Exact computations with approximate arithmetic

Jean-Michel Muller
CNRS - Laboratoire LIP
(CNRS-INRIA-Université de Lyon)
october 2007

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

Floating-Point Arithmetic ?

- used everywhere in scientific calculation;
- $x = m_x \times \beta^{e_x}$;
- "fuzzy" approach: computed value of $x + y = (x + y)(1 + \epsilon)$.

Floating-Point Arithmetic ?

- used everywhere in scientific calculation;
- $x = m_x \times \beta^{e_x}$;
- "fuzzy" approach: computed value of $x + y = (x + y)(1 + \epsilon)$.

Better approach ?

Needs ? A few figures...

- span:

$$\frac{\text{Estimated diameter of observable universe}}{\text{Planck's length}} \approx 10^{62}$$

Needs ? A few figures...

- span:

$$\frac{\text{Estimated diameter of observable universe}}{\text{Planck's length}} \approx 10^{62}$$

- accuracy: some predictions of general relativity or quantum mechanics verified within relative accuracy 10^{-14}

Needs ? A few figures. . .

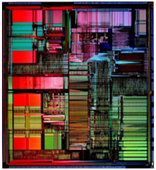
- span:

$$\frac{\text{Estimated diameter of observable universe}}{\text{Planck's length}} \approx 10^{62}$$

- accuracy: some predictions of general relativity or quantum mechanics verified within relative accuracy 10^{-14}
- intermediate calculations: quad precision and smart tricks required for very-long term stability of the Solar system (J. Laskar, Paris Observatory).
  Good news: we seem to be safe for the next 40 million years;

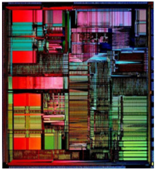
We can do a very poor job...

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



We can do a very poor job...

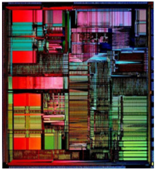
- Pentium 1 division bug: 8391667/12582905 gave $0.666869\cdots$ instead of $0.666910\cdots$;



- On some Cray computers, it was possible to get an overflow when multiplying by 1;

We can do a very poor job…

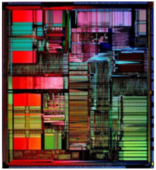
- Pentium 1 division bug: 8391667/12582905 gave $0.666869\cdots$ instead of $0.666910\cdots$;



- On some Cray computers, it was possible to get an overflow when multiplying by 1;
- Maple version 7.0, enter $\frac{5001!}{5000!}$ and you get 1 instead of 5001;

We can do a very poor job…

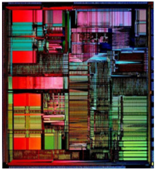
- Pentium 1 division bug: 8391667/12582905 gave $0.666869\cdots$ instead of $0.666910\cdots$;



- On some Cray computers, it was possible to get an overflow when multiplying by 1;
- Maple version 7.0, enter $\frac{5001!}{5000!}$ and you get 1 instead of 5001;
- Version 6.0 was even worse. Enter 214748364810, you get 10. Note that $2147483648 = 2^{31}$.

We can do a very poor job...

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



- On some Cray computers, it was possible to get an overflow when multiplying by 1;
- Maple version 7.0, enter $\frac{5001!}{5000!}$ and you get 1 instead of 5001;
- Version 6.0 was even worse. Enter 214748364810, you get 10. Note that $2147483648 = 2^{31}$.
- Excel'2007, compute $65535 - 2^{-37}$, you get 100000;

Floating-Point System

Parameters:

$$\begin{cases} \text{base} & \beta \geq 2 \\ \text{precision} & p \geq 1 \\ \text{extremal exponents} & E_{\min}, E_{\max} \end{cases}$$

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

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

Real significand, or significand of x the number

$$m = M \times \beta^{1-p},$$

so that $x = m \times \beta^e$.

Normal representation

Goal: uniqueness of representation.

The normal representation of x (if any) is the one for which $1 \leq m < \beta$. It is the one for which the exponent is minimum.

Normal representation

Goal: uniqueness of representation.

The normal representation of x (if any) is the one for which $1 \leq m < \beta$. It is the one for which the exponent is minimum.

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

Normal representation

Goal: uniqueness of representation.

The normal representation of x (if any) is the one for which $1 \leq m < \beta$. It is the one for which the exponent is minimum.

Base 2, the leftmost bit of the significand of a normal number is a "1" $\longrightarrow$ no need to store it (implicit 1 convention).
A subnormal number has the form

$$M \times \beta^{E_{\min}+1-p}.$$

with $|M| \leq \beta^{p-1} - 1$. Such a number has no normal representation.
Corresponds to $\pm 0.xxxxxxxx \times \beta^{E_{\min}}$.

IEEE-754 Standard for FP Arithmetic (1985)

- 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.);
- under revision.

IEEE-754 Standard for FP Arithmetic (1985)

- 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.);
- under revision.

Correct rounding

Definition 1 (Correct rounding)

The user defines an *active rounding mode* among:

- round to the nearest (default) in case of a tie, value whose integral significand is even;
- round towards $+\infty$.
- round towards $-\infty$.
- round towards zero.

An operation whose entries are FP numbers must return 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 mathematical structure in itself, that can be studied.

Error of FP addition (Møller, Knuth, Dekker)

First result: representability. $\mathrm{RN}(x)$ is x rounded to the nearest.

Lemma 2

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.

