AdePT Status Report

EP R&D Software Working Group Meeting
18 Sep 2024

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

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

Recent developments

- New method for Geant4 integration
- New method for scoring
- Refactoring of AdePT into a library
  - Previously, the project consisted on a series of mostly independent examples
  - There has been a major refactoring to reorganise the project into a library, simplifying integration into external applications
  - Example integration with the GeantVal HGCAL Test Beam app
- Gaussino Integration
- Progress towards an asynchronous AdePT backend
- Major development in VecGeom's Surface geometry model

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

Gaussino Integration

- Gaussino is a framework allowing to configure and to steer the different phases of detector simulation
- It provides wrappers for the Geant4 physics constructors and allows to build the physics list (using the Geant4 modular physics list mechanism) using a simple Python configuration
- Gaussino has now been extended with such a wrapper for the AdePTPhysicsPhysicsConstructor (see slide 6)
- this allows to run Geant4+AdePT simulation by simply adding the appropriate line in the Python configuration

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

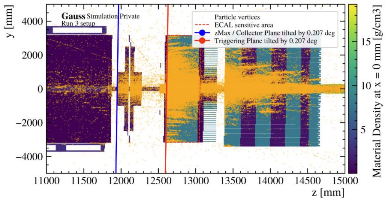
Gaussino integration

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

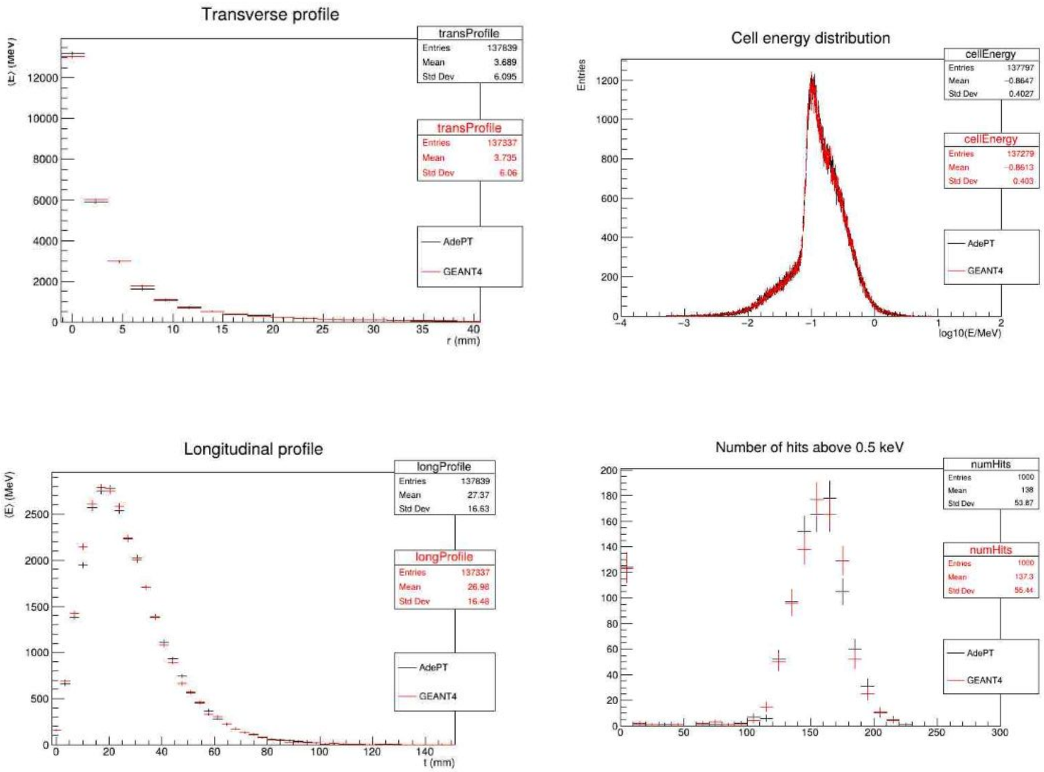
```
GiGaMTRunManagerFAC("GiGaMT.GiGaMTRunManagerFAC").InitCommands = [
   "/adept/setVecGeomGDML calochallenge.gdml",
   "/adept/addGPURegion CaloRegion" #"/adept/setTrackInAllRegions true"]
```

