AdePT status and plans

Accelerated demonstrator of electromagnetic Particle Transport

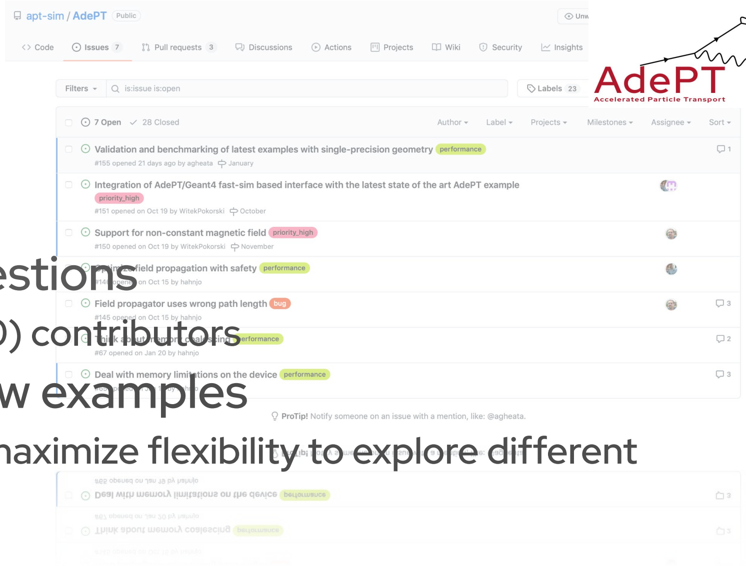
Andrei Gheata for the AdePT Developers

26th INTERNATIONAL CONFERENCE ON COMPUTING IN HIGH ENERGY & NUCLEAR PHYSICS (CHEP2023) - Norfolk, May 8-12, 2023

Simulation on GPU - can we do that?

- **Functionality**: make all simulation components work on GPU
  - Physics, geometry, field, but also user sensitive detector code and hits?
  - Simulate e^+, e^- and γ electromagnetic shower
- **Correctness**: validate results and ensure reproducibility
  - Producing compatible results with Geant4 equivalent?
- **Usability**: integrate in a hybrid CPU-GPU Geant4 workflow
  - Usable within real experiment frameworks?
- **Performance**: understand/address bottlenecks limiting performance
  - Can we actually use the GPU in an efficient/beneficial way?
  - Show stoppers? Bottlenecks? Can we overcome them?

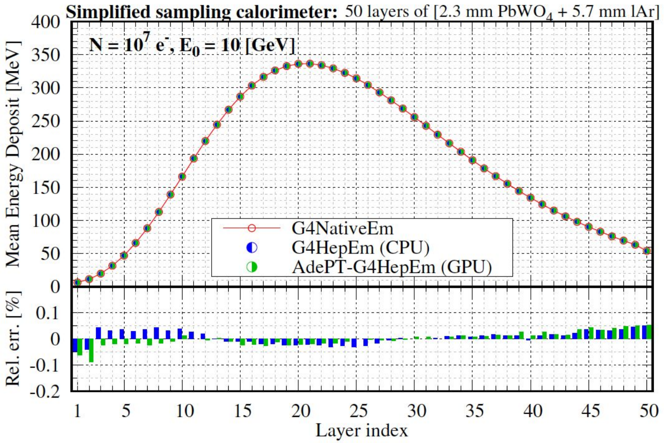
The AdePT project



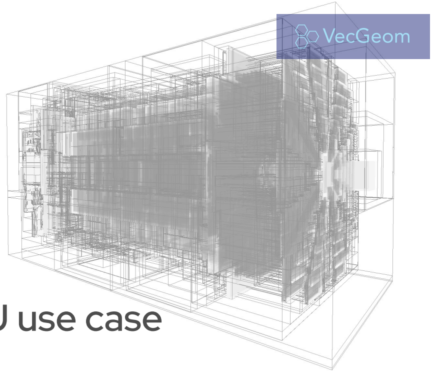
- Trying to get answers for most of these questions
  - GitHub repository, initial commit in Sep 2020, $\mathcal{O}(10)$ contributors
- Strategy: integrate gradually features as new examples
  - No library/framework, just core infrastructure, to maximize flexibility to explore different directions and adapt to different requirements
- Minimal external dependencies
  - Geometry: ***VecGeom*** library, enhancing its GPU-related features
  - Physics: ***G4HepEm*** library, a GPU-friendly port of Geant4 EM interactions
- Understand and improve the integration with experiments and their frameworks
  - Discussions/collaboration with ATLAS, CMS and LHCb ongoing

At a glance: physics

- **G4HepEm**: GPU-friendly compact rewrite of EM processes for HEP
  - Covers the complete physics for e⁻, e⁺ and γ particle transport
- Design of library very supportive for heterogeneous simulations
  - Stateless interfaces working on both CPU and GPU
  - Data: physics tables and other data structures relying on Geant4, but standalone after being copied to GPU
- Verified against Geant4 standalone
  - At ‰ level in the sampling calorimeter test case



