Useful applications of correctly-rounded operators of the form $ab + cd + e$

Tom Hubrecht, Claude-Pierre Jeannerod, Jean-Michel Muller

November 05, 2024

LIP - Équipe AriC

ARITH 2024 Best Paper Award

Introduction

- Introduction of the **FMA** 30 years ago

$$s = \mathsf{RN}\left(ab + c\right)$$

  - Efficient correctly rounded square root, division and double-word algorithms
  - Faster computations (dot product, polynomial evaluation)

- Introduction of the **FD2A** *(FDP, FDPA)* recently, what is possible now ?

$$s = \mathsf{RN}\left(ab + cd + e\right)$$

Assumptions and notation

- Floating-point arithmetic of radix 2 and precision p, unbounded exponent range, conforms to **IEEE-754** assuming no underflows or overflows ;
- $u := 2^{-p}$ is the unit roundoff ;
- $\mathbb{F}$ is the set of FP numbers ;
- RN the rounding function, rounds to nearest, ties to even ;
- $\text{FD2}(a, b, c, d) := \text{RN}(ab + cd)$,
  $\text{FD2A}(a, b, c, d, e) := \text{RN}(ab + cd + e)$,
  with $a, b, c, d, e \in \mathbb{F}$;

Outline

Complex arithmetic

Error Free Transforms

Products of three or four numbers

Discriminants

Dot products

More efficient double word arithmetic

Complex arithmetic

Correctly rounded multiplication

Let $x = a + i\,b$ and $y = c + i\,d$ with $a, b, c, d \in \mathbb{F}$,

- For addition and subtraction, CR results are automatic:

$$(x \pm y) = (a \pm c) + i\,(b \pm d)$$

- Multiplication needs more to avoid cancellation:

$$x \times y = (ac - bd) + i\,(ad + bc)$$

Two problems can arise when using only **FP** multiplications and **FMA** : catastrophic cancellation and failure to compute a deserved zero.

Division and Square Root

$$\frac{a+ib}{c+id}=\frac{ac+bd}{c^2+d^2}+i\frac{bc-ad}{c^2+d^2}$$

Complex division can be computed with two **FP** divisions and three **FD2** operations with a relative error bound of $3u$.

$$\sqrt{a^2+b^2} \text{ can be computed as } \mathsf{RN}\left(\sqrt{\mathsf{RN}\left(a^2+b^2\right)}\right)$$

This results in an error bound less than $\frac{3u}{2}$, compared with $2u$ for the naive algorithms.

Error Free Transforms

Compute the rounded result and the error to the correct result.

- The error of $\times$ can be computed exactly with an **FMA**
- The error of $+$ can be computed with an **Add3** (or AugmentedAddition)
- The CR value of an **FMA** error $\mathsf{RN}\left(ab + c - \mathsf{RN}\left(ab + c\right)\right)$ can be computed with an **FD2A**

They can all be seen as special cases of an FD2A operator, and it also allows for simpler implementations of TwoSum / FastTwoSum which are EFTs of the addition.

Original lemma

Given $a, b \in \mathbb{F}$ such that $\frac{1}{2}a \leq -b \leq 2a$, then $\textsf{RN}(a+b) = a+b$.

For $a, b, c, d \in \mathbb{F}$ and $\frac{1}{2}ab \leq -cd \leq 2ab$, for:

- $P_h = \textsf{RN}(ab+cd)$
- $P_\ell = \textsf{RN}(ab+cd-P_h)$

In this case, the error of an **FD2** operation is a floating-point number, and we have exactly:

$$ab + cd = P_h + P_\ell$$

Hence $ab+cd$ fits in two floats instead of four.

Products of three or four numbers

$abc + d$ and $abcd$

Given $a, b, c, d \in \mathbb{F}$, we can compute:

- $p_h = \mathrm{RN}\left(ab\right)$
- $p_\ell = \mathrm{RN}\left(ab - p_h\right) = ab - p_h$ // *Exact operation*
- $q = \mathrm{RN}\left(p_h c + p_\ell c + d\right) = \mathrm{RN}\left(abc + d\right)$

It is useful for computing elementary functions (e.g. $1 + x^2 P(x)$ for the cosine).

