FERMILAB-CONF-17-045-CD

Performance of GeantV EM Physics Models

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

[1]Parallel Computing Center at São Paulo State University (UNESP), São Paulo, Brazil
[2]CERN, EP Department, Geneva, Switzerland
[3]Bhabha Atomic Research Centre (BARC), Mumbai, India
[4]Fermilab† , MS234, P.O. Box 500, Batavia, IL, 60510, USA
[5]Intel Corporation, Santa Clara, CA, 95052, USA
[6]Mackenzie Presbyterian University, São Paulo, Brazil
[7]Institute of Space Sciences, Bucharest-Magurele, Romania

E-mail: `syjun@fnal.gov`

Abstract. The recent progress in parallel hardware architectures with deeper vector pipelines or many-cores technologies brings opportunities for HEP experiments to take advantage of SIMD and SIMT computing models. Launched in 2013, the GeantV project studies performance gains in propagating multiple particles in parallel, improving instruction throughput and data locality in HEP event simulation on modern parallel hardware architecture. Due to the complexity of geometry description and physics algorithms of a typical HEP application, performance analysis is indispensable in identifying factors limiting parallel execution. In this report, we will present design considerations and preliminary computing performance of GeantV physics models on coprocessors (Intel Xeon Phi and NVidia GPUs) as well as on mainstream CPUs.

1. Introduction

Recent discoveries in experimental High Energy Physics (HEP) would not be possible without leveraging advances in scientific computing, especially in the areas of simulation, reconstruction, and physics analysis for large-scale data sets. For more than two decades, HEP programs have taken advantage of automatic performance gains coming from increases in processor clock speed and high throughput computing using either local clusters (Farms) or distributed resources around the world (GRIDs). However, the hardware landscape has significantly changed toward parallel computing architecture, but the code developed and used within HEP is not yet utilizing the increased length of the number of threads or vector processing units [1]. Furthermore, demands for computing power are ever-increasing for future HEP programs, especially for High-luminosity LHC experiments. In order to overcome computing challenges ahead of the community and to benefit from the full processing power of the latest chips, we must improve our software architecture and increase the amount of work that can be done in parallel within a similar memory budget.

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

Within the HEP software ecosystem, event simulation is one of the most time consuming parts of the work flow. However, the basis of HEP detector simulation is mostly independent from the details of individual experiments and thus is easy to share among experiments. Consequently, any run-time performance improvement in physics event simulation can have a significant impact on the amount and quality of HEP simulation overall.

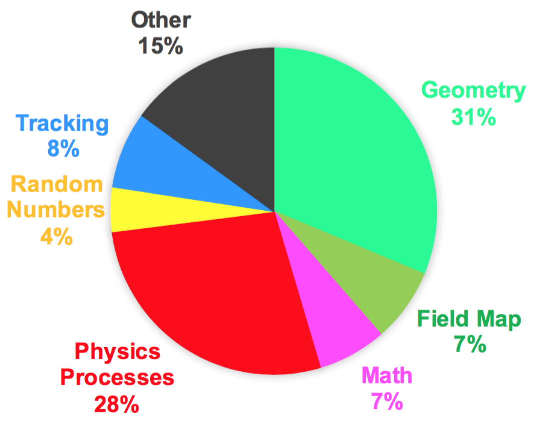
Geant4 [2, 3, 4] is the most widely used simulation toolkit in contemporary HEP experiments, but does not efficiently utilize the vector capability of modern CPUs as it has been designed for sequential executions. To explore emerging computer technologies in order to significantly increase run-time performance of detector simulation, the GeantV project [5, 6] was launched in 2013. The project studies performance gains from propagating multiple tracks from multiple events in parallel, improving code and data locality in the process. Using code specialized to take advantage of the hardware specifics, it aims to leverage both the vector pipelines in modern processors and the availability of coprocessors, including the Xeon Phi and general purpose GPUs. GeantV, as the next generation simulation toolkit, will provide SIMD (Single Instruction Multiple Data) and SIMT (Single Instruction Multiple Threads) capable components of concurrent framework, geometry and navigation, physics and other services for HEP particle transport working on modern hardware. In this paper we focus on the techniques explored to enhance the existing electromagnetic (EM) physics models utilizing parallel hardware architectures.

