AdePT - Enabling GPU electromagnetic transport with Geant4

John Apostolakis[1], *Severin* Diederichs[1], *Andrei* Gheata[1], *Juan* González Caminero[1,*], *Stephan* Hageboeck[1], *Jonas* Hahnfeld[1,2], *Benjamin* Morgan[3], and *Witold* Pokorski[1]

[1]CERN, Geneva (CH)
[2]Goethe University Frankfurt (DE)
[3]University of Warwick (GB)

Abstract. Increasing simulation throughput is a major challenge for LHC experiments as they undergo significant detector upgrades for the high-luminosity phase. GPU-enabled particle transport simulation is a key R&D direction to address this, leveraging the growing availability of GPUs in computing centers. In its first phase, the AdePT project demonstrated that particle transport codes can be adapted for GPUs, integrated into a standard Geant4 workflow, and deliver significant speed-ups for standalone Geant4 setups of varying complexity. The second phase focuses on enabling seamless and efficient GPU usage within experiment frameworks via a Geant4 plugin. To achieve this, GPU transport kernels have been restructured into a header library hidden from the users, exposing only a configurable integration library easy to interface from diverse Geant4 applications. Several performance limitations identified in the first phase have been partially addressed. CPU-GPU scheduling has been improved to process multiple events on the GPU while allowing the CPU to perform asynchronous tasks. In addition, we continued the development of a new GPU-friendly surface-based geometry model, which mitigates some of the geometry-related bottlenecks. The initial integration of AdePT with two experiment frameworks has revealed challenges that will be addressed moving forward. Here, we present the latest results and insights, focusing on the hybrid Geant4-AdePT use case.

1 Introduction and motivation

The computational demands of simulation workflows in the experiments conducted at the Large Hadron Collider (LHC) are growing fast with the advance towards the high-luminosity era. Traditional CPU-based approaches struggle to keep up with the increasing demand for throughput. Graphics Processing Units (GPUs) present a compelling solution due to their massively parallel architecture, which is well-suited for the independent and repetitive nature of particle transport calculations. Moreover, the growing availability of GPUs in modern computing centers, including large-scale high-performance computing (HPC) clusters and cloud infrastructures, makes GPU-accelerated simulation an increasingly viable and scalable

*e-mail: juan.gonzalez.caminero@cern.ch

© The Authors, published by EDP Sciences. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (https://creativecommons.org/licenses/by/4.0/).

approach. Leveraging GPUs for simulation has the potential to improve the simulation efficiency and reduce computational costs, encouraging the deployment of dedicated research and development projects such as AdePT [1, 2] and Celeritas [3, 4].

The AdePT project started in late 2020 to demonstrate a complete Geant4 [5] simulation workflow working on GPUs. Existing CPU simulation components have been adapted for use on GPU. The geometry engine uses the VecGeom [6] library, which offers a 3D solids-based geometry modeler that can be used on both CPU and GPU. Given the complexity and GPU inefficiency of the VecGeom solid modeling, we started a couple of years ago the development of a GPU-friendly surface-based modeler, which is now very promising and close to its pre-release [7]. The G4HepEm [8] library is used for modeling the electromagnetic (EM) interactions of electrons, positrons, and gamma particles on the device. The numerical integration of charged particles in a magnetic field is implemented using a helix stepper for uniform fields and a Runge-Kutta stepper for arbitrary fields.

The project evolved from simple prototype examples demonstrating increasing functionality to more advanced ones showing a hybrid CPU-GPU workflow by integrating AdePT with Geant4 CPU simulation. In the current phase, AdePT has been refactored into a library incorporating the most advanced features from its initial implementation, allowing the development of an initial integration into the software frameworks of LHCb [9] and ATLAS [10] experiments.

This paper describes the latest integration features and the migration of the project into a library, the initial integration efforts into the ATLAS Athena [11] and LHCb Gaussino [12] frameworks, and the latest additions to the project addressing the largest bottlenecks of GPU simulation.

