Accurate Complex Multiplication in Floating-Point Arithmetic

Vincent LEFÈVRE

Joint work with Jean-Michel MULLER (ARITH 26)

AriC, INRIA Grenoble – Rhône-Alpes / LIP, ENS-Lyon

RAIM 2019, Toulon, 2019-11-27

Outline

- Generalities on floating point
- Naive algorithms for complex FP multiplication
- Accurate complex multiplication
- Implementation & experiments
- Other variants (work in progress)
- Conclusion

The floating-point system

The floating-point system:

- Radix $\beta = 2$ (?), precision p.
- Rounding to nearest (RN), with any choice in case of a tie.
- Basic operations: $+$, $-$, $\times$, and FMA/FMS.
- Usual assumption: underflow/overflow do not occur.

Note: tests done with the even-rounding rule (ties to even).

Bound on the relative error when rounding a real number x (the exact result of an operation):

$$|\operatorname{RN}(x) - x| \leqslant \frac{u}{1+u} \cdot |x| \leqslant u \cdot |x|$$

where $u = \frac{1}{2}\beta^{1-p} = 2^{-p}$ (*rounding unit*).

The last inequality is strict when $x \neq 0$.

We will use in the proofs: $|\operatorname{RN}(x) - x| \leqslant u \cdot |x|$ and $|\operatorname{RN}(x)| \leqslant (1+u) \cdot |x|$.

Double-word numbers

To get more accurate results: ***double-word numbers*** (DW).

- A representation of a number x by a pair (x_h, x_ℓ) of FP numbers such that

$$\begin{cases} x = x_h + x_\ell \\ |x_\ell| \leqslant \frac{1}{2}\,\mathrm{ulp}(x) \leqslant u \cdot |x|. \end{cases}$$

  Note: if x is a real number, $x_h = \mathrm{RN}(x)$, and the error term $x_\ell = x - x_h$ is exactly representable, then (x_h, x_ℓ) is a DW number.

- Also called double-double in the literature (with basic FP format = double).

- Algorithms and libraries for manipulating DW numbers:
  QD (Hida, Li & Bailey), Campary (Joldes, Popescu & others).
  Also part of libgcc on PowerPC (C type "long double").

- Use the 2Sum, Fast2Sum & Fast2Mult algorithms (see next slides).

Advanced operations: Error-Free Transforms

In rounding to nearest (RN), the error term of an addition or multiplication of two FP numbers is exactly representable.

Operations that return the rounded result and the error term:

- Addition: one gets $s = \mathrm{RN}(a + b)$ and t such that $a + b = s + t$.
- Multiplication: one gets $\pi = \mathrm{RN}(ab)$ and ρ such that $ab = \pi + \rho$.

$\rightarrow$ Error-Free Transforms.

New in the 2019 revision of the IEEE 754 standard (IEEE 754-2019): recommended *augmented arithmetic operations*.

- They have the above properties (with ties rounded toward zero).
- They are intended to be implemented in hardware.

But no hardware implementations yet.

$\rightarrow$ Use of the classical 2Sum and Fast2Mult algorithms.

2Sum and Fast2Mult algorithms

Expressing $a + b$ as a DW number

2Sum(a, b). Returns s and t such that $s = \mathrm{RN}(a + b)$ and $a + b = s + t$.

$$
\begin{aligned}
s &\leftarrow \mathrm{RN}(a + b)\\
a' &\leftarrow \mathrm{RN}(s - b)\\
b' &\leftarrow \mathrm{RN}(s - a')\\
\delta_a &\leftarrow \mathrm{RN}(a - a')\\
\delta_b &\leftarrow \mathrm{RN}(b - b')\\
t &\leftarrow \mathrm{RN}(\delta_a + \delta_b)
\end{aligned}
$$

Expressing ab as a DW number

Fast2Mult(a, b). Returns π and ρ such that $\pi = \mathrm{RN}(ab)$ and $ab = \pi + \rho$.

$$
\begin{aligned}
\pi &\leftarrow \mathrm{RN}(ab)\\
\rho &\leftarrow \mathrm{RN}(ab - \pi)
\end{aligned}
$$

Naive algorithms for complex FP multiplication

- Straightforward transcription of the formula

$$z = z^R + iz^I = (a + ib) \cdot (c + id) = (ac - bd) + i \cdot (ad + bc)$$

$\rightarrow$ Approximate result $\hat{z}$.

