

The path toward HEP High Performance Computing

John Apostolakis, René Brun, Federico Carminati¹, Andrei Gheata and Sandro Wenzel

European Organization for Nuclear Research (CERN) - Geneva, Switzerland

E-mail: Federico.Carminati@cern.ch

Abstract. High Energy Physics code has been known for making poor use of high performance computing architectures. Efforts in optimising HEP code on vector and RISC architectures have yield limited results and recent studies have shown that, on modern architectures, it achieves a performance between 10% and 50% of the peak one. Although several successful attempts have been made to port selected codes on GPUs, no major HEP code suite has a "High Performance" implementation. With LHC undergoing a major upgrade and a number of challenging experiments on the drawing board, HEP cannot any longer neglect the less-than-optimal performance of its code and it has to try making the best usage of the hardware. This activity is one of the foci of the SFT group at CERN, which hosts, among others, the Root and Geant4 project. The activity of the experiments is shared and coordinated via a Concurrency Forum, where the experience in optimising HEP code is presented and discussed. Another activity is centred on the development of a high-performance prototype for particle transport. Achieving a good concurrency level on the emerging parallel architectures without a complete redesign of the framework can only be done by parallelizing at event level, or with a much larger effort at track level. Apart the shareable data structures, this typically implies a multiplication factor in terms of memory consumption compared to the single threaded version, together with sub-optimal handling of event processing tails. Besides this, the low level instruction pipelining of modern processors cannot be used efficiently to speedup the program. We have implemented a framework that allows scheduling vectors of particles to an arbitrary number of computing resources in a fine grain parallel approach. The talk will review the current optimisation activities within the SFT group with a particular emphasis on the development perspectives towards a simulation framework able to profit best from the recent technology evolution in computing.

1. Introduction

Efficient exploitation of current and future computing architectures is one of the key elements that will influence the data analysis strategy of the High Energy and Nuclear Physics (HENP) experiments and that will ultimately have an impact on the quality and quantity of science that will be done in the field. Higher performances on current architectures are achieved by exploiting the opportunities for parallel execution offered by the hardware, both at the level of the single instructions, via several SIMD techniques, and at the level of the instruction flow, via multi-threading or multitasking.

¹ To whom any correspondence should be addressed.

In spite of its high parallel content, HENP code has been traditionally slow in exploiting it to obtain higher performances, apart from running independent event streams on different cores. The reasons for this have to do both with the nature of the workload and of the problem at hand. The workload is essentially driven by throughput (events per unit time) rather than by time-to-solution (how fast a single job completes), and therefore the gains from multi-threading are more in the realm of reducing memory footprint and improving multi-core utilisation. However these come at the price of a potential complication of the I/O subsystem that has to be carefully designed to avoid losing all gains due to locking, particularly for the very data intensive tasks that characterise our workload.

The actual algorithms used to treat the data are diverse, from simulation to calibration, reconstruction and analysis, however they all share some common features as far as the code and instructions are concerned. A high degree of parallelism is present also here: different detectors are largely independent and different particles as well. However we also see a very high degree of “non uniformity”. Different particles have to be treated differently and even for one particle kind the treatment varies according to its kinematics and position. We find lots of possibilities for MIMD processing but woefully few SIMD opportunities. The code is characterised by a large number of short loops and conditional clauses, and by the navigation in large data structures using pointers and indexes. Performance scales closer to SpecInt (the integer benchmark suite of the Standard Performance Evaluation Corporation [1]) than to SpecFP (the floating-point benchmark suite), in spite of the fact that the floating-point content is far from negligible.

If C++ has served HENP well in the past two decades, it has also introduced some hurdles of its own. Templates and inheritance are elegant and effective ways to deal with complexity and diversity of algorithms. However C++ containers do not guarantee data contiguity or efficiency and heavy usage of virtual interfaces introduces penalties and makes compiler optimisation very hard. Specialised languages such as OpenCL [2] or CUDA [3] are much closer to C than to C++, and this for a good reason. The very large amount of (successful) C++ code written by HEP might quickly become an embarrassing legacy in moving to the brand new world of High Performance Computing (HPC). The assessment of the current situation can be found in a nice paper published last year by the openlab project [4]. This paper shows that there is, at least in principle, some room for the improvement of our code.

