A DECIMAL MULTIPLE-PRECISION INTERVAL ARITHMETIC LIBRARY

RAIM 2017 - LIP, Lyon
October 25, 2017

Stef Graillat, Clothilde Jeangoudoux, and Christoph Lauter
Arithmetic: Decimal vs Binary

Decimal

- \( a = 10^{-1} \times 9.000 \)
- \( b = \log(a) \)
- \( b = -10^{-1} \times 1.05360515... \)
Arithmetic: Decimal vs Binary

Decimal

\[ a = 10^{-1} \times 9.000 \]

\[ b = \log(a) \]

\[ b = -10^{-1} \times 1.05360515... \]

Binary (here float)

\[ \alpha = 2^{-1} \times 1.100110011... \]

\[ \beta = \log(\alpha) \]

\[ \beta = -2^{-4} \times 1.010111110... \]

Decimal arithmetic

Through binary arithmetic
Arithmetic: Decimal vs Binary

Decimal

\[ a = 10^{-1} \times 9.000 \]

\[ b = \log(a) \]

\[ b = -10^{-1} \times 1.05360515\ldots \]

\[ \tilde{b} = -10^{-1} \times 1.05360545\ldots \]

Binary (here float)

\[ \alpha = 2^{-1} \times 1.100110011\ldots \]

\[ \beta = \log(\alpha) \]

\[ \beta = -2^{-4} \times 1.010111110\ldots \]

Through binary arithmetic
Motivations for our work

In what context a decimal multiple-precision interval library may be used?

- Financial applications
- Validation and test of systems subject to strong certification processes:
  - aerospace industry,
  - autonomous car...
Motivations for our work

In what context a decimal multiple-precision interval library may be used?

◆ Financial applications
◆ Validation and test of systems subject to strong certification processes:
  > aerospace industry,
  > autonomous car...

Why use multiple-precision?

◆ multiple-precision arithmetic: in opposition to fixed precision arithmetic, enables the user to choose the precision of each variable.
◆ The precision is only limited by the space and the time needed for the computation.
  > Increasing the precision may increase the accuracy.
Motivations for our work

In what context a decimal multiple-precision interval library may be used?

◆ Financial applications
◆ Validation and test of systems subject to strong certification processes:
  > aerospace industry,
  > autonomous car...

Why use multiple-precision?

◆ multiple-precision arithmetic: in opposition to fixed precision arithmetic, enables the user to choose the precision of each variable.
◆ The precision is only limited by the space and the time needed for the computation.
  > Increasing the precision may increase the accuracy.

Why use interval arithmetic?

◆ interval arithmetic: represent a number with an interval, representable by machine numbers, containing its value. It gives us guarantees on the numerical result.
Our goal

Provide a decimal multiple-precision interval arithmetic through two libraries

- MPD: Multiple Precision Decimal
- MPDI: Multiple Precision Decimal Interval
Our goal

Provide a decimal multiple-precision interval arithmetic through two libraries

- MPD: Multiple Precision Decimal
- MPDI: Multiple Precision Decimal Interval

State of the art libraries

- multiple precision: BigInteger, BigDecimal
- interval arithmetic: JInterval

Our contributions

- Correctly rounded
- Reliable
- Fast
Our goal

Provide a decimal multiple-precision interval arithmetic through two libraries

- MPD: Multiple Precision Decimal
  - Using MPFR and GMP mathematical functions
- MPDI: Multiple Precision Decimal Interval
  - based on MPD the same way MPFI is based on MPFR

- GMP: GNU Multiple Precision Arithmetic Library
- MPFR: Multiple Precision Floating-Point Reliable Library
- MPFI: Multiple Precision Floating-Point Interval Library
MPD and MPDI types

Note on the IEEE Standard for Floating-Point Arithmetic (IEEE-754 2008)

- MPD arithmetic is IEEE 754 compatible, but not compliant (e.g. there is no quantum for now)
- MPD numbers are not normalized: a number does not have a unique representation in MPD

MPD type

\[ MPD : 10^F \times n \]

- significand \( n \) is a GMP signed integer,
- exponent \( F \) is an integer,
- additional information structure (NaN, ±\( \infty \), ±0 or number),
- the decimal precision \( k \) in digits

MPDI type

\[ MPDI : [\text{left}, \text{right}] \]

- Interval of two MPD numbers \( \text{left} \) and \( \text{right} \)
Outline

1. Introduction
2. Mechanisms behind MPD
3. Mechanisms behind MPDI
4. Benchmark
**MPD Library Architecture**

- **mpd_cmp**
  - Comparison

- **mpd_add**
  - Addition with `mpz_add`

- **mpd_sub**
  - Subtraction with `mpz_sub`

- **mpd_mul**
  - Multiplication with `mpz_mul`

- **mpd_get_fr**
  - Conversion from decimal to binary

- **mpd_set_fr**
  - Conversion from binary to decimal

- **mpd_div**
  - Division with `mpfr_div`

- **mpd_sqrt**
  - Square root with `mpfr_sqrt`

- **mpd_exp**
  - Exponential with `mpfr_exp`

- **mpd_log**
  - Logarithm with `mpfr_log`
    - with a special treatment near 1

