Apporter de la confiance aux calculs en arithmétique virgule flottante

Jean-Michel Muller

CNRS - Laboratoire LIP

Septembre 2024

http://perso.ens-lyon.fr/jean-michel.muller/

Floating-Point Arithmetic

- by far the most frequent solution for manipulating real numbers in computers;
- comes from the "scientific notation" used for 3 centuries by the scientific community;

Sometimes a bad reputation... for bad reasons:

- intrinsically approximate...
  - but most data is approximate;
  - but most numerical problems we deal with have no closed-form solution;
  - and in a subtle way (correct rounding), FP arithmetic is exact.
- part of the literature comes from times when it was poorly specified;

$\rightarrow$ too often, viewed as a mere set of cooking recipes.

We wish to show that

- it is a well specified arithmetic, on which one can build trustable calculations;
- one can prove useful properties and build efficient algorithms on FP arithmetic;
- and yet the proofs are complex: formal proof is helpful.

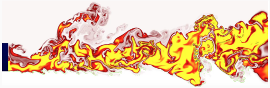
Desirable properties of an arithmetic system

- Speed: tomorrow's weather must be computed in less than 24 hours;
- Reliability: all numerical computing is built upon basic arithmetic. If the arithmetic collapses, everything collapses;
- Accuracy;
- Range: represent big and tiny numbers as well;
- Size: silicon area for hardware, memory consumption for software;
- Power consumption;
- Easiness of implementation and use: If a given arithmetic is too arcane, nobody will use it. . .

. . . of course, you can't win on all fronts.

Much change since the 70's and 80's: i) applications

Numerical simulation

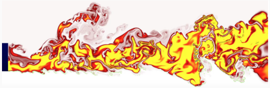


- trillions of operations
- crash? just start again the simulation (but not too often)
- reproducibility may be useful.

Finance

Much change since the 70's and 80's: i) applications

Numerical simulation



- trillions of operations
- crash? just start again the simulation (but not too often)
- reproducibility may be useful.

Finance

Embedded computing

- speed: yes, but no need to be faster than real time;
- crash? ahem...

→ certified calculations.

Entertainment

- Supermario's pizza: no need to carefully follow the laws of physics;
- fluidity matters;
- reproducibility: each player must see the same game landscape.

Artificial intelligence

- neural net training: huge amount of very low precision calculations.

Much change since the 70's and 80's: ii) performance

- the ratio

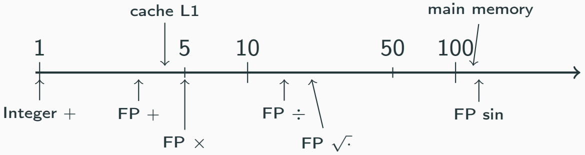
$$\frac{\text{time to read/write in memory}}{\text{time to perform } +, \times, \div, \sqrt{}}$$

has increased by a factor around 140 between 1986 and 2000;

- It has continued to increase after 2000, but at a somehow slower pace;
- the challenge is no longer to design fast arithmetic operators, but to be able to feed them with data at a very high rate;

→ first consequence: many new architectural concepts (multiple levels of cache, pipelining, vector instructions, branch prediction);

→ second consequence: incentive to use small formats whenever possible.



Much change since the 70's and 80's: iii) FP formats

$$\begin{array}{l} \text{single precision (a.k.a. binary32)} \\ \text{double precision (a.k.a. binary64)} \end{array} \Rightarrow \begin{cases} \text{8-bit emerging formats for IA} \\ \text{BFloat16} \\ \text{binary16} \\ \text{binary32} \\ \text{binary64} \\ \text{binary128 (quad)} \end{cases}$$

- Combinatorial explosion of all the possible arithmetic operators of the form `Format 1` $\times$ `Format 2` $\rightarrow$ `Format 3`
- Need to develop and maintain math function libraries for all these formats.

Cleverly using these formats:

Locate when low precision puts us at an unacceptable risk.	$\longleftrightarrow$	Locate when big precision totally destroys performance.

Numerical analysis, abstract interpretation, compilation, computer architecture, formal proof, ...

A few weird arithmetic things

- Excel'2007 (first releases), compute $65535 - 2^{-37}$, you get 100000;
- 2020: in a competition, robotic car crash due to bad handling of floating-point exception

- if you have a Casio FX-92 pocket calculator, compute $11^6/13$, you will get

$$\frac{156158413}{3600}\pi$$

compute $97027288/89521$, you will get

$$345\pi.$$

Base 2, precision-p FP arithmetic

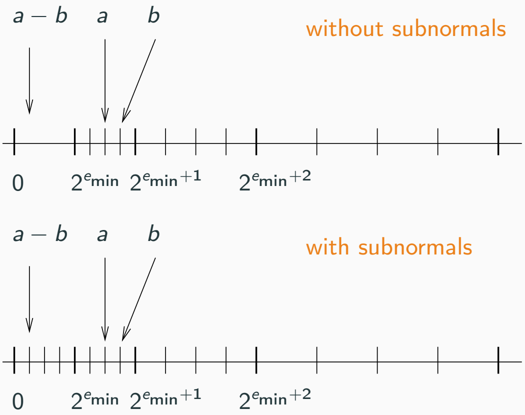
In binary, precision-p Floating-Point (FP) arithmetic, a number x is represented by two integers M (integral significand) and e (exponent):

$$x = \left(\frac{M}{2^{p-1}}\right) \cdot 2^{e} = m_0.m_1 m_2 \cdots m_{p-1} \cdot 2^{e}$$

where $M, e \in \mathbb{Z}$, with $|M| \leq 2^p - 1$ and $e_{\min} \leq e \leq e_{\max}$. Additional requirement: e smallest under these constraints.

- x is normal if $|x| \geq 2^{e_{\min}}$ (implies $|M| \geq 2^{p-1}$, i.e., $m_0 = 1$);
- x is subnormal otherwise ($m_0 = 0$).

