ACAT 2021

AdePT
Accelerated Particle Transport

Offloading electromagnetic shower transport to GPUs

The AdePT project

Guilherme Amadio (CERN), John Apostolakis (CERN), Predrag Buncic (CERN), Gabriele Cosmo (CERN), Daniel Dosaru (EPFL), Andrei Gheata (CERN), Stephan Hageboeck (CERN), Jonas Hahnfeld (CERN), Mark Hodgkinson (Sheffield University), Benjamin Morgan (Warwick University), Mihaly Novak (CERN), Adrian Antonio Petre (ISS Bucharest), Witold Pokorski (CERN), Alberto Ribon (CERN), Graeme A Stewart (CERN), Pere Mato Vila (CERN)

Nov 30, 2021

andrei.gheata@cern.ch

The challenge

- Using GPU cards for detailed simulation of collider experiments
  - Running (part of) the simulation workflow on accelerators
  - Prototype realistic use-case(s) to understand GPU usability
- Focus only on e^+, e^- and γ shower simulation
  - Relevant hotspot for detector simulation
  - Involves all simulation components: physics, geometry, propagation, scoring
    - Adapt/rewrite for GPU
- Support for both GPU standalone and hybrid modes
  - Stealing from Geant4 CPU workload and offloading to GPU
  - Understand the hard limits for functionality and performance
- Bootstrapping SIMT awareness for simulation components & workflow
  - Get (fast) to a vantage point to understand how to evolve the software
  - One year project roadmap

The AdePT project

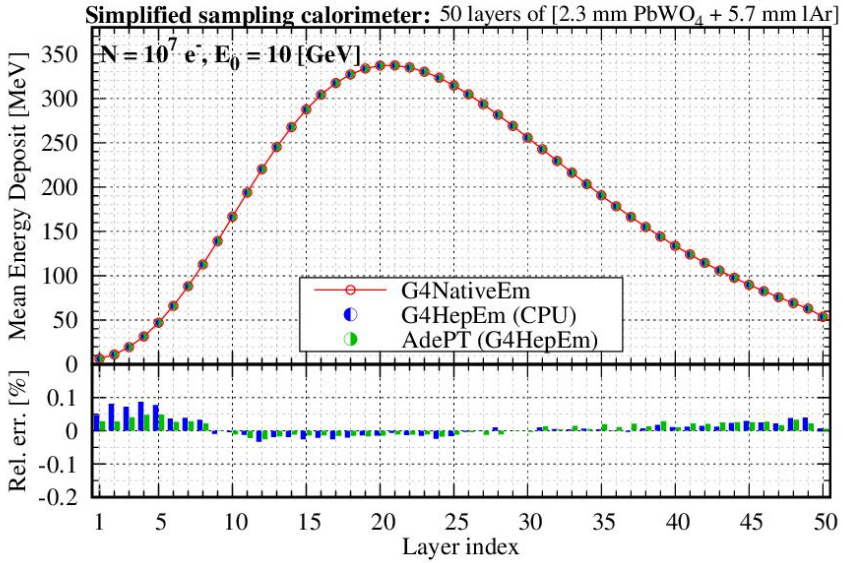
- GitHub [repository]
  - Started in September 2020, 11 contributors so far
- Strategy: evolve to a prototype based on gradually more complex examples
  - Postponed portability, converged to using CUDA directly for fast prototyping
  - Increase example complexity from Fisher-Price toy models to LHC detectors
  - Reuse the stable helpers and interfaces while evolving the examples
- Adapt or re-implement simulation components to be GPU friendly
  - External libraries for geometry: ***VecGeom*** and physics: ***G4HepEm***
  - Keep minimal infrastructure to allow building-up GPU workflows
    - Atomics, containers, error handling macros
- Validation against Geant4 for the realistic setups