2. Vectorization of Physics Models

The typical HEP event consists of a set of particles produced by a primary collision and subsequent secondary interactions or decays. Conventional HEP detector simulation processes all tracks of an event sequentially even though multiple events can be processed simultaneously (event-level parallelism) using multiple processors or threads. In contrast, GeantV explores particle-level parallelism by grouping similar tracks and processing them in a vectorized manner to maximize locality of both data and instructions. To take full advantage of SIMD or SIMT architectures, identical operations should be executed on multiple data which requires substantial re-engineering HEP software and computing models.

Due to the complexity of the geometry and magnetic field description used for general-purpose collider detectors or many other types of experiments, navigation in geometry usually consumes a significant fraction of simulation time. VecGeom [7, 8], as the backbone of the GeantV geometry implementation, has been developed to support multi-particles transport in a vectorized manner on modern hardware platforms using the concept of backend interfaces and has already demonstrated significant performance gains on both CPU and coprocessors. The current status of VecGeom and its performance can be found in detail elsewhere [9]. However, it is important to note that vectorization of physics is also indispensable to avoid scalar bottlenecks in GeantV. The pie chart in Figure 1 shows CPU allocation for simulating $H \to ZZ$ (Z decays to all channels) events generated by PYTHIA [10] in $p\bar{p}$ collisions at the center of mass energy, $\sqrt{s} = 13$ TeV using the CMS detector [11] (geometry and the magnetic field map excerpted from CMS software) and Geant4. Assuming that physics (physics processes, pseudo-random number generation and math library) accounts for about 40% of the total CPU time for a typical HEP detector simulation, the maximum performance gain would be less than 2.5 even if the rest of GeantV components were fully vectorized.

It is very challenging to vectorize physics processes and models as they undergo many conditional branches and heavily depend upon data. Consequently, vectorization of physics models requires not only proper algorithm decompositions but also data reorganization in order to maximize instruction throughput and to minimize memory access. To build coherent strategies to make efficient use of parallel architectures for specific physics tasks, it is also critical to understand computing performance of each component quantitatively.



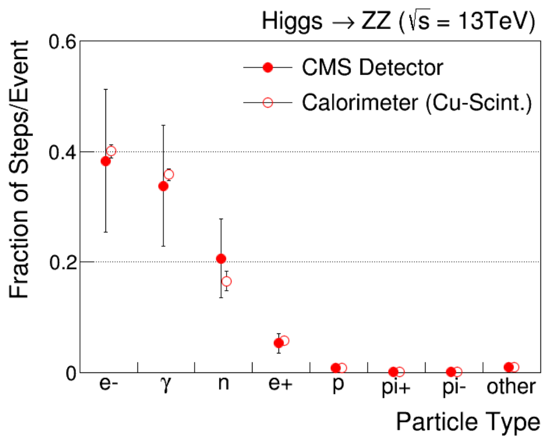


Figure 1. A standalone CMS detector simulation with $H \rightarrow ZZ$ events from $p\bar{p}$ collisions at LHC simulated with Geant4: (left) CPU allocation for linked objects, (right) the fraction of the average number of steps per event by the particle type.

One of approaches of GeantV physics implementation is to develop a vectorized physics module by refactoring existing Geant4 physics codes and leveraging vectors and threads, adopting the same technique used in VecGeom. In Geant4, EM and hadron physics are two major pillars of the description of particle interactions in the passage through detectors. Although the general process of how to track EM particles and hadrons is similar, details of the implementation are significantly different as they undergo different nature of interactions. Since most of the secondary particles produced by primary particle interactions with matter are electrons or photons in typical HEP events and their contribution to the fraction of the average number of steps per event is significant, as shown in the right plot in Figure 1, a set of physics models simulating EM interactions is a good candidate to be vectorized for GeantV. Furthermore, EM physics is a self-contained $e^{\pm}, \gamma$ cascade processes which facilitates re-grouping similar tasks and accumulating a large basket of the same particle type, which may be run on coprocessors efficiently. An example set of EM physics processes and models for high energy electrons and photons is summarized in Table 1.

Table 1. A list of electromagnetic physics processes and models of electron and photon.

Primary	Process	Model	Secondaries	Survivor
γ	Compton Scattering	Klein-Nishina	e^-	γ
	Pair-Production	Bethe-Heitler	e^-e^+	–
	Photo-Electric Effect	Sauter-Gavrila	e^-	–
e^-	Ionization	Moller-Bhabha	e^-	e^-
	Bremsstrahlung	Seltzer-Berger	γ	e^-
	Multiple Scattering	Goudsmit-Saunderson	–	e^-
e^+	Annihilation	Heitler	$\gamma\gamma$	–

In this section, we briefly describe EM physics models and sampling techniques suitable for SIMD and SIMT architectures.

