

Custom-Precision Mathematical Library

Explorations for Code Profiling and Optimization

David Defour[§], Pablo de Oliveira Castro^{*†}, Matei Iştoan^{*†}, and Eric Petit^{†‡}

^{*} University of Versailles – Li-PaRAD, Email: {pablo.oliveira, matei.istoan}@uvsq.fr

[†] Exascale Computing Research, ECR

[‡] Intel Corporation, Email: eric.petit@intel.com

[§] University of Perpignan, Email: david.defour@upvd.fr

Abstract—The typical processors used for scientific computing have fixed-width data-paths. This implies that mathematical libraries were specifically developed to target each of these fixed precisions (binary16, binary32, binary64). However, to address the increasing energy consumption and throughput requirements of scientific applications, library and hardware designers are moving beyond this one-size-fits-all approach. In this article we propose to study the effects and benefits of using user-defined floating-point formats and target accuracies in calculations involving mathematical functions. Our tool collects input-data profiles and iteratively explores lower precisions for each call-site of a mathematical function in user applications. This profiling data will be a valuable asset for specializing and fine-tuning mathematical function implementations for a given application. We demonstrate the tool’s capabilities on SGP4, a satellite tracking application. The profile data shows the potential for specialization and provides insight into answering where it is useful to provide variable-precision designs for elementary function evaluation.

Index Terms—HPC, libm, floating-point, custom-precision, optimization, specialization

I. INTRODUCTION

A growing interest to adapt floating-point formats to the real needs of applications is becoming ever more ubiquitous. This process has been successfully conducted by the AI community which has settled on the BF16 [1] and fp16 [2] formats, in order to increase performance and efficiency. Similar benefits have been achieved in other domains [3], [4] by reducing the precision of basic operations (+, −, *, /) and harnessing hardware-support for multiple internal floating-point formats.

Oberman et al. [5] demonstrate that neglecting optimizations of infrequent operations, such as division and square root, can severely impact performance. We believe that elementary functions should not be neglected either when optimizing for mixed-precision. Even though elementary functions are not widely available in hardware and infrequently used in applications, their impact can be important. HPC Patmos neutronic solver [6] spends 70% of the execution time in the mathematical library (*libm*) functions. This confirms similar observations by CERN [7] on their HPC codes.

Various mathematical libraries specialize evaluation schemes for different accuracy/performance trade-offs [8]. Recent developments such as *metalibm* [9] automatically generate elementary functions to best fit the hardware and

accuracy constraints. These works highlight that a trade-off can be explored between performance, accuracy and precision.

In this paper, we propose a tool for collecting input intervals and output required precision profiles from real applications in order to guide the design of specialized mathematical libraries. Indeed, considering limited input data ranges and application-focused output accuracy could drastically influence the implementation performance. We demonstrate the tool’s capabilities on SGP4, a satellite tracking application. The profile data shows the potential for specialization and provides insight into where it is useful to provide variable-precision designs for elementary function evaluation.

II. RELATED WORKS

Contrary to basic operations, properties of elementary function are not standardized mainly because the correctly rounded property is difficult to achieve [10], [11]. As a consequence, there are numerous available implementations of such functions, either in software or hardware, each representing a different trade-off between, accuracy, performance, hardware requirements and programming language. The most notable mathematical library embedding different trade-offs are the Vector Mathematical Functions from Intel’s MKL library [8], which offers three accuracy modes: High-/Low-accuracy and Enhanced Performance. Another example are Nvidia’s GPUs, which embed dedicated hardware for fast approximation of some functions and software implementation of more accurate and larger input range versions [12]. This has led to the OpenCL 2.2 standard which defines the requirements in terms of accuracy of mathematical functions from `half` to `double` [13].

Developing and maintaining multiple implementations for each function is a daunting endeavor. Several tools have been proposed to automate this task either, for hardware or software implementation of such functions [9], [14].

Porting the concept of memoization to mathematical functions has been explored in [15], [16] where the authors investigated how considering real input-data profile can be used to optimize the evaluation. However, they did not evaluate the potential decrease in accuracy.

III. SIMULATING VARIABLE PRECISION AND RANGE FOR MATHEMATICAL FUNCTIONS

Our approach to simulate mathematical functions with reduced range/precision is twofold: first we transparently interpose calls to mathematical functions, then the `VPREC-libm` library computes the result in a reduced precision.

