The Generic Multiple-Precision Floating-Point Addition With Correct Rounding
(as in the MPFR Library)

Vincent LEFÈVRE
Loria / INRIA Lorraine

6th Conference on Real Numbers and Computers
Schloß Dagstuhl, Germany
15 – 17 November 2004
Introduction

MPFR: Arbitrary-precision floating-point system in base 2.

Considered here: the addition of numbers having the same sign.

- The addition of floating-point numbers: a “simple” operation, easy to understand? But many different cases for the generic addition (with arbitrary precisions).
- In MPFR, the addition had been buggy for a long time (missing particular cases...), despite several patches.
  → I completely rewrote the addition function (October 2001).
- How about the complexity? Seems obvious, but...
The MPFR Floating-Point Addition

Note: The negative case is obtained from the positive case.

Input:

- **Positive numbers** $x$ and $y$ of resp. precisions $m \geq 2$ and $n \geq 2$.
- **Target precision** $p \geq 2$.
- **Rounding mode** $\diamond$ (to $-\infty$, to $+\infty$, to 0, or to the nearest).

Output:

- $\diamond_p (x + y)$, i.e. correctly-rounded result.
- **Sign** of $\diamond_p (x + y) - (x + y)$, called *ternary value*. 
The Floating-Point Representation

- All the values considered here are positive real numbers.
- Floating-point representation in precision $p$:

$$0.b_1b_2b_3\ldots b_p \times 2^e$$

where the $b_i$'s are binary digits (0 or 1) forming the mantissa and $e$ is the exponent (a bounded integer).
- The representation is normalized: $b_1 \neq 0$, i.e. $b_1 = 1$.
- We do not consider subnormals here (MPFR does not support them).
**Computation Steps**

The addition (without considering optimization) consists in:

1. ordering $x$ and $y$ so that $e_x \geq e_y$,
2. computing the exponent difference $d = e_x - e_y$,
3. shifting the mantissa of $y$ by $d$ positions to the right,
4. initializing the exponent $e$ of the result to $e_x$ (temporary value),
5. adding the mantissa of $x$ and the shifted mantissa of $y$ (shifting the result by 1 position to the right and incrementing $e$ if there is a carry),
6. rounding the result (setting the mantissa to 0.1 and incrementing $e$ if a carry is generated due to an upward rounding).
Exponent Considerations

- Assume $e_x \geq e_y$.

- Addition of the aligned mantissas with rounding, with 1 or 2 possible carries (due to rounding and arbitrary precision, e.g. $0.111 + 0.111$ gives $0.10 \times 2^2$ for $p = 2$, rounding upwards).

- Exponent $e_{x+y} = e_x +$ carries.

Underflow: impossible.

Possible overflow, but no practical or theoretical difficulties.

→ Will not be considered here (i.e. assume unbounded exponents).

→ We now concentrate on the addition of the mantissas.
Rounding an Exact Real Value

Canonical infinite mantissa of the exact result: \(0.1b_2b_3b_4b_5\ldots\)

The rounding can be expressed as a function of the rounding mode, the **rounding bit** \(r = b_{p+1}\) and the **sticky bit** \(s = b_{p+2} \lor b_{p+3} \lor \ldots\)

<table>
<thead>
<tr>
<th>(r / s)</th>
<th>downwards</th>
<th>upwards</th>
<th>to the nearest</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 / 0</td>
<td>exact</td>
<td>exact</td>
<td>exact</td>
</tr>
<tr>
<td>0 / 1</td>
<td></td>
<td>+</td>
<td>(-)</td>
</tr>
<tr>
<td>1 / 0</td>
<td></td>
<td>+</td>
<td>(- / +)</td>
</tr>
<tr>
<td>1 / 1</td>
<td></td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

“\(-\)” means: exact mantissa truncated to precision \(p\).

“\(+\)” means: add \(2^{-p}\) to the truncated mantissa (→ possible carry).
Finding an Efficient Algorithm

Trailing bits of $x$ and/or $y$ may have no influence on the result. For instance:

$$0.101010000010010001 + 0.10001 \times 2^{-9}$$

rounded to 4 bits.