Subnormal numbers complicate the implementation of FP multiplication, but...



If a and b are FPN, $a \neq b$ equivalent to "computed $a - b \neq 0$".

Theorem 1 (Hauser)

If the absolute value of the sum/difference of two FP numbers is $\leq 2^{e_{min}+1}$ then it is a floating-point number (i.e., it is exactly representable in FP arithmetic).

Before 1985: a total mess. . .

Machine	Underflow λ	Overflow Λ
DEC PDP-11, VAX, F and D formats	$2^{-128} \approx 2.9 \times 10^{-39}$	$2^127 \approx 1.7 \times 10^38$
DEC PDP-10; Honeywell 600, 6000; Univac 110x single; IBM 709X, 704X	$2^{-129} \approx 1.5 \times 10^{-39}$	$2^127 \approx 1.7 \times 10^38$
Burroughs 6X00 single	$8^{-51} \approx 8.8 \times 10^{-47}$	$8^{76} \approx 4.3 \times 10^{68}$
H-P 3000	$2^{-256} \approx 8.6 \times 10^{-78}$	$2^{256} \approx 1.2 \times 10^{77}$
IBM 360, 370; Amdahl1; DG Eclipse M/600; ...	$16^{-65} \approx 5.4 \times 10^{-79}$	$16^{63} \approx 7.2 \times 10^{75}$
Most handheld calculators	10^{-99}	10^{100}
CDC 6X00, 7X00, Cyber	$2^{-976} \approx 1.5 \times 10^{-294}$	$2^{1070} \approx 1.3 \times 10^{322}$
DEC VAX G format; UNIVAC, 110X double	$2^{-1024} \approx 5.6 \times 10^{-309}$	$2^{1023} \approx 9 \times 10^{307}$

Source: Kahan, *Why do we need a Floating-Point Standard*, 1981.

Before 1985: a total mess...

- Some Cray computers: overflow in FP $\times$ detected just from the exponents of the entries, in parallel with the actual computation of the product;

$\rightarrow$ `1 * x` could overflow;

- still on the Crays, only 12 bits of x were examined to detect a division by 0 when computing y/x

$\rightarrow$ `if (x = 0) then z := 17.0 else z := y/x`
could lead to `zero divide` error message... but since the multiplier too examined only 12 bits to decide if an operand is zero,

```
if (1.0 * x = 0) then z := 17.0 else z := y/x
```

was just fine.

- many systems, not enough "guard bits" for FP $+$ $\rightarrow$ for $x \approx 1$, experts knew that $(0.5 - x) + 0.5$ was much better than $1.0 - x$.

Writing reliable and portable numerical software was a challenge!

IEEE-754 Standard for FP Arithmetic (1985, 2008, 2019)

- put an end to a mess (no portability, variable quality);
- leader: W. Kahan (father of the arithmetic of the HP35 and the Intel 8087);
- formats (in radices 2 and 10);
- specification of operations and conversions;
- exception handling (max+1, 1/0, $\sqrt{-2}$, 0/0, etc.);
- successive versions of the standard: 2008, 2019, and 2029 is already in preparation.

Correct rounding

- the sum, product, . . . of two FP numbers is not, in general, a FP number $\rightarrow$ must be rounded;
- the IEEE 754 Std for FP arithmetic specifies several rounding functions;
- the default function is RN ties to even.

Correctly rounded operation: returns what we would get by exact operation followed by rounding.

- correctly rounded $+$, $-$, $\times$, $\div$, $\sqrt{\cdot}$ are required;

$\rightarrow$ when `c = a + b` appears in a program, we get $c = \mathrm{RN}\,(a + b)$.
$\rightarrow$ somehow deterministic arithmetic (more later).

ulp (unit in the last place), u (unit round-off)

Binary, precision-p FP arithmetic.

- If $|x| \in [2^e, 2^{e+1})$, then $\mathrm{ulp}(x) = 2^{\max\{e, e_{\min}\}-p+1}$.
  - Frequently used for expressing errors of atomic functions;
  - distance between consecutive FP numbers near x;
- if $2^{e_{\min}} \leq |x| \leq \Omega$, then

  $$|x - \mathrm{RN}(x)| \leq \frac{1}{2}\mathrm{ulp}(x) = 2^{\lfloor \log_2 |x| \rfloor - p},$$

  therefore,

  $$|x - \mathrm{RN}(x)| \leq u \cdot |x|, \qquad (1)$$

  with $u = 2^{-p}$. Hence the relative error

  $$\frac{|x - \mathrm{RN}(x)|}{|x|}$$

  (for $x \neq 0$) is $\leq u$.
- u, called unit round-off is frequently used for expressing relative errors.

With the math functions work still needs to be done

library	GNU libc	IML	AMD	Newlib	OpenLibm	Musl	Apple	LLVM	MSVC	FreeBSD	ArmPL	CUDA	ROCm
version	2.40	2024.0.2	4.2	4.4.0	0.8.3	1.2.5	14.5	18.1.8	2022	14.1	24.04	12.2.1	5.7.0
acos	**0.523**	0.531	1.36	0.930	0.930	0.930	1.06		0.934	0.930	1.52	1.53	0.772
acosh	2.25	**0.509**	1.32	2.25	2.25	2.25	2.25		3.22	2.25	2.66	2.52	0.661
asin	**0.516**	0.531	1.06	0.981	0.981	0.981	0.709		1.05	0.981	2.69	1.99	0.710
asinh	1.92	**0.507**	1.65	1.92	1.92	1.92	1.58		2.05	1.92	2.04	2.57	0.661
atan	**0.523**	0.528	0.863	0.861	0.861	0.861	0.876		0.863	0.861	2.24	1.77	1.73
atanh	1.78	**0.507**	1.04	1.81	1.81	1.80	2.01		2.50	1.81	3.00	2.50	0.664
cbrt	3.67	0.523	1.53e22	0.670	0.668	0.668	0.729		1.86	0.668	1.79	**0.501**	**0.501**
cos	**0.516**	0.518	0.919	0.887	0.834	0.834	0.948	Inf	0.897	0.834		1.52	0.797
cosh	1.93	**0.516**	1.85	2.67	1.47	1.04	0.523		1.91	1.47	1.93	1.40	0.563
erf	1.43	**0.773**	1.00	1.02	1.02	1.02	6.41		4.62	1.02	2.29	1.50	1.12
erfc	5.19	**0.826**		4.08	4.08	3.72	10.7		8.46	4.08	1.71	4.51	4.08
exp	0.511	0.530	1.01	0.949	0.949	0.511	0.521	**0.500**	1.50	0.949	0.511	0.928	0.929

