[1]CERN, Geneva
[2]UNESP, São Paulo
[3]Mackenzie Presbyterian University

GeantV - Next generation simulation prototype

Andrei Gheata[1], Guilherme Amadio[2], Calebe de Paula Bianchini[2,3], Federico Carminati[1] *(project PI)*, Sofia Vallecorsa[1] and Sandro Wenzel[1] for the ***GeantV*** Project (geant.web.cern.ch)
andrei.gheata@cern.ch

Introduction

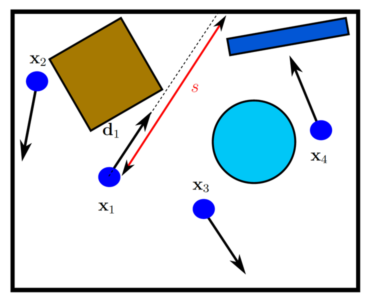
High energy physics experiments such as the ones at the Large Hadron Collider (LHC) at CERN have been using so far most of their worldwide distributed CPU budget – in the range of half a million CPU-years equivalent - to simulate the transport through matter and the effects produced by particles generated in the initial collisions. These simulations are fundamental for understanding both the detector performance and the physics outcome of such an experiment.

The most computing-intensive components of such simulations are the geometry modeling, handling navigation in setups containing millions of objects, and physics, embedding state of the art knowledge of physics models.

Geometry redesign for vectorization

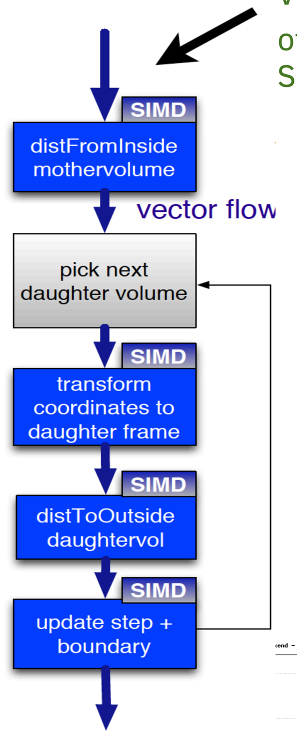
VecGeom is a complete geometry modeler evolved from legacy geometry libraries (Geant4, USolids, ROOT TGeo). It introduces a many-particle API besides the standard scalar one, and relies on templated backend abstraction to enable both platform/architecture specific optimizations and vector/scalar API polymorphism.

Vec(torized)Geom(etry) = Evolved Usolids
+ many-particle API
+ geometry mode/navigation



vectors of particles

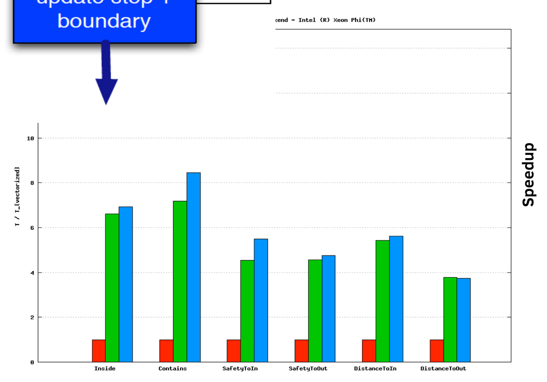
VecGeom can run chains of algorithms in vector/SIMD

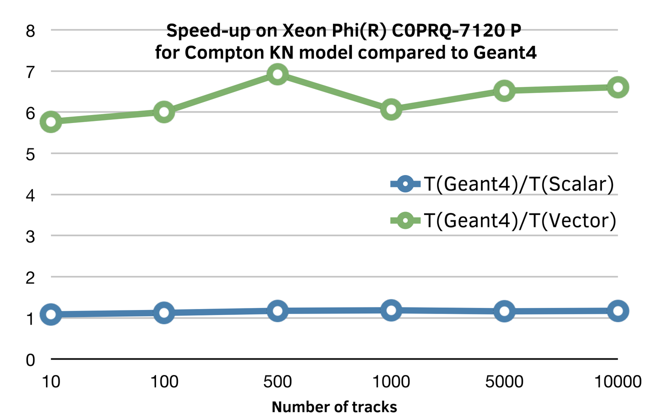


	16 particles	1024 particles	SIMD max
Intel Ivy-Bridge (AVX)	~2.8x	~4x	4x
Intel Haswell (AVX2)	~3x	~5x	4x
Intel Xeon Phi (AVX-512)	~4.1	~4.8	8x

