Ecole thématique PRECIS 2017, Fréjus, 15-19 mai 2017

Précision et Reproductibilité en Calcul et Informatique Scientifique :

garantir la reproductibilité numérique concilier précision et performance

Philippe Langlois, David Parello

DALI, University of Perpignan Via Domitia

LIRMM, UMR 5506 CNRS-UM, France

Acknowledgement and collaboration

Joint work with (past or current) members of the DALI research team in University of Perpignan:

Prof. B. Goossens,
Dr. N. Louvet,
Dr. D. Parello,
Ch. Chohra,
R. Nheili,
C. Porada.

and colleagues:

Ch. Denis (EDF R&D, CMLA),
J.-M. Hervouet (EDF R&D).

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.

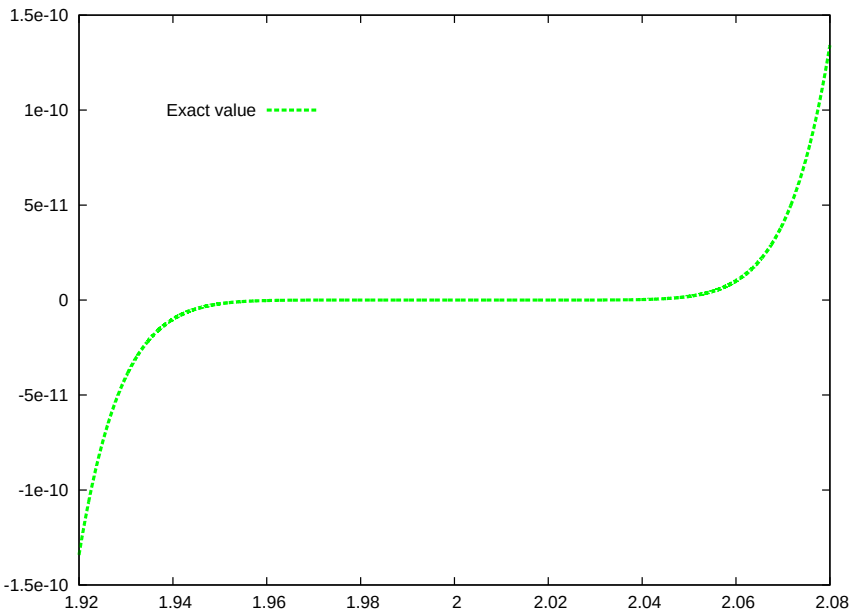
Rounding errors may totally corrupt a floating point computation:

- 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

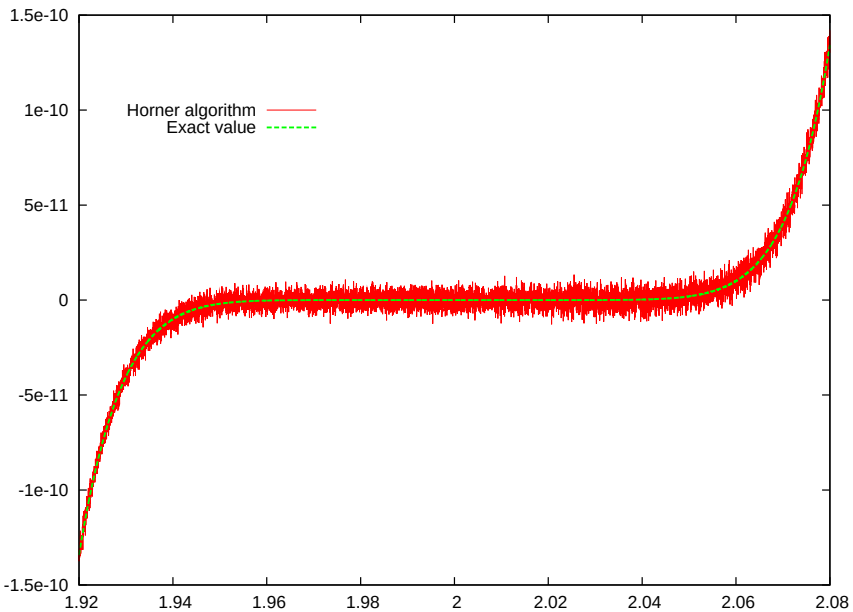


$p(x) = (x-2)^9$ in expanded form
near the multiple root $x = 2$

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

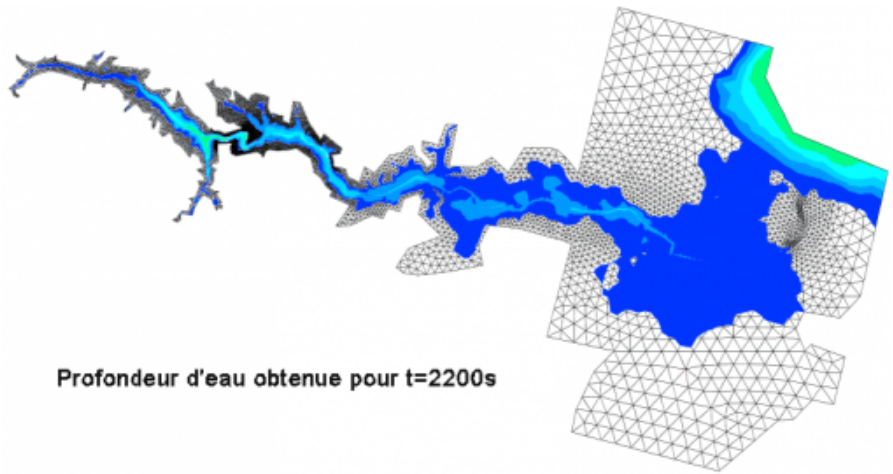
Telemac 2D [5]

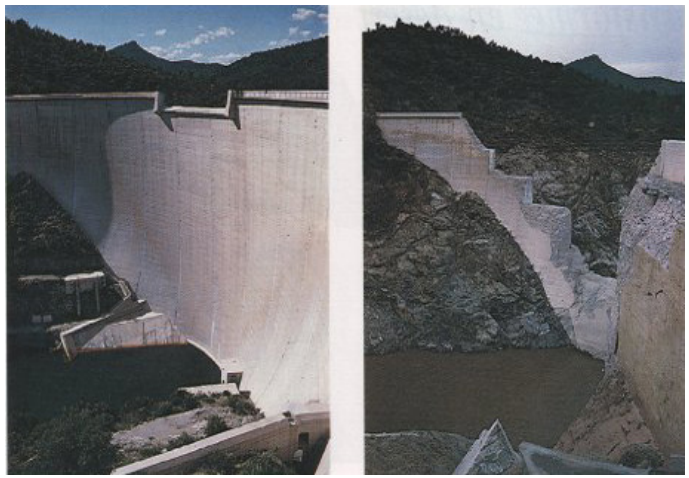
- 2D hydrodynamic: Saint Venant equations
- Finite element method, triangular element mesh, sub-domain decomposition for parallel resolution
- Mesh node unknowns: water depth (H) and velocity (U,V)

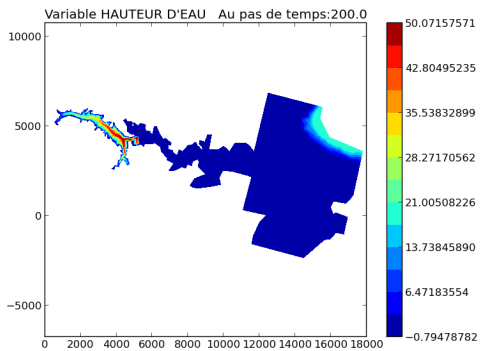
A complex Telemac 2D simulation

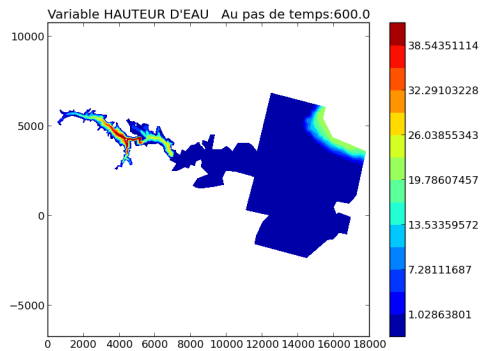
The Malpasset dam break (1959)

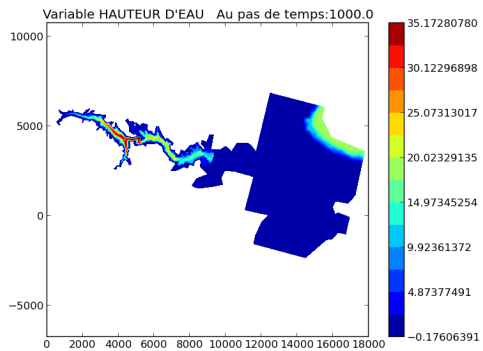
- A five year old dam break: 433 dead people and huge damage
- Triangular mesh: 26000 elements and 53000 nodes
- Simulation: $\rightarrow$35min. after break with a 2sec. time step



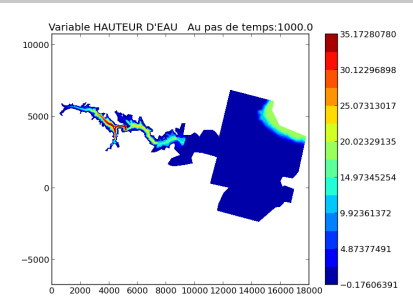








How to trust this complex simulation?



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

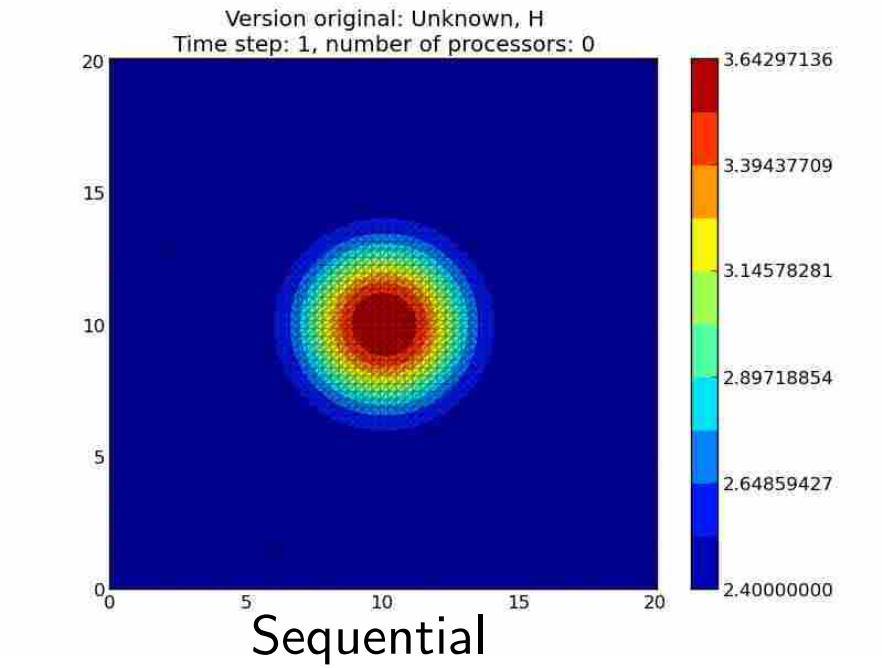
- Up to $\times 2.5$ uncertainty
- The privileged sequential run?

Telemac2D: the simplest gouttedo simulation

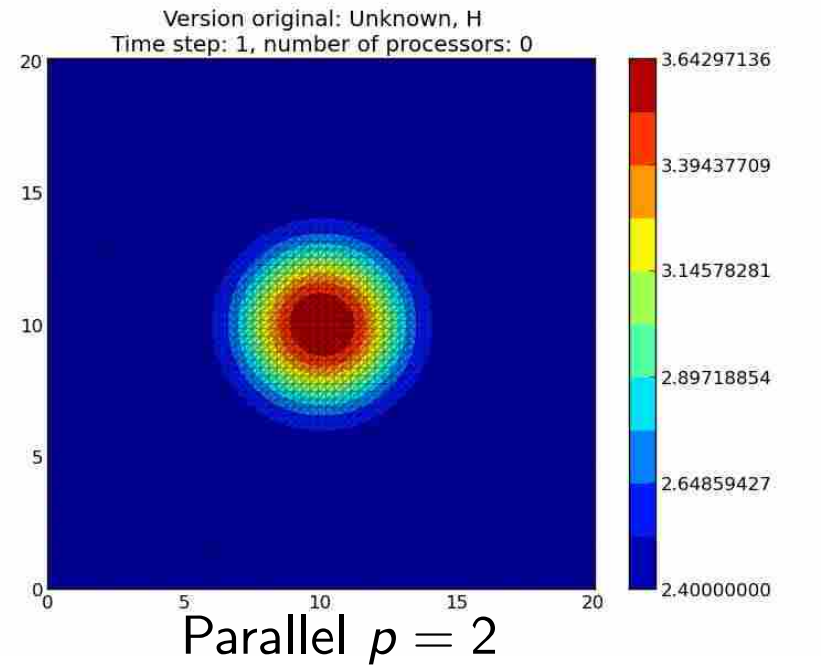
The gouttedo simulation test case

- 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

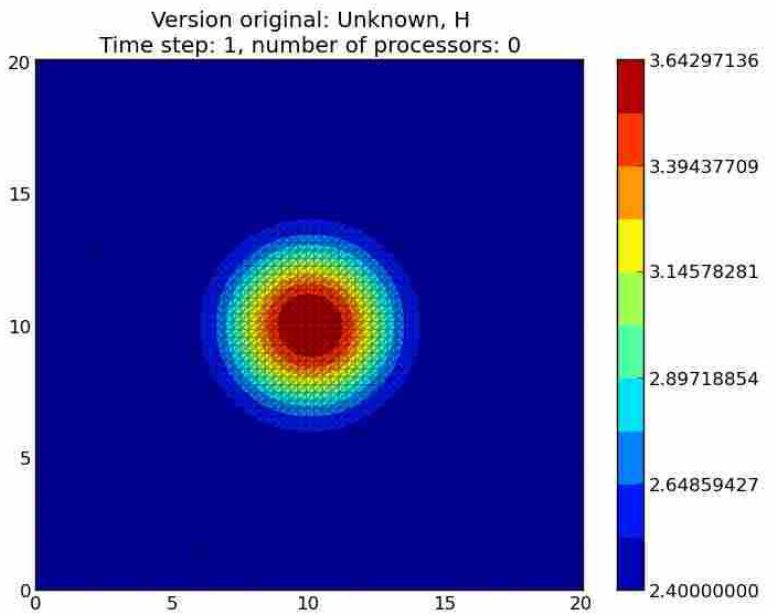


Parallel $p = 2$

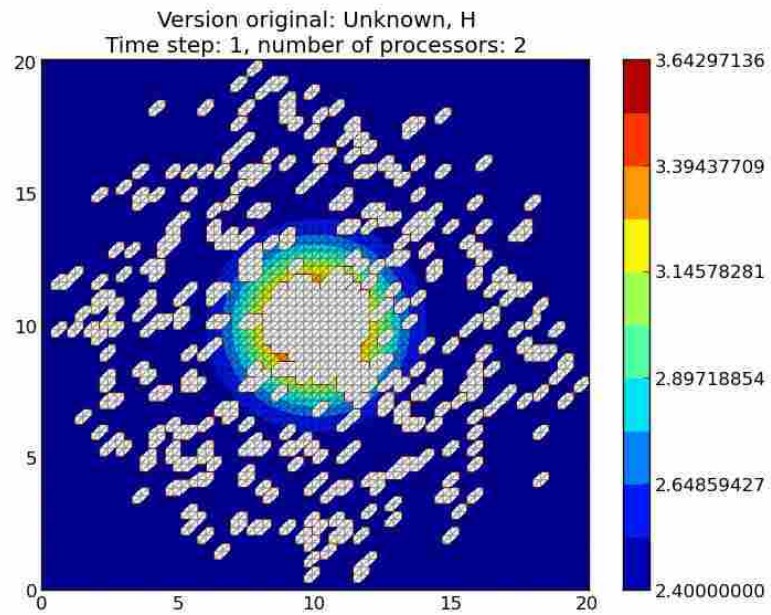
A white plot displays a non-reproducible value

Numerical reproducibility?

time step $= 1$



Sequential

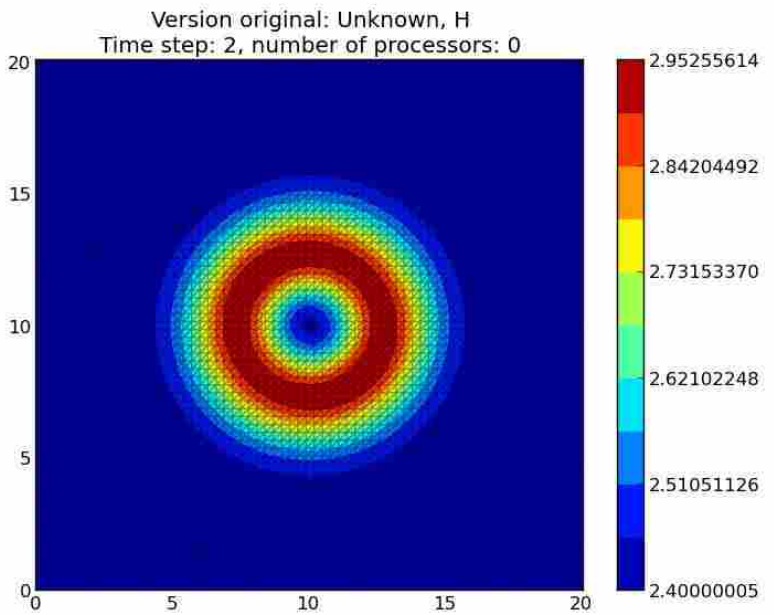


Parallel $p = 2$

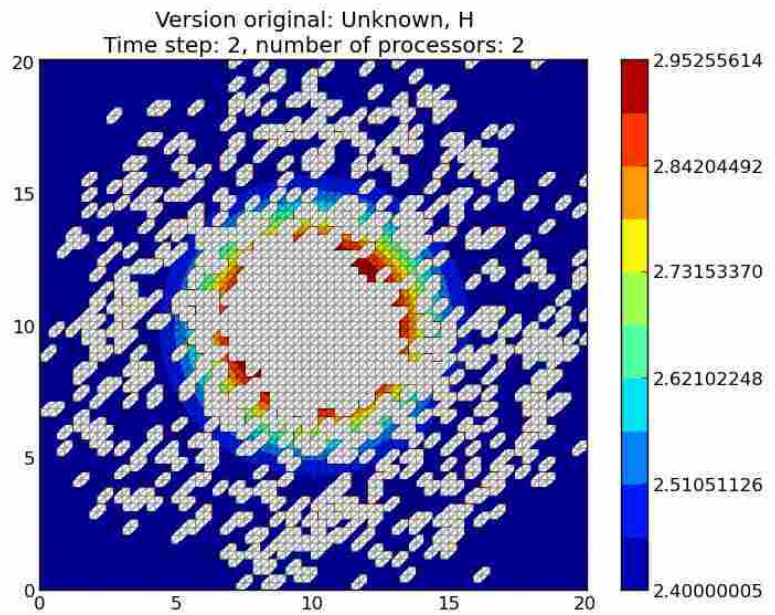
A white plot displays a non-reproducible value

Numerical reproducibility?

time step = 2



Sequential

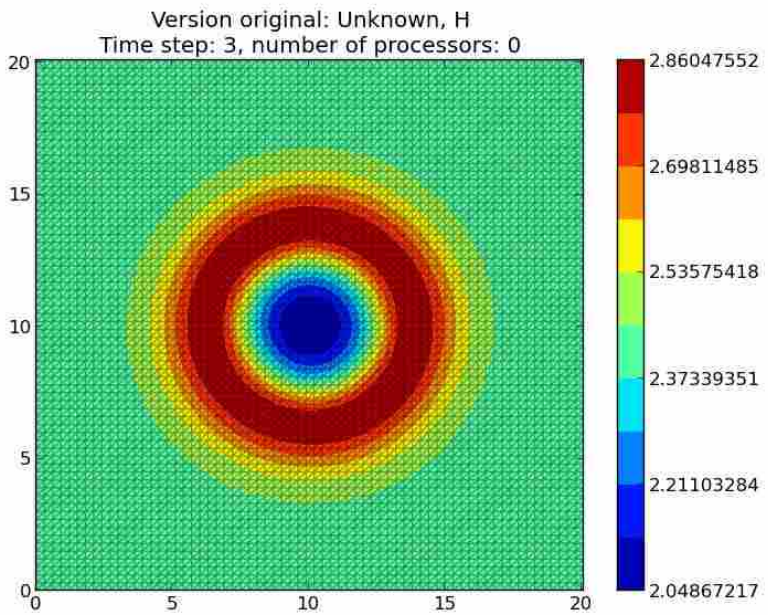


Parallel $p = 2$

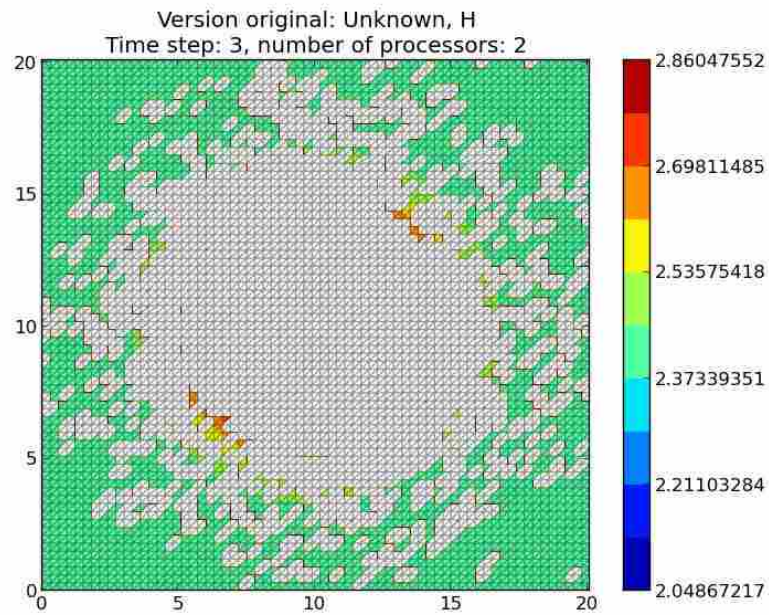
A white plot displays a non-reproducible value

Numerical reproducibility?

time step = 3



Sequential

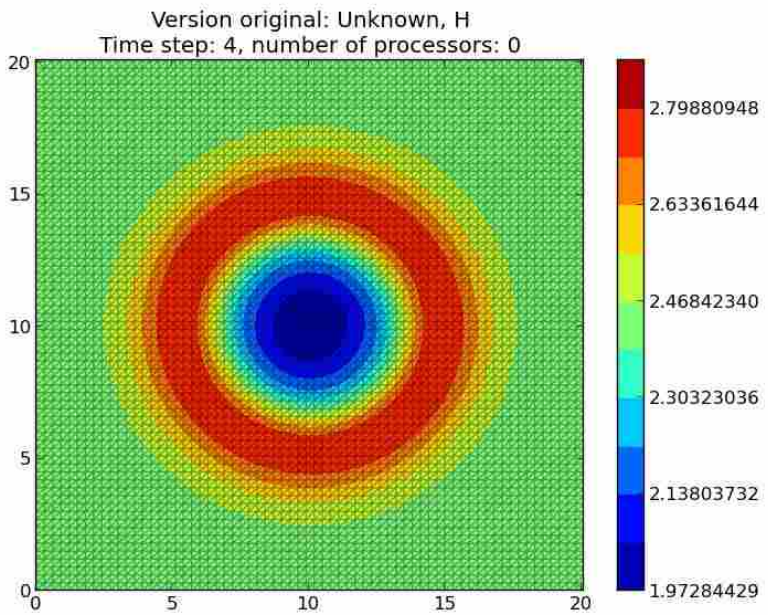


Parallel $p = 2$

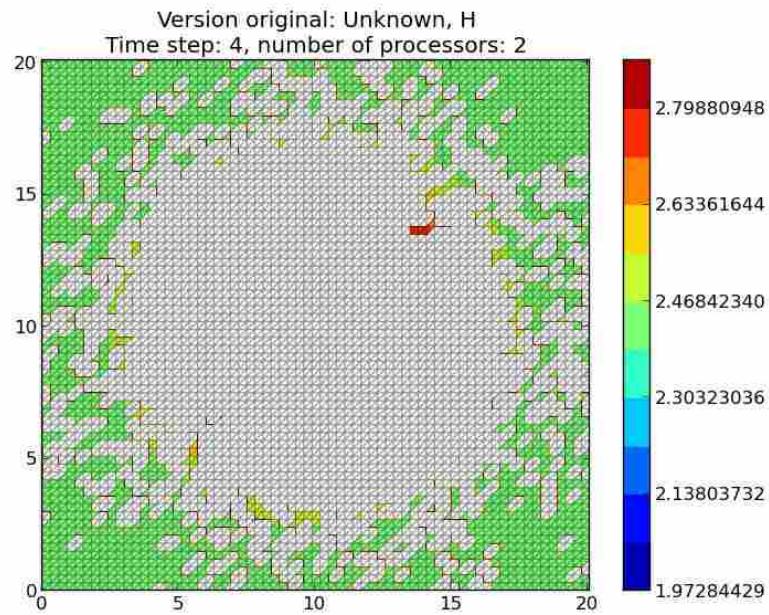
A white plot displays a non-reproducible value

Numerical reproducibility?

time step $= 4$



Sequential

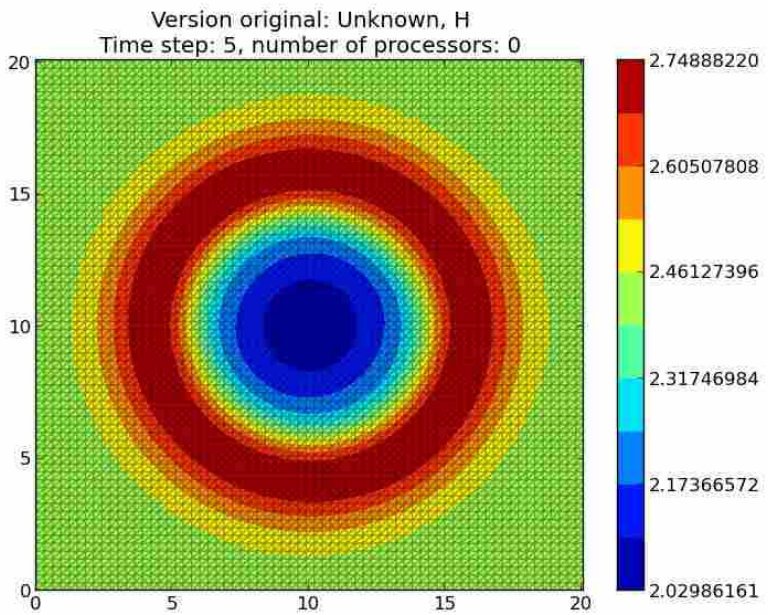


Parallel $p = 2$

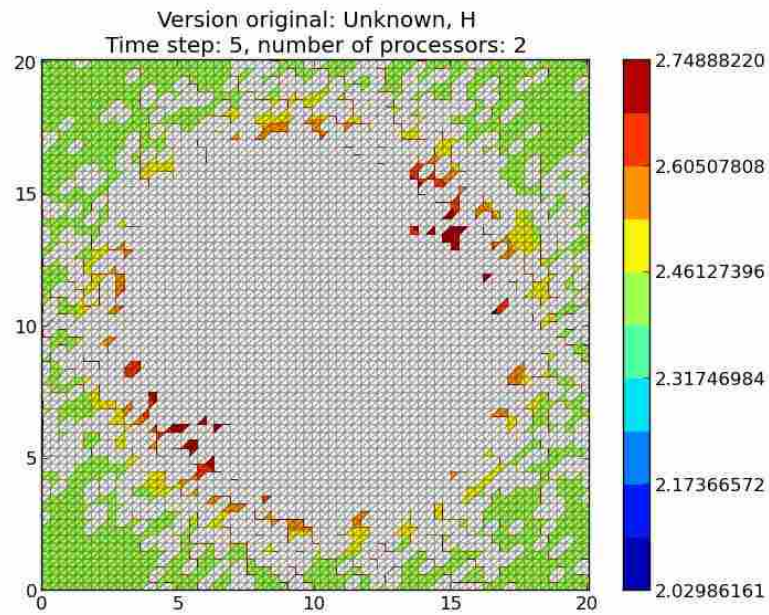
A white plot displays a non-reproducible value

Numerical reproducibility?

time step = 5



Sequential

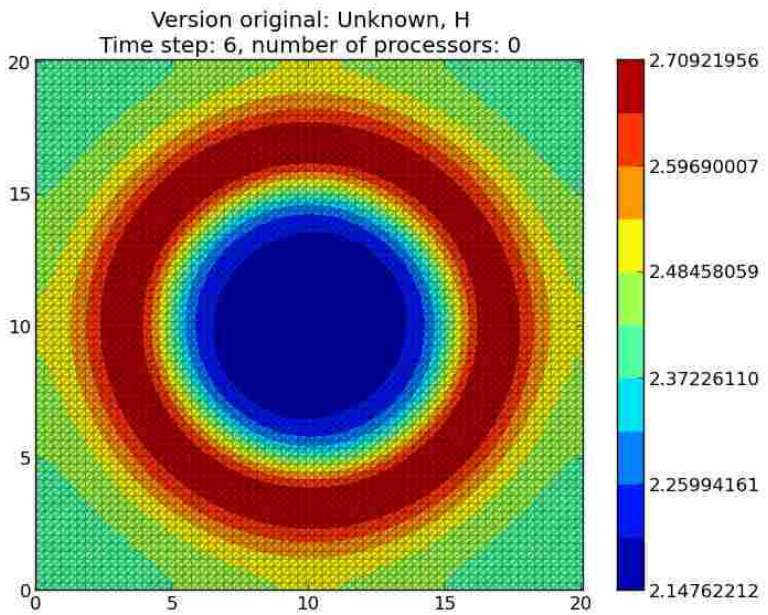


Parallel $p = 2$

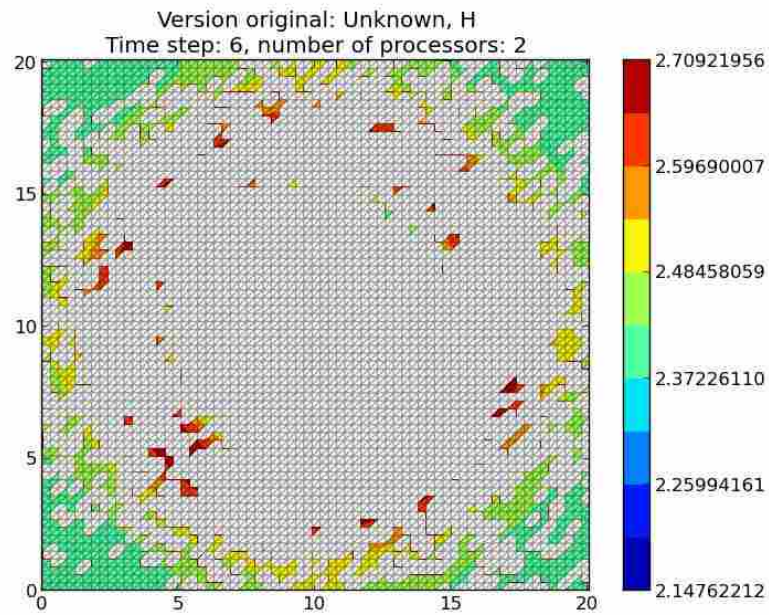
A white plot displays a non-reproducible value

Numerical reproducibility?

time step = 6



Sequential

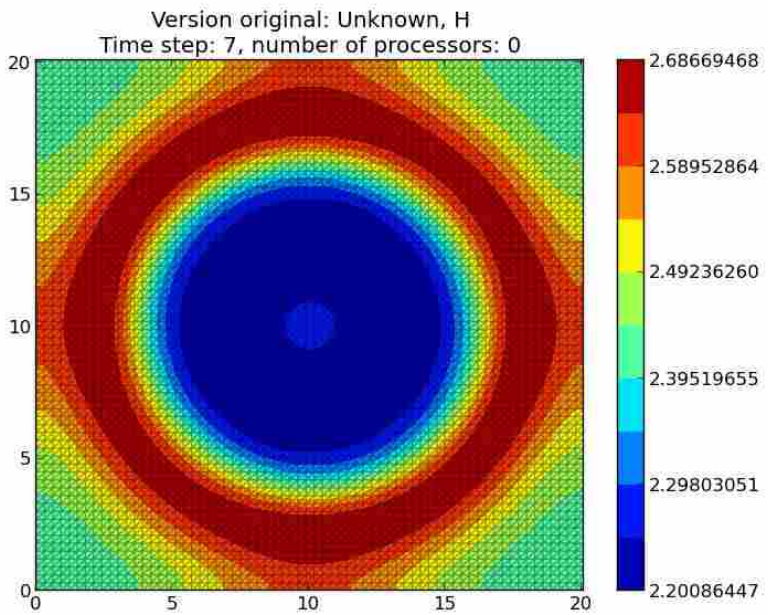


Parallel $p = 2$

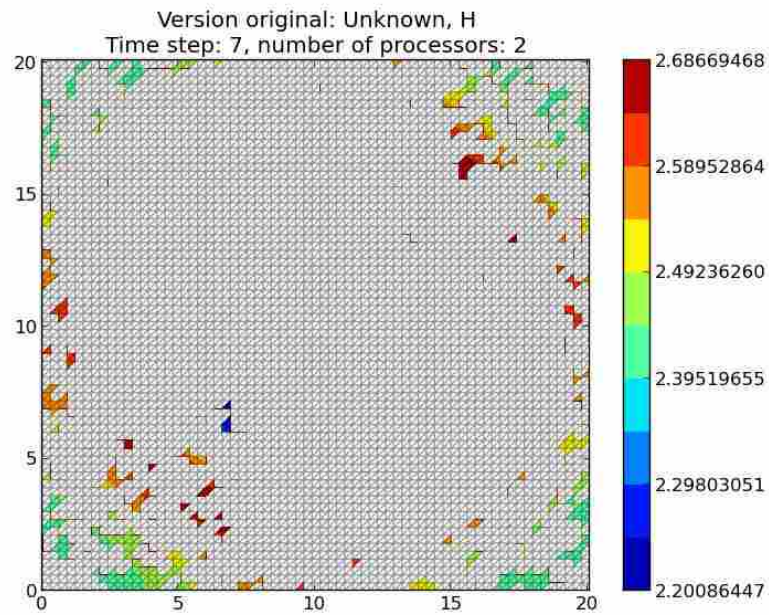
A white plot displays a non-reproducible value

Numerical reproducibility?

time step = 7



Sequential

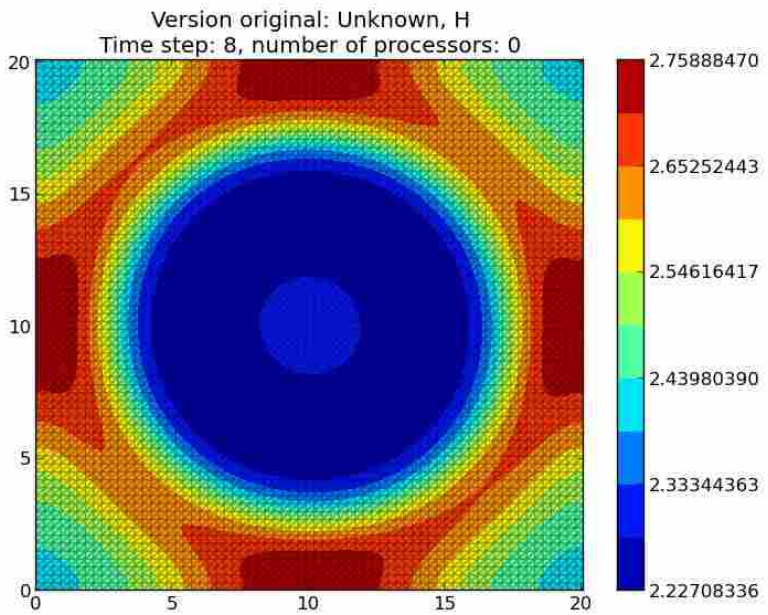


Parallel $p = 2$

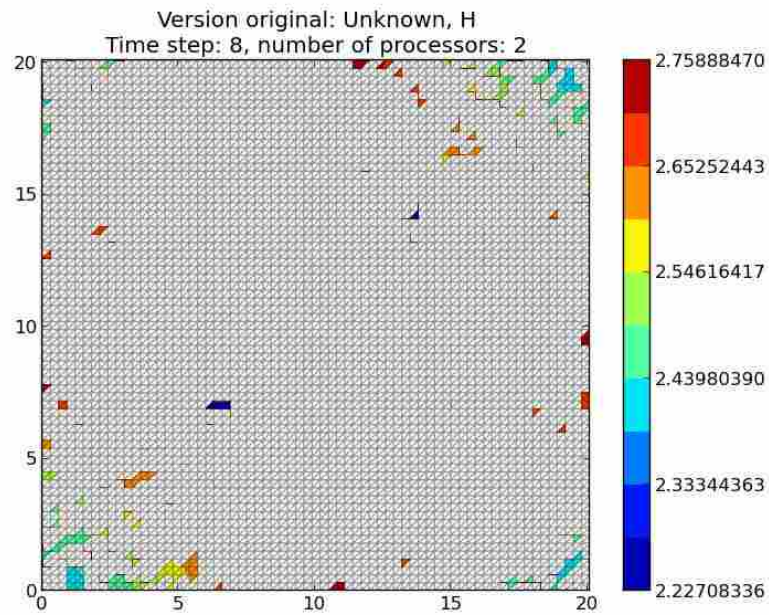
A white plot displays a non-reproducible value

Numerical reproducibility?

time step = 8



Sequential

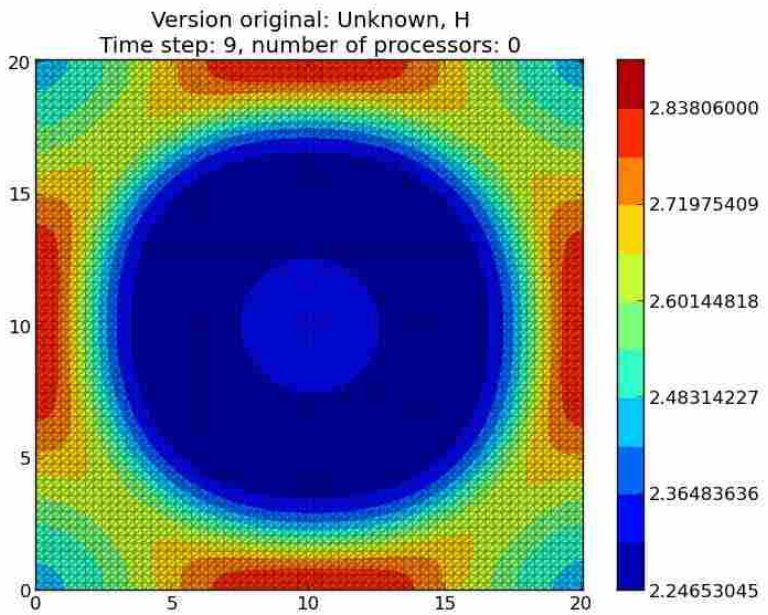


Parallel $p = 2$

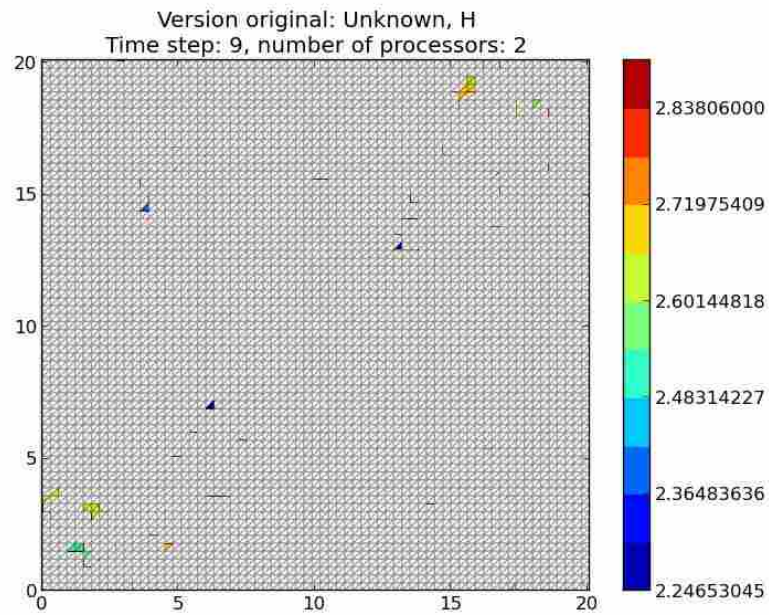
A white plot displays a non-reproducible value

Numerical reproducibility?

time step = 9



Sequential

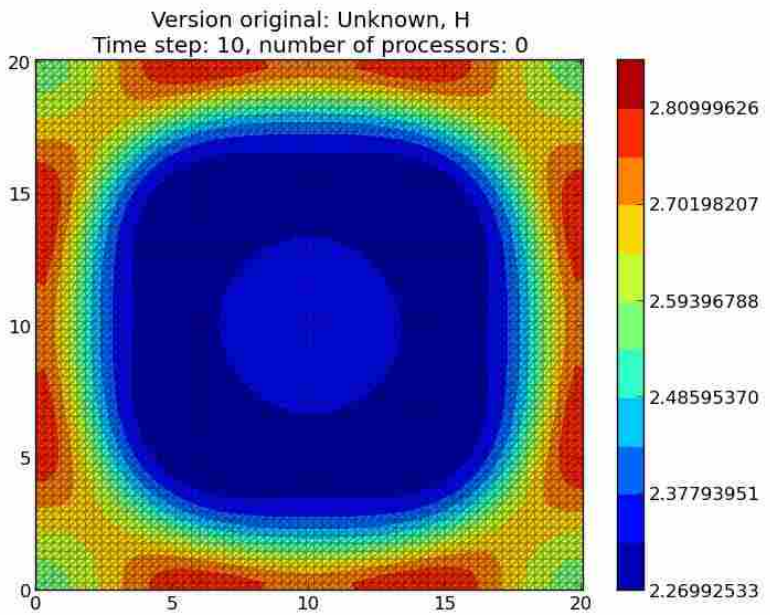


Parallel $p = 2$

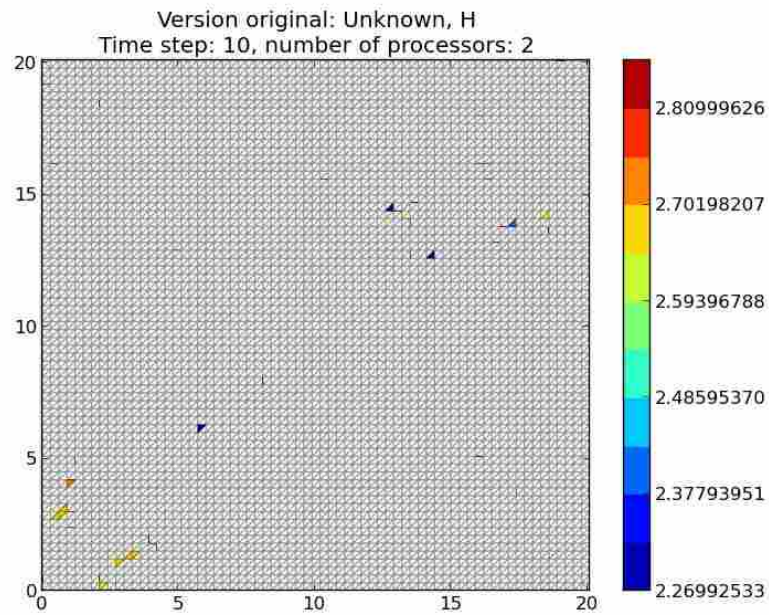
A white plot displays a non-reproducible value

