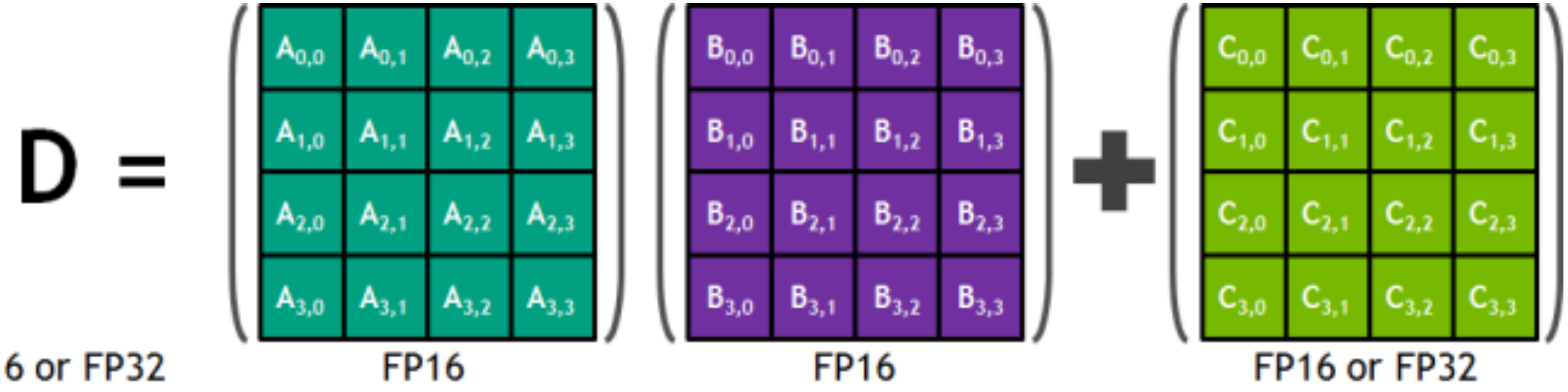
Justin Salmon
Prof Simon McIntosh-Smith
HPC research group
University of Bristol
@simonmcs

Exploiting Hardware-Accelerated Ray Tracing for Monte Carlo Particle Transport with OpenMC

Increased heterogeneity is an important response to the slowing of Moore’s Law

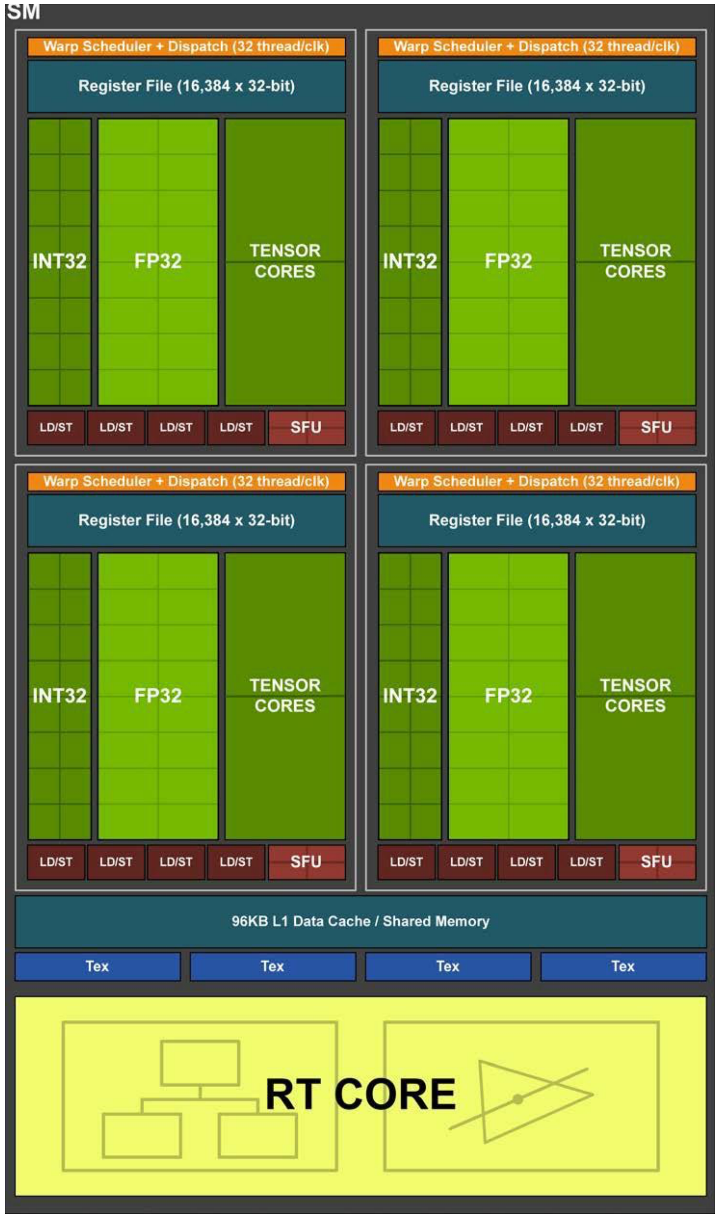
- Expect to see more “application-oriented” optimisations
  - Matrix multiply units in SIMD instruction sets (AVX, SVE)
  - Floating point formats optimized for deep learning (BFLOAT16)

