Exact computations with an arithmetic known to be approximate

MaGiX@LiX conference – 2011

Jean-Michel Muller

CNRS - Laboratoire LIP
(CNRS-INRIA-ENS Lyon-Université de Lyon)

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

Floating-Point Arithmetic

- bad reputation ;
- used everywhere in scientific calculation ;
- "scientific notation" of numbers :

$$6.02214179 \times 10^{23}$$

The number 6.02214179 is the significand (or *mantissa*), and the number 23 is the exponent.

- generalization to radix β : $x = m_x \cdot \beta^{e_x}$, where m_x is represented in radix β. Almost always, β is 2 or 10 ;

Floating-Point Arithmetic

- bad reputation ;
- used everywhere in scientific calculation ;
- "scientific notation" of numbers :

$$6.02214179 \times 10^{23}$$

  The number 6.02214179 is the significand (or *mantissa*), and the number 23 is the exponent.

- generalization to radix β : $x = m_x \cdot \beta^{e_x}$, where m_x is represented in radix β. Almost always, β is 2 or 10 ;

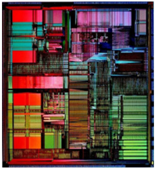
But there is more to say about this. . . later

Desirable properties

- Speed : tomorrow's weather must be computed in less than 24 hours ;
- Accuracy, Range ;
- "Size" : silicon area and/or code size ;
- Power consumption ;
- Portability : the programs we write on a given system must run on different systems without requiring huge modifications ;
- Easiness of implementation and use : If a given arithmetic is too arcane, nobody will use it.

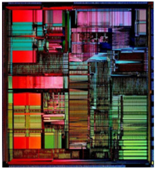
Some can do a very poor job...

- 1994 : Pentium 1 division bug : 8391667/12582905 gave $0.666869\cdots$ instead of $0.666910\cdots$;



Some can do a very poor job. . .

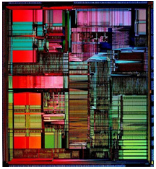
- 1994 : Pentium 1 division bug : 8391667/12582905 gave 0.666869 $\cdots$ instead of 0.666910 $\cdots$;



- Maple version 6.0. Enter 214748364810, you get 10. Note that $2147483648 = 2^{31}$;

Some can do a very poor job. . .

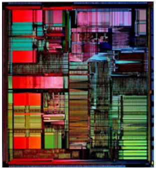
- 1994 : Pentium 1 division bug : 8391667/12582905 gave 0.666869 · · · instead of 0.666910 · · · ;



- Maple version 6.0. Enter 214748364810, you get 10. Note that $2147483648 = 2^{31}$;
- Excel'2007 (first releases), compute $65535 - 2^{-37}$, you get 100000 ;

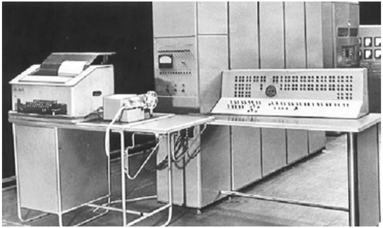
Some can do a very poor job...

- 1994 : Pentium 1 division bug : 8391667/12582905 gave 0.666869 $\cdots$ instead of 0.666910 $\cdots$;



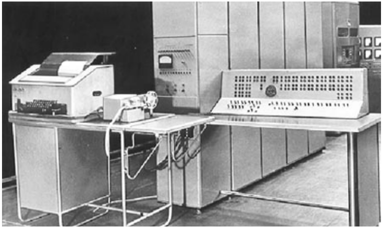
- Maple version 6.0. Enter 214748364810, you get 10. Note that $2147483648 = 2^{31}$;
- Excel'2007 (first releases), compute $65535 - 2^{-37}$, you get 100000 ;
- November 1998, USS Yorktown warship, somebody erroneously entered a «zero» on a keyboard → division by 0 → series of errors → the propulsion system stopped.

Some strange things



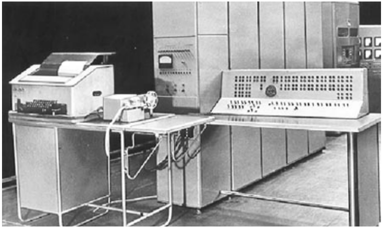
- Setun Computer, Moscow University, 1958. 50 copies ;

Some strange things



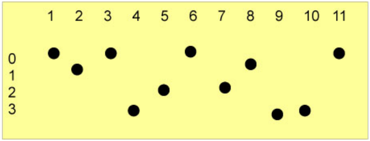
- Setun Computer, Moscow University, 1958. 50 copies ;
- radix 3 and digits -1, 0 and 1. Numbers represented using 18 « trits » ;

Some strange things



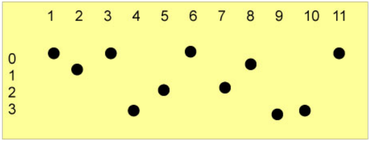
- Setun Computer, Moscow University, 1958. 50 copies ;
- radix 3 and digits -1, 0 and 1. Numbers represented using 18 « trits » ;
- idea : radix β, n digits, you want to represent around M different numbers. “Cost” : $\beta \times n$.

