Améliorer la qualité numérique des calculs

Licence informatique (L3)

Ph. Langlois

Université de Perpignan, France

4 avril 2018

Context and motivations

Sources of errors when computing the solution of a scientific problem in floating point arithmetic:

- mathematical model,
- truncation errors,
- data uncertainties,
- rounding errors.

Context and motivations

Sources of errors when computing the solution of a scientific problem in floating point arithmetic:

- mathematical model,
- truncation errors,
- data uncertainties,
- rounding errors.

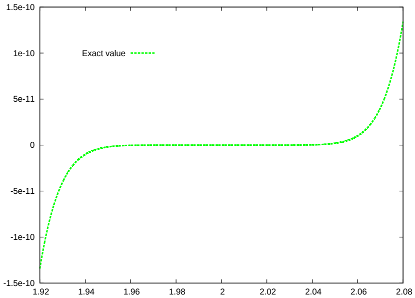
Rounding errors may totally corrupt a floating point computation:

- accumulation of billions of floating point operations,
- intrinsic difficulty to solve the problem accurately.

Example: polynomial evaluation

Evaluation of univariate polynomials with floating point coefficients:

- the evaluation of a polynomial suffers from rounding errors
- example : in the neighborhood of a multiple root

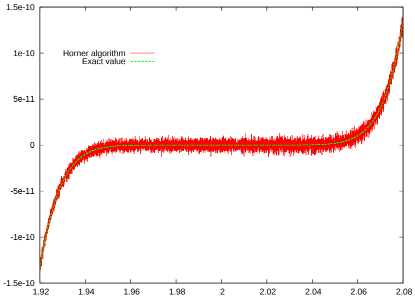


$p(x) = (x-2)^9$ in expanded form near the multiple root $x = 2$

Example: polynomial evaluation

Evaluation of univariate polynomials with floating point coefficients:

- the evaluation of a polynomial suffers from rounding errors
- example : in the neighborhood of a multiple root



$p(x) = (x-2)^9$ in expanded form
near the multiple root $x = 2$
evaluated with the Horner algorithm
in IEEE double precision.

General motivation

How to:

– improve and validate the accuracy of a floating point computation,

– recover the numerical reproducibility of parallel floating-point computation,

– without large computing time overheads ?

- More hardware precision
- Software simulation of more computing precision
- More accurate algorithms
- As fast as possible accurate algorithms
- Parallelism is everywhere
- Reproducibility at least to debug!

Main issues today

Starting point: IEEE-754 floating point arithmetic

- Best possible accuracy for $+, -, \times, /, \sqrt{}$
- Add is **not associative**

Summing *n* floating numbers : focus on accuracy

- Core computation, numerous algos, recently some really smart ones
- Computed sum accuracy : doubled or more, faithful or correctly rounded

Summing *n* floating numbers : focus on running-time and memory print

- Running-time and memory print are discriminant factors when computing the best possible accurate sum
- Appreciating the actual performance of one algo is not an easy task : flop/s ? hardware counters ? compiler options ?

Reliable and significant measure of the time complexity?

The classic way: count the number of flop

- A usual problem: double the accuracy of a computed result
- A usual answer for polynomial evaluation (degree n)

Metric	Horner	CompHorner	DDHorner
Flop count	2n	$22n + 5$	$28n + 4$
Flop count ratio	1	≈ 11	≈ 14
Measured #cycles ratio	1	2.8 – 3.2	8.7 – 9.7

Flop count vs. run-time measures

- Flop counts and measured run-times are not proportional
- Run-time measure is a very difficult experimental process
- Which one trust?

Reproducible HPC numerical simulations?

Numerical reproducibility of parallel computation

- Getting bitwise identical results for every p-parallel run, $p \geq 1$

One industrial scale simulation code

- Simulation of free-surface flows in 1D-2D-3D hydrodynamics
- 300 000 loc. of open source Fortran 90
- 20 years, 4000 registered users, EDF R&D + international consortium

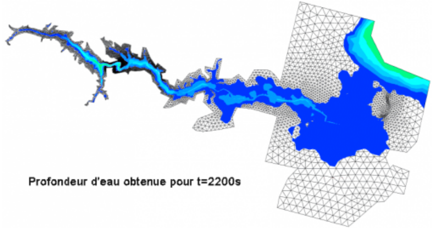
Telemac 2D [5]

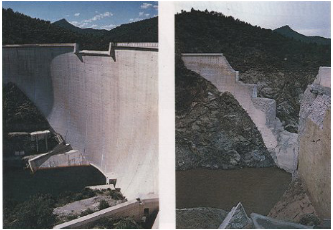
- 2D hydrodynamic: Saint Venant equations
- Finite element method, triangular element mesh, sub-domain decomposition for parallel resolution
- Mesh node unknowns: water depth (H) and velocity (U,V)

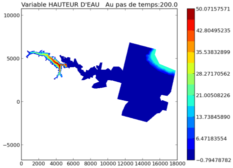
A complex Telemac 2D simulation

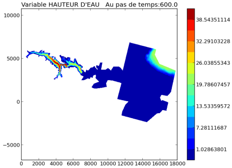
The Malpasset dam break (1959)

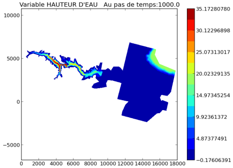
- A five year old dam break: 433 dead people and huge damage
- Triangular mesh: 26000 elements and 53000 nodes
- Simulation: $\rightarrow$35min. after break with a 2sec. time step



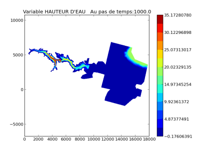








How to trust this complex simulation?



A reproducible simulation?

	velocity U	velocity V	depth H
The sequential run	0.4029747E-02	0.7570773E-02	0.3500122E-01
one 64 procs run	0.4935279E-02	0.3422730E-02	0.2748817E-01
one 128 procs run	0.4512116E-02	0.7545233E-02	0.1327634E-01

Reproducibility failure

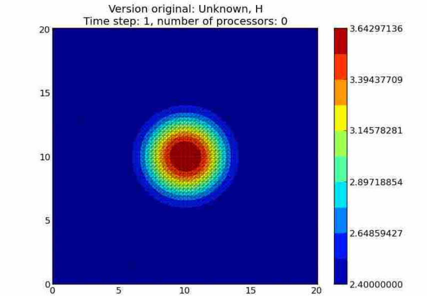
- Up to $\times 2.5$ uncertainty
- The privileged sequential run?

Telemac2D: the simplest gouttedo simulation

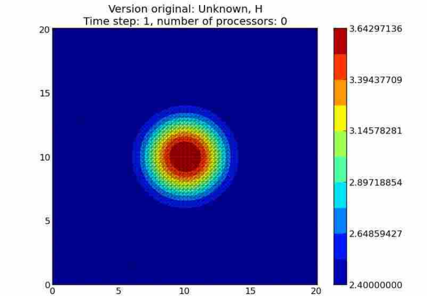
The gouttedo simulation test case

- 2D-simulation of a water drop fall in a square basin
- Unknown: water depth for a 0.2 sec time step
- Triangular mesh: 8978 elements and 4624 nodes

Expected numerical reproducibility (time step = 1, 2, ...)



