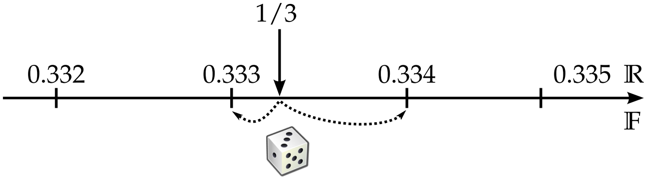
VERROU: New stochastic rounding modes for numerical verification

24/08/23
Workshop ICIAM 2023

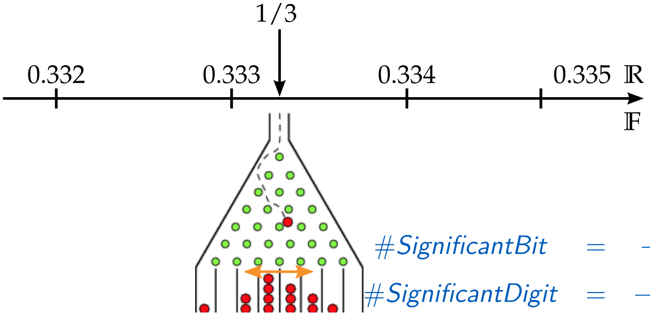
Bruno Lathuilière (EDF R&D)

Common work with:
Nestor Demeure
(Data and analytics services group, National Energy Research Scientific Computing Center, Berkeley).

Stochastic arithmetic for numerical verification



Stochastic arithmetic for numerical verification



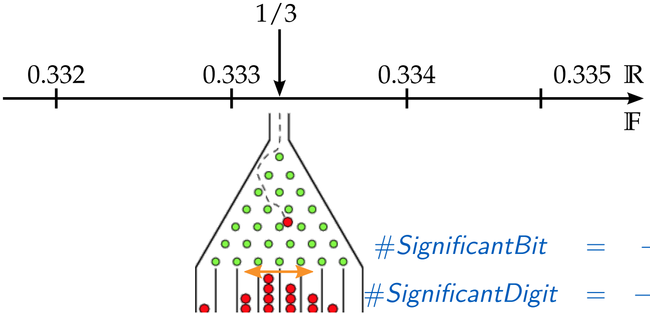
$$\#SignificantBit = -log2\left(\frac{max_i(|X_i - X_{nearest}|)}{|X_{nearest}|}\right)$$

$$\#SignificantDigit = -log10\left(\frac{max_i(|X_i - X_{nearest}|)}{|X_{nearest}|}\right)$$

Instruction	Eval. Nearest	Eval. 1	Eval. 2	Eval. 3
$a = 1/3$	0.333	$0.333_{\downarrow}$	$0.334^{\uparrow}$	$0.334^{\uparrow}$
$b = a \times 3$	0.999	0.999	$1.00_{\downarrow}$	$1.01^{\uparrow}$

$\#SignificantDigit \approx 1.95$

Stochastic arithmetic for numerical verification



$$\#SignificantBit = -log2\left(\frac{max_i(|X_i - X_{nearest}|)}{|X_{nearest}|}\right)$$

$$\#SignificantDigit = -log10\left(\frac{max_i(|X_i - X_{nearest}|)}{|X_{nearest}|}\right)$$

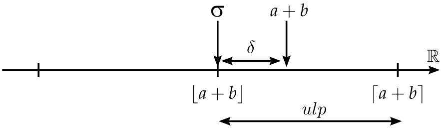
Instruction	Eval. Nearest	Eval. 1	Eval. 2	Eval. 3
$a = 1/3$	0.333	$0.333_{\downarrow}$	$0.334^{\uparrow}$	$0.334^{\uparrow}$
$b = a \times 3$	0.999	0.999	$1.00_{\downarrow}$	$1.01^{\uparrow}$

$\#SignificantDigit \approx 1.95$

- Easily compatible with binary instrumentation (**Verrou** based on valgrind) or low-level LLVM pass (Verificarlo)
- Few false positive detection (due to asynchronous approach) but ...

