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

Mioara Joldes, Jean-Michel Muller, Valentina Popescu

SCAN - September 2016
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 $x_\ell$ such that

$$x_h = \text{RN} \left( x \right).$$
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 $x_\ell$ such that

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

→ Called double-double when using the binary64 standard format.

Example: $\pi$ in double-double

$$p_h = 11.00100100001111110110101000100010001011100011000_2,$$

and

$$p_\ell = 1.00011010000100110001100011000101000111100000000111_2 \times 2^{-53};$$

$$p_h + p_\ell \leftrightarrow 107 \text{ bit FP approx.}$$
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.
Not the same as IEEE 754-2008 standard’s binary128/quadruple-precision.

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

Binary128/quadruple-precision:
Remark

→ Not the same as IEEE 754-2008 standard’s \textit{binary128/quadruple}-precision.

Double-word (using \textit{binary64/double}-precision):
- “wobbling precision” $\geq 107$ bits of precision;

\textbf{Binary128/quadruple}-precision:
- 113 bits of precision;
Remark

Not the same as IEEE 754-2008 standard’s \textit{binary128/quadruple}-precision.

\textbf{Double-word (using \textit{binary64/double}-precision):}

- “wobbling precision” $\geq 107$ bits of precision;
- exponent range limited by \textit{binary64} (11 bits) i.e. $-1022$ to $1023$;

\textbf{Binary128/quadruple}-precision:

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

→ Not the same as IEEE 754-2008 standard’s *binary128/quadruple*-precision.

**Double-word (using *binary64/double*-precision):**
- “wobbling precision” $\geq 107$ 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

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 $(a, b)$)

$\begin{align*}
  s &\leftarrow RN(a + b) \\
  t &\leftarrow RN(s - b) \\
  e &\leftarrow RN(RN(a - t) + RN(b - RN(s - t))) \\
\text{return} (s, e)
\end{align*}$

$\rightarrow$ 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;
- $s = \text{RN}(a + b)$.

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

$s \leftarrow \text{RN}(a + b)$

$z \leftarrow \text{RN}(s - a)$

$e \leftarrow \text{RN}(b - z)$

**return** $(s, e)$

$\rightarrow$ 3 FP operations
Theorem 3 (2ProdFMA algorithm)

Let $a$ and $b$ be FP numbers, $e_a + e_b \geq e_{\text{min}} + p - 1$. 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 $(p, e)$

$\rightarrow$ 2 FP operations

$\rightarrow$ 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.

Notation:
- \( p \) represents the precision of the underlying FP format;
- \( \text{RN}(t) \) stands for \( t \) rounded to the nearest FP number, ties-to-even;
- \( \text{ulp}(x) = 2^\lfloor \log_2 |x| \rfloor - p + 1 \), for \( x \neq 0 \);
- \( u = 2^{-p} = \frac{1}{2} \text{ulp}(1) \) denotes the roundoff error unit.
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.

→ 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.

→ Strong need to “clean up” the existing literature.

Notation:
- \( p \) represents the precision of the underlying FP format;
- \( \text{RN}(t) \) stands for \( t \) rounded to the nearest FP number, ties-to-even;
- \( \text{ulp}(x) = 2^{\lfloor \log_2 |x| \rfloor - p + 1} \), for \( x \neq 0 \);
- \( u = 2^{-p} = \frac{1}{2} \text{ulp}(1) \) denotes the roundoff error unit.
Addition: \( \text{DWPlusFP}(x_h, x_\ell, y) \)

**Algorithm 4**

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

- computing \(\underbrace{x_h + x_\ell + y}_x\);
- 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 \(p \geq 4\).
Algorithm 5

1: \( (s_h, s_\ell) \leftarrow 2\text{Sum}(x_h, y_h) \)
2: \( (t_h, t_\ell) \leftarrow 2\text{Sum}(x_\ell, y_\ell) \)
3: \( c \leftarrow \text{RN}(s_\ell + t_h) \)
4: \( (v_h, v_\ell) \leftarrow \text{Fast2Sum}(s_h, c) \)
5: \( w \leftarrow \text{RN}(t_\ell + v_\ell) \)
6: \( (z_h, z_\ell) \leftarrow \text{Fast2Sum}(v_h, w) \)
7: return \( (z_h, z_\ell) \)

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

Addition: \textbf{AccurateDWPlusDW}(x_h, x_\ell, y_h, y_\ell)
Addition: **AccurateDWPlusDW**(*x*_h, *x*_ℓ, *y*_h, *y*_ℓ)

**Algorithm 5**

1: (*s*_h, *s*_ℓ) ← 2Sum(*x*_h, *y*_h)
2: (*t*_h, *t*_ℓ) ← 2Sum(*x*_ℓ, *y*_ℓ)
3: *c* ← RN(*s*_ℓ + *t*_h)
4: (*v*_h, *v*_ℓ) ← Fast2Sum(*s*_h, *c*)
5: *w* ← RN(*t*_ℓ + *v*_ℓ)
6: (*z*_h, *z*_ℓ) ← Fast2Sum(*v*_h, *w*)
7: return (*z*_h, *z*_ℓ)

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

\[
\begin{align*}
  x_h &= 2^p - 1, & x_\ell &= -(2^p - 1) \cdot 2^{-p-1}, \\
  y_h &= -(2^p - 5)/2, & y_\ell &= -(2^p - 1) \cdot 2^{-p-3},
\end{align*}
\]

which leads to a relative error asymptotically equivalent to 2.25 \(\times\) 2\(^{-2p}\);
Addition: **AccurateDWPlusDW**($x_h, x_{\ell}, y_h, y_{\ell}$)