- Important recent example: TensorCores

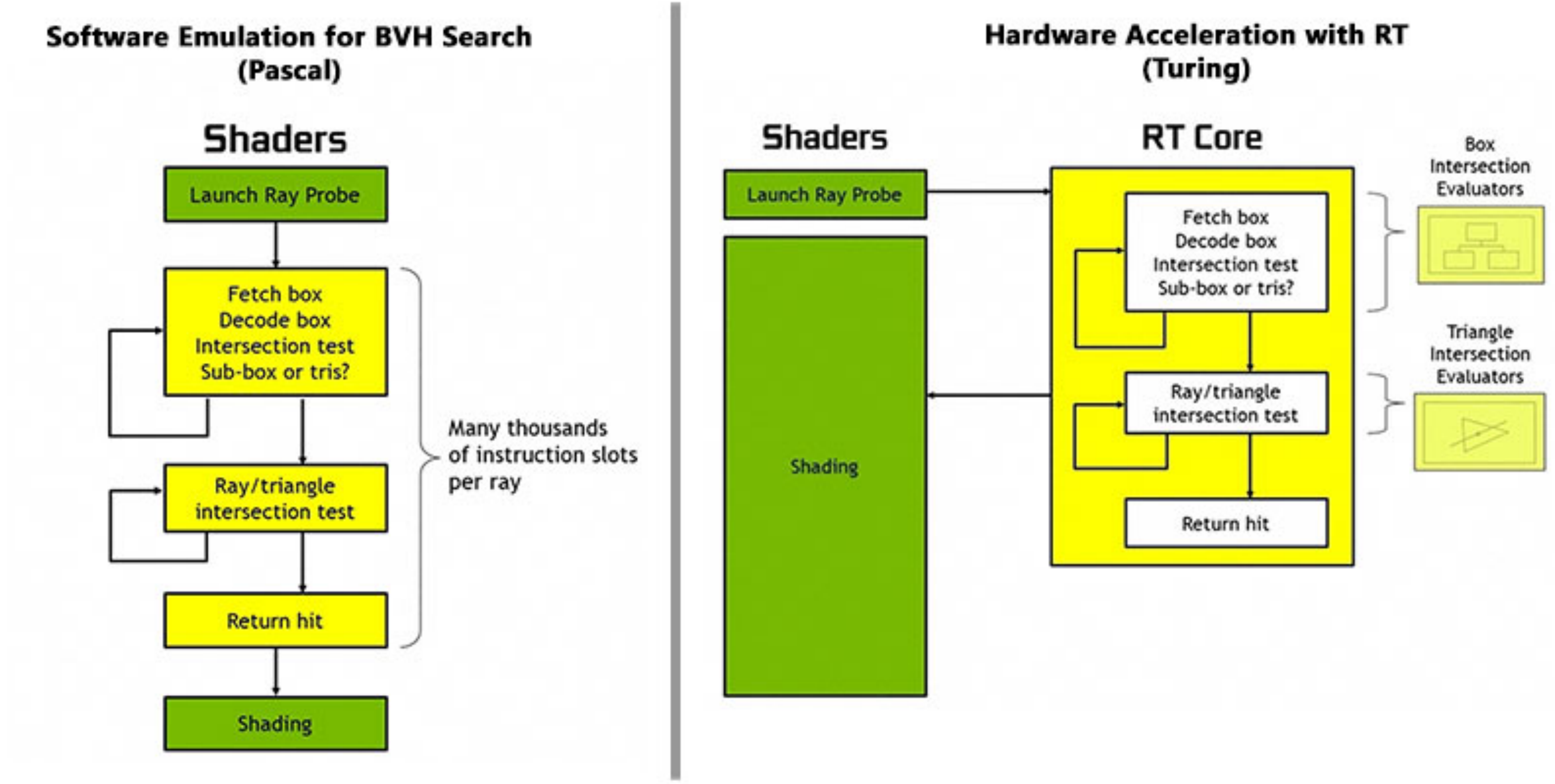


Ray Tracing cores

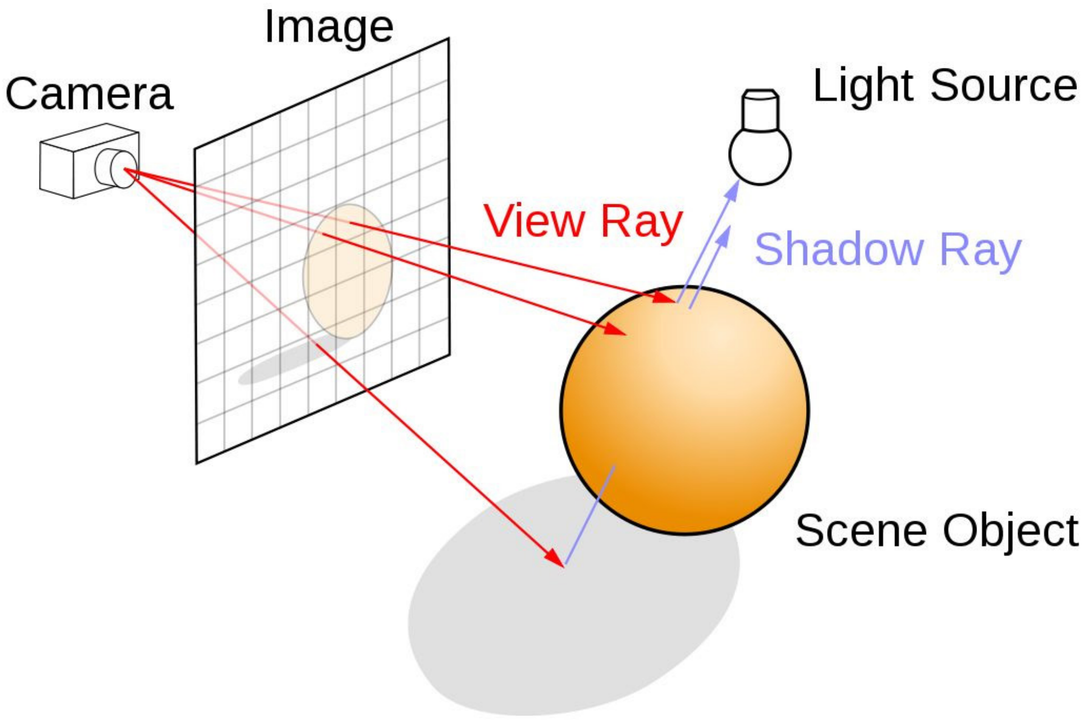
- NVIDIA's latest architectural innovation (Turing-class GPUs)
- Designed to accelerate the ray tracing algorithms used in graphical rendering in games, rather than for HPC
- Potential speedups of up to 10X vs CUDA code on same GPU
- Accelerates ray / surface intersection calculations
  - 10 GigaRays/s on RT cores vs 1-2 GigaRays/s in CUDA on the same GPU