Sequential

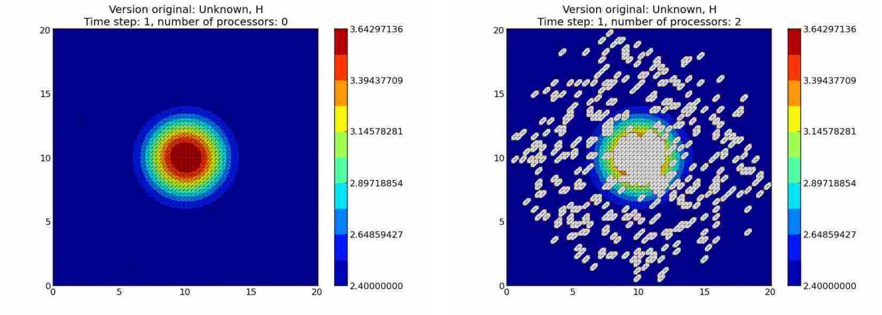


Parallel $p = 2$

A white plot displays a non-reproducible value

Numerical reproducibility?

time step $= 1$

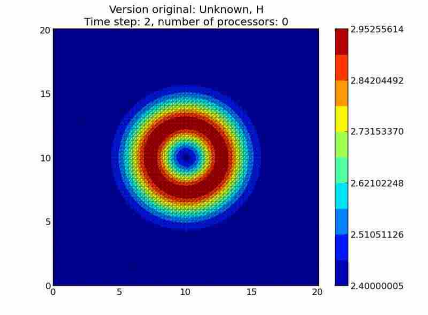


Sequential

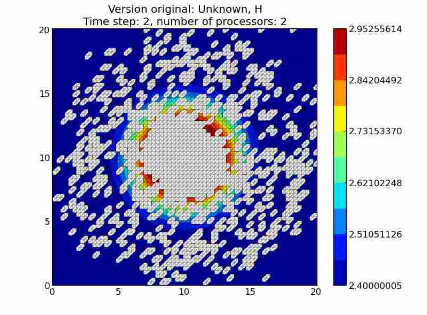
Parallel $p = 2$

Numerical reproducibility?

time step = 2



Sequential

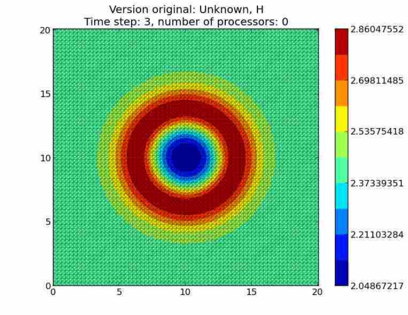


Parallel $p = 2$

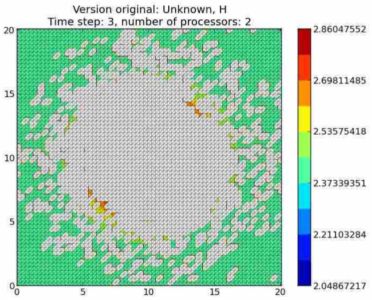
A white plot displays a non-reproducible value

Numerical reproducibility?

time step $= 3$



Sequential

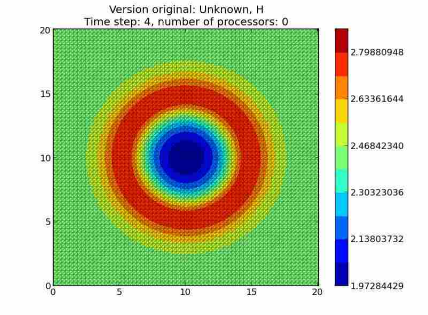


Parallel $p = 2$

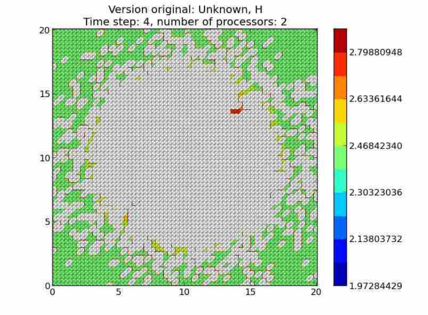
A white plot displays a non-reproducible value

Numerical reproducibility?

time step = 4



Sequential

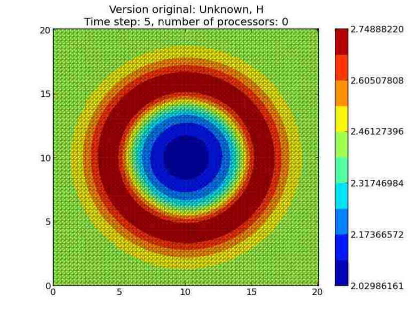


Parallel $p = 2$

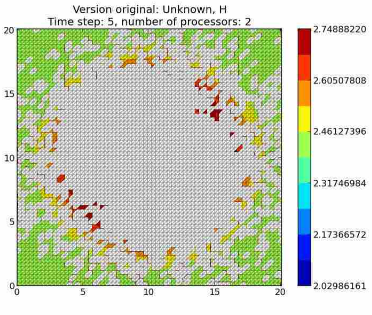
A white plot displays a non-reproducible value

Numerical reproducibility?

time step $= 5$



Sequential

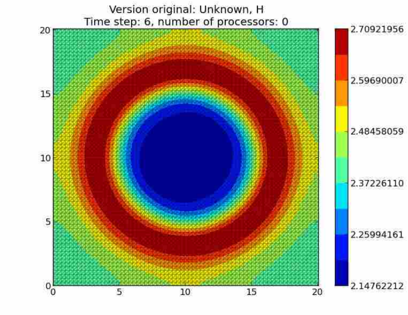


Parallel $p = 2$

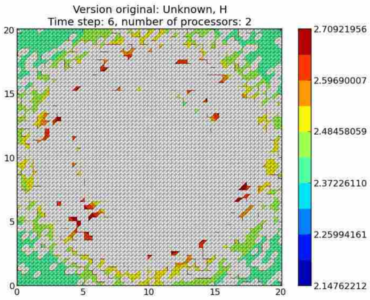
A white plot displays a non-reproducible value

Numerical reproducibility?

time step = 6



Sequential

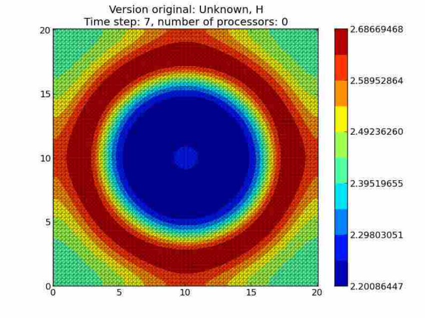


Parallel $p = 2$

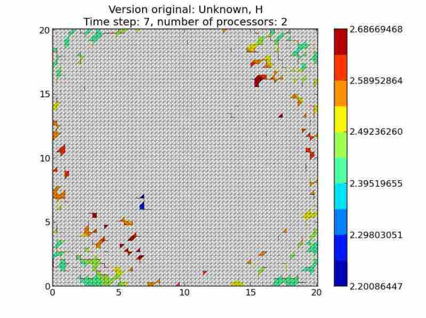
A white plot displays a non-reproducible value

Numerical reproducibility?

time step = 7



Sequential

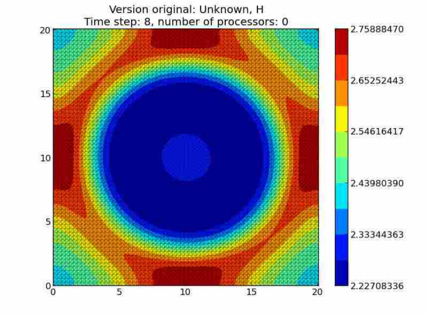


Parallel $p = 2$

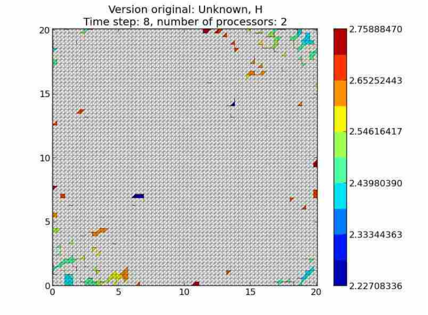
A white plot displays a non-reproducible value

Numerical reproducibility?

time step $= 8$



Sequential

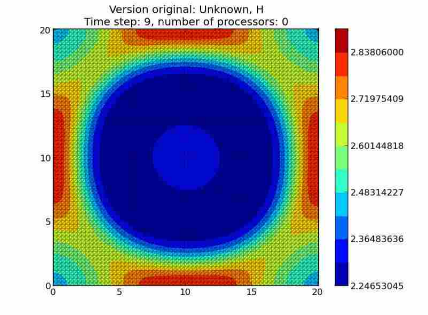


Parallel $p = 2$

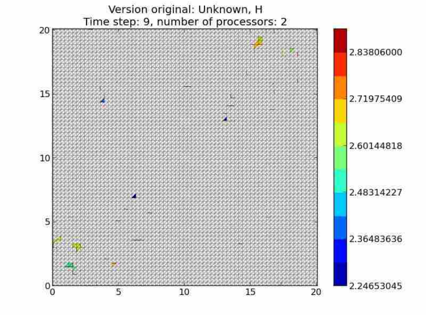
A white plot displays a non-reproducible value

