# LLVM libc math library
# Current status and future directions

Tue Ly (Google)

LLVM Developers’ Meeting 2024

# What is LLVM libc

- Greenfield implementation of C standard library (aka libc) under LLVM umbrella
- Standards: C23, POSIX
- Language(s): C++ with some inlined assembly
- Goals:
  - Complete C23 standard
  - One libc for all:
    - OS: Linux,  MacOS, Windows, Fuchsia, Android, ...
    - CPU: x86-64 / i386, ARM 64/32, RISC-V 64/32
    - Embedded systems (baremetal)
    - GPUs (nVidia / AMD)
- Homepage: https://libc.llvm.org

# LLVM libc math library (libm)

- (Re)implement all math C23 functions focusing on:
  - Accuracy
  - Performance
  - Portability
  - Configurability
- Floating point standard: IEEE 754-2019
- Homepage: https://libc.llvm.org/math

# How accurate?

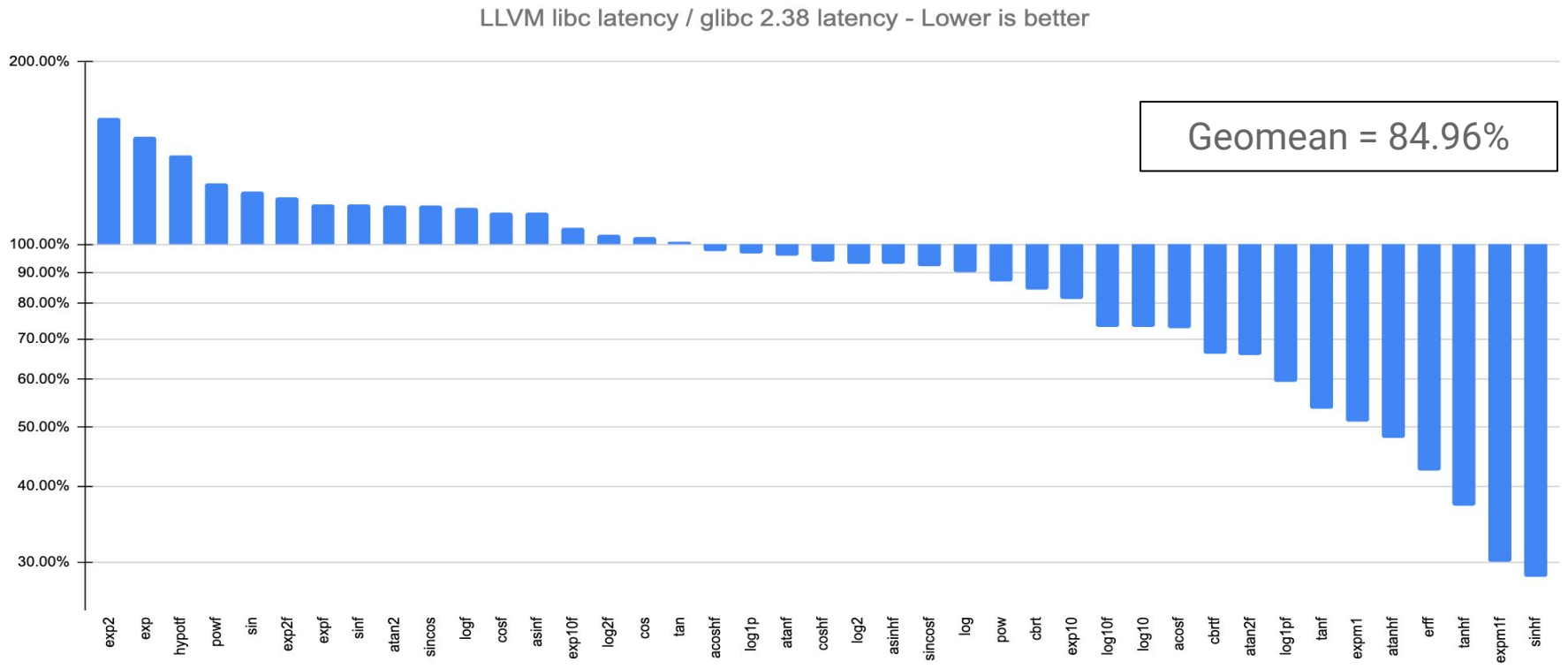
- As accurate as possible, with the ultimate goal to be correctly rounded for all rounding modes:
  - FE_TONEAREST, FE_UPWARD, FE_DOWNWARD, FE_TOWARDZERO,
    (to-be-added for ARM) round-to-nearest tie-away-from-zero
  - Rounding mode is decided by the floating point environment
- Correct rounding → Consistency:
  - Output bits are identical across platforms and versions
  - Reduce the toil of updating / integrating libc
  - Floating point data / model integrity to be stored / read / computed across environments

