Recent and future developments of GNU MPFR

Paul Zimmermann

iRRAM/MPFR/MPC workshop, Dagstuhl, April 18, 2018

The GNU MPFR library

- a software implementation of *binary* IEEE-754

- variable/arbitrary precision (up to the limits of your computer)

- *each variable* has its own precision: `mpfr_init2 (a, 35)`

- global user-defined exponent range (might be huge): `mpfr_set_emin (-123456789)`

- mixed-precision operations: $a \leftarrow b - c$ where a has 35 bits, b has 42 bits, c has 17 bits

- correctly rounded mathematical functions (exp, log, sin, cos, ...) as in Section 9 of IEEE 754-2008

History

- 2000: first public version;
- 2008: MPFR is used by GCC 4.3.0 for *constant folding*:

```
double x = sin (3.14);
```

- 2009: MPFR becomes GNU MPFR;
- 2016: 4th developer meeting in Toulouse.
- Dec 2017: release 4.0.0
- `mpfr.org/pub.html` mentions 2 books, 27 PhD theses, 63 papers citing MPFR
- Apr 2018: iRRAM/MPFR/MPC developer meeting in Dagstuhl

MPFR is used by SageMath

```
SageMath version 8.1, Release Date: 2017-12-07
Type "notebook()" for the browser-based notebook interface.
Type "help()" for help.

sage: x=1/7; a=10^-8; b=2^24
sage: RealIntervalField(24)(x+a*sin(b*x))
[0.142857119 .. 0.142857150]
```

Representation of MPFR numbers (*mpfr_t*)

- precision $p \geq 1$ (in bits);
- sign (-1 or $+1$);
- exponent (between $E_{\min}$ and $E_{\max}$), also used to represent special numbers (NaN, $\pm\infty$, ± 0);
- significand (array of $\lceil p/64 \rceil$ *limbs*), defined only for *regular* numbers (neither NaN, nor $\pm\infty$ and ± 0, which are *singular* values).

The most significant bits are shown on the left.

Regular numbers are *normalized*: the most significant bit of the most significant *limb* must be 1.

Example, $x = 17$ with a precision of 10 bits and *limbs* of 6 bits is represented as follows:

$$\underbrace{10}_{\text{precision}} \quad \underbrace{+1}_{\textit{sign}} \quad \underbrace{5}_{\textit{exponent}} \quad \underbrace{\boxed{100010}}_{\text{limb 1}}\underbrace{\boxed{000000}}_{\text{limb 0}}$$

Major new features in MPFR 4

- major speedup for add, sub, mul, div, sqrt for 1 or 2 words (up to 3 words for add, sub, mul) when all operands have same precision

- partial support of MPFR_RNDF (faithful rounding)

- new functions mpfr_fmma and mpfr_fmms to compute $ab + cd$ and $ab - cd$

- complete rewrite of mpfr_sum

MPFR 3.1.5 compared to MPFR 4.0-dev

`araignee.loria.fr`, Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz, with GMP 6.1.2 and GCC 6.3.0.
GMP and MPFR are configured with `--disable-shared`.

