Emulation of the FMA and ADD3 in rounding-to-nearest floating-point arithmetic

Stef Graillat Jean-Michel Muller

FJWNC 2025, Paris, mar. 12–14
日本の友人のためにフランスへようこそ


The FMA instruction

- correctly rounded evaluation of $ab + c$;
- very useful:
  - software implementation of correctly rounded $\div$ and $\sqrt{}$;
  - (in general) more accurate evaluation of dot products and polynomials;
  - fast and accurate matrix operations (see next talk!);
  - accurate implementation of transcendental functions: $a_0 + x\rho$, where $\rho = a_1 + x(a_2 + x(a_3 + \cdots))$ and $|x|$ small.
- specified by IEEE-754 since 2008 $\rightarrow$ implemented in most general-purpose computing environments;
- notable exceptions: Java Virtual Machine, WebAssembly, many microcontrollers units used for instance in automotive applications.

ADD3

- correctly rounded evaluation of $a + b + c$;
- not specified by IEEE-754 $\rightarrow$ not provided by computing environments;
- and yet, would greatly help
  - final rounding step in correctly-rounded elementary functions (Lauter, 2017);
  - implementation of double-word and triple-word arithmetics (where a high precision number is represented by the unevaluated sum of 2 or 3 FP numbers);
- a fast hardware ADD3 would be a nice replacement for 2Sum (N/A here: we are going to use 2Sum to emulate ADD3!).

Aim of this work

- software emulation of the FMA, ADD3, and the error of these operations;

  (errors: interesting for building compensated algorithms)
- high-level algorithms: FP operations/comparisons only (no use of the internal binary representation of the FP numbers);
- binary, precision-p, rounded-to-nearest FP arithmetic, with unbounded exponent range

  ($\rightarrow$ our results apply to "real life" FP arithmetic provided underflow/overflow do not occur);

Notation

- set $\mathbb{F}$ of the binary, precision-p FP numbers:

$$x = M_x \cdot 2^{e_x - p + 1},$$

  where $M_x, e_x \in \mathbb{Z}$, and either $M_x = 0$, or $2^{p-1} \leq |M_x| \leq 2^p - 1$;
- M_x is the integral significand of x;
- $\mathbb{F}^* = \mathbb{F} \backslash \{0\}$;
- RN: round-to-nearest, ties-to-even rounding function

$$\texttt{x = y + z} \rightarrow x = \mathrm{RN}(y + z)$$

- midpoints the numbers where the value of RN changes. Exactly halfway between two consecutive FP numbers;
- unit round-off: $u = 2^{-p}$. Bounds the relative error due to rounding;

Notation

- $\mathrm{ulp}(t)$ (for $t \in \mathbb{R}$) defined as

$$\begin{cases} 0 & \text{if } t = 0, \\ 2^{\lfloor \log_2 |t| \rfloor - p + 1} & \text{otherwise;} \end{cases}$$

  - If $t \notin \mathbb{F}$, $\mathrm{ulp}(t)$ is the distance between the two consecutive FP numbers that surround t;
  - if $x \in \mathbb{F}$, x is an integer multiple of $\mathrm{ulp}(x)$;
- x is a double-word (DW) number if it is an unevaluated sum $x = x_h + x_\ell$ of two FP numbers s.t. $x_h = \mathrm{RN}(x)$.

Adding 3 numbers is a difficult problem

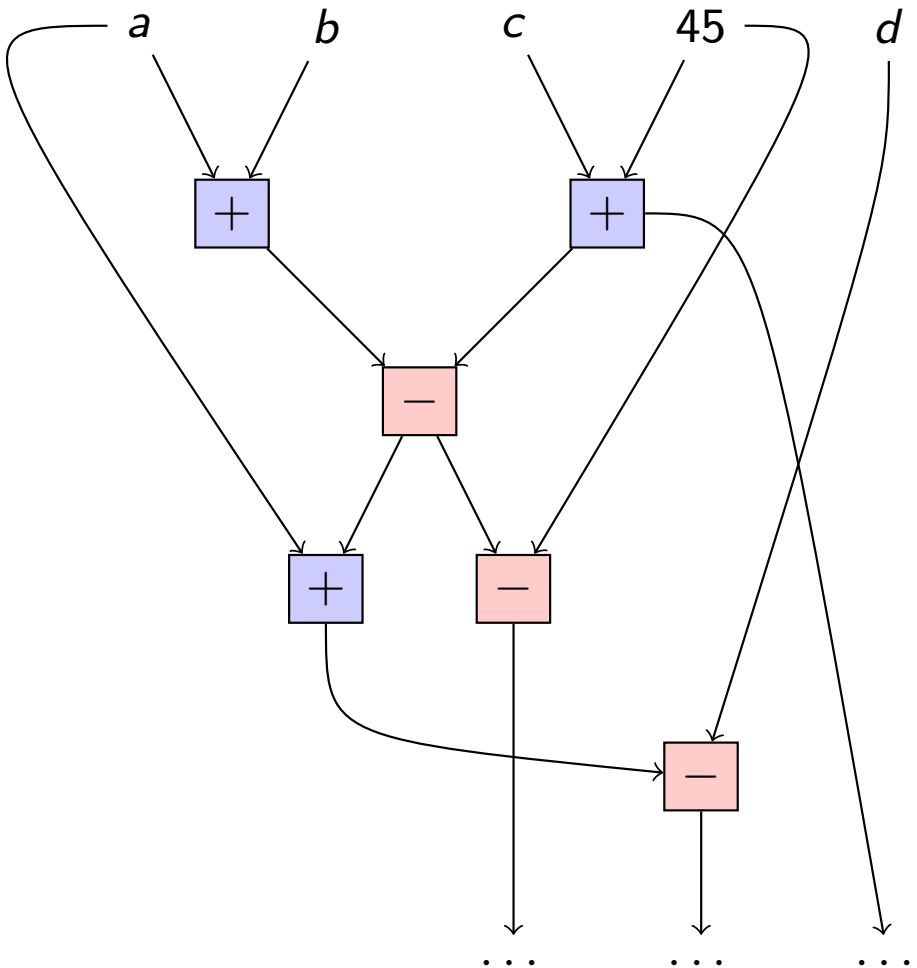
- RN-addition algorithm: only uses operations of the form

$$z \leftarrow \mathrm{RN}(x \pm y)$$

(no comparisons, no tests);

- Kornerup, Lefèvre, Louvet, M. (2013): in binary FP arithmetic with unbounded exponent range, no RN-addition algorithm returns $\mathrm{RN}(a+b+c)$ for all $a, b, c \in \mathbb{F}$.