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 $a+b \rightarrow$ 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 3 (Fast2Sum (Dekker))

$\beta \leq 3$, subnormal numbers available. Let a and b be FP numbers, with exponents s.t. $e_a \geq e_b$ (if $|a| \geq |b|$, will be satisfied). 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;
```

Proof: Show that $s - a$ and $b - z$ are exactly representable.

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 very cheap in front of wrong branch prediction penalty when comparing a and b.

Algorithm 2 (TwoSum)

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

The TwoSum Algorithm (Møller-Knuth)

- no need to compare a and b;
- 6 operations instead of 3 yet 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 very cheap in front of wrong branch prediction penalty when comparing a and b.

Algorithm 2 (TwoSum)

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

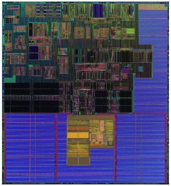
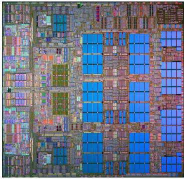
Knuth: $\forall\beta$, if no underflow nor overflow occurs then $a+b=s+r$, and s is nearest $a+b$.

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

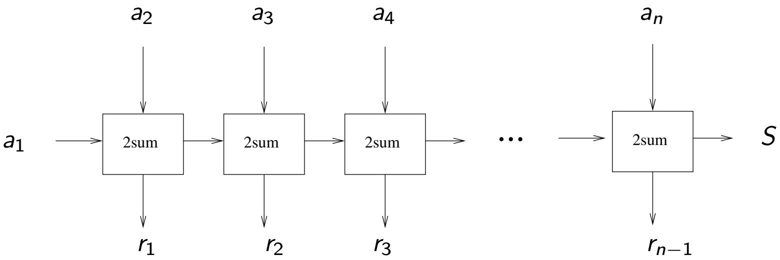
Formal proofs (in Coq) of many useful such algorithms:
http://lipforge.ens-lyon.fr/www/pff/Fast2Sum.html.

How about products ?

- FMA: *fused multiply-add*, computes $\mathrm{RN}\,(ab+c)$. RS6000, Itanium and PowerPC. Will be in IEEE 754-R;
- if a and b are FP numbers, then $r = ab - \mathrm{RN}\,(ab)$ is a FP number;
- obtained by Algorithm TwoMultFMA $\begin{cases} p &= \mathrm{RN}\,(ab) \\ r &= \mathrm{RN}\,(ab-p) \end{cases}$
  $\longrightarrow$ two operations only. $p + r = ab$.
- without a fma, Dekker algorithm: 17 operations ($7\ \times$, $10\ \pm$).

Itanium 2   PowerPC 5

Compensated summation methods (Kahan, Priest, Rump. . .)



- $S + (r_1 + r_2 + \cdots + r_{n-1}) = a_1 + a_2 + \cdots + a_{n-1}$ exactly;
- 1st solution: compute $S + (r_1 + r_2 + \cdots + r_{n-1})$ as usual. If all a_is have same sign, in double precision ($\beta = 2$, $p = 53$) and RN, one can add $\sqrt{2} \times 2^{26}$ values and get error $\leq$ weight of last bit;
- 2nd solution: use again the same trick for adding the r_i's

Evaluating powers

Algorithm 3 (DblMult(a_h,a_ℓ,b_h,b_ℓ))

Computes approx. to $(a_h + a_\ell)(b_h + b_\ell)$.

$$
\begin{array}{lcl}
t & := & RN(a_\ell b_h); \\
s & := & RN(a_h b_\ell + t); \\
(x', u) & := & TwoMultFMA(a_h, b_h); \\
(x'', v) & := & Fast2Sum(x', s); \\
y' & := & RN(u + v); \\
(x, y) & := & Fast2Sum(x'', y');
\end{array}
$$

Not an exact product!

Algorithm 4 (LogPower(x, n), $n \geq 1$)

$i := n$;
$(h, \ell) := (1, 0)$;
$(u, v) := (x, 0)$;
while $i > 1$ **do**
  if (i **mod** 2) $= 1$ **then**
    $(h, \ell) :=$ *DblMult*(h, ℓ, u, v);
  end;
  $(u, v) :=$ *DblMult*(u, v, u, v);
  $i := \lfloor i/2 \rfloor$;
end do;
return *DblMult*(h, ℓ, u, v);

Evaluating powers

If algorithm LogPower is run in double-extended precision ($\beta = 2, p = 64$), and $3 \leq n \leq 284$, then by rounding the final value to the nearest double-precision number, we get a correctly rounded result.

- rather error-prone error analysis;
- special algorithm for computing hardest-to-round cases;

For $3 \leq n \leq 284$, the hardest-to-round case for x^n is for $n = 51$. It is

$$x = 1.0100010111101011011011101010011111100101000111011101$$

