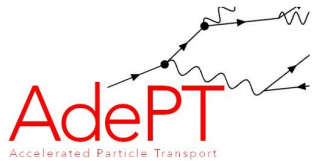


AdePT: Status & Plans

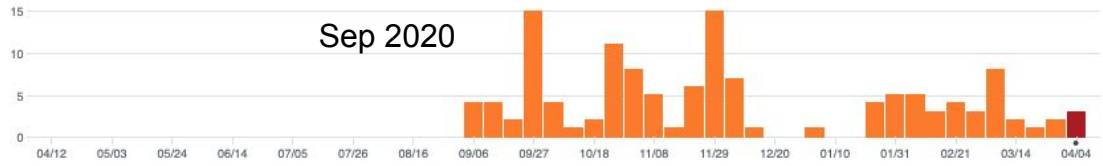
HSF Detector Simulation WG
April 12, 2021

andrei.gheata@cern.ch

R&D for EM physics transport simulation on GPU

- EM particle transport specialisation for GPU
  - GPU plug-in for Geant4 EM transport for specific “regions” as a special “fast simulation” process
  - Possible standalone usage in specific applications
- Implement technical solutions for running particle transport on GPU
  - Understand portability/implementation/optimisation of components for GPU
    - Geometry, EM physics, field propagator
  - Profit from track-level parallelism for EM showers
    - Implement and evaluate different data models, scheduling strategies
- Optimise performance and understand limitations
  - Demonstrating a realistic simulation workflow on GPU and CPU↔GPU
  - Justify investment in a large-scale project

The AdePT project



- GitHub repository
  - Started in September 2020, 10 contributors so far
- Strategy: evolve to a prototype based on gradually more complex examples
  - Core types/macros/abstractions reusable outside AdePT separated into an externalizable library for GPU (***CopCore***)
  - Gradually consolidating infrastructure, common services and types
  - External physics (***G4HepEm***) and geometry (***VecGeom***)
- Rapid evolution on a tight roadmap
  - **Sep 2021:** *First complete AdePT simulation running, including all EM interactions, tracking in non-constant magnetic field (optionally), sensitive detector functionality*
  - **~ Jan 2022:** *HSF meeting discussing the results and planning a community wide-project*

