Accuracy of Mathematical Functions in Single, Double, Double Extended, and Quadruple Precision

Brian Gladman, Vincenzo Innocente[*], John Mather[†], Paul Zimmermann[‡]

August 2024

> *Computer users, most of whom assume they are working with reliable routines, unwittingly accept results from functions where the accuracies vary significantly from one mathematical library to another, from one library function to another, and even over different argument intervals of the same function. [...] Users are not likely to demand an improved situation because most of them, having neither the time nor the inclination to test manufacturer-supplied software, do not know the problem exists. This paper contains the results of such tests of elementary functions from several computer companies. The data [...] demonstrate that the industry does not satisfy the needs of those who require accurate and efficient mathematical software.*

These lines, written nearly four decades ago in 1984 by Black, Burton and Miller [6], are unfortunately still very true today.

The IEEE 754 standard, even in its latest 2019 revision [17], does not *require* correctly rounded mathematical functions, it only *recommends* them. In turn, current mathematical libraries do not provide correct rounding, which is the best possible result. Thus, users might get different results with different libraries, or different versions of the same library. This can have dramatic consequences: for example missed collisions in the Large Hadron Collider [5] or reproducibility issues in neuroimaging [13].

This document compares the accuracy of several mathematical libraries for the evaluation of mathematical functions, in single, double and quadruple precision (respectively `binary32`, `binary64`, and `binary128` in the IEEE 754 standard), and also in the extended double format. For single precision, an exhaustive search is possible for univariate functions, thus the given values are upper bounds. For larger precisions or bivariate functions, since an exhaustive search is not possible with academic resources, we use a black-box algorithm that tries to locate the values with the largest error; the given values are only lower bounds, but comparing them can give an idea of the relative accuracy of different libraries. An interesting fact is that, for several functions, different libraries yield the same largest known error, for the exact same input value, which probably means they use the same code base. Note that some libraries document the largest known errors[1] [8, 14]. Also, in

[*]CERN

[†]Side Effects Software Inc.

[‡]Université de Lorraine, CNRS, Inria, LORIA

[1]For the GNU libc, the values given in the "Known Maximum Errors in Math Functions" section are however not always accurate, since larger values can be found in GNU libc Bugzilla (and in this document).

[20], the authors have proven rigorous upper bounds for some double-precision functions (exp, log, sin, tan) of the Intel Math Library, in some domains (however the code might have changed since then).

Today, at least for single precision and most double precision functions, it is known how to get correct rounding (for all rounding modes, not only for rounding to nearest) at very low cost, and reference implementations exist that outperform current libraries [29, 16, 7].

1 Introduction

In this document we compare the accuracy of the following mathematical libraries: GNU libc 2.40 [15], the Intel Math Library shipped with the Intel oneAPI DPC++ Compiler 2024.0.2 (IML) [18], AMD LibM 4.2 [1], RedHat Newlib 4.4.0 [25], OpenLibm 0.8.3 [27], Musl 1.2.5 [24], the Apple Math Library 14.5 available under Darwin 23.5.0 [2], the LLVM libc 18.1.8 [21], the Microsoft library from Visual Studio 2022 (MSVC) [22], the mathematical library from FreeBSD 14.1 [12], libamath from the Arm Performance Libraries 24.04 [3], the CUDA mathematical library 12.2.1 [9], and the ROCm mathematical library 5.7.0 [26]. We only consider rounding to nearest-even; with other rounding modes one might get disastrous results with some libraries [19], as reported already in 2002 by Vincent Lefèvre.[2] We do not compare to the x87 instructions `fsin` and others, which are known to have bad accuracy [10].

For each function, assuming y is the value returned by the library, and z is the exact result (as with infinite precision), we denote by e the absolute difference between y and z in terms of units-in-last-place of z. The value z is approximated with the GNU MPFR library [11], using a larger precision. Our definition of ulp (unit-in-last-place) is the following: for $2^{e-1} \leq |x| < 2^e$, and precision p, we define $\mathrm{ulp}(x) = 2^{e-p}$. i.e., the distance between two consecutive p-bit floating-point numbers in the binade $[2^{e-1}, 2^e)$, see [23].

The results for GNU libc, AMD LibM, Newlib, OpenLibm, and Musl were obtained on an Intel Core i5-4590 with GCC 13.3.0 under Debian. Those for LLVM libc were obtained on an Intel(R) Xeon(R) Silver 4214; an AMD EPYC2 7352 was used for the Intel Math Library (IML in short), with the Intel compiler version 2024.0.2, using `-fp-model=strict` (including `mathimf.h` instead of `math.h`). Those for the Apple math library were obtained on a Mac M1 (arm64) under Darwin 23.5.0, with clang 15.0.0. The results with the Microsoft library were obtained using the Universal C Runtime Library (UCRT) distributed with Windows SDK version 10.0.22621 for Windows 11, and the code was run on a Dell Precision 5540 with an Intel Core i9 9880H running Windows 11 and on a Dell PowerEdge R7525 with two AMD EPYC2 7352 running Windows Server 2019. The FreeBSD results were obtained with a libvirt image running on a Intel Xeon Silver 4214. The results for libamath from the Arm Performance Libraries were obtained on an AWS Graviton3 c7g.metal instance running Ubuntu 22.04. The CUDA library was tested on a NVidia Grace-Hopper system running CUDA 12.2.1. The tests were also run on a GTX1060 GPU, hosted on an AMD Ryzen 7 1800X, obtaining identical results. The ROCm library was tested on an AMD Radeon Instinct MI50 running ROCm 5.7.0 hosted on a Intel Xeon Silver 4114. Tests were also run on a "Radeon Pro WX 9100", hosted on a AMD Ryzen 9 5900X, obtaining identical results.

Newlib was configured with default flags (in particular, without use of hardware FMA), and with the default configuration.[3]

[2] `https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=153022`

[3] For binary32, by default, the old SunPro functions are used; with `OBSOLETE_MATH_DEFAULT=0`, Newlib will use

In all tables, values of e are given with 3 decimal digits, rounded up; thus for example $e = 2.17$ for a univariate single-precision function means that the relative error is bounded by 2.17 ulps for all `binary32` inputs, and in all other cases (larger formats or bivariate functions) it means the largest *known* error is bounded by 2.17 ulps, with at least one case giving an error of more than 2.16 ulps. Note that correct rounding implies an entry 0.5, but the converse is false: when the "round-to-even" rule applies, if the library returns the "odd" value, the error will still be 0.5 ulp.

It should be noted that one might get different results for a given library on different hardware, for at least two reasons. Firstly some libraries have a runtime dispatcher which invokes different source code for different cpus, for example using the fused multiply-add, or extensions SSE, AVX, AVX2 or AVX512. This is the case of the GNU libc and of the Intel Math library. Secondly the very same binary might produce different results on different hardware due to the use of some assembly instructions that are implemented differently. This is for example the case with the `rsqrt` and `rcp` instructions that differ on Intel and AMD hardware, see §3.2.

2 Single Precision

2.1 Univariate Functions

The IEEE 754 single-precision format (`binary32`) has $2^{32} - 2^{24}$ values, not counting +Inf, -Inf, and NaN. For a function with a single input, it is thus possible to check all values by exhaustive search.

Table 1 summarizes the largest known ulp error for each function and each library. For univariate functions, the corresponding input can easily be found by exhaustive search, while Table 2 gives the corresponding inputs for bivariate functions.

In all tables, empty cells mean the corresponding function is not available.

We see that for all libraries, the sqrt function is correctly rounded for all binary32 inputs, as required by IEEE 754.[4] The following functions are correctly rounded: `rsqrtf` in the Intel Math Library, `cbrtf` in OpenLibm, Musl, the Apple library and FreeBSD, `coshf` in MSVC, `sinhf` and `tanhf` in AMD LibM (which uses the CORE-MATH implementation [29]), and all functions available in the LLVM library.

We get large errors for `j0f`, `j1f`, `y0f`, `y1f`, `lgammaf` and `tgammaf` for all libraries where these functions are available, except for GNU libc, IML, and the Apple library.

2.2 Bivariate Functions

For single precision bivariate functions, it is not possible to perform an exhaustive search with academic resources, since there are up to 2^{64} possible pairs of inputs. For example, for the power function x^y, there are about 2^{61} input pairs $x, y > 0$ that do not yield underflow nor overflow. We thus used the algorithm described in §3.1 to obtain the values of Table 2, which are *lower bounds* for the largest error.

Notes about GNU libc. GNU libc 2.40 implements new C23 functions: `exp10m1`, `exp2m1`, `log2p1`, and `log10p1`, for all formats. However, these functions have dummy implementations: for

instead a new set of mathematical functions provided by Arm, that use binary64 for intermediate computations.

[4] As noticed by Hugues de Lassus, correct rounding implies a maximal error of 0.5 ulp, but the converse is not necessarily true. However, we also checked the results agree with GNU MPFR.

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.

	GNU libc 2.40			IML 2024.0.2		
	x	y	max e	x	y	max e
atan2	-0x1.f9cf48p+49	0x1.f60598p+51	1.52	-0x1.58a7ecp-118	0x1.58a7bep-123	0.550
atan2pi				-0x1.461adep+23	-0x1.74fbc8p+22	0.841
compound				0x1.46674cp+50	0x1.d5738ap-30	0.501
hypot	-0x1.003222p-20	-0x1.6a2d58p-32	0.501	-0x1.003222p-20	-0x1.6a2d58p-32	0.501
pow	0x1.025736p+0	0x1.309f94p+13	0.817	0x1.fe7782p-1	-0x1.c361cap+14	0.515

	AMD LibM 4.2			RedHat Newlib 4.4.0		
	x	y	max e	x	y	max e
atan2	0x1.fffe24p+59	0x1.000adcp+73	0.584	-0x1.f9cf48p+49	0x1.f60598p+51	1.52
hypot	-0x1.0554acp+44	-0x1.6dc9e6p+32	0.501	-0x1.6b05c4p-127	0x1.6b3146p-126	1.21
pow	0x1.10fff4p+0	0x1.58fd76p+10	1.56	0x1.d55902p-1	-0x1.fe037ep+9	169.