GPU-friendly rewrite of EM physics

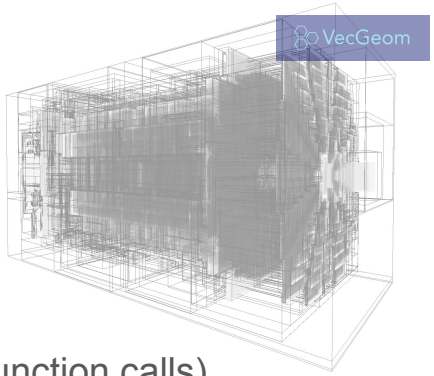
- **G4HepEm**: compact library of EM processes for HEP
  - Covers all interactions of e^-, e^+ and γ, including multiple scattering
  - Initialization of physics tables dependent on Geant4, but usage on GPU standalone and lightweight
- Design of library very supportive for heterogeneous simulations
  - Interfaces: standalone functions without global state
  - Data: physics tables and other data structures copied to GPUs
  - Reusing > 95% of the code from G4HepEm for GPU shower simulation
- For details, see this presentation at the last Geant4 Collaboration meeting

G4HepEm validation

- Previously validated simulation results on GPU against Geant4
  - Agreement at per-mill level in the mean energy deposit (shown below) and other quantities (number of secondaries, number of steps, charged track length)



GPU geometry: VecGeom

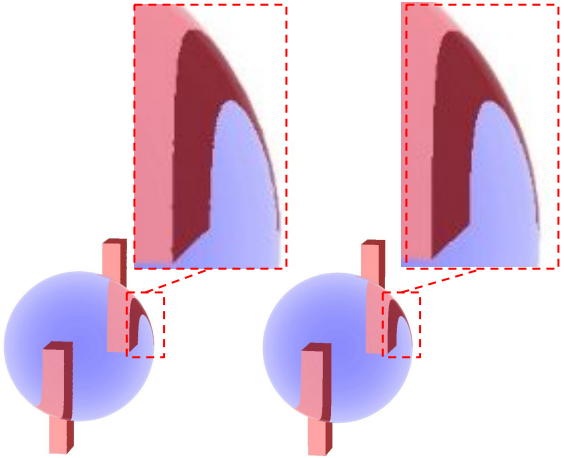


- First implementation of GPU support few years old
  - C++ types re-compiled using ***nvcc*** in a separate namespace/library
    - Transient model and interfaces reused > 90% (including virtual function calls)
  - Global navigation layer customized for GPU use
- Improving gradually GPU support
  - Developed custom optimised navigation state, single-precision support
  - Moving from a simple looper to an optimized **BVH navigator**
  - Adopting modern CMake GPU support
- Moving forward: specializing the VecGeom GPU navigation support
  - Investigating surface models as alternative to solid primitives
  - Important step towards portability and performance