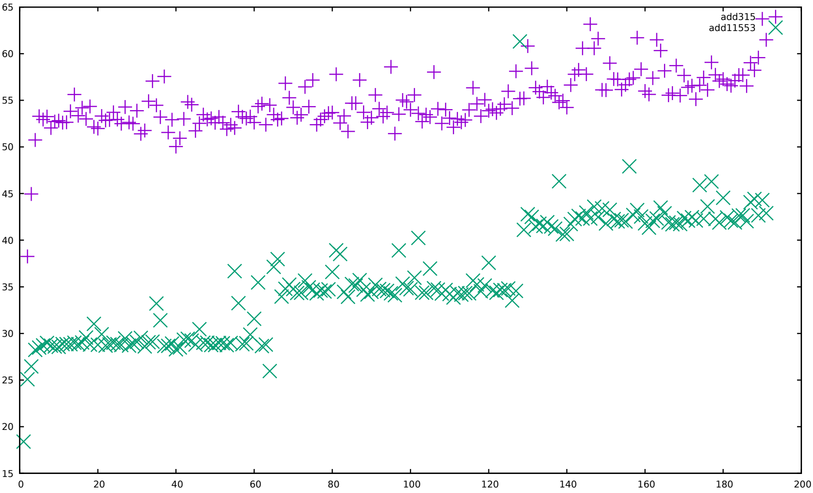
MPFR 3.1.5

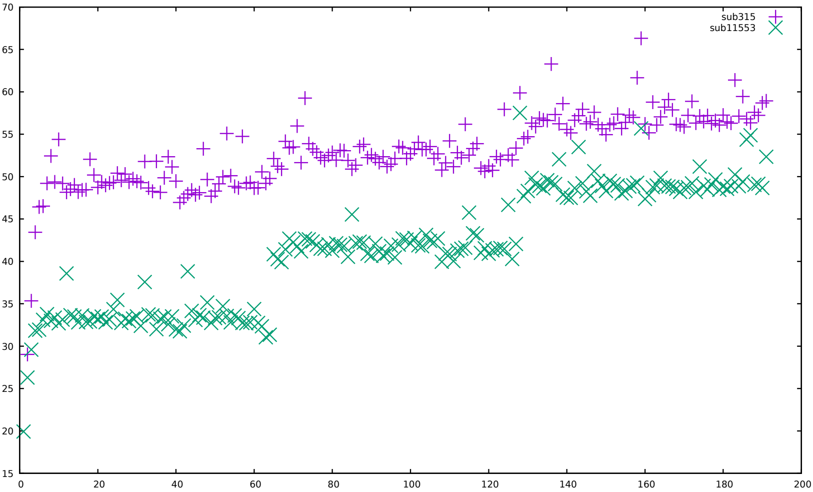
bits	53	113
mpfr_add	52	53
mpfr_sub	49	52
mpfr_mul	49	63
mpfr_sqr	74	79
mpfr_div	134	146
mpfr_sqrt	171	268

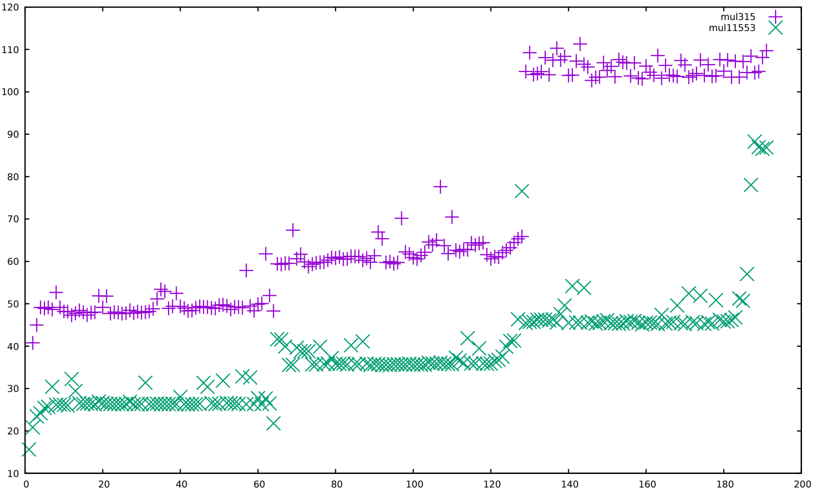
MPFR 4.0-dev

bits	53	113
mpfr_add	**25**	**29**
mpfr_sub	**28**	**33**
mpfr_mul	**23**	**33**
mpfr_sqr	**21**	**29**
mpfr_div	**56 (64)**	**77 (102)**
mpfr_sqrt	**55 (56)**	**84 (133)**

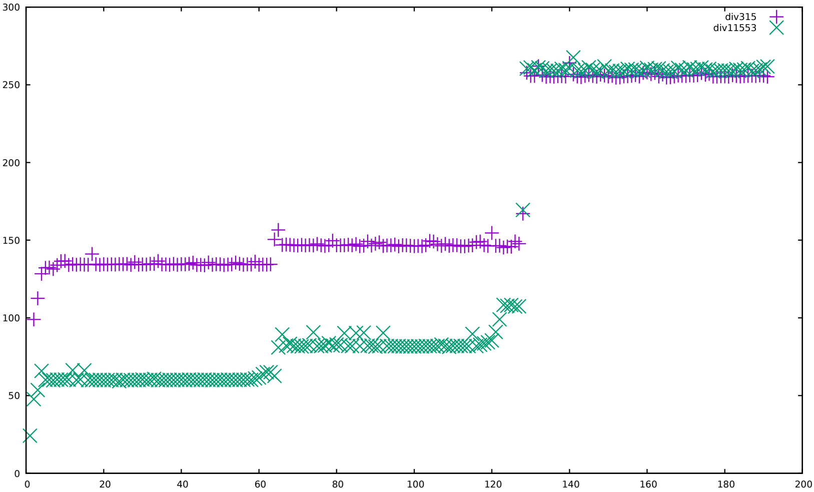
Timings are in cycles.