	OpenLibm 0.8.3			Musl 1.2.5		
	x	y	max e	x	y	max e
atan2	0x1.a10104p+123	0x1.99f182p+125	1.55	0x1.a10104p+123	0x1.99f182p+125	1.55
hypot	-0x1.6b05c4p-127	0x1.6b3146p-126	1.21	0x1.26b188p-127	-0x1.a4f2fp-128	0.927
pow	0x1.343e4ep+0	0x1.af3c4p+8	0.970	1.025736p+0	1.309f94p+13	0.817

	Apple 14.5			LLVM 18.1.8		
	x	y	max e	x	y	max e
atan2	-0x1.ce62cep-116	0x1.cbf9bp-113	0.722			
hypot	-0x1.003222p-20	-0x1.6a2d58p-32	0.501	0x1.5804ccp-40	-0x1.a3bp-52	0.500
pow	0x1.034016p+0	0x1.b782b4p+12	0.515	-0x1.8p-49	0x1.8p+1	0.500

	MSVC 2022			FreeBSD 14.1		
	x	y	max e	x	y	max e
atan2	0x1.fffe24p+59	0x1.000adcp+73	0.584	0x1.a10104p+123	0x1.99f182p+125	1.55
hypot	-0x1.003222p-20	-0x1.6a2d58p-32	0.501	-0x1.6b05c4p-127	0x1.6b3146p-126	1.21
pow	0x1.107fe4p+0	0x1.631e6cp+10	0.568	0x1.343e4ep+0	0x1.af3c4p+8	0.970

	ArmPL 24.04		
	x	y	max e
atan2	-0x1.9be82ap-101	0x1.92cfb2p-101	2.93
pow	0x1.025736p+0	0x1.309f94p+13	0.817

	CUDA 12.2.1			ROCm 5.7.0		
	x	y	max e	x	y	max e
atan2	0x1.0e5beap+6	0x1.016188p+6	2.18	0x1.5c8fdep+25	0x1.cbe722p+24	2.01
hypot	0x1.007594p+1	-0x1.003512p+1	1.03	-0x1.ad2d0ap+111	-0x1.7f456ap+118	1.57
pow	0x1.6794b6p+0	0x1.f9f1bap+7	2.60	0x1.25f64ep-5	0x1.a47bc2p+4	1.40

Table 2: Single precision bivariate functions.

example `exp10m1` calls `exp10` on its input and subtracts 1. As a consequence, the observed largest errors are of a few ulps.

Notes about the Intel Math Library. The `rsqrt` function, which is not yet standardized in C99, is called `invsqrt` in the Intel Math Library. In single precision, it is correctly rounded for all rounding modes.

Notes about AMD LibM. All issues reported previously have been fixed in AMD LibM 4.2.

Notes about Newlib. For negative integers, Newlib `tgammaf` returns an infinite value, whereas other libraries return NaN. We use the `lgammaf_r` function, since we were unable to compile the `lgammaf` function (`undefined reference to '_impure_ptr'`).

Notes about the Apple Math Library. The `erff`, `lgammaf` and `tgammaf` functions seem to call the corresponding double function, which explains the very good accuracy and the very small number of incorrectly-rounded results. The single precision `exp10` function is available as `__exp10f`.

Notes about LLVM-libc. Although the maximum observed distance is 0.5 ulp for `powl`, some inputs are not correctly rounded in the subnormal range, for example for x = `-0x1.8p-49` and y = `0x1.8p+1`, LLVM 18.1.8 yields `-0x1.ap-146` instead of `-0x1.cp-146`.

Notes about the Microsoft mathematical library. The Bessel functions are not available in single precision (they are only available in double precision).

Notes about FreeBSD. We use the generic x86_64 binary from the FreeBSD 14.1 distribution. Some slighly different results might be obtained if you recompile FreeBSD using fused-multiply-add.

3 Double Precision

For double precision it is not possible to perform an exhaustive search with academic resources. We thus use a black-box algorithm—described in §3.1—that tries to find large errors. Therefore, the values in the double-precision tables are only lower bounds of the largest error.
Remark: We did not want to analyze the code of each library, since this approach would need more human work, and would require to start again from scratch for each new version of the library. We did not want either to design code specific to each function: for example for the double precision sine or cosine, one could test inputs near 2^{1024}, to check the argument reduction is correctly done. We expect our search algorithm automatically detects such issues.

3.1 Search Algorithm

The idea of the algorithm is to subdivide recursively the set of values to search for. We describe it for a univariate double precision function, but it works for any IEEE format, as long as there is a corresponding integer type with the same bit-width, and it also works for bivariate functions.

Assume $f(x)$ is a univariate double precision function. The number of possible inputs of f is less than 2^{64}, thus each one can be mapped to a 64-bit integer. Assume we have a conversion

function `to_double` from `uint64_t` to `double`. The algorithm takes as input a range $[a, b]$ of `uint64_t` values, and a threshold t. If $b - a < t$, it checks exhaustively all double precision values `x=to_double(i)` for $a \le i < b$. This means for each x, we compute the ulp-error e between the value $y \approx f(x)$ returned by the corresponding library, and the exact result z (as with infinite precision), as described in §1, and record the largest error.

If $b - a \ge t$, we subdivide the interval $[a, b]$ into two (almost) equal intervals, in each one we generate t random values and compute the corresponding errors. We then recurse in the interval where we found the largest error.

For example with $t = 10^6$, the initial interval has 2^{64} values, thus we compute $f(x)$ on $2t$ random inputs x (t in each sub-range of 2^{63} values), and so on... The recursion stops when the width of the current interval is less than the total number of evaluations done so far in the recursion tree, thus an exhaustive search will at most double the search time.

In practice we use a variant of this algorithm suggested by Eric Schneider: instead of recursing only in the sub-interval giving the largest error on the random sample, we keep at each level of the search tree a list of say 20 intervals with the largest sample errors. Then we subdivide each of these intervals, which yields 40 smaller intervals (or 80 for bivariate functions), and keep again the 20 most promising ones.

We tried three variants of this algorithm, depending on how we choose the "best" sub-interval. The first strategy—described above—keeps the sub-interval with the largest ulp-error. A second strategy keeps the sub-interval with the maximal *average* ulp-error (considering only inputs which yield a meaningful ulp-error, discarding those giving NaN, zero or $\pm\infty$). A third strategy keeps the sub-interval with the largest expected ulp-error; for this, we estimate the mean and standard deviation of the ulp-error on each sub-interval, from which we deduce an estimate of the largest ulp-error for the number of points in the sub-interval [28]. In practice we found the first strategy to be more effective, with the second and third strategies finding sometimes larger errors. Thus when the search program is run on a machine with n cores, we assign one core to the second and third strategies, and $n - 2$ cores to the first one.

The program also keeps track of the worst cases found for each library, and tries these input values for the other libraries. This helps determining the libraries using the same code base. The search programs (`check_sample.c` for univariate functions, and `check_sample2.c` for bivariate functions), the exhaustive search program for `binary32` univariate functions (`check_exhaustive.c`) and the source code of this article (containing in comment the x-values yielding the largest errors for `binary32`) are available from `https://gitlab.inria.fr/zimmerma/math_accuracy`.

We have also used the worst cases found by Vincent Lefèvre, publicly available at `https://www.vinc17.net/research/testlibm/`.

3.2 Results

We used a threshold of at least $t = 10^6$ for all libraries, often on processors with at least 32 cores, and the search program was run multiple times, cycling over all libraries, to detect common large errors.

Table 3 summarizes the largest known errors found using the above algorithm, for example the 0.531 entry for `acos` and IML means that for all inputs tried by the above algorithm, the error for the arc-cosine function with the Intel Math Library was bounded by 0.531 ulp. On each line, bold-face entries correspond to the best largest known error. Detailed tables (Tables 4-5, 6, 7, 8,

9, 10 and 11) give the input values (in hexadecimal) yielding the corresponding ulp-error e, which enables the reader to reproduce our results.

In double precision, the Intel Math Library gives the best results in most cases. However, it was observed that the Intel Math Library gives better results on AMD hardware than on Intel hardware for `acosh`, `asin`, `asinh` and `atan2`; a possible explanation is that these functions use the `rsqrt` instruction, which is known to be more accurate on AMD hardware [4]. The square root function seems to be correctly rounded for all libraries, as required by IEEE 754. All LLVM functions except `cos`, `sin`, `tan` seem to be correctly rounded. Large errors occur for the AMD `cbrt` function, for the `j0`, `j1`, `y0` and `y1` functions for all libraries that provide them except the Intel Math Library, for the `lgamma` function from all libraries but the GNU and Intel libraries, for the `tgamma` function from Newlib, OpenLibm, the Apple library, MSVC and FreeBSD, for the power function from OpenLibm, MSVC and FreeBSD, for the `cos`, `sin`, and `tan` functions from LLVM libc.

Notes about AMD LibM. An issue noticed in 4.1 (maybe it was there earlier) is still in 4.2: for subnormal numbers, `cbrt` gives huge errors, for example for x = `-0x0.01fffffffffffp-1022`, it yields `-0x1.9d23e0bb9777ap-323` instead of the expected value `-0x1.fffffffffffabp-344`.

Notes about Newlib. For negative integers, Newlib `tgamma` returns an infinite value, whereas other libraries return NaN. The C standard says that a domain error or pole error may occur for a negative integer, but due to some non-standard dealing of Newlib with errno, we were unable to check whether it is the case. We however excluded negative integers for Newlib `tgamma` in our tests.

Notes about LLVM-libc. For x = `-0x1.13a5ccd87c9bbp+1008`, the `cos` and `sin` functions return x instead of `0x1.a1fa1068d0b59p-1` and `-0x1.27b3964185d8dp-1` respectively, and `tan` yields `NaN` instead of `-0x1.6a3815320e5cfp-1`.

Notes about the Microsoft mathematical library. The `_y1` function yields NaN for x = +0 instead of -Inf, which explains the Inf value in Table 3.

3.3 Other Functions

For other functions from the C standard like `ldexp`, correct rounding might at first glance seem easier to implement. However, as reported by Fred Tydeman, cutting the smallest subnormal number in half does not always return zero. We noticed that for rounding upwards, for x = `0x1p-1074`, both Newlib 4.4.0 (already reported for 4.3.0) and LLVM 18.1.8 (already reported for 16.0.6) return 0 for `ldexp(x,-1)` instead of x. For both libraries, there is a similar issue for `ldexpf` and `ldexpl`.

