Accurate Complex Multiplication in Floating-Point Arithmetic

Vincent Lefèvre, Jean-Michel Muller

Accurate Complex Multiplication in Floating-Point Arithmetic

Vincent Lefèvre*, Jean-Michel Muller*
* Univ Lyon, EnsL, UCBL, CNRS, Inria, LIP, F-69342, LYON Cedex 07, France

***Abstract*—We deal with accurate complex multiplication in binary floating-point arithmetic, with an emphasis on the case where one of the operands is a "double-word" number. We provide an algorithm that returns a complex product with normwise relative error bound close to the best possible one, i.e., the rounding unit u.**

Keywords. Floating-point arithmetic, Complex multiplication, Rounding error analysis.

I. Introduction and notation

This paper deals with accurate (from a normwise point of view) complex multiplication in binary floating-point arithmetic, with an emphasis on the case where the real and imaginary parts of one of the operands are "double-word" numbers. This is of interest, for instance, when that operand is a root of one (which is the case in fast Fourier transforms), precomputed and stored in higher precision than standard floating-point precision. This can also be of interest for computing iterated products of several complex numbers accurately. In the following, we assume a radix-2, precision-p floating-point arithmetic, with correctly rounded (to nearest) arithmetic operations. We assume that an FMA (Fused Multiply-Add) instruction is available and, to simplify our study, we also assume an unbounded exponent range. This means that the results presented in this paper apply to "real-life" IEEE 754 Floating-Point arithmetic [6] as long as underflows and overflows do not occur.

We will denote $u = 2^{-p}$ the "rounding unit". For instance in the binary64 format of the IEEE 754 Standard (a.k.a. "double precision"), $u = 2^{-53}$. RN is the round-to-nearest function (with any choice in case of a tie). For instance, when performing the operation $a+b$, where a and b are floating-point numbers, the obtained result is $\mathrm{RN}(a+b)$, and it satisfies:

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

When implementing complex operations or functions, if the computed result $\hat{z}$ has the form $\hat{z}^R + i \cdot \hat{z}^I$, where $\hat{z}^R$ and $\hat{z}^I$ are floating-point numbers, and if the exact result is $z = z^R + i \cdot z^I$, one may be interested in minimizing the *componentwise relative error*

$$\max\left\{\left|\frac{\hat{z}^R - z^R}{z^R}\right|; \left|\frac{\hat{z}^I - z^I}{z^I}\right|\right\},$$

or the *normwise relative error*

$$\left|\frac{\hat{z} - z}{z}\right|.$$

An algorithm that would always return the best possible result (i.e., a real part equal to the floating-point number nearest to the exact real part, and an imaginary part equal to the floating-point number nearest to the exact imaginary part) would have worst case componentwise and normwise relative error $u/(1+u) \approx u$: this is therefore the best error bound achievable by an algorithm that returns the real and imaginary parts of the result in floating point. Our goal here is to obtain small *normwise* relative errors when computing complex products, i.e., small values of $|(\hat{z} - z)/z|$, where z is the exact product and $\hat{z}$ is the computed product.

We will need to represent some variables by *double-word* numbers. A double-word number (frequently called "double-double" in the literature, because the underlying floating-point format is, in general, the binary64 format) [9], [4] is a pair of floating-point numbers v_h and v_ℓ that represents a real number v such that

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

To compute an approximation $\hat{z}^R + i\hat{z}^I$ to $z^R + iz^I = (x^R + ix^I) \cdot (y^R + iy^I)$, where x^R, x^I, y^R, y^I, $\hat{z}^R$, and $\hat{z}^I$ are floating-point numbers, one may consider the following "naive" formulas:

- if no FMA instruction is available

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

- if an FMA instruction is available

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

Formulas (1) and (2), as well as the algorithm we give in this paper (Algorithm 3), can lead to large componentwise relative errors. To obtain small componentwise errors, one needs to use significantly different algorithms, such as an algorithm attributed to Kahan by Higham [5, p. 65], analyzed in [8], or Cornea et al's algorithm for $ab + cd$ presented in [2].

Asymptotically optimal bounds on the normwise relative error of formulas (1) and (2) are known: Brent et al. [1] show a bound $\sqrt{5} \cdot u$ for (1), and Jeannerod et al. [7] show a bound $2 \cdot u$ for (2).

We aim at obtaining smaller normwise relative errors, closer to the best possible one, at the cost of more complex algorithms. We consider the product

$$\omega \times x,$$

with

$$x = x^R + i \cdot x^I,$$

where x^R and x^I are floating-point numbers.

In Section II we deal with the case where the real and imaginary parts ω^R and ω^I of ω are double-word numbers and the real and imaginary parts of the product are floating-point numbers. The obtained algorithm (Algorithm 3) will then be somehow simplified to consider two cases of interest: the case where the real and imaginary parts of the product are double-word numbers (Section III-A) and the case where ω^R and ω^I are floating-point numbers (Section III-B).