RN-addition algorithm: DAG whose vertices are FP $+$ or $-$

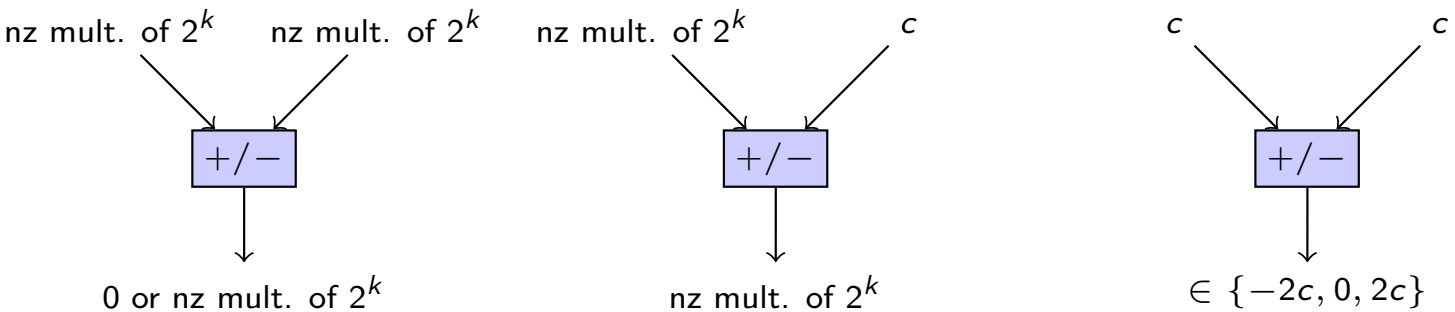


Computing RN($a+b+c$) with a RN-add algorithm?

- DAG of depth r;
- possible constants are nonzero (nz) multiple of 2^k for some $k \in \mathbb{Z}$;
- $a = 2^{k+p}$ and $b = 2^k$ ($\rightarrow a+b$ is a midpoint);
- $|c| \leq 2^{k-p-r}$

$\rightarrow$ RN($a+b+c$) is the FP number immediately below or above $a+b$, depending on the sign of c.

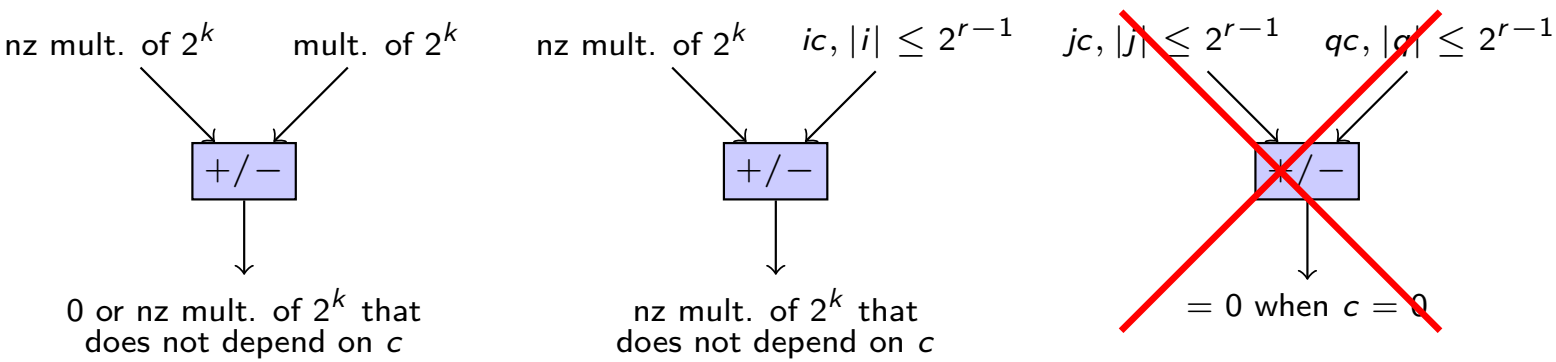
- Operations at depth 1 in the DAG:



- after depth-1 operations, available operands are multiples of 2^k that do not depend on c and elements of $\{-2c, 0, c, 2c\}$.

Computing RN($a+b+c$) with a RN-add algorithm?

- Induction: after depth-m operations, available operands are multiples of 2^k that do not depend on c and elements of $\{-2^m c, \ldots, 0, \ldots, 2^m c\}$.
- Last operation, that outputs the final result:
  - its inputs are not both elements of $\{-2^{r-1}c, \ldots, 0, \ldots, 2^{r-1}c\}$, because when $c = 0$ it must output RN($a+b$);
  - $\rightarrow$ at least one input is a nonzero multiple of 2^k that do not depend on c;
  - $\rightarrow$ the output is a multiple of 2^k that do not depend on c.



- the sign of c cannot change the result.

But in real life, the exponents are bounded?