Largest errors in ulps for double-precision calculation of some math functions. ulp (x) is the distance between two FP numbers in the neighborhood of x (so the largest values should be 0.5 – which is the case with $+$, $-$, $\times$, $\div$, and $\sqrt{\cdot}$).

(Extracted from Gladman, Innocente, Mather, and Zimmermann, *Accuracy of Mathematical Functions. . .*, Aug. 2024)

Exception handling: the show must go on...

- when an exception occurs: the computation must continue (default behaviour);
- two infinities and two zeros, with intuitive rules: $1/(+0) = +\infty$, $5 + (-\infty) = -\infty \ldots$;
- and yet, something a little odd: $\sqrt{-0} = -0$;
- Not a Number (NaN): result of $\sqrt{-5}$, $(\pm 0)/(\pm 0)$, $(\pm\infty)/(\pm\infty)$, $(\pm 0) \times (\pm\infty)$, NaN $+3$, etc.

$$f(x) = 3 + \frac{1}{x^5}$$

will give the very accurate answer 3 for huge x, even if x^5 overflows.

One should be cautious: behavior of

$$\frac{x^2}{\sqrt{x^3 + 1}}$$

for large x.

With correct rounding and standardized exception handling, arithmetic is almost deterministic

- watch the dependency graph of operations (beware of "optimizing" compilers);
- watch the format of the implicit variables (such as the `x+y` in `(x+y)*(z+t)`);
- math functions still a problem unless you use a correctly rounded library such as Zimmermann & Sibidanov's Core Math,[1] or LLVM libc.[2]

With enough care we can prove properties and build specific algorithms.

[1] `https://core-math.gitlabpages.inria.fr/`
[2] `https://libc.llvm.org/`

A useful property: Sterbenz' Theorem

Theorem 2 (Sterbenz)

Let a and b be positive FP numbers. If

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

then $a - b$ is a FP number
($\rightarrow$ computed exactly, whatever the rounding function).

Beware: the "2"s in the formula are not the radix. In radices 10, 16 or 42, the same property holds, still with $\frac{a}{2} \leq b \leq 2a$.

Example of use: implementation of trig. functions in precision-p FP arithmetic

- cosine function: range reduction to small interval followed by polynomial approximation in that interval;
- range reduction: $x \to y = x - k\pi$ such that $|y|$ is small. If done naively this is a very inaccurate operation.
- assuming the largest value of k of interest fits in $m < p$ bits, express π as the sum of two FP numbers π_1 and π_2 such that
  - π_1 is closest to π among the FP numbers whose significand fits in $p - m$ bits;
  - $\pi_2 = \mathrm{RN}(\pi - \pi_1)$.

  Program: $\texttt{y} \leftarrow \texttt{((x - k*}\pi_1\texttt{) - k*}\pi_2\texttt{)}$

By construction, $\Delta = \texttt{k*}\pi_1$ is exact, and by Sterbenz Lemma, $\texttt{x} - \Delta$ is exact.

(Cody-Waite range reduction. Many improvements are possible)

The error of (RN) FP addition is a FPN

Lemma 3

Let a and b be two FP numbers. Let

$$s = RN(a+b) \text{ and } r = (a+b) - s.$$

If no overflow when computing s, then r is a FP number.

Beware: does not always work with rounding functions $\neq$ RN.

Get r: the fast2sum algorithm (Dekker)

Theorem 4 (Fast2Sum (Dekker))

(only radix 2). Let a and b be FP numbers, s.t. $|a| \geq |b|$. Following algorithm: s and r such that

- $s + r = a + b$ *exactly;*
- s *is "the" FP number that is closest to* $a + b$*;*

Algorithm 1 (FastTwoSum)

$s \leftarrow RN(a + b)$

$z \leftarrow RN(s - a)$

$r \leftarrow RN(b - z)$

C Program 1

```
s = a+b;
z = s-a;
r = b-z;
```

Important remark: Proving the behavior of such algorithms requires use of the correct rounding property.

The TwoSum Algorithm (Moller-Knuth)

- no need to compare a and b;
- 6 operations instead of 3 yet, on many architectures, very cheap in front of wrong branch prediction penalty when comparing a and b;
- works in all bases.

Algorithm 2 (TwoSum)

$s \leftarrow RN(a+b)$
$a' \leftarrow RN(s-b)$
$b' \leftarrow RN(s-a')$
$\delta_a \leftarrow RN(a-a')$
$\delta_b \leftarrow RN(b-b')$
$r \leftarrow RN(\delta_a+\delta_b)$

Knuth: if no underflow nor overflow occurs then $a+b=s+r$, and s is nearest $a+b$.

Boldo et al: formal proof + underflow does not hinder the result (overflow does).

TwoSum is optimal (no way of always obtaining the same result with less than 6 $\pm$ operations).

