AdePT status and plans

Accelerated demonstrator of electromagnetic Particle Transport

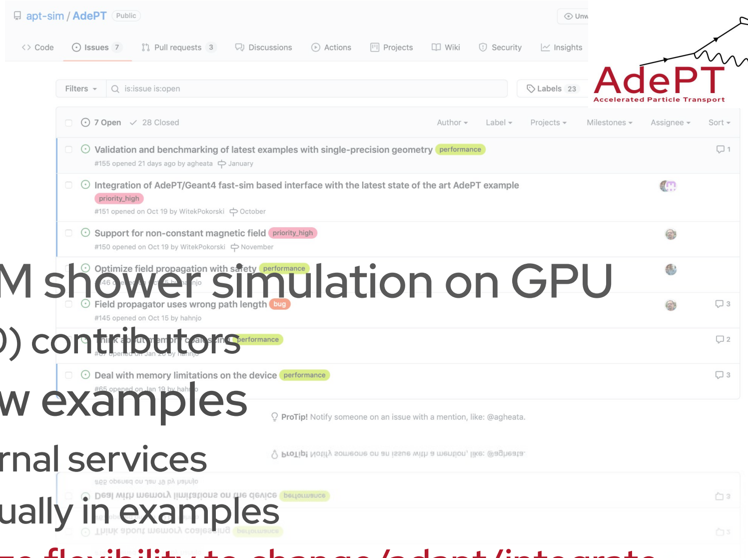
G Amadio, J Apostolakis, P Buncic, G Cosmo, D Dosaru, A Gheata, S Hageboeck, J Hahnfeld, M Hodgkinson, B Morgan, M Novak, A A Petre, W Pokorski, A Ribon, G A Stewart and P M Vila

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

Simulation on GPU – can we do that?

- GPUs are today widely available in HPC centers
  - Silicon that we have to use to increase HEP detector simulation throughput
  - A major challenge given the code complexity for particle transport (Geant4)
- Two main R&D projects spawned ~ 3 years ago
  - AdePT (CERN/SFT + collaborators) & Celeritas (ECP: ORNL, FNAL, Argonne, LBL)
  - Looking at the problem from different angles & learning from each other (++)
  - Inter-project meetings and a community mini-workshop one year ago
    - marking the completion of a first R&D phase
- We can actually run complex (LHC level) simulation on GPUs now
  - Standalone, but also integrated in a Geant4-driven workflow (see this talk)

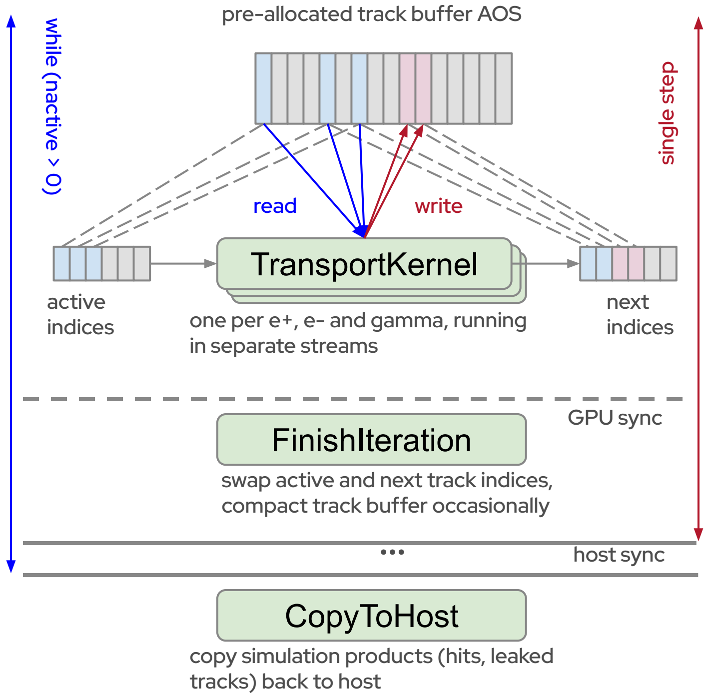
The AdePT project



- ▶ Putting together all pieces needed to run EM shower simulation on GPU
  - GitHub repository, initial commit in Sep 2020, $\mathcal{O}(10)$ contributors
- ▶ Strategy: integrate gradually features as new examples
  - Core infrastructure, physics and geometry as external services
  - Lightweight transport stepping loop evolved gradually in examples
  - Not a framework approach at this point, to maximize flexibility to change/adapt/integrate
- ▶ Minimal external dependencies
  - Geometry: ***VecGeom*** library, enhancing its GPU-related features
  - Physics: ***G4HepEm*** library, a compact GPU-friendly port of Geant4 EM interactions
- ▶ A first integration approach with Geant4 available (via fastsim hooks)
  - Discussions/evaluation with ATLAS, CMS and LHCb for testing and integration

