Recent and future developments of GNU MPFR

Paul Zimmermann
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:
  \[
  \text{double } x = \sin (3.14);
  \]
- 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 $E_{\text{min}}$ and $E_{\text{max}}$), also used to represent special numbers (NaN, $\pm \infty$, $\pm 0$);
- significand (array of $\lceil p/64 \rceil$ limbs), defined only for regular numbers (neither NaN, nor $\pm \infty$ and $\pm 0$, 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:

```
  10  +1  5  100010  000000
   precision  sign  exponent  limb 1  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.

<table>
<thead>
<tr>
<th></th>
<th>MPFR 3.1.5</th>
<th></th>
<th>MPFR 4.0-dev</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>bits</td>
<td></td>
<td>bits</td>
</tr>
<tr>
<td>mpfr_add</td>
<td>52</td>
<td>53</td>
<td>mpfr_add</td>
</tr>
<tr>
<td>mpfr_sub</td>
<td>49</td>
<td>52</td>
<td>mpfr_sub</td>
</tr>
<tr>
<td>mpfr_mul</td>
<td>49</td>
<td>63</td>
<td>mpfr_mul</td>
</tr>
<tr>
<td>mpfr_sqr</td>
<td>74</td>
<td>79</td>
<td>mpfr_sqr</td>
</tr>
<tr>
<td>mpfr_div</td>
<td>134</td>
<td>146</td>
<td>mpfr_div</td>
</tr>
<tr>
<td>mpfr_sqrt</td>
<td>171</td>
<td>268</td>
<td>mpfr_sqrt</td>
</tr>
</tbody>
</table>

Timings are in cycles.
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_n_digits, 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