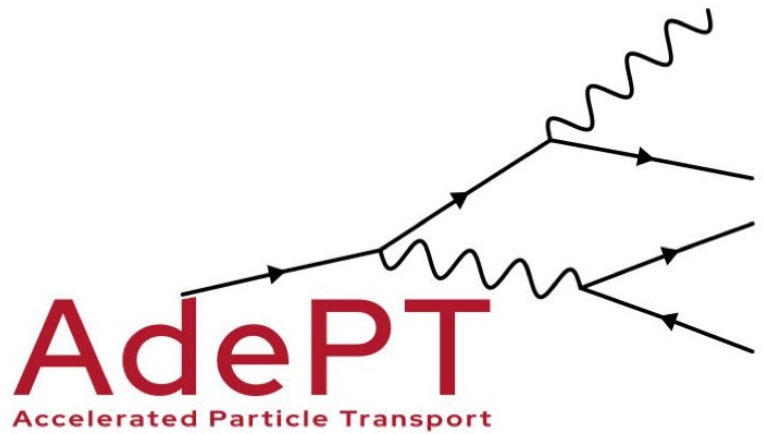


AdePT status report

Witek Pokorski for the AdePT team

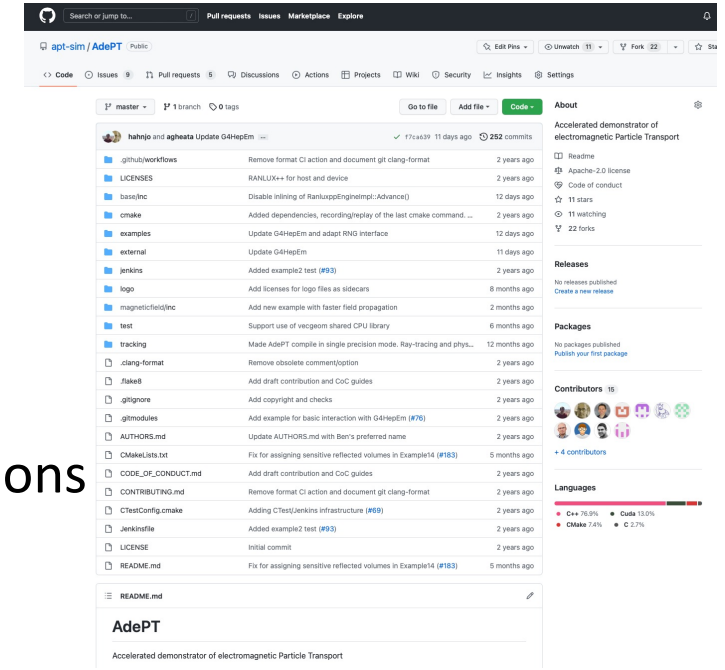
26.09.2022

27th Geant4 Collaboration Meeting

Project targets

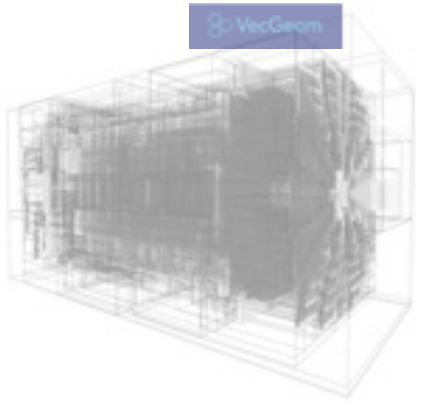
- Understand usability of GPUs for general particle transport simulation
  - Prototype e+, e− and γ EM shower simulation on GPU, evolve to realistic use-cases
- Provide GPU-friendly simulation components
  - Physics, geometry, field, but also data model and workflow
- Ensure correctness and reproducibility
  - Validate the prototype against Geant4 equivalent, ensure reproducible results in all modes
- Integrate in a hybrid CPU-GPU Geant4 workflow
  - Understand possible limitation in such an environment
- Understand bottlenecks and blockers limiting performance
  - Estimate feasibility and effort for efficient GPU simulation

Development approach



- Strategy: integrate gradually features as new examples
  - No library build, maximize flexibility to explore different directions
- Build-up gradually common functionality (services)
  - Infrastructure: custom containers and helpers
  - Geometry: VecGeom library, adapting & developing for GPU
  - Physics: G4HepEm library, a GPU-friendly port of Geant4 EM interactions
- Portability aspects not a major priority in this project phase
  - Initial study identified VecGeom as blocking issue
- Demonstrate usability in native Geant4 workflows
  - Early integration to allow then optimizing a hybrid CPU-GPU workflow
- Git repository
  - Initial commit in Sep 2020, $\mathcal{O}(10)$ contributors

