AdePT - Enabling GPU electromagnetic transport with Geant4

Juan González for the AdePT team

21/10/2024

CHEP 2024

j.gonzalez@cern.ch

Project Targets

- Understand usability of GPUs for general particle transport simulation
  - Seeking potential speed up and/or usage of available GPU resources for HEP simulation
- Provide GPU-friendly simulation components
  - EM Physics, geometry, field, but also data model and workflow
- Integrate in a hybrid CPU-GPU Geant4 workflow

GPU Simulation components

- Physics: G4HepEM
  - Compact rewrite of EM processes, focusing on performance and targeted at HEP detector simulation applications
  - Adapted for GPU
- Geometry: VecGeom
  - GPU adaptation built on top of the original VecGeom GPU/CUDA support
  - Includes several GPU-focused improvements, like an optimised navigation state system, and a BVH navigator.
- Magnetic Field: Uniform field with helix propagator
  - 3D field map and Runge-Kutta integrator in development

Major changes since the last AdePT update at CHEP 2023

- New method for Geant4 integration
- New method for scoring
- Refactoring of AdePT into a library allowing for simple integration
- Successfully tested integration into the
    - ATLAS Athena framework
    - LHCb Gaussino framework
- Asynchronous AdePT backend
- Major development in VecGeom's Surface geometry model

Geant4 Integration

- Before: integration using the Fast-simulation hooks
  - Tracks are intercepted based on detector region
  - Easy way to define a region for GPU transport
  - But: not flexible when trying to do GPU transport in multiple regions or the complete geometry

- Now: integration using a specialized G4VTrackingManager
  - Attaches to all EM particles as a process, we can filter on arbitrary criteria
  - Much more customizable
  - Simplifies the integration from the user’s point of view

Geant4 Integration using the specialized AdePT Tracking Manager

- The user only needs to register the AdePTPhysicsConstructor in their physics list (1 Line!)
- AdePT can be configured through an API or macro commands
- Example integration with the HGCAL Test-beam app by L. Pezzotti for geant-val, see this PR
    - CMake integration and minimal changes to the application

Scoring

- Before: AdePT kernels included a simplified scoring on device
  - Good for validation but not a realistic use case

- Now: Sending back hit information, and calling the user-defined sensitive detector code on CPU
  - Sensitive volume information taken from the geometry
  - GPU hit information is used to reconstruct G4 steps
  - **No changes** to the user SD code are needed

LHCb's Gaussino framework integration

- Gaussino allows to configure and to steer the different phases of detector simulation
- Provides wrappers for the Geant4 physics constructors and allows to build the Geant4 modular physics list using a simple Python configuration
- Gaussino has now been extended with such a wrapper for the AdePTPhysicsConstructor

```
GaussinoSimulation(
    PhysicsConstructors=[
        "GiGaMT_AdePTPhysics",
        "GiGaMT_G4EmStandardPhysics",
        "GiGaMT_G4EmExtraPhysics",
        "GiGaMT_G4DecayPhysics",
        "GiGaMT_G4HadronElasticPhysics",
        "GiGaMT_G4HadronPhysicsFTFP_BERT",
        "GiGaMT_G4StoppingPhysics",
        "GiGaMT_G4IonPhysics",
        "GiGaMT_G4NeutronTrackingCut"])
```

It can now be added to a simulation in a **single line!**

LHCb’s Gaussino framework integration

- Additional AdePT configuration can be passed through the Gaussino wrapper for Geant4 configuration macros

