Tight and rigorous error bounds for basic building blocks of double-word arithmetic

Mioara Joldes, Jean-Michel Muller, Valentina Popescu

JNCF - January 2017

CAMPARY
CudA Multiple Precision ARithmetic librarY

LIP LAAS-CNRS

Floating point arithmetics

A real number X is approximated in a machine by a rational

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

– M_x is the *significand*, a signed integer number of p digits in radix 2 s.t. $2^{p-1} \le |M_x| \le 2^p - 1$;

– e_x is the *exponent*, a signed integer ($e_{min} \le e_x \le e_{max}$).

Floating point arithmetics

A real number X is approximated in a machine by a rational

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

– M_x is the *significand*, a signed integer number of p digits in radix 2 s.t. $2^{p-1} \leq |M_x| \leq 2^p - 1$;
– e_x is the *exponent*, a signed integer ($e_{min} \leq e_x \leq e_{max}$).

IEEE 754-2008: Most common formats

- Single ($binary32$) precision format:

1	8	23
s	e	m

- Double ($binary64$) precision format:

1	11	52
s	e	m

Floating point arithmetics

A real number X is approximated in a machine by a rational

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

– M_x is the *significand*, a signed integer number of p digits in radix 2 s.t. $2^{p-1} \leq |M_x| \leq 2^p - 1$;
– e_x is the *exponent*, a signed integer ($e_{min} \leq e_x \leq e_{max}$).

IEEE 754-2008: Most common formats

- Single ($binary32$) precision format:

1	8	23
s	e	m

- Double ($binary64$) precision format:

1	11	52
s	e	m

Rounding modes

- 4 rounding modes: RD, RU, RZ, RN;
- Correct rounding for: $+, -, \times, \div, \sqrt{}$;
- Portability, determinism.

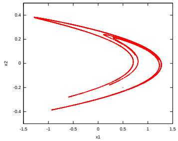
→ Computations with increased (multiple) precision in numerical applications.

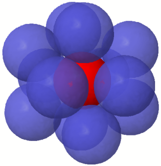
Chaotic dynamical systems:

- bifurcation analysis;
- compute periodic orbits (e.g., Hénon map, Lorenz attractor);
- celestial mechanics (e.g., stability of the solar system).

Experimental mathematics:

- computational geometry (e.g., kissing numbers);
- polynomial optimization etc.





What is a double-word number?

Definition:

A *double-word* number x is the unevaluated sum $x_h + x_\ell$ of two floating-point numbers x_h and x_ℓ such that

$$x_h = \mathsf{RN}(x).$$

What is a double-word number?

Definition:

A *double-word* number x is the unevaluated sum $x_h + x_\ell$ of two floating-point numbers x_h and x_ℓ such that

$$x_h = \mathsf{RN}\,(x).$$

$\rightarrow$ Called *double-double* when using the *binary64* standard format.

Example: π in double-double

$$p_h = 11.001001000011111101101010100010001000010110100011000_2,$$

and

$$p_\ell = 1.0001101001100010011000110011000101000101110000000111_2 \times 2^{-53};$$

$p_h + p_\ell \leftrightarrow$ 107 bit FP approx.

Remark

$\longrightarrow$ Not the same as IEEE 754-2008 standard's *binary128/quadruple*-precision.

Double-word (using *binary64/double*-precision):

***Binary128/quadruple*-precision:**

Remark

$\longrightarrow$ Not the same as IEEE 754-2008 standard's *binary128/quadruple*-precision.

Double-word (using *binary64/double*-precision):

- "wobbling precision" ≥ 107 bits of precision;

***Binary128/quadruple*-precision:**

- 113 bits of precision;

Remark

$\longrightarrow$ Not the same as IEEE 754-2008 standard's *binary128/quadruple*-precision.

Double-word (using *binary64/double*-precision):

- "wobbling precision" ≥ 107 bits of precision;
- exponent range limited by *binary64* (11 bits) i.e. -1022 to 1023;

***Binary128/quadruple*-precision:**

- 113 bits of precision;
- larger exponent range (15 bits): -16382 to 16383;

Remark

$\longrightarrow$ Not the same as IEEE 754-2008 standard's *binary128/quadruple*-precision.