We will need two well-known algorithms of the floating-point literature: Algorithm 2Sum (Algorithm 1 below), that takes two floating-point numbers a and b as input and returns two floating-point numbers s and t such that $s = \mathrm{RN}(a+b)$ and $t = a + b - s$ (that is, t is the error of the floating-point addition of a and b), and Algorithm Fast2Mult (Algorithm 2 below), that requires the availability of an FMA instruction, and takes two floating-point numbers a and b as input and returns two floating-point numbers π and ρ such that $\pi = \mathrm{RN}(ab)$ and $\rho = ab - \pi$ (that is, ρ is the error of the floating-point multiplication of a and b).

ALGORITHM 1: 2Sum(a, b). The 2Sum algorithm [12], [11].

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

ALGORITHM 2: Fast2Mult(a, b)**.** The Fast2Mult algorithm (see for instance [10], [14], [13]). It requires the availability of a fused multiply-add (FMA) instruction for computing $\mathrm{RN}(ab - \pi)$.

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

II. The multiplication algorithm

In this section, we assume that the real and imaginary parts of ω are double-word 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 floating-point numbers that satisfy:

- $|\omega_\ell^R| \leqslant (1/2)\mathrm{ulp}(\omega^R) \leqslant u \cdot |\omega^R|$;
- $|\omega_\ell^I| \leqslant (1/2)\mathrm{ulp}(\omega^I) \leqslant u \cdot |\omega^I|$.

For performing the complex multiplication $\omega \cdot x$, we introduce Algorithm 3 below. The real part (lines 1 to 9) and the imaginary part (lines 10 to 18) can obviously be computed in parallel, and within these parts, additional parallelism is possible. For instance lines 3 and 5 can run in parallel with line 1, and line 7 can run in parallel with line 6. This parallelism is easily exploited by recent compilers. This explains the good performance we obtain (see Section IV). Roughly speaking, Algorithm 3 computes the real part z^R of the result by computing the difference v_h^R of the high-order parts of the products $\omega_h^R x^R$ and $\omega_h^I x^I$, and adding the approximated sum γ_ℓ^R of all the error terms that could have a significant influence on the normwise relative error. The imaginary part z^I of the result is computed in a similar way.

ALGORITHM 3: Accurate complex multiplication $\omega \cdot x$, where the real and imaginary parts of $\omega = (\omega_h^R + \omega_\ell^R) + i \cdot (\omega_h^I + \omega_\ell^I)$ are double-word numbers, and the real and imaginary parts of x are floating-point numbers.

1: $t^R \leftarrow \mathrm{RN}(\omega_\ell^I x^I)$
2: $\pi_\ell^R \leftarrow \mathrm{RN}(\omega_\ell^R x^R - t^R)$
3: $(P_h^R, P_\ell^R) \leftarrow$ Fast2Mult(ω_h^I, x^I)
4: $r_\ell^R \leftarrow \mathrm{RN}(\pi_\ell^R - P_\ell^R)$
5: $(Q_h^R, Q_\ell^R) \leftarrow$ Fast2Mult(ω_h^R, x^R)
6: $s_\ell^R \leftarrow \mathrm{RN}(Q_\ell^R + r_\ell^R)$
7: $(v_h^R, v_\ell^R) \leftarrow 2\mathrm{Sum}(Q_h^R, -P_h^R)$
8: $\gamma_\ell^R \leftarrow \mathrm{RN}(v_\ell^R + s_\ell^R)$
9: **return** $z^R = \mathrm{RN}(v_h^R + \gamma_\ell^R)$ (real part)
10: $t^I \leftarrow \mathrm{RN}(\omega_\ell^I x^R)$
11: $\pi_\ell^I \leftarrow \mathrm{RN}(\omega_\ell^R x^I + t^I)$
12: $(P_h^I, P_\ell^I) \leftarrow$ Fast2Mult(ω_h^I, x^R)
13: $r_\ell^I \leftarrow \mathrm{RN}(\pi_\ell^I + P_\ell^I)$
14: $(Q_h^I, Q_\ell^I) \leftarrow$ Fast2Mult(ω_h^R, x^I)
15: $s_\ell^I \leftarrow \mathrm{RN}(Q_\ell^I + r_\ell^I)$
16: $(v_h^I, v_\ell^I) \leftarrow 2\mathrm{Sum}(Q_h^I, P_h^I)$
17: $\gamma_\ell^I \leftarrow \mathrm{RN}(v_\ell^I + s_\ell^I)$
18: **return** $z^I = \mathrm{RN}(v_h^I + \gamma_\ell^I)$ (imaginary part)

Our main result is

Theorem 1. *As soon as $p \geqslant 4$, the normwise relative error η of Algorithm 3 satisfies*

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

The condition "$p \geqslant 4$" of Theorem 1 always holds in practice. Note that Algorithm 3 can easily be transformed into an algorithm that returns the real and imaginary parts of the product as double-word numbers, in order to reduce the error: it suffices to replace the floating-point additions of lines 9 and 18 by a call to 2Sum (or to the somehow simpler Fast2Sum algorithm, see for instance [13]). We deal with this solution in Section III-A.

