Journal of Physics: Conference Series **513** (2014) 052038 doi:10.1088/1742-6596/513/5/052038

Vectorising the detector geometry to optimise particle transport

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

European Organisation for Nuclear Research (CERN), Geneva, Switzerland

E-mail: sandro.wenzel@cern.ch

Abstract. Among the components contributing to particle transport, geometry navigation is an important consumer of CPU cycles. The tasks performed to get answers to "basic" queries such as locating a point within a geometry hierarchy or computing accurately the distance to the next boundary can become very computing intensive for complex detector setups. So far, the existing geometry algorithms employ mainly scalar optimisation strategies (voxelisation, caching) to reduce their CPU consumption. In this paper, we would like to take a different approach and investigate how geometry navigation can benefit from the vector instruction set extensions that are one of the primary source of performance enhancements on current and future hardware. While on paper, this form of microparallelism promises increasing performance opportunities, applying this technology to the highly hierarchical and multiply branched geometry code is a difficult challenge. We refer to the current work done to vectorise an important part of the critical navigation algorithms in the ROOT geometry library. Starting from a short critical discussion about the programming model, we present the current status and first benchmark results of the vectorisation of some elementary geometry shape algorithms. On the path towards a full vector-based geometry navigator, we also investigate the performance benefits in connecting these elementary functions together to develop algorithms which are entirely based on the flow of vector-data. To this end, we discuss core components of a simple vector navigator that is tested and evaluated on a toy detector setup.

1. Introduction

The Geant-Vector prototype is one of the initiatives [1, 2] that try to recast High-Energy Physics (HEP) particle transport codes into a form that allows them to benefit from all performance dimensions on current and future (commodity) hardware – in particular from both multicore- (multithreading) and micro- parallelism (vectorisation). By doing so, these projects try to overcome the increasing gap between the performance of existing software and the ideal performance limits coming from advances in computing technology [3].

One of the important activities in the context of the Geant-Vector prototype is to factorise the CPU-intensive algorithms contributing to particle transport and to try to reshuffle both geometry and physics code in a vectorisable form. While first efforts were focused on investigating concurrency issues [1], the project now also addresses the opportunities offered by vector microparallelism.

The choice to start our investigation on the geometry component was stimulated by the fact that geometry calculations traditionally consume a considerable CPU budget in a typical detector simulation (up to $40-50\%$ of the transport time; estimating the fraction of the time


Content from this work may be used under the terms of the Creative Commons Attribution 3.0 licence. Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.
Published under licence by IOP Publishing Ltd

spent in the geometry with respect to the total simulation time is more difficult, since there are components [such as digitisation] which are highly experiment dependent). Clearly, it is our hope that the experience gained as a result of vectorising the geometry will help us implement similar strategies in other simulation components, such as physics processes, as well as in experiment dependent components such as digitisation, at a later stage.

Starting from a short review of vectorisation approaches, we will report below on the status of vectorising some of the shape navigation algorithms in the ROOT geometry library [4]. Building on top of these vectorised algorithms, we will then discuss the results for the higher level task of finding the distance to the next boundary within a volume that contains several "daughter volumes" making up a toy detector model. This benchmark allows us to estimate performance benefits according to different origins (gains due to refactoring, vectorisation, code locality, etc.), which will be helpful for extrapolating to more complex algorithms.

2. Vectorisation basics and programming models

Vector-units executing *single instructions on multiple data* (SIMD) in parallel were introduced in commodity hardware in the late 90's in form of extensions to the x86 architecture and instruction set (for example MMX, SSE, 3DNOW, AVX). These vector extensions allow the simultaneous application of particular instructions (say an *add* instruction) on multiple data elements leading to an increase in throughput compared to scalar (serial) handling of the same data elements. Current SIMD extensions (AVX, AVX2) can handle simultaneous operations on 4 doubles (8 floats) at the same time whereas vector units able to handle 8 doubles are already available on the Intel Xeon Phi and are announced in the upcoming AVX3 instruction set extension.

