GeantV

Philippe Canal (FNAL) for the GeantV development team

G.Amadio (UNESP), Ananya (CERN), J.Apostolakis (CERN) , A.Arora (CERN), M.Bandieramonte (CERN), A.Bhattacharyya (BARC), C.Bianchini (UNESP), R.Brun (CERN), Ph.Canal (FNAL), F.Carminati (CERN), L.Duhem (intel), D.Elvira (FNAL), A.Gheata (CERN), M.Gheata (CERN), I.Goulas (CERN), R.Iope (UNESP), S.Y.Jun (FNAL), G.Lima (FNAL), A.Mohanty (BARC), T.Nikitina (CERN), M.Novak (CERN), W.Pokorski (CERN), A.Ribon (CERN), R.Sehgal (BARC), O.Shadura (CERN), S.Vallecorsa (CERN), S.Wenzel (CERN), Y.Zhang (CERN)

Outline

- GeantVectorized – an introduction
  - The problem
  - Existing solutions
  - Challenges, ideas, goals
- Main components and performance
  - Design and infrastructure
  - Vectorization: overheads vs. gains
  - Geometry library
  - Physics processes
- Performance benchmarks
- Results, milestones, plans

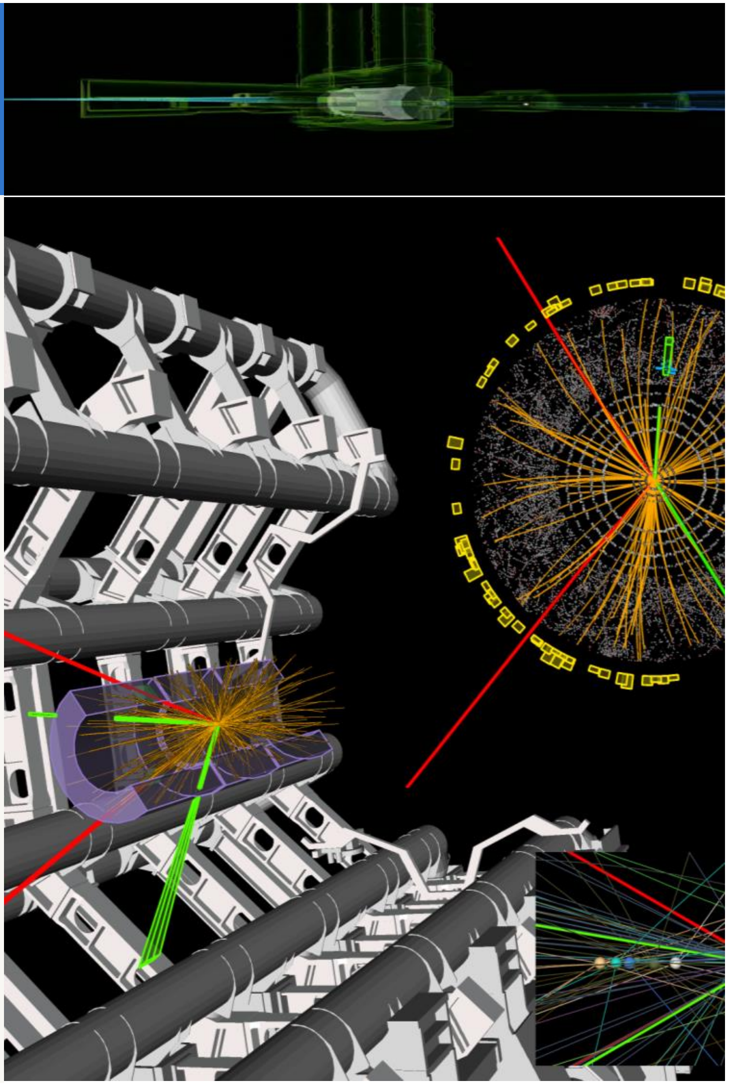
The problem

Detailed simulation of subatomic particle transport and interactions in detector geometries

Using state of the art physics models, propagation in electromagnetic fields in geometries having complexities of millions of parts

Heavy computation requirements, massively CPU-bound, seeking organically HPC solutions…

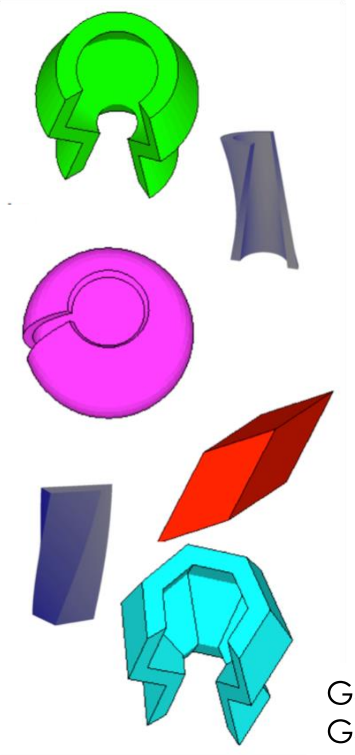
The LHC uses more than 50% of its distributed GRID power for detector simulations (~250.000 CPU years equivalent so far)