Theorem 1 uses the following lemma, that we will prove first.

Lemma 1 (Componentwise absolute error of Algorithm 3)**.** *We have*

$$\begin{aligned} |z^R - \Re(wx)| &\leqslant \alpha n^R + \beta N^R, \\ |z^I - \Im(wx)| &\leqslant \alpha n^I + \beta N^I, \end{aligned} \tag{3}$$

with

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

Proof. We will focus on the calculation of the real part of the complex product (i.e., lines 1 to 9 of the algorithm), since the results for the real part hold for the imaginary part through a simple symmetry argument (namely, the imaginary part of $(a+ib)\cdot(c+id)$ is equal to the real part of $(b-ia)\cdot(c+id)$).

A. Lines 1-2 of Algorithm 3: computation of an approximation π_ℓ^R to $(\omega_\ell^R x^R - \omega_\ell^I x^I)$.

We have

$$|t^R - (\omega_\ell^I x^I)| \leqslant u^2 \cdot |\omega^I x^I|, \tag{4}$$

and $|t^R| \leqslant (u+u^2)\cdot|\omega^I x^I|$, so that

$$\begin{array}{rcl} |\omega_\ell^R x^R - t^R| &\leqslant& u\cdot|\omega^R x^R| + (u+u^2)\cdot|\omega^I x^I| \\ &\leqslant& (u+u^2)\cdot(|\omega^R x^R| + |\omega^I x^I|) \\ &=& (u+u^2)\cdot N^R, \end{array}$$

a consequence of which is

$$|\pi_\ell^R - (\omega_\ell^R x^R - t^R)| \leqslant (u^2+u^3)\cdot N^R, \tag{5}$$

and

$$|\pi_\ell^R| \leqslant (u + 2u^2 + u^3)\cdot N^R.$$

From (4) and (5), we obtain

$$\left|\pi_\ell^R - (\omega_\ell^R x^R - \omega_\ell^I x^I)\right| \leqslant \epsilon_1, \tag{6}$$

with

$$\epsilon_1 = (2u^2+u^3)\cdot N^R.$$

B. Line 3.

We have

$$P_h^R + P_\ell^R = \omega_h^I x^I, \tag{7}$$

with

$$|P_\ell^R| \leqslant u\cdot|\omega_h^I x^I| \leqslant u(1+u)\cdot|\omega^I x^I|,$$

and

$$|P_h^R| \leqslant (1+u)^2\cdot|\omega^I x^I|.$$

C. Line 4.

From

$$\begin{array}{rcl} |\pi_\ell^R - P_\ell^R| &\leqslant& (u+u^2)\cdot|\omega^I x^I| + (u+2u^2+u^3)\cdot N^R \\ &\leqslant& (2u+3u^2+u^3)\cdot N^R, \end{array}$$

we obtain

$$|r_\ell^R - (\pi_\ell^R - P_\ell^R)| \leqslant \epsilon_2, \tag{8}$$

with

$$\epsilon_2 = (2u^2+3u^3+u^4)\cdot N^R,$$

and

$$|r_\ell^R| \leqslant (2u+5u^2+4u^3+u^4)\cdot N^R, \tag{9}$$

and, using (6) and (8),

$$\begin{array}{c} \left|r_\ell^R - (\omega_\ell^R x^R - \omega_\ell^I x^I - P_\ell^R)\right| \\ \leqslant (4u^2+4u^3+u^4)\cdot N^R. \end{array} \tag{10}$$

D. Lines 5 and 6.

We have

$$Q_h^R + Q_\ell^R = \omega_h^R x^R, \tag{11}$$

and

$$|Q_\ell^R| \leqslant u(1+u)\cdot|\omega^R x^R|.$$

From this bound on $|Q_\ell^R|$ and (9), we obtain

$$\left|Q_\ell^R + r_\ell^R\right| \leqslant (3u+6u^2+4u^3+u^4)\cdot N^R,$$

from which we deduce

$$\left|s_\ell^R - (Q_\ell^R + r_\ell^R)\right| \leqslant \epsilon_3, \tag{12}$$

with

$$\epsilon_3 = (3u^2+6u^3+4u^4+u^5)\cdot N^R,$$

and

$$\left|s_\ell^R\right| \leqslant (3u+9u^2+10u^3+5u^4+u^5)\cdot N^R. \tag{13}$$

All this gives

$$\begin{array}{rcl} (Q_h^R - P_h^R + s_\ell^R) &=& (Q_h^R - P_h^R + Q_\ell^R + r_\ell^R) \\ && +\xi_3 \\ &=& (Q_h^R + Q_\ell^R - P_h^R + (\pi_\ell^R - P_\ell^R)) \\ && +\xi_3 + \xi_2 \\ &=& (\omega_h^R x^R - \omega_h^I x^I + \omega_\ell^R x^R - \omega_\ell^I x^I) \\ && +\xi_3 + \xi_2 + \xi_1, \end{array}$$