2.1. EM Physics Models
An essential component of particle interactions with matter is the final state analysis described by a physics model associated with the selected physics process for a given step; the physics process and the distance before interaction are determined by the mean free path analysis beforehand. In most EM physics models, the atomic differential cross section of the underlying physics process plays a central role in updating kinematic states of the primary particle or producing secondary particles if necessary. For example, if the selected physics is the Compton process for a given step, the final state of the scattered photon (angle and energy) is sampled based on the two dimensional probability distribution function (PDF) of the Klein-Nishina differential cross section [12] for the given energy of the incoming photon. Then, a secondary particle, the recoil electron, is produced due to the inelastic scattering of the photon on atom. In Geant4, combined composition and rejection methods [13, 14, 15] are often used to sample variables following PDFs used in EM physics models as inverse functions of their cumulative distributions are not analytically calculable in general. However, composition and rejection methods are not suitable for vectorization due to a loop drawing a random number and testing it until one of the number is selected. Alternative sampling methods which can be effectively vectorized will be considered and some examples are described below.

2.2. Sampling Methods
The alias sampling method [16, 17] is similar to the acceptance-rejection method, but it uses an alias outcome for the rejected case which is thrown away in the traditional acceptance-rejection method. It recasts the original PDF with N equally probable events, each with likelihood $c = 1/N$, but keeps information of the original distribution in the alias table. The alias table consists of the alias and the non-alias probability. Unlike composition and rejection methods, the alias method can be effectively vectorized as each sampling procedure follows the same instructions without a branch or a conditional exit. It is also as accurate as the traditional table look-up method, which is neither vectorizable nor efficient due to its use of a binary search. One drawback of the method is that it may introduce a potential bias in the resulting output distribution if the PDF is significantly non-linear within a bin. Nonetheless, we adopt the alias method for random samplings used in EM physics models whenever appropriate, especially for secondary particle productions.

Another popular Monte Carlo technique that can effectively vectorize sampling processes in conjunction with composition and rejection methods is an iterative shuffling algorithm. It applies the split operation [18] for accepted trials and repeating sampling only for rejected cases until all elements are accepted and packed into the output collection. The method is guaranteed to reproduce the original distribution without any bias. However, there is an irreducible overhead in each shuffling loop for reorganizing data which may out-weight the gains from vectorization. The shuffling method is preferable when the sampling efficiency based on the rejection is poor and there are a large set of vector elements.

The last option is the combination of vector operations and scalar loops, which can be applied to any sampling algorithm. As the performance of this approach intrinsically suffers from the Amdahl's law, it may be only considered when the large portion of sampling procedure can be directly vectorizable while other methods are inefficient for a given algorithm.

3. Implementation
GeantV has been designed from the start to enable the use of multiple modern hardware platforms. It is important for the long-term relevance of the project to preserve the option open to use existing platforms, as well as future hardware or software developments. In this section, we briefly describe the structure of code implementation for vector physics models.

3.1. Architecture Backends
GeantV uses backends, which are software layers between the generic, platform-independent simulation code and the hardware-specific details (e.g. SIMD vector registers or GPU threads) and their software-related constructs like SIMD intrinsics, or CUDA `C++` extensions. The main purpose of the backends is to isolate all the complexity of low-level, high performance details behind simplified abstractions which are then available for use by carefully designed, generic kernels. Examples of currently available backends are the scalar, vector, and cuda backends. The vector backend uses the Vc library [19] or UME::SIMD [20] to promote explicit SIMD vectorization by the client code in the kernels. Alternatively, both the cuda and scalar backends use standard types, since the GPU registers are scalar. Detailed descriptions of a backend can be found elsewhere [21, 22].

3.2. Generic kernels
Kernels are high-performance versions of performance-critical algorithms, developed using generic programming and based on the data structures defined by the backends. Each physics-relevant algorithm is coded into a separate kernel so we speak of algorithms and kernels interchangeably. In order to take full advantage of the performance capabilities of the underlying hardware, some important choices were made:

- Inlined functions are used extensively, to avoid the overhead due to function calls.
- Virtual function calls inside the kernels are avoided and replaced by static polymorphism. Kernels themselves are coded in terms of `C++` templates, with a specific backend type as the template parameter. Platform-specific, high-performance kernels are built at compilation time, based on the generic kernels and the backends, selected by a user request or local hardware configuration.
- Branching of execution flow is strictly minimized. Kernels can only use conditional constructs sparingly, preferentially using constant-expression conditions known at compilation time.