Numerical reproducibility?

time step $= 10$



Sequential

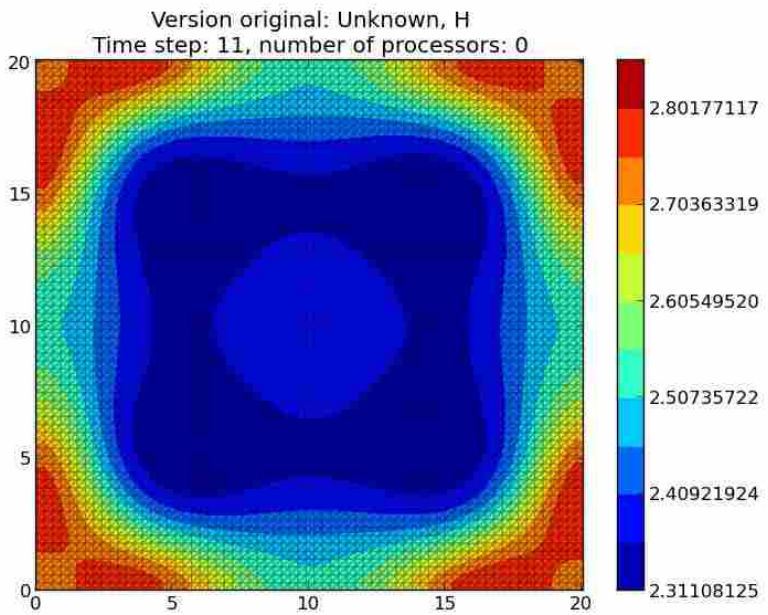


Parallel $p = 2$

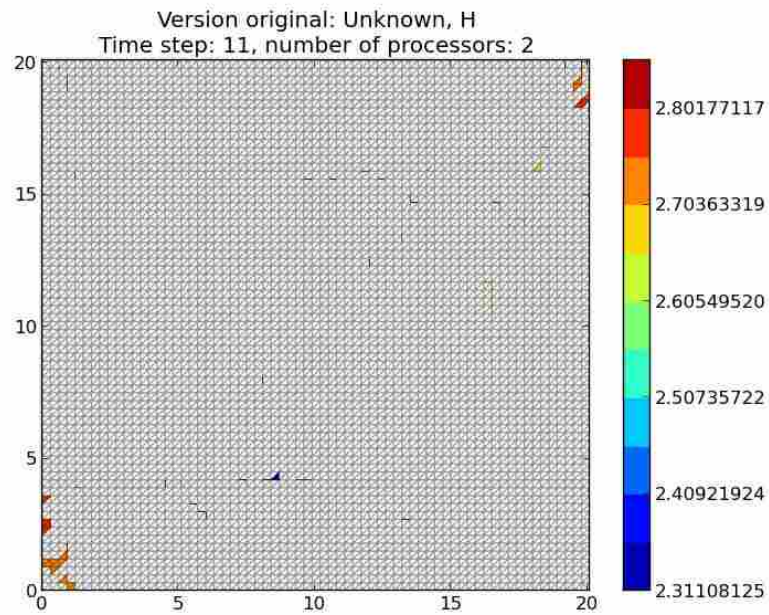
A white plot displays a non-reproducible value

Numerical reproducibility?

time step = 11



Sequential

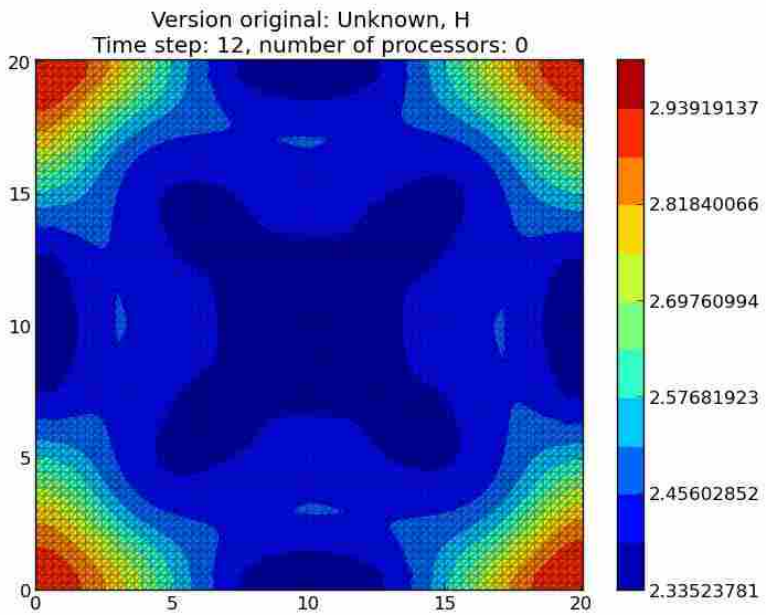


Parallel $p = 2$

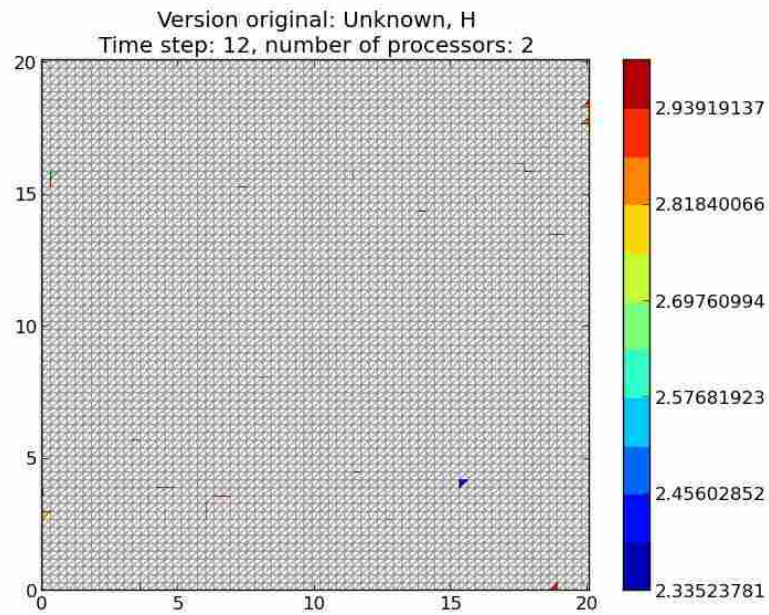
A white plot displays a non-reproducible value

Numerical reproducibility?

time step $= 12$



Sequential

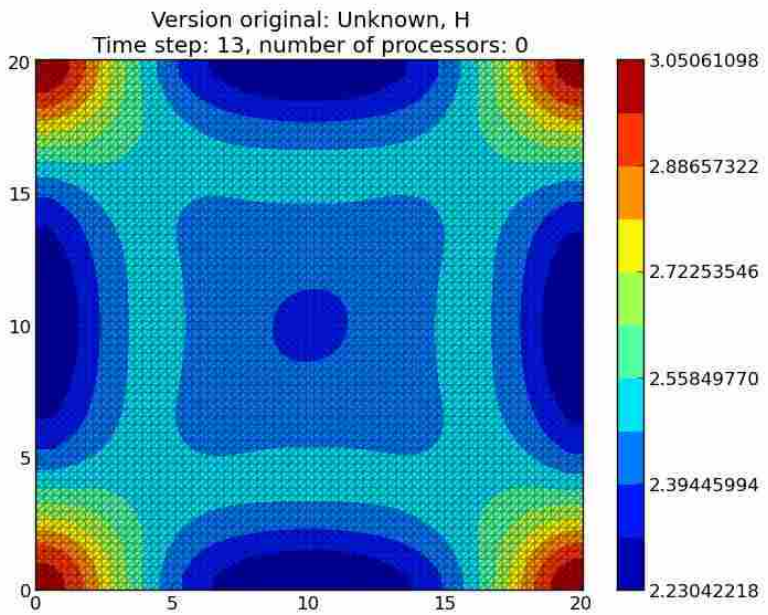


Parallel $p = 2$

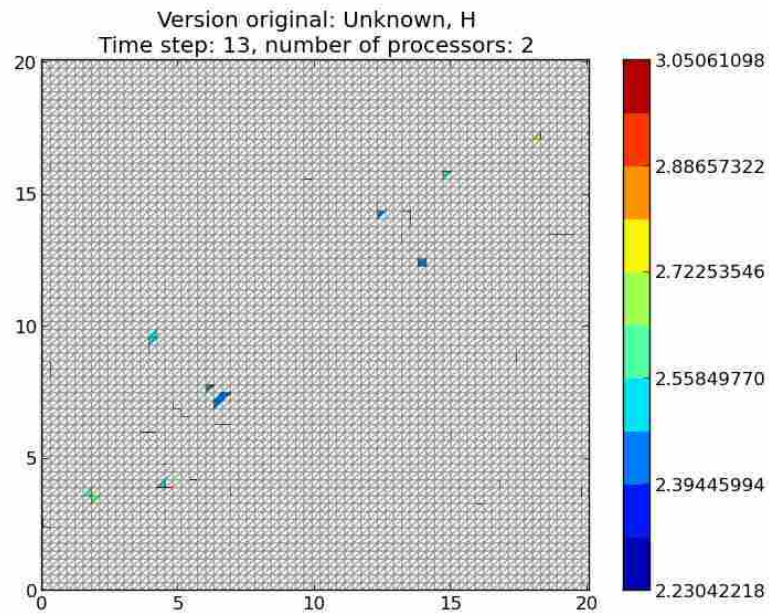
A white plot displays a non-reproducible value

Numerical reproducibility?

time step = 13



Sequential

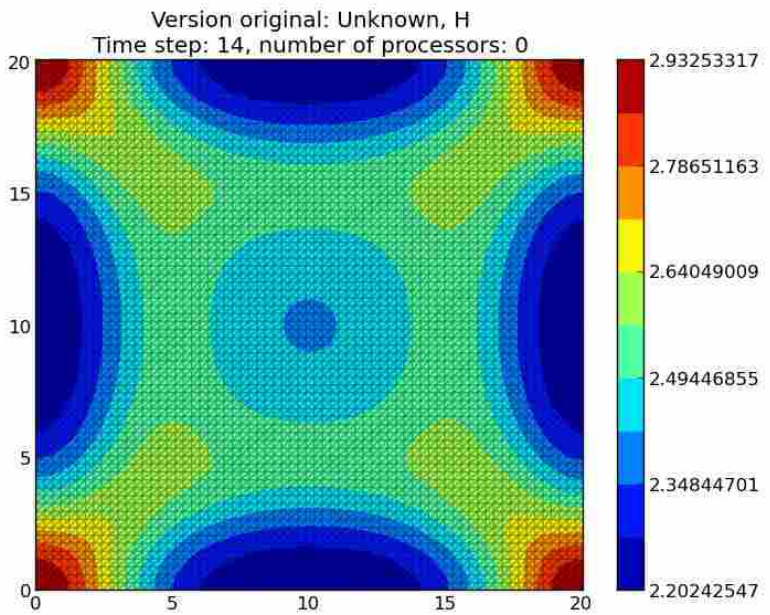


Parallel $p = 2$

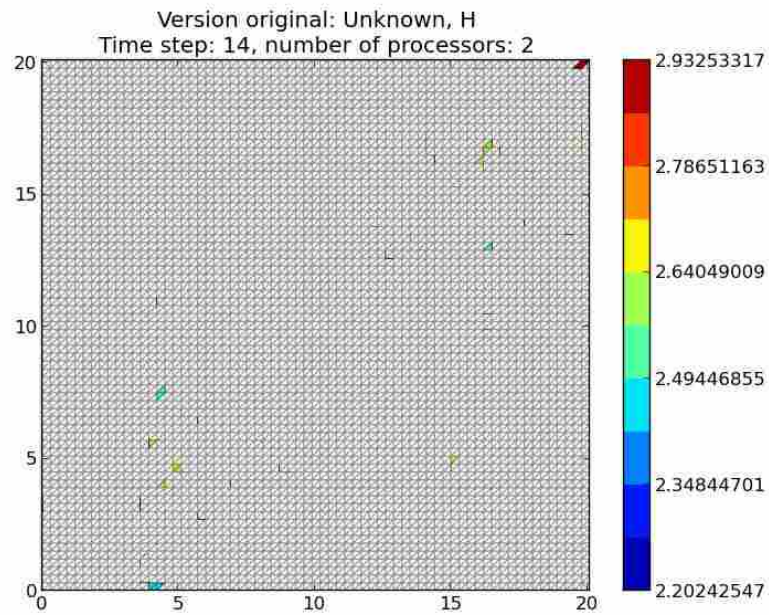
A white plot displays a non-reproducible value

Numerical reproducibility?

time step $= 14$



Sequential

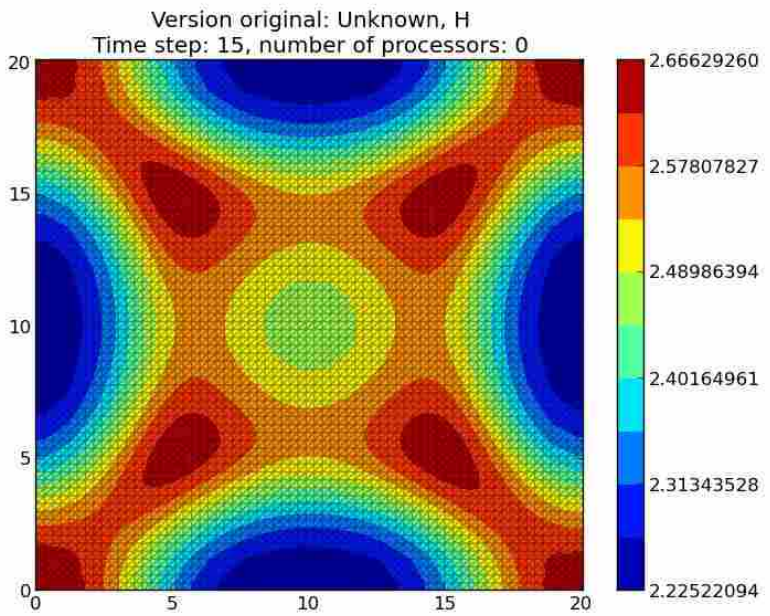


Parallel $p = 2$

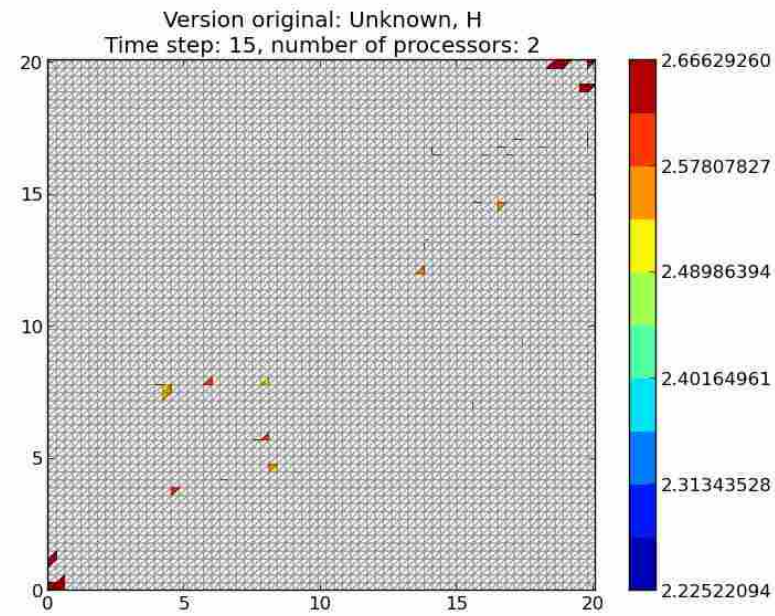
A white plot displays a non-reproducible value

NO numerical reproducibility!

time step = 15



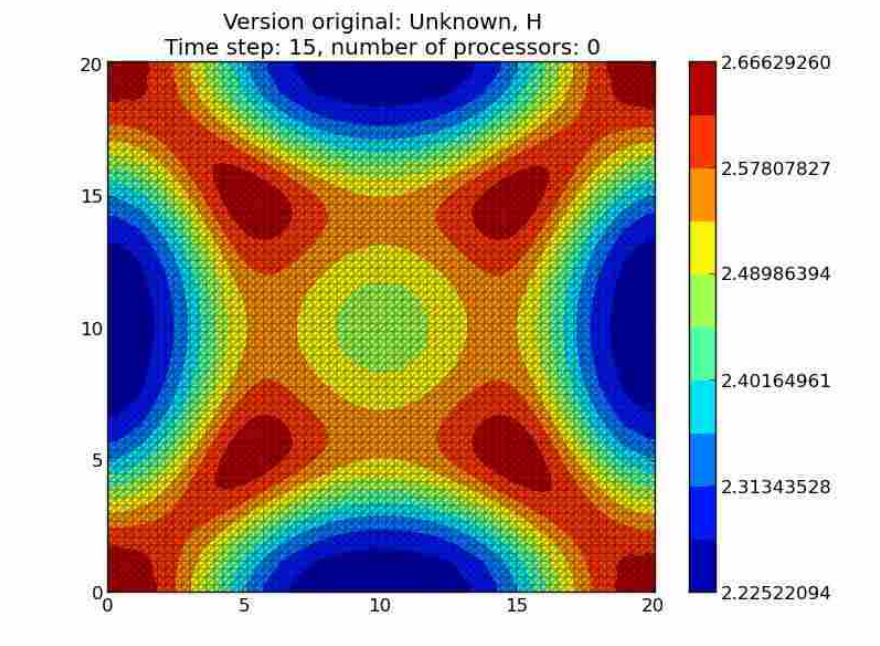
Sequential



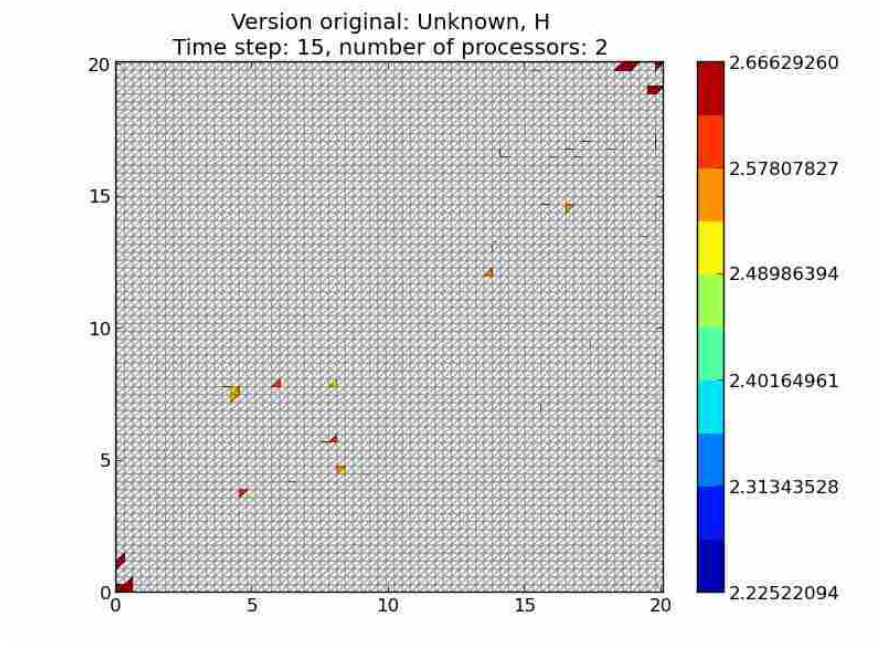
Parallel $p = 2$

Telemac2D: gouttedo

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. Computing sums as accurately as possible
   - iFastSum: distillation again
   - Accsum
   - Hybridsum and OnLineExactSum
4. More EFT and accurate algorithms
   - Some more results
5. 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. Computing sums as accurately as possible
   - iFastSum: distillation again
   - Accsum
   - Hybridsum and OnLineExactSum
4. More EFT and accurate algorithms
   - Some more results
5. More accuracy: conclusion

How to manage accuracy and speed?

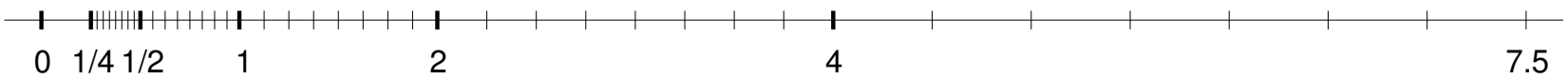
So many ways . . . and a new "best one" every year since 1999

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

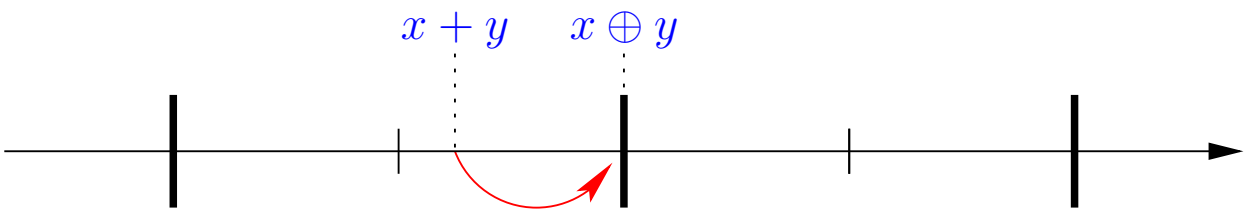
IEEE-754 floating point arithmetic

Floating-point numbers (normal, non zero)

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



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

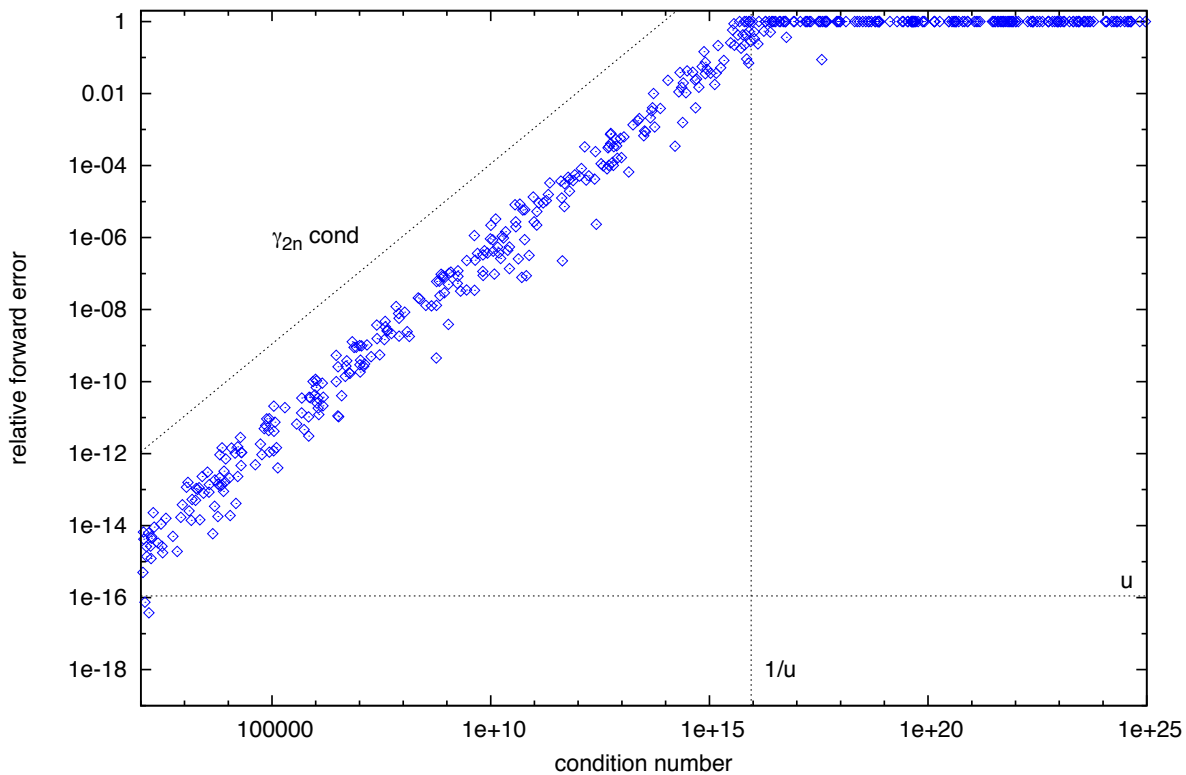
Using the standard model

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. Computing sums as accurately as possible
   - iFastSum: distillation again
   - Accsum
   - Hybridsum and OnLineExactSum
4. More EFT and accurate algorithms
   - Some more results
5. More accuracy: conclusion

Floating-point summation: classic results [HFPA,09],[ASNA,06]

Accuracy for backward stable algorithms

- Accuracy of the computed sum $\leq (n-1) \times \mathit{cond} \times \mathbf{u}$
- $\mathit{cond}(\sum x_i) = \sum |x_i| / |\sum x_i|$
- 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 $\leq (n-1)\sum |x_i|\mathbf{u}$

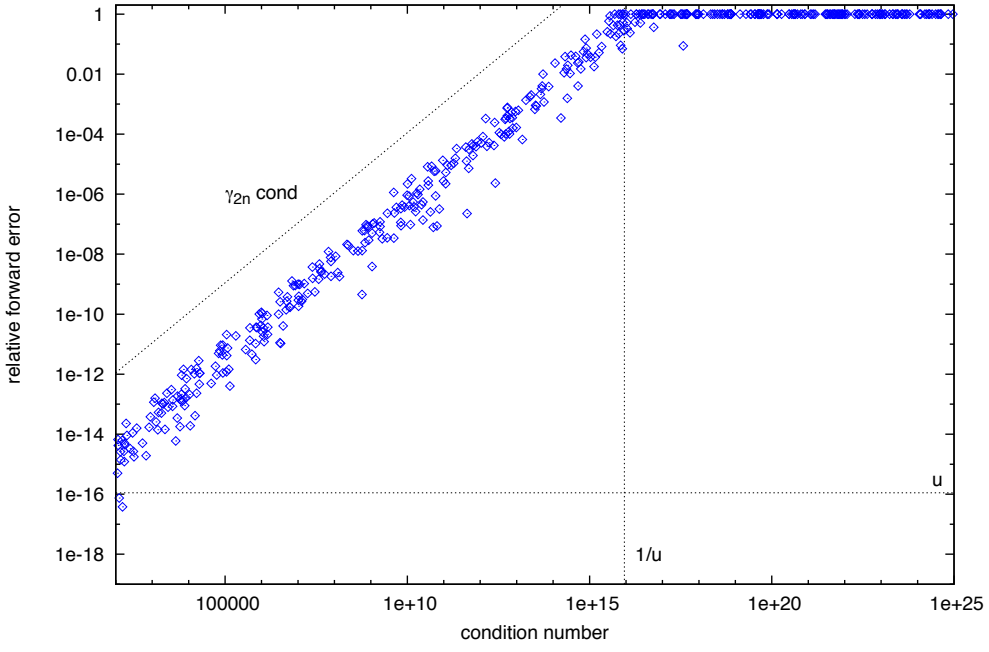
Proposition (Another bound)

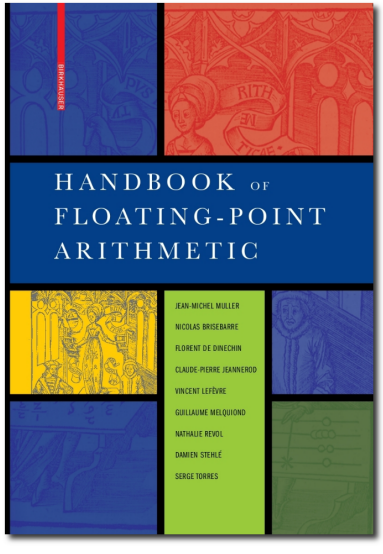
$|\widehat{s_n} - s_n| \leq \mathbf{u} \sum_{i=1}^{n-1} |T_i|$, where T_i is the i-th partial sum in $\mathcal{A}$.

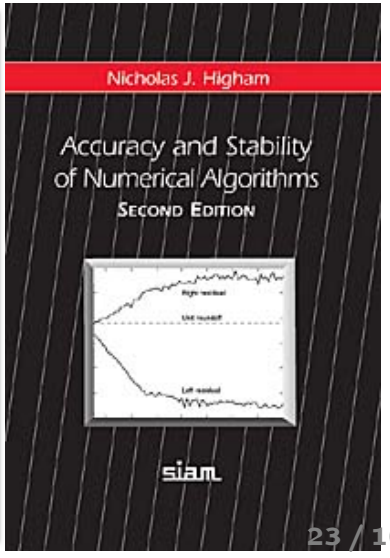
- Minimize this bound or at least every $|T_i|$.
- All the x_i have the same sign: recursive with IOS (good), insertion with IOS (the best).
- Cancellation when $\sum |x_i| \gg |\sum x_i|$: DOS better than IOS.

Conclusion of the basic steps

It exists no universally better accurate summation algorithm





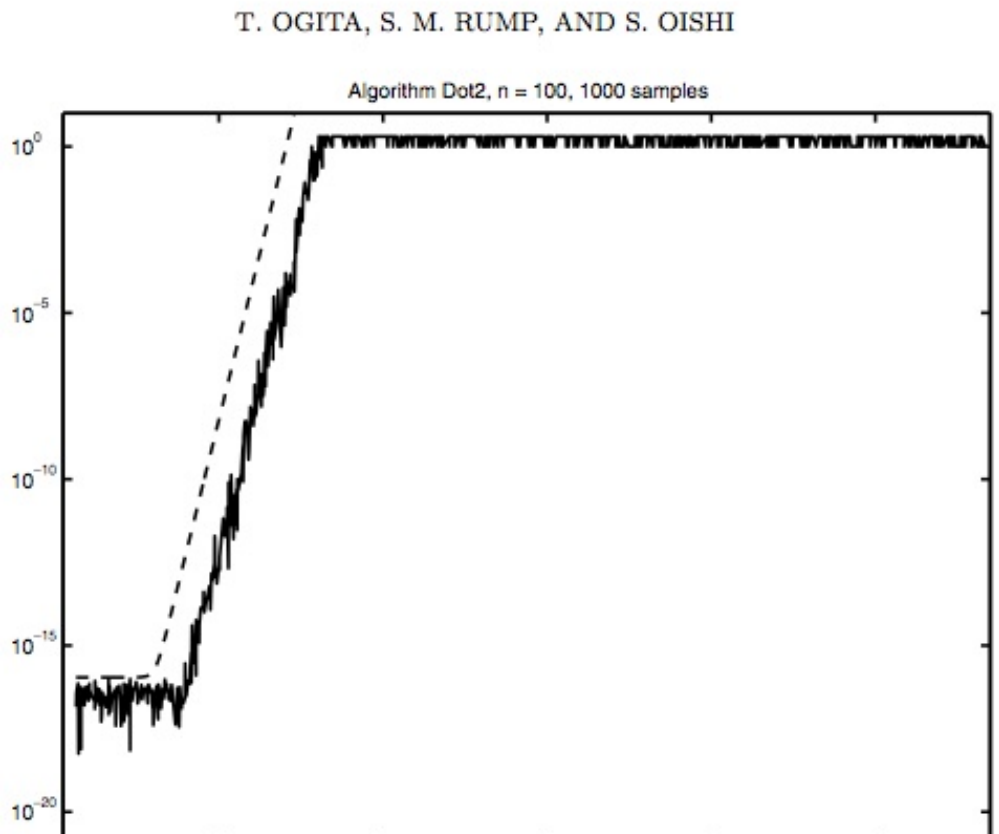


Computing sums more accurately: step 1

More accuracy but still conditioning: overview

Arbitrary precision but *accuracy* still condition dependant

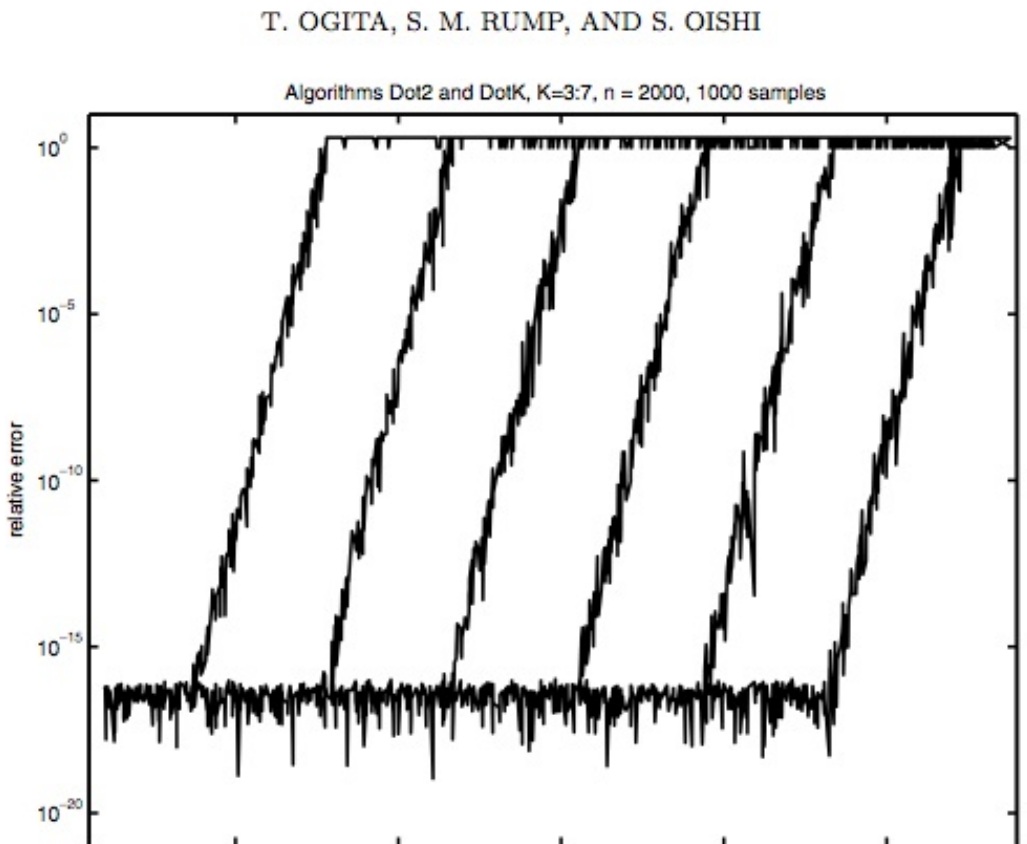
- Simulating more *precision*: double-double, quad-double, . . . , MPFR
- Compensated algorithms: Kahan(65), . . . , Sum2(05), SumK(05)
- Accuracy $\lesssim \mathbf{u} + \mathit{cond} \times \mathbf{u}^K$
- No more n dependency while $n\mathbf{u} \leq 1$.



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 $n\mathbf{u} \leq 1$.



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

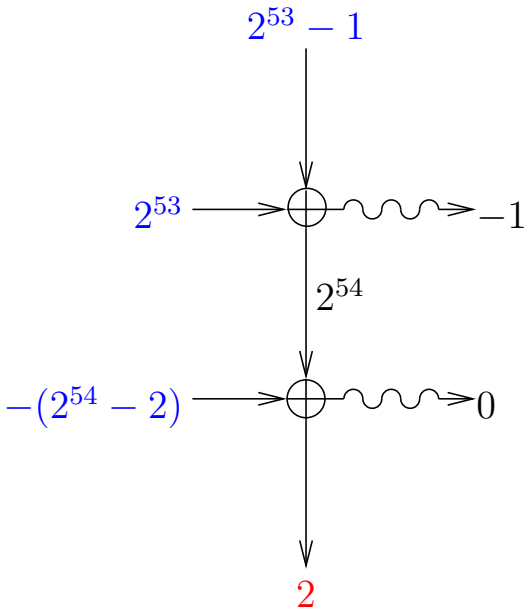
- Accuracy improvement in practice: $-c$ approximates $s \oplus y$'s rounding error.
- Compensated sum accuracy $\leq 2\mathbf{u} + n \sum |x_i| \mathcal{O}(\mathbf{u}^2)$.

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



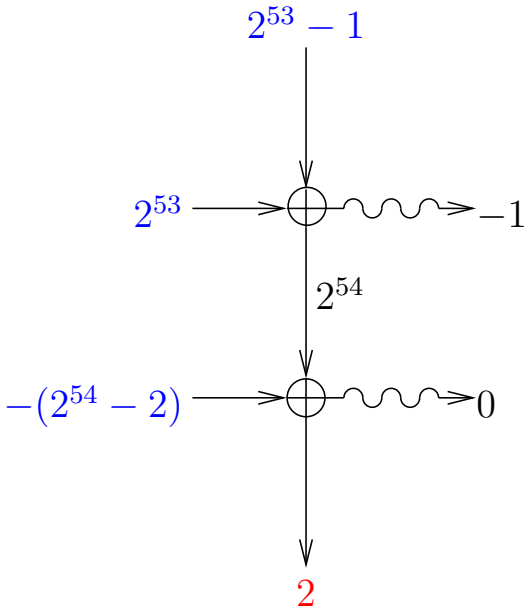
Relative error = 1

One example

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

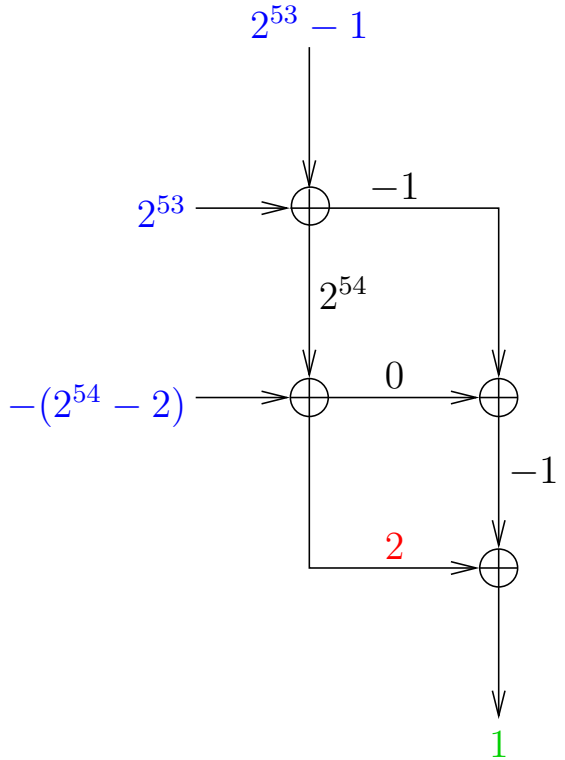
Exact sum: $x_1 + x_2 + x_3 = 1$.

Classic summation



Relative error = 1

Compensation of the rounding errors



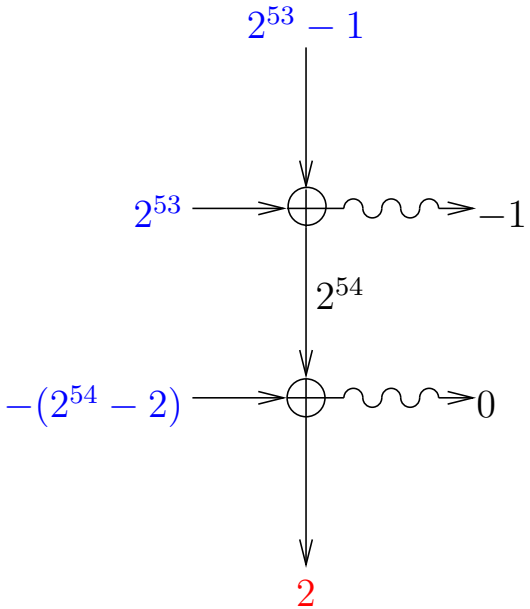
The exact result is computed

One example

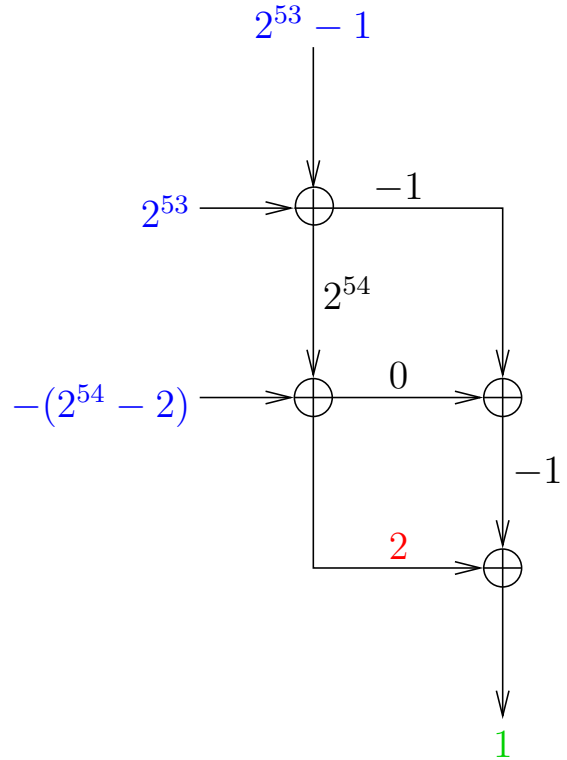
IEEE double precision numbers: $x_1 = 2^{53} - 1$, $x_2 = 2^{53}$ and $x_3 = -(2^{54} - 2)$.

