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

  return 0;
}
Motivations

What we would like to get:
Something close to $d = 0.0$

```c
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 \( d = 0.0 \)

What we actually get:

- Nothing!
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 \( d = 0.0 \)

What we actually get:

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

```c
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:
```c
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 \( d = 0x1p - 52 \approx 2.2204 \times 10^{-16} \)
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 \( d = 0x1p - 52 \approx 2.2204 \cdot 10^{-16} \)
- as a reminder, the smallest subnormal number is \( 0x1p - 1074 \approx 4.9407 \cdot 10^{-324} \)
**IEEE 754-2008 - FP formats**

**Binary format**

\[ (-1)^s \cdot 2^E \cdot m \]

- **s**: 1 bit
- **E + bias**: \( W_E \) bits
- **m**: \( p - 1 \) 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**: 1 bit
- **F + bias**: \( w + 5 \) bits
- **n**: \( 10 \times J \) 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

FP Number

Midpoint
IEEE 754-2008 - Rounding Modes

FP Number

0... | | | Midpoint | x |
IEEE 754-2008 - Rounding Modes

FP Number

0... Midpoint

\( RD(x) \)
\( RZ(x) \)
\( RN(x) \)
\( RU(x) \)
IEEE 754-2008 - Rounding Modes

FP Number

0... Midpoint y
IEEE 754-2008 - Rounding Modes

FP Number

0... Midpoint

RN(x)
RZ(x)
RD(x)
RU(x)

RN(x)
RZ(x)
RD(x)
RU(x)

y
Definitions and properties

- basic arithmetic operations (+, ×, ÷, FMA...)
- exceptions and flags
- heterogenous operations
  - same base, different format/precision
  - e.g. binary32 = binary32 × binary64
IEEE 754-2008 - Arithmetic Operations

Definitions and properties

- basic arithmetic operations (+, ×, ÷, FMA...)
- exceptions and flags
- heterogenous operations
  - same base, different format/precision
  - e.g. binary32 = binary32 × 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: \( \approx 1120 \text{ operations} \)
Designing Mixed-Radix Operations

Considered formats

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

Considering all basic operations: $\approx 1120$ operations

Goal: Automatically generate mixed-radix basic operations

Compromise between efficiency and implementation effort.
Designing Mixed-Radix Operations

Definition: Fused Multiply and Add

\[ \text{FMA}(a, b, c) = \circ(a \times b + c) \]

where \( \circ \in \{\text{RN, RZ, RU, RD}\} \)
Designing Mixed-Radix Operations

Definition: Fused Multiply and Add

\[ \text{FMA}(a, b, c) = \circ(a \times b + c) \quad \text{where} \quad \circ \in \{\text{RN, RZ, RU, RD}\} \]

Correctly Rounded Mixed Radix FMA
Designing Mixed-Radix Operations

Definition: Fused Multiply and Add

\[ \text{FMA}(a, b, c) = \circ(a \times b + c) \]

where \( \circ \in \{\text{RN, RZ, RU, RD}\} \)

Correctly Rounded Mixed Radix FMA

- correctly rounded mixed-radix +, −, ×
Designing Mixed-Radix Operations

Definition: Fused Multiply and Add

\[
\text{FMA}(a, b, c) = \circ(a \times b + c)
\]
where \(\circ \in \{\text{RN, RZ, RU, RD}\}\)

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

\[ FMA(a, b, c) = \circ(a \times b + c) \quad \text{where} \quad \circ \in \{\text{RN, RZ, RU, RD}\} \]

Correctly Rounded Mixed Radix FMA

- correctly rounded mixed-radix +, −, ×
- correctly rounded mixed-radix ÷, √
  > assuming we can represent the midpoint between two FP-numbers, f
Designing Mixed-Radix Operations

Definition: Fused Multiply and Add

\[ \text{FMA}(a, b, c) = \circ(a \times b + c) \]

where \( \circ \in \{\text{RN}, \text{RZ}, \text{RU}, \text{RD}\} \)

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 \( \circ_k \left( \frac{x}{y} \right) \) boils down to computing the sign of \( y \times f - x \)
Designing Mixed-Radix Operations

Definition: Fused Multiply and Add

\[ \text{FMA}(a, b, c) = \circ(a \times b + c) \quad \text{where} \quad \circ \in \{\text{RN, RZ, RU, RD}\} \]

Correctly Rounded Mixed Radix FMA

- correctly rounded mixed-radix +, −, ×
- correctly rounded mixed-radix ÷, √
  - assuming we can represent the midpoint between two FP-numbers, \( f \)
    - the rounding of \( \circ_k \left( \frac{x}{y} \right) \) boils down to computing the sign of \( y \times f - x \)
    - the rounding of \( \circ_k \left( \sqrt{x} \right) \) boils 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 $\tilde{y} = \exp(x)$. 

Correct Rounding in the easy case

Correct Rounding in the hard case

$RN(\tilde{y})$ ?

not enough accuracy, but how much?
Table Maker’s Dilemma

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

Correct Rounding in the easy case

RN($\hat{y}$)

足够的精度
Table Maker’s Dilemma

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