At a glance: geometry



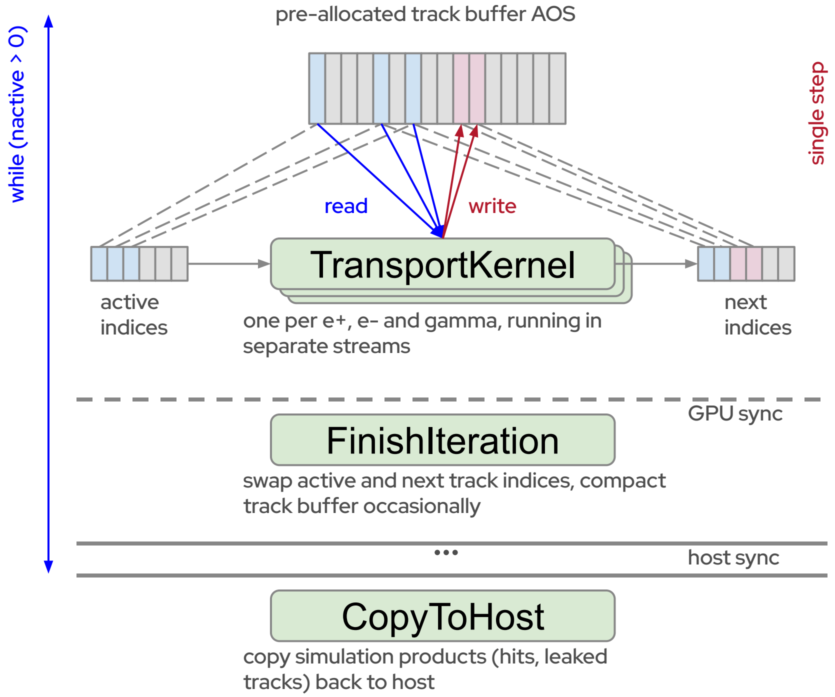
- Relying on the builtin **VecGeom** CUDA support
  - Identical object model for CPU and GPU, non-specialized for the GPU use case
  - CUDA-specific, non-portable
- Improved gradually the GPU support
  - Developed index-based navigation state handling, single-precision support, faster GPU init
  - Moving from a simple non-optimized to a more efficient **BVH navigator**
  - Adopting modern CMake GPU support
- The current geometry approach is a major GPU bottleneck
  - Strong motivation to develop a surface model for GPU support
    - Portable less complex & less divergent code, creating a surface-based view on device
    - Our major work item (see: surface model presentation link)

Parallelization in AdePT

- Simulation is done in steps, moving particles to either boundaries or physics processes
- All active tracks available are stepped at once (Geant4 transports one particle at a time)
  - Much higher degree of parallelism and more uniform work for the GPU
- No “thread-local” state, everything embedded in the track
  - Energy, position/direction, state needed across steps
  - Random number generator state (RANLUX++) per track to ensure reproducibility
    - Strategy to spawn a new sequence for daughter particles from the current state
- Tracks pre-allocated per particle type in thread-safe containers
  - Atomic counter to hand on track slots to be filled by kernels