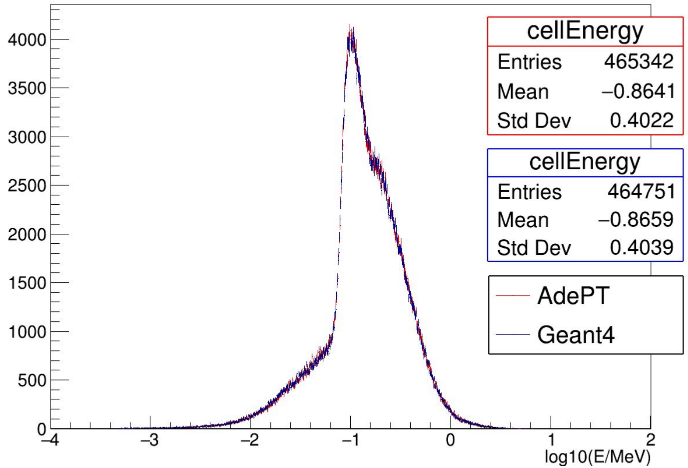
```
GiGaMTRunManagerFAC("GiGaMT.GiGaMTRunManagerFAC").InitCommands = [
   "/adept/setVecGeomGDML calochallenge.gdml",
   "/adept/addGPURegion CaloRegion" #"/adept/setTrackInAllRegions true"]
```

- Using the scoring mechanism discussed on slide 7
  - AdePT calls the appropriate Gaussino sensitive detectors to create hits as in a normal Geant4 simulation

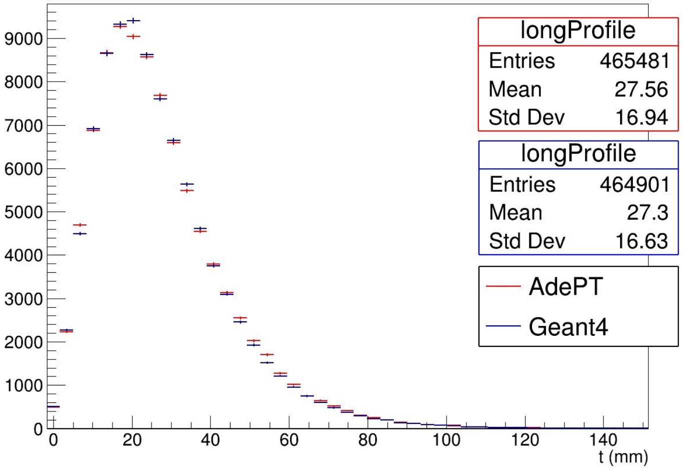
Gaussino integration - CaloChallenge setup

- Physics results show a good agreement with Geant4
- For enough particles sent to the GPU, the gains can be significant
    - Achieved **5x speedup** with 4 CPU threads in initial tests with gamma-only events

Cell energy distribution



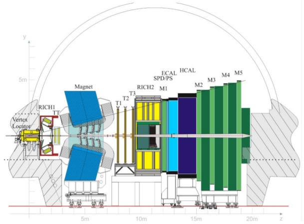
Longitudinal profile

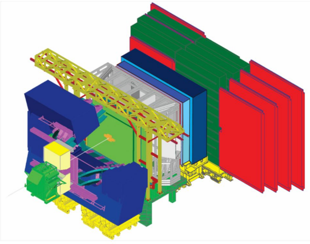


Gaussino integration - status and next steps

- AdePT integration works out of the box except…
  - Full MCtruth information is not available for GPU tracks
  - Not possible to carry over custom ‘user track information’ to the GPU (custom approaches could be implemented)

- Now: Working on full LHCb setup with AdePT through Gaussino
  - Working fine out of the box, with all LHCb sensitive detectors and monitoring functioning
    - Debugging some discrepancy in the number of hits

