Extending GeantV to accelerators

S.Vallecorsa for the GeantV team

Outline:

- Motivation
- Geometry performance
- Physics
- Running the whole example
- Summary and plans

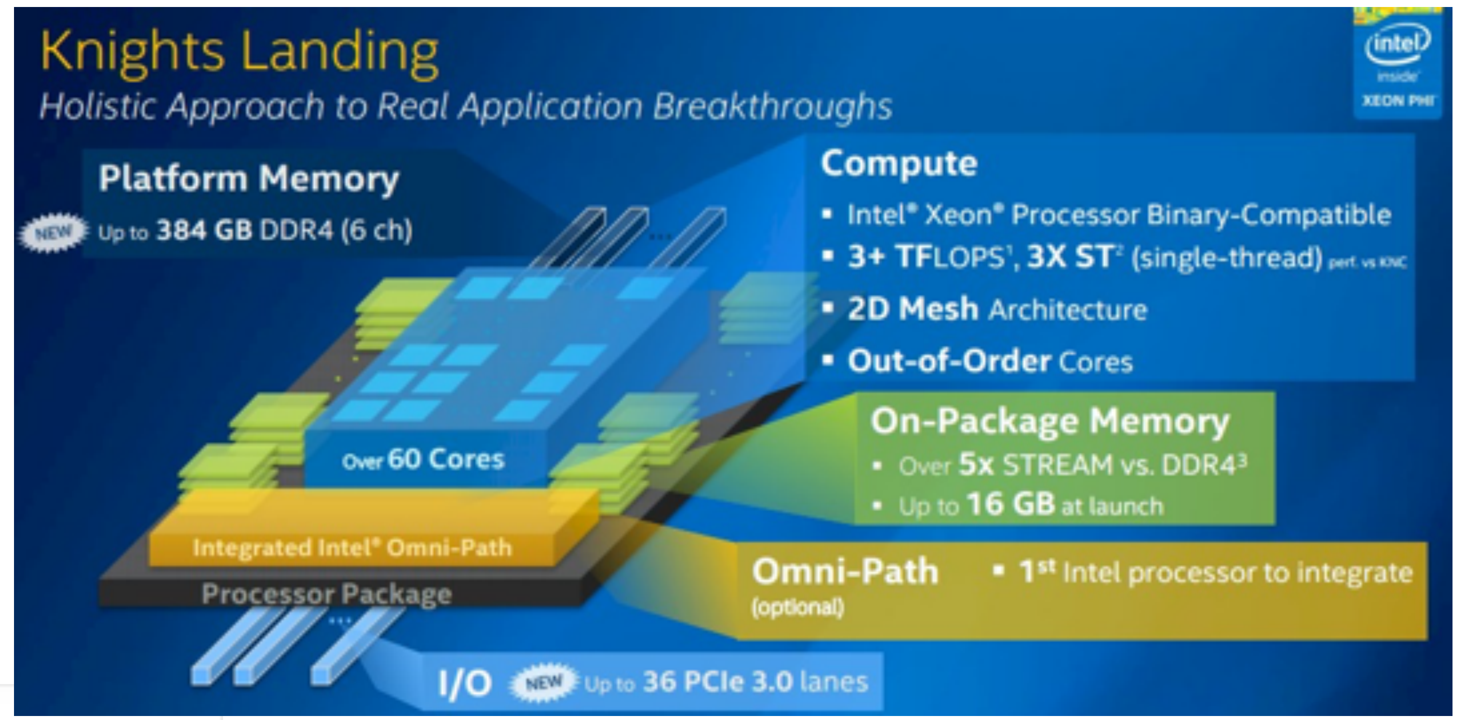
Geant4 collaboration meeting - September 2016

Motivation

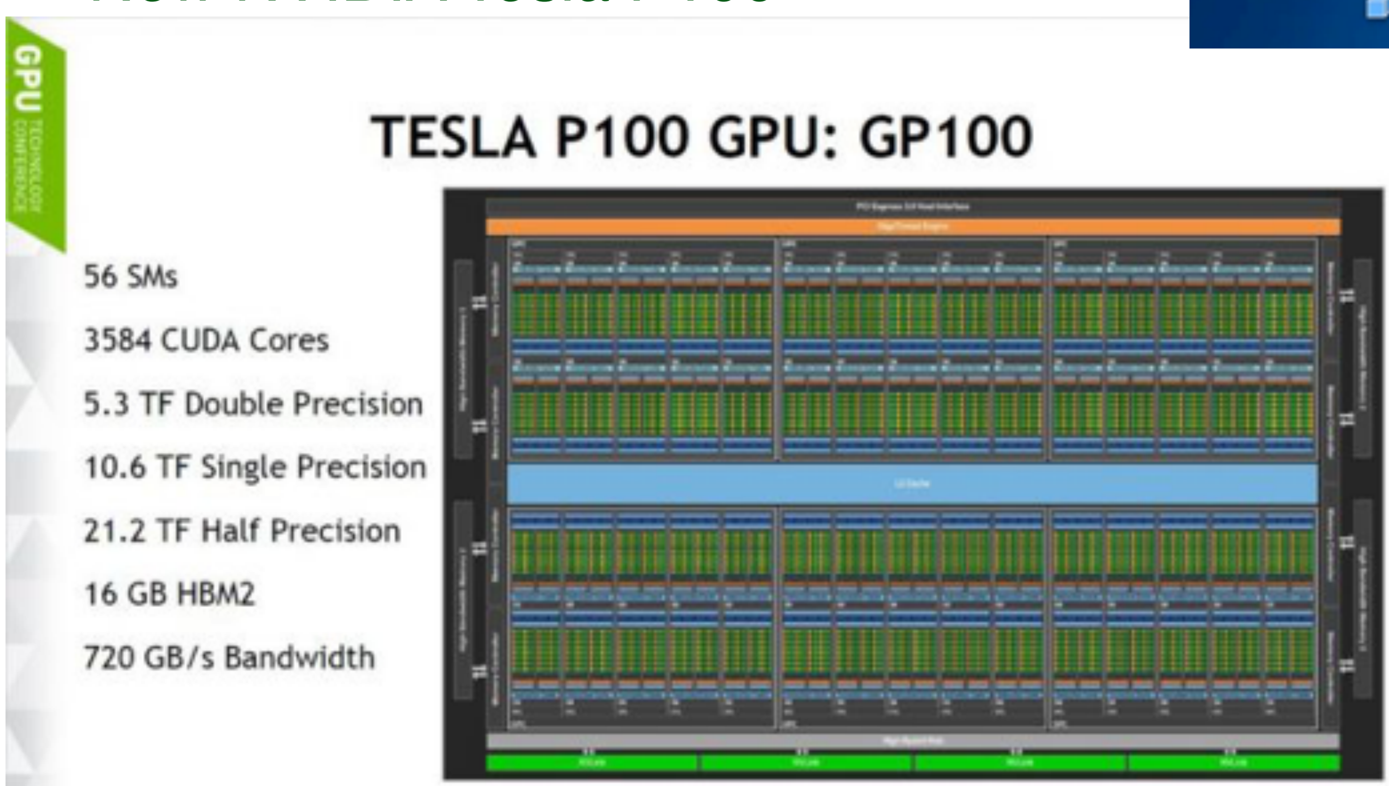
New hardware gets more and more powerful..

