MANCHESTER 1824
The University of Manchester

M\crNA
Manchester Numerical Analysis

Algorithms for Stochastically Rounded Elementary Arithmetic Operations in IEEE 754 Floating-Point Arithmetic

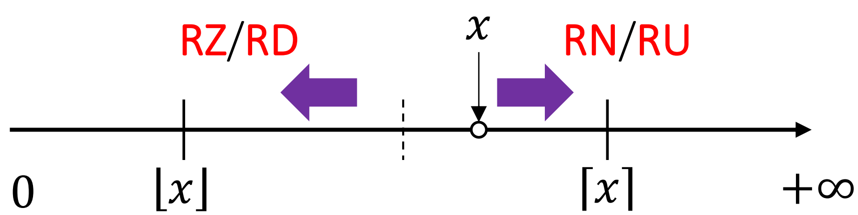
Massimiliano Fasi, Örebro University, Sweden

Mantas Mikaitis, University of Manchester, UK

Contact: *mantas.mikaitis@manchester.ac.uk*

Presentation for SIAM CSE21, 2 March 2021.
Minisymposium on Reduced Precision Arithmetic and Stochastic Rounding.

Motivation of the project



- Rounding modes in the IEEE 754 standards:
  - RN—Round to Nearest, even on ties,
  - RZ—Round towards Zero,
  - RU—Round (Up) towards $+\infty$, and
  - RD—Round (Down) towards $-\infty$.
- *Stochastic Rounding* (SR) is starting to appear in hardware.
- Benefits shown in NLA, PDE and ODE solv., machine learning.
- Simulate SR before it is ubiquitous to
  - test behaviour,
  - develop applications with SR,
  - inform hardware of what is needed.

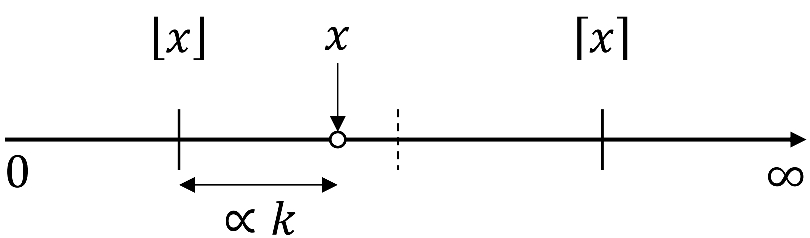
Stochastic rounding (SR)

Definition after Connolly, Higham, and Mary (2021).

Given $x \in \mathbb{R}$ with $\lfloor x \rfloor \leq x \leq \lceil x \rceil$ (with floor and ceiling defined in FP), stochastic rounding (SR) is defined as

$$\mathrm{SR}(x) = \begin{cases} \lceil x \rceil & \text{with the probability } k, \\ \lfloor x \rfloor & \text{with the probability } 1-k. \end{cases}$$

Mode 1	$k = \dfrac{x - \lfloor x \rfloor}{\lceil x \rceil - \lfloor x \rfloor}$
Mode 2	$k = 0.5$



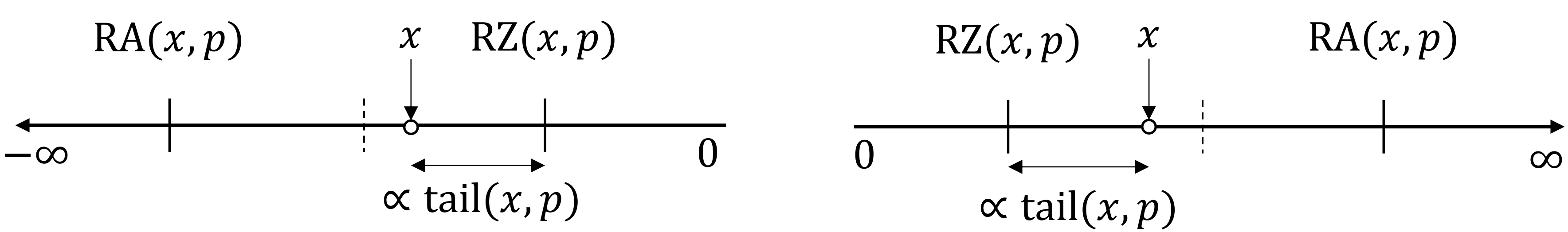
With mode 1, $\mathbb{E}(\mathrm{SR}(x)) = x$.