4 Double Extended Precision

This format corresponds to the C type `long double` on `x86_64` processors. Some libraries do not provide double extended precision, or do not provide mathematical functions for this format. The results are summarized in Table 12, and detailed in Tables 13-15. We see that in this format, the

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	**0.523**	0.531	1.36	0.930	0.930	0.930	1.06		0.934	0.930	1.52	1.53	0.772
acosh	2.25	**0.509**	1.32	2.25	2.25	2.25	2.25		3.22	2.25	2.66	2.52	0.661
asin	**0.516**	0.531	1.06	0.981	0.981	0.981	0.709		1.05	0.981	2.69	1.99	0.710
asinh	1.92	**0.507**	1.65	1.92	1.92	1.92	1.58		2.05	1.92	2.04	2.57	0.661
atan	**0.523**	0.528	0.863	0.861	0.861	0.861	0.876		0.863	0.861	2.24	1.77	1.73
atanh	1.78	**0.507**	1.04	1.81	1.81	1.80	2.01		2.50	1.81	3.00	2.50	0.664
cbrt	3.67	0.523	1.53e22	0.670	0.668	0.668	0.729		1.86	0.668	1.79	**0.501**	**0.501**
cos	**0.516**	0.518	0.919	0.887	0.834	0.834	0.948	Inf	0.897	0.834		1.52	0.797
cosh	1.93	**0.516**	1.85	2.67	1.47	1.04	0.523		1.91	1.47	1.93	1.40	0.563
erf	1.43	**0.773**	1.00	1.02	1.02	1.02	6.41		4.62	1.02	2.29	1.50	1.12
erfc	5.19	**0.826**		4.08	4.08	3.72	10.7		8.46	4.08	1.71	4.51	4.08
exp	0.511	0.530	1.01	0.949	0.949	0.511	0.521	**0.500**	1.50	0.949	0.511	0.928	0.929
exp10	0.513	0.538	1.02	0.896		4.14	0.521	**0.500**			0.510	1.11	1.11
exp2	0.511	0.535	1.05	0.896	0.751	0.511	0.521	**0.500**	2.23	0.751	0.509	0.948	0.947
expm1	0.911	0.512	0.670	0.909	0.909	0.909	0.706	**0.500**	3.06	0.909	2.18	1.18	1.91
j0	4.51e14	**0.600**		9.01e15	4.51e14	4.51e14	3.83e14		1.88e26	4.51e14		3.08e20	1.25e13
j1	4.47e14	**0.615**		9.01e15	1.10e15	1.10e15	1.10e15		3.85e26	1.10e15		1.73e21	4.80e13
lgamma	11.1	**0.515**		4.45e15	4.45e15	4.45e15	2.33e16		5.10e13	4.45e15		5.11e15	4.45e15
log	0.520	0.518	0.610	0.946	0.946	0.520	0.508	**0.500**	0.577	0.946	0.520	0.564	0.663
log10	1.62	0.532	1.09	2.08	0.814	0.814	0.514	**0.500**	0.633	0.814	1.62	1.43	0.785
log1p	0.899	0.521	0.636	0.896	0.896	0.900	0.667	**0.500**	1.44	0.896	1.74	1.50	1.00
log2	0.548	0.505	1.72	2.06	0.921	0.555	0.515	**0.500**	0.812	0.921	0.554	1.31	0.734
sin	**0.516**	0.518	0.895	0.888	0.831	0.831	0.944	Inf	0.799	0.831		1.52	0.800
sinh	1.93	**0.521**	1.49	2.67	1.88	1.88	0.539		1.51	1.88	2.59	1.51	0.868
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	0.619	**0.550**	1.38	1.02	1.02	1.02	3.53	Inf	1.32	1.02		2.09	1.30
tanh	2.21	**0.556**	1.40	2.22	2.22	2.22	0.613		1.44	2.22	2.76	1.48	0.866
tgamma	8.68	**0.519**		2.27e3	1.03e3	16.0	1.03e3		9.01e15	1.03e3		10.1	13.7
y0	5.93e15	**1.14**		1.42e15	1.42e15	1.42e15	1.42e15		3.30e25	1.42e15		1.18e21	1.95e13
y1	5.56e15	**1.25**		5.56e15	5.56e15	5.56e15	5.56e15		Inf	5.56e15		1.17e21	6.14e13
acospi		**0.541**											
asinpi		**0.521**											
atanpi		**0.936**											
cospi		**0.503**								0.812	3.18	1.52	0.894
exp10m1	**3.54**												
exp2m1	**1.93**												
log10p1	**1.95**												
log2p1	**1.88**												
sinpi		**0.532**								0.811	3.16	1.52	0.890
tanpi		1.00								**0.969**			
rsqrt		**0.501**										0.510	1.00
atan2	**0.524**	0.548	1.51	1.55	1.55	1.55	0.747		0.750	1.55	2.23	1.76	1.82
atan2pi		**1.02**											
compound		**0.519**											
hypot	0.792	0.751	1.03	1.21	1.21	1.04	1.21	**0.500**	1.22	1.21		1.89	1.21
pow	**0.523**	1.73	0.762	0.892	636.	0.525	0.757		91.3	636.	**0.523**	1.84	1.40

Table 3: Double precision: Largest known error.

	GNU libc 2.40		IML 2024.0.2	
function	x	max e	x	max e
acos	0x1.dffffb3488a4p-1	0.523	0x1.6c05eb219ec46p-1	0.531
acosh	0x1.0001ff6afc4bap+0	2.25	0x1.01825ca7da7e5p+0	0.509
asin	-0x1.0000045b2c904p-3	0.516	0x1.6c042a6378102p-1	0.531
asinh	-0x1.02657ff36d5f3p-2	1.92	0x1.0003e70f36703p-4	0.507
atan	0x1.f9004c4fef9eap-4	0.523	-0x1.ffff8020d3d1dp-7	0.528
atanh	-0x1.ebb5133a9d9a4p-4	1.78	-0x1.e2cfb2667f17ep-9	0.507
cbrt	0x1.7a337e1ba1ec2p-257	3.67	-0x1.f7af4893d1d51p-616	0.523
cos	-0x1.7120161c92674p+0	0.516	-0x1.d19ebc5567dcdp+311	0.518
cosh	-0x1.633c654fee2bap+9	1.93	-0x1.5a364e6b98134p+9	0.516
erf	0x1.c332bde7ca515p-5	1.43	0x1.c5bba21264fa9p-9	0.773
erfc	0x1.3ff2d63705b29p+0	5.19	0x1.a8cf6bca23f9cp+4	0.826
exp	-0x1.49f33ad2c1c58p+9	0.511	0x1.fce66609f7428p+5	0.530
exp10	-0x1.57449153f316ep-7	0.513	-0x1.5cd9d94d49a85p+1	0.538
exp2	-0x1.1a4ce073ea908p-5	0.511	0x1.f3ffd85f33423p-1	0.535
expm1	0x1.63be411e096ep-2	0.911	-0x1.62fe464c64f65p-8	0.512
j0	0x1.33d152e971b4p+1	4.51e14	0x1.aff859518c846p+7	0.600
j1	-0x1.ea75575af6f09p+1	4.47e14	-0x1.67b5541c7d8b7p+7	0.615
lgamma	-0x1.f613ab0969f81p+1	11.1	-0x1.3f62c60e23b31p+2	0.515
log	0x1.1211bef8f68e9p+0	0.520	0x1.008000db2e8bep+0	0.518
log10	0x1.de02157073b31p-1	1.62	0x1.feda7b62c1033p-1	0.532
log1p	-0x1.2bf183e0344b2p-2	0.899	0x1.000aee2a2757fp-9	0.521
log2	0x1.1406d79e1b574p+0	0.548	0x1.fe01ab6b835b8p-1	0.505
sin	-0x1.f8b791cafcdefp+4	0.516	-0x1.0e16eb809a35dp+944	0.518
sinh	-0x1.633c654fee2bap+9	1.93	-0x1.adc135eb544c1p-2	0.521
sqrt	0x1.fffffffffffffp-1	0.500	0x1.fffffffffffffp-1	0.500
tan	-0x1.317cd745dd37cp+9	0.619	0x1.49adfd996a81dp+18	0.550
tanh	0x1.e0d4673daf76bp-3	2.21	0x1.002629fd74484p+0	0.556
tgamma	-0x1.c18caecc00f7bp+2	8.68	-0x1.3e0001ad3bee3p+6	0.519
y0	0x1.c982eb8d417eap-1	5.93e15	0x1.4cdee58a47eddp-31	1.14
y1	0x1.193bed4dff243p+1	5.56e15	0x1.c513c569fe78ep+0	1.25
acospi			0x1.6a18f7dda5343p-1	0.541
asinpi			0x1.921fabd2a2b25p-750	0.521
atanpi			0x1.a6ba44a8e47acp-2	0.936
cospi			0x1.f09672f5535bbp-8	0.503
exp10m1	0x1.e880c5bafbd41p-2	3.54		
exp2m1	0x1.d047583a6c6dp-1	1.93		
log10p1	-0x1.4c2971893052fp-1	1.95		
log2p1	0x1.a7b725780ff2cp-2	1.88		
sinpi			0x0.07f16ec91c164p-1022	0.532
tanpi			0x1.49b79692667bp+46	1.00
rsqrt			0x1.00018f5913816p-458	0.501

Table 4: Double precision: GNU libc and Intel Math Library (part 1).

function	GNU libc 2.40 x	max e	IML 2024.0.2 x	max e
atan2	0x1.ed6060626eecfp-429 0x1.f42ebb62994dcp-426	0.524	0x1.b77ade79a36d5p-326 0x1.ff6a37b72b52bp-319	0.548
atan2pi			-0x1.026462f302171p-391 0x1.39b157b1210a4p-390	1.02
compound			0x1.63d62cb3d61b8p-118 -0x1.bdc05a687ad2bp+112	0.519
hypot	0x0.603e52daf0bfdp-1022 -0x0.a622d0a9a433bp-1022	0.792	0x0.19deaac345ffap-1022 0x0.92c8727c389b6p-1022	0.751
pow	0x1.010e2e7ee71aep+0 0x1.44bf0047427f6p+17	0.523	0x1.fffff9c61ce4p-1 0x1.c4e304ed4c734p+31	1.73

Table 5: Double precision: GNU libc and Intel Math Library (part 2).