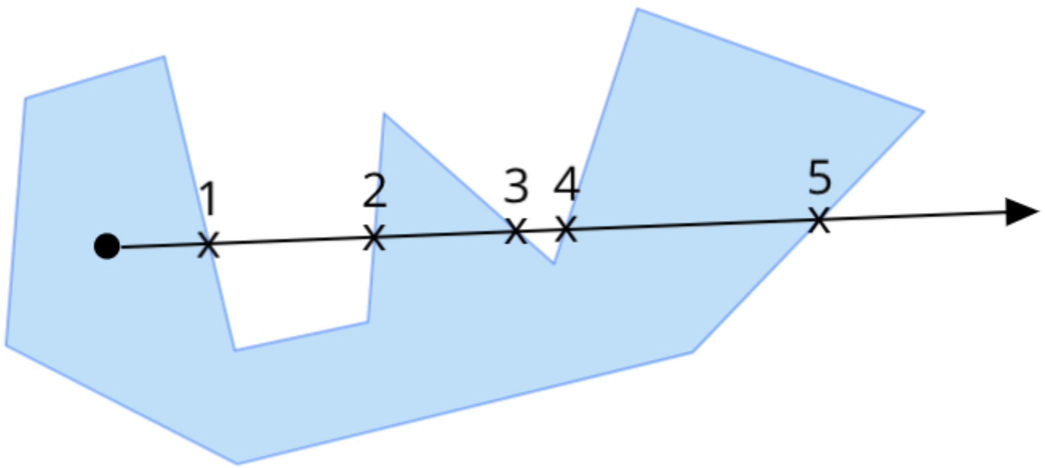


Which parts of ray tracing do the RT cores accelerate?



Observation: Monte Carlo particle transport has similarities to RT





Both require large numbers of linear geometric queries
to be executed over complex 3D geometric models

Monte Carle particle transport

- MC particle transport has applications in fission and fusion reactor design, radiography, and accelerator design
- Requires large numbers of particles >= $O(10^6)$, therefore computationally expensive
- Many codes and mini-apps developed for MC particle transport:
  - OpenMC, MCNP, Quicksilver, Branson, **neutral [1]**
- We've focused on **OpenMC** for this work

University of BRISTOL

[1] M. Martineau and S. McIntosh-Smith. Exploring on-node parallelism with neutral, a Monte Carlo neutral particle transport mini-app. In Cluster Computing (CLUSTER), 2017 IEEE International Conference on, 2017.

ASIMOV

OpenMC

- A Monte Carlo particle transport code focused on neutron criticality simulations, recently developed in the Computational Reactor Physics Group at MIT [1]
- Modern C++
- Being evaluated by the UK Atomic Energy Authority (UKAEA) as a tool for simulating the ITER nuclear fusion reactor [2]
- CPUs only, using OpenMP for on-node parallelism and MPI for inter-node parallelism

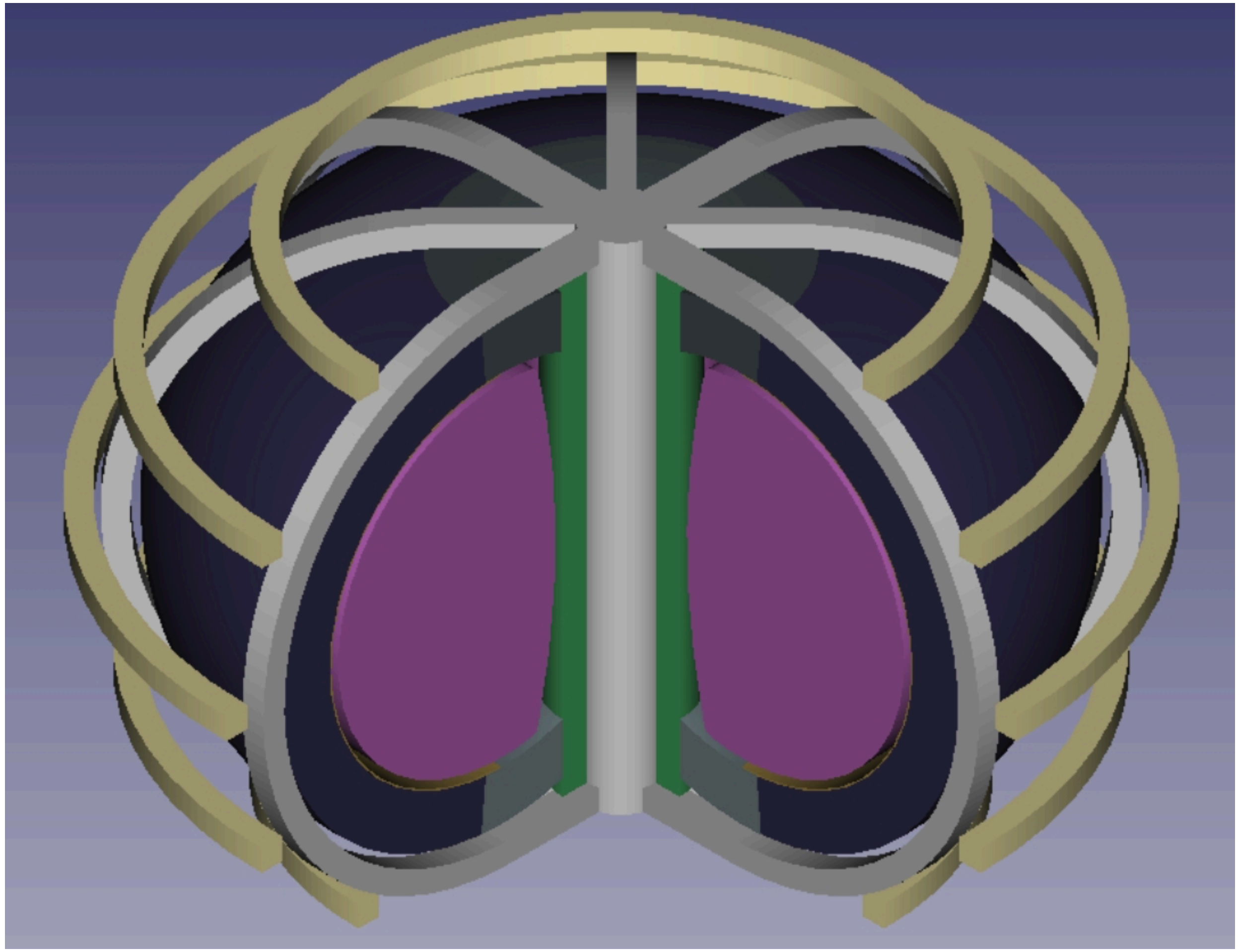
[1] P. K. Romano and B. Forget, "The OpenMC Monte Carlo particle transport code," *Annals of Nuclear Energy*, vol. 51, pp. 274–281, 2013.

