A concurrent vector-based steering framework for particle transport

John Apostolakis, René Brun, Federico Carminati, Andrei Gheata and Sandro Wenzel
European Organization for Nuclear Research (CERN) - Geneva, Switzerland

E-mail: Federico.Carminati@cern.ch

Abstract. High Energy Physics has traditionally been a technology-limited science that has pushed the boundaries of both the detectors collecting the information about the particles and the computing infrastructure processing this information. However, since a few years the increase in computing power comes in the form of increased parallelism at all levels, and High Energy Physics has now to optimise its code to take advantage of the new architectures, including GPUs and hybrid systems. One of the primary targets for optimisation is the particle transport code used to simulate the detector response, as it is largely experiment independent and one of the most demanding applications in terms of CPU resources. The Geant Vector Prototype project aims to explore innovative designs in particle transport aimed at obtaining maximal performance on the new architectures. This paper describes the current status of the project and its future perspectives. In particular we describe how the present design tries to expose the parallelism of the problem at all possible levels, in a design that is aimed at minimising contentions and maximising concurrency, both at the coarse granularity level (threads) and at the micro granularity one (vectorisation, instruction pipelining, multiple instructions per cycle). The future plans and perspectives will also be mentioned.

1. Introduction

High Energy Physics always depended upon the availability of very large computational resources to extract the physics from the collected data and it has been well served by the steady increase in computing power during the last 30 years synthetized by the so-called Moore's law. From the 80's until 2004, this increase came mainly in the form of faster cycle computers, doubling the software performance every 18 months with no need for code refactoring. High Energy Physics made full use of this bonanza and attempts to optimise the offline code to take into account the intrinsic parallelism of the problem being dealt with (independent tracks, events, detectors and so on) and the new architectures appearing on the market (RISC, multicore machines, SIMD capable CPUs, GPU co-processors) were few and of marginal impact, at least in the offline domain.

The situation is now changing due to two concomitant factors. On one side, since the CPU clock frequency reached the 3GHz mark around 2005, fundamental physical limits have blocked its further progress. Computer companies still claim they offer an 18-month doubling of the computing capacity, but this now comes in the form of enhanced parallelism at various levels, which must be skilfully exploited by the application software. In other words, while until 2005 the increase in performance was provided in the form of raw computational speed, now it is offered in terms of "opportunities"

that the user has to seize. It goes without saying that this does not come for free, above all because some applications are more prone to be optimised than others.

The second important factor is that, in the next few years, the new or upgraded experiments will see an increase in the amount of data collected of a factor between 10 and 1000, and therefore require at least a corresponding increase of the computational resources, against a background of dwindling funds, where maintaining a constant budget will be already a substantial success.

These factors combined suggest that it is urgent to explore what the opportunities are to improve the performance of our code. The assessment of the current situation can be found in an interesting paper published last year by the OpenLab project [1]. This paper shows that there is, at least in principle, plenty of scope for the improvement of our code.

Several initiatives have been started to improve and optimise existing code (see for instance [2,3,4] and similar publications at CHEP 2012), however for the moment their only coordination is via the CERN Concurrency Forum [5]. The advantage of starting from existing code is that the improvements can be of immediate use to the community. However such initiatives are also, almost by definition, constrained by the features of the existing code: it is the flip side of their short time usefulness.

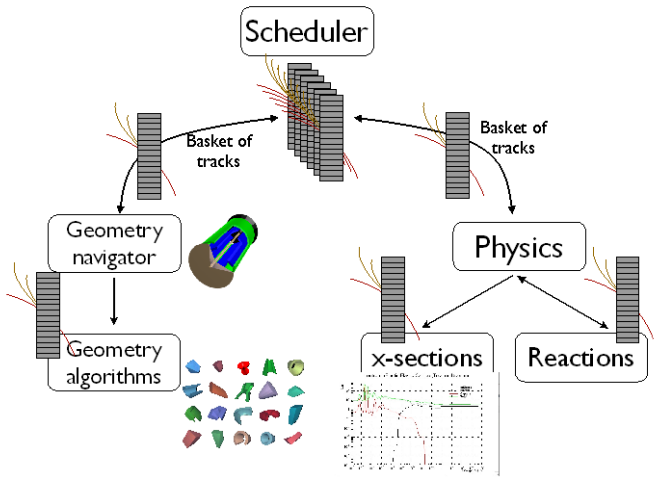

Figure 1. General layout of the Vector Prototype.