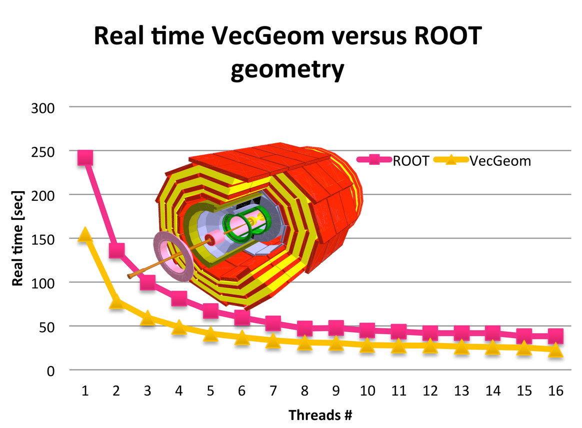
Overall performance for a toy detector (4 boxes, 3 tubes, 2 cones) vs. to ROOT/5.34.17 (http://arxiv.org/pdf/1312.0816.pdf)

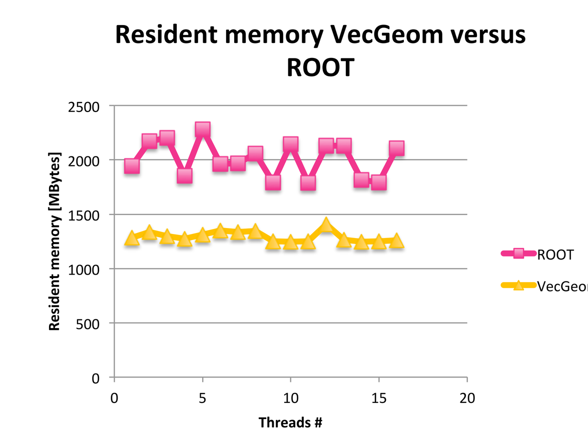
Vectorization performance for shape navigation (left) and physics (right) on a Xeon Phi C0PRQ-7120





We have compared the scalar Haswell performance for GeantV navigation in full CMS (one of the major LHC experiments) geometry. Left, real time for the simulation of 10 pp events at 7TeV using the new VecGom package instead of the existing ROOT geometry. Right side, the resident memory of the full application after compacting the navigation states.

