FERMILAB-CONF-16-625-CD
ACCEPTED

GeantV: from CPU to accelerators

G Amadio[1], A Ananya[2], J Apostolakis[3], A Arora[2], M Bandieramonte[3], A Bhattacharyya[4], C Bianchini[1,5], R Brun[3], P Canal[6], F Carminati[3], L Duhem[7], D Elvira[6], A Gheata[3], M Gheata[3,8], I Goulas[3], R Iope[1], S Jun[6], G Lima[6], A Mohanty[4], T Nikitina[3], M Novak[3], W Pokorski[3], A Ribon[3], R Sehgal[4], O Shadura[3], S Vallecorsa[3], S Wenzel[3], Y Zhang[3]

[1] UNESP, Universidade Estadual Paulista (BR)
[2] IIT- Indian Institute of Technology (IN)
[3] CERN - European Organization for Nuclear Research - Geneva, Switzerland
[4] Bhabha Atomic Research Center (IN)
[5] Mackenzie Presbyterian University (BR)
[6] Fermi National Accelerator Laboratory (US)
[7] Intel Corporation
[8] Institute of Space Sciences (RO)

Andrei.Gheata@cern.ch

Abstract. The *GeantV* project aims to research and develop the next-generation simulation software describing the passage of particles through matter. While the modern CPU architectures are being targeted first, resources such as GPGPU, Intel© Xeon Phi, Atom or ARM cannot be ignored anymore by HEP CPU-bound applications. The proof of concept *GeantV* prototype has been mainly engineered for CPU's having vector units but we have foreseen from early stages a bridge to arbitrary accelerators. A software layer consisting of architecture/technology specific backends supports currently this concept. This approach allows to abstract out the basic types such as scalar/vector but also to formalize generic computation kernels using transparently library or device specific constructs based on Vc, CUDA, Cilk+ or Intel intrinsics. While the main goal of this approach is portable performance, as a bonus, it comes with the insulation of the core application and algorithms from the technology layer. This allows our application to be long term maintainable and versatile to changes at the backend side. The paper presents the first results of basket-based *GeantV* geometry navigation on the Intel© Xeon Phi KNC architecture. We present the scalability and vectorization study, conducted using Intel performance tools, as well as our preliminary conclusions on the use of accelerators for *GeantV* transport. We also describe the current work and preliminary results for using the *GeantV* transport kernel on GPUs.

1. Vertical scaling for particle transport simulation

LHC has intensively used horizontal scaling for the most important HEP computing tasks, namely simulation, reconstruction and analysis. The LHC computing GRID [1] allowed running hundreds of thousands of concurrent jobs taking advantage of distributed computing resources and relying on the fact that events are uncorrelated. Measurements [2] have revealed however a rather poor usage of the silicon per socket, interpreted to be due to instruction fetch and data locality issues. Correlating this

Operated by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the United States Department of Energy.

with the lack of awareness of HEP software to features common to modern processors such as instruction level parallelism and SIMD, we concluded that there was a very large potential for improvement.

The *GeantV* simulation prototype introduced the concept of vectorized processing in particle transport [3], enabled by a so-called *basketizing* procedure that feeds vectors of particles to floating-point intensive geometry and physics algorithms. The goal is to make use of the SIMD vector units available on modern processors and to optimize the use of caches by profiting from the geometry and physics locality as much as possible. The design of the *GeantV* vector prototype has been described in detail in other papers [4,5]. The idea is to leverage the vertical dimension of scalability by making maximum use of the silicon available on a single socket, and to study the feasibility of using coprocessors and accelerators such as Xeon Phi and GPU's. The project aims to eventually provide the path to a long-term solution for particle transport simulation working in both general purpose and HPC environments.

The different pieces composing *GeantV* - such as concurrent scheduling, geometry and physics - are currently being built and in different stages of development. This is an iterative process, as we learn that some approaches are better than others. In terms of porting the vectorized CPU prototype on accelerators, running full *GeantV* in native mode is the preferred but not always possible solution. For the GPU we are offloading modules of *GeantV* as kernels, coming with an additional overhead of having to handle data transfers between the host and device, as shown in the next section.