Double-word (using *binary64/double*-precision):

- "wobbling precision" ≥ 107 bits of precision;
- exponent range limited by *binary64* (11 bits) i.e. -1022 to 1023;
- lack of clearly defined rounding modes;

***Binary128/quadruple*-precision:**

- 113 bits of precision;
- larger exponent range (15 bits): -16382 to 16383;
- defined with all rounding modes

Remark

$\longrightarrow$ Not the same as IEEE 754-2008 standard's *binary128/quadruple*-precision.

Double-word (using *binary64/double*-precision):

- "wobbling precision" ≥ 107 bits of precision;
- exponent range limited by *binary64* $(11$ bits) i.e. -1022 to 1023;
- lack of clearly defined rounding modes;
- manipulated using error-free transforms (next slide).

***Binary128/quadruple*-precision:**

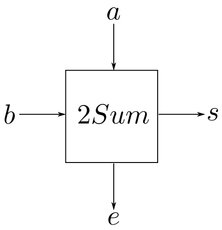
- 113 bits of precision;
- larger exponent range $(15$ bits): -16382 to 16383;
- defined with all rounding modes
- not implemented in hardware on widely available processors.

Theorem 1 (*2Sum* algorithm)

Let a and b be FP numbers. Algorithm 2Sum computes two FP numbers s and e that satisfy the following:

- $s + e = a + b$ *exactly;*
- $s = RN(a + b)$.

(RN stands for performing the operation in rounding to nearest rounding mode.)



Algorithm 1 (*2Sum* (a, b))

$s \leftarrow RN(a + b)$
$t \leftarrow RN(s - b)$
$e \leftarrow RN(RN(a - t) + RN(b - RN(s - t)))$
return (s, e)

$\longrightarrow$ 6 FP operations (proved to be optimal unless we have information on the ordering of $|a|$ and $|b|$)

Theorem 2 (*Fast2Sum* algorithm)

Let a and b be FP numbers that satisfy $e_a \geq e_b (|a| \geq |b|)$. Algorithm Fast2Sum computes two FP numbers s and e that satisfy the following:

- $s + e = a + b$ *exactly;*
- $s = RN(a + b)$.

Algorithm 2 (*Fast2Sum* (a, b))

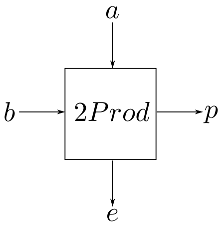
$s \leftarrow RN(a + b)$
$z \leftarrow RN(s - a)$
$e \leftarrow RN(b - z)$
return (s, e)

$\longrightarrow$ 3 FP operations

Theorem 3 (*2ProdFMA* algorithm)

Let a and b be FP numbers, $e_a + e_b \geq e_{min} + p - 1$. Algorithm 2ProdFMA computes two FP numbers p and e that satisfy the following:

- $p + e = a \cdot b$ *exactly;*
- $p = RN(a \cdot b)$.



Algorithm 3 (*2ProdFMA* (a, b))

$p \leftarrow RN(a \cdot b)$
$e \leftarrow fma(a, b, -p)$
return (p, e)

$\longrightarrow$ 2 FP operations

$\longrightarrow$ hardware-implemented FMA available in latest processors.

Previous work:

- concept introduced by Dekker [DEK71] together with some algorithms for basic operations;
- Linnainmaa [LIN81] suggested similar algorithms assuming an underlying wider format;
- library written by Briggs [BRI98] - that is no longer maintained;
- QD library written by Bailey [Li.et.al02].

Previous work:

- concept introduced by Dekker [DEK71] together with some algorithms for basic operations;
- Linnainmaa [LIN81] suggested similar algorithms assuming an underlying wider format;
- library written by Briggs [BRI98] - that is no longer maintained;
- QD library written by Bailey [Li.et.al02].

Problems:

1. most algorithms come without correctness proof and error bound;
2. some error bounds published without a proof;
3. differences between published and implemented algorithms.

Previous work:

- concept introduced by Dekker [DEK71] together with some algorithms for basic operations;
- Linnainmaa [LIN81] suggested similar algorithms assuming an underlying wider format;
- library written by Briggs [BRI98] - that is no longer maintained;
- QD library written by Bailey [Li.et.al02].

Problems:

1. most algorithms come without correctness proof and error bound;
2. some error bounds published without a proof;
3. differences between published and implemented algorithms.

$\longrightarrow$ Strong need to "clean up" the existing literature.