Ex. Officially launched in 2016:

New Intel Xeon Phi processor



New NVIDIA Tesla P100



- Task/threads and data parallelism are essential to exploit new hardware capabilities
- **Combining the two is much faster than either one alone.**

GeantV: introducing parallelism

GeantV restructures particle transport simulation in a new algorithm

- Improving physics models
- Including options for fast simulations
- Introducing parallelism (task/process and data)
  - Group particles exploiting locality (geometry or physics)

    Transport particles in groups (baskets)
  - Multi-threading and multi-tasking
  - Explicit memory management for NUMA aware systems
- Easy portability across different architectures

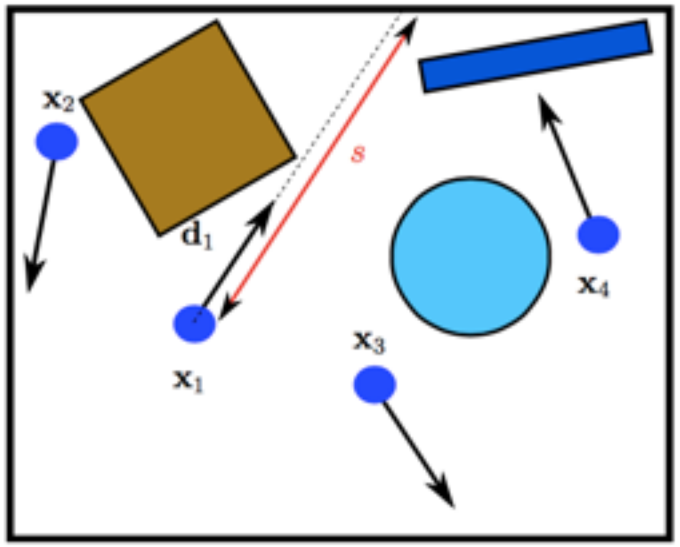
The goal is x3-x5 better performance than “state of the art” Geant4

… and understand how to go beyond

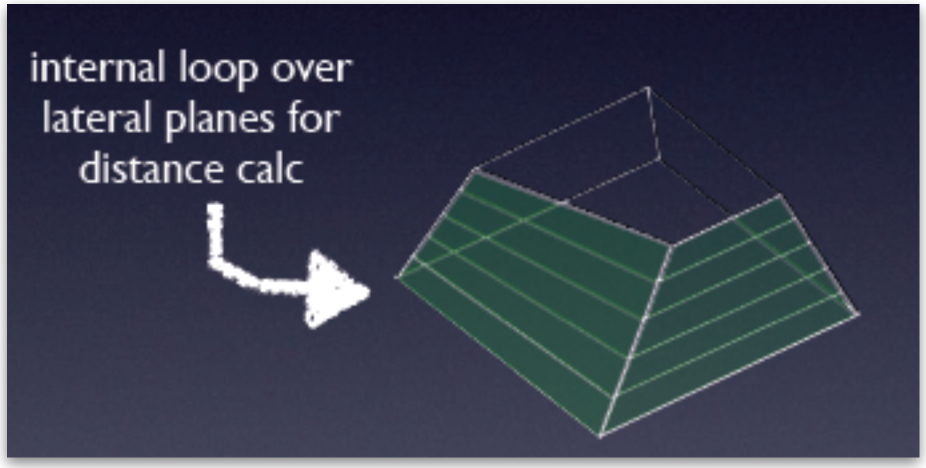
VecGeom

see Sandro's talk on Thursday

- Optimised library of primitive and composite solids
- Core of templated kernels using abstract types
  - compiler optimised code for any combination of primitive shapes ("template-shape specialisation")
  - No virtual function calls/ avoid code multiplication
- SIMD vectorisation & accelerator ready
  - APIs for single & many-track navigation
  - "Inner" vectorisation of complex shapes
  - Uses backends

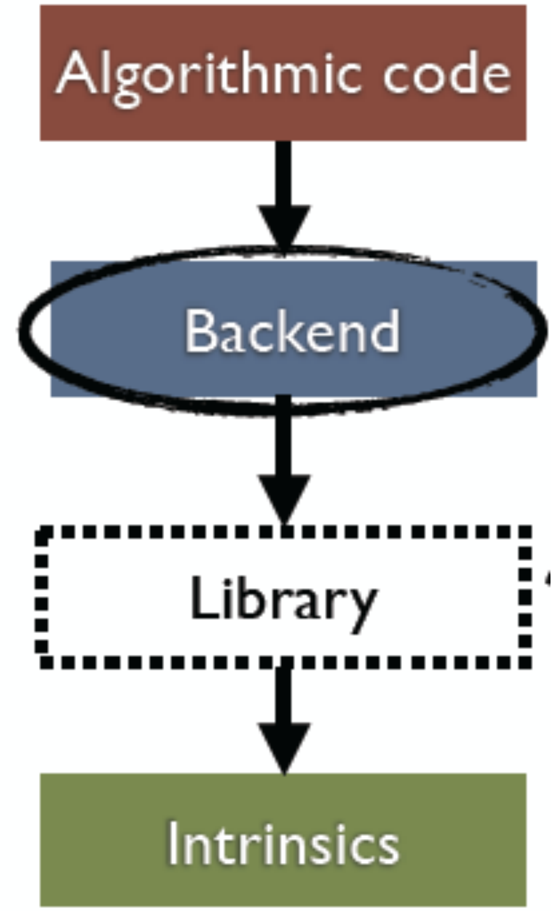


Currently migrating to new improved backend interface (VecCore)



Backends

Portability across platforms & vector types



A trait struct encapsulating standard types & properties for different architectures.

Hides underlying vectorizing hardware from the user

- Abstraction of underlying intrinsics
- Act as a layer between algorithmic code and intrinsics
- additional library layer (Vc/Umesimd)
- Can guide behaviour of algorithms depending on architecture (scalar/vector/GPU)

```
struct kVc {
  typedef Vc::int_v                          int_v;
  typedef Vc::Vector<Precision>              precision_v;
  typedef Vc::Vector<Precision>::Mask        bool_v;
  typedef Vc::Vector<int>                    inside_v;
  constexpr static bool early_returns = false;
```

```
template <class Backend>
VECGEOM_CUDA_HEADER_BOTH
typename Backend::bool_v Contains(
    Vector3D<typename Backend::precision_v> const &point) const;
```