$$x^{51} = \underbrace{1.10110011101001000111001000011001000000101101011101110}_{53 \text{ bits}}\ 1$$

$$\underbrace{00000000000000000000000000000000000000000000000000000000000}_{59 \text{ zeros}}\ 100\cdots \times 2^{17}$$

Error term of a FMA

- Joint work with Sylvie Boldo (2005);
- $\beta = 2$, $p \geq 3$, fma, no underflow nor overflow;
- a, x, y: FP numbers;
- a fma computes $r_1 = \mathrm{RN}(ax + y)$;
- Two questions:
  - how many FP numbers are necessary for representing $r_1 - (ax + y)$?
  - can these numbers be easily computed?

Error term of a FMA

- Joint work with Sylvie Boldo (2005);
- $\beta = 2$, $p \geq 3$, fma, no underflow nor overflow;
- a, x, y: FP numbers;
- a fma computes $r_1 = \mathrm{RN}\,(ax + y)$;
- Two questions:
  - how many FP numbers are necessary for representing $r_1 - (ax + y)$?
  - can these numbers be easily computed?
- Answers:

Error term of a FMA

- Joint work with Sylvie Boldo (2005);
- $\beta = 2$, $p \geq 3$, fma, no underflow nor overflow;
- a, x, y: FP numbers;
- a fma computes $r_1 = \mathrm{RN}\,(ax + y)$;
- Two questions:
  - how many FP numbers are necessary for representing $r_1 - (ax + y)$?
  - can these numbers be easily computed?
- Answers:
  - two numbers;

Error term of a FMA

- Joint work with Sylvie Boldo (2005);
- $\beta = 2$, $p \geq 3$, fma, no underflow nor overflow;
- a, x, y: FP numbers;
- a fma computes $r_1 = \text{RN}\,(ax + y)$;
- Two questions:
  - how many FP numbers are necessary for representing $r_1 - (ax + y)$?
  - can these numbers be easily computed?
- Answers:
  - two numbers;
  - you need 19 operations (1 TwoMultFMA, 2 TwoSum, 2 additions, 1 FastTwoSum);

Error term of a FMA

- Joint work with Sylvie Boldo (2005);
- $\beta = 2$, $p \geq 3$, fma, no underflow nor overflow;
- a, x, y: FP numbers;
- a fma computes $r_1 = \text{RN}(ax + y)$;
- Two questions:
  - how many FP numbers are necessary for representing $r_1 - (ax + y)$?
  - can these numbers be easily computed?
- Answers:
  - two numbers;
  - you need 19 operations (1 TwoMultFMA, 2 TwoSum, 2 additions, 1 FastTwoSum);
  - *I did not trust our proof* before Sylvie wrote it in Coq.

Humans don't need computers to do silly things

- The Mars Climate Orbiter probe crashed on Mars in 1999;

Humans don't need computers to do silly things

- The Mars Climate Orbiter probe crashed on Mars in 1999;
- one of the software teams assumed the unit of length was the meter;

Humans don't need computers to do silly things

- The Mars Climate Orbiter probe crashed on Mars in 1999;
- one of the software teams assumed the unit of length was the meter;
- the other team assumed it was the foot.

Multiplication by "infinitely precise" constants

- Joint work with Nicolas Brisebarre;
- We want $\mathrm{RN}\,(Cx)$, 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 naive method

- replace C by $C_h = \mathrm{RN}(C)$;
- compute $\mathrm{RN}(C_h x)$ (instruction `y = Ch * x`).

p	Prop. of correctly-rounded results
5	0.93750
6	0.78125
7	0.59375
...	...
16	0.86765
17	0.73558
...	...
24	0.66805

Proportion of FP numbers x for which $RN(C_h x) = RN(Cx)$ for $C = \pi$ and various p.

The algorithm

- 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:

$$\begin{cases} C_h &= \mathrm{RN}(C), \\ C_\ell &= \mathrm{RN}(C - C_h), \end{cases}$$

The algorithm

Algorithm 5 (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 5 (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 5 (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;
