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

Vincenzo Innocente and Paul Zimmermann

September 7, 2021

> *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 (see Table I) demonstrate that the industry does not satisfy the needs of those who require accurate and efficient mathematical software.*

These lines, written in 1984 by Black, Burton and Miller [5], are unfortunately still very true today.

The IEEE 754 standard, even in its latest 2019 revision [13], does not require correct rounding of mathematical functions. In turn, current mathematical libraries do not provide correct rounding, which would be the best possible result. Thus, users might get different results with different libraries, or even with different versions of the same library. This can have dramatic consequences: for example missed collisions in the Large Hadron Collider [4] or reproducibility issues in neuroimaging [10].

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 maximal errors [6] or known maximal errors [11].

1 Introduction

In this document we compare the accuracy of the eight following mathematical libraries (in the rounding to nearest mode): GNU libc 2.34 [12], the Intel Math Library shipped with the Intel

oneAPI DPC++ Compiler 2021.2.0 (icx) [14], AMD LibM 3.7 [1], RedHat Newlib 4.1.0 [17], OpenLibm 0.7.5 [18], Musl 1.2.2 [16], the Apple Math Library available under Darwin [2], and the CUDA mathematical library [7]. We do not compare to the x87 instructions `fsin` and others, which are known to have bad accuracy [8].

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 [9], using a larger precision. We also denote by E the absolute difference between y and $Z = \mathrm{RN}(z)$, in terms of units-in-last-place of Z, where Z is the correct rounding of z to nearest to the target precision, obtained with MPFR. Thus e is a real, while E is an integer (except in some corner cases). 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 [15].

The results for GNU libc, AMD LibM, Newlib, OpenLibm and Musl were obtained on an Intel Core i5-4590, with GCC 10.2.1 under Debian (note that the GNU libc results might differ sligthly from one `x86_64` processor to another one, due for example to the use of fused-multiply add or not). Those for the Intel Math Library were obtained on an Intel Xeon (E5-2680 or Gold 6142), with icx version 2021.2.0, using `-fp-model=strict`. Those for the Apple math library were obtained on a Mac M1 (arm64) under Darwin 20.4.0, with clang 12.0.5. The CUDA library was tested on a NVidia Tesla V100-SXM2 running CUDA 11.2.2 hosted on a Intel Xeon Gold 6148. The test were also run on a GTX1060 GPU, hosted on an AMD Ryzen 7 1800X, obtaining identical results. Newlib was configured with default flags (in particular, without use of hardware FMA), and with the default configuration.[1]

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 $2.17\mathrm{ulp}(z)$ for all `binary32` inputs, and in all other cases (larger formats or bivariate functions) it means the largest *known* error is bounded by $2.17\mathrm{ulp}(z)$, with at least one case giving an error of more than $2.16\mathrm{ulp}(z)$.

2 Single Precision

2.1 Univariate Functions

The IEEE 754 single-precision (`binary32`) format has $2^{32} - 2^{24} = 4278190080$ values, not counting +Inf, -Inf, and NaN. For a function with a single input—i.e., excluding the `pow` function for example—it is possible to check all values by exhaustive search.

Table 1 summarizes the maximal value of e for each function and each library. In detailed tables (Tables 2, 3, 4, 5), we indicate the number of inputs with $E \geq 1$ (thus incorrectly rounded), with $E \geq 2$, and the maximum value of e.

In all tables, the notation NA means "Not Available" (`exp10` in OpenLibm, and the Bessel functions `j0`, `j1`, `y0`, `y1` in Apple libm).

We see that for all libraries, the sqrt function is correctly rounded for all binary32 inputs, as required by IEEE 754. The single-precision cubic root function (`cbrt`) is also correctly rounded in

[1] For binary32, by default, the old SunPro functions are used; with `OBSOLETE_MATH_DEFAULT=0`, Newlib will use instead a new set of mathematical functions provided by Arm, that use binary64 for intermediate computations.

library	GNU libc	IML	AMD	Newlib	OpenLibm	Musl	Apple	CUDA
version	2.34	2021.2.0	3.7	4.1.0	0.7.5	1.2.2	11.3.1	11.2.2
acos	0.899	**0.528**	0.669	0.899	0.918	0.918	0.634	1.69
acosh	2.01	**0.501**	0.504	2.01	2.01	NaN	0.502	2.18
asin	0.898	**0.528**	0.861	0.926	0.743	0.743	0.634	2.05
asinh	1.78	0.527	0.518	1.78	1.78	1.78	**0.515**	1.78
atan	0.853	0.541	**0.501**	0.853	0.853	0.853	0.722	2.06
atanh	1.73	0.507	**0.506**	1.73	1.73	1.73	0.511	3.16
cbrt	0.969	0.520	0.548	3.56	**0.500**	**0.500**	**0.500**	1.17
cos	0.561	0.548	0.530	2.91	**0.501**	**0.501**	0.862	1.52
cosh	1.89	0.506	∞	2.51	1.36	1.03	0.589	2.34
erf	0.968	0.507	0.968	0.968	0.943	0.968	**0.501**	1.14
erfc	3.13	**0.502**	3.13	63.9	3.17	3.13	0.750	4.49
exp	**0.502**	0.506	1.00	0.911	0.911	**0.502**	0.576	1.94
exp10	**0.502**	0.507	1.00	1.06	NA	3.88	0.580	2.07
exp2	0.502	0.519	1.00	1.02	**0.501**	0.502	0.570	2.39
expm1	0.813	0.544	**0.537**	0.813	0.813	0.813	0.687	1.45
j0	6.18e6	**0.678**	4.77e9	6.18e6	3.66e6	3.66e6	NA	3.78e10
j1	9.00	**1.69**	7.15e8	1.68e7	2.25e6	2.25e6	NA	7.48e9
lgamma	6.78	0.510	6.78	7.50e6	7.50e6	7.50e6	**0.501**	1.35e7
log	0.818	0.524	0.940	0.888	0.888	0.818	**0.511**	0.865
log10	2.07	0.516	0.626	2.10	0.832	0.832	**0.502**	2.09
log1p	1.30	0.525	**0.504**	1.30	0.839	0.835	0.513	0.887
log2	0.752	0.508	0.586	1.65	0.865	0.752	**0.502**	0.919
sin	0.561	0.546	0.530	1.37	**0.501**	**0.501**	0.846	1.50
sinh	1.89	0.538	**0.501**	2.51	1.83	NaN	0.601	2.94
sqrt	**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	3.10
tanh	2.19	**0.514**	1.27	2.19	2.19	2.19	0.817	1.82
tgamma	7.91	0.510	7.91	239.	**0.501**	**0.501**	**0.501**	11.5
y0	8.98	**3.40**	1.52e10	4.84e6	4.84e6	4.84e6	NA	2.36e10
y1	9.00	**2.07**	4.65e8	6.18e6	4.17e6	3.66e6	NA	4.96e10
atan2	1.52	**0.577**	0.584	1.52	1.55	1.55	0.722	2.18
hypot	**0.500**	**0.500**	**0.500**	1.21	1.21	0.927	**0.500**	1.03
pow	0.817	**0.515**	3.65	169.	6.88e10	0.817	**0.515**	10.3

Table 1: Single precision: maximal value of e (for univariate functions), and largest *known* value of e (for bivariate functions). IML is Intel Math Library (the version is that of the Intel C compiler).

