TOWARDS CORRECTLY ROUNDED MIXED-RADIX IEEE754 ARITHMETIC

RAIM 2018 – Gif-sur-Yvette, France
November 12th, 2018

Clothilde Jeangoudoux and Christoph Lauter
clothilde.jeangoudoux@lip6.fr, christoph.lauter@lip6.fr

Sorbonne Université, CNRS, LIP6 UMR 7606

Motivations

```
int main() {
  _Decimal128 a = 0.1D;
  float b = 10.25;
  _Decimal64 c = -1.025D;
  double d;
  d = a * b + c;


  return 0;
}
```

Motivations

```c
int main() {
  _Decimal128 a = 0.1D;
  float b = 10.25;
  _Decimal64 c = -1.025D;
  double d;
  d = a * b + c;


  return 0;
}
```

What we would like to get:

Something close to `d = 0.0`

Motivations

```
int main() {
  _Decimal128 a = 0.1D;
  float b = 10.25;
  _Decimal64 c = -1.025D;
  double d;
  d = a * b + c;


  return 0;
}
```

What we would like to get:

Something close to `d = 0.0`

What we actually get:

- Nothing!

Motivations

```
int main() {
  _Decimal128 a = 0.1D;
  float b = 10.25;
  _Decimal64 c = -1.025D;
  double d;
  d = a * b + c;


  return 0;
}
```

What we would like to get:

Something close to `d = 0.0`

What we actually get:

- Nothing!
- Compilation with gcc 5.4 yields:
  `error: can't mix operands of decimal float and other float types`

Motivations

```
int main() {
  _Decimal128 a = 0.1D;
  float b = 10.25;
  _Decimal64 c = -1.025D;
  double d;

  d = ((double) a) * b +
      ((double) c);

  return 0;
}
```

What we would like to get:

Something close to `d = 0.0`

What we actually get:

- Nothing!
- Compilation with gcc 5.4 yields:
  `error: can't mix operands of decimal float and other float types`

Let's force it:

Motivations

```
int main() {
  _Decimal128 a = 0.1D;
  float b = 10.25;
  _Decimal64 c = -1.025D;
  double d;

  d = ((double) a) * b +
      ((double) c);

  return 0;
}
```

What we would like to get:

Something close to `d = 0.0`

What we actually get:

- Nothing!
- Compilation with gcc 5.4 yields:
  `error: can't mix operands of decimal float and other float types`

Let's force it:

- the result is $\mathtt{d} = \mathtt{0x1p} - 52 \approx 2.2204 \cdot 10^{-16}$

Motivations

```
int main() {
  _Decimal128 a = 0.1D;
  float b = 10.25;
  _Decimal64 c = -1.025D;
  double d;

  d = ((double) a) * b +
      ((double) c);

  return 0;
}
```

What we would like to get:

Something close to `d = 0.0`

What we actually get:

- Nothing!
- Compilation with gcc 5.4 yields:
  `error: can't mix operands of decimal float and other float types`

Let's force it:

- the result is $\mathtt{d} = \mathtt{0x1p} - 52 \approx 2.2204 \cdot 10^{-16}$
- as a reminder, the smallest subnormal number is $\mathtt{0x1p} - 1074 \approx 4.9407 \cdot 10^{-324}$

IEEE 754-2008 - FP formats

Binary format

$(-1)^s \cdot 2^E \cdot m$

s	$E + \text{bias}$	m
1 bit	W_E bits	$p - 1$ bits

Example, `binary` formats:

- significand: $2^{k_2 - 1} \leq m \leq 2^{k_2} - 1$
- exponent: $E_{\text{min}} \leq E \leq E_{\text{max}}$
  (with subnormals)

with $k_2 = p$.