How can we make use of SIMD? The necessary prerequisite is the availability of (contiguous) data on which the same operations should be carried out. In the Geant-Vector prototype, this is being realised by grouping the particles in the same logical volume (potentially from different events) into a data-parallel container called a *basket* [1]. Basic algorithms can then process particles within the same basket in a loop-like fashion. In this circumstance, modern compilers are in principle able to generate SIMD vector instructions by autovectorising the relevant loop. In reality, our experience is (using different modern compilers) that this currently works only in a few cases (such as simple array operations in tight loops) and often requires substantial refactoring of the code. Finally, autovectorisation is usually an all-or-nothing procedure that, for instance, makes it hard to mix vector and scalar operations; as an example, a single scalar function call within a loop will usually prevent autovectorisation.

At the other extreme, complete control can be given to the developer by directly coding in a vector-oriented way using intrinsics or assembly instructions. Higher-level alternatives to this are currently available in the form of (non-standardised) vector types, available in some compilers or in the form of portable C++ template libraries [5, 6] that encapsulate the low-level details in template classes with a C++ high-level syntax (operator overloading). These libraries still require a reformulation of existing original code but allow, to a very large extent, for portable and maintainable code. For example, the Vc library is a free software library that eases explicit vectorisation of C++ code [5, 7] and which provides for us the essential advantage to write code in which SIMD vector operations are easily mixed with scalar code. Overall we have gained a good experience with Vc and the results reported below have all been obtained by making use of Vc version 0.73.

However, it should be noted that once we have formulated our algorithms in terms of vector operations, a move to alternative vector-oriented programming models/languages seems to be straightforward. Good experience has already been obtained from a direct *statement-to-statement* translation of Vc code into Intel CilkPlus [8] array notation. Equally, it should be straightforward to extend the vector-oriented code to different platforms such as accelerators (for instance GPUs). Ultimately, we have to see how a single code base can be practically maintained

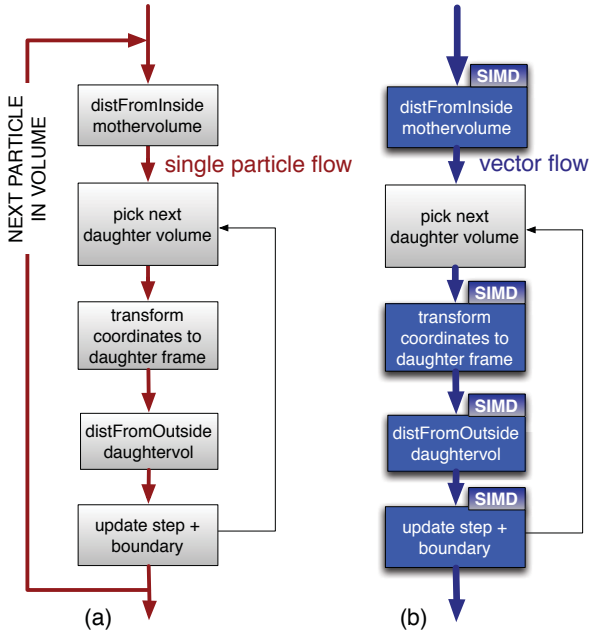


Figure 1. Representation of the core part of a geometry navigator addressing the problem of obtaining the distance to the next hit boundary of particles in a certain detector volume (which itself contains further daughter volumes). (a) Schema of a sequential algorithm versus (b) an algorithm based on flow of vectors.

for all targeted architectures. We are currently starting to investigate how the evolving standards OpenMP/OpenACC [9, 10] can help in this respect.

3. Towards a vector-oriented geometry navigator

The main CPU usage related to geometry during particle transport is due to the execution of navigation algorithms. The geometry navigator calls several elementary algorithms for each particle/track to calculate parameters such as geometrical steps, minimal (safe) distances to (volume) boundaries and to determine the containing volume (localisation) of particles within a complex detector geometry. In order to make efficient use of the SIMD capabilities and the availability of vector data in form of baskets [1, 11], the Geant-Vector prototype has to provide a geometry navigator that is able to process vectors of particles instead of operating on single particles/tracks in a sequential mode. Figure 1 sketches this concept for the core functionalities of the navigator by calculating the geometrical step length to the next boundary along the particle line of flight and the corresponding hit volume. The two essential components are the functions `distFromInside` (mothervolume) and `distFromOutside` (daughtervolume), which are used to determine the distance of a particle to the boundary of its current detector element as well as the distance to other volumes contained within this element along its line of flight.