Numerical reproducibility?

time step = 9



Sequential

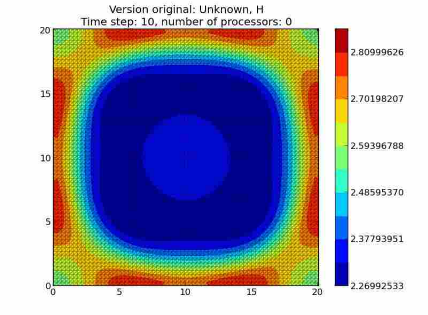


Parallel $p = 2$

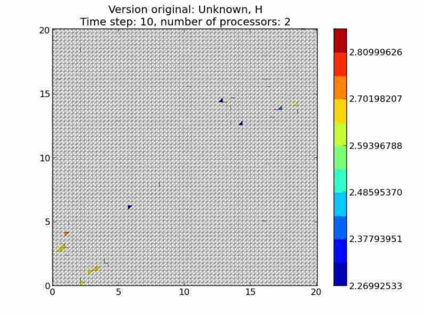
A white plot displays a non-reproducible value

Numerical reproducibility?

time step $= 10$



Sequential

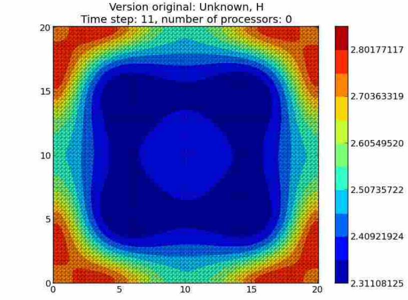


Parallel $p = 2$

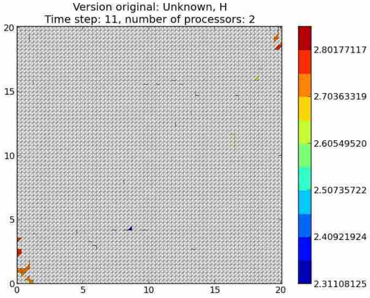
A white plot displays a non-reproducible value

Numerical reproducibility?

time step = 11



Sequential

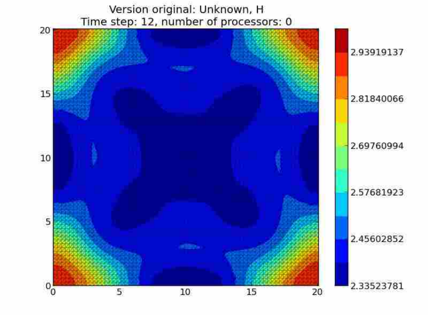


Parallel $p = 2$

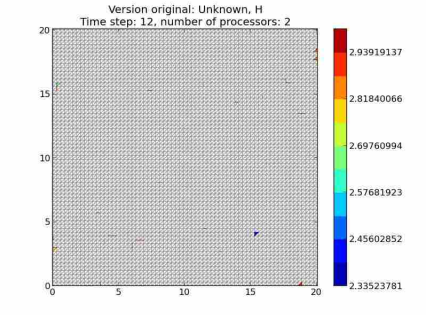
A white plot displays a non-reproducible value

Numerical reproducibility?

time step = 12



Sequential

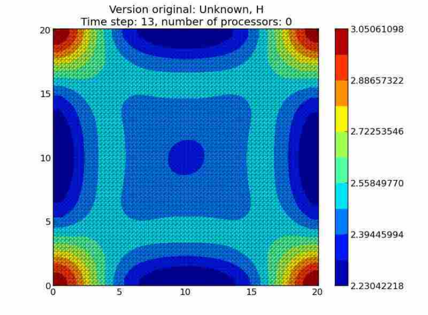


Parallel $p = 2$

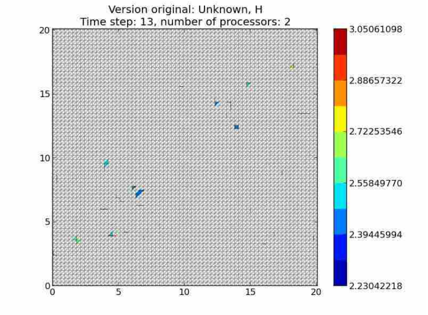
A white plot displays a non-reproducible value

Numerical reproducibility?

time step = 13



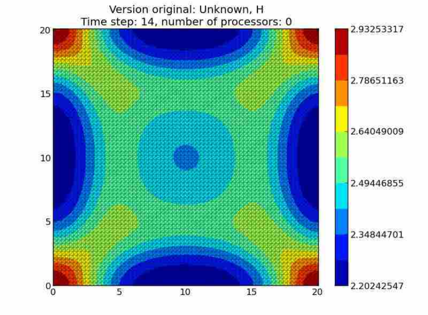
Sequential



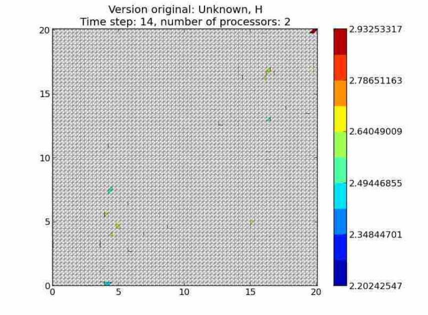
Parallel $p = 2$

Numerical reproducibility?

time step $= 14$



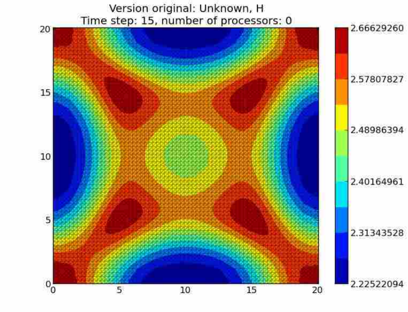
Sequential



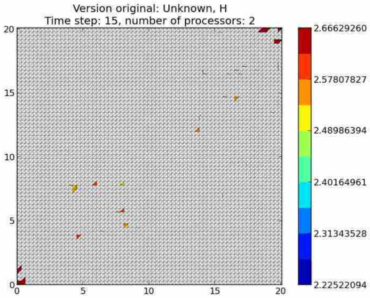
Parallel $p = 2$

NO numerical reproducibility!

time step = 15

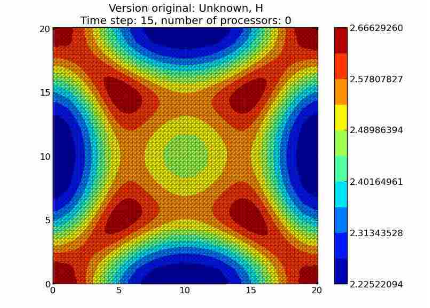


Sequential

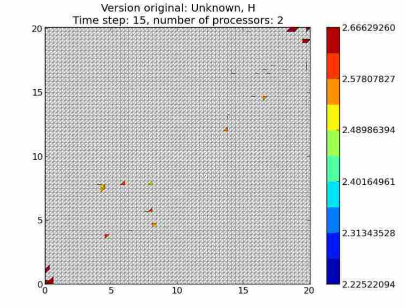


Parallel $p = 2$

NO numerical reproducibility!



Sequential



Parallel $p = 2$

Part 1: More accuracy

1. Summing n floating-point numbers: basic steps
   - Some algorithms
   - Some accuracy bounds
2. Computing sums more accurately
   - Some famous old and magic algorithms
   - Compensation
   - Distillate to understand and go further
3. More EFT and accurate algorithms
   - Some more results
4. More accuracy: conclusion

Part 2: More reproducibility

Part 3: More performance

Part I

More accuracy

1. Summing n floating-point numbers: basic steps
   - Some algorithms
   - Some accuracy bounds
2. Computing sums more accurately
   - Some famous old and magic algorithms
   - Compensation
   - Distillate to understand and go further
3. More EFT and accurate algorithms
   - Some more results
4. More accuracy: conclusion

How to manage accuracy and speed?