2 Current status

2.1 Recent developments

2.1.1 AdePT as a library

The initial AdePT examples demonstrated GPU-only simulation, which later evolved into the more complex examples integrating GPU transport with Geant4, scoring simple quantities such as energy deposition, to validate the physics output obtained in combined CPU-GPU workflows. While this approach offered great flexibility to implement changes and allowed examples to remain fully self-contained, it made integrating AdePT into external projects more challenging. By the end of this exploratory, prototype-driven phase, a satisfactory solution had been reached. To improve integration, we decided to refactor the latest examples into a library.

The current solution consists of two libraries: A header-only library containing all GPU transport-related components, and a precompiled library ‘AdePT_G4_integration‘ which contains the AdePT physics constructor, a specialized *G4VTrackingManager*, and utility functions needed by AdePT for configuration and initialization.

The AdePT header library is kept separated from any Geant4-related components and interacts only with an "IntegrationLayer" providing a certain interface. We provide a specific implementation of the integration layer for Geant4, but this design allows for extending the GPU support to other particle transport engines.

2.1.2 Geant4 integration method

To offload specific tracks to the GPU, the initial standalone AdePT prototypes were integrated into Geant4 simulations using the same mechanism as for invoking custom fast-simulation

code. Users must declare detector regions where tracks should be handled by a custom transport code. In AdePT, electrons, positrons, and gammas are buffered and eventually transported on the GPU. This mechanism proved sufficient for the initial integration tests, however, it was limited to a single detector region at a time. This was rather restrictive, and most importantly, prevented full-detector GPU transport, which is often the most performant configuration.

An alternative approach for integrating into Geant4 is to use a specialized *G4VTrackingManager*, which replaces the default Geant4 tracking loop with a custom one for selected particles. A custom tracking loop gives control over tracks throughout their entire lifetime, and allows fine-grained customization, as tracks can be processed based on their full state rather than only the detector region.

Currently, AdePT's specialized tracking manager keeps a list of GPU regions defined by the user and, whenever tracks enter these regions, they are injected into AdePT, either immediately or buffered, depending on the GPU scheduling strategy. Particles leaking from GPU regions are returned to the Geant4 stack to be transported on the CPU. The GPU transports tracks in parallel in *blocks* of threads, scheduled independently, leading to non-deterministic ordering of tracks in the output buffers. Leaking tracks need to be sorted before being copied back to the CPU to make the simulation reproducible in this scenario.

The G4HepEm library implements models describing all EM particle interactions relevant for HEP detector simulation. It is available on both CPU and GPU, making the physics identical when using the specialized G4HepEm tracking manager in Geant4, apart from gamma-nuclear and lepton-nuclear interactions that are not yet adopted in AdePT. AdePT also imports via G4HepEm all production and transport cuts set by the user, creating the conditions to move tracks between host and device without altering the physics. This allows moving tracks between the CPU and the GPU depending on resource availability, not been implemented so far due to reproducibility problems related to the random state not being currently shareable between host and device.

Integration into a Geant4 application is done via a specialized physics constructor *AdePTPhysics*, which registers the specialized AdePT tracking manager to electrons, positrons, and gammas. This makes adding AdePT to any existing Geant4 application straightforward, as the user only needs to add one line to their physics list.

2.1.3 Scoring approach

In particle transport codes, *scoring* refers to collecting and recording physical quantities of interest during simulation. In Geant4, scoring is defined in sensitive detector code, which is user-defined and called while stepping inside sensitive detectors. Reusing the CPU scoring code generically in GPU-enabled simulation would require porting this code to GPU, enabling user function callbacks during GPU transport, and allowing the user code to create, manage, and copy back to CPU arbitrary data structures. This is difficult to support in general due to resource and abstraction limitations on GPUs but is possibly the most efficient way to implement scoring in a full custom way.