Stepping loop at a glance

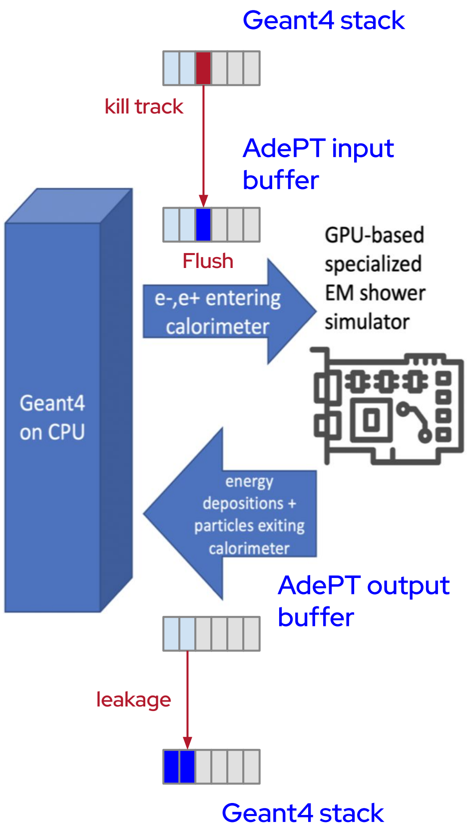
- Simulation is done in steps as in Geant4
  - Difference: All active tracks available are stepped at once (exposing the parallelism to the GPU)
- A step may be limited by physics
  - All EM physics calculations delegated to the external G4HepEm library
    - modeling all interactions for e^+, e^- and γ, verified against Geant4
- … or geometry
  - Calculations delegated to the VecGeom library
  - GPU port not GPU-friendly …
    - The main bottleneck (see this talk)
    - **Radical changes** for geometry GPU support: main optimization work during last year



36 SM GPU (consumer class)≃32 CPU cores in HT mode (dual socket CPU) for simple setups (see backup)

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
  - This approach is under evaluation by several experiments