This algorithm is missing the final localisation of particles in the detector geometry which has been left out for the first investigation. Note also that this is a rather simple form of a navigator without voxelisation techniques. In Figure 1a, the algorithm is based on processing single particles through a series of elementary algorithms while in Figure 1b the same algorithm is presented based on a flow of vector data. Essentially, going from the scalar version to the SIMD optimised vector version of the algorithm requires two steps, namely a refactoring of the API to enable passing vectors across algorithms, followed by the adaptation of each elementary algorithm to profit as much as possible from SIMD instructions. Note that both

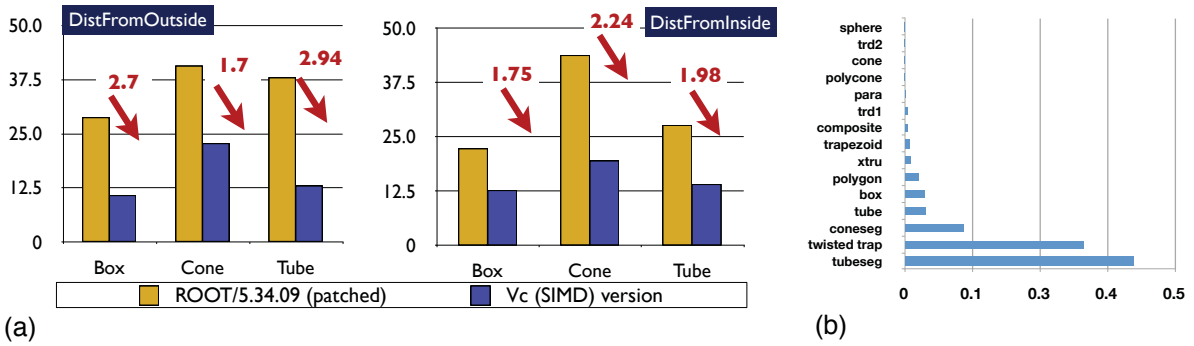


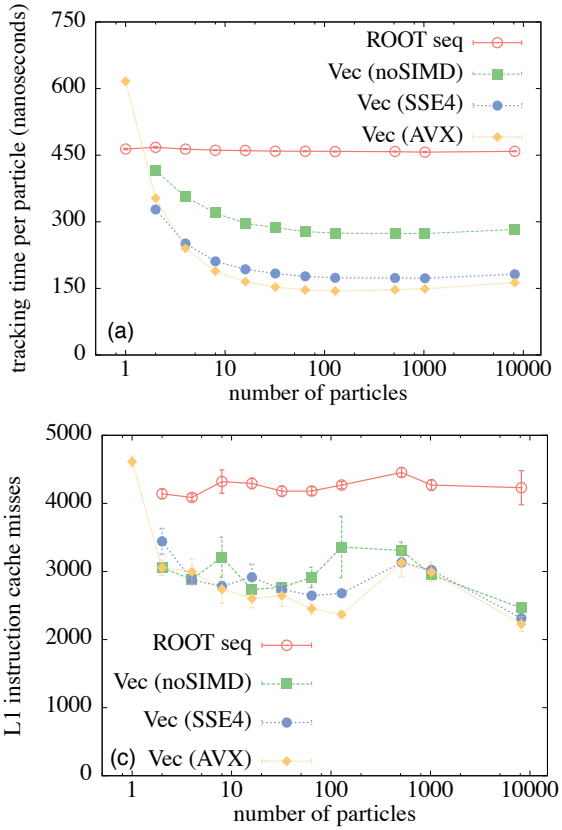
Figure 2. (a) Speedups obtained from vectorising simple algorithms for the box, cone and tube shapes from the ROOT shape library. The functions presented are `distFromOutside` and `distFromInside`, i.e. the distance to the shape boundary from a point outside and from inside the shape respectively. (b) We also show a simple estimate of the relative CPU budget (in percentage on the x-axis) for various shapes based on counting physical shapes in detectors of 33 existing HEP experiments and taking into account the scalar runtime cost for the function `distFromOutside`.