- Without l.o.g., we assume that $1 < x < 2$ and $1 < C < 2$, that C is not exactly representable, and that $C - C_h$ is not a power of 2;

The algorithm

Algorithm 5

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.

Two methods for checking if $\forall x, u_2 = \mathrm{RN}(Cx)$.

- Method 1: simple but does not always give a complete answer;
- Method 2: gives all "bad cases", or certify that there are none, i.e. that the algorithm always returns $\mathrm{RN}(Cx)$.

Analyzing the algorithm

Bound on maximum possible distance between u_2 and Cx:

Property 1

For all FP number x, we have

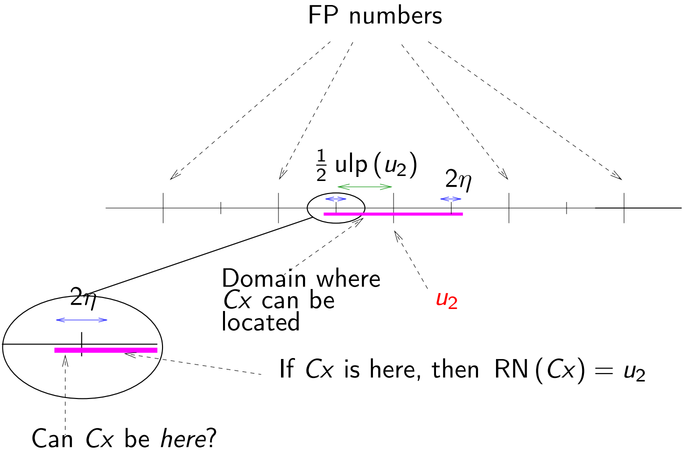
$$|u_2 - Cx| < \frac{1}{2}\, ulp\,(u_2) + 2\, ulp\,(C_\ell).$$

ulp(t) (unit in the last place) = distance between consecutive FP numbers around t. Correct rounding $\leftrightarrow$ error $\leq \frac{1}{2}$ ulp.

$$\text{ulp}\ (t_0.t_1 t_2 \cdots t_{p-1} \times 2^{e_t}) = 2^{e_t - p}$$

Analyzing the algorithm

Reminder: $|u_2 - Cx| < \frac{1}{2}\,\mathrm{ulp}\,(u_2) + \eta$ with $\eta = 2\,\mathrm{ulp}\,(C_\ell)$.



Analyzing the algorithm

- We know that Cx is within $1/2\,\mathrm{ulp}\,(u_2) + 2\,\mathrm{ulp}\,(C_\ell)$ from the FP number u_2.

Analyzing the algorithm

- We know that Cx is within $1/2\,\mathrm{ulp}\,(u_2) + 2\,\mathrm{ulp}\,(C_\ell)$ from the FP number u_2.
- If we prove that Cx cannot be at a distance $\leq \eta = 2\,\mathrm{ulp}\,(C_\ell)$ from the middle of two consecutive FP numbers, then u_2 will be the FP number that is closest to Cx.

A reminder on continued fractions

We will use the following well-known results:

Theorem 4

Let $(p_j/q_j)_{j\geq 1}$ be the convergents of β. For any (p,q), with $0 \leq q < q_{n+1}$, we have

$$|p - \beta q| \geq |p_n - \beta q_n|.$$

Theorem 5

Let p, q be nonzero integers, with $\gcd(p,q) = 1$. If

$$\left|\frac{p}{q} - \beta\right| < \frac{1}{2q^2}$$

then p/q is a convergent of β.

Method 1: use of Theorem 4

- Remark: Cx can be in $[1, 2)$ or $[2, 4)$ $\rightarrow$ two (very similar) cases;
- define $x_{\text{cut}} = 2/C$. Let $X = 2^{p-1}x$ and $X_{\text{cut}} = \lfloor 2^{p-1}x_{\text{cut}} \rfloor$.
- we detail the case $x < x_{\text{cut}}$ below.

Middle of two consecutive FP numbers around Cx: $\frac{2A+1}{2^p}$ where $A \in \mathbb{Z}$, $2^{p-1} \leq A \leq 2^p - 1$ $\rightarrow$ we try to know if there can be such an A such that

$$\left| Cx - \frac{2A+1}{2^p} \right| < \eta.$$

This is equivalent to

$$|2CX - (2A+1)| < 2^p\eta.$$

Method 1: use of Theorem 4 (cont)

We want to know if there exists X between 2^{p-1} and X_{cut} and A between 2^{p-1} and $2^p - 1$ such that

$$|2CX - (2A+1)| < 2^p \eta.$$

- $(p_i/q_i)_{i \geq 1}$: convergents of $2C$;
- k: smallest integer such that $q_{k+1} > X_{\text{cut}}$,
- define $\delta = |p_k - 2Cq_k|$.

Theorem 4 $\Rightarrow \forall B, X \in \mathbb{Z}$, with $0 < X \leq X_{\text{cut}} < q_{k+1}$, $|2CX - B| \geq \delta$.

Method 1: use of Theorem 4 (cont)

Therefore

1. If $\delta \geq 2^p \eta$ then $|Cx - A/2^p| < \eta$ is impossible $\Rightarrow$ the algorithm returns $\mathrm{RN}\,(Cx)$ for all $x < x_{\mathrm{cut}}$;
2. if $\delta < 2^p \eta$, we try the algorithm with $x = q_k 2^{-p+1} \rightarrow$ either we get a counter-example, or we cannot conclude

Case $x > x_{\mathrm{cut}}$: similar (convergents of C instead of those of $2C$)

Example: $C = \pi$, double precision ($p = 53$)