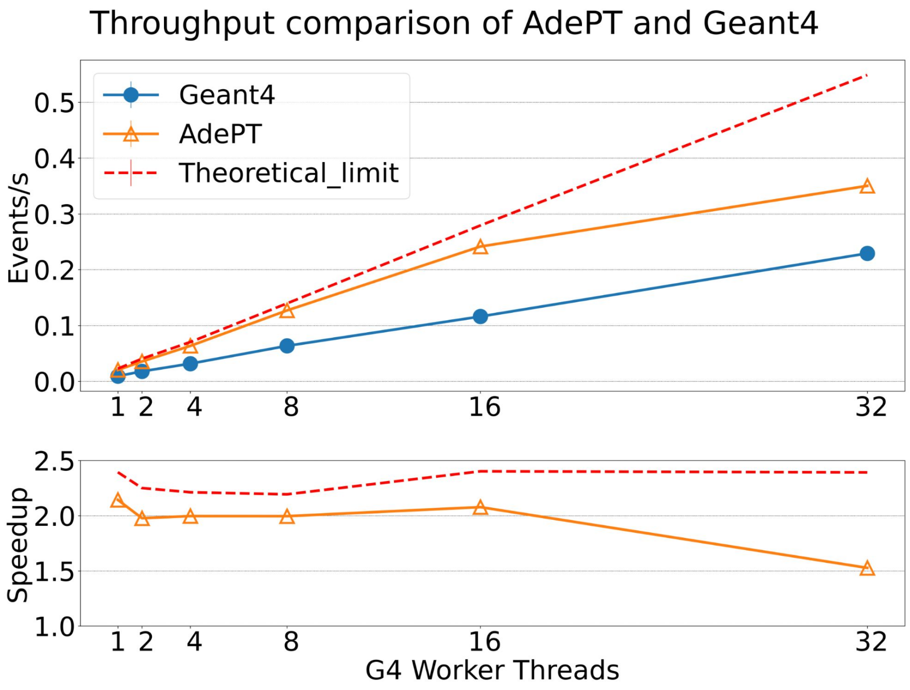




Current performance results with CMS 2018 geometry

- Getting a speedup of around 2 in a realistic use case
- At some point, AdePT's scaling becomes slower
  - Low device occupancy and high divergence mean that a saturated GPU slows down
  - More on how we are addressing this next

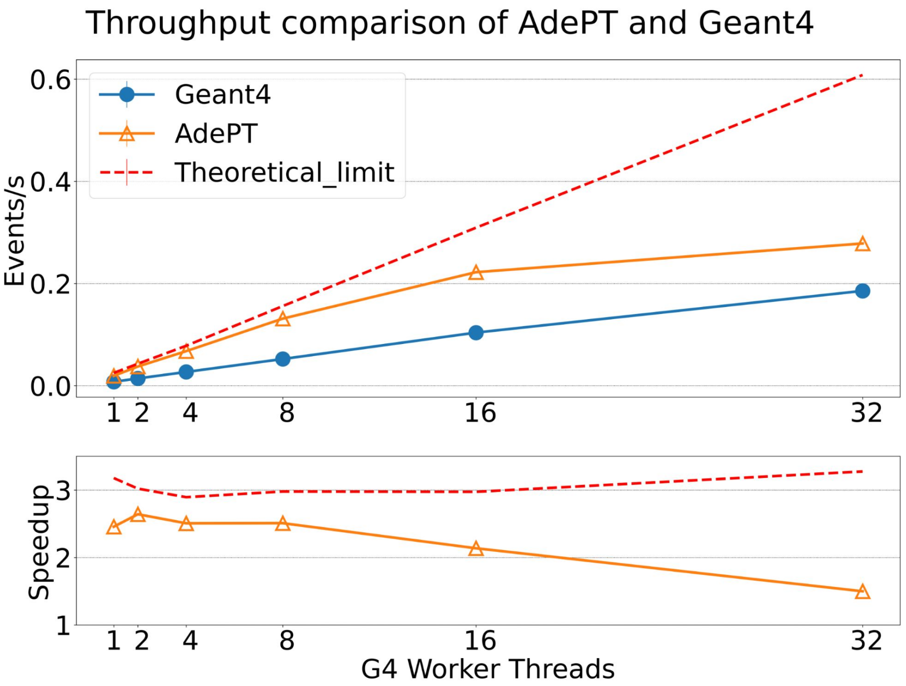
```
GPU: Nvidia A100
Input: 4 TTBar per thread
Geometry: CMS2018
No magnetic field
```



**Theoretical limit: All EM tracks killed as they are produced*

Current performance results with CMS Run 4 geometry

- For now we don't have sensitive volume information for CMS Run4
- Scoring will have some effect on performance but the initial results are promising