steps individually contribute to a performance gain over the scalar version. As will be shown in Section 3.2, the first step decreases the number of function calls, reduces the number of memory moves and improves code locality, while the second step increases throughput via microparallelism.

3.1. Elementary geometry algorithms

In order to progress towards a full vector geometry navigator, we have to provide the basic geometry algorithms (single blocks in Figure 1a) in a recast and optimised form. Among the most important basic algorithms are the calculations of the particle distances to the solids/shapes composing the detector. We hence started our investigation of the potential of SIMD instructions at the level of the shape methods, notably on the functions that calculate the distance of the particles, along the line of flight, to the inside (entering) of a shape or its distance to get outside (leaving), as well as on functions that calculate the minimum distance to any boundary (safety) and functions performing inside/outside checks. Because the Geant-Vector prototype uses the ROOT geometry, the current vectorisation work is based on the existing code in the ROOT geometry library [4]. However, an extension of this work to future standard libraries such as the Unified Solids [12] is foreseen.

The first shape tackled was the box since it is one of the most simple yet important geometrical forms (see Figure 2b for an averaged estimate of the importance of various shapes). The simplicity of the box gave us a good playground to quickly assess the various programming models and memory layouts. Considering the difficulties of trying to get the code to autovectorise, versus the relative ease of programming (and compiler independence) with a library like Vc, we then opted for the second choice for the purpose of this first performance evaluation. At the time of writing, several of the simple shapes, such as boxes, cones, tubes (including their segmented forms), have been successfully ported to Vc code. Figure 2a gives an overview of the speedup achieved so far for the most important methods. These benchmarks were run on an Intel Ivy Bridge machine (`Intel(R) Core(TM) i7-3770 CPU@3.40GHz (1 socket, 4 cores)`, SLC6 `2.6.32-358.18.1.el6.x86_64`, `gcc-4.7`, `Vc-0.73`, `ROOT-5.34.09`) with AVX instruction set and results are reported for (an arbitrarily chosen number of) 1024 particles in a basket. These



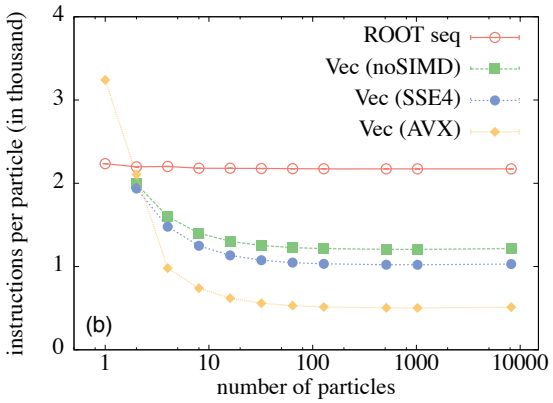


Figure 3. Results from the benchmark comparing the scalar algorithm (ROOT seq) with a vector-oriented algorithm using various degrees of usage of SIMD instructions [VEC(noSIMD), VEC(SSE4) and VEC(AVX)]. (a) Comparison of the runtime per particle showing a speedup factor of roughly 3 comparing the original version to the AVX code. (b) Comparison of the number of instructions executed per particle. (c) Comparison of level-1 instruction cache misses (arbitrary scale).

speedups, measured at between 1.7 and 3, are matching our expectations and provide a first confirmation that we are moving into the right direction.