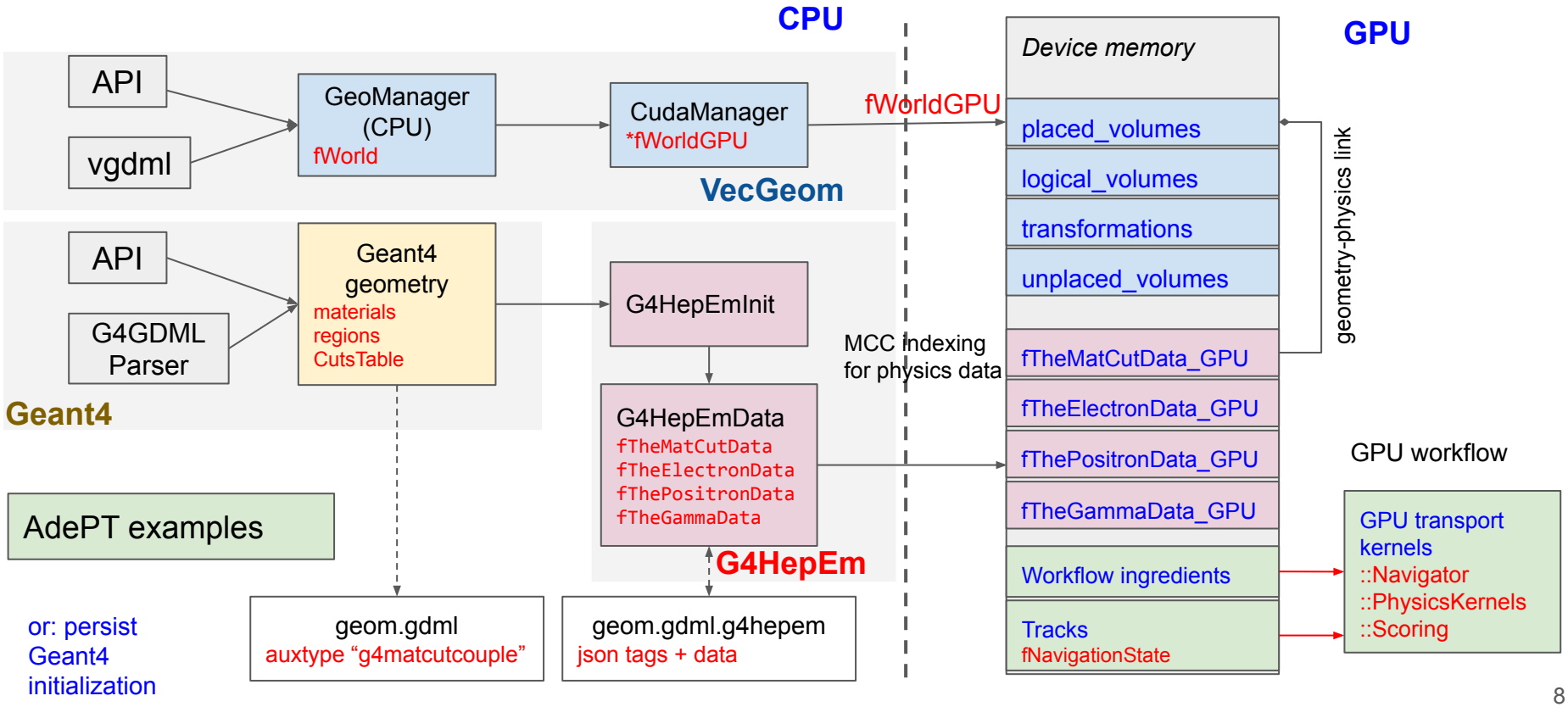
Single-precision geometry - less memory access

- Tested impact on performance in the AdePT examples
  - After doing several fixes for importing VecGeom precision type and using it in the navigators
    - LoopNavigator (simple looper for daughters), and BVHNavigator
- RaytraceBenchmark example (using BVHNavigator)
  - Reading a GDML file and modeling reflections/refractions and specularity
  - Validated by the output image
    - Very simple geometry: 7% faster
    - Complex geometry (trackML): 44% GPU, 14% **CPU!**
- Physics-enabled GPU examples
  - Tracker geometry + loop navigator: 280% faster
  - Tracker geometry + optimized navigator: 30% faster
  - To fix: differences at % level with double-precision