Some strange things



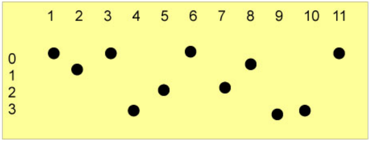
- idea : radix β, n digits, you want to represent around M different numbers. "Cost" : $\beta \times n$. $\rightarrow$ punched card area ;

Some strange things



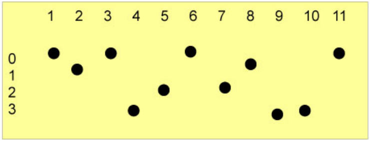
- idea : radix β, n digits, you want to represent around M different numbers. "Cost" : $\beta \times n$. $\rightarrow$ punched card area ;
- if we wish to represent M numbers, minimize $\beta \times n$ knowing that $\beta^n \geq M$.

Some strange things



- idea : radix β, n digits, you want to represent around M different numbers. "Cost" : $\beta \times n$. $\rightarrow$ punched card area ;
- if we wish to represent M numbers, minimize $\beta \times n$ knowing that $\beta^n \geq M$.
- With real variables $\beta = e = 2.718\ldots \approx 3\ldots$ what is the "best" (integral) radix ?

Some strange things

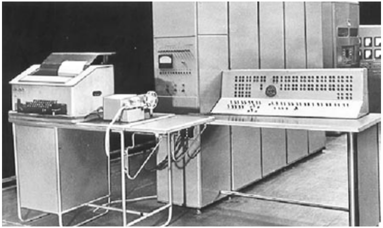


- idea : radix β, n digits, you want to represent around M different numbers. "Cost" : $\beta \times n$. $\rightarrow$ punched card area ;
- if we wish to represent M numbers, minimize $\beta \times n$ knowing that $\beta^n \geq M$.
- With real variables $\beta = e = 2.718\ldots \approx 3\ldots$ what is the "best" (integral) radix ?
- as soon as :

$$M \geq e^{\frac{5}{(2/\ln(2))-(3/\ln(3))}} \approx 1.09 \times 10^{14}$$

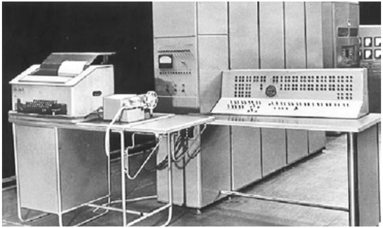
it is always 3

Yes, but...



Building circuits with three-valued logic turned out to be very difficult...

Yes, but. . .



Building circuits with three-valued logic turned out to be very difficult. . .

. . . so that in practice, each “trit” was represented by two bits.

Floating-Point System

Parameters :

$$\begin{cases} \text{radix (or base)} & \beta \geq 2 \text{ (will be 2 in this presentation)} \\ \text{precision} & p \geq 1 \\ \text{extremal exponents} & e_{\min}, e_{\max}, \end{cases}$$

A finite FP number x is represented by 2 integers :

- integral significand : M, $|M| \leq \beta^p - 1$;
- exponent e, $e_{\min} \leq e \leq e_{\max}$.

such that

$$x = M \times \beta^{e+1-p}$$

with $|M|$ largest under these constraints ($\rightarrow |M| \geq \beta^{p-1}$, unless $e = e_{\min}$).
(Real) significand of x : the number $m = M \times \beta^{1-p}$, so that $x = m \times \beta^e$.

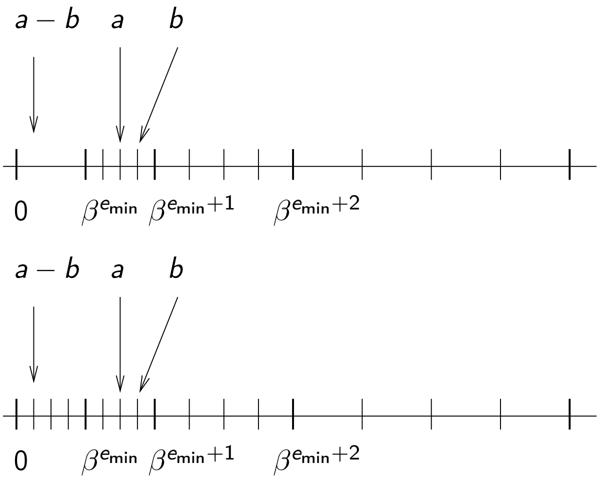
Normal and subnormal numbers

- normal number : of absolute value $\geq \beta^{e_{\min}}$. The absolute value of its integral significand is $\geq \beta^{p-1}$;
- subnormal number : of absolute value $< \beta^{e_{\min}}$. The absolute value of its integral significand is $< \beta^{p-1}$.

normality/subnormality encoded in the exponent.

Radix 2 : the leftmost bit of the significand of a normal number is a "1" $\rightarrow$ no need to store it (implicit 1 convention).