https://github.com/VcDevel/Vc
https://bitbucket.org/edanor/umesimd

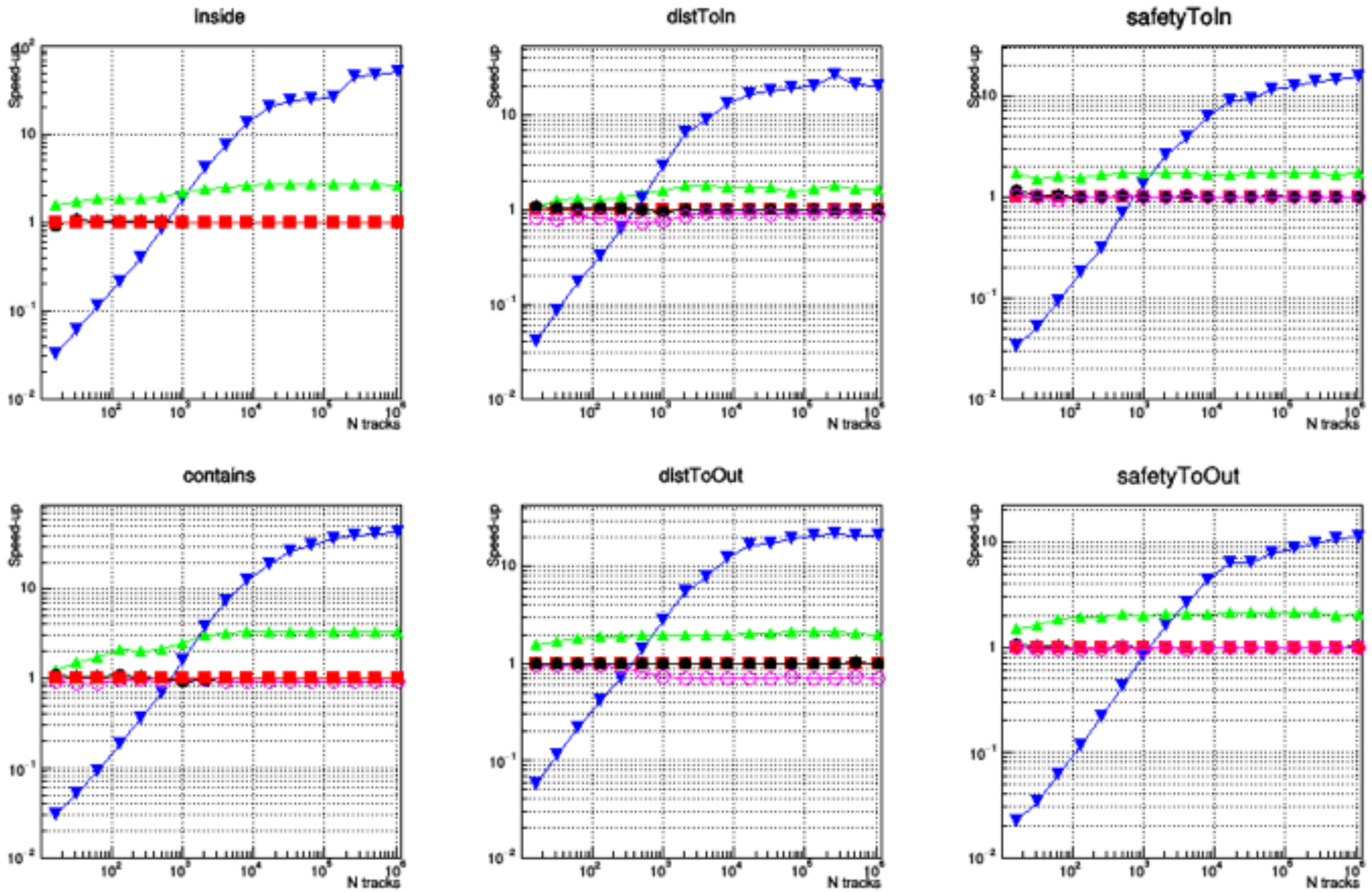
VecGeom on GPU

Speedup for different navigation methods for the BOX shape (normalized to scalar)

– GPU: Nvidia Kepler (K20), 2496 cores @ 0.7 GHz

- Asynchronous data transfer
- Measured only the kernel performance, but providing constant throughput can hide transfer latency
- The die can be saturated with both large track containers, running a single kernel, or with smaller containers dynamically scheduled.

Just a baseline proving we can run the same code on CPU/ accelerators, to be optimized

