Accurate Complex Multiplication in Floating-Point Arithmetic

Vincent Lefèvre   Jean-Michel Muller.

Université de Lyon, CNRS, Inria, France.

Arith26,

Kyoto, June 2019

Accurate complex multiplication in FP arithmetic

- $\omega \cdot x$, emphasis on the case where $\Re(\omega)$ and $\Im(\omega)$ are double-word numbers—i.e., pairs (high-order, low-order) of FP numbers;
- applications: Fourier transforms, iterated products.

Assumptions:

- radix-2, precision-p, FP arithmetic;
- rounded to nearest (RN) FP operations;
- an FMA instruction is available;
- underflow/overflow do not occur.

Bound on relative error of (real) operations:

$$|\mathrm{RN}(a+b)-(a+b)| \leqslant \frac{u}{1+u}\cdot|a+b| < u\cdot|a+b|,$$

where u (rounding unit) equals 2^{-p}.

Some variables: **double-word** (DW) numbers

- also called double-double in the literature;
- $v \in \mathbb{R}$ represented by a pair of FP numbers v_h and v_ℓ such that

$$\begin{aligned} &v = v_h + v_\ell, \\ &|v_\ell| \leqslant \tfrac{1}{2}\mathrm{ulp}(v) \leqslant u \cdot |v|. \end{aligned}$$

- algorithms and libraries for manipulating DW numbers: QD (Hida, Li & Bailey), Campary (Joldes, Popescu & others),
- use the 2Sum, Fast2Sum & Fast2Mult algorithms (see later).

Naive algorithms for complex FP multiplication

- straightforward transcription of the formula

$$z = (x^R + ix^I) \cdot (y^R + iy^I) = (x^R y^R - x^I y^I) + i \cdot (x^I y^R + x^R y^I);$$

- bad solution if componentwise relative error is to be minimized;
- adequate solution if normwise relative error is at stake.
  ($\hat{z}$ approximates z with normwise error $|(\hat{z} - z)/z|$)

Algorithms:

- if no FMA instruction is available

$$\begin{cases} \hat{z}^R &= \mathrm{RN}(\mathrm{RN}(x^R y^R) - \mathrm{RN}(x^I y^I)), \\ \hat{z}^I &= \mathrm{RN}(\mathrm{RN}(x^R y^I) + \mathrm{RN}(x^I y^R)). \end{cases} \tag{1}$$

- if an FMA instruction is available

$$\begin{cases} \hat{z}^R &= \mathrm{RN}(x^R y^R - \mathrm{RN}(x^I y^I)), \\ \hat{z}^I &= \mathrm{RN}(x^R y^I + \mathrm{RN}(x^I y^R)). \end{cases} \tag{2}$$

Naive algorithms for complex multiplication

- if no FMA instruction is available

$$\left\{ \begin{array}{lcl} \hat{z}^R & = & \mathrm{RN}(\mathrm{RN}(x^R y^R) - \mathrm{RN}(x^I y^I)), \\ \hat{z}^I & = & \mathrm{RN}(\mathrm{RN}(x^R y^I) + \mathrm{RN}(x^I y^R)). \end{array} \right. \tag{1}$$

- if an FMA instruction is available

$$\left\{ \begin{array}{lcl} \hat{z}^R & = & \mathrm{RN}(x^R y^R - \mathrm{RN}(x^I y^I)), \\ \hat{z}^I & = & \mathrm{RN}(x^R y^I + \mathrm{RN}(x^I y^R)). \end{array} \right. \tag{2}$$

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

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

Accurate complex multiplication

Our goal:

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

We consider the product

$$\omega \cdot x,$$

with

$$\omega = \omega^R + i \cdot \omega^I \text{ and } x = x^R + i \cdot x^I,$$

where:

- ω^R and ω^I are DW numbers (special case FP considered later)
- x^R and x^I are FP numbers.