```
> method1(Pi/2,53);
Ch = 884279719003555/562949953421312
Cl = 4967757600021511/81129638414606681695789005144064
xcut = 1.2732395447351626862, Xcut = 5734161139222658
eta = .8069505497e-32
pk/qk = 6134899525417045/1952799169684491
delta = .9495905771e-16
OK for X < 5734161139222658
etaprime = .1532072145e-31
pkprime/qkprime = 12055686754159438/7674888557167847
deltaprime = .6943873667e-16
OK for 5734161139222658 <= X < 9007199254740992
```

$\Rightarrow$ We always get a correctly rounded result for $C = 2^k\pi$ and $p = 53$, with $C_h = 2^{k-48} \times 884279719003555$ and
$C_\ell = 2^{k-105} \times 4967757600021511$.

Consequence 1

Correctly rounded multiplication by π: in double precision one multiplication and one fma.

Method 2

- Again, two cases. Here: $x > x_{\text{cut}}$ (case $x < x_{\text{cut}} = 2/C$ similar);

Method 2

- Again, two cases. Here: $x > x_{\mathrm{cut}}$ (case $x < x_{\mathrm{cut}} = 2/C$ similar);
- We recall the notations: $C_h = \mathrm{RN}(C)$, $C_\ell = \mathrm{RN}(C - C_h)$,

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

- Again, $X_{\mathrm{cut}} = 2^{p-1} x_{\mathrm{cut}}$;

Method 2

- Again, two cases. Here: $x > x_{\text{cut}}$ (case $x < x_{\text{cut}} = 2/C$ similar);
- We recall the notations: $C_h = \text{RN}(C)$, $C_\ell = \text{RN}(C - C_h)$,

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

- Again, $X_{\text{cut}} = 2^{p-1} x_{\text{cut}}$;
- We want to determine integers X, $X_{\text{cut}} \leq X \leq 2^p - 1$ s.t. $\exists A \in \mathbb{Z}$, $2^{p-1} \leq A \leq 2^p - 1$ with

$$\left| C \frac{X}{2^{p-1}} - \frac{2A+1}{2^{p-1}} \right| \leq 2 \, \text{ulp}(C_\ell).$$

Method 2

- Again, two cases. Here: $x > x_{\text{cut}}$ (case $x < x_{\text{cut}} = 2/C$ similar);
- We recall the notations: $C_h = \text{RN}(C)$, $C_\ell = \text{RN}(C - C_h)$,

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

- Again, $X_{\text{cut}} = 2^{p-1} x_{\text{cut}}$;
- We want to determine integers X, $X_{\text{cut}} \leq X \leq 2^p - 1$ s.t. $\exists A \in \mathbb{Z}$, $2^{p-1} \leq A \leq 2^p - 1$ with

$$\left| C \frac{X}{2^{p-1}} - \frac{2A+1}{2^{p-1}} \right| \leq 2 \,\text{ulp}(C_\ell).$$

- Once we know the X candidate, we compute u_2 and $\text{RN}(Cx)$ to check if they coincide or not.

Method 2

- We are looking for $x = X/2^{p-1},\ X_{\text{cut}} \leq X \leq 2^p - 1$ s.t. $\exists A$ with
$$\left| C\frac{X}{2^{p-1}} - \frac{2A+1}{2^{p-1}} \right| \leq 2\,\mathrm{ulp}\,(C_\ell). \tag{1}$$

Method 2

- We are looking for $x = X/2^{p-1}$, $X_{\text{cut}} \le X \le 2^p - 1$ s.t. $\exists A$ with
$$\left| C \frac{X}{2^{p-1}} - \frac{2A+1}{2^{p-1}} \right| \le 2 \operatorname{ulp}(C_\ell). \qquad (1)$$
- We know that $\operatorname{ulp}(C_\ell) \le 2^{-2p}$;

Method 2

- We are looking for $x = X/2^{p-1}$, $X_{\text{cut}} \le X \le 2^p - 1$ s.t. $\exists A$ with
$$\left| C \frac{X}{2^{p-1}} - \frac{2A+1}{2^{p-1}} \right| \le 2\,\text{ulp}\,(C_\ell). \tag{1}$$
- We know that $\text{ulp}\,(C_\ell) \le 2^{-2p}$;
- Two cases: $\text{ulp}\,(C_\ell) \le 2^{-2p-1}$ and $\text{ulp}\,(C_\ell) = 2^{-2p}$.

Method 2

First, we assume $\mathrm{ulp}(C_\ell) \leq 2^{-2p-1}$.
In that case, the integers X that satisfy (1) satisfy

$$\left|2C - \frac{2A+1}{X}\right| < \frac{1}{2X^2}:$$

- $(2A+1)/X$ is a convergent of $2C$ from Theorem 5.

Method 2

First, we assume $\mathrm{ulp}(C_\ell) \leq 2^{-2p-1}$.
In that case, the integers X that satisfy (1) satisfy

$$\left|2C - \frac{2A+1}{X}\right| < \frac{1}{2X^2} :$$

- $(2A+1)/X$ is a convergent of $2C$ from Theorem 5.
- It suffices then to check all the convergents of $2C$ of denominator less than or equal to $2^p - 1$.

Method 2

Now, assume $\mathrm{ulp}\,(C_\ell) = 2^{-2p}$.

- We are led to the following problem: determine the X, $X_{\mathrm{cut}} \le X \le 2^p - 1$ s.t.

$$\left\{ X(C_h + C_\ell) + \frac{1}{2^{p+1}} \right\} \le \frac{1}{2^p},$$

  where $\{y\}$ is the fractional part of y: $\{y\} = y - \lfloor y \rfloor$.
- Algorithm (see later) to determine the integers $X, X_{\mathrm{cut}} \le X \le 2^p - 1$ solution of this inequality;