Only the first 6 bits 101010 of $x$ (and none for $y$) are necessary to deduce the result and the ternary value.

The goal: **take into account as few input bits as possible**.

Note: bits are grouped into words in memory. To simplify, we give here a bit-based description of the algorithm.
The addition can be written $x + y = t + \varepsilon$, where

- $t$ (main term) is computed with the first $p + 2$ bits of $x$ and the corresponding $\max(p + 2 - d, 0)$ bits of $y$,
- $\varepsilon$ (error term) satisfies $0 \leq \varepsilon < 2^{e_x - p - 1} \leq (1/2) \text{ulp}(x + y)$, with equality if there are no carries.

Graphically:

```
  t
 /\ /
|  |
| x' |
|    |
|    | x''
| y' |
|    |
|    | y''
```

where $x''$ may be empty and either $y'$ or $y''$ may be empty (and $x''$ may end after $y''$, and if $y'$ is empty, $y''$ may start after $x''$ ends).
Computing the Main Term

The main term $t$ is computed and written in time $\Theta(p)$:

- an $\Omega(p)$ time is necessary to fill the $p + 2$ bits;
- a linear time is obviously sufficient.

Note: different ways to compute the main term, due to different overlappings and trailing zeros (see the paper for the details concerning the MPFR implementation).

Possible carry detection (to avoid a separate shift) by looking at the most significant bits of $x$ and $y$ first (not implemented in MPFR).

Special bits: \[
\begin{align*}
\text{Bit } p + 1: & \text{ temporary rounding bit } r_t. \\
\text{Bit } p + 2: & \text{ following bit } f.
\end{align*}
\]
If a Carry Was Generated...

Then $p + 3$ bits of the result have really been computed (instead of $p + 2$).

→ In the implementation, consider that the bit $p + 3$ comes from the first iteration of the processing described in a few slides and must be taken into account accordingly.

→ In the following tables, we may assume that $p + 2$ bits of the result have been computed and the bit $p + 3$ is part of the error term.
Following Bit and Error → Rounding and Sticky Bits

Let \( u \) denote the weight \( 2^{-(p+2)} \) of the bit \( p + 2 \) (following bit). So, \( 0 \leq \varepsilon < 2u \).

<table>
<thead>
<tr>
<th>( f )</th>
<th>( \varepsilon )</th>
<th>( r )</th>
<th>( s )</th>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \varepsilon = 0 )</td>
<td>=</td>
<td>0</td>
<td>( 1000_r.0_f + 0.0000 )</td>
</tr>
<tr>
<td>0</td>
<td>( \varepsilon &gt; 0 )</td>
<td>=</td>
<td>1</td>
<td>( 1000_r.0_f + 1.1101 )</td>
</tr>
<tr>
<td>1</td>
<td>( \varepsilon &lt; u )</td>
<td>=</td>
<td>1</td>
<td>( 1000_r.1_f + 0.1101 )</td>
</tr>
<tr>
<td>1</td>
<td>( \varepsilon = u )</td>
<td>+</td>
<td>0</td>
<td>( 1111_r.1_f + 1.0000 )</td>
</tr>
<tr>
<td>1</td>
<td>( \varepsilon &gt; u )</td>
<td>+</td>
<td>1</td>
<td>( 1000_r.1_f + 1.0001 )</td>
</tr>
</tbody>
</table>