with

$$|\xi_3| \leqslant \epsilon_3 = (3u^2+6u^3+4u^4+u^5)\cdot N^R,$$

(from (12)), and

$$|\xi_2| \leqslant \epsilon_2 = (2u^2+3u^3+u^4)\cdot N^R,$$

(from (8)), and

$$|\xi_1| \leqslant \epsilon_1 = (2u^2+u^3)\cdot N^R,$$

(from (6), (7), and (11)). This implies

$$\begin{array}{c} \left|(Q_h^R - P_h^R + s_\ell^R) - (\omega^R x^R - \omega^I x^I)\right| \\ \leqslant (7u^2+10u^3+5u^4+u^5)\cdot N^R. \end{array} \tag{14}$$

E. Lines 7 to 9.

We have

$$\begin{array}{rcl} |Q_h^R - P_h^R| &=& |\omega_h^R x^R - \omega_h^I x^I - Q_\ell^R + P_\ell^R| \\ &\leqslant& n^R + (2u+u^2)\cdot N^R, \end{array}$$

hence,

$$|v_\ell^R| \leqslant u\cdot n^R + (2u^2+u^3)\cdot N^R,$$

and

$$|v_h^R| \leqslant (1+u)\cdot n^R + (2u+3u^2+u^3)\cdot N^R.$$

Therefore, using (13),

$$|v_\ell^R + s_\ell^R| \leqslant u\cdot n^R + (3u+11u^2+11u^3+5u^4+u^5)\cdot N^R,$$

so that

$$\begin{array}{c} \left|\gamma_\ell^R - (v_\ell^R + s_\ell^R)\right| \\ \leqslant u^2\cdot n^R \\ +(3u^2+11u^3+11u^4+5u^5+u^6)\cdot N^R, \end{array} \tag{15}$$

and

$$\begin{aligned}\left|\gamma_\ell^R\right| &\leqslant (u+u^2)\cdot n^R\\ &+(3u+14u^2+22u^3+16u^4+6u^5+u^6)\cdot N^R.\end{aligned}$$

This gives

$$\begin{aligned}\left|v_h^R+\gamma_\ell^R\right| &\leqslant (1+2u+u^2)\cdot n^R\\ &+(5u+17u^2+23u^3+16u^4+6u^5+u^6)\cdot N^R,\end{aligned}$$

so that the error of the final addition $v_h^R+\gamma_\ell^R$ is bounded by

$$\begin{aligned}&(u+2u^2+u^3)\cdot n^R\\ &+(5u^2+17u^3+23u^4+16u^5+6u^6+u^7)\cdot N^R.\end{aligned} \tag{16}$$

The bound on the total error (i.e., $|z^R-\Re(wx)|$) is obtained by adding the bounds (14), (15), and (16), i.e.,

$$\begin{aligned}&(u+3u^2+u^3)\cdot n^R\\ &+(15u^2+38u^3+39u^4+22u^5+7u^6+u^7)\cdot N^R.\end{aligned} \tag{17}$$

This gives Lemma 1. □

Let us now prove Theorem 1.

Proof. Lemma 1 gives a bound on the componentwise *absolute* error of Algorithm 3. However, it does not allow one to infer a useful bound on the componentwise *relative* error, since $|N^R/n^R|$ and $|N^I/n^I|$ can be arbitrarily large. However, the normwise relative error is always small, as we are going to see.

The square of the normwise relative error η is

$$\eta^2=\frac{(z^R-\Re(\omega x))^2+(z^I-\Im(\omega x))^2}{\left(\Re(\omega x)\right)^2+\left(\Im(\omega x)\right)^2}.$$

From (3), we have

$$\begin{aligned}&(z^R-\Re(\omega x))^2+(z^I-\Im(\omega x))^2\\ &\leqslant \alpha^2\left(\left(n^R\right)^2+\left(n^I\right)^2\right)+2\alpha\beta\left(n^RN^R+n^IN^I\right)\\ &+\beta^2\left(\left(N^R\right)^2+\left(N^I\right)^2\right).\end{aligned}$$

We also have

$$\begin{aligned}&\left(\Re(\omega x)\right)^2+\left(\Im(\omega x)\right)^2=\left(n^R\right)^2+\left(n^I\right)^2\\ &=\left(\omega^Rx^R\right)^2+\left(\omega^Ix^I\right)^2+\left(\omega^Rx^I\right)^2+\left(\omega^Ix^R\right)^2.\end{aligned}$$

Hence,

$$\begin{aligned}\eta^2 &\leqslant \alpha^2+2\alpha\beta\frac{n^RN^R+n^IN^I}{\left(n^R\right)^2+\left(n^I\right)^2}\\ &+\beta^2\frac{\left(N^R\right)^2+\left(N^I\right)^2}{\left(n^R\right)^2+\left(n^I\right)^2}.\end{aligned} \tag{18}$$