Previous work:

- concept introduced by Dekker [DEK71] together with some algorithms for basic operations;
- Linnainmaa [LIN81] suggested similar algorithms assuming an underlying wider format;
- library written by Briggs [BRI98] - that is no longer maintained;
- QD library written by Bailey [Li.et.al02].

Problems:

1. most algorithms come without correctness proof and error bound;
2. some error bounds published without a proof;
3. differences between published and implemented algorithms.

$\longrightarrow$ Strong need to "clean up" the existing literature.

Notation:

- p represents the precision of the underlying FP format;
- $\text{ulp}(x) = 2^{\lfloor \log_2 |x| \rfloor - p + 1}$, for $x \neq 0$;
- $u = 2^{-p} = \frac{1}{2}\,\text{ulp}(1)$ denotes the roundoff error unit.

Addition: **DWPlusFP**(x_h, x_ℓ, y)

Algorithm 4

1: $(s_h, s_\ell) \leftarrow$ *2Sum*(x_h, y)
2: $v \leftarrow$ *RN*$(x_\ell + s_\ell)$
3: $(z_h, z_\ell) \leftarrow$ *Fast2Sum*(s_h, v)
4: **return** (z_h, z_ℓ)

– implemented in the QD library;
– no previous error bound published;
– relative error bounded by

$$\frac{2 \cdot 2^{-2p}}{1 - 2 \cdot 2^{-p}} = 2 \cdot 2^{-2p} + 4 \cdot 2^{-3p} + 8 \cdot 2^{-4p} + \cdots,$$

which is less than $2 \cdot 2^{-2p} + 5 \cdot 2^{-3p}$ as soon as $p \geq 4$;

Addition: **DWPlusFP**(x_h, x_ℓ, y)

Algorithm 4

1: $(s_h, s_\ell) \leftarrow$ *2Sum*(x_h, y)
2: $v \leftarrow$ *RN*$(x_\ell + s_\ell)$
3: $(z_h, z_\ell) \leftarrow$ *Fast2Sum*(s_h, v)
4: **return** (z_h, z_ℓ)

– implemented in the QD library;
– no previous error bound published;
– relative error bounded by

$$\frac{2 \cdot 2^{-2p}}{1 - 2 \cdot 2^{-p}} = 2 \cdot 2^{-2p} + 4 \cdot 2^{-3p} + 8 \cdot 2^{-4p} + \cdots,$$

which is less than $2 \cdot 2^{-2p} + 5 \cdot 2^{-3p}$ as soon as $p \geq 4$;

Asymptotically optimal bound

Let $x_h = 1$, $x_\ell = (2^p - 1) \cdot 2^{-2p}$, and $y = -\frac{1}{2}(1 - 2^{-p})$. Then:
– $z_h + z_\ell = \frac{1}{2} + 3 \cdot 2^{-p-1}$ and;
– $x + y = \frac{1}{2} + 3 \cdot 2^{-p-1} - 2^{-2p}$;
– relative error

$$\frac{2 \cdot 2^{-2p}}{1 + 3 \cdot 2^{-p} - 2 \cdot 2^{-2p}} \approx 2 \cdot 2^{-2p} - 6 \cdot 2^{-3p}.$$

Sketch of the proof

Lemma 4 (Sterbenz Lemma)

Let a and b be two positive FP numbers. If

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

then $a - b$ is a floating-point number, so that $RN(a - b) = a - b$.

Lemma 5

Let a and b be FP numbers, and let $s = RN(a + b)$. If $s \neq 0$ then

$$s \geq \max\left\{\frac{1}{2}\, ulp(a), \frac{1}{2}\, ulp(b)\right\}.$$

Sketch of the proof

W.l.o.g. $|x_h| \geq |y|$; x_h positive; $1 \leq x_h \leq 2 - 2u$.

Sketch of the proof

W.l.o.g. $|x_h| \geq |y|$; x_h positive; $1 \leq x_h \leq 2 - 2u$.

[Case1:] **If $-x_h \leq y \leq -x_h/2$:** from Sterbenz Lemma $s_h = x_h + y$ and $s_\ell = 0$.
From Lemma 5 $|s_h| \geq \frac{1}{2}\,\mathrm{ulp}\,(x_h)$, so $|s_h| \geq |x_\ell|$.
Hence we can use Algorithm Fast2Sum at line 3 of the algorithm, so that $z_h + z_\ell = s_h + v = x + y$ exactly.