- Bad solution if the componentwise relative error is to be minimized.
- Adequate solution if the normwise relative error $|(\hat{z} - z)/z|$ is at stake.

Algorithms:

- If no FMA instruction is available:

$$\begin{cases} \hat{z}^R = \mathrm{RN}(\mathrm{RN}(ac) - \mathrm{RN}(bd)) \\ \hat{z}^I = \mathrm{RN}(\mathrm{RN}(ad) + \mathrm{RN}(bc)) \end{cases} \quad (1)$$

- If an FMA instruction is available:

$$\begin{cases} \hat{z}^R = \mathrm{RN}(ac - \mathrm{RN}(bd)) \\ \hat{z}^I = \mathrm{RN}(ad + \mathrm{RN}(bc)) \end{cases} \quad (2)$$

Naive algorithms for complex FP multiplication [2]

Algorithms:

- If no FMA instruction is available:

$$\begin{cases} \hat{z}^R = \mathrm{RN}(\mathrm{RN}(ac) - \mathrm{RN}(bd)) \\ \hat{z}^I = \mathrm{RN}(\mathrm{RN}(ad) + \mathrm{RN}(bc)) \end{cases} \quad (1)$$

- If an FMA instruction is available:

$$\begin{cases} \hat{z}^R = \mathrm{RN}(ac - \mathrm{RN}(bd)) \\ \hat{z}^I = \mathrm{RN}(ad + \mathrm{RN}(bc)) \end{cases} \quad (2)$$

Asymptotically optimal bounds on the normwise relative error of (1) and (2) are known:

- for (1): bound $\sqrt{5} \cdot u$ (Brent et al., 2007);
- for (2): bound $2 \cdot u$ (Jeannerod et al., 2017).

Accurate complex multiplication

Our goal:

- smaller normwise relative errors,
- closer to the best possible one ($\approx u$, unless we output DW numbers),
- but at the cost of more complex algorithms.

We consider the product $(a + ib) \cdot (c + id)$, where

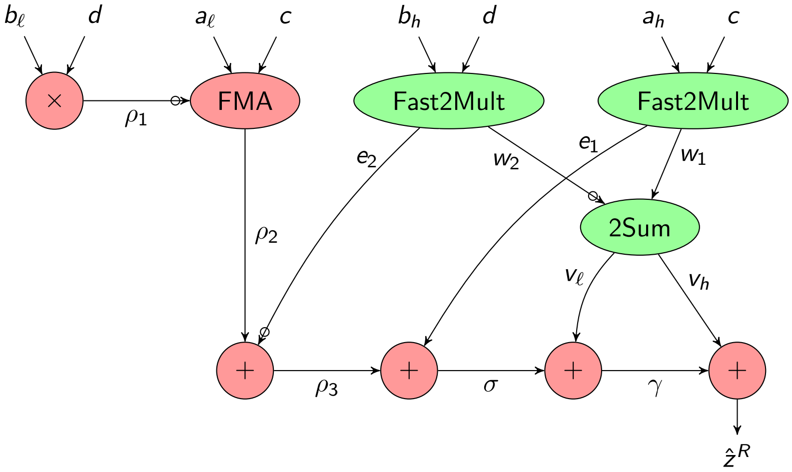
- a and b are DW numbers (special FP case considered later), i.e. $a = a_h + a_\ell$ with $|a_\ell| \leqslant \frac{1}{2}\,\mathrm{ulp}(a)$ and $b = b_h + b_\ell$ with $|b_\ell| \leqslant \frac{1}{2}\,\mathrm{ulp}(b)$,
- c and d are FP numbers.

Real part z^R of the result (similar for the imaginary part):

- difference v_h of the high-order parts of ac and bd;
- add approximated sum γ of all the error terms that may have a significant influence on the normwise relative error.

