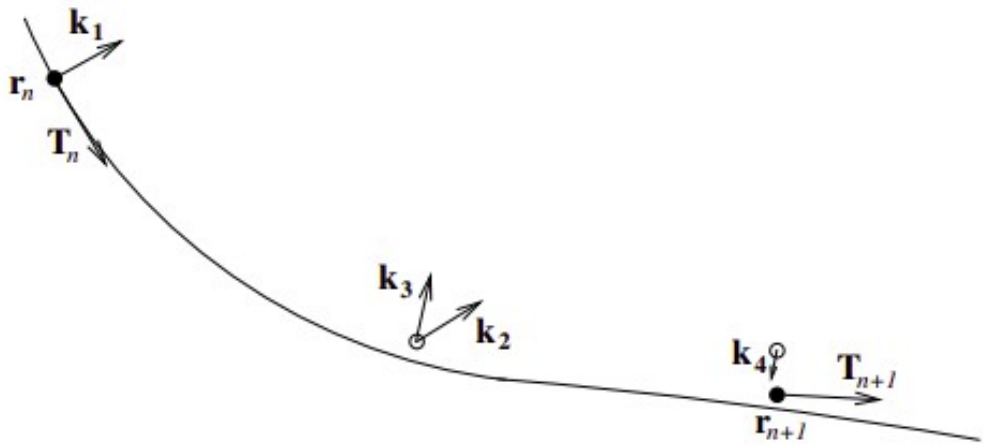
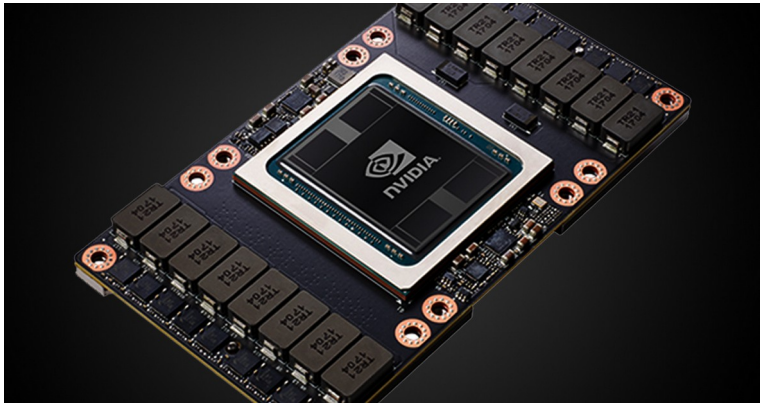


ACTS Propagation on GPUs

Xiaocong Ai

ACTS Workshop 2020, May 28, 2020



Beginner experience with GPU/CUDA

- What are working
  - Template
  - Virtual function
  - Eigen library

- C++ STL containers (e.g. std::vector, std::array) and algorithms (e.g. std::sort) are not usable
  - Use C array or Eigen::array/Eigen::matrix (static memory allocation)
  - Use explicit heap memory allocation where needed
- Problem of using pointer data member
  - A host pointer is just invalid on device!

ACTS propagator from Host to Device

Host

Device

An Acts::Propagator

An Acts::EigenStepper

An Acts::InterpolatedBFieldMap

An Acts::InterpolatedBFieldMapper

An Acts::Grid

std::vector<T> m_values
replaced by T* m_values

Large size grid values requires allocation on heap

cudaMemcpy

An Acts::Propagator

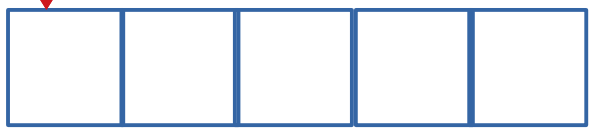
An Acts::EigenStepper

An Acts::InterpolatedBFieldMap

An Acts::InterpolatedBFieldMapper

An Acts::Grid

T* m_values (point to host memory)



Host memory

ACTS propagator from Host to Device

Host

Device

An Acts::Propagator

An Acts::EigenStepper

An Acts::InterpolatedBFieldMap

An Acts::InterpolatedBFieldMapper

An Acts::Grid

std::vector<T> m_values
replaced by T* m_values

cudaMemCpy

An Acts::Propagator

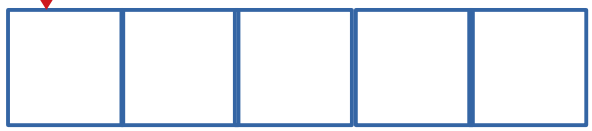
An Acts::EigenStepper

An Acts::InterpolatedBFieldMap

An Acts::InterpolatedBFieldMapper

An Acts::Grid

T* m_values (explicitly set to point to grid values on device)

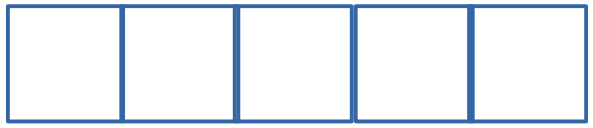


Host memory

cudaMemcpy

Explicit copy of the grid values!

works!