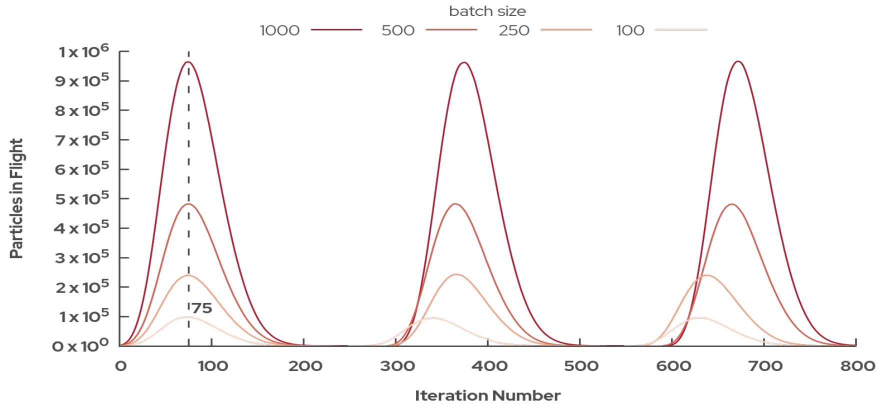
Stepping loop

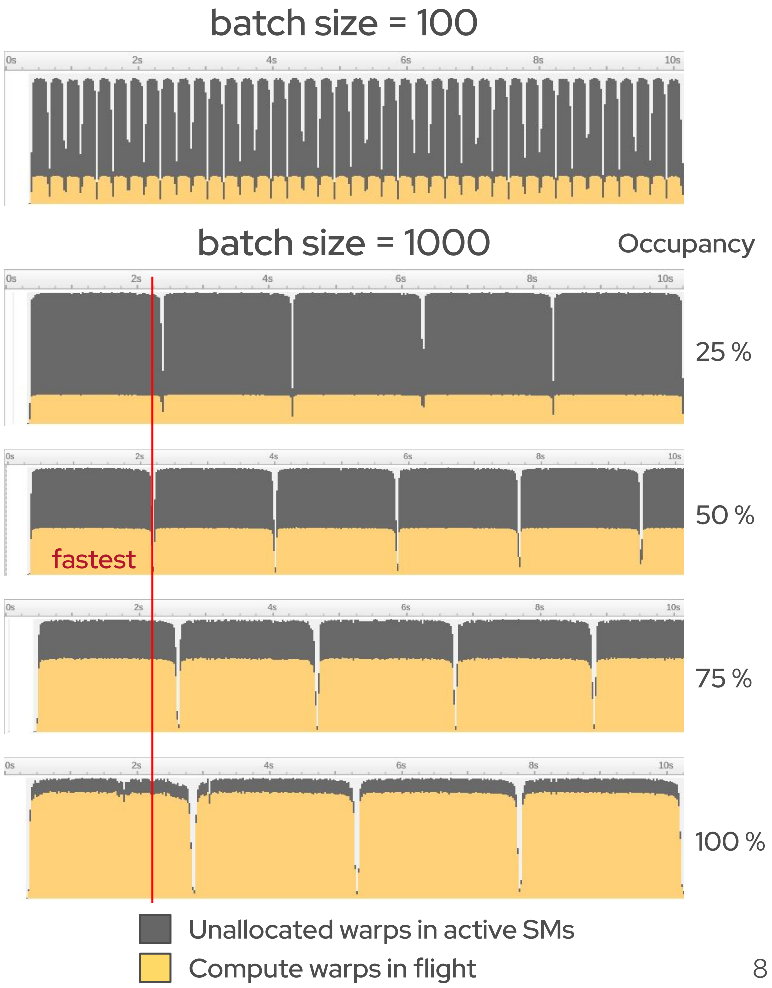
- Pre-allocated track buffer
- Separate kernels per particle type
  - Separate kernels for continuous and different discrete processes also possible
- Double buffer of active/next track indices
  - Atomic access, next-unused slot track allocation policy
  - Dead tracks leave holes, track container compacted occasionally
- Copy to host simulation products at the end (hits, leaked tracks)



Run Time Characteristics

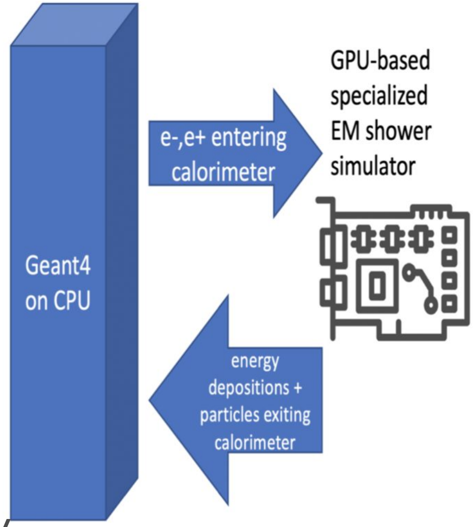
- putting more work per batch does more work in the same #iterations (steps)
  - limited by available memory AND available tracks
- hints already to using strategies to fill the gaps
  - e.g. more CPU threads doing concurrent events
- performance: sweet spot at about 50% occupancy (register-hungry code)
- 36 SM GPU ≃64 CPU threads: a consumer card can double the throughput of a dual socket machine

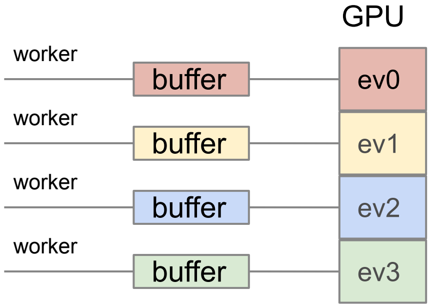




AdePT-Geant4 integration

- AdePT only provides EM physics for e^+, e^- and γ
  - Cannot be used standalone for simulating a full experiment
  - In a first phase it could be used as accelerator for the EM part, in the same way as fast simulation models can be used in Geant4
- Developed an integration interface allowing a Geant4 region to become the "GPU region"
  - Intercepting and buffering for GPU particles sent asynchronously by Geant4 threads
    - Available from Geant4 11.1, patches available for older versions
  - Sensitive detector code run on device, hits+leaked tracks sent back to host
  - An initial approach under evaluation by several experiments