| library | GNU libc | IML | AMD | Newlib | OpenLibm | Musl | Apple | LLVM | MSVC | FreeBSD | ArmPL | CUDA | ROCm |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| version | 2.40 | 2024.0.2 | 4.2 | 4.4.0 | 0.8.3 | 1.2.5 | 14.5 | 18.1.8 | 2022 | 14.1 | 24.04 | 12.2.1 | 5.7.0 |
| acos | 0.899 | 0.528 | 0.897 | 0.899 | 0.918 | 0.918 | 0.634 | **0.500** | 0.669 | 0.918 | 1.32 | 1.34 | 1.47 |
| acosh | 2.01 | 0.501 | 0.504 | 2.01 | 2.01 | 2.01 | 0.502 | **0.500** | 2.89 | 2.01 | 2.79 | 2.18 | 0.564 |
| asin | 0.898 | 0.528 | 0.781 | 0.926 | 0.743 | 0.743 | 0.634 | **0.500** | 0.861 | 0.743 | 2.41 | 1.36 | 2.54 |
| asinh | 1.78 | 0.527 | 0.518 | 1.78 | 1.78 | 1.78 | 0.515 | **0.500** | 1.99 | 1.78 | 3.57 | 1.78 | 0.573 |
| atan | 0.853 | 0.541 | 0.501 | 0.853 | 0.853 | 0.853 | 0.722 | **0.500** | 0.501 | 0.853 | 2.88 | 1.21 | 2.10 |
| atanh | 1.73 | 0.507 | 0.547 | 1.73 | 1.73 | 1.73 | 0.511 | **0.500** | 2.35 | 1.73 | 3.09 | 3.16 | 0.574 |
| cbrt | 0.969 | 0.520 | 0.548 | 3.56 | **0.500** | **0.500** | **0.500** | | 1.83 | **0.500** | 1.53 | 1.17 | 1.14 |
| cos | 0.561 | 0.548 | 0.729 | 2.91 | 0.501 | 0.501 | 0.846 | **0.500** | 0.530 | 0.501 | 0.561 | 1.52 | 1.61 |
| cosh | 1.89 | 0.506 | 1.03 | 2.51 | 1.36 | 1.03 | 0.579 | **0.500** | **0.500** | 1.36 | 1.89 | 2.34 | 0.567 |
| erf | 0.968 | 0.780 | 0.531 | 0.968 | 0.943 | 0.968 | 0.501 | **0.500** | 3.99 | 0.890 | 1.93 | 1.04 | 1.51 |
| erfc | 3.13 | 0.934 | | 63.9 | 3.17 | 3.13 | **0.750** | | 6.66 | 3.18 | 1.64 | 4.49 | 3.33 |
| exp | 0.502 | 0.506 | 0.501 | 0.911 | 0.911 | 0.502 | 0.514 | **0.500** | 0.501 | 0.911 | 0.502 | 1.94 | 1.00 |
| exp10 | 0.502 | 0.507 | 0.501 | 1.06 | | 3.88 | 0.514 | **0.500** | | | | 2.07 | 1.00 |
| exp2 | 0.502 | 0.519 | 0.501 | 1.02 | 0.501 | 0.502 | 0.514 | **0.500** | 2.14 | 0.501 | 0.502 | 2.39 | 0.871 |
| expm1 | 0.813 | 0.544 | 0.536 | 0.813 | 0.813 | 0.813 | 0.687 | **0.500** | 3.02 | 0.813 | 1.51 | 1.45 | 1.45 |
| j0 | 9.00 | **0.678** | | 6.18e6 | 3.66e6 | 3.66e6 | | | | 3.66e6 | | 3.78e10 | 7.60e7 |
| j1 | 9.00 | **1.69** | | 1.68e7 | 2.25e6 | 2.25e6 | | | | 2.25e6 | | 7.48e9 | 7.53e7 |
| lgamma | 6.78 | 0.510 | | 7.50e6 | 7.50e6 | 7.50e6 | **0.501** | | 2.92e5 | 7.50e6 | | 1.35e7 | 7.50e6 |
| log | 0.818 | 0.519 | 0.577 | 0.888 | 0.888 | 0.818 | 0.511 | **0.500** | 0.562 | 0.888 | 0.818 | 0.865 | 1.89 |
| log10 | 2.07 | 0.516 | 1.40 | 2.10 | 0.832 | 0.832 | 0.502 | **0.500** | 0.626 | 0.832 | 0.82 | 2.09 | 1.71 |
| log1p | 1.30 | 0.525 | 0.501 | 1.30 | 0.839 | 0.835 | 0.513 | **0.500** | 1.44 | 0.839 | 2.02 | 0.887 | 0.579 |
| log2 | 0.752 | 0.508 | 0.766 | 1.65 | 0.865 | 0.752 | 0.502 | **0.500** | 2.04 | 0.865 | 0.752 | 0.919 | 1.00 |
| sin | 0.561 | 0.546 | 0.530 | 1.37 | 0.501 | 0.501 | 0.846 | **0.500** | 0.530 | 0.501 | 0.561 | 1.50 | 1.61 |
| sinh | 1.89 | 0.538 | **0.500** | 2.51 | 1.83 | 1.83 | 0.579 | **0.500** | 0.501 | 1.83 | 2.26 | 2.94 | 0.922 |
| sqrt | **0.500** | **0.500** | **0.500** | **0.500** | **0.500** | **0.500** | **0.500** | **0.500** | **0.500** | **0.500** | | **0.500** | **0.500** |
| tan | 1.48 | 0.520 | 0.509 | 3.48 | 0.800 | 0.800 | 0.746 | **0.500** | 0.502 | 0.800 | 3.30 | 3.10 | 2.33 |
| tanh | 2.19 | 0.514 | **0.500** | 2.19 | 2.19 | 2.19 | 0.817 | **0.500** | 1.27 | 2.19 | 2.59 | 1.82 | 1.41 |
| tgamma | 7.91 | 0.510 | | 239 | **0.501** | **0.501** | **0.501** | | 3.58e5 | **0.501** | | 4.34 | 1.68e7 |
| y0 | 8.98 | **3.40** | | 4.84e6 | 4.84e6 | 4.84e6 | | | | 4.84e6 | | 2.36e10 | 7.53e7 |
| y1 | 9.00 | **2.07** | | 6.18e6 | 4.17e6 | 3.66e6 | | | | 4.17e6 | | 4.96e10 | 9.35e7 |
| acospi | | **0.504** | | | | | | | | | | | |
| asinpi | | **0.506** | | | | | | | | | | | |
| atanpi | | **0.545** | | | | | | | | | | | |
| cospi | | **0.501** | | | | | | | | **0.501** | 2.65 | 0.966 | 0.966 |
| exp10m1 | **2.46** | | | | | | | | | | | | |
| exp2m1 | **1.66** | | | | | | | | | | | | |
| log10p1 | **2.04** | | | | | | | | | | | | |
| log2p1 | **1.88** | | | | | | | | | | | | |
| sinpi | | **0.501** | | | | | | | | 0.755 | 2.49 | 0.967 | 0.967 |
| tanpi | | 1.00 | | | | | | | | **0.800** | | | |
| rsqrt | | **0.500** | | | | | | | | | | 1.52 | 0.864 |
| atan2 | 1.52 | **0.550** | 0.584 | 1.52 | 1.55 | 1.55 | 0.722 | | 0.584 | 1.55 | 2.93 | 2.18 | 2.01 |
| atan2pi | | **0.841** | | | | | | | | | | | |
| compound | | **0.501** | | | | | | | | | | | |
| hypot | 0.501 | 0.501 | 0.501 | 1.21 | 1.21 | 0.927 | 0.501 | **0.500** | 0.501 | 1.21 | | 1.03 | 1.57 |
| pow | 0.817 | 0.515 | 1.56 | 169. | 0.970 | 0.817 | 0.515 | **0.500** | 0.568 | 0.970 | 0.817 | 2.60 | 1.40 |