To preserve scoring generality without imposing strict restrictions on the quantities to be recorded, AdePT takes a different approach: transferring the data for all steps made in sensitive volumes (hits) back to the host. GPU steps store all relevant information needed to create native Geant4 step objects on the host to be provided to the user scoring code via the standard Geant4 interface.

In this approach, no change to the user scoring needs to be made on the user side for the AdePT-enabled simulation. During transport, when a GPU thread steps a particle within a sensitive volume, it registers all available track information and stores it in a buffer. When

this buffer reaches a predefined threshold, it is copied back to the host, where Geant4 steps are reconstructed, and then each step is processed by the worker associated with the respective event as if the step had been produced during the CPU simulation.

The main advantage is that scoring works transparently for the user as in the CPU simulation, the limit being that custom information a user may associate with tracks during stepping cannot be propagated on the GPU without a custom implementation. We are currently working on understanding how to propagate custom information for a certain number of use cases, observed while integrating AdePT into experimental frameworks.

2.2 Integration with experiment frameworks

After the reorganization of the AdePT code into two libraries as described in the previous section, the integration with experimental frameworks has been largely simplified. AdePT can be activated by calling a specific constructor in the active physics list. This has been done, in particular, for the LHCb Gaussino simulation framework. In Gaussino, the Geant4 physics list is constructed dynamically based on a Python configuration file containing physics constructors to include. This is achieved through a specialized factory-based code implementation for each Geant4 physics constructor. We adopted the same approach for AdePT, including it by adding a single line to the Python configuration file. Additional AdePT parameters can be passed through standard Geant4 UI commands.

The AdePT integration layer is implemented such that the same sensitive detector code used for the Geant4 simulation is also called for the hits recorded by AdePT. The Geant4 logical volumes marked as "sensitive" are propagated to AdePT during the conversion from Geant4 to VecGeom geometry. This makes the integration of AdePT transparent to the Gaussino application once connected via the configuration file.

The AdePT-Gaussino integration was first tested with the Calo Challenge [13] setup, originally developed for testing machine learning-based fast simulation models. This setup implements a simplified calorimeter and produces several monitoring histograms to verify the correctness of the results. Running this setup with AdePT enabled inside Gaussino, showed good physics agreement with the Geant4-only simulation while providing a significant speedup.

The full LHCb detector simulation has been considered as the next step. The first issues we encountered and are currently addressed were related to the MC truth handling, where some of the 'user' actions were expecting particles having MC truth information attached. Apart from the MC truth handling, the LHCb simulation with AdePT can now run with the standard Gaussino code. Certain discrepancies in the number of hits remain to be understood after resolving some inconsistencies in propagating the user-defined cuts to AdePT.

Similar to the LHCb approach, we have started to work on integrating AdePT with the ATLAS Athena framework, in a common effort done with the Celeritas and ATLAS simulation teams. After successfully enabling GPU support and including AdePT and its dependencies in Athena, we were able to complete preliminary simulation runs involving the TileCal [14] detector. The work of integration and validation is currently ongoing.

2.3 Identified bottlenecks and ongoing improvements

2.3.1 Synchronous mode

Initially, AdePT could only be operated in a "synchronous" mode: During the simulation, Geant4 workers transport particles as usual. Whenever an EM track enters a GPU region, it is extracted from Geant4 and added to a buffer, while Geant4 continues transporting the

remaining tracks. When this buffer is full, the Geant4 worker stops tracking particles on the CPU and switches to the GPU transport loop, steering it until no more particles are left in flight. At this point, any particles that have leaked out of GPU regions are added back to the Geant4 stack and the simulation continues on the CPU.

Scoring is done on the CPU as described in section 2.1.3. This means that while steering the GPU transport loop, the Geant4 worker may need to process the scoring data produced on GPU, but this processing does not overlap with GPU transport.

