusing on CPU, CUDA, and SYCL
    - HIP and std::par will be investigated as well
Hit Clusterization

- Connected Component Labeling (CCL)
  - Cluster making algorithm

- Measurement creation
  - Calculate the weighted average of cluster cell positions and covariances

- Spacepoint formation
  - local to global transformation
  - input to seeding algorithm

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

Connected Component Labeling (CCL)

Spacepoint formation
Current Progress in Hit Clusterization

- 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.
Seeding

- Binning spacepoints
  - Grouping hits on two dimension grid

- Seed finding
  - Doublet search
  - Triplet search
  - Triplet filter

- Track parameter estimation
  - Global to local transformation on surface
  - Input to Kalman filtering

Seed finding

Track parameter estimation
Seeding Benchmarks

- Each sub-algorithm of seeding parallelizes over spacepoints, doublets and triplets.

- For 200 pileups of ttbar events in trackML detector, one order of magnitude of speedup is achieved from CUDA and SYCL with single precision.

- Interestingly, SYCL showed ~10% better speedup compared to CUDA.
Prospect for Kalman Filtering

- Two parallelization schemes under discussion:
  - one track per thread
  - one track per block

- one track per thread
  - Algebra is easy to implement
  - Will suffer from branching conditions

- one track per block
  - Hard to optimize the algebra in a given dimension of threads
  - Free from branching conditions

- In Acts design, Kalman Filter is an extension of the propagator, hence closes the loop to detray project

- **per thread scheme**

- **per block scheme**
Summary

- Acts R&D projects are being developed to offload tracking algorithms onto GPUs
- **vecmem** is the core library for defining detector geometry and event data model
- **algebra-plugin** provides essential algebras to detray and traccc while efficient matrix algebra is under development and discussion
- **detray** constructs the tracking geometry for (combinatorial) Kalman filtering
  - In principle, detector is fully usable in host and device side
  - Benchmark study on propagation is promising
  - Still a lot of works remain to be done for more flexible detector design and Acts geometry translation
- **traccc** is the downstream project for GPU tracking demonstration
  - Successfully demonstrated seeding algorithm on CUDA and SYCL
  - Plans to harmonize with detray detector for Kalman filtering development

FAQ Participation?
- acts-parallelization@cern.ch
- Bi-weekly Acts Parallelization Meeting, Fri at 16:00 [indico]
BACKUP
## detray Project Status

<table>
<thead>
<tr>
<th>Types</th>
<th>CPU</th>
<th>GPU (CUDA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detector</td>
<td></td>
<td></td>
</tr>
<tr>
<td>volume container</td>
<td></td>
<td></td>
</tr>
<tr>
<td>surface container</td>
<td></td>
<td></td>
</tr>
<tr>
<td>transform container</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mask container</td>
<td></td>
<td></td>
</tr>
<tr>
<td>surface grid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Field</td>
<td></td>
<td></td>
</tr>
<tr>
<td>constant</td>
<td></td>
<td></td>
</tr>
<tr>
<td>realistic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tools</td>
<td></td>
<td></td>
</tr>
<tr>
<td>local navigation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>global navigation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Runge-Kutta stepping</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Propagation</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Green**: Merged
- **Yellow**: Work in progress
- **Gray**: Not yet started
## traccc Project Status

<table>
<thead>
<tr>
<th>Types</th>
<th>Algorithms</th>
<th>CPU</th>
<th>CUDA</th>
<th>SYCL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hit clusterization</td>
<td>CCL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>measurement creation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>spacepoint formation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Track finding</td>
<td>binning spacepoints</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>seed finding</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>track param estimation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Combinatorial KF</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Track fitting</td>
<td>KF</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Merged**
- **Work in progress**
- **Not yet started**