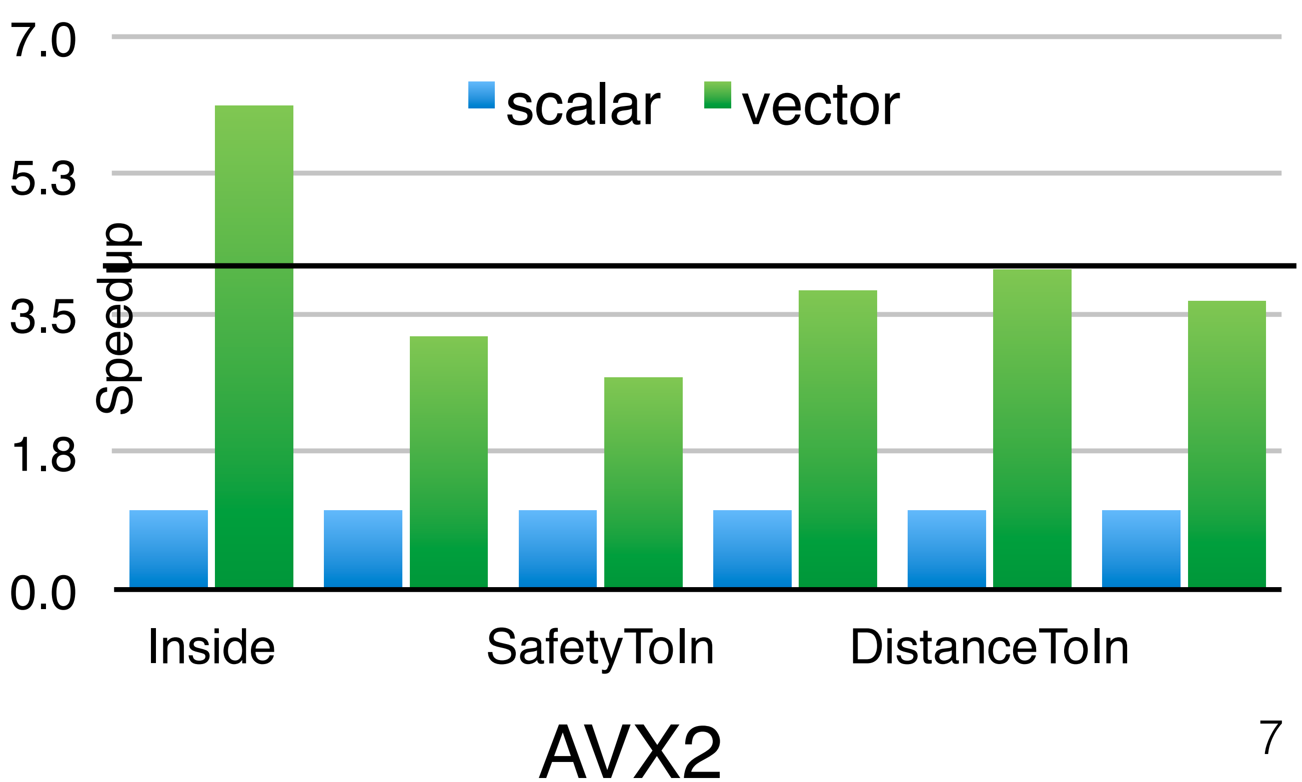


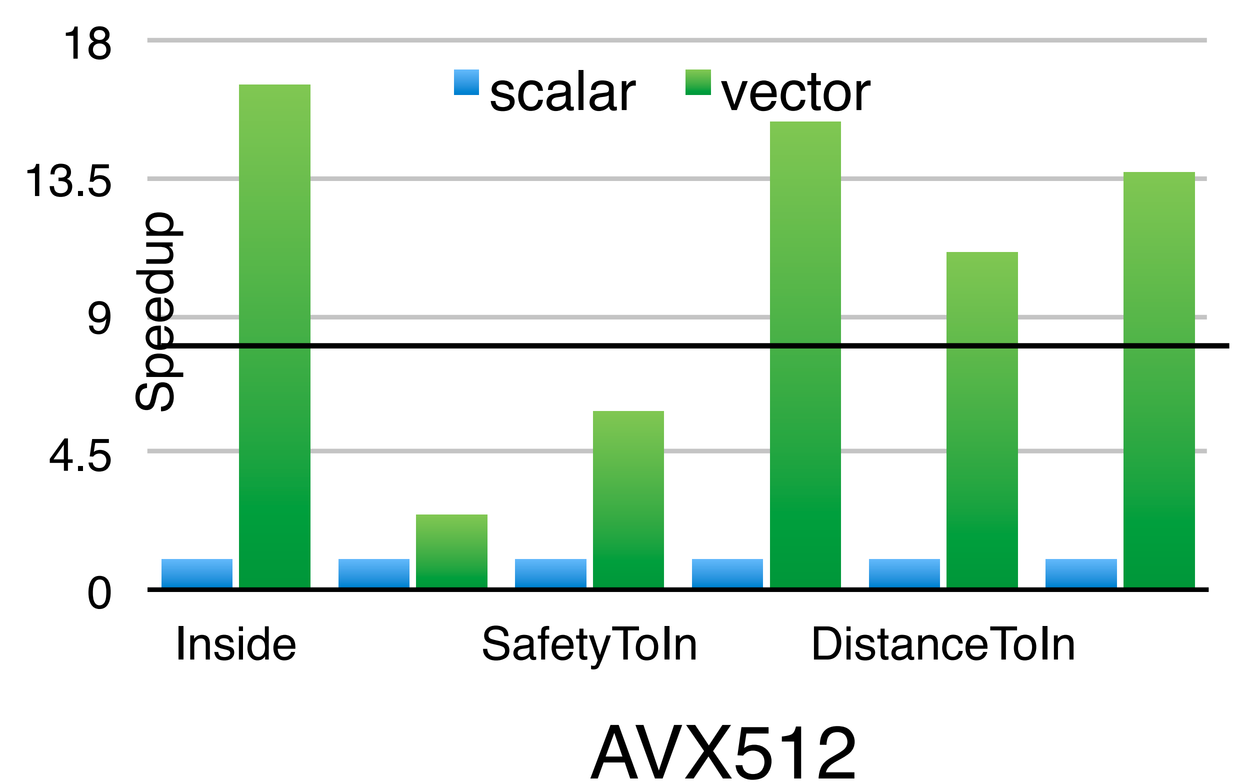
Scalar (specialized/unspecialized) Vector GPU (K20) ROOT

VecGeom on KNL

- **Running set of standard navigation benchmarks using UME::SIMD backend.**
- KNL systems use 512 bit registers corresponding to 8DP and 16SP floating points
- Vector versus scalar speed-up using AVX2 and AVX512
- **Super-linear speedup for some methods**
- Investigating if it is compiler-related

Intel® Xeon Phi™ CPU 7210 @ 1.30GHz, 64 cores



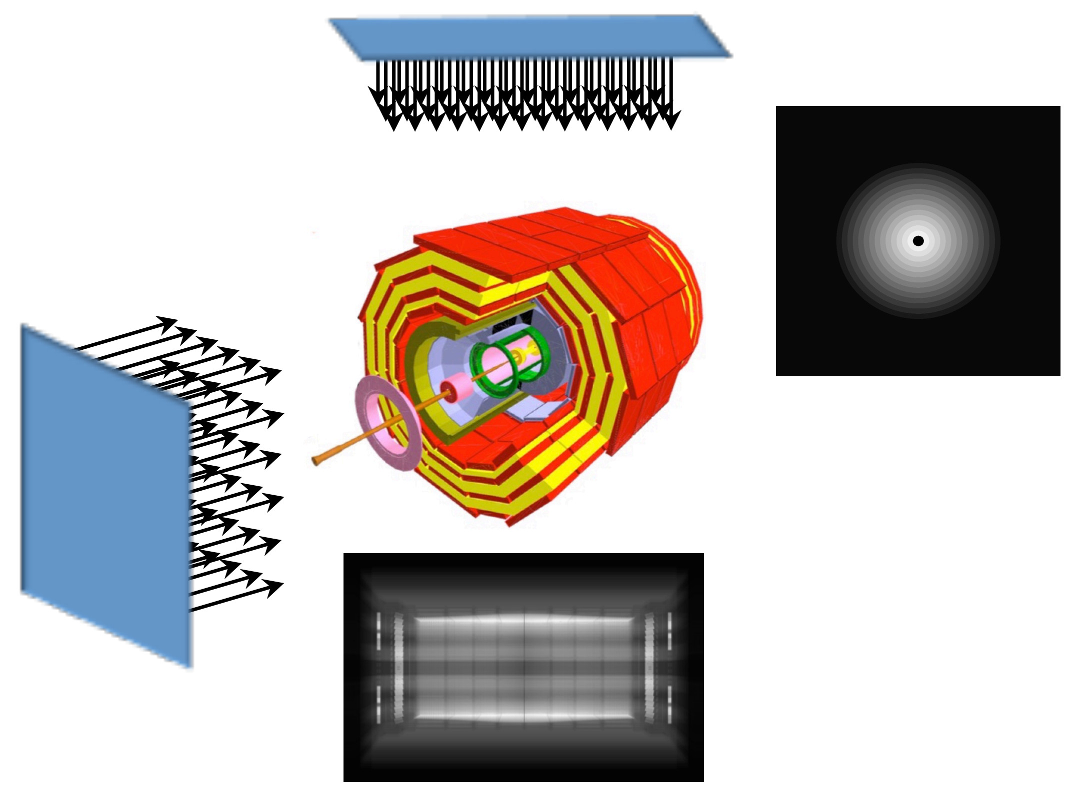


The x-ray benchmark

Benchmarking full geometry navigation of a toy detector

presented @ ISC16

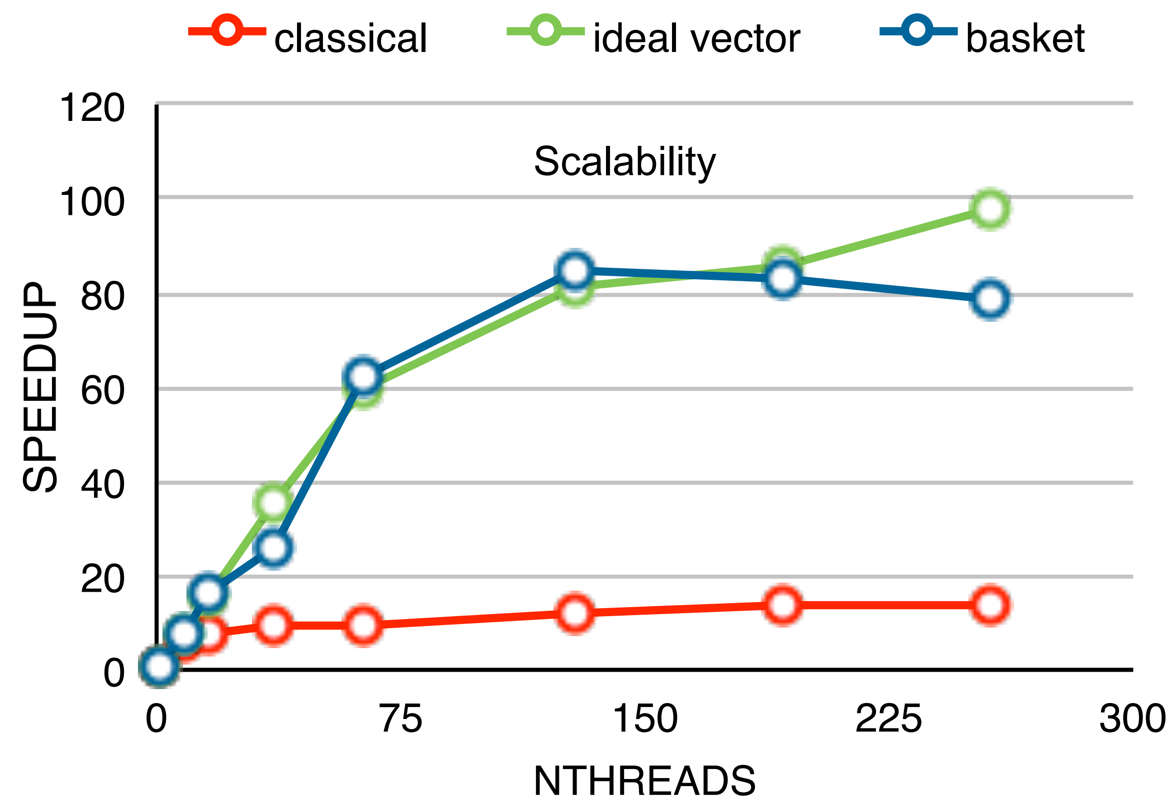
- "X-Ray" scan of a simple geometry
  - A concentric set of **tubes** emulating a tracker detector
  - Trace one ray per pixel from one edge to the opposite
- **Test the global navigation algorithm**
- **Stress vector API + basket transport** tracing multiple identical tracks through the same grid
  - **Test OMP parallelism** producing multiple identical images