Bit-level definition of SR in floating-point

Mode 1 SR: given $x \in \mathbb{R}$, a random number $Z \in [0,1)$ from a uniform distribution and the target precision p,

$$\mathrm{SR}(x,p) = \begin{cases} \mathrm{RA}(x,p) & \text{if } Z < \mathrm{tail}(x,p), \\ \mathrm{RZ}(x,p) & \text{if } Z \geq \mathrm{tail}(x,p), \end{cases}$$

where $\mathrm{tail}(x,p) \in [0,1)$ is a value encoded by the trailing bits that do not fit into precision p, RZ—round towards zero, RA—round away from zero.



SR in software and hardware

- Can round numbers from *hardware precision* to lower prec:
  - `Chop` (MATLAB) by Higham and Pranesh (2019).
  - `floatp` (MATLAB) by Meurant (2020) includes floats, fixed point, and posits with SR.
  - `CPFloat` (C) by Fasi and Mikaitis (2020)—very efficient bit-wise implementation.

- Hardware (details not always provided):
  - Davies et al. (2018) included SR in the Intel Loihi chip (inside the MAC units).
  - Graphcore IPU (2020) includes binary16 arithmetic and SR.
  - There are various HW prototypes and patents appearing from the ML community.

Contributions

Problem

- Current simulators round (with SR) to p precision using at least $2p$ precision.
- If we wish to simulate SR of a highest precision in some hardware, we cannot use current simulators.
- Except with arbitrary precision software (Advanpix, MPFR).

Our contributions

- Algorithms for $+, -, \times, \div, \sqrt{}$ with SR in precision p.
- Generalization of binary64 algorithms by Févote and Lathuilière (2016) (use RN only).

No precision different to p required.

Error-free transformations

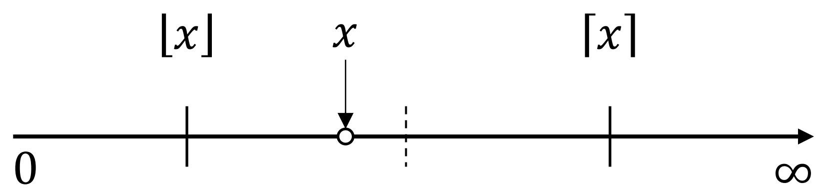
Algorithm I : TwoSum augmented addition.

```
function TwoSum(a ∈ F, b ∈ F, ∘ : ℝ → F)
  Compute σ, τ ∈ F s.t. σ + τ = a + b.
    σ ← ∘(a + b);
    a′ ← ∘(σ − b);
    b′ ← ∘(σ − a′);
    δ_a ← ∘(a − a′);
    δ_b ← ∘(b − b′);
    τ ← ∘(δ_a + δ_b);
    return (σ, τ);
```

Algorithm II : TwoProdFMA augmented multiplication.

```
function TwoProdFMA(a ∈ F, b ∈ F, ∘ : ℝ → F)
  If a, b satisfy (5.1), compute σ, τ ∈ F s.t. σ + τ = a · b.
    σ ← ∘(a × b);
    τ ← ∘(a × b − σ);
    return (σ, τ);
```

Simulating SR of (and using) precision p



- Get the distance to x and use it for SR.
- Use *error-free transforms* to compute