We consider it necessary to complement these initiatives with a wide-ranging exploration unhindered by any backward compatibility consideration. For this we have chosen the particle transport application used in detector simulation. The Geant Vector Prototype project (in short Geant-V) started based on a set of preliminary studies performed in 2012. High Energy and Nuclear Physics has a long established tradition of developing particle transport codes to simulate particle detectors both in the design phase and to evaluate the efficiency of particle detection in the data analysis phase ([6,7,8,9,10,11]), and there is substantial expertise in the community on this kind of problems. Monte Carlo simulation is one of the most demanding computing task, due to the slow decrease of the statistical fluctuations around the estimated mean, which is proportional to the inverse of the square root of the number of events simulated. Moreover particle transport simulation is one of the most experiment-independent applications in High Energy Physics. Unlike reconstruction and analysis, where most of the code is specific to a given experimental apparatus, most of the simulation code is generic, and it is expected that any improvement reached in this code could profit all existing and future experiments.

2. The Geant-V Project

The objective of the Geant-V project is to explore different design options to maximise the efficiency of the transport code on modern CPUs and GPUs, in order to obtain the best possible performance in terms of events produced per unit time. For this, parallelism at all levels has to be exposed and exploited.

A particle transport problem has a large intrinsic parallelism, insofar all events are independent and different tracks within the same events are also independent. Exploitation of this parallelism is necessary, but alone not sufficient, as it may improve memory footprint but not increase the number of events per second actually simulated. There is also significant potential for fine grain parallelism in treating low-level operations including geometry and physics. This too must be exposed and exploited, increasing the number of instruction per cycle and exploiting instruction pipelining and vector processing within the CPUs and GPUs.

Present particle transport is essentially scalar and sequential. A particle is followed from its birth to its end, across whatever portion of the detector it traverses. This means that different geometry paths and materials are loaded and then unloaded for each particle history, with a considerable danger of exceeding the capacity of the faster levels of the cache ("trashing" of the cache memory). Particle transport is mostly local, in the sense that the majority of the "transport steps" happen in a small portion of the volumes, typically 50% of the steps in less than 1% of the volumes. This is the result of our preliminary studies that we have presented at CHEP 2012 [12]. We started from this consideration to design a particle transport programme that could exploit this locality by transporting vectors of particles (baskets) that are contained in the same logical volume and are transported in the same material.

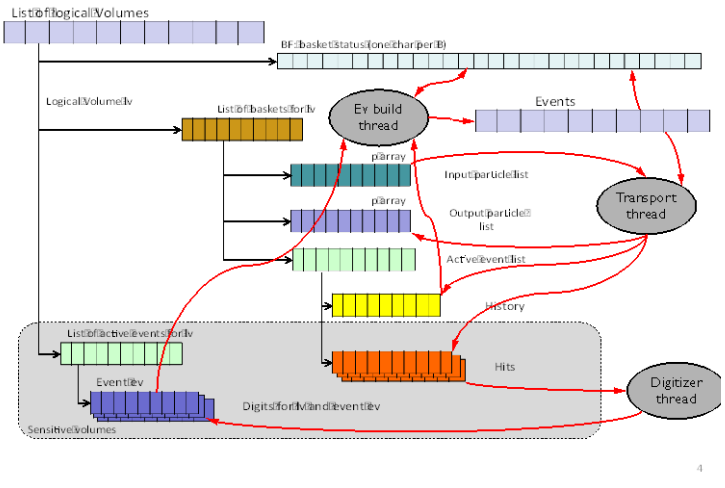

Figure 2. Layout of the scheduler.

In exploring the possibilities to recast particle simulation so that it takes advantage from all performance dimensions and technologies we are concentrating our efforts in two dimensions:

1. Coarse grain parallelism. This exploits the "trivial parallelism" intrinsic in Monte Carlo simulations and leads to reduction of memory footprint on shared memory, multi-core machines. On systems 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 or vectorisation. This allows increasing throughput and ultimately is the source of the improved performance. It is expected that the gain in performance derive not only from the parallelism at the instruction level (vectorisation, instruction pipelining and multiple instructions per cycle), but also from a better use of both data and instruction caches due to the enhanced "locality" of the code. This requires a redesign of the data structures and processing strategy that usually also increases locality thereby reducing the cache misses.

In the following we will give a short report of the current situation 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 orchestrate 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 the distance to the boundaries and to the volumes contained into the logical volume. When on a boundary, the navigator determines the next volume that is to be entered by the particles in the global geometry. The navigator gives back to the scheduler a new basket of particles in various logical volumes that need to be re-dispatched to the 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 is supposed to happen, it generates the secondary particles and again feeds back particles in baskets to the scheduler for further re-dispatching. Each basket is handled by a separate task, while one more task handles the scheduler.