	GNU libc 2.34			icx 2021.2.0		
function	$E \geq 1$	$E \geq 2$	max e	$E \geq 1$	$E \geq 2$	max e
acos	5422146	0	0.899	66001	0	0.528
acosh	243413455	2698	2.01	283	0	0.501
asin	4581700	0	0.898	470024	0	0.528
asinh	619608176	2748	1.78	963558	0	0.527
atan	21089464	0	0.853	505178	0	0.541
atanh	52062348	5790	1.73	240154	0	0.507
cbrt	453492162	0	0.969	15275450	0	0.520
cos	28209642	0	0.561	15732888	0	0.548
cosh	17868534	3558	1.89	139708	0	0.506
erf	126805016	0	0.968	908674	0	0.507
erfc	20494449	302363	3.13	1761	0	0.502
exp	170648	0	0.502	250299	0	0.506
exp10	169838	0	0.502	386017	0	0.507
exp2	168362	0	0.502	717434	0	0.519
expm1	12920601	0	0.813	539655	0	0.544
j0	1321697598	259702448	9.01e5	11960	0	0.678
j1	1323780342	259270196	9.00	8400236	2	1.69
lgamma	500354453	8246657	6.78	100287	0	0.510
log	416908	0	0.818	1060	0	0.524
log10	29787060	62225	2.07	151499	0	0.516
log1p	11534111	0	1.30	254793	0	0.525
log2	313550	0	0.752	276	0	0.508
sin	29362812	0	0.561	12374252	0	0.546
sinh	71328448	34776	1.89	247226	0	0.538
sqrt	0	0	0.500	0	0	0.500
tan	83411250	0	1.48	694770	0	0.520
tanh	118674314	729782	2.19	164068	0	0.514
tgamma	209259574	20924067	7.91	3971282	0	0.510
y0	1295653875	179694586	8.98	6785	1	3.40
y1	1178850345	128483641	9.00	10384	1	2.07
	x	y	e	x	y	e
atan2	`-0x1.f9cf48p+49`	`0x1.f60598p+51`	1.52	`-0x1.ff67c2p+57`	`0x1.ff7f62p+60`	0.577
hypot	`1.3ac98p+67`	`-1.ba5ec2p+77`	0.500	`1.3ac98p+67`	`-1.ba5ec2p+77`	0.500
pow	`1.025736p+0`	`1.309f94p+13`	0.817	`0x1.fe7782p-1`	`-0x1.c361cap+14`	0.515

Table 2: Single precision: GNU libc and Intel Math Library.

OpenLibm, Musl and the Apple library. The Intel and Apple libraries give in general more accurate results.

The `j1`, `y0`, and `y1` functions give large errors for all libraries where there are available, except for the Intel Math Library and the GNU libc, similarly for `j0` which yields large errors except for the Intel Math Library.

Notes about AMD LibM. We noticed a regression in AMD LibM 3.7: for $x =$ `-0x1.6cp+20`, `coshf` returns -Inf in AMD LibM 3.7, whereas it returned +Inf in AMD LibM 3.5 and 3.6 (where `coshf` was correctly rounded). The maximal error for `exp` is 1.00 since for $x =$ `-0x1.9d1da2p+6`, it yields 0 instead of the smallest subnormal 2^{-149}, where $\exp(x)$ is slightly smaller than the smallest subnormal. The same issue arises with `exp2` and $x =$ `-0x1.2a0002p+7` and with `exp10` and $x =$ `-0x1.66d3eap+5`.

Notes about Newlib. We used the `lgammaf_r` function, since we were unable to compile the `lgammaf` function (with lgamma Newlib says `undefined reference to '_impure_ptr'`). While this article only considers rounding to nearest, we noticed that Newlib 4.1.0 `sqrtf` function was not correctly rounded for directed rounding modes.

Notes about OpenLibm. For `x=0x1.ffffecp-1` and `y=-0x1.000002p+27`, the `powf` function yields `+Inf`, whereas the correct result is `0x1.557a86p115`.

Notes about Musl. For `x=-0x1.1e6ae8p+5`, the `acosh` function returns `-0x1.2f63acp+3` instead of the expected NaN value, which explains NaN in the max e column. A similar issue happens for `sinh` and `x=0x1.62e4p+6`, where it returns NaN instead of `0x1.ffe808p+126`. Those issues were fixed by Szabolcs Nagy in February 2021, but no new Musl release was published since that time.

Notes about the Apple Math Library. The `erf`, `lgamma` and `tgamma` 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`.

2.2 Bivariate Functions

For 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 at the end of Tables 2, 3, 4, and 5, which are *lower bounds* for the maximal error. For the `hypot` function, we found no incorrectly-rounded result for GNU libc, IML, AMD Libm and Apple Libm (but there might be).

3 Double Precision

For double precision it is not possible to perfom an exhaustive search with academic resources. We thus designed a black-box algorithm that tries to find large errors. (We did not want to analyze the code of each library, since this approach would need more human work, and requires to start again

	AMD LibM 3.7			RedHat Newlib 4.1.0		
function	$E \geq 1$	$E \geq 2$	max e	$E \geq 1$	$E \geq 2$	max e
acos	707885	0	0.669	5422146	0	0.899
acosh	21852	0	0.504	244658623	2698	2.01
asin	2454466	0	0.861	2358230	0	0.926
asinh	185582	0	0.518	542122908	2748	1.78
atan	3534	0	0.501	6406812	0	0.853
atanh	50260	0	0.506	52062348	5790	1.73
cbrt	10626352	0	0.548	1799139486	116334632	3.56
cos	2876076	0	0.530	209833072	6	2.91
cosh	1036795883	1020078851	∞	23905668	7706	2.51
erf	126805016	0	0.968	126741900	0	0.968
erfc	20494449	302363	3.13	21247299	1131209	63.9
exp	114132	0	1.00	17982847	0	0.911
exp10	102461	0	1.00	18423203	0	1.06
exp2	86902	0	1.00	18401203	0	1.02
expm1	102330	0	0.537	12920601	0	0.813
j0	1353797232	310998452	4.77e9	1338235574	279528826	6.18e6
j1	1369557306	337817680	7.15e8	1818091384	1376362116	1.68e7
lgamma	500354453	8246657	6.78	510903809	13277834	7.50e6
log	72371093	0	0.940	13363494	0	0.888
log10	2418509	0	0.626	30061115	91958	2.10
log1p	73898	0	0.504	11534111	0	1.30
log2	179825	0	0.586	602745869	258	1.65
sin	2866930	0	0.530	206155238	0	1.37
sinh	2	0	0.501	74587762	38924	2.51
sqrt	0	0	0.500	0	0	0.500
tan	529444	0	0.509	83455936	32	3.48
tanh	4314486	0	1.27	118674314	729782	2.19
tgamma	209259574	20924067	7.91	2028164922	1833526367	239.
y0	1314115311	207848357	1.52e10	1306144386	191859954	4.84e6
y1	1213975420	177569092	4.65e8	1201178797	153321647	6.18e6
	x	y	e	x	y	e
atan2	`1.fffe24p+59`	`1.000adcp+73`	0.584	`-0x1.f9cf48p+49`	`0x1.f60598p+51`	1.52
hypot	`1.3ac98p+67`	`-1.ba5ec2p+77`	0.500	`-0x1.6b05c4p-127`	`0x1.6b3146p-126`	1.21
pow	`-0x1.007fdcp+0`	`0x1.60d4p+15`	3.65	`0x1.d55902p-1`	`-0x1.fe037ep+9`	169.