Subnormal numbers difficult to implement efficiently, but...



$a \neq b$ equivalent to "computed $a - b \neq 0$".

IEEE-754 Standard for FP Arithmetic (1985 and 2008)

- 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 ;
- specification of operations and conversions ;
- exception handling (max+1, 1/0, $\sqrt{-2}$, 0/0, etc.) ;
- new version of the standard : August 2008.

Correct rounding

Definition 1 (Correct rounding)

The user chooses a *rounding function* among :

- round toward $-\infty$: $\mathrm{RD}(x)$ is the largest FP number $\leq x$;
- round toward $+\infty$: $\mathrm{RU}(x)$ is the smallest FP number $\geq x$;
- round toward zero : $\mathrm{RZ}(x)$ is equal to $\mathrm{RD}(x)$ if $x \geq 0$, and to $\mathrm{RU}(x)$ if $x \leq 0$;
- round to nearest : $\mathrm{RN}(x) =$ FP number closest to x. If exactly halfway between two consecutive FP numbers : the one whose integral significand is even (default mode)

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

Correct rounding

IEEE-754 (1985) : Correct rounding for $+$, $-$, $\times$, $\div$, $\sqrt{}$ and some conversions. Advantages :

- if the result of an operation is exactly representable, we get it ;
- if we just use the 4 arith. operations and $\sqrt{}$, deterministic arithmetic : one can elaborate algorithms and proofs that use the specifications ;
- accuracy and portability are improved ;
- playing with rounding towards $+\infty$ and $-\infty$ $\rightarrow$ certain lower and upper bounds : interval arithmetic.

FP arithmetic becomes a structure in itself, that can be studied.

First example : Strebenz Lemma

Lemma 2 (Sterbenz)

Radix β,with subnormal numbers available. 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, in any rounding mode).

Proof : straightforward using the notation $x = M \times \beta^{e+1-p}$.

Error of FP addition (Møller, Knuth, Dekker)

First result : representability. RN (x) is x rounded to nearest.

Lemma 3

Let a and b be two FP numbers. Let

$$s = RN(a + b)$$

and

$$r = (a + b) - s.$$

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

Same thing for $\times$.

Error of FP addition (Møller, Knuth, Dekker)

Proof : Assume $|a| \geq |b|$,

1. s is "the" FP number nearest $a + b \rightarrow$ it is closest to $a + b$ than a is. Hence $|(a + b) - s| \leq |(a + b) - a|$, therefore

$$|r| \leq |b|.$$

Error of FP addition (Møller, Knuth, Dekker)

Proof : Assume $|a| \geq |b|$,

1. s is "the" FP number nearest $a+b \longrightarrow$ it is closest to $a+b$ than a is. Hence $|(a+b)-s| \leq |(a+b)-a|$, therefore
$$|r| \leq |b|.$$
2. denote $a = M_a \times \beta^{e_a-p+1}$ and $b = M_b \times \beta^{e_b-p+1}$, with $|M_a|, |M_b| \leq \beta^p - 1$, and $e_a \geq e_b$.
$a+b$ is multiple of $\beta^{e_b-p+1} \Rightarrow s$ and r are multiple of β^{e_b-p+1} too
$\Rightarrow \exists R \in \mathbb{Z}$ s.t.
$$r = R \times \beta^{e_b-p+1}$$
but, $|r| \leq |b| \Rightarrow |R| \leq |M_b| \leq \beta^p - 1 \Rightarrow r$ is a FP number.

Get r : the fast2sum algorithm (Dekker)

Theorem 4 (Fast2Sum (Dekker))

$\beta \leq 3$, subnormal numbers available. 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.

Proof in the case $\beta = 2$

$$\begin{aligned} s &= \mathrm{RN}\,(a+b) \\ z &= \mathrm{RN}\,(s-a) \\ t &= \mathrm{RN}\,(b-z) \end{aligned}$$

- if a and b have same sign, then $|a| \leq |a+b| \leq |2a|$ hence (radix $2 \rightarrow 2a$ is a FP number, rounding is increasing) $|a| \leq |s| \leq |2a| \rightarrow$ (Sterbenz Lemma) $z = s - a$. Since $r = (a+b) - s$ is a FPN and $b - z = r$, we find $\mathrm{RN}\,(b-z) = r$.
- if a and b have opposite signs then
  1. either $|b| \geq \frac{1}{2}|a|$, which implies (Sterbenz Lemma) $a+b$ is a FPN, thus $s = a+b$, $z = b$ and $t = 0$;
  2. or $|b| < \frac{1}{2}|a|$, which implies $|a+b| > \frac{1}{2}|a|$, hence $s \geq \frac{1}{2}|a|$ (radix $2 \rightarrow \frac{1}{2}a$ is a FPN, and rounding is increasing), thus (Sterbenz Lemma) $z = \mathrm{RN}\,(s-a) = s - a = b - r$. Since $r = (a+b) - s$ is a FPN and $b - z = r$,we get $\mathrm{RN}\,(b-z) = r$.