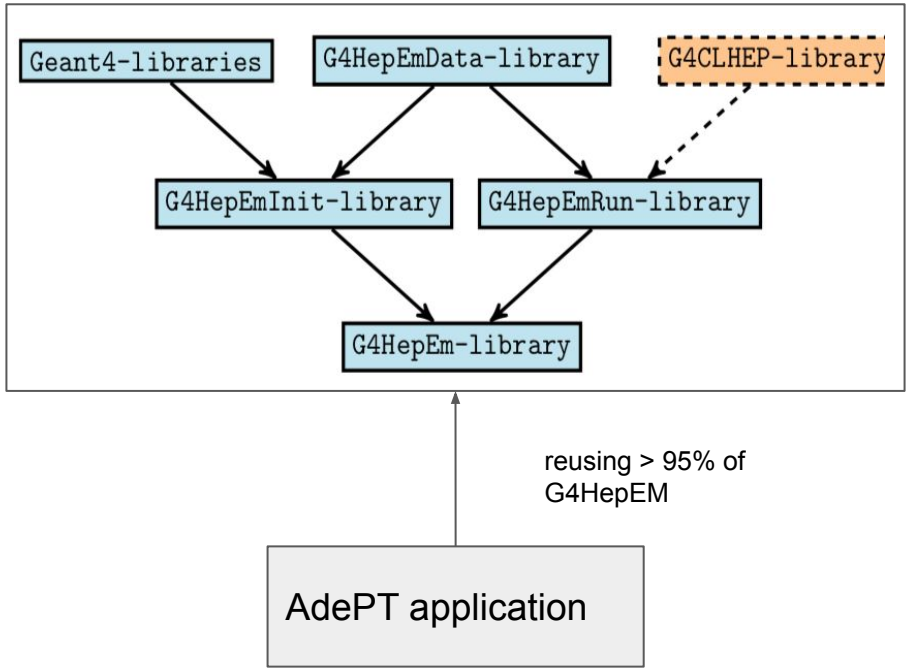
Components: physics

- ***G4HepEm*** library, a Geant4 EM physics working group R&D
  - Compact specialized rewrite of $e^+/e^-/\gamma$ physics
    - Much simplified compared to the current Geant4 implementation
  - Clear **separation** between *initialization* (relying on Geant4) and *run-time* (independent)
    - Storing internally in simple structures just the properties needed at run-time
  - Presentation from M. Novak available here
- Provides physics tables and interfaces usable at run-time on GPU
  - Initialize the setup using Geant4 and construct/transport physics tables to GPU
    - Restricted ranges, macroscopic cross sections, final states differential cross sections
  - Functional programming approach for sampling the physics step length and final state w/o needing Geant4

Components: physics

M. Novák, SFT R&D meeting, 8 Dec 2020

- G4HepEmData: data structure used run-time
- G4HepEmRun: run-time functionality
  - reading/interpolation of datas structures
  - Wrapper class to abstract the PRNG
- Can run on CPU and GPU
  - Same code on host & device
  - Ideal for comparing performance
- Integration with AdePT complete
  - Ongoing work for importing the *G4HepEm* data and geometry via files



J. Hahnfeld, SFT R&D meeting, 9 Feb 2020

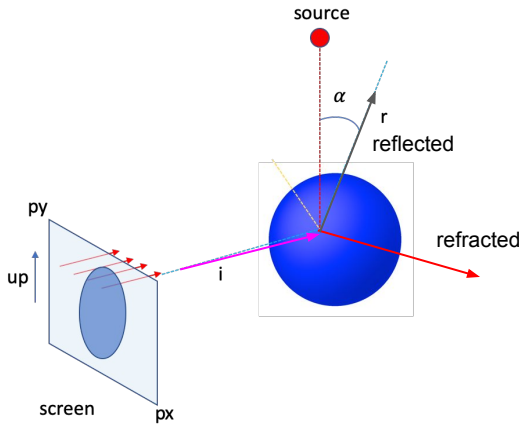
Components: geometry

- Using VecGeom as geometry engine
  - Using indexed navigation states to reduce memory footprint per track
  - Loading geometry using *vecgeom::vgdml* parser and using *vecgeom::CudaManager* to transfer the geometry to device
  - Keeping the navigation layer in AdePT (for now)
- Advantages:
  - Ability to read existing geometry setups using GDML
  - Reuse of the same algorithms on CPU and GPU
  - Requiring minimal effort for having full geometry functionality for prototyping
- Disadvantages:
  - CUDA-specific, not usable as-is with portability frameworks
  - Virtual functions and algorithm diversity
    - Reduced compiler optimisation opportunities (stack size, register pressure)
    - Bad warp synchronisation