Same reasoning: Assuming extremal exponents $e_{\min}$ and $e_{\max}$, an RN-addition algorithm of depth r cannot always return $\mathrm{RN}(a+b+c)$ as soon as

$$r \leq e_{\max} - e_{\min} - 2p.$$

Binary64/double precision: an RN-addition algorithm that always returns $\mathrm{RN}(a+b+c)$ (if such an algorithm exists!) has depth ≥ 1939.

$\rightarrow$ Adding 3 numbers is a difficult problem! We cannot avoid tests/comparisons and/or use of various rounding functions.

Classical results 1: Sterbenz theorem

Theorem 1 (Sterbenz Theorem)

Let $a, b \in \mathbb{F}$. *If*

$$\frac{a}{2} \leq b \leq 2a,$$

then $a - b \in \mathbb{F}$.

Implies that the subtraction $a - b$ is performed exactly in FP arithmetic.

Classical results 2: Fast2Sum and 2Sum

$x_h \leftarrow \mathrm{RN}(a+b)$
$z \leftarrow \mathrm{RN}(x_h - a)$
$x_\ell \leftarrow \mathrm{RN}(b - z)$
return (x_h, x_ℓ)

Alg. 1: Fast2Sum(a, b). Returns $(x_h, x_\ell) \in \mathbb{F}^2$ such that x_h is the FP number nearest $a+b$, and, if $|a| \geq |b|$ or $a = 0$, $x_\ell = (a+b) - x_h$.

$x_h \leftarrow \mathrm{RN}(a+b)$
$a' \leftarrow \mathrm{RN}(x_h - b)$
$b' \leftarrow \mathrm{RN}(x_h - a')$
$\delta_a \leftarrow \mathrm{RN}(a - a')$
$\delta_b \leftarrow \mathrm{RN}(b - b')$
$x_\ell \leftarrow \mathrm{RN}(\delta_a + \delta_b)$
return (x_h, x_ℓ)

Alg. 2: 2Sum(a, b). Returns $(x_h, x_\ell) \in \mathbb{F}^2$ such that x_h is the FP number nearest $a+b$, and $x_\ell = (a+b) - x_h$.

Classical results 3: The Dekker-Veltkamp multiplication

Require: $K = 2^s + 1$
Require: $2 \leq s \leq p - 2$
$\gamma \leftarrow \mathrm{RN}(K \cdot x)$
$\delta \leftarrow \mathrm{RN}(x - \gamma)$
$x_h \leftarrow \mathrm{RN}(\gamma + \delta)$
$x_\ell \leftarrow \mathrm{RN}(x - x_h)$
return (x_h, x_ℓ)

Alg. 3: Split(x, s). x_h fits in $p - s$ bits, x_ℓ fits in $s - 1$ bits, $x_h + x_\ell = x$.

Require: $s = \lceil p/2 \rceil$
$(a_h, a_\ell) \leftarrow \mathrm{Split}(a, s)$
$(b_h, b_\ell) \leftarrow \mathrm{Split}(b, s)$
$\pi_h \leftarrow \mathrm{RN}(a \cdot b)$
$t_1 \leftarrow \mathrm{RN}(-\pi_h + \mathrm{RN}(a_h \cdot b_h))$
$t_2 \leftarrow \mathrm{RN}(t_1 + \mathrm{RN}(a_h \cdot b_\ell))$
$t_3 \leftarrow \mathrm{RN}(t_2 + \mathrm{RN}(a_\ell \cdot b_h))$
$\pi_\ell \leftarrow \mathrm{RN}(t_3 + \mathrm{RN}(a_\ell \cdot b_\ell))$
return (π_h, π_ℓ)

Alg. 4: DekkerProd(a, b). $\pi_h = \mathrm{RN}(ab)$ and $\pi_h + \pi_\ell = ab$.

(A bit less) classical result Round-to-Odd

$$\mathrm{RO}(t) = \begin{cases} t \text{ if } t \in \mathbb{F} \\ \text{the FPN with an odd integral significand nearest } t \text{ otherwise.} \end{cases}$$