The TwoSum Algorithm (Møller-Knuth)

- no need to compare a and b ;

The TwoSum Algorithm (Møller-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.

Algorithm 2 (TwoSum)

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

The TwoSum Algorithm (Møller-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.

Algorithm 2 (TwoSum)

$$
\begin{aligned}
s &\leftarrow RN(a+b) \\
a' &\leftarrow RN(s-b) \\
b' &\leftarrow RN(s-a') \\
\delta_a &\leftarrow RN(a-a') \\
\delta_b &\leftarrow RN(b-b') \\
r &\leftarrow RN(\delta_a+\delta_b)
\end{aligned}
$$

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

The TwoSum Algorithm (Møller-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.

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

Boldo et al : (formal proof) in radix 2, underflow does not hinder the result (overflow does).

The TwoSum Algorithm (Møller-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.

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

Boldo et al : (formal proof) in radix 2, underflow does not hinder the result (overflow does).

TwoSum is optimal, in a way we are going to explain.

TwoSum is optimal

Assume an algorithm satisfies :

- it is without tests or min/max instructions ;
- it only uses rounded to nearest additions/subtractions : at step i we compute $\mathrm{RN}(u+v)$ or $\mathrm{RN}(u-v)$ where u and v are input variables or previously computed variables.

If that algorithm algorithm always computes the same results as 2Sum, then it uses at least 6 *additions/subtractions (i.e., as much as 2Sum).*

- proof : most inelegant proof award ;

TwoSum is optimal

Assume an algorithm satisfies :

- it is without tests or min/max instructions ;
- it only uses rounded to nearest additions/subtractions : at step i we compute $\text{RN}(u+v)$ or $\text{RN}(u-v)$ where u and v are input variables or previously computed variables.

If that algorithm algorithm always computes the same results as 2Sum, then it uses at least 6 *additions/subtractions (i.e., as much as 2Sum).*

- proof : most inelegant proof award ;
- 480756 algorithms with 5 operations (after suppressing the most obvious symmetries) ;

TwoSum is optimal

Assume an algorithm satisfies :

- it is without tests or min/max instructions ;
- it only uses rounded to nearest additions/subtractions : at step i we compute $\text{RN}(u+v)$ or $\text{RN}(u-v)$ where u and v are input variables or previously computed variables.

If that algorithm algorithm always computes the same results as 2Sum, then it uses at least 6 *additions/subtractions (i.e., as much as 2Sum).*

- proof : most inelegant proof award ;
- 480756 algorithms with 5 operations (after suppressing the most obvious symmetries) ;
- each of them tried with 2 well-chosen pairs of input values.

Adding n numbers : $x_1 + x_2 + x_3 + \cdots + x_n$

Pichat, Ogita, Rump, and Oishi's algorithm RN : rounding to nearest

Algorithm 3

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

Adding n numbers : $x_1 + x_2 + x_3 + \cdots + x_n$

Theorem 5 (Ogita, Rump and Oishi)

Let

$$\mathbf{u} = \frac{1}{2}\beta^{-p+1}$$

and

$$\gamma_n = \frac{n\mathbf{u}}{1 - n\mathbf{u}}.$$

Applying the algorithm of P.,O., R., and O. to x_i, $1 \le i \le n$, *and if* $n\mathbf{u} < 1$, *then, even in case of underflow (but without overflow), the final result* σ *satisfies*

$$\left|\sigma - \sum_{i=1}^{n} x_i\right| \le \mathbf{u}\left|\sum_{i=1}^{n} x_i\right| + \gamma_{n-1}^2 \sum_{i=1}^{n} |x_i|.$$

ULP : Unit in the Last Place

Radix β, precision p. In the following, $x \in \mathbb{R}$ and X is a FP number that approximates x.

Definition 6

If $|x| \in [\beta^e, \beta^{e+1})$ then $\text{ulp}(x) = \beta^{\max(e, e_{\min}) - p + 1}$.

Property 1

In radix 2,

$$|X - x| < \frac{1}{2}\, ulp(x) \Rightarrow X = RN(x).$$

Not true in radix ≥ 3. Not true (even in radix 2) if we replace $\text{ulp}(x)$ by $\text{ulp}(X)$.

ULP : Unit in the Last Place

Property 2

In any radix,

$$X = RN(x) \Rightarrow |X - x| \leq \frac{1}{2} ulp(x).$$

Property 3

In any radix,

$$X = RN(x) \Rightarrow |X - x| \leq \frac{1}{2} ulp(X).$$

Division using Newton-Raphson iteration and an FMA

Simplified version of an algorithm used on the Intel/HP Itanium. Precision p, radix 2. To simplify, assume we compute $1/b$. We assume $1 \leq b < 2$ (significands of normal FP numbers).

- Newton-Raphson iteration to compute $1/b$:

$$y_{n+1} = y_n(2 - by_n)$$

- we lookup $y_0 \approx 1/b$ in a table addressed by the first (typically from 6 to 10) bits of b ;
- FMA : computes $\text{RN}\,(xy + z)$ (RS 6000, Power PC, Itanium. . .) ;
- the NR iteration is decomposed into 2 FMA instructions :

$$\begin{cases} e_n & = & \text{RN}\,(1 - by_n) \\ y_{n+1} & = & \text{RN}\,(y_n + e_n y_n) \end{cases}$$

Notice that $e_{n+1} \approx e_n^2$.