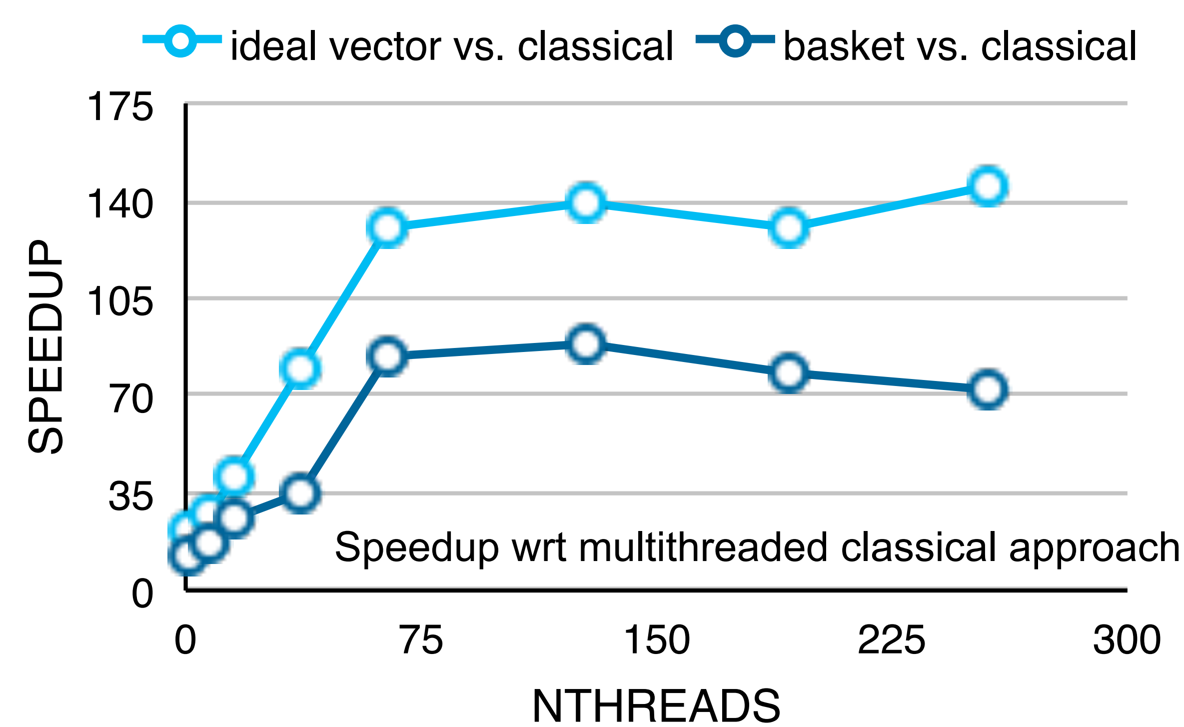


Scalability

Intel® Xeon Phi™ CPU 7210 @ 1.30GHz,
64 cores

- Compare basketized navigation to scalar Geant4 (one navigator per thread)
- vectorization enforced by API,
  (UME::SIMD backend for AVX512)
- Scalability reaches ~100x for the ideal and basket versions
- Preliminary results: the whole GeantV scheduler is being redesigned