GPU geometry:VecGeom



- Built on top of the original VecGeom GPU/CUDA support
  - C++ types re-compiled using nvcc in a separate namespace/library
  - In AdePT we wrote a custom global navigation layer calling lower level VecGeom APIs
- Improving gradually GPU support
  - Developed custom optimised navigation state, single-precision support
  - Moved from a simple “loop” navigator to an optimized BVH navigator
  - Adopting modern CMake GPU support
- Specializing the VecGeom GPU navigation support
  - Portable, less complex code
- New ability to read GDML files allowing to run with almost any geometry
- Although being a working first solution, CSG-based approach seems to be a bottleneck on GPU
  - investigating surface-based models (see geometry parallel session)

GPU-friendly rewrite of EM physics

- G4HepEm: compact library of EM processes for HEP
  - Covers the complete physics for e−, e+ and γ particle transport
  - Initialization of physics tables dependent on Geant4, but usage on GPU standalone and lightweight
  - Excellent physics agreement between Geant4 processes and G4HepEm
- Design of library very supportive for heterogeneous simulations
  - Interfaces: standalone functions without global state
  - Data: physics tables and other data structures copied to GPUs
  - Reusing > 95% of the code from G4HepEm for GPU shower simulation

GPU workflow

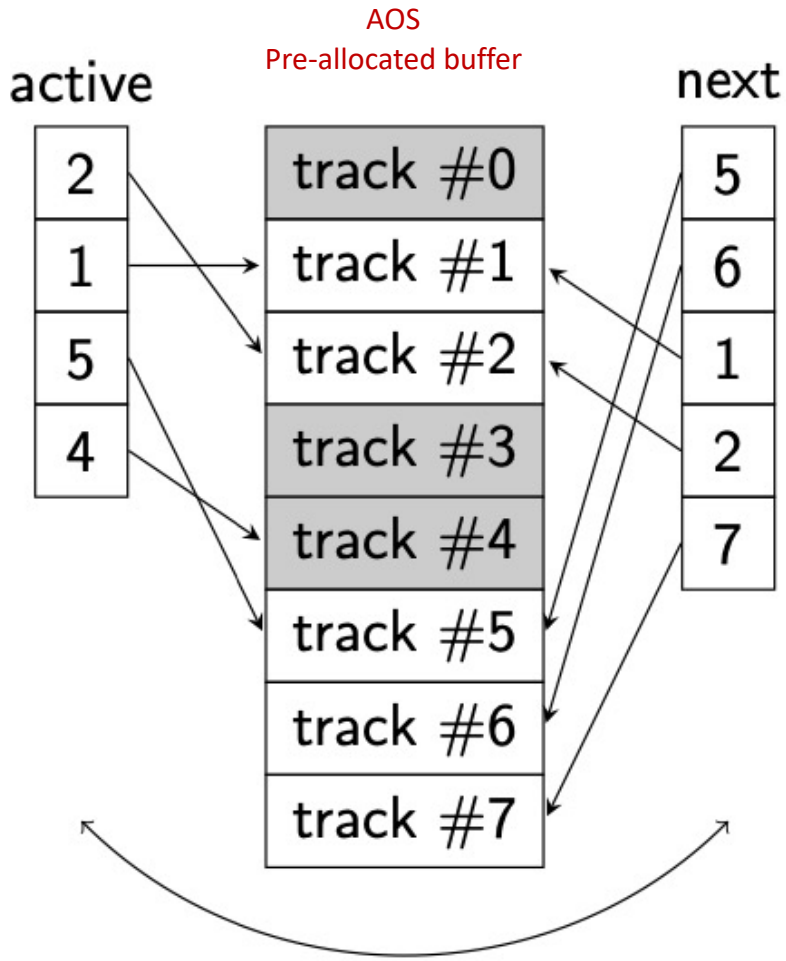
- The GPU workflow has probably the largest impact on performance
  - Very different on GPU compared to CPU, it has to be massively parallelizable
    - AdePT ‘steps’ all active tracks at once
- Different properties to the simulation workflow of Geant4
  - No “thread-local” state, everything associated with a track
  - At the same time: track must be as lightweight as possible
  - Data structures must not create bottlenecks (prefer atomics)
- In AdePT we adopted so far an approach based on active track slots queues scheduled for per-particle kernels (see following slides)
    - A “per-event” approach so far, easier to integrate in realistic simulation workflows

