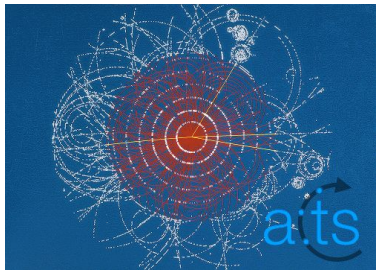


Summary of Thursday Session:

R&D Development

Nick Styles

ACTS Workshop Closeout

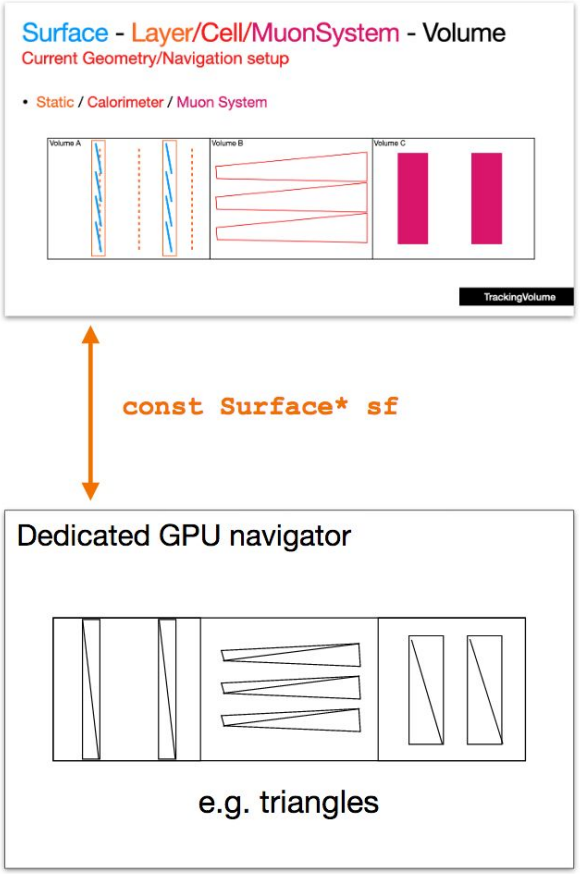
29/05/2020

Introduction

- While Wednesday session focused largely on core-functionality developments, Thursday was for R&D developments. This means both
  - Developments that make ACTS more useful as a testbench for new approaches
  - And longer-term plans about how ACTS itself may develop
- Two main prongs envisaged for this
  - New computing architectures; how to support them and how to make best use of them eventually
  - New algorithmic approaches; e.g. Machine Learning-based tools, and any other things outside the normal tried-and-tested workhorses
- Session ended up focusing exclusively on the first topic
  - Nevertheless, was a very useful and illuminating discussion
  - Many thanks to all who participated!

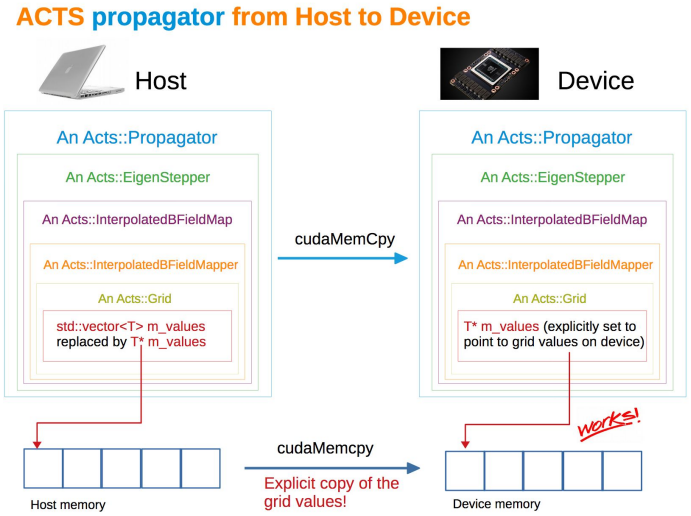
Geometry on GPUs

- Andi presented some ideas for how to represent detector geometries natively on GPUs
  - Use triangles to build up any and all surfaces
  - Dealing with large numbers of triangles is the “bread and butter” of GPUs since this is how computer graphics are typically represented
- Comparisons made with GeantV
  - There are some similarities primarily with the VecGeom that was a ‘side product’ of GeantV development
- Existing tools for navigating with such meshes may be available
  - Some APIs etc, may already be available that can avoid re-inventing the wheel for this
- Discussion on ray-tracing approaches
  - Such geometry representations allow this, but there is still much to think about
  - Leads nicely into next topic...



CUDA Propagation Example

- Xiaocong shared her experiences of porting the ACTS extrapolator to CUDA
  - Required re-thinking the way Acts::Grid was stored
  - Once done this showed some encouraging speed-ups for many threads with >30k extrapolations
  - NB so far only parameter propagation



- Discussion on use of “Managed Memory”
  - Largely a programming convenience that does copying of memory “behind the scenes” without needing to do it explicitly
  - Not guaranteed to be most performant solution, especially for large numbers of threads
- Discussion on best ideas for memory management
  - Optimal solution may be “pre-categorization” into groups of similar tracks (e.g. similar pT, number of hits, etc) that can be efficiently processed by a kernel
  - Important point to address is the synchronization (CUDA handles this if sticking to that world, but anything more general will need its own solution)

Including CUDA, etc, in code base

- Inspired by pull request for inclusion of CUDA seeding by Beomki Yeo
  - Initiated discussion on best approach for inclusion of such code
  - Currently handled by a template flag
- Interesting point raised: Will CUDA code ever want to be run on CPUs?
  - Changes made to allow algorithms to run in CUDA have in some cases been seen to also be more efficient for CPUs too
- CMake can also be used more cleverly to make such inclusions easier
  - CMake native CUDA support is rather good by now
  - Different compilation units can make the problem more tractable
- At this point my internet connection dropped out so I will leave it to others to summarise further points I missed ;-)
  - The major drawback for me so far of holding a workshop remotely...
  - ...aside from the lack of a social workshop dinner of course!