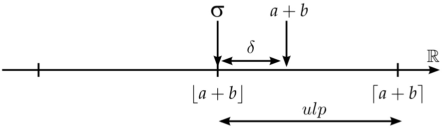
Verrou: implementation of stochastic rounding

- Error Free Transformation:
  - $a \circ b = \sigma + \delta$,
  - $\sigma = fl(a \circ b)$



Verrou: implementation of stochastic rounding

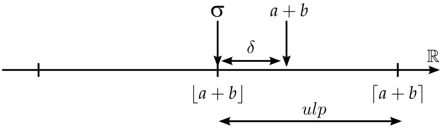
- Error Free Transformation:
  - $a \circ b = \sigma + \delta$,
  - $\sigma = fl(a \circ b)$



- If $\delta < 0$:
  $\lfloor a \circ b \rfloor = fl(a \circ b) - ulp$,
  $\lceil a \circ b \rceil = fl(a \circ b)$.
- If $\delta = 0$:
  $\lfloor a \circ b \rfloor = fl(a \circ b)$
  $\lceil a \circ b \rceil = fl(a \circ b)$.
- If $\delta > 0$:
  $\lfloor a \circ b \rfloor = fl(a \circ b)$,
  $\lceil a \circ b \rceil = fl(a \circ b) + ulp$.

Verrou: implementation of stochastic rounding

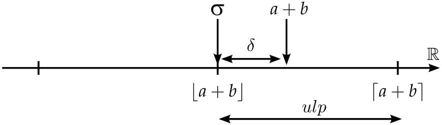
- Error Free Transformation:
  - $a \circ b = \sigma + \delta$,
  - $\sigma = fl(a \circ b)$



- If $\delta < 0$:
  $\lfloor a \circ b \rfloor = fl(a \circ b) - ulp$,
  $\lceil a \circ b \rceil = fl(a \circ b)$.
- If $\delta = 0$:
  $\lfloor a \circ b \rfloor = fl(a \circ b)$
  $\lceil a \circ b \rceil = fl(a \circ b)$.
- If $\delta > 0$:
  $\lfloor a \circ b \rfloor = fl(a \circ b)$,
  $\lceil a \circ b \rceil = fl(a \circ b) + ulp$.

Verrou: implementation of stochastic rounding

- Error Free Transformation:
  - $a \circ b = \sigma + \delta$,
  - $\sigma = fl(a \circ b)$



- If $\delta < 0$:
  $\lfloor a \circ b \rfloor = fl(a \circ b) - ulp$,

  $\lceil a \circ b \rceil = fl(a \circ b)$.

- If $\delta = 0$:
  $\lfloor a \circ b \rfloor = fl(a \circ b)$

  $\lceil a \circ b \rceil = fl(a \circ b)$.

- If $\delta > 0$:
  $\lfloor a \circ b \rfloor = fl(a \circ b)$,

  $\lceil a \circ b \rceil = fl(a \circ b) + ulp$.

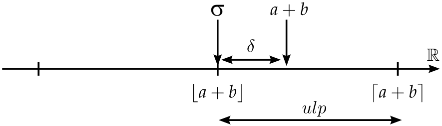
`random` **mode**:

$$fl_{random}(a \circ b) = \left| \begin{array}{ll} \lfloor a \circ b \rfloor & with\, p = 1/2 \\ \lceil a \circ b \rceil & with\, p = 1/2 \end{array} \right.$$

Pseudo random number generator in $\{0, 1\}$: (tinyMT or xoshiro256plus)+ bit shift.

Verrou: implementation of stochastic rounding

- Error Free Transformation:
  - $a \circ b = \sigma + \delta$,
  - $\sigma = fl(a \circ b)$



- If $\delta < 0$:
  $\lfloor a \circ b \rfloor = fl(a \circ b) - ulp$,
  $\lceil a \circ b \rceil = fl(a \circ b)$.
- If $\delta = 0$:
  $\lfloor a \circ b \rfloor = fl(a \circ b)$
  $\lceil a \circ b \rceil = fl(a \circ b)$.
- **If $\delta > 0$:**
  $\lfloor a \circ b \rfloor = fl(a \circ b)$,
  $\lceil a \circ b \rceil = fl(a \circ b) + ulp$.

average mode:

$$fl_{average}(a \circ b) = \left| \begin{array}{ll} \lfloor a \circ b \rfloor & \text{with } p = \frac{1-\delta}{|ulp|} \\ \lceil a \circ b \rceil & \text{with } p = \frac{\delta}{|ulp|} \end{array} \right.$$

Pseudo random number generator in $\mathbb{F} \cap [0, 1]$: tinyMT or xoroshiro128plus.
Equivalent to MCA RR to machine precision: often called `SR_nearness`.

False positive with stochastic rounding

The application developer may use properties fulfilled with nearest rounding mode (intentionally or not) which are no more fulfilled by stochastic rounding.

```
double a1=foo(42.);
double a2=foo(42.);
assert(a1==a2);
```

➧ **assert fail**

```
float x = foo (42);
if(x>0) return sqrt(foo(42));
else  return sqrt(-foo(42));
```

➧ **NaN**

In numerical verification context, it is not always possible/suitable to rewrite code to be compliant with stochastic rounding.

random_det and average_det

Idea: insure the determinism inside one verrou execution to the floating point instruction level.