Track storage

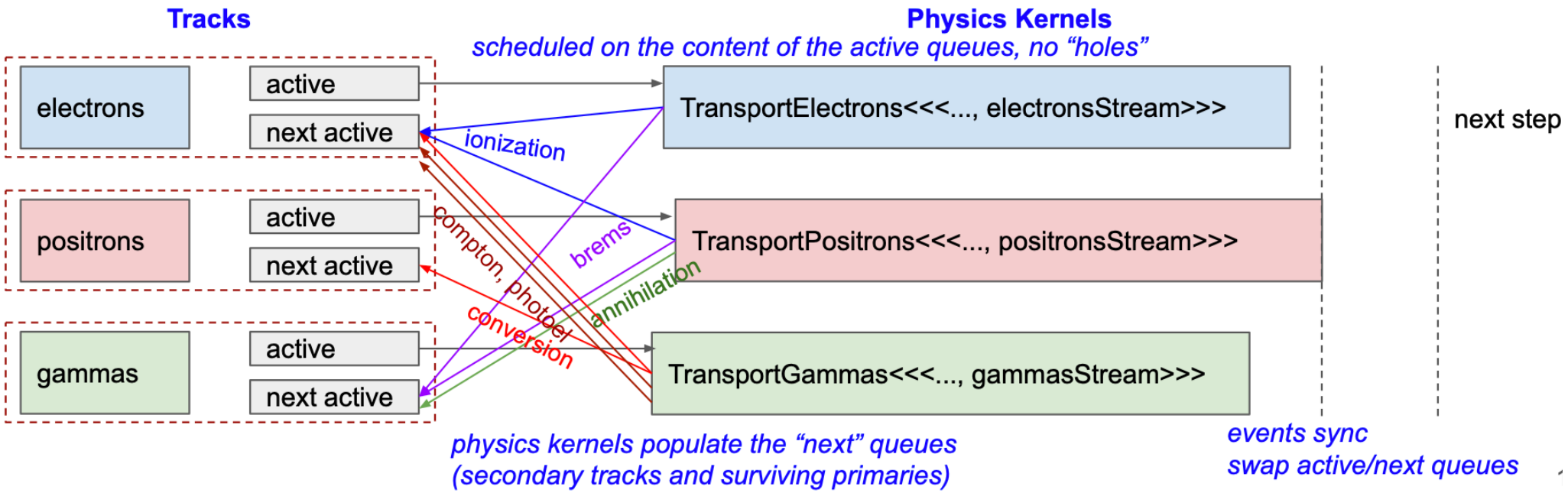
- Properties stored per track:
  - Random number generator state
  - Kinetic energy
  - Position, direction, and current navigation state (volume)
  - State to be preserved across steps (number-of-interaction-left, MSC properties)
- Pre-allocate arrays of tracks per particle type (array of structures)
  - One for electrons, one for positrons, one for gammas
  - Advantage: can call specialized kernels, potentially specialize stored properties
  - Atomic counter to hand out “slots” (to allow compaction)
- Properties *not* stored per track:
  - Particle type / PDG number (implicit from array)
  - Charge, mass (can be inferred from particle type)

Arrays of active and next tracks

- Store indices of active tracks (per particle type)
  - Parallelize transportation kernels over these indices
- Queue indices for “next” active tracks
  - Both secondaries and “surviving” tracks
  - Implemented with atomic counter
  - Tracks are killed by not enqueuing
- Run transportation kernels stepping the active tracks
  - Here track #1, #2 and #5 survive, track #4 dies, and track #6 and #7 are produced
- Swap ‘active’ with ‘next’ before next iteration
  - Compacting unused slots now possible



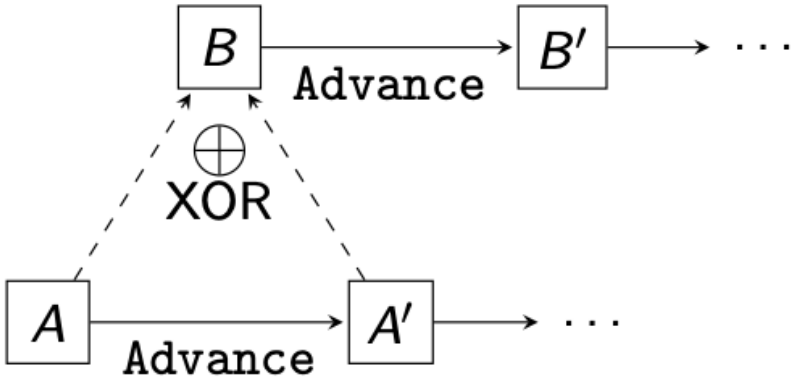
Stepping workflow – first approach



- Can start kernels for particle types in parallel streams (transport is independent)
- Synchronization means overhead
  - Synchronize with host once at the end of the step (stepping loop control)
- Main optimization playground
  - Better work balancing between warps, reducing impact of tails, better device occupancy
  - Experimenting with smaller kernels (separating discrete and continuous interactions)

Random number handling

- To assure reproducibility, RNG state needs to be associated with each track
  - Guarantees identical results no matter the parallel execution order and kernel configuration
  - Essential for debugging during development and production
- Need to initialize new RNG state for secondary particles
  - Must only depend on parent track to guarantee reproducibility
  - Can re-use RNG state of dying track in annihilation or conversion



Input: state A
Output: states A' and B'

1. Advance to state A'
2. XOR bits in A and A' to get B
3. Advance state B to break correlations

Random number generator: RANLUX++

- Based on the well-known RANLUX generator
  - Uses the equivalent LCG and therefore faster
  - Excellent statistical properties: inherited from RANLUX, only shared by MIXMAX
    - (XORWOW used by default in cuRAND known to fail some statistical tests)
- Portable implementation available, written with GPUs in mind
  - See J. Hahnfeld, L. Moneta: A Portable Implementation of RANLUX++
- Advantage over MIXMAX: smaller state
  - Even for N = 17, the default generator in Geant4 (148 bytes of state)
  - Compared to 80 bytes for RANLUX++