2. GeantV in native and offload mode

The design of *GeantV* has foreseen creating *baskets* (vectors of tracks located in the same geometry volume of having similar physics properties) of filtered tracks and dispatching them to processing units. These can be threads of the same process, different processes or arbitrary co-processors acting as accelerators. The track data is passed to a broker that can collect as many tracks as deemed efficient for the co-processor, taking care of shipping the data to and from the device. This run mode can only be efficient if the data transfer back and forth is asynchronous and concurrent in order to mitigate the data transfer latency. The GPU broker, for example, collects baskets until the number of contained tracks is equal to the number of GPU cores, and then serializes them to GPU. Each track is given to a different thread, which propagates it until the next geometry boundary or physics process. The broker receives the tracks back and gets them "*basketized*" again by the CPU scheduler, as shown in Figure 1.

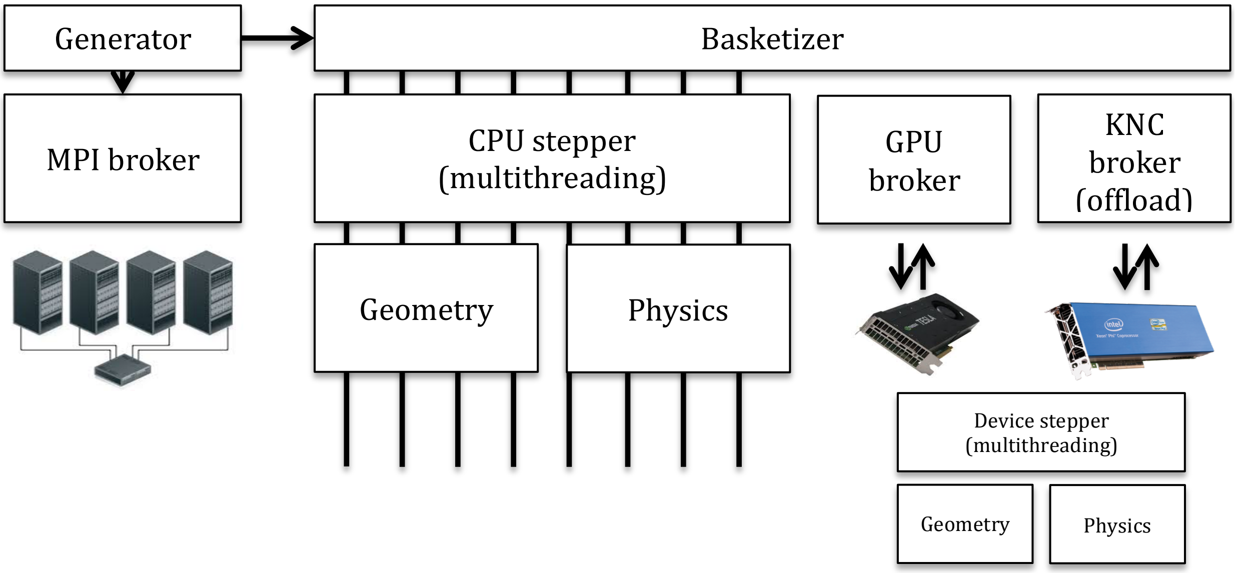

Figure 1 ***GeantV*** **scheduling in native and offload modes. The coprocessor brokers can work fine grain processing a set of baskets and sending transported tracks back to host, or MPI mode fetching data from an event of file server.**

The workflow described above has been tested for the GPU *Kepler* cards using *CUDA7*, and currently is being optimized. The same approach was initially foreseen also for the Intel® KNC device broker but it was eventually dropped due to difficulties in managing asynchronous data transfers from the host, using the KNC offload mechanism. We decided to rather use the Xeon Phi co-processor in native/offload modes to test the behavior of specific *GeantV* geometry and physics workloads, without asynchronous communication. This helped us understanding performance features on this architecture for some of our main software components, targeting a full *GeantV* deployment on the future Intel®KNL. For the offload mode the device library was cross-compiled on the host and the deployment of the work was done using *OpenMP's #pragma offload* directive within a dedicated *MicVec* backend.