Table 3: Single precision: AMD LibM and RedHat Newlib.

	OpenLibm 0.7.5			Musl 1.2.2		
function	$E \geq 1$	$E \geq 2$	max e	$E \geq 1$	$E \geq 2$	max e
acos	5717768	0	0.918	1700216587	0	0.918
acosh	244658828	2698	2.01	319260148	23345165	NaN
asin	4220748	0	0.743	4220748	0	0.743
asinh	542176908	2748	1.78	642880516	2730	1.78
atan	6483278	0	0.853	1717759310	0	0.853
atanh	52089660	5790	1.73	52062556	5740	1.73
cbrt	0	0	0.500	0	0	0.500
cos	647594	0	0.501	647594	0	0.501
cosh	23865830	0	1.36	16675588	0	1.03
erf	126619324	0	0.943	127569522	0	0.968
erfc	24416748	343931	3.17	19695704	302363	3.13
exp	19194854	0	0.911	170646	0	0.502
exp10	NA	NA	NA	41421106	3446689	3.88
exp2	102250	0	0.501	168362	0	0.502
expm1	12920593	0	0.813	12920592	0	0.813
j0	1332943954	268167202	3.66e6	1422932510	271739106	3.66e6
j1	1337823280	270477302	2.25e6	1320601392	117301706	2.25e6
lgamma	508702215	10980627	7.50e6	504159259	10758067	7.50e6
log	13361747	0	0.888	416908	0	0.818
log10	12305116	0	0.832	12305116	0	0.832
log1p	11588705	0	0.839	11678873	0	0.835
log2	11476491	0	0.865	313550	0	0.752
sin	625106	0	0.501	625106	0	0.501
sinh	72347778	31216	1.83	72812234	31516	NaN
sqrt	0	0	0.500	0	0	0.500
tan	303818252	0	0.800	303818252	0	0.800
tanh	70733480	729768	2.19	112377586	290564	2.19
tgamma	2	0	0.501	3	0	0.501
y0	1303825112	186524308	4.84e6	1309884649	190741595	4.84e6
y1	1198288693	144090005	4.17e6	1171308902	67527225	3.66e6
	x	y	e	x	y	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.ffffecp-1`	`-0x1.000002p+27`	6.88e10	`1.025736p+0`	`1.309f94p+13`	0.817

Table 4: Single precision: OpenLibm and Musl.

	Apple 11.3.1			CUDA 11.2.2		
function	$E \geq 1$	$E \geq 2$	max e	$E \geq 1$	$E \geq 2$	max e
acos	475118	0	0.634	162888432	1461	1.69
acosh	6196	0	0.502	243330719	4123	2.18
asin	581248	0	0.634	21377944	358624	2.05
asinh	369750	0	0.515	151988118	5660	1.78
atan	1726005038	0	0.722	181296102	101660	2.06
atanh	92652	0	0.511	190056816	7899670	3.16
cbrt	0	0	0.500	551571624	0	1.17
cos	359705126	0	0.862	429339510	8	1.52
cosh	899792	0	0.589	65403872	286368	2.34
erf	2	0	0.501	364685356	0	1.14
erfc	51477493	0	0.750	230105768	23856924	4.49
exp	1693613	0	0.576	159912805	95287	1.94
exp10	1693714	0	0.580	162340068	104557	2.07
exp2	1645495	0	0.570	189192458	983637	2.39
expm1	2978469	0	0.687	26510788	0	1.45
j0	NA	NA	NA	3530150566	893872806	3.78e10
j1	NA	NA	NA	1743976120	871824990	7.48e9
lgamma	3	0	0.501	705883207	49167626	1.35e7
log	874	0	0.511	72516687	0	0.865
log10	511	0	0.502	763162986	10416179	2.09
log1p	110493	0	0.513	42499890	0	0.887
log2	604	0	0.502	12044078	0	0.919
sin	358753536	0	0.846	426015576	0	1.50
sinh	942638	0	0.601	38397002	411336	2.94
sqrt	0	0	0.500	0	0	0.500
tan	164196812	0	0.746	836917130	30145620	3.10
tanh	5804012	0	0.817	39501004	948272	1.82
tgamma	3	0	0.501	316531085	15827516	11.5
y0	NA	NA	NA	1630315303	629603489	2.36e10
y1	NA	NA	NA	1403359302	603385922	4.96e10
	x	y	e	x	y	e
atan2	`-0x1.ce62cep-116`	`0x1.cbf9bp-113`	0.722	`0x1.a59982p-74`	`0x1.915c9p-74`	2.18
hypot	`0x1.3ac98p+67`	`-0x1.ba5ec2p+77`	0.500	`0x1.007594p+1`	`-0x1.003512p+1`	1.03
pow	`0x1.034016p+0`	`0x1.b782b4p+12`	0.515	`0x1.714882p-1`	`-0x1.0f68b4p+8`	10.3

Table 5: Single precision: Apple and CUDA.

from scratch for each new version of the library.) Therefore, the values in the double-precision tables are only lower bounds of the maximal error.

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_uint64` 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_uint64(i)` for $a \leq i < b$. 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.

If $b - a \geq t$, we subdivide the interval $[a, b]$ into two equal intervals, in each interval 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 recursive algorithm would perform more function evaluations than trying all the values in the current interval.

In practice we used 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 those intervals, which yields 40 smaller intervals (or 80 for bivariate functions), and keep again the 20 better 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 maximal ulp-error. A second strategy keeps the sub-interval with the maximal *average* ulp-error (considering only inputs which yield a non-zero ulp-error, i.e., 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 [19]. In practice we found the first strategy to be more effective, with the second and third strategies finding sometimes larger maximal 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 those 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 6 summarizes the maximal known errors found using the above algorithm, for example the 0.531 entry for `acos` and icx means that for all inputs tried by the above algorithm, the ulp-error e 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 smallest maximal error (which should not be taken as an upper bound, since the search is not exhaustive). Detailed tables (Tables 7, 8, 9 and 10) give the input values (in hexadecimal) yielding the corresponding ulp-error e, which enables the reader to reproduce our results.

Like for single precision, the Intel Math Library gives the best results in most cases (for 22 of the 30 univariate functions, and for the `hypot` function). 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 those functions use the `rsqrt` instructions, which is known to be more accurate on AMD hardware [3]. The square root function seems to be correctly rounded for all libraries, as required by IEEE 754. Large errors occur for the AMD `acosh`, `atanh` and `log1p` functions, for the `j0`, `j1`, `y0` and `y1` functions for all libraries except the Intel Math Library, for the `lgamma` function from Newlib, OpenLibm, Musl, the Apple and CUDA libraries, for the `tgamma` function from Newlib, OpenLibm and the Apple library, and for the power function from AMD LibM, Newlib and OpenLibm.