Property 4

If

$$\left|\frac{1}{b} - y_n\right| < \alpha 2^{-k},$$

where $1/2 < \alpha \leq 1$ *and* $k \geq 1$, *then*

$$\begin{array}{rcl} \left|\frac{1}{b} - y_{n+1}\right| & < & b\left(\frac{1}{b} - y_n\right)^2 + 2^{-k-p} + 2^{-p-1} \\ & < & 2^{-2k+1}\alpha^2 + 2^{-k-p} + 2^{-p-1} \end{array}$$

$\Rightarrow$ it seems that we can get arbitrarily closer to error 2^{-p-1} (i.e., $1/2\,\mathrm{ulp}\,(1/b)$), without being able to show a bound below $1/2\,\mathrm{ulp}\,(1/b)$.

Example : double precision of the IEEE-754 standard

Assume $p = 53$ and $|y_0 - \frac{1}{b}| < 2^{-8}$ (small table), we find

- $|y_1 - 1/b| < 0.501 \times 2^{-14}$
- $|y_2 - 1/b| < 0.51 \times 2^{-28}$
- $|y_3 - 1/b| < 0.57 \times 2^{-53} = 0.57\,\mathrm{ulp}\,(1/b)$

Going further ?

Property 5

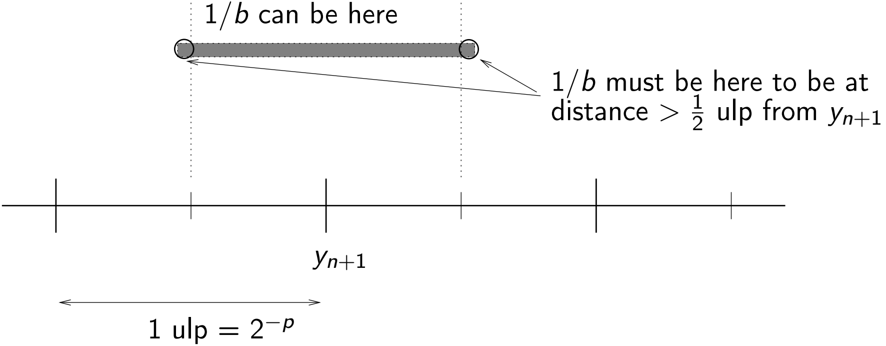
When y_n approximates $1/b$ within error $< 1\,ulp\,(1/b) = 2^{-p}$, then, since b is multiple of 2^{-p+1} and y_n is multiple of 2^{-p}, $1 - by_n$ is multiple of 2^{-2p+1}.

But $|1 - by_n| < 2^{-p+1} \rightarrow 1 - by_n$ is exactly representable in FP arithmetic with a p-bit precision $\rightarrow$ exactly computed by one FMA.

$$\Rightarrow \left|\frac{1}{b} - y_{n+1}\right| < b\left(\frac{1}{b} - y_n\right)^2 + 2^{-p-1}.$$

$$\left|y_n - \frac{1}{b}\right| < \alpha 2^{-p} \Rightarrow \left|y_{n+1} - \frac{1}{b}\right| < b\alpha^2 2^{-2p} + 2^{-p-1}$$

(assuming $\alpha < 1$)



What can be deduced ?

- to be at distance $> 1/2$ ulp from y_{n+1}, $1/b$ must be within $b\alpha^2 2^{-2p} < b2^{-2p}$ from the midpoint of two consecutive FP numbers ;
- implies that distance between y_n and $1/b$ has the form $2^{-p-1} + \epsilon$, with $|\epsilon| < b2^{-2p}$;
- implies $\alpha < \frac{1}{2} + b2^{-p}$ hence

$$\left| y_{n+1} - \frac{1}{b} \right| < \left(\frac{1}{2} + b2^{-p} \right)^2 b2^{-2p} + 2^{-p-1}$$

- so, to be at distance $> 1/2$ ulp from y_{n+1}, $1/b$ must be within $\left(\frac{1}{2} + b2^{-p}\right)^2 b2^{-2p}$ from the midpoint of two consecutive FP numbers.