Another use case foreseen for *GeantV* is running in native mode on some of the accelerators. This will be the preferred run mode on *KNL* considering the overhead introduced by offloading. The pre-requisites are the native compilation of the full *GeantV* library on the device and a custom tuning of the scheduling parameters to optimize the workflow depending on the device architecture.

An important feature of the *GeantV* library allowing compiling the same code on both host and device modes is the concept of backend interfaces [6, 7]. The idea behind this is that the main differences across platforms related to data types, library-dependent implementations, or scalar versus vector handlers, can be wrapped into backend-dependent structures. Template specializing on the backend type makes then a common source code available across libraries and different technologies in both scalar and SIMD vector mode when available. A backend approach insulates the code base from the heterogeneous technology layer and makes maintenance of the code much easier. It is clear that a considerable effort shifts into the development and maintenance of the backend interfaces – currently the supported ones being the *scalar*, *CUDA*, *Vc*, *MicVec* and *UME::SIMD* (using Intel native intrinsics).

We are currently re-shaping the *GeantV* scheduler to allow running in native mode on the KNL Xeon Phi, implying a deep redesign to provide better scalability and NUMA awareness. On the GPU side, the CUDA backend allows already running a full transport step (propagation of a vector of particles to the current geometry volume limits or the simulation of the next physics process) but currently the rescheduling has to be done still on the host. We do not exclude the possibility to eliminate this limitation in future. Dispatching work to accelerators running in native mode can be done either by MPI or their native communication interface, considering that the input data required by particle transport is lightweight compared to the amount of processing performed per track.

3. Physics and geometry benchmarks on accelerators

We have investigated in a first step the performance of *GeantV* geometry and physics benchmarks on *Kepler K20* and *Intel®Xeon Phi C0PRQ-7120* (KNC) cards. We have tried to understand the performance limits on such devices for elementary algorithms such as geometry intersection for different solids or simple physics models. The two architectures are complementary as many-core topologies, exposing different parallelism use cases. While the *Kepler* provides 2496 lightweight cores which are suitable for track-level parallelism, the *Intel®KNC* provides 60 cores running up to 4 threads per core with access to 512 bits vector registers, allowing to profit from the *GeantV* basket model, enabling multi-track vectorization.

For the geometry case, we have tested the speedup compared to *Geant4* [8] and *ROOT* [9] implementations across different platforms and for different vector lengths. Table 1 presents such a comparison, combining both scalar optimizations (with respect to the equivalent ROOT algorithms) and vector gains for a simplified detector setup using a simple navigation, which examines all daughter volumes (i.e. without geometry voxel optimizations). Standard intersection benchmarks for solids such as boxes, trapezoids or tubes are showing very good acceleration of often more than a factor of 4 on KNC. We note that for the most complex algorithms such as the polycone or polyhedron intersection, the vector gains are reduced (sometimes less than x2) due to their multiple branches, which gets penalized when mapped to mask operations. In such complex cases we are seeking alternatives, e.g to profit from vectorizing inner loops on surfaces or to provide better algorithms. Figure 2 presents the performance for the different box shape algorithms on the K20, showing how the die can be saturated with large track containers running in a single kernel.