Intel Math library is better than all other libraries for all functions it provides, both univariate and bivariate, except for the `tgamma` and `hypot` functions.

Notes about GNU libc. We noticed that for extended double precision, the GNU libc can give different results on x86_64 hardware, whether it is Intel or AMD hardware. For example for x = `0xb.1722e675ab2f4c1p-5`, `expm1l` yields `0xd.413ec8a7dda564ap-5` on AMD cpu, and `0xd.413ec8a7dda5648p-5` on Intel cpu.

Notes about the Intel Math Library. The `j0l`, `j1l`, `y0l`, and `y1l` functions call the corresponding quadruple precision function, which explains why the largest error is near 0.5 ulp in our experiments (for the `j1l`, `y0l` and `y1l` functions, we found inputs that are not correctly rounded thanks to the BaCSeL software tool). For `x=-0x4.179563a9af206c1p+60` which is a negative integer, `tgammal` returns -0 instead of NaN.

Notes about Newlib. Newlib only provides long double functions for platforms where `long double` is the same as `double` (which is not the case of the `x86_64` processor) with two exceptions: `sqrt` and `hypot`. However, in Newlib 4.4.0, the `hypotl` function does not work properly: for $x \geq 2^{8192}$, the call `hypotl(x,0)` gives infinity.

Notes about OpenLibm. The `powl` function does not seem to be thread-safe, the `tgammal` function yields `+Inf` for `x=-0x6.db747ae147ae148p+8` instead of `0x0.01dbd551da54538p-16385`, and `expm1l` yields NaN instead of `0xf.ffffcfce79e56d5p+16380` for `x=0x2.c5c85fdf170c604cp+12`.

Notes about the Apple Math Library. The Apple Darwin ABI for ARM processors maps the C long double type to double, thus there is no real "double extended" format. The same holds for the Microsoft math library.

Notes about LLVM-libc. The LLVM-libc library only implements the square root function in double-extended precision, and for this function we could not find any error larger than 0.5 ulp

function	AMD LibM 4.2 x	max e	RedHat Newlib 4.4.0 x	max e
acos	0x1.35b03e336a82bp-1	1.36	-0x1.0068b067c6feep-1	0.930
acosh	0x1.209fae707a0edp+0	1.32	0x1.0001ff6afc4bap+0	2.25
asin	-0x1.00d44cccfa99p-1	1.06	-0x1.004d1c5a9400bp-1	0.981
asinh	0x1.005ae8d126f7ep+0	1.65	-0x1.02657ff36d5f3p-2	1.92
atan	-0x1.60370d15396b7p-1	0.863	0x1.62ff6a1682c25p-1	0.861
atanh	-0x1.d8fb311a52173p-2	1.04	-0x1.f97fabc0650c4p-4	1.81
cbrt	0x0.7fffffffffffp-1022	1.53e22	-0x1.00ddafe7d9deep-885	0.670
cos	0x1.91e60af551108p-1	0.919	-0x1.4ae182c1ab422p+21	0.887
cosh	0x1.ff76fb3f476d5p+0	1.85	0x1.633cc2ae1c934p+9	2.67
erf	0x1.11642f2eab9edp+0	1.00	-0x1.c57541b55c8ebp-16	1.02
erfc			0x1.5182d8799b84bp+0	4.08
exp	0x1.b97dc8345c55p+5	1.01	0x1.2e8f20cf3cbe7p+8	0.949
exp10	-0x1.285d82b75258fp+2	1.02	0x1.ce7ef793d4b0ap-2	0.896
exp2	0x1.fffbff4152bafp+9	1.05	-0x1.ff95ecb4e6331p-2	0.896
expm1	0x1.9a57c1ec06a74p-2	0.670	0x1.62ff47a01658fp-2	0.909
j0			0x1.45f3067a0f4b2p+847	9.01e15
j1			0x1.45f3066f80258p+325	9.01e15
lgamma			-0x1.3a7fc9600f86cp+1	4.45e15
log	0x1.10803885617a6p+0	0.610	0x1.48ae5a67204f5p+0	0.946
log10	0x1.10fdf4211fd45p+0	1.09	0x1.55535a0140a21p+0	2.08
log1p	0x1.e0013fd35cbbp-4	0.636	-0x1.2bf1de6b04a8ap-2	0.896
log2	0x1.0b541b6746bd1p+0	1.72	0x1.68d778f076021p+0	2.06
sin	-0x1.85e624577c23ep-1	0.895	-0x1.842d8ec8f752fp+21	0.888
sinh	0x1.1feb2a79f307p+3	1.49	-0x1.633cae1335f26p+9	2.67
sqrt	0x1.fffffffffffffp-1	0.500	0x1.fffffffffffffp-1	0.500
tan	0x1.371a47b7e4eb2p+11	1.38	0x1.3f9605aaeb51bp+21	1.02
tanh	-0x1.fde5bd2769a01p-1	1.40	-0x1.e134557098e37p-3	2.22
tgamma			-0x1.535175475cc8dp+7	2.27e3
y0			0x1.c982eb8d417eap-1	1.42e15
y1			0x1.193bed4dff243p+1	5.56e15
atan2	-0x1.ec2003fe89e36p-433 0x1.ec173ed5fd8f5p-427	1.51	-0x1.358bb5eb25bdcp+813 0x1.2f86b82481a0ap+815	1.55
hypot	0x1.ea74e4b1da32bp+1002 -0x1.0a9efc37c242cp+1003	1.03	0x1.6a0a41410b1abp-1004 -0x0.a24afe71b539fp-1022	1.21
pow	0x1.00a000205d461p+1 -0x1.fd35c41fc20bbp+9	0.762	0x1.b44c681a51345p-822 0x1.3e262867f583fp-11	0.892

Table 6: Double precision: AMD LibM and RedHat Newlib.

function	OpenLibm 0.8.3 x	max e	Musl 1.2.5 x	max e
acos	-0x1.0068b067c6feep-1	0.930	-0x1.0068b067c6feep-1	0.930
acosh	0x1.0001ff6afc4bap+0	2.25	0x1.0001ff6afc4bap+0	2.25
asin	-0x1.004d1c5a9400bp-1	0.981	-0x1.004d1c5a9400bp-1	0.981
asinh	-0x1.02657ff36d5f3p-2	1.92	-0x1.0240f2bdb3f25p-2	1.92
atan	0x1.62ff6a1682c25p-1	0.861	0x1.62ff6a1682c25p-1	0.861
atanh	-0x1.f97fabc0650c4p-4	1.81	-0x1.f8a404597baf4p-4	1.80
cbrt	-0x1.13a5ccd87c9bbp+1008	0.668	-0x1.13a5ccd87c9bbp+1008	0.668
cos	-0x1.34e729fd08086p+21	0.834	-0x1.34e729fd08086p+21	0.834
cosh	-0x1.6310ab92794a8p+9	1.47	-0x1.502bf5ad80729p+0	1.04
erf	-0x1.c57541b55c8ebp-16	1.02	-0x1.c57541b55c8ebp-16	1.02
erfc	0x1.5182d8799b84bp+0	4.08	0x1.527f4fb0d9331p+0	3.72
exp	0x1.2e8f20cf3cbe7p+8	0.949	-0x1.18209ecd19a8cp+6	0.511
exp10			-0x1.fe8c27141c94ap+3	4.14
exp2	-0x1.ff1eb5acee46bp+9	0.751	-0x1.1a4ce073ea908p-5	0.511
expm1	0x1.62ff47a01658fp-2	0.909	0x1.62ff47a01658fp-2	0.909
j0	0x1.33d152e971b4p+1	4.51e14	-0x1.33d152e971b4p+1	4.51e14
j1	-0x1.ea75575af6f09p+1	1.10e15	0x1.ea75575af6f09p+1	1.10e15
lgamma	-0x1.3a7fc9600f86cp+1	4.45e15	-0x1.3a7fc9600f86cp+1	4.45e15
log	0x1.48ae5a67204f5p+0	0.946	0x1.dc0b586f2b26p-1	0.520
log10	0x1.553e1cb579ee9p+0	0.814	0x1.553e1cb579ee9p+0	0.814
log1p	-0x1.2bf1de6b04a8ap-2	0.896	-0x1.2bf32aaf122e2p-2	0.900
log2	0x1.67eaf07ce24d1p+0	0.921	0x1.0b53197bd66c8p+0	0.555
sin	0x1.4d84db080b9fdp+21	0.831	0x1.4d84db080b9fdp+21	0.831
sinh	-0x1.63324af2fb5b7p-1	1.88	-0x1.63324af2fb5b7p-1	1.88
sqrt	0x1.fffffffffffffp-1	0.500	0x1.fffffffffffffp-1	0.500
tan	0x1.3f9605aaeb51bp+21	1.02	0x1.3f9605aaeb51bp+21	1.02
tanh	-0x1.e134557098e37p-3	2.22	-0x1.e134557098e37p-3	2.22
tgamma	-0x1.540b170c4e65ep+7	1.03e3	-0x1.fc4b534c8eccp+2	16.0
y0	0x1.c982eb8d417eap-1	1.42e15	0x1.c982eb8d417eap-1	1.42e15
y1	0x1.193bed4dff243p+1	5.56e15	0x1.193bed4dff243p+1	5.56e15
atan2	-0x1.358bb5eb25bdcp+813 0x1.2f86b82481a0ap+815	1.55	-0x1.358bb5eb25bdcp+813 0x1.2f86b82481a0ap+815	1.55
hypot	0x1.6a0a41410b1abp-1004 -0x0.a24afe71b539fp-1022	1.21	0x1.00014d4b1c6b9p-1015 -0x1.000105ba9bf4p-1015	1.04
pow	0x1.000002c5e2e99p+0 0x1.c9eee35374af6p+31	636.	0x1.010e2e7ec0c83p+0 0x1.44bf00479249dp+17	0.525

Table 7: Double precision: OpenLibm and Musl.