Decimal format

$(-1)^s \cdot 10^F \cdot n$

s	$F + \text{bias}$	n
1 bit	$w + 5$ bits	$10 \times J$ bits

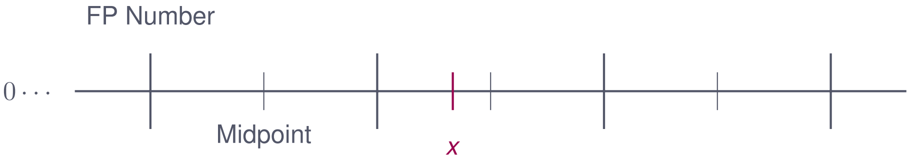
Example, `decimal`$\{k\}$ format:

- significand: $1 \leq n \leq 10^{k_{10}} - 1$
- exponent: $F_{\text{min}} \leq F \leq F_{\text{max}}$
- binary BID encoding

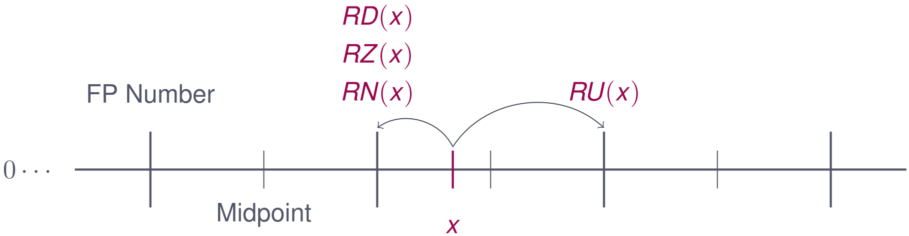
with $k_{10} = 9 \times k/32 - 2$.

IEEE 754-2008 - Rounding Modes

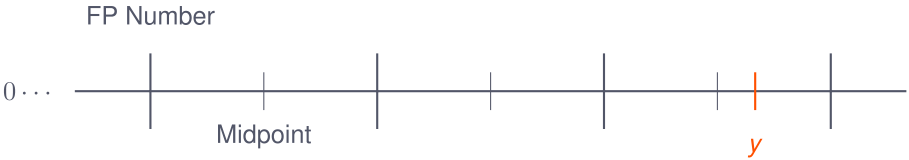
IEEE 754-2008 - Rounding Modes



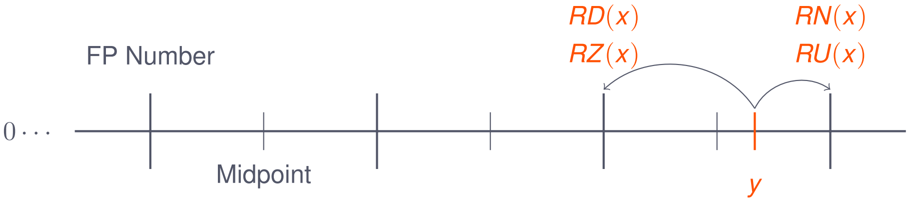
IEEE 754-2008 - Rounding Modes



IEEE 754-2008 - Rounding Modes



IEEE 754-2008 - Rounding Modes



IEEE 754-2008 - Arithmetic Operations

Definitions and properties

- basic arithmetic operations ($+$, $\times$, $\div$, *FMA*...)
- exceptions and flags
- heterogenous operations
  - same base, different format/precision
  - e.g. $\texttt{binary32} = \texttt{binary32} \times \texttt{binary64}$

IEEE 754-2008 - Arithmetic Operations

Definitions and properties

- basic arithmetic operations ($+$, $\times$, $\div$, *FMA*...)
- exceptions and flags
- heterogenous operations
  - same base, different format/precision
  - e.g. $\texttt{binary32} = \texttt{binary32} \times \texttt{binary64}$

Goal: mixed-radix operations

Enrich the IEEE 754-2008 standard with heterogenous operations in base 2 and 10.

Designing Mixed-Radix Operations

Considered formats

- Binary: `binary32`, `binary64`, `binary128`
- Decimal: `binary64`, `binary128`

Considering all basic operations: ≈ 1120 **operations**

Designing Mixed-Radix Operations

Considered formats

- Binary: `binary32`, `binary64`, `binary128`
- Decimal: `binary64`, `binary128`

Considering all basic operations: ≈ 1120 **operations**

Goal: Automatically generate mixed-radix basic operations

Compromise between efficiency and implementation effort.