**Algorithm 5**

1: $(s_h, s_{\ell}) \leftarrow 2\text{Sum}(x_h, y_h)$
2: $(t_h, t_{\ell}) \leftarrow 2\text{Sum}(x_{\ell}, y_{\ell})$
3: $c \leftarrow \text{RN}(s_{\ell} + t_h)$
4: $(v_h, v_{\ell}) \leftarrow \text{Fast2Sum}(s_h, c)$
5: $w \leftarrow \text{RN}(t_{\ell} + v_{\ell})$
6: $(z_h, z_{\ell}) \leftarrow \text{Fast2Sum}(v_h, w)$
7: return $(z_h, z_{\ell})$

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

\[
\begin{align*}
x_h &= 2^p - 1, & x_{\ell} &= -(2^p - 1) \cdot 2^{-p-1}, \\
y_h &= -(2^p - 5)/2, & y_{\ell} &= -(2^p - 1) \cdot 2^{-p-3},
\end{align*}
\]

which leads to a relative error asymptotically equivalent to $2.25 \times 2^{-2p}$;
– 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.
**Algorithm 6**

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

– 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).
Algorithm 7

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

– 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: \( \text{DWDivFP1}(x_h, x_\ell, y) \)

**Algorithm 8**

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

- algorithm suggested by Bailey;
- previously known error bound [Li.et.al02] of \( 4 \cdot 2^{-2p} \);
### Algorithm 8

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

- 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.

\[\Rightarrow\] New algorithm
Algorithm 9

1: \( t_h \leftarrow RN(x_h/y) \)
2: \((\pi_h, \pi_\ell) \leftarrow Fast2Mult(t_h, y)\)
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: \((z_h, z_\ell) \leftarrow Fast2Sum(t_h, t_\ell)\)
8: return \((z_h, z_\ell)\)

– less FP operations, but mathematically equivalent;
– slightly improved error bound:
\[
\frac{7}{2} \cdot 2^{-2p},
\]
as soon as \( p \geq 4 \).
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Previously known bound</th>
<th>Our bound</th>
<th>Largest relative error found in experiments</th>
<th># of FP ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>DWPlusFP</td>
<td>?</td>
<td>$2u^2 + 5u^3$</td>
<td>$2u^2 - 6u^3$</td>
<td>10</td>
</tr>
<tr>
<td>SloppyDWPlusDW</td>
<td>N/A</td>
<td>N/A</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>AccurateDWPlusDW</td>
<td>$2u^2$ (wrong)</td>
<td>$3u^2 + 13u^3$</td>
<td>$2.25u^2$</td>
<td>20</td>
</tr>
<tr>
<td>DWTimesFP1</td>
<td>$4u^2$</td>
<td>$2u^2$</td>
<td>$1.5u^2$</td>
<td>10</td>
</tr>
<tr>
<td>DWTimesFP2</td>
<td>?</td>
<td>$3u^2$</td>
<td>$2.517u^2$</td>
<td>7</td>
</tr>
<tr>
<td>DWTimesFP3 (fma)</td>
<td>N/A</td>
<td>$2u^2$</td>
<td>$1.984u^2$</td>
<td>6</td>
</tr>
<tr>
<td>DWTimesDW1</td>
<td>$11u^2$</td>
<td>$7u^2$</td>
<td>$4.9916u^2$</td>
<td>9</td>
</tr>
<tr>
<td>DWTimesDW2 (fma)</td>
<td>N/A</td>
<td>$5u^2$</td>
<td>$3.936u^2$</td>
<td>9</td>
</tr>
<tr>
<td>DWDivFP1*</td>
<td>$4u^2$</td>
<td>$3.5u^2$</td>
<td>$2.95u^2$</td>
<td>16</td>
</tr>
<tr>
<td>DWDivFP2*</td>
<td>N/A</td>
<td>$3.5u^2$</td>
<td>$2.95u^2$</td>
<td>10</td>
</tr>
<tr>
<td>DWDivDW1*</td>
<td>?</td>
<td>$15u^2 + 56u^3$</td>
<td>$8.465u^2$</td>
<td>24</td>
</tr>
<tr>
<td>DWDivDW2*</td>
<td>N/A</td>
<td>$15u^2 + 56u^3$</td>
<td>$8.465u^2$</td>
<td>18</td>
</tr>
<tr>
<td>DWDivDW3 (fma)</td>
<td>N/A</td>
<td>$9.8u^2$</td>
<td>$5.922u^2$</td>
<td>31</td>
</tr>
</tbody>
</table>
Conclusions

– many similar algorithms with small differences;
– no correctness proofs and error bounds;
– need to clean up the literature and implementation;

[9x256]Conclusions

[14x198]– many similar algorithms with small differences;
– no correctness proofs and error bounds;
– need to clean up the literature and implementation;

[2014x2157]+ we looked at 13 algorithms, both old and new;
+ we compared them and provided correctness proofs and error bounds;
+ code soon to be available online at: http://homepages.laas.fr/mmjoldes/campary/.

A

MPA R

CudA  Multiple  Precision  ARithmetic  librarY

[118x93]AMPA R

CudA  Multiple  Precision  ARithmetic  librarY

[346x1]15 / 15
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 soon to be available online at: http://homepages.laas.fr/mmjoldes/campary/.