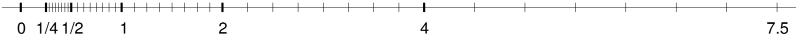
So many ways . . . and a new “best one” every year since 1999

1965 Møller, Ross
1969 Babuska, Knuth
1970 Nickel
1971 Dekker, Malcolm
1972 Kahan, Pichat
1974 Neumaier
1975 Kulisch/Bohlender
1977 Bohlender, Mosteller/Tukey
1981 Linnaimaa
1982 Leuprecht/Oberaigner
1983 Jankowski/Semoktunowicz/-Wozniakowski
1985 Jankowski/Wozniakowski
1987 Kahan

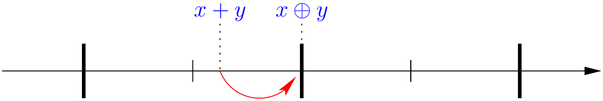
1991 Priest
1992 Clarkson, Priest
1993 Higham
1997 Shewchuk
1999 Anderson
2001 Hlavacs/Uberhuber
2002 Li et al. (XBLAS)
2003 Demmel/Hida, Nievergelt, Zielke/Drygalla
2005 Ogita/Rump/Oishi, Zhu/Yong/Zeng
2006 Zhu/Hayes
2008 Rump/Ogita/Oishi
2009 Rump, Zhu/Hayes
2010 Zhu/Hayes

Floating-point numbers (normal, non zero)

$$x = (-1)^s \cdot m \cdot 2^e = \pm \underbrace{1.x_1 x_2 \dots x_{p-1}}_{p \text{ bits of mantissa}} \times 2^e,$$



Rounding



- The standard model:
  $\mathrm{fl}(a \circ b) = (1 + \varepsilon)(a \circ b)$, with $|\varepsilon| \leq \mathbf{u} = 2^{-p}$, or 2^{1-p}.

IEEE-754 (1985, 2008)

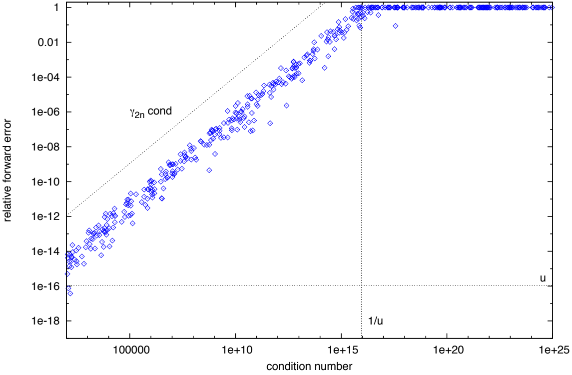
- formats, rounding modes : $+, -, \times, /, \sqrt{}$ are as accurate as possible, exceptions
- $\mathbf{u} = 2^{-53} \approx 10^{-16}$ for b64 in IEEE-754

Using the standard model

1. Summing n floating-point numbers: basic steps
   - Some algorithms
   - Some accuracy bounds
2. Computing sums more accurately
   - Some famous old and magic algorithms
   - Compensation
   - Distillate to understand and go further
3. More EFT and accurate algorithms
   - Some more results
4. More accuracy: conclusion

Accuracy for backward stable algorithms

- Accuracy of the computed sum $\leq (n-1) \times cond \times \mathbf{u}$
- $cond(\sum x_i) = \sum |x_i| / |\sum x_i|$
- No more significant digit in IEEE-b64 for large cond, *i.e.* $> 10^{16}$
- The length also matters for large n



Backward stable summation algorithms

Algorithm ($\mathcal{A}$ describes a class of summation algorithms)

Let $\mathcal{S} = \{x_1, x_2, \ldots, x_n\}$.

`while` $\mathcal{S}$ contains more than one element `do`

  Remove two numbers x and y from S and add $x \oplus y$ to S;

`end while`

Return the remaining element of S.

- Recursive summation...
- ...with increasing order sorting (IOS): $|x_1| \leq |x_2| \leq \cdots \leq |x_n|$,
- ...with decreasing order (DOS),
- insertion summation with IOS,
- pairwise summation.

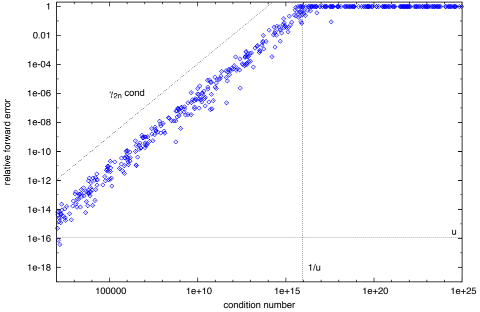
Accuracy of the computed sum $\leq (n-1)\sum |x_i|\mathbf{u}$

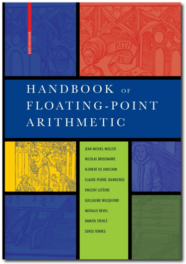
Proposition (Another bound)

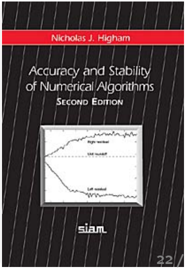
$|\widehat{s_n} - s_n| \leq \mathbf{u}\sum_{i=1}^{n-1}|T_i|$, where T_i is the i-th partial sum in $\mathcal{A}$.

- Minimize this bound or at least every $|T_i|$.
- All the x_i have the same sign: recursive with IOS (good), insertion with IOS (the best).
- Cancellation when $\sum |x_i| \gg |\sum x_i|$: DOS better than IOS.

It exists no universally better accurate summation algorithm







Computing sums more accurately: step 1

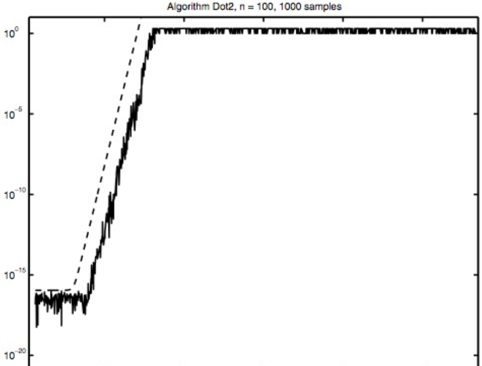
1. Summing n floating-point numbers: basic steps
   - Some algorithms
   - Some accuracy bounds
2. Computing sums more accurately
   - Some famous old and magic algorithms
   - Compensation
   - Distillate to understand and go further
3. More EFT and accurate algorithms
   - Some more results
4. More accuracy: conclusion

More accuracy but still conditioning: overview

Arbitrary precision but *accuracy* still condition dependant

- Simulating more *precision*: double-double, quad-double, ..., MPFR
- Compensated algorithms: Kahan(65), ..., Sum2(05), SumK(05)
- Accuracy $\lesssim \mathbf{u} + \mathit{cond} \times \mathbf{u}^K$
- No more n dependency while $n\mathbf{u} \leq 1$.

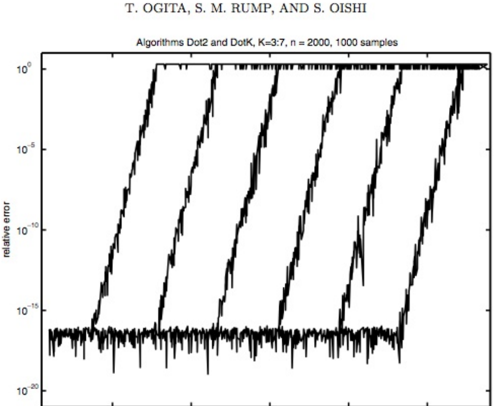
T. OGITA, S. M. RUMP, AND S. OISHI



More accuracy but still conditioning: overview

Arbitrary precision but *accuracy* still condition dependant

- Simulating more *precision*: double-double, quad-double, . . . , MPFR
- Compensated algorithms: Kahan(65), . . . , Sum2(05), SumK(05)
- Accuracy $\lesssim \mathbf{u} + \mathit{cond} \times \mathbf{u}^K$
- No more n dependency while $n\mathbf{u} \le 1$.