Basic building blocks: Error-Free Transforms

Expressing $a + b$ as a DW number

Algorithm 1: **2Sum**(a, b). Returns s and t such that $s = \mathrm{RN}(a + b)$ and $t = a + b - s$

```
$s \leftarrow \mathrm{RN}(a + b)$
$a' \leftarrow \mathrm{RN}(s - b)$
$b' \leftarrow \mathrm{RN}(s - a')$
$\delta_a \leftarrow \mathrm{RN}(a - a')$
$\delta_b \leftarrow \mathrm{RN}(b - b')$
$t \leftarrow \mathrm{RN}(\delta_a + \delta_b)$
```

Expressing $a \cdot b$ as a DW number

Algorithm 2: **Fast2Mult**(a, b). Returns π and ρ such that $\pi = \mathrm{RN}(ab)$ and $\rho = ab - \pi$

```
$\pi \leftarrow \mathrm{RN}(ab)$
$\rho \leftarrow \mathrm{RN}(ab - \pi)$
```

The multiplication algorithm

- $\omega^R = \Re(\omega)$ and $\omega^I = \Im(\omega)$: DW numbers, i.e.,

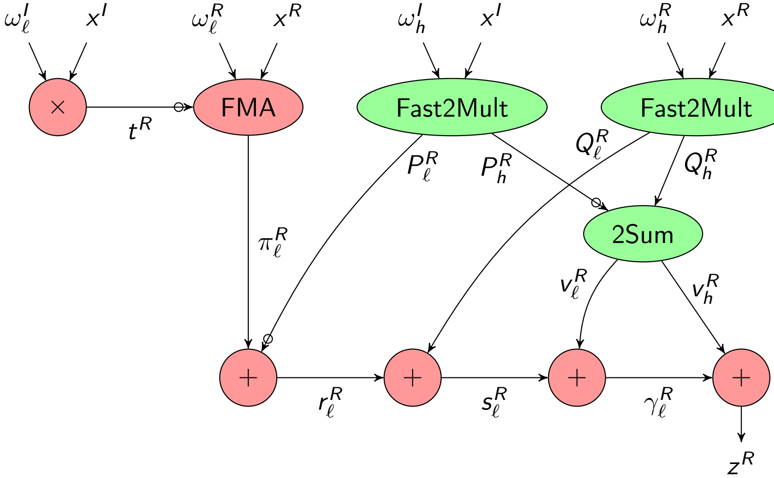
  $$\omega = \omega^R + i \cdot \omega^I = (\omega_h^R + \omega_\ell^R) + i \cdot (\omega_h^I + \omega_\ell^I),$$

  where ω_h^R, ω_ℓ^R, ω_h^I, and ω_ℓ^I are FP numbers that satisfy:
  - $|\omega_\ell^R| \leqslant \frac{1}{2}\mathrm{ulp}(\omega^R) \leqslant u \cdot |\omega^R|$;
  - $|\omega_\ell^I| \leqslant \frac{1}{2}\mathrm{ulp}(\omega^I) \leqslant u \cdot |\omega^I|$.

- Real part z^R of the result (similar for imaginary part):
  - difference v_h^R of the high-order parts of $\omega_h^R x^R$ and $\omega_h^I x^I$,
  - add approximated sum γ_ℓ^R of all the error terms that may have a significant influence on the normwise relative error.

- rather straightforward algorithms: the tricky part is the error bounds.

Real part $(\omega_h^R + \omega_\ell^R) \cdot x^R - (\omega_h^I + \omega_\ell^I) \cdot x^I$