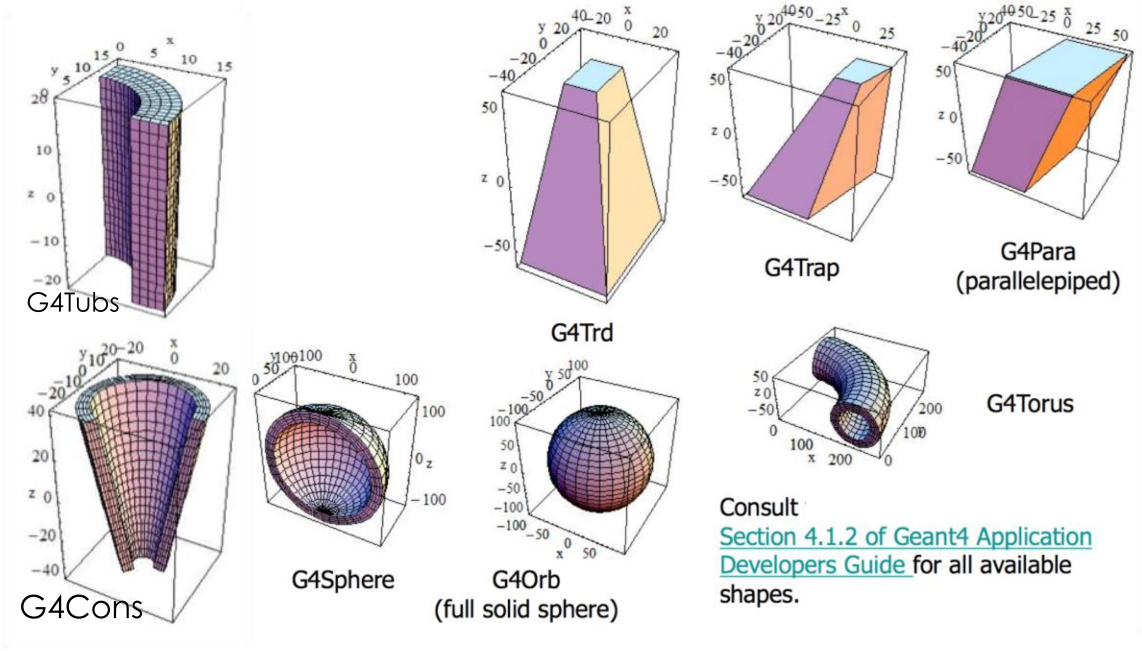


http://atlas.ch

Geometry

A large collection of solids are defined (similar to Geant4 below)

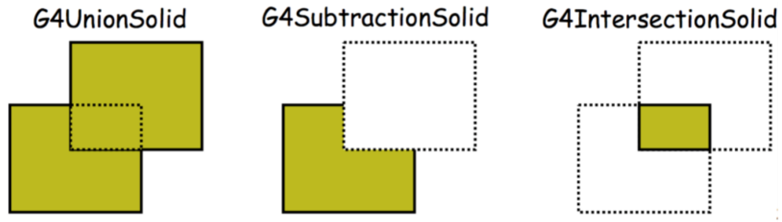




Consult [Section 4.1.2 of Geant4 Application Developers Guide](#) for all available shapes.

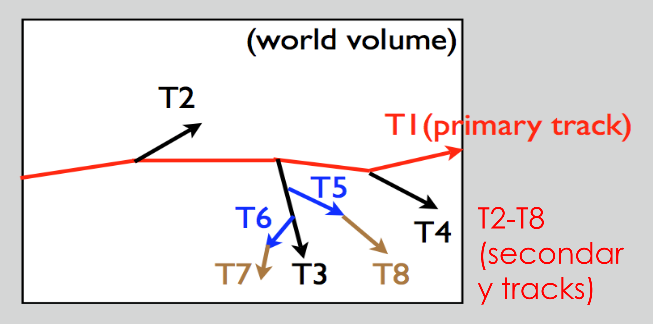
G4Polycone, G4 Polyhedra, G4Hype,
G4TwistedTubs, G4TwistedTrap

Also Boolean operations such as:

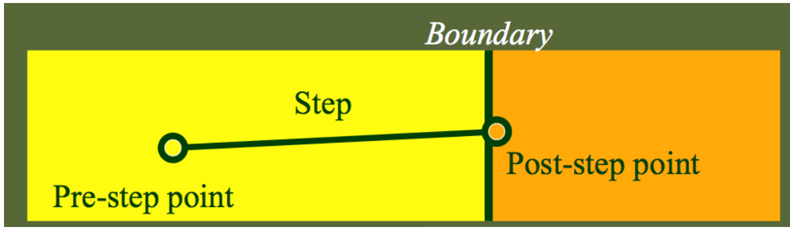


Transportation

A Track also includes the info for transporting the particle through the detector ➔ we typically use Particle = Track

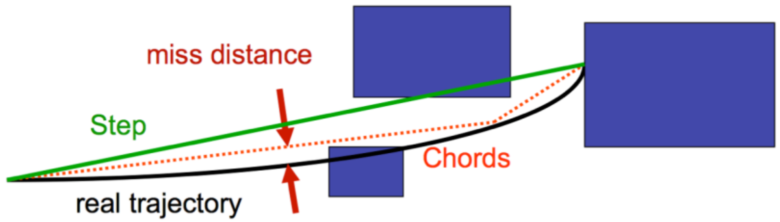


The track information is updated before and after every step in the particle (track) propagation



Magnetic Field

Particle propagated in EM field by integration of equation of motion using the Runge-Kutta method (others also available)



- Curved path broken into linear *chord* segments to minimize the *sagitta* (maximum chord-trajectory distance)
- Chords used to interrogate navigator on whether the track has crossed a volume boundary
- *miss distance* parameter used to tune volume intersection accuracy

Need to supports user defined, uniform, and non-uniform (static or time dependent) magnetic fields

Physics

- A set of *models* for physics processes which include the *cross sections* and final state information for each particle type
- The *process categories* are electromagnetic, hadronic, decay, optical, photolepton_hadron, parameterization, transportation
- Three kinds of *process actions*: AtRest (decay, annihilation), AlongStep (continuous interactions like ionization), PostStep (point-like interactions like decay in flight, EM, hadronic)

Processes are defined for both primary and secondary particles

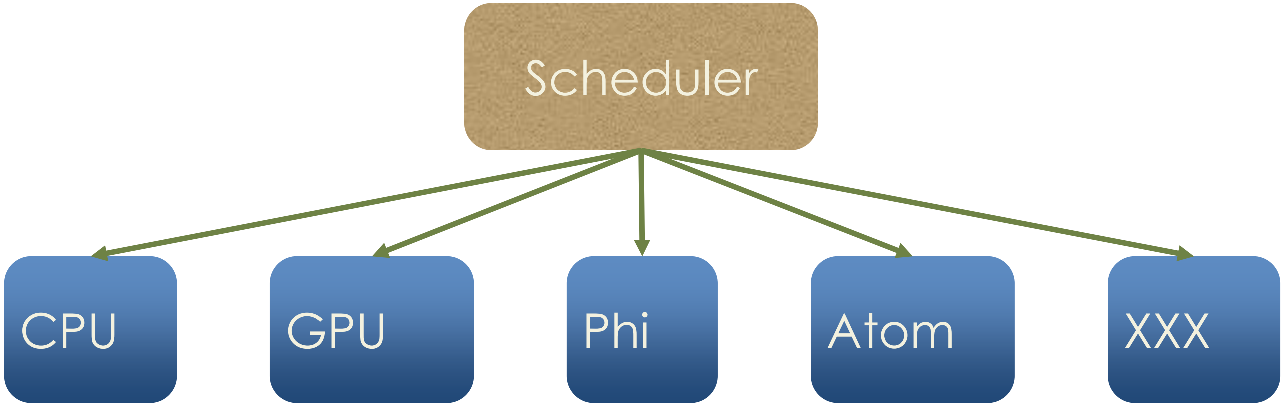
- Particles tracked down to zero kinematic energy
- Production cuts on secondary particles set to a default of 1 mm, secondaries with $E_{kin} < E_{Th}$ to travel 1 mm are stopped

What do we want to do?