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

Classic summation



Compensation of the rounding errors



Relative error = 1

The exact result is computed

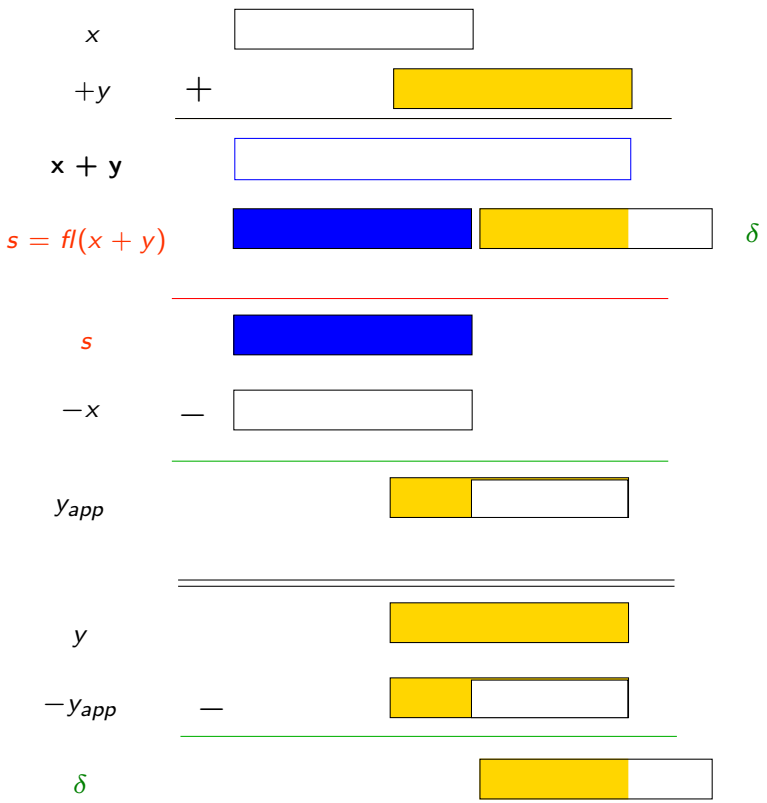
The rounding errors are computed thanks to *error-free transformations*.

Compensation

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 $n \leq n'_{max} = 1/3\mathbf{u}$.

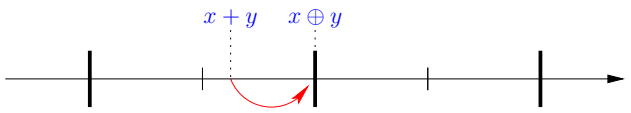
Priest's doubly compensated summation (92)

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

Computing sums more accurately: step 2

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

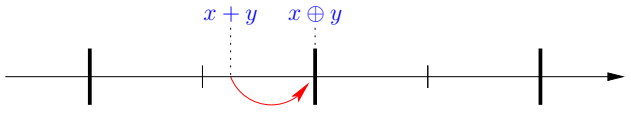
- Long accumulator, hardware oriented: Malcolm (71), Kulish (80)
- Cutting the summands: AccSum (SISC-08), FastAccSum (SISC-09)
- Summation at a given exponent: HybridSum (SISC-09), OnLineExact (TOMS-10)

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)

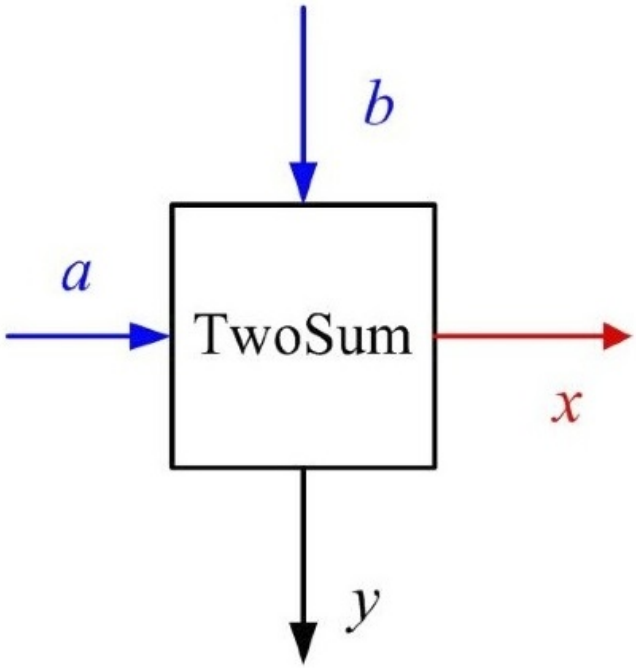
From faithful rounding to correct rounding

$\longrightarrow$ **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)

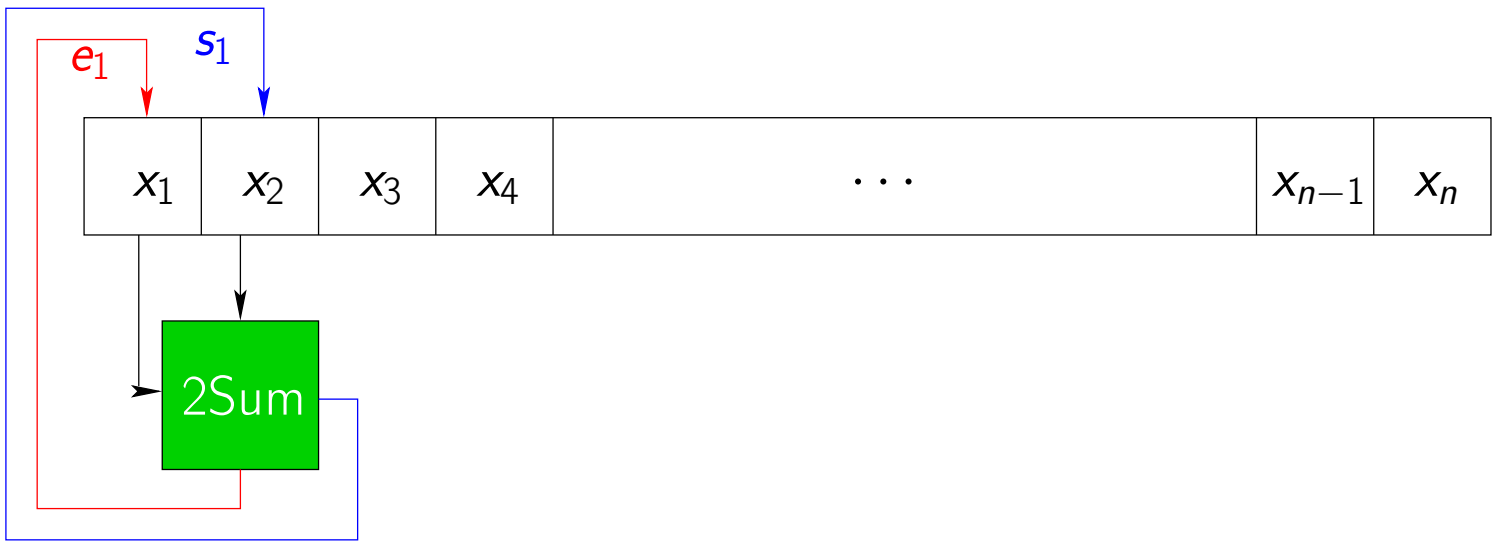
$x = a \oplus b$

$z = x \ominus a$

$y = b \ominus z$

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

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

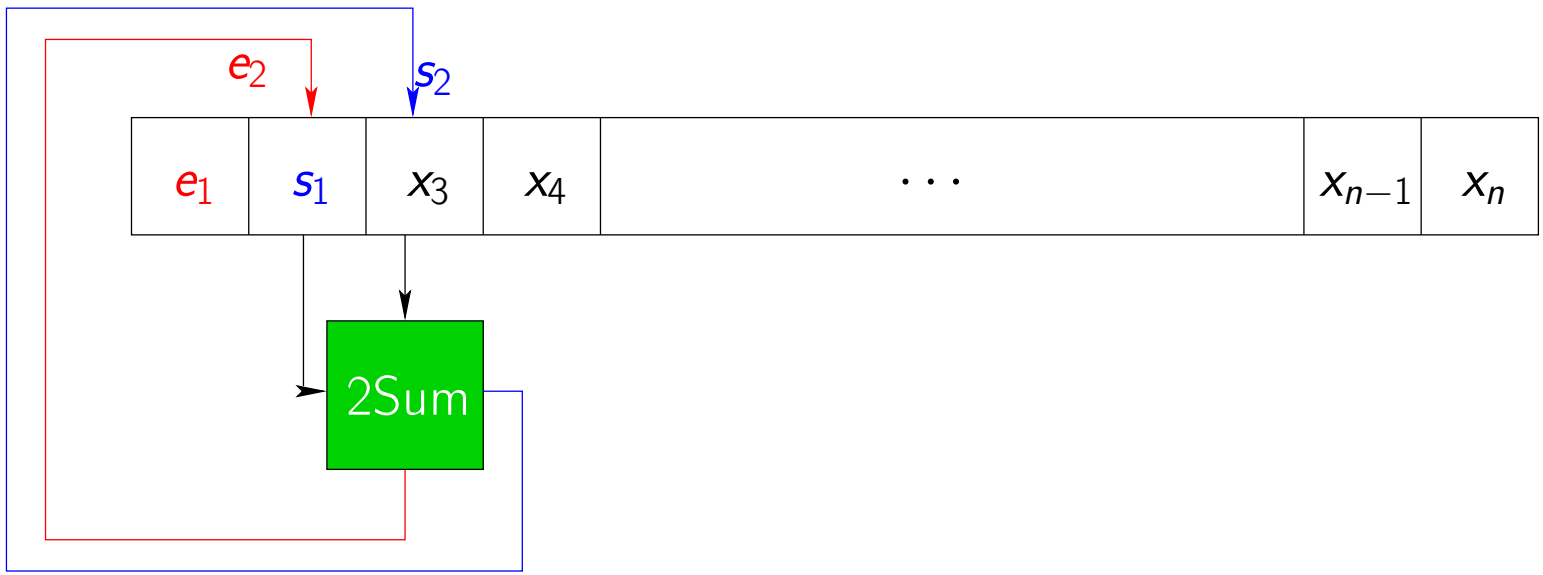


Distillate $\sum x_i, x_i \in \mathbb{F}$

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

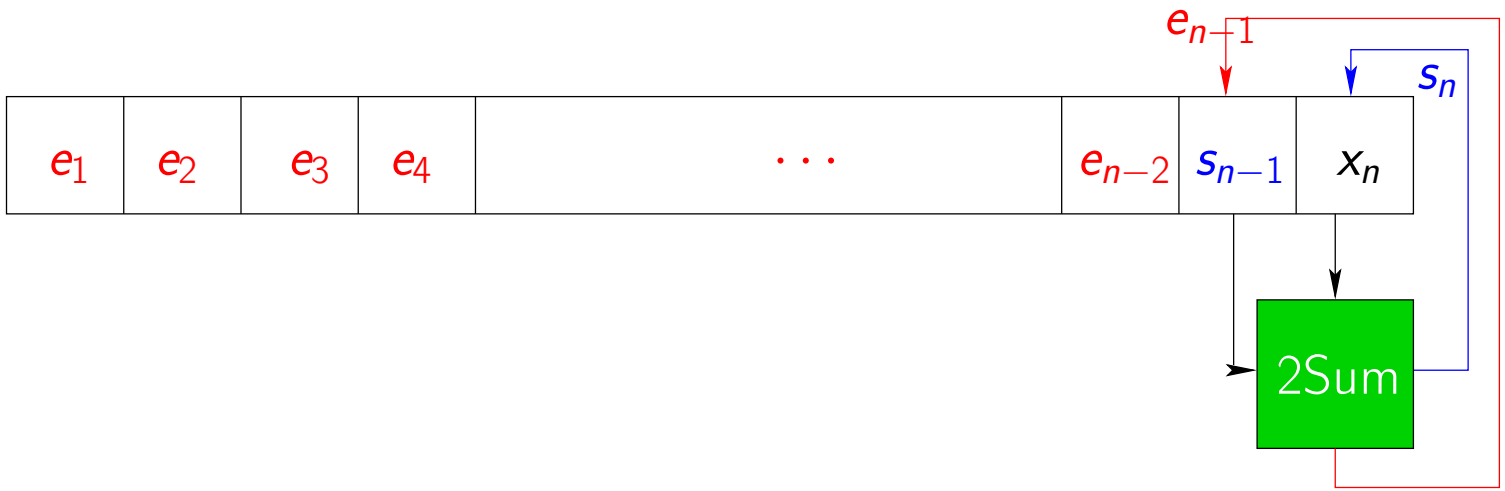
e_1	e_2	s_2	x_4	$\cdots$	x_{n-1}	x_n

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

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

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

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



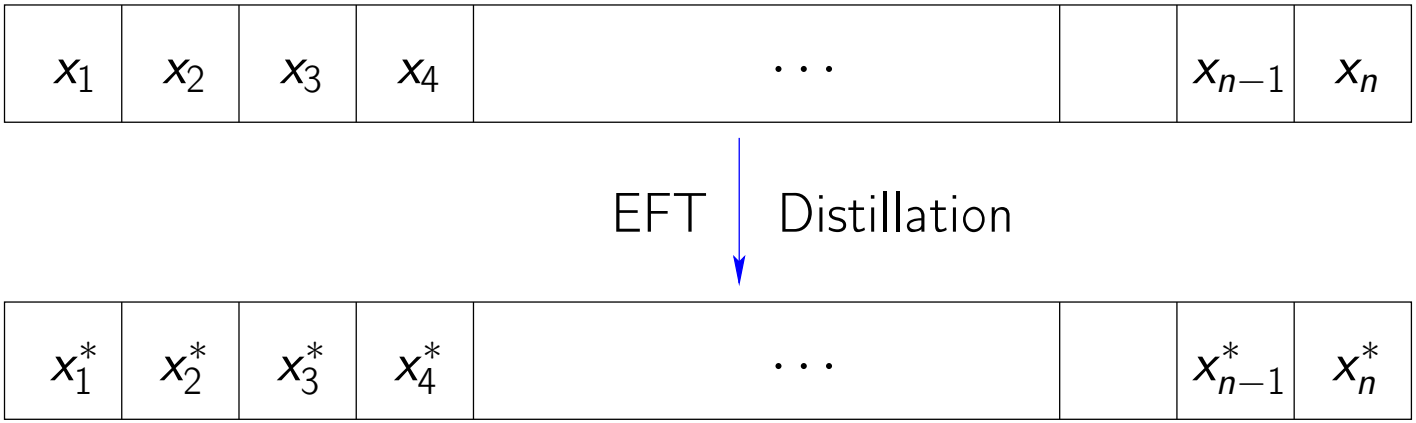
Distillate $\sum x_i, x_i \in \mathbb{F}$

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!



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.

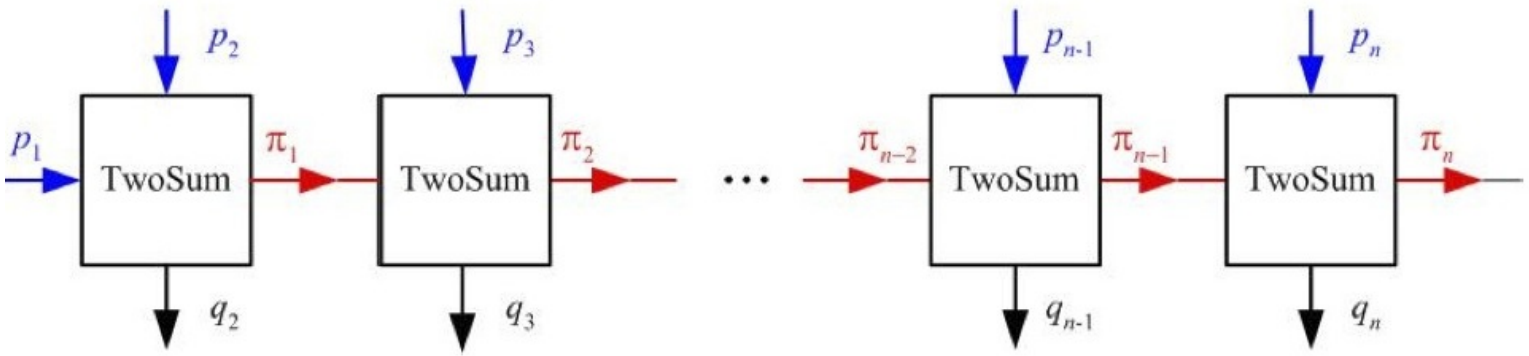
We need a good stopping criteria!

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)

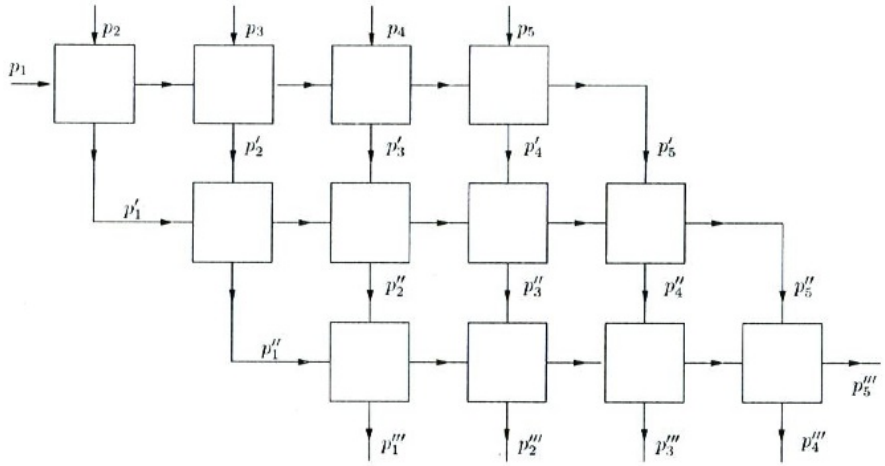


FIG. 4.2. *Outline of Algorithm 4.8 for $n = 5$ and $K = 4$.*

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

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

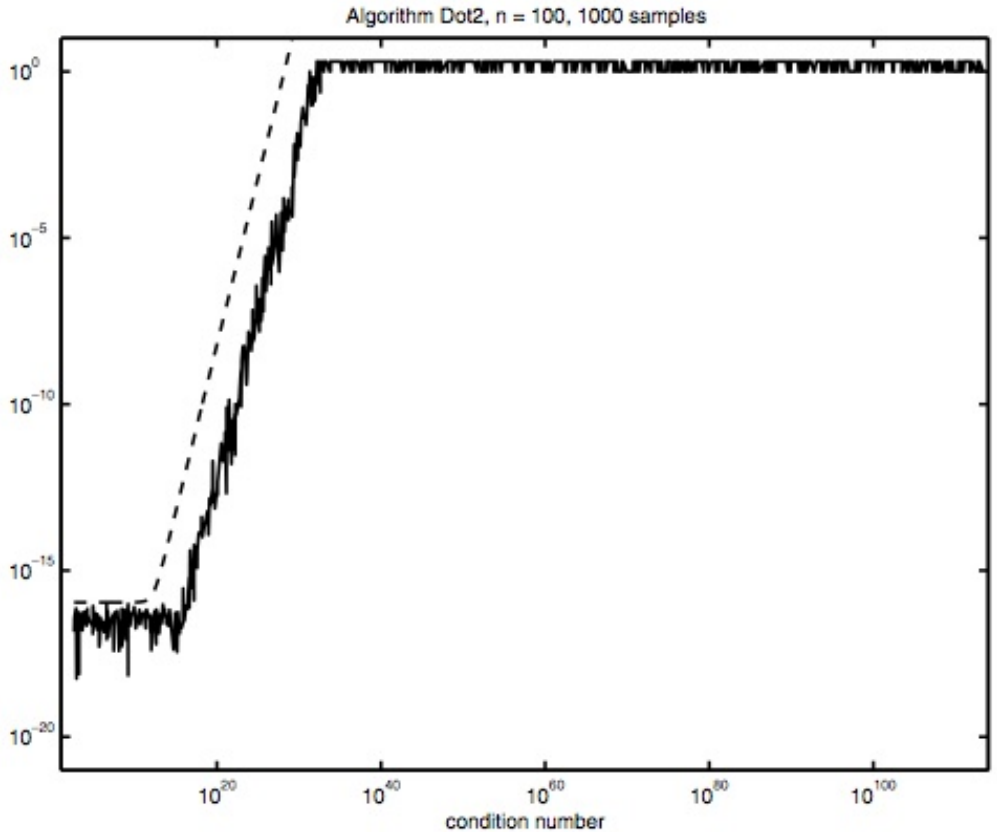
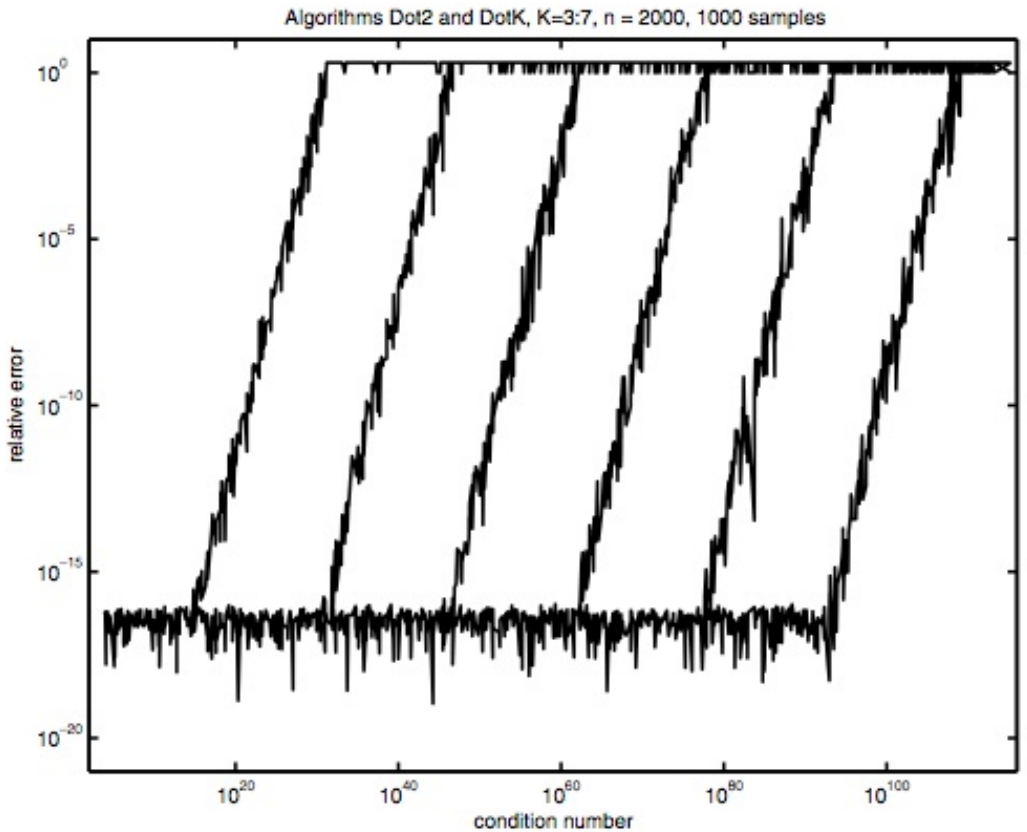


FIG. 6.1. *Test results for Algorithm* 5.3 (`Dot2`), $n = 100$, 1000 *samples.*

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

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



Step 2: towards faithfully or correctly rounded sums

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. Computing sums as accurately as possible
   - iFastSum: distillation again
   - Accsum
   - Hybridsum and OnLineExactSum
4. More EFT and accurate algorithms
   - Some more results
5. More accuracy: conclusion

No more cond: new solutions

Distillation iFastSum (09)

- Rounding errors quickly vanish in practice.
- So distillation soon iterates on shorter vectors.
- Recursive calls only for difficult cases, *e.g.* $1 + 2^{-53} + 2^{-106}$.

How to compute exact partial sums?

Exact summation: cut the summands

- Faithfully rounded sum: AccSum (08), FastAccSum (09)
- Correctly rounded sum: NearSum (09)

Exact summation: partial sum by exponant + distillation

- Faithfully rounded sum: HybridSum (09), OnLineExact (10)

Computing sums as accurately as possible

iFastSum (09)

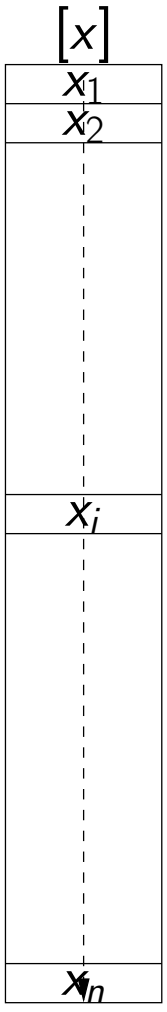
iFastSum = distallation SumK ($K \geq 2$) with a dynamic control of the residual error until the faithfully or correctly rounded value.

Memory overhead = $\mathcal{O}(1)$

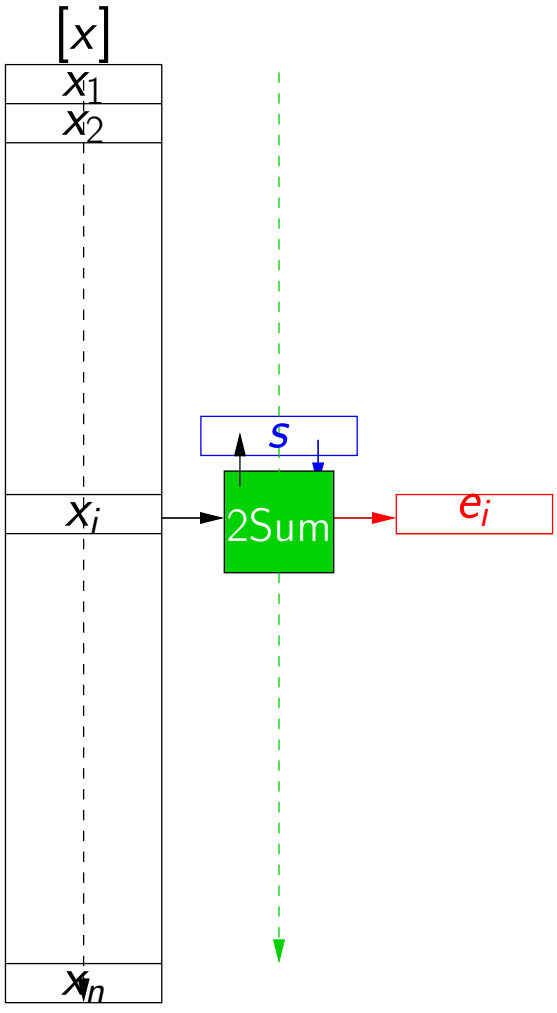
No more cond limitation: always faithfully or correctly rounded sum.

- Rounding errors quickly vanish in practice.
- So distillation soon iterates on shorter vectors.
- Recursive calls only for difficult cases, *e.g.* $1 + 2^{-53} + 2^{-106}$.

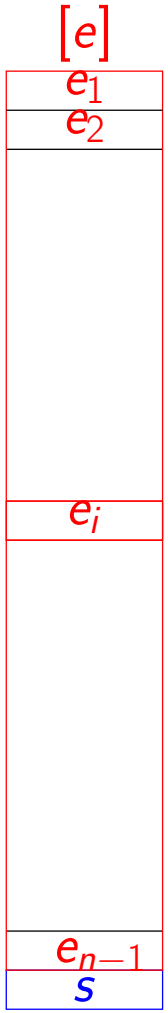
iFastSum (Zhu-Hayes,09)



iFastSum (Zhu-Hayes,09)

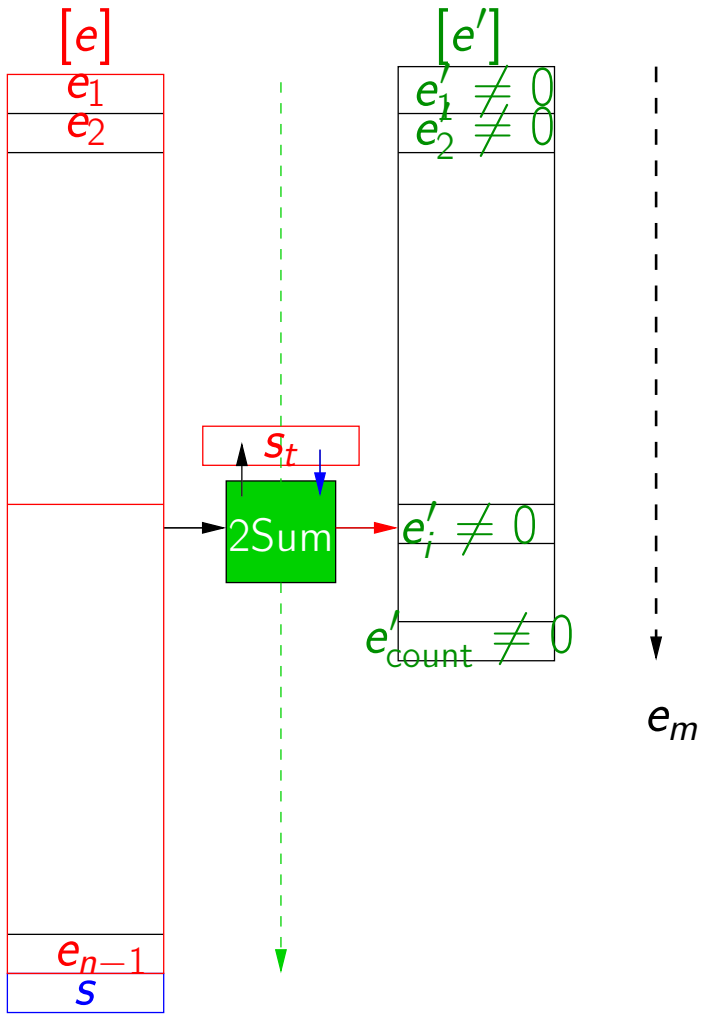


iFastSum (Zhu-Hayes,09)



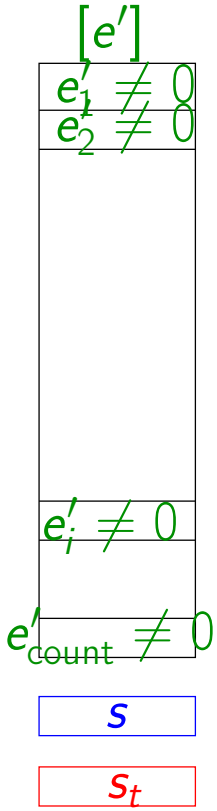
$$\sum_{1}^{n} x_i = s + \sum_{1}^{n-1} e_i$$

iFastSum (Zhu-Hayes,09)



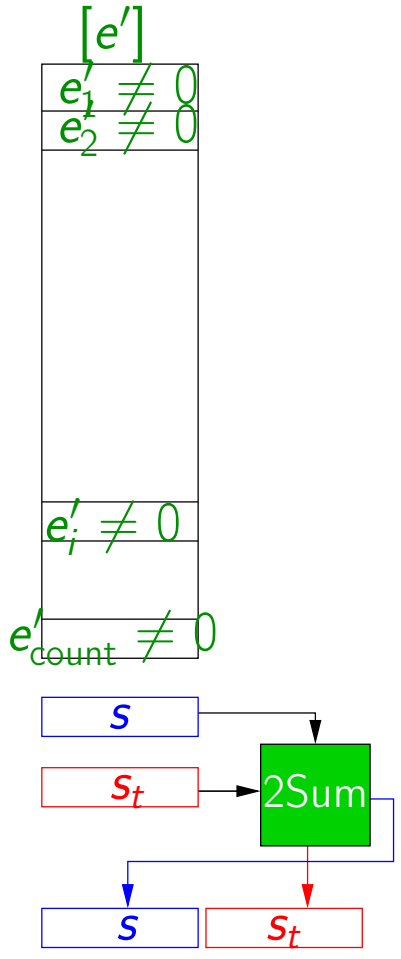
Bounding the residual error $e_m = (\text{count}) \times 1/2 ulp(\max |s_t|)$

iFastSum (Zhu-Hayes,09)



$$\sum_{1}^{n} x_i = s + s_t + \sum_{1}^{\text{count}} e'_i$$

iFastSum (Zhu-Hayes,09)

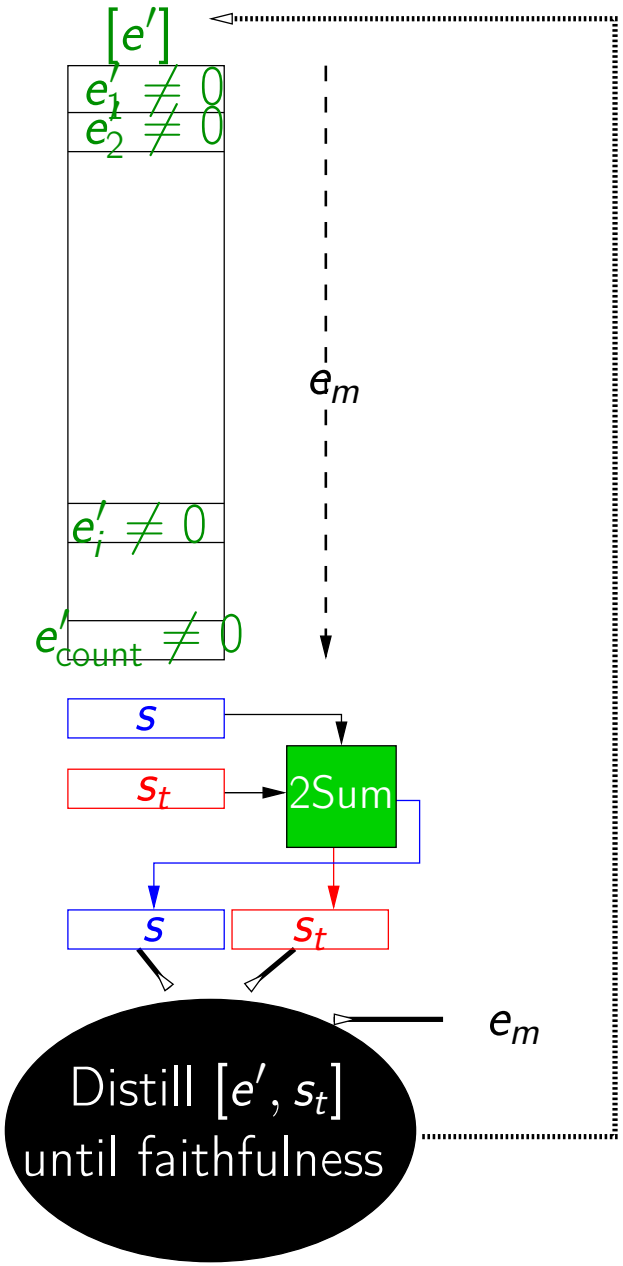


$$\sum_{1}^{n} x_i = s + s_t + \sum_{1}^{\text{count}} e'_i$$

iFastSum (Zhu-Hayes,09)

Stopping criteria in 2 steps

1. Stop the distillation when
   $e_m = 0$ or $e_m < \mathrm{HalfUlp}(s)$
2. Correct rounding: effect of e_m to s and s_t
   - ▶ 2 "recursive" calls of the iFastSum's main part
   if $s_t \pm e_m$ modifies s or
   $Round3(s, (s_t \pm e_m)_h, (s_t \pm e_m)_l) \neq s$.



How to stop to distillate!

Principe

Sharp approximation of the exact sum of 3 non overlapping values: 2 floating-points and 1 sign (the value does not matter).

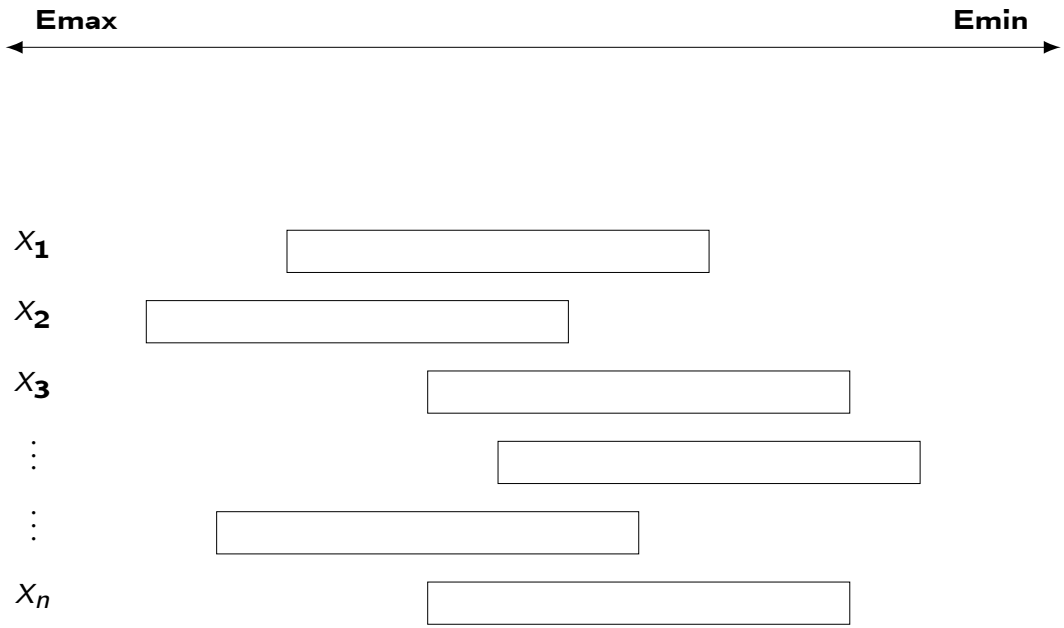
$\mathrm{Round3}(s_0, s_1, \mathrm{sign}(\widehat{s_2})) = \mathrm{fl}(s_0 + s_1 + \widehat{s_2})$

- One test of s_1
- exponent manipulation
- but no change of the rounding mode

Computing sums as accurately as possible

Faithful accuracy: AccSum (08), FastAccSum (09)

AccSum uses EFT ExtractVector (Fast2Sum)

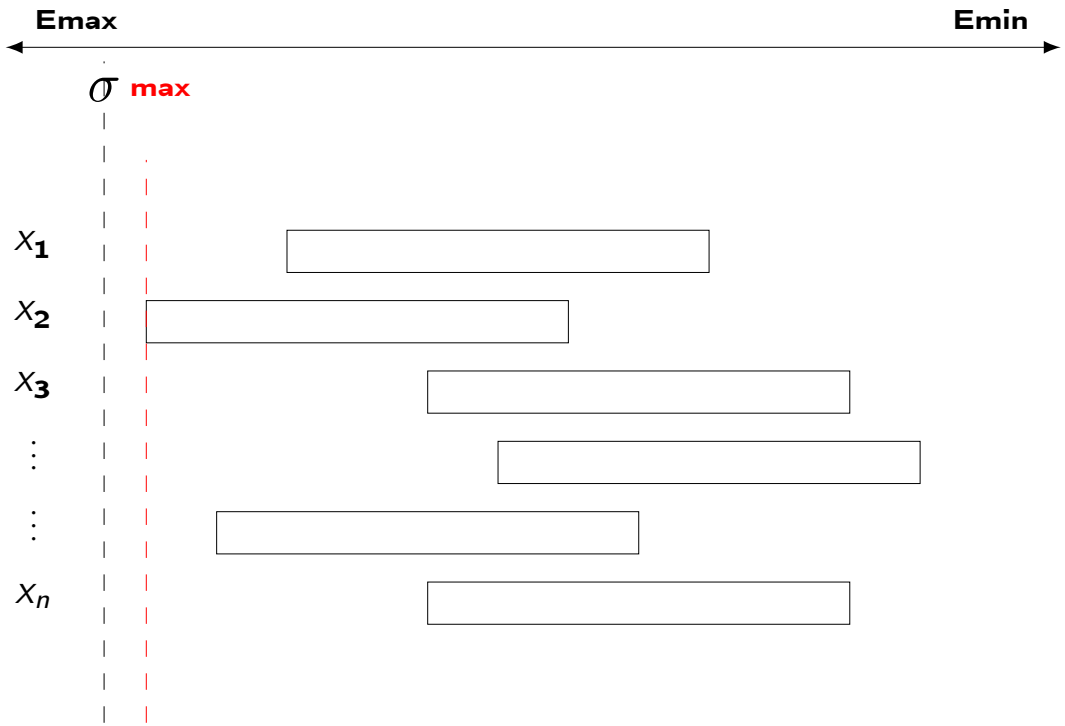


Steps for sequential

- Compute max.

Faithful accuracy: AccSum (08), FastAccSum (09)

AccSum uses EFT ExtractVector (Fast2Sum)

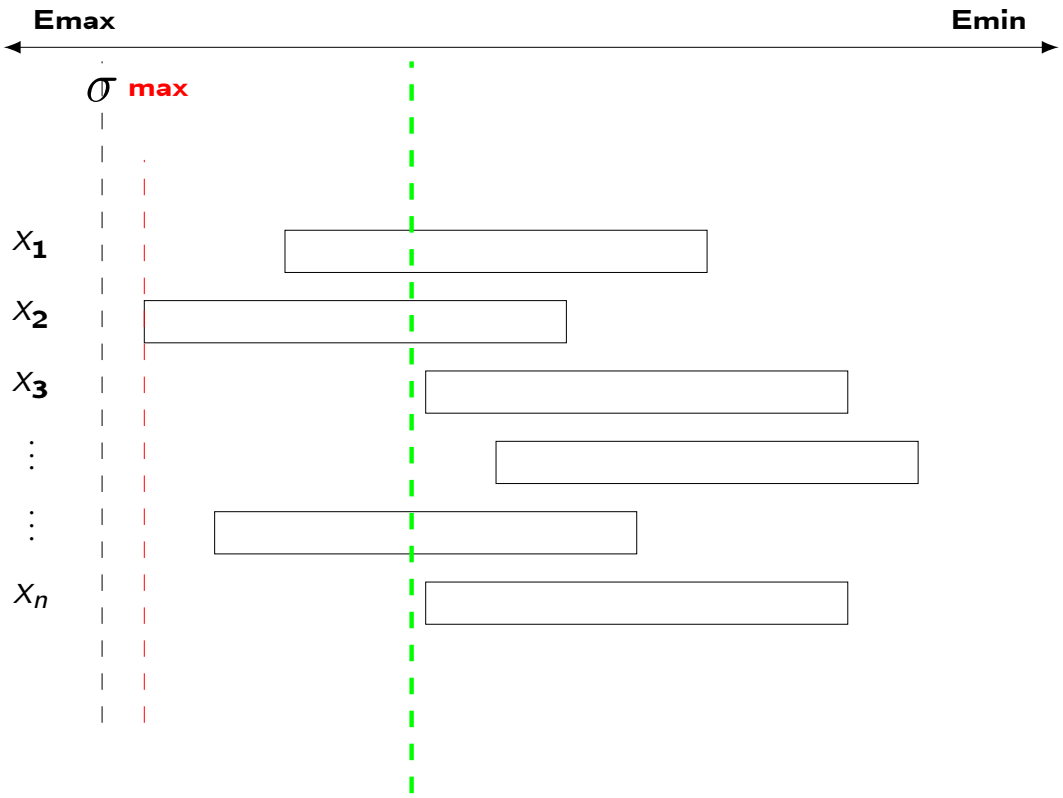


Steps for sequential

- Compute max.

