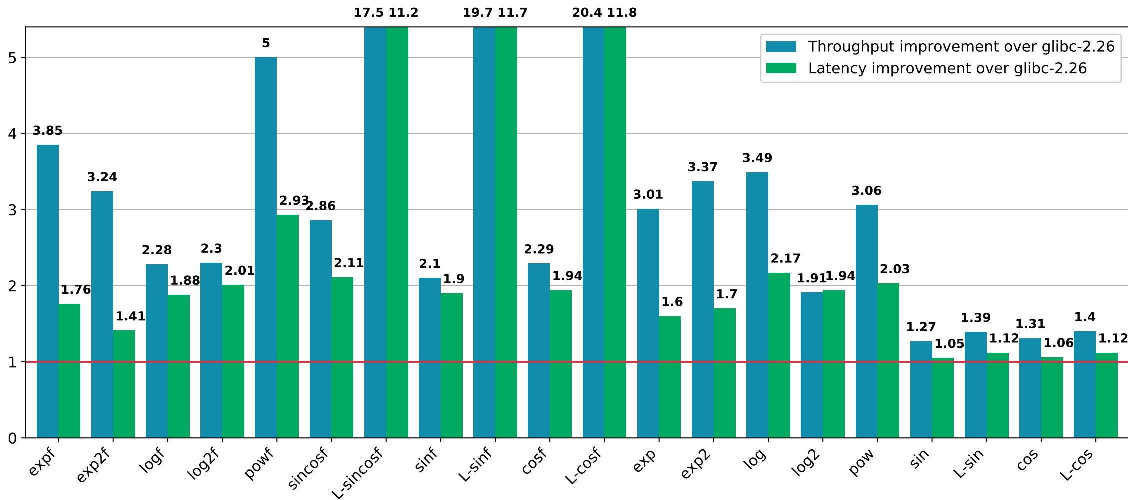
Optimizing Math Routines

Szabolcs Nagy <szabolcs.nagy@arm.com>

September 8, 2018

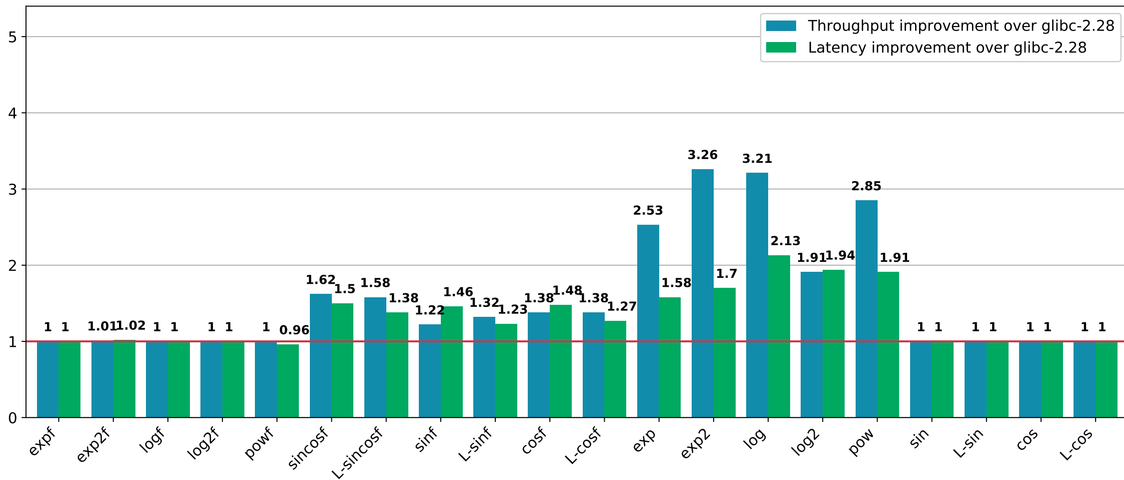
© 2018 Arm Limited

Improvements on Cortex-A72 over glibc 2.26



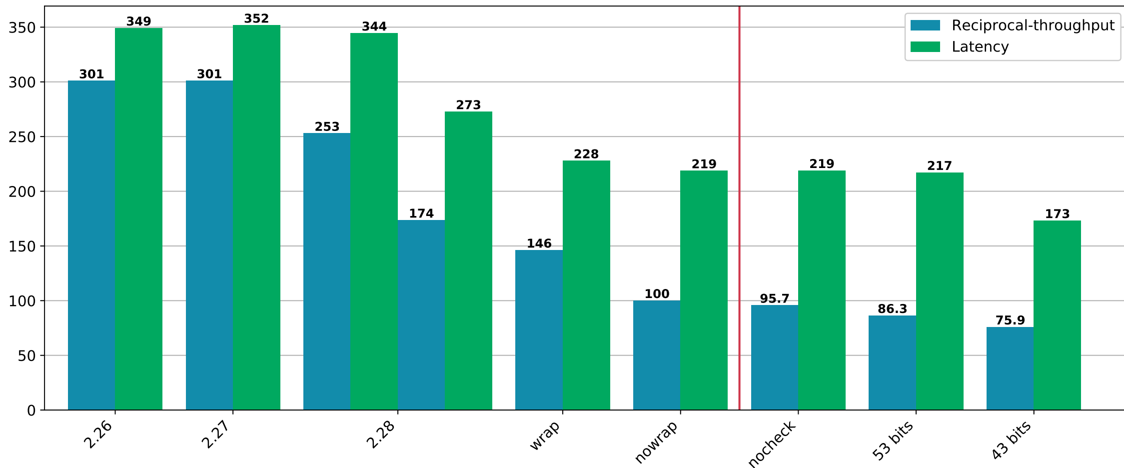
© 2018 Arm Limited
arm

Improvements on Cortex-A72 over glibc 2.28



© 2018 Arm Limited
arm

Performance of exp on Cortex-A72



© 2018 Arm Limited
arm

Size of libm.so on AArch64

text	data	bss	dec	filename	diff
689568	1288	8	690864	libm-2.26.so	
719886	756	12	720654	libm-2.27.so	+29790
706067	756	12	706835	libm-2.28.so	-13819
625357	756	12	626125	libm-wrap.so	-80710
625187	756	12	625955	libm-nowrap.so	-170

© 2018 Arm Limited

Single precision

- expf, exp2f, logf, log2f, powf (glibc-2.27), sinf, cosf, sincosf, tanf (glibc-2.29).
- Use double precision arithmetics.
  - Not for SIMD or GPU.
- Use table based methods.
  - Reduces argument range: 2x table size $\longrightarrow$ 0.5x interval.