We can notice that

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

and

$$\begin{aligned}&\frac{\left(N^R\right)^2+\left(N^I\right)^2}{\left(n^R\right)^2+\left(n^I\right)^2}\\ &=1+\frac{4\cdot\left|\omega^Rx^R\omega^Ix^I\right|}{\left(\omega^Rx^R\right)^2+\left(\omega^Ix^I\right)^2+\left(\omega^Rx^I\right)^2+\left(\omega^Ix^R\right)^2}.\end{aligned} \tag{19}$$

In the denominator of the right-hand part of (19), the sum of the two terms $\left(\omega^Rx^R\right)^2$ and $\left(\omega^Ix^I\right)^2$ is larger than or equal to $2\cdot\left|\omega^Rx^R\omega^Ix^I\right|$, since

$$\begin{aligned}&\left(\omega^Rx^R\right)^2+\left(\omega^Ix^I\right)^2-2\cdot\left|\omega^Rx^R\omega^Ix^I\right|\\ &=\left(\left|\omega^Rx^R\right|-\left|\omega^Ix^I\right|\right)^2\geqslant 0.\end{aligned}$$

The same holds for the sum of the two terms $\left(\omega^Rx^I\right)^2$ and $\left(\omega^Ix^R\right)^2$. This immediately gives

$$1+\frac{4\cdot\left|\omega^Rx^R\omega^Ix^I\right|}{\left(\omega^Rx^R\right)^2+\left(\omega^Ix^I\right)^2+\left(\omega^Rx^I\right)^2+\left(\omega^Ix^R\right)^2}\leqslant 2.$$

Combined with (18), this gives

$$\eta^2\leqslant\alpha^2+4\alpha\beta+2\beta^2,$$

from which we obtain

$$\begin{aligned}\eta^2 &\leqslant u^2+66u^3+793u^4+2958u^5+5937u^6\\ &+7696u^7+6982u^8+4596u^9+2216u^{10}\\ &+772u^{11}+186u^{12}+28u^{13}+2u^{14}.\end{aligned} \tag{20}$$

This bound can be rewritten

$$\eta^2\leqslant\left(u+33u^2\right)^2-(296-\nu)\cdot u^4$$

with

$$\begin{aligned}\nu &= 2958u+5937u^2+7696u^3+6982u^4+4596u^5\\ &+2216u^6+772u^7+186u^8+28u^9+2u^{10}.\end{aligned}$$

One easily checks that for $u\leqslant 1/16$ (i.e., $p\geqslant 4$), $\nu<296$ (since it is an increasing function of u, it suffices to check its value for $u=1/16$). Theorem 1 immediately follows from that.

□

III. Two special cases

A. Obtaining the real and imaginary parts of the product as double-word numbers

As explained above, one may wish to obtain the real and imaginary parts of the product $\omega\cdot x$ as double-word numbers, by replacing the floating-point addition $z^R=\mathrm{RN}(v_h^R+\gamma_\ell^R)$ of line 9 of Algorithm 3 by a call to $\mathrm{2Sum}(v_h^R,\gamma_\ell^R)$, and by replacing the floating-point addition $z^I=\mathrm{RN}(v_h^I+\gamma_\ell^I)$ of line 18 by a call to $\mathrm{2Sum}(v_h^I,\gamma_\ell^I)$. The resulting componentwise error bound is obtained by adding (14) and (15), so that the values α and β of Lemma 1 must be replaced by

$$\alpha'=u^2$$

and

$$\beta'=10u^2+21u^3+16u^4+6u^5+u^6.$$

With these new values, the square η'^2 of the normwise error now satisfies

$$\begin{aligned}\eta'^2 \quad \leqslant \quad & 241u^4 + 924u^5 + 1586u^6 \\ & + 1608u^7 + 1060u^8 + 468u^9 \\ & + 136u^{10} + 24u^{11} + 2u^{12},\end{aligned} \tag{21}$$

so that the normwise relative error becomes less than $\sqrt{241} \cdot u^2 + \mathcal{O}(u^3) \approx 15.53u^2 + \mathcal{O}(u^3)$.

That variant is of interest if one wishes to accurately evaluate the product

$$z_1 \times z_2 \times \cdots \times z_n$$

of n complex numbers. One can evaluate that product iteratively, keeping the real and imaginary parts of the partial product of all already considered terms as double-word numbers, and just using the unmodified Algorithm 3 (i.e., a simple FP addition and not a 2Sum at lines 9 and 18) for the last multiplication. Still assuming that overflow or underflow do not occur (which may become unlikely if n is very large and the z_i are arbitrary numbers), the total relative error is bounded by

$$(1 + \eta')^{n-2} \cdot (1 + \eta) - 1.$$

For instance, assuming binary64 arithmetic ($u = 2^{-53}$), one can multiply 1000 numbers and still have a normwise relative error bounded by $1.000000000001724u$.