RTX 2080 SUPER

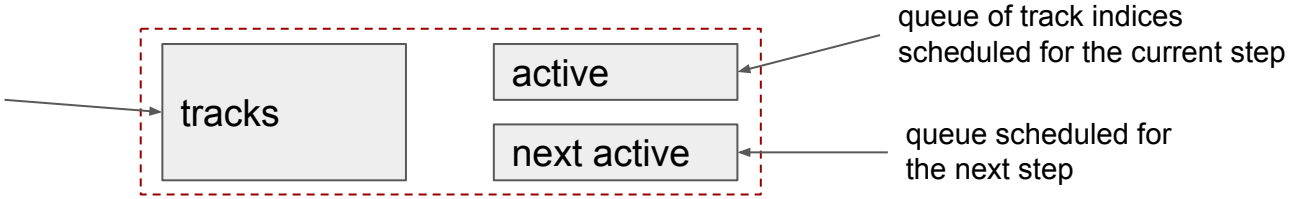


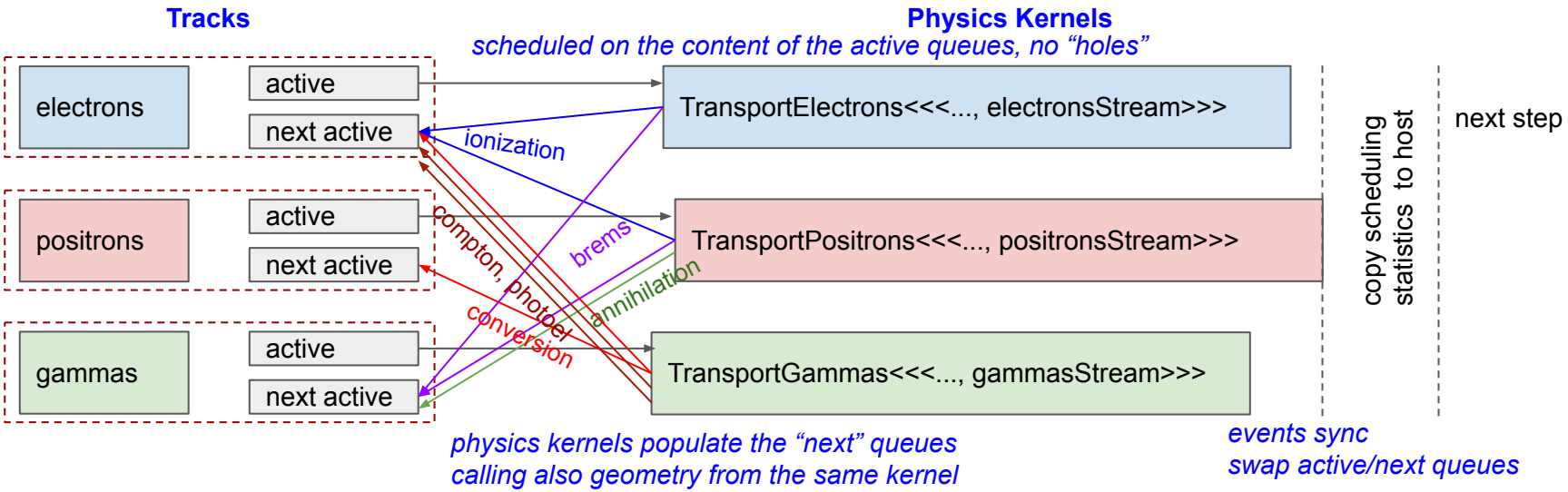
The AdePT “cookbook”