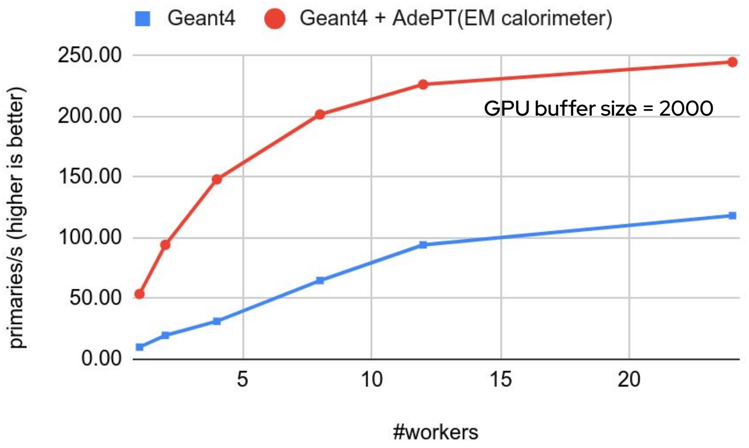




Integration performance

- Performance in this approach increases with :
  - Fraction of time spent in the GPU-accelerated region (Amdahl law)
  - GPU buffer size and event size (to fill it)
- Performance degrades with :
  - Number of exchanges CPU-GPU per event
  - Number of CPU threads (GPU saturation)
- Why not the full detector on GPU?
  - Not limited by geometry
  - Possible for EM particles
    - Except lepto-nuclear processes (rare) that can be delegated to CPU
  - Limited by sensitive detector code GPU awareness - incentive to write GPU-friendly scoring

Throughput for batches of 100 x 10 GeV e^-/event gun, 85% of simulation time in the EM calorimeter



cms_2018 setup, Xeon(R) CPU E5-2630v3 + RTX2070

Hooking user code

- AdePT advanced examples provide a mechanism to implement Geant4-like sensitive detector code
  - Scoring type to be implemented and aliased as *AdeptScoring*
  - Transport kernels templated on this type, calling back directly on GPU
- Fairly straightforward interfaces
  - GPU data management (hits) – allocation and cleanup, copy to host
    - A very simple atomic calorimeter cell accumulator as example
  - *AdeptScoring::Score* method to intercept current step as in Geant4
- One of the main challenges for experiment code integration
  - Cannot be identical with Geant4 code (different types)
  - Working directly with experiments to understand realistic cases

AdePT

```
electrons.cuh

template <typename Scoring>
__global__ void
TransportElectrons(Scoring *s)
{
…
  s->Score(track_state);
}
```

User code

```
SimpleScoring.h

struct SimpleScoring
{
  __device__ void Score(
           TrackState const&);
  …
};

using AdeptScoring =
SimpleScoring;
```

Experiment integrations

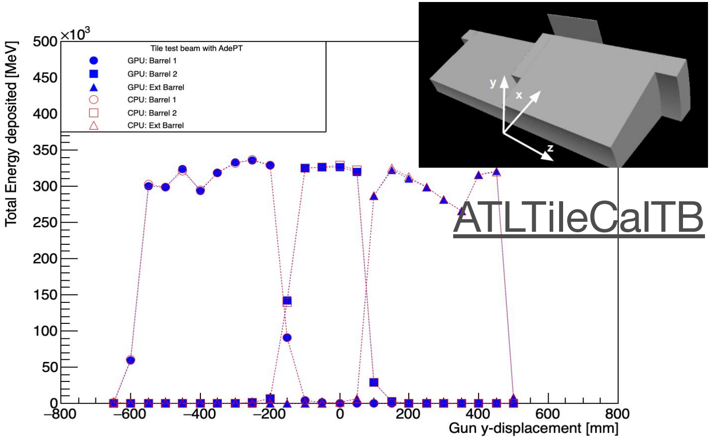
- AdePT is not a framework/library at this stage
  - Compiling the specific experiment integration will compile AdePT
  - This makes easier to fit on specific user scoring requirements
- Interacting with experiments in different phases
  - Understand how AdePT works: modifying advanced examples and adding a custom detector module
  - Understanding which detectors/workflows may benefit from such GPU integration
  - “Biting the bullet” and actually dealing with the concrete case integration problems
    - AdePT dependencies, experiment framework, specific detector scoring code