Magnetic Field: Runge-Kutta field propagation

- Propagation using Dormand-Prince 4/5th order RK method
- Field Propagator templated on field type, driver
- RK Driver templated on stepper, equation of motion

First results with TestEm3, Bz= 3.8T

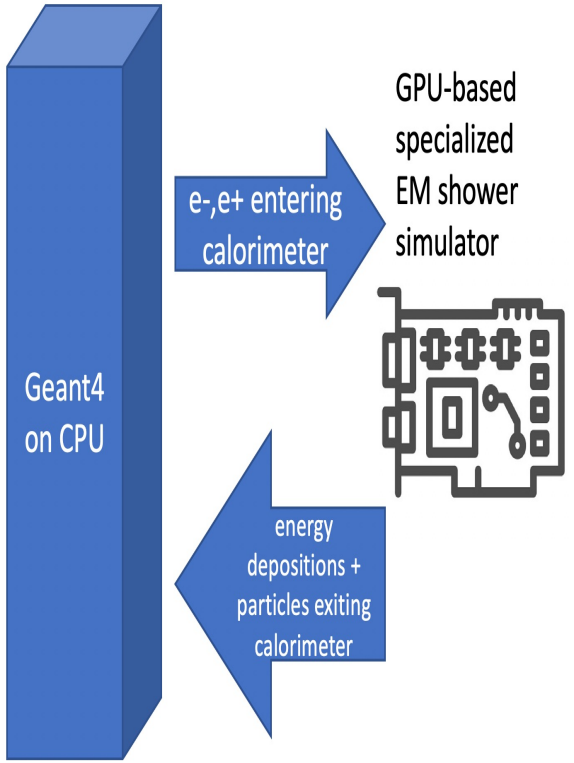
- Better than per-mille agreement in observables with helix results
- Improved handling of particles ‘stuck’ at boundaries - flip volume at boundary
- Runtime about 1.5x helix (98 s vs 65 s in TestEm3 3.8T test case)

Next steps - further testing and performance evaluation

- Use semi-realistic field (simplified CMS or ACTS ‘texture’-based interpolation)

