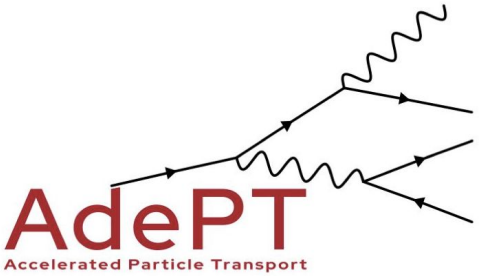




# AdePT Progress and Plans

Juan Gonzalez for the AdePT team
27/3/2024


# Project overview: **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
  - Physics, geometry, field, but also data model and workflow
- Integrate in a hybrid CPU-GPU Geant4 workflow

# Project overview: **GPU Simulation components**

- Physics: G4HepEM
  - G4HepEM is a compact rewrite of EM processes, focusing on performance and targeted at HEP detector simulation applications
  - It was adapted for use on the 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: Work in progress on a Runge Kutta field propagator
  - Currently only a uniform field with a helix propagator is validated

# Geant4 integration

- Previously AdePT integrated into Geant4 applications using the Fast-simulation hooks
  - They provided an easy way to define a region for GPU transport
  - However, this approach is not flexible when trying to do GPU transport in multiple regions or even the complete geometry

- A new integration approach uses a specialized G4VTrackingManager
  - Much more customizable than the Fast-simulation hooks
  - Simplifies the integration from the user's point of view

# Geant4 integration

- AdePT Integration using the specialized AdePT Tracking Manager
  - The user only needs to register the AdePTPhysicsConstructor in their physics list
  - AdePT can be configured through an API, for simplicity we also provide several macro commands for this
  - We recently provided an example integration with the HGCAL Test-beam application developed by Lorenzo Pezzotti for geant-val, which can be seen in this PR
    - Besides the CMake integration of AdePT, there are minimal changes needed in the user application

# Scoring

- Previously, the AdePT kernels included a simplified scoring that was done on device
  - Only the final information about Edep per volume was returned
  - The main motivation was avoiding the overhead of regularly sending information back to the host
  - One of the main questions about scoring on device is how to implement more realistic scoring codes
- The current approach is sending back hit information, and calling the user-defined sensitive detector code on CPU
  - No significant overhead has been observed
  - No changes to the SD code are needed
  - The information retrieved from the GPU should cover most use cases
  - In case of complex scoring code however, this reduces the code fraction that can be accelerated on GPU

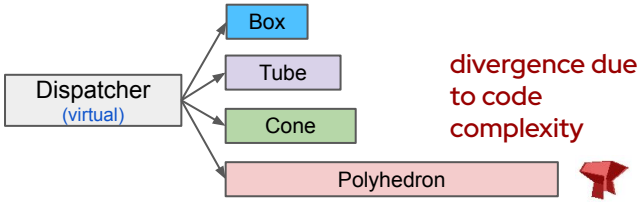
# Library version of AdePT

- Until recently, the project consisted on a series of mostly independent examples
  - Useful for exploration, but difficult to integrate into external applications

- AdePT has been re-organised as a library
  - There has been a major refactoring of the project, as a result integrating with AdePT is now as simple as linking against this library, and registering the AdePT physics constructor

# Major issues

- We have identified two major performance bottlenecks: Geometry and kernel scheduling.
  - 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




  - The current approach to kernel scheduling blocks the calling thread while the GPU transports a batch of particles
    - This has a very visible effect when the GPU is saturated