- Using the scoring mechanism discussed on slide 7
  - AdePT reads which volumes in Gaussino (Geant4 geometry) are sensitive and marks them for the GPU
  - Scores the energy depositions in those volumes
  - Sends them back to the host and calls the appropriate Gaussino sensitive detectors to create hits as in a normal Geant4 simulation
- No modification to the user scoring code are (in principle) required

Gaussino integration - Calo Challenge setup

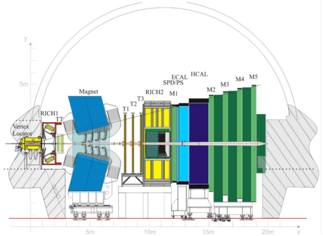


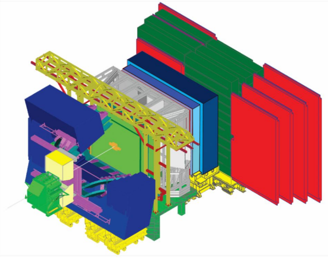
- Gaussino - AdePT integration has been successfully used for the Calo Challenge setup
- Physics results show a very good agreement with Geant4
- In events of 1000 x 1GeV gammas:
  - x7 speedup in single-thread mode
  - x5 speedup with 4 threads
- If there are enough particles sent to the GPU, the gains can be significant
  - these numbers can be considered as upper limits for real events



Gaussino integration - issues and next steps

- AdePT integration through Gaussino works out of the box except…
  - When user code expects to have access to full MCtruth information associated to the hits
    - We assign the hits to the particle which entered the GPU region
      - More sophisticated approaches possible but need to be application-specific
  - When user code relies on custom ‘user track information’ attached to tracks
    - We can’t carry it over in the GPU regions, so some specific approaches need to be adopted
- Next steps
  - Currently working on full LHCb setup with AdePT integrated through Gaussino
    - Working fine out of the box, with all LHCb sensitive detectors and monitoring functioning
      - Debugging some discrepancy in the number of hits
        - Probably due to some cuts
  - Tests with min-bias events on the way