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

If ω^I and ω^R are floating-point numbers (i.e., if $\omega_\ell^I = \omega_\ell^R = 0$), Algorithm 3 becomes simpler, and we obtain Algorithm 4 below. Each separate part (computation of the real part, lines 1 to 6, or computation of the imaginary part, lines 7 to 12) in Algorithm 4 is similar to Cornea et al's algorithm for $ab + cd$ presented in [2] (with an addition replaced here by a call to 2Sum), and to Algorithm 5.3 in [15] (with an inversion in the order of summation of P_ℓ^R, Q_ℓ^R, and v_ℓ^R for the real part, and of P_ℓ^I, Q_ℓ^I, and v_ℓ^I for the imaginary part).

ALGORITHM 4: Accurate complex multiplication $\omega \cdot x$, where the real and imaginary parts of ω and the real and imaginary parts of x are FP numbers, derived from Algorithm 3.

1: $(P_h^R, P_\ell^R) \leftarrow$ Fast2Mult(ω^I, x^I)
2: $(Q_h^R, Q_\ell^R) \leftarrow$ Fast2Mult(ω^R, x^R)
3: $s_\ell^R \leftarrow$ RN$(Q_\ell^R - P_\ell^R)$
4: $(v_h^R, v_\ell^R) \leftarrow$ 2Sum$(Q_h^R, -P_h^R)$
5: $\gamma_\ell^R \leftarrow$ RN$(v_\ell^R + s_\ell^R)$
6: **return** $z^R =$ RN$(v_h^R + \gamma_\ell^R)$ (real part)
7: $(P_h^I, P_\ell^I) \leftarrow$ Fast2Mult(ω^I, x^R)
8: $(Q_h^I, Q_\ell^I) \leftarrow$ Fast2Mult(ω^R, x^I)
9: $s_\ell^I \leftarrow$ RN$(Q_\ell^I + P_\ell^I)$
10: $(v_h^I, v_\ell^I) \leftarrow$ 2Sum(Q_h^I, P_h^I)
11: $\gamma_\ell^I \leftarrow$ RN$(v_\ell^I + s_\ell^I)$
12: **return** $z^I =$ RN$(v_h^I + \gamma_\ell^I)$ (imaginary part)

Of course the error bounds given by (20) and Theorem 1 still apply. However, one can redo the calculations taking into account the zero terms, and obtain new error bounds with smaller higher-order terms, more precisely, we find

$$\begin{aligned}\eta^2 \quad \leqslant \quad & u^2 + 38u^3 + 299u^4 + 782u^5 + 1025u^6 \\ & + 768u^7 + 336u^8 + 80u^9 + 8u^{10}.\end{aligned} \tag{22}$$

Therefore

$$\eta^2 \leqslant \left(u + 19u^2\right)^2 - (62 - \nu) \cdot u^4$$

with

$$\nu = 782u + 1025u^2 + 768u^3 + 336u^4 + 80u^5 + 8u^6.$$

This gives

Theorem 2. *As soon as $p \geqslant 4$, the normwise relative error η of Algorithm 4 satisfies*

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

IV. Implementation and experiments

Algorithm 3, implemented in binary64 arithmetic (i.e., $p = 53$ and $u = 2^{-53}$), was compared with other solutions, using a loop over N random inputs, itself inside another loop doing K iterations. The goal of the external loop is to get precise timings without having to choose a large value of N, with input data that would not fit in the cache: we do not want to include memory transfers in the timings. For each test, we chose $(N, K) = (1024, 65536)$, $(2048, 32768)$ and $(4096, 16384)$.

The other considered solutions were: use of the naive formula (1) in binary64 arithmetic; use of (1) in binary128 (a.k.a. "quad precision") arithmetic; use of GNU MPFR [3] with precision ranging from 53 to 106 bits either with fused multiplications/subtractions fmma/fmms (thus implementing the formulas, correctly rounded) or with separate additions, subtractions and multiplications.

The tests were run on two machines with a hardware FMA:

- an x86_64 machine with Intel Xeon E5-2609 v3 CPUs, under Linux (Debian/unstable), with GCC 8.2.0 and a Clang 8 preversion, using `-march=native`;
- a ppc64le machine with POWER9 CPUs, under Linux (CentOS 7), with GCC 8.2.1, using `-mcpu=power9`.

The following optimization options were used: `-O3` and `-O2`. With GCC, `-O3 -fno-tree-slp-vectorize` was also used in order to avoid a loss of performance with some vectorized codes. In all the cases, `-static` was used to avoid the overhead due to function calls to dynamic libraries.

The tests were run on several random data sets, giving a range of timings and a range of ratios. The smallest 10 % values and largest 10 % values have been excluded to take into account inaccuracies in the timings.