Refactoring and optimisation work on more complicated shapes such as polycones (sequence of connected cone and tube segments) has started. First positive results for polycones with few z segments have been already obtained, while for polycones with many z segments the availability of optimised sequential algorithms (with voxelisation and table lookup techniques) makes it hard to better expose the parallelism in a way that can be exploited by SIMD-capable hardware because different particles will in general follow slightly different code paths. It will be one of the future challenges to adapt these special sequential optimisations in complicated shapes with microparallelism. However, note that we will anyway gain from the re-factored interface of those shapes alone by exploiting better code locality and less function calls.

Besides the shape algorithms, we have successfully SIMD-optimised various other elementary algorithms needed by a vector-oriented geometry navigator, such as coordinate transformations needed for coordinate frame conversions or min/max algorithms.

3.2. From elementary to complex vector algorithms

Building on top of the vector-enabled basic components, we have made a first implementation of the vector-oriented geometry navigator as shown in Figure 1b. This simple approach allows for a more realistic evaluation of the performance gains coming from the combined usage of vector-optimised algorithms used in the vector prototype.

To this end, we compare the performance of the sequential algorithm (Figure 1a) using the standard scalar approach from the existing ROOT package with the newly implemented version. To estimate a best-case scenario, a small toy detector setup was made out of already optimised shapes and which should serve as a first (standard) benchmark. The toy detector

Table 1. Statistics for instructions executed in the algorithm of Figure 1. The first line of numbers shows the relative reduction of the total instruction count (ALL) with respect to the sequential algorithm. The numbers in the second block show the fractions for simple memory moves (MOV), call instructions (CALL), SIMD instructions (ALL SIMD) as well as the subset of arithmetic SIMD instructions (ARITHM SIMD) relative to the total number of instructions within each column. These numbers are obtained for 16 particles and a comparison is done between the four algorithmic versions as explained in the text.

Instruction (type)	ROOT seq	VEC(noSIMD)	VEC(SSE4)	VEC(AVX)
ALL	1	0.6	0.52	0.29
MOV	0.296	0.116	0.132	0.163
CALL	0.036	0.0023	0.0026	0.0048
ALL SIMD	0.043	0.188	0.641	0.57
ARITHM SIMD	0.023	0.039	0.289	0.30

consists of a tubular mother volume containing two other tubes (beampipe, shield), four boxes representing detector plates and two cones (as endcaps). The intent was to have a non-trivial setup for the simple algorithm presented here. For more complex setups we would have to combine vectorisation with scalar optimisation techniques, such as voxelisation, which has to be addressed in a future step.

We filled the exclusive part of the mother volume with a large pool of random particle positions and directions. To run the benchmark, we pick N consecutive particles in memory starting from a random position in this pool and process them with both the sequential and vector algorithms to obtain the distances and next hit boundaries for all of them. We repeat this process P times and in each benchmark we keep the product of $N \times P$ constant to give the same amount of work to each benchmark run.

Besides comparing the runtime between the scalar and vectorised navigators, we also looked into different metrics given by the hardware performance counters of the CPU. We could directly measure and compare the code locality in terms of L1 instruction cache misses, branch misses and data cache misses. For this we interface with the perfmon library [13] with which we are able to read out the counters right before and after the specific code section. Additionally, details of the actual instructions executed in the relevant code sections are obtained using a custom binary instrumentation tool using the Intel Pin API [14].

A couple of key results from this study are shown in Figure 3 where data is included for the original scalar and sequential algorithm, for the refactored algorithm based on vector flow but without SIMD optimisation, and for the vector algorithms with SIMD optimisations (labeled in the Figure as ROOTseq, Vec(noSIMD), VEC(SSE4|AVX) respectively). Our main result is that we are currently able to speed-up the example algorithm of Figure 1 by a factor of $\gtrsim 3$ and that considerable gains are even seen for rather small number of particles in a basket. The speedup originates from various contributions as shown in the plots: just by refactoring into a vector interface [version VEC(noSIMD)], a performance gain with a factor ≈ 1.5 is seen in terms of the runtime. The SIMD instructions, using instruction sets SSE4 or AVX, then give the actual gains from microparallelism. We can track the origin of the gains by analysing the dynamic instruction mix actually executed in the benchmark. Some important numbers obtained from this analysis are summarised in Table 1. Going from a scalar to vector interface allows to reduce the number of function call instructions accompanied by a massive reduction in simple memory moves (such as those used to save registers on the stack). When introducing SIMD optimisations with Vc, the overall number of instructions further shrinks and the CPU vector unit is used to

a much higher degree. Using the hardware performance counters, we have also confirmed that the number of instruction cache misses is considerable reduced due to better code locality when using the vectorised interfaces (Figure 3c). We expect this effect to become even more important with more complex algorithms.