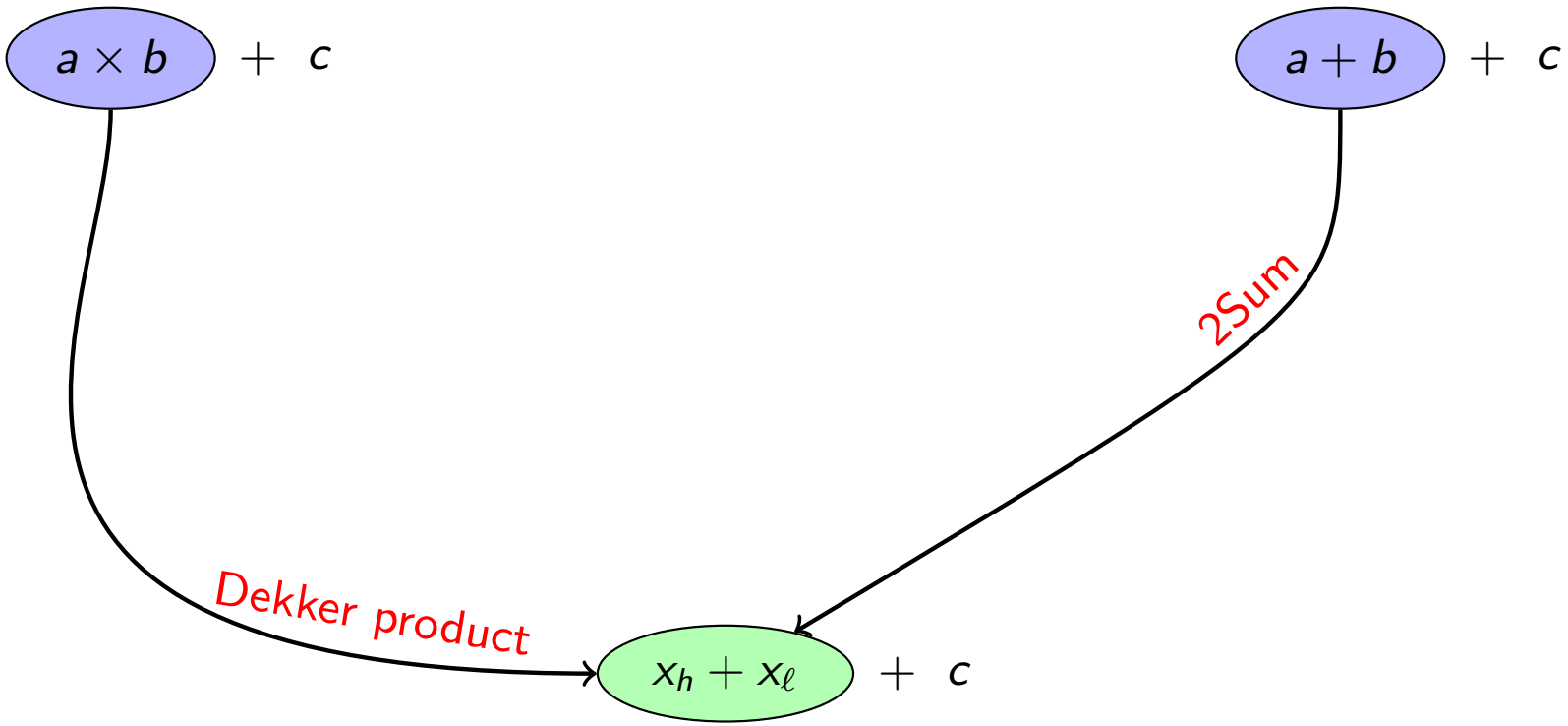
- not specified by IEEE-754
- not implemented in current processors of commercial significance
- known emulation uses internal binary representation (*not doable with "high level" algorithms?*)
- many nice properties, among them:

Lemma 2 (Boldo & Melquiond, 2008)

Let $x \in \mathbb{F}$ and $z \in \mathbb{R}$. If $p \geq 4$ and $6 \cdot |z| \leq x$ then

$$\mathrm{RN}(x + \mathrm{RO}(z)) = \mathrm{RN}(x + z).$$

FMA and ADD3 $\rightarrow$ DW number+ FP number



$\rightarrow$ we focus on: compute $x_h + x_\ell + c$, where $|x_\ell| \leq \frac{1}{2}\text{ulp}(x_h)$.

Easy if we have Round-to-Odd...

$$\mathrm{RN}(x_h + x_\ell + c)$$

- define $(s_h, s_\ell) = 2Sum(x_h, c)$, so that $x_h + x_\ell + c = s_h + s_\ell + x_\ell$;
- Lemma 2 implies

$$\mathrm{RN}(s_h + s_\ell + x_\ell) = \mathrm{RN}(s_h + \mathrm{RO}(s_\ell + x_\ell)).$$

- Boldo and Melquiond give a solution for emulating $\mathrm{RO}(s_\ell + x_\ell)$.

```
struct DW
{double h, l;};

double OddRoundSum (double x, double y){
INTDOUBLE myvh;
struct DW v;
v = TwoSum(x,y);
myvh.real = v.h;
if (v.l != 0.0){
if (!(myvh.integer & 1)){
if ((v.h > 0.0) ^ (v.l < 0.0)){myvh.integer++;}
  else {myvh.integer--;}}
  }
return myvh.real;
}
```

Determining if $x \in \mathbb{F}^* = \pm 2^k$ or $\pm 3 \cdot 2^k$, $k \in \mathbb{Z}$

Theorem 3

In binary, precision-p (with $p \geq 4$), FP arithmetic, assuming no overflow occurs, $x \in \mathbb{F}^*$ is of the form $\pm 2^k$ or $\pm 3 \cdot 2^k$, with $k \in \mathbb{Z}$, if and only if

$$\mathrm{RN}\left[\mathrm{RN}\left(\left(2^{p-2}+1\right)\cdot x\right) - 2^{p-2}x\right] = x. \tag{1}$$

- uses the fact that $\left(2^{p-2}+1\right)\cdot x$ is a FPN (hence, is computed exactly) iff $x = 0$ or $x = \pm 2^k$ or $\pm 3 \cdot 2^k$, and Sterbenz theorem for the subtraction;

$2^k = 100\cdots0$	$\times$	$\underbrace{100\cdots01}_{p-1 \text{ bits}}0$	$=$	$\underbrace{100\cdots01}_{p-1 \text{ bits}}0$	exact
$3\cdot 2^k = 110\cdots0$	$\times$	$\underbrace{100\cdots01}_{p-1 \text{ bits}}0$	$=$	$\underbrace{110\cdots011}_{p \text{ bits}}$	exact
any other nonzero FPN	$\times$	$\underbrace{100\cdots01}_{p-1 \text{ bits}}0$	$=$	$\underbrace{1xx\cdots xxxxxx}_{\geq p+1 \text{ bits}}$	inexact

- related to Split & Alg. NextPowerTwo of Rump, Ogita, Oishi;
- condition $p \geq 4$ is necessary: a counterexample with $p = 3$ is $x = 6$.

Determining if $x \in \mathbb{F}^* = \pm 2^k$ or $\pm 3 \cdot 2^k$, $k \in \mathbb{Z}$

Require: $P = 2^{p-2} + 1$
Require: $Q = 2^{p-2}$
  $L \leftarrow \mathrm{RN}(P \cdot x)$
  $R \leftarrow \mathrm{RN}(Q \cdot x)$
  $\Delta \leftarrow \mathrm{RN}(L - R)$
  return $(\Delta \neq x)$

Alg. 5: IsNot1or3TimesPowerOf2(x). Returns true iff $|x| \neq 2^k$ or $3 \cdot 2^k$.