**Two central functions**
- Conversion: binary → decimal
- Conversion: decimal → binary

**Two groups of functions**
- with GMP
- with MPFR
Conversion algorithm: binary to decimal

Indecision with the rounding mode to the nearest

- $a$ and $b$ two representable decimal numbers in the current precision
Conversion algorithm: binary to decimal

Indecision with the rounding mode to the nearest

- $a$ and $b$ two representable decimal numbers in the current precision
- Convert the binary number $\alpha_1$ into $a$ or $b$
  - Enclose $\alpha_1$ with two values
  - Compare the enclosure of $\alpha_1$ with the midpoint of $[a, b]$
  - Round $\alpha_1$ to $a$

![Diagram showing a and b with a midpoint indicating indecision.](Image)
Conversion algorithm: binary to decimal

Indecision with the rounding mode to the nearest

- $a$ and $b$ two representable decimal numbers in the current precision
- Convert the binary number $\alpha_1$ into $a$ or $b$
  - Enclose $\alpha_1$ with two values
  - Compare the enclosure of $\alpha_1$ with the midpoint of $[a, b]$
  - Round $\alpha_1$ to $a$
Conversion algorithm: binary to decimal

Indecision with the rounding mode to the nearest

- $a$ and $b$ two representable decimal numbers in the current precision
- Convert the binary number $\alpha_1$ into $a$ or $b$
  - Enclose $\alpha_1$ with two values
  - Compare the enclosure of $\alpha_1$ with the midpoint of $[a, b]$
  - Round $\alpha_1$ to $a$
- Convert the binary number $\alpha_2$ into $a$ or $b$
  - Comparison leads to indecision: increase the precision and compare again
  - Ziv’s loop
Conversion algorithm: binary to decimal

Indecision with the rounding mode to the nearest

- $a$ and $b$ two representable decimal numbers in the current precision
- Convert the binary number $\alpha_1$ into $a$ or $b$
  - Enclose $\alpha_1$ with two values
  - Compare the enclosure of $\alpha_1$ with the midpoint of $[a, b]$
  - Round $\alpha_1$ to $a$
- Convert the binary number $\alpha_2$ into $a$ or $b$
  - Comparison leads to indecision: increase the precision and compare again
  - Ziv’s loop
Conversion algorithm: binary to decimal

Indecision with the rounding mode to the nearest

- $a$ and $b$ two representable decimal numbers in the current precision
- Convert the binary number $\alpha_1$ into $a$ or $b$
  - Enclose $\alpha_1$ with two values
  - Compare the enclosure of $\alpha_1$ with the midpoint of $[a, b]$
  - Round $\alpha_1$ to $a$
- Convert the binary number $\alpha_2$ into $a$ or $b$
  - Comparison leads to indecision: increase the precision and compare again
  - Ziv’s loop
Conversion algorithm: binary to decimal

Indecision with the rounding mode to the nearest

- \( a \) and \( b \) two representable decimal numbers in the current precision
- Convert the binary number \( \alpha_1 \) into \( a \) or \( b \)
  - Enclose \( \alpha_1 \) with two values
  - Compare the enclosure of \( \alpha_1 \) with the midpoint of \([a, b]\)
  - Round \( \alpha_1 \) to \( a \)
- Convert the binary number \( \alpha_2 \) into \( a \) or \( b \)
  - Comparison leads to indecision: increase the precision and compare again
  - Ziv’s loop
Conversion algorithm: binary to decimal

Indecision with the rounding mode to the nearest

- $a$ and $b$ two representable decimal numbers in the current precision
- Convert the binary number $\alpha_1$ into $a$ or $b$
  - Enclose $\alpha_1$ with two values
  - Compare the enclosure of $\alpha_1$ with the midpoint of $[a, b]$
  - Round $\alpha_1$ to $a$
- Convert the binary number $\alpha_2$ into $a$ or $b$
  - Comparison leads to indecision: increase the precision and compare again
  - Ziv’s loop
Conversion algorithm: binary to decimal

Indecision with the rounding mode to the nearest

- $a$ and $b$ two representable decimal numbers in the current precision
- Convert the binary number $\alpha_1$ into $a$ or $b$
  - Enclose $\alpha_1$ with two values
  - Compare the enclosure of $\alpha_1$ with the midpoint of $[a, b]$
  - Round $\alpha_1$ to $a$
- Convert the binary number $\alpha_2$ into $a$ or $b$
  - Comparison leads to indecision: increase the precision and compare again
  - Ziv’s loop
Conversion algorithm: binary to decimal