A. Library Call Interposition

In Linux, the dynamic loader offers the possibility to intercept dynamic library calls, so that a custom library is called instead. This is achieved by setting the `LD_PRELOAD` environment variable. This interception method works out-of-the-box with a compiled binary and is transparent to the user.

We use it to replace standard calls to the `libm` with custom calls to our own `VPREC-libm` library which simulates non-standard precisions and range. This approach is flexible, but has two limitations:

- it is not applicable to statically linked programs: for those, the user must manually re-link the program against the `VPREC-libm`;
- it only intercepts library calls; to ensure that we intercept all operations we disable compiler optimizations which replace calls by hardware intrinsics (such as `sqrtsd` assembly instruction in IA-64). Fortunately, the `-fno-builtin` flag disables these optimizations in most standard compilers.

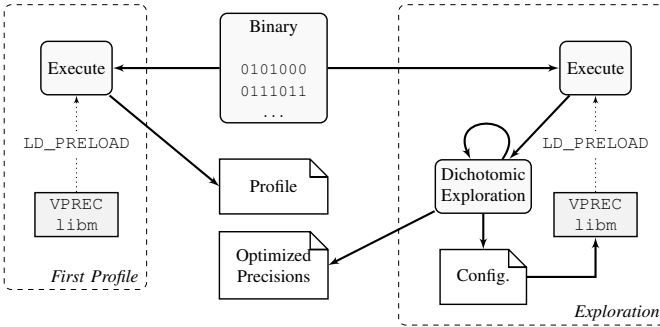


Fig. 1: VPREC-libm optimization process overview

B. Implementation of the VPREC-libm

The interposed mathematical call is handled by `VPREC-libm`, which returns the result in the target floating-point format. For example in double precision, one can customize the bit length of the pseudo-exponent $r \in [1, 11]$ and the pseudo-mantissa $p \in [0, 52]$.

`VPREC-libm` operates in two steps. First, it computes a `binary128` result \tilde{z} by calling the corresponding mathematical function from the GCC's `libquadmath`.

Then, \tilde{z} is converted to the target format using `Verificarlo-VPREC` [3]. If \tilde{z} is representable in the target range, a faithfully rounded result at target precision p is returned. If \tilde{z} is outside the target range, `VPREC` returns $\pm\infty$ for overflows and ± 0 for underflows. Rounding is achieved by adding a ulp at precision $p + 1$ followed by a truncation ($\lfloor \tilde{z} + 2^{e_z - p - 1} \rfloor_p$).

C. Exploring Precision Requirements Using VPREC-libm

`VPREC-libm` can be used in two modes: *profiling*, and *execution*. In profiling mode, `VPREC-libm` creates a profile of the executed code. For each call, it updates the boundaries of the operands and output intervals, the number of occurrences for the unique program address and stack trace from which the call is made. This information is aggregated and processed to produce execution statistics.

In execution mode, `VPREC-libm` accepts a configuration file which specifies the floating-point formats to use at each call-site. An initial config file is generated automatically after a profiling run, containing information for all encountered call-sites, which the user can later modify as it is required.

Taking advantage of this functionality, we developed a method for exploring the optimization potential of a given floating-point code. On a broad scale, we perform a dichotomic search for the minimal output precision of the `VPREC-libm` functions which meets the user-specified correctness criteria. This search is applied sequentially per call-site and converges in logarithmic time, in the size of the mantissa. As we are dealing with a vast search-space, the order in which this optimization is performed is quite important. According to Amdahl's law, optimizing the code in which most of the time is spent results in the biggest performance gains. Therefore we prioritize the call-site exploration by call frequency as a heuristic. Figure 1 summarizes this process.

IV. EXPERIMENTAL RESULTS

The example discussed throughout this section illustrates the potential of our proposed method, applied to a real-world astrophysics application used to predict the position and the velocity of Earth-orbiting objects (most notably satellites).

A. Satellite Tracker: SGP4

In order to track the position and the velocity of satellites, a common technique is to use *simplified general perturbations* (SGP) models, which predict the influence of drag, the Earth's shape, as well as that of the sun and the moon on the trajectory. Work on SGP started in the 1960's, but at the beginning of the 1980s NORAD¹ released the equations and the source code to predict satellite positions [17]. NORAD maintains and periodically refines data sets on all resident space objects, ensuring the accuracy of trajectory predictions. The data-sets were made available to users, through NASA, being the only source of readily available orbital data.