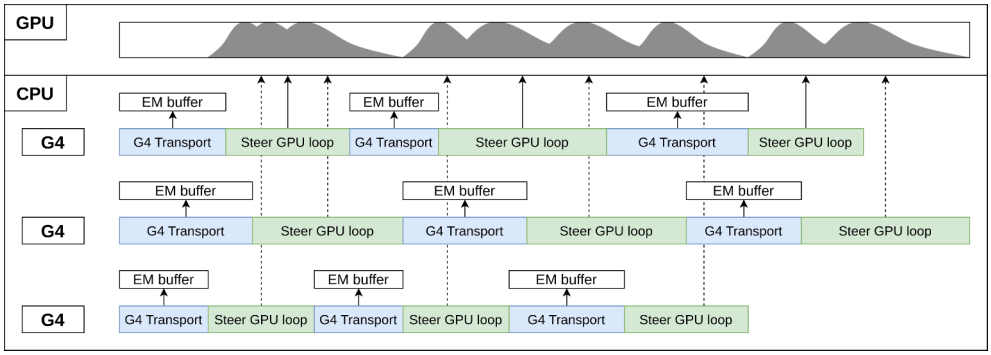

Figure 1. Synchronous AdePT workflow. The x-axis represents time, the y-axis shows the communications between Geant4 workers, and the track population on the GPU.

The main limitation of this model is that CPU and GPU transport are executed sequentially: In the transport loop, the Geant4 worker threads have to steer and complete the GPU transport loop before transporting hadrons or performing user actions on the host. If the number of Geant4 worker threads is relatively small, this inefficiency still allows for a significant gain because the GPU can transport the particles in the buffer faster than the CPU would. This changes when the number of CPU threads increases, providing many tracks and saturating the GPU. In this case, the GPU becomes a serious bottleneck, as the CPU is blocked for longer periods, reducing the speedup relative to Geant4 running on CPU-only in multi-threaded mode.

There are other sources of inefficiency in synchronous transport, as a consequence of injecting bunches of tracks to be transported together. As shown in Figure 1, when a thread triggers GPU transport, the shower size starts small, increases to a maximum, and then it has a long tail until all tracks are finished. A single CPU thread is only able to fill the GPU when the track population is above a certain threshold, which means that the GPU is under-utilized during tails. This is mitigated when more CPU threads are running, but GPU utilization is still highly dependent on when each thread is doing GPU transport. Since threads don't do CPU simulation while steering the GPU loop, no new particles can be injected, and the shower size depends only on the initial number of tracks.

In addition, each Geant4 worker maintains their separate buffers of particles to be transported on the GPU, scoring data, and has access to a portion of the space allocated for tracks in GPU memory. As events have different needs at different points in their execution, this can mean that at times a thread may be memory-starved while others have plenty of unused space.

2.3.2 Asynchronous mode

To address the synchronous execution bottleneck, a new mode of operation has been developed: In the "Asynchronous" mode, a dedicated CPU thread manages the GPU transport loop, freeing the Geant4 worker threads from directly interacting with GPU transport kernels. Instead, the workers simply transfer tracks to a shared buffer for offloading and immediately resume the CPU-based transport.

At the start of each GPU transport loop iteration, the dedicated thread fetches the available new tracks from the CPU buffer and adds them to the active particle pool on the device. The GPU loop runs continuously as long as there are particles to be transported, without blocking the Geant4 CPU workers. Since tracks can exit GPU regions, the GPU transport thread is responsible for transferring their states back to the CPU, reconstructing them as Geant4 tracks, and passing them to the appropriate workers for further processing.

As shown in Figure 2, Geant4 workers no longer interrupt CPU-based transport to launch GPU transport kernels. Owing to the asynchronous mode, the Geant4 worker threads continuously generate new EM particles, which are added to the buffer and injected into the GPU by the GPU transport thread, helping maintain a high usage of the device.