We checked that the various timings were globally consistent, in particular between the three chosen parameters for (N, K), and rejected some anomalies manually: for Algorithm 3, $(N, K) = (2048, 32768)$ and $(4096, 16384)$ with

TABLE I

SUMMARY OF THE TIMINGS ON AN X86_64 MACHINE (IN SECONDS, FOR $NK = 2^{26}$ OPERATIONS). "A3" STANDS FOR "ALGORITHM 3" (IN BINARY64 ARITHMETIC), "SW" CORRESPONDS TO THE NAIVE FORMULA (1) IN BINARY64 ARITHMETIC, "DW" CORRESPONDS TO (1) IN BINARY128 ARITHMETIC, "CR" IS GNU MPFR WITH FUSED MULTIPLICATIONS/SUBTRACTIONS FMMA/FMMS, AND "NA" IS GNU MPFR WITH SEPARATE ADDITIONS, SUBTRACTIONS AND MULTIPLICATIONS.

		minimums			maximums		
$N \rightarrow$		1024	2048	4096	1024	2048	4096
`gcc -O3`	a3	0.94	0.97	0.97	0.96	1.02	1.02
	sw	0.61	0.62	0.62	0.61	0.62	0.62
	dw	21.02	21.17	21.20	21.18	21.25	21.28
	cr	15.76	15.99	16.08	21.48	21.63	21.66
	na	12.46	12.88	12.99	23.16	23.23	23.22
`gcc -O3 -f...`	a3	0.92	0.97	0.97	0.95	1.02	1.02
	sw	0.61	0.61	0.62	0.61	0.62	0.62
	dw	21.32	21.44	21.46	21.43	21.53	21.54
	cr	15.87	16.11	16.16	21.54	21.73	21.78
	na	12.59	13.01	13.12	22.72	22.85	22.80
`gcc -O2`	a3	0.91	0.97	0.97	0.95	1.02	1.02
	sw	0.61	0.62	0.62	0.61	0.62	0.62
	dw	20.90	21.03	21.08	21.01	21.10	21.13
	cr	15.93	16.17	16.26	21.57	21.70	21.75
	na	12.31	12.74	12.85	23.11	23.20	23.18
`clang -O3`	a3	0.86	1.09	1.10	0.96	1.15	1.15
	sw	0.39	0.61	0.63	0.47	0.65	0.66
	dw	21.65	21.77	21.81	21.74	21.87	21.88
	cr	16.00	16.24	16.32	21.46	21.69	21.71
	na	12.24	12.63	12.72	22.91	22.94	22.97
`clang -O2`	a3	0.88	1.08	1.10	0.96	1.14	1.15
	sw	0.40	0.61	0.63	0.48	0.65	0.66
	dw	21.33	21.45	21.50	21.49	21.57	21.59
	cr	15.38	15.62	15.70	21.62	21.79	21.87
	na	12.15	12.54	12.65	23.15	23.21	23.21

Clang gave running times larger than those obtained with GCC, affecting the comparison with GNU MPFR. The timings are given in Table I (x86_64) and Table II (Power9). Note that reading the inputs is included in the timings (thus the ratios will be closer to 1 than one could expect), but these inputs are already in the right format for each implementation.

As a summary from the tables:

- **Implementation based on the naive formula (1) in binary64** (inlined code): It is about two times as fast as our implementation of Algorithm 3, but it is significantly less accurate.
- **Implementation based on the naive formula in binary128,** using the `__float128` C type (inlined code): On the x86_64 platform, it is from 19 to 25 times as slow as our implementation of Algorithm 3, the reason being that this format is implemented in software. The case of the POWER9 platform is particularly interesting as it has binary128 support in hardware. Here, the implementation is about 2.3 times as slow. This shows that even though one has hardware support for binary128, there is still interest in algorithms using a mix of binary64 and double-binary64.
- **Implementation based on GNU MPFR,** using precisions from 53 (corresponding to binary64) to 106 (roughly corresponding to double-binary64). Both codes based on fmma/fmms (thus implementing the formulas, correctly rounded) and based on separate additions, subtraction and multiplication operations were tested. This is from 11 to 26 times as slow as our implementation of Algorithm 3 on x86_64, and from 17 to 31 times as slow on POWER9.

TABLE II

SUMMARY OF THE TIMINGS ON A POWER9 MACHINE (IN SECONDS, FOR $NK = 2^{26}$ OPERATIONS). "A3" , "SW", "DW", "CR", AND "NA" HAVE THE SAME MEANING AS IN TABLE I.

		minimums			maximums		
$N \rightarrow$		1024	2048	4096	1024	2048	4096
`gcc -O3`	a3	0.97	0.97	0.97	0.99	0.99	1.00
	sw	0.47	0.47	0.51	0.48	0.48	0.52
	dw	2.22	2.22	2.22	2.24	2.24	2.24
	cr	19.44	19.56	19.62	23.94	24.07	24.06
	na	16.41	16.60	16.66	30.07	30.34	30.63
`gcc -O3 -f...`	a3	0.97	0.97	0.97	0.98	0.99	1.00
	sw	0.47	0.47	0.51	0.48	0.48	0.52
	dw	2.22	2.22	2.22	2.24	2.24	2.24
	cr	19.45	19.59	19.61	24.11	24.08	24.07
	na	16.42	16.59	16.66	30.06	30.39	30.44
`gcc -O2`	a3	0.98	0.98	0.98	0.99	1.01	1.01
	sw	0.47	0.47	0.51	0.47	0.47	0.51
	dw	2.22	2.22	2.22	2.24	2.24	2.24
	cr	19.50	19.66	19.68	24.14	24.11	24.05
	na	16.36	16.58	16.63	30.29	30.29	30.49