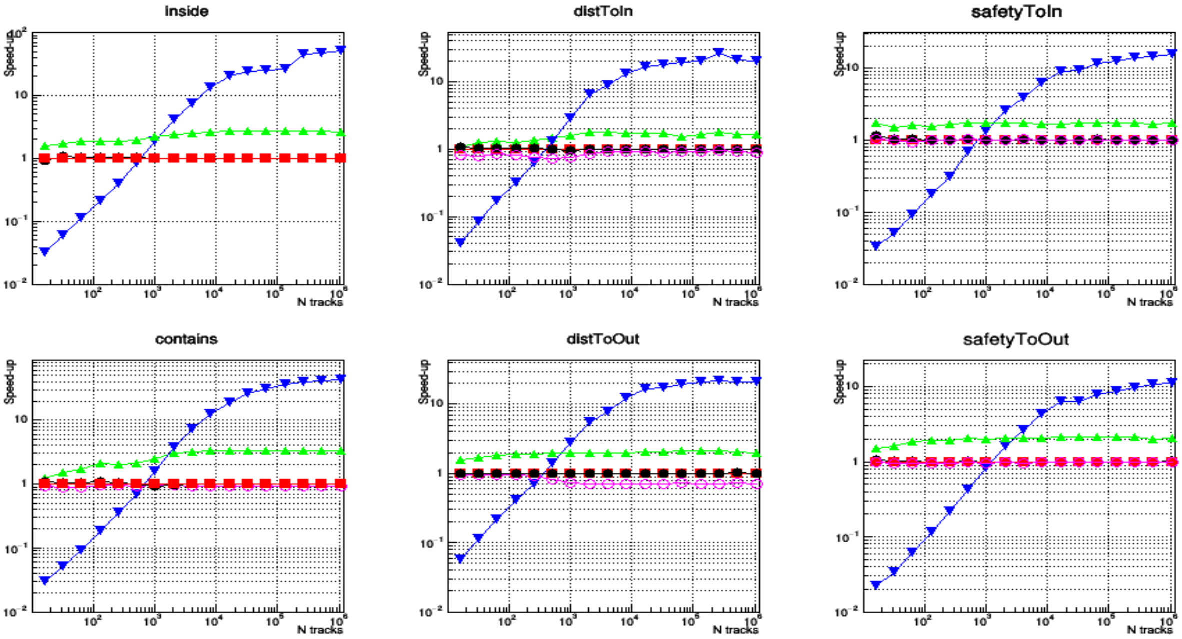

Figure 2 Performance of box navigation algorithms on K20 GPU compared to scalar/vector modes on CPU and compared to equivalent ROOT algorithms. The speedup is computed as $T_{scalar}/T_{vector/GPU}$

The latest versions of CUDA allow running multiple kernels on the same die, for workloads processing smaller data units such as the *GeantV* baskets. Note that, due to its lower performance per core the GPU needs at minimum about 10^3 tracks to match the scalar CPU performance.

Architecture	16 particles	1024 particles	SIMD max
Intel © Ivy-Bridge (AVX)	2.8	4.0	4.0
Intel © Haswell (AVX2)	3.0	5.0	4.0
Intel © Xeon Phi (Intel®IMCI)	4.1	4.8	8.0

Table 1 Overall performance for particles traversing a simplified detector, compared to ROOT v5.34.17. We compare the speedup of the vector version compared to the scalar version of ROOT running on the same architecture.

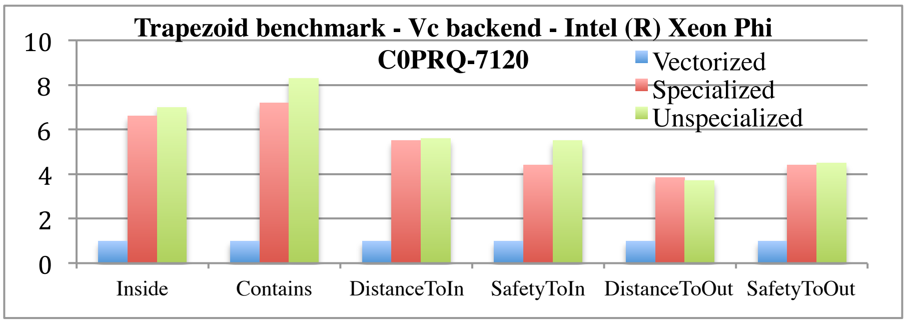

Figure 3 Relative time normalized to the vectorized case for different trapezoid shape intersection algorithms (Xeon®Phi C0PRQ-7120 P

The development of vector-aware physics algorithms has recently started in the *GeantV* project. We are trying to optimize at first common algorithms for distribution sampling, random number generators and an initial set of physics model implementations for gamma processes.

