Custom-Precision Mathematical Library Explorations for Code Profiling and Optimization

Matei Istoan*[1], Pablo De Oliveira Castro[2], David Defour , and Eric Petit

[1]Université de Versailles Saint-Quentin-en-Yvelines – Exascale ComputingResearch Lab – France
[2]Laboratoire dÍnformatique Parallélisme Réseaux Algorithmes Distribués – Université de Versailles Saint-Quentin-en-Yvelines – France

Résumé

The typical processors used for scientific computing have historically been designed with fixed-width data-paths.
Mathematical libraries (such as the GNU libm, Intel's MKL VML etc.) targeting these architectures were specifically developed and optimized for each of these predetermined floating-point formats, among which the more popular being binary16, binary32 or binary64.
However, to address the increasing performance and throughput needs, as well as the ever more limited energy consumption requirements of scientific applications, library and hardware designers should move beyond this one-size-fits-all approach, opting instead for libraries which are optimized for their applications' needs, in a given usage scenario.

In this presentation we set out to demonstrate how using custom user-defined floating-point formats and accuracies for computations involving the mathematical functions (sin, exp, log, erf ...) can be beneficial.
We show how our proposed tool explores the minimal required precision for each call-site to a mathematical function, for a given code.
This results in an optimal configuration of the floating-point formats and required accuracies for the mathematical function calls.
Producing this type of function-call profiles is a valuable asset when generating and optimizing mathematical function implementations, for the specific context of a user's application.
The advantages of our proposed tool are twofold.
First, we ensure a priori that the target application maintains its correctness, as per an user defined criteria, by generating a mathematical library which is guaranteed to satisfy the accuracy requirements.
Second, as demonstrated in the presentation, we allow for performance increase, by minimizing datapaths and reducing computational effort, where this effort would not bring any improvements in the program output.

We propose a three-steps tool that: analyses a given code, finds the minimal accuracies of the calls to the mathematical library and automatically makes the necessary changes to the code.
It is often unnecessary to compute with an accuracy equal to the maximum one allowed by a floating-point format.
Our goal is, then, to find a minimal configuration of the accuracies of the calls to the mathematical library.

*Intervenant

The precision gives an upper bound on the maximum accuracy that can be attained on the given format: finding the minimal accuracy will also give the optimal floating-point format.
Our proposed methodology consists of first creating a profile of the input dataset, to determine input data ranges and the frequency of calls to the different mathematical functions.
Based on this profile, we then proceed to a dichotomic exploration to find the minimal precisions and ranges for the outputs of the mathematical functions in a user application.
This process is done heuristically, in the decreasing order of the number of calls at each call-site path, as, according to Amdahl's law, optimizing the functions in which most of the time is spent will result in the biggest performance gains.
The minimization process is over once a user-defined output-accuracy criteria is unmet (e.g. relative or absolute errors increasing over a given threshold).
The last step is replacing the calls to the mathematical library with equivalent calls to a library that allows the control of the accuracy of the functions (e.g. Intel's MKL VML).

We demonstrate the tool's capabilities on two applications: SGP4, a widely-used satellite tracking application, and PATMOS, a Monte-Carlo neutron transport code for pin-by-pin full core depletion calculations for large nuclear power reactors.

In the case of SGP4, our experiments show that, depending on the input dataset, many of the calls to the mathematical library can have the input and output exponent size considerably reduced.
For tracking most of the satellites, the accuracy can be reduced to an important degree in around 30% of the calls, therefore allowing us to lower the precision of the floating-point formats, to the point of being able to use binary32 implementations instead of binary64.
As one would expect, using a more comprehensive dataset, that tests the algorithm's corner cases, the accuracy can be reduced to a lesser degree.
Only 15-20% of the calls can be converted from binary32 to binary64.

For PATMOS, our findings and conclusions are quite similar to those for SGP4, as many of the call-sites can benefit from reduced precision and range, while not influencing the final results.
We also measured the performance gains, in terms of reduction of the execution time by replacing the calls to the mathematical library with their equivalents from Intel's MKL VML, which trades accuracy for speed, while still maintaining the same floating-point format.

The control is quite coarse though, the user only being able to select between predefined accuracy targets: high accuracy, low accuracy or enhanced performance.
We found that by fine-tuning the accuracy at each calls-site, as per our profiling data, we can achieve a reduction of 5-10% of the execution time, as compared to using all calls in their default accuracy setting.
An alternative to using the Intel MKL VML is the Intel MKL SVML, which also allows the user to control the accuracy of the calls to its functions, though on an even coarser, compile-unit level.
Our first experiments with the Intel SVML show the potential for a 2x speedup, as compared to an implementation using the GNU libm, which encourages us to go deeper into that direction and allow SVML tuning per call-site.

The profiling data produced by the proposed methodology hints at real performance gains to be achieved through specialization, and pinpoints the location where it is useful to provide variable accuracy and precision designs for elementary function evaluation.
Our methodology has been implemented inside Verificarlo, a tool for debugging, assessing numerical stability and reproducibility, and optimizing floating point computations.
We provide an open-source implementation, to be released on the public github repository of Verificarlo.

The work is part of the Intel Exascale Research Lab (CEA-UVSQ-Intel).