- b is a FP number between 1 et 2 $\Rightarrow b = B/2^{p-1}$ where $B \in \mathbb{N}$, $2^{p-1} < B \leq 2^p - 1$;
- the midpoint of two consecutive FP numbers in the neighborhood of $1/b$ has the form $g = (2G+1)/2^{p+1}$ where $G \in \mathbb{N}$, $2^{p-1} \leq G < 2^p - 1$;
- we deduce

$$\left| g - \frac{1}{b} \right| = \left| \frac{2BG + B - 2^{2p}}{B.2^{p+1}} \right|$$

- the distance between $1/b$ and the midpoint of two consecutive FP numbers is a multiple of $1/(B.2^{p+1}) = 2^{-2p}/b$. It is $\neq 0$

Distance between $\frac{1}{b}$ and g, when $\left|\frac{1}{b} - y_{n+1}\right| > \frac{1}{2}\,\mathrm{ulp}\left(\frac{1}{b}\right)$

- has the form $k2^{-2p}/b$, $k \in \mathbb{Z}$, $k \neq 0$;
- we must have

$$\frac{|k| \cdot 2^{-2p}}{b} < \left(\frac{1}{2} + b2^{-p}\right)^2 b2^{-2p}$$

  therefore

$$|k| < \left(\frac{1}{2} + b2^{-p}\right)^2 b^2$$

- since $b < 2$, as soon as $p \geq 4$, the only solution is $|k| = 1$;
- moreover, for $|k| = 1$, elementary manipulation shows that the only possible solution is

$$b = 2 - 2^{-p+1}.$$

How do we procede ?

- we want
$$B = 2^p - 1,$$
$$2^{p-1} \leq G \leq 2^p - 1$$
$$B(2G+1) = 2^{2p} \pm 1$$
Only one solution : $B = 2^p - 1$ and $G = 2^{p-1}$: comes from $2^{2p} - 1 = (2^p - 1)(2^p + 1)$;
- except for that B (thus for the corresponding value $b = B/2^{p-1}$ of b), we are certain that $y_{n+1} = \text{RN}\,(1/b)$;
- for $B = 2^p - 1$: we try the algorithm with the two values of y_n within one ulp from $1/b$ (i.e. $1/2$ and $1/2 + 2^{-p}$). In practice, it works (otherwise : do dirty things).

Application : double precision ($p = 53$)

We start from y_0 such that $|y_0 - \frac{1}{b}| < 2^{-8}$. We compute :

$$\begin{array}{rcll} e_0 & = & \mathrm{RN}\left(1 - by_0\right) & \\ y_1 & = & \mathrm{RN}\left(y_0 + e_0 y_0\right) & \\ e_1 & = & \mathrm{RN}\left(1 - by_1\right) & \\ y_2 & = & \mathrm{RN}\left(y_1 + e_1 y_1\right) & \\ e_2 & = & \mathrm{RN}\left(1 - by_1\right) & \\ y_3 & = & \mathrm{RN}\left(y_2 + e_2 y_2\right) & \text{error} \leq 0.57 \text{ ulps} \\ e_3 & = & \mathrm{RN}\left(1 - by_2\right) & \\ y_4 & = & \mathrm{RN}\left(y_3 + e_3 y_3\right) & 1/b \text{ rounded to nearest} \end{array}$$

In practice : two iterations

Markstein iterations

$$\begin{cases} e_n & = & \mathrm{RN}\,(1 - b y_n) \\ y_{n+1} & = & \mathrm{RN}\,(y_n + e_n y_n) \end{cases}$$

More accurate ("self correcting"), sequential

Goldschmidt iterations

$$\begin{cases} e_{n+1} & = & \mathrm{RN}\,(e_n^2) \\ y_{n+1} & = & \mathrm{RN}\,(y_n + e_n y_n) \end{cases}$$

Less accurate, faster (parallel)

In practice : we start with Goldschmidt iterations, and switch to Markstein iterations for the final steps.

Double roundings

C program :

```
double a = 1848874847.0;
double b = 19954562207.0;
double c;
c = a * b;
printf("c = %20.19e\n", c);
return 0;
```

Depending on the environment, we obtain 3.6893488147419103232e+19 or 3.6893488147419111424e+19 (which is the binary64 number closest to the exact product).

Double roundings

- several FP formats supported in a given environment → difficult to know in which format some operations are performed ;
- may make the result of a sequence of operations difficult to predict ;
- for instance, the C99 Std states :

  > *the values of operations with floating operands and values subject to the usual arithmetic conversions and of floating constants are evaluated to a format whose range and precision may be greater than required by the type.*

Double roundings

Assume the various declared variables of a program are of the same format. Two phenomenons may occur when a wider format is available :

- for implicit variables such as the result of “a+b” in “d = (a+b)*c”) : not clear in which format they are computed ;
- explicit variables may be first computed in the wider format, and then rounded to their destination format → sometimes leads to a problem called double rounding.

What happened in the example ?

The exact value of a*b is 36893488147419107329. In binary :

$$\overbrace{\underbrace{10000000000000000000000000000000000000000000000000000}_{53\text{ bits}}\,10000000000}^{64\text{ bits}}\,01$$

If it is first rounded to the INTEL “double-extended” format, we get

$$\overbrace{\underbrace{10000000000000000000000000000000000000000000000000000}_{53\text{ bits}}\,10000000000}^{64\text{ bits}}\times 4$$

if that intermediate value is rounded to the binary64 destination format, this gives (round-to-nearest-even rounding mode)

$$\underbrace{10000000000000000000000000000000000000000000000000000}_{53\text{ bits}}\times 2^{13}$$
$$= 36893488147419103232_{10},$$

$\longrightarrow$ rounded down, whereas it should have been rounded up.