	Apple 14.5		LLVM 18.1.8	
function	x	max e	x	max e
acos	-0x1.8d313198a2e03p-53	1.06		
acosh	0x1.00007fb3703ddp+0	2.25		
asin	0x1.eae75e3d82b6fp-2	0.709		
asinh	-0x1.fdefd03df4cd7p-3	1.58		
atan	0x1.01e0be37af68fp+1	0.876		
atanh	0x1.ffd834a270fp-10	2.01		
cbrt	0x1.fed1fe9f1122dp+11	0.729		
cos	0x1.2f29eb4e99fa2p+7	0.948	-0x1.13a5ccd87c9bbp+1008	Inf
cosh	-0x1.62dabd4848dc4p-2	0.523		
erf	-0x1.e057e7a0e494cp-2	6.41		
erfc	0x1.bba14dc3507ccp+1	10.7		
exp	-0x1.4133f4fd79c1cp-13	0.521	0x1p-53	0.5
exp10	-0x1.c37443e446523p-16	0.521	0x1.bcb7b1526e50dp-55	0.5
exp2	-0x1.b3d9b47ad1b2fp-13	0.521	0x1.71547652b82fep-53	0.5
expm1	0x1.e7f93188565ecp-5	0.706	0x1p-52	0.5
j0	0x1.6148f5b2c2e45p+2	3.83e14		
j1	-0x1.ea75575af6f09p+1	1.10e15		
lgamma	-0x1.bffcbf76b86fp+2	2.33e16		
log	0x1.490af72a25a81p-1	0.508	0x1.5b6e7e4e96f86p+2	0.500
log10	0x1.2501ee5628b08p-1	0.514	0x1.e12d66744ff81p+429	0.500
log1p	-0x1.ffffff3fffffdp-28	0.667	0x1p-53	0.500
log2	0x1.6b015f8d9a784p-1	0.515	0x1.b4ebe40c95a01p+0	0.500
sin	-0x1.07e4c92b5349dp+4	0.944	-0x1.13a5ccd87c9bbp+1008	Inf
sinh	0x1.d7131e11fc6b3p-2	0.539		
sqrt	0x1.fffffffffffffp-1	0.500	0x1.fffffffffffffp-1	0.500
tan	-0x1.a81d98fc58537p+6	3.53	-0x1.13a5ccd87c9bbp+1008	Inf
tanh	0x1.00cf9f273d84p+1	0.613		
tgamma	-0x1.5456e56919a19p+7	1.03e3		
y0	0x1.c982eb8d417eap-1	1.42e15		
y1	0x1.193bed4dff243p+1	5.56e15		
atan2	-0x1.6a539153430d8p-416 0x1.d2b5b9dc716d8p-415	0.747		
hypot	0x1.6a0a41410b1abp-1004 -0x1.4495fce36a73ep-1023	1.21	0x1.a308e1455f447p+0 0x1.9d931a83ef879p+0	0.500
pow	0x1.111616f835fb1p-72 0x1.c6cfa07925d49p+3	0.757		

Table 8: Double precision: Apple and LLVM.

	MSVC 2022		FreeBSD 14.1	
function	x	max e		
acos	-0x1.010fd0ad6aa41p-1	0.934	-0x1.0068b067c6feep-1	0.930
acosh	0x1.0007fd4307b75p+0	3.22	0x1.0001ff6afc4bap+0	2.25
asin	-0x1.0239000439deep-1	1.05	-0x1.004d1c5a9400bp-1	0.981
asinh	-0x1.00212bb59c31ep-4	2.05	-0x1.02657ff36d5f3p-2	1.92
atan	-0x1.60370d15396b7p-1	0.863	0x1.62ff6a1682c25p-1	0.861
atanh	-0x1.ffbe8dd88527fp-9	2.50	-0x1.f97fabc0650c4p-4	1.81
cbrt	-0x1.55cd285f321f6p-64	1.86	-0x1.13a5ccd87c9bbp+1008	0.668
cos	-0x1.9200634d4471fp-1	0.897	-0x1.34e729fd08086p+21	0.834
cosh	-0x1.1ff088806d82ep+3	1.91	-0x1.6310ab92794a8p+9	1.47
erf	0x1.755dca4d8b458p+0	4.62	-0x1.c57541b55c8ebp-16	1.02
erfc	0x1.f6094003e85d6p+1	8.46	0x1.5182d8799b84bp+0	4.08
exp	-0x1.74046dfefd9d1p+9	1.50	0x1.2e8f20cf3cbe7p+8	0.949
exp2	-0x1.72286b6f94510p-2	2.23	-0x1.ff1eb5acee46bp+9	0.751
expm1	-0x1.62d7c116d2e32p-1	3.06	0x1.62ff47a01658fp-2	0.909
j0	0x1.bbbd6a9201265p+657	1.88e26	0x1.33d152e971b4p+1	4.51e14
j1	0x1.a635d8219ad13p+157	3.85e26	-0x1.ea75575af6f09p+1	1.10e15
lgamma	-0x1.bffe071eea741p+2	5.10e13	-0x1.3a7fc9600f86cp+1	4.45e15
log	0x1.0ffc349469a2fp+0	0.577	0x1.48ae5a67204f5p+0	0.946
log10	0x1.e005e1e891807p-1	0.633	0x1.553e1cb579ee9p+0	0.814
log1p	-0x1.8000000000000p-53	1.44	-0x1.2bf1de6b04a8ap-2	0.896
log2	0x1.160732376ad7fp+0	0.812	0x1.67eaf07ce24d1p+0	0.921
sin	-0x1.11b624b546894p+9	0.799	0x1.4d84db080b9fdp+21	0.831
sinh	-0x1.aff899f6fcad6p+4	1.51	-0x1.63324af2fb5b7p-1	1.88
sqrt	0x1.fffffffffffffp-1	0.500	0x1.fffffffffffffp-1	0.500
tan	-0x1.4d7c8b8320237p+11	1.32	0x1.3f9605aaeb51bp+21	1.02
tanh	-0x1.fb52ec8460d82p-1	1.44	-0x1.e134557098e37p-3	2.22
tgamma	-0x1.5c00000003c15p+7	9.01e15	-0x1.547cf3ddaaddap+7	1.03e3
y0	0x1.c1d741dc52512p+744	3.30e25	0x1.c982eb8d417eap-1	1.42e15
y1	+0	Inf	0x1.193bed4dff243p+1	5.56e15
cospi			-0x1.fe3bb5207682dp-3	0.812
sinpi			0x1.0806840ac80ap+3	0.811
tanpi			0x1.c9542a6e8f18fp+0	0.969
atan2	-0x1.f1037a6756bfep-881 0x1.959f99be632e6p+142	0.750	-0x1.358bb5eb25bdcp+813 0x1.2f86b82481a0ap+815	1.55
hypot	-0x1.6a5a0ce661358p+890 -0x1.0151c108425b1p+890	1.22	0x1.6a0a41410b1abp-1004 -0x1.4495fce36a73ep-1023	1.21
pow	0x1.fffff9c61ce40p-1 0x1.c4e304ed4c734p+31	91.3	0x1.000002c5e2e99p+0 0x1.c9eee35374af6p+31	636.

Table 9: Double precision: Microsoft and FreeBSD Math Libraries.

	ArmPL 24.04	
function	x	max e
acos	0x1.251869c3f7881p-1	1.52
acosh	0x1.071334daf83adp+0	2.66
asin	0x1.0479b37d95e5cp-1	2.69
asinh	-0x1.000eeed78380ap+0	2.04
atan	0x1.032b4811f3dc5p+0	2.24
atanh	-0x1.e7c1f36602014p-4	3.00
cbrt	0x1.fffad1cec59fep-332	1.79
cosh	-0x1.628af341989dap+9	1.93
erf	0x1.0000187085e56p-8	2.29
erfc	0x1.46cffdf330b13p+4	1.71
exp	-0x1.49f33ad2c1c58p+9	0.511
exp10	-0x1.5acf96f42165bp-7	0.510
exp2	-0x1.f7087fb1cf9e8p+9	0.509
expm1	0x1.633f993a730c9p-2	2.18
log	0x1.1211bef8f68e9p+0	0.520
log10	0x1.de02157073b31p-1	1.62
log1p	-0x1.2e496d25897ecp-2	1.74
log2	0x1.0b53f741fb8c4p+0	0.554
sinh	0x1.9fa32b1149d35p-2	2.59
tanh	-0x1.c416448380debp-3	2.76
cospi	0x1.5a33cc258p-22	3.18
sinpi	0x1.ffffe4b85e77ap-2	3.16
atan2	0x1.d5de7a294d493p-935 0x1.d030d7b608be1p-935	2.23
pow	0x1.010e2e7ee71aep+0 0x1.44bf0047427f6p+17	0.523

Table 10: Double precision: ArmPL.