Unfortunately the problem cannot be dodged, as every performance improvement will have a profound impact on the future of the field. New or upgraded experiments will see an increase in the amount of data collected of a factor between 10 and 1000, and therefore an at least corresponding increase of the computational resources, against a background of dwindling funds, where maintaining a constant budget will be already a substantial success. It has been said times before, but it now really seems the case: HENP really risks to compromise physics because of lack of computing resources. If a mere 10 % improvement in code performance might not seem worth any effort, it is enough to translate this in terms of money on a WLCG grid that counts more than 300,000 cores to see what could be the potential impact.

One more thing to be said about the difficulty of optimising HENP code is that the evolution rate of the code is very high, and non-professional programmers write a large part of it. Even after an algorithm has been fine-tuned for maximal performance, it is bound to be sooner or later modified by a physicist who may be an expert or not. But even if the person changing it is an expert, it is doubtful that the fine-tuning procedure can be repeated at every change during the frantic activity preceding a conference or the preparation of a large production campaign.

Last but not least, portability to different architectures will remain a must, as different centres on the Grid or the Cloud will acquire their own high-end equipment and experiments survive several generations of new computers. The result of the optimisation work should be a code that is performing well on the present machines and that is a good base to transition to the next generation. It should not

be a code that obtain near-optimal performance on the current hardware at the price of being too hardware specific and require a complete rewrite to be ported. Here standards would help, however they are slow in coming due to the high dynamicity of the high-end hardware and software vendors, which in itself is one of our best allies.

2. Current research directions

It is clear that intense R&D is necessary before embarking into any large development project aimed at refactoring and rewriting existing code. The amount of combinations to be tested is daunting. Different compilers, languages, hardware platforms and programming models can to a large extent be used in a number of combinations. Our community has, in principle, the ability to effectively tackle this large amount of work, thanks to its mix of expertise and its knowledge of the application domain, however to be effective, appropriate communication channels have to be established to share knowledge and build consensus on common solutions.

To address this issue, in 2011 the CERN SFT group created the Forum on Concurrent Programming Models and Frameworks. The stated objective of this forum is to “share knowledge among interested parties that should work together to develop 'demonstrators' and agree minimally on technology so that they can share code and compare results.” [5]. The forum is meeting bi-weekly and it has spawned the development of sixteen demonstrators and two major projects, the Concurrent Framework Project (GaudiHive) (see for instance [6] at this conference) and the Geant Vector Prototype project (Geant-V) described by this paper (see also [7] [8] at this conference and [9]). The Concurrency Forum has an instrumental role in bringing people together and in fostering dialogue on the subject of HPC in our field.

Another initiative that is important to mention is the realisation of a multi-threaded version of the Geant4 Simulation Toolkit, the de-facto standard in HEP for detector simulation (see [10] at this conference and [11]). This work started in 2008 and led to the production of an early production prototype [12] that has been now merged with the main development version of Geant4 in version 10. Parallelisation has been achieved at the event level, while relevant information is shared between the different threads. The resulting code shows excellent scalability with the number of cores, both on Intel servers and on Intel Xeon Phi devices. Memory footprint is substantially reduced with respect to the sequential version. This is a very encouraging development, as it is the first large scale common physics code enabled for parallelism to enter production. Particular attention has been devoted to maintain backward compatibility, so that only minimal changes are needed to the user code. There are now plans to extend the parallelism at the track level, even if preliminary studies show that this would imply larger modifications and a reduced backward compatibility. If this development is remarkable under several points of view, still it is not intended to address the question of the optimisation at the micro-architectural level that alone can provide a net increase in the number of events produced for unit time.

Multithreading conversions of HEP code are important, however they need to be complemented with extensive investigations on the micro-parallelisation of the code, to increase actual throughput. Several initiatives have been presented to the Concurrency Forum, reporting good results on specific applications. It is here important also to mention the seminal work done by openlab ([13] and [14] at this conference) in training on optimisation of the code at several levels and consultancy.