Is it a problem ?

- In most applications, these phenomenons are innocuous ;
- they make the behavior of some numerical programs difficult to predict (interesting examples given by Monniaux) ;
- most compilers offer options that prevent this problem. However,
  - restricts the portability of numerical programs : e.g., difficult to make sure that one will always use 2Sum with the right compilation switches ;
  - may have a bad impact on the accuracy of programs, since it is in general more accurate to perform the intermediate calculations in a wider format.

→ examine which properties remain true when double roundings occur.

Notation

- precision-p target format, and precision-$(p+p')$ wider "internal" format ;
- when the precision is omitted, it is p (e.g. "FPN" means "precision-p FPN") ;
- $\mathrm{RN}_k(u)$ means u rounded to the nearest precision-k FP number (assuming round to nearest even) ;

Double rounding → the error of $a+b$ may not be a FPN

Consider $a = 1\underbrace{xxxx\cdots x}_{p-3 \text{ bits}}01$, where $xxxx\cdots x$ is any $(p-3)$-bit bit-chain.

Also consider, $b = 0.0\underbrace{111111\cdots 1}_{p \text{ ones}} = \frac{1}{2} - 2^{-p-1}$. We have :

$$a+b = \underbrace{1xxxx...x01}_{p \text{ bits}}.0\underbrace{111111...1}_{p \text{ bits}},$$

so that if $1 \leq p' \leq p$, $u = RN_{p+p'}(a+b) = 1xxxx...x01.100...0$, we have

$$s = RN_p(u) = 1xxxx...x10 = a+1$$

Therefore,

$$s-(a+b) = a+1-(a+\frac{1}{2}-2^{-p-1}) = \frac{1}{2}+2^{-p-1} = 0.\underbrace{10000\cdots 01}_{p+1 \text{ bits}},$$

which is not exactly representable in precision-p FP arithmetic.

Double roundings and double rounding biases

When the arithmetic operation $x \top y$ appears in a program :

- a double rounding occurs if what is actually performed is

$$\mathrm{RN}_p\left(\mathrm{RN}_{p+p'}(x \top y)\right),$$

- a double rounding bias occurs if a double rounding occurs and the obtained result differs from $\mathrm{RN}_p(x \top y)$.

2Sum and double roundings

Algorithm 4 (2Sum-with-double-roundings(a, b))

(1) $s \leftarrow RN_p(RN_{p+p'}(a+b))$ *or* $RN_p(a+b)$
(2) $a' \leftarrow RN_p(RN_{p+p'}(s-b))$ *or* $RN_p(s-b))$
(3) $b' \leftarrow \circ(s-a')$
(4) $\delta_a \leftarrow RN_p(RN_{p+p'}(a-a'))$ *or* $RN_p(a-a')$
(5) $\delta_b \leftarrow RN_p(RN_{p+p'}(b-b'))$ *or* $RN_p(b-b')$
(6) $t \leftarrow RN_p(RN_{p+p'}(\delta_a+\delta_b))$ *or* $RN_p(\delta_a+\delta_b)$

$\circ(u)$: $\mathrm{RN}_p(u)$, $\mathrm{RN}_{p+p'}(u)$, or $\mathrm{RN}_p(\mathrm{RN}_{p+p'}(u))$, or any faithful rounding.

Theorem 7

If $p \geq 4$ and $p + p'$, with $p' \geq 2$. If a and b are precision-p FPN, and if no overflow occurs, then Algorithm 4 satisfies :

- *if no double rounding bias occurred when computing s then $t = (a + b - s)$ exactly ;*
- *otherwise, $t = RN_p(a + b - s)$.*

Rump, Ogita and Oishi's cascaded summation algorithm

Algorithm 5

$s \leftarrow a_1$
$e \leftarrow 0$
for $i = 2$ *to* n ***do***
  $(s, e_i) \leftarrow$ *2Sum-with-double-roundings*(s, a_i)
  $e \leftarrow RN_p(RN_{p+p'}(e + e_i))$
end for
return $RN_p(RN_{p+p'}(s + e))$

Pichat, Rump, Ogita and Oishi's summation algorithm

Theorem 8

Assuming $p \geq 8$, $p' \geq 4$, *and* $n < \frac{1}{2u'}$, *the final value* σ *returned by Algorithm 5 satisfies*

$$\left|\sigma - \sum_{i=1}^{n} a_i\right| \leq \left(2^{-p} + 2^{-p-p'} + 2^{-2p-p'}\right) \cdot \sum_{i=1}^{n} a_i + 2^{-2p} \cdot \left(4n^2 - 10n - 5\right) \cdot \left(1 + 2^{-p'+1} + \frac{3}{200}\right) \cdot \sum_{i=1}^{n} |a_i|.$$

Rump, Ogita and Oishi's K-fold summation algorithm

Algorithm 6 (VecSum(a), where $a = (a_1, a_2, \ldots, a_n)$)