Input data for the algorithms come as kernel arguments, triggering the compile-time instantiation of the binary objects appropriate for the hardware used.

3.3. Data Organization
The basic flow of data in vector physics models is that a group of particles (a basket of tracks received from the GeantV scheduler or high level interfaces of physics processes) with similar properties is processed by the parallel map pattern [18] that applies a function to every element of collected data in parallel or in a vectorized way. Each track contains a set of elements (data members) describing the state of the particle during the course of tracking. Position, momentum or energy of the track are frequently queried and updated throughout the physics process and should be laid out contiguously in memory for efficient vector or parallel operations. Therefore, the organization of track data is one of important considerations to achieve efficient memory accesses for both SIMD and SIMT - SIMT (GPU architect) is distinct from SIMD which requires the multiple data elements for a single instruction to be strictly aligned in a vector register.

In general, the Structure of Arrays (SoA) is more efficient than the Array of Structures (AoS) both for SIMD and SIMT [23]. Currently, interfaces to physics processes specialized only for the vector backend take track data in the SoA format. With SoA tracks, each vector kernel processes a set of vector instructions on coalesced chunks of SoA track elements for the number of iterations equivalent to the number of tracks within the basket divided by the corresponding vector size of SIMD instruction sets (2 for SSE, 4 for AVX, 8 for IMCI(MIC) and AVX512 for the double precision).

Another consideration for *effective* vectorization is related to the table look-up used in the *vectorized* sampling procedure with alias tables. Sampling using the alias technique involves scattered memory accesses to get the final state variable in parallel. For example, sampling outgoing photon energies in the Compton process using the alias table randomly selects target bins with values that usually are not contiguous in memory. For the vector backend, gather operations are used to rearrange queried data (scattered in memory) into a contiguous memory segment so that subsequent instructions can be executed through vector pipelines. The scatter operation is also required to store the vector of results back into the original track data. Since a gather operation itself is an additional sequential operation, it introduces an overhead in the performance of the sampling kernel.

4. Performance

As vectorized EM physics models adopt generic implementations for different architectures, it is critical to understand the computing performance of physics kernels and to validate results. Since physics kernels are designed to be architecture-independent, they can be executed in the exactly same way for different backends, allowing direct comparison of simulation results. To test the implementation of sampling algorithms, we extended the verification to execute the same operations using the original Geant4 library. Simulated quantities such as the final energy and angle of the primary track as well as kinematic distributions of secondary particles have been compared and verified with respect to results obtained by Geant4. Verification of EM physics models developed for parallel computing architectures in the GeantV project is described in detail elsewhere [24, 25].

To have correct and efficient parallel code, performance analysis is an essential part of the development cycle. As the primary measure of the computing performance, we define the relative speedup as the ratio between the time taken by a set of kernels with a specific backend (Scalar, Vector, Cuda and etc) and by the Geant4 code to execute the same task. For the purpose of performance measurements, input particles are generated according to an exponentially falling spectrum within a valid energy range for each model - for this paper, the energy range from 2 MeV to 20 MeV where all tested EM models are valid. Even though the relative speedup is not an absolute measure of the speedup, because the efficiency of sampling varies as a function of the energy, it can be used as a general guideline for performance comparisons, to identify potential problems, and to tune models optimized for a specific architecture.

Figure 2 shows preliminary performance results of the alias method tested on Intel® Xeon Phi 5110P (Knight Corner, KNC): (left) the relative speedup of the scalar backend for the alias method with respect to Geant4 composition and rejection methods as a function of number of tracks, (right) the relative speedup of the vector backend using Vc and the IMCI (MIC) instruction set (8 vector pipelines for double precision). The alias method for all models tested except Klein-Nishina (the Compton scattering) performs better than the composition and rejection method owing to the algorithmic change in sampling shown in the left plot. The alias method for the Compton scattering is example where memory transaction (alias table look-up) is computationally more expensive than arithmetic calculations (the composition and rejection algorithm for the Klein-Nishina distribution). Nevertheless, overall vector gains from SIMD with respect to the scalar code are about 3.3 to 6.5 on Xeon Phi (KNC) depending on the number of tracks and EM models shown in the right plot.