4 Double Extended Precision

This format corresponds to the C type `long double` on `x86_64` processors. The results are summarized in Table 11, and detailed in Tables 12 and 13. We see that in this format, the Intel Math library is better than all other libraries for all functions, both univariate and bivariate.

For the Intel compiler, the j0, j1, y0, and y1 functions call the corresponding quadruple precision function, which explains why the maximal error is 0.5 ulp in our experiments[2] (assuming the quadruple precision functions are correctly rounded, an incorrect rounding can only occur when the last $113 - 64 = 49$ bits are exactly 100...000, which occurs with probability 2^{-49}). AMD Libm does not provide long double functions. 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 4.1.0, the `hypotl` function does not work properly: for $x \geq 2^{8192}$, the call `hypotl(x,0)` gives infinity. OpenLibm does not provide the following long double functions: `exp10`, `j0`, `j1`, `y0` and `y1`, its `powl` does not seem to be thread-safe, its `tgammal` function yields `+Inf` for `x=-0xd.b6e8f5c28f5c29p+7` instead of `0xe.deaa8ed2a29cp-16396`. Musl does not provide `j0`, `j1`, `y0`, and `y1` either.

The Apple Darwin ABI for ARM processors maps the C long double type to double, thus there is no real "double extended" format.

[2]Except for j0 where we found an input that is not correctly rounded.

library	GNU libc	IML	AMD LibM	Newlib	OpenLibm	Musl	Apple	CUDA
version	2.34	icx 2021.2.0	3.7	4.1.0	0.7.5	1.2.2	11.3.1	11.2.2
acos	**0.523**	0.531	0.938	0.930	0.930	0.930	1.06	1.53
acosh	2.25	**0.509**	3.35e7	2.25	2.25	2.25	2.25	2.48
asin	**0.516**	0.531	1.06	0.981	0.981	0.981	0.746	1.99
asinh	1.92	**0.507**	1.28	1.92	1.92	1.92	1.58	2.48
atan	**0.523**	0.528	0.864	0.861	0.861	0.861	0.869	1.75
atanh	1.81	**0.507**	1.67e7	1.81	1.81	1.80	2.01	2.49
cbrt	3.67	0.523	0.502	0.670	0.668	0.668	0.729	**0.501**
cos	**0.516**	0.518	0.800	0.887	0.834	0.834	0.947	1.51
cosh	1.93	**0.516**	1.93	2.67	1.47	1.04	0.523	1.40
erf	1.43	**0.507**	1.43	1.02	1.02	1.02	6.22	1.48
erfc	5.12	**0.505**	5.12	4.04	4.04	3.72	10.7	4.42
exp	**0.511**	0.530	0.756	0.949	0.949	**0.511**	0.521	0.904
exp10	2.01	**0.538**	0.769	0.896	NA	4.14	0.520	1.10
exp2	**0.511**	0.535	0.775	0.895	0.751	**0.511**	0.520	0.943
expm1	0.914	**0.512**	0.724	0.907	0.907	0.907	0.706	1.18
j0	4.51e14	**0.600**	4.51e14	9.01e15	4.51e14	4.51e14	4.51e14	3.36e19
j1	4.47e14	**0.615**	4.47e14	9.01e15	1.10e15	1.10e15	1.10e15	2.19e19
lgamma	11.0	**0.515**	11.0	4.45e15	4.45e15	4.45e15	2.33e16	5.11e15
log	0.520	**0.518**	0.578	0.944	0.944	0.520	0.508	0.563
log10	1.62	**0.532**	0.634	2.08	0.813	0.813	0.514	1.42
log1p	0.903	**0.521**	2.52e8	0.895	0.895	0.900	0.667	1.50
log2	0.554	**0.504**	0.617	2.06	0.921	0.554	0.515	1.29
sin	**0.516**	0.518	0.800	0.888	0.831	0.831	0.944	1.51
sinh	1.93	**0.521**	1.53	2.67	1.88	1.88	0.538	1.51
sqrt	**0.500**	**0.500**	**0.500**	**0.500**	**0.500**	**0.500**	**0.500**	**0.500**
tan	0.619	**0.550**	1.40	1.02	1.02	1.02	3.53	2.07
tanh	2.22	**0.555**	1.47	2.22	2.22	2.22	0.613	1.48
tgamma	8.85	**0.518**	10.3	2.27e3	1.03e3	15.3	9.01e15	9.62
y0	5.93e15	**1.14**	5.93e15	1.42e15	1.42e15	1.42e15	1.42e15	2.99e19
y1	5.56e15	**1.25**	5.56e15	5.56e15	5.56e15	5.56e15	5.56e15	5.85e19
atan2	**0.524**	0.550	0.750	1.55	1.55	1.55	0.747	1.76
hypot	0.987	**0.751**	0.942	1.21	1.21	1.04	1.21	1.88
pow	**0.523**	1.73	926.	636.	636.	0.525	0.757	1.40

Table 6: Double precision: Maximal known error.

function	GNU libc 2.34 x	max e	icx 2021.2.0 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	-0x1.00064c7d4d16ep-4	0.507
atan	-0x1.f90219fb2af9ap-4	0.523	-0x1.ffff8020d3d1dp-7	0.528
atanh	-0x1.f97fabc0650c4p-4	1.81	-0x1.e2cfb2667f17ep-9	0.507
cbrt	0x1.7a13d2b82be1ap-254	3.67	-0x1.f7a4b333bb2adp+20	0.523
cos	0x1.1feeccb9e4bf7p+5	0.516	-0x1.d19ebc5567dcdp+311	0.518
cosh	-0x1.633c654fee2bap+9	1.93	-0x1.5a364e6b98134p+9	0.516
erf	0x1.c332bde7ca515p-5	1.43	0x1.00b4cd58903b2p+2	0.507
erfc	0x1.3feefa52464d8p+0	5.12	0x1.5d164509e8235p-1	0.505
exp	-0x1.571eb1496dab3p+9	0.511	0x1.fce66609f7428p+5	0.530
exp10	0x1.334ab33a9aaep-2	2.01	-0x1.5cd9d94d49a85p+1	0.538
exp2	-0x1.1b4f4659c5c9ep-5	0.511	0x1.f3ffd85f33423p-1	0.535
expm1	0x1.62f69d171fa65p-2	0.914	-0x1.635e445cc416bp-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.f60d7b63e60d9p+1	11.0	-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	-0x1.2c10396268852p-2	0.903	0x1.000aee2a2757fp-9	0.521
log2	0x1.0b524a705b134p+0	0.554	0x1.00b0d7b252144p+0	0.504
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.c673a473b3503p+3	0.619	0x1.49adfd996a81dp+18	0.550
tanh	-0x1.e134557098e37p-3	2.22	-0x1.0018308fc500dp+0	0.555
tgamma	-0x1.70578d333203ap+4	8.85	-0x1.3e002bee87875p+6	0.518
y0	0x1.c982eb8d417eap-1	5.93e15	0x1.f78ea64a68b96p-63	1.14
y1	0x1.193bed4dff243p+1	5.56e15	0x1.c50658fc9bc2dp+0	1.25
atan2	0x1.ed6060626eecfp-429 0x1.f42ebb62994dcp-426	0.524	0x1.e4b0731c7640dp-611 0x1.fc94bba330aedp-606	0.550
hypot	-0x0.5a934b7eac967p-1022 -0x0.b5265a7e06b82p-1022	0.987	0x0.89c127b351f2bp-1022 0x0.4fd9b86bb4fe8p-1022	0.751
pow	0x1.010e2e7ee71aep+0 0x1.44bf0047427f6p+17	0.523	0x1.fffff9c61ce4p-1 0x1.c4e304ed4c734p+31	1.73