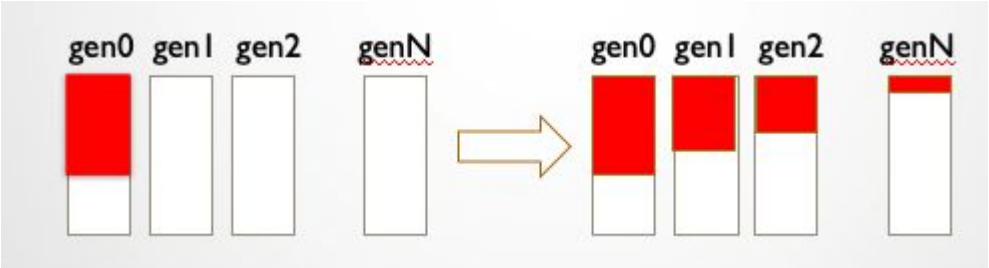
Components: geometry

- Currently using a very simple *LoopNavigator* doing reductions for *Locate/ComputeDistance* functionality
  - Main bottleneck in all examples using non-trivial geometry
  - Ongoing work
- Ongoing work for “de-virtualising” the navigation calling sequence
  - PR 106 in AdePT and MR 797 in VecGeom - *switch* statement dispatch brings ~10% gain
    - Possible since the implementation layer in VecGeom is standalone
  - Preliminary study of using *std::variant*: this would be the ‘clean’ C++17 approach, not supported directly yet by CUDA (*std::visit*)
- Enhanced ray-tracing example in AdePT, including secondary ray generation
- Work on CUDA-friendly BVH acceleration structure on the critical path
- Work on using single-precision in geometry

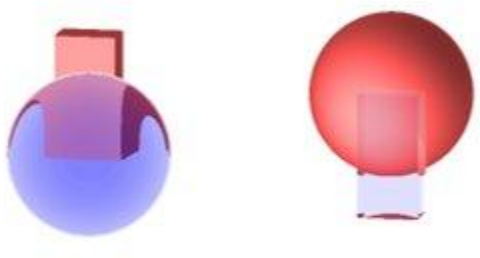
Ray-tracing + reflection / refraction / attenuation



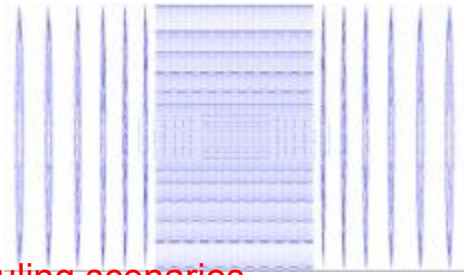
A. Petre, SFT R&D meeting, 9 Mar 2021



Transport to boundary only, then generate reflected ray and increase generation, moving to next container. Continue tracking the refracted ray to next boundary. AdePT PR #105



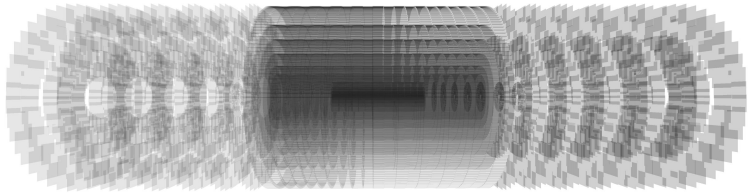
trackML



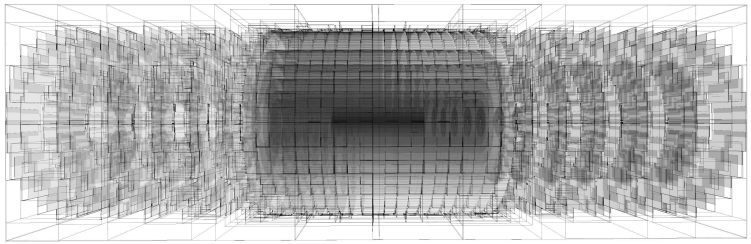
excellent playground for different scheduling scenarios

BVH acceleration

- Acceleration structure for reducing the number of candidate checks
  - Concurrent collision detection against aligned bounding boxes as first step
  - Used natively in RTX hardware, available via Nvidia Optix
  - A software implementation can also help reducing the current geometry bottleneck
- Implementation ongoing
  - Also followed in Issue #39
  - Goal: integrate eventually in VecGeom (usable then by multiple clients)