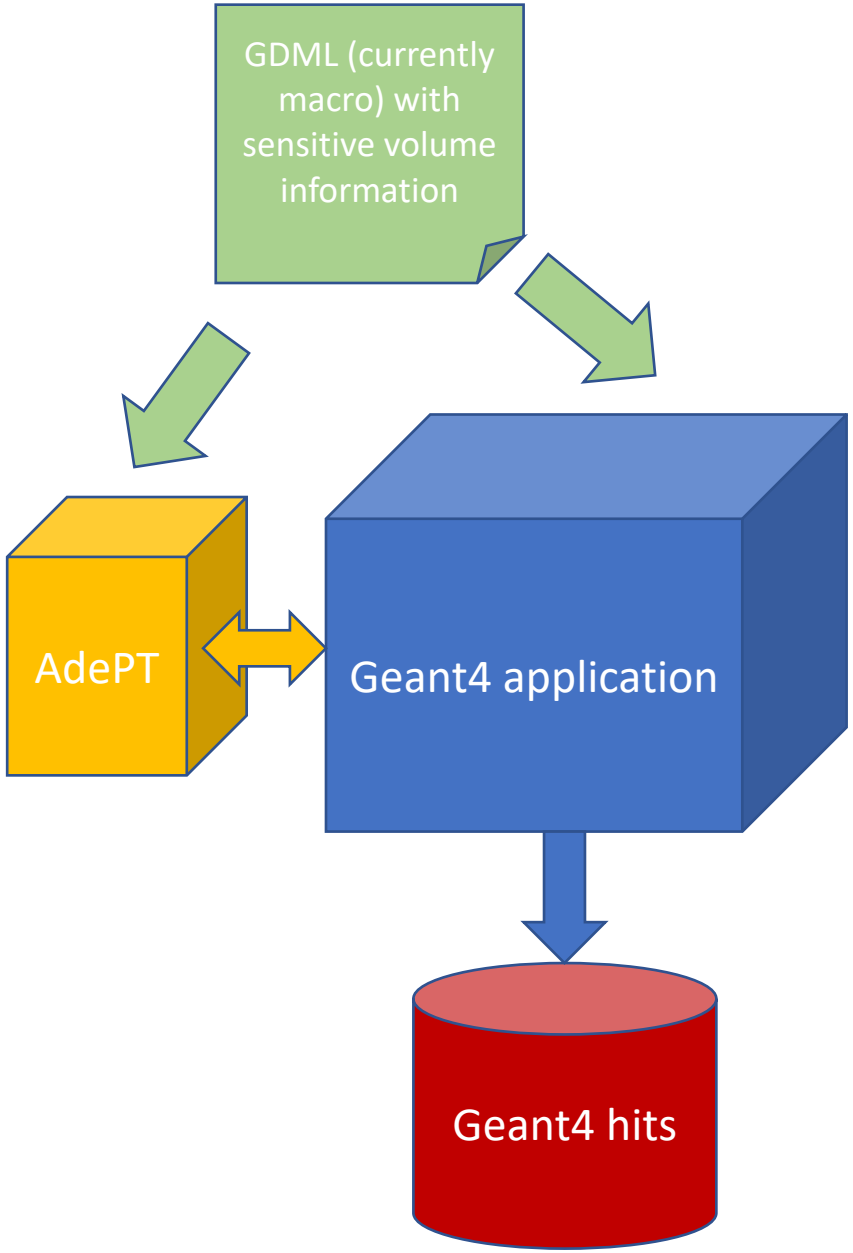
Prototype integration strategy

- region-based approach for delegating simulation to an external transport
  - particle killed on the Geant4 side and passed to the other transport engine
  - energy depositions and 'outgoing' (from that region) particles returned
- this follows 'fast-simulation' approach in Geant4
  - allows the use of (most of the) existing fast-simulation hooks
  - easy integration with the physics list
  - ability to switch between full Geant4 and Geant4 + AdePT at runtime (from macro file)
- one difference:
  - we buffer particles to process them together when some threshold is reached (or when there are no more Geant4 particles on the stack)



Scoring

- sensitive volumes marked on the GPU with a flag while initializing geometry
  - list of sensitive volumes provided in Geant4 macro file or read from GDML auxiliary information (not fully implemented yet)
- energy deposition per volume (per event) recorded by AdePT in sensitive volumes
  - other types of ‘hits’ can be implement by user
- array of energy depositions per volume is transfered to the host once the AdePT ‘shower’ finished
  - indices of volumes on GPU mapped to the Geant4 ones
- SensitiveDetector::ProcessHit method (overloaded) called to translate this array of energy deposition into Geant4 hits
  - the output looks the same regardless running full Geant4 or Geant4 + AdePT
    - can be then processed/analysed in the same way



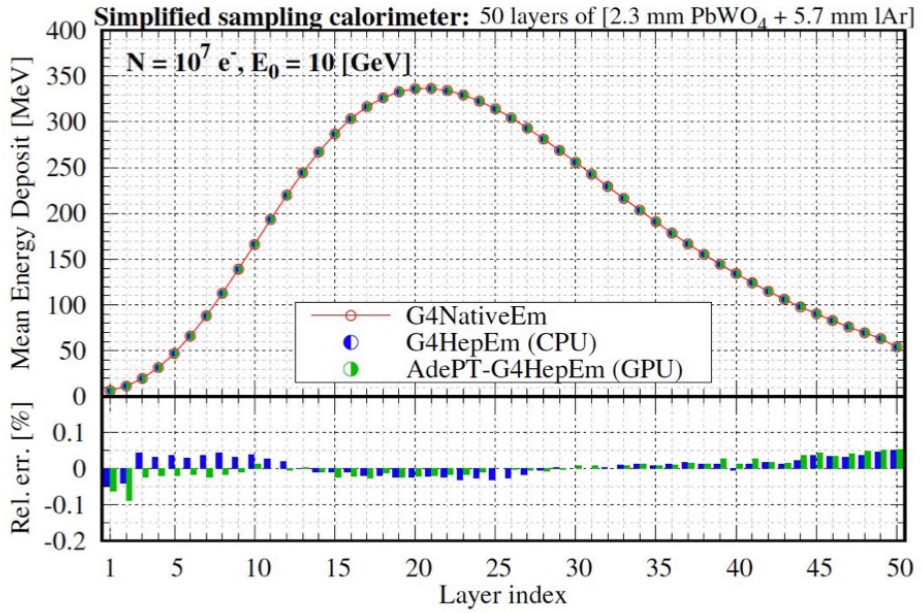
‘Outgoing’ particles

- if a particle leaves AdePT region (or can’t be handled by AdePT physics) it is put in the ‘from device’ buffer
- after AdePT shower has finished, ‘from device’ buffer is transformed in Geant4 tracks and put on the Geant4 stack
  - Geant4 continues the event loop to process those particles
    - to guarantee reproducibility (also in MT) particles ‘from device’ are sorted according to some unique key
- event finishes when no more particles are in the AdePT buffer (‘to device’) and Geant4 stacks are empty