[2] A. Turner, "Investigations into alternative radiation transport codes for ITER neutronics analysis," in Transactions of the American Nuclear Society, 2017.

University of BRISTOL

ASIMOV

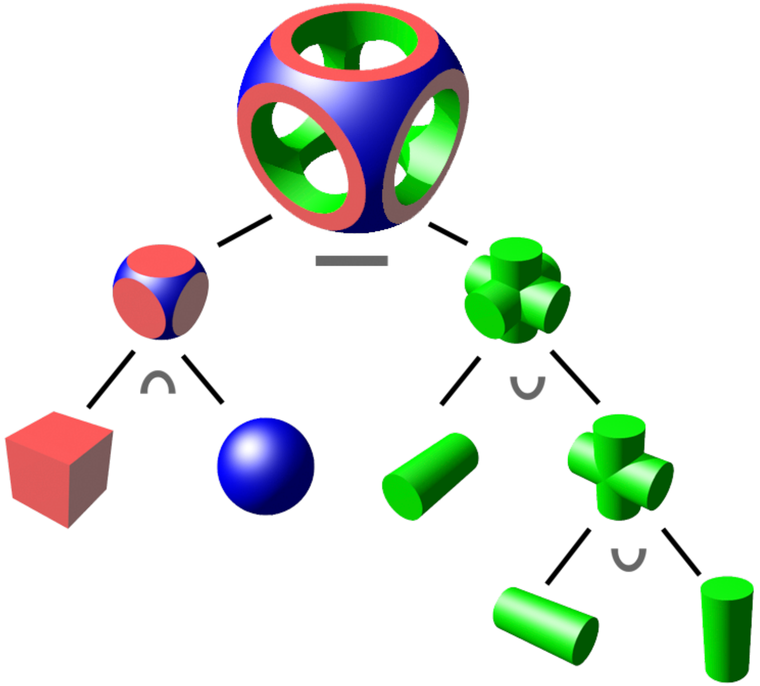
Motivation – fusion reactor design

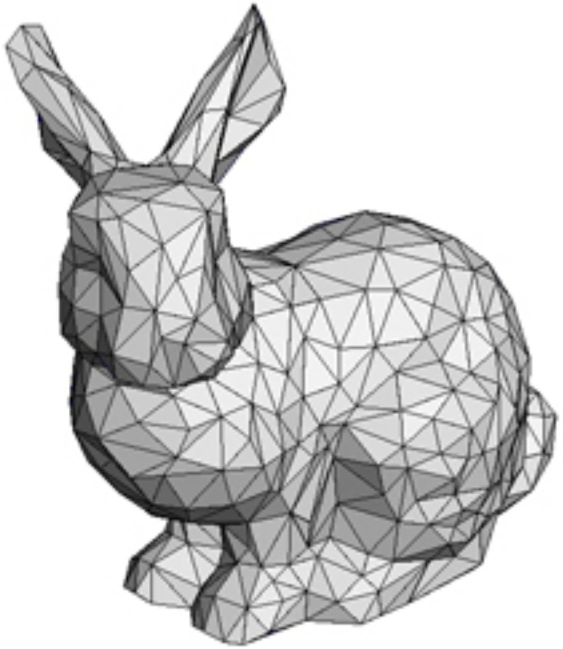


- Tokamak model from UKAEA
- CAD model
- $O(10^8)$ triangles in mesh
- O(10) GBytes of data