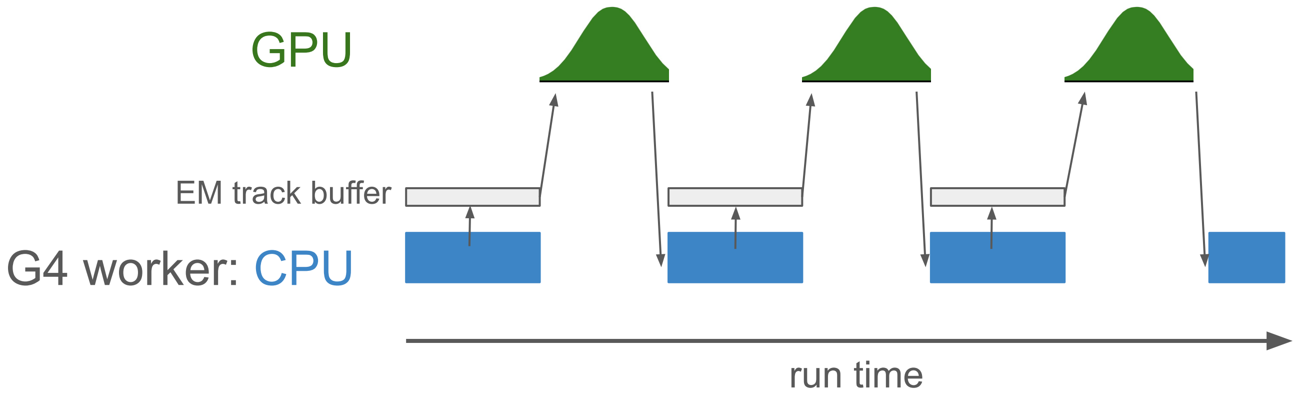



```
GPU: Nvidia A100
Input: 4 TTBar per thread
Geometry: CMS Run 4 (No Scoring)
No magnetic field
```

Two performance bottlenecks identified

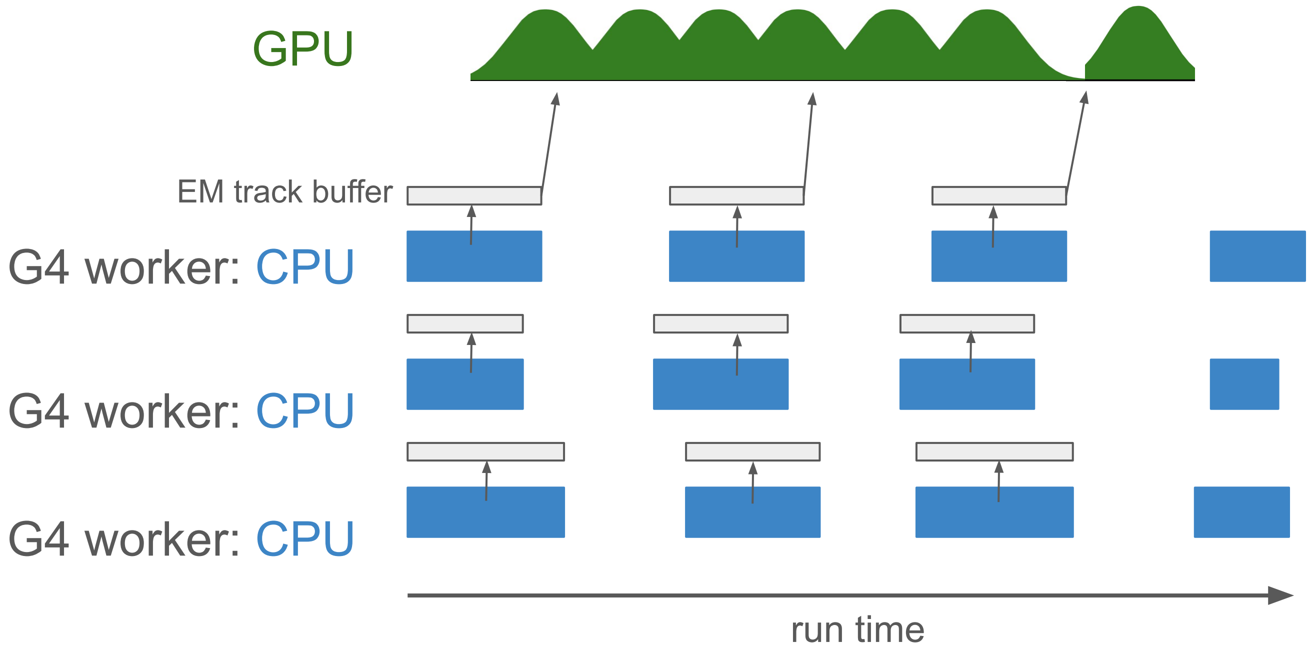
- Geometry
  - The current solid-based geometry has two main issues on GPU:
    - The relatively large number of solid types causes warp divergence
    - The code is complex and register-hungry, which limits the maximum occupancy
- Kernel scheduling
  - The current approach to kernel scheduling blocks the calling thread while the GPU transports a batch of particles
    - This becomes more relevant as the GPU gets saturated and slows down