2.1. The scheduler

It is possible to obtain the desired performance improvements only if we manage to transport sets of particles "together" to maximise the locality of the operations. This calls for the efficient handling of lists of particles to be fed both to the geometry and to the physics methods, which should be in turn properly optimised to treat them. A scheduler handles these lists of tracks during transport. The general layout of the scheduler can be seen in **Figure 2**. The general schema of the scheduler is the following. Each logical volume has a list of baskets to which corresponds a list of status flags. Each basket has one input and one output buffers of particles, and a list of active events. 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 which has the status flag "ready for transport" is associated with a transport task (run by a thread) which 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. The basket is marked "ready for dispatching".

2) A dispatching task (again associated with a thread) looks for baskets "ready for dispatching" and puts their particles in buffers that are "ready for filling". When a basket is full, it is marked "ready for transport".

3) When the basket "ready for transport" is dispatched, it is marked as "ready for filling".

A detailed analysis of the program flow leads us to believe that locks are minimised in this process flow. 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.

As we have seen (see [12]), the difference in population of the different logical volumes can be very relevant. During a single event some volumes may never reach a population of particles convenient to handle via the basketised transport. To make sure that this does not happen, our design allows the transport of several events in parallel. This is an important modification of the transport schema now in operation in various Monte Carlo programmes, and it will call for an attentive revision of the scoring strategy of the user code.

When an event is finished, i.e. all its particles have been transported, a separated task will be used to digitise the event by going through all the baskets of the sensitive volumes and collecting the hits for this event, transforming them into digits. We are investigating SIMD optimisations for this task because all hits of a detector have, by definition, a similar structure.

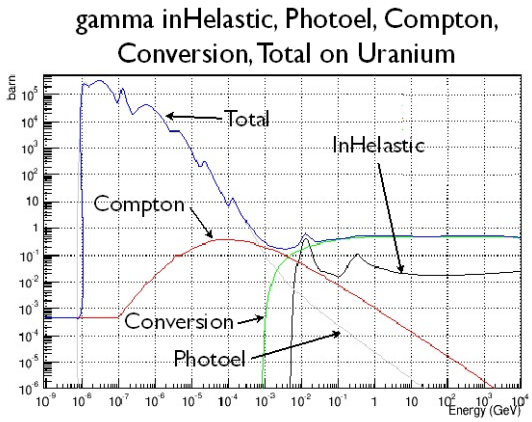

Figure 3. Example of cross section tabulation from Geant4.

2.2. Physics

The plan for the prototype includes optimising physics algorithms, including cross section tables, sampling and generation of secondaries, passing them lists of particles and optimising their SIMD performance. This however has to be regarded as a long-term goal, as the development of a SIMD optimised physics package may take a substantial amount of time. Nevertheless, for the purpose of verifying our design, we needed both a realistic physics shower development and a realistic geometry package. Our problem has very little chance to be linear, so simple scaling from "toy models" is most probably meaningless. We will explain in the following sub-section how we are dealing with the geometry. To be able to simulate a realistic shower development without interfacing with large packages, such as Geant4, we decided to extract from Geant4 both the tabulated cross sections and a sampling of the final states and use them in the prototype. To do so, we have selected the major mechanisms, namely bremsstrahlung, e+ annihilation, Compton scattering, discrete ionisation, elastic scattering, inelastic scattering, pair production, photoelectric effect and capture, and for each particle we have tabulated all Geant4 cross sections for all materials from Hydrogen to Uranium in the range 10 keV to 1 TeV. Using 100 logarithmically spaced bins produces an ~90 MB file. For each energy bin, each reaction, each particle and each element we generate a number of final states that we store in yet another file. When sampling 20 final states per bin, with 100 bins as before, we have a file of ~500 MB (including multiple scattering and energy loss, see later). This means that for a typical detector, which is composed of less than 30 elements, we can store the whole physics in ~300 MB in memory (taking into account Root compression), which is still a reasonable size for an in-memory database.