Correct Rounding in the easy case

$\text{RN}(\hat{y})$

- enough accuracy

Correct Rounding in the hard case

$\text{RN}(\hat{y})$

- not enough accuracy, but how much?
Algorithm 1 Binary FMA $d = \circ(a \times b + c)$

1: if $\frac{a \times b}{c} \not\in \left[\frac{1}{2}, 2\right]$ then
2: \hspace{1em} $d = \text{farpath\_addition}(a \times b, c)$
3: else
4: \hspace{1em} $d = \text{nearpath\_subtraction}(a \times b, c)$
5: end if

far-path addition

- when $\frac{a \times b}{c} \not\in \left[\frac{1}{2}, 2\right]$
- 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

**Observation**

Mixed-radix addition almost always inexact.
Overcoming the TDM - Mixed-Radix unified format

<table>
<thead>
<tr>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 is divisible by 2.</td>
</tr>
</tbody>
</table>
Overcoming the TDM - Mixed-Radix unified format

Observations

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

**Observations**

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

\[
2^{k_2'} - 1 \leq |t_a| < 2^{k_2'}; \quad t_a \in \mathbb{Z}
\]
Overcoming the TDM - Mixed-Radix unified format

**Observations**

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

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

\[ \min(F_{\text{min}} - k_{10} - k_2', E_{\text{min}} - k_2 - k_2') \leq N_a \leq \max(F_{\text{max}} - k_2' + 1, E_{\text{max}} - k_2' + 1) \]
Overcoming the TDM - Mixed-Radix unified format

Observations

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

\[ 2^{k'_2 - 1} \leq |t_a| < 2^{k'_2}; \quad t_a \in \mathbb{Z} \]

\[ \min(F_{\text{min}} - k_{10} - k'_2, E_{\text{min}} - k_2 - k'_2) \leq N_a \leq \max(F_{\text{max}} - k'_2 + 1, E_{\text{max}} - k'_2 + 1) \]

\[ F_{\text{min}} - k_{10} + \Lambda_{5}^{\text{min}} + 1 \leq P_a \leq F_{\text{max}} + \Lambda_{5}^{\text{max}} + 1; \quad N_a, P_a \in \mathbb{Z}. \]
Overcoming the TDM - Mixed-Radix unified format

Observations

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

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

\[ \min(F_{\text{min}} - k_{10} - k_2', E_{\text{min}} - k_2 - k_2') \leq N_a \leq \max(F_{\text{max}} - k_2' + 1, E_{\text{max}} - k_2' + 1) \]

\[ F_{\text{min}} - k_{10} + \Lambda_5^{\text{min}} + 1 \leq P_a \leq F_{\text{max}} + \Lambda_5^{\text{max}} + 1; \quad N_a, P_a \in \mathbb{Z}. \]

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

**Observations**

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

Observations

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

<table>
<thead>
<tr>
<th></th>
<th>(\eta)</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary64, decimal64</td>
<td>(2^{-177.61})</td>
</tr>
<tr>
<td>binary128, decimal128</td>
<td>(2^{-360.25})</td>
</tr>
</tbody>
</table>
Performances issues of this exact addition

Size of the long accumulator

- actual 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

<table>
<thead>
<tr>
<th>Format</th>
<th>$AC_{\text{max}}$ (bits)</th>
<th>$AC_{\text{max}}$ (words)</th>
<th>Free bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary64, decimal64</td>
<td>4225</td>
<td>67</td>
<td>63</td>
</tr>
<tr>
<td>binary128, decimal128</td>
<td>62158</td>
<td>972</td>
<td>14</td>
</tr>
</tbody>
</table>
Performances issues of this exact addition

Size of the long accumulator

- actual 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

<table>
<thead>
<tr>
<th></th>
<th>$AC_{\text{max}}$ (bits)</th>
<th>$AC_{\text{max}}$ (words)</th>
<th>free bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>binary64, decimal64</td>
<td>4225</td>
<td>67</td>
<td>63</td>
</tr>
<tr>
<td>binary128, decimal128</td>
<td>62158</td>
<td>972</td>
<td>14</td>
</tr>
</tbody>
</table>

Observation

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 \( d = \circ(a \times b + c) \)

1: Multiplication \( \psi \leftarrow a \times b \)

2: if it is an “addition” or \( \frac{\psi}{c} \notin \left[ \frac{1}{2}; 2 \right] \) then

3: \( \phi \leftarrow \text{“far-path” binary addition} \)

4: else

5: \( \phi \leftarrow \text{“near-path” binary subtraction} \)

6: end if

7: \( \rho \leftarrow \text{Conversion of } \phi \text{ to the output format} \)

8: if \( \rho \) can round correctly then

9: return \( d \leftarrow \rho \) correctly rounded to output format

10: else

11: Compute integer rounding boundary significand \( f \)

12: \( \alpha \leftarrow \text{Exact decimal addition} \)

13: Correct \( \rho \) using \( f \) and the sign of \( \alpha \)

14: return \( d \leftarrow \rho \) correctly rounded to output format

15: 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: reasonably 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?