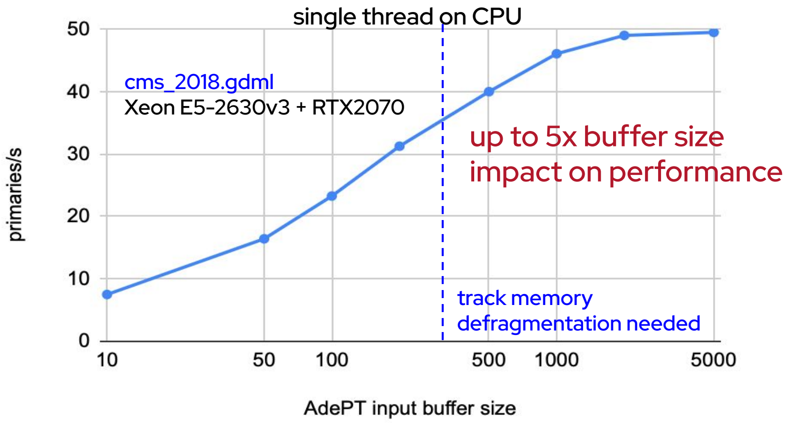


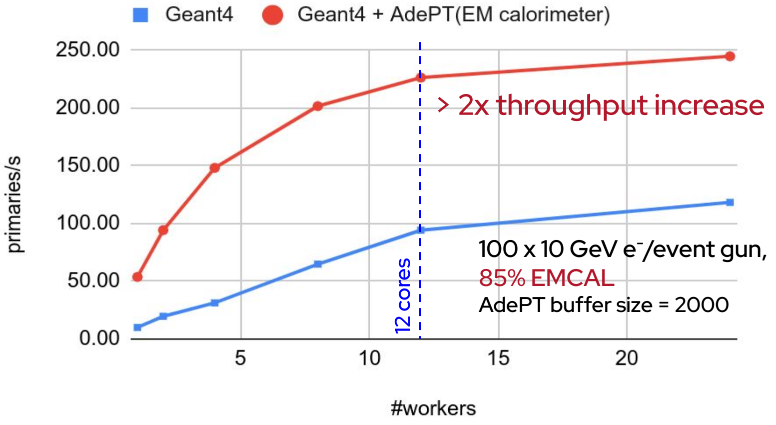
Integration performance

- Performance in this approach increases with :
  - Fraction of time spent in the GPU-accelerated region (Amdahl's law)
  - GPU buffer size (up to 5x impact)
- Performance degrades with :
  - Number of exchanges CPU-GPU per event
  - Number of CPU threads (GPU saturation, CPU transport stalled while GPU loop running)
- Why not the full EM on GPU?
  - Not limited by geometry or physics
    - Lepto-nuclear processes are rare and can be delegated to CPU
  - Limited by sensitive detector code GPU awareness
    - Incentive to write GPU-friendly sensitive detector scoring even for e.g. trackers

Higher is better

CMS benchmark



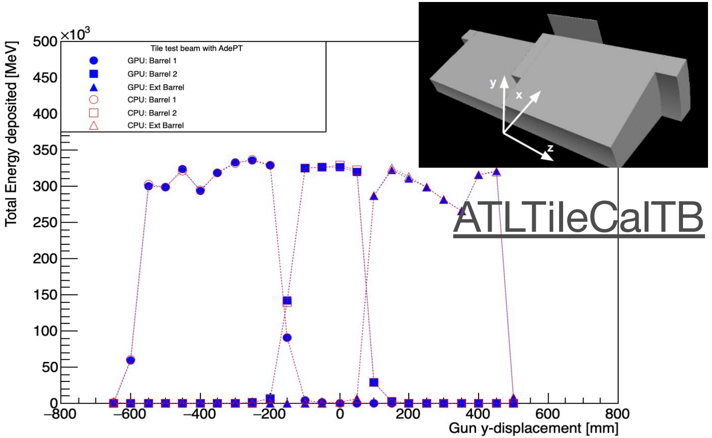


Towards integration with experiments

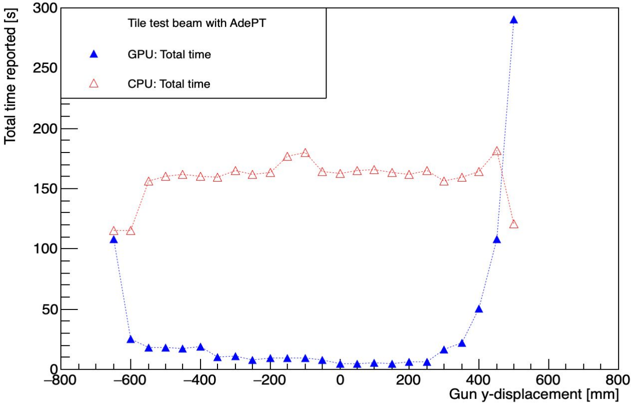
- CMS: targeting Phase 2 setup, in particular CMS HGCal
  - First steps already made
    - Loading the CMSSW-exported geometry setup in AdePT integration example
    - Configure **HGCalRegion** to offload electrons, positrons and gammas
    - Load *HepMC3* file with minimum bias events
    - Integrate G4HepEm in CMSSW builds, usable in an optional physics list
  - Particularly challenging due to the large number of channels, so sparsity needs to be used
  - Other challenges: requesting GPU resources and handling CPU-GPU exchanges in CMSSW
- ATLAS: typical “try out and adapt to my framework path”, see next
- LHCb: initial discussions and an integration project started
  - Discussed the possibility to stop particles entering LHCb EMCAL and run AdePT via Gaussino outside the Geant4 simulation step

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), "a couple of days effort to have something running"
  - More complex scoring (e.g. Birk's law on device)
  - How to avoid code duplication CPU/GPU?
  - Thinking about integration with *FullSimLight*