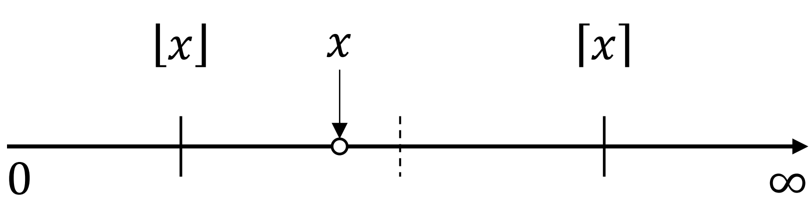
  $\sigma \in \{\lfloor x \rfloor, \lceil x \rceil\}$, and $\tau = x - \sigma$, $|\tau| \in [0, 2^{e_x}\varepsilon)$,

  with e_x exponent of x, and $\varepsilon = 2^{1-p}$.
- Scale random number to be in $[0, 2^{e_x}\varepsilon)$ rather than computing the $\mathrm{tail}(x, p)$ (avoid division).
- We use `TwoSum` and `TwoProdFMA` for $+$ and $\times$.
- Transforms for $\div$ and $\sqrt{}$ exist, but small error in τ.

Class 1: SR addition (using RN/RZ/RU/RD)

Algorithm III : Stochastically rounded addition.

function $\textsc{Add}(a \in \mathcal{F}, b \in \mathcal{F})$
Compute $\varrho = \mathrm{SR}(a+b) \in \mathcal{F}$.
  $Z \leftarrow \mathrm{rand}()$;
  $(\sigma, \tau) \leftarrow \textsc{TwoSum}(a, b, \mathrm{RN})$;
  $\eta \leftarrow \mathrm{get\_exponent}(\mathrm{RZ}(a+b))$;
  $\pi \leftarrow \mathrm{sign}(\tau) \times Z \times 2^{\eta} \times \varepsilon$;
  if $\tau \geq 0$ **then**
    $\circ = \mathrm{RD}$;
  else
    $\circ = \mathrm{RU}$;
  $\varrho \leftarrow \circ(\diamond(\tau + \pi) + \sigma)$;
  return ϱ;

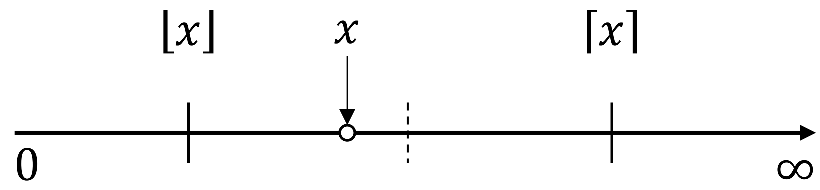


- $Z \in [0,1)$ is a precision-p random value.
- Repeated addition with RZ deals with cases where $\lceil x \rceil$ is a power of 2.
- We choose RD or RU to round towards $\boldsymbol{\sigma}$.
- Comparison in SR def. is performed by the last addition step.

Class 2: SR addition (using RN only)

Algorithm IV : A helper function for stochastic rounding.

1 **function** SRROUND($\sigma \in \mathcal{F}, \tau \in \mathcal{F}, Z \in \mathcal{F}$)
Compute round $\in \mathcal{F}$.
2 **if** $\mathrm{sign}(\tau) \neq \mathrm{sign}(\sigma)$ **then**
3 $\eta \leftarrow$ get_exponent(pred($|\sigma|$));
4 **else**
5 $\eta \leftarrow$ get_exponent(σ);
6 $\mathrm{ulp} \leftarrow \mathrm{sign}(\tau) \times 2^{\eta} \times \varepsilon$;
7 $\pi \leftarrow \mathrm{ulp} \times Z$;
8 **if** $|\mathrm{RN}(\tau + \pi)| \geq |\mathrm{ulp}|$ **then**
9 round $=$ ulp;
10 **else**
11 round $= 0$;
12 **return** round;



- Approach similar to binary64 implementation in VERROU package by Févote and Lathuilière (2016).
- Function pred() allows to avoid requirement of RZ.
- On line 8, comparison replaces RD/RU addition.
- Returns 0 (stay), ulp (go forward), or $-ulp$ (go backward).

SR addition (using only RN)

Algorithm V : Stochastically rounded addition without the change of the rounding mode.

function $\textsc{Add2}(a \in \mathcal{F},\ b \in \mathcal{F})$
  Compute $\varrho = \mathrm{SR}(a+b) \in \mathcal{F}$.
    $Z \leftarrow \mathrm{rand}()$;
    $(\sigma, \tau) \leftarrow \textsc{TwoSum}(a, b, \mathrm{RN})$;
    $\mathrm{round} \leftarrow \mathrm{SR}\textsc{Round}(\sigma, \tau, Z)$;
    $\varrho \leftarrow \mathrm{RN}(\sigma + \mathrm{round})$;
    return ϱ;

In summary, algorithms of class 2 are expected to be faster on Intel, while class 1 faster where switching rounding modes has no cost.