However, a user could not just go ahead and use any prediction model she wished, she had to use the same model employed to generate the data-set, even if the user's choice might have had better performance, an aspect highlighted in [17]. This made SGP, especially its SGP4/SDP4 variants, commonplace among users wishing to use NORAD's orbital data, distributed as *two-line element* (TLE) sets. *Near-Earth* objects (period less than 225 minutes) are tracked using SGP4, while *deep-space* objects (period over 225 minutes) are tracked

¹North American Space Defense Command

using SDP4, which also models the gravitational effects of the moon and the sun and certain Earth harmonics.

In the period following the original release of SGP, a multitude of code variations came to exist, making interoperability and compatibility an issue. For experimental purposes, we will use the SGP4 version documented in [18], which is a community effort to keep a version up to date with the models and TLE data-sets used by NORAD. The mathematical models used throughout are those presented in [19]. The C++ version used for the experiments is the one provided by *Celestrak*, available [online](https://celestrak.com/publications/AIAA/2006-6753/)². Only minimal changes were made, in order to ensure that the program runs correctly on Linux. This has not affected the program outputs, as verified against the provided test-suites.

B. Results

We applied our method to the data-set `sgp4-all.tle` discussed in [18]. Figure 2 illustrates our profiling and precision exploration results for the 50 *libm* call-sites with the most calls. At the top we show the number of occurrences for different *libm* call-sites. In the middle graph we show for each call-site the dynamic range of the input data, as used in the original code, extracted from the profile produced by `VPREC-libm`. Finally, in the bottom graph we show for each call-site the required precision of the outputs determined by our exploration with the `VPREC-libm` method.

We can observe that the second and third call-sites are almost twice more frequent than the next ten entries. The output precision cannot be significantly decreased for the two first call-sites, as shown in the bottom sub-figure. Indeed they occur at the very end of the algorithm, directly influencing the outputs. On the other hand, the dynamic range of the input is quite reduced at these two call-sites.

Analyzing the code, we notice that these calls to `sin()` and `cos()` take the same argument, as they are part of a rotation. Therefore, they could be replaced with a call to `sincos()`, effectively reducing their combined workload by almost a factor of two, similar to what is done in [20].

Analyzing the rest of the call-sites, we can observe two general trends. The first are call-sites where the required precision is close to the default one; here we only manage to save 5–6-bits. The second are call-sites where the required precision hovers around the 28-bits mark. A plausible explanation for these results can be found through a bit of computer-science archaeology. SGP4 was first developed in *FortranIV*³, on a *Honeywell-6000*⁴ series computer [17]. This machine had 36-bit words, so floating-point numbers had a 8-bit exponent and a 28- or 64-bit significand, for single- or double-precision, respectively. As noted in [18], the code originally used a mix of single- and double-precision computations. With the evolution of the underlying hardware, the code was moved to double-precision throughout, which made for a smoother behavior, but did not improve the accuracy. These observations are indeed

coherent with our findings on optimizing the precision of the outputs of the mathematical functions.

Figure 3 shows the results of our method on to a data-set containing just the satellite number 5, which is the first satellite in the `sgp4-all.tle` dataset. This is a near-Earth satellite, which means that its trajectory is tracked using SGP4, not SDP4. Its period, perigee and eccentricity ensure that no corner case is triggered in the model. The only particularity of this example is that it uses the *TEME*⁵ coordinate system, which requires a conversion to a more standard coordinate systems. Indeed, avoiding exceptional cases in the model shows that considerably lower precisions can be used throughout. The trends for the precision of the math functions’ output remains mostly the same. We notice, that over a third of the functions could be evaluated in single-precision, requiring at most 23-bits of precision at the output.

It should be noted that the x-axis indices in Fig. 2 and 3 do not necessarily match, as the execution paths can differ, due to the different nature of the two data-sets.

V. CONCLUSION AND FUTURE WORK

In this paper, we focus on providing a software tool and methodology to profile the mathematical library usage in a full scale application. The objective is to measure the potential and drive future *ad-hoc* optimizations of the math library.

Usually, elementary functions are implemented following a four step scheme [21]: special-case handling, argument reduction, reduced domain splitting and interpolation (*e.g.* polynomial or iterative).

When limiting the input domain, the first two steps can be optimized. Furthermore, reducing the required accuracy and input domain may lower the interpolation complexity. It can also diminish the implementation cost in special purpose architecture designs or re-programmable architectures (FPGA).