- Separate special cases quickly.
  - Don’t use error handling wrapper.
- Reduce dependency (polynomial eval).

© 2018 Arm Limited

Double precision

- exp, exp2, log, log2, pow, (sin, cos, sincos).
- Correct rounding is too slow.
- Cannot test all inputs.
- Rounding errors matter.
- Need extra bits of precision efficiently.
- Presence of fma matters.

© 2018 Arm Limited
arm

expf

```
1  float expf (float x)
2  {
3    uint64_t ki, t;
4    double_t kd, z, r, xd = x;
5    if (is_special (x)) { if (really_special (x)) return spec (x); }
6    z = (InvLn2 * N) * xd;                    // x = ln2/N (k + r)
7    ki = converttoint (z);
8    kd = roundtoint (z);
9    r = z - kd;                               // r ∈ [-1/2, 1/2]
10   t = T[ki % N] + (ki << (52-Nbits));     // 2^(k/N)
11   return asdouble (t) * poly (r);          // 2^(k/N) 2^(r/N)
12 }
```

Comments: $x = \frac{\ln 2}{N}(k + r)$; $r \in [-\frac{1}{2}, \frac{1}{2}]$; $2^{\frac{k}{N}}$; $2^{\frac{k}{N}} 2^{\frac{r}{N}}$