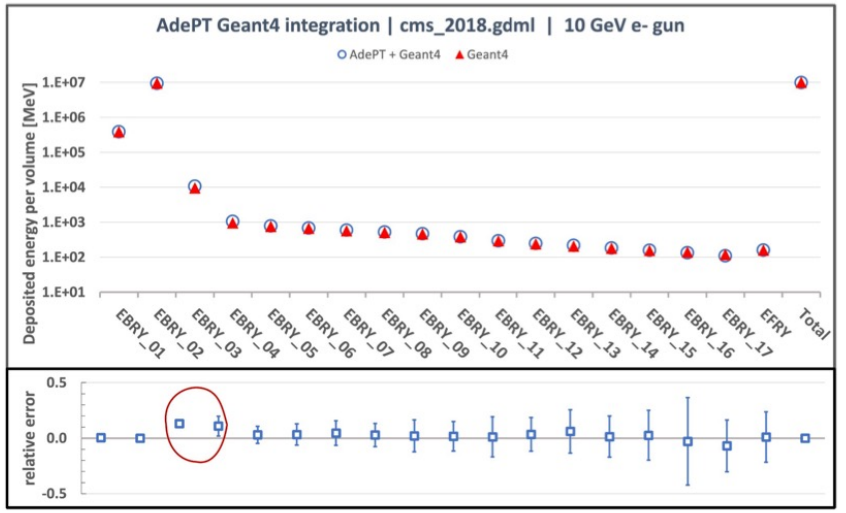
Validation

- Validation against Geant4 standalone is essential
  - Comparisons to CPU references (in general Geant4-based) done for each added item of functionality
  - Both for standalone and Geant4 integration examples
- EM physics now fully validated
  - At ‰ level in the sampling calorimeter test case
  - Still working on the last bugs/features in the hybrid workflow