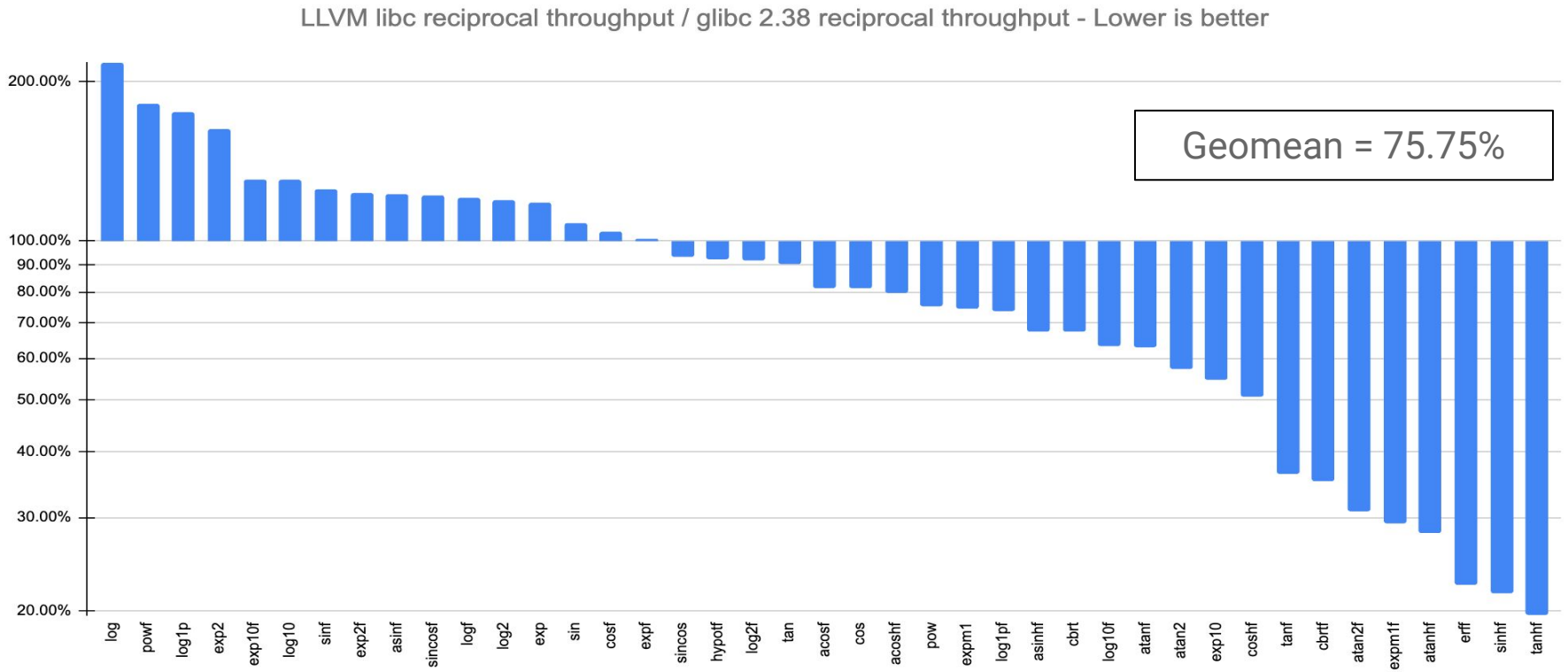
Table 1: Single precision: largest value of $e$ (for univariate functions), and largest *known* value of $e$ (for bivariate functions). Empty cells mean the corresponding function is not available.