“\( = \)” means: the rounding bit is the temporary rounding bit \( p + 1 \).
“\( + \)” means: 1 must be added to the temporary rounding bit \( p + 1 \).
<table>
<thead>
<tr>
<th>$r_t$</th>
<th>$f$</th>
<th>$\varepsilon$</th>
<th>$r$</th>
<th>$s$</th>
<th>downwards</th>
<th>upwards</th>
<th>to the nearest</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>$\varepsilon = 0$</td>
<td>0</td>
<td>0</td>
<td>exact</td>
<td>exact</td>
<td>exact</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>$\varepsilon &gt; 0$</td>
<td>0</td>
<td>1</td>
<td>$-$</td>
<td>$+$</td>
<td>$-$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$\varepsilon &lt; u$</td>
<td>0</td>
<td>1</td>
<td>$-$</td>
<td>$+$</td>
<td>$-$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$\varepsilon = u$</td>
<td>1</td>
<td>0</td>
<td>$-$</td>
<td>$+$</td>
<td>$-$ / $+$</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>$\varepsilon &gt; u$</td>
<td>1</td>
<td>1</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$\varepsilon = 0$</td>
<td>1</td>
<td>0</td>
<td>$-$</td>
<td>$+$</td>
<td>$-$ / $+$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>$\varepsilon &gt; 0$</td>
<td>1</td>
<td>1</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$\varepsilon &lt; u$</td>
<td>1</td>
<td>1</td>
<td>$-$</td>
<td>$+$</td>
<td>$+$</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$\varepsilon = u$</td>
<td>0</td>
<td>0</td>
<td>exact</td>
<td>exact</td>
<td>exact</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>$\varepsilon &gt; u$</td>
<td>0</td>
<td>1</td>
<td>$-$</td>
<td>$+$</td>
<td>$-$</td>
</tr>
</tbody>
</table>
Iteration Over the Remaining Bits

Assume one iterates over bits $p + 3$, $p + 4$, $p + 5$… (best solution?). At each iteration, the mantissa of the temporary result has the form: $0.1z_2z_3 \ldots z_prff\ldots fff$ with an error in the interval $[0, 2) \text{ulp}$, and one iterates as long as the bits after the (temporary) rounding bit are identical.

- $f = 0$: while $x_i = y_{i-d} = 0$.
- $f = 1$: while $x_i + y_{i-d} = 1$. If $x_i = y_{i-d} = 1$, then point $f = 0$.

**Particular case:** $y$ hasn’t been read yet, i.e. $d \geq p + 2$.
If $f = 0$, take into account the fact that $y_1 = 1$: $s = 1$. 
The Complexity

We assume that:

- the mantissa bits are 0 and 1 with equal probabilities,
- \( x \) and \( y \) are independent numbers.

Time complexity in \( \Omega(p) \) and in \( O(m + n + p) \).
Worst case in \( \Theta(m + n + p) \). Average case in \( \Theta(p) \).

In some cases: many possible orders to test the trailing bits.

Note: As the natural distribution of the real numbers is logarithmic, in a very theoretical point of view, it is better to start with the least significant bits for the 0 equality test (i.e. when \( f = 0 \)).
The MPFR Implementation

- Bits grouped into *limbs* (32-bit or 64-bit unsigned integer).
- Bit-based algorithm → limb-based algorithm (not difficult, but more cases to deal with!).
- Bits $p + 1$ and $p + 2$ in variables $rb$ and $fb$, determined on the fly, as soon as they are known (again, many cases...).
- In addition to the $p$ bits of the target, more bits may be taken into account for the main term (to fill the least significant limb).

Various cases in the main term computation; in particular: whether $d$ is a multiple of the limb size. Very dependent on the GMP functions.
Various cases for the error term:

- $x''$ has not entirely been read and $y''$ has not been read yet.
- $x''$ and $y''$ overlap.
- $x''$ has not entirely been read and $y''$ has entirely been read.
- $x''$ has entirely been read and $y''$ has not been read yet.
- $x''$ has entirely been read and $y''$ has not entirely been read.
- $x''$ and $y''$ have entirely been read.

In the overlapping case: two limbs are added. The loop ends as soon as the result is different from 0 for $f = 0$ or the maximum limb value MP_LIMB_T_MAX for $f = 1$. 
Conclusion

• Not so simple, after all…

• The (bit-based) theoretical analysis could help to improve the current MPFR implementation.

• The theoretical analysis could also be useful to provide a full mechanically-checked proof.

• Future work: deal with the subtraction, but more difficult (e.g. possible cancellation, when subtracting very close numbers).