Faithful accuracy: AccSum (08), FastAccSum (09)

AccSum uses EFT ExtractVector (Fast2Sum)

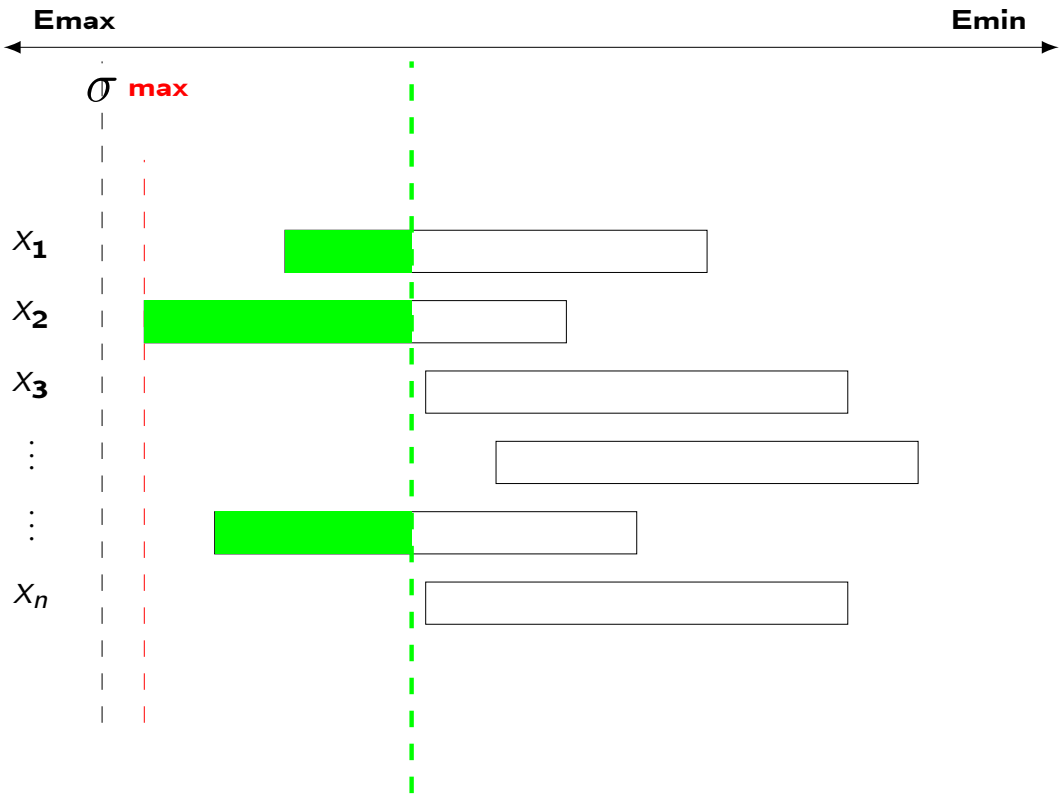


Steps for sequential

- Compute max.

Faithful accuracy: AccSum (08), FastAccSum (09)

AccSum uses EFT ExtractVector (Fast2Sum)

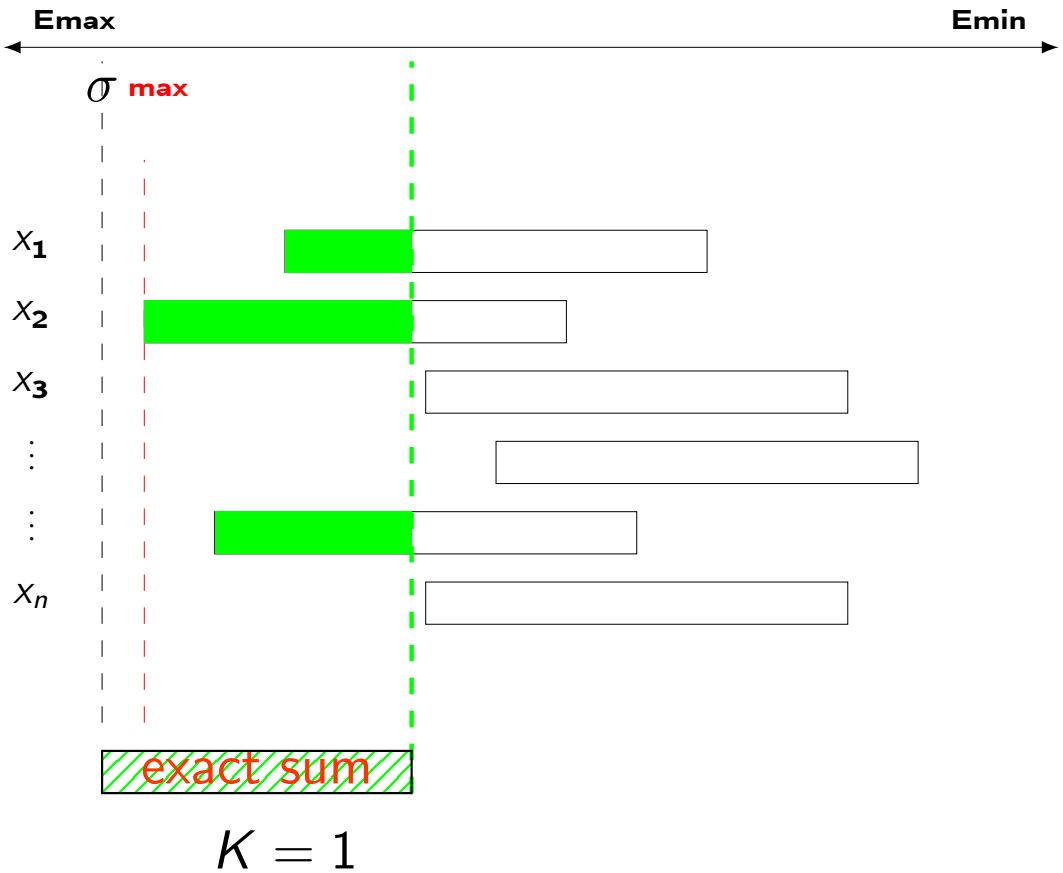


Steps for sequential

- Compute max.
- Sum each shrunk

Faithful accuracy: AccSum (08), FastAccSum (09)

AccSum uses EFT ExtractVector (Fast2Sum)

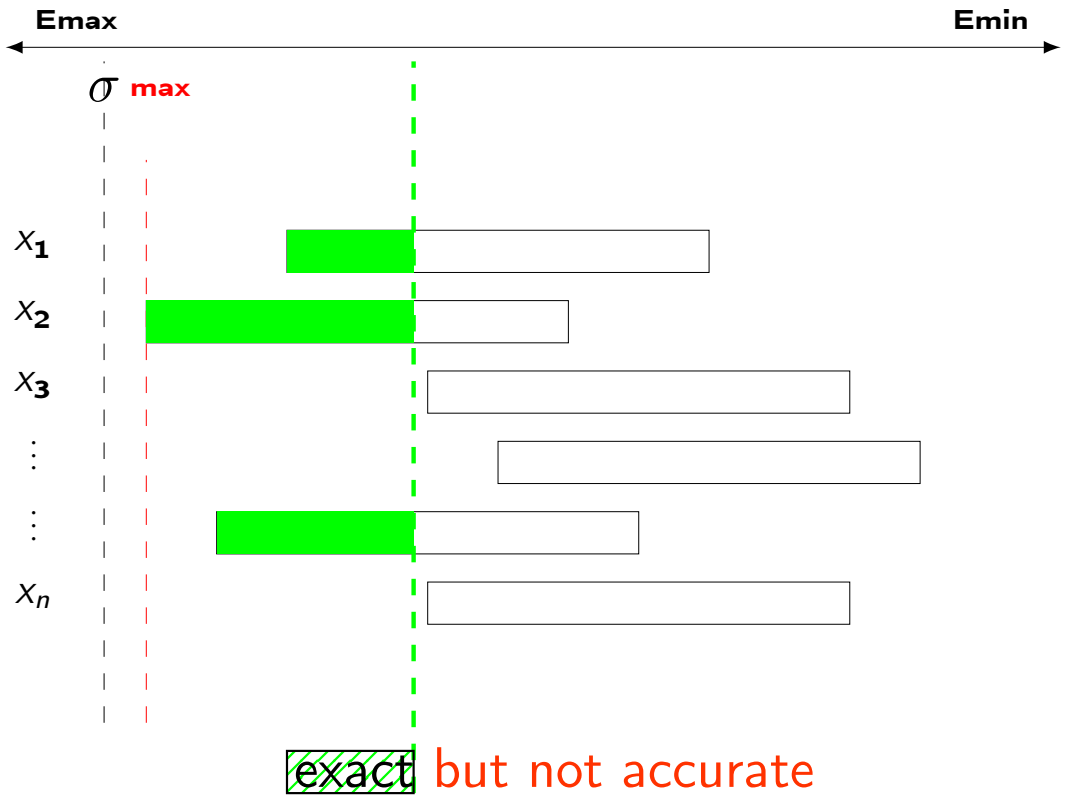


Steps for sequential

- Compute max.
- Sum each shrunk

Faithful accuracy: AccSum (08), FastAccSum (09)

AccSum uses EFT ExtractVector (Fast2Sum)

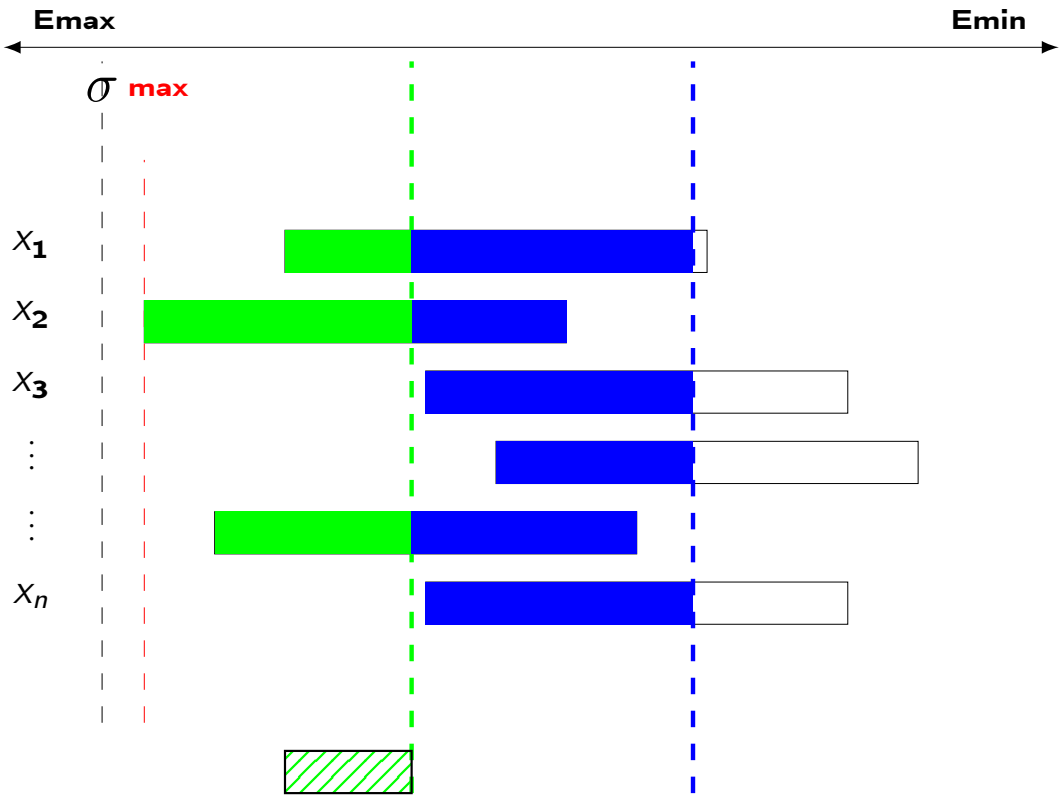


Steps for sequential

- Compute max.
- Sum each shrunk

Faithful accuracy: AccSum (08), FastAccSum (09)

AccSum uses EFT ExtractVector (Fast2Sum)

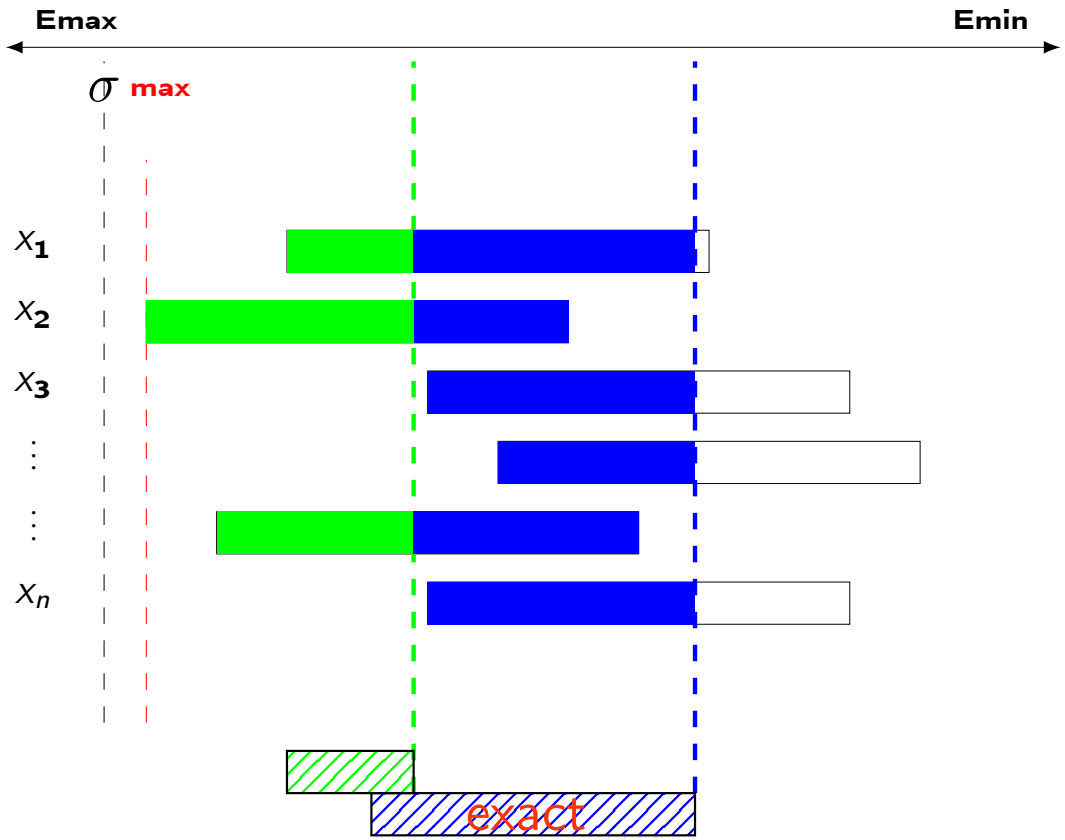


Steps for sequential

- Compute max.
- Sum each shrunk

Faithful accuracy: AccSum (08), FastAccSum (09)

AccSum uses EFT ExtractVector (Fast2Sum)

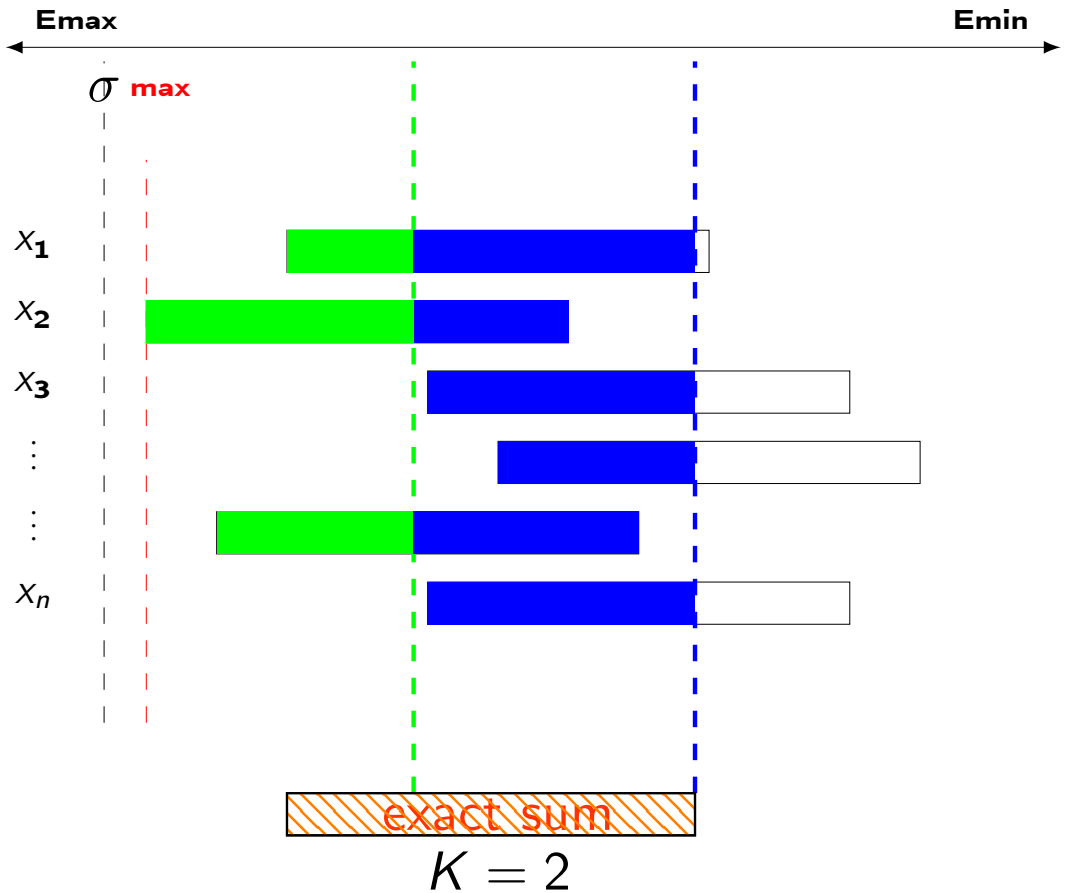


Steps for sequential

- Compute max.
- Sum each shrunk

Faithful accuracy: AccSum (08), FastAccSum (09)

AccSum uses EFT ExtractVector (Fast2Sum)

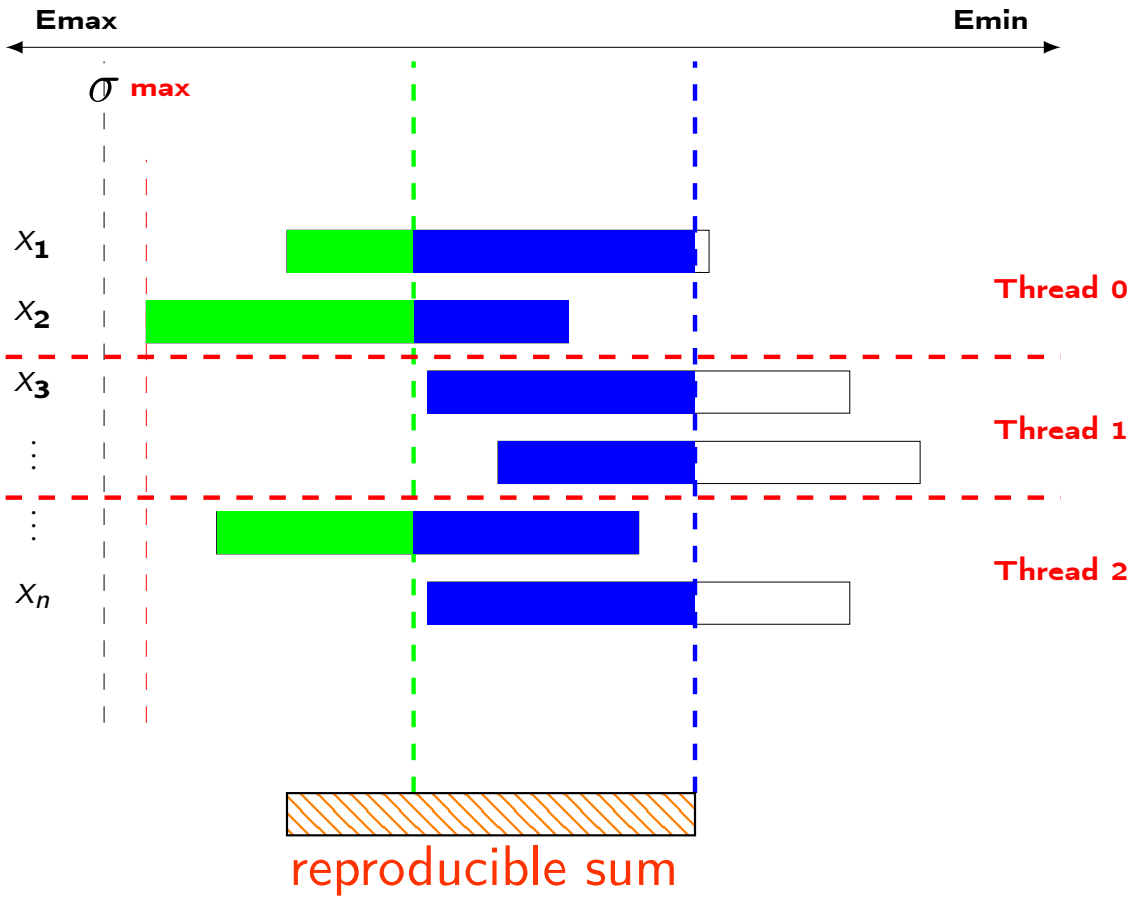


Steps for sequential

- Compute max.
- Sum each shrunk

Faithful accuracy: AccSum (08), FastAccSum (09)

AccSum uses EFT ExtractVector (Fast2Sum)



Steps for sequential

- Compute max.
- Sum each shrunk

Steps for parallel

- Local max (compute)
- Max (reduce)
- Local sum (compute)
- Sum (reduce)

AccSum proportional to $\log_2(\text{cond})$.

In practice: how to compute σ and how to extract q_i from p_i:

- $M = \lceil log_2(n+2) \rceil$
- $\sigma = 2^M \cdot max\{|p_i|\}$, for $n + 2 \leq 2^M$
- Extraction of the higher part q_i of p_i: $[q_i, p_i'] = Fast2Sum(\sigma, p_i)$

Properties

- Number of extractions: proportional to $\log_2(\text{cond})$
- Number of summands limitations: $n \lesssim 1/\sqrt{\mathbf{u}} \approx 7 \cdot 10^7$ (alternative exist for larger n).

AccSum proportional to $\log_2(\text{cond})$.

In practice: number of extractions and flop

TABLE 4.3
Minimum treatable length n for condition number $\texttt{eps}^{-1}$ *in IEEE 754 double precision.*

$m = 2$	$m = 3$	$m = 4$
4094	$1.0 \cdot 10^6$	$6.7 \cdot 10^7$

TABLE 4.4
Floating-point operations needed for different dimension and condition number.

n	cond	AccSum	Sum2	XBLAS
1000	10^6	$7n$	$7n$	$10n$
1000	10^{16}	$11n$	$7n$	$10n$
10^6	10^{16}	$15n$	$7n$	$10n$

Computing sums as accurately as possible: step 2

Partial sums by exponent (sorting) + distillation

Zhu-Hayes : HybridSum (09), OnLineExact (10)

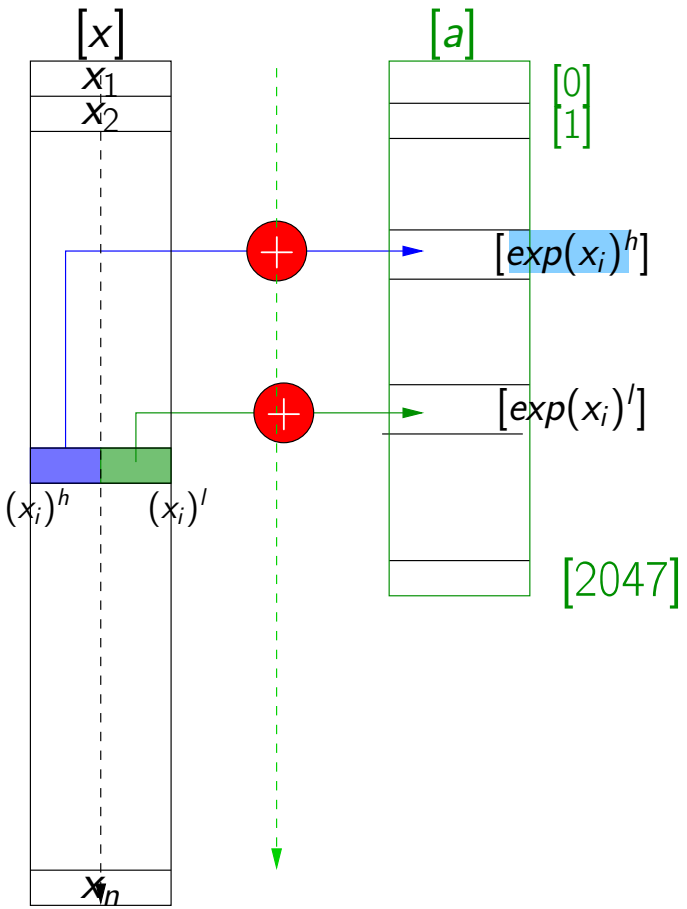
Exponent extraction and sorting, then distillation

Principle :

1. With no loss, transforming the entry vector to one or two (short) vectors which size equals the exponent range: 2048 for IEEE-b64 ;
   - HybridSum : split + 1 short vector
   - OnLineExact : 2Sum+ 2 vectors concatenated in a very short vector (of non zero entries)
2. then distillate this very short vector.

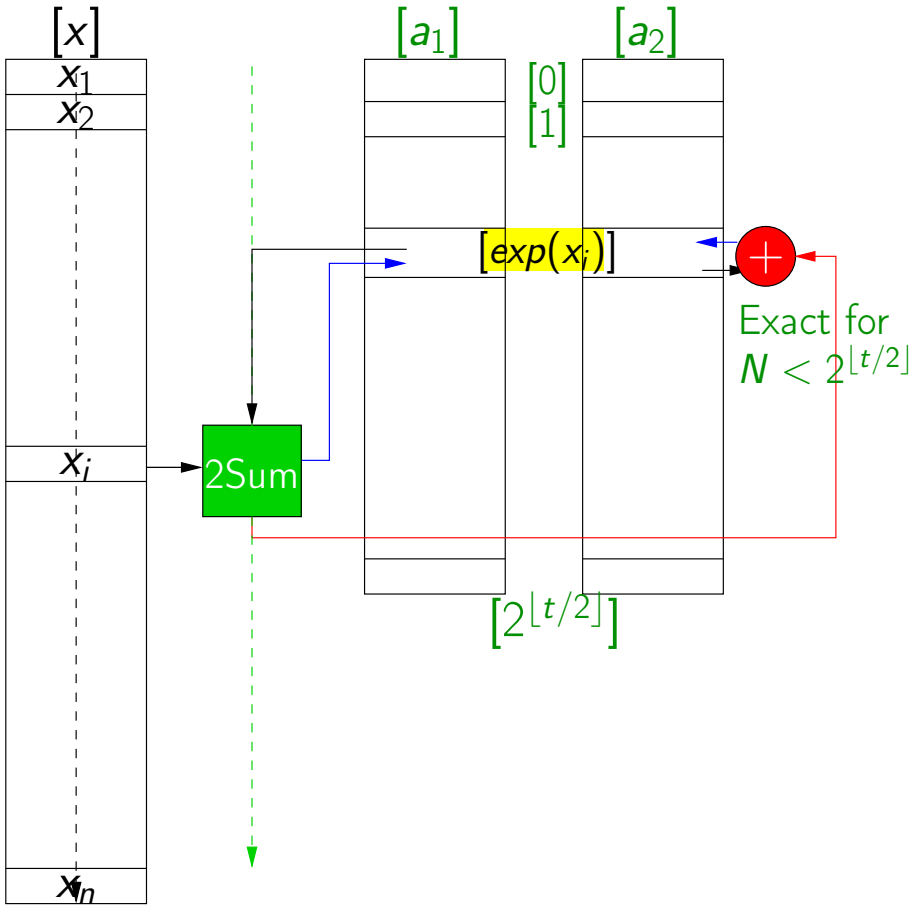
HybridSum (09) while $n \leq 7 \cdot 10^7$ in IEEE-b64

Idea : split and exact accumulation with $\bigoplus$

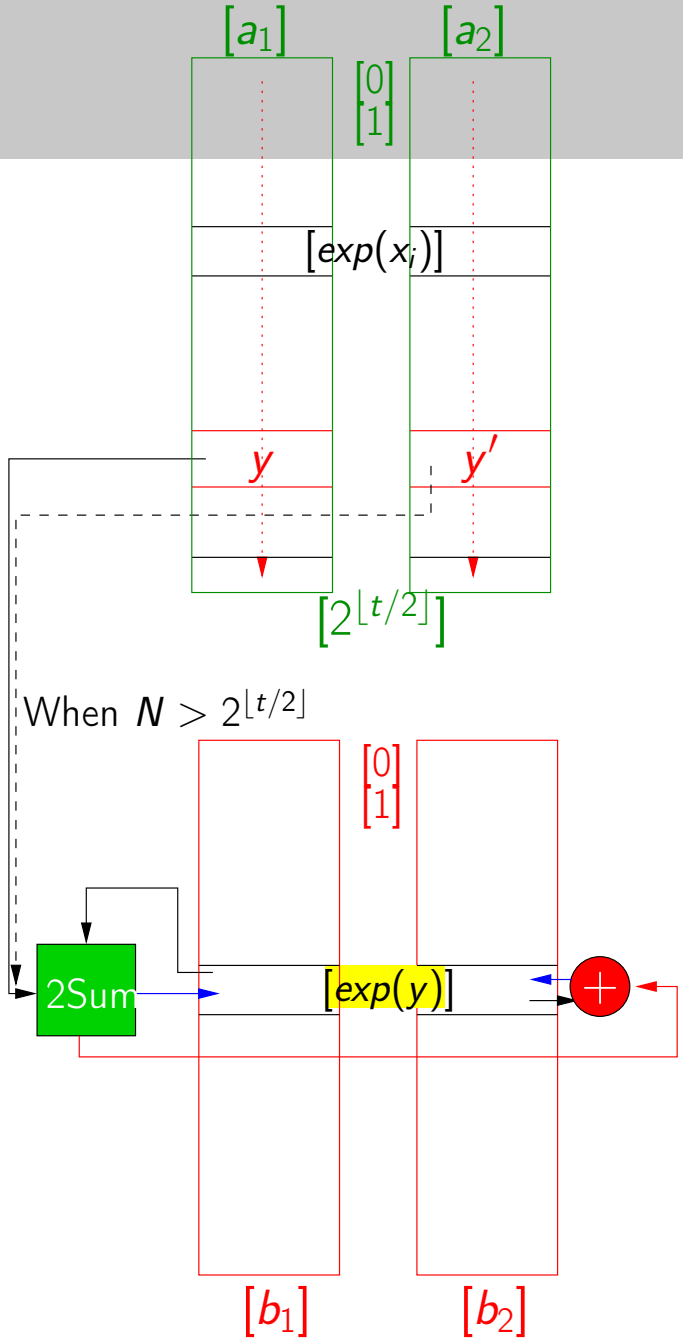


Exact for
$N < 2^{\lfloor t/2 \rfloor}$

OnLineExact (10)

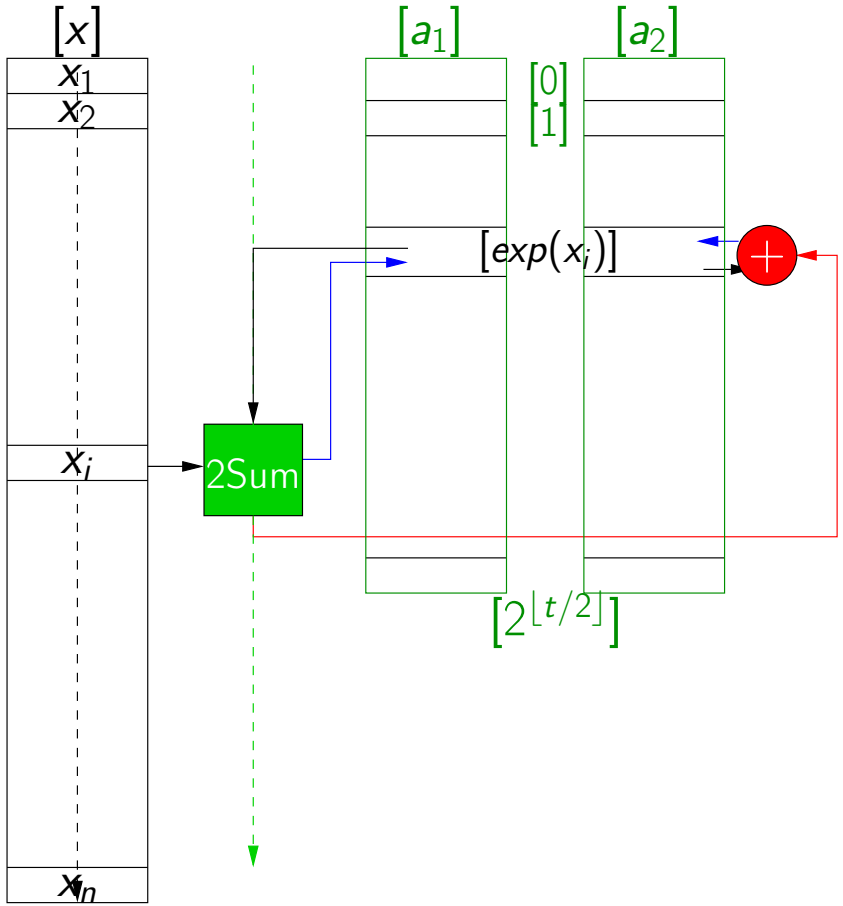


OnLineExact (10)

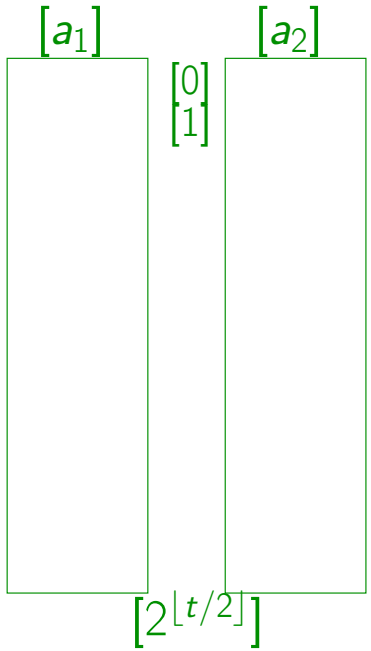


Then Swap($[b_1, b_2], [a_1, a_2]$)

OnLineExact (10)



OnLineExact (10)



a := $[a_1]$&$[a_2]$;
Remove 0;
Return iFastSum(a, length(a));

More EFT and accurate algorithms

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)

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

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

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)

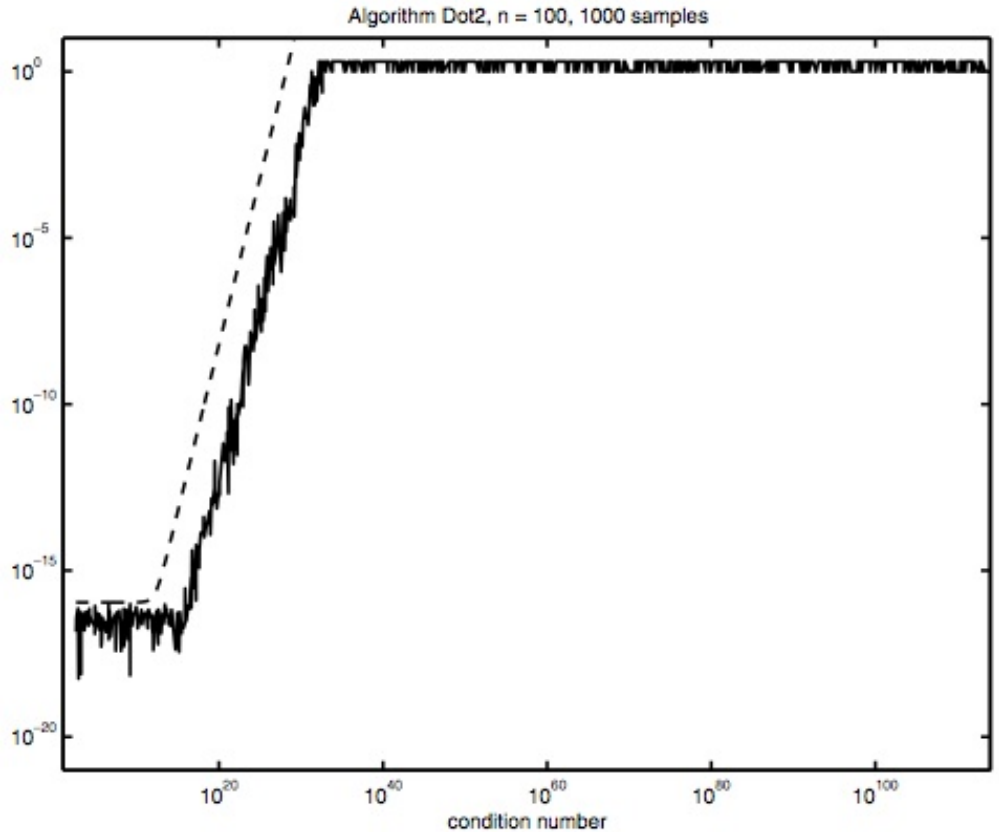
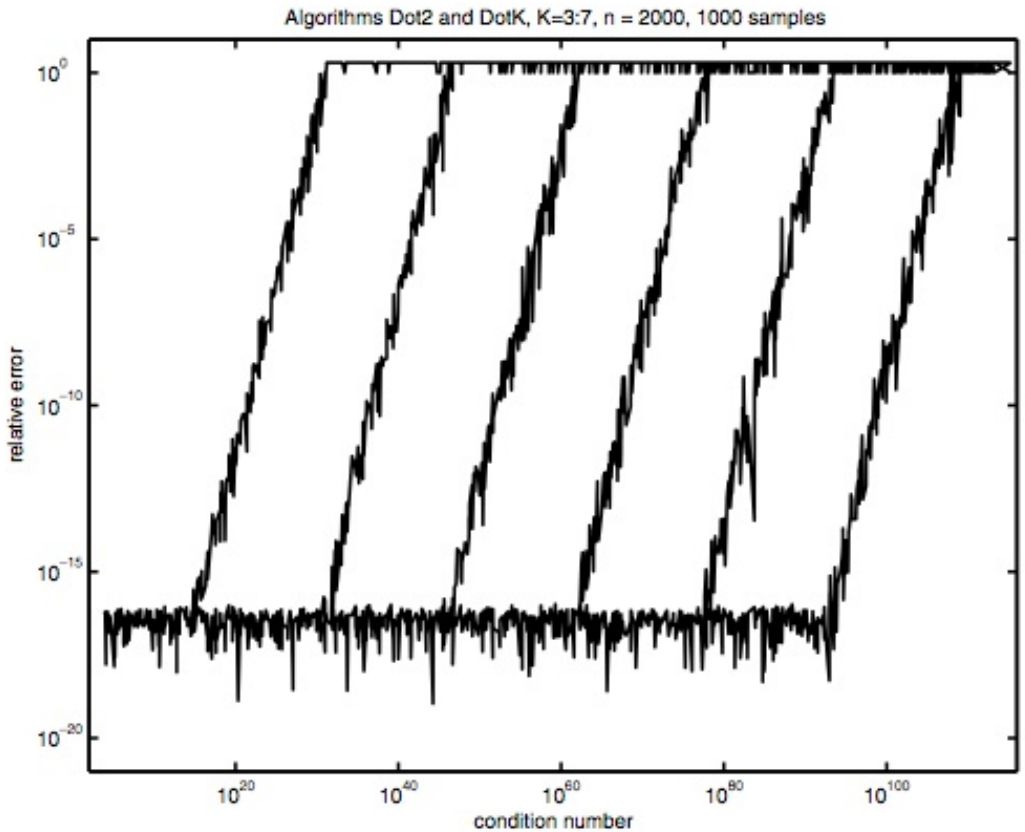


FIG. 6.1. *Test results for Algorithm* 5.3 (`Dot2`), $n = 100$, 1000 *samples.*

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

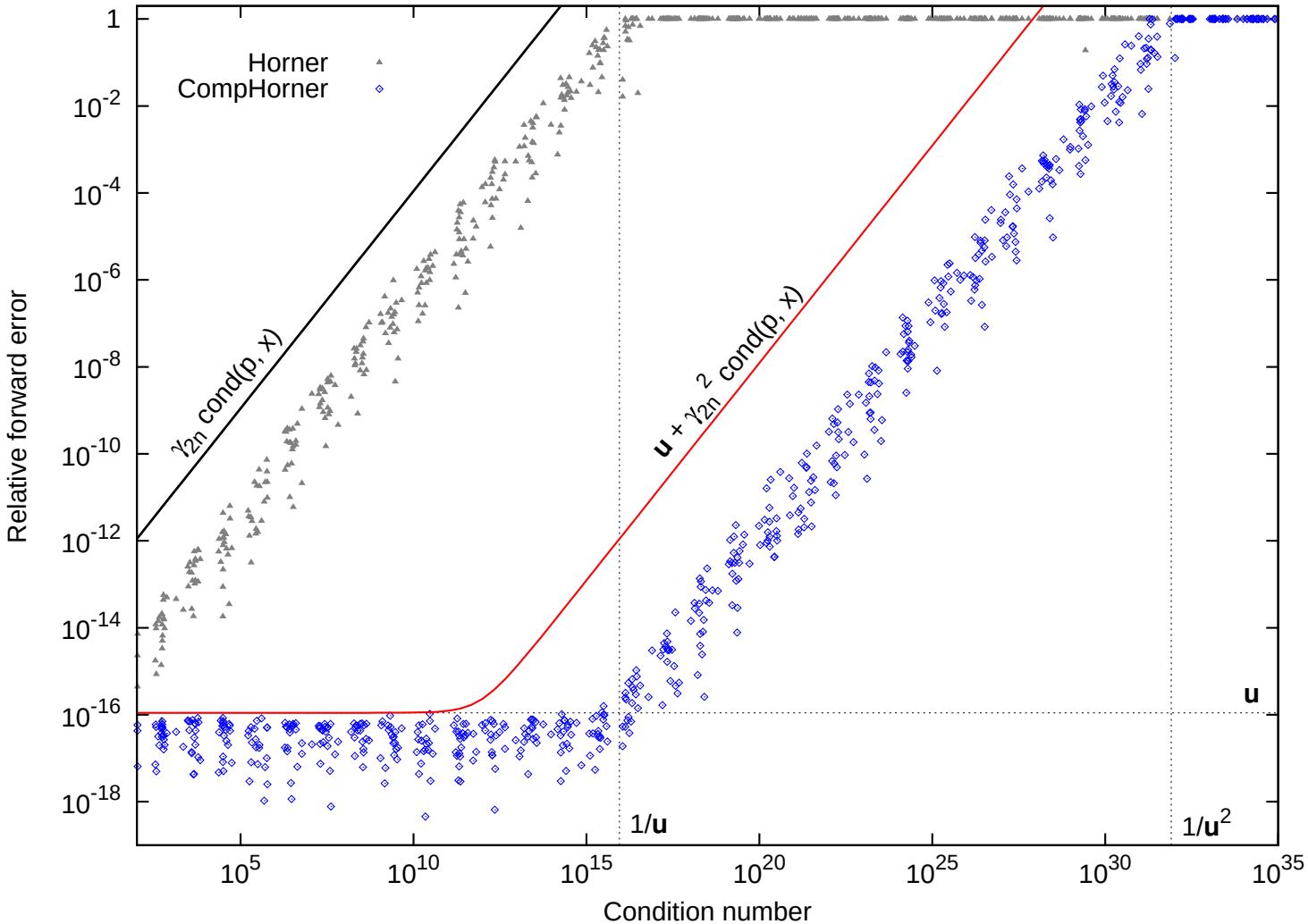
Algorithm (Compensated Horner)

$r_n = a_n$;
`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;

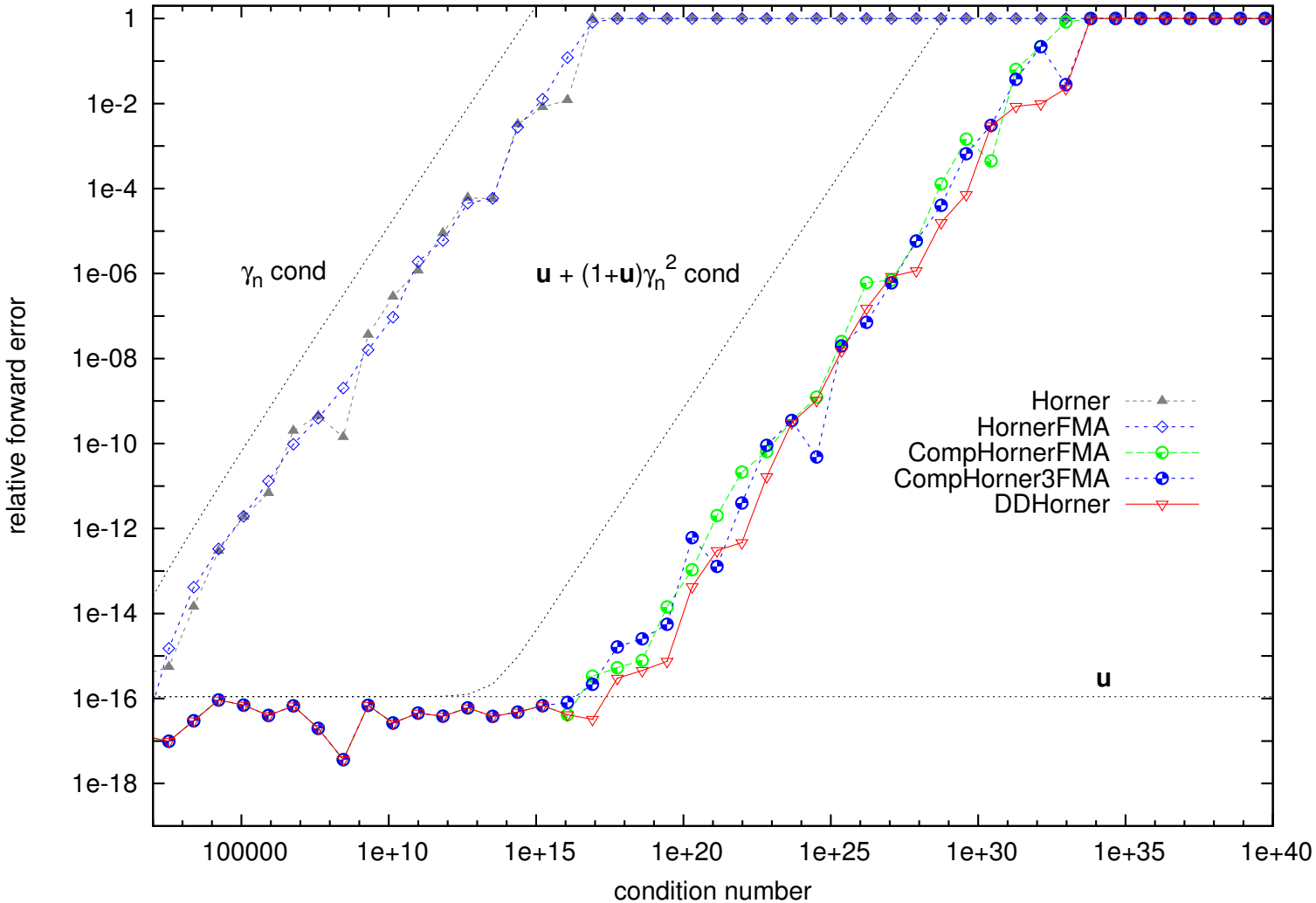
Theorem (Langlois-Louvet, 07)

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. Computing sums as accurately as possible
   - iFastSum: distillation again
   - Accsum
   - Hybridsum and OnLineExactSum
4. More EFT and accurate algorithms
   - Some more results
5. 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!