Using the same method, we can do :

- $q_h = \mathrm{RN}\left(abc\right)$
- $q_\ell = \mathrm{RN}\left(abc - q_h\right)$
- $\hat{r} = \mathrm{RN}\left(q_h d + q_\ell d\right) \simeq abcd$ with a relative error $\leq u$

Discriminants

Considering $ax^2 + bx + c = 0$, we have: $\Delta = b^2 - 4ac$
which can be obtained (*with correct rounding*) in one **FD2** instruction.

Furthermore,

- $\Delta_{+\times} = \mathsf{RN}\left(\mathsf{RN}\left(b^2\right) - \mathsf{RN}\left(4ac\right)\right)$
- $\Delta_{FMA_1} = \mathsf{RN}\left(\mathsf{RN}\left(b^2\right) - 4ac\right)$
- $\Delta_{FMA_2} = \mathsf{RN}\left(b^2 - \mathsf{RN}\left(4ac\right)\right)$

Those computation don't even guarantee the sign of the discriminant, e.g.

- $(a, b, c) = (\frac{1}{4} - \frac{u}{2}, 1, 1 + 2u) \implies \Delta_{+\times} = 0, \quad \Delta < 0$
- $(a, b, c) = (\frac{1}{4} - \frac{u}{4}, 1 - u, 1 - u) \implies \Delta_{FMA_1} < 0, \quad \Delta = 0$

Depressed cubic equations

$$x^3 + bx + c = 0 \text{ gives } \Delta = -4b^3 - 27c^2$$

Again, an implementation with or without an **FMA** can give a result with the wrong sign.

$$\begin{aligned} \delta_1 &= \mathsf{RN}\left(4b^3\right) \\ \delta_2 &= \mathsf{RN}\left(27c^2\right) \\ \Delta_{\mathsf{FD2}} &= \mathsf{RN}\left(-\delta_1 - \delta_2\right) \end{aligned}$$

This computation guarantees that when the result is non-zero, it will have the same sign as Δ because **RN** is increasing.

Dot products

Inner products and sums

Given $x, y \in \mathbb{F}^n$ and $r = x^T y$, we can compute an approximation of r with n **FMA** operations such that $|r_{\text{FMA}} - x^T y| \leq nu|x|^T|y|$

With an **FD2A** we can divide the bounds by 2:

$$|r_{\text{FD2A}} - x^T y| \leq mu|x|^T|y| \text{ in } m \text{ operations, } m = \left\lceil \frac{n}{2} \right\rceil$$

Compensated algorithms

One can also use the following recursions :

- $r_i := \mathsf{RN}\left(r_{i-1} + x_i y_i\right)$
- $e_i := \mathsf{RN}\left(r_{i-1} + x_i y_i - r_i\right)$

At the end, $r_{\text{COMP}} = \mathsf{RN}\left(r_{\text{FMA}} + e\right)$ such that

$$|r_{\text{COMP}} - x^T y| \leq u|x^T y| + \lambda_{\text{COMP}}|x|^T|y|$$

with $\lambda_{\text{COMP}} = (\frac{1}{2}n^2 + n)u^2 + (\frac{1}{4}n^3 + \frac{1}{4}n^2)u^3$.

This value is approximately half of the one in the classic approach, using $n+1$ FMA, $n-1$ FD2A and $\lfloor \frac{n}{2} \rfloor$ ADD3, $10\times$ less compared with the classical $\sim 25n$ flops.

More efficient double word arithmetic

The availability of an **FD2A** allows simpler algorithms to be written, with better error bounds.

Algorithm 1 : TwoSum_Add3(a, b).

1: $s \leftarrow \mathrm{RN}\,(a + b)$
2: $e \leftarrow \mathrm{RN}\,(a + b - s)$ // $a + b = e + s$ (EFT)

Algorithm 2 : DWTimesFP_FD2A(a_h, a_ℓ, b). Computes a DW approximation z to ab.

1: $s \leftarrow \mathrm{RN}\,(a_h \cdot b + a_\ell \cdot b)$
2: $e \leftarrow \mathrm{RN}\,(a_h \cdot b + a_\ell \cdot b - s)$
3: $(z_h, z_\ell) \leftarrow$ TwoSum_Add3(s, e)

The most accurate current algorithm takes **10** operations and has an error bound of $\frac{3}{2}u^2 + 4u^3$, compared with $\frac{u^2}{2}$ in **4** operations here.