Generating the input geometries

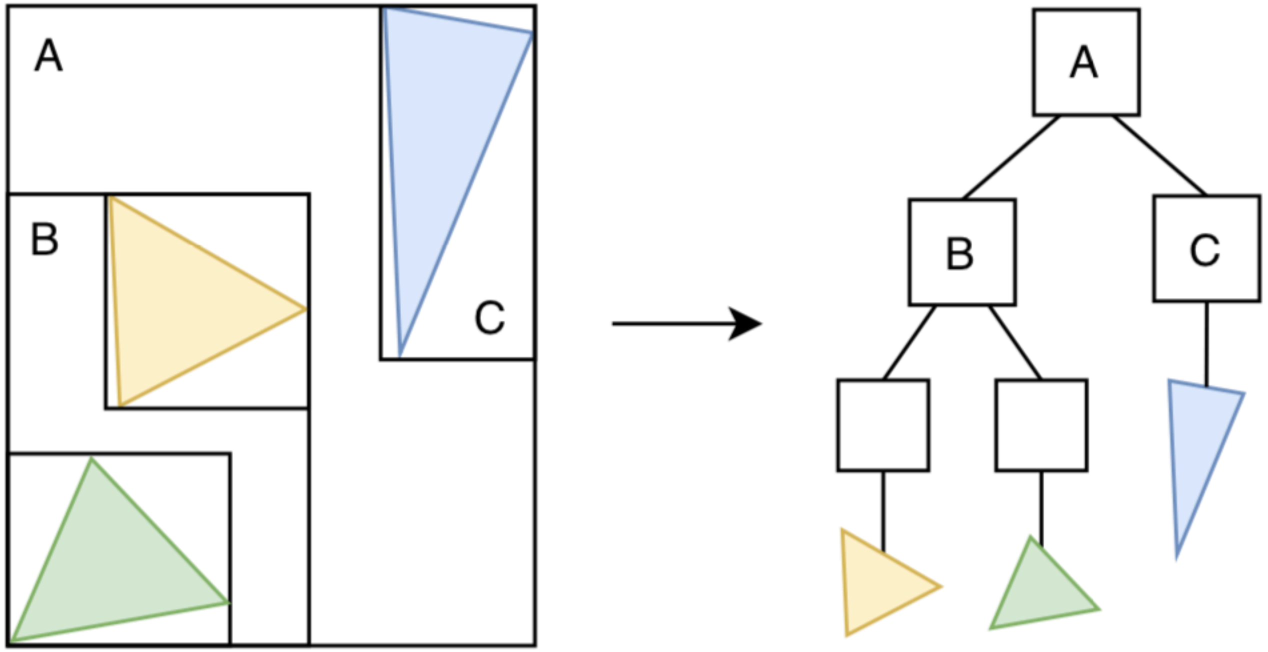
- Can be Constructive Solid Geometry (CSG) or meshes of many small triangles from CAD tools





Acceleration structures

- For large models, finding the intersection points is expensive
- Acceleration structures use a hierarchy of progressively smaller bounding boxes around model sub-regions
- These boxes are then tested for intersection in a binary tree style search, massively reducing the number of surfaces that need to be tested
- E.g. Bounding Volume Hierarchy (BVH) trees, Octrees and Kd- trees



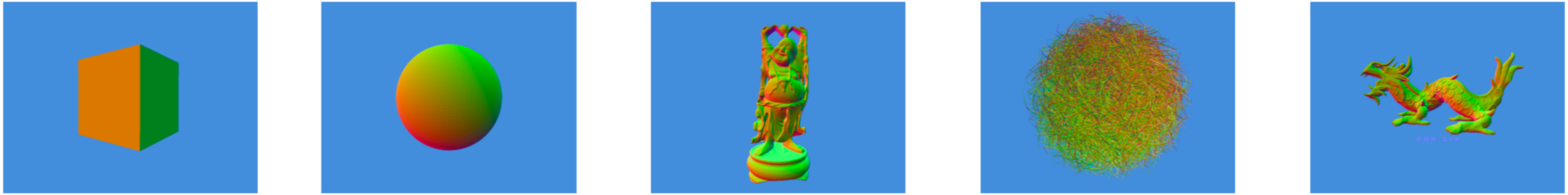
How the RT cores work

- Each SM on the GPU has access to its own RT core to which it can issue “ray probe” requests
- Each RT core has triangle intersection and BVH traversal units
- Can cache triangle vertex and BVH tree data
- The two units in the RT core execute the ray probe asynchronously, writing the result back to an SM register once complete. The SM can perform other work in parallel.

Implementation

- It's not yet possible to program the RT cores directly via CUDA et al
- Have to use a library: **NVIDIA's OptiX™** ray tracing library, or Vulkan, Microsoft DXR, …
- In OptiX, the user provides a set of CUDA-like kernel programs as PTX strings, each of which performs a specific function in the ray tracing pipeline
  - E.g. generating rays, handling intersections or handling rays which miss the geometry entirely
- These programs are then compiled on-the-fly by OptiX and woven into a single "mega kernel"
- OptiX then handles scheduling of kernel launches internally, automatically balancing load across the GPU