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

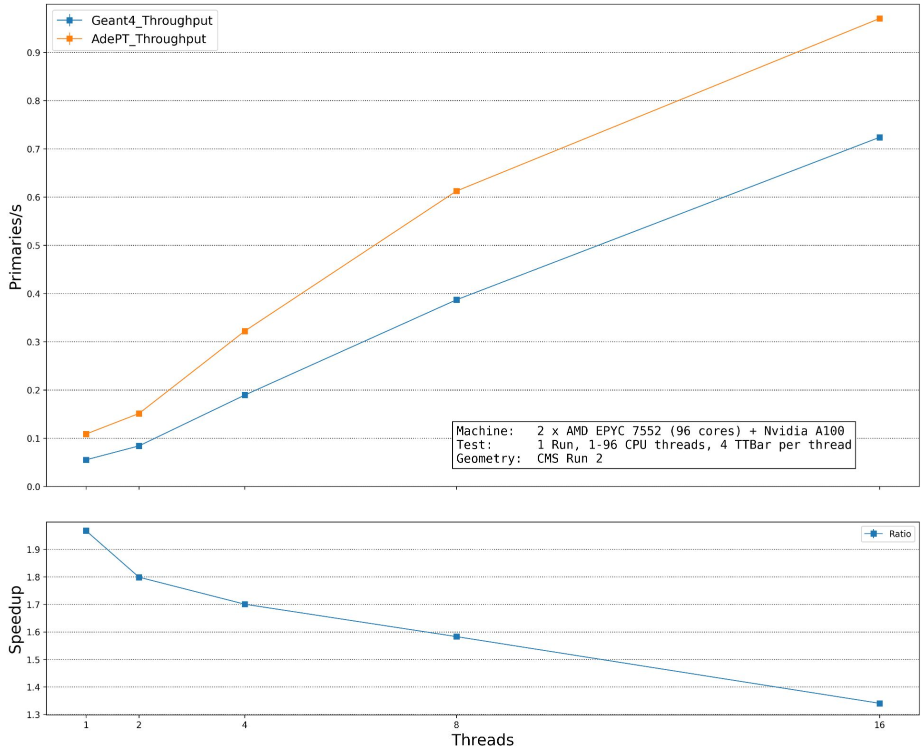
Current performance results

- 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

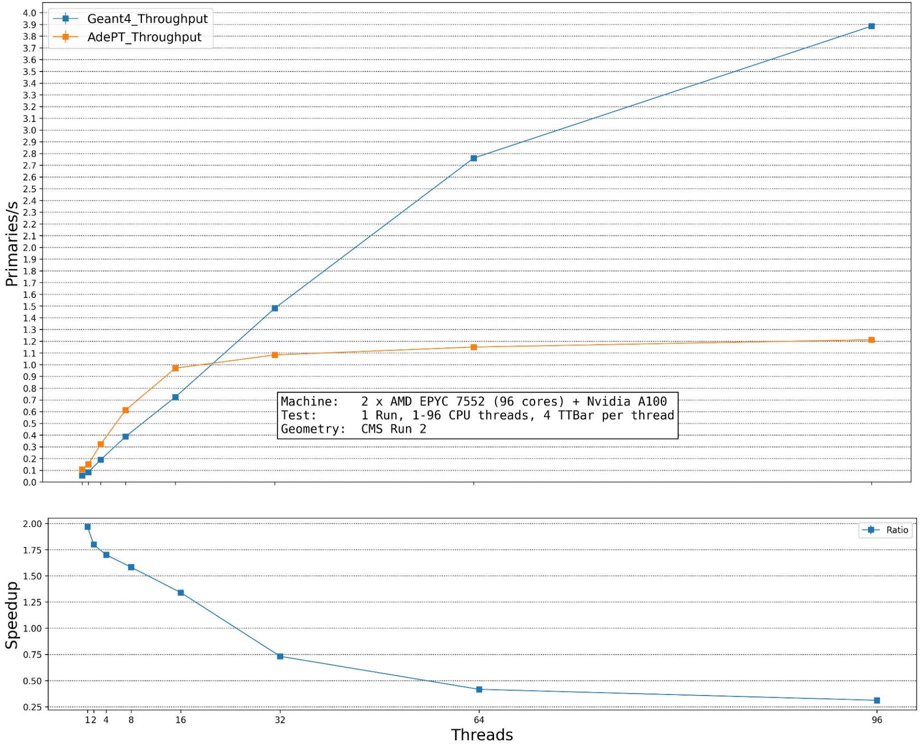
CMS Run2, TTbar, full detector EM transport on the GPU, No field, Nvidia A100, 1-16 Threads

- 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

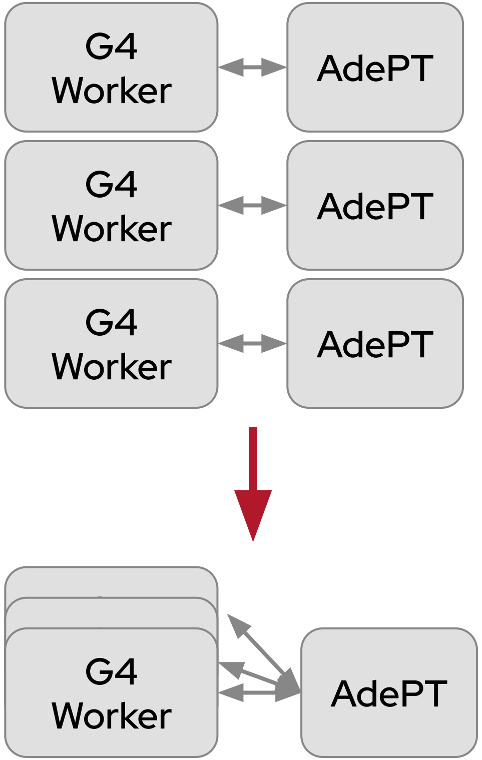


Asynchronous kernel scheduling

- Currently, CPU threads track particles across the geometry, and buffer EM particles entering marked GPU regions
- When the buffer is full, the thread triggers the transport on GPU
  - The caller blocks until the GPU finishes tracking
- As long as the GPU is not saturated this is inefficient, but we still see a speedup
- However when the GPU becomes slower, it stops the CPU from tracking particles in other regions