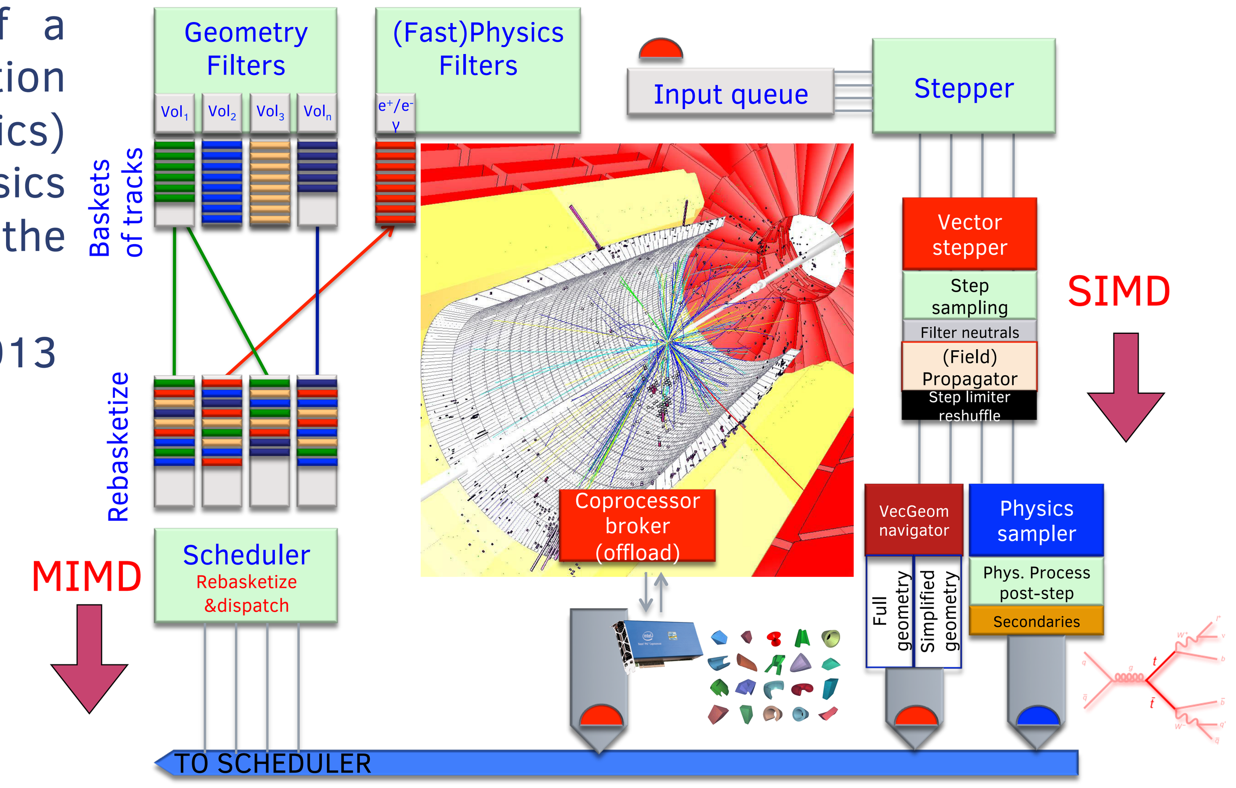




Rethinking particle transport to leverage vectorization

The scope of the project is the development of a community supported, open-source, next generation particle transport code for HEP (High Energy Physics) integrating both detailed and fast simulation physics models and transport algorithms, optimized for the emerging parallel and vector architectures.

- CERN/FNAL/BARC/UniCt-OACT joint project since 2013
- Two Intel®PCCs (CERN via openlab & UNESP)



- Group particles by locality into vectors (baskets)
- Invoke geometry to determine particle position
- Invoke physics models to predict stochastically a process location (interactions with detector material, decays, …)
- Validate proposed physics step against geometry
- Propagate vector of tracks and regroup baskets

Backends and interfaces

Long-term maintainability of the code implies writing one single version of each algorithm and specializing it for the different platforms/technologies using template programming and low level optimized libraries.

- A Xeon®Phi MicVec backend based on intrinsics is in production, inheriting from *F64vec8* class, allowing also to run in offload mode
- A general vectorized backend is implemented using the Vc library (code.compeng.uni-frankfurt.de/projects/vc)
- Backends exist for scalar, CUDA, CILK+, Vc and can be extended to platform/library dependent implementations

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

Scalar interface

```
Vc::double_v distance( Vc::double_v );
```

Vector interface

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

```
struct ScalarBackend
{
   typedef double double_t;
   typedef bool   bool_t;
   static const bool IsScalar=true;
   static const bool IsSIMD=false;
};
```

```
struct VectorBackend
{
   typedef Vc::double_v double_t;
   typedef Vc::double_m bool_t;
   static const boolIsScalar=false;
   static const bool IsSIMD=true;
};
```

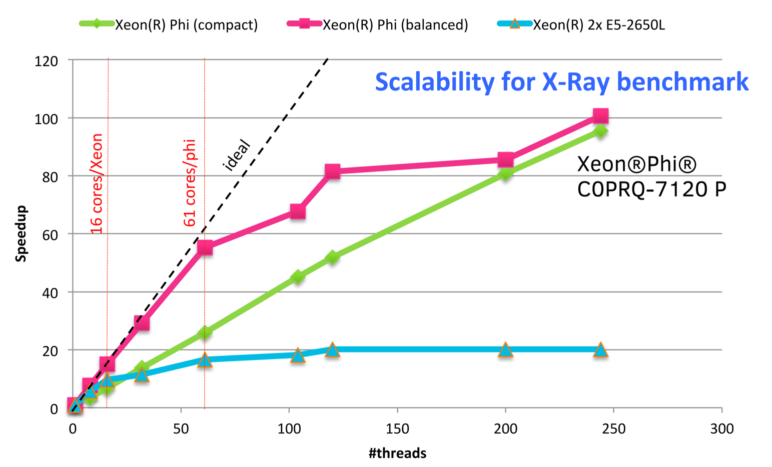
The X-Ray benchmark: Can we harness the Phi for detector simulations?