TrackML



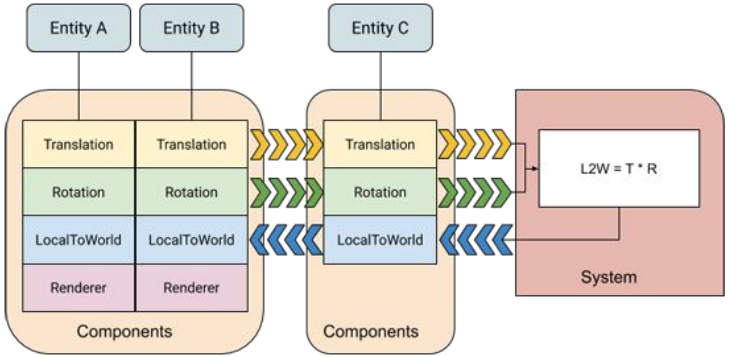
G. Amadio, SFT R&D meeting, 23 Mar 2021

Data layout investigations

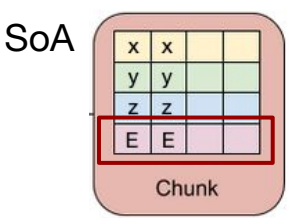
- The data layout has major importance on GPUs
  - Coalesced accesses are much faster
  - Reading more data than actually used is expensive
- Data should be laid out as compact as possible
  - This is partly the case for geometry and physics tables
  - Coalescing memory may be quite challenging depending on the scheduling strategy (followed in issue #67)
- The track data structure design and interface are important
  - Minimizing memory reads
  - Making code dependent on “components” rather than on a given track type

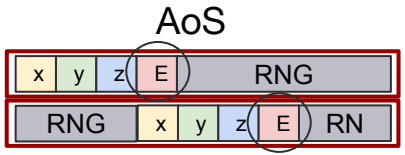
Testing an entity component approach

- Entity components used extensively in gaming
- SoA vs. AoS
  - partial vs. full reads (no need to read x,y,z if we need just E)
  - naturally coalesced in memory
  - more difficult to program accessing elements
    - like: *tracks.E[i]*
    - can be dealt with “views” allowing to program like: *tracks[i].E -> SoR (structure of references)*
  - what about holes left by tracks dying?
  - special example demonstrating this approach



S. Hageboeck, SFT R&D meeting, 23 Mar 2021





SoA(R) versus AoS

- Various operations on track data structure from one-field writes to particle transport with random numbers
- SoA consistently faster, better cache locality, less memory transfers
- **Can program SoA in a struct-like fashion** if using structs of references
- Structs of references are as fast as direct access
- Merging blocks of tracks when having many holes have **minimal cost**
- See ECS example

		CPU	GPU	GPU 2nd run (hot cache)	GPU run 100x (hide launch overhead)
Enumerate particles	SoA	0.93	0.10	0.02	0.40
	AoS	13.73	0.36	0.32	33.63
	SoA/AoS	14.7 x	3.6 x	13.8 x	84.7 x
Seed RNG	SoA		132.55		
	AoS		120.53		
	SoA/AoS		0.9 x		
Initialise position and momentum, Compute energy	SoA	11.09	0.09	0.08	7.32
	AoS	17.74	1.77	1.83	220.36
	SoA/AoS	1.6 x	20.3 x	22.9 x	30.1 x
Transport particle by random distance	SoA	16.88	0.55		53.00
	AoS	19.46	0.92		97.58
	SoA/AoS	1.2 x	1.7 x		1.8 x

AdePT examples

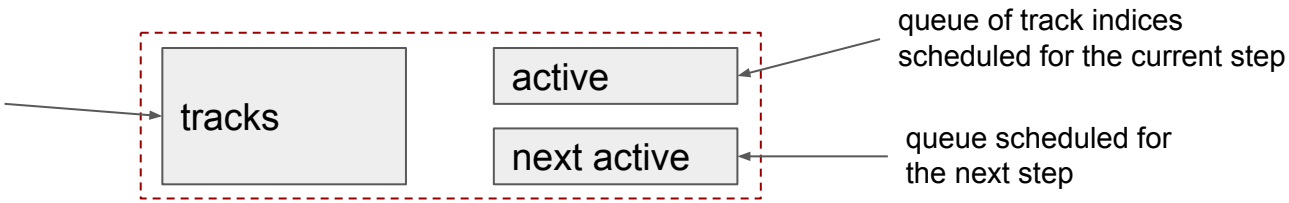
- Many scenarios and hypothesis implemented as examples
  - Starting with preliminary embryonic ideas implementing a Fisher-Price-like workflow
    - Now being removed...
- Series of numbered examples implementing new/improved features
  - magnetic field, geometry propagation/relocation, more physics processes, more materials
- Series of specialized examples, testing specific functionality: data layout, raytracing, Alpaka or OneAPI conversion
- As implementing common approaches, they get reused in several examples