Designing Mixed-Radix Operations

Definition: Fused Multiply and Add

$$\texttt{FMA}(a, b, c) = \circ(a \times b + c) \qquad \texttt{where } \circ \in \{\texttt{RN}, \texttt{RZ}, \texttt{RU}, \texttt{RD}\}$$

Designing Mixed-Radix Operations

Definition: Fused Multiply and Add

$$\texttt{FMA}(a, b, c) = \circ(a \times b + c) \qquad \texttt{where } \circ \in \{\mathrm{RN}, \mathrm{RZ}, \mathrm{RU}, \mathrm{RD}\}$$

Correctly Rounded Mixed Radix FMA

Designing Mixed-Radix Operations

Definition: Fused Multiply and Add

$$\mathtt{FMA}(a, b, c) = \circ(a \times b + c) \qquad \texttt{where } \circ \in \{\mathtt{RN}, \mathtt{RZ}, \mathtt{RU}, \mathtt{RD}\}$$

Correctly Rounded Mixed Radix FMA

- correctly rounded mixed-radix $+, -, \times$

Designing Mixed-Radix Operations

Definition: Fused Multiply and Add

$$\mathtt{FMA}(a, b, c) = \circ(a \times b + c) \qquad \mathtt{where} \circ \in \{\mathrm{RN}, \mathrm{RZ}, \mathrm{RU}, \mathrm{RD}\}$$

Correctly Rounded Mixed Radix FMA

- correctly rounded mixed-radix $+, -, \times$
- correctly rounded mixed-radix $\div, \sqrt{\ }$

Designing Mixed-Radix Operations

Definition: Fused Multiply and Add

$$\mathtt{FMA}(a, b, c) = \circ(a \times b + c) \qquad \qquad \mathtt{where}\ \circ \in \{\mathrm{RN}, \mathrm{RZ}, \mathrm{RU}, \mathrm{RD}\}$$

Correctly Rounded Mixed Radix FMA

- correctly rounded mixed-radix $+, -, \times$
- correctly rounded mixed-radix $\div, \sqrt{}$
  - assuming we can represent the midpoint between two FP-numbers, f

Designing Mixed-Radix Operations

Definition: Fused Multiply and Add

$$\mathtt{FMA}(a, b, c) = \circ(a \times b + c) \qquad \texttt{where } \circ \in \{\mathrm{RN}, \mathrm{RZ}, \mathrm{RU}, \mathrm{RD}\}$$

Correctly Rounded Mixed Radix FMA

- correctly rounded mixed-radix $+, -, \times$
- correctly rounded mixed-radix $\div, \sqrt{\ }$
  - assuming we can represent the midpoint between two FP-numbers, f
    - the rounding of $\circ_k\left(\frac{x}{y}\right)$ boilds down to computing the sign of $y \times f - x$

Designing Mixed-Radix Operations

Definition: Fused Multiply and Add

$$\mathtt{FMA}(a, b, c) = \circ(a \times b + c) \qquad\qquad \texttt{where } \circ \in \{\mathtt{RN}, \mathtt{RZ}, \mathtt{RU}, \mathtt{RD}\}$$

Correctly Rounded Mixed Radix FMA

- correctly rounded mixed-radix $+, -, \times$
- correctly rounded mixed-radix $\div, \sqrt{\ }$
  - assuming we can represent the midpoint between two FP-numbers, f
    - the rounding of $\circ_k\left(\frac{x}{y}\right)$ boilds down to computing the sign of $y \times f - x$
    - the rounding of $\circ_k\left(\sqrt{x}\right)$ boilds down to computing the sign of $f \times f - x$

Mixed Radix Arithmetic

What are the operations available in binary and decimal format?

Mixed Radix Arithmetic

What are the operations available in binary and decimal format?

- Conversions as defined in IEEE 754

Mixed Radix Arithmetic

What are the operations available in binary and decimal format?

