Speeding up HEP experiment software with a library of fast and auto-vectorisable mathematical functions

Danilo Piparo, Vincenzo Innocente, Thomas Hauth
CERN CH-1211, Switzerland
E-mail: danilo.piparo@cern.ch, vincenzo.innocente@cern.ch, thomas.hauth@cern.ch

Abstract. During the first years of data taking at the Large Hadron Collider (LHC), the simulation and reconstruction programs of the experiments proved to be extremely resource consuming. In particular, for complex event simulation and reconstruction applications, the impact of evaluating elementary functions on the runtime is sizeable (up to one fourth of the total), with an obvious effect on the power consumption of the hardware dedicated to their execution. This situation clearly needs improvement, especially considering the even more demanding data taking scenarios after the first LHC long shut down. A possible solution to this issue is the VDT (VectorisD maTh) mathematical library. VDT provides the most common mathematical functions used in HEP in an open source product. The function implementations are fast, can be inlined, provide an approximate accuracy and are usable in vectorised loops. Their implementation is portable across platforms: x86 and ARM processors, Xeon Phi coprocessors and GPGPUs. In this contribution, we describe the features of the VDT mathematical library, showing significant speedups with respect to the LibM library and comparable accuracies. Moreover, taking as examples simulation and reconstruction workflows in production by the LHC experiments, we show the benefits of the usage of VDT in terms of runtime reduction and stability of physics output.

1. Target Accuracy
Target accuracy is a performance optimisation pattern applicable to HEP software. It consists in the tradeoff of numerical accuracy in mathematical calculations for an enhanced runtime performance, preserving or minimally altering the values of the physics quantities yielded. Therefore, in the context of a complex computation, the goal of target accuracy consists of choosing the minimal accuracy in the intermediate steps to obtain a correct final result. The amount of runtime accountable to mathematical functions in typical HEP workflows like event reconstruction and simulation is significant. The tradeoff between accuracy and execution speed of mathematical functions in HEP should therefore be of concern.

Different optimised mathematical libraries exist. Some solutions are free, for example LibM part of the GNU C Library [1] or the AMD LibM [2], other are commercial, like the Intel Math Kernel Library (MKL) [3]. This paper focusses on Vectorised maTh (VDT) [4], a free and open source (LGPL3) mathematical library we developed. The performance and numerical accuracy of VDT functions is discussed, also in the context of real life LHC experiment data processing workflows.
2. The VDT Mathematical Library

The VDT library is a collection of single and double precision highly optimised mathematical functions, yielding results which have an adequate accuracy. It is written in standard C++. This choice has several advantages over the usage of assembly. For example, it allows for maximum portability: the C/C++ compiler is responsible for producing machine code executable on the desired platform. In addition, this approach allows the source code to automatically scale for future hardware: in order to achieve top performance, VDT leverages the capability of compilers to emit machine instructions which are optimised for the target architecture.

Having chosen this role for the compiler, maximum portability is easily achieved: VDT can be run on every platform for which a compiler exists. VDT symbols are meant to be inline and have different names from the traditional ones, i.e.: `vdt::fast_<math function name>`, where `math function name` stands for example for sin, cos, atan. This gives the possibility to mix VDT functions with the pre-existing ones in a given package in a controlled way. On the other hand, the package comes with a flexible CMake [5] build system which allows, if desired, to build a shared library usable with a preload strategy, therefore automatically replacing all function calls to the system’s default mathematical library with VDT ones.

In addition to being fast, VDT functions are inline and vectorisable. This allows them to be used in autovectorised loops. Within the library itself, this powerful feature is exploited to offer both scalar and array signatures, i.e. with signatures like `T (T)` and `void (int, T*, T*)` respectively. The array signatures can be used to calculate the value of a mathematical function on an array of values in bulk, exploiting vectorisation.

The implementation of the functions of VDT is influenced by the well known Cephes [6] library. The double precision functions rely on Padé’ approximants [7], while the single precision ones on polynomial approximants. An exception to this scheme is represented by the inverse square root (useful for example for vector normalisation) which was inspired by the “Quake 3 fast inverse square root” [8], both for the single and double precision versions.

Sections 2.1 and 2.3 review the runtime performance and numerical accuracy of VDT functions while in section 3 the influence of VDT functions on real life HEP workflows is discussed.