- check the algorithm (i.e., compute u_2 and compare with $\mathrm{RN}\,(Cx)$) with these values of X.

An example: multiplication by $1/\pi$ in double precision

Consider the case $C = 4/\pi$ and $p = 53$

- Method 1 gives a (family of) counterexample(s): $x = 6081371451248382 \times 2^{\pm k}$.
- Method 2 certifies that $x = 6081371451248382 \times 2^{\pm k}$ are the only FP values for which our algorithm fails.

Implementation

- Maple programs that implement Methods 1 and 2;
- These programs (along with explanations) can be downloaded from

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

Some results

C	p	Method 1	Method 2
π	8	Does not work for 226	AW unless $X =$ 226
π	24	unable	AW
π	53	AW	AW
π	64	unable	AW
π	113	AW	AW

Table: *The results given for constant C hold for all values $2^{\pm j} C$. "AW" means "always works" and "unable" means "the method is unable to conclude".*

Some results

C	p	Method 1	Method 2
$1/\pi$	24	unable	AW
$1/\pi$	53	Does not work for 6081371451248382	AW unless $X =$ 6081371451248382
$1/\pi$	64	AW	AW
$1/\pi$	113	unable	AW
$\ln 2$	24	AW	AW
$\ln 2$	53	AW	AW
$\ln 2$	64	AW	AW
$\ln 2$	113	AW	AW

Table: *The results given for constant C hold for all values $2^{\pm j}C$.*

Conclusion on multiplication by a constant

The two methods make it possible to check whether correctly rounded multiplication by an “infinite precision” constant C is feasible at a low cost (one multiplication and one `fma`).

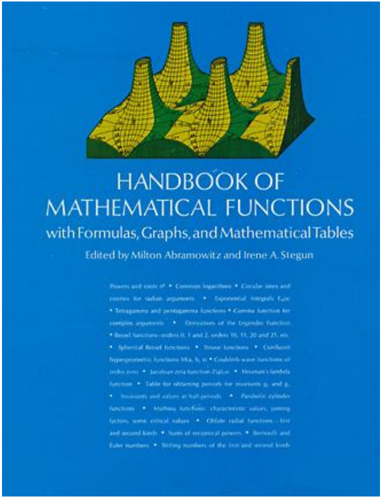
- method 1 does not always allow one to conclude, but is quite simple: use it at compile time?
- method 2 always gives the counter-examples or certifies that the algorithm always works.

The IEEE-754 Std is under revision

- near the end of the process (hopefully);
- main ideas remain the same;
- fma, base 10, some considerations on the elementary functions (sin, cos, exp, log, etc.) and their correct rounding;
- watch `http://754r.ucbtest.org/`

The Table Maker's Dilemma

The Table Maker's Dilemma



The Table Maker's Dilemma

Consider the double precision FP number ($\beta = 2, p = 53$)

$$x = \frac{8520761231538509}{2^{62}}$$

We have

$$2^{53+x} = 9018742077413030.{\color{blue}99999999999999999}8805240837303\cdots$$

The Table Maker's Dilemma

Consider the double precision FP number ($\beta = 2, p = 53$)

$$x = \frac{8520761231538509}{2^{62}}$$

We have

$$2^{53+x} = 9018742077413030.99999999999999999 8805240837303 \cdots$$

So what ?

The Table Maker's Dilemma

Consider the double precision FP number ($\beta = 2, p = 53$)

$$x = \frac{8520761231538509}{2^{62}}$$

We have

$$2^{53+x} = 9018742077413030.999999999999999999805240837303\cdots$$

So what ?

Hardest-to-round case for function 2^x and double precision FP numbers.

Joint work with Vincent Lefèvre.

The Hall of Shame (Ng)

System	$\sin(10^{22})$
exact result	$-0.8522008497671888017727\cdots$
HP 48 GX	-0.852200849762
HP 700	0.0
HP 375, 425t (4.3 BSD)	$-0.65365288\cdots$
matlab V.4.2 c.1 for Macintosh	0.8740
matlab V.4.2 c.1 for SPARC	-0.8522
SPARC	-0.85220084976718879
IBM RS/6000 AIX 3005	$-0.852200849\cdots$
DECstation 3100	NaN
Casio fx-8100, fx180p, fx 6910 G	Error
TI 89	Trig. arg. too large

No standard for the elementary functions.

Correct rounding of the elementary functions

- base 2, precision p;
- FP number x and integer m (with $m > p$) $\rightarrow$ one can compute an approximation y to $f(x)$ whose error on the significand is $\leq 2^{-m}$.
- can be done with a possible wider format, or using algorithms such as TwoSum, TwoMultFMA, Dekker product, etc.
- getting a correct rounding of $f(x)$ from y: not possible if y is too close to a breakpoint: a point where the rounding function changes.

Correct rounding of the elementary functions

- RN mode,

$$\overbrace{\underbrace{1.xxxxx\cdots xxx}_{p \text{ bits}}1000000\cdots 000000}^{m \text{ bits}}xxx\cdots$$

or

$$\overbrace{\underbrace{1.xxxxx\cdots xxx}_{p \text{ bits}}0111111\cdots 111111}^{m \text{ bits}}xxx\cdots\ ;$$

- other modes,

$$\overbrace{\underbrace{1.xxxxx\cdots xxx}_{p \text{ bits}}0000000\cdots 000000}^{m \text{ bits}}xxx\cdots$$

or