Example of application: computing $x_1 + x_2 + x_3 + \cdots + x_n$

Naive algorithm:

$s \leftarrow x_1$
for $i = 2$ to n **do**
  $s \leftarrow \mathrm{RN}(s + x_i)$
end for
return s

Pichat, Ogita, Rump, and Oishi:

$s \leftarrow x_1$
$e \leftarrow 0$
for $i = 2$ to n **do**
  $(s, e_i) \leftarrow \mathrm{2Sum}(s, x_i)$
  $e \leftarrow \mathrm{RN}(e + e_i)$
end for
return $\mathrm{RN}(s + e)$

Error bounds:

$$(n-1) \cdot u \sum |x_i|$$

$$u \left| \sum_{i=1}^{n} x_i \right| + \left(\frac{(n-1)u}{1-(n-1)u} \right)^2 \sum_{i=1}^{n} |x_i|$$

(remember: $u = 2^{-p}$)

What about products ?

- If a and b are FP numbers, then (under mild conditions), $r = ab - \mathrm{RN}\,(ab)$ is a FP number;
- We use the *fused multiply-add* (fma) instruction. It computes $\mathrm{RN}\,(ab + c)$. First appeared in IBM RS6000, Intel/HP Itanium, PowerPC... Specified since 2008.
- obtained with algorithm TwoMultFMA $\begin{cases} p &= & \mathrm{RN}\,(ab) \\ r &= & \mathrm{RN}\,(ab - p) \end{cases}$

  $\rightarrow$ 2 operations only, gives $p + r = ab.$

Just an example: $ad - bc$ with fused multiply-add

Kahan's algorithm for $x = ad - bc$:

$\hat{w} \leftarrow \mathrm{RN}(bc)$
$e \leftarrow \mathrm{RN}(\hat{w} - bc)$
$\hat{f} \leftarrow \mathrm{RN}(ad - \hat{w})$
$\hat{x} \leftarrow \mathrm{RN}(\hat{f} + e)$
Return $\hat{x}$

- we have proven (2011):

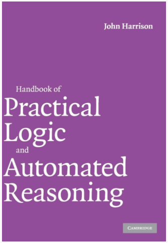
$$|\hat{x} - x| \le 2u|x|$$

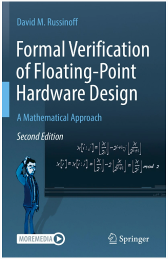
  "asymptotically optimal" error bound.

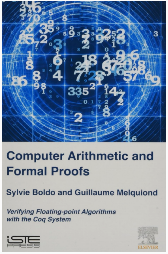
- $\rightarrow$ rotations, complex arithmetic.

Formal verification of FP algorithms

- starting point: the Pentium division bug (1994)
- J. Harrison formalized FP arithmetic in HOL Light, formally proved the division and sqrt algorithms of the Intel Itanium, and some elementary function algorithms (around 1999);
- D. Russinoff: similar things for AMD (more on the hardware side);
- Sylvie Boldo and Guillaume Melquiond use the Coq proof assistant (Flocq library, Gappa tool).







Double-Word arithmetic

- Fast2Sum, 2Sum and 2MultFMA return their result as the unevaluated sum of two FP numbers.
- idea: manipulate such unevaluated sums to perform more accurate calculations in critical parts of a numerical program.

$\rightarrow$ "double word" or "double-double" arithmetic. Most recent avatar: Rump and Lange's "pair arithmetic" (2020).

Definition 5

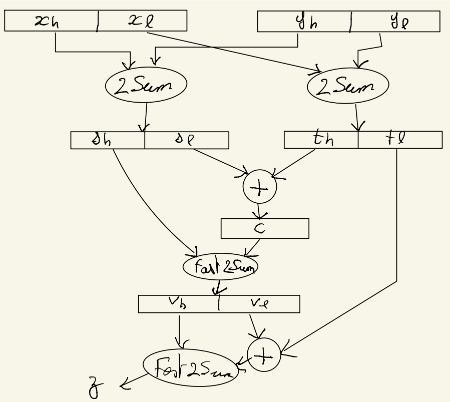
A double-word (DW) number x is the unevaluated sum $x_h + x_\ell$ of two floating-point numbers x_h and x_ℓ such that

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

Sum of two DW numbers. There exist a “quick & dirty” algorithm, but its relative error is unbounded.

DWPlusDW

$(s_h, s_\ell) \leftarrow \text{2Sum}(x_h, y_h)$
$(t_h, t_\ell) \leftarrow \text{2Sum}(x_\ell, y_\ell)$
$c \leftarrow \text{RN}\,(s_\ell + t_h)$
$(v_h, v_\ell) \leftarrow \text{Fast2Sum}(s_h, c)$
$w \leftarrow \text{RN}\,(t_\ell + v_\ell)$
$(z_h, z_\ell) \leftarrow \text{Fast2Sum}(v_h, w)$
return (z_h, z_ℓ)



DW+DW: “accurate version”

We have (after a very long and tedious proof):

Theorem 6

If $p \geq 3$, the relative error of Algorithm DWPlusDW is bounded by

$$\frac{3u^2}{1-4u} = 3u^2 + 12u^3 + 48u^4 + \cdots, \tag{2}$$

That theorem has an interesting history. . .

ALGORITHM 6: – **AccurateDWPlusDW**$(x_h, x_\ell, y_h, y_\ell)$. Calculation of $(x_h, x_\ell) + (y_h, y_\ell)$ in binary, precision-p, floating-point arithmetic.

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

Li et al. (2000, 2002) claim that in binary64 arithmetic ($p = 53$) the relative error of Algorithm 6 is upper bounded by $2 \cdot 2^{-106}$. This bound is inco…

$x_h = 900719$…
$x_\ell = -9007$…
$y_h = -9007$…
$y_\ell = -9007$…

then the relative error of Algorithm 6 is

2.249999999…

Note that this example is somehow "gene… $2^p - 1, x_\ell = -(2^p - 1) \cdot 2^{-p-1}, y_h = -(2^p - 5)/2,$… that is asymptotically equivalent (as p goes to infini…

Now let us try to find a relative error bound. We are g…

in which one of the opera…
he returned result is 2Sum$(x_\ell, y_\ell,$
Now, without loss of generality, we
nonzero. Notice that $1 \le x_h <$

Theorem 3.1. *If $p \ge 3$, then the relative error of Algorithm 6* … *by*

$$\frac{3u^2}{1-4u} = 3u^2 + 12u^3 + 48u^4 + \cdots,$$

which is less than $3u^2 + 13u^3$ as soon as $p \ge 6$.