Sketch of the proof

W.l.o.g. $|x_h| \geq |y|$; x_h positive; $1 \leq x_h \leq 2 - 2u$.

[Case1:] **If $-x_h \leq y \leq -x_h/2$:** from Sterbenz Lemma $s_h = x_h + y$ and $s_\ell = 0$.
From Lemma 5 $|s_h| \geq \frac{1}{2}\,\text{ulp}\,(x_h)$, so $|s_h| \geq |x_\ell|$.
Hence we can use Algorithm Fast2Sum at line 3 of the algorithm, so that $z_h + z_\ell = s_h + v = x + y$ exactly.

[Case2:] **If $-x_h/2 < y \leq x_h$,** then $\frac{1}{2} \leq \frac{x_h}{2} < x_h + y \leq 2x_h$, so that $s_h \geq 1/2$.
One can prove that $|x_\ell + y_\ell| \leq 3u$ (two cases), so $|v| \leq 3u$, s.t. $s_h > |v|$: we can use Algorithm Fast2Sum at line 3 of the algorithm.

W.l.o.g. $|x_h| \geq |y|$; x_h positive; $1 \leq x_h \leq 2 - 2u$.

[Case1:] **If $-x_h \leq y \leq -x_h/2$:** from Sterbenz Lemma $s_h = x_h + y$ and $s_\ell = 0$.
From Lemma 5 $|s_h| \geq \frac{1}{2}\,\mathrm{ulp}\,(x_h)$, so $|s_h| \geq |x_\ell|$.
Hence we can use Algorithm Fast2Sum at line 3 of the algorithm, so that $z_h + z_\ell = s_h + v = x + y$ exactly.

[Case2:] **If $-x_h/2 < y \leq x_h$,** then $\frac{1}{2} \leq \frac{x_h}{2} < x_h + y \leq 2x_h$, so that $s_h \geq 1/2$.
One can prove that $|x_\ell + y_\ell| \leq 3u$ (two cases), so $|v| \leq 3u$, s.t. $s_h > |v|$: we can use Algorithm Fast2Sum at line 3 of the algorithm.

[Case2a:] **If $x_h + y \leq 2$** then $|s_\ell| \leq u$, so that $|x_\ell + s_\ell| \leq 2u$, hence, $v = x_\ell + s_\ell + \varepsilon$, with $|\varepsilon| \leq u^2$.
Therefore $z_h + z_\ell = s_h + v = x + y + \varepsilon$ and the relative error

$$\frac{\varepsilon}{|x+y|} \leq \frac{\varepsilon}{\frac{1}{2} - u} \leq \frac{2u^2}{1-2u}.$$

Addition: **AccurateDWPlusDW**$(x_h, x_\ell, y_h, y_\ell)$

Algorithm 5

1: $(s_h, s_\ell) \leftarrow$ *2Sum*(x_h, y_h)
2: $(t_h, t_\ell) \leftarrow$ *2Sum*(x_ℓ, y_ℓ)
3: $c \leftarrow$ *RN*$(s_\ell + t_h)$
4: $(v_h, v_\ell) \leftarrow$ *Fast2Sum*(s_h, c)
5: $w \leftarrow$ *RN*$(t_\ell + v_\ell)$
6: $(z_h, z_\ell) \leftarrow$ *Fast2Sum*(v_h, w)
7: **return** (z_h, z_ℓ)

– previously published relative error bound [Li.et.al02]: $2 \cdot 2^{-2p}$;

Addition: **AccurateDWPlusDW**$(x_h, x_\ell, y_h, y_\ell)$

Algorithm 5

1: $(s_h, s_\ell) \leftarrow$ *2Sum*(x_h, y_h)
2: $(t_h, t_\ell) \leftarrow$ *2Sum*(x_ℓ, y_ℓ)
3: $c \leftarrow$ *RN*$(s_\ell + t_h)$
4: $(v_h, v_\ell) \leftarrow$ *Fast2Sum*(s_h, c)
5: $w \leftarrow$ *RN*$(t_\ell + v_\ell)$
6: $(z_h, z_\ell) \leftarrow$ *Fast2Sum*(v_h, w)
7: **return** (z_h, z_ℓ)