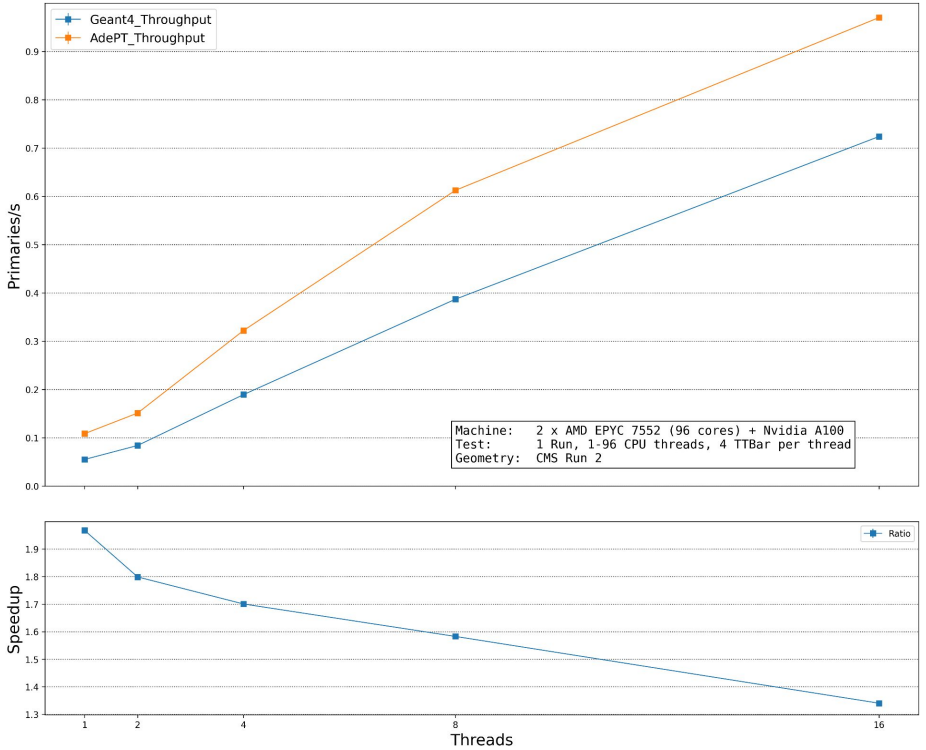
# Performance results

- Previous AdePT performance results were obtained using electron-only events
  - Good for showing the potential speedup in an ideal scenario, but not realistic

- The current version of AdePT has been benchmarked with TTbar events
  - The tests shown here were done using the CMS2018 geometry, transporting particles on GPU across the entire detector

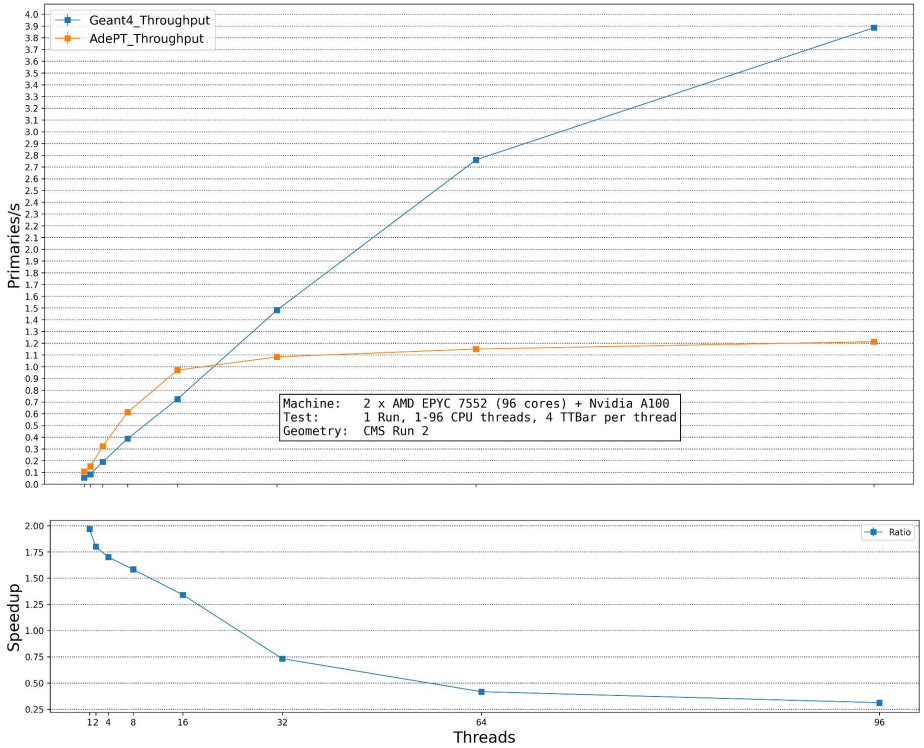
# CMS Run2, TTbar, full detector EM transport on the GPU, no field, Nvidia A100, 1-16 Threads

- For low numbers of threads, offloading the EM transport to the GPU gives some speedup, which decreases as the number of CPU workers increases and the GPU becomes more saturated.



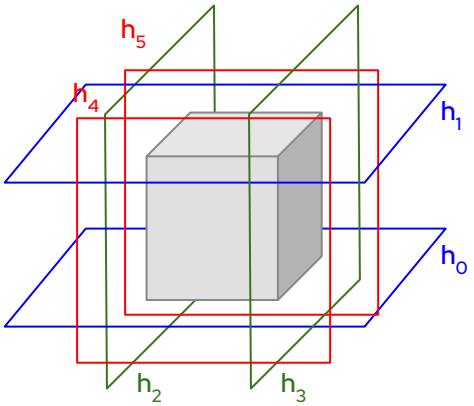
# CMS Run2, TTbar, full detector EM transport on the GPU, no field, Nvidia A100, 1-96 Threads

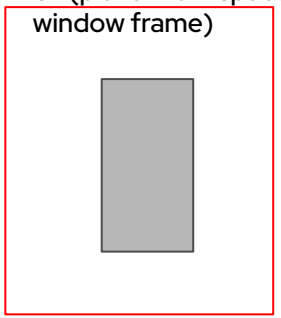
- At a certain point the GPU becomes saturated. Geometry is a major factor in how early this happens
- Due to the current way of scheduling the kernel launches, this means that the GPU starts blocking the CPU threads
  - The Geant4 worker is currently steering the GPU loop, being idle while the GPU kernels are running
- More research into non-blocking scheduling strategies is ongoing