Note that the conditions on p ($p \ge 3$ for the bound (3) to hold, $p \ge 6$ for the simplified bound $3u^2 + 13u^3$) are satisfied in all practical cases.

Proof. First, we exclude the straightforward case in which one of the operands is zero. We can also quickly proceed with the case $x_h + y_h = 0$: The returned result is 2Sum(x_ℓ, y_ℓ), which is equal to $x + y$, that is, the computation is errorless. Now, without loss of generality, we assume $1 \le x_h < 2$, $x \ge |y|$ (which implies $x_h \ge |y_h|$), and $x_h + y_h$ nonzero. Notice that $1 \le x_h < 2$ implies $1 \le x_h \le 2 - 2u$, since x_h is a FP number.

Define ϵ_1 as the error committed at Line 3 of the algorithm:

$$\epsilon_1 = c - (s_\ell + t_h) \tag{4}$$

and ϵ_2 as the error committed at Line 5:

$$\epsilon_2 = w - (t_\ell + v_\ell). \tag{5}$$

1. If $-x_h < y_h \le -x_h/2$. Sterbenz Lemma, applied to the first line of the algorithm, implies $s_h = x_h + y_h$, $s_\ell = 0$, and $c = \mathrm{RN}(t_h) = t_h$.

Define

$$\sigma = \begin{cases} 2 \text{ if } y_h \le -1, \\ 1 \text{ if } -1 < y_h \le -x_h/2. \end{cases}$$

We have $-x_h < y_h \le (1-\sigma) + \frac{x_h}{2}(\sigma - 2)$, so $0 \le x_h + y_h \le 1 + \sigma \cdot (\frac{x_h}{2} - 1) \le 1 - \sigma u$. Also, since x_h is a multiple of $2u$ and y_h is a multiple of σu, $s_h = x_h + y_h$ is a multiple of σu. Since s_h is nonzero, we finally obtain

$$\sigma u \le s_h \le 1 - \sigma u. \tag{6}$$

We have $|x_\ell| \le u$ and $|y_\ell| \le \frac{\sigma}{2} u$, so

$$|t_h| \le \left(1 + \frac{\sigma}{2}\right) u \quad \text{and} \quad |t_\ell| \le u^2. \tag{7}$$

From Equation (6), we deduce that the floating-point exponent of s_h is at least $-p + \sigma - 1$. From Equation (7), the floating-point exponent of $c = t_h$ is at most $-p + \sigma - 1$. Therefore, the Fast2Sum algorithm introduces no error at line 4 of the algorithm, which implies

$$v_h + v_\ell = s_h + c = s_h + t_h = x + y - t_\ell.$$

Equations (6) and (7) imply

$$|s_h + t_h| \le 1 + \left(1 - \frac{\sigma}{2}\right) u \le 1 + \frac{u}{2},$$

so $|v_h| \le 1$ and $|v_\ell| \le \frac{u}{2}$. From the bounds on $|t_\ell|$ and $|v_\ell|$, we obtain:

$$|\epsilon_2| \le \frac{1}{2}\mathrm{ulp}(t_\ell + v_\ell) \le \frac{1}{2}\mathrm{ulp}\left(u^2 + \frac{u}{2}\right) = \frac{u^2}{2} \tag{8}$$

and

$$|\epsilon_2| \le \frac{1}{2}\mathrm{ulp}\left[\frac{1}{2}\mathrm{ulp}(x_\ell + y_\ell) + \frac{1}{2}\mathrm{ulp}\left((x+y) + \frac{1}{2}\mathrm{ulp}(x_\ell + y_\ell)\right)\right]. \tag{9}$$

Lemma 2.1 and $|s_h| \ge \sigma u$ imply that either $s_h + t_h = 0$, or $|v_h| = |\mathrm{RN}(s_h + c)| = |\mathrm{RN}(s_h + t_h)| \ge \sigma u^2$. If $s_h + t_h = 0$, then $v_h = v_\ell = 0$ and the sequel of the proof is straightforward. Therefore, in the following, we assume $|v_h| \ge \sigma u^2$.

Now,

- If $|v_h| = \sigma u^2$, then $|v_\ell + t_\ell| \le u|v_h| + u^2 = \sigma u^3 + u^2$, which implies $|w| = |\mathrm{RN}(t_\ell + v_\ell)| \le \sigma u^2 = |v_h|$;
- If $|v_h| > \sigma u^2$, then, since v_h is a FP number, $|v_h|$ is larger than or equal to the FP number immediately above σu^2, which is $\sigma(1 + 2u)u^2$. Hence $|v_h| \ge \sigma u^2/(1-u)$, so $|v_h| \ge u \cdot |v_h| + \sigma u^2 \ge |v_\ell| + |t_\ell|$. So, $|w| = |\mathrm{RN}(t_\ell + v_\ell)| \le |v_h|$.

Therefore, in all cases, Fast2Sum introduces no error at line 6 of the algorithm, and we have

$$z_h + z_\ell = v_h + w = x + y + \epsilon_2. \tag{10}$$

Directly using Equation (10) and the bound $u^2/2$ on $|\epsilon_2|$ to get a relative error bound would result in a large bound, because $x + y$ may be small. However, when $x + y$ is very small, some simplification occurs thanks to Sterbenz Lemma. First, $x_h + y_h$ is a nonzero multiple of σu. Hence, since $|x_\ell + y_\ell| \le (1 + \frac{\sigma}{2})u$, we have $|x_\ell + y_\ell| \le \frac{3}{2}(x_h + y_h)$. Let us now consider the two possible cases:

- If $-\frac{3}{2}(x_h + y_h) \le x_\ell + y_\ell \le -\frac{1}{2}(x_h + y_h)$, which implies $-\frac{3}{2}s_h \le t_h \le -\frac{1}{2}s_h$, then Sterbenz lemma applies to the floating-point addition of s_h and $c = t_h$. Therefore line 4 of the algorithm results in $v_h = s_h$ and $v_\ell = 0$. An immediate consequence is $\epsilon_2 = 0$, so $z_h + z_\ell = v_h + w = x + y$: the computation of $x + y$ is errorless;

ALGORITHM 6: – **AccurateDWPlusDW**$(x_h, x_\ell, y_h, y_\ell)$. Calculation of $(x_h, x_\ell) + (y_h, y_\ell)$ in binary, precision-p, floating-point arithmetic.