- Develop an all-particle transport simulation software with
  - Geant4 physics models
  - A performance between 2 and 5 times greater than Geant4
  - Full simulation and various options for fast simulation
  - Portable on different architectures, including accelerators (GPUs and Xeon Phi's)
- Understand the limiting factors for a one-order-of-magnitude (10x) improvement

Challenges

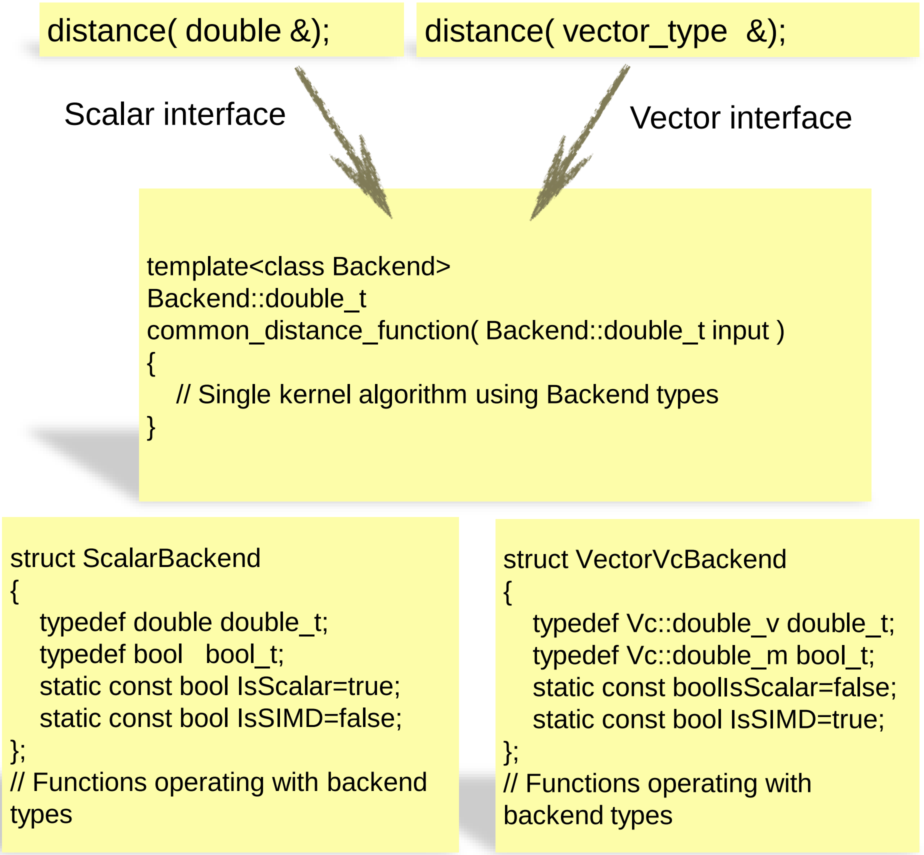
- Overhead from reshuffling particle lists should not offset SIMD gains
- Exploit the hardware at its ***best***, while maintaining ***portability***



- Test from the onset on a “large” setup (LHC-like detector)
  - Toy models tell us very little – complexity is the problem

The ideas

- Transport particles in groups (vectors) rather than one by one
  - Group particles by geometry volume or same physics
  - No free lunch: data gathering overheads < vector gains
- Dispatch SoA to functions with vector signatures
  - Use backends to abstract interface: vector, scalar
  - Use backends to insulate technology/library: Vc, Cilk+, VecMic, …
- Redesign the library and workflow to target fine grain parallelism
  - CPU, GPU, Phi, Atom, …
  - Aim for a 3x-5x faster code, understand hard limits for more



```
distance( double &);
```

```
distance( vector_type  &);
```

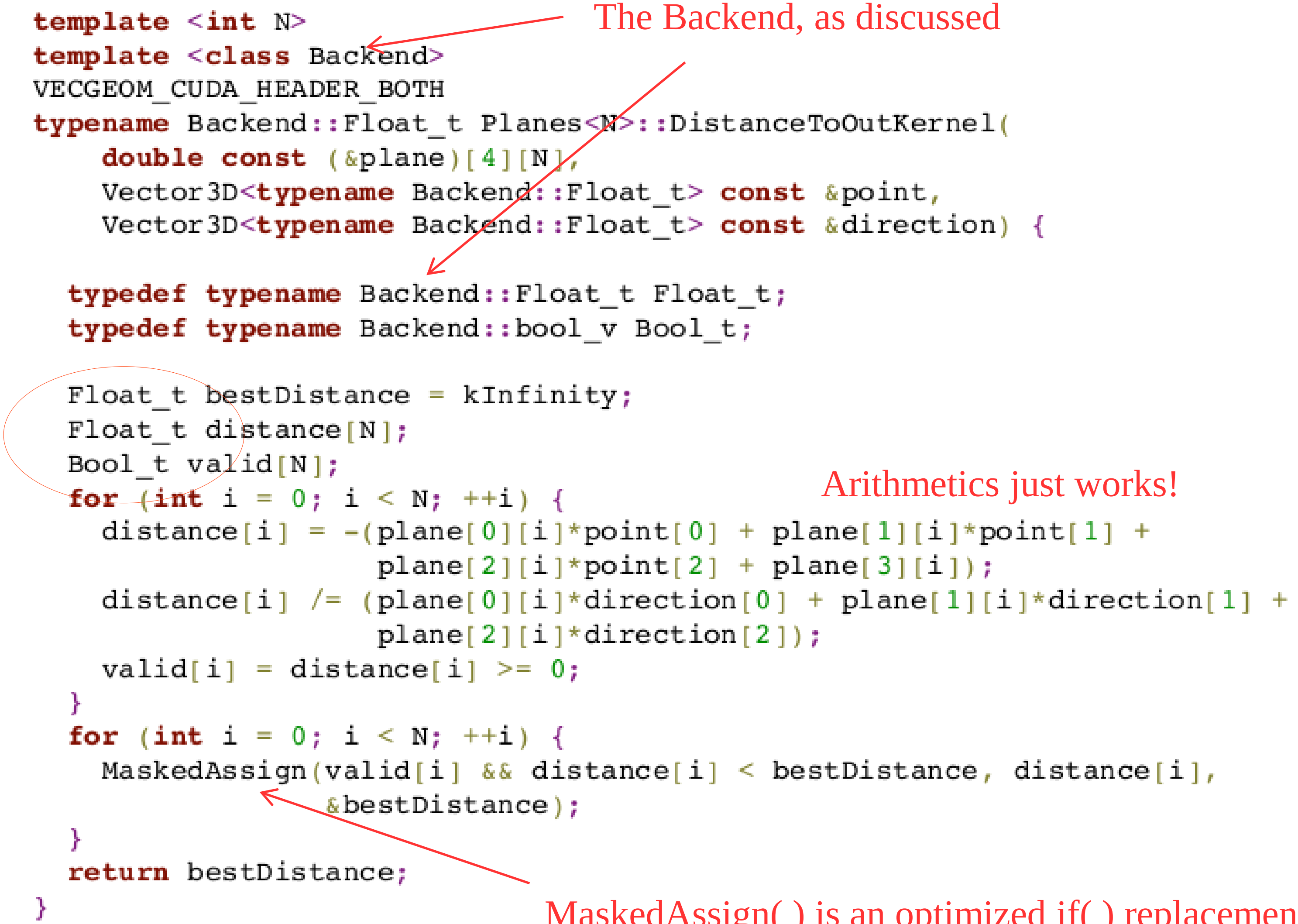
```
template<class Backend>
Backend::double_t
common_distance_function( Backend::double_t input )
{
   // Single kernel algorithm using Backend types
}
```

```
struct ScalarBackend
{
   typedef double double_t;
   typedef bool   bool_t;
   static const bool IsScalar=true;
   static const bool IsSIMD=false;
};
// Functions operating with backend
types
```

```
struct VectorVcBackend
{
   typedef Vc::double_v double_t;
   typedef Vc::double_m bool_t;
   static const boolIsScalar=false;
   static const bool IsSIMD=true;
};
// Functions operating with
backend types
```