Implementation: replace the PRnG by a **hash function** of the following parameters:

- `arg1, [arg2, [arg3]]`: the arguments of the operation.
- `verrou_seed`: the 64 bit seed (different for each verrou execution).
- `Op`: enum operator ($\oplus$,$\ominus$,$\otimes$,$\oslash$, *fma*, *cos*, *sin* ...).

For `random_det` (resp `average_det`) the image space is $\{0, 1\}$ (resp $\mathbb{F} \cap [0, 1]$)

Depending on hash function, we can prefer the following parameters:

- `arg1, [arg2, [arg3]]`
- `verrou_seed ^ Op`

The results in this presentation are based on XXH3 hash (`https://github.com/Cyan4973/xxHash`) function with `verrou_seed ^ Op` trick.

Commutative determinist stochastic rounding

Problem:

```
assert(dot(x,y)==dot(y,x))
```

Solution: Introduction of `[random,average]_comdet` which guarantee $x\,op\,y$ is rounded as $y\,op\,x$ if op is commutative.

Implementation:

- We replace:
  - $hash(arg1, arg2, op)$ by $hash(min(arg1, arg2), max(arg1, arg2), op)$ if op is in $(\oplus, \otimes)$.
  - $hash(arg1, arg2, arg3, FmaEnum)$ par $hash(min(arg1, arg2), max(arg1, arg2), arg3, FmaEnum)$.

The commutative determinist stochastic rounding

Signed variant

Problem:

```
assert(dot(x,y)==dot(-x,-y))
```

Solution: Introduction of `[random,average]_scomdet` which guarantee the commutativity and the following sign properties:
$\forall a, b, c \in \mathbb{F}^3$

- $a \oplus b = -(-a \oplus (-b))$
- $a \oplus (-b) = -(-a \oplus b)$
- $a \ominus b = a \oplus (-b)$
- $a \otimes b = (-a) \otimes (-b)$
- $a \otimes b = -(a \otimes (-b))$
- $a \oslash b = (-a) \oslash (-b)$
- $a \oslash b = -a \oslash (-b)$
- $a \oslash b = -(-a \oslash (b))$

- $fma(a, b, 0) = a \otimes b$
- $fma(a, b, c) = fma(-a, -b, c)$
- $fma(-a, b, -c) = -fma(a, b, c)$
- $fma(a, -b, -c) = -fma(a, b, c)$
- $cast(a) = -cast(-a)$

List of hash constraints

Let $x, y, z, a, b \in \mathbb{F}^5$ *with* $a > 0, b > 0$

OP		$\oplus$	$\ominus$	$\otimes$	$\oslash$	fma		
x	y					z>0	z=0	z<0
a	b	$r_\oplus$	$r_\ominus$	$r_\otimes$	$r^1_\oslash$	r^1_{fma}	$r_\otimes$	$\overline{r^2_{fma}}$
b	a	$r_\oplus$	$\overline{r_\ominus}$	$r_\otimes$	$r^2_\oslash$	r^1_{fma}	$r_\otimes$	$\overline{r^2_{fma}}$
-a	-b	$\overline{r_\oplus}$	$\overline{r_\ominus}$	$r_\otimes$	$r^1_\oslash$	r^1_{fma}	$r_\otimes$	$\overline{r^2_{fma}}$
-b	-a	$\overline{r_\oplus}$	$r_\ominus$	$r_\otimes$	$r^2_\oslash$	r^1_{fma}	$r_\otimes$	$\overline{r^2_{fma}}$
a	-b	$r_\ominus$	$r_\oplus$	$\overline{r_\otimes}$	$\overline{r^1_\oslash}$	r^2_{fma}	$\overline{r_\otimes}$	$\overline{r^1_{fma}}$
-b	a	$r_\ominus$	$\overline{r_\oplus}$	$\overline{r_\otimes}$	$\overline{r^2_\oslash}$	r^2_{fma}	$\overline{r_\otimes}$	$\overline{r^1_{fma}}$
-a	b	$\overline{r_\ominus}$	$\overline{r_\oplus}$	$\overline{r_\otimes}$	$\overline{r^1_\oslash}$	r^2_{fma}	$\overline{r_\otimes}$	$\overline{r^1_{fma}}$
b	-a	$\overline{r_\ominus}$	$r_\oplus$	$\overline{r_\otimes}$	$\overline{r^2_\oslash}$	r^2_{fma}	$\overline{r_\otimes}$	$\overline{r^1_{fma}}$