The multiplication algorithm

Algorithm 3: Computes $\omega \cdot x$, where the real & imaginary parts of $\omega = (\omega_h^R + \omega_\ell^R) + i \cdot (\omega_h^I + \omega_\ell^I)$ are DW, and the real & im. parts of x are FP.

```
$t^R \leftarrow \mathrm{RN}(\omega_\ell^I x^I)$
$\pi_\ell^R \leftarrow \mathrm{RN}(\omega_\ell^R x^R - t^R)$
$(P_h^R, P_\ell^R) \leftarrow \mathrm{Fast2Mult}(\omega_h^I, x^I)$
$r_\ell^R \leftarrow \mathrm{RN}(\pi_\ell^R - P_\ell^R)$
$(Q_h^R, Q_\ell^R) \leftarrow \mathrm{Fast2Mult}(\omega_h^R, x^R)$
$s_\ell^R \leftarrow \mathrm{RN}(Q_\ell^R + r_\ell^R)$
$(v_h^R, v_\ell^R) \leftarrow \mathrm{2Sum}(Q_h^R, -P_h^R)$
$\gamma_\ell^R \leftarrow \mathrm{RN}(v_\ell^R + s_\ell^R)$
return $z^R = \mathrm{RN}(v_h^R + \gamma_\ell^R)$ (real part)
$t^I \leftarrow \mathrm{RN}(\omega_\ell^I x^R)$
$\pi_\ell^I \leftarrow \mathrm{RN}(\omega_\ell^R x^I + t^I)$
$(P_h^I, P_\ell^I) \leftarrow \mathrm{Fast2Mult}(\omega_h^I, x^R)$
$r_\ell^I \leftarrow \mathrm{RN}(\pi_\ell^I + P_\ell^I)$
$(Q_h^I, Q_\ell^I) \leftarrow \mathrm{Fast2Mult}(\omega_h^R, x^I)$
$s_\ell^I \leftarrow \mathrm{RN}(Q_\ell^I + r_\ell^I)$
$(v_h^I, v_\ell^I) \leftarrow \mathrm{2Sum}(Q_h^I, P_h^I)$
$\gamma_\ell^I \leftarrow \mathrm{RN}(v_\ell^I + s_\ell^I)$
return $z^I = \mathrm{RN}(v_h^I + \gamma_\ell^I)$ (imaginary part)
```

The multiplication algorithm

Theorem 1

As soon as $p \geqslant 4$, *the normwise relative error* η *of Algorithm 3 satisfies*

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

(remember: the best possible bound is u)

Remarks:

- Condition "$p \geqslant 4$" always holds in practice;
- Algorithm 3 easily transformed (see later) into an algorithm that returns the real and imaginary parts of z as DW numbers.

Sketch of the proof

- first, we show that

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

  with

$$\begin{array}{rcl} N^R & = & |\omega^R x^R| + |\omega^I x^I|, \\ n^R & = & |\omega^R x^R - \omega^I x^I|, \\ N^I & = & |\omega^R x^I| + |\omega^I x^R|, \\ n^I & = & |\omega^R x^I + \omega^I x^R|, \\ \alpha & = & u + 3u^2 + u^3, \\ \beta & = & 15u^2 + 38u^3 + 39u^4 + 22u^5 + 7u^6 + u^7; \end{array}$$

- then we deduce

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

- the theorem follows, by using

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

Obtaining the real and imaginary parts of z as DW numbers

- replace the FP addition $z^R = \mathrm{RN}(v_h^R + \gamma_\ell^R)$ of line 9 of Algorithm 3 by a call to $\mathrm{2Sum}(v_h^R, \gamma_\ell^R)$,
- replace the FP addition $z^I = \mathrm{RN}(v_h^I + \gamma_\ell^I)$ of line 18 by a call to $\mathrm{2Sum}(v_h^I, \gamma_\ell^I)$.
- resulting relative error

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

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

Interest:

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

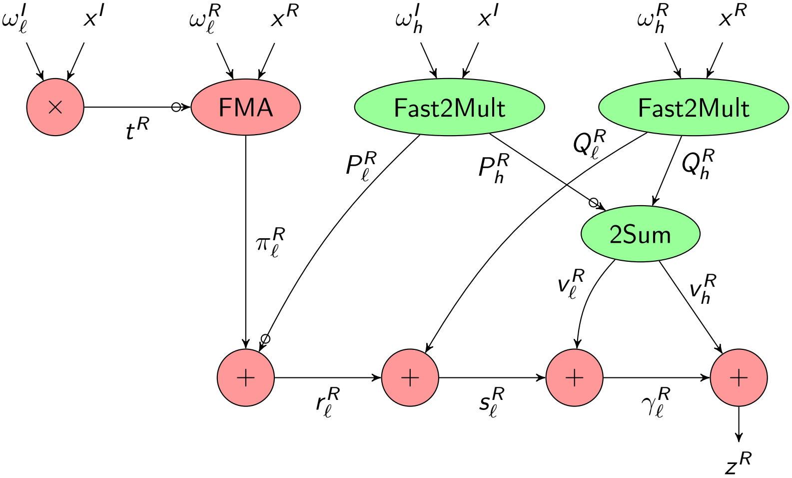
If ω^I and ω^R are floating-point numbers

$\omega_\ell^I = \omega_\ell^R = 0 \Rightarrow$ Algorithm 3 becomes simpler:

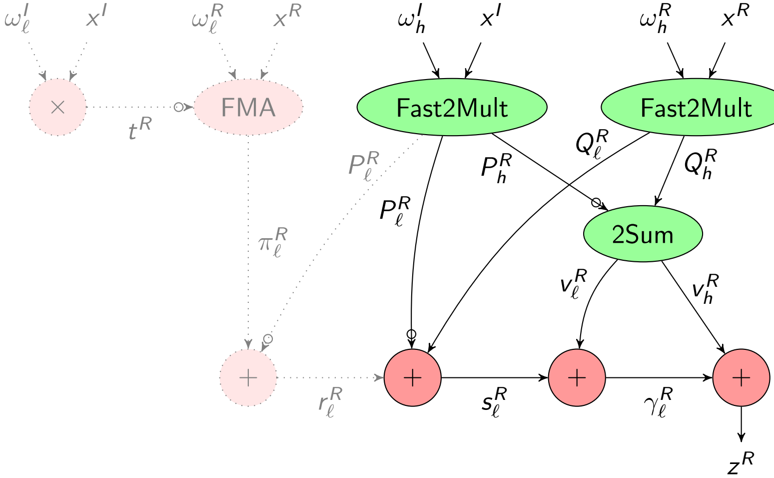
Algorithm 4: Complex multiplication $\omega \cdot x$, where $\Re(\omega)$ and $\Im(\omega)$ are FP numbers.

```
$(P_h^R, P_\ell^R) \leftarrow$ Fast2Mult$(\omega^I, x^I)$
$(Q_h^R, Q_\ell^R) \leftarrow$ Fast2Mult$(\omega^R, x^R)$
$s_\ell^R \leftarrow$ RN$(Q_\ell^R - P_\ell^R)$
$(v_h^R, v_\ell^R) \leftarrow$ 2Sum$(Q_h^R, -P_h^R)$
$\gamma_\ell^R \leftarrow$ RN$(v_\ell^R + s_\ell^R)$
**return** $z^R =$ RN$(v_h^R + \gamma_\ell^R)$ (real part)
$(P_h^I, P_\ell^I) \leftarrow$ Fast2Mult$(\omega^I, x^R)$
$(Q_h^I, Q_\ell^I) \leftarrow$ Fast2Mult$(\omega^R, x^I)$
$s_\ell^I \leftarrow$ RN$(Q_\ell^I + P_\ell^I)$
$(v_h^I, v_\ell^I) \leftarrow$ 2Sum$(Q_h^I, P_h^I)$
$\gamma_\ell^I \leftarrow$ RN$(v_\ell^I + s_\ell^I)$
**return** $z^I =$ RN$(v_h^I + \gamma_\ell^I)$ (imaginary part)
```

Real part

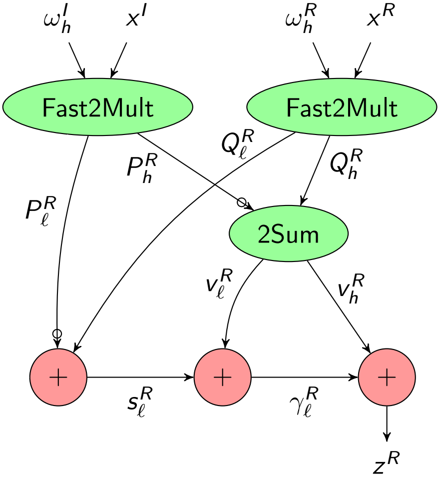

ω_ℓ^I
x^I
ω_ℓ^R
x^R
ω_h^I
x^I
ω_h^R
x^R
×
FMA
Fast2Mult
Fast2Mult
t^R
P_ℓ^R
P_h^R
Q_ℓ^R
Q_h^R
π_ℓ^R
2Sum
v_ℓ^R
v_h^R
+
+
+
+
r_ℓ^R
s_ℓ^R
γ_ℓ^R
z^R

Real part



ω_ℓ^I
x^I
ω_ℓ^R
x^R
ω_h^I
x^I
ω_h^R
x^R
×
t^R
FMA
Fast2Mult
Fast2Mult
P_ℓ^R
P_ℓ^R
P_h^R
Q_ℓ^R
Q_h^R
π_ℓ^R
2Sum
v_ℓ^R
v_h^R
+
+
+
+
r_ℓ^R
s_ℓ^R
γ_ℓ^R
z^R