Sampling calorimeter example

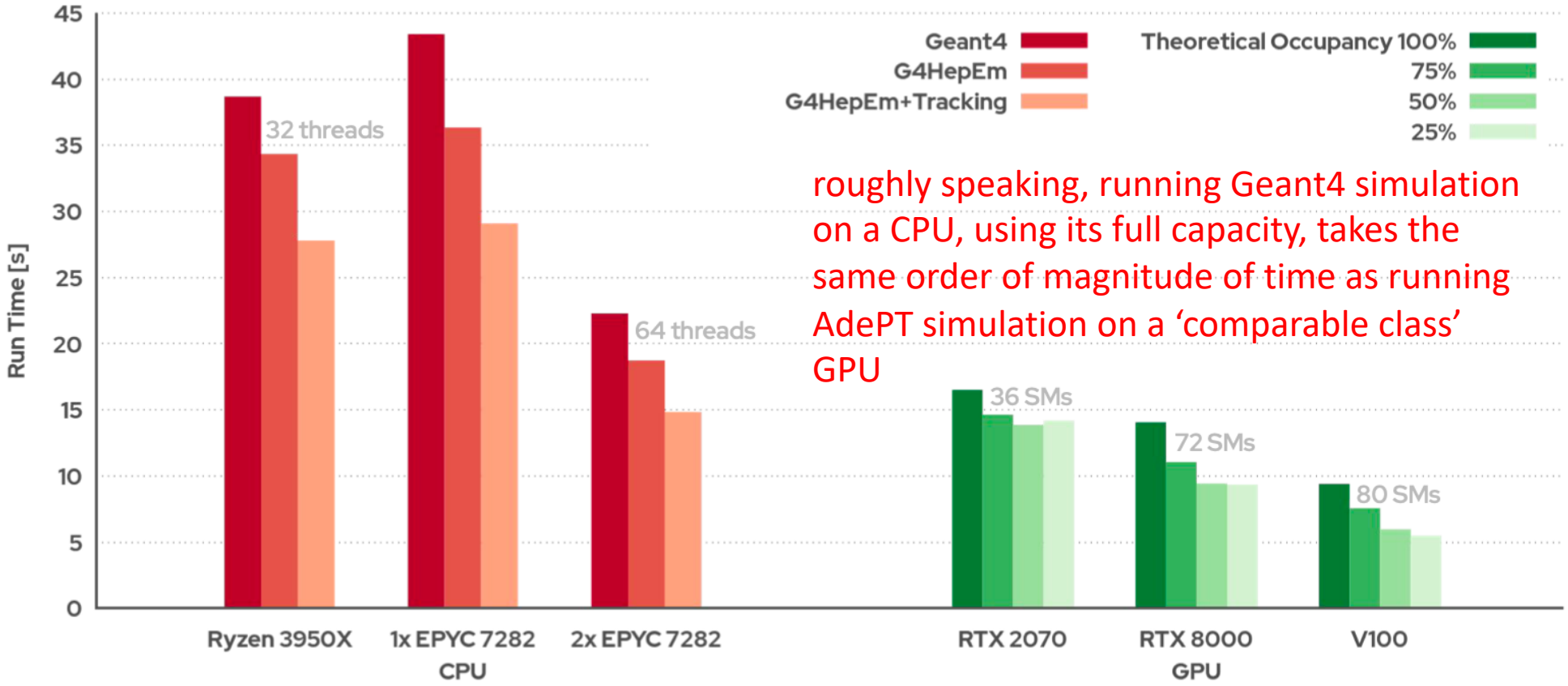


AdePT integration with Geant4



CPU vs GPU Performance

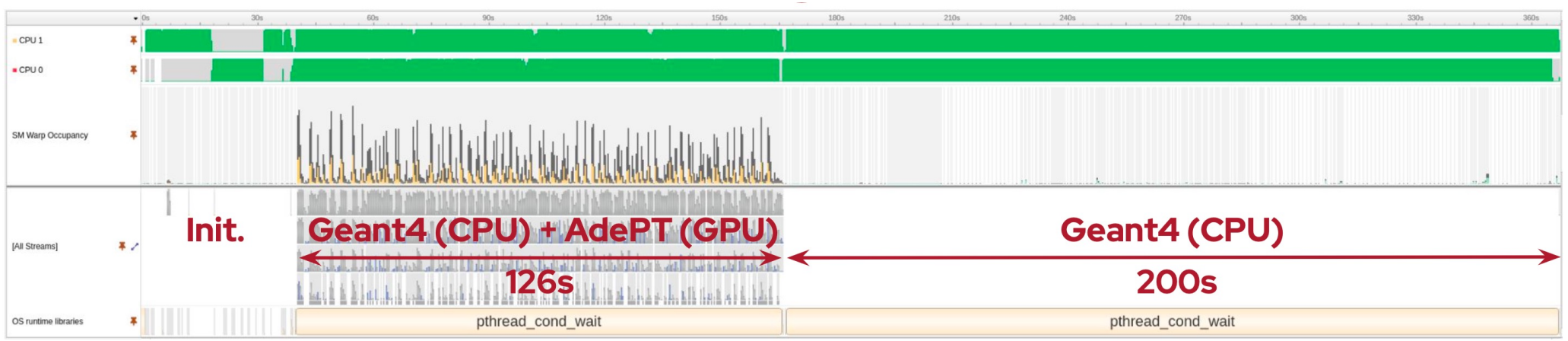
(Sampling Calorimeter example)



roughly speaking, running Geant4 simulation on a CPU, using its full capacity, takes the same order of magnitude of time as running AdePT simulation on a 'comparable class' GPU

AMD Ryzen 3950X (16 cores, 32 threads, 3.5-4.7GHz), AMD EPYC 7282 (16 cores, 32 threads, 2.8-3.2GHz)

CMS simulations: integrated and standalone

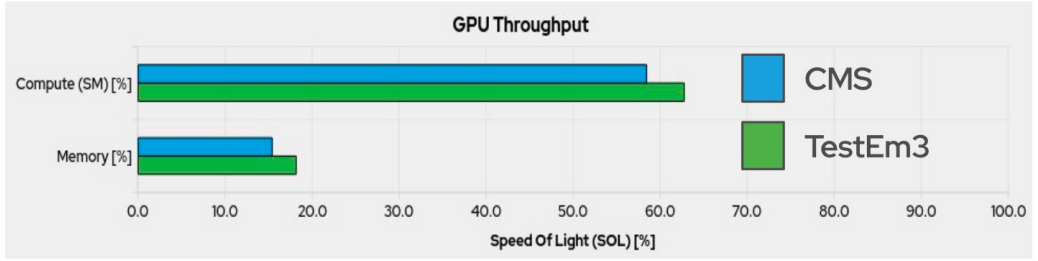


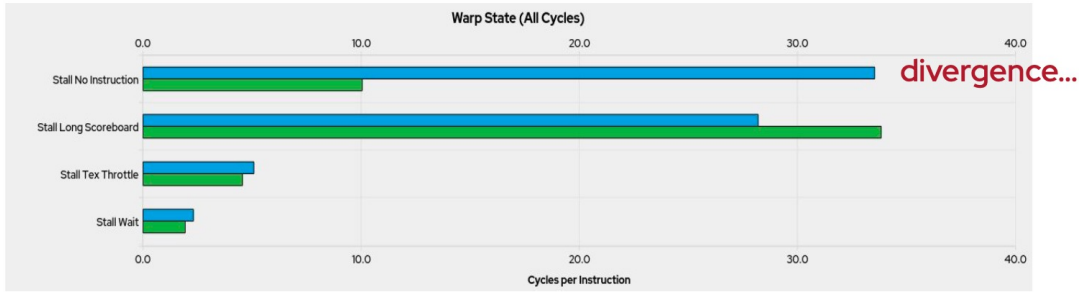
Comparison to Geant4

Above is the timeline of CMS simulation comparing AdePT integrated into Geant4 to Geant4 (Ryzen 3950X, RTX2070), with a speedup of 37% when using 2 CPU threads + 1 GPU vs only 2 CPU threads.

Impact of detector geometry

On the right, 10^6 electrons at 10GeV on Nvidia Tesla V100 with TestEm3 geometry vs the CMS geometry. The total simulation run time for the simplified calorimeter (TestEm3) setup is 549s vs 1455s for the CMS geometry