ATLTileCalTB



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

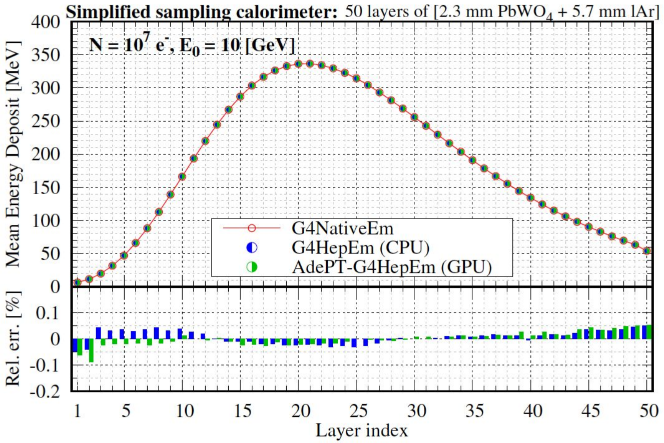
Outlook

- A challenging project, evolving through a second phase
  - Two R&D projects, AdePT and Celeritas, collaborating and working on alternative strategies to tackle simulation on GPU
  - A first phase completed, answering most of the initial R&D questions
    - We can now run complete EM shower simulation on GPU, both standalone and integrated with Geant4
  - Efficiency bottlenecks identified
    - New VecGeom project on GPU-friendly geometry surface modeling
- Main work focused on adapting the workflow to integrate with experiments
  - Working with experiments on specific challenges
    - framework integration, sensitive detector code requirements
  - A required step for validating GPUs as accelerating alternative for HEP simulation

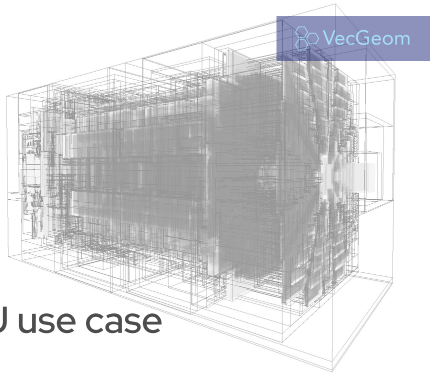
Backup

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: geometry presentation)