© 2018 Arm Limited

arm

Reduce dependency

- Polynomial evaluation:

```
1    // Horner's: 4 fma, latency = 4 fma
2    (((A[0]*x + A[1])*x + A[2])*x + A[3])*x + A[4];
3    // Pipelined: 4 fma + 1 mul, latency = 3 fma
4    x2 = x*x;
5    (A[0]*x2 + (A[1]*x + A[2]))*x2 + (A[3]*x + A[4]);
```

- Rounding errors might change, largest terms may need special care.
- Parallel round and convert (assume single instructions):

```
1    ki = lround (z);
2    kd = (double)ki; // bad: depends on ki
3    kd = round (z);  // good: can be done in parallel
```


© 2018 Arm Limited
arm

Tweaks

- Consts:
  - Tweaked to become simpler (e.g. immediate in a compare instruction).
  - Optimize access of loaded consts: struct in a separate TU (sharing tables, single address).
  - Loaded data can be more complicated/bigger if it makes computation simpler:
    - 3 doubles per table entry $\longrightarrow$ 4 doubles is faster to index.
    - Split a const into separate top half and bottom half if that helps.
    - The exp table can be adjusted so the code is `T[i]|k<<S` instead of `T[i]|(k&mask)<<S`.
- __glibc_unlikely for slow path.
- Error handling is in tail calls, no call frame is needed.

© 2018 Arm Limited
arm

Target variations

- Portable round/convert is slightly slower than single instructions:

```
1    kd = narrow_eval (kd + 0x1.8p52);
2    ki = asuint64 (kd);
3    kd -= 0x1.8p52;
```

- ... and only works in nearest rounding mode: use more accurate polynomial.
- Separate code paths using __builtin_fma if __FP_FAST_FMA.
- (Float vs integer compare in threshold checks) - removed.
- Different results (without separate code path):
  - FLT_EVAL_METHOD != 0.
  - FMA: glibc uses -ffp-contract=fast, results can depend on what is contracted.
- Configuration options that are not used in glibc but left in the code for documentation:
  - WANT_ROUNDING: code is only needed for non-nearest rounding mode.
  - WANT_ERRNO: code is only needed for errno setting.
  - WANT_ERRNO_UFLOW: code is only needed for errno setting in case of underflow in non-nearest rounding.

© 2018 Arm Limited

Rounding errors - Mul, Add

op	ulp error bound	notes
xy	0.5	black x, y means exact fp value
xy	1.5	red x, y means inexact (has a rounding error)
xx	1.913	
xy	2.5	
$x + y$	0.5	
$x + y$	1	same sign
$x + y$	1.25	same sign
$x + y$	0	opposite sign, $0.5 \leq \|x/y\| \leq 2$
$x + y$	2^{b-1}	opposite sign, $0.5 \leq \|x/y\| \leq 2$, $\text{ulp}(x+y) = 2^{-b}\,\text{ulp}(x)$
$x + y$	$0.5 + 2^{-b-1}$	large exact y, small inexact x, $\text{ulp}(x+y) = 2^{b}\,\text{ulp}(x)$

© 2018 Arm Limited

arm

Extra precision - Add

- Exact Add if $|a| \geq |b|$

```
1 r = a + b;
2 t = a - r + b; // exact!
```

- Correct-rounding check:

```
1 if (r + c*t == r)
```


© 2018 Arm Limited

arm

Extra precision - Mul

```
1  r = a * b;
2  t = a * b - r;  // exact! ..with fma(a, b, -r)
```


© 2018 Arm Limited

arm

Extra precision - Mul - Veltkamp-Dekker

```
1  r = a * b;
2
3  split = 0x1p27 + 1;
4
5  s = a*split;
6  ha = a - s + s;  // ~ a + a*0x1p27 - a*0x1p27
7  ta = a - ha;
8
9  s = b*split;
10 hb = b - s + s;
11 tb = b - hb;
12
13 t = -r + ha*hb + ha*tb + hb*ta + ta*tb;
```