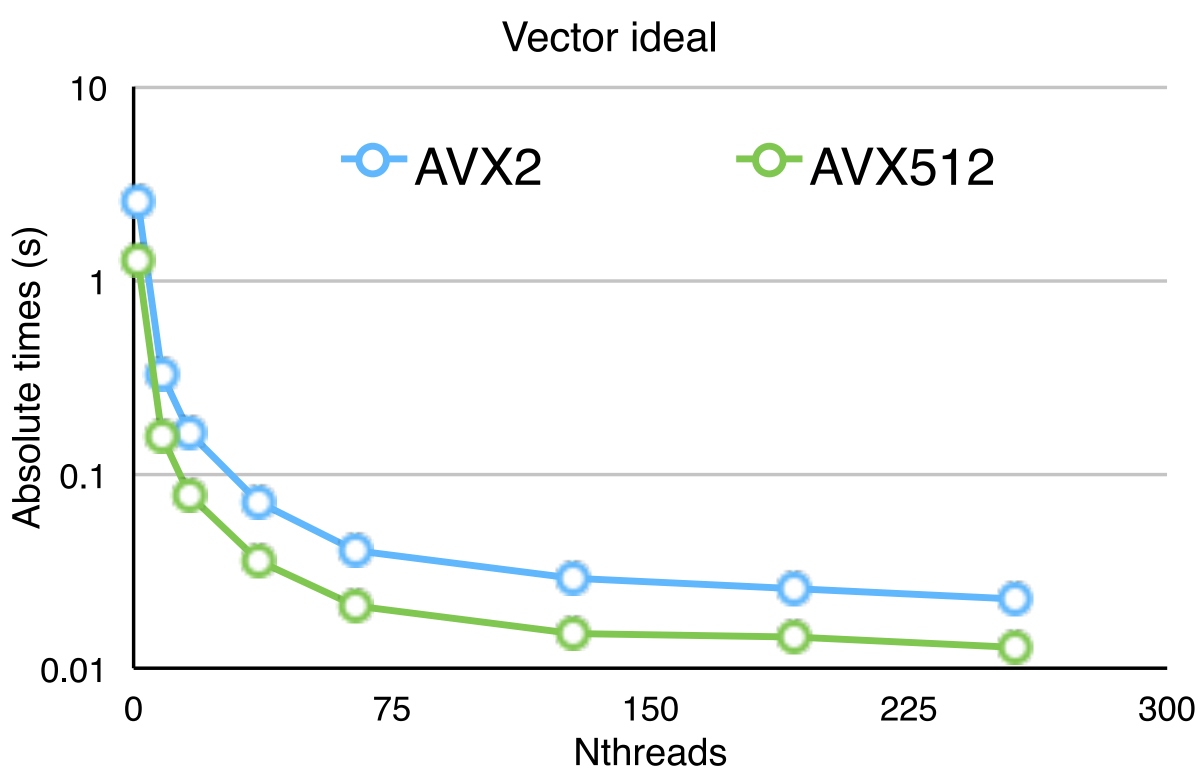


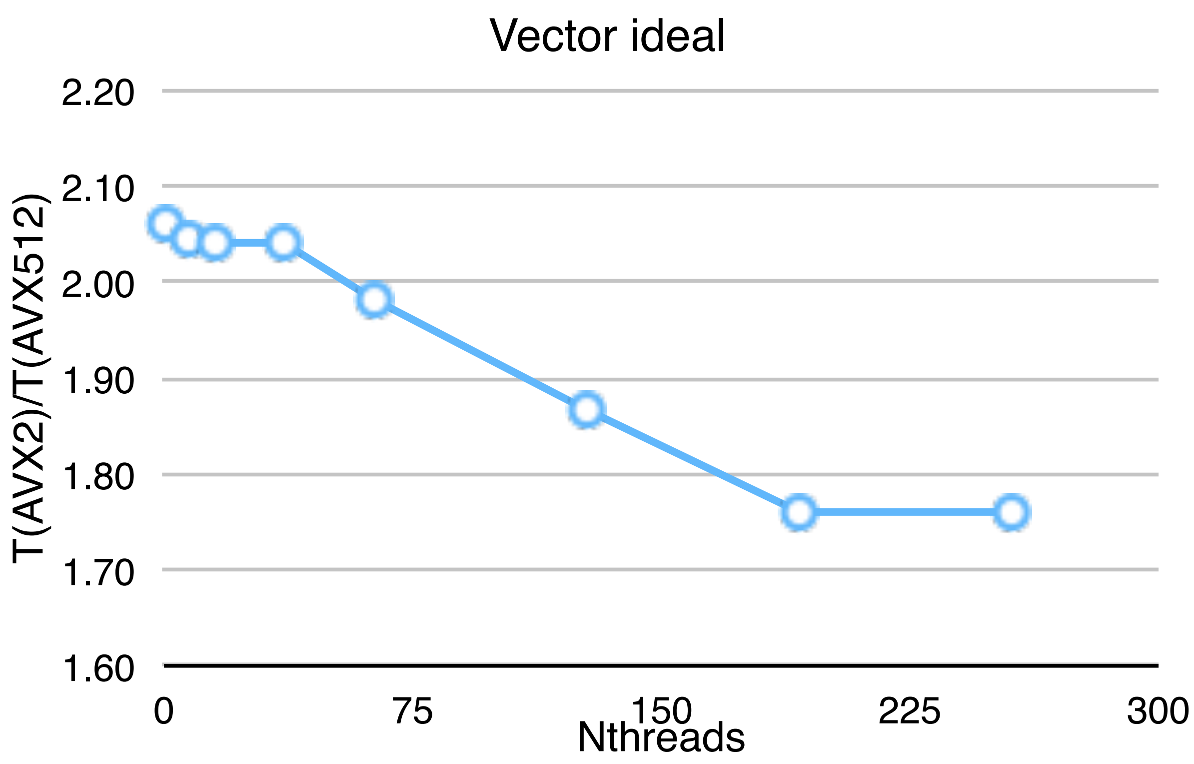


Vectorization

High vectorization intensity achieved for both ideal and basketized cases

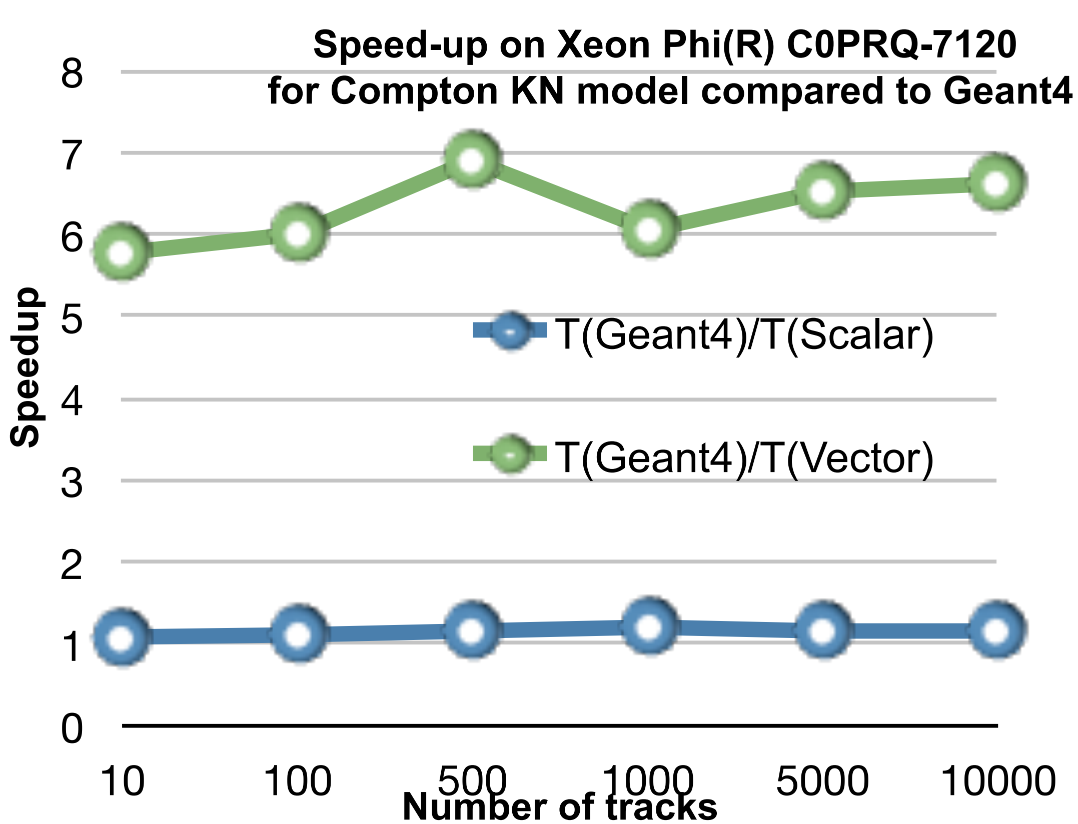
AVX512 brings an extra factor of ~2 to our benchmark



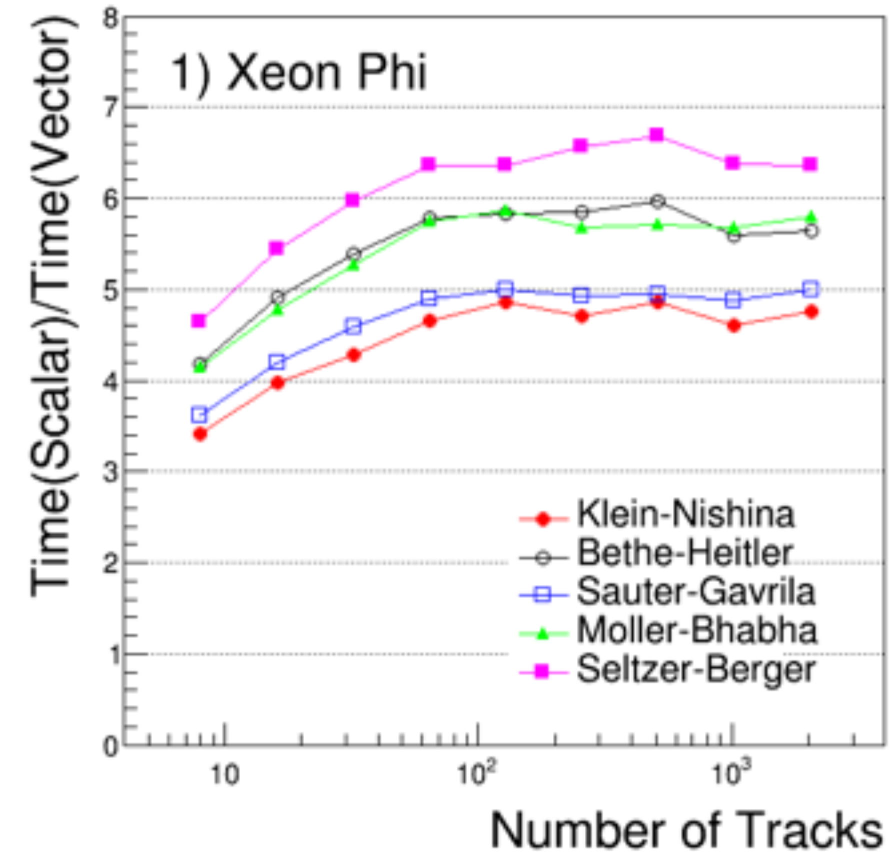


What about physics?