Rewriting *ad-hoc* custom elementary functions with a target accuracy on a given input interval is a costly and error prone task. We propose to explore existing tools to assist or automate these optimizations and measure empirical speedup on real use cases. An easy first step would be to automatically select the best fitting implementation among existing libraries, such as Intel MKL VML [8]. Finally, approaches such as *metalibm* [9] for math function code generation could be leveraged to produce specialized libraries.

To conclude, the work presented in this paper shows promising results on co-designing mathematical libraries from application profiles. One weakness of the approach is that the profile is data-input dependent; further experiments on a larger set of use-cases will be done to demonstrate the generalization of the approach and how one can deal with the speculative aspect of profile-guided optimization for math libraries.

REFERENCES

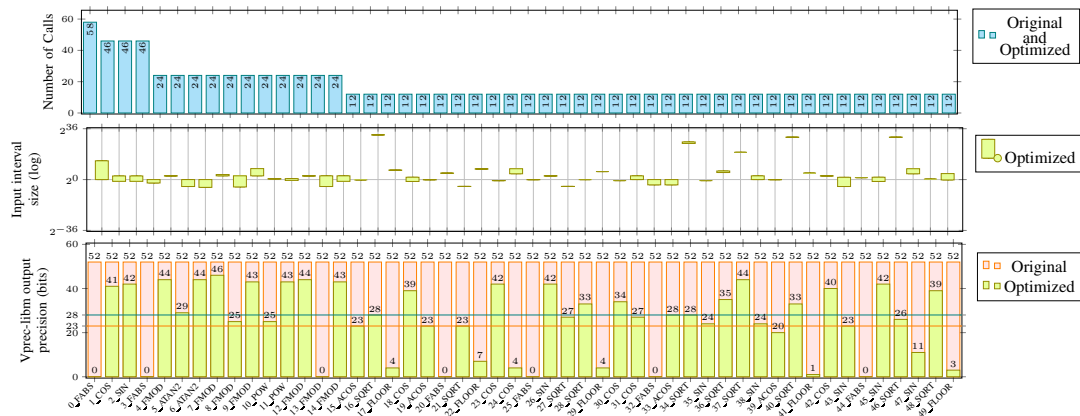
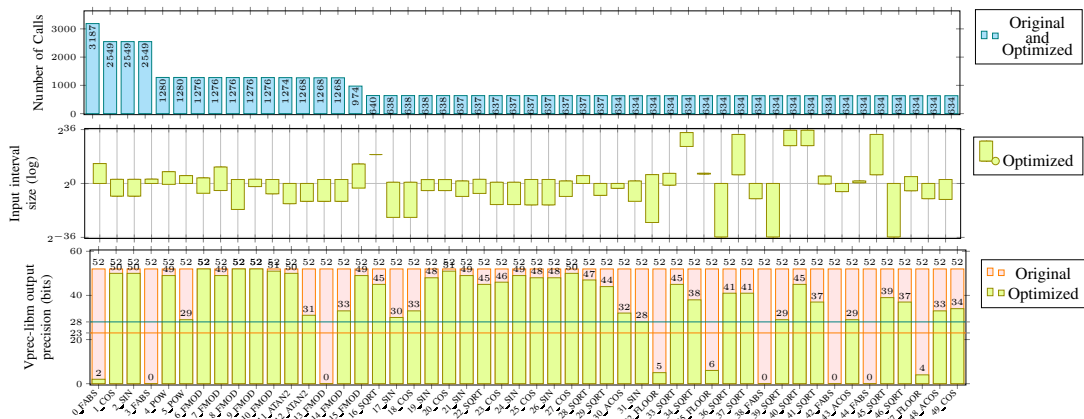
- [1] Intel, “White paper: Bfloat16 hardware numerics definition,” Tech. Rep., November 2018. [Online]. Available: <https://software.intel.com/sites/default/files/managed/40/8b/bf16-hardware-numeric-de>