$$\overbrace{\underbrace{1.xxxxx\cdots xxx}_{p \text{ bits}}1111111\cdots 111111}^{m \text{ bits}}xxx\cdots\ .$$

Finding m beyond which there is no problem ?

- function f: sin, cos, arcsin, arccos, tan, arctan, exp, log, sinh, cosh,

Finding m beyond which there is no problem ?

- function f: sin, cos, arcsin, arccos, tan, arctan, exp, log, sinh, cosh,
- Lindemann's theorem ($z \neq 0$ algebraic $\Rightarrow e^z$ transcendental) $\rightarrow$ except for straightforward cases (e^0, $\ln(1)$, $\sin(0)$, ...), if x is a FP number, there exists an m, say m_x, s.t. rounding the m_x-bit approximation $\Leftrightarrow$ rounding $f(x)$;

Finding m beyond which there is no problem ?

- function f: sin, cos, arcsin, arccos, tan, arctan, exp, log, sinh, cosh,
- Lindemann's theorem ($z \neq 0$ algebraic $\Rightarrow e^z$ transcendental) $\rightarrow$ except for straightforward cases (e^0, $\ln(1)$, $\sin(0)$, $\ldots$), if x is a FP number, there exists an m, say m_x, s.t. rounding the m_x-bit approximation $\Leftrightarrow$ rounding $f(x)$;
- finite number of FP numbers $\rightarrow \exists m_{\max} = \max_x(m_x)$ s.t. $\forall x$, rounding the $m_{\max}$-bit approximation to $f(x)$ is equivalent to rounding $f(x)$;

Finding m beyond which there is no problem ?

- function f: sin, cos, arcsin, arccos, tan, arctan, exp, log, sinh, cosh,
- Lindemann's theorem ($z \neq 0$ algebraic $\Rightarrow e^z$ transcendental) $\longrightarrow$ except for straightforward cases (e^0, $\ln(1)$, $\sin(0)$, ...), if x is a FP number, there exists an m, say m_x, s.t. rounding the m_x-bit approximation $\Leftrightarrow$ rounding $f(x)$;
- finite number of FP numbers $\rightarrow \exists m_{\max} = \max_x(m_x)$ s.t. $\forall x$, rounding the $m_{\max}$-bit approximation to $f(x)$ is equivalent to rounding $f(x)$;
- this reasoning does not give any hint on the order of magnitude of $m_{\max}$. Could be huge.

A bound derived from a result due to Baker (1975)

- $\alpha = i/j$, $\beta = r/s$, with $i, j, r, s < 2^p$;
- $C = 16^{200}$;

$$|\alpha - \log(\beta)| > (p2^p)^{-Cp\log p}$$

Application: To evaluate ln et exp in double precision ($p = 53$) with correct rounding, it suffices to compute an approximation accurate to around

A bound derived from a result due to Baker (1975)

- $\alpha = i/j$, $\beta = r/s$, with $i, j, r, s < 2^p$;
- $C = 16^{200}$;

$$|\alpha - \log(\beta)| > (p2^p)^{-Cp\log p}$$

Application: To evaluate ln et exp in double precision ($p = 53$) with correct rounding, it suffices to compute an approximation accurate to around

$$10^{244} \text{ bits}$$

A bound derived from a result due to Baker (1975)

- $\alpha = i/j$, $\beta = r/s$, with $i, j, r, s < 2^p$;
- $C = 16^{200}$;

$$|\alpha - \log(\beta)| > (p2^p)^{-Cp \log p}$$

Application: To evaluate ln et exp in double precision ($p = 53$) with correct rounding, it suffices to compute an approximation accurate to around

$$10^{244} \text{ bits}$$

Fortunately, in practice, much less (≈ 100).

Some insight, but no proof...

- the infinitely precise significand y of $f(x)$ has the form:

$$y = y_0.y_1y_2\cdots y_{p-1}\overbrace{01111111\cdots11}^{kbits}xxxxx\cdots$$

or

$$y = y_0.y_1y_2\cdots y_{p-1}\overbrace{10000000\cdots00}^{kbits}xxxxx\cdots$$

with $k \geq 1$.

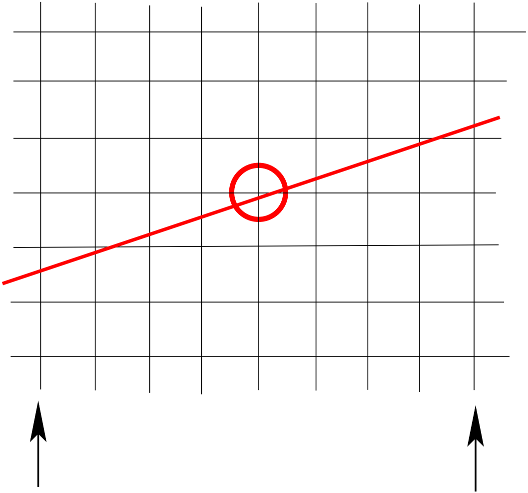
- Assuming that after the p^{th} position the "1" and "0" are equally likely, the "probability" of having $k \geq k_0$ is 2^{1-k_0};
- if we consider N input FP numbers, around $N \times 2^{1-k_0}$ values for which $k \geq k_0$;

$\rightarrow$ no longer happens as soon as k_0 is significantly larger than $\log_2(N)$ (for one given value of the exponent, as soon as $k_0 \gg p$).

Worst cases for double precision

- Lefèvre's method: split the domain $\rightarrow$ piecewise linear approximation to the functions for a pre-filtering, so that there remain only a few cases to be checked with big precision;
- pre-filtering: variant of the Euclidean algorithm;
- double precision: why ?
  - by far the most used;
  - computing all sines of the 2^{32} single-precision numbers: a few hours only;