	CUDA 12.2.1		ROCm 5.7.0	
function	x	max e	x	max e
acos	0x1.266637a3d2bbcp-1	1.53	-0x1.36b1482765f6dp-1	0.772
acosh	0x1.1d7bc19163966p+0	2.52	0x1.0aaab62cc290dp+0	0.661
asin	-0x1.2ef2481799c7cp-1	1.99	0x1.df27e1c764802p-2	0.710
asinh	0x1.0ab3fc30267c2p-1	2.57	0x1.2aae7abf26ce3p-2	0.661
atan	0x1.52184b1b9bd9bp+0	1.77	-0x1.0684fa9fa7481p+0	1.73
atanh	-0x1.f586714622a66p-3	2.50	-0x1.2493fecd07e5p-3	0.664
cbrt	-0x1.588a24f9a953fp+535	0.501	0x1.1e0ef6faa076p+175	0.501
cos	0x1.25133ca3904dfp+20	1.52	0x1.2a33ae49ab15dp+1	0.797
cosh	0x1.e7ffe229fe99ep+1	1.40	-0x1.e7fa36b6eb43p+1	0.563
erf	0x1.340ff534d52bfp-2	1.50	-0x1.10c4c3d3b6cdbp+0	1.12
erfc	0x1.8659a03b35abcp-7	4.51	0x1.f1193828dcc1ep-19	4.08
exp	-0x1.625f1b359729ep+9	0.928	-0x1.625f1c27780c8p+9	0.929
exp10	-0x1.a7d980016dc5ap+0	1.11	0x1.5c1ece7fea4bep+0	1.11
exp2	-0x1.ff40169bd093bp+9	0.948	-0x1.ff3ffea62d3d7p+9	0.947
expm1	0x1.a0e95d59498e9p-2	1.18	0x1.632cfb1033275p-2	1.91
j0	-0x1.0e126bbcb3e65p+25	3.08e20	0x1.ddca13ef271d2p+3	1.25e13
j1	-0x1.635ab5a8baf45p+26	1.73e21	0x1.aa5baf310e5a2p+3	4.80e13
lgamma	-0x1.fa471547c2fe5p+1	5.11e15	-0x1.3a7fc9600f86cp+1	4.45e15
log	0x1.69e7aa6da2df5p-1	0.564	0x1.5556123e8a2bp-1	0.663
log10	0x1.803dea263187fp-1	1.43	0x1.55558196a2cbp+0	0.785
log1p	-0x1.ffffffbaefe27p-2	1.50	-0x1.5efad5491a79bp-1022	1.00
log2	0x1.670c5aa6680abp+0	1.31	0x1.5556d5fbb94cbp+0	0.734
sin	-0x1.1c49ad613ff3bp+19	1.52	-0x1.f05e952d81b89p+5	0.800
sinh	0x1.be64384e3ac1ep+0	1.51	-0x1.ff9faf9b69235p-5	0.868
sqrt	0x1.fffffffffffffp-1	0.500	0x1.fffffffffffffp-1	0.500
tan	0x1.da7a85a88bbecp+11	2.09	-0x1.66af736e8555p+18	1.30
tanh	-0x1.19398a9a24319p-1	1.48	0x1.00433533940cdp-4	0.866
tgamma	-0x1.2baa17692a3f2p+7	10.1	-0x1.201a11d80c13dp+2	13.7
y0	0x1.16bad92479879p+25	1.18e21	0x1.ab8e1c4a1e74ap+3	1.95e13
y1	0x1.2391e4c8faa6p+26	1.17e21	0x1.e9e480605283cp+4	6.14e13
cospi	-0x1.ae7b6f6da3747p-2	1.52	-0x1.7e00a005862afp+5	0.894
sinpi	-0x1.4778e04770c45p-4	1.52	-0x1.eff4d839e0c49p+2	0.890
rsqrt	0x1.f80d8004b3ae9p+479	0.510	0x1.0000000000002p+484	1.00
atan2	0x1.9cde4ff190e45p+931 0x1.37d91467e558bp+931	1.76	0x1.401ec07d65549p+888 0x1.3c3976605bb0cp+888	1.82
hypot	-0x1.41fcfeeb2e246p+420 -0x1.8d4d41eacdeccp+420	1.89	0x1.afa7134ad6d8p-403 0x1.6a0ff6e086067p-384	1.21
pow	0x1.6b2d4fdb85ba1p-1 -0x1.f0d1d713b0262p+10	1.84	0x1.17efb14831458p-421 0x1.f8c34d6504b2p-7	1.40

Table 11: Double precision: CUDA and ROCm.

(for rounding to nearest). Since a single function is implemented, we don't mention LLVM-libc in Tables 12-15.

Notes about FreeBSD. The FreeBSD extended double power function is not thread safe (like the OpenLibm one), and for `x=-0x6.e000000000000008p+8` which is very near a negative integer, the FreeBSD `tgammal` function yields -0 instead of `-0x4.b40cdf839d0bbp-16392`.

5 Quadruple Precision

Only the GNU libc and the Intel Math Library support quadruple precision, through the `_Float128` type in GNU libc, and `_Quad` in the Intel Math Library (using the option of the Intel C compiler `-Qoption,cpp,--extended_float_types`). The results are summarized in Table 16, and detailed in Table 17. Only the square root function is correctly rounded (or at least seems to be). The Intel Math Library gives better results than the GNU libc for all functions it provides, except for `lgamma` and `tgamma`. Apart from these two functions, and from the Bessel functions `j0`, `j1`, `y0`, `y1`, the observed error for the Intel Math Library is at most 1.4 ulps. The GNU libc has large errors for `j0`, `j1`, `y0`, `y1`, and `pow`.

Acknowledgements. The authors thank Claude-Pierre Jeannerod, Vincent Lefèvre and Terje Mathisen who helped improving that article, Alexei Sibidanov who helped compiling Newlib, Eric Schneider, Nick Timmons, Hugues de Lassus and Fred J. Tydeman for interesting discussions. Joseph Myers suggested to included the double extended format. Experiments presented in this article were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see `https://www.grid5000.fr`). This work was also supported by the French "Ministère de l'Enseignement Supérieur et de la Recherche", by the "Conseil Régional de Lorraine", and by the European Union, through the "Cyber-Entreprises" project. Experiments on GPU were performed on hardware made available by CERN. Experiments with the Microsoft library were possible thanks to Brice Goglin and the TADaaM project-team from Inria.

References

[1] AMD LibM version 4.2. `https://developer.amd.com/amd-aocl/amd-math-library-libm/`, 2024.

[2] Apple Math Library (MacOS 14.5, Apple M1).

[3] Arm Performance Libraries version 24.04. `https://developer.arm.com/downloads/-/arm-performance-libraries`, 2024.

[4] Arnold, J. M. A study of the `rsqrt` and `rcp` instructions on Intel and AMD platforms. `https://github.com/jeff-arnold/math_routines.git`, 2016. 22 pages.

[5] Bailey, D. H. Variable precision computing: Applications and challenges. Slides presented at the ICERM workshop on Variable Precision in Mathematical and Scientific Computing, 2020. `https://www.davidhbailey.com/dhbtalks/dhb-icerm-2020.pdf`.

library	GNU libc	Intel Math Library	OpenLibm	Musl	FreeBSD
version	2.40	IML 2024.0.2	0.8.3	1.2.5	14.1
acos	1.75	**0.505**	0.938	1.75	0.938
acosh	2.99	**0.502**	3.14	2.99	2.24
asin	1.15	**0.506**	1.03	2.00	1.03
asinh	2.96	**0.506**	3.19	2.96	1.63
atan	0.640	**0.501**	1.10	0.640	1.10
atanh	2.88	**0.501**	85.4	3.19	1.52
cbrt	0.824	**0.503**	0.890	0.890	0.890
cos	1.51	**0.502**	0.799	0.799	0.799
cosh	3.40	**0.502**	4.86	3.73	0.936
erf	1.17	**0.518**	1.17	1.17	0.995
erfc	4.73	**0.527**	5.77	5.12	1.38e8
exp	1.27	**0.501**	2.00	1.54	0.752
exp10	1.50	**0.501**		40.1	
exp2	0.788	**0.501**	2.18	0.788	0.753
expm1	3.08	**0.502**	Inf	9.71e3	0.517
j0	9.79e17	**0.501**			
j1	3.38e18	**0.501**			
lgamma	12.2	**0.549**	9.08e19	9.08e19	1.65e20
log	0.998	**0.501**	1.22	0.998	0.512
log10	1.36	**0.502**	1.22	1.36	0.511
log1p	2.49	**0.501**	2.60	2.49	0.516
log2	0.995	**0.502**	1.64	0.995	0.509
sin	1.51	**0.502**	0.799	0.799	0.799
sinh	3.40	**0.503**	4.85	9.71e3	0.803
sqrt	**0.500**	**0.500**	**0.500**	**0.500**	**0.500**
tan	1.76	**0.504**	1.02	1.02	1.02
tanh	3.22	**0.506**	2.56	2.95	0.640
tgamma	**9.77**	Inf	Inf	3.69e19	4.24e16
y0	1.38e18	**0.501**			
y1	4.61e18	**0.501**			
cospi					**0.797**
exp10m1	**3.57**				
exp2m1	**3.20**				
log10p1	**4.71**				
log2p1	**3.95**				
sinpi					**0.797**
tanpi					**1.50**
rsqrt		**0.501**			
atan2	0.751	**0.501**	1.69	0.751	1.69
compound		**0.502**			
hypot	**0.584**	0.751	0.981	1.08	0.981
pow	0.914	**0.501**	3.77e4	3.77e4	3.77e4

Table 12: Double extended precision: Largest known error.

function	GNU libc 2.40 x	max e	IML 2024.0.2 x	max e
acos	0xf.fe002cabd608585p-4	1.75	0x8.af256cd27462348p-4	0.505
acosh	0x1.1ecdb5b8f0c5d79p+0	2.99	0x1.1f9c4feedfe4f2cp+0	0.502
asin	0x8.171fd358c4cb27bp-4	1.15	-0x8.018aef8787e5a6bp-4	0.506
asinh	-0x8.0bb656992eac437p-4	2.96	0x7.ff15da44c3651abp-4	0.506
atan	-0x1.0411ae010d4c5b1ep+0	0.640	-0x8.00f60592e42d79p+8	0.501
atanh	-0x3.337ceaccc9025258p-4	2.88	0x3.e7be418257523408p-4	0.501
cbrt	-0xc.f4fd71a450e6a0bp-14732	0.824	-0x2.320375fd33ed311cp-13376	0.503
cos	-0x3.d067a048093bdf94p+9160	1.51	-0x4.b0df0d7d55044918p+8	0.502
cosh	0x2.c5d375f827733ac4p+12	3.40	-0x7.f6a09874512cf768p-4	0.502
erf	0xd.7fe64ab05cf75e8p-4	1.17	-0x1.c55160e785ee1cbap-4	0.518
erfc	0x1.59723d7ee47e3034p+0	4.73	0x3.03c7b9f943690558p-4	0.527
exp	0x5.8b9111182b4467ep-4	1.27	0x2.c590e6ab0d71c77p+12	0.501
exp10	0x1.2da9675e95849c3ep+12	1.50	-0x1.2ab76ac25255a1aap+12	0.501
exp2	-0x7.3f819acf048f1678p-4	0.788	-0x3.fe9a346527a75d98p-16	0.501
expm1	0x5.8b910bbe3c26818p-4	3.08	-0x1.0040016b56008656p-8	0.502
j0	-0x2.67a2a5d2e367f784p+0	9.79e17	-0x1.6a09e667f3bd238cp-32	0.501
j1	0x3.d4eaaeb5ede115p+0	3.38e18	0x8.001819d5fc886dap-4	0.501
lgamma	-0x3.ec9403f23a1f21cp+0	12.2	-0x4.07fe15510b6a28p+0	0.549
log	0x1.20dad075f537ae56p+0	0.998	0x1.1001246349edf00cp+0	0.501
log10	0x1.272b7c3bbb08ae12p+0	1.36	0x1.010141e1049fce68p+0	0.502
log1p	-0x6.451f6c3fd0d4a218p-4	2.49	-0xe.fefa23913fa3eb7p-8	0.501
log2	0x1.058f12b8b3ac44bep+0	0.995	0x1.01004bfffe4316bep+0	0.502
sin	-0x6.e2368c0ed74e5698p+16	1.51	-0xc.141cf155623856bp+8	0.502
sinh	0x2.c5d375f827733ac4p+12	3.40	0x7.b0af44fc25df3efp-4	0.503
sqrt	0xf.ffffffffffffffp-4	0.500	0xf.ffffffffffffffp-4	0.500
tan	0x1.974cd2181086913p+8	1.76	0xc.845cb771b06f4c5p+0	0.504
tanh	0x3.b9979a543d0fbfa8p-4	3.22	0x7.fb808a1ef99076ep-4	0.506
tgamma	-0x1.70a55b2628a7cb68p+4	9.77	-0x4.179563a9af206c1p+60	Inf
y0	0xe.4c175c6a0bf51e8p-4	1.38e18	0x1.000213a50d97fd8ep+0	0.501
y1	0xb.bfc89c6a1903022p+0	4.61e18	0x4.002362c1b67ad6cp+0	0.501
exp10m1	0x2.c7fd02fd98797bf4p-4	3.57		
exp2m1	0x9.3c1d13fc7c58944p-4	3.20		
log10p1	0x4.a486e7fa771f839p-4	4.17		
log2p1	0x5.843d01be597f38fp-4	3.95		
rsqrt			0x1.61e30a1ac16221eap+12600	0.501
atan2	-0x7.9301460b8463cbp+15368 0xf.25cd5eb1280b4d1p+15372	0.751	-0x5.c0c9cc5a59632f88p+16340 0x5.db7810fba1ce4908p+16348	0.501
compound			0xa.613dc3f1daa2566p+5432 -0x4.fcf34863ee4e7a38p-20	0.502
hypot	-0x2.97b86706043d619p+7240 0x1.8256bdd12d2e163ep+7240	0.584	-0x3.00bad8a56d87a0cp-16384 -0xe.6d794db04791398p-16388	0.751
pow	0x2.21dda4bcec55b158p-3616 0x7.ef1ef5fbe3df50dp-16	0.914	0xc.b80572af668bb57p+152 -0x6.8a6d3d7b442f3c18p+4	0.501