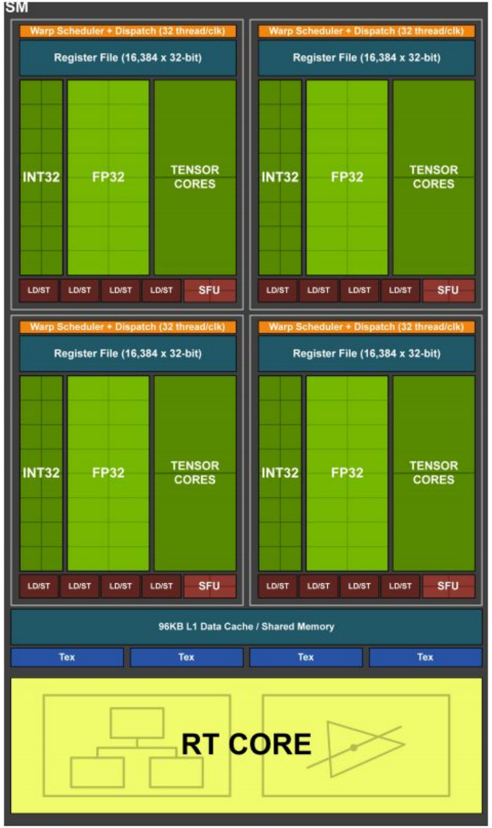
Kernel Launch Configurations

- 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 ↓

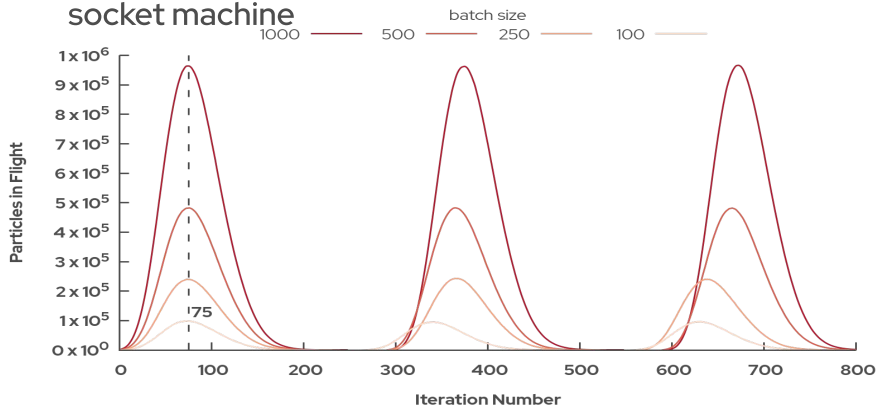
Faster Threads ↑

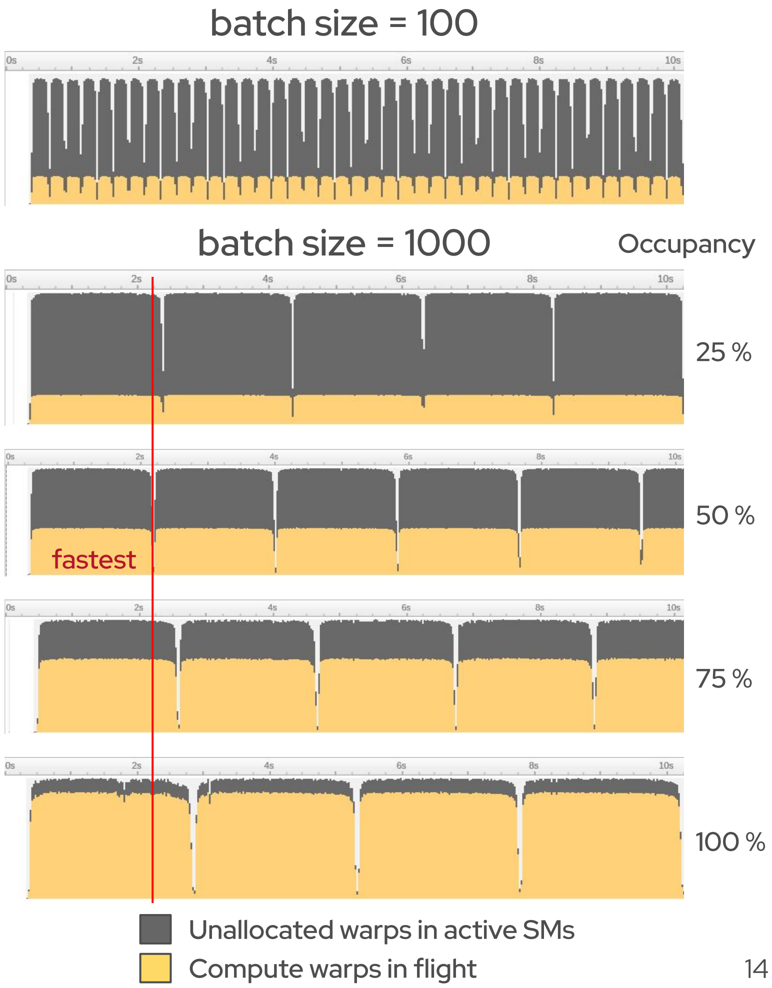
Turing SM