Thread Divergence on GPU

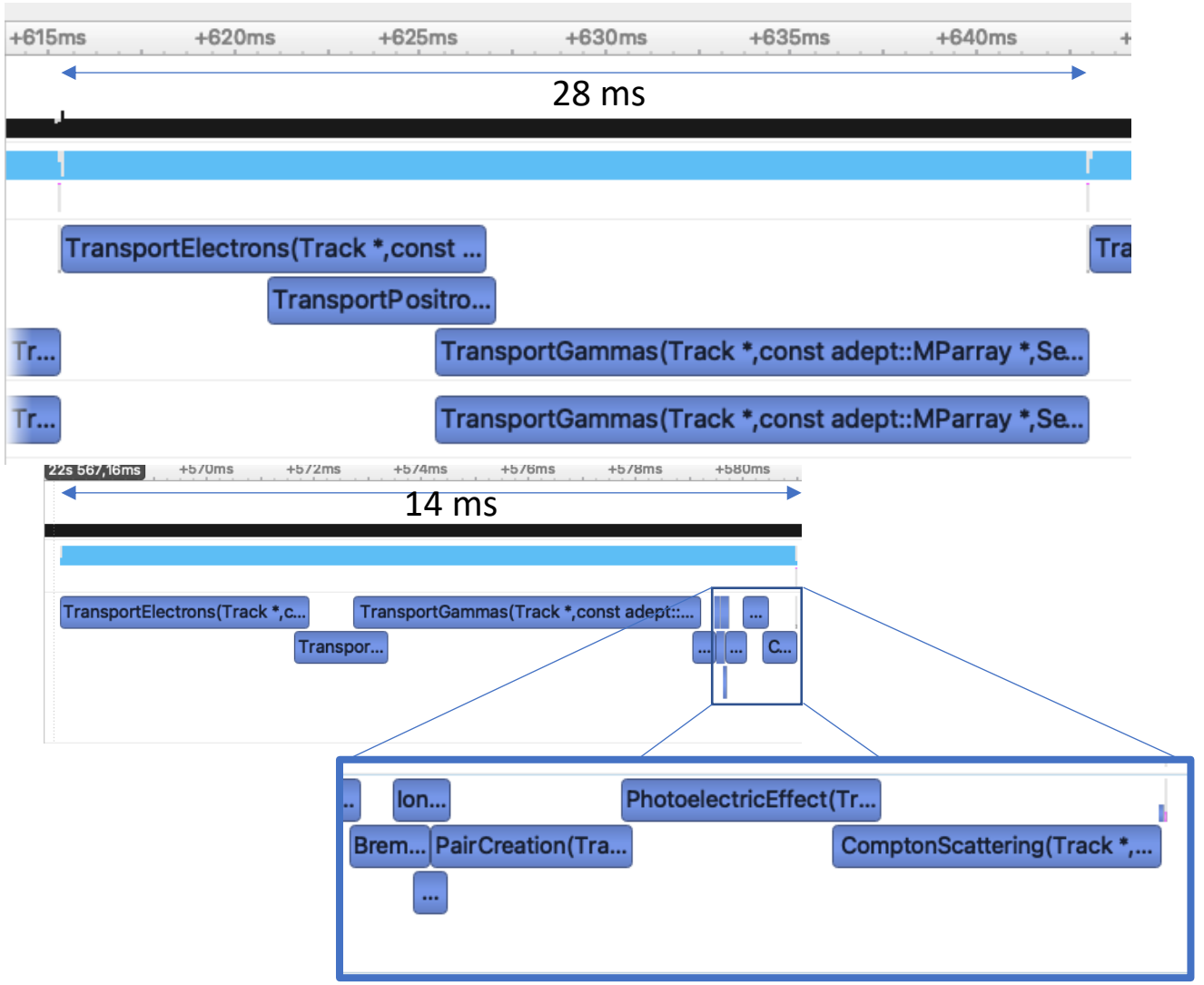
Problem:

- Threads in transport kernels diverge over time; the longer the kernel runs the more this happens
- Navigation and interactions branch frequently
- **5.5 / 32 threads active** on average (Example 18, 10k primaries, CMS 2018 geo)
- GPU underutilised, even though all queues are full

AdePT Example 19:

- Split off interaction computations from cross-section and geometry kernels
- Test on Tesla V100, 10k primaries, CMS geo
- **4.5 / 32** threads active for TransportGammas (= physics + navigation)
- **{6.4,12.8,20.8} / 32** threads active for {pair creation, photoelectric effect, Compton scattering}
- **Run time:** 23.5 s → 15.4 s

Conclusion: Strive for thread coherence where possible



Ongoing and future work

- geometry: trying out alternatives to reduce code complexity and divergence -> surface models
- magnetic field: transforming workflows to balance the work and reduce the tails better
- scheduling and kernels refactoring and simplification for smaller register footprint and better work balancing
- understanding better and optimizing the G4 integration workflow, now penalized by too many CPU-GPU exchanges
- aiming for fully GPU-confined simulation, learning how to deal with GPU-produced tracker hits for example

Common points to both AdePT and Celeritas projects – looking forward to establish closer collaboration on those.

Summary

- challenging project, simulation clearly not an easy problem for GPUs
  - need to recast the diverse physics interactions and geometry elements for the regularity of the GPU
- AdePT GPU prototype provides full EM physics and geometry support to run simulations of CMS calorimeter complexity in standalone and Geant4 integrated modes
  - encouraging and motivating first result!
- current geometry model is a big bottleneck, we are addressing it with development of a new surface-based model
- further work on integration with experiments software frameworks will allow to better understand other potential stumbling blocks
- we invite collaborators to join this R&D !