- Conversions as defined in IEEE 754
- Exact comparisons
  Comparison between binary and decimal floating-point numbers, N. Brisebarre, C. L., M. Mezzarobba, J.-M. Muller [2016]
  - study of the feasibility of mixed-radix comparison,
  - implementation of two algorithms that have been proven and thoroughly tested,

Mixed Radix Arithmetic

What are the operations available in binary and decimal format?

- Conversions as defined in IEEE 754
- Exact comparisons
  Comparison between binary and decimal floating-point numbers, N. Brisebarre, C. L., M. Mezzarobba, J.-M. Muller [2016]
  - study of the feasibility of mixed-radix comparison,
  - implementation of two algorithms that have been proven and thoroughly tested,

Goal: mixed-radix FMA

- an emerging need for mixed-radix arithmetic
- implementation of all basic arithmetic operations with one slightly more precise FMA

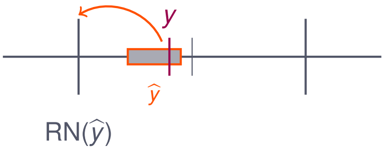
Table Maker's Dilemma

Example: consider the exact transcendental number $y = e^x$ and the computed result $\widehat{y} = \exp(x)$.

Table Maker's Dilemma

Example: consider the exact transcendental number $y = e^x$ and the computed result $\widehat{y} = \exp(x)$.

Correct Rounding in the easy case

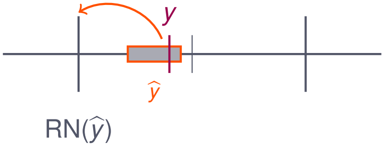


- enough accuracy

Table Maker's Dilemma

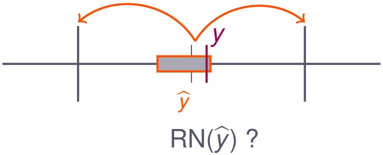
Example: consider the exact transcendental number $y = e^x$ and the computed result $\widehat{y} = \exp(x)$.

Correct Rounding in the easy case



- enough accuracy

Correct Rounding in the hard case



- not enough accuracy, but how much?

SORBONNE UNIVERSITÉ SAFRAN

Classical Binary FMA

Algorithm 1 Binary FMA $d = \circ(a \times b + c)$

1: **if** $\frac{a \times b}{c} \notin \left[\frac{1}{2}, 2\right]$ **then**
2: d = *farpath*_addition($a \times b$, c)
3: **else**
4: d = *nearpath*_subtraction($a \times b$, c)
5: **end if**

far-path addition

- when $\frac{a \times b}{c} \notin \left[\frac{1}{2}, 2\right]$
- simple logic with sticky guard bit

near-path subtraction

- when $\frac{a \times b}{c} \in \left[\frac{1}{2}, 2\right]$
- **Sterbenz's lemma**: $(a \times b) - c$ is exactly representable

Mixed-Radix Inexact Cancellation Cases

Mixed-Radix *near-path* subtraction is INEXACT!

- at a certain precision
- cannot compute the result with enough accuracy for correct rounding

Mixed-Radix Inexact Cancellation Cases

Mixed-Radix *near-path* subtraction is INEXACT!

- at a certain precision
- cannot compute the result with enough accuracy for correct rounding

Mixed-Radix *far-path* addition is not always exact!

- no simple sticky bit

Mixed-Radix Inexact Cancellation Cases

Mixed-Radix *near-path* subtraction is INEXACT!

- at a certain precision
- cannot compute the result with enough accuracy for correct rounding

Mixed-Radix *far-path* addition is not always exact!

- no simple sticky bit

Obervation

Mixed-radix addition almost always inexact.

Overcoming the TDM - Mixed-Radix unified format

Obervations

10 is divisible by 2.

Overcoming the TDM - Mixed-Radix unified format

Obervations

10 is divisible by 2.

Binary and decimal formats can be unified as

$$a = (-1)^{s_a} \cdot 2^{N_a} \cdot 5^{P_a} \cdot t_a;$$

Overcoming the TDM - Mixed-Radix unified format

Obervations

10 is divisible by 2.