Table 7: Double precision: GNU libc and Intel Math Library.

function	AMD LibM 3.7 x	max e	RedHat Newlib 4.1.0 x	max e
acos	-0x1.01c3d975759c3p-1	0.938	-0x1.0068b067c6feep-1	0.930
acosh	0x1.40c044a37af12p+0	3.35e7	0x1.0001ff6afc4bap+0	2.25
asin	-0x1.0181c4fd1ff3fp-1	1.06	-0x1.004d1c5a9400bp-1	0.981
asinh	-0x1.00238008c0c6fp+0	1.28	-0x1.02657ff36d5f3p-2	1.92
atan	0x1.601c5ffc19b43p-1	0.864	0x1.62ff6a1682c25p-1	0.861
atanh	-0x1.34124dadbcda1p-1	1.67e7	-0x1.f97fabc0650c4p-4	1.81
cbrt	0x1.09806cdccbfa1p-748	0.502	-0x1.00a7f4c3c1deep-885	0.670
cos	0x1.293f72677a7e7p+13	0.800	-0x1.4ae182c1ab422p+21	0.886
cosh	0x1.1ffff6b05a77fp+4	1.93	0x1.633cc2ae1c934p+9	2.67
erf	0x1.c332bde7ca515p-5	1.43	-0x1.c57541b55c8ebp-16	1.02
erfc	0x1.3feefa52464d8p+0	5.12	-1 0x1.531fe30327333p+0	4.04
exp	-0x1.6237c669d1a9fp+9	0.756	0x1.2e8f20cf3cbe7p+8	0.949
exp10	-0x1.33b58776304ebp+8	0.769	0x1.3450246086766p-3	0.896
exp2	-0x1.ff03ffe8ed867p+9	0.775	-0x1.fff1e028e2058p-2	0.895
expm1	0x1.9a579b4e7005fp-2	0.724	0x1.63a87a0bce7dbp-2	0.906
j0	0x1.33d152e971b4p+1	4.51e14	-0x1.45f306d16c7cap+915	9.01e15
j1	-0x1.ea75575af6f09p+1	4.47e14	0x1.45f3066f80258p+325	9.01e15
lgamma	-0x1.f60d7b63e60d9p+1	11.0	-0x1.3a7fc9600f86cp+1	4.45e15
log	0x1.0ffda7808ae17p+0	0.578	0x1.48a807fa56469p+0	0.944
log10	0x1.e0030380b742fp-1	0.634	0x1.55109a890357dp+0	2.08
log1p	0x1.10730dfffffffp-4	2.52e8	-0x1.2bef5ca35157ap-2	0.895
log2	0x1.e007c3c7bee53p-1	0.617	0x1.68d778f076021p+0	2.06
sin	0x1.59c91f12040dep+5	0.801	-0x1.842d8ec8f752fp+21	0.888
sinh	0x1.1ff604247b163p+3	1.53	-0x1.633cae1335f26p+9	2.67
sqrt	0x1.fffffffffffffp-1	0.500	0x1.fffffffffffffp-1	0.500
tan	-0x1.6842486cdd221p+12	1.40	0x1.3f9605aaeb51bp+21	1.02
tanh	-0x1.ff58c8cce5385p-1	1.47	-0x1.e134557098e37p-3	2.22
tgamma	-0x1.c033cc426752fp+2	10.3	-0x1.535175475cc8dp+7	2.27e3
y0	0x1.c982eb8d417eap-1	5.93e15	0x1.c982eb8d417eap-1	1.42e15
y1	0x1.193bed4dff243p+1	5.56e15	0x1.193bed4dff243p+1	5.56e15
atan2	0x1.ccbbbcfef3c02p-523 0x1.924bf639c1a94p+500	0.750	-0x1.358bb5eb25bdcp+813 0x1.2f86b82481a0ap+815	1.55
hypot	0x1.95678c86e5d12p-189 -0x1.2bfba48a14a49p-188	0.942	0x1.6a0a41410b1abp-1004 -0x0.a24afe71b539fp-1022	1.21
pow	0x1.fffffca9dd1a7p-1 0x1.344c8c25b78dfp+32	926.	0x1.000002c5e2e99p+0 0x1.c9eee35374af6p+31	636.

Table 8: Double precision: AMD LibM and RedHat Newlib.

function	OpenLibm 0.7.5 x	max e	Musl 1.2.2 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	0x1.502bacfb2c9ap+0	1.04
erf	-0x1.c57541b55c8ebp-16	1.02	-0x1.c57541b55c8ebp-16	1.02
erfc	0x1.531fe30327333p+0	4.04	0x1.527f4fb0d9331p+0	3.72
exp	0x1.6f5ea0e012c38p+6	0.948	-0x1.18209ecd19a8cp+6	0.511
exp10	NA	NA	-0x1.fe8c27141c94ap+3	4.14
exp2	-0x1.ff1eb5acee46bp+9	0.751	-0x1.1b4f4659c5c9ep-5	0.511
expm1	0x1.63a87a0bce7dbp-2	0.907	0x1.63a87a0bce7dbp-2	0.907
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	0x1.48a807fa56469p+0	0.944	0x1.dc0b612d39136p-1	0.520
log10	0x1.5536e3c4fc059p+0	0.813	0x1.5536e3c4fc059p+0	0.813
log1p	-0x1.2bef5ca35157ap-2	0.895	-0x1.2bf32aaf122e2p-2	0.900
log2	0x1.67eaf07ce24d1p+0	0.921	0x1.0b524a705b134p+0	0.554
sin	0x1.b8b6d07237443p+21	0.831	0x1.b8b6d07237443p+21	0.831
sinh	-0x1.63218f059a8f1p-1	1.88	0x1.6320943636f24p-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.ff2496d78b383p+2	15.6
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 9: Double precision: OpenLibm and Musl.

