A CORRECTLY ROUNDED MIXED-RADIX FUSED-MULTIPLY-ADD

ARITH25 - Amherst, USA
June 25th, 2018

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

Sorbonne Université, CNRS, LIP6 UMR 7606
Motivations

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

    return 0;
}
```

What we would like to get:

```
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 = 0x1p522
```

as a reminder, the smallest subnormal number is

```
0x1p1074
```

1/16 Clothilde JEANGOUĐOUX / ARITH 25 / June 25th, 2018 / MIXED-RADIX FMA
Motivations

What we would like to get:
Something close to \( d = 0.0 \)

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

    return 0;
}
```
Motivations

```c
int main() {
    _Decimal64 a = 0.1D;
    double 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() {
    _Decimal64 a = 0.1D;
    double 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

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() {
  _Decimal64 a = 0.1D;
  double b = 10.25;
  _Decimal64 c = -1.025D;
  double d;
  d = ((double) a) * b + ((double) c);
  return 0;
}
```
Motivations

```c
int main() {
    _Decimal64 a = 0.1D;
    double 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 \( d = 0x1p_{-52} \approx 2.2204 \cdot 10^{-16} \)
Motivations

```c
int main() {
  _Decimal64 a = 0.1D;
  double 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
- \(E\): \(W_E\) bits
- \(m\): \(p - 1\) bits

Example, binary64 format:
- significand: \(2^{52} \leq m \leq 2^{53} - 1\)
- exponent: \(-1074 \leq E \leq 971\) (with subnormals)

**Decimal format**
\[(−1)^s \cdot 10^F \cdot n\]

- \(s\): 1 bit
- \(F\): \(w + 5\) bits
- \(n\): \(10 \times J\) bits

Example, decimal64 format:
- significand: \(1 \leq n \leq 10^{16} - 1\)
- exponent: \(-398 \leq F \leq 369\)
- binary BID encoding
IEEE 754-2008 - Rounding Modes

FP Number

0...  
Midpoint  

x
IEEE 754-2008 - Rounding Modes

FP Number

0...

Midpoint

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

x
IEEE 754-2008 - Rounding Modes

FP Number

0⋯

Midpoint

y
IEEE 754-2008 - Rounding Modes

FP Number

0... Midpoint

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

y
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
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. binary32 = binary32 $\times$ binary64

Goal: mixed-radix operations

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

Definition

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

Definition

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

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

Why a Mixed-Radix FMA?
Fused Multiply and Add

Definition

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

Why a Mixed-Radix FMA?

- correctly rounded FMA \( \Rightarrow \) correctly rounded \(+, -, \times\)
Fused Multiply and Add

Definition

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

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

Why a Mixed-Radix FMA?

- correctly rounded FMA \( \Rightarrow \) correctly rounded \(+, -, \times\)
- but also, with few more bits of precision, correctly rounded FMA \( \Rightarrow \) correctly rounded \(\div, \sqrt{\}\)
Fused Multiply and Add

Definition

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

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

Why a Mixed-Radix FMA?

- correctly rounded FMA \( \Rightarrow \) correctly rounded \(+, -, \times\)
- but also, with few more bits of precision, correctly rounded FMA \( \Rightarrow \) correctly rounded \(\div, \sqrt{\} \)
  - assuming we can represent the midpoint between two FP-numbers
Fused Multiply and Add

Definition

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

Why a Mixed-Radix FMA?

- correctly rounded FMA ⇒ correctly rounded +, −, ×
- but also, with few more bits of precision, correctly rounded FMA ⇒ correctly rounded \(\div, \sqrt{\cdot}\)
  - assuming we can represent the midpoint between two FP-numbers
  - compromise between efficiency and implementation effort, e.g. for binary64 and decimal64 combinations:
    - 5 operations +, −, ×, \(\div, \sqrt{\cdot}\) in 20 mixed-radix versions
    - 1 FMA operation in 10 versions
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 $\tilde{y} = \exp(x)$. 
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

$\hat{y}$

$y = \exp(x)$

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

Correct Rounding in the hard case

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

\( y = \exp(x) \)

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

\( y = \exp(x) \)

- enough accuracy
- not enough accuracy, but how much?
Classical Binary FMA

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

1: if $\frac{a \times b}{c} \notin \left[\frac{1}{2}, 2\right]$ then
2:   $d = \text{farpath}\_\text{addition}(a \times b, c)$
3: else
4:   $d = \text{nearpath}\_\text{subtraction}(a \times b, c)$
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

near-path subtraction is INEXACT!

- at a certain precision
- cannot compute the result with enough accuracy for correct rounding
Mixed-Radix Inexact Cancellation Cases

**near-path** subtraction is INEXACT!
- at a certain precision
- cannot compute the result with enough accuracy for correct rounding

**far-path** addition is not always exact!
- no simple sticky bit
Mixed-Radix Inexact Cancellation Cases

near-path subtraction is INEXACT!

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

far-path addition is not always exact!

- no simple sticky bit

Observation

Mixed-radix addition almost always inexact.
Overcoming the TDM

Observations

- 10 is divisible by 2
- at a certain precision, binary to decimal conversion becomes exact.
Overcoming the TDM

Observations

- 10 is divisible by 2
- at a certain precision, binary to decimal conversion becomes exact.

Mixed-Radix unified format

Binary64 and decimal64 formats can be unified as

\[ 2^J \cdot 5^K \cdot r \]

with \( 2^{54} \leq |r| < 2^{55}, \quad r \in \mathbb{Z} \)

\(-1130 \leq J \leq 969; \quad -421 \leq K \leq 385; \quad J, K \in \mathbb{Z}. \)
Overcoming the TDM

<table>
<thead>
<tr>
<th>Observations</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 is divisible by 2</td>
</tr>
<tr>
<td>at a certain precision, binary to decimal conversion becomes exact.</td>
</tr>
</tbody>
</table>

**Mixed-Radix unified format**

Binary64 and decimal64 formats can be unified as $2^J \cdot 5^K \cdot r$

with $2^{54} \leq |r| < 2^{55}$; $r \in \mathbb{Z}$

$-1130 \leq J \leq 969; -421 \leq K \leq 385$;

$J, K \in \mathbb{Z}$.

**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 $c = 2^N \cdot 5^P \cdot t$

$$\left| \frac{s}{t} - 2^{N-L} \cdot 5^{P-M} \right| \leq \eta = 2^{-177.61}$$

- computed using one sided approximations
Performances issues of this exact addition

Size of the 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
- We are sure that we can compute $\alpha$ and store it on 4225 bits, that is 67 words of 64 bits, leaving 63 "free" bits.
Performances issues of this exact addition

Size of the 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
- We are sure that we can compute $\alpha$ and store it on 4225 bits, that is 67 words of 64 bits, leaving 63 "free" bits.

Observation

In a lot of cases, a quick and not so accurate addition can be enough to perform correct rounding in the output format.
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: $\phi \leftarrow “far\text{-}path”$ binary addition
4: else
5: $\phi \leftarrow “near\text{-}path”$ binary subtraction
6: end if
7: $\rho \leftarrow$ Conversion of $\phi$ 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: $\alpha \leftarrow$ 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

- more formats!
- mixed-radix FMA of heterogeneous precision
Thank you! Questions?