Looking at this end of the problem, a group at FNAL is exploring the micro-optimisation of the Geant4 code [15] [16]. This group has developed a prototype of a Geant4-based massively parallel high-energy electromagnetic particle transportation in the Graphic Processor Unit (GPU). The component of the transportation prototype consists of a transportation process under a non-uniform magnetic field, a geometry navigation with a set of solid shapes and materials, electromagnetic physics processes for electrons and photons, and an interface to a framework that dispatches bundles of tracks in a highly vectorised manner optimizing for spatial locality and throughput. The code is

C/C++ and designed to be compatible with CUDA and OpenCL and generic enough for future variations of programming models and hardware architectures. This project and the Geant-V one, which will be described in detail in the next sections, are now building a close collaboration converging to a common code.

3. The Geant-V project

When facing the micro-parallelisation of a larger framework code, one is immediately confronted with the need of reworking the algorithms but above all reorganise the data so that these can be fed to the processing units contiguously and with as little dependence and logical branches as possible. This immediately entails deep changes in the code, which make the development process particularly laborious for large code systems.

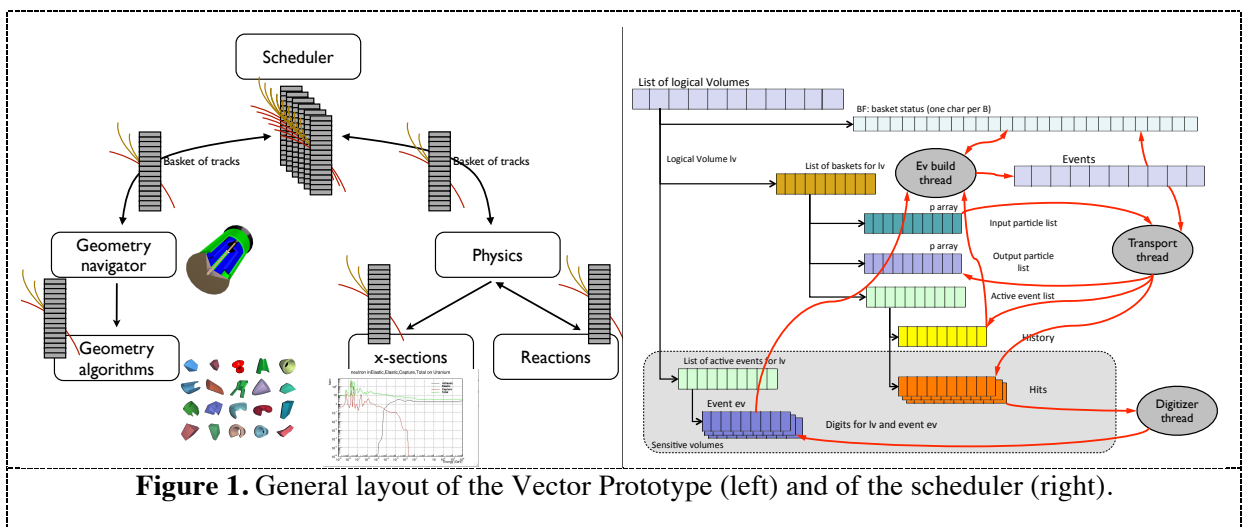


Figure 1. General layout of the Vector Prototype (left) and of the scheduler (right).

To reduce the difficulty of this work and produce meaningful results that could guide the development, we decided to launch an initiative to develop *ab initio* a prototype for particle transport that could take maximum advantage from both the multithreading and the micro-parallelisation aspect. The choice fell quite naturally on the simulation programme for several reasons.

HEP long experience in developing simulation programmes ([17] [18] [19] [20] [21]) and the proximity of a substantial Geant4 group was considered a big advantage. The simulation code is comparatively experiment-independent if compared to reconstruction or analysis, and it is less data-intensive than the latter. Moreover, and not less importantly, simulation is one of the main consumers of CPU time and this will probably become ever more so in the future. The Geant-V started at the beginning of 2012 based on a set of preliminary studies performed in 2011.

The objective of the Geant-V project is to explore different design options to maximise the efficiency of the transport code. For this parallelism at all levels has to be exposed and exploited.

A particle transport problem has a large intrinsic parallelism: all events and all tracks within events are independent. Both this coarse parallelism and finer grain parallelism have to be exposed and exploited, in order to maximise microarchitectural efficiency within the CPUs and GPUs.