Part II

More reproducibility

6. Motivation

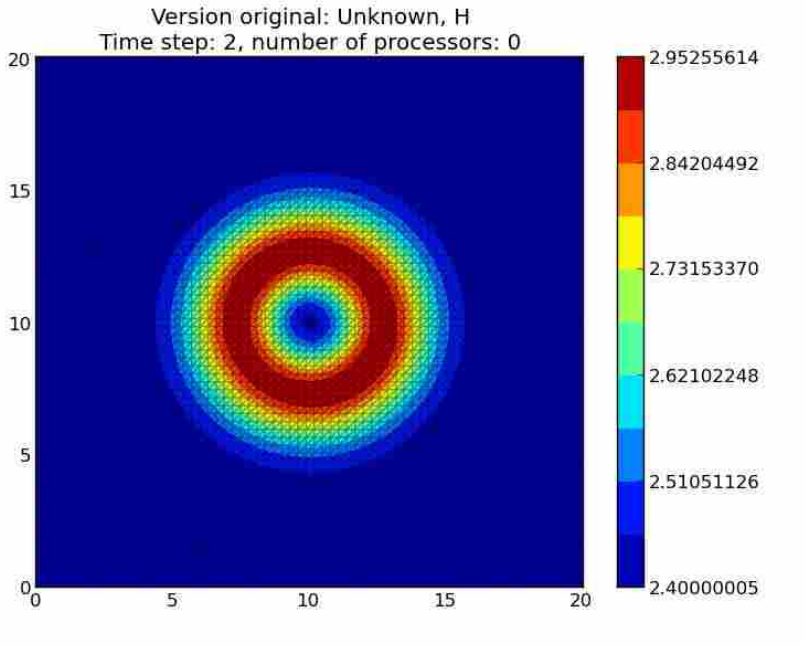
7. Reproducibility failure in a finite element simulation
   - Sequential and parallel FE assembly
   - Sources of non reproducibility in Telemac2D

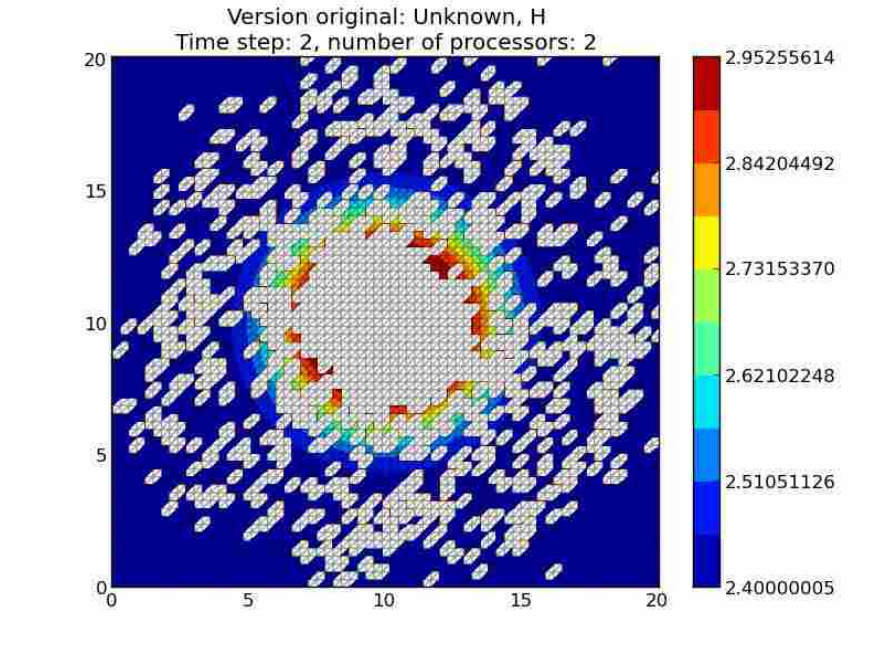
8. Recovering reproducibility
   - Reproducible parallel FE assembly
   - A first application: reproducible Tomawac
   - Reproducible algebraic operations
   - Reproducible conjugate gradient
   - Reproducible Telemac2D

9. Efficiency

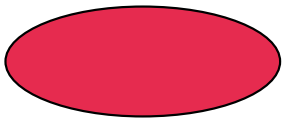
10. More reproducibility: conclusion

Reproducibility failure in Telemac2D simulation

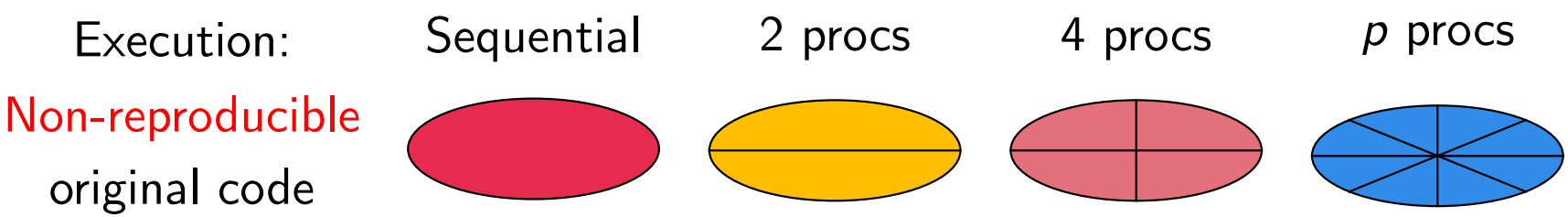




Recovering numerical reproducibility

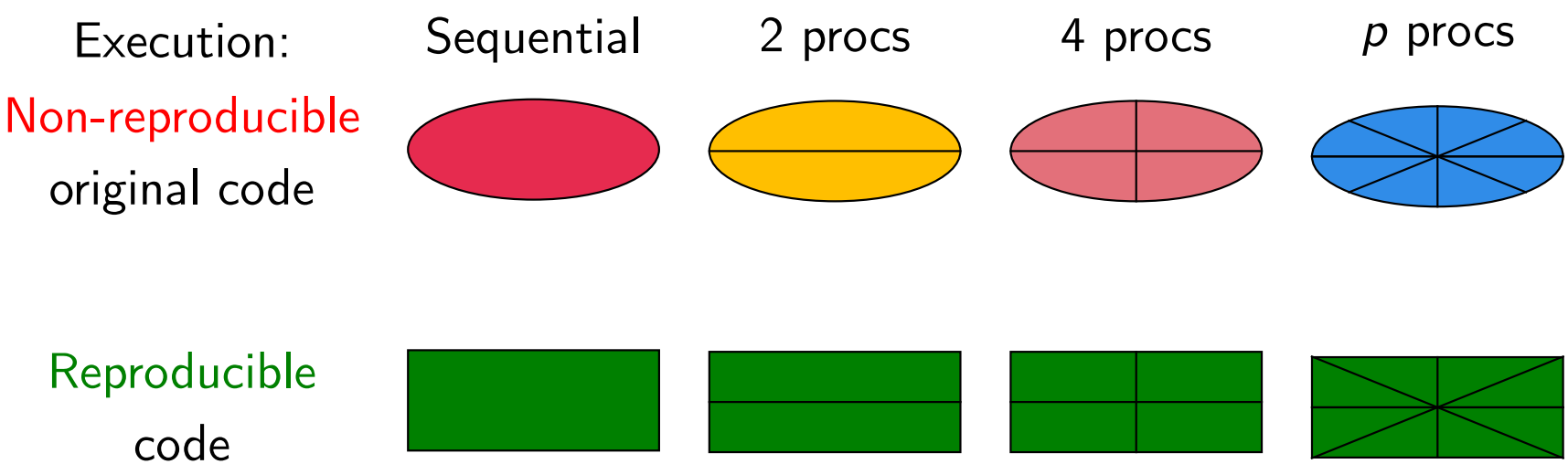
Execution:	Sequential	2 procs	4 procs	p procs
Original code				

Recovering numerical reproducibility



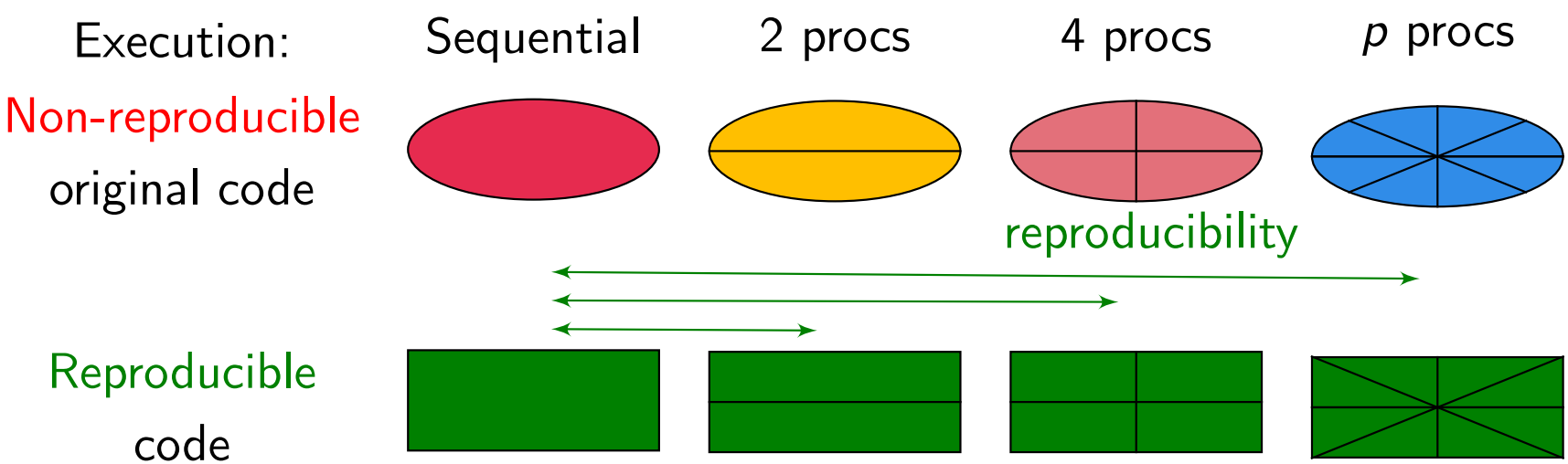
Recovering numerical reproducibility

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



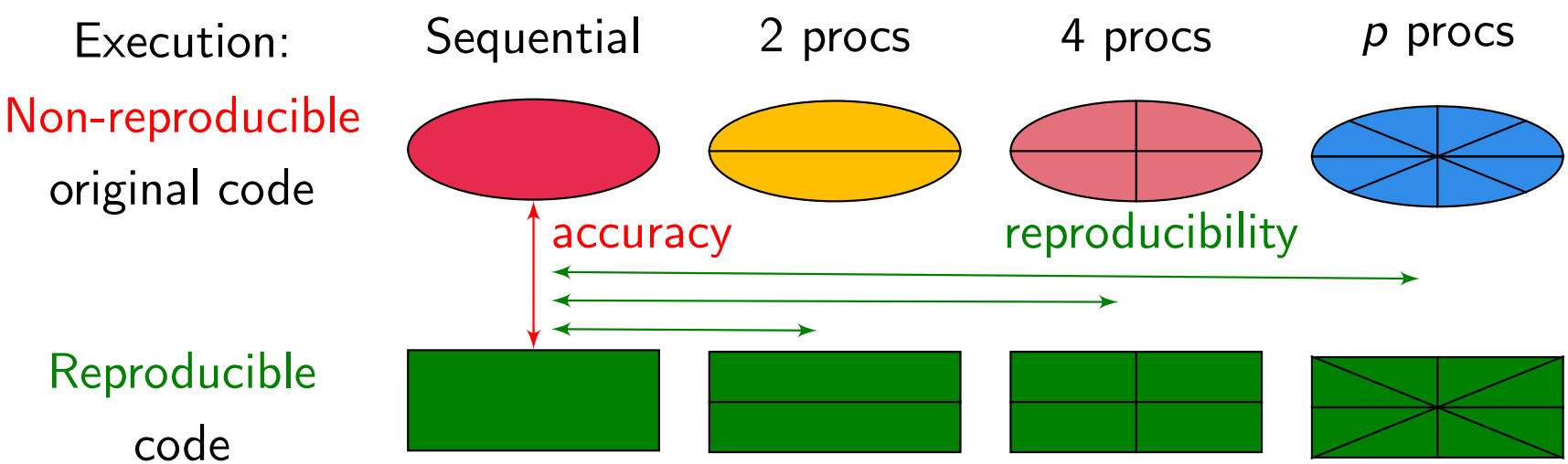
Recovering numerical reproducibility

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



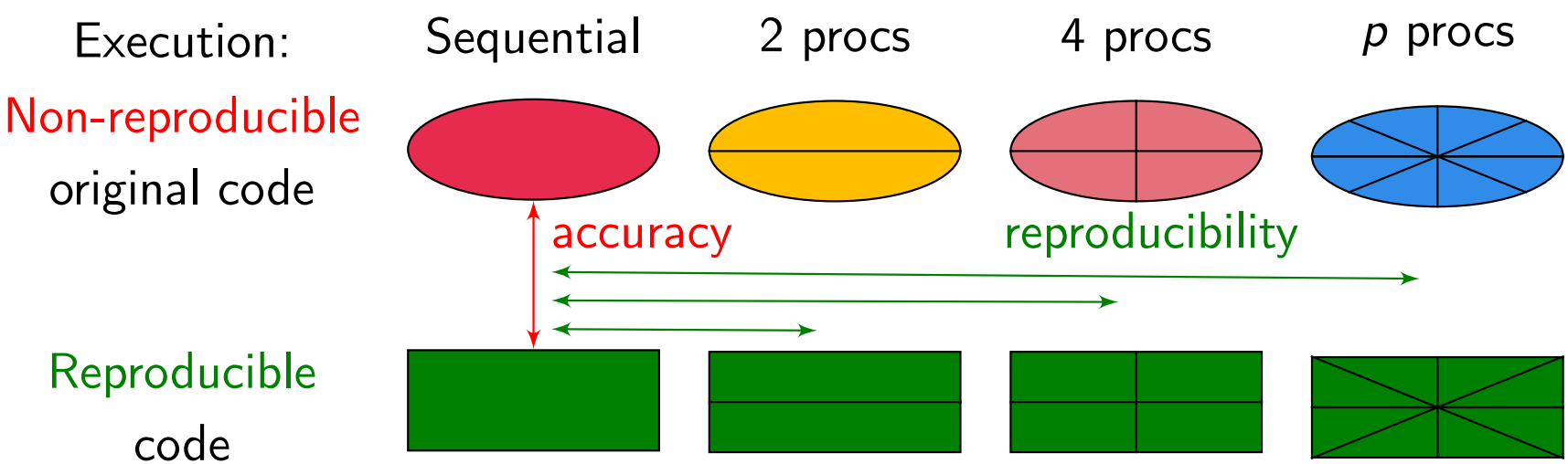
Recovering numerical reproducibility

- Reproducibility: bitwise identical results for every p-parallel run, $p \geq 1$
- Reproducibility $\neq$ Accuracy



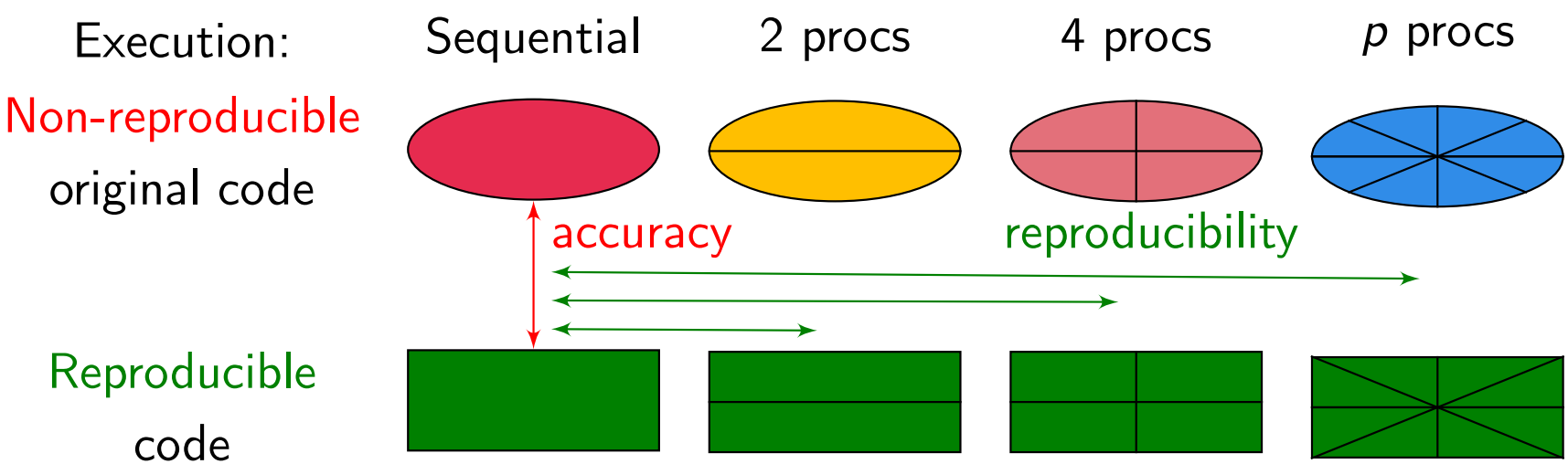
Recovering numerical reproducibility

- Reproducibility: bitwise identical results for every p-parallel run, $p \geq 1$
- Reproducibility $\neq$ Accuracy
- Failures reported in numerical simulation for energy [21], dynamic weather science [4], dynamic molecular [20], dynamic fluid [17]



Recovering numerical reproducibility

- Reproducibility: bitwise identical results for every p-parallel run, $p \geq 1$
- Reproducibility $\neq$ Accuracy
- Failures reported in numerical simulation for energy [21], dynamic weather science [4], dynamic molecular [20], dynamic fluid [17]
- How to debug? How to test? How to validate? How to receive legal agreements?



Today's issues

Case study

- Industrial scale software: openTelemac-Mascaret
- Finite element simulation, domain decomposition, linear system solving
- 2 modules: Tomawac, Telemac2D

Today's issues

Case study

- Industrial scale software: openTelemac-Mascaret
- Finite element simulation, domain decomposition, linear system solving
- 2 modules: Tomawac, Telemac2D

Feasibility

- How to recover reproducibility?
- Sources of non-reproducibility?
- Do existing techniques apply? how easily?
- Compensation yields reproducibility here!

Today's issues

Case study

- Industrial scale software: openTelemac-Mascaret
- Finite element simulation, domain decomposition, linear system solving
- 2 modules: Tomawac, Telemac2D

Feasibility

- How to recover reproducibility?
- Sources of non-reproducibility?
- Do existing techniques apply? how easily?
- Compensation yields reproducibility here!

Efficiency

- How much to pay for reproducibility?
- $\times 1.2 \leftrightarrow \times 2.3$ extra-cost which decreases as the problem size increases
- OK to debug, to validate and even to simulate!

6. Motivation

7. Reproducibility failure in a finite element simulation
   - Sequential and parallel FE assembly
   - Sources of non reproducibility in Telemac2D

8. Recovering reproducibility
   - Reproducible parallel FE assembly
   - A first application: reproducible Tomawac
   - Reproducible algebraic operations
   - Reproducible conjugate gradient
   - Reproducible Telemac2D

9. Efficiency

10. More reproducibility: conclusion

Parallel reduction and compensation techniques

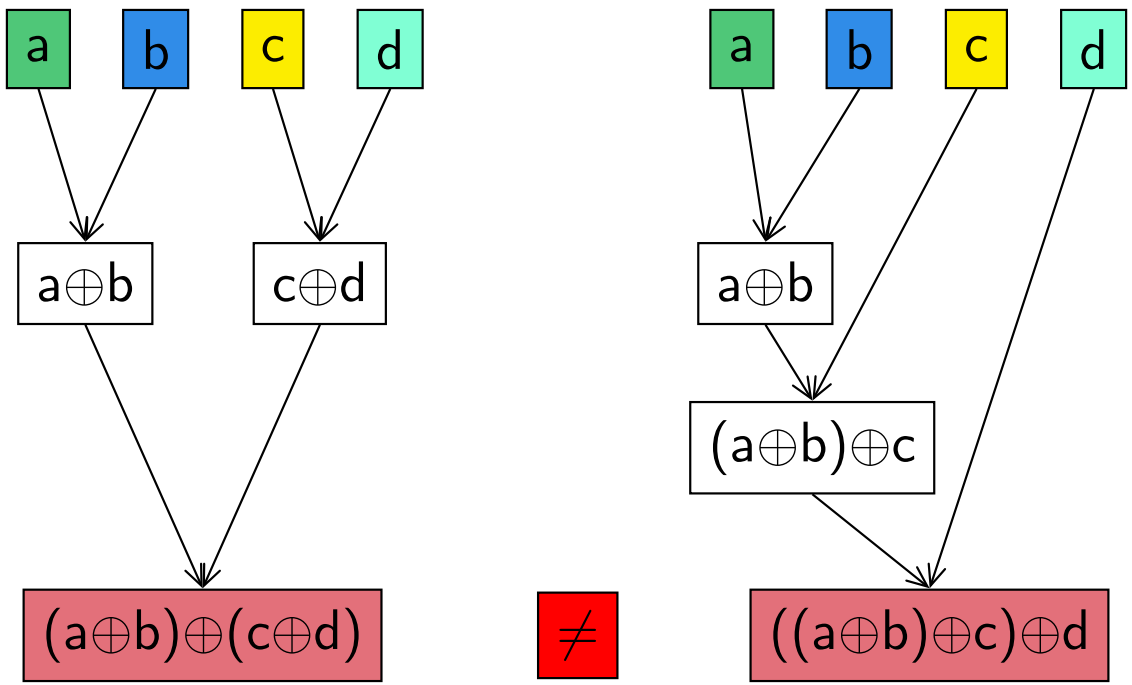
Non associative floating point addition

- The computed value depends on the operation order

Parallel reduction and compensation techniques

Non associative floating point addition

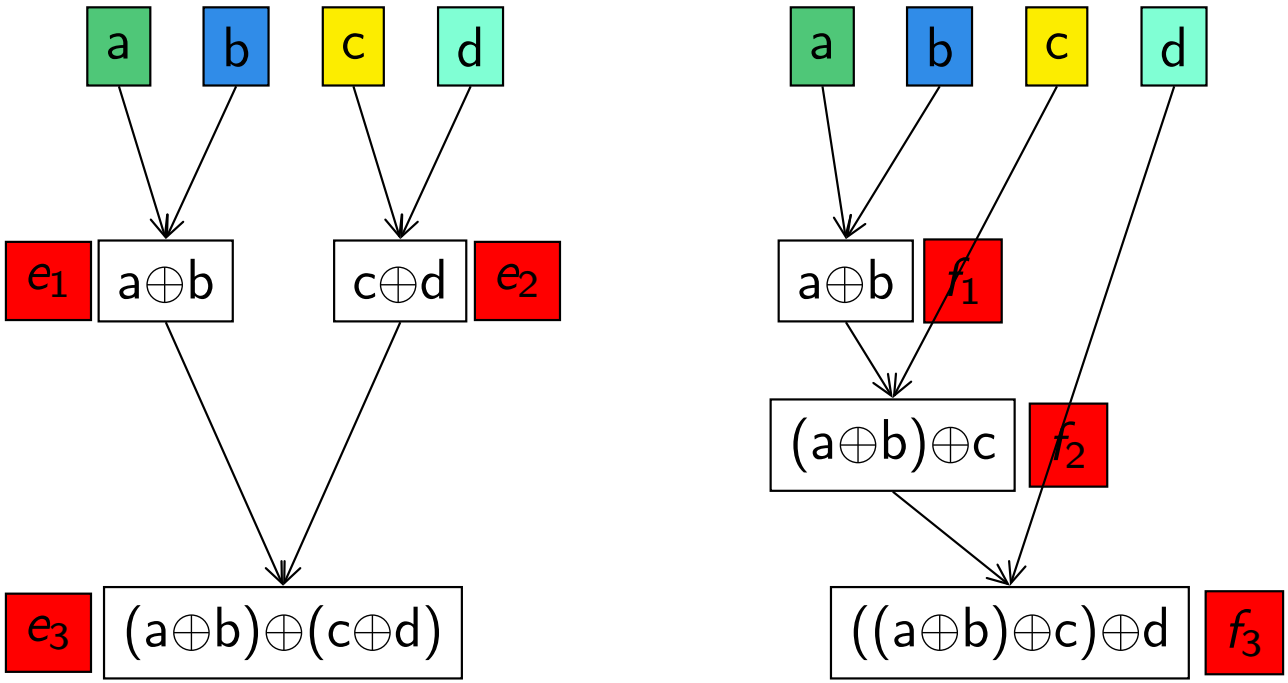
- The computed value depends on the operation order
- Parallel reduction of undefined order generates reproducibility failure



Parallel reduction and compensation techniques

Non associative floating point addition

- The computed value depends on the operation order
- Parallel reduction of undefined order generates reproducibility failure
- Compensate rounding errors with error free transformations



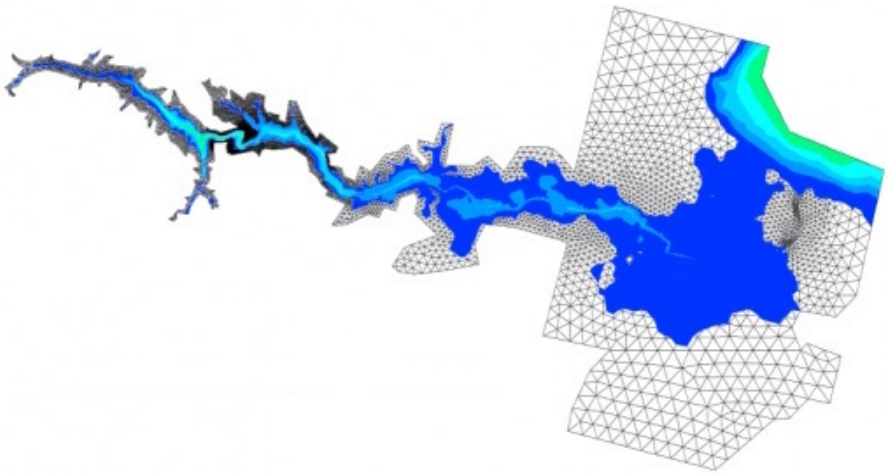
$$((a\oplus b)\oplus(c\oplus d)) \oplus ((e_1 \oplus e_2) \oplus e_3) = (((a\oplus b)\oplus c)\oplus d) \oplus((f_1 \oplus f_2) \oplus f_3)$$

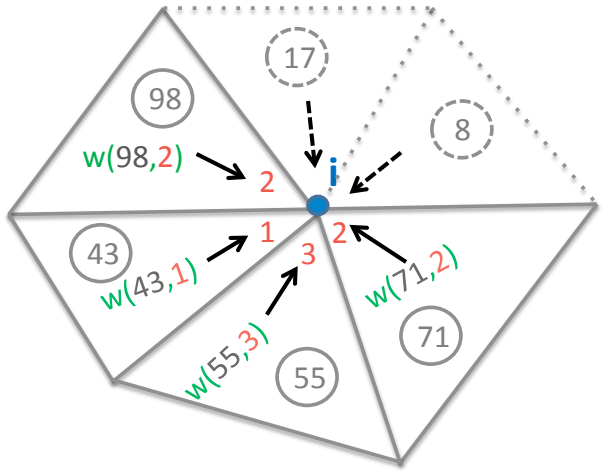
Should be repeted for too ill-conditionned sums

Finite element assembly: the sequential case

The assembly step: $V(i) = \sum_{elements} W_{el}(i)$

- compute the inner node values $V(i)$
- accumulating local W_{el} for every el that contains i





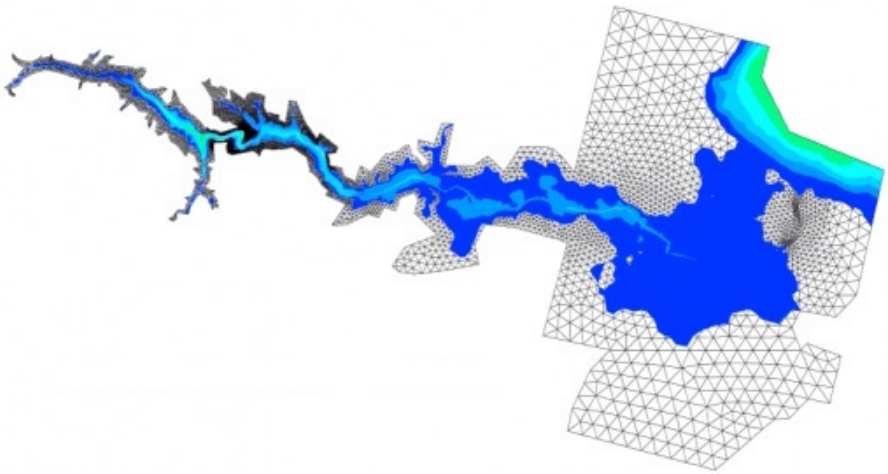
The assembly loop

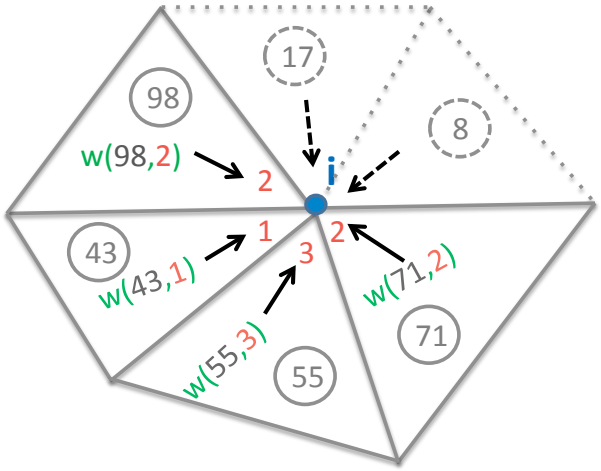
```
for  p = 1,np          //p: triangular local number (np=3)
  for el = 1,nel
     i = IKLE(el,p) %
     V(i) = V(i) +  W(el,p)       //i: domain global number
```

Finite element assembly: the sequential case

The assembly step: $V(i) = \sum_{elements} W_{el}(i)$

- compute the inner node values $V(i)$
- accumulating local W_{el} for every el that contains i





The assembly loop

```
for  p = 1,np          //p: triangular local number (np=3)
  for el = 1,nel
     i = IKLE(el,p) %            <-- LOOP INDEX INDIRECTION
     V(i) = V(i) +  W(el,p)        //i: domain global number
```

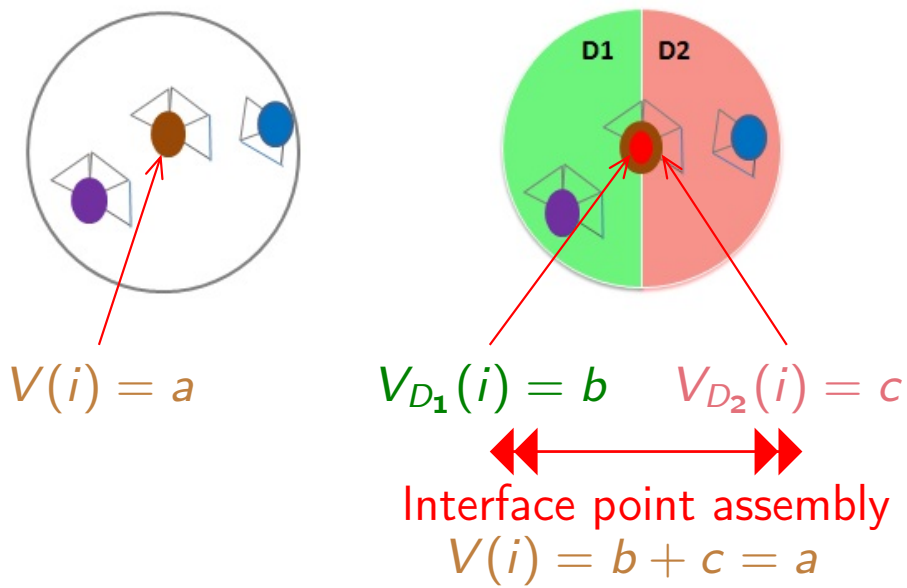
Finite element assembly: the parallel case

Interface point assembly: communications and reductions

$V(i) = \sum_{D_k} V(i)$ for sub-domains $D_k, k = 1...p$

sequential parallel sub-domains

inner nodes $\longrightarrow$ interface points



Exact arithmetic

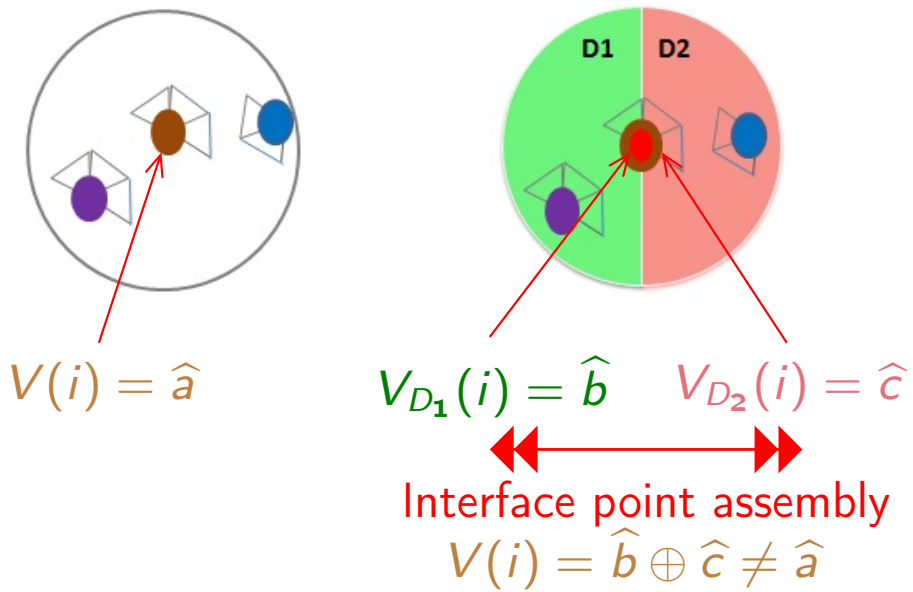
Finite element assembly: the parallel case

Interface point assembly: communications and reductions

$V(i) = \bigoplus_{D_k} V(i)$ for sub-domains $D_k, k = 1...p$

sequential parallel sub-domains

inner nodes $\longrightarrow$ interface points



Floating point arithmetic

Basic ingredients

Sources of non reproducibility in Telemac2D

Culprits: theory

1. Building step: interface point assembly
2. Solving step (conjugate gradient): parallel matrix-vector and dot products

Sources of non reproducibility in Telemac2D

Culprits: theory

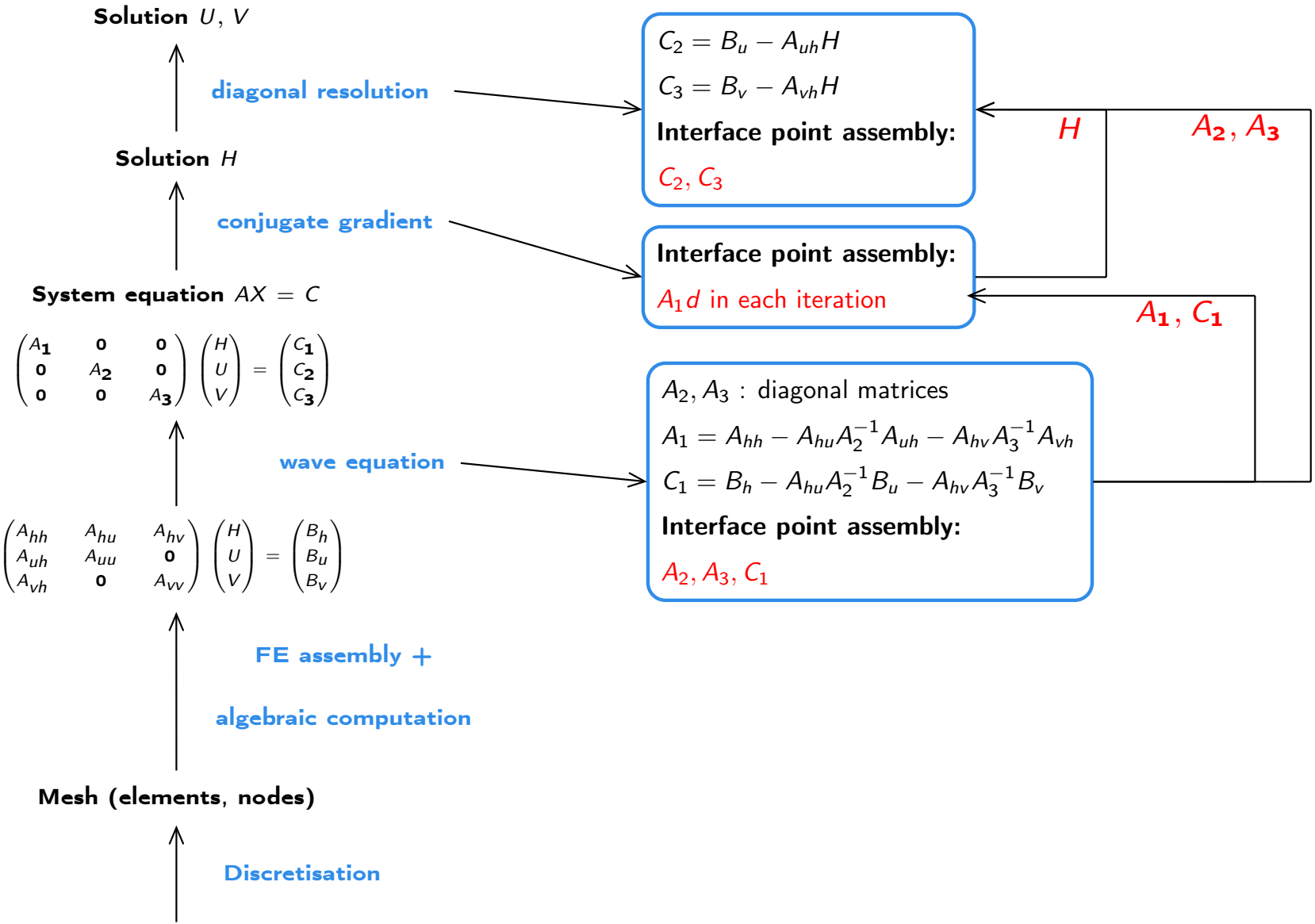
1. Building step: interface point assembly
2. Solving step (conjugate gradient): parallel matrix-vector and dot products

Culprits: practice = optimization

- Interface point assembly and linear system solving are merged
- Element-by-element storage (EBE) of the FE matrix
  - EBE parallel matrix-vector product: no BLAS
  - Everything is vector, no matrix!
- Wave equation, "mass-lumping" and associated algebraic transformations
  - System decoupling and many diagonal matrices
  - Everything is vector, no matrix!