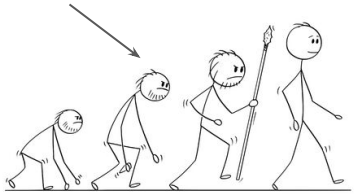
Stepping loop workflow v1.0

- contiguous
- next slot atomic
- no slot reuse
- different for e+/e-/gamma





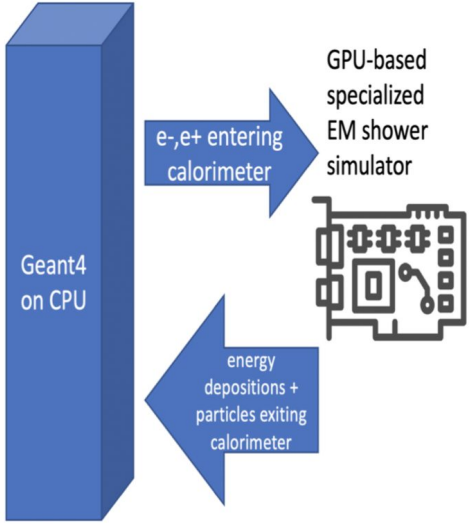
Evolved examples

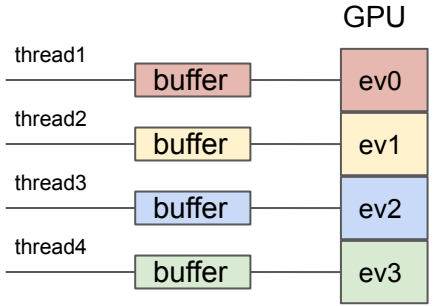


- Sampling calorimeter (TestEm3)
  - Constant field on/off, Physics = ALL, Geant4 and GPU versions
  - Similar throughput for Geant4 MT on AMD Ryzen 9 3900 (12C/24T), versus AdePT on GeForce RTX 2070 SUPER, without magnetic field
  - MT example shows limited benefits in the current workflow due to lock contention in the CUDA runtime
- Arbitrary geometry with scoring per volume (Example13)
  - Field on/off, Physics = ALL, AdePT version only for now
  - Doing initialization of Geant4, G4HepEm and VecGeom data structures based on GDML info
  - Generic scoring in all volumes, in the future only for sensitive volumes
  - Example recipe now adopted in the Geant4 integration prototype

AdePT-Geant4 integration

- AdePT integration using fast simulation hooks in Geant4
  - Intercept, buffer and shower on GPU all EM particles entering the calorimeter, recover leaked particles and hits
  - Flush the AdePT buffer until the event content is exhausted
- Synchronous processing per batch of tracks
  - After flushing the AdePT buffer, results have to come back to CPU before continuing
  - One AdePT pipeline per Geant4 CPU thread, partitioned GPU track data
  - Tracks from different events are not mixed in the same buffer
- An evolving workflow implementation
  - The batches per event may not be large enough to fill the GPU
  - Mixing buffers from many threads (event mixing) may be needed





GPU Performance: high-level view

Example13, cms_2018 setup / no field / RTX2070

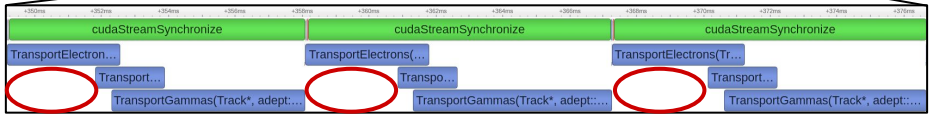
≈30s

20 particles per batch

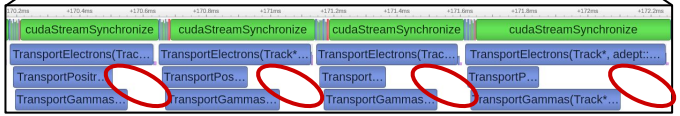
≈7.5s

200 particles per batch

Already ≈4x faster, but another 2x seems feasible



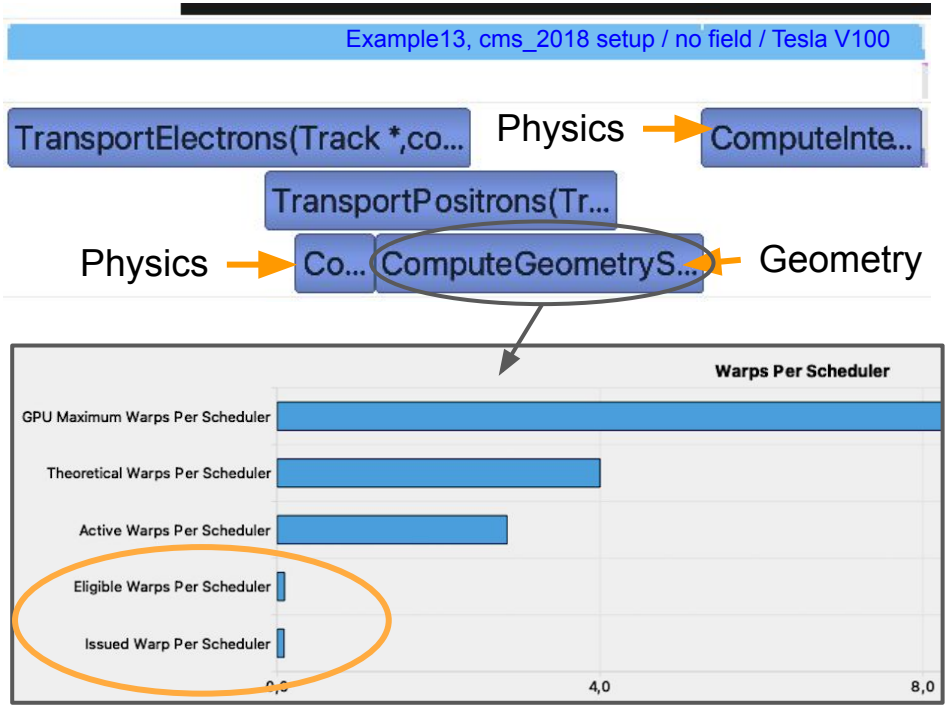
GPU is busy, so it cannot start new kernels until it has processed some warps from the current running kernel.