Present particle transport is scalar and sequential. A particle is followed from start to end across all materials and volumes it traverses. This means that different geometry paths (including volume parameters and geometry transformations) and materials are loaded and then unloaded for each boundary crossing of each particle history, with a considerable rapid exchange of the data in memory

implying long latencies for data fetching². Particle transport is mostly local, in the sense that the majority of the “transport steps” happen in a small portion of the volumes. In a typical test we made with the geometry of the ATLAS detector, we found that 0.7 % of the volume types contain 50 % of the total steps. From this result, presented and commented at CHEP 2012 [22] and ACAT 2013 [9] we set about to design a code that could exploit locality by transporting vectors of particles (baskets) that contained in the same logical volume and transported in the same material. In the development of our prototype, we are concentrating our efforts in two dimensions:

1. Coarse grain parallelism. This exploits the “trivial parallelism” intrinsic in MonteCarlo simulations and on shared memory, multi-core machines leads to reduction of memory footprint. On system with attached processors (GPUs or Xeon Phis) this allows to have several of them working in parallel, trading the minimum amount of data between them and the main processor.
2. In-core micro-parallelisation and in particular vectorisation. This allows increasing throughput and ultimately is the source of the improved performance. Our studies show that part of the gain comes from the actual exploitation of the SIMD capabilities of the processors, but a large gain also comes from the improved locality deriving from the processing of several particles together.

In the following we will review the progress in the four main areas in which we have factorised our problem: the scheduler, the geometry navigator, the basic geometry routines and the physics. The relation between these elements in the transport process is shown in Figure 1. The scheduler has the role to manage the queues of baskets. The input baskets are given to the geometry navigator and to the physics subsystem for handling. The geometry subsystem passes them to the geometry algorithms to determine various quantities such as the distance to the next boundary along the line of flight and the closest distance to any boundary in all directions (safety). The navigator hands back to the scheduler an output basket of particles in various logical volumes, which need to be re-dispatched to the appropriate logical volume baskets for further processing. The physics subsystem determines the distance to the next interaction for the particles of a basket and, when the interaction happens, it generates the secondary particles that are fed back to baskets to the scheduler for further re-dispatching. Each basket is handled by a separate task, while one more task handles the scheduler.

3.1. The scheduler

The scheduler handles the particle baskets and moves them efficiently across the geometry and physics subsystems. Its main roles are to keep track of the basket workload and to maximize their track content while maintaining a good concurrency level for the transport tasks. The scheduler general layout is shown in Figure 1. Each logical volume has a list of “baskets” holding tracks being currently transported in one of the physical instances of that volume. There is one basket manager per logical volume, dealing with track “basketizing” and pushing “*ready for transport*” baskets to the common work queue. The transportability criteria depend on the track flow in and out a given volume and on the vectorisation performance. Each basket has one input and one output buffer of particles, and an active event list. For each event in this list there is one or more buffer of hits, if this is a sensitive detector, and a history buffer. The scheduler performs the following actions:

- 1) A basket with the status flag “ready for transport” is added to the common concurrent work queue feeding a number of transport tasks (run by separate threads). The transport task that takes the basket transports all particles in the input buffer and, as they leave the volume, interact or stop, puts them in the output buffer for further processing. At the end of this “local” transport procedure

² In common computing parlance this is called memory or cache “thrashing”.

the input buffer is empty³. Any daughter produced during the transport is also put in the output buffer. The task then sends back to the scheduler the transported basket via a result queue.

- 2) A dispatching task (again associated with a thread) takes all the baskets available in the result queue which are now “ready for dispatching”, and moves their particles to the appropriate volume basket managers and their “ready for filling” baskets. The loop is closed when these baskets become “ready for transport” and are sent to the work queue. This operation implies a real copy of particles in order to maintain contiguity of data in the baskets.
- 3) The dispatched basket are marked as “ready for filling” and recycled to the basket manager owner.