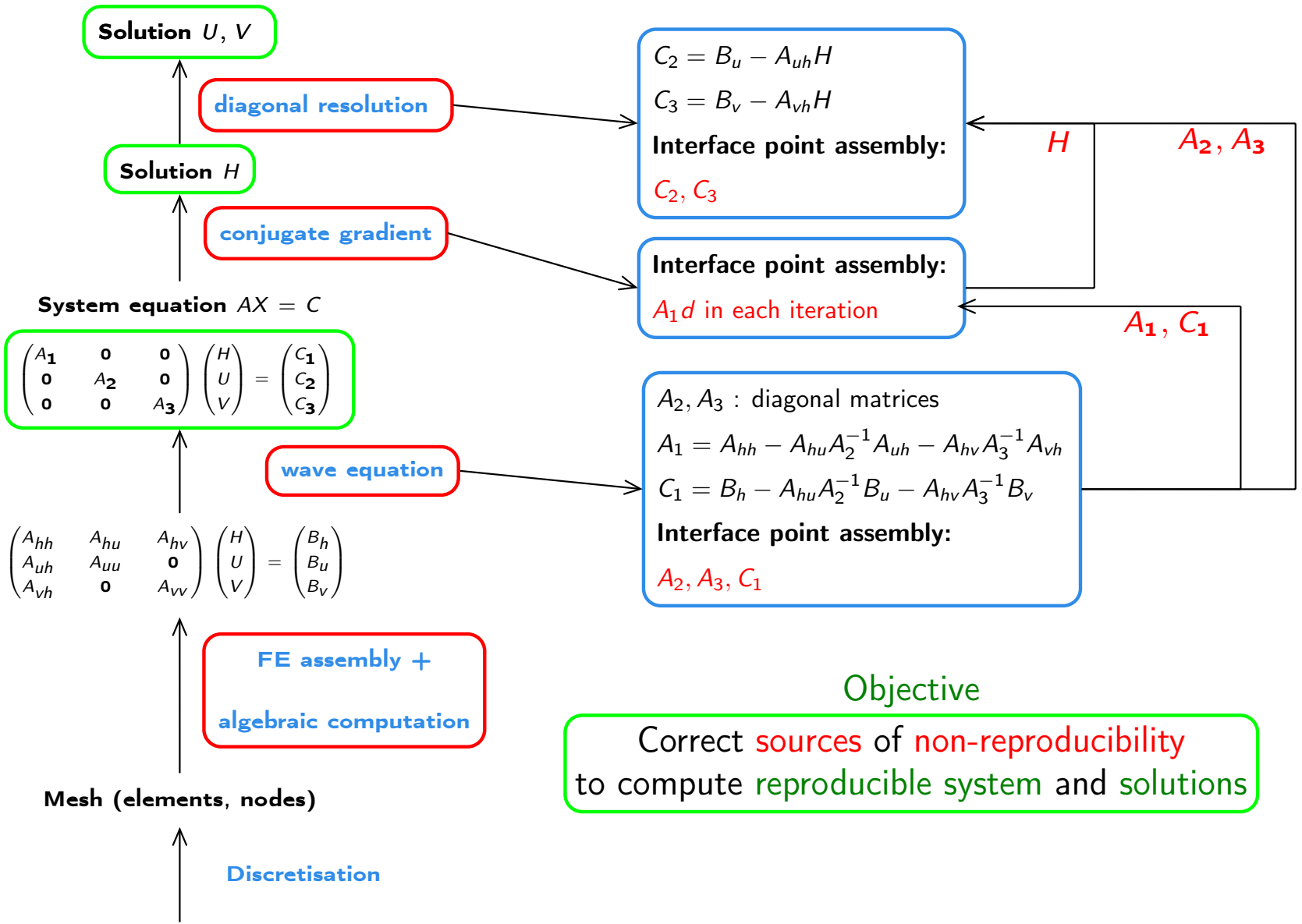
Sources of non reproducibility in Telemac2D

The Telemac2D FE steps



Sources of non reproducibility in Telemac2D

The Telemac2D FE steps



Objective

Correct sources of non-reproducibility
to compute reproducible system and solutions

Recovering reproducibility in a finite element resolution

Recovering reproducibility in Telemac2D

Sources

- FE assembly: diagonal of the matrices and second members
- Resolution: EBE matrix-vector and dot products
- Wave equation: algebraic transformations and diagonal resolutions

Reproducible resolution: principles

- vector $V \rightarrow [V, E_V] \rightarrow V + E_V$
- Computes E_V in the FE assembly of V
- Propagates E_V over each V operation
- Compensates all nodes while assembling the Interface Point
- Compensate MPI parallel dot products that include MPI reduction

Reproducible FE assembly

i: any node; $W_{el}(i)$: contribution for every el that contains i

Original FE assembly: $V(i) = \sum_{elements} W_{el}(i)$

$$V(i) = W_{el_1}(i) + W_{el_2}(i) + \cdots + W_{el_{ni}}(i)$$

Modified FE assembly: $[V(i), E_V(i)] = \text{ReprodAss}_{elements} W_{el}(i)$

$$V(i) = W_{el_1}(i) + W_{el_2}(i) + \cdots + W_{el_{ni}}(i)$$
$$\downarrow \qquad \downarrow \qquad \downarrow$$
$$e_1 \qquad e_2 \qquad e_{ni}$$

$$E_V(i) = e_1 + e_2 + \cdots + e_{ni-1}$$

Reproducible interface point assembly

i: one interface point of $D_1, D_2, \cdots, D_{k-1}, D_k$

Original IP assembly: $V(i) = \sum_{D_k} V(i)$

Communicate V_{D_i} to $D_1, D_2, \cdots$ and compute:

$$V(i) = V_{D_1}(i) + V_{D_2}(i) + \cdots + V_{D_{k-1}}(i) + V_{D_k}(i)$$

Modified IP assembly: $[V(i), E_V(i)] = \text{ReprodAss}_{D_k}[V(i), E_{V_{D_k}}(i)]$

Communicate $[V_{D_i}, E_{D_{V_i}}]$ to $D_1, D_2, \cdots$ and compute:

$$V(i) = V_{D_1}(i) + V_{D_2}(i) + \cdots + V_{D_{k-1}}(i) + V_{D_k}(i)$$

$$\downarrow \quad \downarrow \quad \downarrow$$

$$\delta_1 \quad \delta_2 \quad \delta_{k-1}$$

$$E_V(i) = (E_{V_{D_1}}(i) + E_{V_{D_2}}(i) + \delta_1) + \cdots + (E_{V_{D_{k-1}}}(i) + E_{V_{D_k}}(i) + \delta_{k-1})$$

Reproducibility: at the IP assembly step

$[V, E_V] \longrightarrow V + E_V$ compensates every node value

FE assembly: implementation

Main steps

- For every vector value: FE assembly + IP assembly

ASSVEC subroutine

```
1  !X refers to VEC and ERRX refers to SVEC%E
2  DO IDP = 1 , NDP
3    DO IELEM = 1 , NELEM
4
5       X(IKLE(IELEM,IDP))=X(IKLE(IELEM,IDP))+W(IELEM,IDP)
6
7
8
9
10
11   ENDDO
12 ENDDO
```

```
1  !FE assembly in ASSVEC
2
3  CALL ASSVEC(VEC, IKLE, NPT ,NELEM,NELMAX,IELM1,
4  &   T,INIT,LV,MSK,MASKEL,NDP)
5
6
7
8
9  !Interface point assembly in PARCOM
10 IF(ASSPAR) CALL PARCOM(SVEC,2,MESH)
11
12
13
14
15
16
```

Interface point assembly: implementation

```
1  ! Receive step
2  DO IL=1,NB_NEIGHB
3    IKA = NB_NEIGHB_PT(IL)
4    IPA = LIST_SEND(IL)
5    CALL P_IREAD(BUF_RECV(1,IL),IAN*IKA*NPLAN*8,IPA,PARACO_MSG_TAG,RECV_REQ(IL))
6
7  ENDDO
8  ! Send step
9  DO IL=1,NB_NEIGHB
10   IKA = NB_NEIGHB_PT(IL)
11   IPA = LIST_SEND(IL)
12   ! Initializes the communication arrays
13   K = 1
14   DO J=1,NPLAN
15     DO I=1,IKA
16       II=NH_COM(I,IL)
17       BUF_SEND(K,IL) =V1(II,J)
18
19       K=K+1
20     ENDDO
21   ENDDO
22   CALL P_IWRIT(BUF_SEND(1,IL),IAN*IKA*NPLAN*8,IPA,PARACO_MSG_TAG,SEND_REQ(IL))
23
24 ENDDO
25 ! Wait received messages
26 DO IL=1,NB_NEIGHB
27   IKA = NB_NEIGHB_PT(IL)
28   IPA = LIST_SEND(IL)
29   CALL P_WAIT_PARACO(RECV_REQ(IL),1)
30   K=1
31   DO J=1,NPLAN
32     DO I=1,IKA
33       II=NH_COM(I,IL)
34       V1(II,J)=V1(II,J)+ BUF_RECV(K,IL)
35
36
37
38
39       K=K+1
40     ENDDO
41   ENDDO
42 ENDDO
```

Reproducible FE assembly: implementation

Main steps

- For every vector value: FE assembly + IP assembly + compensation

ASSVEC subroutine

```
!X refers to VEC and ERRX refers to SVEC%E
DO IDP = 1 , NDP
  DO IELEM = 1 , NELEM
    IF (MODASS.EQ.1)
 &    X(IKLE(IELEM,IDP))=X(IKLE(IELEM,IDP))+W(IELEM,IDP)
    ELSEIF (MODASS.EQ.3) THEN
      CALL 2SUM(X(IKLE(IELEM,IDP)),
 &    W(IELEM,IDP),X(IKLE(IELEM,IDP)),ERROR)
      ERRX(IKLE(IELEM,IDP))=ERRX(IKLE(IELEM,IDP))+ERROR
    ENDIF
  ENDDO
ENDDO
```

```
! Note: VEC is a reference to SVEC%R
IF(MODASS.EQ.1) THEN
   CALL ASSVEC(VEC, IKLE, NPT ,NELEM,NELMAX,IELM1,
&   T,INIT,LV,MSK,MASKEL,NDP)
ELSEIF (MODASS.EQ.3) THEN
   CALL ASSVEC(VEC, IKLE, NPT ,NELEM,NELMAX,IELM1,
&  T,INIT,LV,MSK,MASKEL,NDP,SVEC%E)
ENDIF
! Implicit modification in PARCOM
IF(ASSPAR) CALL PARCOM(SVEC,2,MESH)
IF(ASSPAR.AND.MODASS.EQ.3) THEN
!The compensation of all the values
  DO I = 1 , MESH%NPOIN
    VEC(I)= VEC(I)+SVEC%E(I)
  ENDDO
ENDIF
```

Reproducible interface point assembly: implementation

```
1  ! Receive step
2  DO IL=1,NB_NEIGHB
3    IKA = NB_NEIGHB_PT(IL)
4    IPA = LIST_SEND(IL)
5    CALL P_IREAD(BUF_RECV(1,IL),IAN*IKA*NPLAN*8,IPA,PARACO_MSG_TAG,RECV_REQ(IL))
6    CALL P_IREAD(BUF_RECV_ERR(1,IL),IAN*IKA*NPLAN*8,IPA,PARACO_MSG_TAG,RECV_REQ(IL))
7  ENDDO
8  ! Send step
9  DO IL=1,NB_NEIGHB
10   IKA = NB_NEIGHB_PT(IL)
11   IPA = LIST_SEND(IL)
12   ! Initializes the communication arrays
13   K = 1
14   DO J=1,NPLAN
15     DO I=1,IKA
16       II=NH_COM(I,IL)
17       BUF_SEND(K,IL) =V1(II,J)
18       BUF_SEND_ERR(K,IL) =ERRX(II)
19       K=K+1
20     ENDDO
21   ENDDO
22   CALL P_IWRIT(BUF_SEND(1,IL),IAN*IKA*NPLAN*8,IPA,PARACO_MSG_TAG,SEND_REQ(IL))
23   CALL P_IWRIT(BUF_SEND_ERR(1,IL),IAN*IKA*NPLAN*8,IPA,PARACO_MSG_TAG,SEND_REQ(IL))
24 ENDDO
25 ! Wait received messages
26 DO IL=1,NB_NEIGHB
27   IKA = NB_NEIGHB_PT(IL)
28   IPA = LIST_SEND(IL)
29   CALL P_WAIT_PARACO(RECV_REQ(IL),1)
30   K=1
31   DO J=1,NPLAN
32     DO I=1,IKA
33       II=NH_COM(I,IL)
34 ! Original version: V1(II,J)=V1(II,J)+ BUF  RECV(K,IL)
35       CALL 2SUM(V1(II,J),BUF_RECV(K,IL),V1(II,J),ERROR1)
36       CALL 2SUM(ERRV(II),BUF_RECV_ERR(K,IL),ERRV(II),ERROR2)
37       ERROR=ERROR1+ERROR2
38       ERRV(II)=ERRV(II)+ERROR
39       K=K+1
40     ENDDO
41   ENDDO
```

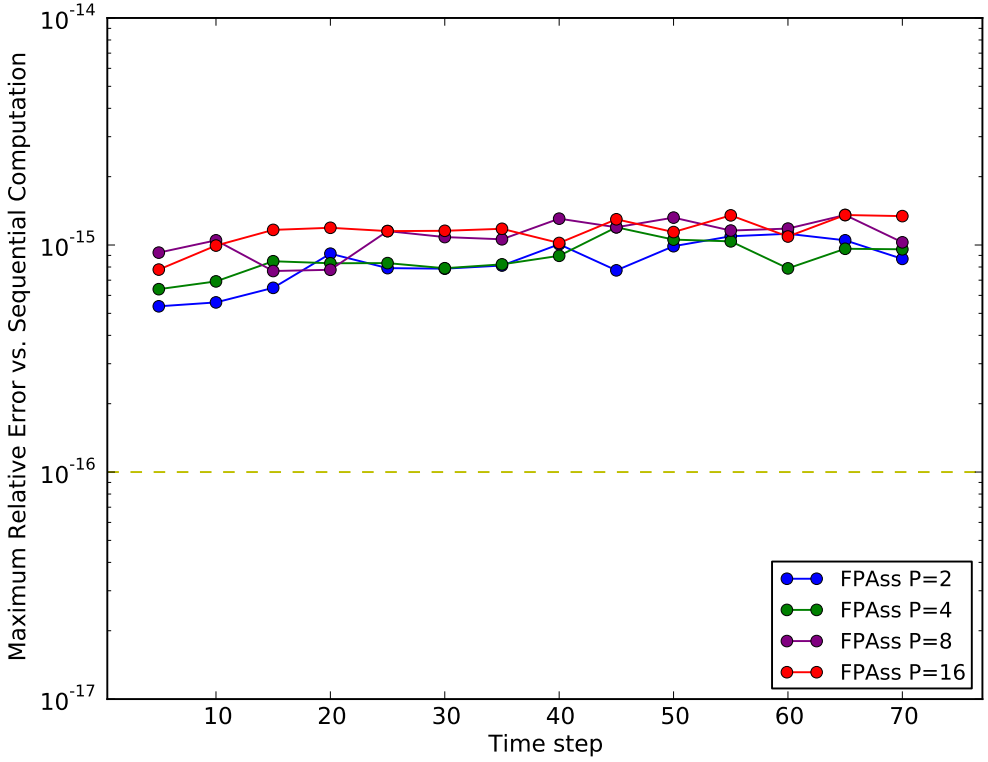
Reproducible algebraic operations

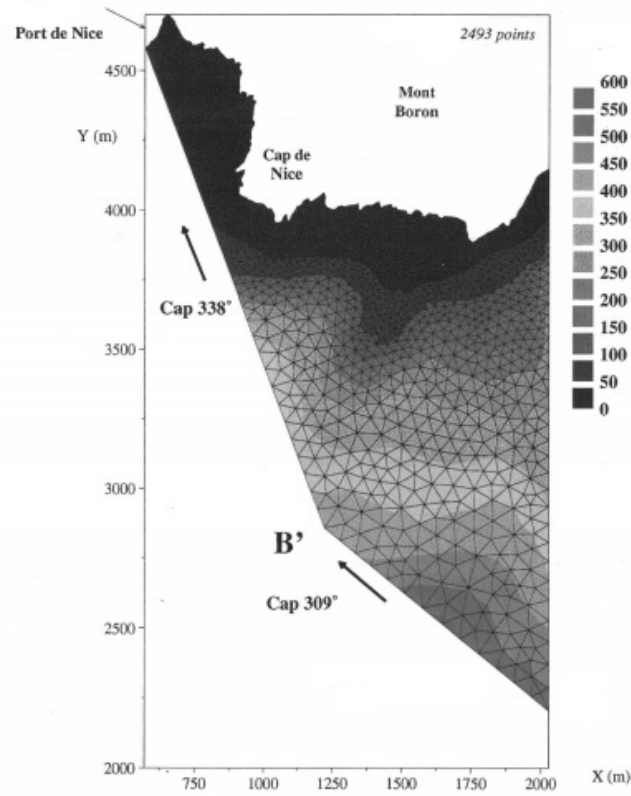
Tomawac case

Non-reproducible Tomawac

- Wave propagation in coastal areas
- Mesh node unknowns:

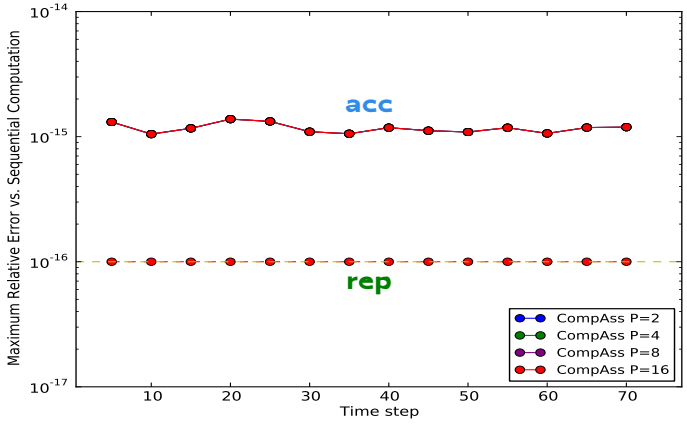
  height, frequency and direction of the waves

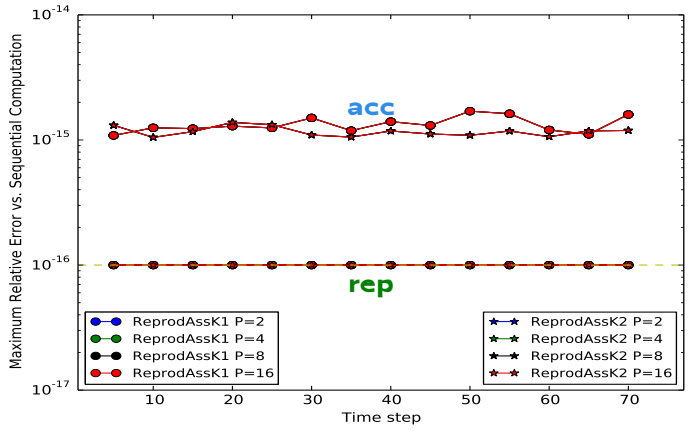


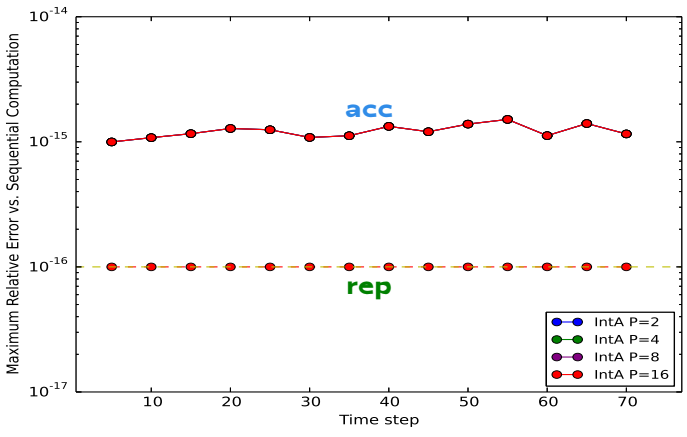


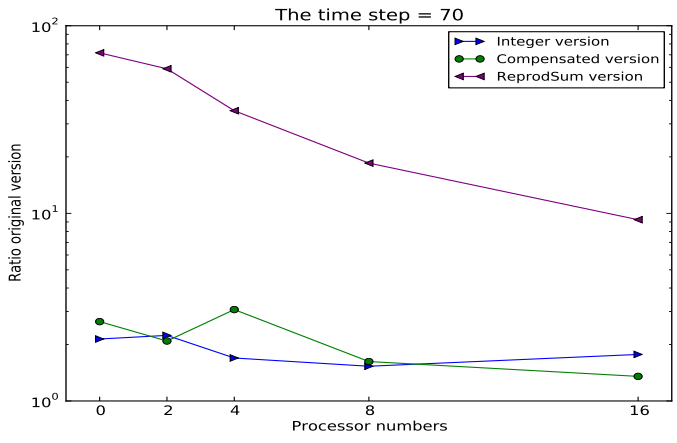
Reproducible Tomawac

- Accurate compensated summation [15]
- Integer conversion [16]
- Demmel-Nguyen's reproducible sums [2]









Reproducible algebraic operations

Algebraic operation: the vector case

Reproducible algebraic vector operations

- openTelemac's included library: BIEF
- Entry-wise vector ops: copy, opp/inv., add/sub, Hadamard prod., ...
- Applies also for diagonal matrix
- Propagate rounding errors to compensate while assembling IP

Example: Hadamard product

Original version

$X, Y \longmapsto V = X \circ Y$

$V(i) = X(i) \cdot Y(i)$

Modified version

$[X, E_X], [Y, E_Y] \longmapsto [V, E_V]$

with $(V, e_V) = \mathrm{2Prod}(X, Y)$

and $E_V = X \circ E_Y + Y \circ E_X + e_V$

What is reproducible now?

Most of the linear system:

- FE assembly
- algebraic vector operations
- interface point assembly

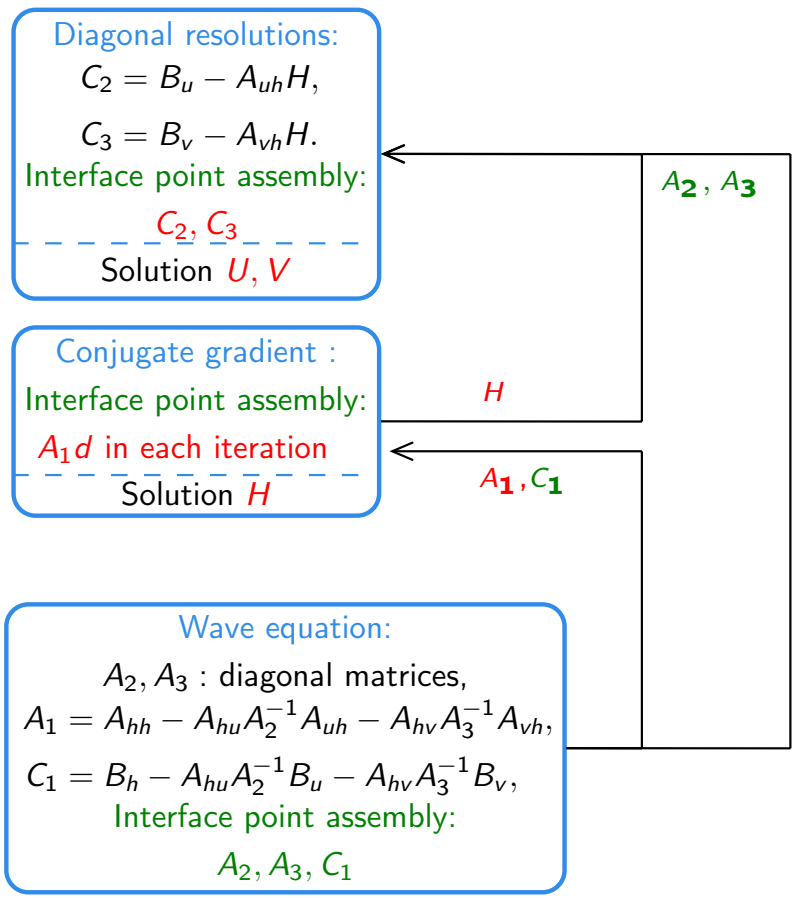
except:

- the matrix of the H system
- its dependencies: the second members of the U and V systems

Next step:

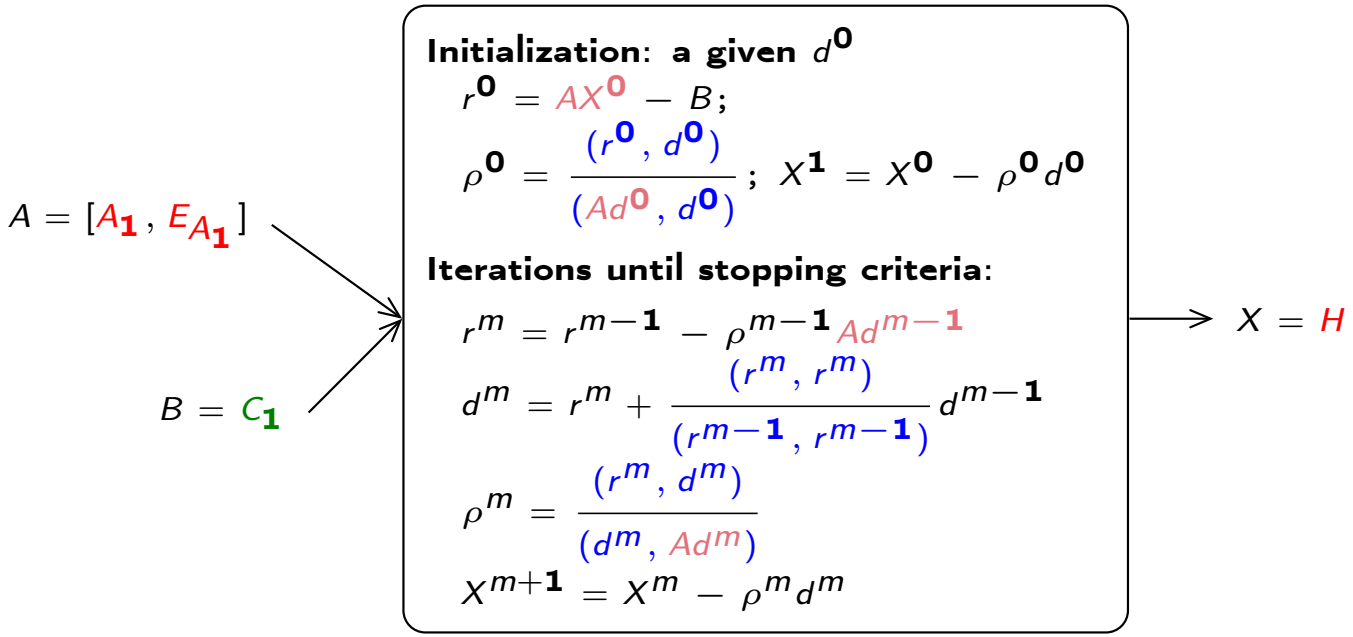
- conjugate gradient

Partially reproducible Telemac2D



Recovering reproducibility in a finite element resolution

Towards a reproducible conjugate gradient



Non-reproducibility sources

- EBE matrix-vector product
- dot product
  - MPI reduction
  - weighted dot products for IP shared by p sub-domains

The EBE storage

$$M = D + \sum_{el=1}^{nel} X_{el}$$

- nodes $i \in [1, np]$, elements $el \in [1, nel]$, element vertices $j, k, l \in el$
- M is decomposed as:
  1. assembled diagonal $D[np]$: $\quad D = [D(1), \cdots, D(np)]$
  2. elementary extra-diagonal $X_{el}[6]$:

$$X_{el} = \begin{pmatrix} \cdot & X_{jk}(el) & X_{jl}(el) \\ X_{kj}(el) & \cdot & X_{kl}(el) \\ X_{lj}(el) & X_{lk}(el) & \cdot \end{pmatrix} = [X_{el}(1), \cdots, X_{el}(6)]$$

The EBE Matrix-Vector product

$$R = M \cdot V = D \cdot V + \sum_{el=1}^{nel} X_{el} \cdot V_{el}$$

Steps of the EBE matrix-vector product

1. $R_1(i) = D(i) \cdot V(i), \quad i \in [1, np]$
2. $X_{el}.V_{el} = [X_{el}(1) \cdot V(k), X_{el}(2) \cdot V(l), \cdots, X_{el}(6) \cdot V(k)], \quad el \in [1, nel]$
3. FE assembly $\rightarrow R_2[np]$: $R_2 = \sum_{el=1}^{nel} X_{el} \cdot V_{el}$
4. $R = R_1 + R_2$
5. IP assembly: $R(i) = \sum_{D_k} R(i)$ for all IP i

Reproducible EBE matrix-vector product

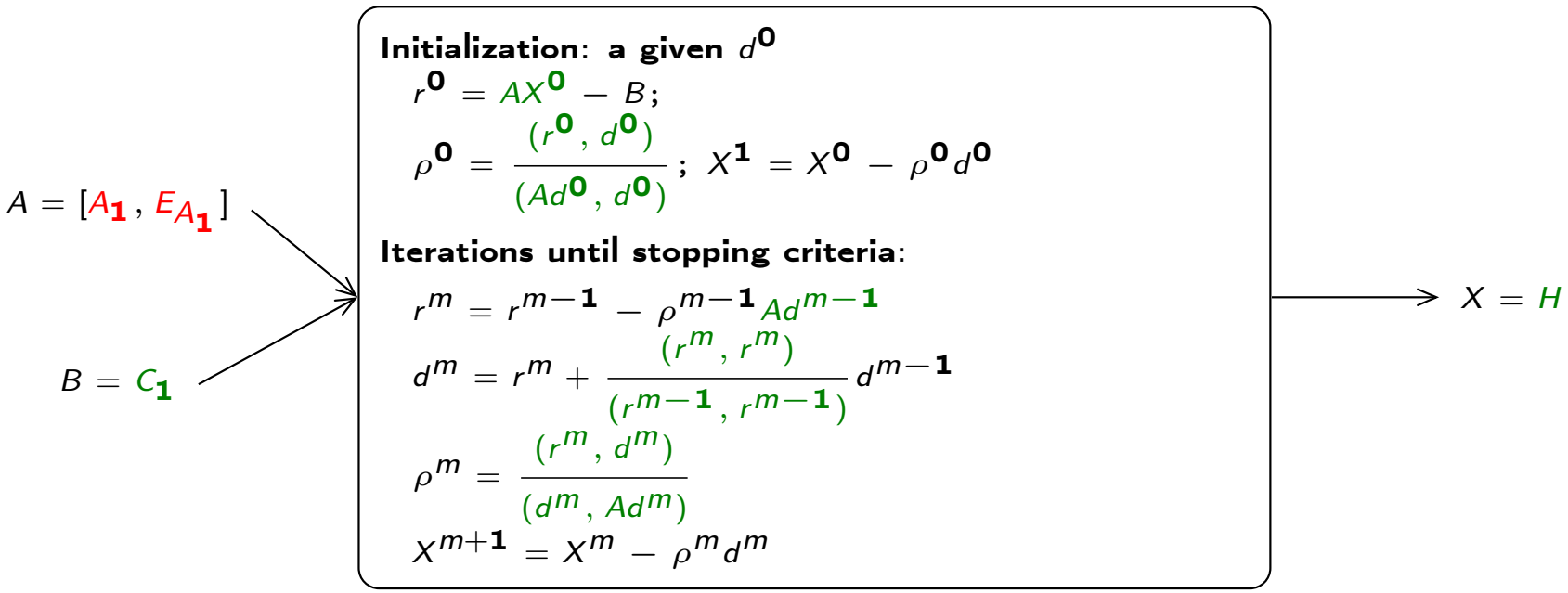
Original EBE matrix-vector product

- $R = D \cdot V + \sum_{el=1}^{nel} X_{el} \cdot V_{el}$
- $R(i) = \sum_{D_k} R(i)$

Reproducible EBE matrix-vector product

- $[R, E_R] = [D, E_D] \circ V + \text{ReprodAss}_{el=1}^{nel} X_{el} \cdot V_{el}$
- $R(i) = \text{ReprodAss}_{D_k} [R(i), E_R(i)]$
- Compensation: $R + E_R$

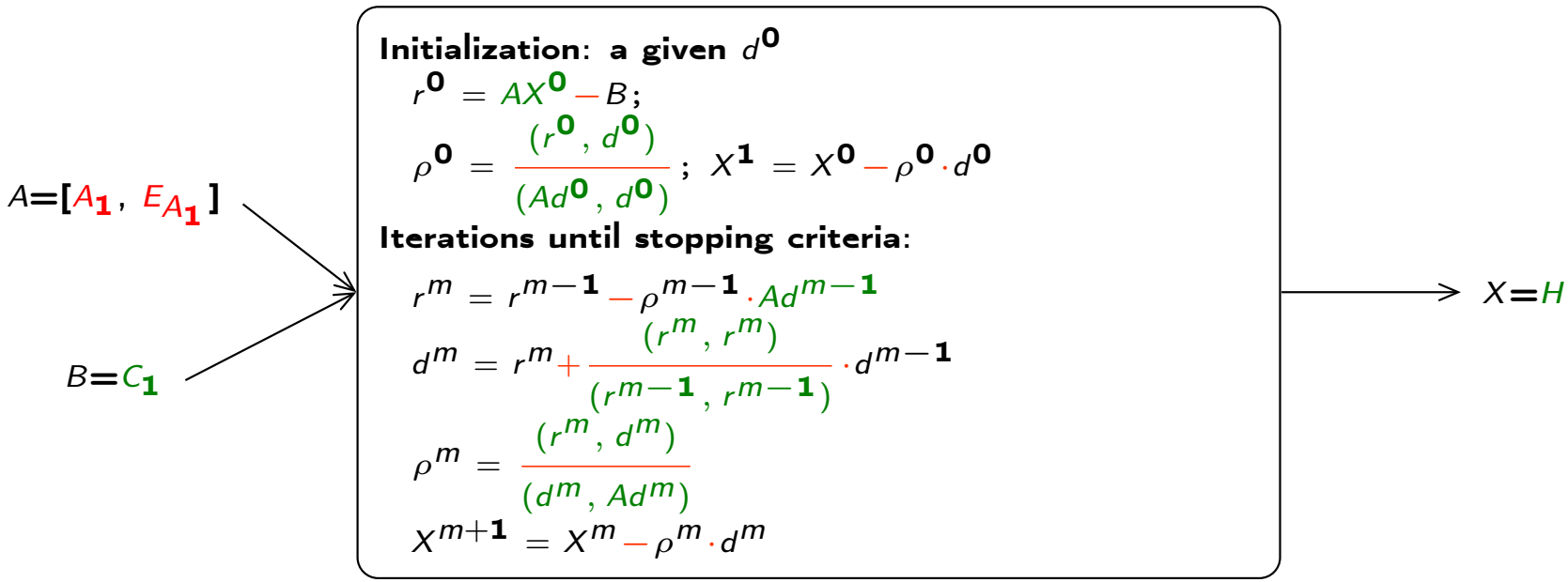
A reproducible conjugate gradient



Non-reproducibility: sources and solutions

- Reproducible EBE matrix-vector product
- dot product
  - MPI reduction: a parallel compensated dot2
  - weights: $(1/k, 1/k, \ldots, 1/k) \rightarrow (1, 0, \ldots, 0)$
- Reproducible operations $\longrightarrow$ reproducible results

A reproducible conjugate gradient



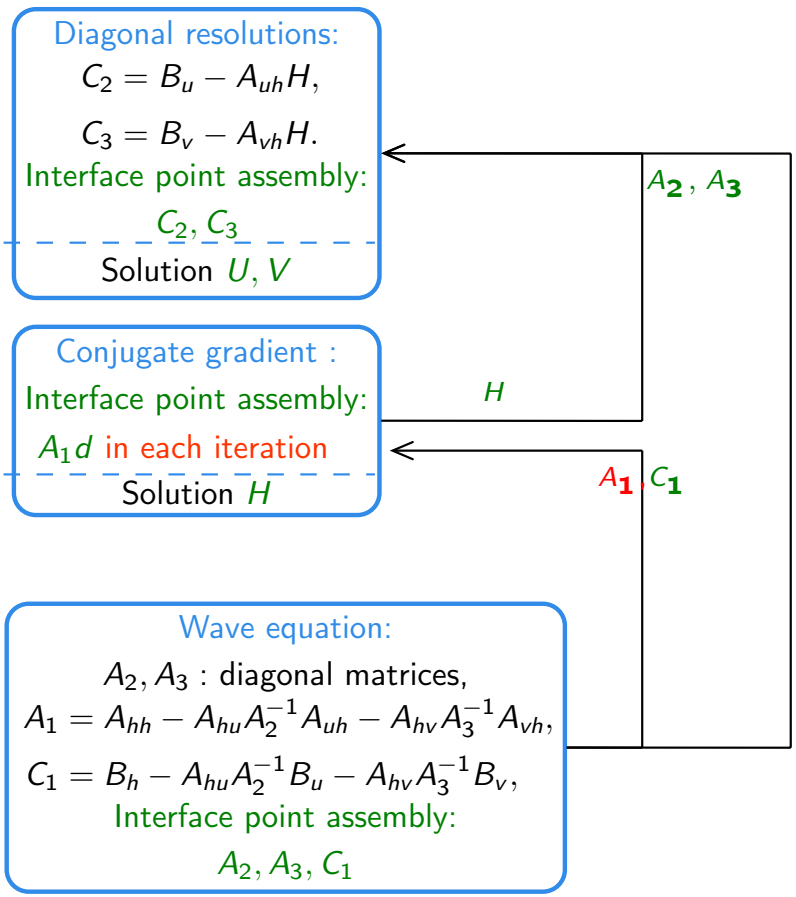
Non-reproducibility: sources and solutions

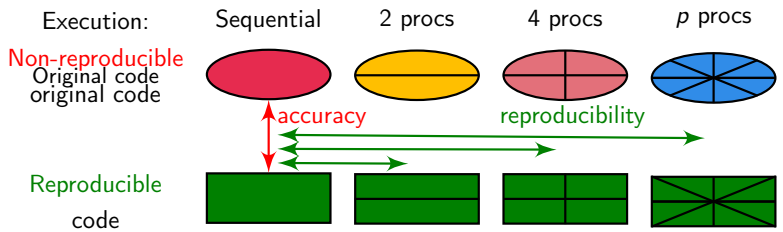
- Reproducible EBE matrix-vector product
- dot product
  - MPI reduction: a parallel compensated `dot2`
  - weights: $(1/k, 1/k, \ldots, 1/k) \rightarrow (1, 0, \ldots, 0)$
- Reproducible operations $\longrightarrow$ reproducible results
- Same errors for both sequential and parallel executions

Recovering reproducibility in a finite element resolution

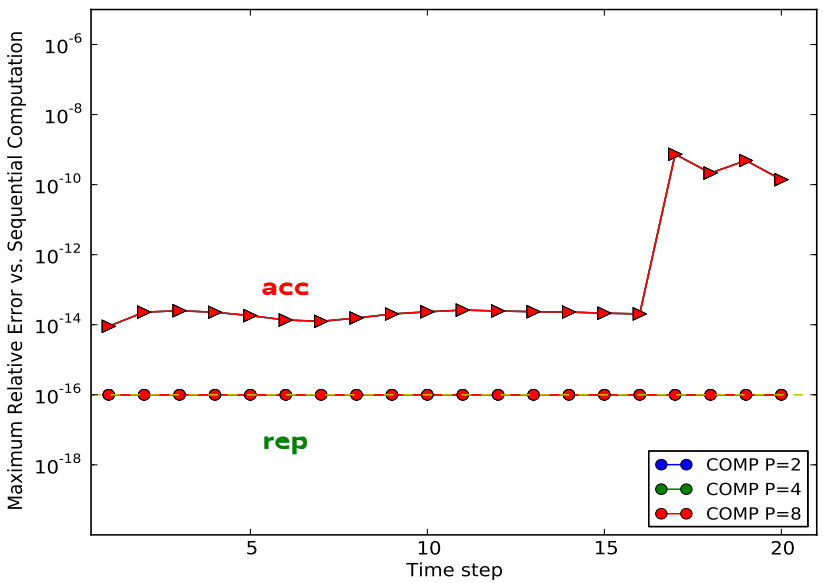
Reproducible Telemac2D

Reproducible Telemac2D





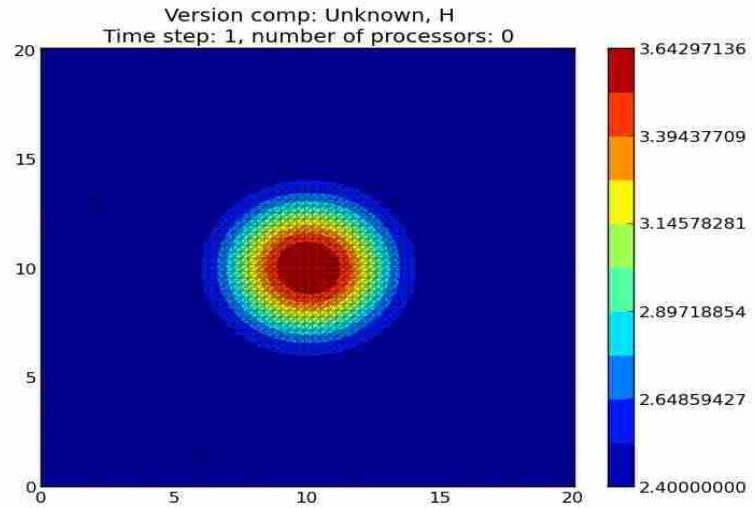
Maximum relative error
gouttedo test case



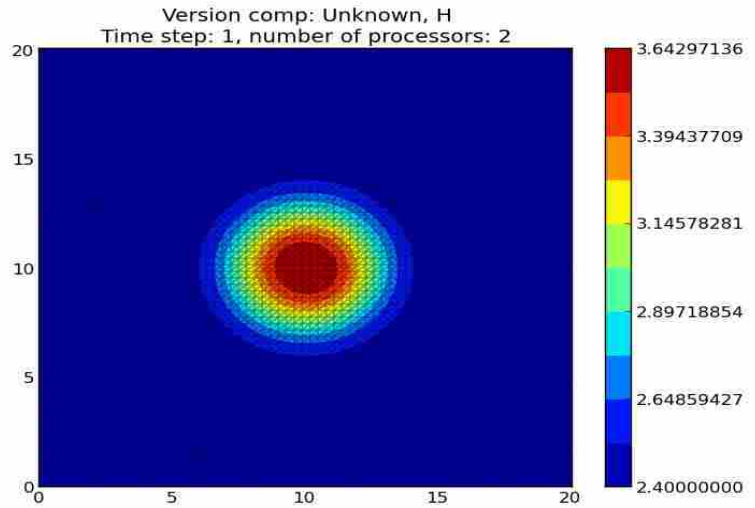
Reproducible gouttedo!