$r_{op} = hash(min(|x|, |y|), max(|x|, |y|), op) \quad \forall op \in \{\oplus, \otimes, \ominus\}$
$r^1_\oslash = hash(|x|, |y|, \oslash)$ and $r^2_\oslash = hash(|x|, -|y|, \oslash)$
$r^1_{fma} = hash(min(|x|, |y|), max(|x|, |y|), |z|, fma)$ and
$r^2_{fma} = hash(min(|x|, |y|), max(|x|, |y|), -|z|, fma)$

$\overline{r} = 1 - r$: if r imply `upward` then $\overline{r}$ implie `downward` (and reciproqually)

The cas x=0 and y=0 are ignored because the floating point operations are exact.

Monotonic stochastic rounding

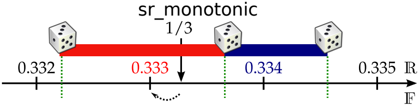
Problem:

```
assert((x <= y) == (a+x <= a+y) )
```

Solution: Introduction of `sr_monotonic`:

$$fl_{monotonic}(a \circ b) = \begin{vmatrix} \lfloor a \circ b \rfloor & \text{if } a \circ b < threshold \\ \lceil a \circ b \rceil & \text{if } a \circ b > threshold \end{vmatrix}$$

with *threshold* sampled (once by program execution) according to an uniform distribution over $[\lfloor a \circ b \rfloor, \lceil a \circ b \rceil]$. In practice *threshold* is obtain thanks a hash function of $round\_toward\_zero(a \circ b)$.



Monotonic stochastic rounding

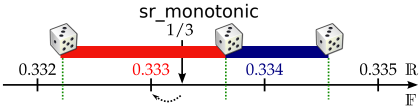
Problem:

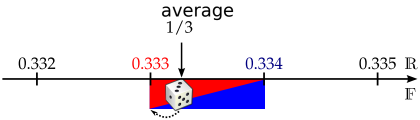
```
assert((x <= y) == (a+x <= a+y) )
```

Solution: Introduction of `sr_monotonic`:

$$fl_{monotonic}(a \circ b) = \left| \begin{array}{ll} \lfloor a \circ b \rfloor & \textit{if } a \circ b < \textit{threshold} \\ \lceil a \circ b \rceil & \textit{if } a \circ b > \textit{threshold} \end{array} \right.$$

with *threshold* sampled (once by program execution) according to an uniform distribution over $[\lfloor a \circ b \rfloor, \lceil a \circ b \rceil]$. In practice *threshold* is obtain thanks a hash function of $round\_toward\_zero(a \circ b)$.





Remarks: In comparison with `average`:

- we loose the independency (inside one verrou execution),
- if the results of each operation are in different intervals, average and sr_monotonic are equivalent.

Monotonic stochastic rounding

Signed version

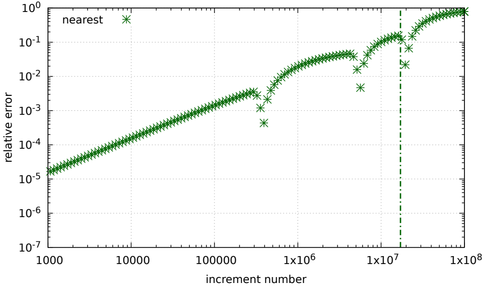
Introduction of `sr_smonotonic`:

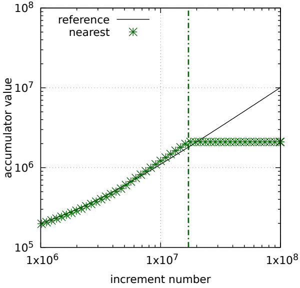
$$fl_{monotonic}(a \circ b) = \begin{vmatrix} \lfloor a \circ b \rfloor & \textit{if } a \circ b < \textit{threshold} & & \\ \lceil a \circ b \rceil & \textit{if } a \circ b > \textit{threshold} & & \\ \lfloor a \circ b \rfloor & \textit{if } a \circ b = \textit{threshold} & \textit{and} & a \circ b < 0 \\ \lceil a \circ b \rceil & \textit{if } a \circ b = \textit{threshold} & \textit{and} & a \circ b > 0 \end{vmatrix}$$

with *threshold* sampled (once by program execution) according to an uniform distribution over $[\lfloor a \circ b \rfloor, \lceil a \circ b \rceil]$
with the following constraint

$$\textit{threshold}([\lfloor -(a \circ b) \rfloor, \lceil -(a \circ b) \rceil]) = -\textit{threshold}([\lfloor a \circ b \rfloor, \lceil a \circ b \rceil])$$