Figure 3 shows additional performance results for the new EM models tested on the latest Intel® Xeon Phi 7120 (Knight Landing, KNL) using the UME::SIMD vector backend and the AVX512 instruction set (left), and on NVidia GPU (Kepler K20M) (right) for simulating interactions and sampling secondary particles using the alias method - the host used for performance evaluation is Intel® Xeon E5-2620 for both KNL and GPU. Performance gains by the UME::SIMD vector backend on KNL is about 2 to 3 for 8-64 tracks - note that performance

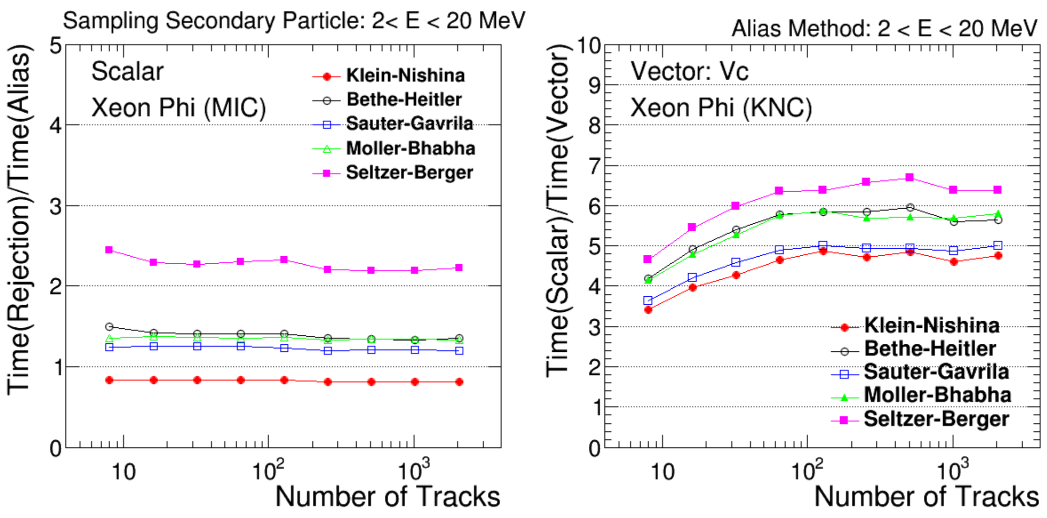


Figure 2. Performance results of simulating particles that undergo EM processes using the alias sampling method tested on Intel® Xeon Phi 5110P (Knight Corner): (left) the ratio between the CPU time taken by the alias method and by the Geant4 composition and rejection method (both scalar codes), (right) the relative speedup of the vector backend using Vc and the IMCI (MIC) instruction set with respect to the scalar.

of UME::SIMD is underestimated as a vectorized pRNG for UME::SIMD is not implemented yet. The plot on the right shows that performance potential on GPU is about 30 even without any optimization for the CUDA backend. However, it requires around 10^4 tracks per process for GPU tasks to be efficient as the number of physical cores of K20M is 2496.

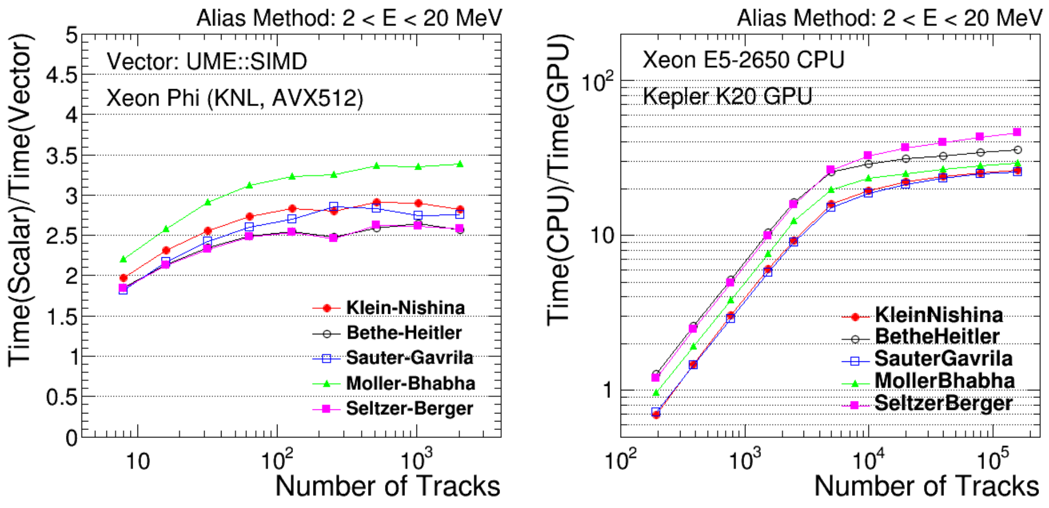


Figure 3. Performance results of simulating particles that undergo EM processes using the alias sampling method: (left) the relative speedup of the vector backend tested on Intel® Xeon Phi 7120 (Knight Landing) using UME::SIMD and the AVX512 instruction set with respect to the scalar code, (right) the ratio between the time taken by the cuda code on NVidia GPU (Tesla K20M) and by the scalar code on Intel Xeon (E5-2620).