One old and famous solution

Algorithm (Kahan's compensated summation (1965))

$s = x_1$;

$c = 0$;

for $i = 2 : n$ do

$y = x_i \ominus c$;

$t = s \oplus y$;

$c = (t \ominus s) \ominus y$;

$s = t$

end for

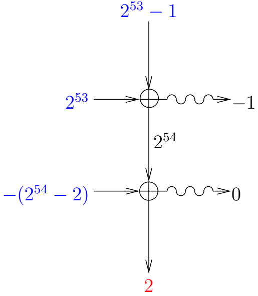
- Accuracy improvement in practice: $-c$ approximates $s \oplus y$'s rounding error.
- Compensated sum accuracy $\leq 2\mathbf{u} + n \sum |x_i| \mathcal{O}(\mathbf{u}^2)$.

One example

IEEE double precision numbers: $x_1 = 2^{53} - 1$, $x_2 = 2^{53}$ and $x_3 = -(2^{54} - 2)$.

Exact sum: $x_1 + x_2 + x_3 = 1$.

Classic summation



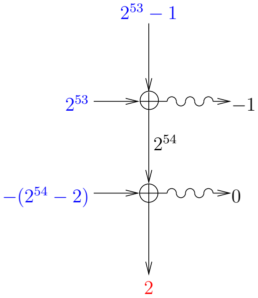
Relative error = 1

One example

IEEE double precision numbers: $x_1 = 2^{53} - 1$, $x_2 = 2^{53}$ and $x_3 = -(2^{54} - 2)$.

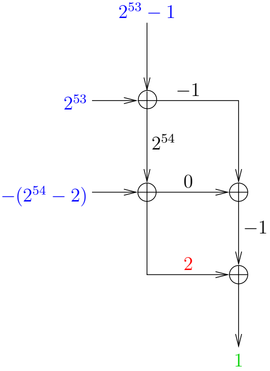
Exact sum: $x_1 + x_2 + x_3 = 1$.

Classic summation



Relative error = 1

Compensation of the rounding errors



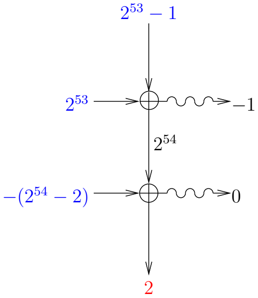
The exact result is computed

One example

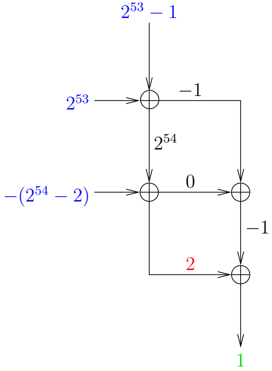
IEEE double precision numbers: $x_1 = 2^{53} - 1$, $x_2 = 2^{53}$ and $x_3 = -(2^{54} - 2)$.

Exact sum: $x_1 + x_2 + x_3 = 1$.

Classic summation



Compensation of the rounding errors



Relative error = 1

The exact result is computed

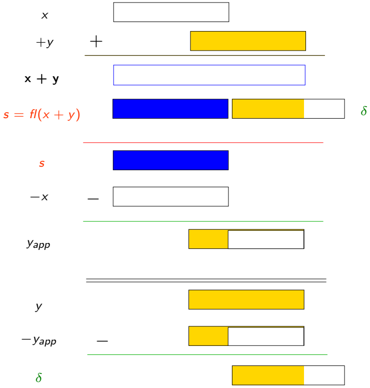
The rounding errors are computed thanks to *error-free transformations*.

1. Summing n floating-point numbers: basic steps
   - Some algorithms
   - Some accuracy bounds
2. Computing sums more accurately
   - Some famous old and magic algorithms
   - Compensation
   - Distillate to understand and go further
3. More EFT and accurate algorithms
   - Some more results
4. More accuracy: conclusion

The underlined idea of compensated algorithm

Compensation

1. Let's compute the generated rounding errors
2. and use it further to compensate the whole computation.



Compensating: the later the better?

Approximations in Kahan's compensated sum

- $x_1 = 2^{p+1}, x_2 = 2^{p+1} - 2, x_3 = x_4 = \cdots = x_6 = -(2^p - 1)$
  computed sum = 1, compensated sum = 3, exact sum = 2 (TP).
- $|s| \geq |y|$ or "exponent-wise" at least, round-to-nearest, order 2 rounding error in y.

Pichat and Neumaier's compensated summation (72,74)

- Accumulate rounding errors and one final correction.
- Compensated accuracy $\leq \mathbf{u}|\sum x_i| + (0.75n^2 + n)\mathbf{u}^2 \sum |x_i|$.
- No more cond, no initial sort and $n \leq n'_{max} = 1/3\mathbf{u}$.

Priest's doubly compensated summation (92)

- Doubly compensated accuracy $\leq 2\mathbf{u}|\sum x_i|$.
- No more cond but initial sort of the summands and $n \leq n_{max} = 1/4\mathbf{u}$.

Doubly compensated summation (92)

Algorithm (Priest (1992))

- Sort the x_i so that $|x_1| \geq |x_2| \geq \cdots \geq |x_n|$

$s_1 = x_1; c_1 = 0$

$\texttt{for}\ k = 2 : n\ \texttt{do}$

$\quad y_k = c_{k-1} + x_k;$

$\quad u_k = x_k - (y_k - c_{k-1});$

$\quad t_k = y_k + s_{k-1};$

$\quad v_k = y_k - (t_k - s_{k-1});$

$\quad z_k = u_k + v_k;$

$\quad s_k = t_k + z_k;$

$\quad c_k = z_k - (s_k - t_k)$

$\texttt{end for}$

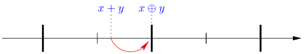
- Need to sort the entries (need to know them!)
- Doubly compensated sum: almost correctly rounded sum! accuracy $\leq 2\mathbf{u}|\sum x_i|$ when $n \leq 2^{p-3}$ (precision p).

Computing sums more accurately: step 2

1. Summing n floating-point numbers: basic steps
   - Some algorithms
   - Some accuracy bounds
2. Computing sums more accurately
   - Some famous old and magic algorithms
   - Compensation
   - Distillate to understand and go further
3. More EFT and accurate algorithms
   - Some more results
4. More accuracy: conclusion

Next step: towards the best accuracy

Distillation: iterate until faithful or correct rounding



- Error free transformation (EFT) $[x] \to [x^{(1)}] \to \cdots \to [x^*]$ s.t. $\sum x_i = \sum x_i^*$ et $[x^*]$ returns the expected rounded value.
- Kahan (87), . . . , Zhu-Hayes: iFastSum (SISC-09)

More space to keep everything

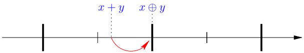
- Long accumulator, hardware oriented: Malcolm (71), Kulish (80)
- Cutting the summands: AccSum (SISC-08), FastAccSum (SISC-09)
- Summation at a given exponent: HybridSum (SISC-09), OnLineExact (TOMS-10)

From faithful rounding to correct rounding

- Choosing the "right side": expensive in the break-point neighborhood
- *e.g.* $1 + 2^{-53} \pm 2^{-106}$

Next step: towards the best accuracy

Distillation: iterate until faithful or correct rounding



- Error free transformation (EFT) $[x] \to [x^{(1)}] \to \cdots \to [x^*]$ s.t. $\sum x_i = \sum x_i^*$ et $[x^*]$ returns the expected rounded value.
- Kahan (87), ..., Zhu-Hayes: iFastSum (SISC-09)

More space to keep everything

- Long accumulator, hardware oriented: Malcolm (71), Kulish (80)
- Cutting the summands: AccSum (SISC-08), FastAccSum (SISC-09)
- Summation at a given exponent: HybridSum (SISC-09), OnLineExact (TOMS-10)

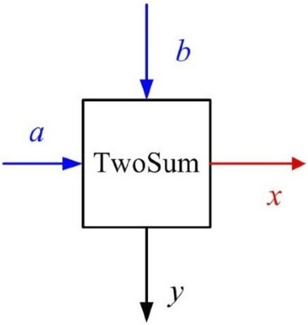
From faithful rounding to correct rounding