[raim2019.tex 123312 2019-11-27 01:07:07Z vinc17/zira]

Computation of the real part (JMM's version)



The multiplication algorithm (JMM's version)

Computation of the real part of $(a + ib) \cdot (c + id)$. [(DW,FP) $\rightarrow$ FP]

Computes $z^R = ac - bd$, where $a = a_h + a_\ell$ and $b = b_h + b_\ell$ are DW numbers, and c and d are FP numbers.

$$
\begin{aligned}
(w_1, e_1) &\leftarrow \mathrm{Fast2Mult}(a_h, c) \\
(w_2, e_2) &\leftarrow \mathrm{Fast2Mult}(b_h, d) \\
(v_h, v_\ell) &\leftarrow \mathrm{2Sum}(w_1, -w_2) \\
\rho_1 &\leftarrow \mathrm{RN}(b_\ell d) \\
\rho_2 &\leftarrow \mathrm{RN}(a_\ell c - \rho_1) \\
\rho_3 &\leftarrow \mathrm{RN}(\rho_2 - e_2) \\
\sigma &\leftarrow \mathrm{RN}(\rho_3 + e_1) \\
\gamma &\leftarrow \mathrm{RN}(v_\ell + \sigma) \\
\hat{z}^R &\leftarrow \mathrm{RN}(v_h + \gamma)
\end{aligned}
$$

Error analysis

Only rounding errors (no ignored terms). Let us recall:

- Bounds on rounding errors: $|\epsilon_x| = |\operatorname{RN}(x) - x| \leqslant u \cdot |x|$.
- Bounds on the variables: $|\operatorname{RN}(x)| \leqslant |x| + |\epsilon_x| \leqslant (1+u) \cdot |x|$.

For the real part, we introduce: $N = |ac| + |bd|$ and $n = |ac - bd|$.

As N can be much larger than the absolute value of the exact result n, bounds should be related to n if possible ($\rightarrow$ n can appear only with the main term), otherwise related to N.

For the main term:

- $|v_h + v_\ell| = |w_1 - w_2| \leqslant n + f(u) \cdot N$, with $f(u) = \mathcal{O}(u)$;
- $|v_\ell| \leqslant u \cdot n + u \cdot f(u) \cdot N$;
- $|v_h| \leqslant (1+u) \cdot n + (1+u) \cdot f(u) \cdot N$.

Bounds for the error terms (before v_ℓ is considered):
product of some function of u (in $\mathcal{O}(u^2)$) by N.

Error analysis [2]

- We show that

$$\begin{aligned} |\hat{z}^R - \Re(z)| &\leqslant \alpha n^R + \beta N^R, \\ |\hat{z}^I - \Im(z)| &\leqslant \alpha n^I + \beta N^I, \end{aligned}$$

  with

$$\begin{aligned} N^R &= |ac| + |bd|, \\ n^R &= |ac - bd|, \\ N^I &= |ad| + |bc|, \\ n^I &= |ad + bc|, \\ \alpha &= u + 3u^2 + u^3, \\ \beta &= 15u^2 + 38u^3 + 39u^4 + 22u^5 + 7u^6 + u^7. \end{aligned}$$

- Then we deduce

$$\eta^2 = \frac{\left(\hat{z}^R - \Re(z)\right)^2 + \left(\hat{z}^I - \Im(z)\right)^2}{(\Re(z))^2 + (\Im(z))^2} \leqslant \alpha^2 + (2\alpha\beta + \beta^2) \cdot \frac{\left(N^R\right)^2 + \left(N^I\right)^2}{\left(n^R\right)^2 + \left(n^I\right)^2}.$$

- Then we use

$$\frac{\left(N^R\right)^2 + \left(N^I\right)^2}{\left(n^R\right)^2 + \left(n^I\right)^2} \leqslant 2.$$

Error bound

Theorem 1

As soon as $p \geqslant 4$, the normwise relative error $\eta = |(\hat{z} - z)/z|$ of the algorithm from the previous slide satisfies

$$\eta < u + 33\,u^2.$$

Remember: the best possible bound is $u/(1+u) \approx u$.