code.compeng.uni-frankfurt.de/projects/vc

A generic kernel



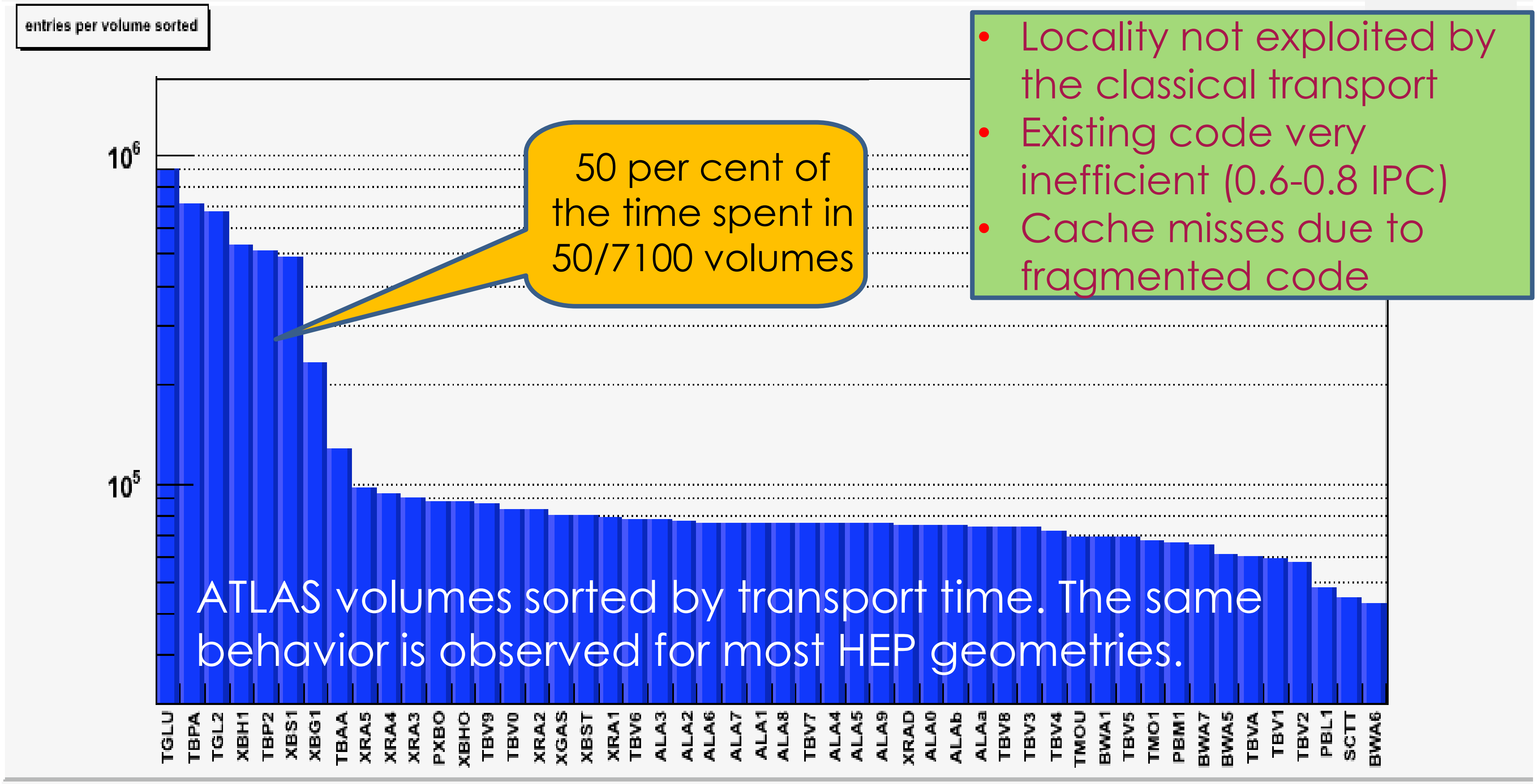
```
template <int N>
template <class Backend>
VECGEOM_CUDA_HEADER_BOTH
typename Backend::Float_t Planes<N>::DistanceToOutKernel(
    double const (&plane)[4][N],
    Vector3D<typename Backend::Float_t> const &point,
    Vector3D<typename Backend::Float_t> const &direction) {

  typedef typename Backend::Float_t Float_t;
  typedef typename Backend::bool_v Bool_t;

  Float_t bestDistance = kInfinity;
  Float_t distance[N];
  Bool_t valid[N];
  for (int i = 0; i < N; ++i) {
    distance[i] = -(plane[0][i]*point[0] + plane[1][i]*point[1] +
                    plane[2][i]*point[2] + plane[3][i]);
    distance[i] /= (plane[0][i]*direction[0] + plane[1][i]*direction[1] +
                    plane[2][i]*direction[2]);
    valid[i] = distance[i] >= 0;
  }
  for (int i = 0; i < N; ++i) {
    MaskedAssign(valid[i] && distance[i] < bestDistance, distance[i],
                 &bestDistance);
  }
  return bestDistance;
}
```

The Backend, as discussed

Arithmetics just works!