© 2018 Arm Limited
arm

Extra precision - Mul - practice

```
1 ha = asdouble((asuint64(a) + (1<<26)) & -1ULL<<27); // precise
2
3 ha = asdouble(asuint64(a) & -1ULL<<32); // ≈ 20 extra bits
4 ta = a - ha;
5 // hb = ...
6 head = ha*hb;
7 tail = ha*tb + hb*ta + ha*tb;
```


© 2018 Arm Limited
arm

Precision bottlenecks - exp, log

- $\exp(x) = 2^{k/N}\mathrm{poly}(r)$

```
1  pow2[i] * (1 + p);                            // ≈ 2ulp
2  pow2[i] + pow2[i] * p;                        // ≈ 1ulp
3  pow2[i] + (pow2tail[i] + pow2[i]*p);          // ≈ 0.5ulp, exponent fix
4  pow2[i] + pow2[i]*(pow2tail[i] + p);          // ≈ 0.5ulp
```

- $\log(x) = \log(2^k z) = k\ln 2 + \log(c) + \mathrm{poly}(z/c - 1)$

```
1  r = z*invc[i] - 1;                    // z/c - 1
2  poly = c1*r + c2*r*r + r*r*r*p;       // ≈ log(r + 1)
3  result = k*ln2 + logc[i] + poly       // k ln 2 + log(c) + log(z/c)
```


© 2018 Arm Limited
arm

Precision bottlenecks - log

```
1      r = fma(z, InvC[i], -1.0); // z*invc - 1
2      B1 = k*Ln2hi + LogChi[i];  // big: k*ln2 + log(c)
3      B2 = B1 + r;               // big: k*ln2 + log(c) + r
4      S1 = k*Ln2lo + LogClo[i];  // small: error of k*ln2 + log(c)
5      S2 = B1 - B2 + r;          // small: error of ... + r
6      r2 = r * r;                // r*r
7      r3 = r * r2;               // r*r*r
8      B3 = B2 - 0.5*r2;          // big: k*ln2 + log(c) + r - r*r/2
9      S3 = -0.5*fma(r, r, -r2);  // small: error of r*r/2
10     S4 = B2 - B3 - 0.5*r2;     // small: error of ... - r*r/2
11     p = A[1] + r*A[2] + r2*(A[3] + r*A[4] + r2*(A[5] + r*A[6]));
12     S5 = S1 + S2 + S3 + S4 + r3*p;
13     ret = B3 + S5;             // k*ln2 + log(c) + r - r^2/2 + r^3 p
```


© 2018 Arm Limited
arm

Topics

- Compiler wishes
- Language wishes
- Glibc wishes
- Tools
- Bugs
- Correct rounding

© 2018 Arm Limited

the end

The Arm trademarks featured in this presentation are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners.

www.arm.com/company/policies/trademarks

© 2018 Arm Limited

Compiler

- Known optimization issues
  - slow path can affect fast path
  - address computations
  - math errno (fortran..)
- fenv
  - barriers
  - separate round, except, call barrier
  - iso c is not fine grain
- excess prec: `double_t` vs gcc semantics

© 2018 Arm Limited
arm

Language

- assoc arithmetics annot (poly eval, etc)
- fast fma contract annot
- automatic struct layout
  - symbol pointing into the middle of a struct
- fine grain math barriers

© 2018 Arm Limited
arm

Glibc

- separate `/wordsize64/` code.
- (unmaintained) targets with separate code.
- c files included in several places (ifunc hacks).
- cannot build/test/benchmark a single component.
- no sub ulp in test results.
- separate ulp files for targets that could share it.
- builtins vs __ prefixed names.
- wrapper magic.
- ChangeLog, NEWS file conflicts.