function	Apple 11.3.1 x	max e	CUDA 11.2.2 x	max e
acos	-0x1.8d313198a2e02p-53	1.06	0x1.26664e9c7c4fcp-1	1.53
acosh	0x1.00007fb3703ddp+0	2.25	0x1.1d7183dc5c72p+0	2.48
asin	0x1.eaeb8b58c0655p-2	0.746	-0x1.2e8209031f6a1p-1	1.99
asinh	-0x1.fdefd03df4cd7p-3	1.58	0x1.0ab7ab2dd559ap-1	2.48
atan	0x1.13af5bed374ep+1	0.869	-0x1.535197a4ac8fcp+0	1.75
atanh	0x1.ffd834a270fp-10	2.01	0x1.f55ef33b30c76p-3	2.49
cbrt	0x1.facdbe3653d0ep-1	0.729	-0x1.4082043b4a743p+238	0.50
cos	0x1.2f29e9e42f109p+7	0.947	0x1.e419decf094c5p+28	1.51
cosh	-0x1.62dabf317e5bbp-2	0.523	-0x1.e800c88ccc189p+1	1.40
erf	-0x1.e057e942cd2b8p-2	6.22	-0x1.32a154fb79376p-2	1.48
erfc	0x1.bba14dc3507ccp+1	10.7	0x1.1523c9af3d2bbp-6	4.42
exp	-0x1.94e37f85b6199p-11	0.521	0x1.e7fe6bd658aa2p+1	0.904
exp10	-0x1.eba008abaa79dp-14	0.520	0x1.5c0db5e8019c2p+0	1.10
exp2	-0x1.8457c9f4p-31	0.520	-0x1.ff3daa34e82bp+9	0.943
expm1	0x1.e7f93188565ecp-5	0.706	0x1.29728a0720ca1p+5	1.18
j0	0x1.33d152e971b4p+1	4.51e14	-0x1.277684658e8f8p+25	3.36e19
j1	-0x1.ea75575af6f09p+1	1.10e15	-0x1.0226cad652692p+26	2.19e19
lgamma	-0x1.bffcbf76b86fp+2	2.33e16	-0x1.fa471547c2fe5p+1	5.11e15
log	0x1.490af72a25a81p-1	0.508	0x1.6a010f195ab29p-1	0.563
log10	0x1.2501ee5628b08p-1	0.514	0x1.808c5c181ad9bp-1	1.42
log1p	-0x1.ffffff3fffffdp-28	0.667	-0x1.ffffffbaefe27p-2	1.50
log2	0x1.6b0163c4e8596p-1	0.515	0x1.68eddbcbdeb01p+0	1.29
sin	-0x1.07e4c92b5349dp+4	0.944	0x1.cfa11e299c922p+16	1.51
sinh	0x1.d7131e5c70263p-2	0.538	0x1.bda830d0e2c74p+0	1.51
sqrt	0x1.fffffffffffffp-1	0.500	0x1.fffffffffffffp-1	0.500
tan	-0x1.a81d84a1133b6p+6—	3.53	0x1.002fe489734eep+12	2.07
tanh	0x1.00cf9f273d84p+1	0.613	-0x1.1931761a1bbe8p-1	1.48
tgamma	-0x1.55f5b67b0b1edp+7	9.01e15	-0x1.2ba89f2297cf1p+7	9.62
y0	0x1.c982eb8d417eap-1	1.42e15	0x1.3f5cf973a65c9p+25	2.99e19
y1	0x1.193bed4dff243p+1	5.56e15	0x1.512c1dcdad492p+26	5.85e19
atan2	-0x1.6a539153430d8p-416 0x1.d2b5b9dc716d8p-415	0.747	0x1.9cde4ff190e45p+931 0x1.37d91467e558bp+931	1.76
hypot	0x1.2719987d4c1fep-1023 -0x1.6a09f20598aa9p-1004	1.21	-0x1.587b2ce3e8f17p+33 0x1.7a8b3312418c8p+33	1.88
pow	0x1.30163fad53c3ep-592 0x1.ba23601ed047ep+0	0.757	0x1.342ee652dbe58p-833 0x1.fe8471dcdf40dp-8	1.40

Table 10: Double precision: Apple and CUDA.

library	GNU libc	Intel Math Library	OpenLibm	Musl
version	2.34	icx 2021.2.0	0.7.5	1.2.2
acos	1.75	**0.505**	0.938	1.75
acosh	2.99	**0.502**	3.14	2.99
asin	1.15	**0.506**	1.03	2.00
asinh	2.96	**0.506**	3.19	2.96
atan	0.640	**0.501**	1.10	0.640
atanh	2.88	**0.501**	85.4	3.19
cbrt	0.824	**0.503**	0.890	0.890
cos	1.51	**0.502**	0.799	0.798
cosh	3.40	**0.502**	4.86	3.73
erf	1.16	**0.517**	1.16	1.16
erfc	4.68	**0.526**	5.67	5.11
exp	1.27	**0.501**	2.00	1.54
exp10	1.50	**0.501**	NA	40.1
exp2	0.788	**0.501**	2.18	0.788
expm1	3.08	**0.502**	1.94	9.71e3
j0	9.79e17	**0.501**	NA	NA
j1	3.38e18	**0.500**	NA	NA
lgamma	12.1	**0.548**	9.08e19	9.08e19
log	0.998	**0.501**	1.22	0.998
log10	1.36	**0.502**	1.21	1.36
log1p	2.49	**0.501**	2.60	2.49
log2	0.995	**0.502**	1.64	0.995
sin	1.51	**0.502**	0.798	0.798
sinh	3.39	**0.503**	4.85	9.71e3
sqrt	**0.500**	**0.500**	**0.500**	**0.500**
tan	1.75	**0.504**	1.01	1.02
tanh	3.22	**0.506**	2.56	2.95
tgamma	9.77	**0.554**	Inf	3.69e19
y0	1.38e18	**0.500**	NA	NA
y1	4.61e18	**0.500**	NA	NA
atan2	0.751	**0.501**	1.69	0.751
hypot	0.981	**0.751**	0.981	1.08
pow	0.914	**0.501**	533.	533.

Table 11: Double extended precision: Maximal known error.