# Ongoing development: **Bounded surface modeling**

- Working on a 3D solid representation closer to tessellations used by the graphics systems
  - Simpler code than 3D primitives, friendlier to GPU processing
- 3D bodies represented as Boolean operation of half-spaces
  - First and second order, infinite
- Storing in addition the solid imprint (frame) on each surface: FramedSurface
  - More memory needed than for unbound models, but easier to optimize/accelerate

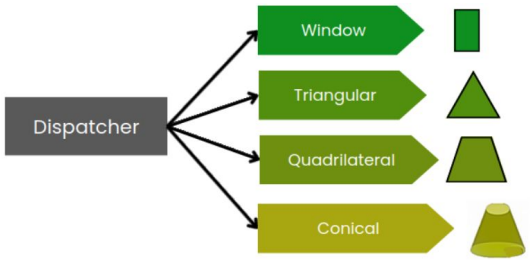


# Objectives for the surface model

- Portable GPU-friendly header library
  - Algorithmic part independent on the backend, compilable with any native/portability compiler
  - Headers templated on the precision type to allow for a single-precision mode
  - Reduced set of simple surface/frame algorithms (vs. ~20 primitive solids now)
    - Reducing divergence and register usage on the GPU
- Automatic conversion from VecGeom transient solid model
  - Transparent creation of the data structures and copy to GPU
  - Preserve awareness of the Geant containment feature as powerful optimization
- Code simplification and GPU performance
  - No virtual calls, no recursions, more work-balanced
  - Better device occupancy and kernel coherence

Reduced divergence using surfaces



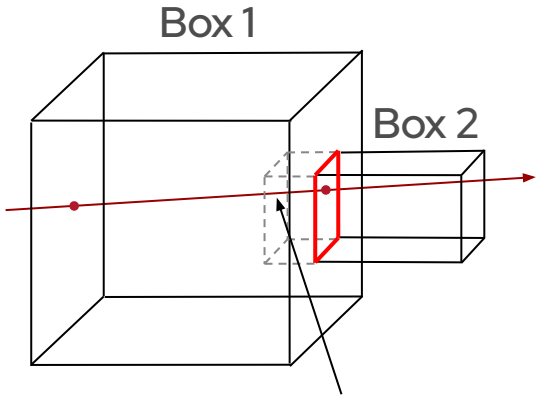
# Current status of the surface model

Surface model supports **all solids** required by the experiments


Currently debugging complex geometries (CMS, LHCb):
**Overlaps** require special treatment

**Next:**

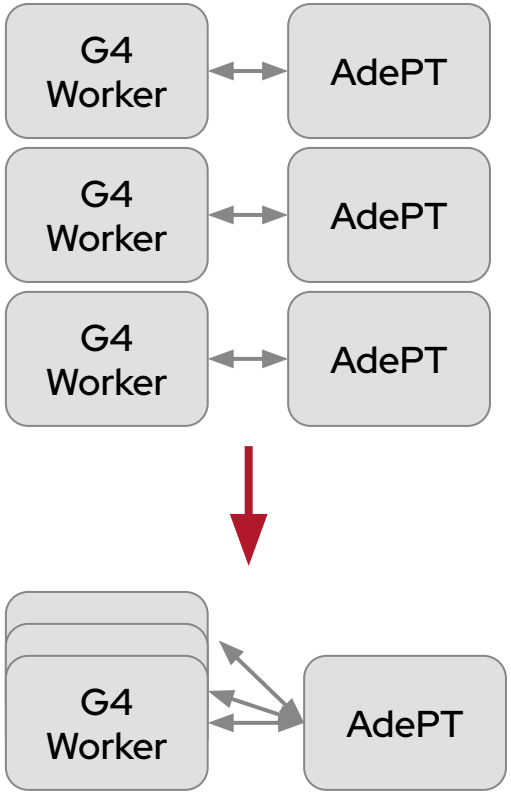
- Adding Bounding Volume Hierarchy accelerating structure
- Performance measurements, testing calorimeters (ATLAS EMEC, CMS HGCAL, LHCb ECAL)
- Lightweight portability layer (instead of pure CUDA)




Missing the overlapping entering surface leads to missing Box 2 entirely

# Future investigations: **Asynchronous kernel scheduling**

- **One** instance of AdePT runs in the background
- Transport loop runs continuously
- All G4 workers communicate with AdePT asynchronously
  - Host threads can continue with CPU work (e.g. Hadrons) while transport runs in the background



# Future investigations: **Asynchronous kernel scheduling**

- Results so far have been promising

- The new scheduling is only implemented in examples using the fast-simulation hooks integration, it will be ported to the new AdePT version for further testing