© 2018 Arm Limited
arm

Genericness

- arch/isa dependence
  - round/convert toint
  - fma
  - special case checks (float vs int, immediates)
- uarch dependence
  - schedule, assoc math, poly eval
  - struct layout

© 2018 Arm Limited
arm

Implementation bugs

- glibc `/wordsize64/` log2
- sinf/cosf namespace violation
- exp subnormal 0.75 ulp error
- exp missed underflow
- pow sign of 0 with `!WANT_ROUNDING`
- exp nofma underflow
- pow large ulp error with nofma

© 2018 Arm Limited
arm

Correct rounding

- table makers dilemma
- nice properties (monotonicity, $-f(x) = f(-x)$ etc.)
- 1991 Ziv: fast impl possible (falls back to 768 bit prec)
- 2001 IBM contributes cr math code to glibc
- 2004 INRIA: fast impl, worst cases (crlibm, 119 bits)
- 2013 glibc no longer wants cr math
  - non-nearest bug
  - extreme latency spikes (1ms pow instead of 100ns)
  - large tables
  - complex code, hard to maintain
  - caller needs to deal with rounding errors anyway
- 2018 Wilco's work (drop slow paths)

© 2018 Arm Limited
arm

Rounding errors - Background

- algorithm template:
  - special case handling
  - argument reduction
  - approximation in a small interval (polynomial)
  - result reconstruction
- errors:
  - approx error (polynomial)
  - rounding error

© 2018 Arm Limited

Rounding errors - Formulas

- $x = \pm 2^k m, \quad m \in [1, 2)$
- IEEE binary64: $k \in [-1022, 1023], \quad 2^{52} m \in \mathbb{N}, \quad \varepsilon = 2^{-52}$
- ulp: $\mathrm{ulp}(x) = 2^{k-52} = 2^k \varepsilon = \varepsilon |x| / m$
- $\frac{1}{2}\varepsilon|x| < \mathrm{ulp}(x) \leq \varepsilon|x|$
- absolute error of rounding:
  - $|\mathrm{R}(x) - x| \leq \mathrm{ulp}(x)/2 \leq \frac{1}{2}\varepsilon|x|$
  - $\mathrm{R}(x) = x + a, \quad |a| \leq \mathrm{ulp}(x)/2$
- relative error of rounding:
  - $|(\mathrm{R}(x) - x)/x| < \frac{1}{2}\varepsilon$
  - $\mathrm{R}(x) = x(1 + r), \quad |r| < \frac{1}{2}\varepsilon$

© 2018 Arm Limited
arm

Rounding errors - Example - Error of Mul

- ULP error of $x \cdot y$?
- $\left| \mathrm{R}\left(\mathrm{R}(x)\, \mathrm{R}(y)\right) - xy \right| = ?\ \mathrm{ulp}(xy)$

© 2018 Arm Limited
arm

Rounding errors - Example - Relative error

- $\mathrm{R}(\mathrm{R}(x)\,\mathrm{R}(y)) = xy(1 + r_x)(1 + r_y)(1 + r_{xy}) \approx xy(1 + r_x + r_y + r_{xy})$
- $|\,\mathrm{R}(\mathrm{R}(x)\,\mathrm{R}(y)) - xy| = |(r_x + r_y + r_{xy})xy| < \frac{3}{2}\varepsilon|xy| < 3\,\mathrm{ulp}(xy)$

© 2018 Arm Limited
arm

Rounding errors - Example - ULP error

- $\mathrm{R}(x)\,\mathrm{R}(y) \to (x + \mathrm{ulp}(x)/2)(y + \mathrm{ulp}(y)/2) \approx xy + x\,\mathrm{ulp}(y)/2 + y\,\mathrm{ulp}(x)/2$
- $\mathrm{R}(\mathrm{R}(x)\,\mathrm{R}(y)) \to xy + x\,\mathrm{ulp}(y)/2 + y\,\mathrm{ulp}(x)/2 + \mathrm{ulp}(\mathrm{R}(x)\,\mathrm{R}(y))/2$
- $x\,\mathrm{ulp}(y) + y\,\mathrm{ulp}(x) = 2^{k_x} m_x 2^{k_y - 52} + 2^{k_y} m_y 2^{k_x - 52} = 2^{k_x + k_y - 52}(m_x + m_y)$
  $= (m_x + m_y)\,\mathrm{ulp}(xy) < 3\,\mathrm{ulp}(xy) \quad \text{if} \quad m_x m_y < 2$
  $= (m_x + m_y)\,\mathrm{ulp}(xy)/2 < 2\,\mathrm{ulp}(xy) \quad \text{if} \quad m_x m_y \geq 2$
- $|\,\mathrm{R}(\mathrm{R}(x)\,\mathrm{R}(y)) - xy| < 3\,\mathrm{ulp}(xy)/2 + \mathrm{ulp}(xy)/2 = 2\,\mathrm{ulp}(xy)$