Table 13: Double extended precision: GNU libc and Intel Math Library.

function	OpenLibm 0.8.3 x	max e	Musl 1.2.5 x	max e
acos	-0x8.040541d0054d89p-4	0.938	0xf.fe002cabd608585p-4	1.75
acosh	0x1.10384b24aec007fcp+0	3.14	0x1.1ecdb5b8f0c5d79p+0	2.99
asin	0x8.0519515d1e15a6bp-4	1.03	-0x3.fff0a397b8dea17cp-8	2.00
asinh	-0x5.c9866cb231f2c7c8p-4	3.19	-0x8.0bb656992eac437p-4	2.96
atan	0x6.fffde214a06fb5f8p-4	1.10	-0x1.0411ae010d4c5b1ep+0	0.640
atanh	-0xf.fffffffffffe78p-32	85.4	0x3.344a915e34e5e6b8p-4	3.19
cbrt	-0x3.fffffffa5623708p+4588	0.890	-0x3.fffffffa5623708p+4588	0.890
cos	0x3.e0db6fa4b23541ap+4	0.799	0x3.e0db6fa4b23541ap+4	0.799
cosh	0x2.c5d374f9436efd1p+12	4.86	0x2.c5d37484e4c162bp+12	3.73
erf	0xd.7fe64ab05cf75e8p-4	1.17	0xd.7fe64ab05cf75e8p-4	1.17
erfc	0x1.5cc0e1cc32a3dc98p+0	5.77	0x1.5c9262fa4210902p+0	5.12
exp	0x8.aa2253c0d601dedp+0	2.00	-0x2.c5a1073a0f38b61cp+12	1.54
exp10			0xd.41cfea690e121b5p+8	40.1
exp2	-0xf.ffffd9f32ee1e06p-12	2.18	-0x7.3f819acf048f1678p-4	0.788
expm1	0x2.c5c85fdf170c604cp+12	Inf	0x2.c5c85fdf170c604cp+12	9.71e3
lgamma	-0x2.74ff92c01f0d82acp+0	9.08e19	-0x2.74ff92c01f0d82acp+0	9.08e19
log	0xb.504a14384e9b137p-4	1.22	0x1.20dad075f537ae56p+0	0.998
log10	0xb.ffac4b4c47e00c3p-4	1.22	0x1.272b7c3bbb08ae12p+0	1.36
log1p	-0x4.c669bd1813ec8bd8p-4	2.60	-0x6.451f6c3fd0d4a218p-4	2.49
log2	0x1.6646b082fd1065cep+0	1.64	0x1.058f12b8b3ac44bep+0	0.995
sin	-0x2.a2a4a117ff34b034p+8	0.799	-0x2.a2a4a117ff34b034p+8l	0.799
sinh	-0x2.c5d375cbe7e4a81cp+12	4.85	0x2.c5c85fdbc1ccc354p+12	9.71e3
sqrt	0xf.ffffffffffffffp-4	0.500	0xf.ffffffffffffffp-4	0.500
tan	-0x6.fae45244b0bc104p+8	1.02	-0x6.fae45244b0bc104p+8	1.02
tanh	0x3.8b2602d43bdf4c28p-4	2.56	0x4.024182351388d15p-4	2.95
tgamma	-0x6.db747ae147ae148p+8	Inf	-0x2.8d19fd20f3aa62cp+4	3.69e19
atan2	0x3.d34c9d81dcd29354p+5568 0xf.3afc4f6c9f5c4a2p+5568	1.69	-0x7.9301460b8463cbp+15368 0xf.25cd5eb1280b4d1p+15372	0.751
hypot	0x1.73f339f61eda21dp-16384 0x2.e45f9f9500877e2p-16384	0.981	0x2.00007da75fd5903cp-8960 0x2.d42207352184bff4p-8960	1.08
pow	0xf.a795000b7dae5b4p-4 -0x7.e4a42355b11a8098p+16	3.77e4	0xf.a795000b7dae5b4p-4 -0x7.e4a42355b11a8098p+16	3.77e4

Table 14: Double extended precision: OpenLibm and Musl.

	FreeBSD 14.1	
function	x	max e
acos	-0x8.040541d0054d89p-4	0.938
acosh	0x1.0001fe8811e16ee2p+0	2.24
asin	0x8.0519515d1e15a6bp-4	1.03
asinh	-0x8.4d30725ad98215ap-4	1.63
atan	0x6.fffde214a06fb5f8p-4	1.10
atanh	-0x7.ff5cbca0e5646c78p-12	1.52
cbrt	-0x3.fffffffa5623708p+4588	0.890
cos	0x3.e0db6fa4b23541ap+4	0.799
cosh	-0xf.c52a6a14a334e77p-4	0.936
erf	0x1.c5a9ec676d7e551ep-24	0.995
erfc	0x3.fffffff7fffffc34p-36	1.38e8
exp	-0x2.c5b2c28ca01620dcp+12	0.752
exp2	-0x3.ffe0d002661900a8p+12	0.753
expm1	0x3.80e4f0677c553158p-4	0.517
lgamma	-0x2.74ff92c01f0d82acp+0	1.65e20
log	0x1.01007581714f057ap+0	0.512
log10	0x1.0101ff27167c589ap+0	0.511
log1p	0x1.0071fc209b87c88p-8	0.516
log2	0x1.0101e3c5cde465ccp+0	0.509
sin	-0x2.a2a4a117ff34b034p+8	0.799
sinh	0xd.ae04a309c0411f3p-4	0.803
sqrt	0xf.ffffffffffffffp-4	0.500
tan	-0x6.fae45244b0bc104p+8	1.02
tanh	0x1.80371de2d031a66ep+0	0.640
tgamma	-0x6.e000000000000008p+8	4.24e16
cospi	-0x3.f25066fd5ea7265p-4	0.797
sinpi	-0x4.019c9d05624aa028p-4	0.797
tanpi	-0x1.7fffffffd731f108p+0	1.50
atan2	0x3.d34c9d81dcd29354p+5568 0xf.3afc4f6c9f5c4a2p+5568	1.69
hypot	0x1.73f339f61eda21dp-16384 0x2.e45f9f9500877e2p-16384	0.981
pow	0xf.a795000b7dae5b4p-4 -0x7.e4a42355b11a8098p+16	3.77e4

Table 15: Double extended precision: FreeBSD.

library version	GNU libc 2.40	Intel Math Library IML 2024.0.2
acos	1.28	**0.502**
acosh	4.00	**0.501**
asin	1.20	**0.502**
asinh	3.95	**0.501**
atan	1.41	**0.501**
atanh	3.89	**0.501**
cbrt	0.736	**0.501**
cos	1.52	**0.501**
cosh	1.92	**0.501**
erf	1.42	**0.501**
erfc	4.38	**0.504**
exp	0.751	**0.501**
exp10	2.00	**0.501**
exp2	1.08	**0.501**
expm1	1.64	**0.501**
j0	4.10e32	**2.90e28**
j1	3.57e33	**3.33e31**
lgamma	**13.0**	2.79e30
log	1.05	**0.501**
log10	2.01	**0.501**
log1p	3.51	**0.501**
log2	3.31	**0.501**
sin	1.52	**0.501**
sinh	2.07	**0.501**
sqrt	**0.500**	**0.500**
tan	1.06	**0.502**
tanh	2.39	**0.501**
tgamma	**10.7**	8.20e3
y0	1.69e33	**4.79e27**
y1	3.47e33	**1.45e30**
exp10m1	**3.25**	
exp2m1	**2.23**	
log10p1	**3.47**	
log2p1	**3.35**	
rsqrt		**0.501**
atan2	1.89	**0.501**
compound		**0.501**
hypot	0.792	**0.501**
pow	30.3	**1.40**

Table 16: Quadruple precision: Largest known error.