Performance

- Implemented SR $+, -, \times, \div, \sqrt{}$ in binary64.
- Comparison with the approach that uses high-precision library.
- `MPFR 4.0.1` for computing x in higher than binary64 precision.
- 100 pairs of binary64 random numbers.
- Each op with each pair is repeated 10M times.
- Report mean throughput of ops (Mop/s) averaged over 100 pairs.
- Intel Xeon Gold 6130
- `gcc 8.2.0`, `-mfma -mfpmath=sse -msse2` (avoid 80-bit arithmetic).
- `-O0` for algs. that change rounding modes and `-O3` for RN-only algs.

Performance

Throughput (Mop/s):

	sr_mpfr_add			ADD	ADD2	sr_mpfr_mul			MUL	MUL2	sr_mpfr_div			DIV	DIV2	sr_mpfr_sqrt			SQRT	SQRT2
MPFR bits	61	88	113	–	–	61	88	113	–	–	61	88	113	–	–	61	88	113	–	–
min	3.5	3.7	3.5	18.5	62.5	3.7	3.7	3.7	32.2	66.6	3.3	3.4	3.4	31.2	62.5	3.6	3.0	2.8	28.5	52.6
max	3.8	4.2	4.0	31.2	71.4	4.2	4.1	4.0	34.4	76.9	3.6	3.6	3.6	33.3	71.4	4.2	3.6	3.6	30.3	58.8
mean	3.7	3.8	3.8	28.2	68.8	3.9	3.9	3.8	33.9	72.4	3.5	3.5	3.5	32.2	67.2	4.1	3.5	3.5	29.5	57.4
↪ *speedup*	0.9×	1.0×	1.0×	7.3×	17.9×	1.0×	1.0×	1.0×	8.7×	18.6×	1.0×	1.0×	1.0×	9.1×	19.0×	1.1×	1.0×	1.0×	8.3×	16.3×
deviation	0.1	0.1	0.1	2.3	2.5	0.1	0.1	0.1	0.6	2.3	0.1	0.1	0.1	0.4	1.9	0.1	0.1	0.1	0.4	1.7

7.3× to 19× speedup over an approach that depends on the arithmetic of $> 2p$ precision.

Summary

- Hardware with SR is not yet widely available.
- Simulating SR before hardware available is an option.
- We have proposed alternative algorithms for simulating SR.
- Two classes: switch rounding modes or use only RN.
- Algorithms require only IEEE 754 operations, comparisons and some bit-level ops.
- $7.3\times$ to $19\times$ speedup compared with algorithms that require `MPFR` or similar.
- Preprint at http://eprints.maths.manchester.ac.uk/2790/.
- Implementations in `C` and `MATLAB`, and code for experiments available at

  https://github.com/mmikaitis/stochastic-rounding-evaluation.

References

- M. Connolly, N. J. Higham, and T. Mary. Stochastic rounding and its probabilistic backward error analysis. SIAM J. Sci. Comput., vol 43, 2021.
- N. Higham and S. Pranesh. Simulating low precision floating-point arithmetic. SIAM J. Sci. Comput., vol 41, 2019.
- G. Meurant. https://gerard-meurant.pagesperso-orange.fr/. 2020.
- M. Fasi and M. Mikaitis. CPFloat: a C library for emulating low-precision arithmetic. Preprint, 2020.
- M. Davies et al. Loihi: a neuromorphic manycore processor with on-chip learning. IEEE Micro, vol 38, 2018.
- Graphcore IPU. https://docs.graphcore.ai/projects/ipu-overview/en/latest/about_ipu.html. 2020.
- F. Févote and B. Lathuilière. VERROU: assessing floating-point accuracy without recompiling. Preprint, 2016.