CORE-MATH: progress report

Alexei Sibidanov and Paul Zimmermann

FPtalks 2023, 6th July

CORE-MATH Mission

CORE-MATH Mission: provide on-the-shelf open-source mathematical functions with **correct rounding** that can be integrated into current mathematical libraries (GNU libc, Intel Math Library, AMD Libm, Newlib, OpenLibm, Musl, Apple Libm, llvm-libc, CUDA libm, ROCm)

Cf `https://core-math.gitlabpages.inria.fr/`

CORE-MATH license: MIT to make integration easier.

Correct rounding

The best possible result for a given rounding mode.

Already required by IEEE 754 for basic arithmetic (addition, subtraction, multiplication, division, square root, fused multiply-add).

CORE-MATH secret goal: implement correct rounding for mathematical functions, as efficient (or even faster) than current mathematical libraries, so that these libraries integrate the CORE-MATH code, and users of these libraries get their computations correctly rounded (thus reproducible).

Current mathematical libraries

	binary32	binary64	binary80	binary128
GNU libc	√	√	√	√
Intel Math Library	√	√	√	√
AMD Libm	√	√		
Redhat Newlib	√	√		
OpenLibm	√	√	√	
Musl	√	√	√	
Apple libm	√	√		
LLVM libm	√	√		
Microsoft libm	√	√		
CUDA	√	√		
ROCm	√	√		

Note: only LLVM libm targets correct rounding (for now).

Largest Known Errors (binary32)

Excluding the gamma, lgamma and Bessel functions, and infinite values:

GNU libc 2.37	3.13 (erfc)
Intel Math Library 2023.0.0	0.550 (atan2)
AMD Libm 4.0	1.56 (tanh)
Redhat Newlib 4.3.0	169 (pow)
OpenLibm 0.8.1	3.17 (erfc)
Musl 1.2.3	3.88 (exp10)
Apple libm 12.1	0.862 (cos)
LLVM libm 15.0.7	0.776 (cos)
Microsoft libm 2022	6.66 (erfc)
CUDA 11.8.0	10.3 (pow)
ROCm 5.4.0	3.33 (erfc)

Source: `https://members.loria.fr/PZimmermann/papers/accuracy.pdf`

Largest Known Errors (binary64)

Excluding the gamma, lgamma and Bessel functions, and infinite values:

GNU libc 2.37	5.19 (erfc)
Intel Math Library 2023.0.0	1.73 (pow)
AMD Libm 4.0	2.79 (atan)
Redhat Newlib 4.3.0	4.08 (erfc)
OpenLibm 0.8.1	636 (pow)
Musl 1.2.3	4.14 (exp10)
Apple libm 12.1	10.7 (erfc)
LLVM libm 15.0.7	2.91e24 (tan)
Microsoft libm 2022	91.3 (pow)
CUDA 11.8.0	4.51 (erfc)
ROCm 5.4.0	4.08 (erfc)

Source: `https://members.loria.fr/PZimmermann/papers/accuracy.pdf`

Which functions?

Target the 39 functions defined in IEEE 754-2019:

- `exp`, `expm1`, `exp2`, `exp2m1`, `exp10`, `exp10m1`;
- `log`, `log2`, `log10`, `logp1`, `log2p1`, `log10p1`;
- `hypot`;
- `rSqrt`;
- `compound`;
- `rootn`, `pown`, `pow`, `powr`;
- `sin`, `cos`, `tan`, `sinPi`, `cosPi`, `tanPi`;
- `asin`, `acos`, `atan`, `asinPi`, `acosPi`, `atanPi`;
- `atan2`, `atan2Pi`;
- `sinh`, `cosh`, `tanh`;
- `asinh`, `acosh`, `atanh`.

Which target formats

- single precision (`binary32`);
- double precision (`binary64`);
- extended double precision (`long double` on x86_64);
- quadruple precision (`binary128`).