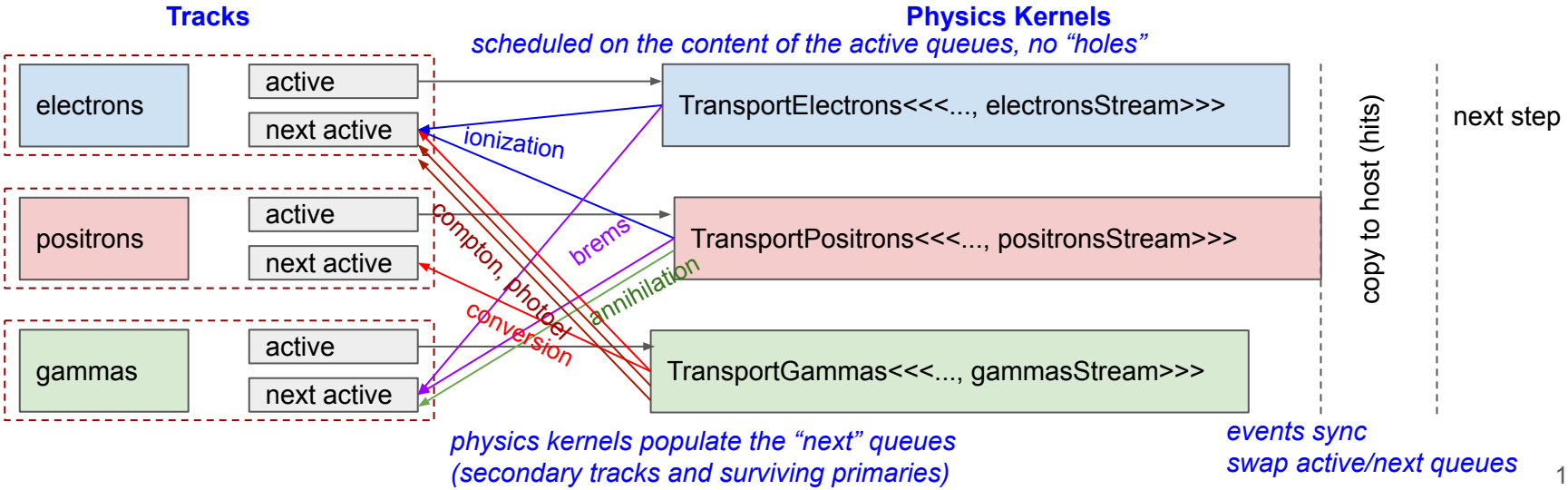
State-of-the art example 9

- Geometry loaded via VecGeom (TrackML for now)
- Bz constant field
- Physics processes for e+/e-/gamma using G4HepEm (no **MSC** yet)
- AoS tracks, storing one RANLUX++ state per track for reproducibility
- Separate containers per particle type, monotonic slot numbers (not reused)
- Overlapping TransportXXX kernels in different streams per particle type.
  - Synchronization via CUDA events to finish “stepping loop”
- Scheduling queues of indices of tracks in flight (indirection, no “hole” dispatch)
- Basic scoring transferred to host
  - number of hits, number of secondaries and energy deposit