Scalability of the basketizer behaves better using OMP balanced

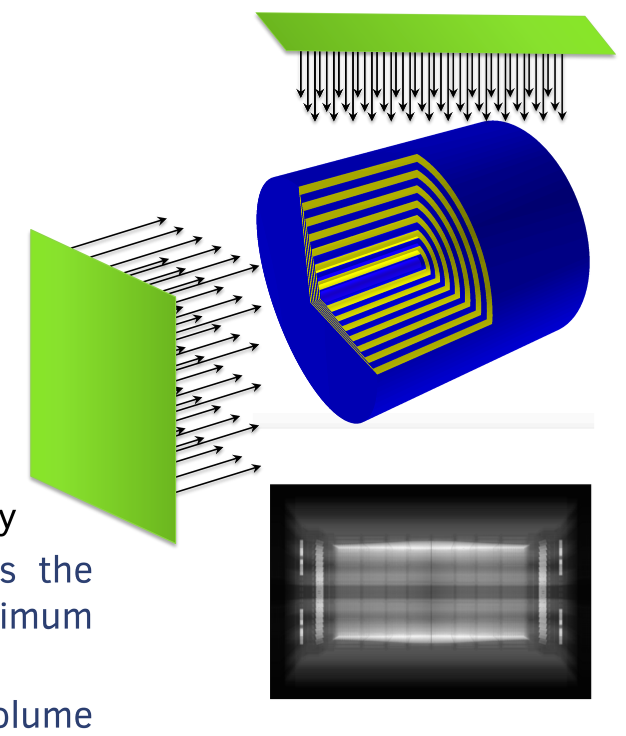
- Approaches well the ideal curve up to native cores count
- Expected performance degradation as more threads are allocated

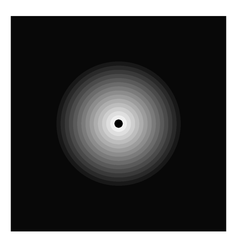
The balanced model converges towards the compact model as all the thread slots are filled

- It's worth to run Xeon Phi saturated for our application



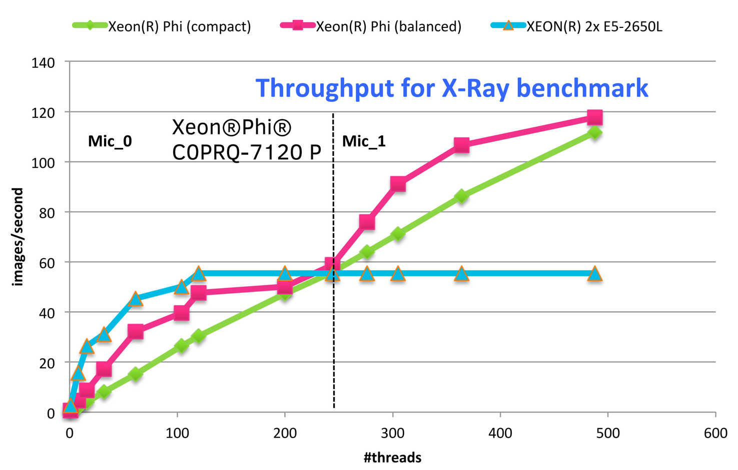
The X-Ray benchmark tests geometry navigation in a real detector geometry, which is one of the main components of GeantV. *X-Ray* comes from the fact that one takes a detector volume (can be the full detector) and scans it with virtual rays (with starting points disposed in a grid) along a given direction and with a given resolution (input parameters). Each ray is propagated from boundary using the *VecGeom* navigator, and the number of crossings is counted until the volume is exited on the other side.