© 2018 Arm Limited
arm

Rounding errors - Example - Test cases

x	0x1.0000001ffffff8p+0	0x1.0000002d413cb8p+0
y	0x1.ffffffbffffff8p+0	0x1.ffffffa57d8698p+0
x y	0x1.ffffffffffffe000000010000004p+0	0x1.fffffffffffff8000010425c5d4p+0
$R(x)\ R(y)$	0x1.fffffffffffff8p+0	0x1.000000000000080000076c293cp+1
$R(R(x)\ R(y))$	0x1p+1	0x1.0000000000001p+1
ulp error	2.0	2.5

© 2018 Arm Limited
arm

Architecture

- +10-15 bits prec arithmetics
- top half round
- load/store single prec into double prec
  - pow table compression
- less performance anomalies..

© 2018 Arm Limited

arm

mathbench

```
./mathbench -m 100 exp log
   exp rthruput:    9.20 ns/elem    9195255 ns in [-9.9 9.9]
   exp  latency:   20.02 ns/elem   20015955 ns in [-9.9 9.9]
   exp rthruput:    9.20 ns/elem    9204355 ns in [0.5 1]
   exp  latency:   20.02 ns/elem   20016215 ns in [0.5 1]
   log rthruput:   11.23 ns/elem   11233422 ns in [0.01 11.1]
   log  latency:   23.28 ns/elem   23280187 ns in [0.01 11.1]
   log rthruput:   11.73 ns/elem   11729524 ns in [0.999 1.001]
   log  latency:   21.27 ns/elem   21266180 ns in [0.999 1.001]
```


© 2018 Arm Limited

ulp test

```
 ./ulp2 log 0.5 1.0 100000
log(0x1.09b91bf53bfdp-1) got -0x1.4fce2df975ffep-1 want -0x1.4fce2df975ffdp-1 -0.498839 ulp err -0.00116126 0x1.d0241530f5f73p-63
log(0x1.21ab10064571cp-1) got -0x1.23a118e330c45p-1 want -0x1.23a118e330c44p-1 -0.49859 ulp err -0.00140987 0x1.446f61f11dacap-62
log(0x1.3a69e109f84a3p-1) got -0x1.f3516f894d72bp-2 want -0x1.f3516f894d72cp-2 +0.498567 ulp err 0.0014331 -0x1.81389f43e07cp-63
log(0x1.4e692ab8ed99p-1) got -0x1.b42d65778e63cp-2 want -0x1.b42d65778e63dp-2 +0.49838 ulp err 0.00161991 -0x1.f277a297bf15ap-63
log(0x1.5234f2a56e7c7p-1) got -0x1.a89e776115941p-2 want -0x1.a89e776115942p-2 +0.497455 ulp err 0.00254536 -0x1.9247db7252eb4p-62
log(0x1.6801e3cdca09fp-1) got -0x1.68a723c62a52ep-2 want -0x1.68a723c62a52fp-2 +0.49732 ulp err 0.00268049 -0x1.f2c683af2e9a2p-62
log(0x1.8fb777eedca8ep-1) got -0x1.fb0544fb5692bp-3 want -0x1.fb0544fb5692ap-3 -0.496893 ulp err -0.00310708 0x1.9b4007af97a8cp-62
log(0x1.902c1c8a92e1bp-1) got -0x1.f8aff96e9d9b9p-3 want -0x1.f8aff96e9d9b8p-3 -0.496187 ulp err -0.00381315 0x1.fb09db563718dp-62
log(0x1.9fe5adc3250a5p-1) got -0x1.a9c06cbf23353p-3 want -0x1.a9c06cbf23352p-3 -0.491103 ulp err -0.00889657 0x1.5e944f874f7eap-60
log(0x1.d3f181f03905dp-1) got -0x1.708c099cf5756p-4 want -0x1.708c099cf5755p-4 -0.489803 ulp err -0.0101972 0x1.d033ca55bb814p-60
PASS log [0x1p-1;0x1p+0] round n errlim inf maxerr 0.0101972 cnt 100000 cnt1 112 0.112% cnt2 0 0% cntfail 0 0%
```