Vincenzo Innocente, John Mather, and Paul Zimmermann, "*Accuracy of Mathematical Functions in Single, Double, Double Extended, and Quadruple Precision Brian Gladman*", (August 2024)
https://members.loria.fr/PZimmermann/papers/accuracy.pdf

# Accuracy at what cost? (aka performance)



# Performance - Reciprocal Throughput



# Portability

- Most functions are platform-agnostic
- Can be built on platforms without floating point unit or even without floating point support in the runtime libraries - We have all the pieces of a generic soft-float library:
  - Integer-based implementations of basic floating point operations
  - Templated BigInt class
- Platform-dependent errno / floating-point exception can be omitted
  → standalone libm library

# Configurability

- Provide support for various use-cases at build time
- Pick only functions, headers, config options that you need in config/<target>/<entrypoints/headers/config>.txt
  - LIBC_CONF_PATH
- Provide cmake config option LIBC_CONF_MATH_OPTIMIZATIONS with various options:
  - LIBC_MATH_NO_ERRNO
  - LIBC_MATH_NO_EXCEPT
  - LIBC_MATH_DEFAULT_ROUNDING_ONLY
  - LIBC_MATH_SMALL_TABLES
  - LIBC_MATH_SKIP_ACCURATE_PASS
    - Provable upper error bounds can be provided
  - LIBC_MATH_FAST = all of the above (aka, the “real” fast math)
- Anything else?