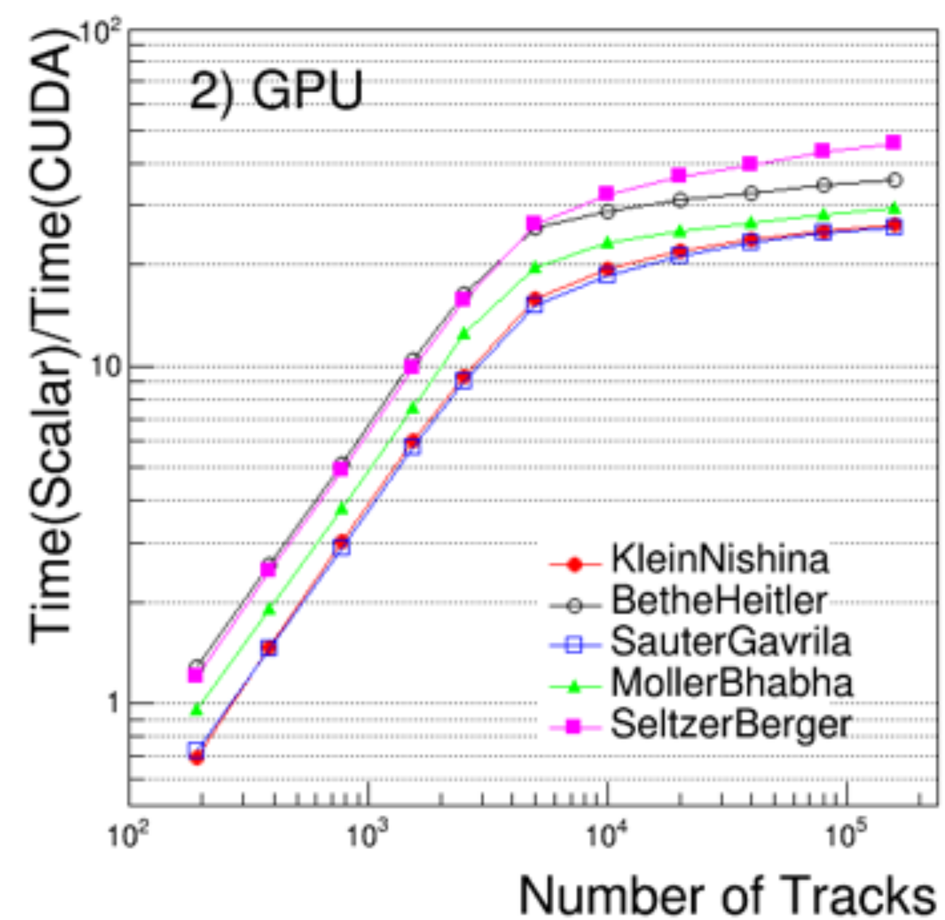
- Objective: a vector/accelerator friendly physics code
- Started with the electromagnetic processes
- The vectorised Compton scattering shows good performance gains



Intel Xeon Phi 5110P 60 cores @ 1.053 GHz



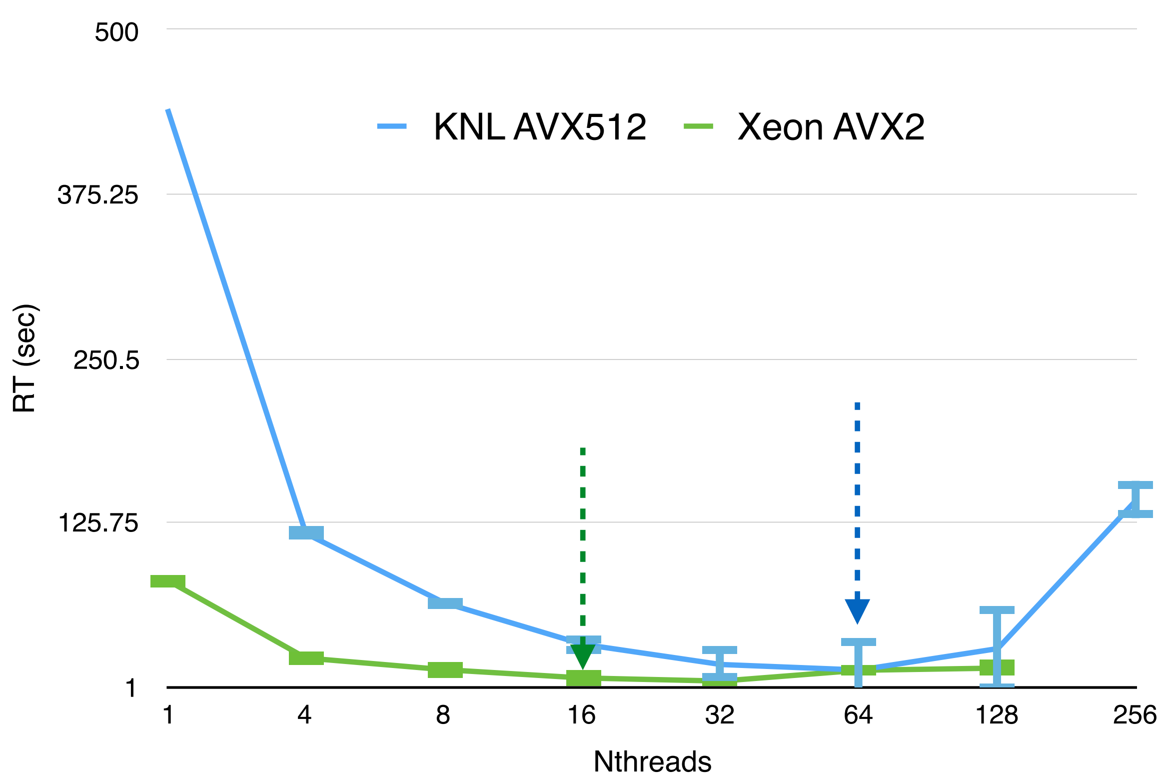
GPU: Nvidia K20
Host: Intel Xeon E5 – 2650 @ 2.60 GHz