5. Conclusion

As the GeantV project assembles the various pieces of the infrastructure: scheduler, geometry, and physics, we demonstrate feasibility of implementing electromagnetic physics processes and models for SIMD/SIMT architectures with common source codes and see tantalizing hints that the goal of significantly improving the performance of physics simulation applications is achievable. For example, as shown in this paper, vectorizing physics code improves simulation speed by a factor 2 to 6 depending on the hardware architecture and vector backends tested on Xeon Phi coprocessors. Further investigation is necessary to verify this result and implement the same type of improvements to other EM physics models. We must see if they can be similarly improved and if these improvements carry through when applied within a full GeantV simulation.

Acknowledgments

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

References

[1] Sverre J 2012 Many-core experience with HEP software at CERN openlab *J. Phys.: Conf. Series* **396** 042043

[2] Allison J *et al* 2006 Geant4 Developments and Applications *IEEE Trans. on Nucl. Sci.* **53** No. 1 270-278

[3] Agostinelli S *et al* 2003 Geant4 - A Simulation Toolkit *Nucl. Instrum. Methods Phys. Res.* A **506** 250-303

[4] Ahn S *et al* 2014 Geant4-MT: bringing multi-threading into Geant4 production *Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2013 (SNA + MC 2013)*, 04213

[5] Apostolakis J *et al* 2014 A concurrent vector-based steering framework for particle transport *J. Phys.: Conf. Series* **523** 012004

[6] http://git.cern.ch/pub/geant

[7] Apostolakis J *et al* 2015 *J. Phys.: Conf. Series* **608** 012023

[8] https://gitlab.cern.ch/VecGeom/VecGeom

[9] Wensel S and Zhang Y 2016 Accelerating Navigation in the VecGeom Geometry Modeller *Computing in High Energy Physics 2016*

[10] Sjöstrand T, Mrenna S and Skands P 2006 PYTHIA 6.4 Physics and Manual *JHEP* **0605**

[11] CMS Collaboration 1994 CMS, the Compact Muon Solenoid: Technical proposal, CERN-LHCC-94-38, CERN-LHCC-P-1

[12] Klein O and Nishina Y Z 1929 *Physik* **52** 853

[13] Butcher J C and Messel H 1960 *Nucl. Phys.* **20** 15

[14] Messel H and Crawford D 1970 Electron-Photon shower distribution, Pergamon Press

[15] Ford R and Nelson W 1985 SLAC-265, UC-32

[16] Walker A J 1977 *ACM Trans. Math. Software. 3*, **3**, 253-256

[17] Brown F J, Martin W R, and Calahan D A 1981 *Trans. Am. Nucl. Soc.* **38** 354-355

[18] McCool M, Robison A, and Reinders J 2012 Structured Parallel Programming, Morgan Kaufmann Publishers

[19] Kretz M and Lindenstruth V 2011 Vc: A C++ library for explicit vectorization *Software: Practice and Experience*. Online at `http://dx.doi.org/10.1002/spe.1149`

[20] UME::SIMD A library for explicit simd vectorization. Online at `https://github.com/edanor/umesimd`

[21] Wenzel S 2014 Towards a high performance geometry library for particle-detector simulation *16th International workshop on Advanced Computing and Analysis Techniques in physics research (ACAT)*

[22] de Fine Licht J 2014 First experience with portable high-performance geometry code on GPU *GPU Computing in High Energy Physics 2014*

[23] Canal P *et al* 2013 High energy electromagnetic particle transportation on the GPU *Computing in High Energy Physics 2014*

[24] Amadio G *et al* 2016 Verification of Electromagnetic Physics Models for Parallel Computing Architectures in the GeantV project *Computing in High Energy Physics 2016*

[25] Amadio G 2016 Electromagnetic Physics Models for Parallel Computing Architectures *J. Phys.: Conf. Series* **762** 012014