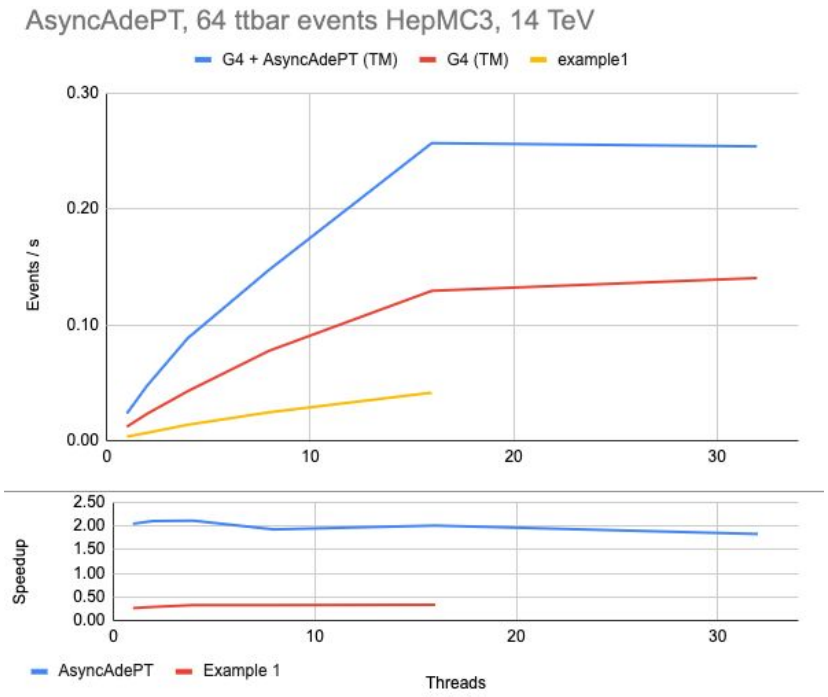
Asynchronous kernel scheduling

- Asynchronous scheduling prototype
- Only one instance of AdePT running in the background
- It continuously runs the transport loop
- All G4 workers communicate with AdePT asynchronously
    - Host threads can continue with CPU work (e.g. Hadrons) while transport runs in the background



Asynchronous kernel scheduling

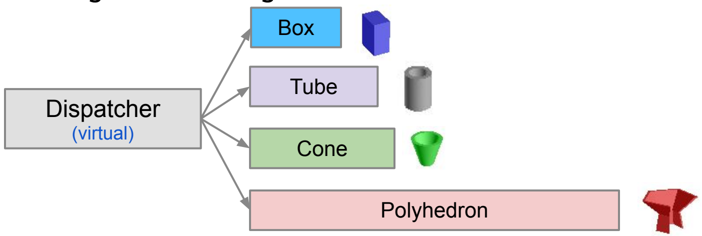
- Promising results in early tests
- In this example, the GPU was never saturated
- The single-threaded speedup is preserved when increasing the number of threads