$\longrightarrow$ **Running-time and memory print are the discriminant factors**

EFT to sum two floating-point numbers

2Sum (Knuth, 65), Fast2Sum (Dekker, 71) for base ≤ 2 and RTN.

$$a + b = x + y, \text{ with } a, b, x, y \in \mathbb{F} \text{ and } x = a \oplus b.$$



Algorithm (Knuth)

function [x,y] = 2Sum(a,b)

$x = a \oplus b$

$z = x \ominus a$

$y = (a \ominus (x \ominus z)) \oplus (b \ominus z)$

Algorithm ($|a| > |b|$, Dekker)

function [x,y] = Fast2Sum(a,b)

$x = a \oplus b$

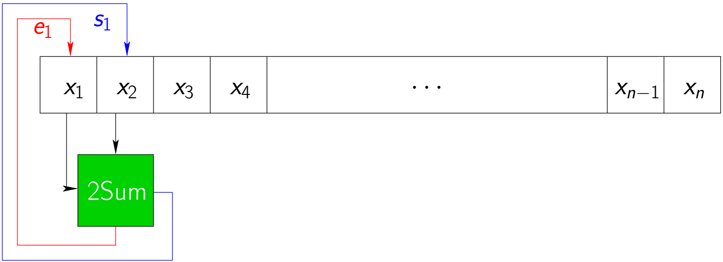
$z = x \ominus a$

$y = b \ominus z$

Distillate $\sum x_i, x_i \in \mathbb{F}$

x_1	x_2	x_3	x_4	$\cdots$	x_{n-1}	x_n

Distillate $\sum x_i, x_i \in \mathbb{F}$

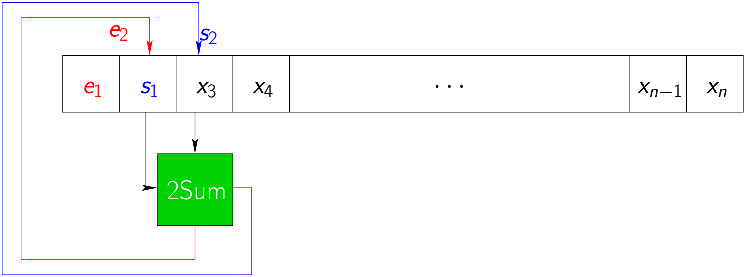


Distillate $\sum x_i, x_i \in \mathbb{F}$

e_1	s_1	x_3	x_4	$\cdots$	x_{n-1}	x_n

$$\sum_{1}^{n} x_i = e_1 + s_1 + \sum_{3}^{n} x_i$$

Distillate $\sum x_i, x_i \in \mathbb{F}$



Distillate $\sum x_i, x_i \in \mathbb{F}$

e_1	e_2	s_2	x_4	$\cdots$	x_{n-1}	x_n

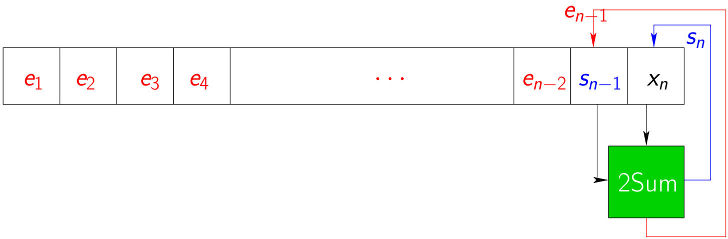
$$\sum_{1}^{n} x_i = e_1 + e_2 + s_2 + \sum_{4}^{n} x_i$$

Distillate $\sum x_i, x_i \in \mathbb{F}$

e_1	e_2	e_3	e_4	$\cdots$	e_{n-2}	s_{n-1}	x_n

$$\sum_{1}^{n} x_i = \sum_{1}^{n-2} e_i + s_{n-1} + x_n$$

Distillate $\sum x_i, x_i \in \mathbb{F}$



Distillate $\sum x_i, x_i \in \mathbb{F}$

e_1	e_2	e_3	e_4	$\cdots$	e_{n-2}	e_{n-1}	s_n

$$\sum_1^n x_i = \sum_1^{n-1} e_i + s_n = \sum_1^n x_i^{(1)}$$

and one iterate within this new vector $[x^{(1)}]$ ...

Let's distillate!

x_1	x_2	x_3	x_4	$\cdots$		x_{n-1}	x_n

$$\text{EFT} \Big\downarrow \text{Distillation}$$

x_1^*	x_2^*	x_3^*	x_4^*	$\cdots$		x_{n-1}^*	x_n^*

Theorem (Zhu-Hayes,09)

Terminating for IEEE-754 : Iterate the distillation $[x] \to [x^{(1)}] \to \cdots \to [x^{(k)}] \to \cdots$ *converges towards a stable state* $[x^*]$ *such that* $|x_1^*| < |x_2^*| < \cdots < |x_n^*|$, $\quad x_i^* \oplus x_{i+1}^* = x_{i+1}^*$ *et* $\sum x_i = \sum x_i^*$.

Rmk : the convergence proof uses the round-to-even tie breaking rule.

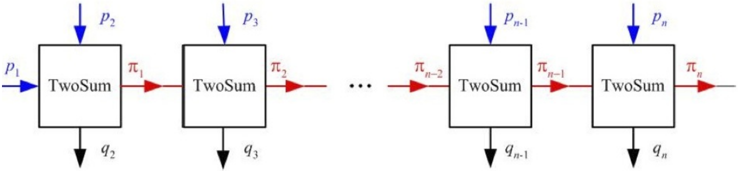
We need a good stopping criteria!

EFT to sum n floating-point numbers

Rien ne se perd, rien ne se crée, tout se transforme (Anaxagore–Lavoisier)

EFT vector transformation: (Ogita-Rump-Oishi, 05)

$[p_1, p_2, \cdots, p_n] \longmapsto [q_2, q_3, \cdots, q_n, \pi_n]$



For $(p_i)_{1 \le i \le n} \in \mathbb{F}$, $p_1 + p_2 = \pi_2 + q_2, p_2 + p_3 = \pi_3 + q_3, \cdots,$

$$\sum_{i=1}^{n} p_i = \pi_n + \sum_{i=2}^{n} q_i, \text{ with } \pi_n = \mathrm{fl}(\sum_{i=1}^{n} p_i).$$

A priori stopping criteria: go away conditioning!

Sum2, SumK (Ogita-Rump-Oishi,05)

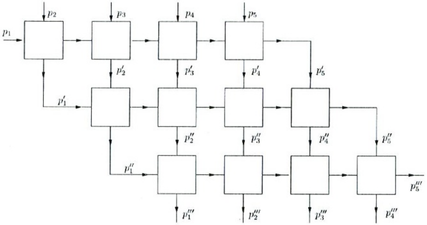


FIG. 4.2. *Outline of Algorithm 4.8 for $n = 5$ and $K = 4$.*

Theorem (OgRO,05)

For every p_i in $\mathbb{F}$,

$$\sum p_i = \sum p'_i = \cdots = \sum p_i^K = \cdots,$$

$$cond(\sum p_i^K) = \mathcal{O}(\mathbf{u}^K) \times \ cond(\sum p_i)$$

Accuracy $\lesssim \mathbf{u} + cond \times \mathbf{u}^K$

T. OGITA, S. M. RUMP, AND S. OISHI

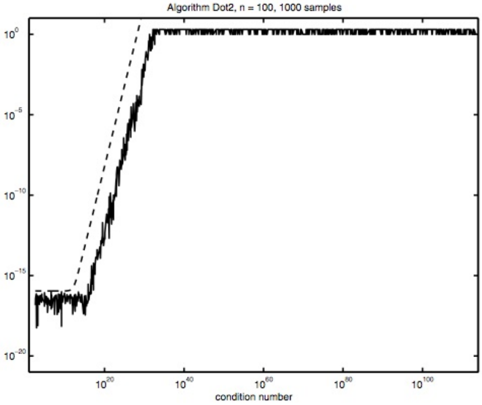
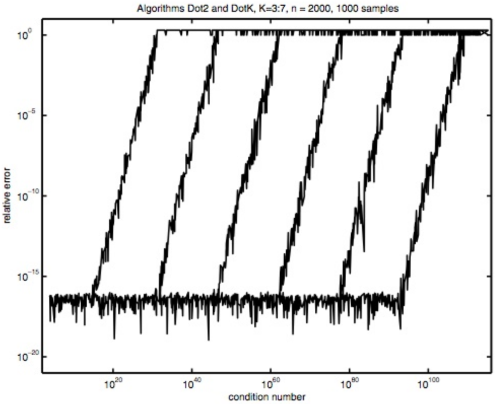