Remarks:

- The condition "$p \geqslant 4$" always holds in practice.
- This algorithm can easily be transformed into an algorithm that returns the real and imaginary parts of z as DW numbers (see later).
- In the error terms, we bounded and simplified $u/(1+u)$ by u. This could yield a small overestimation of the order-2 term in Theorem 1.

Real and imaginary parts of z as DW numbers

To obtain the real and imaginary parts of z as DW numbers:

- Replace the last FP addition $\hat{z}^R \leftarrow \mathrm{RN}(v_h + \gamma)$ by a call to $\mathrm{2Sum}(v_h, \gamma)$.
- Similar change for the imaginary part.
- Resulting relative error

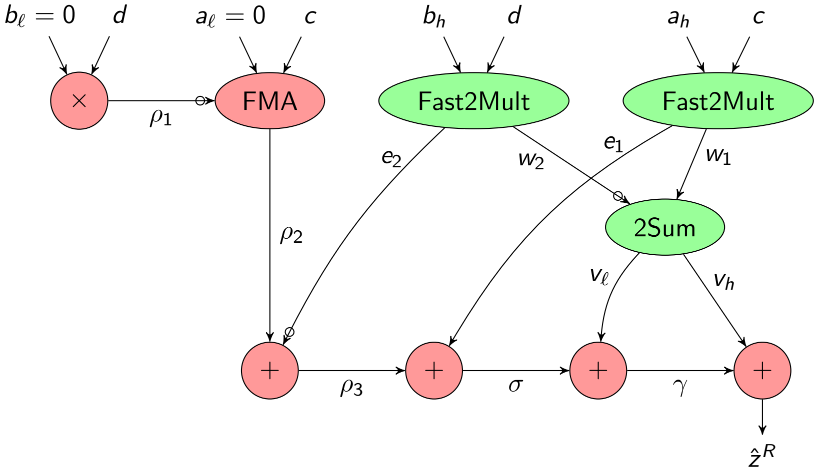
$$\sqrt{241} \cdot u^2 + \mathcal{O}(u^3) \approx 15.53\, u^2 + \mathcal{O}(u^3)$$

(instead of $u + 33\, u^2$).

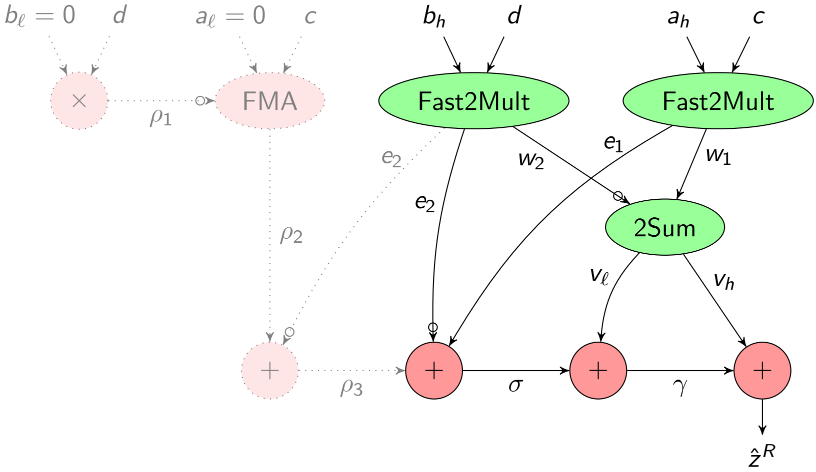
Interest:

- **iterative product $z_1 \times z_2 \times \cdots \times z_n$:** keep the real and imaginary parts of the partial products as DW numbers;
- **Fourier transforms:** when computing $z_1 \pm \omega z_2$, keep $\Re(\omega z_2)$ and $\Im(\omega z_2)$ as DW numbers before the $\pm$.