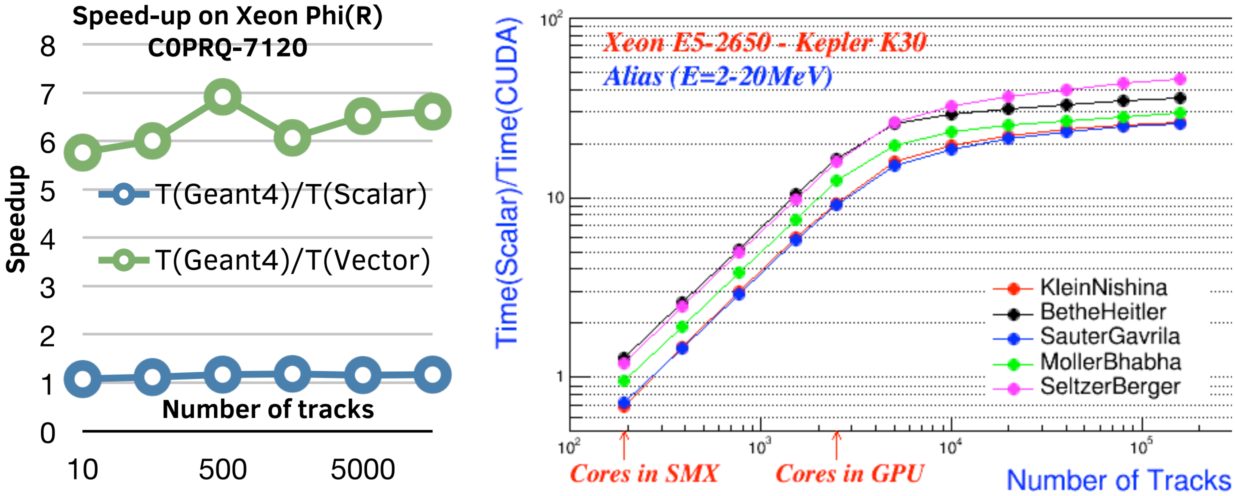

Figure 4 Vectorized Compton process on KNC (left). Performance of the alias sampling method for different process cross-sections (right).

Figure 4 (left) presents the performance of the vectorized Compton physics model algorithm on KNC card, which uses the vector unit of the accelerator close to its ideal performance, while the scalar version has basically the same performance as the corresponding *Geant4* one. We have tested alias sampling on different electromagnetic models' distributions (having quite different profiles) on the *Kepler* card [10]. The performance profile keeps the same shape but can vary with as much as 50% for different distributions.

4. Performance of combined algorithms on accelerators: the X-ray benchmark.

An important question is the relevance of vector optimizations of elementary algorithms for the complete simulation workflow: is it possible to maintain the performance improvements shown above in the combined algorithm case? It is clear that some of the processing steps connecting different vectorized algorithms may represent scalar bottlenecks that will have direct impact on performance. It is very important to first identify these scalar bottlenecks, then try to minimize them.

We have tried to understand the vector optimization procedure by running a composite benchmark on the Intel® Xeon Phi *C0PRQ-7120* card. The standard Xeon-based benchmark uses a part of the full detector geometry and runs multi-particle propagation traversing through it as shown in Figure 5. This operation is performed during the *GeantV* stepping but there it competes with the active physics models. This benchmark performs just the geometry navigation part of the task and creates an "X-ray" virtual image on the other side. A pixel's gray level depends on the number of different volumes traversed by each track. We looked at some basic quantities such as scalability and throughput, and moved on to in-depth *VTune* profiling to understand the bottlenecks.