Synchronous kernel scheduling uses GPU *sequentially*



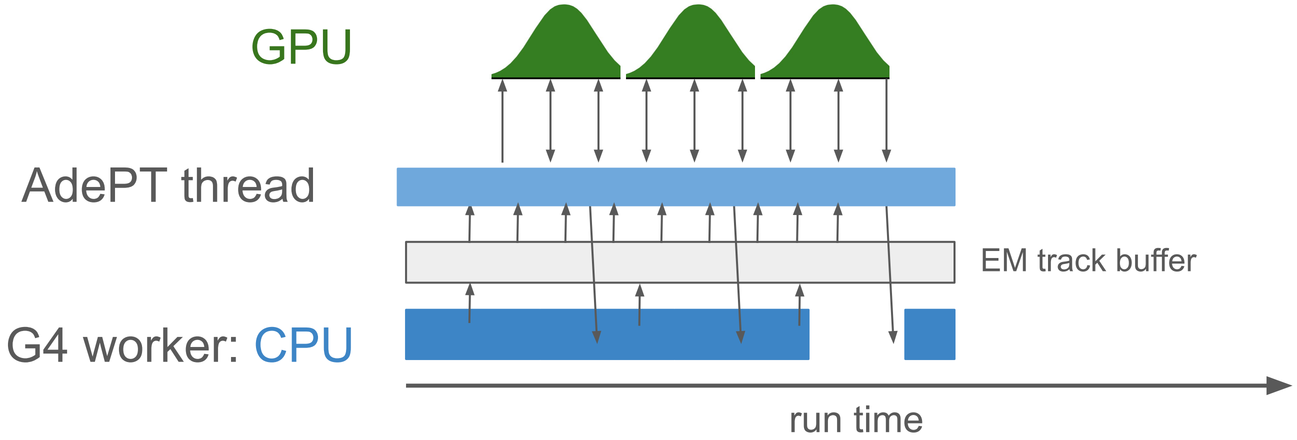
- Each G4 worker thread calls the GPU individually
  - Launching parallel transport of the buffered tracks
- Computation on CPU and GPU are done **sequentially**