VecGeom solid model is a huge bottleneck on GPU

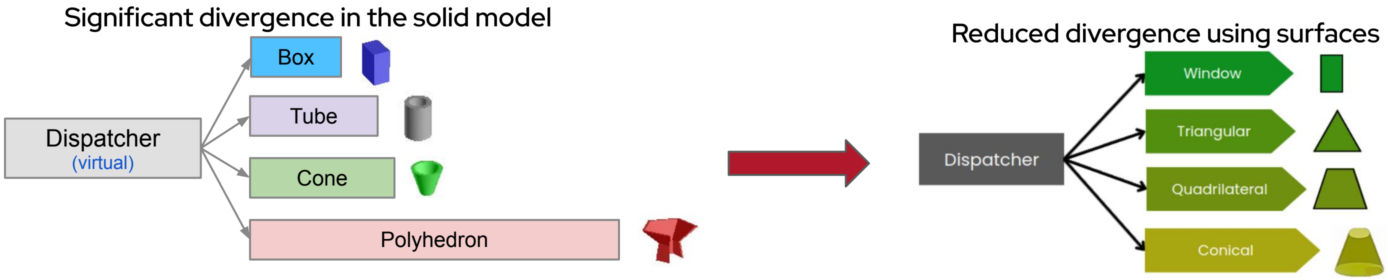
- recursive calls, virtual functions, complicated algorithms → high register and stack usage → low occupancy on GPU
- very different complexity per solid → high divergence
- uncoalesced memory accesses → high latency
- relies on small pushes for knowing in which volume one is → requires double precision

Significant divergence in the solid model



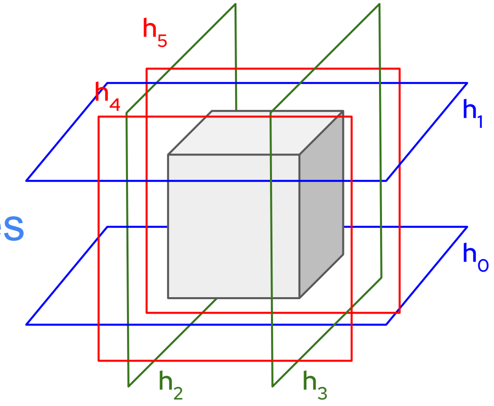
VecGeom surface model optimized for GPUs

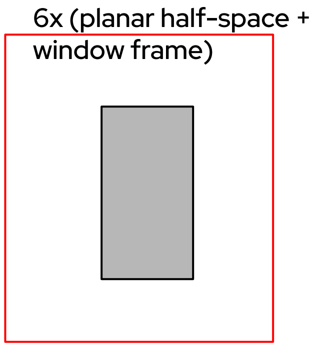
- ~~recursive calls, virtual functions,~~ less complicated algorithms ➔lower register and stack usage➔higher occupancy on GPU?
- reduced complexity per surface ➔lower divergence?
- uncoalesced memory accesses ➔high latency (intrinsic to geometry)
- State is known by navigation, no pushes required, enables potential use of mixed precision

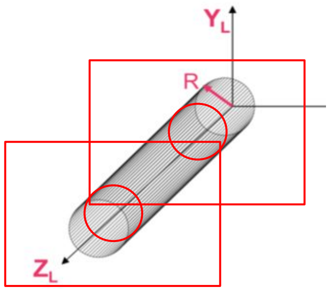


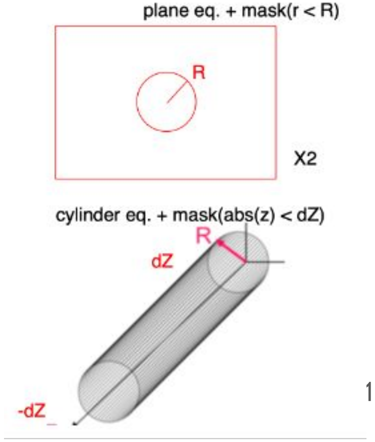
VecGeom uses a bounded surface approach

- 3D bodies represented as Boolean operation of half-space CommonSurfaces
  - First and second order, infinite
  - Just intersections for convex primitives
    e.g. box = h_0 & h_1 & h_2 & h_3 & h_4 & h_5
- Storing the solid imprint (frame) as a bounded FramedSurface