The detailed results of this procedure are not reproduced in this paper but we have clearly identified the particle relocation algorithm as one of the important scalar bottlenecks. A possibility to optimize relocation is to understand geometry navigation from a heuristic perspective. As a simple example, a calorimeter detector is composed of possibly millions of elements, out of which often just a few logically different volumes. In addition, these volumes are disposed symmetrically and in a lattice manner, so that each cell has a very limited number of neighbors. We are currently investigating and prototyping methods to optimize navigation for this kind of topologies, giving very promising results so far and very likely be used in future by the framework.

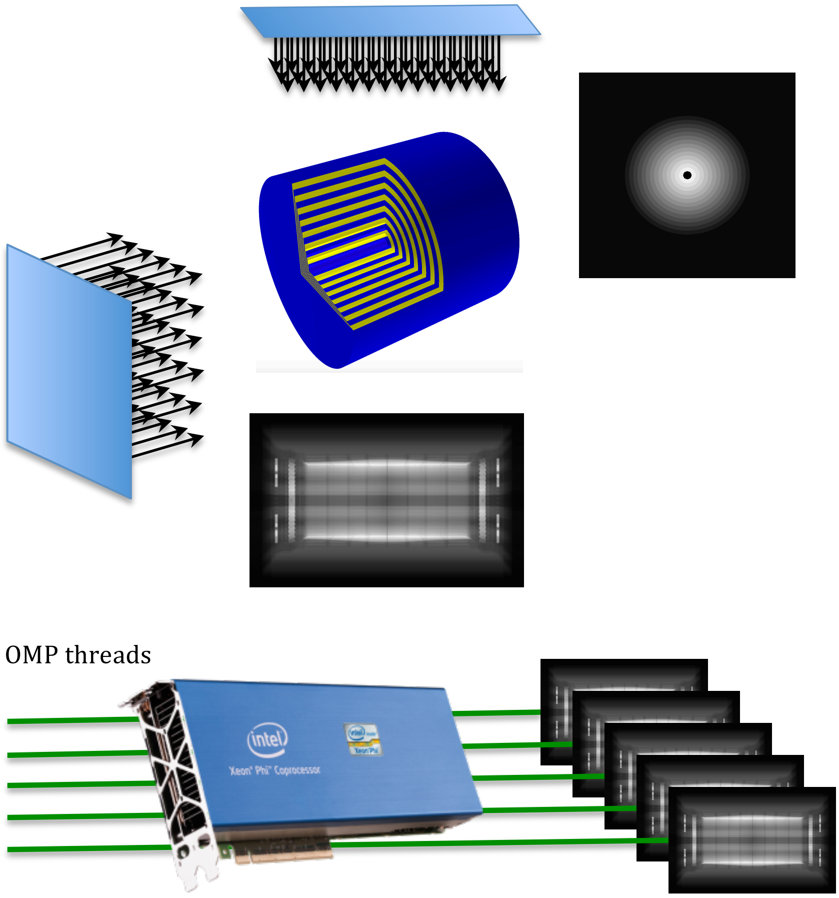

Figure 5 The X-Ray benchmark tests geometry navigation in real detector geometry, which is one of the main components of GeantV. The name comes the fact that one takes a detector geometry module and shoots virtual rays (with starting points disposed in a grid) along a given direction and with a given resolution (input parameters). Each ray is propagated from boundary to boundary using the VecGeom navigator, and the number of crossings is counted until the volume is exited on the other side. The model for parallelism uses *OpenMP* to dispatch different scans to different threads.

For our optimization purpose, we have used a simplified geometry setup emulating concentric cylindrical tracking detectors, where crossing volume boundaries are predictive. We attached a basketizer to each of these detectors, aiming to form vectors of tracks of fixed size before invoking geometry navigation algorithms. This reproduces to some extent the actual re-basketizing schema used in *GeantV*, including the rundown phase when the partially filled baskets have to be processed at the end of the simulation.

The setup is presented in Figure 5. The exercise's objective was to saturate the processing pipelines of a Xeon Phi card. To achieve this goal we steered multithreading using OpenMP, parallelizing on the loop of X-ray scans to be performed: each thread had to complete full scans over a fixed-size grid of 1024x1024 pixels. We have tested both the compact and balanced OpenMP thread scheduling, using an increasing number of threads, up to 4 threads per physical core of the Xeon Phi card. We have measured both scalability and the image throughput, comparing them with the same benchmark done on the *IvyBridge* dual processor machine hosting the KNC card.