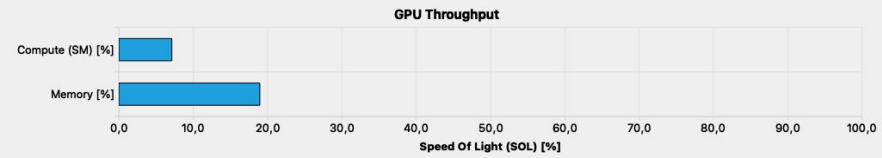


GPU does not have enough work, can start all kernels at once and has idle time between steps.

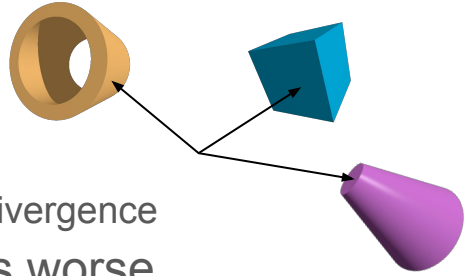
GPU Performance: low-level view

- Running time roughly equal for geometry and physics (step length + interactions)
- Here: profile geometry kernel
- Theoretically 4 warps / scheduler due to resource usage, 2.8 active on avg
- **But: 0.1 warps eligible for execution**
  - Memory latency of detector description
  - Thread divergence when navigating
- **GPU issues 1 instruction / 34 cycles**
- Considerable headroom for more performance, but this requires a new geometry description





Performance: an initial insight



- GPU is very hungry
  - Needs many (at least 10^5) tracks in parallel for high efficiency
- Current code seems still too complex for the GPU
  - Deep stacks use too many registers → low occupancy, high thread divergence
- Long tails pointing to load unbalancing - in mag field it gets worse
- Memory access patterns and thread divergence are performance blockers
  - Tracks need to run different code paths: traverse BVH, check distance to volumes
  - Tracks have different physics interactions (e.g. Compton scatt., pair prod., photoelectric eff.)
- Big performance gains yet to be reaped
  - Initial performance analysis insights give reasonable hope that speedups of several factors are still feasible

Performance portability

- oneAdePT - port to oneAPI of an AdePT snapshot
  - core utilities, magnetic field, RNG, G4HepEM
  - Attempt to use legacy CUDA code compiled in VecGeom
- Many obstacles for migrating CUDA to DPC++ code
  - SYCL limitations in calling virtual functions or function pointers, non-const globals, support for std:: math functions, support for CUDA compiled libraries, documentation
- Triggered investigations and work in VecGeom
  - Non-virtual dispatch and CUDA compilation using clang, deeper restructuring needed
  - Specializing geometry for GPU needed for both portability and better performance

Overlook

- Converging to a prototype combining complete particle transport features
  - Most functionality demonstrated in examples, standalone and combined CPU-GPU workflows
- R&D work for improving GPU friendliness
  - Re-written physics, GPU-aware geometry, optimized navigation
- A first look on performance shows large potential for improvement
  - Identified bottlenecks related to workflow, code complexity, work imbalance
- We can answer the question “can it work?”, now assessing the “how efficient?” part
  - Only a realistic full prototype can give us an unbiased measure of performance
  - **Early 2022:** HSF meeting to plan next steps and seek convergence of efforts on a common project