Year 2022 status

Reference: The CORE-MATH project, Alexei Sibidanov, Paul Zimmermann, Stéphane Glondu, Proceedings of ARITH 2022.

All C99 binary32 functions implemented: `acosf`, `acoshf`, `asinf`, `asinhf`, `atanf`, `atanhf`, `atan2f`, `cbrtf`, `cosf`, `coshf`, `erff`, `erfcf`, `expf`, `exp2f`, `exp10f`, `expm1f`, `hypotf`, `logf`, `log2f`, `log10f`, `log1pf`, `powf`, `sinf`, `sinhf`, `tanf`, `tanhf`

Three C99 binary64 functions implemented: `acos`, `cbrt` `exp`.

9 (out of 26) binary32 functions faster than GNU libc, Intel Math Library and LLVM libc.

Speed improvements

Reciprocal throughput (in AMD EPYC 7282 cycles) of some CORE-MATH routines:

function	CORE-MATH 2022	CORE-MATH 2023	GNU libc 2.36	Intel Math Library 2023.0.0
`acosf`	30	**17**	*30*	*25*
`asinf`	24	**17**	*28*	*25*
`asinhf`	25	**16**	*38*	*17*
`erfcf`	47	***25***	*55*	*42*
`exp`	32	18	***13***	*23*

Timings produced in our test environment (might be different in other conditions).

IML timings through docker image `intel/oneapi-hpckit`.

Year 2023 status

All C99 binary32 functions implemented (as in 2022) plus new C2X functions: `acospi`, `asinpi`, `atanpi`, `atan2pi`, `sinpi`, `cospi`, `tanpi`, `exp10m1`, `exp2m1`, `log10p1`, `log2p1`.

New C99/C2X binary64 functions: `acosh`, `asin`, `asinh`, `atan`, `atanh`, `cosh`, `cospi`, `erf`, `erfc`, `exp2`, `hypot`, `log`, `log10`, `log1p`, `log2`, `rsqrt`, `sinh`, `sinpi`, `tanh`, `tanpi`.

For the new C2X functions, CORE-MATH is the only library providing them so far (apart from GNU MPFR).

The binary64 power function

Work with Tom Hubrecht (ENS Paris) and Claude-Pierre Jeannerod (INRIA Lyon).

Complete code with full paper proof, soon to be integrated in CORE-MATH.

Technical details in Arith 2023 paper (soon online) and presentation.

Comparison of some implementations of the binary64 power function, in terms of i7-8700 cpu cycles:

	GNU libc 2.36	MathLib	CRLIBM	CORE-MATH
rec. throughput	*43*	123	211	66
latency	*79*	166	275	111

Some colleagues at Inria (Laurence Rideau and Laurent Théry) started a formal proof with the Coq formal proof engine.

The logarithm function

Guillaume Melquiond and Paul Geneau de Lamarlière (Inria Saclay) have formally proven a part of the argument reduction of the CORE-MATH binary64 logarithm.

Given $t \in [\sqrt{2}/2, \sqrt{2}]$, CORE-MATH approximates

$$\log t \approx P(z) + z + C_{i,1} + C_{i,2},$$

where $z = t \cdot r_i$ is close to 1 for some tabulated value r_i, and $C_{i,1} + C_{i,2}$ is a double-double approximation of $-\log r_i$.

They have formally proven:

$$|P(z) - (\log(1+z) - z)| \leq 2^{-68.72}$$

in three lines of Coq, using the tools Gappa and CoqInterval.

Work to be presented at Arith 2023.

How can I contribute?

- find a bug in the published functions
- submit a faster CR implementation
- find hard-to-round cases for binary64, binary80 or binary128
- make a formal proof of some implementation
- work on the integration in some mathematical library

References

`https://core-math.gitlabpages.inria.fr/`: main page

`https://gitlab.inria.fr/core-math/core-math/`: source code

`https://sympa.inria.fr/sympa/arc/core-math/`: mailing list