Example9 stepping loop workflow

- contiguous
- next slot atomic
- no slot reuse
- different for e+/e-/gamma

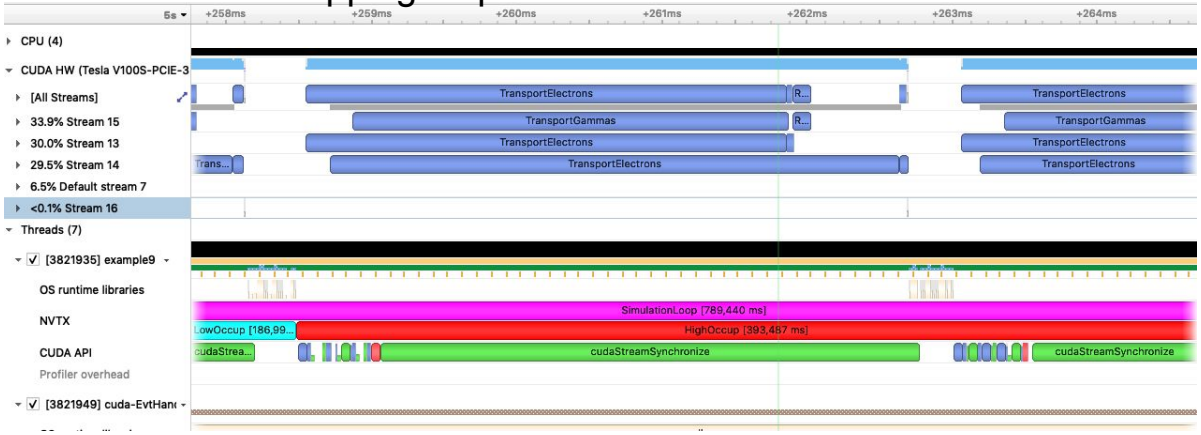




Preliminary performance analysis

- Tesla V100
- Default arguments
  - One 100GeV electron
  - ~ 3500 particles peak
  - TrackML geometry
- Workflow:
  - 2.5 s non-GPU init
  - 2.2 s VecGeom init
  - 0.8 s simulation
- Kernels run in parallel
- Low device occupancy ~10%
  - V100 can handle much more work

one stepping loop in the middle of the simulation



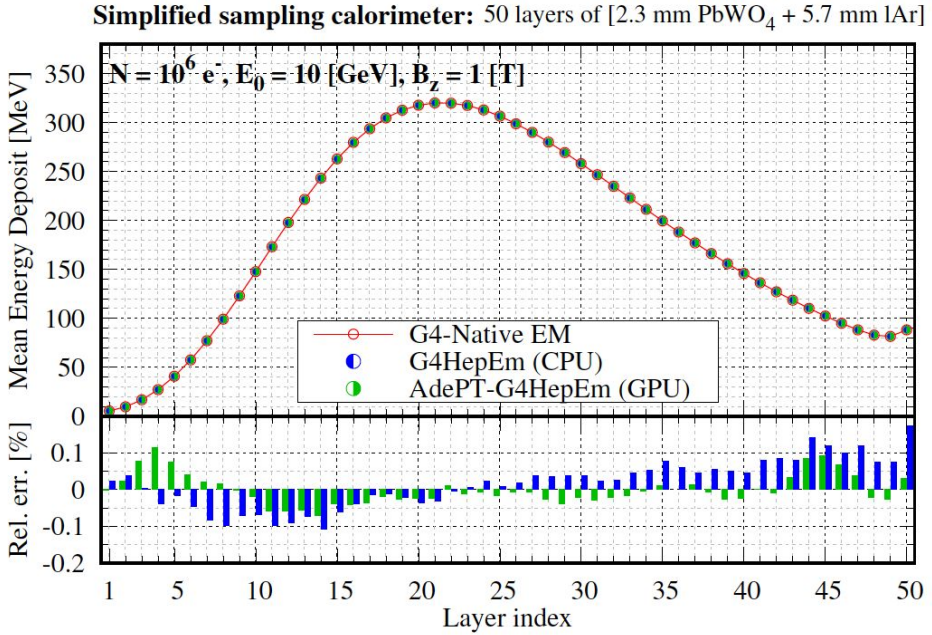
Main hotspot: LoopNavigator.h:101 calling virtual function *DistanceToIn* for daughter volumes - warps effectively serialised