4. Summary and Outlook

Focusing on the geometry component in particle transport codes, we described the status of our vectorisation effort within the Geant-Vector prototype. On the basis of an explicit vector-oriented programming model with a high level C++ template library (Vc), we reported on significant performance improvements for important distance methods of simple shape classes in the ROOT geometry library. These SIMD improvements together with a new vector API, allowing to pass vectors of data across the basic algorithmic components, add up to a total performance gain of the order of 300% for the example of a simple vector-oriented navigation algorithm in a toy-geometry. We would like to note that the current work serves as a first proof of principle exercise for studying the sustainability of vector algorithms at different granularity. As such, the present demonstrator has a limited scope, in particular because we are aware that at the full complexity the geometry algorithms include optimisation methods such as voxelisation techniques. Combining such advanced techniques with vectorisation may require a complete redesign of such optimisations.

There are a multitude of challenges to be tackled in the future: the simple navigation algorithm above has to be extended to be a full geometry navigator, we have to SIMD optimise more complex geometrical shapes and most importantly we have to figure out how more complicated algorithms (voxelisation) can make use of SIMD instructions. We have also to extend and validate our findings to other hardware, such as GPUs.

Acknowledgments

We thank Marilena Bandieramonte, Raman Sehgal, Juan Valles Martin for contributions to the Vc coding. We also would like to thank Pere Mato, Vincenzo Innocente (PH-SFT, CERN), Laurent Duhem (Intel), Andrzej Nowak (Openlab, CERN) and Matthias Kretz (University of Frankfurt) for very stimulating discussions.

References

[1] Apostolakis J, Brun R, Carminati F and Gheata A 2012 *Journal of Physics: Conference Series* **396** 022014 URL http://stacks.iop.org/1742-6596/396/i=2/a=022014

[2] Canal P, Elvira D, Hatcher R, Jun S Y and Mrenna S 2013 *ArXiv e-prints* (*Preprint* 1307.7452)

[3] Jarp S, Lazzaro A and Nowak A 2012 *Journal of Physics: Conference Series* **396** 052058 URL http://stacks.iop.org/1742-6596/396/i=5/a=052058

[4] Gheata A The root geometry URL ftp://root.cern.ch/root/doc/18Geometry.pdf

[5] Kretz M and Lindenstruth V 2012 *Software: Practice and Experience* **42** 1409 URL http://dx.doi.org/10.1002/spe.1149

[6] Esterie P, Gaunard M, Falcou J, Lapresté J T and Rozoy B 2012 *PACT* ed Yew P C, Cho S, DeRose L and Lilja D J (ACM) p 431 ISBN 978-1-4503-1182-3 URL http://doi.acm.org/10.1145/2370816.2370881

[7] The Vc homepage URL http://code.compeng.uni-frankfurt.de/projects/vc

[8] The Intel CilkPlus homepage URL https://www.cilkplus.org

[9] OpenMP-Consortium URL http://openmp.org/wp/openmp-specifications/

[10] OpenACC-Consortium 2013 URL http://www.openacc-standard.org

[11] Apostolakis J, Brun R, Carminati F, Gheata A and Wenzel S 2013 *Journal of Physics: Conference Series (CHEP2013)*

[12] Gayer M, Apostolakis J, Cosmo G, Gheata A, Guyader J M and Nikitina T 2012 *Journal of Physics: Conference Series* **396** 052035 URL http://stacks.iop.org/1742-6596/396/i=5/a=052035

[13] The perfmon/libpfm homepage URL http://perfmon2.sourceforge.net/

[14] The Intel Pin homepage URL http://www.pintool.org