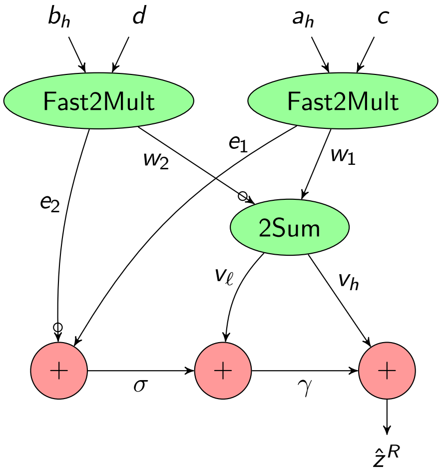
If a and b are FP numbers: original algorithm



If a and b are FP numbers: simplification ($a_\ell = b_\ell = 0$)



If a and b are FP numbers: new algorithm



If a and b are FP numbers: new algorithm [2]

Computation of the real part of $(a + ib) \cdot (c + id)$. [(FP,FP) → FP]

Computes $z^R = ac - bd$, where a, b, c, d are FP numbers.

$$
\begin{aligned}
(w_1, e_1) &\leftarrow \mathrm{Fast2Mult}(a_h, c)\\
(w_2, e_2) &\leftarrow \mathrm{Fast2Mult}(b_h, d)\\
(v_h, v_\ell) &\leftarrow \mathrm{2Sum}(w_1, -w_2)\\
\sigma &\leftarrow \mathrm{RN}(e_1 - e_2)\\
\gamma &\leftarrow \mathrm{RN}(v_\ell + \sigma)\\
\hat{z}^R &\leftarrow \mathrm{RN}(v_h + \gamma)
\end{aligned}
$$

Similar to:

- Cornea, Harrison and Tang's algorithm for $ab + cd$, with a "+" replaced by a 2Sum;
- Algorithm 5.3 in Ogita, Rump and Oishi's *Accurate sum & dot product* (with a different order of summation of e_1, e_2 and v_ℓ).

If a and b are FP numbers: new algorithm [3]

The error bound $u + 33\,u^2$ of Theorem 1 still applies, but it can be slightly improved:

Theorem 2

As soon as $p \geqslant 4$, the normwise relative error $\eta = |(\hat{z} - z)/z|$ of the algorithm from the previous slide satisfies

$$\eta < u + 19\,u^2.$$

Implementation & experiments

- Main algorithm [(DW,FP) $\to$ FP] implemented in binary64 (a.k.a. double-precision) arithmetic, and compared with other solutions:
  - naive formula in binary64 arithmetic;
  - naive formula in binary128 arithmetic;
  - GNU MPFR with precision ranging from 53 to 106 bits.
- Loop over N random inputs, itself inside another loop doing K iterations.
- Goal of the external loop: get accurate timings without having to choose a large N, with input data that would not fit in the cache.
- For each test, we chose $(N, K) = (1024, 65536)$, $(2048, 32768)$, and $(4096, 16384)$.

Implementation & experiments [2]

- Tests run on two computers with a hardware FMA:
  - x86_64 with Intel Xeon E5-2609 v3 CPUs, under Linux (Debian/unstable), with GCC 8.2.0 and a Clang 8 preversion, using `-march=native`;
  - ppc64le with POWER9 CPUs (from the GCC Compile Farm[1]), under Linux (CentOS 7), with GCC 8.2.1, using `-mcpu=power9`.
- Options `-O3` and `-O2`.
- With GCC, `-O3 -fno-tree-slp-vectorize` also used to avoid a loss of performance with some vectorized codes.[2]
- In all cases, `-static` used to avoid the overhead due to function calls to dynamic libraries.