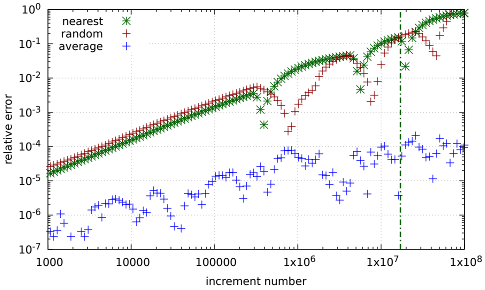
Stagnation behavior: constant increment

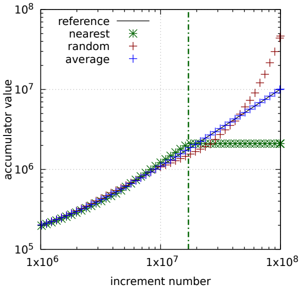




Error (and value) of accumulator (initialized to 100000) after *i* additions of 0.1. The vertical bars represent the begining of stagnation.

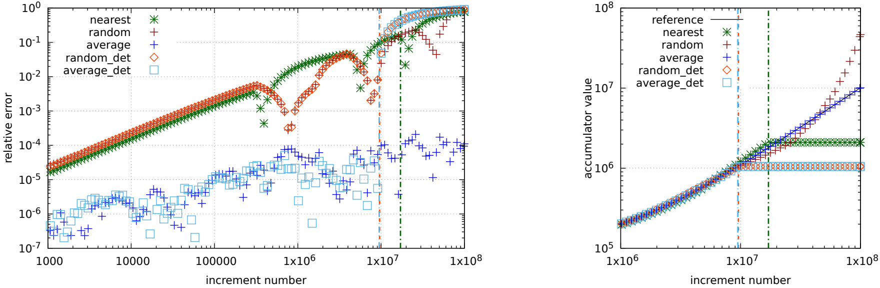
Stagnation behavior: constant increment





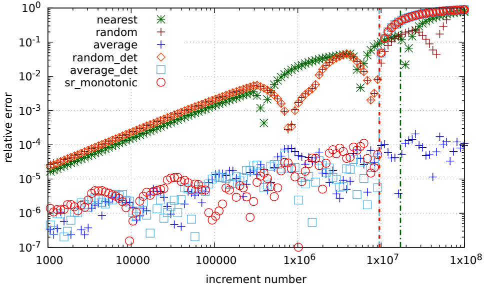
Error (and value) of accumulator (initialized to 100000) after *i* additions of 0.1. The vertical bars represent the begining of stagnation.

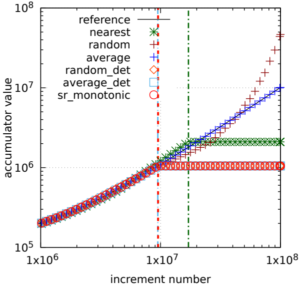
Stagnation behavior: constant increment



Error (and value) of accumulator (initialized to 100000) after i additions of 0.1. The vertical bars represent the begining of stagnation.

Stagnation behavior: constant increment





Error (and value) of accumulator (initialized to 100000) after i additions of 0.1. The vertical bars represent the begining of stagnation.

Performances

Program: stencil with float/double compiled with O0/O3 (implemented with fma)

type	**double**		float	
compilation option	O0	**O3**	O0	O3
nearest	x12.3	**x25.0**	x12.4	x35.3
random	x18.8	**x49.6**	x19.3	x78.0
random_det	x20.5	**x54.8**	x21.0	x89.9
random_comdet	x20.8	**x55.5**	x21.6	x94.1
random_scomdet	x25.2	**x72.0**	x26.7	x125.3
average	x22.7	**x60.7**	x23.3	x100.9
average_det	x24.8	**x66.7**	x26.2	x113.0
average_comdet	x26.1	**x71.1**	x28.1	x132.3
average_scomdet	x26.8	**x75.4**	x29.1	x139.1
sr_monotonic	x24.7	**x68.3**	x27.2	x123.9
sr_smonotonic	x24.7	**x68.7**	x27.5	x125.1

Conclusions and perspectives

The new rounding modes `[random,average]_[[s]com]det` and `sr_[s]monotonic`

- are now available in VERROU master branch: `https://github.com/edf-hpc/verrou`,
- suppress false-positive without code modification nor recompilation,
- implie *acceptable* slowdown,

Open question

- Can be used outside the numerical verification context?