5. Results and conclusions

The benchmark was run with both compact and balanced OMP modes. In balanced mode it showed an excellent scalability approaching the ideal case up to the physical core count. We observed expected limited performance increase as more threads are allocated. The balanced model converges towards the compact model as all thread slots are filled. The conclusion was that running Xeon Phi fully saturated brought the maximum benefit for this application. 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.

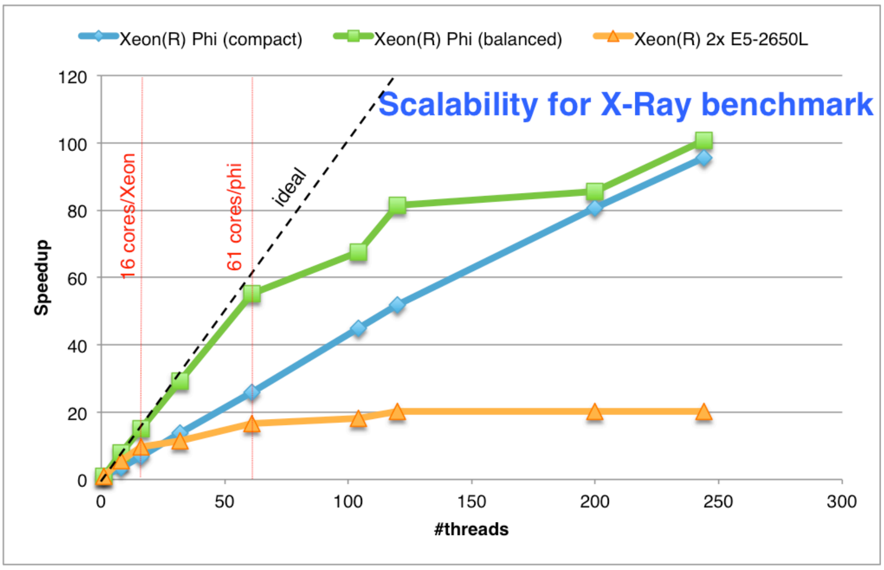

Figure 6 X-ray benchmark scalability on Xeon Phi and IvyBridge for compact and balanced thread allocation. The Xeon server shows unexpected benefits even running beyond its hyper-threading regime, hinting to vectorization as main responsible.

To understand the vectorization features of the model, we have run the benchmark and studied three different cases:

- *Scalar case*: Simple loop over pixels, generating a ray for each pixel and transporting it in scalar mode to the exit of the detector.
- *Ideal vectorization case*: Filling each vector with N times the same X-ray, and then providing this as input to our geometry propagator vector API. This is of course not realistic as the work for each track is redone the vector size times but this was useful to be used as a reference for the maximum achievable vectorization.
- *Filtering/regrouping case*: Filling vectors of tracks grouped per geometry volume and regrouping tracks to the appropriate baskets as particles are entering the next volume (similar to *GeantV approach*). Transport is triggered when vectors reach a given size.

In Figure 7 we plotted the speedup of the ideal and realistic cases with respect to the scalar approach with the same number of threads. We also plotted the vector performance on the IvyBridge host for comparison. While some of the vector efficiency is lost compared to the ideal case, we observed gains up to 4.5 from vectorization in basketized mode, with vector starvation showing up beyond the core count. A more detailed analysis using Intel *VTune* is ongoing in order to understand better the source of contention.

An important conclusion to our benchmark was that navigation specialization could bring benefits to the vectorization performance of *GeantV*. While studying a non-trivial yet idealized case, we have confirmed that a new level of optimization for geometry navigation is possible and is needed for obtaining additional performance on both standard Xeon processors and Xeon Phi cards. In addition, the results on KNC are promising for future migration of *GeantV* code to many core architectures such as the Intel®Knights Landing.