Indecision with the rounding mode to the nearest

- $a$ and $b$ two representable decimal numbers in the current precision
- Convert the binary number $\alpha_1$ into $a$ or $b$
  - Enclose $\alpha_1$ with two values
  - Compare the enclosure of $\alpha_1$ with the midpoint of $[a, b]$
  - Round $\alpha_1$ to $a$
- Convert the binary number $\alpha_2$ into $a$ or $b$
  - Comparison leads to indecision: increase the precision and compare again
  - Ziv’s loop
Multiplication algorithm

Simple algorithm without conversion

We want to compute $10^F \times n = (10^{F_1} \times n_1) \times (10^{F_2} \times n_2)$

- $F = F_1 + F_2$
- $t = \text{mpz\_mul}(n_1, n_2)$
- $\tilde{n} = \text{mpd\_set\_z}(t)$
- add the exponents
- multiplication of the significand with the GMP function
- Round the result to the decimal precision $k$
Addition algorithm

Simple algorithm without conversion

We want to compute $10^F \times n = (10^{F_1} \times n_1) + (10^{F_2} \times n_2)$

- if needed, align the two decimal numbers with multiplications (instead of divisions as in binary)
- set the exponent $F$
- $t = \text{mpz_add}(n_1, n_2)$ add the significands
- $\tilde{n} = \text{mpd_set_z}(t)$ round the result to the decimal precision $k$
Square root algorithm

Decimal (MPD)

\[ a = 10^F \times n \]
with \( k \) digit precision

Binary (MPFR)

\[ \text{mpfr_sqrt}() \]

\[ e_b = \text{decimal sqrt}(a) \]
with \( \log_2(10) \) \( p \) digit precision

\[ \text{mpd_sqrt}(a) \]
with \( k \) digit precision

\[ \text{final result} \]
Square root algorithm

**Decimal (MPD)**

\[ a = 10^F \times n \]

with \( k \) digit precision

**Binary (MPFR)**

\[ \alpha = 2^{E_\alpha} \times m_\alpha \]

with \( p = \log_{10}(2) \times k \) bit prec
Square root algorithm

Decimal (MPD)

\[ a = 10^F \times n \]
with \( k \) digit precision

conversion

Binary (MPFR)

\[ \alpha = 2^{E_\alpha} \times m_\alpha \]
with \( p = \log_{10}(2) \times k \) bit prec

\[ \beta = 2^{E_\beta} \times m_\beta \]
with \( p = \log_{10}(2) \times k \) bit prec

\text{mpfr\_sqrt}(\alpha)
Square root algorithm

**Decimal (MPD)**

\[ a = 10^F \times n \]
with \( k \) digit precision

\[ \tilde{b} = \text{decimal sqrt}(a) \]
with \( \log_2(10) \times p \) digit prec

**Binary (MPFR)**

\[ \alpha = 2^{E_\alpha} \times m_\alpha \]
with \( p = \log_{10}(2) \times k \) bit prec

\[ \text{mpfr_sqrt}(\alpha) \]

\[ \beta = 2^{E_\beta} \times m_\beta \]
with \( p = \log_{10}(2) \times k \) bit prec

Conversion
Square root algorithm

Decimal (MPD)

\[ a = 10^F \times n \]
with \( k \) digit precision

\[ \tilde{b} = \text{decimal sqrt}(a) \]
with \( \log_2(10) \times p \) digit prec

Binary (MPFR)

\[ \alpha = 2^{E_\alpha} \times m_\alpha \]
with \( p = \log_{10}(2) \times k \) bit prec

\[ \beta = 2^{E_\beta} \times m_\beta \]
with \( p = \log_{10}(2) \times k \) bit prec

Ziv’s loop

conversion

\( \text{mpfr_sqrt}(\alpha) \)

conversion

Final result
Square root algorithm

**Decimal (MPD)**

\[ a = 10^F \times n \]
with \( k \) digit precision

\[ \tilde{b} = \text{decimal sqrt}(a) \]
with \( \log_2(10) \times p \) digit prec

**Binary (MPFR)**

\[ \alpha = 2^{E_\alpha} \times m_\alpha \]
with \( p = \log_{10}(2) \times k \) bit prec

\[ \beta = 2^{E_\beta} \times m_\beta \]
with \( p = \log_{10}(2) \times k \) bit prec

Ziv's loop

\[ \checkmark \text{ Proof} \]