A detailed analysis of the program flow has convinced us that locks are actually minimised in this approach. We are now carefully designing the handling of the special cases of the beginning and the end of the cycle and of the “flushing” of events to avoid memory inflation. This is done associating “policies” for the different logical volumes. Here are some of the policies that we currently apply:

- Mixing tracks from several events to maximize the basket content per logical volume. The scheduler can invoke the event generator to boost the vector content of baskets when approaching the end of cycle and only few tracks are left in the detector.
- Prioritizing baskets with tracks from “old” events in order to “flush them” and make room for new ones. This is implemented using a double-ended work queue and allows keeping the memory under control.
- Dynamically adjusting the “ready to transport” threshold and the number of baskets per volume. A volume that is rarely traversed by tracks will have a lower threshold than one that is a major contributor to transport.
- Scheduling policies applied within the transport tasks. In order to allow vectorisation, the track buffer container is implemented as a structure of arrays (SOA) allocated in contiguous aligned memory. During the different transport stages the baskets may lose content (like interacting particles or tracks crossing boundaries) and become inefficient in a vector approach. The scheduler may decide to postpone these baskets and rather “re-basketize” to more efficient ones. Also, both physics and geometry methods often require selecting a given category of particles (like “electrons only” or “far from boundary” ones) to actually allow for vectorisation. For this purpose the track vector container allows for dynamic reshuffling of its content and removal of “holes” produced by particle state changes.

The transport of several events in parallel requires modifying the data structures and reviewing the scoring strategy, both on framework and user sides. The user scoring and digitization are very resource demanding and should not spoil the overall performance, both in terms of concurrency and memory management. This implies extending the scheduler to such tasks and in addition providing an alternative for the memory management for user data (such as factories for hits and digits).

When an event is finished, i.e. all its particles have been transported, a separated task will be scheduled to digitize the event by going through all the baskets of the sensitive volumes and collecting the hits for this event, transforming them into digits. Note that this task should be highly suited to a SIMD optimisation because all hits of a detector have, by definition, a similar structure.

³ Or the number of particles in it has fallen under the “transport threshold” and the particles still to be transported are copied “as such” to the output buffer.

3.2. Physics and geometry optimisation

For the purpose of verifying our design, we need both a realistic physics shower development and a realistic geometry package. To be able to simulate a realistic shower development, we decided to extract from Geant4 both the tabulated cross sections and a sampling of the final states and use them in the prototype. For continuous processes we follow the same direction. We have tabulated the specific energy loss, the average angle of multiple scattering and the average dispersion for multiple-scattering angle. All this information is to be used at transport time to generate continuous (along the step) processes and discrete processes. A SIMD implementation of the methods to extract cross sections and sample final states will come next.

As far as geometry is concerned, for this first version of the prototype we decided to start from the Root geometry navigator [23]. To use it in a multi-threaded context, we separated the stateful description of the position and direction of a particle flying through the detector from a stateless, read-only description of the geometry in memory. This allowed us to transport several particles in parallel holding their status in thread-local data, while all threads were using a shared geometry description. We have currently SIMD-optimized the core algorithms of several shapes and developed a reduced version of the navigator able to accept vectors and pass them down to the geometry algorithms. An extensive description of this work has been presented at this conference [7] and we refer the reader to it.

4. Conclusions

Work toward the realisation of a high performance particle transport prototype optimised for SIMD machines (the Geant Vector Prototype) is progressing steadily. The major components have been identified and architected and their implementation is in advanced state. We believe we will be able to demonstrate a full-scale prototype before the end of the year.

5. Acknowledgement

We would like to thank Pere Mato of the SFT group for useful discussions. We would also like to thank Marilena Bandieramonte from Catania University, Raman Shegal from the Bhabha Atomic Research Centre, Mumbai, India and Laurent Durhem of Intel Corporation for their collaboration. We also gratefully acknowledge the help of Andrzej Nowak from CERN openlab.

6. Bibliography

- [1] Standard Performance Evaluation Corporation. (2013, November) SPEC - Standard Performance Evaluation Corporation. [Online]. <http://www.spec.org>
- [2] The Khronos Group. (2013, October) OpenCL - The open standard for parallel programming of heterogeneous systems. [Online]. <http://www.khronos.org/opencl/>
- [3] NVIDIA Corporation. (2013, October) NVIDIA Developer Zone. [Online]. <https://developer.nvidia.com/category/zone/cuda-zone>
- [4] Jarp Sverre, Alfio Lazzaro, and Andrzej Nowak, "The future of commodity computing and many-core versus the interests of HEP software," in *International Conference on Computing in High Energy and Nuclear Physics*, vol. 396, 2012, pp. 52-58, doi:10.1088/1742-6596/396/5/052058.
- [5] CERN-SFT Group. (2013, October) Forum on Concurrent Programming Models and Frameworks Main menu. [Online]. <http://concurrency.web.cern.ch>
- [6] B Hegner, P Mato Vila, and D Piparo, "Introducing Concurrency in the Gaudi Data Processing Framework," in *20th International Conference on Computing in High Energy and Nuclear Physics (CHEP)*, 2013.
- [7] J Apostolakis, R Brun, F Carminati, A Gheata, and S Wenzel, "Vectorizing the detector geometry