A pixel is produced for each ray having a grey value proportional to the number of crossings.

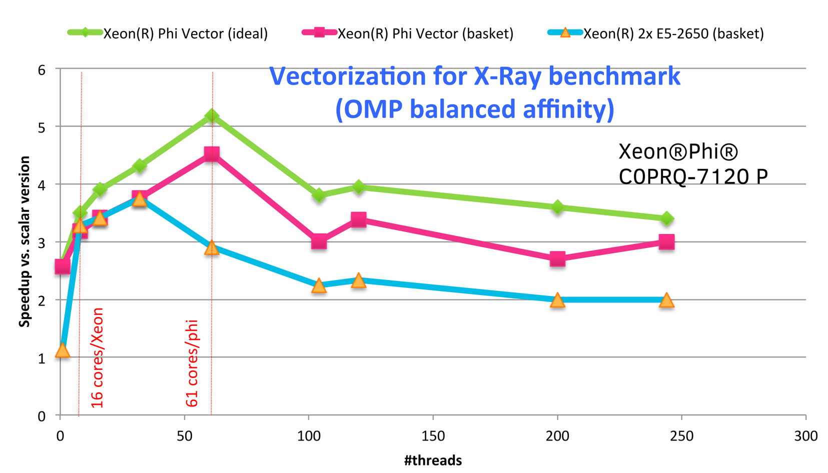
Using a simplified geometry setup emulating a detector tracking system (embedded cylinders). Dispatching one full scan (image) per task



Scalar case: Simple loop over pixels, generating a ray

Ideal vectorization case: Fill vectors with N times the same X-ray, using this as reference for the maximum achievable vectorization

Realistic (basket) case: Fill baskets per geometry volume as particles are entering (as in GeantV)



Gaining up to 4.5 from vectorization when making use of all vector pipelines in the realistic basket case, approaching the ideal vectorization case (when no regrouping of vectors is done) .

- Vector starvation starts to pop-in fast when filling more thread slots than the core count, but the performance loss is not dramatic
- We get expected better vectorization compared to the Sandy-Bridge host

The throughput tests were currently done on a single KNC card C0PRQ-7120P, extended to reflect a 2 card scenario

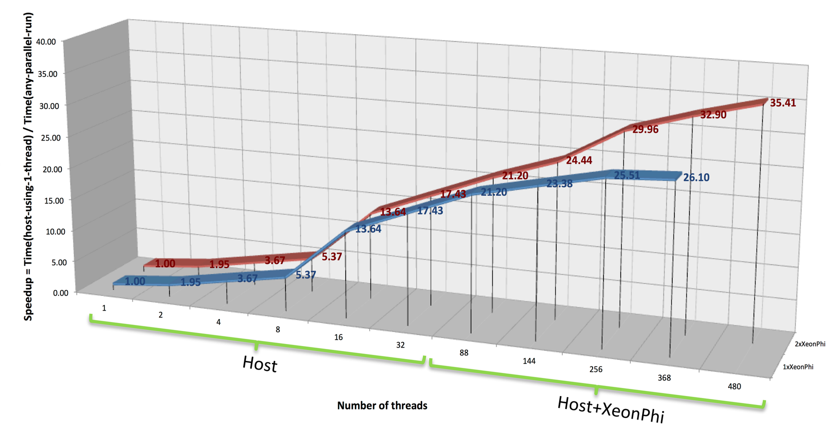
- The throughput performance for a saturated KNC is equivalent (for this setup) to the dual Xeon E5-2650L@1.8GHz server which hosts the card.

Offloading simulations on the KNC

We have tested the functionality of running GeantV tasks (scalar X-Ray benchmark) in offload mode, in a heterogeneous environment having one host and 2 Xeon© Phi cards. This was a preliminary performance measurement before enabling vectorization in our benchmark. The offload was split among the host and 2 Xeon Phi cards, demonstrating good scalability.