FIG. 6.1. *Test results for Algorithm* 5.3 (Dot2), $n = 100$, 1000 *samples.*

Accuracy $\lesssim \mathbf{u} + \textit{cond} \times \mathbf{u}^K$

T. OGITA, S. M. RUMP, AND S. OISHI



1. Summing n floating-point numbers: basic steps
   - Some algorithms
   - Some accuracy bounds
2. Computing sums more accurately
   - Some famous old and magic algorithms
   - Compensation
   - Distillate to understand and go further
3. More EFT and accurate algorithms
   - Some more results
4. More accuracy: conclusion

More accuracy: more results

Other EFT to compensate

- Multiplication: $a \times b = (a \otimes b) + e$
- FMA: $a \times b + c = FMA(a, b, c) + e_1 + e_2$, and $FMA(a, b, c) = RN(a \times b + c)$
- Polynomial evaluation

Other compensated algorithms

- Dot product
- Horner and derivative (dsynthetic div.), de Casteljau (Bernstein), Clenshaw (Chebychev)

Sterbenz's lemma: when subtraction is exact

Theorem (Sterbenz, 72)

In a radix-β floating-point system with subnormal numbers, if x and y are finite floating-point numbers such that

$$\frac{y}{2} \leq x \leq 2y,$$

then $x - y$ is a floating-point number.

Comments

- Exact subtraction for the four IEEE-754 rounding-modes
- Exact subtraction *vs.* catastrophic cancellation?
- Used in the previous EFT sum proofs

EFT: the multiplication case

Theorem (Dekker)

The multiply rounding-error $xy - (x \otimes y)$ is a floating-point number when no overflow occurs.

When one FMA is available, next algorithm computes the multiply-EFT:

$$xy = r_1 + r_2.$$

Algorithm (2ProdFMA)

$r_1 = FMA(x, y, 0); \qquad // r_1 = x \otimes y$

$r_2 = FMA(x, y, -r_1); \qquad // r_2 = xy - r_1$

The multiplication case when no available FMA

Let $C = \beta^s + 1$ where $s < p$, the next algo splits the p-digit floating-point number x in two parts x_h, x_l of $p - s$ and s digits.

Algorithm (split - Veltkamp, 68)

$r\_shift = C \otimes x$

$head = x \ominus r\_shift$

$x_h = r\_shift \oplus head$

$x_l = x \ominus x_h$

Theorem (Dekker, 72; Boldo, 06)

When no overflow occurs,

$$x = x_h + x_l.$$

The multiplication case when no available FMA

When no overflow occurs, next algorithm computes the multiply-EFT:

$$xy = r_1 + r_2.$$

Algorithm (2prod - Dekker, 72)

$(x_h, x_l) = split(x);$

$(y_h, y_l) = split(y);$

$r_1 = x \otimes y;$

$high = -r_1 \oplus (x_h \otimes y_h);$

$mid_1 = high \oplus (x_h \otimes y_l);$

$mid_2 = mid_1 \oplus (x_l \otimes y_h);$

$r_2 = mid_2 \oplus (x_l \otimes y_l);$ //low part

The compensated dot product (Ogita-Rump-Oishi, 05)

EFT-DotProd: $\sum_{i=1}^{n} x_i y_i = p + \sum_{i=1}^{n} \pi_i + \sum_{i=1}^{n-1} \sigma_i = \sum_{i=1}^{2n} z_i$.

Algorithm (Compensated DotProd)

$(s_1, c_1) = 2Prod(x_1, y_1)$;

`for` $i = 2 : n$ `do`

  $(p_i, \pi_i) = 2Prod(x_i, y_i)$;

  $(s_i, \sigma_i) = 2Sum(p_i, s_{i-1})$;

  $c_i = c_{i-1} \oplus (\pi_i \oplus \sigma_i)$;

`end for`

`return` $s_n \oplus c_n$;

Theorem (Ogita-Rump-Oishi, 05)

Compensated dot product accuracy $\leq \mathbf{u} |\sum_{i=1}^{n} x_i y_i| + \mathcal{O}(n^2 \mathbf{u}^2) \sum |x_i y_i|$.

The compensated dot product (Ogita-Rump-Oishi, 05)

T. OGITA, S. M. RUMP, AND S. OISHI

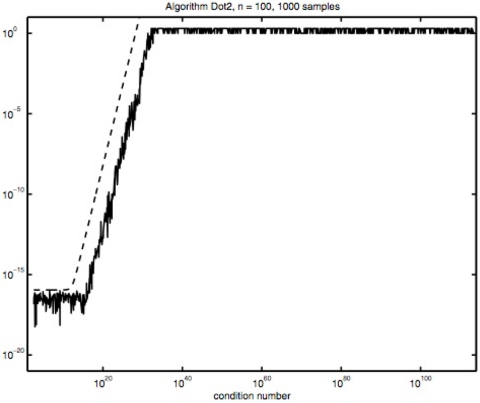
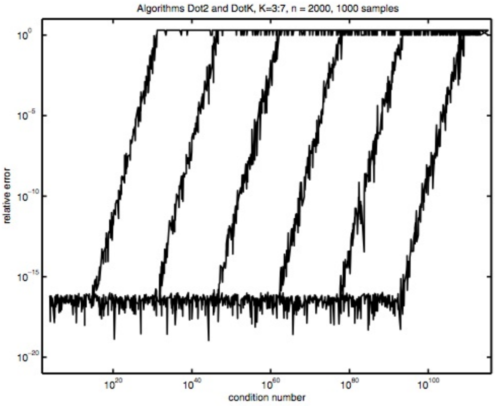


FIG. 6.1. *Test results for Algorithm* 5.3 (`Dot2`), $n = 100$, 1000 *samples.*

The compensated dot product (Ogita-Rump-Oishi, 05)

T. OGITA, S. M. RUMP, AND S. OISHI



The Horner polynomial evaluation (Graillat-Langlois-Louvet, 07)

EFT-Horner: $\sum_{i=0}^{n} a_i x^i = p + \sum_{i=0}^{n-1} \pi_i x^i + \sum_{i=0}^{n-1} \sigma_i y^i = p + \pi(x) + \sigma(x)$.

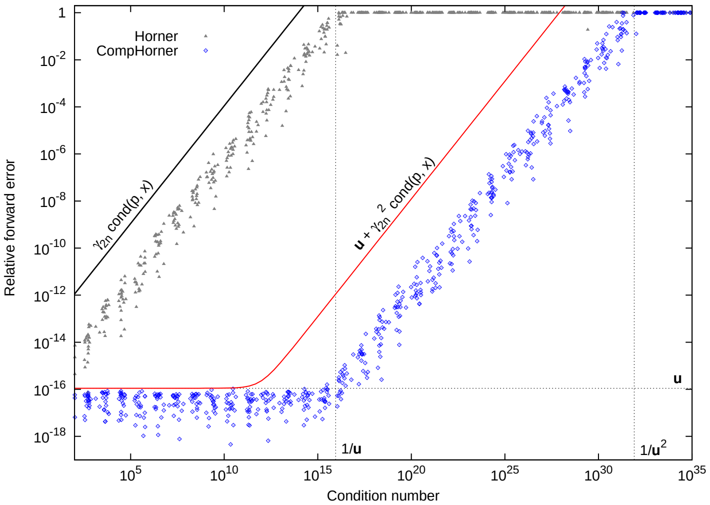
Algorithm (Compensated Horner)

$r_n = a_n$;
`for` $i = n - 1 : (-1) : 0$ `do`
  $(p_i, \pi_i) = 2Prod(r_{i+1}, x)$;
  $(r_i, \sigma_i) = 2Sum(p_i, a_i)$;
  $q_i = \pi_i \oplus \sigma_i$; //correcting polynomial coefficient
`end for`;
$r = r_0 \oplus Horner(q, x)$;
`return` r;