function	GNU libc 2.34 x	max e	icx 2021.2.0 x	max e
acos	0xf.fe0016ad10f90d9p-4l	1.75	0x8.af256cd27462348p-4l	0.505
acosh	0x1.1ecdb5b8f0c5d79p+0l	2.99	0x1.1f9c4feedfe4f2cp+0l	0.502
asin	0x8.171fd358c4cb27bp-4l	1.15	-0x8.018aef8787e5a6bp-4l	0.506
asinh	-0x8.0bb656992eac437p-4l	2.96	0x7.ff15da44c3651abp-4l	0.506
atan	-0x1.0411ae010d4c5b1ep+0l	0.640	-0x8.00f60592e42d79p+8l	0.501
atanh	-0x3.341a81bdff966918p-4l	2.88	0x3.e7be418257523408p-4l	0.501
cbrt	0xc.f5414f1e9486ffep+16084l	0.824	-0x2.320375fd33ed311cp-13376l	0.503
cos	-0x3.d067a048093bdf94p+9160l	1.51	-0x4.b0df0d1cf1e0f24p+8l	0.502
cosh	0x2.c5d375f827733ac4p+12l	3.40	-0x7.f6a09874512cf768p-4l	0.502
erf	0xd.7304333df36e803p-4l	1.16	-0x1.c5f27c68de6b5a76p-4l	0.517
erfc	0x1.5969338308783d16p+0l	4.68	0x2.f35504f2fa42e838p-4l	0.526
exp	0x5.8b910f170240d878p-4l	1.27	0x2.c590e6ab0d71c77p+12l	0.501
exp10	0x1.2e56ca82b0ab4a66p+12l	1.50	-0x1.2ab76ac25255a1aap+12l	0.501
exp2	-0x7.3f819acf048f1678p-4l	0.788	-0x3.ef2d27bf02a002bp-16l	0.501
expm1	0x5.8b9235e9c8acc8a8p-4l	3.08	-0x1.00400264d53fedf8p-8l	0.502
j0	-0x2.67a2a5d2e367f784p+0l	9.79e17	-0x1.6a09e667f3bd238cp-32l	0.501
j1	0x3.d4eaaeb5ede115p+0l	3.38e18	-0x1.8p-16444l	0.500
lgamma	-0x3.ec8f6f9589ea99f4p+0l	12.1	-0x4.088ad64714fd4768p+0l	0.548
log	0x1.20db2d121134e12ep+0l	0.998	0x1.0ffd0d3bcf067118p+0l	0.501
log10	0x1.272b71572c9fc862p+0l	1.36	0x1.01002619de32e7cp+0l	0.502
log1p	-0x6.451f6c3fd0d4a218p-4l	2.49	-0xe.fefa23913fa3eb7p-8l	0.501
log2	0x1.05980f1e8cabcfbp+0l	0.995	0x1.01004edb8a2eb6e8p+0l	0.502
sin	-0x6.e2368c0ed74e5698p+16l	1.51	-0xc.141cf155623856bp+8l	0.502
sinh	0x2.c5d376167f4052f4p+12l	3.39	0x7.b0af44fc25df3efp-4l	0.503
sqrt	0xf.ffffffffffffffp-4l	0.500	0xf.ffffffffffffffp-4l	0.500
tan	0x1.974ccdb290851e7cp+8l	1.75	0xc.845cb9b2c6a1d57p+0l	0.504
tanh	0x3.b9979a543d0fbfa8p-4l	3.22	0x7.fb80104d932bceep-4l	0.506
tgamma	-0x1.70a55b2628a7cb68p+4l	9.77	-0x6.9c7a8fb06c63eb5p+8l	0.554
y0	0xe.4c175c6a0bf51e8p-4l	1.38e18	0x1.8a4b874528c14f1cp+2652l	0.500
y1	0xb.bfc89c6a1903022p+0l	4.61e18	0xd.749961e354cf884p-4284l	0.500
atan2	-0x7.9301460b8463cbp+15368l 0xf.25cd5eb1280b4d1p+15372l	0.751	-0x5.c0c9cc5a59632f88p+16340l 0x5.db7810fba1ce4908p+16348l	0.501
hypot	0x1.73f339f61eda21dp-16384l 0x2.e45f9f9500877e2p-16384l	0.981	-0x3.00bad8a56d87a0cp-16384l -0xe.6d794db04791398p-16388l	0.751
pow	0x2.21dda4bcec55b158p-3616l 0x7.ef1ef5fbe3df50dp-16l	0.914	0xc.b80572af668bb57p+152l -0x6.8a6d3d7b442f3c18p+4l	0.501

Table 12: Double extended precision: GNU libc and Intel Math Library.

function	OpenLibm 0.7.5 x	max e	Musl 1.2.2 x	max e
acos	-0x8.040541d0054d89p-4l	0.938	0xf.fe002cabd608585p-4l	1.75
acosh	0x1.10384b24aec007fcp+0l	3.14	0x1.1ecdb5b8f0c5d79p+0l	2.99
asin	0x8.0519515d1e15a6bp-4l	1.03	-0x3.fff0a397b8dea17cp-8l	2.00
asinh	-0x5.c9866cb231f2c7c8p-4l	3.19	-0x8.0bb656992eac437p-4l	2.96
atan	0x6.fffde214a06fb5f8p-4l	1.10	-0x1.0411ae010d4c5b1ep+0l	0.640
atanh	-0xf.fffffffffffe78p-32l	85.4	-0x3.336379ca42d23c18p-4l	3.19
cbrt	-0x3.fffffffa5623708p+4588l	0.890	-0x3.fffffffa5623708p+4588l	0.890
cos	0x3.e0d9ca90829608dp+4l	0.799	0x3.e0d6be5b4e1853d4p+4l	0.798
cosh	0x2.c5d374f9436efd1p+12l	4.86	0x2.c5d37484e4c162bp+12l	3.73
erf	0xd.7304333df36e803p-4l	1.16	0xd.7304333df36e803p-4l	1.16
erfc	0x1.5cc0e11b5d9aa3e4p+0l	5.67	0x1.5b14b83148c0ce86p+0l	5.11
exp	0x8.aa2409947e8ea78p+0l	2.00	-0x2.c5a106889ba12ac4p+12l	1.54
exp10	NA	NA	0xd.41cfea4b9c78d61p+8l	40.1
exp2	-0xf.fffff464e1ae63ep-12l	2.18	-0x7.3f819acf048f1678p-4l	0.788
expm1	0x6.651489ac8f591f58p-4l	1.94	0x2.c5c85fdf170c604cp+12l	9.71e3
j0	NA	NA	NA	NA
j1	NA	NA	NA	NA
lgamma	-0x2.74ff92c01f0d82acp+0l	9.08e19	-0x2.74ff92c01f0d82acp+0l	9.08e19
log	0x1.672126d956ceeb06p+0l	1.22	0x1.20db1c276f1d2656p+0l	0.998
log10	0xb.fff4d3443e63bbap-4l	1.21	0x1.272b7c3bbb08ae12p+0l	1.36
log1p	-0x4.c669bd1813ec8bd8p-4l	2.60	-0x6.451f6c3fd0d4a218p-4l	2.49
log2	0x1.6646b082fd1065cep+0l	1.64	0x1.05980f1e8cabcfbp+0l	0.995
sin	-0x2.a2a4aca336af4538p+8l	0.798	-0x2.a2a4aca336af4538p+8l	0.798
sinh	-0x2.c5d375cbe7e4a81cp+12l	4.85	0x2.c5c85fdbc1ccc354p+12l	9.71e3
sqrt	0xf.ffffffffffffffp-4l	0.500	0xf.ffffffffffffffp-4l	0.500
tan	-0x6.fae45260a35437f8p+8l	1.01	-0x6.fae4525c1c348edp+8l	1.02
tanh	0x3.8b2602d43bdf4c28p-4l	2.56	0x4.024182351388d15p-4l	2.95
tgamma	-0x6.db747ae147ae148p+8l	Inf	-0x2.8d19fd20f3aa62cp+4l	3.69e19
y0	NA	NA	NA	NA
y1	NA	NA	NA	NA
atan2	0x3.d34c9d81dcd29354p+5568l 0xf.3afc4f6c9f5c4a2p+5568l	1.69	-0x7.9301460b8463cbp+15368l 0xf.25cd5eb1280b4d1p+15372l	0.751
hypot	0x1.73f339f61eda21dp-16384l 0x2.e45f9f9500877e2p-16384l	0.981	0x2.00007da75fd5903cp-8960l 0x2.d42207352184bff4p-8960l	1.08
pow	0xc.f620c9ea4p+16380l -0x4.0ffffcp-48l	533.	0xc.f620c9ea4p+16380l -0x4.0ffffcp-48l	533.

Table 13: Double extended precision: OpenLibm and Musl.