Device memory

ACTS propagate call as CUDA Kernel

Acts::Propagator propagate call (without destination surface)

```
template <typename parameters_t, typename propagator_options_t,
          typename path_aborter_t = PathLimitReached>
Result<action_list_t_result_t<
    CurvilinearParameters, typename propagator_options_t::action_list_type>>
propagate(const parameters_t& start,
          const propagator_options_t& options) const;
```

Propagate CUDA Kernel

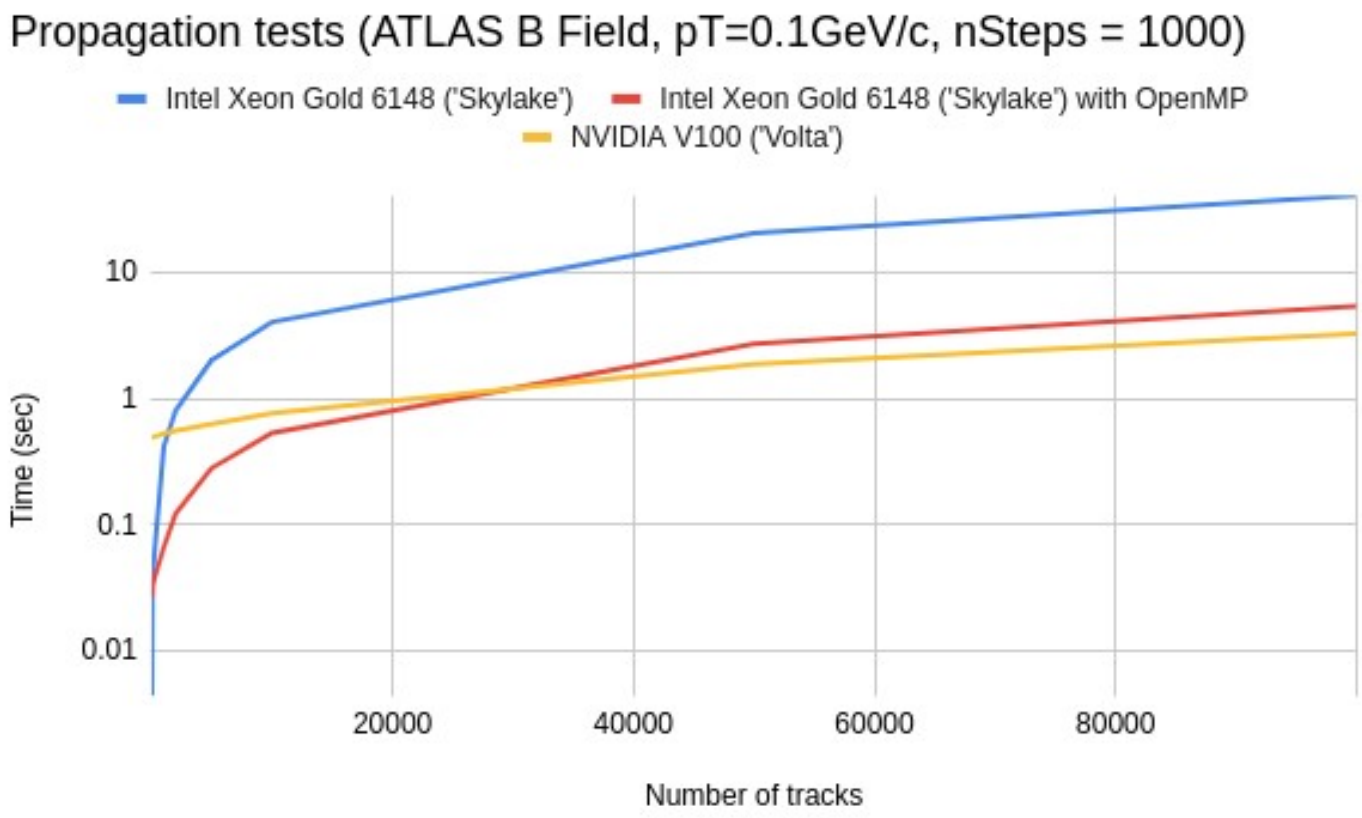
```
// Device code
__global__ void propKernel(PropagatorType *propagator, TrackParameters *tpars,
                           PropagatorOptions *propOptions,
                           PropResultType *propResult, Vector3D *gridValPtr,
                           int N) {
  // Awkwardly make the grid values pointer to point to memeory on device
  // explicitly
  propagator->refStepper().refField().refMapper().refGrid().refValues() =
      gridValPtr;

  int i = blockDim.x * blockIdx.x + threadIdx.x;
  if (i < N) {
    propagator->propagate(tpars[i], *propOptions, propResult[i]);
  }
}
```

The explicit reset of grid value pointer inside kernel

Propagation on GPU

- Results on GPU are validated
- GPU win over CPU at many threads when $N_{propgations}$ > 30k



Conclusion&Discussion

- Dynamic memory allocation on host (in particular for a deep nested object) could be a problem for CUDA data transfer (cudaMemcpy())
  - The copied object to device might be ‘incomplete’
- Allocating Unified Memory (cudaMallocManaged()) on device could returns a pointer that is valid on both host and device
- Some relevant and necessary classes might be designed to be templated on the memory allocation mode, i.e. on host or device (with GPU present)
  - If specialized to device allocation mode, the class should be immediately usable after copied from host to device