$p \leftarrow a$
for $i = 2$ *to* n ***do***
  $(p_i, p_{i-1}) \leftarrow \mathit{2Sum}(p_i, p_{i-1})$
end for
return p

Algorithm 7 (K-fold summation algorithm)

for $k = 1$ *to* $K - 1$ ***do***
  $a \leftarrow \mathit{VecSum}(a)$
end for
$c = a_1$
for $i = 2$ *to* $n - 1$ ***do***
  $c \leftarrow RN(c + a_i)$
end for
return $RN(a_n + c)$

Rump, Ogita and Oishi's K-fold summation algorithm

- without double roundings, if $4nu < 1$, the FPN σ returned by Algorithm 7 satisfies

$$\left|\sigma - \sum_{i=1}^{n} a_i\right| \le (u + \gamma_{n-1}^2)\left|\sum_{i=1}^{n} a_i\right| + \gamma_{2n-2}^K \sum_{i=1}^{n} |a_i|. \quad (1)$$

- if a double-rounding bias occurs in the first call to VecSum, not possible to show an error bound better than prop. to $2^{-2p}\sum_{i=1}^{n}|a_i|$;

Multiplication by "infinitely precise" constants

- We want $\text{RN}(Cx)$, where x is a FP number, and C a real constant (i.e., known at compile-time).
- Typical values of C : π, $1/\pi$, $\ln(2)$, $\ln(10)$, e, $1/k!$, $B_k/k!$, $1/10^k$, $\cos(k\pi/N)$ and $\sin(k\pi/N)$, ...
- another frequent case : $C = \frac{1}{\text{FP number}}$ (division by a constant) ;

The algorithm

- introduced by Brisebarre and M.,
- Cx with correct rounding (assuming rounding to nearest even) ;
- C is not a FP number ;
- A correctly rounded fma instruction is available. Operands stored in a binary FP format of precision p ;
- We assume that the two following FP numbers are pre-computed :

$$\left\{ \begin{array}{lcl} C_h & = & \mathrm{RN}\,(C), \\ C_\ell & = & \mathrm{RN}\,(C - C_h), \end{array} \right.$$

The algorithm

Algorithm 8 (Multiplication by C with a product and an `fma`)

From x, compute

$$\begin{cases} u_1 = RN(C_\ell x), \\ u_2 = RN(C_h x + u_1). \end{cases}$$

Returned result : u_2.

The algorithm

Algorithm 8 (Multiplication by C with a product and an `fma`)

From x, compute

$$\begin{cases} u_1 & = & RN(C_\ell x), \\ u_2 & = & RN(C_h x + u_1). \end{cases}$$

Returned result : u_2.

Warning ! There exist C and x s.t. $u_2 \neq \mathrm{RN}(Cx)$ – easy to build.

The algorithm

Algorithm 8 (Multiplication by C with a product and an `fma`)

From x, compute

$$\begin{cases} u_1 & = & RN(C_\ell x), \\ u_2 & = & RN(C_h x + u_1). \end{cases}$$

Returned result : u_2.

Warning ! There exist C and x s.t. $u_2 \neq \mathrm{RN}(Cx)$ – easy to build.

Fast methods for analyzing a given C

Examples

Theorem 9 (Correctly rounded multiplication by π)

The algorithm always returns a correctly rounded result in double precision with $C = 2^j\pi$, where j is any integer, provided no under/overflow occur.

- Same thing with $C = \ln(2)$;
- with $C = 1/\pi$, the only numbers x for which the algorithm does not work in double precision are of the form

$$6081371451248382 \times 2^{\pm k}.$$

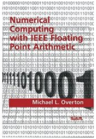
Conclusion

- operations fully specified (the double rounding problem should partly vanish when IEEE 754-2008 becomes widely implemented) ;
- derive algorithms, as well as proofs of properties ;
- formal proof investigated by several people ;

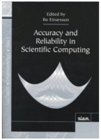
Floating-point arithmetic on the web

- W. Kahan :
  http://http.cs.berkeley.edu/~wkahan/
- Goldberg's paper "What every computer scientist should know about Floating-Point arithmetic"
  http://www.validlab.com/goldberg/paper.pdf
- D. Hough :
  http://www.validlab.com/754R/
- The Arenaire team of lab. LIP (ENS Lyon)
  http://www.ens-lyon.fr/LIP/Arenaire/
- my own web page
  http://perso.ens-lyon.fr/jean-michel.muller/

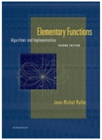
Books on Floating-Point Arithmetic



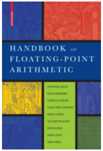
Michael Overton
Numerical Computing with IEEE Floating Point Arithmetic
Siam, 2001



Bo Einarsson
Accuracy and Reliability in Scientific Computing
Siam, 2005



Jean-Michel Muller
Elementary Functions, algorithms and implementation, 2ème édition
Birkhauser, 2006



Brisebarre, de Dinechin, Jeannerod, Lefèvre, Melquiond, Muller (coordinator), Revol, Stehlé and Torres
A Handbook of Floating-Point Arithmetic
Birkhauser, 2010.