2.1. Runtime Performance

The platform used for this test was a Scientific Linux 6 machine equipped with a processor offering SSE, AVX, AVX2 and fma instruction sets (an Intel Core i7 – 4770K at 3.50GHz [9]). The glibc and VDT versions installed are the 2.12-1.107.el6_4.4 and v0.3.6 respectively, while the compiler used was GCC 4.8.1.

Table 1 shows the time necessary in ns to calculate a particular mathematical function on a certain value, in the scalar and vector case. LibM functions are taken as reference. For the test, a set of five millions random numbers was considered, ranging according to the function domains at most from -5000 to 5000. In order to reduce the statistical uncertainty affecting the mean time values, the measurements were repeated 250 times. For the measurements influenced by vectorisation, different instruction sets were used, namely SSE, AVX, AVX2 and AVX2 in combination with FMA.

LibM is a rock-solid reference for HEP calculations. Nevertheless, the usage of a more optimised mathematical library like VDT can be a clear advantage when dealing with performance critical applications.

1 The only C++ feature used is the namespace keyword.
Table 1: Time in ns needed to calculate mathematical function on one particular value. The symbols terminating with \( f \) denote a single precision function. Already in scalar mode, VDT offers significant speed-ups. The usage of array signatures shows the advantage of vectorisation. Since values of polynomials are calculated, the FMA instruction set is a considerable advantage, especially when overheads due for example to range reductions are negligible or not present. *The measurement for Sinf, Cosf and Atanf was carried out also in the \([-10,10]\) domain, obtaining for timings \(23.61 \pm 1.38, 24.87 \pm 1.46 \) and \(29.05 \pm 1.71\) respectively.

| Func | LibM  | VDT   | VDT+fma | SSE     | AVX    | AVX2   | AVX2+fma |
|------|-------|-------|---------|---------|--------|--------|----------|
| Exp  | 102.00 ± 5.2 | 8.00 ± 0.50 | 5.80 ± 0.40 | 3.55 ± 0.28 | 3.58 ± 0.28 | 1.73 ± 0.17 | 1.32 ± 0.17 |
| Log  | 32.28 ± 1.66 | 11.48 ± 0.66 | 9.84 ± 0.59 | 4.35 ± 0.31 | 3.93 ± 0.29 | 2.20 ± 0.18 | 1.47 ± 0.18 |
| Sin  | 77.77 ± 3.93 | 16.50 ± 0.90 | 16.51 ± 0.91 | 6.21 ± 0.39 | 5.66 ± 0.37 | 2.65 ± 0.20 | 1.94 ± 0.20 |
| Cos  | 77.60 ± 3.92 | 14.41 ± 0.80 | 13.24 ± 0.75 | 5.12 ± 0.34 | 4.77 ± 0.33 | 2.31 ± 0.19 | 1.48 ± 0.18 |
| Tan  | 89.69 ± 4.56 | 10.59 ± 0.62 | 8.96 ± 0.54 | 4.42 ± 0.31 | 4.17 ± 0.30 | 3.17 ± 0.23 | 3.14 ± 0.26 |
| Asin | 21.25 ± 1.13 | 8.94 ± 0.54 | 6.95 ± 0.45 | 5.84 ± 0.38 | 5.02 ± 0.34 | 5.03 ± 0.32 | 5.03 ± 0.34 |
| Acos | 21.63 ± 1.15 | 9.11 ± 0.55 | 7.31 ± 0.47 | 5.96 ± 0.38 | 5.00 ± 0.34 | 5.09 ± 0.32 | 5.05 ± 0.34 |
| Atan | 15.56 ± 0.86 | 8.41 ± 0.52 | 6.66 ± 0.43 | 5.59 ± 0.36 | 4.99 ± 0.34 | 5.09 ± 0.32 | 5.08 ± 0.34 |
| Isqrt | 5.66 ± 0.40 | 4.27 ± 0.33 | 2.76 ± 0.26 | 1.83 ± 0.20 | 1.42 ± 0.18 | 0.39 ± 0.11 | 0.11 ± 0.13 |
| Atan2 | 36.38 ± 1.87 | 19.94 ± 1.08 | 18.90 ± 1.03 | 12.68 ± 0.70 | 8.32 ± 0.48 | 8.37 ± 0.48 | 7.71 ± 0.44 |

2.2. **VDT on ARM**

Being implemented in plain C code, VDT can be easily used also on non-x86 platforms like ARM as long as a C/C++ compiler for those platforms exist. Table 2 shows the results of a test identical to the one described in section 2.1 on a smaller scale (0.5M numbers and 150 repetitions) on a Cortex-A9 ARMv7 32 bits processor at 2.0 GHz [10].