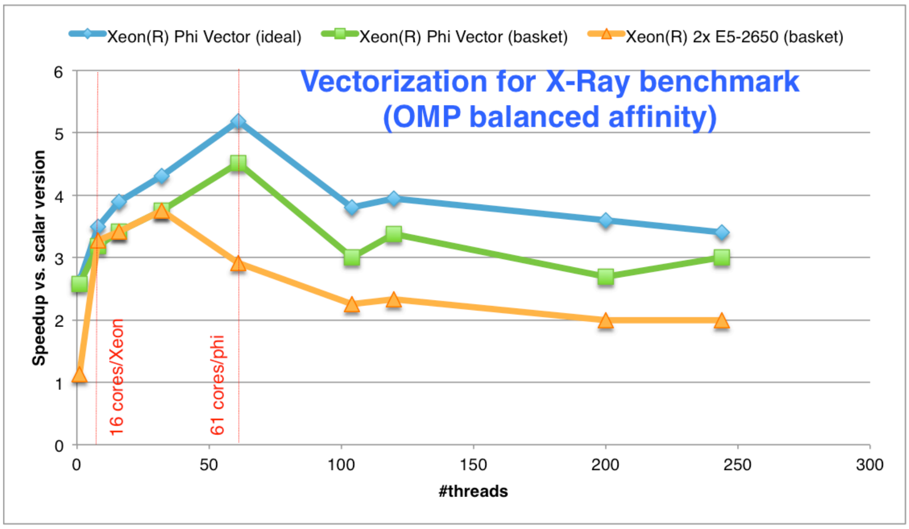

Figure 7 Vectorization performance on Xeon Phi (KNC) for the basketized X-ray benchmark, compared to idealized performance of the same setup and with IvyBridge. The speedup is defined as T_{scalar}/T_{vector} for the same number of threads. The decreasing slope at large number of threads is not an absolute loss in performance but a relative one (i.e. the scalar version scales better than the vector)

In parallel with the *GeantV* steering framework, the development of vectorized algorithms based on our approach using backends is well advanced in case of geometry and in an initial phase in case of physics. Both geometry and physics benchmarks demonstrate good performance so far on co-processors. Development plans for the next phase aim to provide full built-in accelerator awareness for *GeantV*-based simulation on both GPGPU and KNL processors.

Acknowledgements

The authors wish to acknowledge the contribution of Intel to the *GeantV* project through the Intel Performance Computing Centre (IPCC) program. We also want to acknowledge the very useful technical contributions of the CERN openlab, which allowed performing these benchmarks.

References

[1] M Lamanna 2004 *N.I.M. in Phys. Res.* **A 534**(1):1-6.
[2] S Jarp, A Lazzaro, J Leduc and A Nowak 2012 *Comp. High Energy and Nucl. Phys. (CHEP)* https://indico.cern.ch/event/149557/session/6/contribution/486
[3] Apostolakis J, Brun R, Carminati F and Gheata A 2012 *J.Phys: Conf. Ser.* **396** 022014 http://iopscience.iop.org/1742-6596/396/2/022014
[4] Apostolakis J, Brun R, Carminati F, Gheata A and Wenzel S 2014 *J. Physics: Conf. Ser.* **513** 052006
[5] J Apostolakis et al 2015 *J. Phys.: Conf. Ser.* **608** 012003
[6] Wenzel S 2014 Towards a high performance geometry library for particle-detector simulation *16th Intl. workshop on Adv. Comp. and Analysis Techniques in phys. Res. (ACAT)*
[7] de Fine Licht J 2014 First experience with portable high-performance geometry code on GPU *GPU Computing in High Energy Physics*
[8] S Agostinelli et al 2003 *Nuclear Instruments and Methods* **A 506** (53pp)
[9] http://root.cern.ch
[10] G Amadio et al 2016 Electromagnetic Physics Models for Parallel Computing Architectures *17th Intl. workshop on Adv. Comp. and Analysis Techniques in phys. Res. (ACAT)*