function	GNU libc 2.40 x	max e	IML 2024.0.2 x	max e
acos	0x9.fdbe71e81d65064f0f24b2602998p-4	1.28	0xf.f80616c2416bf63c33a739ae3a08p-4	0.502
acosh	0x1.0f97586eba090200118df0902f99p+0	4.00	0x1.004ae7a1e9d7b621b12baeda616dp+0	0.501
asin	0x7.79659a0b568bad280c8ec7eb8278p-4	1.20	0x7.ff86cc20db4e6f7fd33ce212282cp-8	0.502
asinh	0x5.a924236647ffb723576b172b52fcp-4	3.95	0x1.0000f6bea05a0cafd1e775e627d3p-4	0.501
atan	0x3.7ff864717fc99760d470d1a994cp-4	1.41	-0x1.15eb4e54ee6ca35bf8b1764f30d4p+0	0.501
atanh	0x2.c02a24f3472c7840afbd8cfb68bap-4	3.89	-0xd.9fe29c463116c87fa567e436489p-8	0.501
cbrt	-0x5.a837d1198a72e5a89695db79896cp-13792	0.736	-0x2.10d29fbb2036d1d7ffdd8bf63184p+10912	0.501
cos	-0x3.08db9df46e0cd142071fdec7eb6p+64	1.52	-0x6.081f6e15f81d27ac2a6038eed3bp+2232	0.501
cosh	-0x2.c5d376fd225ce5739bef59cb0e16p+12	1.92	-0x2.ba5adc2ddaf3f5466db2cd018394p+4	0.501
erf	0xd.f3a140b19b0e7d0fafae7eec5ebp-4	1.42	0x5.a5182e2e3fce6963a492839ebb3cp-8	0.501
erfc	0x1.517e84504890cba9f9f65ff93206p+0	4.38	0x6.0a5ca72c4efcd78f90acc0aefbbp+0	0.504
exp	-0x2.c5b323ac8f24d66ed41ee61ab6bap+12	0.751	-0x5.6622c128e27c6a8c991743947adcp-8	0.501
exp10	0x3.e9d3cc7e0cbdc5bc7fdfc1932fd6p+0	2.00	0x1.1e2a2ef09a4f66e4d3648a85045bp+12	0.501
exp2	0x1.fffe69758fd951b5213a6d47be1ap+0	1.08	-0x7.cab667376a3dd98217d7b028adccp-8	0.501
expm1	0x5.a1195b05aec378d0b236943f4a18p-4	1.64	0x8.ca3ec068eee81b45c0adcae049ap+4	0.501
j0	-0x8.a75ab6666f64eae68f8eb383dad8p+0	4.10e32	0x3.7c3f883498c0d5e0dab7e54a98b2p+4	2.90e28
j1	-0x1.7059c8d303730c6b82b12d9941b9p+8	3.57e33	-0x1.7059c8d303730c6b82b12d9941b9p+8	3.33e31
lgamma	-0x3.ec2152452b5eaf0f070d215b3418p+0	13.0	-0x3.24c1b793cb35efb8be699ad3d9bap+0	2.79e30
log	0xf.d016f49074a9c4fe793af2394278p-4	1.05	0xc.4806c5e4877bbeb4b44ed03d9f18p-5364	0.501
log10	0x1.6a291ea0aa11fb374f1df8b3ac6bp+0	2.01	0x1.9b621e77f399e4a8c1a85a964e94p-12364	0.501
log1p	0x6.a0aed5f6dad05d6ff33ecd883dc8p-4	3.51	-0x6.2611e37be5cf4388865319f859b4p-12	0.501
log2	0xb.54170d5cfa8fd72a47d6bda19068p-4	3.31	0xf.f63cee8e97ac6783532625273eap-4	0.501
sin	0x5.6a5005df151cc2274e119666a9c8p+64	1.52	0x4.246e3c1f1094e4159999f13cff24p+5604	0.501
sinh	0x6.7e79f3aada38698b910c300b19b8p-4	2.07	-0x1.6606d9c89bc66d481844a8589dcbp+0	0.501
sqrt	0xf.ffffffffffffffffffffffffffp-4	0.500	0xf.ffffffffffffffffffffffffffp-4	0.500
tan	-0x3.832b771f9462df46117b6a863fa2p+8	1.06	0xb.eb95e948d6f2a74a1d3a7694bd88p+3816	0.502
tanh	-0x3.c26abeca541298cca288adbd1e12p-4	2.39	-0x2.01d7bf6773e2b04acd388c84cd4ep-4	0.501
tgamma	-0x1.62ab0823decc5cf957d9a218cf27p+4	10.7	0x2.00003274fc8659f8ed68e96e0378p-16224	8.20e3
y0	0x6.b99c822052e965e1754eb5ffeb08p+4	1.69e33	0x3.9561432d16442ec543c74876d1c8p+4	4.79e27
y1	0x2.3277da9bfe485c85c35e5bcc806p+0	3.47e33	0x2.80bc307275f6a6a3feb2ab211838p+4	1.45e30
exp10m1	0xb.2ee9062818e91afe8e80fae1818p-4	3.25		
exp2m1	0xb.2ee9062818e91afe8e80fae1818p-4	2.23		
log10p1	-0x6.eec527c6a8d6e31ca9f0dcdd747cp-4	3.47		
log2p1	-0x7.fff3a57fdd2666bcecba33bd89f4p-4	3.35		
rsqrt			0x1.00db76159f986d3a3614199fd36fp-188	0.501
atan2	0x1.41df5aa214612c7e019fa6ade88p-13316 0x5.e53b26a270a29eb9f77ef8ef7af8p-13316	1.89	-0x1.fb41ff205f5ade930a9fcbba8ea8p-16384 0x2.23f098fd6b8799dbeb03219bfa08p-10520	0.501
compound			0x2.660c17a54ded3960195d40db8ee4p-11652 0xd.45349a19be26b49582eb8c9bb588p+11660	0.501
hypot	-0x1.80e7403e1b344c4a78edeced92e4p-16384 -0x2.986c750d01c32e4c807c12ad685p-16384	0.792	0x8.79ec30b61f9b839fe507bbdf414p-11908 0xb.94f6832f64d0729ebd68035ed7a8p-11908	0.501
pow	0x1.364dcbbad0512d7bacaae2a8d56bp+0 -0xe.68759219434c37725fdf30d17d2p+12	30.3	0x4p-16496 0x3.ffffff39c102f0aa11bb2c8a91dp-128	1.40

Table 17: Quadruple precision: GNU libc and Intel Math Library.

[6] Black, C. M., Burton, R. P., and Miller, T. H. The need for an industry standard of accuracy for elementary-function programs. *ACM Trans. Math. Softw. 10*, 4 (1984), 361–366.

[7] Brisebarre, N., Hanrot, G., Muller, J.-M., and Zimmermann, P. Correctly-rounded evaluation of a function: why, how, and at what cost? working paper or preprint, May 2024.

[8] CUDA C Programming Guide v12.2.1, Section H Mathematical Functions. https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#mathematical-functions-appendix, 2023.

[9] CUDA Math Library. https://developer.nvidia.com/cuda-math-library, 2023.

[10] Ferguson, W., Cornea, M., Anderson, C., and Schneider, E. The difference between x87 instructions fsin, fcos, fsincos, and fptan and mathematical functions sin, cos, sincos, and tan, 2015. https://software.intel.com/content/dam/develop/external/us/en/documents/x87trigonometricinstructionsvsmathfunctions.pdf.

[11] Fousse, L., Hanrot, G., Lefèvre, V., Pélissier, P., and Zimmermann, P. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw. 33*, 2 (2007), article 13.

[12] FreeBSD libc version 14.1. https://www.freebsd.org/releases/14.1R/announce/, 2024.

[13] Glatard, T., Lewis, L. B., da Silva, R. F., Adalat, R., Beck, N., Lepage, C., Rioux, P., Rousseau, M., Sherif, T., Deelman, E., Khalili-Mahani, N., and Evans, A. C. Reproducibility of neuroimaging analyses across operating systems. *Frontiers Neuroinformatics 9* (2014), 12.

[14] GNU libc: Known maximum errors in math functions. http://www.gnu.org/software/libc/manual/html_node/Errors-in-Math-Functions.html, 2023.

[15] GNU libc version 2.40. https://www.gnu.org/software/libc/, 2024.

[16] Hubrecht, T., Jeannerod, C.-P., and Zimmermann, P. Towards a correctly-rounded and fast power function in binary64 arithmetic. In *ARITH 2023 - 30th IEEE Symposium on Computer Arithmetic* (2023). Long version available at https://inria.hal.science/hal-04159652.

[17] IEEE standard for floating-point arithmetic, 2019. 84 pages.

[18] Intel Math Library. Distributed with the Intel oneAPI DPC++ Compiler 2024.0.2, 2024.

[19] Kong, S., Gao, S., and Clarke, E. M. Floating-point bugs in embedded GNU C library. Tech. Rep. CMU-CS-13-130, Carnegie Mellon University, 2013. Available at http://reports-archive.adm.cs.cmu.edu/anon/2013/CMU-CS-13-130.pdf.

[20] Lee, W., Sharma, R., and Aiken, A. On automatically proving the correctness of math.h implementations. In *Proceedings of the ACM on Programming Languages (POPL)* (2017), pp. 41:1–47:32. https://doi.org/10.1145/3158135.

[21] LLVM-libc C standard library 18.1.8. https://github.com/llvm/llvm-project/releases, 2024.

[22] Microsoft Visual Studio 2022, 2022.

[23] Muller, J.-M. On the definition of ulp(x). Research Report RR-5504, LIP RR-2005-09, INRIA, LIP, Feb. 2005.

[24] Musl version 1.2.5. https://musl.libc.org/, 2024.

[25] Redhat Newlib version 4.4.0. https://sourceware.org/newlib/, 2023.

[26] ROCm Math Library. https://github.com/RadeonOpenCompute/ROCm, 2023.

[27] OpenLibm version 0.8.3. https://openlibm.org/, 2024.

[28] Petzold, M. A note on the first moment of extreme order statistics from the normal distribution. Tech. rep., Göteborg University. School of Business, Economics and Law, 2000. 6 pages, https://gupea.ub.gu.se/handle/2077/3092.

[29] Sibidanov, A., Zimmermann, P., and Glondu, S. The CORE-MATH Project. In *ARITH 2022 - 29th IEEE Symposium on Computer Arithmetic* (virtual, France, Sept. 2022).