[1] https://cfarm.tetaneutral.net/
[2] https://gcc.gnu.org/bugzilla/show_bug.cgi?id=65847

Implementation & experiments: Timings (x86_64, GCC)

Table 1: Timings on x86_64 (in seconds, for $NK = 2^{26}$ operations) with GCC. GNU MPFR is used with separate $\pm$ and $\times$.

		minimums			maximums		
	$N \rightarrow$	1024	2048	4096	1024	2048	4096
gcc -O3 -f...	(DW,FP) $\rightarrow$ FP	0.92	0.97	0.97	0.95	1.02	1.02
	Naive, binary64	0.61	0.61	0.62	0.61	0.62	0.62
	Naive, binary128	21.32	21.44	21.46	21.43	21.53	21.54
	GNU MPFR	12.59	13.01	13.12	22.72	22.85	22.80
gcc -O2	(DW,FP) $\rightarrow$ FP	0.91	0.97	0.97	0.95	1.02	1.02
	Naive, binary64	0.61	0.62	0.62	0.61	0.62	0.62
	Naive, binary128	20.90	21.03	21.08	21.01	21.10	21.13
	GNU MPFR	12.31	12.74	12.85	23.11	23.20	23.18

Implementation & experiments: Timings (x86_64, Clang)

Table 2: Timings on x86_64 (in seconds, for $NK = 2^{26}$ operations) with Clang. GNU MPFR is used with separate $\pm$ and $\times$.

		minimums			maximums		
	$N \rightarrow$	1024	2048	4096	1024	2048	4096
`clang -O3`	(DW,FP) $\rightarrow$ FP	0.86	1.09	1.10	0.96	1.15	1.15
	Naive, binary64	0.39	0.61	0.63	0.47	0.65	0.66
	Naive, binary128	21.65	21.77	21.81	21.74	21.87	21.88
	GNU MPFR	12.24	12.63	12.72	22.91	22.94	22.97
`clang -O2`	(DW,FP) $\rightarrow$ FP	0.88	1.08	1.10	0.96	1.14	1.15
	Naive, binary64	0.40	0.61	0.63	0.48	0.65	0.66
	Naive, binary128	21.33	21.45	21.50	21.49	21.57	21.59
	GNU MPFR	12.15	12.54	12.65	23.15	23.21	23.21

Implementation & experiments: Timings (POWER9)

Table 3: Timings on a POWER9 (in seconds, for $NK = 2^{26}$ operations). The POWER9 has hardware support for binary128.

		minimums			maximums		
	$N \rightarrow$	1024	2048	4096	1024	2048	4096
gcc -O3 -f...	(DW,FP) → FP	0.97	0.97	0.97	0.98	0.99	1.00
	Naive, binary64	0.47	0.47	0.51	0.48	0.48	0.52
	Naive, binary128	2.22	2.22	2.22	2.24	2.24	2.24
	GNU MPFR	16.42	16.59	16.66	30.06	30.39	30.44
gcc -O2	(DW,FP) → FP	0.98	0.98	0.98	0.99	1.01	1.01
	Naive, binary64	0.47	0.47	0.51	0.47	0.47	0.51
	Naive, binary128	2.22	2.22	2.22	2.24	2.24	2.24
	GNU MPFR	16.36	16.58	16.63	30.29	30.29	30.49

Implementation & experiments: Timings summary

- **Naive formula in binary64** (inlined code) $\approx$ two times as fast as our implementation of the main algorithm, but significantly less accurate;
- **Naive formula in binary128** using the `__float128` C type (inlined code):
  - on x86_64: from 19 to 25 times as slow as our main algorithm;
  - on POWER9: 2.3 times as slow as our main algorithm.
- **GNU MPFR** using precisions from 53 to 106: from 11 to 26 times as slow as our main algorithm on x86_64, and from 17 to 31 times as slow on POWER9.

Implementation & experiments: Errors

Largest errors found by random tests:

- In binary32 arithmetic ($p = 24$), with

$$\begin{aligned} a &= \text{0x1.fbec1ep}{-36} + {-\text{0x1.0ddbc2p}{-61}} \\ b &= \text{0x1.ed2492p}{-1} + \text{0x1.2d60a2p}{-27} \\ c &= \text{0x1.09ca04p}{-1} \\ d &= \text{0x1.e85856p}{-28} \end{aligned}$$

  the normwise relative error η is $\approx 0.99999990056894153671\,\mathrm{u}$.