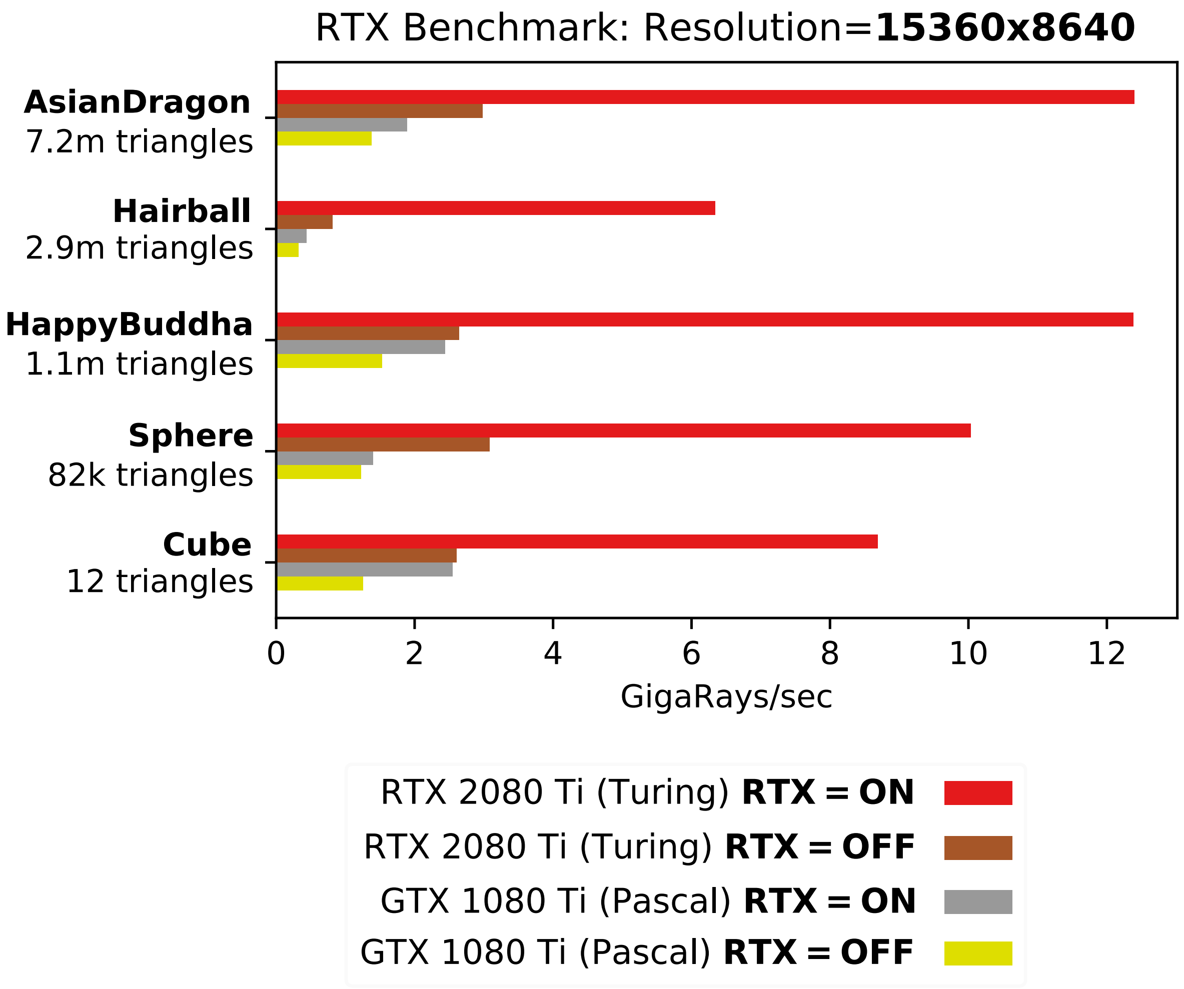
Benchmarking RT cores for raytracing



(a) Cube, 12 (b) Sphere, 89k (c) Happy Buddha, 1.1m (d) Hairball, 2.9m (e) Asian Dragon, 7.2m

- Developed a simple benchmark to evaluate the raw ray tracing performance of RT cores
- Renders frames of a 3D triangle mesh scene as fast as possible
- Each thread handles a single ray and writes the computed pixel colour to an output buffer, which is then interpreted as an image
- Five 3D models were selected to use as rendering targets, from a trivial 12 triangles, to over 7 million triangles.

Raytracing speedup using the RT cores and OptiX



- 4.6X speedup on average for the Turing GPU
- Over 12 GigaRays/sec for the Happy Buddha and Asian Dragon models
- The Hairball model is the most geometrically complex, achieves an 11.8X speedup.