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

Li et al. (2000, 2002) claim that
is upper bounded by $2 \cdot 2^{-106}$. Th

y_h
$y_\ell = -$

then the relative error of Algorithm 6 is

2.24999999999999956 ⋯

Note that this example is somehow "generic": In precision-p FP arith
$2^p - 1$, $x_\ell = -(2^p - 1) \cdot 2^{-p-1}$, $y_h = -(2^p - 5)/2$, and $y_\ell = -(2^p - 1) \cdot 2^{-p-3}$ leads to a relative error that is asymptotically equivalent (as p goes to infinity) to $2.25u^2$.

Now let us try to find a relative error bound. We are going to show the following result.

Theorem 3.1. *If $p \geq 3$, then the relative error of Algorithm 6 (AccurateDWPlusDW) is bounded by*

$$\frac{3u^2}{1-4u} = 3u^2 + 12u^3 + 48u^4 + \cdots, \quad (3)$$

which is less than $3u^2 + 13u^3$ as soon as $p \geq 6$.

Note that the conditions on p ($p \geq 3$ for the bound (3) to hold, $p \geq 6$ for the simplified bound $3u^2 + 13u^3$) are satisfied in all practical cases.

Proof. First

and ϵ_2 as the err

$$\epsilon_2 = w - (t_\ell + v_\ell). \quad (5)$$

1. If $-x_h < y_h \leq -x_h/2$. Sterbenz Lemma, applied to the first line of the algorithm, implies $s_h = x_h + y_h$, $s_\ell = 0$, and $c = \mathrm{RN}(t_h) = t_h$.

Define

$$\sigma = \begin{cases} 2 \text{ if } y_h \leq -1, \\ 1 \text{ if } -1 < y_h \leq -x_h/2. \end{cases}$$

We have $-x_h < y_h \leq (1-\sigma) + \frac{x_h}{2}(\sigma - 2)$, so $0 \leq x_h + y_h \leq 1 + \sigma \cdot (\frac{x_h}{2} - 1) \leq 1 - \sigma u$. Also, since x_h is a multiple of $2u$ and y_h is a multiple of σu, $s_h = x_h + y_h$ is a multiple of σu. Since s_h is nonzero, we finally obtain

$$\sigma u \leq s_h \leq 1 - \sigma u. \quad (6)$$

have $|x_\ell| \leq u$ and $|y_\ell| \leq \frac{\sigma}{2}u$, so

$$\leq \left(1 + \frac{\sigma}{2}\right)u \quad \text{and} \quad |t_\ell| \leq u^2. \quad (7)$$

g-point exponent of s_h is at least $-p + \sigma - 1$. From is at most $-p + \sigma - 1$. Therefore, the Fast2Sum hm, which implies

$= x + y - t_\ell$.

$\left(1 - \frac{\sigma}{2}\right)u \leq 1 + \frac{u}{2}$,

ounds on $|t_\ell|$ and $|v_\ell|$, we obtain:

$$|\epsilon_2| \leq \frac{1}{2}\mathrm{ulp}(t_\ell + v_\ell) \leq \frac{1}{2}\mathrm{ulp}\left(u^2 + \frac{u}{2}\right) = \frac{u^2}{2} \quad (8)$$

and

$$|\epsilon_2| \leq \frac{1}{2}\mathrm{ulp}\left[\frac{1}{2}\mathrm{ulp}(x_\ell + y_\ell) + \frac{1}{2}\mathrm{ulp}\left((x+y) + \frac{1}{2}\mathrm{ulp}(x_\ell + y_\ell)\right)\right]. \quad (9)$$

Lemma 2.1 and $|s_h| \geq \sigma u$ imply that either $s_h + t_h = 0$, or $|v_h| = |\mathrm{RN}(s_h + c)| = |\mathrm{RN}(s_h + t_h)| \geq \sigma u^2$. If $s_h + t_h = 0$, then $v_h = v_\ell = 0$ and the sequel of the proof is straightforward. Therefore, in the following, we assume $|v_h| \geq \sigma u^2$.

Now,

- If $|v_h| = \sigma u^2$, then $|v_\ell + t_\ell| \leq u|v_h| + u^2 = \sigma u^3 + u^2$, which implies $|w| = |\mathrm{RN}(t_\ell + v_\ell)| \leq \sigma u^2 = |v_h|$;
- If $|v_h| > \sigma u^2$, then, since v_h is a FP number, $|v_h|$ is larger than or equal to the FP number immediately above σu^2, which is $\sigma(1 + 2u)u^2$. Hence $|v_h| \geq \sigma u^2/(1-u)$, so $|v_h| \geq u \cdot |v_h| + u^2 \geq |v_\ell| + |t_\ell|$. So, $|w| = |\mathrm{RN}(t_\ell + v_\ell)| \leq |v_h|$.