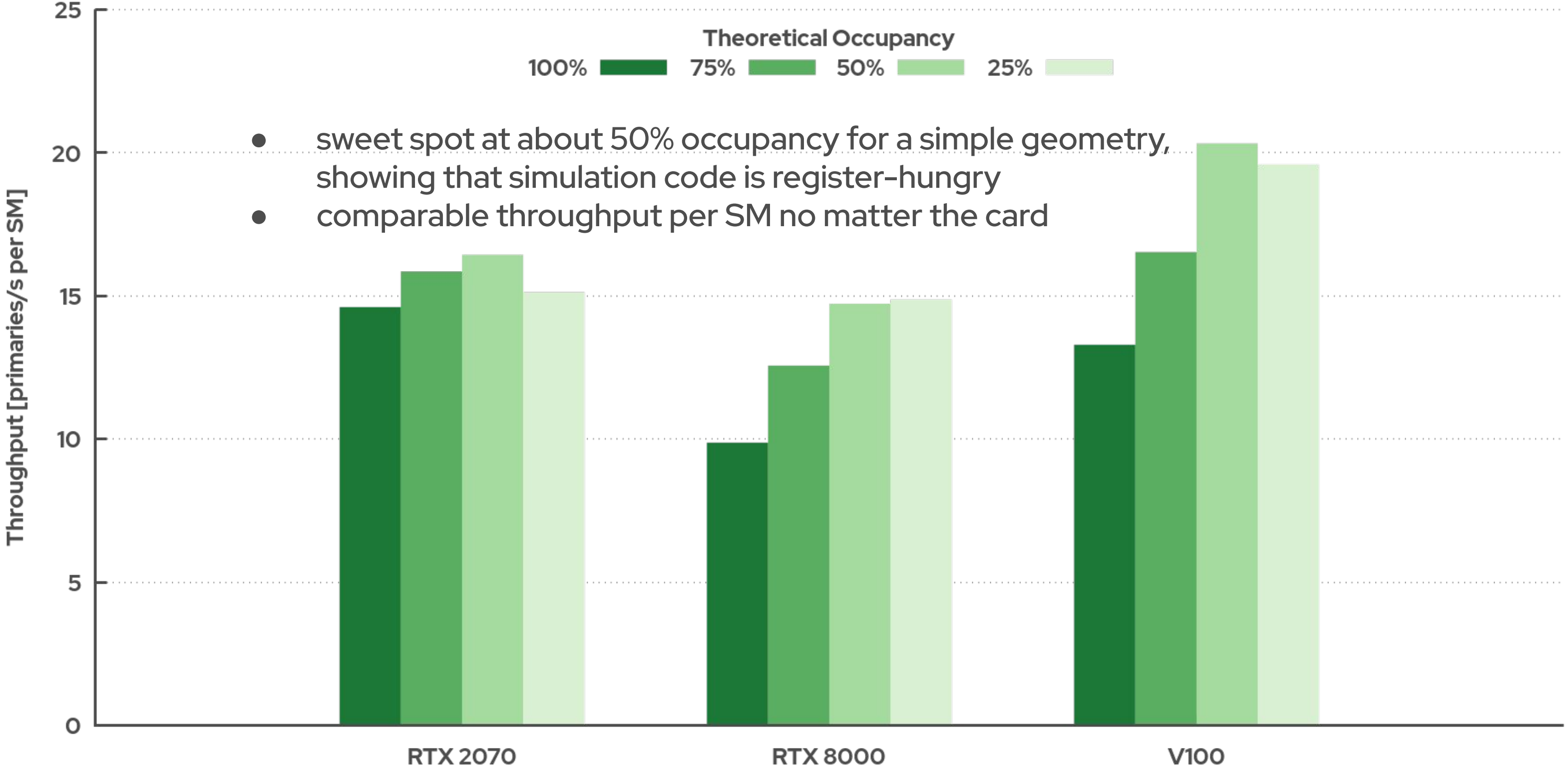
Run Time Characteristics

- 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 ≃32 CPU cores in HT mode (64 threads): a consumer card can double the throughput of a dual socket machine