MaskedAssign( ) is an optimized if( ) replacement

HEP transport is mostly local !



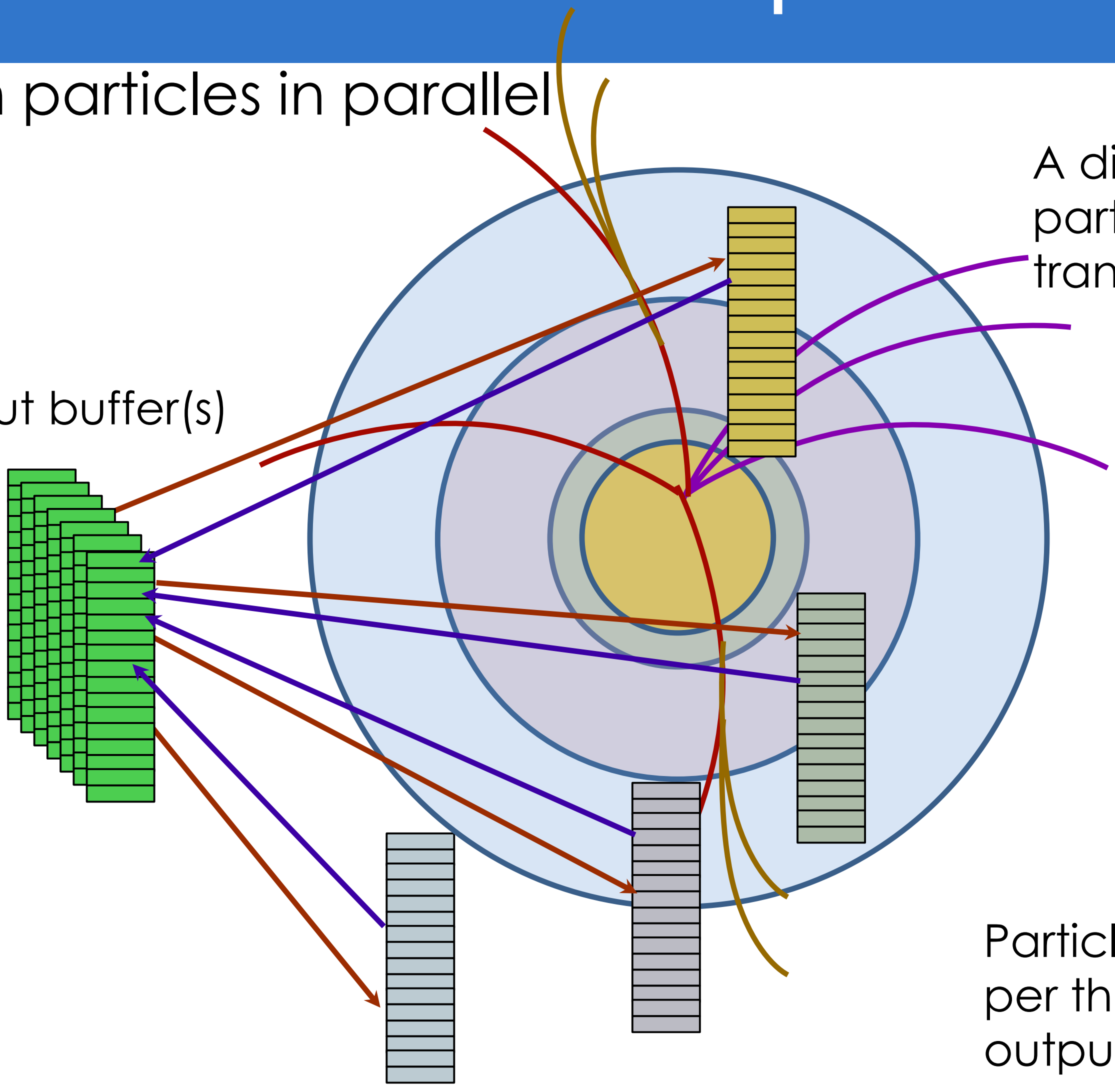
50 per cent of the time spent in 50/7100 volumes

- Locality not exploited by the classical transport
- Existing code very inefficient (0.6-0.8 IPC)
- Cache misses due to fragmented code

ATLAS volumes sorted by transport time. The same behavior is observed for most HEP geometries.