Synchronous kernel scheduling uses GPU *sequentially*



Multiple threads can fill the GPU but the CPU still needs to wait

Asynchronous kernel scheduling uses GPU in *parallel*

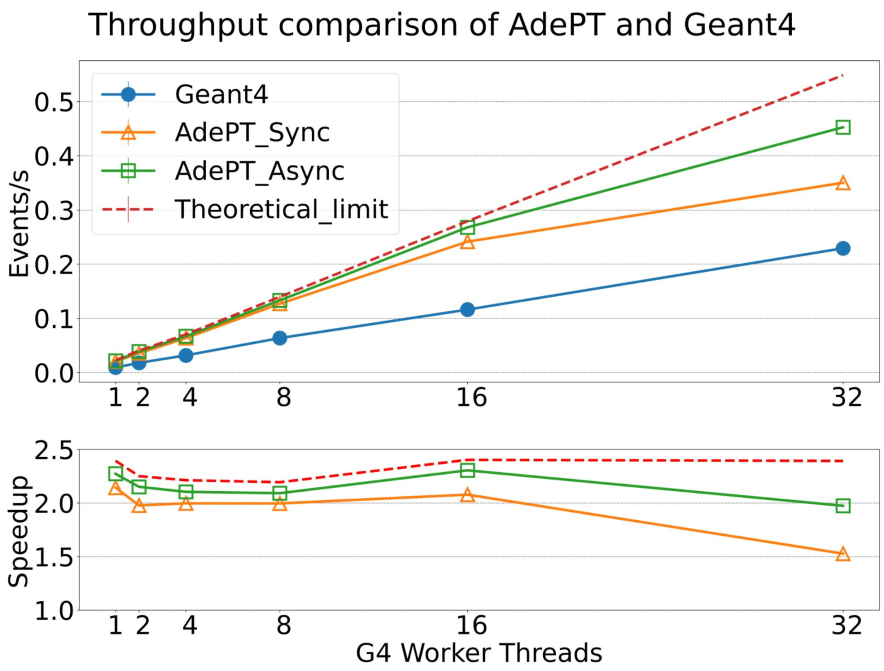


- Each G4 worker thread writes to the same buffer
- **Separate AdePT thread** takes care of kernel scheduling (and scoring)
- Better CPU utilization due to **parallel** computation
  - Host threads can continue with other work (e.g. Hadrons)
  - Synchronization is only needed at the end of an event

Asynchronous kernel scheduling (CMS 2018)

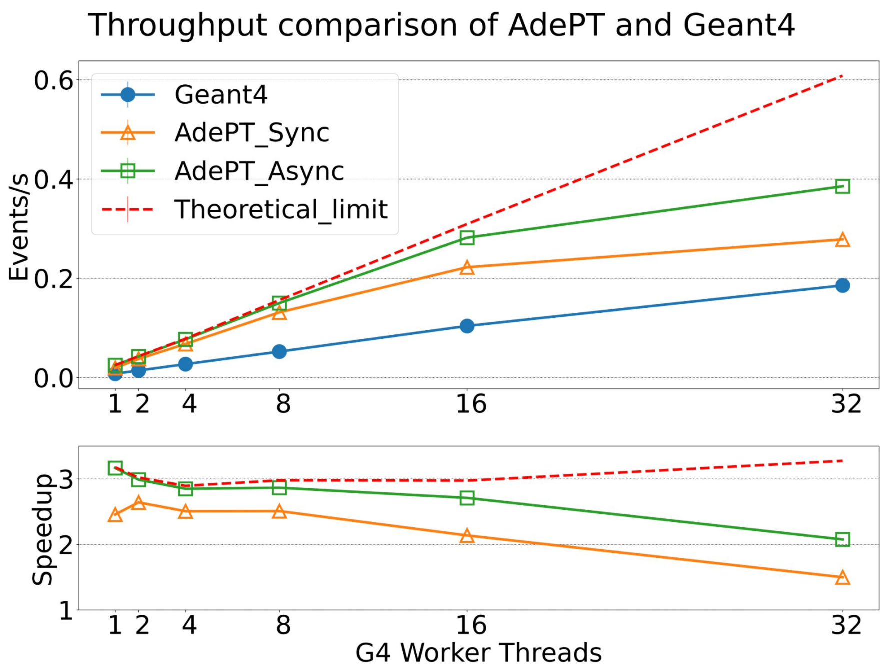
- Promising results in early tests
- The single-threaded speedup is better preserved when increasing the number of threads

```
GPU: Nvidia A100
Input: 4 TTBar per thread
Geometry: CMS2018
No magnetic field
```