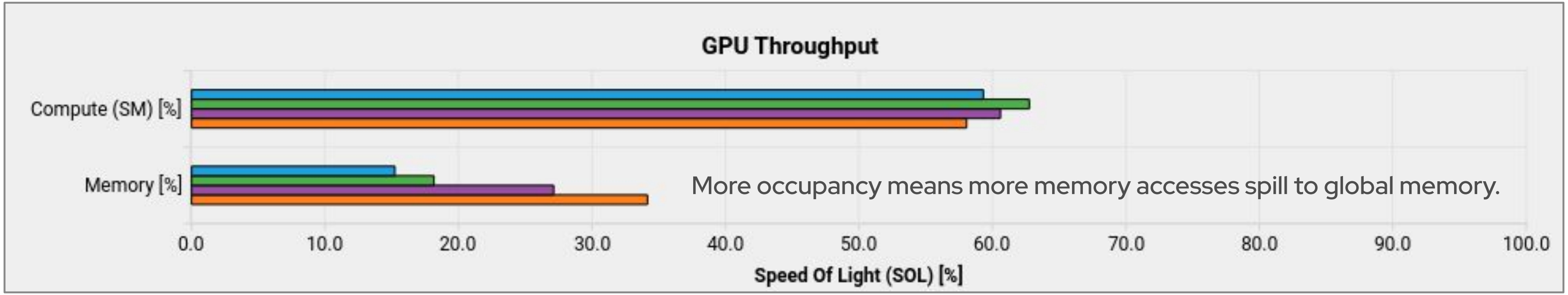


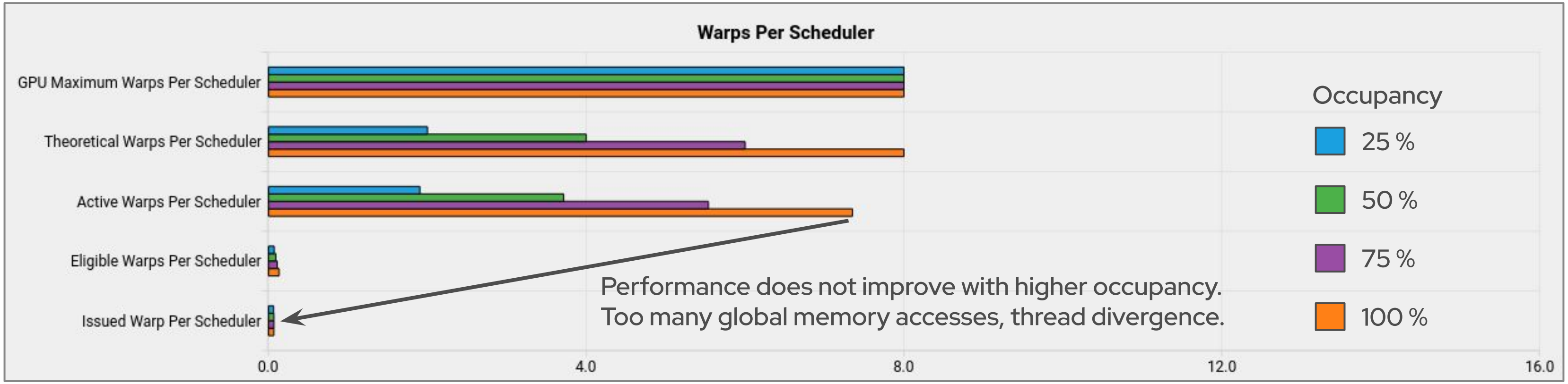
Relative Performance per SM



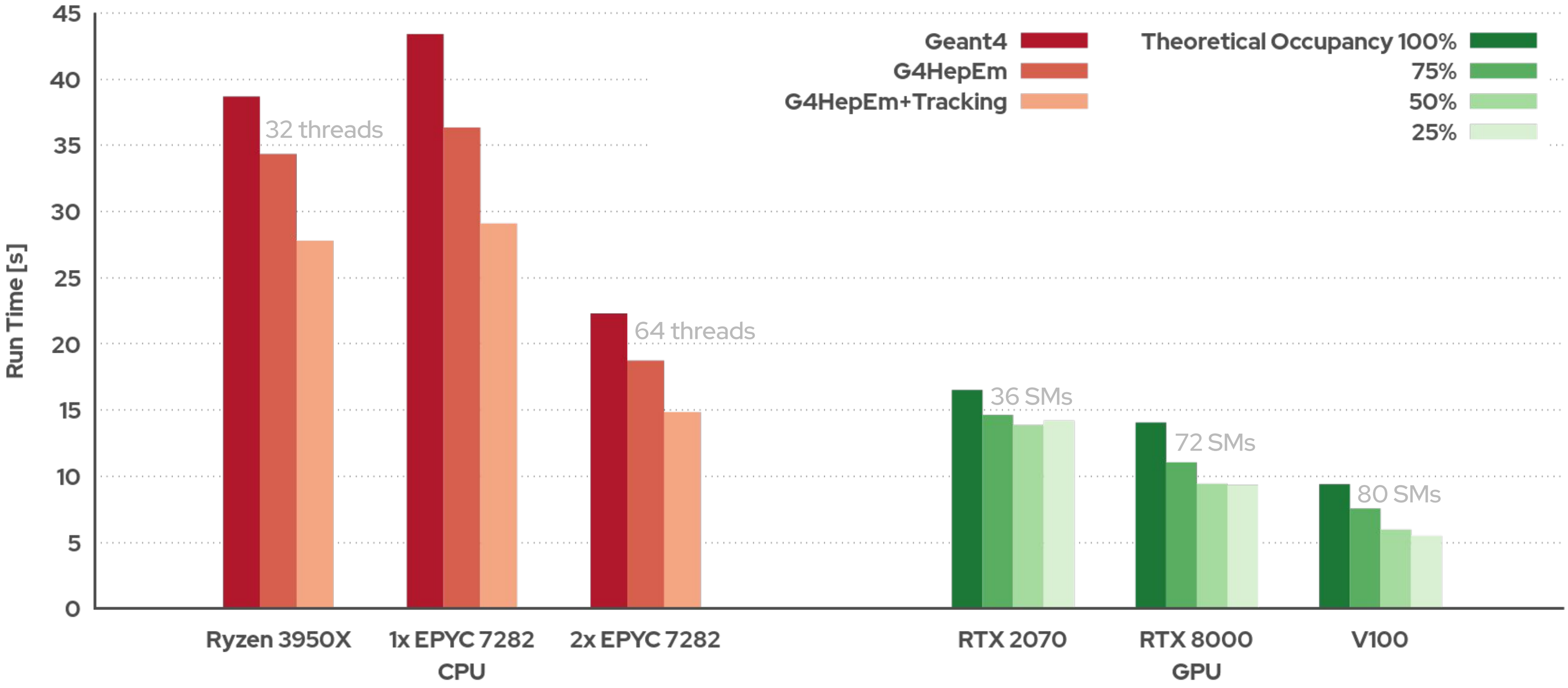
- sweet spot at about 50% occupancy for a simple geometry, showing that simulation code is register-hungry
- comparable throughput per SM no matter the card