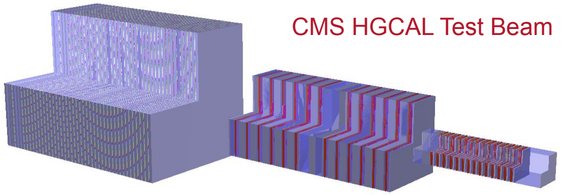


All solids supported

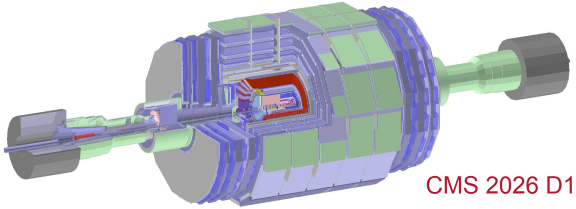
Conversion time and memory footprint under control

	# touchables [million]	conversion time [s]	memory [MB]
cms_2018	2.1	5.1	307
cms_TB_HGCAL	0.06	0.8	51.4
cms_2026D110	13.1	59.8	673
LHCb_Upgrade	18.5	92.8	173
LHCb_ECal_HCal	18.4	0.8	6.7
ATLAS_EMEC	0.08	1.4	132

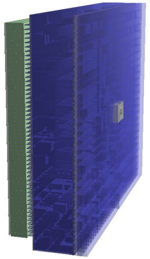
Correctness tested with randomized raytracing



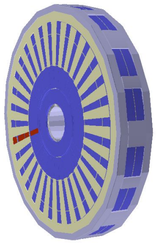
CMS HGCAL Test Beam



CMS 2026 D110



LHCb ECal HCal



ATLAS EMEC

BVH acceleration structure for factors of speedup

- **Bounding Volume Hierarchies** (BVH) are used to speed up collision detection within 3D objects. Number of checks scale with **Log(n)**
- VecGeom has a BVH navigation algorithm for the solids model
- Original BVH adapted using the bounding boxes of surfaces

	HGCAL Test Beam	LHCb Calorimeters	CMS 2026 D110
Looper	1.097 s	3.007 s	26.78 s
With BVH	0.226 s	1.006 s	2.60 s
Speedup	**4.9x**	**3.0x**	**10.3x**

Run time of ray tracing with 10M rays

Surface model still slightly slower than solid model

AdePT
HGCAL Test beam: 100 primary electrons with 10 GeV

Solid model: **2.62 s**
Surface model: **2.77 s**

LHCb calorimeters: 8 ttbar events

Solid model: **21.22 s**
Surface model: **26.25 s**

Why slower after all the advertisement?

Ongoing optimizations

Lower register usage?

AdePT is using a *single kernel* which is still at the maximum registers / thread

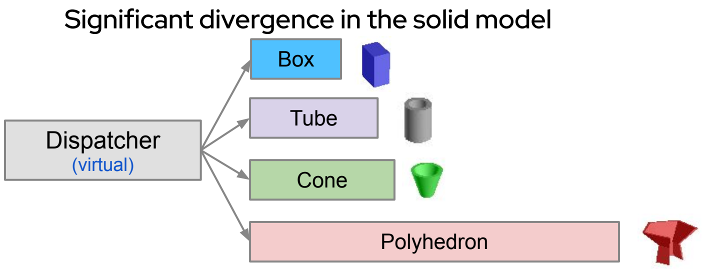
→ Kernel must be split (separate physics, geometry, magnetic field etc)

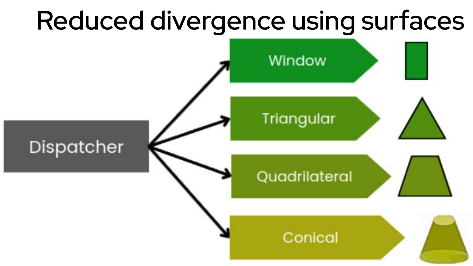
	FindNextVolume	Relocation	Max. Occupancy
Solids	256	220	16%
Surfaces (double)	146	142	25%
Surfaces (mixed)	123	122	33%

Mixed precision: **2x speedup** in HGCAL TestBeam over solid model in raytracing due to 128 registers/thread breakpoint (work in progress, not fully correct)

Ongoing optimizations

Lower divergence?