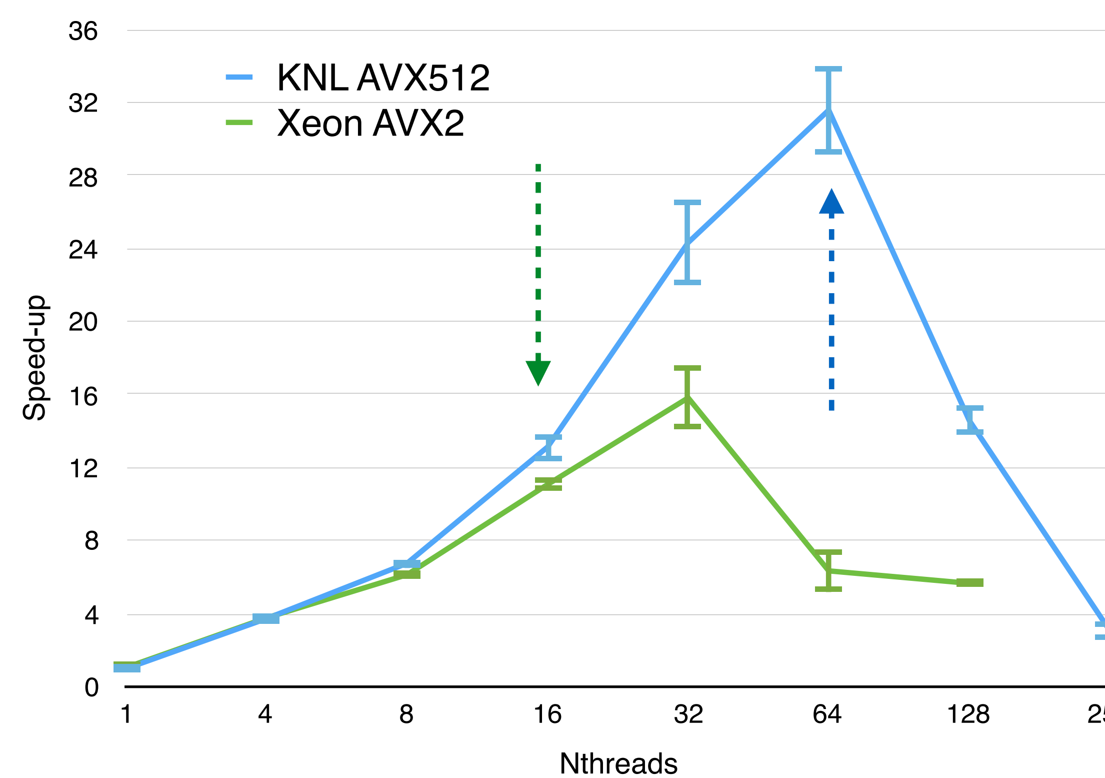


Full GeantV prototype

- First GeantV full navigation benchmark on KNL
  - Tabulated physics
  - Simplified detector geometry
  - Test track transport and basketization procedure
- Use UME::SIMD backend for AVX512

Good scalability up to the number of physical cores then rebasketizing sync problems (see Andrei's talk)

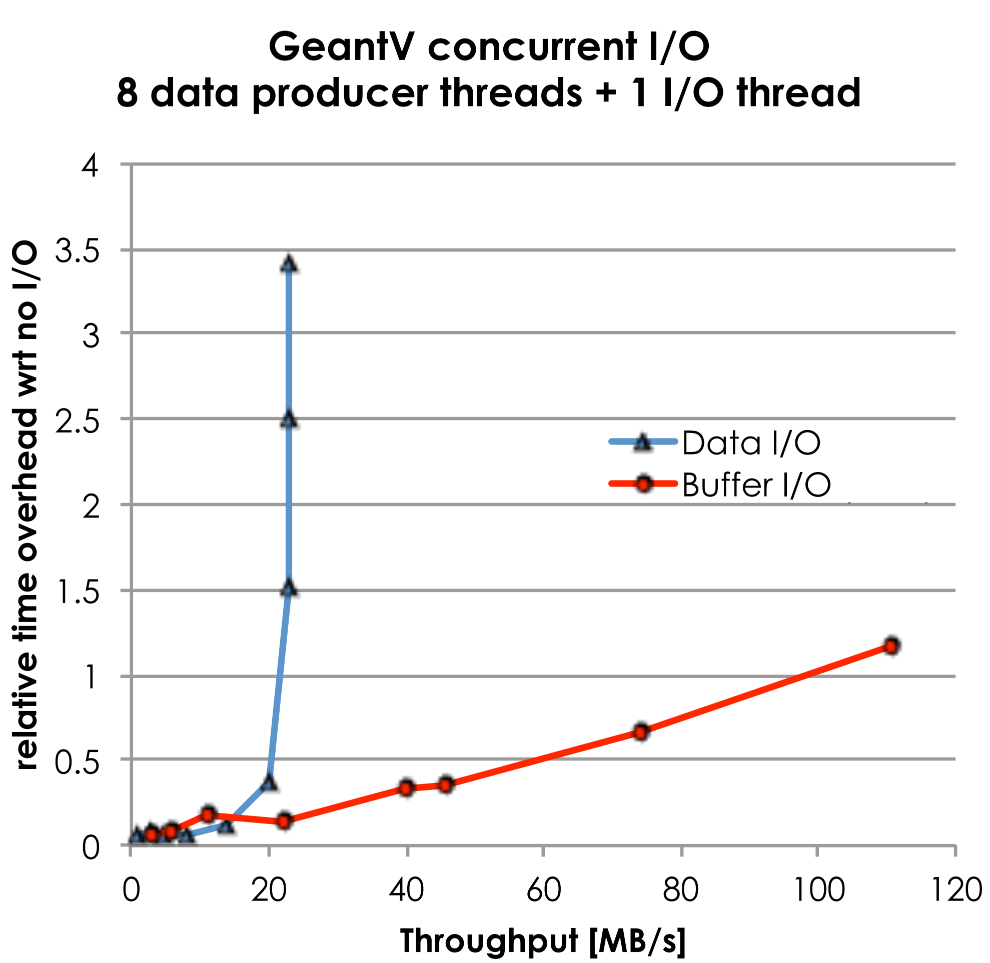




Intel Xeon Phi 7210 @1.30 GHz
Intel Xeon E5-2630 v3 @2.40GHz

Concurrent output

- Simulated hits are produced concurrently by the different threads
- Thread-safe queues have been implemented to handle asynchronous generation of hits by several threads
- Dedicated Output thread transfers the data from the output queues to ROOT I/O



“Data” mode (sequential)
Send concurrently data to one thread dealing with full I/O

“Buffer” mode (concurrent)
Send concurrently local trees connected to memory files produced by workers to one thread dealing with merging/write to disk

GPU schema

- Broker adapts baskets to the coprocessor
  - Selects tracks that are efficiently processed on coprocessor
  - Gathers in chunk large enough (e.g. 4096 tracks on NVidia K20)
  - Transfers data to and from coprocessor
  - Executes kernels
- On NVidia GPU, we are effectively using implicit vectorization
  - Rather than one thread per basket, we use 4096 threads each processing one of the tracks in the basket
- Cost of data transfer is mitigated by overlapping kernel execution and data transfer
  - We can send fractions of the full GPU's work asynchronously using streams

Summary & plans

- A first successful iteration on KNL for geometry benchmarks was demonstrated last June at ISC'16
  - Overhead due to track reshuffling is under control
- Now testing and optimising the whole prototype:
  - Full realistic detector geometry benchmark on KNL by SC16
- Concerning GPU:
  - Broker with 'geometry only' kernel works
  - Tabulated physics code transfers data to GPU (memory fetching from tables maybe a bottleneck!)
  - Next steps:
    - Adapt to new navigation code and incorporate physics code into CUDA Kernel
    - Running the full prototype using GPU as co-processor, understand performance issues and optimise

Benchmark baseline

We compare GeantV basketized navigation to scalar Geant4

- Classical approach (Geant4, ROOT) uses a navigator per thread and works in scalar mode
- “Basket” approach uses one navigator per volume in vectorized mode
- Crossing a layer “feeds” the next basket