Binary and decimal formats can be unified as

$$a = (-1)^{s_a} \cdot 2^{N_a} \cdot 5^{P_a} \cdot t_a;$$

with

$$2^{k_2'-1} \leq |t_a| < 2^{k_2'}; \; t_a \in \mathbb{Z}$$

Overcoming the TDM - Mixed-Radix unified format

Obervations

10 is divisible by 2.

Binary and decimal formats can be unified as

$$a = (-1)^{s_a} \cdot 2^{N_a} \cdot 5^{P_a} \cdot t_a;$$

with

$$2^{k_2'-1} \leq |t_a| < 2^{k_2'};\ t_a \in \mathbb{Z}$$

$$\min(F^{\min} - k_{10} - k_2', E^{\min} - k_2 - k_2') \leq N_a \leq \max(F^{\max} - k_2' + 1, E^{\max} - k_2' + 1)$$

Overcoming the TDM - Mixed-Radix unified format

Obervations

10 is divisible by 2.

Binary and decimal formats can be unified as

$$a = (-1)^{s_a} \cdot 2^{N_a} \cdot 5^{P_a} \cdot t_a;$$

with

$$2^{k_2'-1} \leq |t_a| < 2^{k_2'};\ t_a \in \mathbb{Z}$$

$$\min(F^{\text{min}} - k_{10} - k_2', E^{\text{min}} - k_2 - k_2') \leq N_a \leq \max(F^{\text{max}} - k_2' + 1, E^{\text{max}} - k_2' + 1)$$

$$F^{\text{min}} - k_{10} + \Lambda_5^{\text{min}} + 1 \leq P_a \leq F^{\text{max}} + \Lambda_5^{\text{max}} + 1;\ N_a, P_a \in \mathbb{Z}.$$

Overcoming the TDM - Mixed-Radix unified format

Obervations

10 is divisible by 2.

Binary and decimal formats can be unified as

$$a = (-1)^{s_a} \cdot 2^{N_a} \cdot 5^{P_a} \cdot t_a;$$

with

$$2^{k_2'-1} \leq |t_a| < 2^{k_2'};\; t_a \in \mathbb{Z}$$
$$\min(F^{\min} - k_{10} - k_2', E^{\min} - k_2 - k_2') \leq N_a \leq \max(F^{\max} - k_2' + 1, E^{\max} - k_2' + 1)$$
$$F^{\min} - k_{10} + \Lambda_5^{\min} + 1 \leq P_a \leq F^{\max} + \Lambda_5^{\max} + 1;\; N_a, P_a \in \mathbb{Z}.$$

$$k_2' = \max(k_2 + 1, \lceil \log_2(10^{k_{10}} - 1) \rceil + 1), \Lambda_5^{\min} = \left\lceil \log_5 \left(\frac{1}{2^{k_2'} - 1} \right) \right\rceil \text{ and } \Lambda_5^{\max} = \left\lfloor \log_5 \left(\frac{1}{2^{k_2'-1}} \right) \right\rfloor.$$

Overcoming the TDM

Obervations

At a certain precision, binary to decimal conversion becomes exact.

Overcoming the TDM

Obervations

At a certain precision, binary to decimal conversion becomes exact.

Bound on the worst case of cancellation

- occurs when $(a \times b) - c$ is relatively small
- if $a \times b = 2^L \cdot 5^M \cdot s$ and $c = 2^N \cdot 5^P \cdot t$

$$\left| \frac{s}{t} - 2^{N-L} \cdot 5^{P-M} \right| \geq \eta$$

- computed using one sided approximations

	η
binary64, decimal64	$2^{-177.61}$
binary128, decimal128	$2^{-360.25}$

Performances issues of this exact addition

Size of the long accumulator

- acutal computation $\alpha = (a \times b) + c - f$
- a, b and c inputs of the FMA, $(a \times b)$ the exact multiplication bounded into the internal mixed-radix format
- f the closest midpoint bounded into the internal mixed-radix format