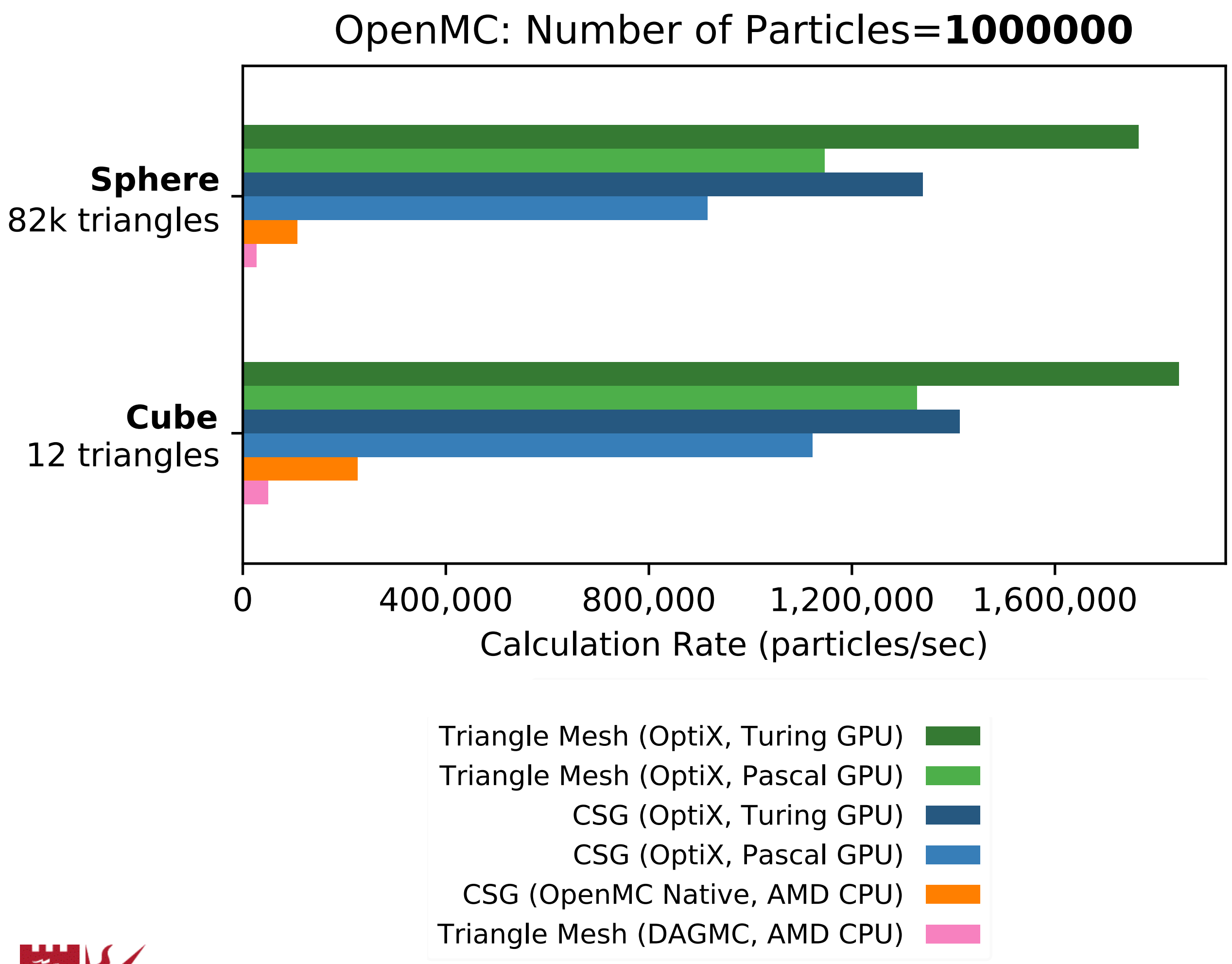
Porting OpenMC to GPUs

- Ported the main kernel to CUDA and to OptiX for comparisons
- In the absence of a real fusion reactor model, we used the same five models from the RT benchmark
- Each model is filled with a fissionable material (^{235}U) and is surrounded by a bounding cube filled with a vacuum
- Particles are terminated if they hit the edge of the bounding cube
- The particle source is set inside the model
- Each model was simulated for 2 generations using 2 batches of N particles, where N ranges from 10^3 up to 10^7.

Methodology

- Ran native OpenMC on a 16-core AMD Ryzen 7 2700 CPU using GCC 7.4
  - Only using the Cube and Sphere models as needed to be CSG for CPU
- Also used the same two GPUs as before: RTX 2080 Ti (Turing) and GTX 1080 Ti (Pascal)
- We collected the particle calculation rate (measured in particles/s) and wallclock time spent in particle transport.

CUDA OpenMC results on simple geometries



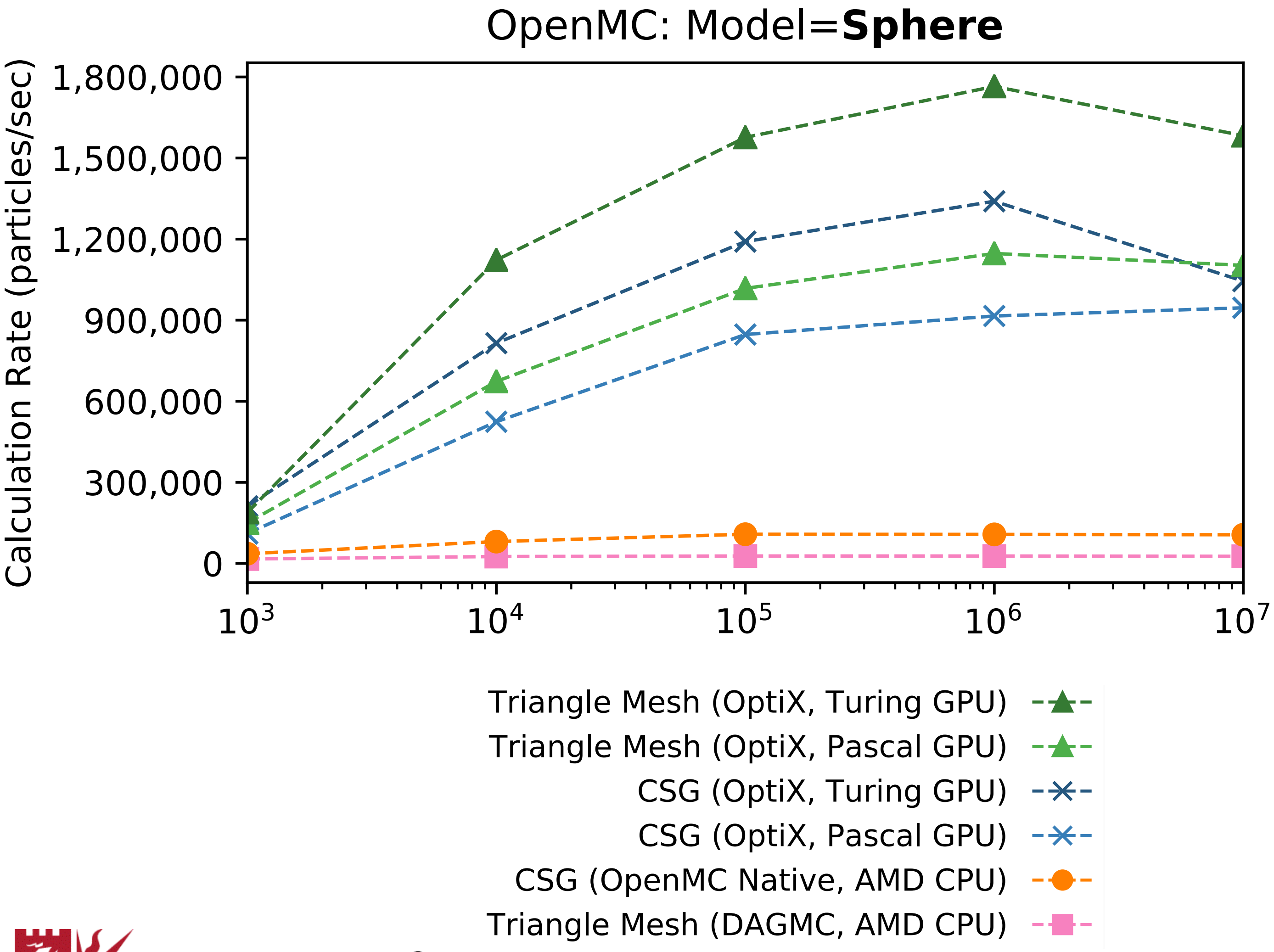
- GPU version ~13X the native CPU version
- Turing ~1.4X faster than Pascal