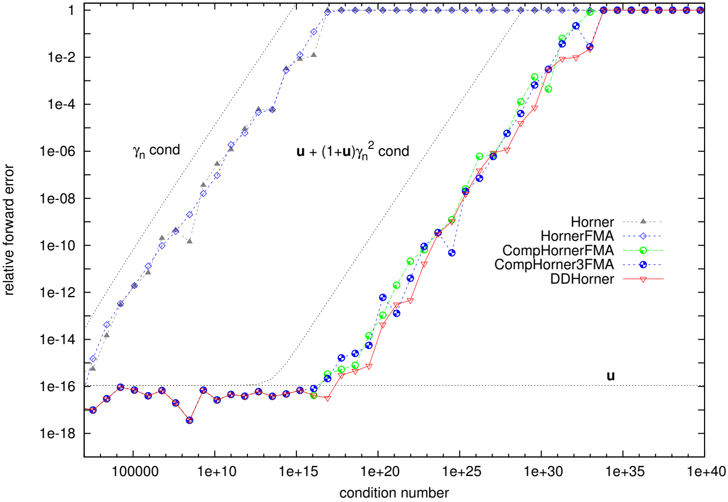
Theorem (Langlois-Louvet, 07)

Compensated Horner accuracy $\leq \mathbf{u}|p(x)| + \mathcal{O}(4n^2\mathbf{u}^2) \sum_{i=0}^{n} |a_i||x^i|$.

The compensated Horner evaluation (Louvet's PhD, 07)



The compensated Horner evaluation: FMA variations



Other compensated polynomial evaluations

Recent extensions

- derivative Horner evaluation — synthetic division (Jiang et al., 12)
- Clenshaw algorithm for Chebychev basis (Jiang et al., 11)
- de Casteljau algorithm for Bernstein basis (Jiang et al., 10)

To conclude this Part 1

1. Summing n floating-point numbers: basic steps
   - Some algorithms
   - Some accuracy bounds
2. Computing sums more accurately
   - Some famous old and magic algorithms
   - Compensation
   - Distillate to understand and go further
3. More EFT and accurate algorithms
   - Some more results
4. More accuracy: conclusion

More accurate sums: conclusion

An "ultimately accurate" floating-point summation algorithm cannot be very simple. *J.M. Muller et al. [HFPA,09]*

- A lot of algorithms!
- Condition dependency and accuracy bounds for classic ones
- Arbitrarily accurate compensated ones: Sum2, SumK
- Faithfully or correctly rounded recent ones: AccSum, FastAccSum, iFastSum, HybdridSum, OnLineExact
- "In place" solutions vs. exponent manipulations

How to choose? Running time performance!

References I

D. H. Bailey.
Twelve ways to fool the masses when giving performance results on parallel computers.
Supercomputing Review, pages 54–55, Aug. 1991.

J. W. Demmel and H. D. Nguyen.
Fast reproducible floating-point summation.
In *Proc. 21th IEEE Symposium on Computer Arithmetic*. Austin, Texas, USA, 2013.

B. Goossens, P. Langlois, D. Parello, and E. Petit.
PerPI: A tool to measure instruction level parallelism.
In K. Jónasson, editor, *Applied Parallel and Scientific Computing - 10th International Conference, PARA 2010, Reykjavík, Iceland, June 6-9, 2010, Revised Selected Papers, Part I*, volume 7133 of *Lecture Notes in Computer Science*, pages 270–281. Springer, 2012.

Y. He and C. Ding.
Using accurate arithmetics to improve numerical reproducibility and stability in parallel applications.
J. Supercomput., 18:259–277, 2001.

References II

J.-M. Hervouet.
Hydrodynamics of free surface flows: Modelling with the finite element method.
John Wiley & Sons, 2007.

N. J. Higham.
Accuracy and Stability of Numerical Algorithms.
SIAM, 2nd edition, 2002.

H. Jiang, R. Barrio, H. Li, X. Liao, L. Cheng, and F. Su.
Accurate evaluation of a polynomial in chebyshev form.
Applied Mathematics and Computation, 217(23):9702 – 9716, 2011.

H. Jiang, S. Graillat, C. Hu, S. Li, X. Liao, L. Cheng, and F. Su.
Accurate evaluation of the k-th derivative of a polynomial and its application.
J. Comput. Appl. Math., 243:28–47, May 2013.

H. Jiang, S. Li, L. Cheng, and F. Su.
Accurate evaluation of a polynomial and its derivative in bernstein form.
Computers & Mathematics with Applications, 60(3):744 – 755, 2010.

References III

P. Langlois and N. Louvet.

How to ensure a faithful polynomial evaluation with the compensated Horner algorithm?

In P. Kornerup and J.-M. Muller, editors, *18th IEEE International Symposium on Computer Arithmetic*, number ISBN 0-7695-2854-6, pages 141–149. IEEE Computer Society, June 2007.

P. Langlois, R. Nheili, and C. Denis.

Numerical Reproducibility: Feasibility Issues.

In *NTMS'2015: 7th IFIP International Conference on New Technologies, Mobility and Security*, pages 1–5, Paris, France, July 2015. IEEE, IEEE COMSOC & IFIP TC6.5 WG.

P. Langlois, R. Nheili, and C. Denis.

Recovering numerical reproducibility in hydrodynamic simulations.

In J. H. P. Montuschi, M. Schulte, S. Oberman, and N. Revol, editors, *23rd IEEE International Symposium on Computer Arithmetic*, number ISBN 978-1-5090-1615-0, pages 63–70. IEEE Computer Society, July 2016.

(Silicon Valley, USA. July 10-13 2016).

References IV

N. Louvet.

Algorithmes compensés en arithmétique flottante : précision, validation, performances.

PhD thesis, Université de Perpignan Via Domitia, Nov. 2007.

J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres.

Handbook of Floating-Point Arithmetic.

Birkhäuser Boston, 2010.

T. Ogita, S. M. Rump, and S. Oishi.

Accurate sum and dot product.

SIAM J. Sci. Comput., 26(6):1955–1988, 2005.

Open TELEMAC-MASCARET. v.7.0, Release notes.

`www.opentelemac.org`, 2014.

R. W. Robey, J. M. Robey, and R. Aulwes.

In search of numerical consistency in parallel programming.

Parallel Comput., 37(4-5):217–229, 2011.

References V

S. M. Rump.
Ultimately fast accurate summation.
SIAM J. Sci. Comput., 31(5):3466–3502, 2009.

S. M. Rump, T. Ogita, and S. Oishi.
Accurate floating-point summation – part I: Faithful rounding.
SIAM J. Sci. Comput., 31(1):189–224, 2008.

M. Taufer, O. Padron, P. Saponaro, and S. Patel.
Improving numerical reproducibility and stability in large-scale numerical simulations on gpus.
In *IPDPS*, pages 1–9. IEEE, 2010.

O. Villa, D. G. Chavarría-Miranda, V. Gurumoorthi, A. Márquez, and S. Krishnamoorthy.
Effects of floating-point non-associativity on numerical computations on massively multithreaded systems.
In *CUG 2009 Proceedings*, pages 1–11, 2009.

References VI

V. Weaver and J. Dongarra.
Can hardware performance counters produce expected, deterministic results?
In *3rd Workshop on Functionality of Hardware Performance Monitoring, 2010*, pages 1–11, Atlanta, USA, 2010.

D. Zaparanuks, M. Jovic, and M. Hauswirth.
Accuracy of performance counter measurements.
In *IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2009, April 26-28, 2009, Boston, Massachusetts, USA*, pages 23–32, 2009.

Y.-K. Zhu and W. B. Hayes.
Correct rounding and hybrid approach to exact floating-point summation.
SIAM J. Sci. Comput., 31(4):2981–3001, 2009.

Y.-K. Zhu and W. B. Hayes.
Algorithm 908: Online exact summation of floating-point streams.
ACM Trans. Math. Software, 37(3):37:1–37:13, Sept. 2010.

version: April 9, 2018