Computation of RN($x_h + x_\ell + c$)

1: $(s_h, s_\ell) \leftarrow 2\mathrm{Sum}(x_h, c)$
2: $(v_h, v_\ell) \leftarrow 2\mathrm{Sum}(x_\ell, s_\ell)$
3: **if** IsNot1or3TimesPowerOf2(v_h) **or** $v_\ell = 0$ **then**
4: $\quad z \leftarrow \mathrm{RN}(s_h + v_h)$
5: **else**
6: $\quad$ **if** v_ℓ and v_h have the same sign **then**
7: $\quad\quad z \leftarrow \mathrm{RN}\left(s_h + \mathrm{RN}\left(1.125 v_h\right)\right)$
8: $\quad$ **else**
9: $\quad\quad z \leftarrow \mathrm{RN}\left(s_h + \mathrm{RN}\left(0.875 v_h\right)\right)$
10: $\quad$ **end if**
11: **end if**
12: **return** z

Alg. 6: CR-DWPlusFP(x_h, x_ℓ, c). Computes RN($x_h + x_\ell + c$).

Remarks:

- $x_h + x_\ell + c = s_h + v_h + v_\ell$;
- The constants $1.125 = 9/8$ and $0.875 = 7/8$ that appear in Alg. 6 are exactly representable as soon as $p \geq 4$;

Analysis of Algorithm CR-DWPlusFP

Theorem 4

If $p \geq 5$, the number z returned by Algorithm CR-DWPlusFP (Algorithm 6) satisfies

$$z = \mathrm{RN}(x_h + x_\ell + c).$$

We just give a sketch of the proof.

Define $\Sigma = \mathrm{RN}(x_h + x_\ell + c)$. First, 2Sum $\rightarrow$ variables s_h, v_h and v_ℓ in Algorithm 6 satisfy

$$s_h + v_h + v_\ell = x_h + x_\ell + c, \text{ so that } \Sigma = \mathrm{RN}(s_h + v_h + v_\ell),$$
$$|v_\ell| \leq \tfrac{1}{2}\mathrm{ulp}(v_h).$$

$\rightarrow$ discussion on $s_h + v_h + v_\ell$.

Analysis of Algorithm CR-DWPlusFP

1: $(s_h, s_\ell) \leftarrow 2\mathrm{Sum}(x_h, c)$
2: $(v_h, v_\ell) \leftarrow 2\mathrm{Sum}(x_\ell, s_\ell)$
3: **if** IsNot1or3TimesPowerOf2(v_h) **or** $v_\ell = 0$ **then**
4: $z \leftarrow \mathrm{RN}(s_h + v_h)$
5: **else**
6: **if** v_ℓ and v_h have the same sign **then**
7: $z \leftarrow \mathrm{RN}\left(s_h + \mathrm{RN}\left(\frac{9}{8}v_h\right)\right)$
8: **else**
9: $z \leftarrow \mathrm{RN}\left(s_h + \mathrm{RN}\left(\frac{7}{8}v_h\right)\right)$
10: **end if**
11: **end if**
12: **return** z

- If $p \geq 5$, when $v_h = \pm 2^k$ or $\pm 3 \cdot 2^k$, the terms $(7/8)v_h$ and $(9/8)v_h$ are FP numbers;
- case $x_h = 0$ straightforward;
- If x_h, x_ℓ, and c are multiplied by $\pm 2^k$, then s_h, s_ℓ, v_h, v_ℓ, z and Σ are multiplied by $\pm 2^k$;
- If we interchange x_h and c, same result $\rightarrow$ prove the theorem in the case $|x_h| \geq |c|$;

$\Rightarrow$ We focus on $1 \leq x_h \leq 2 - 2u$ and $|c| \leq x_h$.

Analysis of Algorithm CR-DWPlusFP

We focus on $1 \leq x_h \leq 2 - 2u$ and $|c| \leq x_h$.

1: $(s_h, s_\ell) \leftarrow \mathrm{2Sum}(x_h, c)$
2: $(v_h, v_\ell) \leftarrow \mathrm{2Sum}(x_\ell, s_\ell)$
3: **if** IsNot1or3TimesPowerOf2(v_h) **or** $v_\ell = 0$ **then**
4: $z \leftarrow \mathrm{RN}(s_h + v_h)$
5: **else**
6: **if** v_ℓ and v_h have the same sign **then**
7: $z \leftarrow \mathrm{RN}\left(s_h + \mathrm{RN}\left(\frac{9}{8} v_h\right)\right)$
8: **else**
9: $z \leftarrow \mathrm{RN}\left(s_h + \mathrm{RN}\left(\frac{7}{8} v_h\right)\right)$
10: **end if**
11: **end if**
12: **return** z

- If $-x_h \leq c \leq -\frac{x_h}{2}$, Sterbenz $\Rightarrow s_\ell = 0 \Rightarrow$ straightforward;
- we focus on

$$-\frac{x_h}{2} < c \leq x_h,$$

which implies

$$\frac{1}{2} \leq s_h \leq 4 - 4u.$$

Analysis of Algorithm CR-DWPlusFP

1: $(s_h, s_\ell) \leftarrow \text{2Sum}(x_h, c)$
2: $(v_h, v_\ell) \leftarrow \text{2Sum}(x_\ell, s_\ell)$
3: **if** IsNot1or3TimesPowerOf2(v_h) **or** $v_\ell = 0$ **then**
4: $z \leftarrow \text{RN}(s_h + v_h)$
5: **else**
6: **if** v_ℓ and v_h have same sign **then**
7: $z \leftarrow \text{RN}\left(s_h + \text{RN}\left(\frac{9}{8} v_h\right)\right)$
8: **else**
9: $z \leftarrow \text{RN}\left(s_h + \text{RN}\left(\frac{7}{8} v_h\right)\right)$
10: **end if**
11: **end if**
12: **return** z