- In binary64 arithmetic ($p = 53$), with

$$\begin{aligned} a &= \text{0x1.ca8960d0529ap}{-50} + {-\text{0x1.d3bbedca6980bp}{-104}} \\ b &= \text{0x1.5d23517609dcp}{-1} + {-\text{0x1.9cd4b29e547d9p}{-57}} \\ c &= \text{0x1.776a8388a7d6cp}{-1} \\ d &= \text{0x1.defea2385e587p}{-79} \end{aligned}$$

  the normwise relative error η is $\approx 0.99999974195846572521\,\mathrm{u}$.

Note: smaller than the normwise relative error $u/(1+u)$ of the trivial case $(1+u)\cdot 1$, i.e. with $a_h = 1$, $a_\ell = u$, $b_h = b_\ell = 0$, $c = 1$, $d = 0$.

Other variants (work in progress)

The addition of the 5 error terms $a_\ell c - b_\ell d + e_1 - e_2 + v_\ell$ can be recombined in various ways. Possible goals:

- Minimize the number of operations by adding both $a_\ell c$ and $-b_\ell d$ with FMAs. But what about the error?
- Minimize the proven error bound (the worst case is unknown).
- Minimize the average error (in absolute value) for some distribution.

Proposed change to minimize the number of operations (5 to 7% faster):

Old

$$\begin{aligned} \rho_1 &\leftarrow \mathrm{RN}(b_\ell d) \\ \rho_2 &\leftarrow \mathrm{RN}(a_\ell c - \rho_1) \\ \rho_3 &\leftarrow \mathrm{RN}(\rho_2 - e_2) \\ \sigma &\leftarrow \mathrm{RN}(\rho_3 + e_1) \end{aligned}$$

$\rightarrow$

New

$$\begin{aligned} r_1 &\leftarrow \mathrm{RN}(a_\ell c + e_1) \\ r_2 &\leftarrow \mathrm{RN}(b_\ell d + e_2) \\ \sigma &\leftarrow \mathrm{RN}(r_1 - r_2) \end{aligned}$$

New normwise relative error bound: $\eta < u + 23\,u^2$ (instead of $\eta < u + 33\,u^2$).

Other variants (work in progress) [2]

Old

$$\begin{aligned} \rho_1 &\leftarrow \mathrm{RN}(b_\ell d) \\ \rho_2 &\leftarrow \mathrm{RN}(a_\ell c - \rho_1) \\ \rho_3 &\leftarrow \mathrm{RN}(\rho_2 - e_2) \\ \sigma &\leftarrow \mathrm{RN}(\rho_3 + e_1) \end{aligned}$$

$\rightarrow$

New

$$\begin{aligned} r_1 &\leftarrow \mathrm{RN}(a_\ell c + e_1) \\ r_2 &\leftarrow \mathrm{RN}(b_\ell d + e_2) \\ \sigma &\leftarrow \mathrm{RN}(r_1 - r_2) \end{aligned}$$

Two improvements concerning the proven error bound:

- Fewer rounding errors: FMA instead of a multiplication and an addition (this should also reduce the average error).
- More symmetry, used in the proof:
  - Old: $|a_\ell c - \rho_1| \leqslant u \cdot |ac| + (u + u^2) \cdot |bd|$
    $\leqslant (u + u^2) \cdot (|ac| + |bd|) = (u + u^2) \cdot N,$
    i.e. an additional overestimation of $u^2 \cdot |ac|$.
  - New: $|r_1| \leqslant (2\,u + 3\,u^2 + u^3) \cdot |ac|,$
    $|r_2| \leqslant (2\,u + 3\,u^2 + u^3) \cdot |bd|,$
    thus $|r_1 - r_2| \leqslant (2\,u + 3\,u^2 + u^3) \cdot N.$

Conclusion

- Main algorithm:
  - the real and imaginary parts of one of the operands of the multiplication are DW, and for the other one they are FP;
  - normwise relative error bound close to the best one ($u/(1+u) \approx u$) that one can guarantee,
  - only twice as slow as a naive multiplication,
  - much faster than binary128 or multiple-precision software.
- Multiple variants:
  - depending on the types of the inputs (can all be FP) and the output (can be DW);
  - depending on the combination of the sums of the error terms in the algorithm.