Time step 1

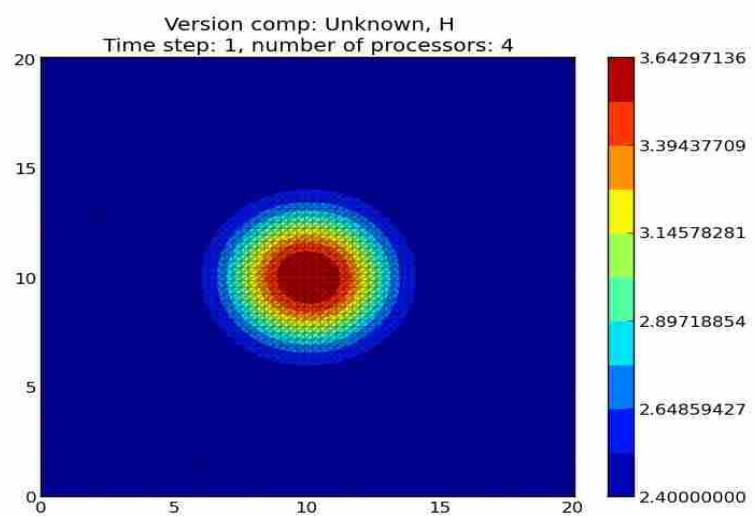
p=1



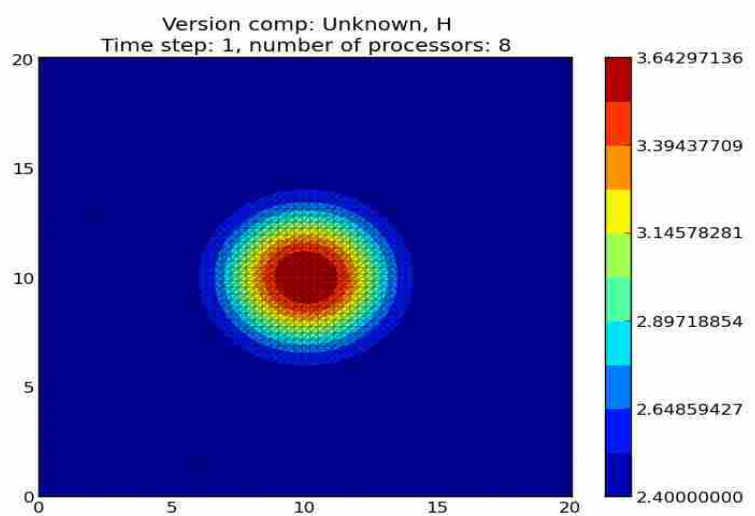
p=2



p=4



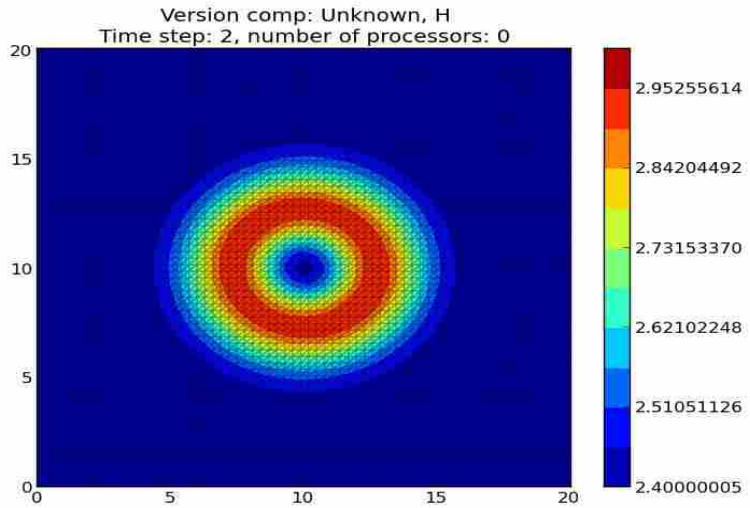
p=8



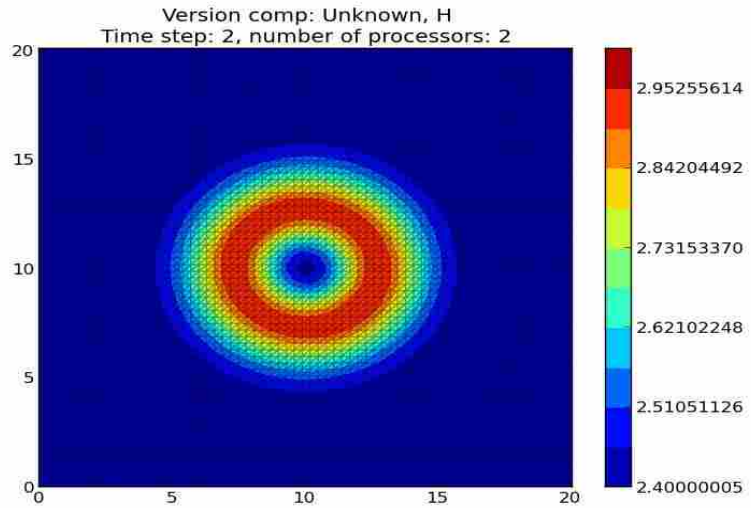
Reproducible gouttedo!

Time step 2

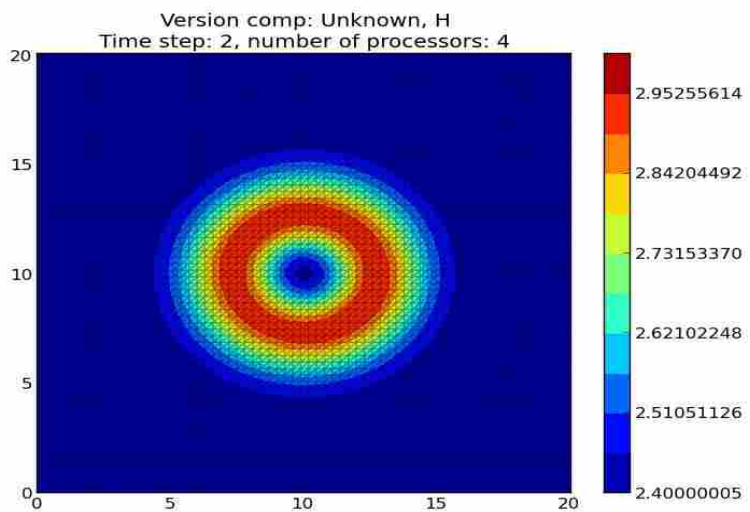
p=1



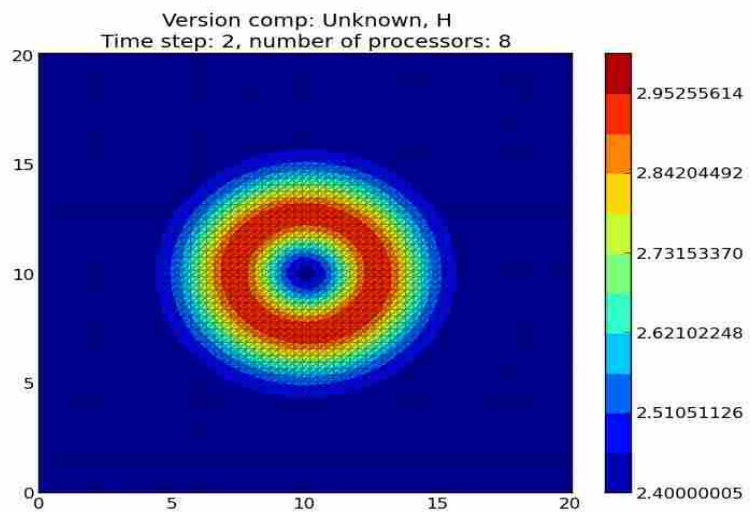
p=2



p=4



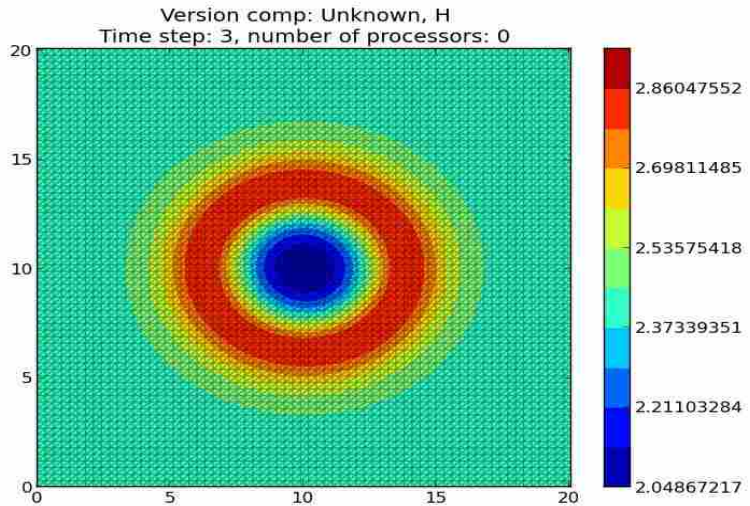
p=8



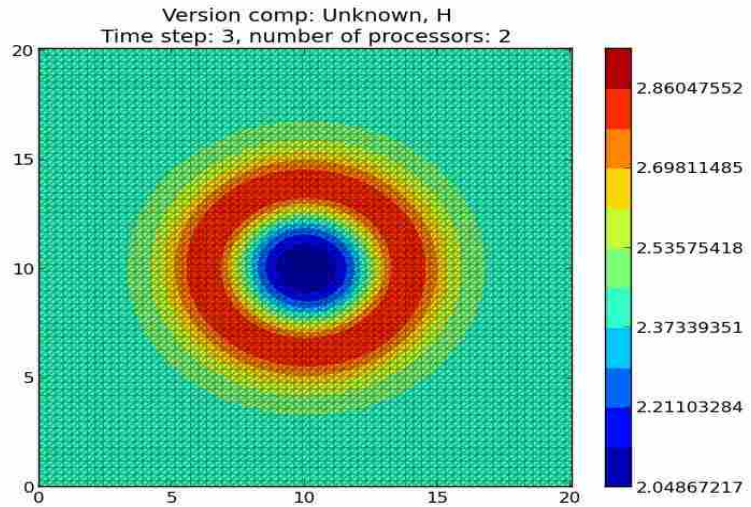
Reproducible gouttedo!

Time step 3

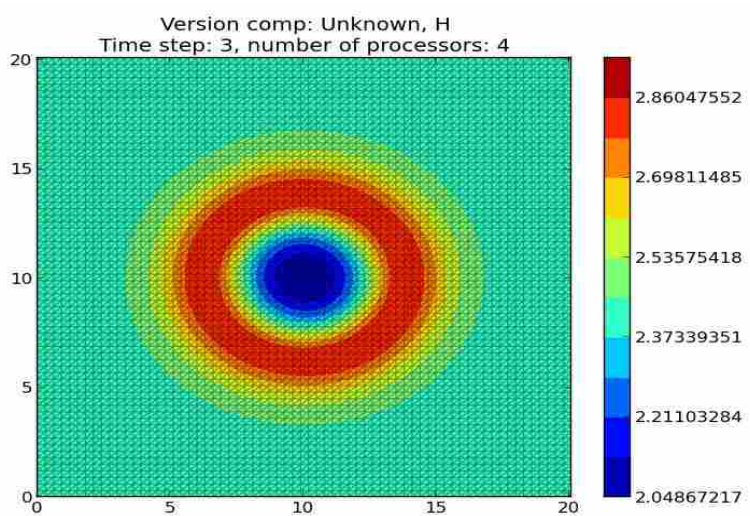
p=1



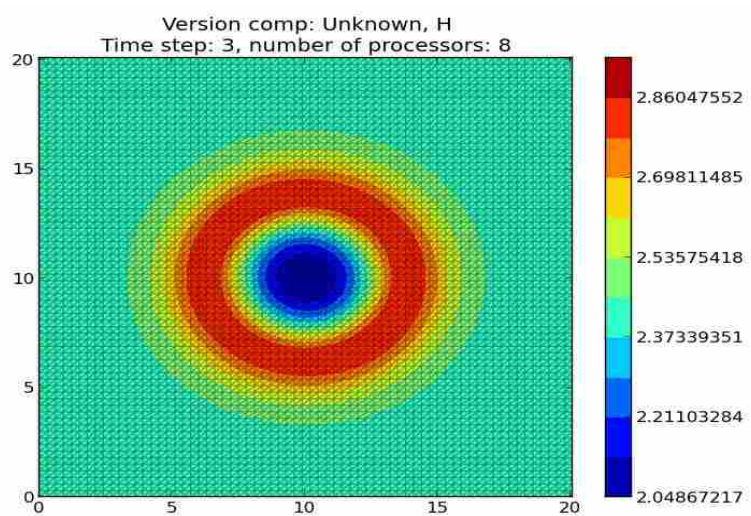
p=2



p=4



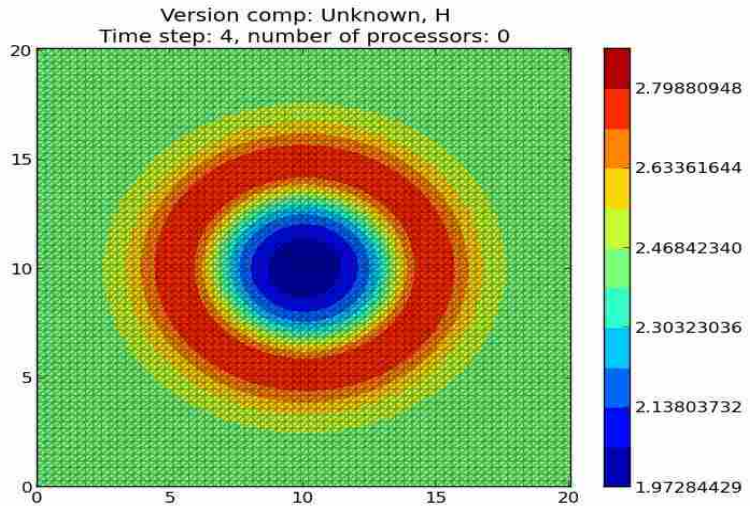
p=8



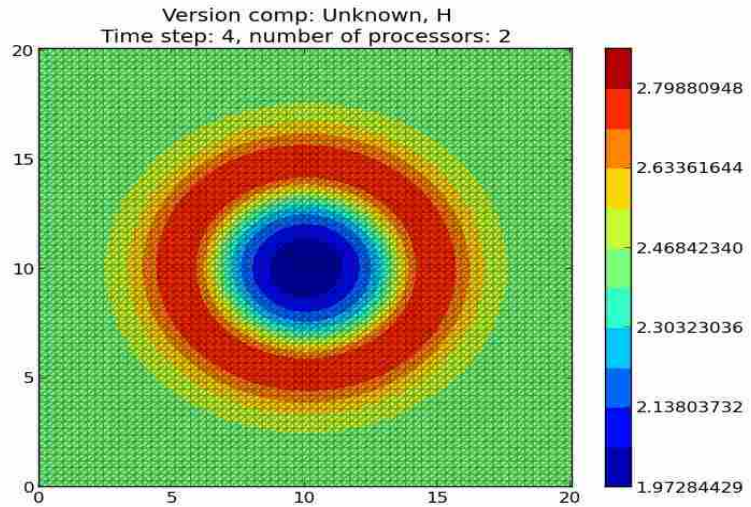
Reproducible gouttedo!

Time step 4

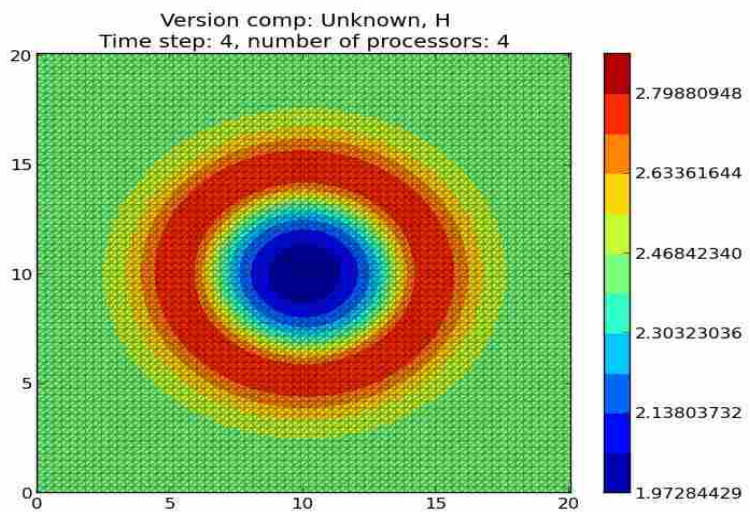
p=1



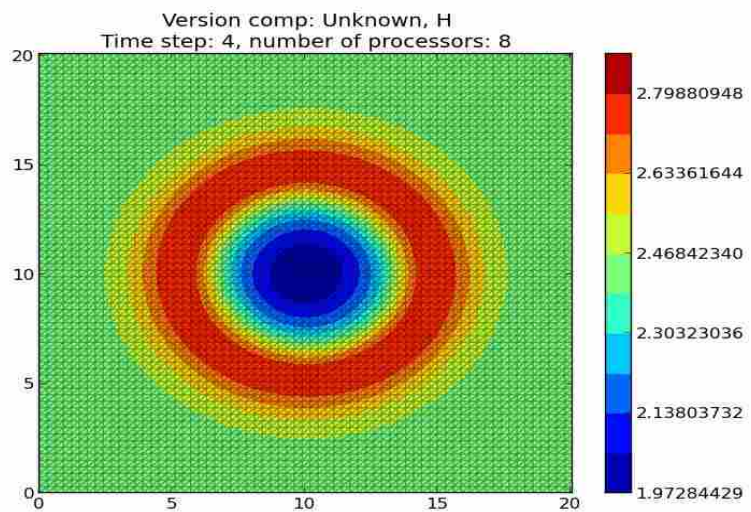
p=2



p=4



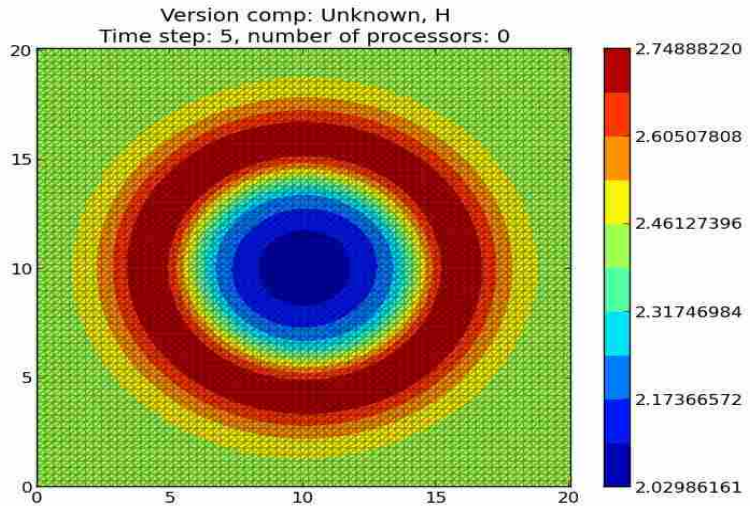
p=8



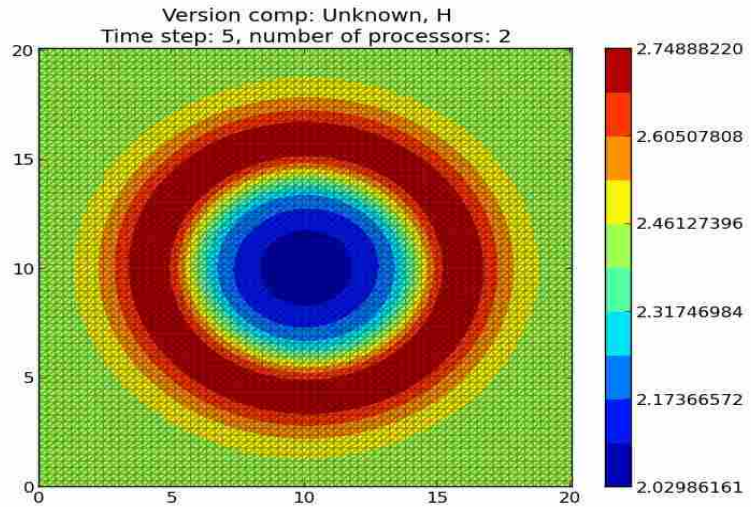
Reproducible gouttedo!

Time step 5

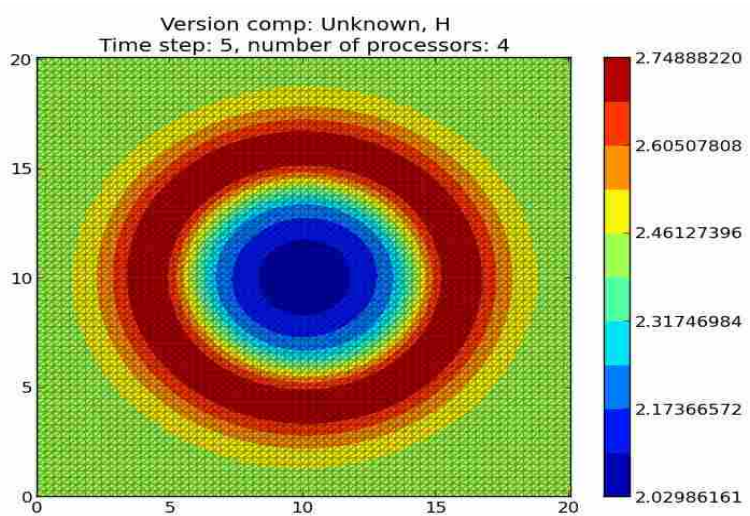
p=1



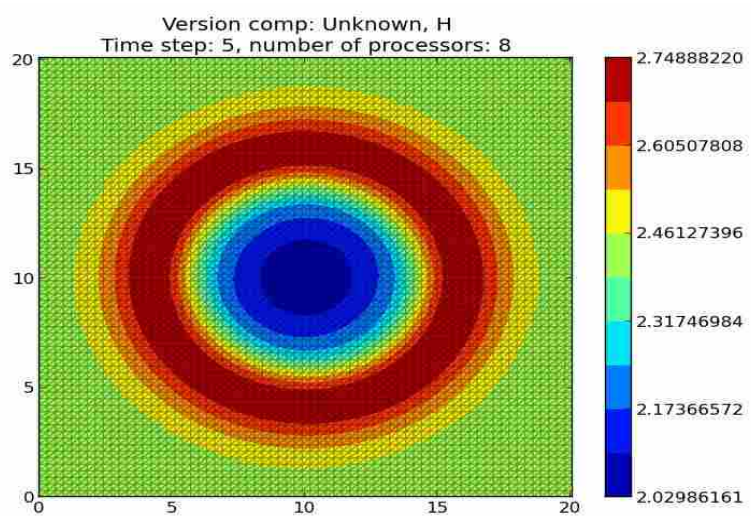
p=2



p=4



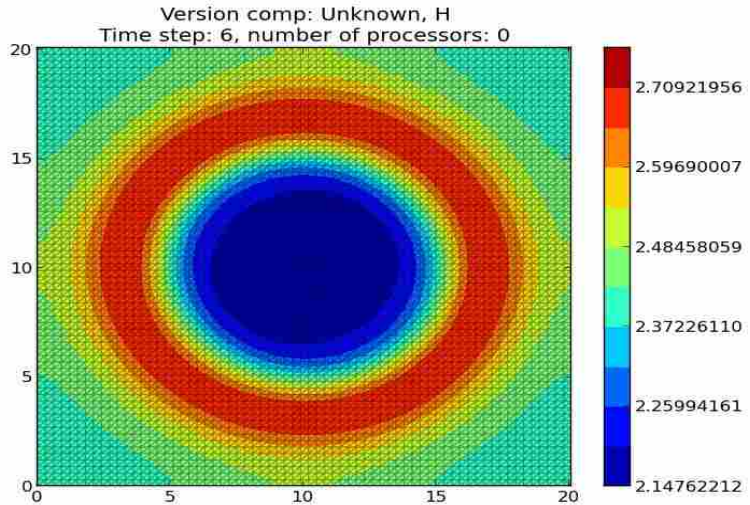
p=8



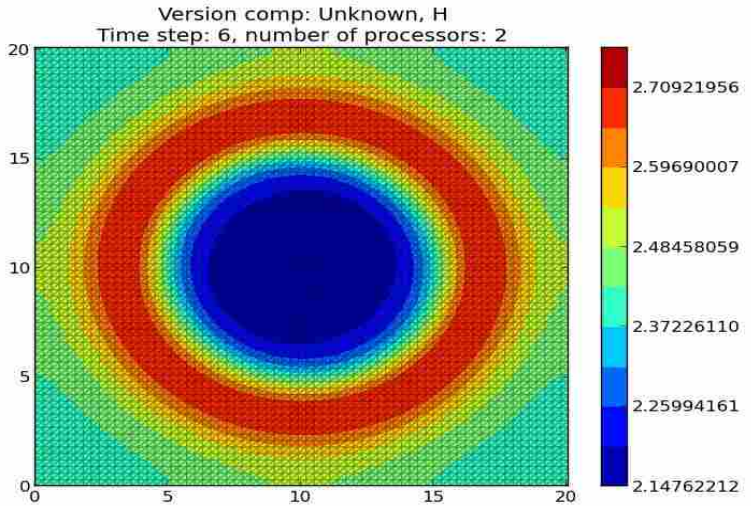
Reproducible gouttedo!

Time step 6

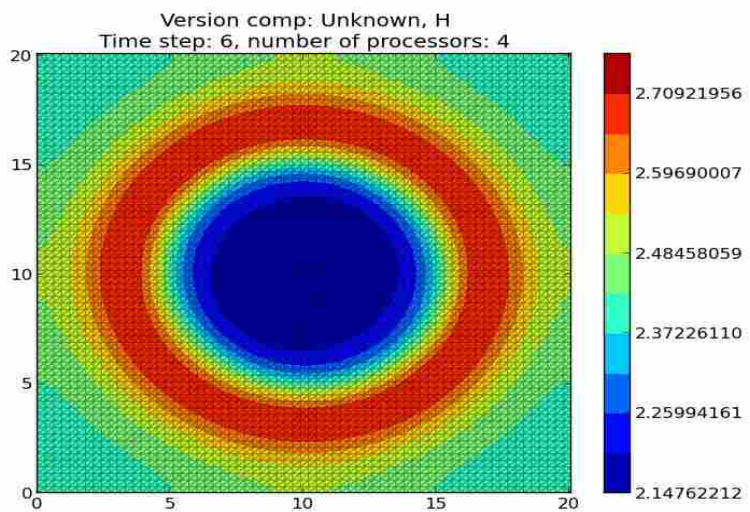
p=1



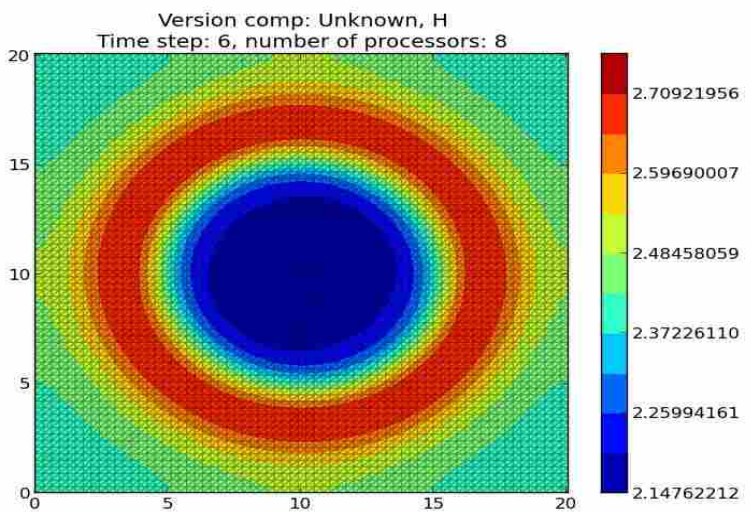
p=2



p=4



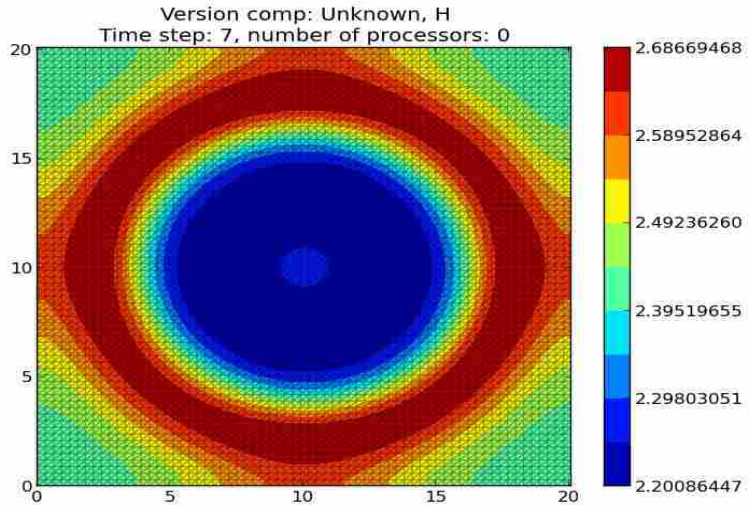
p=8



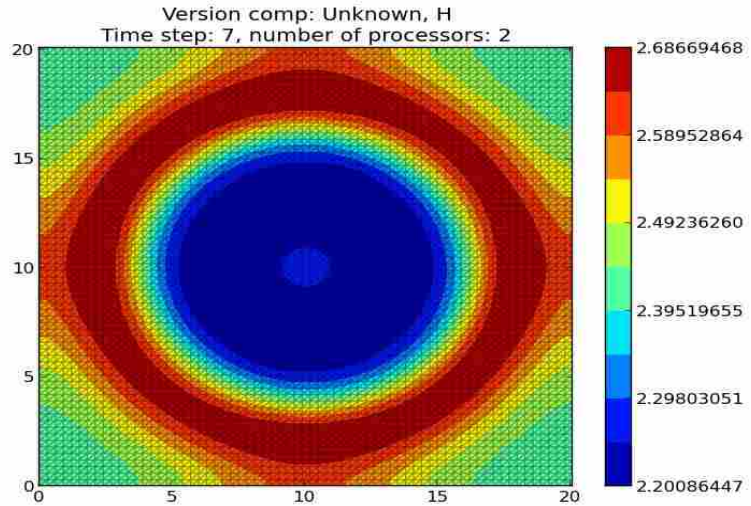
Reproducible gouttedo!

Time step 7

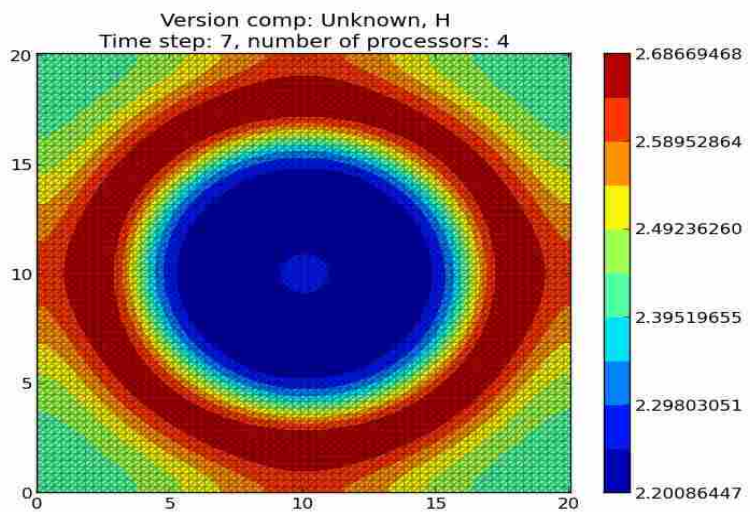
p=1



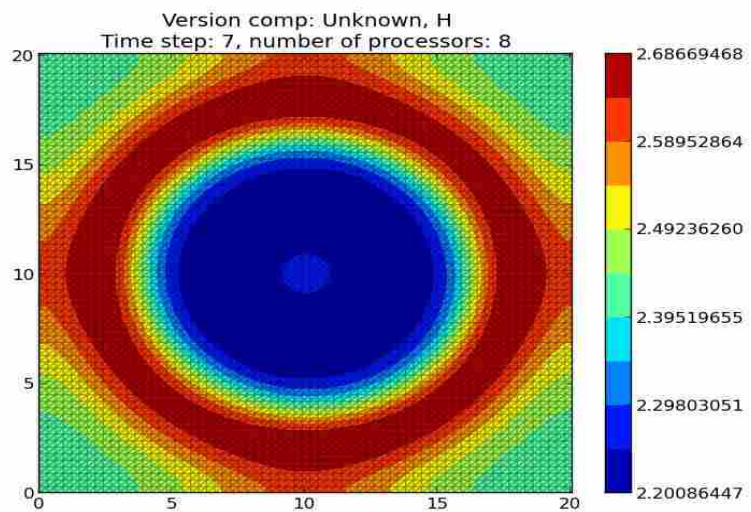
p=2



p=4



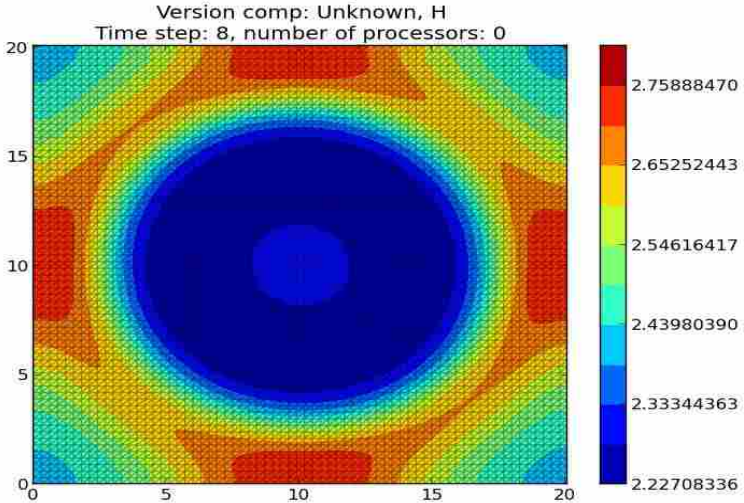
p=8



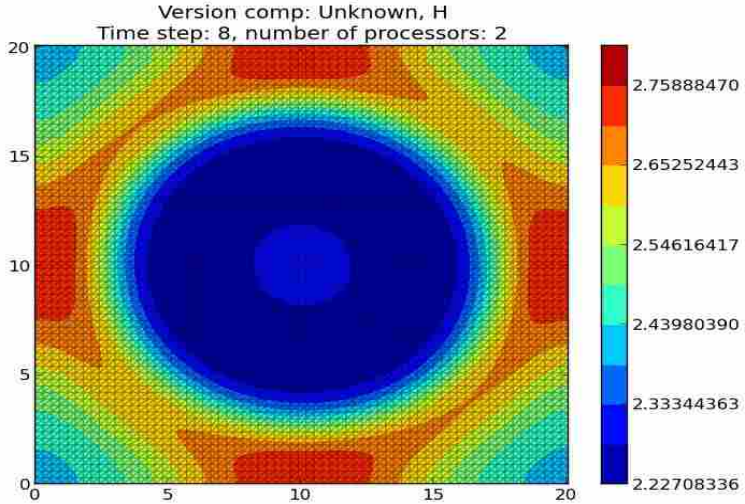
Reproducible gouttedo!

Time step 8

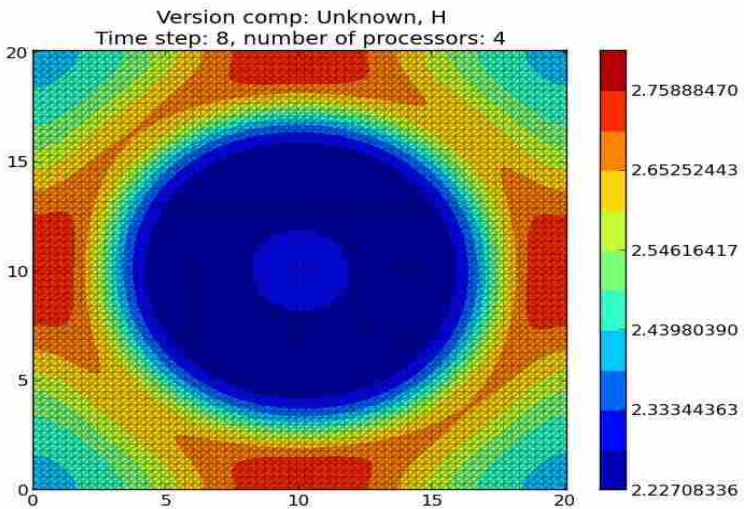
p=1



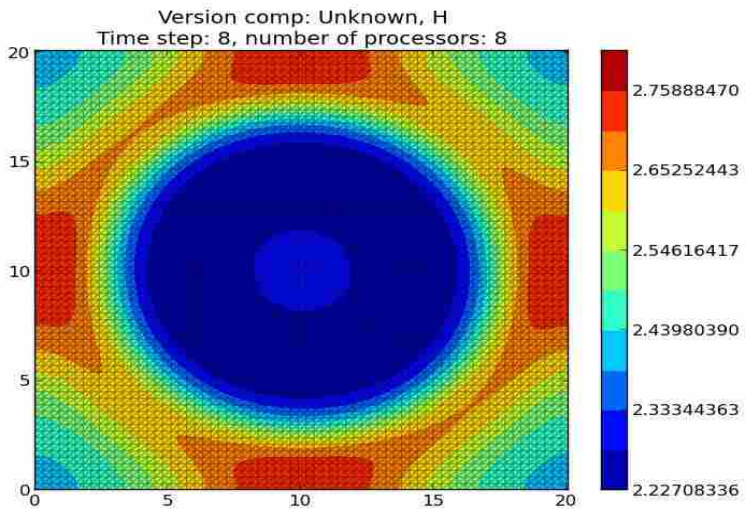
p=2



p=4



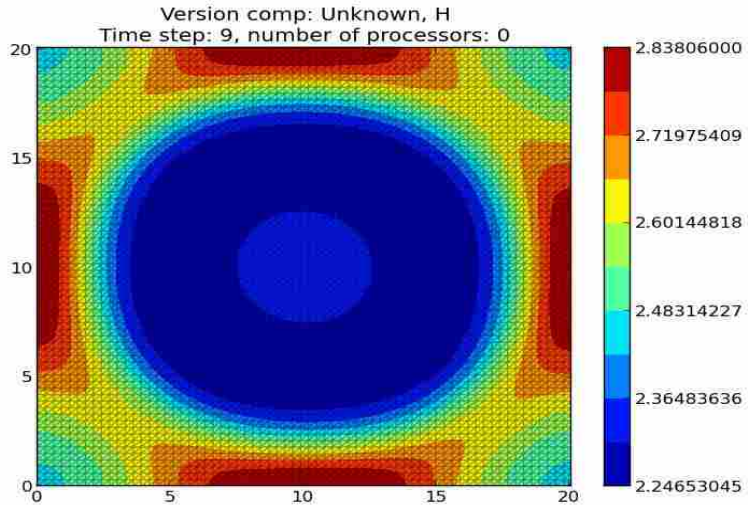
p=8



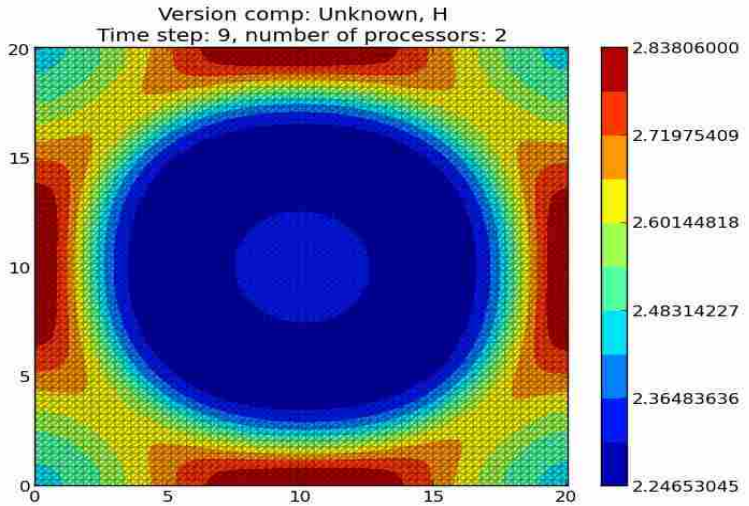
Reproducible gouttedo!