add315
add11553
65
60
55
50
45
40
35
30
25
20
15
0
20
40
60
80
100
120
140
160
180
200

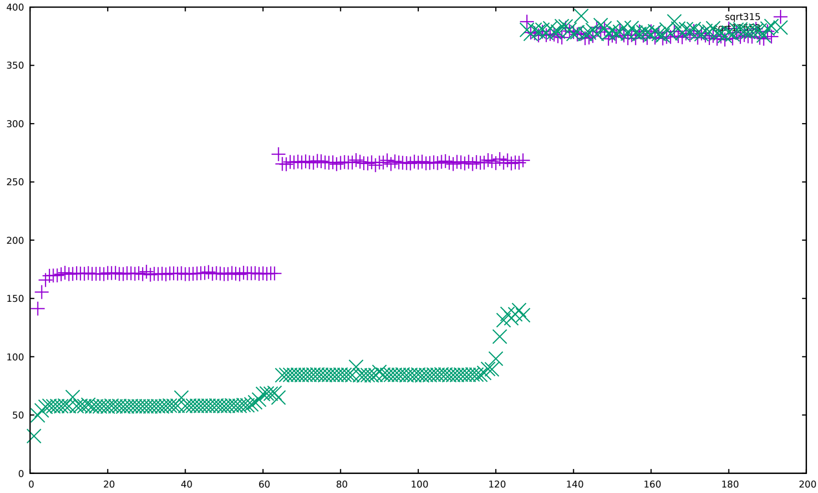

sub315
sub11553
70
65
60
55
50
45
40
35
30
25
20
15
0
20
40
60
80
100
120
140
160
180
200



mul315
mul11553
120
110
100
90
80
70
60
50
40
30
20
10
0
20
40
60
80
100
120
140
160
180
200



div315
div11553
0
50
100
150
200
250
300
0
20
40
60
80
100
120
140
160
180
200


sqrt315
400
350
300
250
200
150
100
50
0
0
20
40
60
80
100
120
140
160
180
200

Faithful Rounding

MPFR 4 also includes a new rounding mode, RNDF, for *faithful rounding*.

With RNDF, the result is either that for RNDD (toward $-\infty$) or RNDU (toward $+\infty$), and might depend on the platform.

In particular, when the result is exact, only this exact value is possible.

The ternary value gives no information.

What's new in the development version?

- improved test coverage from 96.3% to 98.2% of code for x86_64 (and found a few bugs while doing this)

- replaced __float128 (GCC extension) by _Float128 (ISO/IEC TS 18661)

- new function mpfr_get_str_ndigits, that gives the number of digits output by mpfr_get_str when its digits argument is zero

Future plans

- formally prove the low-level code, in particular the special algorithms (and code) added in MPFR 4 for 1 and 2 limbs

- improve the test coverage to at least 99% on 64-bit ABI, and also on other platforms (32-bit ABI)

Small tasks

- implement a function mpfr_hash
- update the timings web page for MPFR 4.0.1
- implement new functions from the C++17 standard (see `http://en.cppreference.com/w/cpp/numeric/special_math`): assoc_laguerre, assoc_legendre, comp_ellint_1, comp_ellint_2, comp_ellint_3, cyl_bessel_i, cyl_bessel_j, cyl_bessel_k, cyl_neumann, ellint_1, ellint_2, ellint_3, hermite, legendre, laguerre, sph_bessel, sph_legendre, sph_neumann
- implement mpfr_get_decimal128 and mpfr_set_decimal128
- implement mpfr_q_sub, mpfr_z_div, mpfr_q_div
- implement mpfr_pow_q and variants with two integers (native or mpz)
- improve test coverage
- put here your favorite task