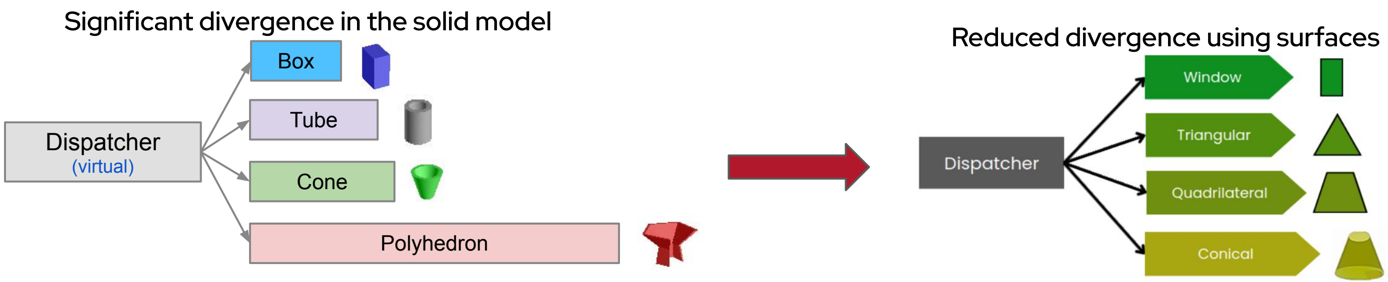
Asynchronous kernel scheduling (CMS Run 4)

GPU: Nvidia A100
Input: 4 TTBar per thread
Geometry: CMS Run 4 (No Scoring)
No magnetic field



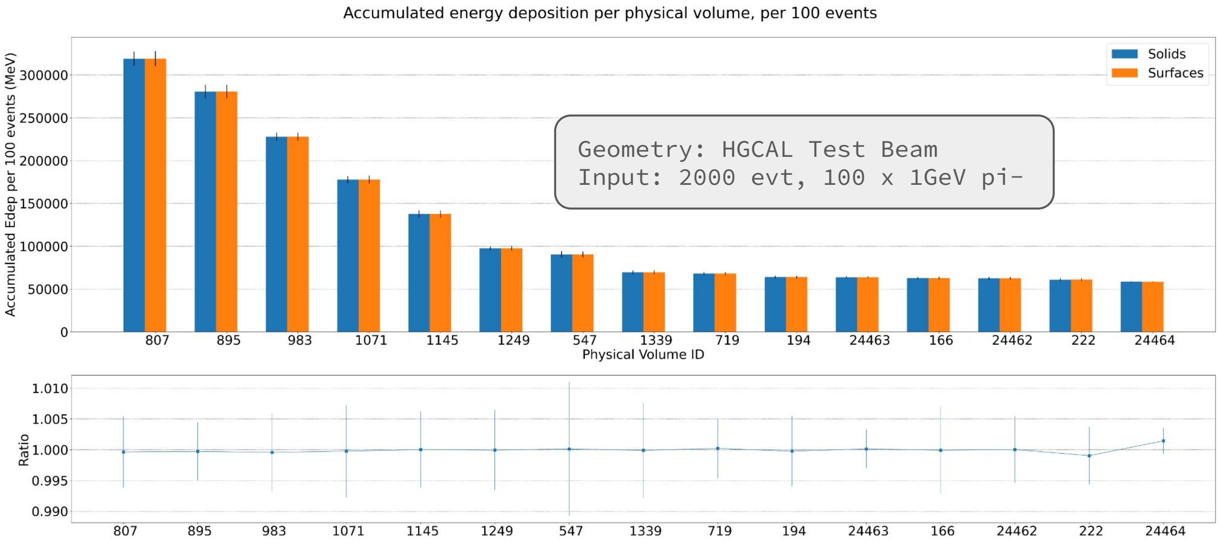
VecGeom surface model

- Simpler algorithms reduce register and stack usage
- Reduced number of primitives and lower complexity reduce divergence
- Potential to navigate using mixed precision



Status of the Surface Model

- Surface navigation already integrated into AdePT
  - Results pre-validated against solids
  - Similar performance to the solid model
    - Optimization still ongoing, with drastic performance improvements during the last months
    - Working on smaller AdePT kernels and a mixed-precision mode



Summary and outlook

- The G4VTrackingManager and new scoring approach ease integration into G4 applications
  - Further collaboration with experiments needed to find missing functionality and implement setup-specific solutions

- Two bottlenecks were tackled:
  - Poor GPU performance in solid-based geometry model
    ➡ New surface model with correct results implemented, optimization ongoing
  - Suboptimal kernel scheduling blocked CPU performance
    ➡ New asynchronous mode significantly improves performance