GPU Throughput (RTX 2070)



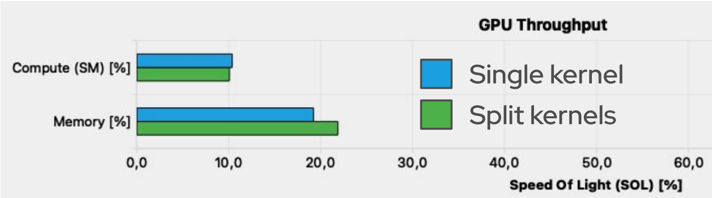


CPU vs GPU Performance



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



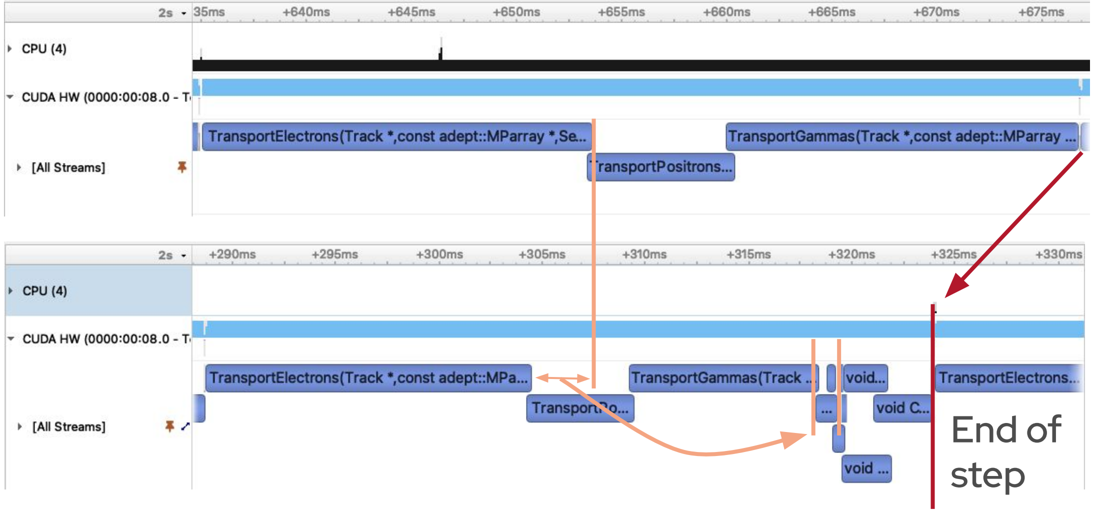
Problem: Threads in transport kernels diverge because of diverging interactions
→ 13 / 32 threads active on average

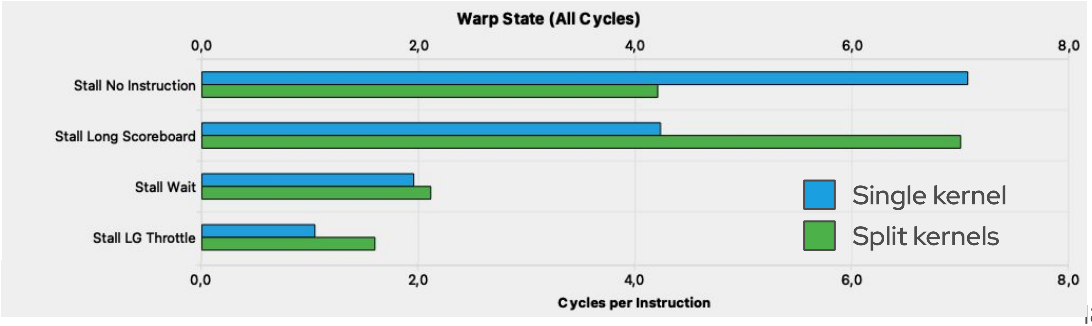
Here: Split off interaction computations from cross-section and geometry kernels (one kernel for pair creation, one for ionisation, …)

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

V100





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;
```