University of BRISTOL

https://uob-hpc.github.io/

ASIMOV

Increasing particle counts



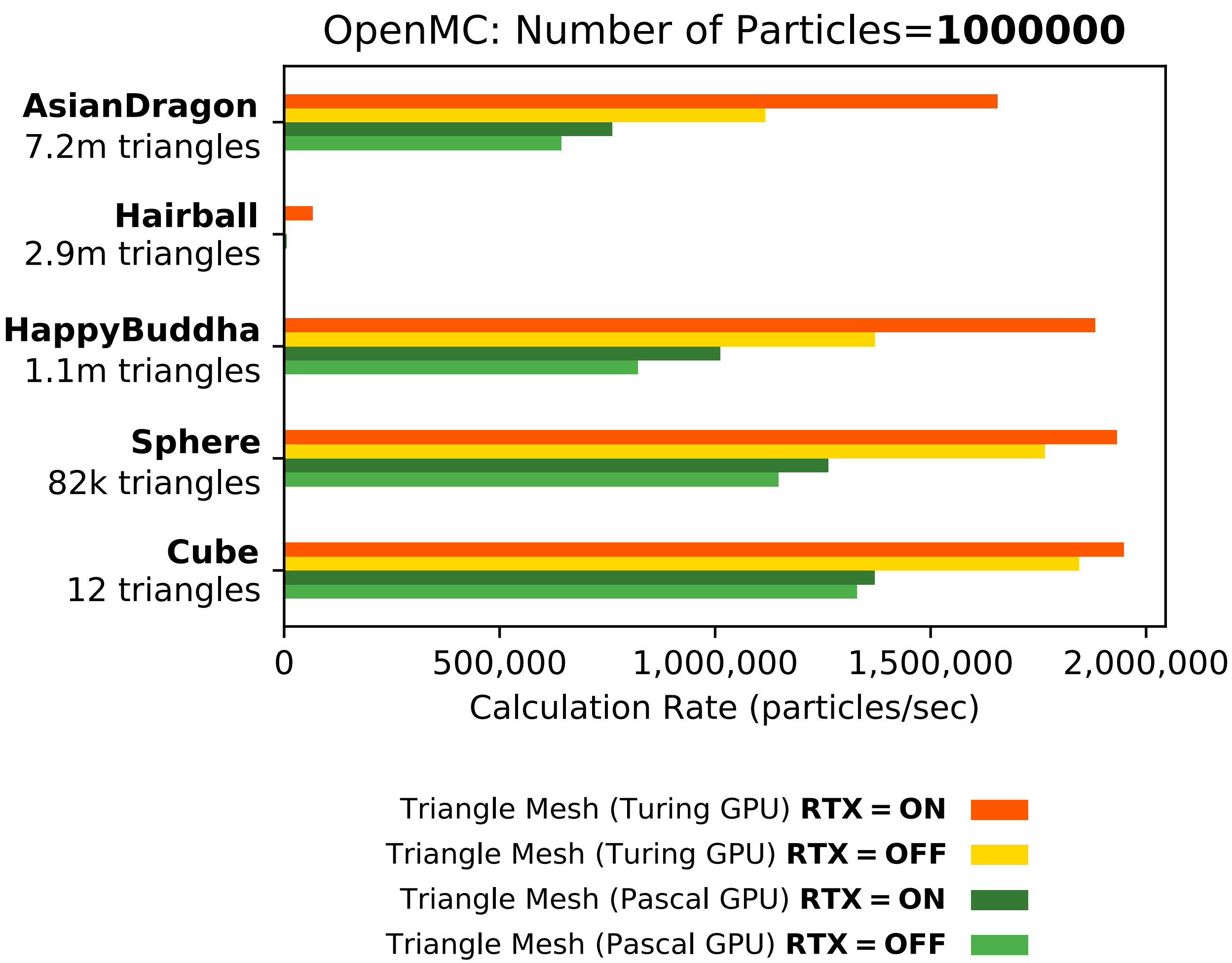
- Peak speedup is 16.4X over the CPU
- Peak GPU performance at 1M particles

University of BRISTOL

https://uob-hpc.github.io/

ASIMOV

RT (OptiX) OpenMC results



- RTX mode on the Turing GPU is the fastest in all cases, being 30-50% faster on the larger geometries
- The Hairball model shows the biggest difference, being 20.1x faster with RTX mode on Turing.

Conclusions

- Monte Carlo-based particle transport can port well to GPUs
- Ray tracing hardware holds promise for accelerating this application, with results from 1.3X to 20X over CUDA alone
- Currently hard to program these cores – have to go through a graphics API to do it
- AMD, Intel, Arm and others are also adding RT hardware
- Potentially other uses of RT hardware that can be explored

For more information

M. Martineau and S. McIntosh-Smith. Exploring on-node parallelism with neutral, a Monte Carlo neutral particle transport mini-app. In Cluster Computing (CLUSTER), 2017 IEEE International Conference on, 2017. DOI: 10.1109/CLUSTER.2017.83

On the Porting and Optimisation of Physics Simulations for Heterogeneous Parallel Processors. Matt Martineau, PhD thesis, University of Bristol, January 2019.

Bristol HPC group: https://uob-hpc.github.io/

Isambard: http://gw4.ac.uk/isambard/

Acknowledgements: ASiMoV is funded by EPSRC, grant no. EP/S005072/1.