- to optimize particle transport," in *20th International Conference on Computing in High Energy and Nuclear Physics (CHEP)*.
- [8] J Apostolakis et al., "Parallelization of particle transport with INTEL TBB," in *20th International Conference on Computing in High Energy and Nuclear Physics (CHEP)*, 2013.
- [9] J Apostolakis, R Brun, F Carminati, A Gheata, and S Wenzel, "A concurrent vector-based steering framework for particle transport," in *15th International Workshop on Advanced Computing and Analysis Techniques in Physics (ACAT)*, 2013, In course of publication.
- [10] G Cosmo, "Geant4 - Towards major release 10," in *20th International Conference on Computing in High Energy and Nuclear Physics (CHEP)*, 2013.
- [11] S Ahn and et al., "Geant4-MT: bringing multi-threaded Geant4 into production," in *Joint International Conference on Supercomputing in Nuclear Applications and Monte Carlo 2013 (SNA + MC 2013)*, Paris, 2013, In course of publication.
- [12] X Dong , G Cooperman, and J Apostolakis, "Multithreaded Geant4: Semi-Automatic Transformation into Scalable Thread- Parallel Software," in *Euro-Par2010 - Parallel Processing*, vol. 6272, 2010, pp. 287-300.
- [13] CERN openlab. (2013, Octobre) CERN openlab. [Online]. <http://openlab.cern.ch>
- [14] A Nowak, "Is the Intel Xeon Phi processor fit for HEP workloads?," in *20th International Conference on Computing in High Energy and Nuclear Physics (CHEP)*, 2013.
- [15] P Canal, D Elvira, R Hatcher, S Y Jun, and S Mrenna, A Vision on the Status and Evolution of HEP Physics Software Tools, 2013, arXiv:1307.7452v1.
- [16] P Canal et al., "High Energy Electromagnetic Particle Transportation on the GPU," in *20th International Conference on Computing in High Energy and Nuclear Physics (CHEP)*, 2013.
- [17] G Battistoni et al., "The FLUKA code: Description and benchmarking," in *Hadronic Shower Simulation Workshop 2006, Fermilab 6-8 September 2006*, vol. 896, 2007, pp. 31-49.
- [18] R Brun, "GEANT Detector Description and Simulation Tool (Version 3.21)," Application Software Group CN Div, CERN, Geneve, CERN Program Library W5013, 1993.
- [19] J Allison and et al., "Geant4 developments and applications," *Nuclear Science, IEEE Transactions*, vol. 53, no. 1, pp. 270-278, Feb 2006.
- [20] R R Johnston, "A General Monte Carlo Neutronics Code," Los Alamos Scientific Laboratory , Los Alamos, LAMS 2856 , 1963.
- [21] R L Ford and W R Nelson, "The EGS code system – Version 3," Stanford Linear Accelerator Center, Stanford, SLAC 210, 1978.
- [22] John Apostolakis, Rene Brun, Federico Carminati, and Andrei Gheata, "Rethinking particle transport in the many-core era towards GEANT 5," in *Computing in High Energy Physics 2012 (CHEP 2012)*, vol. 396-2, 2012.
- [23] A Gheata. (2013, May) ROOT | A Data Analysis Framework. [Online]. <ftp://root.cern.ch/root/doc/18Geometry.pdf>
- [24] J Apostolakis, R Brun, F Carminati, and A Gheata, "Rethinking particle transport in the many-core era towards GEANT 5," in *Computing in High Energy Physics 2012 (CHEP 2012)*, vol. 396-2, 2012, doi:10.1088/1742-6596/396/2/022014.