Algorithm 3 : DWPlusFP_FD2A(a_h, a_ℓ, b). Computes a DW approximation c to $a + b$.

1: $(s, f) \leftarrow$ TwoSum_Add3(a_h, b)
2: $e \leftarrow$ RN $(f + a_\ell)$
3: $(c_h, c_\ell) \leftarrow$ TwoSum_Add3(s, e)

This algorithm has the same asymptotic error bound of $2u$ as the conventional algorithm, but without mixing **TwoSum** and **FastTwoSum**, simplifying the error analysis.

Smaller error bounds

Without designing new algorithms, error bounds are automatically improved thanks to more precise intermediate steps.

Algorithm 4 : DWDivDW_FD2A$(a_h, a_\ell, b_h, b_\ell)$. Computes a DW approximation z to a/b.

1: $t_h \leftarrow \mathrm{RN}\left(1/b_h\right)$
2: $r_h \leftarrow \mathrm{RN}\left(1 - t_h \cdot b_h\right)$
3: $r_\ell \leftarrow -\mathrm{RN}\left(t_h \cdot b_\ell\right)$
4: $(e_h, e_\ell) \leftarrow \mathrm{TwoSum\_Add3}(r_h, r_\ell)$
5: $(\delta_h, \delta_\ell) \leftarrow \mathrm{DWTimesFP\_FD2A}(e_h, e_\ell, t_h)$
6: $(m_h, m_\ell) \leftarrow \mathrm{DWPlusFP\_FD2A}(\delta_h, \delta_\ell, t_h)$
7: $(z_h, z_\ell) \leftarrow \mathrm{DWTimesDW\_FD2A}(a_h, a_\ell, m_h, m_\ell)$

The error bound is $7.8u^2$ instead of $9.8u^2$ without changing anything.

Many things are possible!

- Easier error analyses
- Simpler algorithms (design and analysis)
- Better accuracy and more correctly rounded results
- More examples in our paper, e.g. Horner's evaluation, Rounded to nearest multiplication by real constants such as e or π
- ...

Some references

[1] R. K. Montoye, E. Hokonek, and S. L. Runyan. "Design of the IBM RISC System/6000 floating-point execution unit". In: *IBM J. Res. Dev.* (1990).

[2] P. Markstein. "Computation of Elementary Functions on the IBM RISC System/6000 Processor". In: *IBM J. Res. Dev.* (1990).

[3] Benoît Dupont de Dinechin, Julien Hascoet, and Orégane Desrentes. "In-Place Multicore SIMD Fast Fourier Transforms". In: *HPEC Proc.* 2023.

[4] Orégane Desrentes, Benoît Dupont de Dinechin, and Florent de Dinechin. "Exact fused dot product add operators". In: *ARITH Proc.* 2023.

[5] Marat Dukhan, Richard Vuduc, and Jason Riedy. *Wanted: Floating-Point Add Round-off Error instruction*. Preprint (https://arxiv.org/abs/1603.00491) and PMMA16 slides (https://blogs.fau.de/hager/bofs/isc16-workshop). 2016.

[6] Jean-Michel Muller and Laurence Rideau. "Formalization of double-word arithmetic, and comments on "Tight and rigorous error bounds for basic building blocks of double-word arithmetic"". In: *ACM Trans. Math. Soft.* (2022). DOI: 10.1145/3484514.

[7] N. Louvet. "Algorithmes Compensés en Arithmétique Flottante : Précision, Validation, Performances". PhD thesis. U. Perpignan, 2007.

[8] Jean-Michel Muller et al. *Handbook of Floating-Point Arithmetic*. English. Birkhäuser, 2018, p. 627.

[9] Takeshi Ogita, Siegfried M. Rump, and Shin'ichi Oishi. "Accurate Sum and Dot Product". In: *SIAM J. Sci. Comput.* (2005). DOI: 10.1137/030601818.

[10] N. Brisebarre and J.-M. Muller. "Correctly rounded multiplication by arbitrary precision constants". In: *IEEE Trans. Comp.* (2008). DOI: 10.1109/TC.2007.70813.

[11] IEEE. *IEEE Standard for Floating-Point Arithmetic (IEEE Std 754-2019)*. 2019. DOI: 10.1109/IEEESTD.2019.8766229.