# Current progress toward complete C23(++)

- New floating point types
  - Half precision (_Float16)
  - Quad precision (_Float128??)
  - Fixed point(s)
- Completed all basic math functions for all 5 floating types (304) 🎉🎉🎉

| | |
|---|---|
| float | 57 |
| double | 62 |
| long double | 67 |
| float16 | 52 |
| float128 | 66 |

- https://libc.llvm.org/math/index.html#basic-operations

## New library features

### New headers

- <stdbit.h>
- <stdckdint.h>

### Library features

- Extended binary floating-point math functions
  - This section is incomplete
    Reason: List TBD
- Decimal floating-point math functions
  - -d*N* variants for existing and new floating-point math functions
  - quantized*N*()
  - samequantumd*N*()
  - quantumd*N*()
  - llquantexpd*N*()
  - encodedecd*N*()
  - decodedecd*N*()
  - encodebind*N*()
  - decodebind*N*()

https://en.cppreference.com/w/c/23

# Current progress toward complete C23(++)

- Implemented 76/254 higher math functions:
  - all but 3 are correctly rounded to all rounding modes (those 3 are 1-ULP)

| | |
|---|---|
| float | 32 / 50 |
| double | 20 / 51 |
| long double | 4 / 52 |
| float16 | 15 / 49 |
| float128 | 5 / 52 |

- https://libc.llvm.org/math/index.html#higher-math-functions

# New C23 type support - Half precision

- C23 type: _Float16
- Kicked off with Google Summer of Code (GSoC) 2024
  - https://blog.llvm.org/posts/2024-08-31-half-precision-in-llvm-libc/
  - Student tech talk tomorrow: https://llvm.swoogo.com/2024devmtg/session/2512743/student-technical-talks
- Have great performance potential with future hardware.
- _Float16 runtime + built-in shenanigan:
  - Which casts / builtins work on x86-64 / aarch64 / riscv64? clang vs gcc? compiler-rt vs libgcc?
  - https://gist.github.com/overmighty/a9a9de847eb11c667ba6b257375afe83

# New C23 type support - Quad precision

- C23 type: _Float128
- Long double replacement for platform consistency
  - long double = fp80 on x86 Linux
  - long double = fp64 on Windows, MacOS
  - long double = fp128 on ARM 64, RISC-V 64/32 Linux, Android (all CPUs)
  - long double = double-double on PowerPC
- Quad precision type definition shenanigan:
  - _Float128, __float128, or long double?
  - C or C++?
  - clang/clang++ vs gcc/g++
  - https://github.com/llvm/llvm-project/pull/78017

# New type support - Fixed point

- _Fract and _Accum types and the header <stdfix.h> are introduced in ISO/IEC TR 18037:2008, C extensions to support embedded processors.
- clang & LLVM libc are currently the only open source option supporting fixed point out of the box.
- Provide significant speed up and code size reduction compared to soft floats when applicable.
  - Replace soft floats in Pixel Bud
- Current support:
  - printf
  - abs*, exp*, round*, sqrt*, *sqrtu*
- https://libc.llvm.org/math/stdfix.html

# Near future

- Finish C23 higher math functions for (ordered by priority, highest to lowest):
  - float
  - double
  - float16 / float128
  - long double
- More fixed point math functions and fixed point configurations
- Complex support
  - Underway - https://libc.llvm.org/complex.html
- Expand platform supports:
  - Attain parity with x86-64 for i386, ARM 64/32, RISC-V 64/32, GPU nVidia/AMD, baremetal
  - More OSes: MacOS, Windows, Android, ...
- Expand LIBC_MATH_* config support to more functions

# Near + far future

- Further optimizations:
  - Performance
  - Code size
- Vector math library (libmvec)
- Decimal floating point support
- C++17 special math functions:
  - https://en.cppreference.com/w/cpp/numeric/special_functions
- C++26 constexpr math functions (hand-in-hand++)
- Shared implementations with compiler-rt / builtins and other runtime libraries (hand-in-hand-in-hand?)
- More (binary) floating point types?
  - bf16, _Float80, _Float256, …

# Thanks

- All past and present LLVM libc contributors
- Google Summer of Code
  - https://summerofcode.withgoogle.com/
- The CORE-MATH project
  - https://core-math.gitlabpages.inria.fr/
- The RLIBM project
  - https://people.cs.rutgers.edu/~sn349/rlibm/
- You, the audience