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 14, and detailed in Table 15. 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, except for `lgamma` and `tgamma`. Apart from those 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` and `y1`.

Acknowledgements. The authors thank Claude-Pierre Jeannerod and Vincent Lefèvre who helped to improve that article, Alexei Sibidanov who helped to compile Newlib, Eric Schneider and Nick Timmons 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. Access to the Intel C Compiler and thus to the Intel Math Library was possible thanks to the PlaFRIM experimental testbed, supported by Inria, CNRS (LABRI and IMB), Université de Bordeaux, Bordeaux INP and Conseil Régional d'Aquitaine (see `https://www.plafrim.fr/`). Experiments on GPU were performed on hardware made available by CERN.

References

[1] AMD LibM version 3.7. `https://developer.amd.com/amd-aocl/amd-math-library-libm/`, 2021.

[2] Apple Math Library (MacOS 11.3.1, Apple M1).

[3] 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.

[4] 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`.

[5] 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.

[6] CUDA C Programming Guide v11.3.1, Section G Mathematical Functions. `https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#mathematical-functions-appendix`, 2021.

[7] Cuda math library. `https://developer.nvidia.com/cuda-math-library`, 2021.

library version	GNU libc 2.34	Intel Math Library icx 2021.2.0
acos	1.28	**0.501**
acosh	4.00	**0.502**
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	1.41	**0.501**
erfc	4.34	**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	3.48	**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**
atan2	1.89	**0.501**
hypot	0.985	**0.501**
pow	30.2	**1.40**

Table 14: Quadruple precision: Maximal known error.

function	GNU libc 2.34 x	max e	icx 2021.2.0 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	0x3.7ffc1cc0aa51559b8de53438aacp-4	1.41	-0x1.15eb4e54ee6ca15bfaa70b134223p+0	0.501
atanh	0x2.c02a24f3472c7840afbd8cfb68bap-4	3.89	-0xd.9fe29c463116c87fa567e436489p-8	0.501
cbrt	-0xb.5096fcd4c900f68990fe999ba7fp+11884	0.736	-0x2.10d29fbb2036d1d7ffdd8bf63184p+10912	0.501
cos	0xe.6672d458b05edf50af4fab1a42p+40	1.52	-0x6.081f6e15f81d27ac2a6038eed3bp+2232	0.501
cosh	-0x2.c5d376eefc912b9e0d7a88c86b38p+12	1.92	-0x2.b927c1a77ac2aced38aad663b41ap+4	0.501
erf	0xd.f293d1740ef439a569f47bbb36dp-4	1.41	0x5.a5182e2e3fce6963a492839ebb3cp-8	0.501
erfc	0x1.5140ab37feea57de6fd09f1281d9p+0	4.34	0x6.0a5ca72c4efcc505d9b7cf9a88e8p+0	0.504
exp	-0x2.c5b323ac8f0d3eea10dbd13f0d22p+12	0.751	-0x5.6622c128e27c6a9113094ad0fd64p-8	0.501
exp10	0xe.72e681b1f4f21cfe05aac3578cb8p-4	2.00	0x1.1e2a2ef09a4f66e4ec2574b62c49p+12	0.501
exp2	0xf.ffffa0ed8d14e72c9a27c16c32c8p-4	1.08	-0x7.cac40d04ef369e25cd005dffe05p-8	0.501
expm1	0x5.a1f428076faa87bb8d8482af6ad4p-4	1.64	0x8.ca3ec0644d8a8b06d144ce6a3e7p+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.ec25f2bf2a4927e95ff5ea041c6ep+0	13.0	-0x3.24c1b793cb35efb8be699ad3d9bap+0	2.79e30
log	0xf.d016f49074a9c4fe793af2394278p-4	1.05	0x2.b77cdc74c184c83993bb7bca672p-4912	0.501
log10	0x1.6a291ea0aa11fb374f1df8b3ac6bp+0	2.01	0x1.9b621e77f3988205a03658827222p-12364	0.501
log1p	0x6.a2681ee1c8522a86c4c7cc3cca28p-4	3.48	-0x6.2611e37be5cf4388865319f859b4p-12	0.501
log2	0xb.54170d5cfa8fd72a47d6bda19068p-4	3.31	0xf.f63cee8e97ac6783532625273eap-4	0.501
sin	0x5.6a5005df151cc2274e115647e9acp+64	1.52	0x4.246e3c1f108f5c75d0f326fc622p+5604	0.501
sinh	0x6.7e79f3aada38698b910c300b19b8p-4	2.07	-0x1.6606d9a4e7ae1b644bb0a547f06ap+0	0.501
sqrt	0xf.ffffffffffffffffffffffffffff8p-4	0.500	0xf.ffffffffffffffffffffffffffff8p-4	0.500
tan	-0x3.832b771f9462df46117b6a863fa2p+8	1.06	0x2.d93a999513c01714079e4f7b8f0ap+36	0.502
tanh	-0x3.c26abeca541298cca288adbd1e12p-4	2.39	-0x2.01ccac403cc83f179b0e4c355f92p-4	0.501
tgamma	-0x1.62ab0823decc5cf957d9a218cf27p+4	10.7	0x4.000047bf7dd56b027a0eb6672638p-15344	820e1
y0	0x6.b99c822052e965e1754eb5ffeb08p+4	1.69e33	0x3.9561432d16442ec543c74876d1c8p+4	4.79e27
y1	0x2.3277da9bfe485c85c35e5bcc806p+0	3.47e33	0x2.80bc307275f6a6a3feb2ab211838p+4	1.45e30
atan2	0x1.41df5aa214612c7e019fa6ade88p-13316 0x5.e53b26a270a29eb9f77ef8ef7af8p-13316	1.89	-0x1.fb41ff205f5ade930a9fcbba8ea8p-16384 0x2.23f098fd6b8799dbeb03219bfa08p-10520	0.501
hypot	-0x2.d8311789103b76133ea1d5bc38c4p-16384 -0x1.6d85492006d7dcc6cc52938684p-16384	0.985	0x8.79ec30b61f9b839fe507bbdf414p-11908 0xb.94f6832f64d0729ebd68035ed7a8p-11908	0.501
pow	0x1.36627b3005e120b290ed016d6087p+0 -0xe.6019c4717345a8dec925ee6219fp+12	30.2	0x4p-16496 0x3.ffffff39c102f0aa11bb2c8a91dp-128	1.40

Table 15: Quadruple precision: GNU libc and Intel Math Library.

[8] 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.

[9] 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.

[10] 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.

[11] GNU libc 2.34: Known maximum errors in math functions. http://www.gnu.org/software/libc/manual/html_node/Errors-in-Math-Functions.html, 2021.

[12] GNU libc version 2.34. https://www.gnu.org/software/libc/, 2021.

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

[14] Intel Math Library. Distributed with the Intel oneAPI DPC++ Compiler 2021.2.0, 2021.

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

[16] Musl version 1.2.2. https://musl.libc.org/, 2021.

[17] Redhat Newlib version 4.1.0. https://sourceware.org/newlib/, 2020.

[18] OpenLibm version 0.7.5. https://openlibm.org/, 2021.

[19] 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.