Towards integration with CMSSW

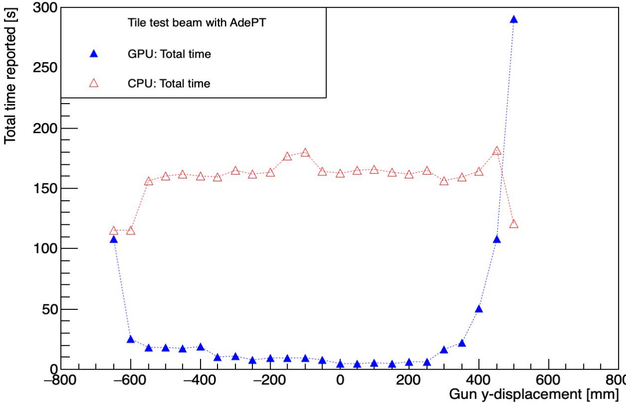
- Targeting Phase 2 setup, in particular CMS HGCal
  - Load geometry setup in AdePT Example14 (exported from CMSSW)
  - Configure HGCalRegion to offload electrons, positrons and gammas
  - Load HepMC3 file with minimum bias events
- Started with integration fo G4HepEm on CPU
  - Library built with CMSSW since November 2022
  - Integrate as option into EMM physics list, only for e- below 100 MeV
- Investigate sensitive detector code on GPUs
  - Right now only accumulated energy deposit
  - Also need to deal with sparsity of HGcal hits (important data volume)
- Prepare prototype for integration with CMSSW
  - How to request GPU resources from multiple threads
  - Ways to extract particles for the GPU, send back results and feed into framework

ATLAS trying out AdePT

- ▶ Forked AdePT & modified example14(17)
  - Taking a test beam setup geometry GDML
    - ▷ Scintillator as active element
  - Modifying BasicScoring.cu
    - ▷ Adapting to specific Geant4 scoring code
  - Scanning with electron gun tilted along Y axis
    - ▷ Getting same results + speedup GPU vs. CPU
  - Main challenge: adapting G4Step-based scoring
- ▶ Take-away & next steps
  - "Ideal environment to build a sensitive detector" (working on GPU)
  - More complex scoring (e.g. Birk's law on device)
  - Code duplication? How to handle?
  - Thinking about integration with *FullSimLight*



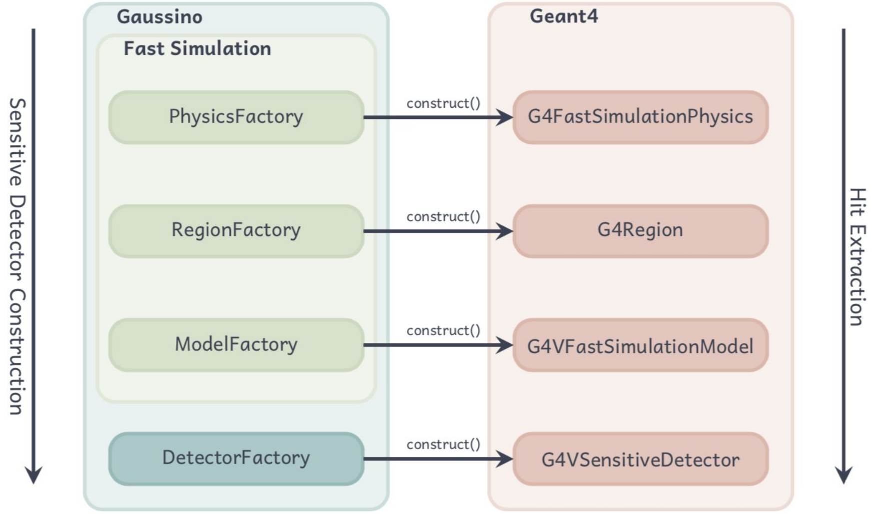
ATLTileCalTB



credits to D. Costanzo, A. Dell'Acqua & R. M. Bianchi

Integration as fast simulation algorithm?

- Different versions of LHCb upgrade geometry usable with AdePT advanced examples
  - Ongoing investigation enabling AdePT with the EMCAL, using MCParticles at entrance taken from realistic simulation
- Ongoing discussions about AdePT integration in Gaussino
  - Via AdePT buffer and FastSim hook (requires patching Geant4)
  - Stopping particles entering the ECAL and giving them to Gaussino calling AdePT as fast simulation algorithm



credits to G.Corti & M. Mazurek

Outlook

- A challenging project, the problem is far from a perfect match for GPU
  - Full EM shower transport functionality implemented and validated
  - A first phase of evaluation completed, answering most of the initial R&D questions
  - Efficiency blockers identified, triggering a new project on GPU geometry
- Prototypes for standalone and Geant4-integrated workflows available
  - Realistic examples for LHC setups, GPUs can be used in a Geant4 native application
  - Optimization work ongoing, performance not yet on a *GPU-efficient* baseline
- GPUs appears to be a valid accelerating alternative for particle transport simulation
  - Still to be validated by integration with concrete experiment use cases
  - Discussions and collaboration with experiments ongoing