For continuous processes we follow the same approach. We have tabulated the specific energy loss, the average angle of multiple scattering and the average dispersion for the multiple-scattering angle. All this information is to be used at transport time to generate continuous (i.e. along the step) processes and discrete processes. When a physics process is selected, the set of final states

corresponding to the energy closest to the one of the incoming particle is determined, and one of these final states is randomly selected and rotated along the direction of the incoming particle, with a random azimuthal angle. In future we will also rescale the energy of the final state to preserve energy balance. We expect that this give us realistic shower development even if the quality of the physics will be probably very diverse depending on the problem and the range of energy treated. It could however be a nice basis for a possible fast simulation development. In **Figure 3** we show an example of cross section tabulation with all the reactions of a photon on Uranium.

For the moment we consider only natural mixtures of isotopes, but a future development will extend this method to the single isotopes. The cross sections are held in C-like data structures wrapped in lightweight Root classes with minimal use of virtuality, to make it easier to port the code on different co-processors. The extracted cross sections and the final states are held in two Root files. For the moment the methods used to extract cross sections and sample final states have only scalar signatures; the implementation of vector signatures and the SIMD optimization will come next.

2.3. The geometry navigator

In order to have a first version of the prototype capable of tackling complex geometries, we needed to start from a mature navigator and we chose the Root one. The first obstacle we encountered was that the Root navigator was not designed for a multi-threaded context, and the geometry description was carrying stateful navigation information to optimise the search algorithms. The first job was to separate the state describing the current position and direction of a particle flying through the detector from a stateless, read-only description of the geometry in memory. Once this was achieved, we could transport several particles in parallel holding their status in thread-local data, while all threads were using a shared geometry description.

This enabled us to do multi-threaded particle transport with the Root geometry TGeo. It is worth to note that this new version of TGeo is now used in production, as it is in effect a more rationalised and efficient code even for sequential transport and geometry navigation. This code is also a step in the direction of the attached processors like GPUs, insofar the geometry description (and physics tables) can be loaded once and for all in the coprocessor and only the stateful particle data have to be traded.

The next step will be to be able to transport the baskets of particles. For this a “vectorised” version of the navigator will have to be developed. We have started the design phase of this new navigator.

2.4. The geometry algorithms

We have used the basic geometry routines as a playground with which to experiment with in-core micro-parallelism and vectorisation (kernel optimisation). The procedure here is essentially to recast the algorithms so that they would handle not one but several particles at the time. This is achieved via the introduction of a new “vector” signature, and it corresponds to the idea of “moving” the loop from outside the method to its inside. However, apart from the trivial gain via the reduction of the number of calls, this is not nearly enough to exploit the SIMD features of the CPU. In order to use SIMD CPU capabilities, the compiler needs to emit special assembly instructions (e.g. “add” versus “vaddp”) to the hardware. For this multiple options exist:

1. Autovectorisation: This consists of exposing the parallelism in the code and letting the compiler exploit the opportunities to replace scalar instructions with vector ones (without code changes). This would of course be the ideal option for portability and maintenance, but in practice it very rarely works without pragmas or heavy re-factoring of the code.
2. Explicit vector-oriented programming via intrinsic instructions. This manually instructs the compiler to use vector instructions. This requires refactoring and code changes to work, and

at the lowest level can be very machine and architecture dependent. Template-based API hiding low-level details like the Vc library [13] can be very helpful. Our experience shows that this approach gives good performance, portability and only little platform dependency. However it does require some code changes and refactoring.

3. Language extensions, such as Intel Cilk Plus Array notation [14] or OpenACC [15]. Our investigation in this area is just beginning and we will not elaborate more on it in this paper.

Given the above, we concentrated on the second approach above. We intend to continue studying the other two approaches, however, in order to come quickly to an "order of magnitude" answer as to whether micro-parallelisation and vectorisation could give us the expected performance benefits. So we started with the optimisation of methods to calculate the distance from inside and outside for three simple geometrical shapes (box, cone and tube) using the Vc library on an Intel Xeon machine. The first results can be shown in **Figure 4** and are indeed quite encouraging. It must be stressed that this is a very preliminary result and there is a lot of work ahead in SIMD-optimizing more complicated shapes, extending to other methods and to the coordinate transformation that uses a good fraction of the CPU time spent in geometry.