– previously published relative error bound [Li.et.al02]: $2 \cdot 2^{-2p}$;
– FALSE, showed by the counterexample:

$$x_h = 2^p - 1,\ x_\ell = -(2^p - 1) \cdot 2^{-p-1},$$
$$y_h = -(2^p - 5)/2,\ y_\ell = -(2^p - 1) \cdot 2^{-p-3},$$

which leads to a relative error asymptotically equivalent to 2.25×2^{-2p};

Addition: **AccurateDWPlusDW**$(x_h, x_\ell, y_h, y_\ell)$

Algorithm 5

1: $(s_h, s_\ell) \leftarrow$ *2Sum*(x_h, y_h)
2: $(t_h, t_\ell) \leftarrow$ *2Sum*(x_ℓ, y_ℓ)
3: $c \leftarrow$ *RN*$(s_\ell + t_h)$
4: $(v_h, v_\ell) \leftarrow$ *Fast2Sum*(s_h, c)
5: $w \leftarrow$ *RN*$(t_\ell + v_\ell)$
6: $(z_h, z_\ell) \leftarrow$ *Fast2Sum*(v_h, w)
7: **return** (z_h, z_ℓ)

– previously published relative error bound [Li.et.al02]: $2 \cdot 2^{-2p}$;
– FALSE, showed by the counterexample:

$$x_h = 2^p - 1,\ x_\ell = -(2^p - 1) \cdot 2^{-p-1},$$
$$y_h = -(2^p - 5)/2,\ y_\ell = -(2^p - 1) \cdot 2^{-p-3},$$

which leads to a relative error asymptotically equivalent to 2.25×2^{-2p};
– rigorous proven error bound less than

$$3 \cdot 2^{-2p} + 13 \cdot 2^{-3p},$$

as soon as $p \geq 6$;
– sloppy version available, but less accurate.

Multiplication: **DWTimesFP**(x_h, x_ℓ, y)

Algorithm 6

1: $(c_h, c_{\ell 1}) \leftarrow$ *Fast2Mult*(x_h, y)
2: $c_{\ell 2} \leftarrow$ *RN*$(x_\ell \cdot y)$
3: $c_{\ell 3} \leftarrow$ *RN*$(c_{\ell 1} + c_{\ell 2})$
4: $(z_h, z_\ell) \leftarrow$ *Fast2Sum*$(c_h, c_{\ell 3})$
5: **return** (z_h, z_ℓ)

– implemented in Briggs and Bailey's libraries;
– no previously published error bound;
– we proved that if $p \geq 3$ the relative error is less than

$$3 \cdot 2^{-2p};$$

– speed and accuracy can be improved if an FMA instruction is available (merging lines 2 and 3).

Multiplication: **DWTimesDW**$(x_h, x_\ell, y_h, y_\ell)$

Algorithm 7

1: $(c_h, c_{\ell 1}) \leftarrow$ *Fast2Mult*(x_h, y_h)
2: $t_{\ell 1} \leftarrow$ *RN*$(x_h \cdot y_\ell)$
3: $t_{\ell 2} \leftarrow$ *RN*$(x_\ell \cdot y_h)$
4: $c_{\ell 2} \leftarrow$ *RN*$(t_{\ell 1} + t_{\ell 2})$
5: $c_{\ell 3} \leftarrow$ *RN*$(c_{\ell 1} + c_{\ell 2})$
6: $(z_h, z_\ell) \leftarrow$ *Fast2Sum*$(c_h, c_{\ell 3})$
7: **return** (z_h, z_ℓ)

– suggested by Dekker and implemented in Briggs and Bailey's libraries;
– Dekker proved a relative error bound of $11 \cdot 2^{-2p}$;
– we improved it, proving that if $p \geq 4$ the relative error is less than

$$7 \cdot 2^{-2p};$$

– speed and accuracy can be improved if an FMA instruction is available.