 © 2018 Arm Limited

arm

sollya script

```
N=128; b=log(2)/(2*N); deg=4;

approx = proc(poly) {
  d = degree(poly);
  return remez(exp(x)-poly(x), deg-1-d, [-b;b], x^(d+1), 1e-10);
};
poly = 1 + x;
for i from 2 to deg do {
  p = roundcoefficients(approx(poly), [|D ...|]);
  poly = poly + x^i*coeff(p,0);
};
display = hexadecimal;
print("rel error:", accurateinfnorm(1-poly(x)/exp(x), [-b;b], 30));
print("abs error:", accurateinfnorm(exp(x)-poly(x), [-b;b], 30));
print("coeffs:"); for i from 0 to deg do coeff(poly,i);
```


© 2018 Arm Limited
arm

sollya output

```
Warning ...
Display mode is hexadecimal numbers.
rel error: 0x1.6f08ef8p-53
abs error: 0x1.6e0adeep-53
coeffs:
0x1p0
0x1p0
0x1.fffffffffd43p-2
0x1.55555c75adbb2p-3
0x1.55555da646206p-5
```


© 2018 Arm Limited

Tools

- math tool: hexfloat, representation, libc results, rounding mode, fenv.
- sollya.gforge.inria.fr: polynomial design (remez, fpminimax)
- ulp measure: worst ULP error in an interval (compared to quad prec)
- mathbench: throughput, latency in an interval
- table generation tools (log)
- plotting tool (error plots)

 © 2018 Arm Limited

arm

Monotonicity

$$\mathrm{ulp}(x)f'(x)/\,\mathrm{ulp}(f(x)) > 2b - 1$$

© 2018 Arm Limited

GCC issues

- pc relative literal access (many adrp vs one)
- float to int reinterpret (stack vs fmov)
- slow path affects fast path: register allocation
- slow path affects fast path: calls
- simple float consts are not synthesized
- fail to use simple const for compare.
- insn scheduling: load of coeffs around fmas
- insn scheduling: inline asm vs builtin
- missed tailcall opt (PR 80237, fixed)
- suboptimal address computations
- -fmath-errno is default (e.g. sqrt and lrint inlining)
- -frounding-math fails for narrowing conv (i386 PR 57245)
- clang: no fenv support.

© 2018 Arm Limited

Obscure stuff

- asuint, asfloat, asuint64, asdouble.
- poly eval.
- literals, immediates, tiny model.
- wrapper, errno, _LIB_VERSION.
- aarch64 vs aarch32 vs softfloat.
- gpu, simd.
- double $\neq$ ieee binary64 (not handled).
- Mixed endian repr (not handled).
- FLT_EVAL_METHOD $\neq 0$ (double_t).
- underflow errno.

© 2018 Arm Limited
arm

Algorithms - exp, log

- exp

$$x = k\frac{\ln 2}{N} + r$$

$$\exp(x) = 2^{\frac{k}{N}} \exp(r)$$

- log

$$x = 2^k z$$

$$\log(x) = k \ln 2 + \log(c) + \log(z/c)$$

$$\log(z/c) = \text{poly}(z/c - 1)$$

© 2018 Arm Limited
arm