- badly needs geometry acceleration structure
- good news: there is much performance to harvest
- Performance optimisation phase just started

Benchmarking example: TestEm3

- Simplified sampling calorimeter
- All features in example 9
  - no field / 1 T Bz field
- G4HepEm equivalent example on CPU
  - validation and comparison for batch size = 26

	Geant4	G4HepEm			
	G4Em (CPU)	HepEm (CPU)	**Rel. err. [%]**	AdePT (GPU)	**Rel. err. [%]**
Energy deposit per material [MeV]					
$PbWO_4$	6002.50	6002.66	0.00267	6005.69	0.05321
lAr	3165.12	3165.73	0.01927	3160.66	-0.14097
Number of secondaries					
γ	4375.94	4375.69	-0.00574	4374.93	-0.02312
e^-	6620.39	6618.69	-0.02565	6621.62	0.01855
e^+	429.189	429.161	-0.00645	429.139	-0.01165
Number of steps					
charged	18381.5	18034.6	-1.88722*	19554.6	6.38196*
neutral	58975.2	58954.4	-0.03527	59278.5	0.51428



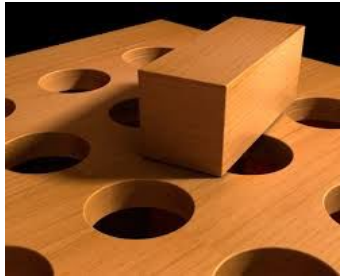
J. Hahnfeld, M. Novak, SFT R&D meeting, 23 Mar 2021

Performance of TestEM3

- No magnetic field; 10,000 electrons of 10 GeV
- AMD Ryzen 9 3900 (12C/24T), GeForce RTX 2070 SUPER
  - Geant4-10.7.1 (1 thread): 497 seconds (G4HepEm: 489 s)
  - AdePT (GPU): 115 seconds (default batch size of 26 particles) - 41.3 since last Friday
  - Geant4-10.7.1 (24 threads): 43 seconds (G4HepEm: 43 s)

- Baseline version of EM shower simulation on GPU
  - Physics processes for e−, e+, γ generating secondaries, VecGeom geometry and field
  - Results validated against Geant4
  - Baseline for performance analysis and testing different ideas
    - We seem to have a large potential for improving!

Portability



- Several directions being tried out
  - Portability libraries: Alpaka, OneAPI, …
  - Redefining CUDA keywords + Allen-like framework kernel launch
  - Macro-based decoration (à la VecGeom) + template specialization of the launch API
- Target: maintaining a single code base running on CPU/GPU
  - Common algorithmic part, possibly decorated with macros
  - Getting same/compatible results on CPU/GPU
- Started by trying Alpaka and OneAPI for some of AdePT examples

OneAPI port

- Development done in a separate repository to minimize interference
- Main goals: understand portability effort to oneAPI in the AdePT context
  - Single source on multiple devices for the main demonstrator example
  - Study performance aspects CPU/GPU, oneAPI GPU/CUDA
- Current blockers: incorporating dependencies on VecCore/VecGeom/G4HepEm
  - Use of virtual functions in navigation
    - Addressed by adding a non-virtual dispatch layer in AdePT
  - User RNG wrapped in G4HepEm
    - Calls via function pointer not supported by OpenCL, so not provided by SYCL

Outlook

- The AdePT R&D is ramping up fast, a first version of physics has been validated on GPU against Geant4 for a simple calorimeter example
  - Active development on several directions
    - physics (adding MSC), geometry (de-virtualization, navigation acceleration), magnetic field (adding a Runge-Kutta integrator), data layout, portability
- Performance optimisation started already
  - Detailed profiling to understand the current baseline examples
  - Testing different workflow scenarios and adding new features (e.g. data management)
- A tight schedule aiming to deliver a realistic prototype in September, followed by a community discussion very early 2022