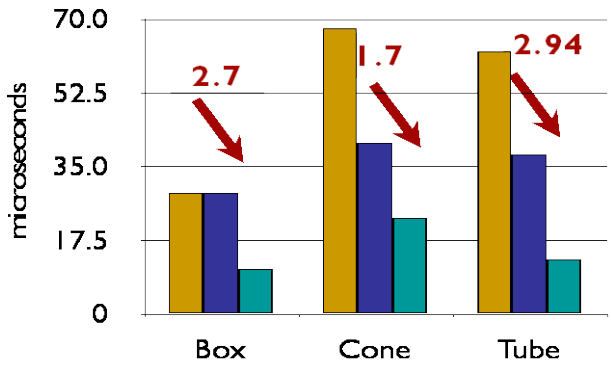

Figure 4. Performance improvement of the SIMD version of the method calculating the distance from a point inside the shape along the flypath for 1024 particles. The first column corresponds to the use of Root 5.34.09, the second an optimised version of Root and the third the SIMD version of the algorithms implemented via the Vc library with the AVX vector instructions. The numbers show the speedup of the SIMD version with respect to the optimised Root.

3. 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 designed and their implementation is in an advanced state. Preliminary results on the SIMD optimisation of the geometry routines are promising. The next steps will be the implementation of the Scheduler, the extension of the tests with the optimised SIMD geometry routines and the optimisation of more shapes, the completion of the sampling of the final states of the interactions and the realisation of a vector navigator. This will allow us to assemble a complete application that could do some basic particle transport and perform accurate timings. An important factor will also be the consideration of how our design could take advantage of GPUs.

Another important item that we will have to tackle is the estimation of the overheads implied in the creation and handling of the baskets and in the creation of thread-safe data structure. With the parallelization of the processing, keeping the results will be an additional challenge, involving also work on the random number generator. Inevitably this too will have an overhead that again we have to measure and qualify.

In conclusion we can say that the first steps in the design of a high performance detector simulation application have been taken and they are promising, while a vast and interesting programme of work lies ahead.

Acknowledgements

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 and Raman Shegal from the Bhabha Atomic Research Centre, Mumbai, India for their collaboration.

Bibliography

1. Jarp S, Lazzaro A and Nowak A 2012 The future of commodity computing and many-core versus the interests of HEP software. *International Conference on Computing in High Energy and Nuclear Physics* **396** pp 52-58 doi:10.1088/1742-6596/396/5/052058
2. Dongx X, Cooperman G, Apostolaks J, Nowak A, Jarp S, Asai M and Brandt D 2012 Creating and Improving Multi-Threaded Geant4 *International Conference on Computing in High Energy and Nuclear Physics* **396** doi:10.1088/1742-6596/396/5/052029
3. Dong X, Cooperman G and Apostolakis J 2010 Multithreaded Geant4: Semi-automatic Transformation into Scalable Thread-Parallel Software *Euro-Par 2010 - Parallel Processing* **6272** pp 287–303
4. Canal P, Elvira D, Hatcher R, Jun S and Mrenna S 2013 A Vision on the Status and Evolution of HEP Physics Software Tools *arXiv:1307.7452v1*
5. Group C 2013 *Forum on Concurrent Programming Models and Frameworks* http://concurrency.web.cern.ch
6. Johnston R 1963 A General Monte Carlo Neutronics Code *LAMS* **2856** Los Alamos Scientific Laboratory, Los Alamos
7. Cashwell E, Neergaard J, Taylor W and Turner G 1972 MCN: A Neutron Monte Carlo Code *Los Alamos Scientific Laboratory Report LA* **4751** Los Alamons National Laboratory, Los Alamos
8. Ford R and Nelson W 1978 The EGS code system – Version 3 *SLAC* **210** Stanford Linear Accelerator Center, Stanford
9. Brun R et al 1993 GEANT Detector Description and Simulation Tool (Version 3.21) *CERN Program Library* **W5013**, CERN, Geneve
10. Battistoni G, Muraro S, Sala P, Cerutti F, Ferrari A, Roesler S, Fassò A and Ranft J 2007 The FLUKA code: Description and benchmarking 2006 *Albrow M and Raja R eds. Hadronic Shower Simulation Workshop 2006,* Fermilab 6-8 September 2006 **896** pp 31-49
11. Allison J et al 2006 Geant4 developments and applications Nuclear Science *IEEE Transactions* **53** 1 pp 270-278
12. Apostolakis J, Brun R, Carminati F and Gheata A 2012 Rethinking particle transport in the many-core era towards GEANT 5 *Computing in High Energy Physics 2012* **396-2**

doi:10.1088/1742-6596/396/2/022014
13. Kretz M and Lindenstruth V 2012 Vc: A C++ library for explicit vectorization. Software: Practice and Experience 42 **11** pp 1409–1430
14. Intel Corporation 2013 *Cilk Plus* https://www.cilkplus.org
15. OpenACC Consortium 2013 *OpenACC Home* http://www.openacc-standard.org