Algorithm 3 has also been tested on random inputs to search for large normwise relative errors. For binary32, the input values (in ISO C99 / IEEE 754-2008 hexadecimal format) and the corresponding largest error found until now are

$$\begin{aligned} \omega^R &= \text{0x1.b3fdfcp}-1 + \text{0x1.77f658p}-26 \\ \omega^I &= \text{0x1.53c918p}-28 + -\text{0x1.ca53e6p}-53 \\ x^R &= \text{0x1.2ca11ep}-1 \\ x^I &= \text{0x1.9c641ap}-18 \\ \eta &\simeq 0.99999933401292962563\,u \end{aligned}$$

and for binary64, we have obtained

$$\begin{aligned} \omega^R &= \text{0x1.d1ef9ea4aa013p}-1 + \text{0x1.ae88ba2a277ep}-56 \\ \omega^I &= \text{0x1.f5c28321df365p}-81 + \text{0x1.c4c3e7b506d06p}-135 \\ x^R &= \text{0x1.194f298b4d152p}-1 \\ x^I &= \text{0x1.5c1fdca444f7cp}-14 \\ \eta &\simeq 0.99999900913907117123\,u. \end{aligned}$$

This corroborates the bound given by Theorem 1.

CONCLUSION

We have given algorithms for complex multiplication in floating-point arithmetic, that either return the real and imag-

inary parts of the product as floating-point numbers with a normwise relative error bound close to the best one that one can guarantee, namely $u/(1+u)$, or as double-word numbers. Our implementation is only twice as slow as a significantly less accurate naive implementation. It is much faster than an implementation based on binary128 or multiple-precision software.

ACKNOWLEDGMENT

The authors thank the anonymous reviewers for their very helpful comments. This work has been partly supported by the FastRelax project of the French Agence Nationale de la Recherche (ANR-14-CE25-0018-01).

REFERENCES

[1] R. P. Brent, C. Percival, and P. Zimmermann. Error bounds on complex floating-point multiplication. *Mathematics of Computation*, 76:1469–1481, 2007.

[2] M. Cornea, J. Harrison, and P. T. P. Tang. *Scientific Computing on Itanium®-based Systems*. Intel Press, Hillsboro, OR, 2002.

[3] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR: A multiple-precision binary floating-point library with correct rounding. *ACM Transactions on Mathematical Software*, 33(2), 2007. 15 pages. Available at https://www.mpfr.org/.

[4] Y. Hida, X. S. Li, and D. H. Bailey. Algorithms for quad-double precision floating-point arithmetic. In *15th IEEE Symposium on Computer Arithmetic (ARITH-15)*, pages 155–162, June 2001.

[5] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, PA, 2nd edition, 2002.

[6] IEEE Computer Society. *IEEE Standard for Floating-Point Arithmetic*. IEEE Standard 754-2008, August 2008. Available at https://doi.org/10.1109/IEEESTD.2008.4610935.

[7] C.-P. Jeannerod, P. Kornerup, N. Louvet, and J.-M. Muller. Error bounds on complex floating-point multiplication with an FMA. *Mathematics of Computation*, 86(304):881–898, 2017.

[8] C.-P. Jeannerod, N. Louvet, and J.-M. Muller. Further analysis of Kahan's algorithm for the accurate computation of 2×2 determinants. *Mathematics of Computation*, 82(284):2245–2264, 2013.

[9] M. Joldeş, J.-M. Muller, and V. Popescu. Tight and rigourous error bounds for basic building blocks of double-word arithmetic. *ACM Transactions on Mathematical Software*, 44(2), 2017.

[10] W. Kahan. Lecture notes on the status of IEEE-754. Available at https://people.eecs.berkeley.edu/~wkahan/ieee754status/IEEE754.PDF, 1997.

[11] D. E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, Reading, MA, 3rd edition, 1998.

[12] O. Møller. Quasi double-precision in floating-point addition. *BIT*, 5:37–50, 1965.

[13] J.-M. Muller, N. Brunie, F. de Dinechin, C.-P. Jeannerod, M. Joldes, V. Lefèvre, G. Melquiond, N. Revol, and S. Torres. *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2018. ACM G.1.0; G.1.2; G.4; B.2.0; B.2.4; F.2.1., ISBN 978-3-319-76525-9.

[14] Y. Nievergelt. Scalar fused multiply-add instructions produce floating-point matrix arithmetic provably accurate to the penultimate digit. *ACM Transactions on Mathematical Software*, 29(1):27–48, 2003.

[15] T. Ogita, S. M. Rump, and S. Oishi. Accurate sum and dot product. *SIAM Journal on Scientific Computing*, 26(6):1955–1988, 2005.