2.3. **Numerical Performance: Accuracy**

Table 3 summarises the numerical performance of VDT. The quantities studied are the maximum and average different bit between the values yielded by VDT and LibM with the same argument, using a large (one million) set of input values. The values yielded by VDT are slightly different from the ones of LibM, but the achieved accuracy is probably sufficient for numerous applications.
Table 2: The gains of VDT are decisive also on RISC architectures. The library can be used transparently: the difficulties of tailoring the machine code for the target architecture are solved by the compiler. *The measurement for Sinf, Cosf and Atanf was carried out also in the [-10,10] domain, obtaining for timings 92.93 ± 7.61, 94.15 ± 7.69 and 119.27 ± 9.77 respectively.*

| Func  | Double Precision | Single Precision |
|-------|------------------|------------------|
|       | LibM             | VDT              | LibM             | VDT              |
| Exp   | 154.75 ± 13.68   | 71.41 ± 6.84     | 96.71 ± 8.84     | 61.98 ± 6.00     |
| Log   | 152.67 ± 13.47   | 64.64 ± 6.28     | 59.78 ± 5.83     | 59.73 ± 5.84     |
| Sin   | 201.85 ± 17.55   | 57.95 ± 5.74     | 1006.18 ± 85.36* | 43.84 ± 4.53     |
| Cos   | 198.81 ± 17.31   | 54.91 ± 5.49     | 992.92 ± 84.55*  | 40.85 ± 4.30     |
| Tan   | 290.29 ± 24.90   | 96.45 ± 8.88     | 1026.74 ± 87.09* | 77.35 ± 7.26     |
| Asin  | 99.16 ± 9.08     | 77.96 ± 7.38     | 103.95 ± 9.43    | 50.87 ± 5.11     |
| Acos  | 94.43 ± 8.69     | 78.91 ± 7.43     | 118.76 ± 10.65   | 51.91 ± 5.20     |
| Atan  | 127.98 ± 11.46   | 75.39 ± 7.17     | 64.50 ± 6.20     | 56.77 ± 5.58     |
| Isqrt | 24.73 ± 3.09     | 52.00 ± 5.27     | 14.09 ± 2.21     | 19.19 ± 2.60     |
| Atan2 | 187.17 ± 16.47   | 89.68 ± 8.51     | 95.84 ± 8.85     | 56.39 ± 5.63     |

Table 3: VDT numerical performance: average and maximum different bit in comparison with LibM.

| Func  | Double Precision | Single Precision |
|-------|------------------|------------------|
|       | Avg              | Max              | Avg              | Max              |
| Exp   | .14              | 2                | 3.4              | 6                |
| Log   | .42              | 2                | .26              | 2                |
| Sin   | .25              | 2                | .24              | 6                |
| Cos   | .25              | 2                | .24              | 6                |
| Tan   | .35              | 2                | .52              | 6                |
| Asin  | .32              | 2                | .6               | 3                |
| Acos  | .39              | 8                | .48              | 7                |
| Atan  | .33              | 1                | .37              | 2                |
| Isqrt | .45              | 2                | 3.7              | 7                |
| Atan2 | .27              | 2                | .35              | 2                |

3. Selected Examples from HEP data processing workflows

Real life workflows from the LHC experiments were studied in order to assess the effect of the usage of VDT mathematical functions in terms of code and physics performance. The performance gains reported are conservative, indeed VDT functions were not used as inlined symbols but compiled in a shared library which was preloaded, therefore replacing all calls to LibM with VDT. It can be noted that this strategy, chosen since it requires no modification in the code, leads to a runtime performance worse than the one that could be obtained using inlining of symbols. The experiment software used for the test dates from the beginning of 2013.
3.1. CMS Simulation and Reconstruction

CMS adopted VDT in 2011 for some of its reconstruction algorithms and the package is part of the CMS Software externals [11]. Two workflows were analysed for CMS: simulation and reconstruction, both considering Monte Carlo input. The process studied was $t\bar{t}$ quark pairs production because the decays of the Top quark result in heavily boosted leptons and hadronic jets, therefore exercising in various ways the experiment’s software. For the reconstruction, on average 25 pileup events were overlayed to the main one. Figure 1 shows the relative speed-ups achievable using VDT as a mathematical library for the reconstruction and simulation, i.e. 5.3% and 6.6% respectively.