ll cases, Fast2Sum introduces no error at line 6 of the algorithm, and we have

$$z_h + z_\ell = v_h + w = x + y + \epsilon_2. \quad (10)$$

y using Equation (10) and the bound $u^2/2$ on $|\epsilon_2|$ to get a relative error bound would result in a large bound, because $x + y$ may be small. However, when $x + y$ is very small, some simplification occurs thanks to Sterbenz Lemma. First, $x_h + y_h$ is a nonzero multiple of σu. Hence, since $|x_\ell + y_\ell| \leq (1 + \frac{\sigma}{2})u$, we have $|x_\ell + y_\ell| \leq \frac{3}{2}(x_h + y_h)$. Let us now consider the two possible cases:

- If $-\frac{3}{2}(x_h + y_h) \leq x_\ell + y_\ell \leq -\frac{1}{2}(x_h + y_h)$, which implies $-\frac{3}{2}s_h \leq t_h \leq -\frac{1}{2}s_h$, then Sterbenz lemma applies to the floating-point addition of s_h and $c = t_h$. Therefore line 4 of the algorithm results in $v_h = s_h$ and $v_\ell = 0$. An immediate consequence is $\epsilon_2 = 0$, so $z_h + z_\ell = v_h + w = x + y$: the computation of $x + y$ is errorless;

ither $s_h + t_h = 0$, or $|v_h| = |\mathrm{RN}(s_h + c)| = |\mathrm{RN}$

d the sequel of the proof is straightforward. Th

- If $-\frac{3}{2}(x_h + y_h) \leq x_\ell + y_\ell \leq -\frac{1}{2}(x_h + y_h)$

applies to the floating-p

- If $-\frac{1}{2}(x_h + y_h) < x_\ell + y_\ell \le \frac{3}{2}(x_h + y_h)$, then $\frac{5}{2}(x_\ell + y_\ell) \le \frac{3}{2}(x_h + y_h + x_\ell + y_\ell) = \frac{3}{2}(x + y)$, and $-\frac{1}{2}(x + y) < \frac{1}{2}(x_\ell + y_\ell)$. Hence, $|x_\ell + y_\ell| < |x + y|$, so $\mathrm{ulp}(x_\ell + y_\ell) \le \mathrm{ulp}(x + y)$. Combined with Equation (9), this gives

$$|\epsilon_2| \le \frac{1}{2}\mathrm{ulp}\left(\frac{3}{2}\mathrm{ulp}(x+y)\right) \le 2^{-p}\mathrm{ulp}(x+y) \le 2 \cdot 2^{-2p} \cdot (x+y).$$

2. If $-x_h/2 < y_h \le x_h$

Notice that we have $x_h/2 < x_h + y_h \le 2x_h$, so $x_h/2 \le s_h \le 2x_h$. Also notice that we have $|x_\ell| \le u$.

- If $\frac{1}{2} < x_h + y_h \le 2 - 4u$. Define

We have

Elementary calculus shows that fo

When $\sigma = 1$, we h
$x_h \le 2 - 2u$ implies $|y_\ell$
$(1 + \sigma/2)u$, therefore

The bound (3) is probabl

$$|t_h| \le \left(1 + \frac{\sigma}{2}\right)u$$

Now, $|s_\ell + t_h| \le (1 + \sigma)u$, so

$$|c| \le (1 + \sigma)u \quad \text{and} \quad |\epsilon_1| \le \sigma u^2. \tag{13}$$

Since $s_h \ge 1/2$ and $|c| \le 3u$, if $p \ge 3$, then Algorithm Fast2Sum introduces no error at line 4 of the algorithm, that is,

$$v_h + v_\ell = s_h + c.$$

Therefore $|v_h + v_\ell| = |s_h + c| \le \sigma(1 - 2u) + (1 + \sigma)u \le \sigma$. This implies

$$|v_h| \le \sigma \quad \text{and} \quad |v_\ell| \le \frac{\sigma}{2}u. \tag{14}$$

Thus $|t_\ell + v_\ell| \le u^2 + \frac{\sigma}{2}u$, so

$$|w| \le \frac{\sigma}{2}u + u^2 \quad \text{and} \quad |\epsilon_2| \le \frac{\sigma}{2}u^2. \tag{15}$$

From Equations (11) and (13), we deduce $s_h + c \ge \frac{\sigma}{2} - u(2\sigma + 1)$, so $|v_h| \ge \frac{\sigma}{2} - u(2\sigma + 1)$. If $p \ge 3$, then $|v_h| \ge |w|$, so Algorithm Fast2Sum introduces no error at line 6 of the algorithm, that is, $z_h + z_\ell = v_h + w$.

Therefore,

$$z_h + z_\ell = x + y + \eta,$$

with $|\eta| = |\epsilon_1 + \epsilon_2| \le \frac{3\sigma}{2}u^2$. Since

$$x + y \ge (x_h - u) + (y_h - u/2) > \begin{cases} \frac{1}{2} - \frac{3}{2}u & \text{if} \quad \sigma = 1, \\ 1 - 4u & \text{if} \quad \sigma = 2, \end{cases}$$

the relative error $|\eta|/(x + y)$ is upper bounded by

$$\frac{3u^2}{1 - 4u}.$$

- If $2 - 4u < x_h + y_h \le 2x_h$, then $2 - 4u \le s_h \le \mathrm{RN}(2x_h) = 2x_h \le 4 - 4u$ and $|s_\ell| \le 2u$. We have

$$t_h + t_\ell = x_\ell + y_\ell,$$

with $|x_\ell + y_\ell| \le 2u$, hence $|t_h| \le 2u$, and $|t_\ell| \le u^2$. Now, $|s_\ell + t_h| \le 4u$, so $|c| \le 4u$, and $|\epsilon_1| \le 2u^2$. Since $s_h \ge 2 - 4u$ and $|c| \le 4u$, if $p \ge 3$, then Algorithm Fast2Sum introduces no error at line 4 of the algorithm. Therefore,

$$v_h + v_\ell = s_h + c \le 4 - 4u + 4u = 4,$$

so $v_h \le 4$ and $|v_\ell| \le 2u$. Thus, $|t_\ell + v_\ell| \le 2u + u^2$. Hence, either $|t_\ell + v_\ell| < 2u$ and $|\epsilon_2| \le$ ^{1}ulp$(t_\ell + v_\ell) \le u^2$, or $2u \le t_\ell + v_\ell \le 2u + u^2$, in which case $w = \mathrm{RN}(t_\ell + v_\ell) = 2u$ and $\le u^2$. In all cases, $|\epsilon_2| \le u^2$. Also, $s_h \ge 2 - 4u$ and $|c| \le 4u$ imply $v_h \ge 2 - 8u$, and $| \le 2u + u^2$ implies $|w| \le 2u$. Hence if $p \ge 3$, then Algorithm Fast2Sum introduces at line 6 of the algorithm.
is gives

$$z_h + z_\ell = v_h + w = x + y + \eta,$$

ith $|\eta| = |\epsilon_1 + \epsilon_2| \le 3u^2$.