Time step 9

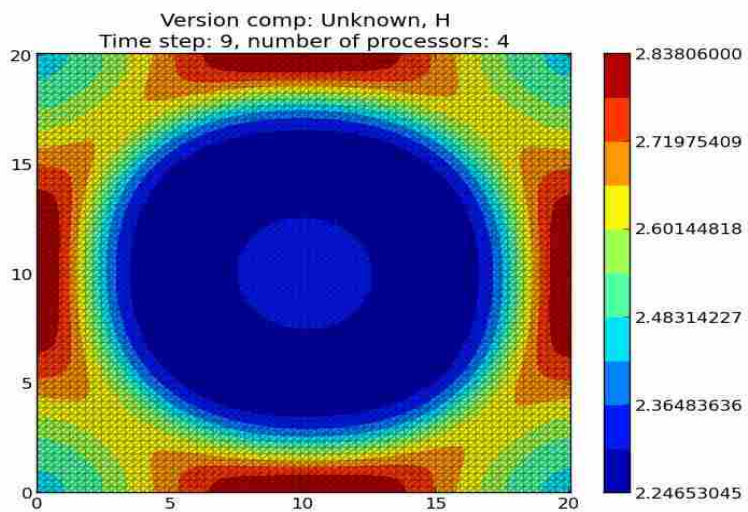
p=1



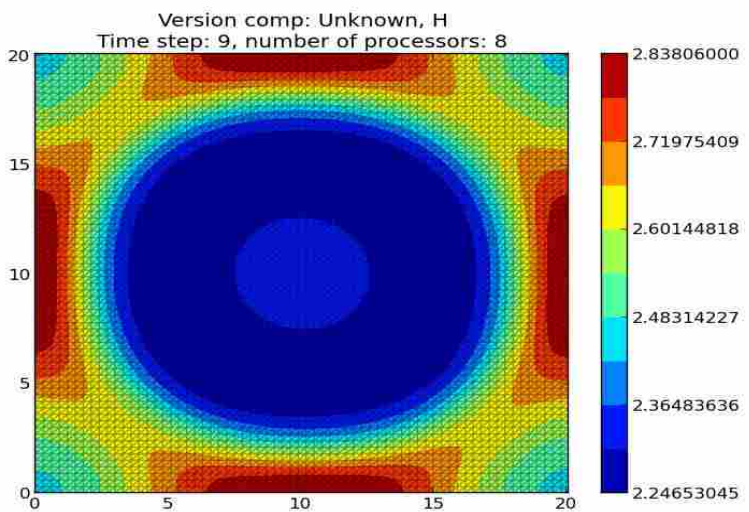
p=2



p=4



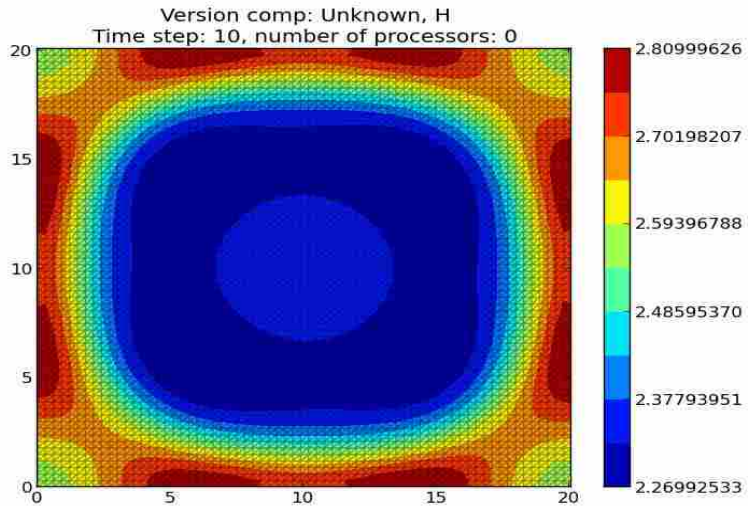
p=8



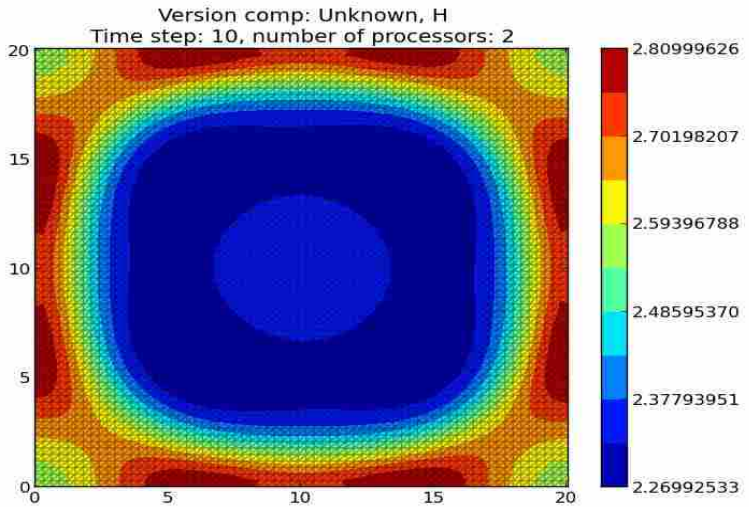
Reproducible gouttedo!

Time step 10

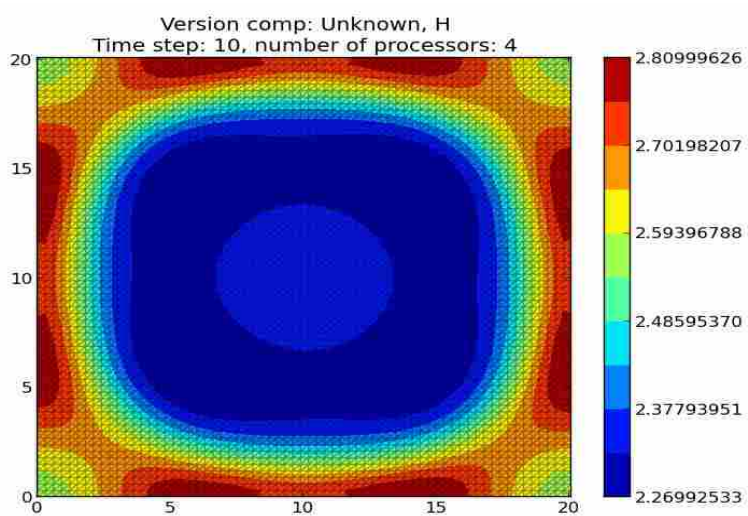
p=1



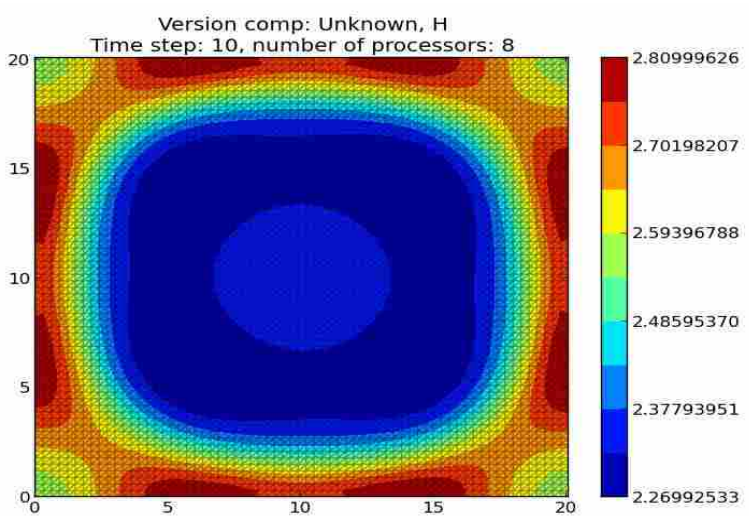
p=2



p=4



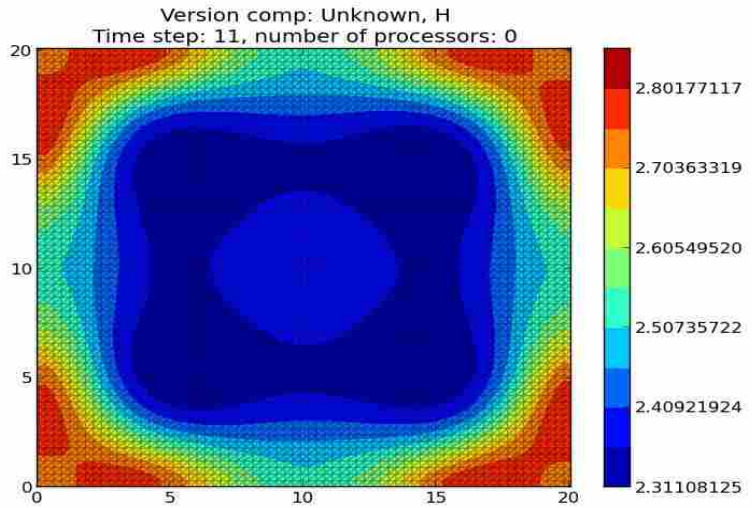
p=8



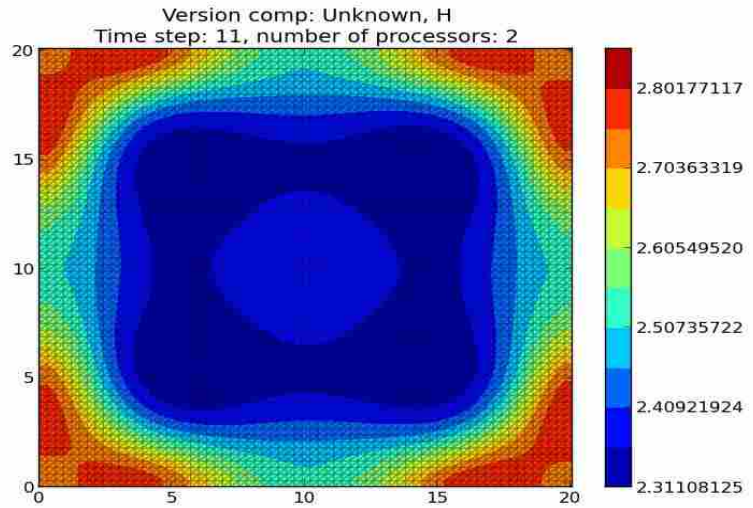
Reproducible gouttedo!

Time step 11

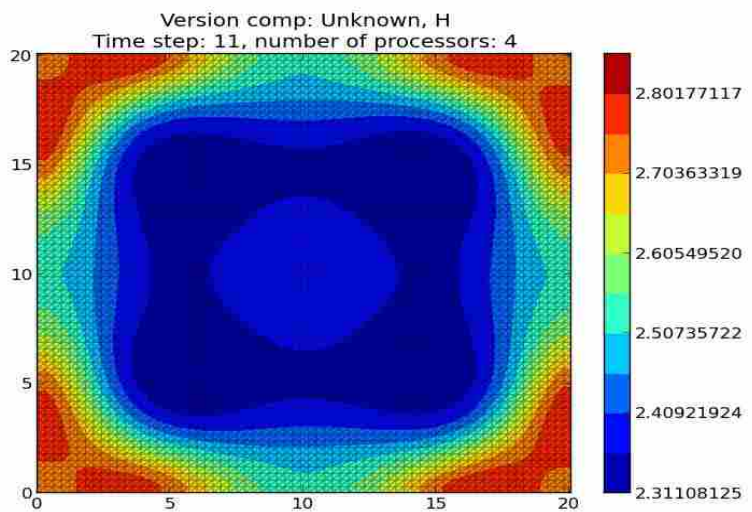
p=1



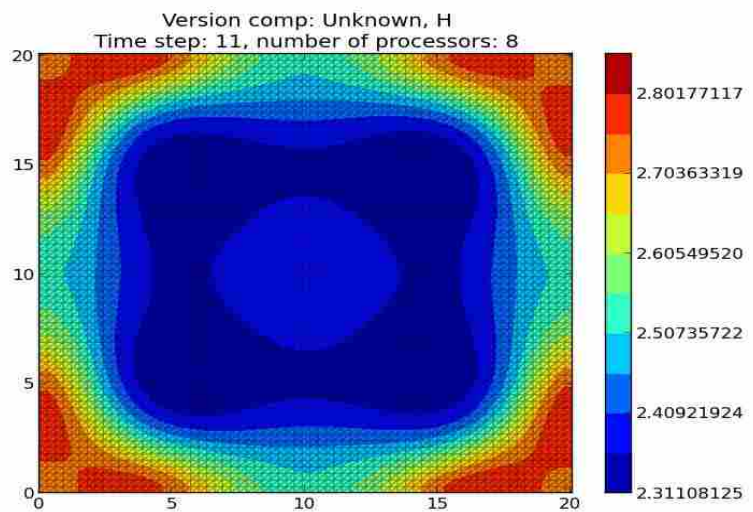
p=2



p=4



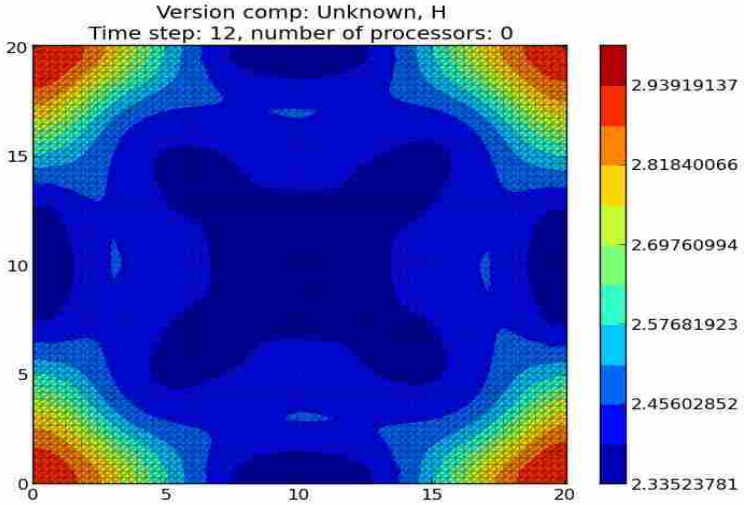
p=8



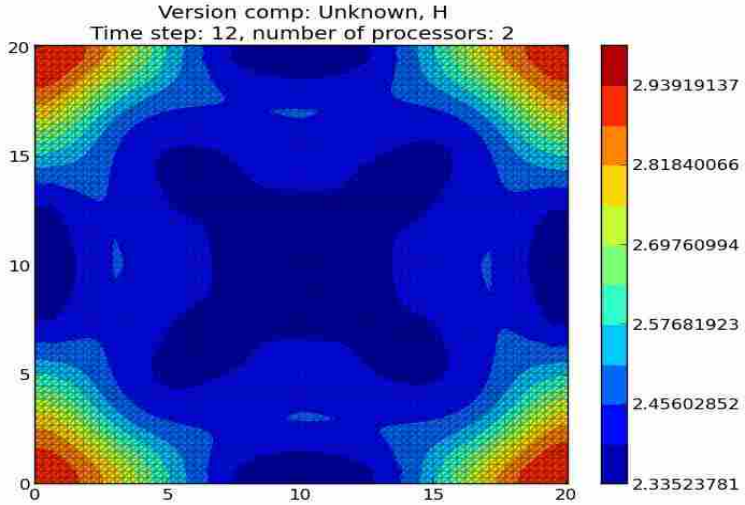
Reproducible gouttedo!

Time step 12

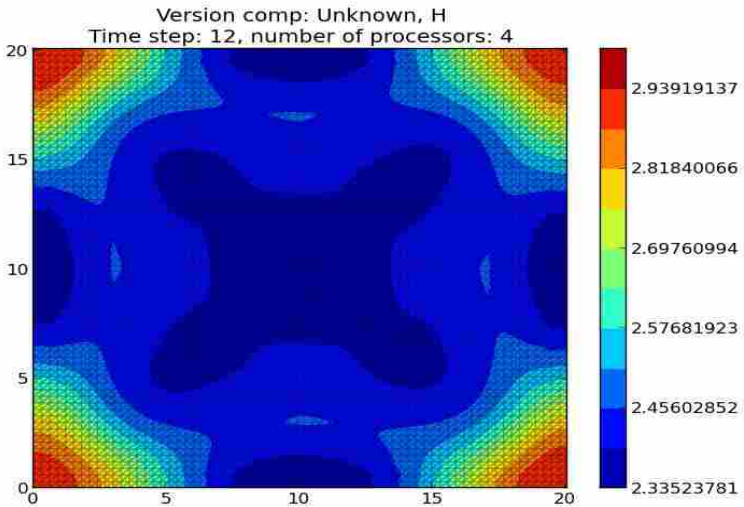
p=1



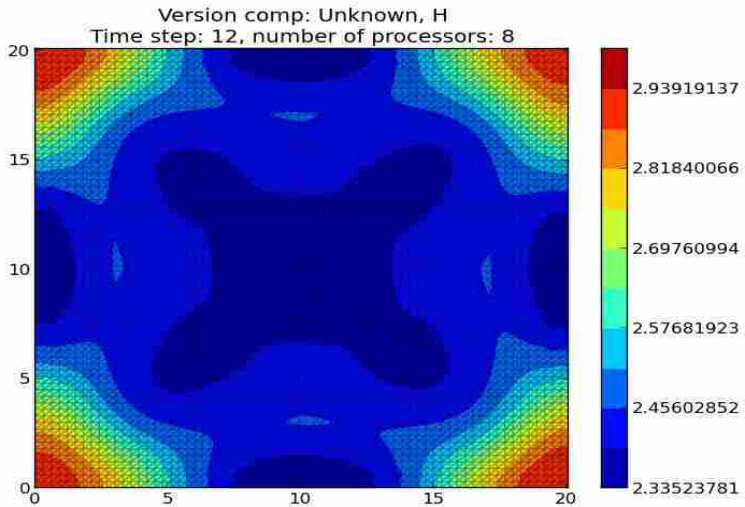
p=2



p=4



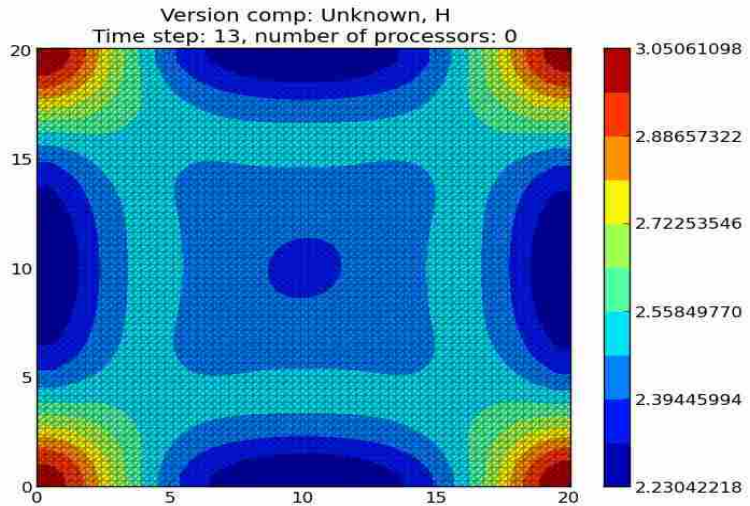
p=8



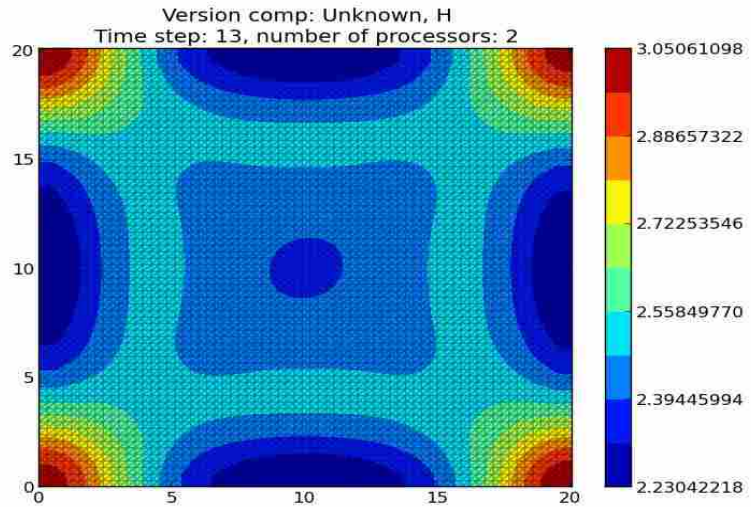
Reproducible gouttedo!

Time step 13

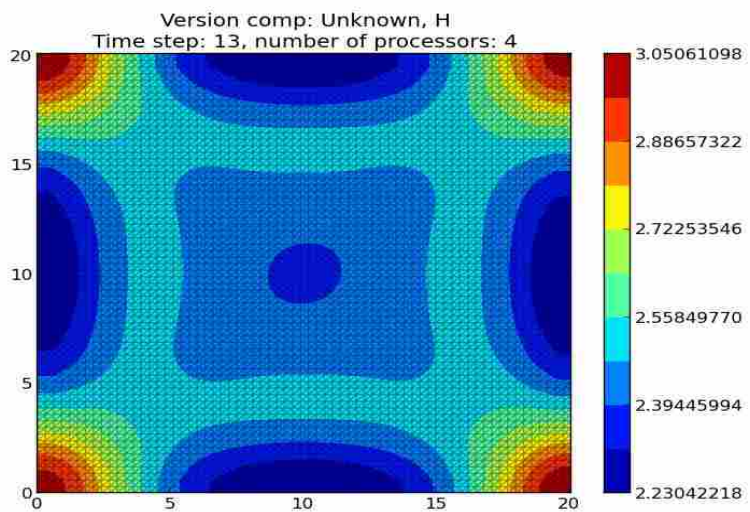
p=1



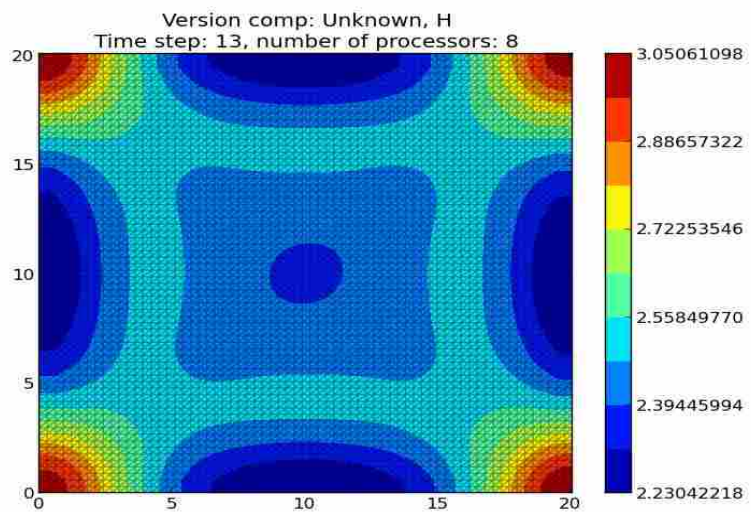
p=2



p=4



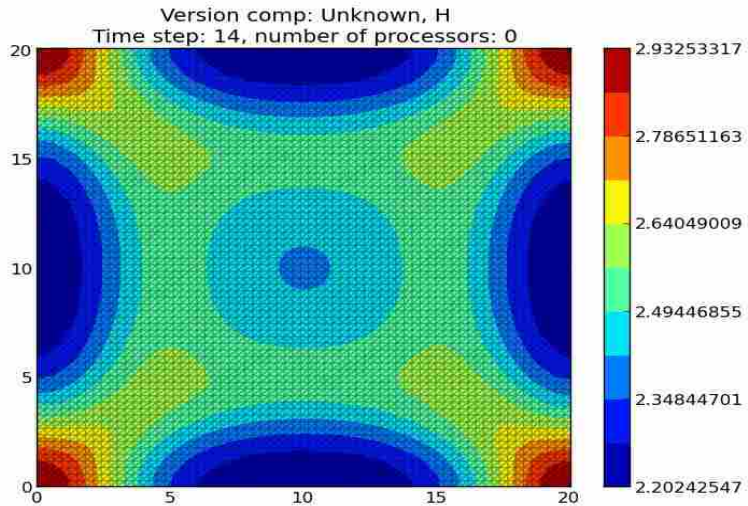
p=8



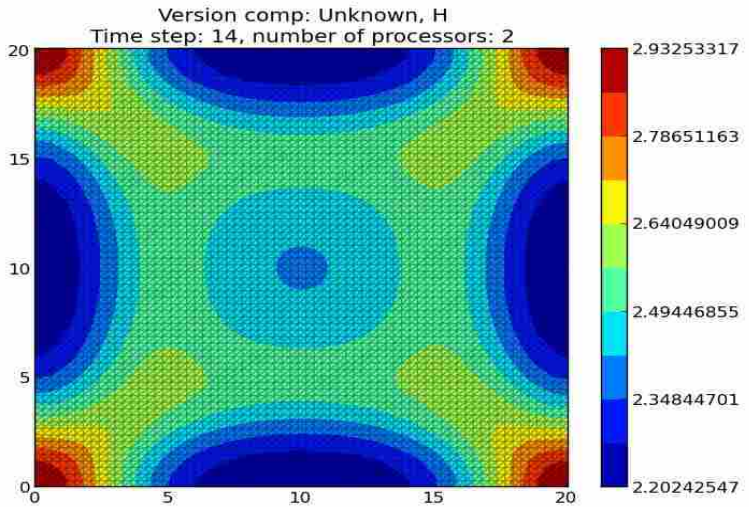
Reproducible gouttedo!

Time step 14

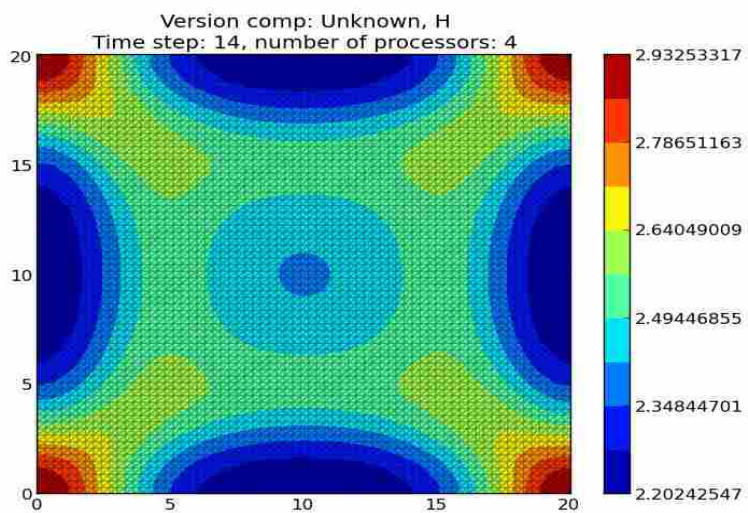
p=1



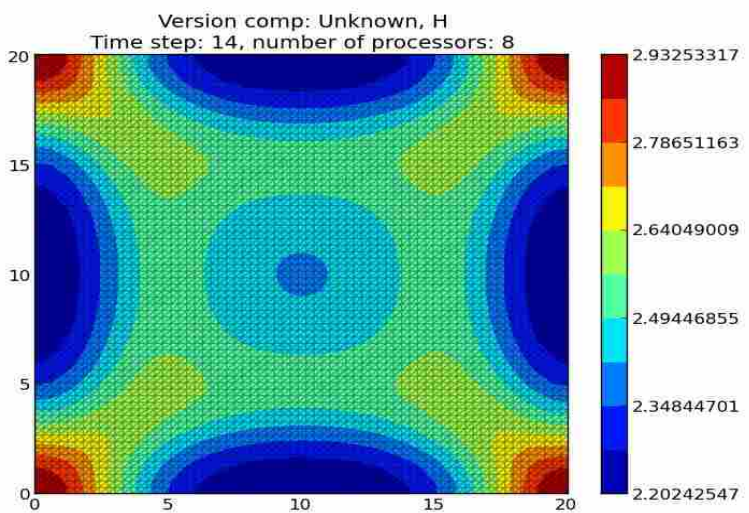
p=2



p=4



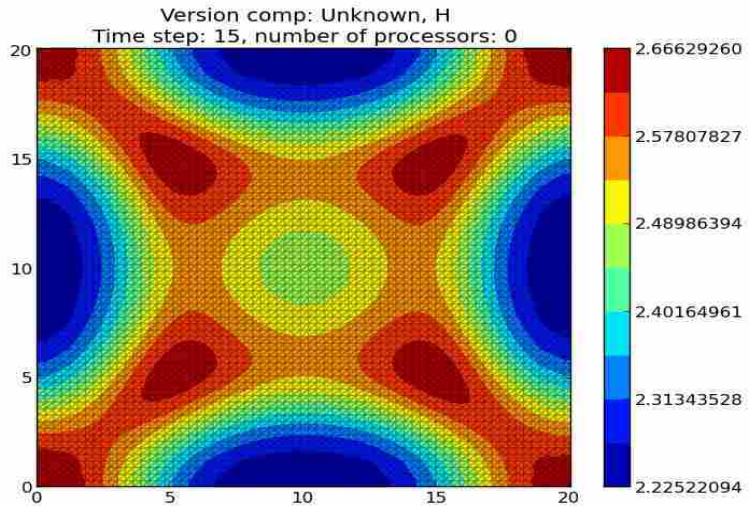
p=8



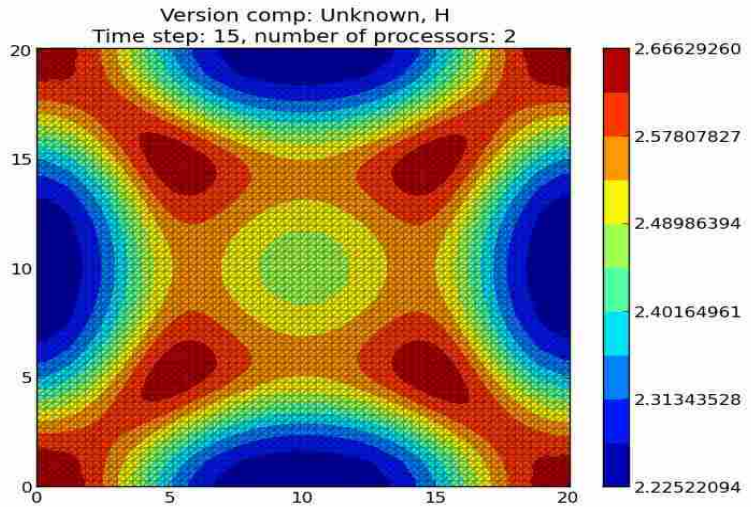
Reproducible gouttedo!

Time step 15

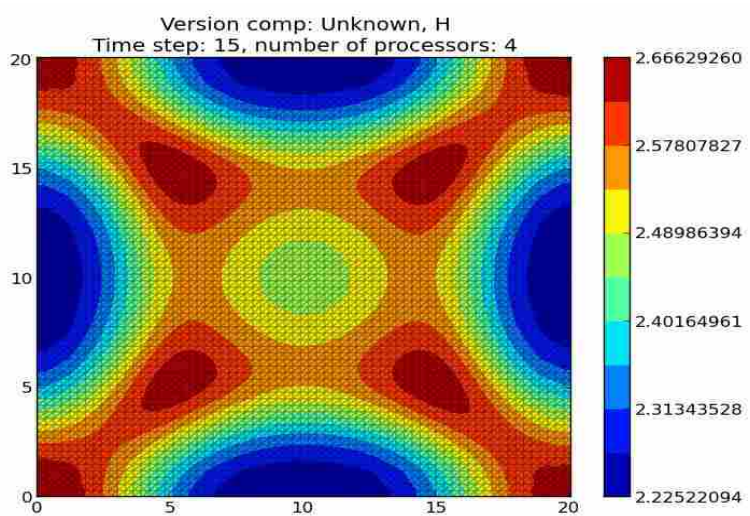
p=1



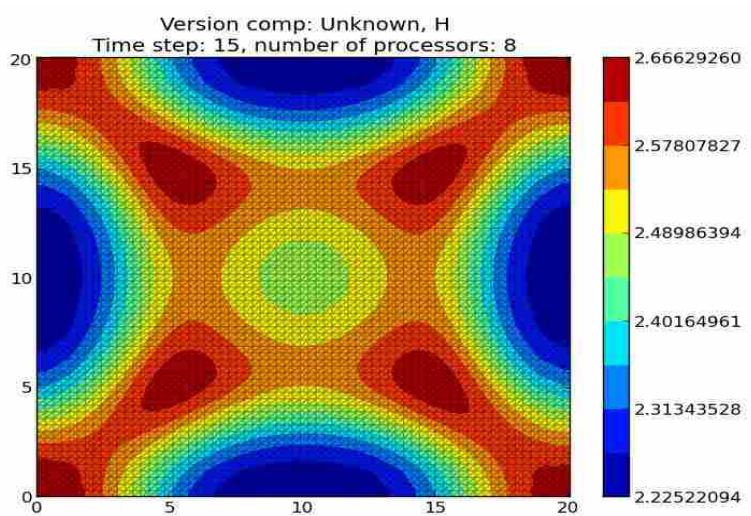
p=2



p=4



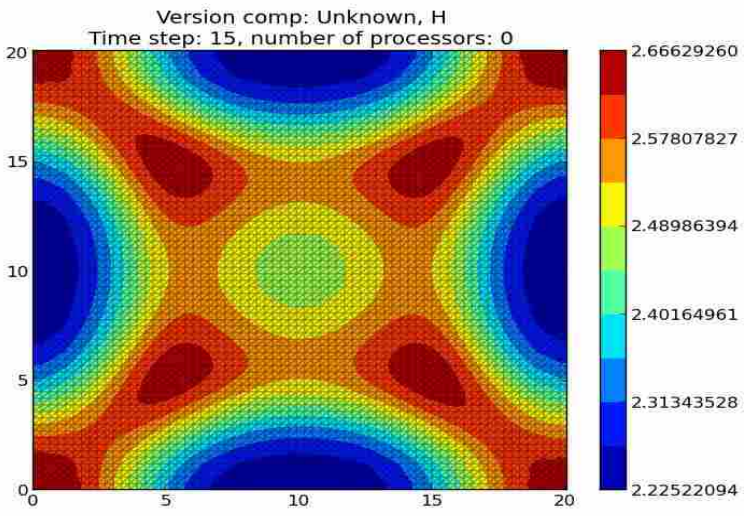
p=8



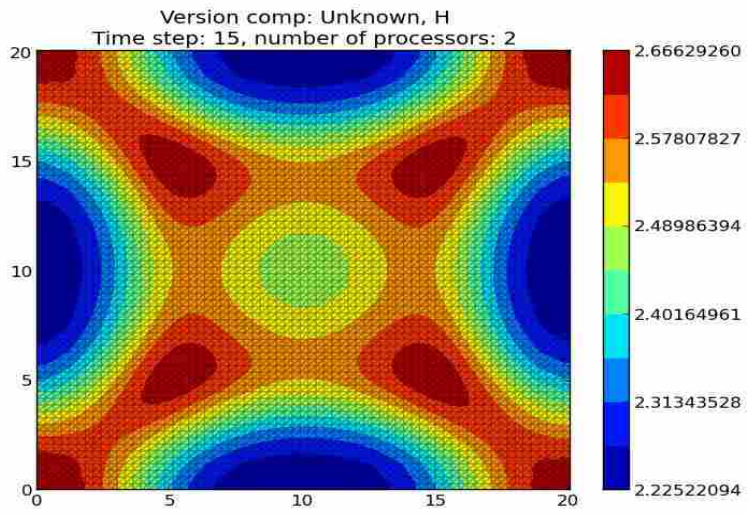
Reproducible gouttedo!

Time step 15

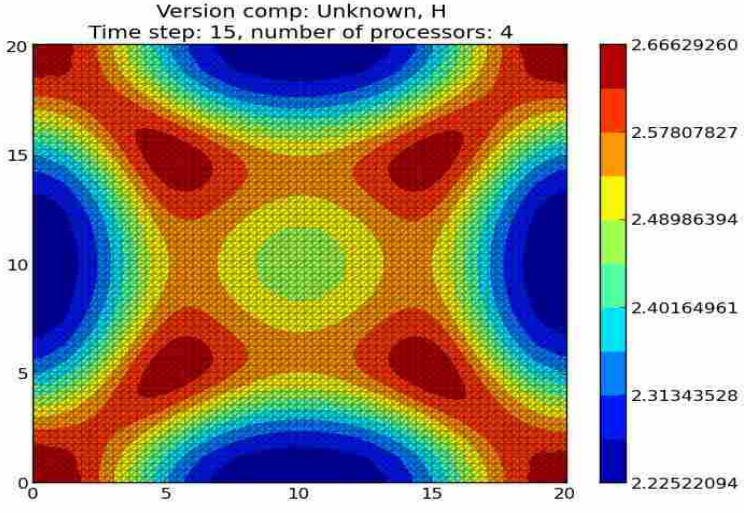
p=1



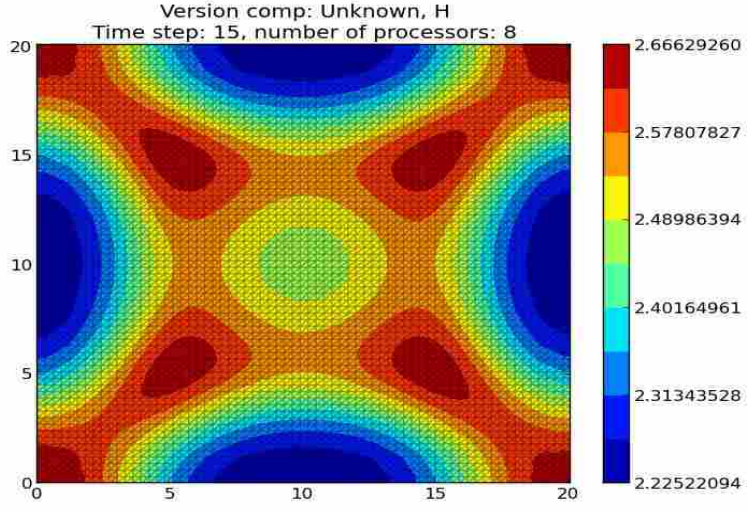
p=2



p=4



p=8



Efficiency

Runtime extra-cost for reproducible simulations

Measures, test cases and mesh sizes

- hardware cycle counter: `rdtsc`
- `gouttedo`
- mesh sizes: 4624, 18225, 72361 nodes (1, $\approx \times 4$, $\approx \times 16$)

	#nodes		
#proc.	4624	18225	72361
2	72	143	280
4	304	674	1368
8	501	1152	2020

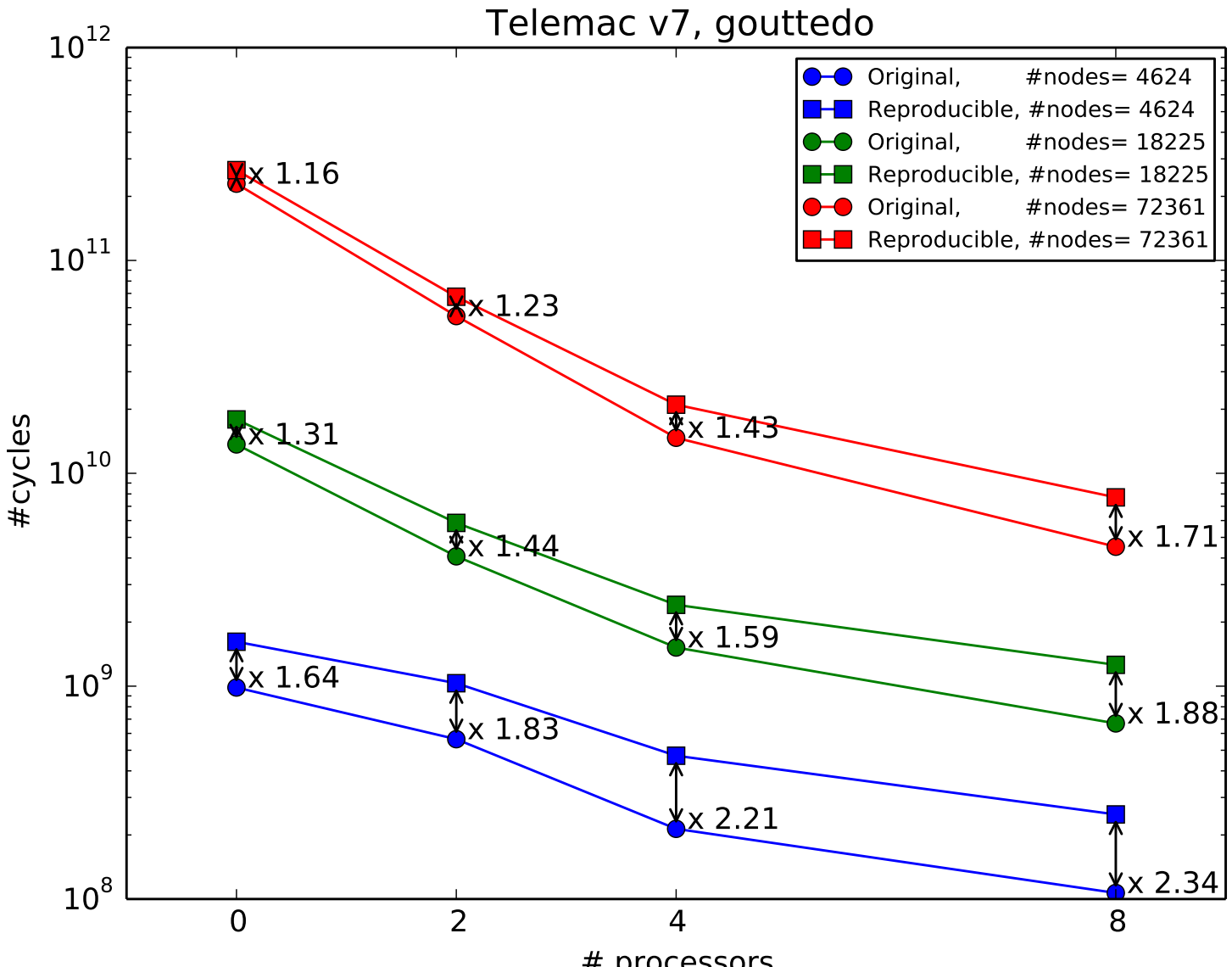
Number of IP

Hardware and software env.

- openTelemac v7.2
- socket: Intel Xeon E5-2660 2.20GHz (L3 cache = 20 M)
- 2 sockets of 8 cores each
- GNU Fortran 4.6.3, -O3
- OpenMPI 1.5.4
- Linux 3.5.0-54-generic

The core runtime extra-cost for reproducible gouttedo

gouttedo core: no input/output steps, just building+solving



Time to conclude

Conclusion

Recovering numerical reproducibility

- Industrial scale software: openTelemac-Mascaret
- Finite element simulation, domain decomposition, linear system solving
- 2 reproducible modules: Tomawac, Telemac2D
- Integration in the next openTelemac version: current work

Conclusion

Recovering numerical reproducibility

- Industrial scale software: openTelemac-Mascaret
- Finite element simulation, domain decomposition, linear system solving
- 2 reproducible modules: Tomawac, Telemac2D
- Integration in the next openTelemac version: current work

Feasibility

- How to recover reproducibility? Sources of non-reproducibility? Do existing techniques apply? how easily?
- Hand-made analysis of the computing workflow
- Compensation yields reproducibility here!
- Fits well to the openTelemac's vector library
- Other existing techniques also apply and more or less easily [11]

Conclusion

Efficiency

- How much to pay for reproducibility?
- $\times 1.2 \leftrightarrow \times 2.3$ extra-cost which decreases as the problem size increases
- OK to debug, to validate and even to simulate!

Conclusion

Efficiency

- How much to pay for reproducibility?
- $\times 1.2 \leftrightarrow \times 2.3$ extra-cost which decreases as the problem size increases
- OK to debug, to validate and even to simulate!

Reproducibility at a larger scale: the whole openTelemac software suite

- Does it still working for complex, large and real-life simulations?
- The two FE test cases are significant enough to validate the methodology
- Localization of the failure sources is difficult to automatize
- but the methodology application is easy for the software developpers

Other existing techniques

Existing techniques to recover numerical reproducibility in summation

- Accurate compensated summation [15]
- Demmel-Nguyen's reproducible sums [2]
- Integer convertion [16]

Part III

More performance

- Développé en TP.

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: May 22, 2017