Reminder: $\Sigma = \text{RN}(s_h + v_h + v_\ell)$, and $\frac{1}{2} \leq s_h \leq 4 - 4u$

- Case A: $\frac{1}{2} \leq s_h \leq 1 - u$, s_h multiple of u, $|s_\ell| \leq \frac{u}{2}$, $|v_h| \leq \frac{3u}{2}$, $|v_\ell| \leq u^2$;
- Case B: $1 \leq s_h \leq 2 - 2u$, s_h mult. of $2u$, $|s_\ell| \leq u$, $|v_h| \leq 2u$, $|v_\ell| \leq u^2$;
- Case C: $2 \leq s_h \leq 4 - 4u$, s_h mult. of $4u$, $|s_\ell| \leq 2u$, $|v_h| \leq 3u$, $|v_\ell| \leq 2u^2$.
- In all cases distance between $s_h + v_h$ and a midpoint multiple of $\text{ulp}(v_h)$. As $|v_\ell| \leq \frac{1}{2}\text{ulp}(v_h)$, If $s_h + v_h$ is not a midpoint, no midpoint between $s_h + v_h$ and $s_h + v_h + v_\ell$ $\rightarrow \Sigma = \text{RN}(s_h + v_h)$.

When can $s_h + v_h$ be a midpoint?

The midpoints are the odd multiples of $\frac{u}{4}$ in $[\frac{1}{4}, \frac{1}{2})$, the odd multiples of $\frac{u}{2}$ in $[\frac{1}{2}, 1)$, and the odd multiples of u in $[1, 2)$. Therefore, if $s_h + v_h$ is a midpoint:

- In Case A: if $s_h = \frac{1}{2}$ then $v_h \in \{-\frac{3u}{4}, -\frac{u}{4}, \frac{u}{2}, \frac{3u}{2}\}$, and if $\frac{1}{2} < s_h \leq 1 - u$ then $v_h \in \{-\frac{3u}{2}, -\frac{u}{2}, \frac{u}{2}, \frac{3u}{2}\}$;
- In Case B: if $s_h = 1$ then $v_h \in \{-\frac{3u}{2}, -\frac{u}{2}, u\}$, and if $1 < s_h \leq 2 - 2u$ then $v_h \in \{-u, u\}$;
- In Case C: if $s_h = 2$ then $v_h \in \{-3u, -u, 2u\}$ and if $2 < s_h \leq 4 - 4u$ then $v_h \in \{-2u, 2u\}$.

In all cases, v_h is of the form $\pm 2^k$ or $\pm 3 \cdot 2^k$.

Analysis of Algorithm CR-DWPlusFP

1: $(s_h, s_\ell) \leftarrow \text{2Sum}(x_h, c)$
2: $(v_h, v_\ell) \leftarrow \text{2Sum}(x_\ell, s_\ell)$
3: **if** IsNot1or3TimesPowerOf2(v_h) **or** $v_\ell = 0$ **then**
4: $\quad z \leftarrow \text{RN}(s_h + v_h)$
5: **else**
6: $\quad$ **if** v_ℓ and v_h have the same sign **then**
7: $\quad\quad z \leftarrow \text{RN}\left(s_h + \text{RN}\left(\frac{9}{8} v_h\right)\right)$
8: $\quad$ **else**
9: $\quad\quad z \leftarrow \text{RN}\left(s_h + \text{RN}\left(\frac{7}{8} v_h\right)\right)$
10: $\quad$ **end if**
11: **end if**
12: **return** z

- when v_h is not of the form $\pm 2^k$ or $\pm 3 \cdot 2^k$, $s_h + v_h$ is not a midpoint, so that

$$\Sigma = \text{RN}(s_h + v_h);$$

- when v_h is of the form $\pm 2^k$ or $\pm 3 \cdot 2^k$, case-by-case analysis.

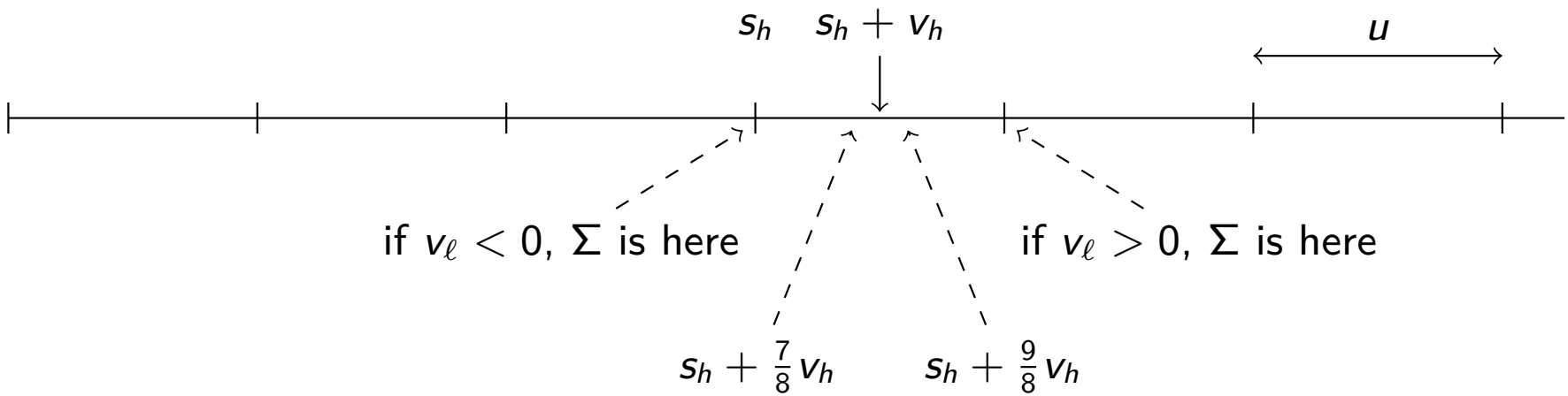


Figure 1: The subcase $\frac{1}{2} < s_h < 1 - u$ and $v_h = +\frac{u}{2}$.