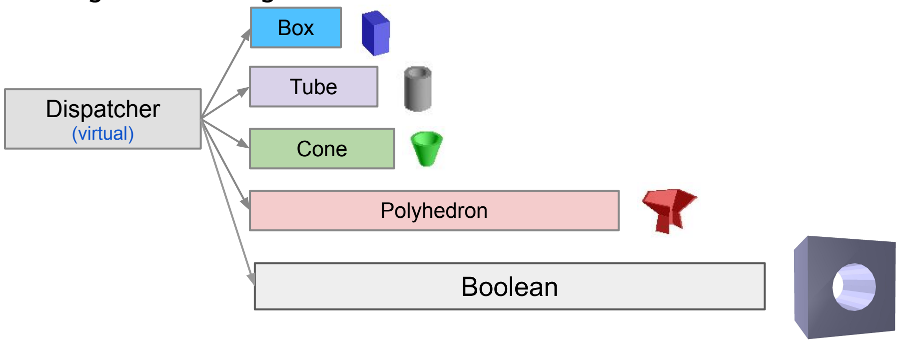




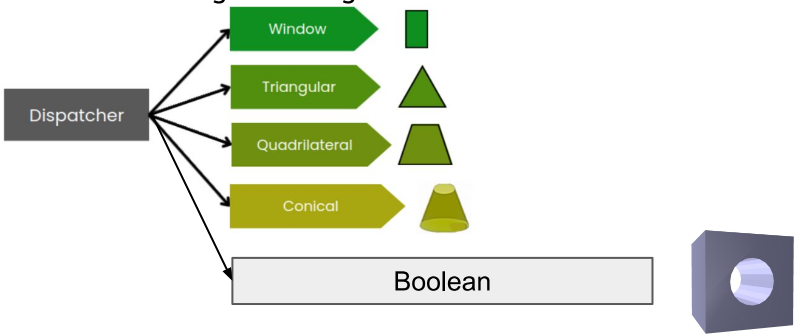
Ongoing optimizations

Lower divergence?

Significant divergence in the solid model



Reduced divergence using surfaces



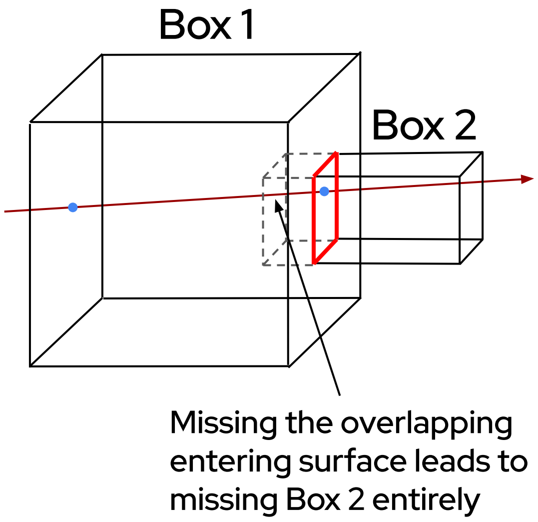
Boolean solids can have virtual surfaces → require full logic evaluation of all surfaces → expensive!

Solution: reduce number of boolean hits by marking “safe” surfaces at construction

Ongoing optimizations

Lower divergence?

- **Overlaps** in the geometry lead to wrong results
- Correctness achieved with overlap detection + relocation
- Relocation expensive & source of divergence



Other sources of divergence need to be further evaluated (different BVH tree, different amount of frames per surface)

Ongoing optimizations

Memory accesses pattern?

Surface model seems to do worse than solid model:

- Memory access random in both cases but solid model seems to do more compute per access
- Surface model seems to use only a small fraction of the 32 bytes per memory transaction

Low level solutions:

- improve memory read per memory transaction,
- improve compute per memory read (recomputing over storing data)

Summary and outlook

AdePT can run complex setups and has been validated.

Two major bottlenecks were identified:

1. **Kernel scheduling**

- Asynchronous kernel scheduling very promising, further optimization in progress
- Currently testing kernel split to increase occupancy

2. **Geometry**

- VecGeom surface model has drastically improved over the last month, still slightly slower than solid model.
- Promising avenue with mixed precision
- Many other optimizations underway