“Basketised” transport

Deal with particles in parallel



A dispatcher thread puts particles back into transport buffers

Everything happens asynchronously and in parallel

The challenge is to minimise locks

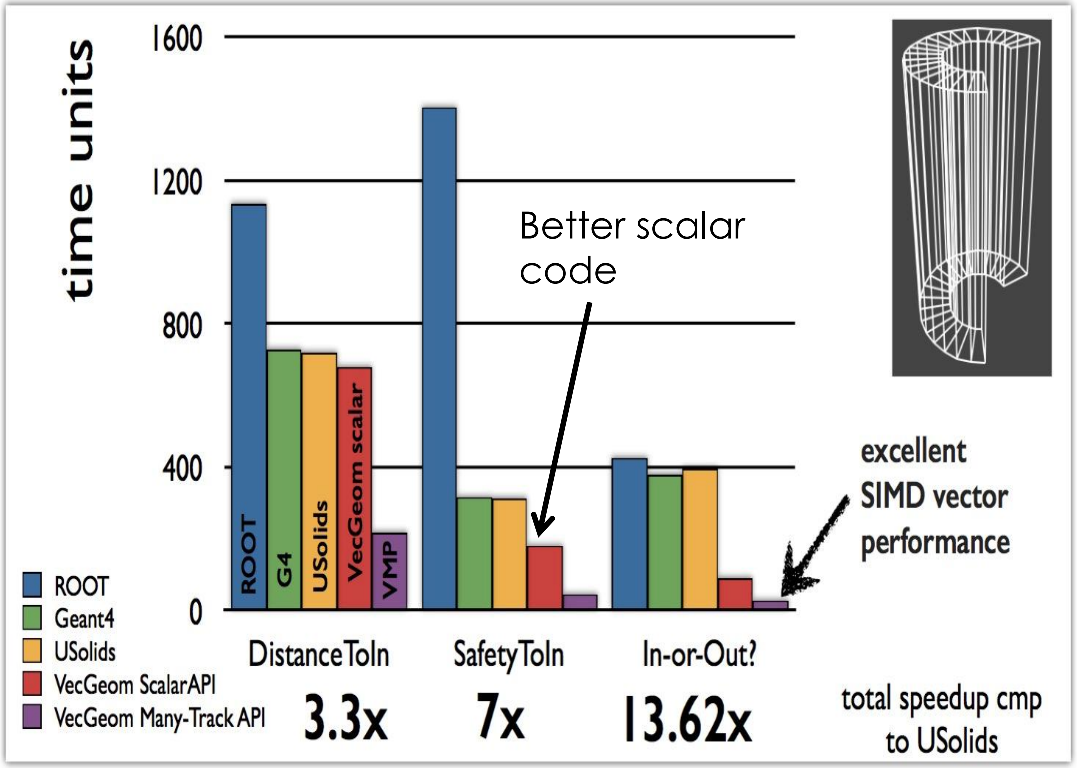
Keep long vectors

Avoid memory explosion

Particles are transported per thread and put in output buffers

Geometry - VecGeom

- Geometry takes 30-40% CPU time of typical Geant4 HEP Simulation
- A library of vectorised geometry algorithms to take maximum advantage of SIMD architectures
- Substantial performance gains also in scalar mode

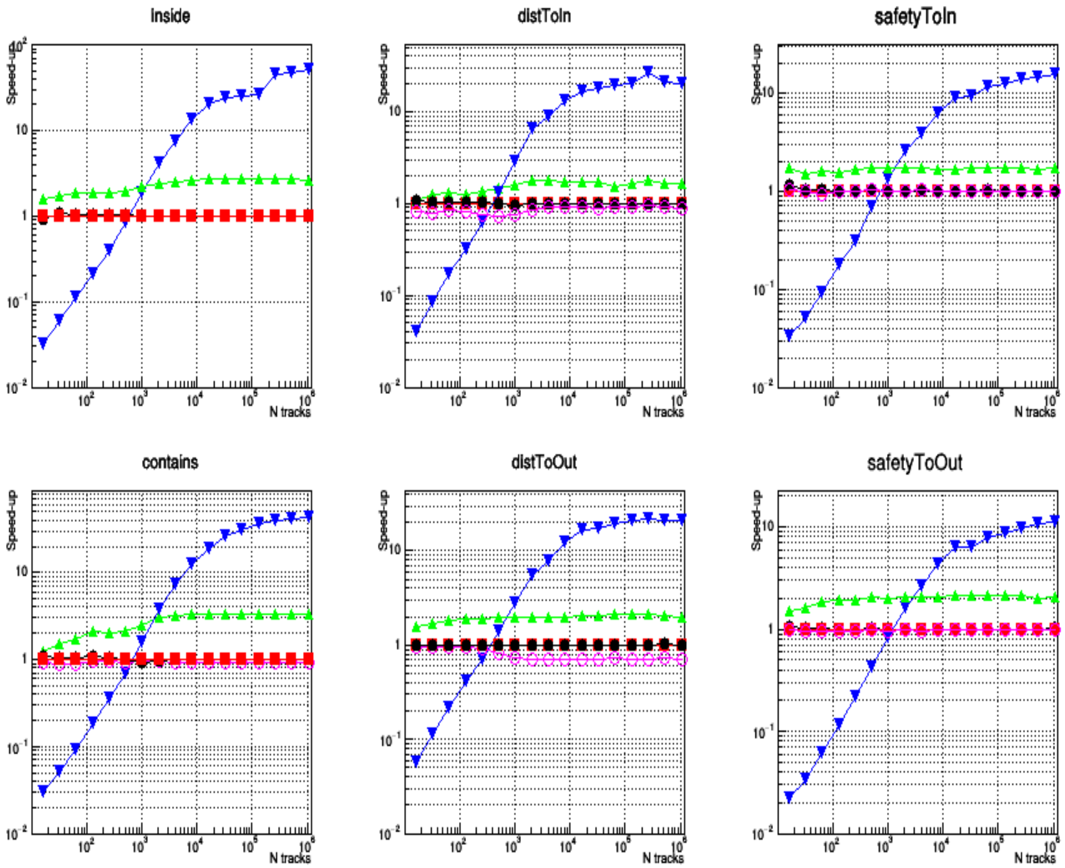


Better scalar code

excellent SIMD vector performance

Geometry performance on K20

- Speedup for different navigation methods of the box shape, normalized to scalar CPU
  - Scalar (specialized/unspecialized)
  - Vector
  - GPU (Kepler K20)
  - ROOT
- Data transfer in/out is asynchronous
  - 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