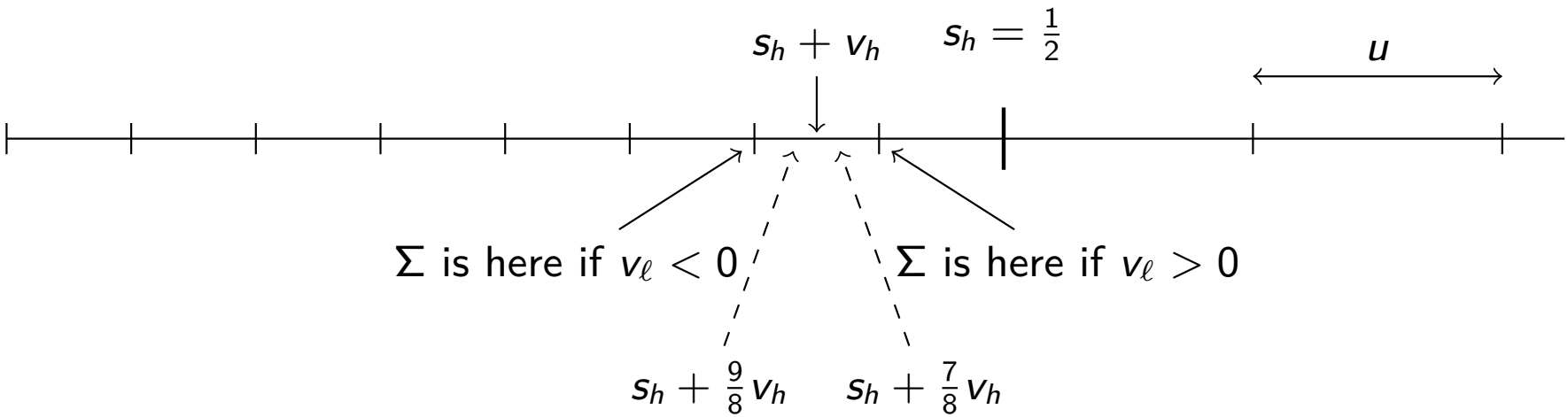


Figure 2: The subcase $s_h = \frac{1}{2}$ and $v_h = -\frac{3u}{4}$.

Emulation of ADD3 and the FMA

1: $(x_h, x_\ell) \leftarrow \mathrm{2Sum}(a, b)$
2: **return** CR-DWPlusFP(x_h, x_ℓ, c)

Alg. 7: Computation of $\mathrm{RN}(a + b + c)$.

1: $(x_h, x_\ell) \leftarrow \mathrm{DekkerProd}(a, b)$
2: **return** CR-DWPlusFP(x_h, x_ℓ, c)

Alg. 8: Computation of $\mathrm{FMA}(a, b, c) = \mathrm{RN}(ab + c)$.

Theorem 5

In a binary, precision-p (with $p \geq 5$), FP arithmetic, Alg. 7 returns $\mathrm{RN}(a + b + c)$ *and Alg. 8 returns* $\mathrm{RN}(ab + c)$ *for all* a, b, $c \in \mathbb{F}$.

Error of these operations?

Same as previously: we reduce FMA and ADD3 to $x_h + x_\ell + c$.

1: $(s_h, s_\ell) \leftarrow \text{2Sum}(x_h, c)$
2: $(v_h, v_\ell) \leftarrow \text{2Sum}(x_\ell, s_\ell)$
3: $(w_h, w_\ell) \leftarrow \text{Fast2Sum}(s_h, v_h)$
4: **if** IsNot1or3TimesPowerOf2(v_h) **or** $v_\ell = 0$ **then**
5: $\quad \delta \leftarrow w_\ell$
6: $\quad z \leftarrow w_h$
7: **else**
8: $\quad$ **if** v_ℓ and v_h have the same sign **then**
9: $\quad\quad z \leftarrow \text{RN}\left(s_h + \text{RN}\left(\frac{9}{8} v_h\right)\right)$
10: $\quad$ **else**
11: $\quad\quad z \leftarrow \text{RN}\left(s_h + \text{RN}\left(\frac{7}{8} v_h\right)\right)$
12: $\quad$ **end if**
13: $\quad \alpha \leftarrow \text{RN}(z - w_h)$
14: $\quad \delta \leftarrow \text{RN}(w_\ell - \alpha)$
15: **end if**
16: **return** (z, δ, v_ℓ)

Alg. 9: CR-DWPlusFP-with-error. Computes $z = \text{RN}(x_h + x_\ell + c)$, and δ and v_ℓ such that $z + \delta + v_\ell = x_h + x_\ell + c$.

Error of these operations?

Theorem 6

If $p \geq 5$ *and* $|x_\ell| \leq \frac{1}{2}\mathsf{ulp}(x_h)$, *the numbers* z, δ, *and* v_ℓ *returned by Algorithm 9 satisfy*

$$z = \mathsf{RN}(x_h + x_\ell + c)$$

and

$$\delta + v_\ell = x_h + x_\ell + c - z.$$

1: $(x_h, x_\ell) \leftarrow \mathsf{DekkerProd}(a, b)$
2: **return** CR-DWPlusFP-with-error(x_h, x_ℓ, c)

Alg. 10: – **FMA-with-error**(a, b, c). Computes $z = \mathsf{RN}(ab + c)$ and δ and v_ℓ such that $ab + c = z + \delta + v_\ell$.