The physics performance of the code was checked both for simulation and reconstruction. Almost 120,000 pairs of CMS official data quality monitoring histograms obtained with and without preloading the VDT mathematical library were compared using the RelMon tool [13]. Although this cannot be considered a detailed validation, no significant change was spotted. A plot representative of the differences introduced is shown in figure 2.

The time necessary to start-up the CMS framework and simulate the first event was also
benchmarked, since this measure represent the start-up time of the application and is relevant in general but in particular for example during the development cycle. The usage of VDT instead of LibM leads to a speed-up of 25%.

The CMSSW version used for this test was 6.2, in combination with GCC 4.7 and Geant 4.9.6.p02. The test machine was an Intel Core i7-3770, with a clock frequency of 3.40 GHz.

3.2. Simulation of ALICE
A study similar to the one described in 3.1 was carried out also considering ALICE simulation performed with Geant4. Figure 3 shows the effect of the usage of VDT.

![Figure 3: Left: runtime performance of the ALICE simulation. An overall speed-up of 7 percent is achieved.](image)

The AliRoot version used for this test was r63037, in combination with GCC 4.7, ROOT 5.34.09 and geant4.9.6.p02. The test machine an Intel Core i7-3770, with a clock frequency of 3.40 GHz.

4. Conclusions
A highly optimised, vectorisable, extremely portable, free and open source library was developed and is presently distributed: VDT. Given the different names of VDT mathematical functions, it can be used in combination with other mathematical libraries and being developed in plain C, is portable on all platform for which a C/C++compiler exist. decisive speed-ups (factors of 10 are not uncommon) could be measured on different platforms, while the numerical accuracy with respect to LibM remains high, i.e. on average less than 1 ULP (Unit in Last Place) both for single and double precision implementations. Some experiments like CMS already adopted it for some of their algorithms.

VDT was exercised on real life workflows of the LHC experiments. The potential performance gain obtainable with the library was measured to be approximatively 6% for CMS simulation and reconstruction and 7% for Alice’s simulation with Geant4. The improvement in the start-up time of CMS simulation was on the other hand measured to be equivalent to 25%. A preliminary study of the physics performance has shown that no significant deviation from the results obtained with LibM was present when using the approximate VDT mathematical functions.

5. The future
Even if VDT will continue to be distributed as a standalone package, it is being integrated at the time of writing in ROOT [14]. Further improvements of the numerical accuracy are planned.
as well as the implementation of new mathematical functions.

The studies described in this paper stress the fact that an optimised mathematical library can lead with a very limited effort (mainly validation) to significant performance improvements. On the other hand, the authors believe that much more can be done, for example with the extensive usage of polynomial approximations of full formulas.

The ultimate goal would be represented by a veritable meta mathematical library, which could allow to tailor the perfect mathematical function for a given context, parametrised by a certain needed accuracy and input argument range.

References

[1] The GNU C library https://www.gnu.org/software/libc
[2] AMD LibM http://developer.amd.com/tools-and-SDKs/CPU-Development/libm
[3] Intel’s MKL http://software.intel.com/en-us/intel-mkl
[4] The VDT mathematical library. https://svnweb.cern.ch/trac/vdt
[5] CMake http://www.cmake.org
[6] The Cephes library http://www.netlib.org/cephes
[7] Cuyt A 1984 Pade Approximants for Operators: Theory and Applications (Springer)
[8] Quake 3 fast isqrt ftp://ftp.idsoftware.com/idstu/source/quake3-1.32b-source.zip
[9] Intel corei7 4400k http://ark.intel.com/products/75123/Intel-Core-i7-4770K-Processor-8M-Cache-up-to-3.90-GHz/wapkw=i7-4770k.
[10] ARMv7 Cortex-A8 http://infocenter.arm.com/help/topic=/com.arm.doc.ddi0406b/index.html.
[11] Hauth T, Innocente V and Piparo D 2012 Development and Evaluation of Vectorised and Multi-Core Event Reconstruction Algorithms within the CMS Software Framework J. of Phys.: Conf. Ser. vol 396
[12] The Geant4 Collaboration 1998 Geant4 user’s documents (CERN, Geneva)
[13] Piparo D 2012 Relmon: A general approach to qa, validation and physics analysis through comparison of large sets of histograms. J. of Phys.: Conf. Ser. vol 396
[14] Brun R et al. 1997 Nucl. Inst. Meth. In Phys. vol 389