Real part


ω_h^I
x^I
ω_h^R
x^R
Fast2Mult
Fast2Mult
Q_ℓ^R
P_h^R
Q_h^R
P_ℓ^R
2Sum
v_ℓ^R
v_h^R
+
+
+
s_ℓ^R
γ_ℓ^R
z^R

If ω^I and ω^R are floating-point numbers

- Real and complex parts of Algorithm 4 similar to:
  - Cornea, Harrison and Tang's algorithm for $ab + cd$ (with a "+" replaced by a 2Sum),
  - Alg. 5.3 in Ogita, Rump and Oishi's *Accurate sum & dot product* (with different order of summation of P_ℓ^R, Q_ℓ^R & v_ℓ^R).
- The error bound $u + 33u^2$ of Theorem 1 still applies, but it can be slightly improved:

Theorem 2

As soon as $p \geqslant 4$, *the normwise relative error* η *of Algorithm 4 satisfies*

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

Implementation and experiments

- Main algorithm (Algorithm 3) implemented in binary64 (a.k.a. double-precision) arithmetic, 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 and experiments

- 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, 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.
- In all cases, `-static` used to avoid the overhead due to function calls to dynamic libraries.

Implementation and experiments

Table 1: Timings on x86_64 (in secs, for $NK = 2^{26}$ ops) with GCC. GNU MPFR is used with separate ± and ×.

		minimums			maximums		
	$N \rightarrow$	1024	2048	4096	1024	2048	4096
`gcc -O3 -f...`	Algorithm 3	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`	Algorithm 3	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 and experiments

Table 2: Timings on x86_64 (in secs, for $NK = 2^{26}$ ops) with Clang.

		minimums			maximums		
	$N \rightarrow$	1024	2048	4096	1024	2048	4096
`clang -O3`	Algorithm 3	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`	Algorithm 3	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 and experiments

Table 3: Timings on a POWER9 (in secs, for $NK = 2^{26}$ ops). The POWER9 has hardware support for Binary128.

		minimums			maximums		
	$N \rightarrow$	1024	2048	4096	1024	2048	4096
`gcc -O3 -f...`	Algorithm 3	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`	Algorithm 3	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 and experiments

- **Naive formula in binary64** (inlined code) ≈ two times as fast as our implementation of Algorithm 3, but significantly less accurate;
- **Naive formula in binary128,** using the `__float128` C type (inlined code):
  - x86_64: from 19 to 25 times as slow as Algorithm 3,
  - on POWER9: 2.3 times as slow.
- **GNU MPFR** using precisions from 53 to 106: from 11 to 26 times as slow as Algorithm 3 on x86_64, and from 17 to 31 times as slow on POWER9.

The error bound of Theorem 1 is tight: In Binary64 arithmetic, with

$$\begin{aligned}
\omega^R &= \mathrm{0x1.d1ef9ea4aa013p{-}1} + \mathrm{0x1.ae88ba2a277ep{-}56}\\
\omega^I &= \mathrm{0x1.f5c28321df365p{-}81} + \mathrm{0x1.c4c3e7b506d06p{-}135}\\
x^R &= \mathrm{0x1.194f298b4d152p{-}1}\\
x^I &= \mathrm{0x1.5c1fdca444f7cp{-}14}
\end{aligned}$$

the normwise relative error is $0.99999900913907117123\,\mathrm{u}$.

Conclusion

- Main algorithm:
  - the real and imaginary parts of one of the operands are DW, and for the other one they are FP,
  - normwise relative error bound close to the best one (u) that one can guarantee,
  - only twice as slow as a naive multiplication,
  - much faster than binary128 or multiple-precision software.
- 2 variants:
  - real and imaginary parts of the output are DW,
  - real and imaginary parts of the inputs are FP.

Conclusion

- Main algorithm:
  - the real and imaginary parts of one of the operands are DW, and for the other one they are FP,
  - normwise relative error bound close to the best one (u) that one can guarantee,
  - only twice as slow as a naive multiplication,
  - much faster than binary128 or multiple-precision software.
- 2 variants:
  - real and imaginary parts of the output are DW,
  - real and imaginary parts of the inputs are FP.

Thank you!