Backup

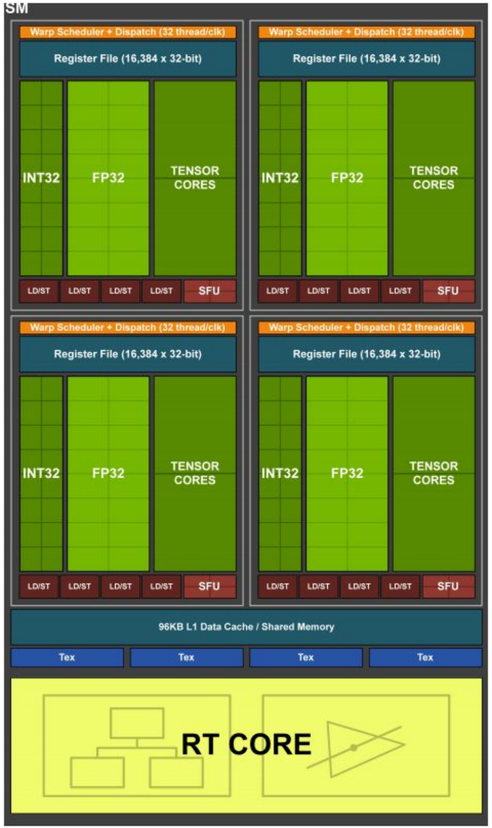
Kernel Launch Configurations

- 1024 Threads / SM
  - 4 schedulers x 8 warps/scheduler x 32 threads/warp
- 65536 Registers / SM
  - 4 register files x 16384 registers
  - 1 float = 1 register, 1 double = 2 registers
- 96 KB L1 Data Cache / Shared Memory
- Theoretical Occupancy (–maxrregcount or __launch_bounds__)
  - 256 regs/thread (256 threads, 8 warps) ⇒ 25%
  - 160 regs/thread (320 threads, 10 warps) ⇒ 38%
  - 128 regs/thread (512 threads, 16 warps) ⇒ 50%
  - 96 regs/thread (640 threads, 20 warps) ⇒ 63%
  - 80 regs/thread (768 threads, 24 warps) ⇒ 75%
  - 64 regs/thread (1024 threads, 32 warps) ⇒ 100%