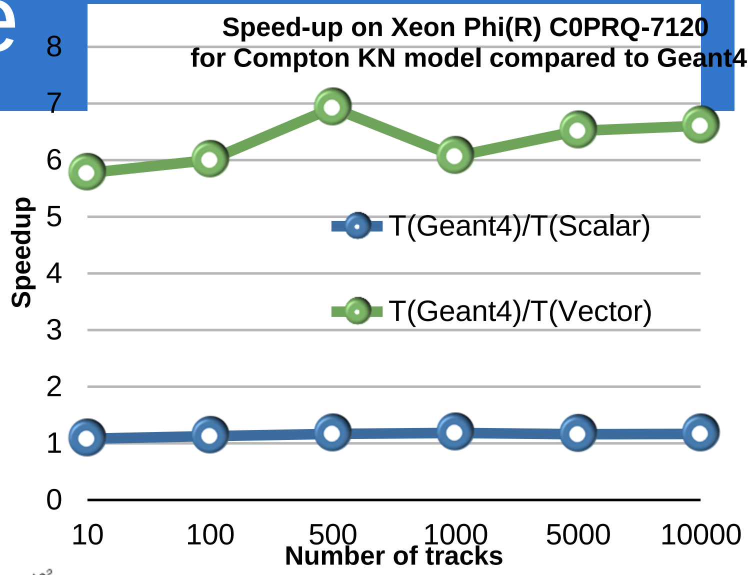


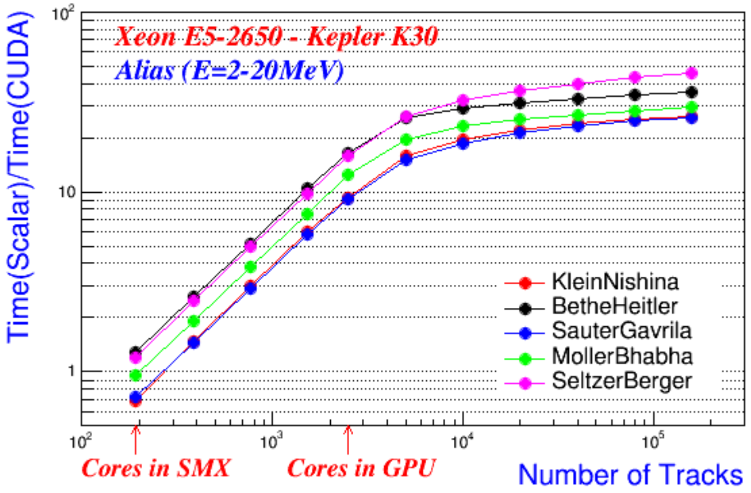
What about physics?

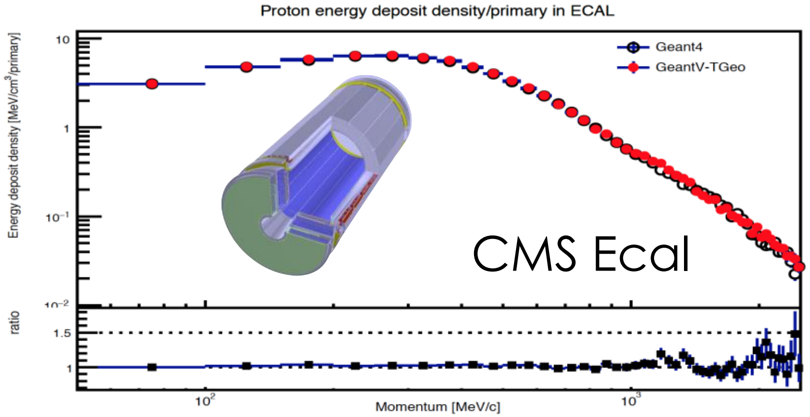
- Needed a “reasonable” shower development
- Developed a library of sampled interactions and tabulated x-sections for GeantV
  - Back ported to Geant4 for verification and comparison
- A quick tool for developing realistic showers
  - Potentially for developing into a fast simulation tool

Physics Performance

- Objective: a vector/accelerator friendly re-write of physics code
- Started with the electromagnetic processes
- The vectorised Compton scattering shows good performance gains
- Current prototype able to run an exercise at the scale of an LHC experiment (CMS)
  - Simplified (tabulated) physics but full geometry, RK propagator in field
  - Very preliminary results needing validation, but hinting to performance improvements of factors







Scheme for GPU

- Broker adapts baskets to the coprocessor
  - Selects tracks that are efficiently processed on coprocessor
  - Gather in chunk large enough (e.g. 4096 tracks on NVidia K20)
  - Transfer data to and from coprocessor
  - Execute kernels
- On NVidia GPU, we are effectively using implicit vectorization
  - Rather than one thread per basket, on the GPUs 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

GPU Status

- Have working broker with ‘geometry only’ kernel
- Just finished updating the tabulated physics code to easily transfer the underlying data tables to the GPU.
- Next
  - Adapt to new navigation code
  - Incorporate Physics code into CUDA Kernel
  - Running the full prototype using GPU as co-processor
  - Understand performance issues and latency limitations of the whole kernel code on NVidia K20
    - Currently no customization of the algorithm for GPU
  - Optimize the application for the GPU
    - Enhance kernel(s) and tune scheduling parameters

Challenges

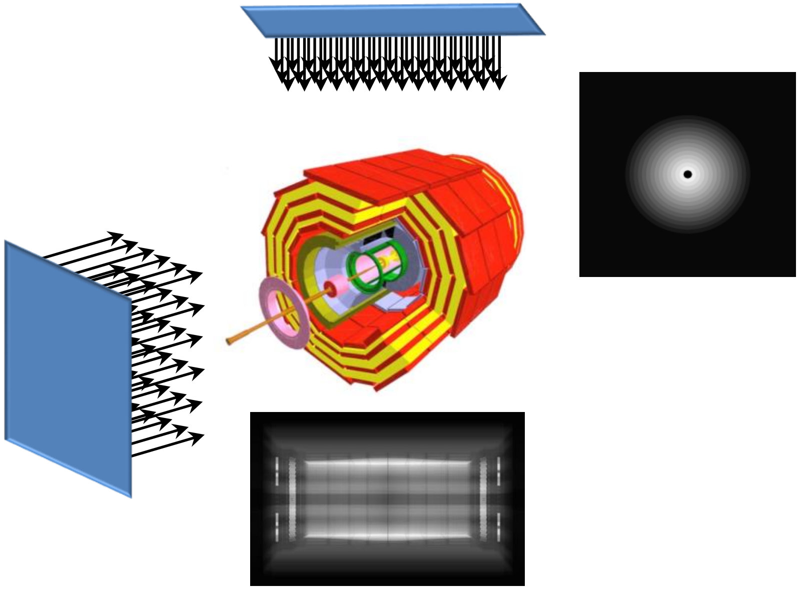
- Large somewhat heterogeneous CUDA kernel
  - One technique tested in previous incarnation of the code is to split the kernel into smaller part and sort the tracks on the device to gather the tracks that will go through the same ‘branch’.
- Historically CUDA profiler gave information on the kernel as a whole
  - Makes it hard to pin point bottleneck or slower portion of the code
  - Is that still the case?
- Tabulated physics ‘likely’ bottleneck will be the high number of often not coalesced memory fetch into the tables.

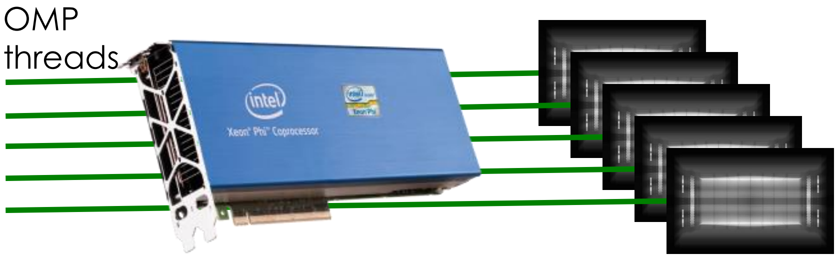
Thank you!