Error of the FMA *when there is a hardware FMA*

- an algorithm was suggested by Boldo and M. in 2005;
- Alg. 10: Dekker Product then Alg. 9 $\rightarrow RN(ab+c)$ and error of that operation;
- however, on platforms with an FMA:
  - to obtain x_h and x_ℓ, no need to use Dekker product, since $x_h = \mathrm{RN}(ab)$ and $x_\ell = ab - x_h$ are obtained with a multiplication and an FMA;
  - the tests needed to compute z in Alg. 9 are no longer necessary: $z = \mathrm{RN}(ab+c)$ is obtained with an FMA;
  - . . . other simplifications that would need to dive into the proof of Theorem 6.

Error of the FMA *when there is a hardware FMA*

1: $z_h \leftarrow \mathrm{RN}(ab + c)$
2: $x_h = \mathrm{RN}(ab)$
3: $x_\ell = \mathrm{RN}(ab - x_h)$
4: $(s_h, s_\ell) \leftarrow \mathrm{2Sum}(x_h, c)$
5: $(v_h, v_\ell) \leftarrow \mathrm{2Sum}(x_\ell, s_\ell)$
6: $\alpha' \leftarrow \mathrm{RN}(z_h - s_h)$
7: $\delta' \leftarrow \mathrm{RN}(v_h - \alpha')$
8: **return** (z, δ', v_ℓ)

Alg. 11: Computes $z = \mathrm{RN}(ab + c)$ and δ' and v_ℓ s.t. $ab + c = z + \delta' + v_\ell$.

Our ADD3 vs Boldo and Melquiond's

Table 1: Time (in seconds) to perform 5×10^9 ADD3 operations in binary64, using the Boldo-Melquiond algorithm and Algorithm 7, on different environments. Each operand: $K \times s \times F$, where F is uniform in $[0, 1]$, $s = \pm 1$ (each with probability $1/2$), and $K \in \{1, 2^{\pm 20}, 2^{\pm 40}, 2^{\pm 60}, 2^{\pm 80}\}$ (each with probability $1/9$).

Architecture/System	compiler and options	Boldo-Melquiond	Algorithm 7
Intel Corei7	clang (v. 16.0.0)	177	153
under MacOS	clang -O3	22	19
Apple M3Pro	clang (v. 16.0.0)	142	144
under MacOS	clang -O3	7	9
AMD Opteron 6272	gcc (v. 12.2.0)	759	659
under Linux	gcc -O3	127	104
	clang -O3	168	93
Intel Xeon Gold 6444Y	gcc (v. 12.2.0)	95	84
under Linux	gcc -O3	18	20
	clang -O3 (v. 14.0.6)	18	15

Our FMA vs Boldo and Melquiond's

Table 2: Time (in seconds) to perform 5×10^9 FMA operations in binary64, using the Boldo-Melquiond (BM) algorithm, Algorithm 8, and the FMA provided by the environment . Each operand: $K \times s \times F$, where F is uniform in $[0, 1]$, $s = \pm 1$ (each with probability $1/2$), and $K \in \{1, 2^{\pm 20}, 2^{\pm 40}, 2^{\pm 60}, 2^{\pm 80}\}$ (each with probability $1/9$).

Architecture/System	compiler and options	BM	Alg. 8	native
Intel Corei7	clang (v. 16.0.0)	258	200	41
under MacOS	clang -O3	31	25	10
Apple M3Pro	clang (v. 16.0.0)	228	162	7
under MacOS	clang -O3	10	9	4
AMD	gcc (v. 12.2.0)	1068	856	75
Opteron 6272	gcc -O3 -lm	190	110	42
under Linux	clang -O3 -lm	181	95	43
Intel Xeon	gcc -lm (v. 12.2.0)	109	98	10
Gold 6444Y	gcc -O3 -lm	25	24	10
under Linux	clang -O3 -lm (v. 14.0.6)	25	21	10

Our error of the FMA vs Boldo and M.'s

Table 3: Time (in seconds) to compute 5×10^9 errors of FMA operations in binary64, using the Boldo-Muller algorithm and Algs 10 and 11, on different environments. Each operand: $K \times s \times F$, where F is uniform in $[0, 1]$, $s = \pm 1$ (each with probability $1/2$), and $K \in \{1, 2^{\pm 20}, 2^{\pm 40}, 2^{\pm 60}, 2^{\pm 80}\}$ (each with probability $1/9$).

Architecture/System	compiler and options	Boldo-M.	Alg. 10	Alg. 11
Intel Corei7	clang (v. 16.0.0)	166	280	168
under MacOS	clang -O3	30	39	33
Apple M3Pro	clang (v. 16.0.0)	151	298	143
under MacOS	clang -O3	7	13	7
AMD Opteron 6272	gcc (v. 12.2.0)	742	1252	736
under Linux	gcc -O3	134	143	140
	clang -O3	117	122	130
Intel Xeon Gold 6444Y	gcc (v. 12.2.0)	89	144	87
under Linux	gcc -O3	23	30	24
	clang -O3 (v. 14.0.6)	22	28	22

Discussion

- primary disadvantage of Algs 6, 7, 8, and 10: presence of tests;
- however:
  - we have seen that ADD3 with only rounded to nearest $+/-$ is impossible;
  - the test (*is some variable of the form* $\pm 2^k$ *or* $\pm 3 \cdot 2^k$*?*) almost always return **false** $\rightarrow$ in practice branch prediction works very well.
- ADD3 and FMA:
  - performance slightly better or similar (ADD3), or always better (FMA) than Boldo and Melquiond's algs (because they test on parity, which is much harder to predict?)
  - high-level algorithms.
- Error of the FMA:
  - if no hardware FMA, no real choice;
  - if hardware FMA: check on your environment, with your applications;
- better error bounds for several double-word or triple-word algorithms.