⁵True Equator, Mean Equinox

²<https://celestrak.com/publications/AIAA/2006-6753/>

³https://en.wikipedia.org/wiki/Fortran#FORTRAN_IV

⁴https://en.wikipedia.org/wiki/Honeywell_6000_series



- [2] NVIDIA, "White paper: Training with mixed precision," Tech. Rep., November 2018. [Online]. Available: <https://docs.nvidia.com/deeplearning/sdk/pdf/Training-Mixed-Precision-Use-Guide.pdf>
- [3] Y. Chatelain, E. Petit, P. de Oliveira Castro, G. Lartigue, and D. Defour, "Automatic exploration of reduced floating-point representations in iterative methods," in *Euro-Par 2019 Parallel Processing - 25th International Conference*, ser. Lecture Notes in Computer Science. Springer, 2019.
- [4] A. Haidar, A. Abdelfattah, M. Zounon, P. Wu, S. Pranesht, S. Tomov, and J. Dongarra, "The design of fast and energy-efficient linear solvers: On the potential of half-precision arithmetic and iterative refinement techniques," in *International Conference on Computational Science*. Springer, 2018, pp. 586–600.
- [5] S. F. Oberman and M. J. Flynn, "Design issues in division and other floating-point operations," *IEEE Transactions on Computers*, vol. 46, no. 2, pp. 154–161, Feb 1997.
- [6] E. Brun, S. Chauveau, and F. Malvagi, "Patmos: A prototype Monte Carlo transport code to test high performance architectures," in *Proceedings of International Conference on Mathematics & Computational Methods Applied to Nuclear Science & Engineering*, Jeju, Korea, 2017.
- [7] D. Piparo, V. Innocente, and T. Hauth, "Speeding up HEP experiment software with a library of fast and auto-vectorisable mathematical functions," *Journal of Physics: Conference Series*, vol. 513, no. 5, 2014.
- [8] "Intel® math kernel library, developer reference," 2020. [Online]. Available: <https://software.intel.com/en-us/mkl-developer-reference-c>
- [9] N. Brunie, F. d. Dinechin, O. Kupriianova, and C. Lauter, "Code generators for mathematical functions," in *2015 IEEE 22nd Symposium on Computer Arithmetic*, June 2015, pp. 66–73.
- [10] C. M. Black, , and R. B. Thomas H. Miller, "The need for an industry standard of accuracy for elementary-function programs," *ACM Trans. Math. Softw.*, vol. 10, no. 4, pp. 361–366, Dec. 1984.
- [11] D. Defour, G. Hanrot, V. Lefèvre, J.-M. Muller, N. Revol, and P. Zimmermann, "Proposal for a standardization of mathematical function implementation in floating-point arithmetic," *Numerical Algorithms*, vol. 37, no. 1, pp. 367–375, 2004.
- [12] S. Collange, M. Daumas, and D. Defour, "Graphic processors to speed-up simulations for the design of high performance solar receptors," in *2007 IEEE International Conf. on Application-specific Systems, Architectures and Processors (ASAP)*, July 2007, pp. 377–382.
- [13] K. O. W. Group, "The OpenCL C specification version: 2.2," Web document., 2019. [Online]. Available: <https://www.khronos.org/registry/OpenCL/specs/opencl-2.0-opencl-c.pdf>
- [14] F. D. Dinechin and B. Pasca, "Floating-point exponential functions for DSP-enabled FPGAs," in *2010 International Conference on Field-Programmable Technology*, Dec 2010, pp. 110–117.
- [15] S. Collange, M. Daumas, D. Defour, and R. Olivès, "Fonctions élémentaires sur GPU exploitant la localité de valeurs," in *SYMposium en Architectures nouvelles de machines*, 2008, p. 12.
- [16] A. Suresh, "Intercepting functions for memoization," Theses, Université Rennes 1, May 2016.
- [17] F. R. Hoots and R. L. Roehrich, "Spacetrack Report No. 3: Models for propagation of NORAD element sets," Aerospace Defense Center, Peterson Air Force Base, Tech. Rep., 1980.
- [18] D. Vallado, P. Crawford, R. Hujsak, and T. Kelso, *Revisiting Spacetrack Report #3*. [Online]. Available: <https://arc.aiaa.org/doi/abs/10.2514/6.2006-6753>
- [19] F. R. Hoots, P. W. Schumacher, and R. A. Glover, "History of Analytical Orbit Modeling in the U. S. Space Surveillance System," *Journal of Guidance, Control, and Dynamics*, vol. 27, no. 2, pp. 174–185, 2004.
- [20] P. Markstein, "Accelerating sine and cosine evaluation with compiler assistance," in *2003 IEEE 16th Symposium on Computer Arithmetic*, June 2003, pp. 137–140.
- [21] C. Lauter and M. Mezzarobba, "Semi-automatic floating-point implementation of special functions," in *2015 IEEE 22nd Symposium on Computer Arithmetic*, June 2015, pp. 58–65.