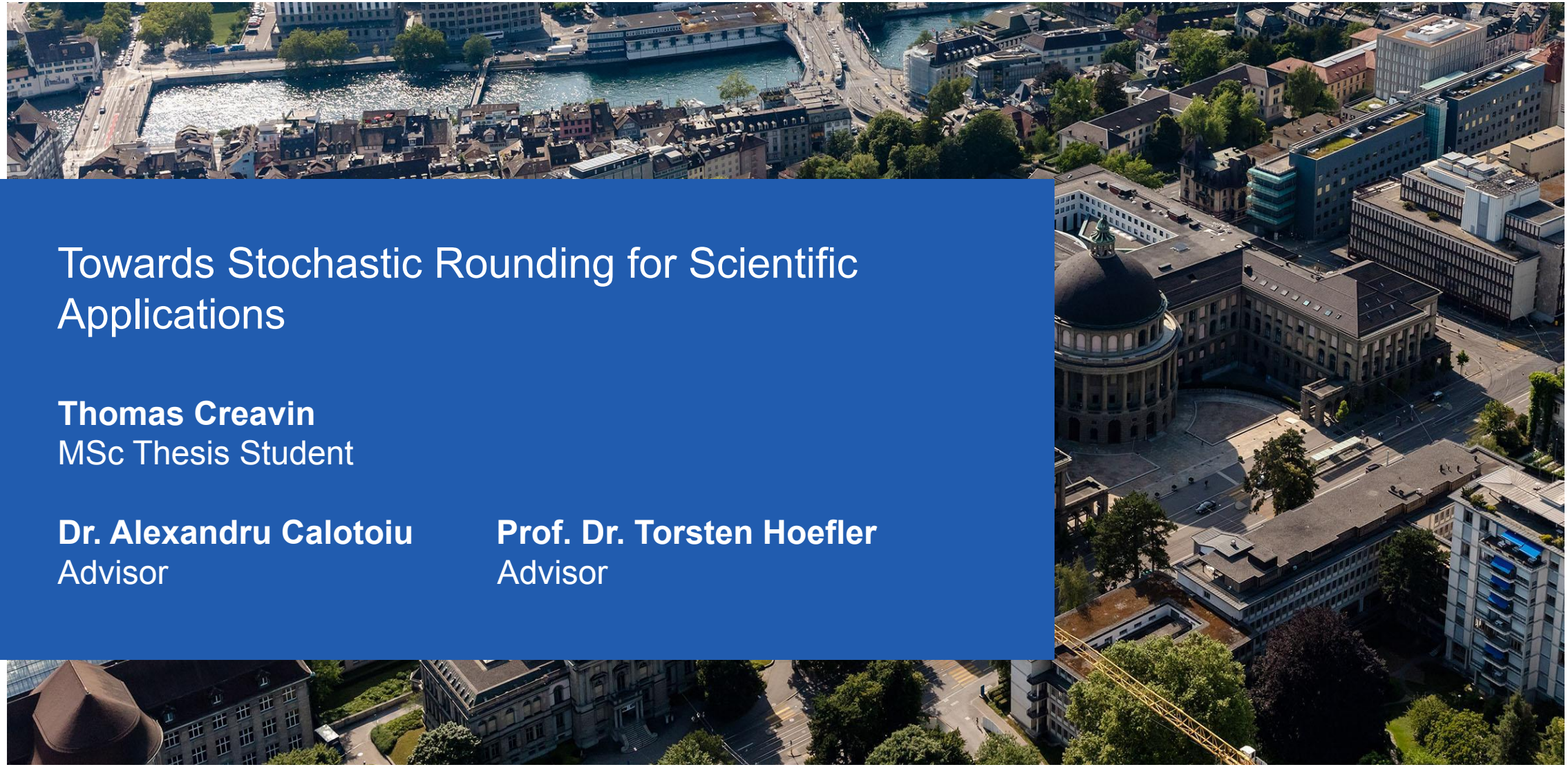
ETH zürich



Towards Stochastic Rounding for Scientific Applications

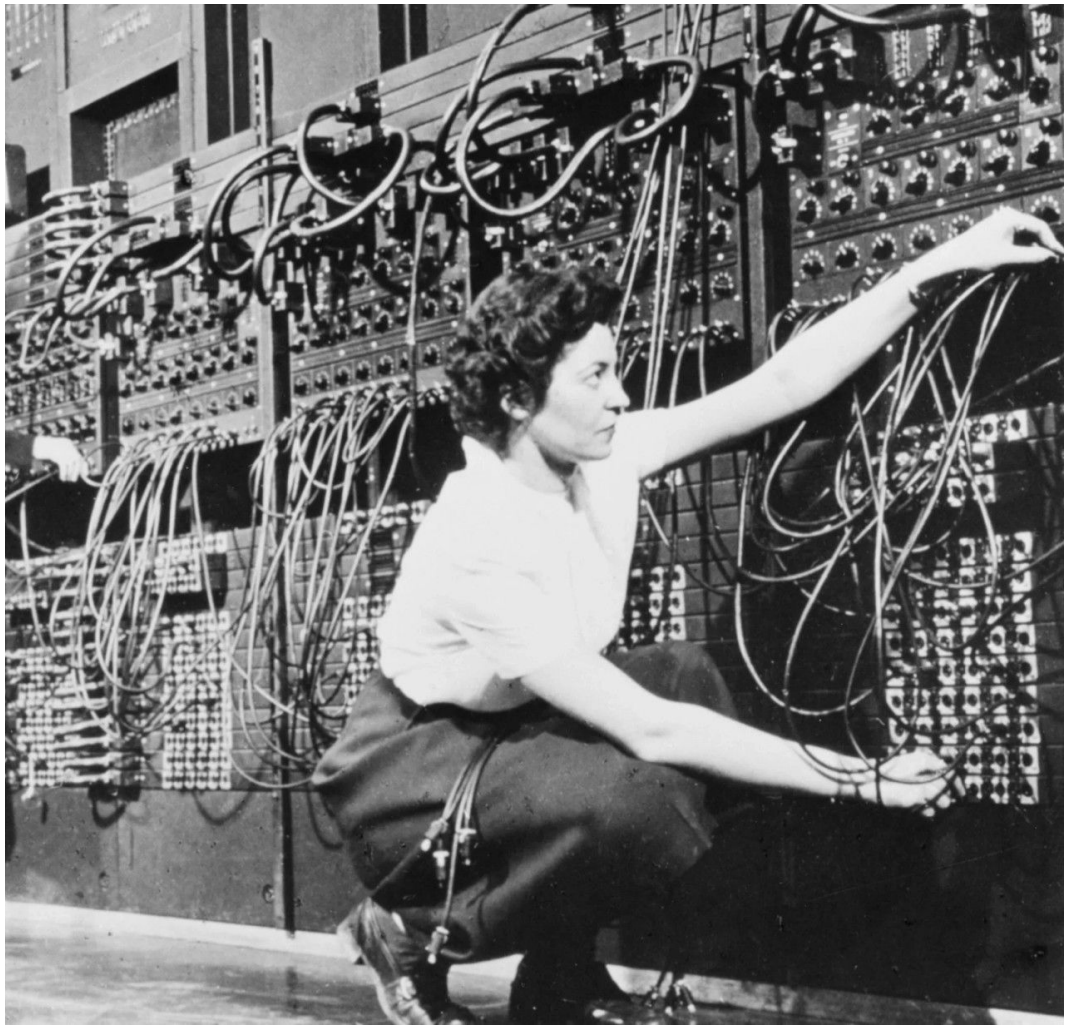
Thomas Creavin
MSc Thesis Student

Dr. Alexandru Calotoiu
Advisor

Prof. Dr. Torsten Hoefler
Advisor

11 September 2025

Overview



“Everything old is new again.”

— Proverb

U.S. National Archives

Problem

Specification	GB200	GH200
FP64	80 TFLOPS	34 TFLOPS
FP64 Tensor Core	80 TFLOPS	67 TFLOPS
FP32	160 TFLOPS	67 TFLOPS
TF32 Tensor Core	2.5 PFLOPS	494 TFLOPS
FP16/BF16 Tensor Core	5 PFLOPS	990 TFLOPS
FP8 Tensor Core	10 PFLOPS	990 TFLOPS
INT8 Tensor Core	10 POPS	1,979 TOPS
FP4 Tensor Core	20 PFLOPS	–

Table 1.1: NVIDIA GB200 and GH200 specifications showing the increased performance of low-precision floats and in particular that of the low precision tensor cores.

- Modern hardware increasingly relies on lower-precision floating-point arithmetic
- Challenge for traditional double-precision climate simulations

Our Approach

- Integrated Stochastic Rounding (SR) into DaCe framework
- Extended NPBench for error measurement
- Analyzed RNG performance to balance speed & accuracy
- Enabled automated SR transformation with minimal developer effort

Key Results

- Replicated literature results + expanded to various kernels & ICON model
- Demonstrated significant SR benefit for long accumulation chains & stagnation-prone simulations
- Demonstrated marginal benefits for ICON & similar kernels