↓ Higher parallelism

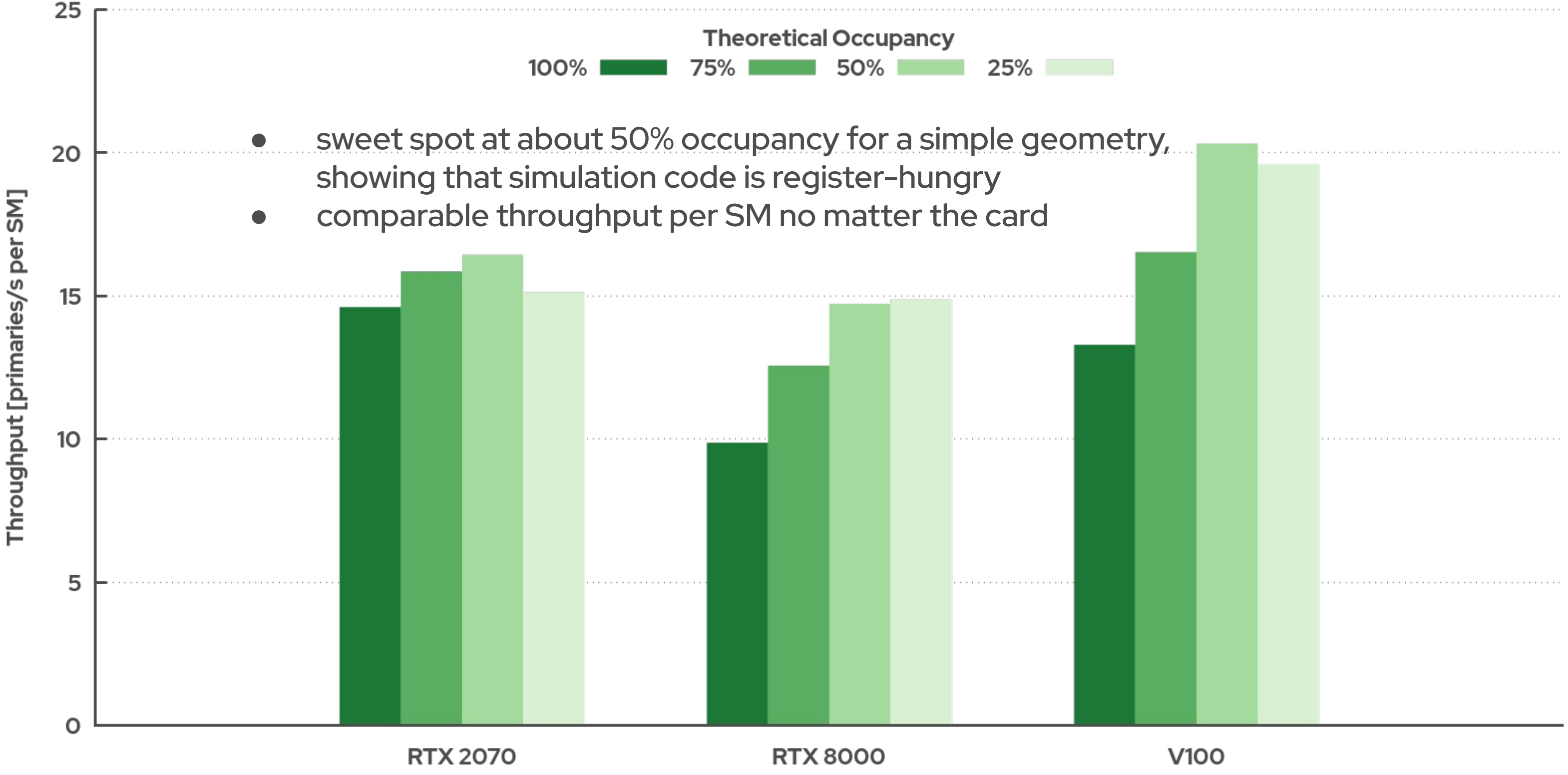
↑ Faster Threads

Turing SM

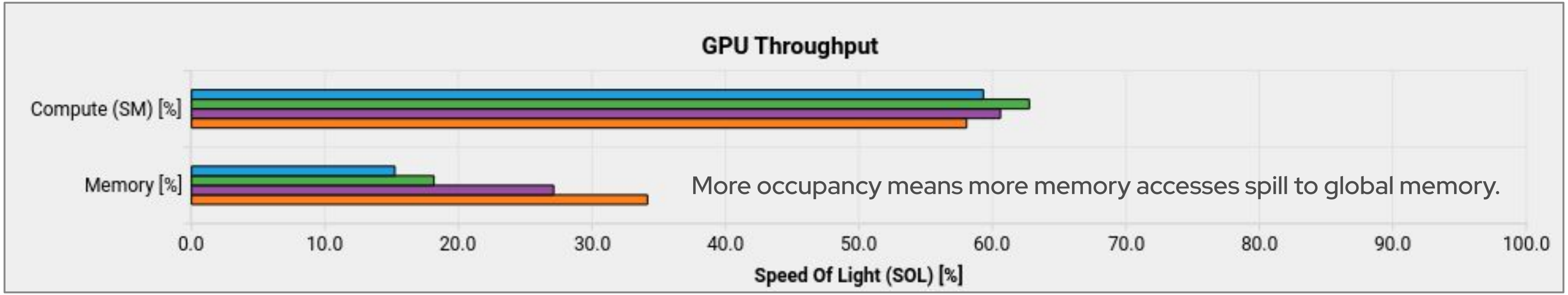


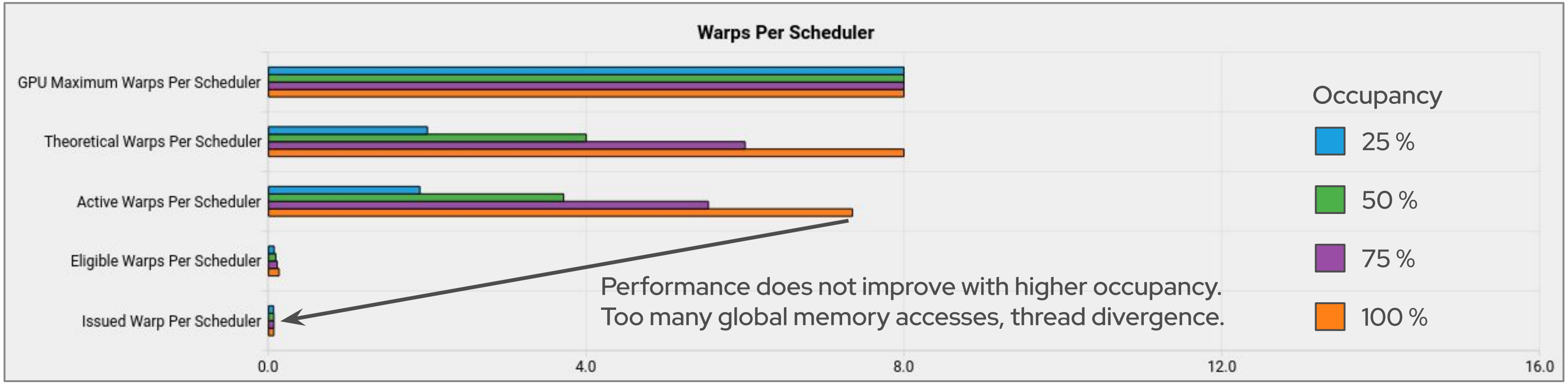
Relative Performance per SM

- sweet spot at about 50% occupancy for a simple geometry, showing that simulation code is register-hungry
- comparable throughput per SM no matter the card