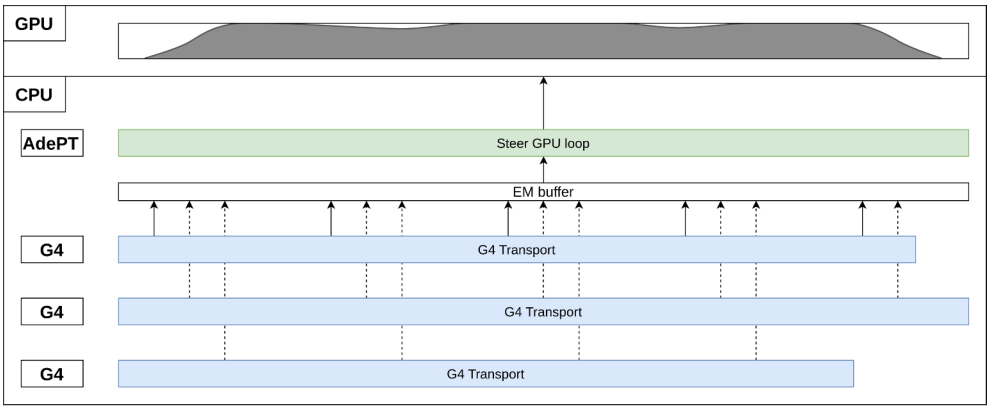

Figure 2. Asynchronous AdePT workflow. The x-axis represents time, the y-axis shows the communications between Geant4 workers, and the track population on the GPU.

The asynchronous mode adds complexity by mixing and processing particles from different events, requiring scoring data and leaked tracks to be sent to the appropriate CPU threads. In addition, different events can have different workloads and be at different stages of execution. For example, all tracks of one particular event may have finished while other events still have plenty in flight, but this should not prevent the thread in charge of the finished event from moving on to the next.

To prevent overloading the GPU transport thread, a dedicated AdePT "scoring management" thread is used, which takes care of the transfer of the scoring data from the device to the host memory, and subsequently hands the scoring data to their respective Geant4 CPU worker threads.

To manage the lifetime of different events, a per-event state machine is maintained, allowing the different events to be processed independently. Continuing with the previous example, if all particles of an event have been transported, a flush of the scoring data is requested by the GPU transport thread, enabling the Geant4 worker to finish the event as quickly as possible.

The asynchronous mode achieves better performance than the synchronous mode, as shown in Fig. 3, despite the usage of additional resources in the form of two extra threads. The main advantage comes from the Geant4 workers regaining CPU time, as they no longer need to steer the GPU transport loop. The performance gain is more pronounced when using a higher number of CPU threads.

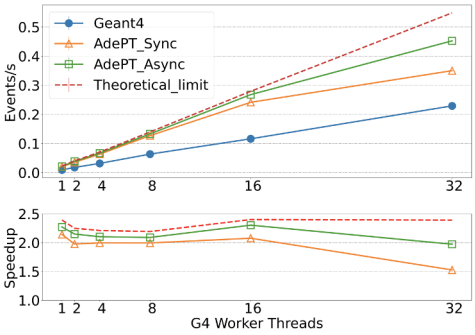

Figure 3. Throughput comparison in events per second of the different modes of AdePT and Geant4 in the CMS2018 geometry using TTbar events.

2.3.3 VecGeom's surface model

VecGeom's surface model was originally presented at CHEP 2023 [7]. It aims to accelerate the geometry modeling in the particle transport simulation by adopting a bounded surface representation of the geometry.

The main performance limitations of VecGeom's solid model on the GPU stem from high register usage and thread divergence. Both contribute to low *achieved occupancy*, which is defined as the ratio of active warps per multiprocessor to the maximum possible active warps.

