Exact rounding in multiple precision: the MPFR library

Sylvie Boldo, Guillaume Hanrot, Fabrice Rouillier, Jean-Luc Rémy, Paul Zimmermann

PolKA project, INRIA Lorraine and LORIA, Nancy

Plan of the talk

1. Current situation and goals
2. An example: the square root function
3. Future and applications

Current situation

Fixed precision: IEEE 754 standard + implementations (hardware, programming language support, compiler support)

$\Longrightarrow$ enables fast and reliable numerical filters (cf. O. Devillers, project Prisme)

Multiple precision: no standard, only interval arithmetic based implementations, doubtful portability

```
    |\^/|     Maple V Release 5 (INRIA)
._|\|   |/|_. Copyright (c) 1981-1997 by Waterloo Maple Inc. All rights
 \  MAPLE  /  reserved. Maple and Maple V are registered trademarks of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> x:= -163*Pi+892*ln(2)-184*gamma:
> evalf(x, 1);
                                      -100.

> evalf(x, 2);
                                       10.

> evalf(x, 3);
                                        0

> evalf(x, 4);
                                       -.1

> evalf(x, 20);
                                                      -6
                                  .18245287177 10
```

Goals

- standard for multiple precision exact rounding (cf. J.-M. Muller)
- implementation as a C library
- efficiency and portability $\rightarrow$ based on GMP (cf. T. Granlund)

MPFR = Multiple Precision Floating-Point Reliable library

```
mpfr_t x, y, z;

mpfr_init2(x, 10);
mpfr_init2(y, 30);
mpfr_init2(z, 20);
mpfr_set_d(y, 1.0, GMP_RNDN);
mpfr_set_d(z, 3.0, GMP_RNDN);
mpfr_div(x, y, z, GMP_RNDU);
mpfr_out_str(stdout, 10, 0, x, GMP_RNDU);
```

Current state

- `mpfr_add`, `mpfr_sub` (based on `mpn_add_n`, `mpn_sub_n`)
- `mpfr_mul` (based on `mpn_mul`)
- `mpfr_div` (based on `mpn_divrem`)
- `mpfr_sqrt` (based on `mpn_sqrtrem`)
- `mpfr_agm`, `mpfr_log` (Sylvie Boldo)
- `mpfr_exp` (Brent's formula)

First results

Sun/Solaris:

```
mpfr_exp differs from libm.a for x=9.24535725028549748572e-01, rnd=0
libm.a gave  2.52069769179743818910e+00,
mpfr_exp got 2.52069769179743774501e+00, dif=1 ulp

mpfr_log differs from libm.a for a=1.08984024310291838233e+00, rnd=1
 double calculus gives 8.60311195385869836860e-02
 mpfr_log        gives 8.60311195385869698082e-02 (1 ulp)
```

PC/Linux:

```
mpfr_exp differs from libm.a for x=9.46206190776594624481e-01, rnd=3
libm.a gave  2.57591855469639297027e+00,
mpfr_exp got 2.57591855469639341436e+00, dif=1 ulp
```

```
mpfr_exp differs from libm.a for x=-4.76816273782620044130e+02, rnd=1
libm.a gave  8.34302249460712934333e-208,
mpfr_exp got 8.34302249460647058187e-208, dif=711 ulp
```

How to guarantee exact rounding?

1. Compute an approximation y of $f(x)$ with $n + \delta$ bits with relative error at most 2^k ulps

2. If $o(y) = o(y \pm 2^k \mathrm{ulp}(y))$, return $o(y)$

3. Increase δ and go to 1.

Optimal guard bits

δ more bits $\rightarrow$ failure $\sim 2^{-\delta}$

Average cost $\sim \frac{1}{1-2^{-\delta}} C(n+\delta)$

For $C(n) = O(n^{1+\epsilon})$, optimal for $\delta 2^{\delta} \sim n$ i.e. $\delta \sim \log_2 n$

Average relative overhead: $O(\frac{1}{n})$

The square root function

Newton's iteration

$$x_{k+1} = x_k + \frac{a - x_k^2}{2x_k}$$

takes $5M(n)$ or $\frac{3}{2}K(n)$ operations ($D(n) = 4M(n)$ or $2K(n)$).

Algorithm $\texttt{SqrtRem}(n = a_3 b^3 + a_2 b^2 + a_1 b + a_0)$

Input: $0 \le a_i < b$ with $a_3 \ge b/4$

Output: (s, r) such that $s^2 \le n = s^2 + r < (s+1)^2$

$(s', r') \leftarrow \texttt{SqrtRem}(a_3 b + a_2)$

$(q, u) \leftarrow \texttt{DivRem}(r' b + a_1, 2s')$

$s \leftarrow s' b + q$

$r \leftarrow ub + a_0 - q^2$

if $r < 0$ **then**

$r \leftarrow r + 2s - 1$

$s \leftarrow s - 1$

return (s, r)

Complexity $\frac{3}{2}K(n)$ ($K(n)$ without remainder).

Proof of the algorithm

1. after $\texttt{SqrtRem}(a_3 b + a_2)$, $n = s'^2 b^2 + r' b^2 + a_1 b + a_0$

2. after $\texttt{DivRem}(r'b + a_1, 2s')$,
$n = s'^2 b^2 + 2s'qb + ub + a_0 = (s'b + q)^2 + ub + a_0 - q^2 \Rightarrow n = s^2 + r$

3. $\geq (s'b + t)^2 \Rightarrow 2s'bt \leq r'b^2 + a_1 b + a_0$. Since $a_0 < b$, then
$2s't \leq r'b + a_1 \Rightarrow t \leq \lfloor \frac{r'b + a_1}{2s'} \rfloor = q$.

4. $a_3 \geq b/4 \Rightarrow s' \geq b/2 \Rightarrow q = \lfloor \frac{r'b + a_1}{2s'} \rfloor \leq \lfloor \frac{r'b + a_1}{b} \rfloor \leq r' \leq 2s'$ since $a_1 < b$

5. $q \leq \lfloor \frac{2s'b + a_1}{2s'} \rfloor \leq b$

6. 4 and 5 imply $q^2 \leq 2s'b \Rightarrow q^2 < 2(s'b + q) = 2s \Rightarrow r \geq 0$ after the if-statement

Some timings

$\sqrt{x}$ where $x = \sqrt{2}$ to n digits in seconds on a PII-366

n	Maple	Pari/GP	mpn_sqrtrem	SqrtRem
1000	0.01	0.0008	0.0004	0.0002
10000	1.5	0.08	0.021	0.008
100000	153	6.9	0.89	0.33
1000000	NA	NA	40.5	15.0

Applications

- simulation of low-precision hardware and/or microcode (division, square root, ...)
- multiple precision interval arithmetic (cf. S. Rump)
- speeding up computations with large integers (random generation, real root isolation, cf. M.-F. Roy)
- computer algebra kernel

Future

- other elementary and special functions (using class properties, for example linear recurrences for holonomic functions, cf. J. van der Hoeven)
- functional operators (derivative, integrals) ?
- adding special algorithms for low precision (with Arénaire)?
- complex numbers?