Conversion
Square root algorithm

Decimal (MPD)

\[ a = 10^F \times n \]
with \( k \) digit precision

\[ \tilde{b} = \text{decimal sqrt}(a) \]
with \( \log_2(10) \times p \) digit prec

\[ b = \text{mpd_sqrt}(a) \]
with \( k \) digit precision

Binary (MPFR)

\[ \alpha = 2^{E_{\alpha}} \times m_{\alpha} \]
with \( p = \log_{10}(2) \times k \) bit prec

\[ \beta = 2^{E_{\beta}} \times m_{\beta} \]
with \( p = \log_{10}(2) \times k \) bit prec

conversion

Ziv's loop

\( \checkmark \) Proof

comparison
Logarithm algorithm

\[
\log(a) = \log(\alpha(1 + \varepsilon)) = \log(\alpha) \cdot \left(1 + \frac{1}{\log(\alpha)} \cdot \frac{\log(1 + \varepsilon)}{\varepsilon} \cdot \varepsilon\right)
\]

Problem when \( \alpha \) is around 1

- Afar from 1
  - implement the decimal logarithm with \texttt{mpfr\_log} in the same way as the square root

- Around 1
  - compute in decimal \( b = a - 1 \) with \texttt{mpd\_sub}
  - convert the decimal \( b \) into the binary \( \beta \) with \texttt{mpd\_set\_fr}
  - perform the logarithm operation with \texttt{mpfr\_log1p}
    - this function implements \( \log 1p(\beta) = \log(1 + \beta) \)
Multiple Precision Decimal Interval

So far we have:

- MPFR: a correctly rounded multiple precision binary library
- MPFI: a correctly rounded interval multiple precision binary library
- MPD: a correctly rounded multiple precision decimal library
Multiple Precision Decimal Interval

So far we have:

- MPFR: a correctly rounded multiple precision binary library
- MPFI: a correctly rounded interval multiple precision binary library
- MPD: a correctly rounded multiple precision decimal library

We want to implement:

- MPDI: a correctly rounded interval multiple precision decimal library
Two solutions:

- Copy MPD methodology
  - convert the decimal interval \([a, b]\) into the binary one \([\alpha, \beta]\)
  - compute the binary result with MPFI functions

- Take inspiration from the MPFI code
  - use MPD functions with directed rounding to compute the decimal interval
List of implemented functions

MPD implementation
- init and clear functions
- set default and current precision
- mpd_set: convert in decimal from a decimal (MPD), a binary floating point (MPFR), a binary integer (GMP).
- mpd_get: convert a decimal into a binary
- arithmetic functions: add, sub, mul, div, sqrt.
- transcendental functions: log, exp.
- comparison functions: cmp, cmp_ui.
- absolute value and negation.

MPDI implementation
- init and clear
- set the prec, set the default prec
- mpdi_set: set a decimal interval MPDI from a decimal number MPD
- arithmetic functions: add, sub, mul, div, sqrt.
- transcendental functions: log, exp.
Outline

1. Introduction
2. Mechanisms behind MPD
3. Mechanisms behind MPDI
4. Benchmark
Do we have the expected output?

**Decimal MPD**

prec \( k = 8 \)

\[
\begin{align*}
a &= 10^{-1} \times 9.000 \\
b &= \text{mpd}_\log(a) \\
b &= -10^{-1} \times 1.0536052
\end{align*}
\]

**Binary MPFR**

prec \( p = 24 \)

\[
\begin{align*}
\alpha &= 2^{-1} \times 1.100110011... \\
\beta &= \text{mpfr}_\log(\alpha) \\
\beta &= -2^{-4} \times 1.010111110...
\end{align*}
\]
Do we have the expected output?

**Decimal MPD**

prec $k = 8$

\[
a = 10^{-1} \times 9.000
\]

\[
b = \text{mpd}_\log(a)
\]

\[
b = -10^{-1} \times 1.0536052
\]

\[
\tilde{b} = -10^{-1} \times 1.0536055
\]

**Binary MPFR**

prec $p = 24$

\[
\alpha = 2^{-1} \times 1.100110011\ldots
\]

\[
\beta = \text{mpfr}_\log(\alpha)
\]

\[
\beta = -2^{-4} \times 1.010111110\ldots
\]
ADD and MUL execution time

Comparison of decimal ADD algorithms

Comparison of decimal MUL algorithms

Time (s) vs. Precision $k$ for different algorithms:
- mpd
- mpdi
- mpd loop
EXP and LOG execution time

Comparison of decimal EXP algorithms

Comparison of decimal LOG algorithms

precision $k$

Time (s)

- mpfr
- mpfi
- mpd
- mpdi
Conclusion and future work

Conclusion

- Development of two libraries, MPD and MPDI of a correctly rounded, reliable and fast decimal multiple precision arithmetic.
- Proof of the conversion algorithm
- Implementation of the basic mathematical functions, and the exponential and logarithm

Perspectives

- Add other functions: trigonometric functions (cos, sin, tan...)
- Proof of the other algorithms
- Expand the set of tests
- Source code available on demand
Questions

Thank you!