As shown in table 1, using the surface model in AdePT largely reduces the register usage when running smaller transport kernels, which increases the *theoretical occupancy* in Nvidia GPUs (defined as the highest possible fraction of active warps per multiprocessor, limited by the register and shared memory usage). For reference, the monolithic kernels always require the maximum amount of registers available, regardless of the geometry model used.

This increase in theoretical occupancy, however, does not translate into an increase in achieved occupancy, which remains similar to that obtained with the solids model. The main reason for this is the large amounts of data accessed by each thread, leading to long delays between instructions. An improved memory model is currently being developed to overcome this limitation.

3 Summary and outlook

Recent developments in AdePT, most importantly the transition from the initial prototype to a library, have simplified integration into existing Geant4 applications. The new approach has been successfully used for the initial integrations into the ATLAS Athena and LHCb Gaussino frameworks. Further work for completing and validating these integrations needs to be done in collaboration with experiments, to identify missing functionality and adapt the current implementation.

Kernel	Solids model	Surface model
ElectronHowFar	220 (16.6%)	118 (33.3%)
ElectronPropagation	254 (16.6%)	204 (16.6%)
ElectronMSC	220 (16.6%)	127 (33.3%)
ElectronRelocation	220 (16.6%)	144 (25%)
ElectronInteractions	118 (33.3%)	118 (33.3%)

Table 1. Register usage for the split AdePT kernels with the solid and surface geometry models. The number in parentheses shows the theoretical occupancy of these kernels in an Nvidia RTX 4080 GPU.

The two major identified bottlenecks have been partially addressed: A new asynchronous transport mode provides better performance than the initial approach, improving the usage of both CPU and GPU and mitigating the issue of device saturation. The new VecGeom surface model is already available for GPU simulation, achieving currently comparable performance to the solids model. Further work is needed to make the surface model more beneficial for GPU simulations.

Work is now ongoing on several fronts: Optimizing the current magnetic field stepper for non-uniform magnetic fields and adding the ability to use arbitrary 3D field maps on device, implementing modifications needed to use the AdePT G4VTrackingManager for full detector transport in ATLAS Athena, identifying and fixing the sources of discrepancies with Geant4 in the LHCb Gaussino framework, and developing a new memory model for improving the performance of the split transport kernels.

References

[1] G Amadio et al., Offloading electromagnetic shower transport to GPUs, J. Phys.: Conf. Ser. **2438** 012055 (2023)
[2] AdePT https://github.com/apt-sim/AdePT
[3] Celeritas https://github.com/celeritas-project/celeritas
[4] S C Tognini et al., Celeritas: GPU-accelerated particle transport for detector simulation in High Energy Physics experiments, arXiv:2203.09467 [physics.data-an] (2022)
[5] Agostinelli S et al., Geant4 — a simulation toolkit, NIM-A **506** 250–303, (2003)
[6] J Apostolakis et al., Towards a high performance geometry library for particle-detector simulations, J. Phys.: Conf. Ser. **608** 012023 (2015)
[7] J Apostolakis et al., Surface-based GPU-friendly geometry modeling for detector simulation, EPJ Web of Conferences **295**, 03039 (2024)
[8] G4HepEm https://github.com/mnovak42/g4hepem
[9] The LHCb Collaboration et al, The LHCb Detector at the LHC. (2008) JINST 3 S08005.
[10] The ATLAS Collaboration et al, The ATLAS Experiment at the CERN Large Hadron Collider . (2008) JINST 3 S08003.
[11] ATLAS Collaboration. (2021). Athena (21.0.127). Zenodo.
[12] M Mazurek et al., Gauss and Gaussino: the LHCb simulation software and its new experiment agnostic core framework, PoS ICHEP2022 **225** (2022)
[13] CaloChallenge code repository https://github.com/CaloChallenge/homepage
[14] TileCal Collaboration, The atlas tile calorimeter project. Nucl. Phys. B Proc. Suppl. **44** 82-87, (1995)
[15] HGCALTB https://github.com/geant-val/HGCALTB