	AC^{max} (bits)	AC^{max} (words)	free bits
binary64, decimal64	4225	67	63
binary128, decimal128	62158	972	14

Performances issues of this exact addition

Size of the long accumulator

- acutal computation $\alpha = (a \times b) + c - f$
- a, b and c inputs of the FMA, $(a \times b)$ the exact multiplication bounded into the internal mixed-radix format
- f the closest midpoint bounded into the internal mixed-radix format

	AC^{max} (bits)	AC^{max} (words)	free bits
binary64, decimal64	4225	67	63
binary128, decimal128	62158	972	14

Obervation

In a lot of cases, a quick and not so accurate addition can be enough to perform correct rounding in the output format.

FMA Mixed Radix Algorithm

Algorithm 2 Mixed-Radix FMA $d = \circ(a \times b + c)$

```
Multiplication ψ ← a × b
if it is an "addition" or ψ/c ∉ [1/2; 2] then
    φ ← "far-path" binary addition
else
    φ ← "near-path" binary subtraction
end if
ρ ← Conversion of φ to the output format
if ρ can round correctly then
    return  d ← ρ correctly rounded to output format
else
    Compute integer rounding boundary significand f
    α ← Exact decimal addition
    Correct ρ using f and the sign of α
    return  d ← ρ correctly rounded to output format
end if
```

Test Environment and Reference implementations

Test Environment

- Intel i7-7500U quad-core processor
- clocked at maximally 2.7GHz
- running Debian/GNU Linux 4.9.0-5 in x86-64 mode

Test Environment and Reference implementations

Test Environment

- Intel i7-7500U quad-core processor
- clocked at maximally 2.7GHz
- running Debian/GNU Linux 4.9.0-5 in x86-64 mode

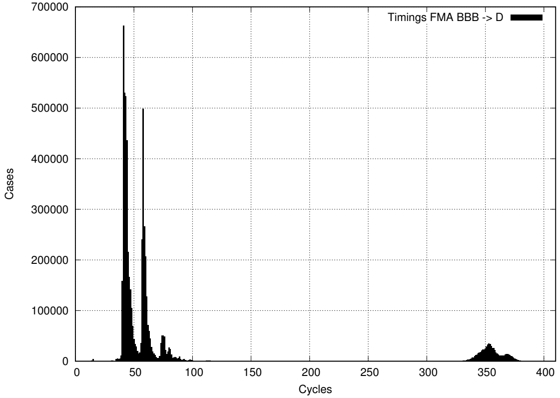
GNU Multiple Precision Library (GMP)

- mixed-radix FMA designed in a limited timeframe
- using GMP rational numbers
- Goal: reasonabely fast but easy to design

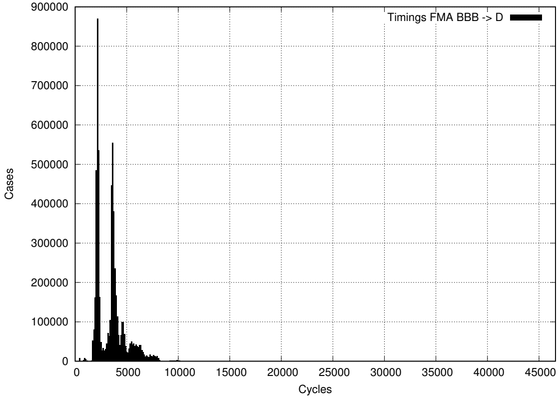
Sollya

- exact representation of numerical expressions
- evaluated at any precision without spurious rounding

Performance Testing



Our implementation



GMP reference implementation

Conclusion and Perspectives

Correctly Rounded Mixed-Radix FMA

- two formats: `binary64` and `decimal64`
- pen and paper proof of the algorithm
- overcoming the TDM and worst case of cancellation in the mixed-radix case
- implementation faster than expected and extensively tested

Going further

- finalize code generator implementation
- optimize FMA for heterogenous precision

Thank you! Questions?