Since $x + y \ge (x_h - u) + (y_h - u) > 2 - 6u$, the relative error $|\eta|/(x + y)$ is upper bounded by

$$\frac{3u^2}{2 - 6u},$$

The largest bound obtained in the various cases we have analyzed is

$$\frac{3u^2}{1 - 4u}.$$

Elementary calculus shows that for $u \in [0, 1/64]$ (i.e., $p \ge 6$) this is always less than $3u^2 + 13u^3$. □

The bound (3) is probably not optimal. The largest relative error we have obtain through many tests is around $2.25 \times 2^{-2p} = 2.25u^2$. An example is the input values given in Equation (2), for which, with $p = 53$ (binary64 arithmetic), we obtain a relative error equal to $2.24999999999999956 \cdots \times 2^{-106}$.

DW+DW: "accurate version"

So the theorem gives an error bound

$$\frac{3u^2}{1-4u} \simeq 3u^2 \ldots$$

As said before, that theorem has an interesting history:

- the authors of the first paper where a bound was given (in 2000) claimed (without published proof) that the relative error was always $\leq 2u^2$ (in binary64 arithmetic);
- when trying (without success) to prove their bound, we found an example with error $\approx 2.25u^2$;
- we finally proved the theorem, and Laurence Rideau (Inria Nice) started to write a formal proof in Coq;
- of course, that led to finding a (minor) flaw in our proof...

DW+DW: "accurate version"

- fortunately the flaw was quickly corrected (before final publication of the paper... Phew)!
- still, the gap between worst case found ($\approx 2.25u^2$) and the bound ($\approx 3u^2$) was frustrating, so I spent months trying to improve the theorem...
- and of course this could not be done: it was the worst case that needed spending time!
- we finally found that with

$$
\begin{aligned}
x_h &= 1 \\
x_\ell &= u - u^2 \\
y_h &= -\frac{1}{2} + \frac{u}{2} \\
y_\ell &= -\frac{u^2}{2} + u^3.
\end{aligned}
$$

error $\frac{3u^2-2u^3}{1+3u-3u^2+2u^3}$ is attained. With $p = 53$ (binary64 arithmetic), gives error $2.99999999999999877875 \cdots \times u^2$.

DW+DW: “accurate version”

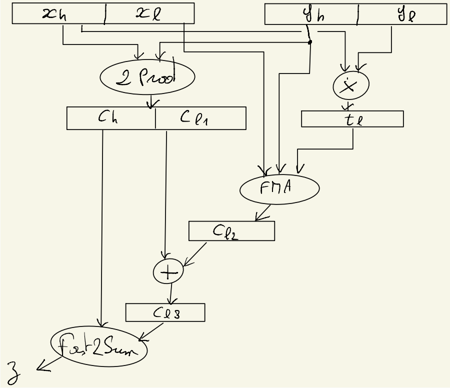
- We suspect the initial authors hinted their claimed bound by performing zillions of random tests
- in this domain, the worst cases are extremely unlikely: you must build them. Almost impossible to find them by chance.

DW $\times$ DW

- Product $z = (z_h, z_\ell)$ of two DW numbers $x = (x_h, x_\ell)$ and $y = (y_h, y_\ell)$;
- several algorithms $\rightarrow$ tradeoff speed/accuracy. We just give one of them.

DWTimesDW

$(c_h, c_{\ell 1}) \leftarrow \text{2Prod}(x_h, y_h)$
$t_\ell \leftarrow \text{RN}\,(x_h \cdot y_\ell)$
$c_{\ell 2} \leftarrow \text{RN}\,(t_\ell + x_\ell y_h)$
$c_{\ell 3} \leftarrow \text{RN}\,(c_{\ell 1} + c_{\ell 2})$
$(z_h, z_\ell) \leftarrow \text{Fast2Sum}(c_h, c_{\ell 3})$
return (z_h, z_ℓ)



DW $\times$ DW

We have

Theorem 7 (Error bound for Algorithm DWTimesDW)

If $p \geq 5$, the relative error of Algorithm DWTimesDW2 is less than or equal to

$$\frac{5u^2}{(1+u)^2} < 5u^2.$$

and that theorem too has an interesting history!

- initial bound $6u^2$;
- again, we tried formal proof... and it turned out the proof was based on a wrong lemma.

DW $\times$ DW

- after a few nights of very bad sleep, we found a turn-around. . . that also improved the bound !
- no proof of asymptotic optimality, but in binary64 arithmetic, we have examples with error $> 4.98u^2$;
- (real consolation or lame excuse?) without the flaw, we would never have found the better bound;
- without the formal proof effort, the error would probably have remained unnoticed (in this case, without serious consequence since the property was true anyway, but. . .).

Conclusion

- (almost) fully specified arithmetic: one can prove properties of (small enough) programs, and build algorithms;
- ongoing effort for also standardizing a kernel of math functions (at least exp, sin, cos, log);
- all of numerical computing is built from computer arithmetic: it must be reliable;
- for some algorithms (e.g., DW arithmetic, FP division algorithms) the "paper proofs" are terrible: use of formal proof and computer algebra.