Division: **DWDivFP1**(x_h, x_ℓ, y)

Algorithm 8

$t_h \leftarrow$ *RN* (x_h/y)
$(\pi_h, \pi_\ell) \leftarrow$ *Fast2Mult*(t_h, y)
$(\delta_h, \delta') \leftarrow$ *2Sum*$(x_h, -\pi_h)$
$\delta'' \leftarrow$ *RN* $(x_\ell - \pi_\ell)$
$\delta_\ell \leftarrow$ *RN* $(\delta' + \delta'')$
$\delta \leftarrow$ *RN* $(\delta_h + \delta_\ell)$
$t_\ell \leftarrow$ *RN* (δ/y)
$(z_h, z_\ell) \leftarrow$ *Fast2Sum*(t_h, t_ℓ)
return (z_h, z_ℓ)

– algorithm suggested by Bailey;
– previously known error bound [Li.et.al02] of $4 \cdot 2^{-2p}$;

Division: **DWDivFP1**(x_h, x_ℓ, y)

Algorithm 8

1: $t_h \leftarrow RN(x_h/y)$
2: $(\pi_h, \pi_\ell) \leftarrow \mathit{Fast2Mult}(t_h, y)$
3: $(\delta_h, \delta') \leftarrow \mathit{2Sum}(x_h, -\pi_h)$
4: $\delta'' \leftarrow RN(x_\ell - \pi_\ell)$
5: $\delta_\ell \leftarrow RN(\delta' + \delta'')$
6: $\delta \leftarrow RN(\delta_h + \delta_\ell)$
7: $t_\ell \leftarrow RN(\delta/y)$
8: $(z_h, z_\ell) \leftarrow \mathit{Fast2Sum}(t_h, t_\ell)$
9: **return** (z_h, z_ℓ)

– algorithm suggested by Bailey;
– previously known error bound [Li.et.al02] of $4 \cdot 2^{-2p}$;
– Improvement: we showed that the addition in line 3 is always exact.

$\Longrightarrow$ new algorithm

Division: **DWDivFP2**(x_h, x_ℓ, y)

Algorithm 9

1: $t_h \leftarrow RN(x_h/y)$
2: $(\pi_h, \pi_\ell) \leftarrow \mathit{Fast2Mult}(t_h, y)$
3: $\delta_h \leftarrow RN(x_h - \pi_h)$
4: $\delta_\ell \leftarrow RN(x_\ell - \pi_\ell)$
5: $\delta \leftarrow RN(\delta_h + \delta_\ell)$
6: $t_\ell \leftarrow RN(\delta/y)$
7: $(z_h, z_\ell) \leftarrow \mathit{Fast2Sum}(t_h, t_\ell)$
8: **return** (z_h, z_ℓ)

– less FP operations, but mathematically equivalent;
– slightly improved error bound:

$$\frac{7}{2} \cdot 2^{-2p},$$

as soon as $p \geq 4$.

Overview

Algorithm	Previously known bound	Our bound	Largest relative error found in experiments	♯ of FP ops
DWPlusFP	?	$2u^2 + 5u^3$	$2u^2 - 6u^3$	10
SloppyDWPlusDW	N/A	N/A	1	11
AccurateDWPlusDW	$2u^2$ (wrong)	$3u^2 + 13u^3$	$2.25u^2$	20
DWTimesFP1	$4u^2$	$2u^2$	$1.5u^2$	10
DWTimesFP2	?	$3u^2$	$2.517u^2$	7
DWTimesFP3 (fma)	N/A	$2u^2$	$1.984u^2$	6
DWTimesDW1	$11u^2$	$7u^2$	$4.9916u^2$	9
DWTimesDW2 (fma)	N/A	$5u^2$	$3.936u^2$	9
DWDivFP1*	$4u^2$	$3.5u^2$	$2.95u^2$	16
DWDivFP2*	N/A	$3.5u^2$	$2.95u^2$	10
DWDivDW1*	?	$15u^2 + 56u^3$	$8.465u^2$	24
DWDivDW2*	N/A	$15u^2 + 56u^3$	$8.465u^2$	18
DWDivDW3 (fma)	N/A	$9.8u^2$	$5.922u^2$	31

Conclusions

– many similar algorithms with small differences;

– no correctness proofs and error bounds;

– need to clean up the literature and implementation;

Tight and rigorous error bounds for basic building blocks of double-word arithmetic. Submitted to ACM TOMS journal. `hal.archives-ouvertes.fr/hal-01351529`

Conclusions

– many similar algorithms with small differences;

– no correctness proofs and error bounds;

– need to clean up the literature and implementation;

+ we looked at 13 algorithms, both old and new;

+ we compared them and provided correctness proofs and error bounds;

+ code available online at: http://homepages.laas.fr/mmjoldes/campary/.

Tight and rigorous error bounds for basic building blocks of double-word arithmetic. Submitted to ACM TOMS journal. `hal.archives-ouvertes.fr/hal-01351529`