BACKUPS

The X-Ray benchmark

- The X-Ray benchmark tests geometry navigation in a real detector geometry
- X-Ray scans a module with virtual rays in a grid corresponding to pixels on the final image
  - Each ray is propagated from boundary to boundary
  - Pixel gray level determined by number of crossings
- A simple geometry example (concentric tubes) emulating a tracker detector used for Xeon©Phi benchmark
  - To probe the vectorized geometry elements + global navigation as task
  - OMP parallelism + “basket” model

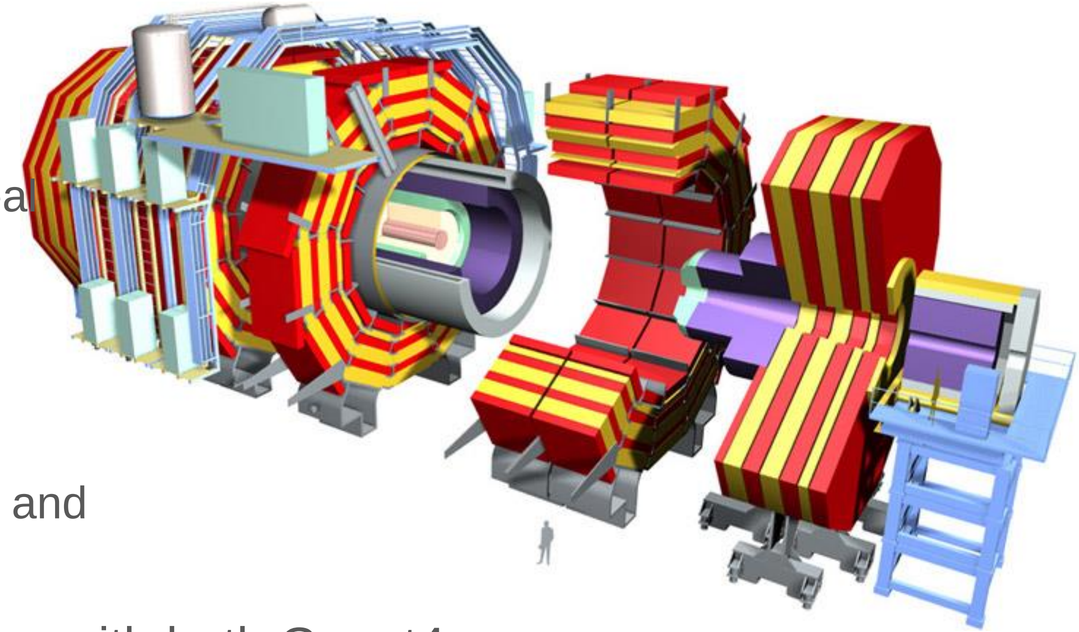




Yardstick: CMS With Tabulated Physics

Realistic Scale Simulation

- pp collisions @ 14TeV minimum bias events produced by Pythia 8
- 2015 CMS detector
- 4T uniform magnetic field
  - Decent approximation of the real solenoidal field
- Low energy cut at 1MeV
- ‘Tabulated’ Physics
  - Library of sampled interactions and tabulated x-sections
- ***Same*** test (described above) run with both Geant4 and GeantV with various versions of the Geometry library.



Putting It All Together - CMS Yardstick

Semantic changes

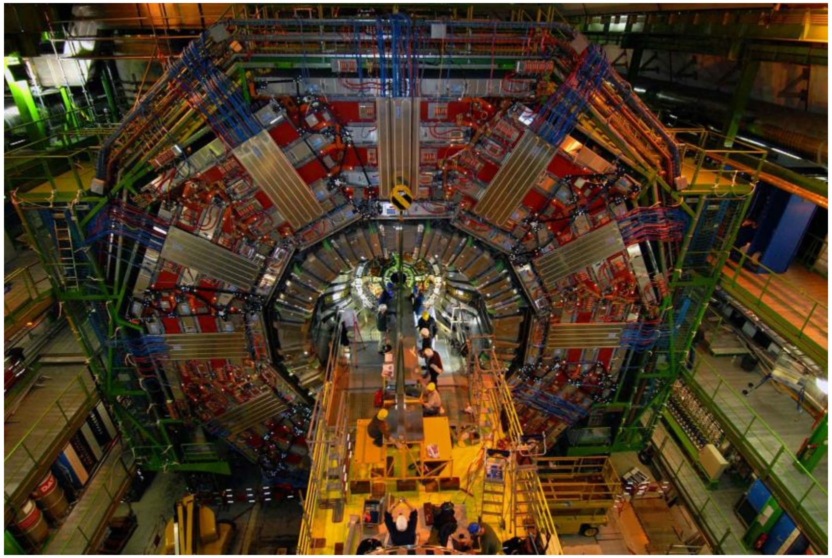
Scheduler	Geometry	Physics	Magnetic Field Stepper
Geant4 only	Legacy G4	Various Physics Lists	Various RK implementations
Geant4 or GeantV	VecGeom 2016 scalar	• Tabulated Physics • Scalar Physics Code	• Helix • Cash-Karp Runge-Kutta
GeantV only	• VecGeom 2015 • VecGeom 2016 vector • Legacy TGeo	Vector Physics Code	Vectorized RK Implementation

Putting It All Together - CMS Yardstick

Semantic changes

Scheduler	Geometry	Physics	Magnetic Field Stepper
Geant4 only	**Legacy G4**	Various Physics Lists	Various RK implementations
Geant4 or GeantV	**VecGeom 2016 scalar** 3	• **Tabulated Physics** • Scalar Physics Code	• Helix (Fixed Field) • **Cash-Karp Runge-Kutta**
GeantV only	• **VecGeom 2015** 2 • VecGeom 2016 vector • **Legacy TGeo** 1	Vector Physics Code	Vectorized RK Implementation

Putting It All Together - CMS Yardstick



- Some of the improvements can be back ported to G4
- Overhead of basket handling is under control
- Ready to take advantage of vectorization throughout.

Improvement Factors (total) with respect to G4

Legacy (TGeo) Geometry library:

- **1.5** → Algorithmic improvements in infrastructure.

2015 VecGeom (estimate)

- **2.4** → Algorithmic improvements in Geometry

Upcoming VecGeom (early result)

- **3.3** → Further Geometric algorithmic improvements and some vectorization