  - precisions higher than double seem out of reach (maybe double extended in a few years, thanks to Moore's law).
- algorithm of better complexity, based on LLL: Stehlé, Lefèvre, Zimmermann. Similar in practice.

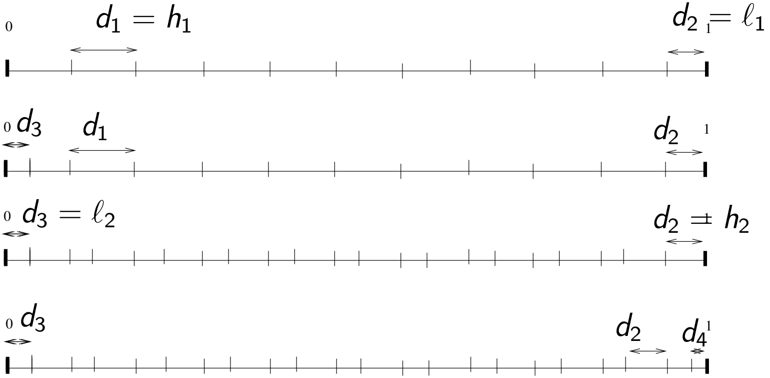
Worst cases for double precision



- grid: FP numbers and "breakpoints";
- scaling $\rightarrow$ integers;
- is the line very close to a point of the grid?
- it is very likely that the answer is "no".

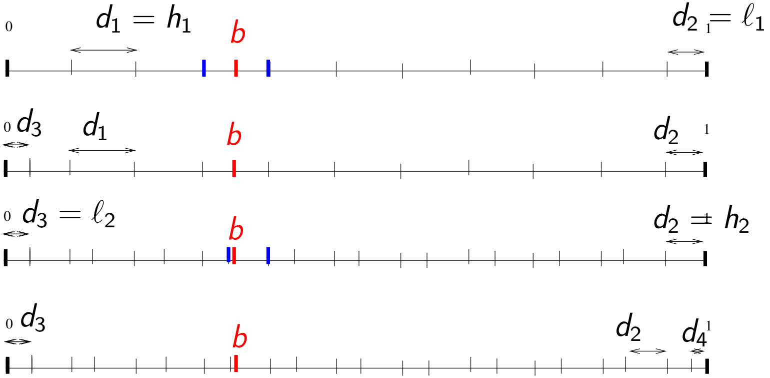
tiny $an - b \mod 1$

$an \mod 1$



- infinitely many times: two lengths;
- $h_{i+1} = \max(h_i - \ell_i, \ell_i)$, $\ell_{i+1} = \min(h_i - \ell_i, \ell_i) \to$ GCD.

Smallest distance between $an \mod 1$ and b, $n < T$?



- don't build all the points: just count them (to stop as soon as more than T);
- just build the two points that surround b, and update the distance to the left one;

Complexity ?

- $N = 2^p$ points;
- scaling $P(t) = Nf(t/N) \rightarrow$ integers;
- T points in each subinterval;
- accuracy of the filtering:

$$|P(t) \mod 1| < \frac{1}{M},$$

- if $f(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)^2 + \cdots,$

$$\left| a_2 \frac{T^2}{N} \right| < \frac{1}{M},$$

- we expect T/M cases in each subinterval $\rightarrow$ we assume $T \ll M$;
- gives $T \ll N^{1/3}$;
- we have to consider $N/T \approx N^{2/3}$ subintervals.

Table: *Worst cases for exponentials of double precision FP numbers.*

Interval	worst case (binary)
$[-\infty, -2^{-30}]$	$\exp(-1.1110110100110001100011101111101101100010011111101010 \times 2^{-27})$ $= 1.111111111111111111111111100 \cdots 0111000100 \quad 1 \quad 1^{59}0001\ldots \times 2^{-1}$
$[-2^{-30}, 0)$	$\exp(-1.0000000000000000000000000000000000000000000000000001 \times 2^{-51})$ $= 1.111111111111111 \cdots 111111111111111100 \quad 0 \quad 0^{100}1010\ldots \times 2^{-1}$
$(0, +2^{-30}]$	$\exp(1.1111111111111111111111111111111111111111111111111111 \times 2^{-53})$ $= 1.0000000000000000000000000000000000000000000000000000 \quad 1 \quad 1^{104}0101\ldots$
$[2^{-30}, +\infty]$	$\exp(1.0111111111111110011111111111111011100000000000100100 \times 2^{-32})$ $= 1.0000000000000000000000000000000101111111111111101000 \quad 0 \quad 0^{57}1101\ldots$
	$\exp(1.1000000000000001011111111111111011011111111111011100 \times 2^{-32})$ $= 1.0000000000000000000000000000000110000000000000010111 \quad 1 \quad 1^{57}0010\ldots$
	$\exp(1.1001111010011100101110111111110101100000100000001011 \times 2^{-31})$ $= 1.0000000000000000000000000000000110011110100111001011 1 \quad 1 \quad 0^{57}1010\ldots$
	$\exp(110.00001111010100101111001101111010111011001111110100)$ $= 110101100.01010000101101000000100111001000101011101110 \quad 0 \quad 0^{57}1000\ldots$

Table: *Worst cases for logarithms of double precision FP numbers.*

Interval	worst case (binary)
$[2^{-1074}, 1)$	$\log(1.1110101001110001110110000101110011101110000000100000 \times 2^{-509})$ $= -101100000.00101001011010100110011010110100001011111111 \quad 1 \quad 1^{60}0000\ldots$
	$\log(1.1001010001110110111000110000010011001101011111000111 \times 2^{-384})$ $= -100001001.10110110000011001010111101000111101100110101 \quad 1 \quad 0^{60}1010\ldots$
	$\log(1.0010011011101001110001001101001100100111100101100000 \times 2^{-232})$ $= -10100000.101010110010110000100101111001101000010000100 \quad 0 \quad 0^{60}1001\ldots$
	$\log(1.0110000100111001010101011101110010000000001011111000 \times 2^{-35})$ $= -10111.111100000010111110011011101011110110000000110101 \quad 0 \quad 1^{60}0011\ldots$
$(1, 2^{1024}]$	$\log(1.0110001010101000100001100001001101100010100110110110 \times 2^{678})$ $= 111010110.01000111100111101011101001111100100101110001 \quad 0 \quad 0^{64}1110\ldots$

Conclusion

- $N = 2^p$ FP values $\rightarrow O(N^{2/3})$ sub-intervals;
- work by Stehlé, Lefèvre, Zimmermann and Hanrot using lattice reduction: better complexity $O(N^{3/5+\epsilon})$, but a big hidden constant.
- correct rounding of the most usual functions is feasible at reasonable cost;
- recommended in the current draft of the IEEE 754 revision;
- CRLIBM library available at `https://lipforge.ens-lyon.fr/projects/crlibm/` (within 10% from LIBM)

Thank you!