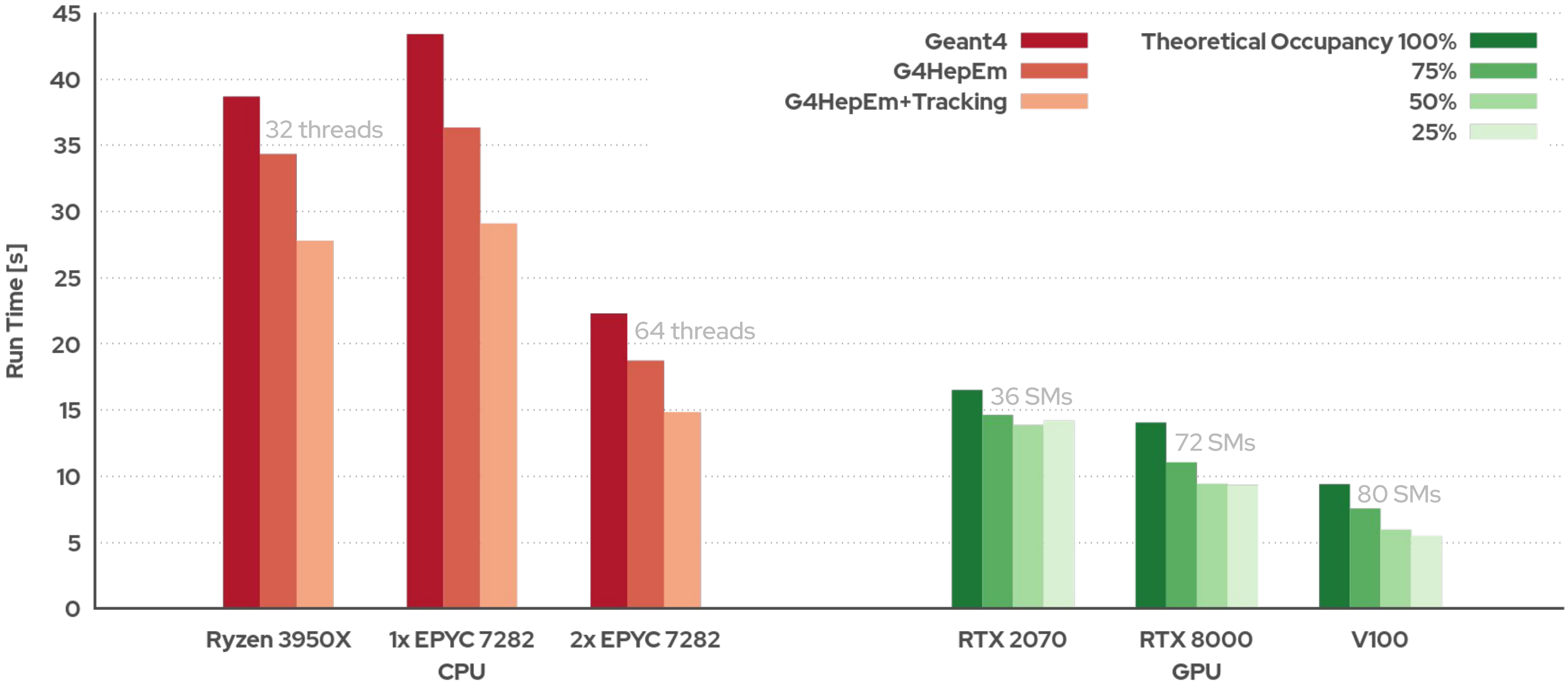


GPU Throughput (RTX 2070)



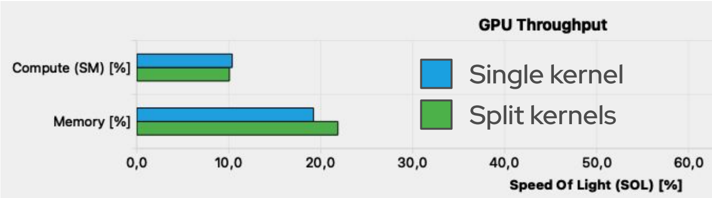


CPU vs GPU Performance



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

Case Study: Thread Divergence



Problem: Threads in transport kernels diverge because of diverging interactions
→ 13 / 32 threads active on average

Here: Split off interaction computations from cross-section and geometry kernels (one kernel for pair creation, one for ionisation, ...)

Result: 17 / 32 threads active for physics + geo
29 / 32 threads active for Bremsstr.
Run time: 6.4 s → 5.5 s

Conclusion: Keeping threads coherent is key for detector simulation
Generally difficult; stochastic processes