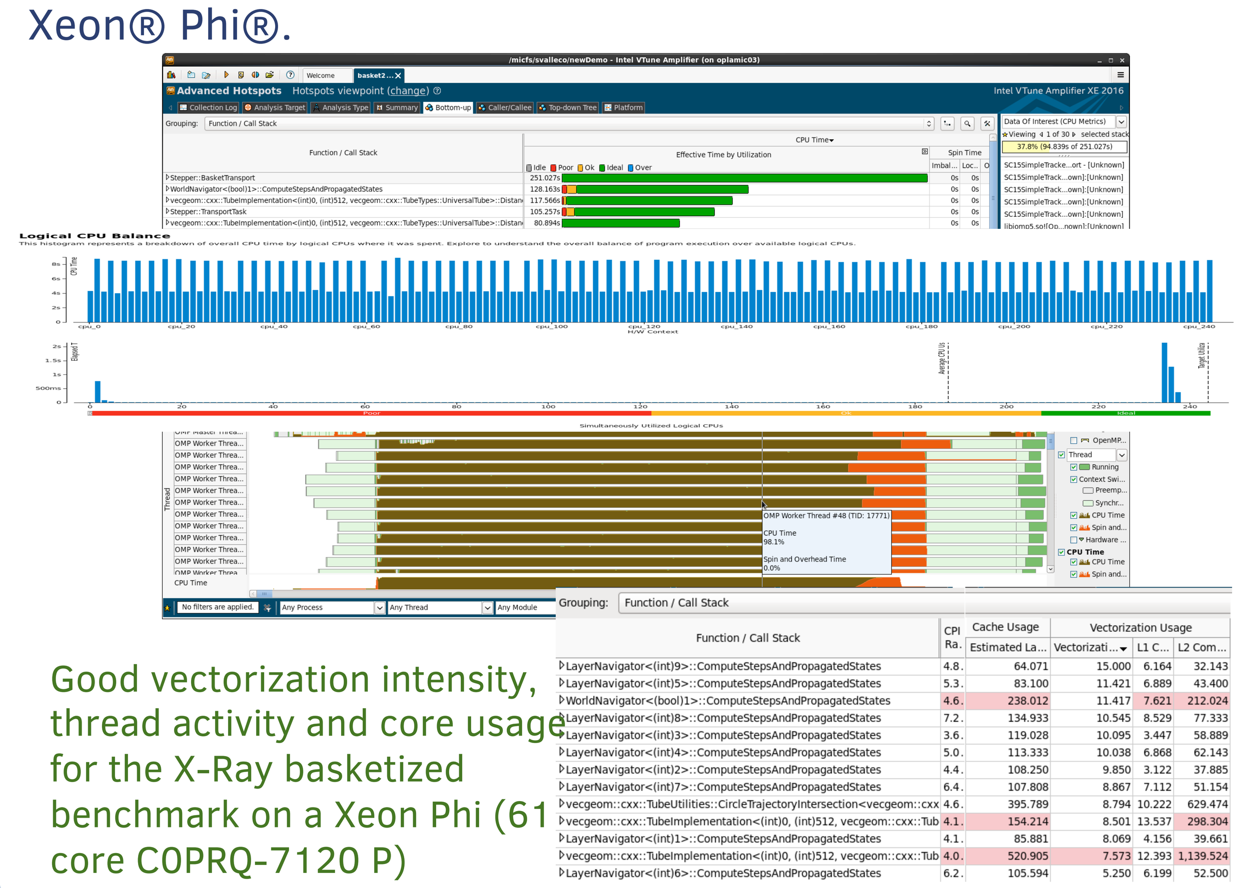
2 x Intel(R) Xeon(R) CPU E5-2670
(2 x 16 threads, 2.60GHz, 64GB RAM)

2x Xeon Phi C0QS-3120 P/A
(224 threads, 57 cores, 6GB GDDR5)



Profiling with Intel Performance Tools

The performance tools were extensively used to understand the current performance of GeantV. Below is an illustration of the VTune outputs for the X-Ray benchmark done on a Xeon® Phi®.



Good vectorization intensity, thread activity and core usage for the X-Ray basketized benchmark on a Xeon Phi (61 core C0PRQ-7120 P)