Background

- Floating-point formats
- Rounding Modes
- Stochastic rounding

Floating-point Formats

$$13_{10} = 8 + 4 + 1 = 2^3 + 2^2 + 2^0 = 1101_2$$

$$3.125_{10} = 2 + 1 + \frac{1}{8} = 2^1 + 2^0 + 2^{-3} = 11.001_2$$

$$3.125_{10} = 1.5625_{10} \times 2^1 = 1.1001_2 \times 2^1$$

Floating-point Formats

Precision	Format	Sign bit	Significand bits	Exponent bits
Double	Floating-point 64	1	52	11
Single	Floating-point 32	1	23	8
Half	Floating-point 16	1	10	5

Table 2.1: IEEE 754 Floating-Point Number Representations

Floating-point Formats

Precision	Format	Sign bit	Significand bits	Exponent bits
BFloat16	Brain Floating-point 16	1	7	8
TF32	TensorFloat-32	1	10	8
Mini Float	Mini Floating-point 8	1	4	3
BFP16	Block Floating Point 16	1	15	8 (shared)

Table 2.2: Alternative Floating-Point Number Representations

IEEE 754 Rounding Modes

1. Round to Nearest

- **Ties to Even** – Nearest value; ties go to even digit *(default for binary FP)*

- **Ties Away from Zero** – Nearest value; ties go away from zero

2. Directed Roundings

- **Toward 0** – Truncate toward zero

- **Toward +∞** – Round up (ceiling)

- **Toward −∞** – Round down (floor)

Stochastic Rounding

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

$$p(x) = 1 - \frac{\lceil x \rceil - x}{\lceil x \rceil - \lfloor x \rfloor}$$

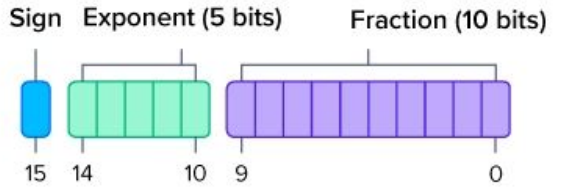
Why Is This Beneficial?

- Consider FP16
- The real numbers 10,000 ≤ x＜10,025 can only be represented by:

  10000, 10008, 10016, 10024

- With RTN, 10,003 + 0.99 always results in 10,000
- With SR, ~50% 10,000, ~50% 10,008
  - Expected value is ~10,004

Half Precision (FP16) Spec



Applications of Stochastic Rounding

- Primarily used in machine learning to perform training and inference at low precision (≤ 16bits)
- Survey paper mentions use in:
  - Numerical linear algebra
  - Numerical verification software e.g CADNA
  - ODE and PDE solvers
  - Quantum computing
  - Digital signal processing

Why Is It Not More Widely Applied?

- Not available in hardware
  - Except in limited research and IPU chips
  - Patents by NVIDIA, AMD, IBM
- Software emulation costs 5× to 10× more computer
  - Largest cost is random number generation

graphcore.ai; intel.com; spinnaker2.gitlab.io

Why Are We Interested in Stochastic Rounding?

- Recent studies show that numerical precision can be reduced while still preserving accuracy
- Theoretical speedups: 2× to 4× higher throughput when precision is halved, quartered etc
- Practical motivation: Transitioning from double-precision CPU computations to accelerators (e.g., GPUs/TPUs) with vastly more single-precision ALUs

State of the Art for Reducing Precision

- Transitioning to Single-precision
- Scientific Applications of Stochastic Rounding

Transitioning to Single-precision: ECMWF Case Study

- ECMWF successfully moved Integrated Forecasting System (IFS) from double to single precision while preserving accuracy.
- Transition took ~4 years, from initial research to operational single-precision forecasts.
- ~40% runtime reduction without losing forecast quality.
- Precision can be easily changed with FORTRAN KIND parameter
- Significant additional effort needed for accurate results

Transitioning to Single-precision: ECMWF Case Study

- Remove hard-coded constants (e.g., 10.E+100 → huge(x))
- Adapt linear algebra & MPI interfaces
- Modify I/O for binary data precision
- Adjusted radiation scheme time-stepping
- Legendre transforms kept in double-precision
- Precompute operators in double then truncate to single

Summation of the Harmonic Series

$$\sum_{n=1}^{\infty} \frac{1}{n} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots$$

Format	Sum at $i = 5 \times 10^6$	Error at $i = 5 \times 10^6$
FP64	16.002	0
FP32	15.404	0.598
FP16	7.086	8.916
BFloat16	5.063	10.94
s16.15 RTN	11.938	4.064
s16.15 RD	10.553	5.449
s8.7 RTN	6.414	9.588
s8.7 RD	5.039	10.963
s16.15 SR	16.002 (s.d. 0.012)	-1.4×10^{-4}
FP32 SR	16.002 (s.d. 8×10^{-4})	-3.5×10^{-5}
s8.7 SR	11.205 (s.d. 0.242)	4.797
FP16 SR	11.638 (s.d. 0.012)	4.364
BFloat16 SR	15.355 (s.d. 0.639)	0.647

Lorenz Simulation

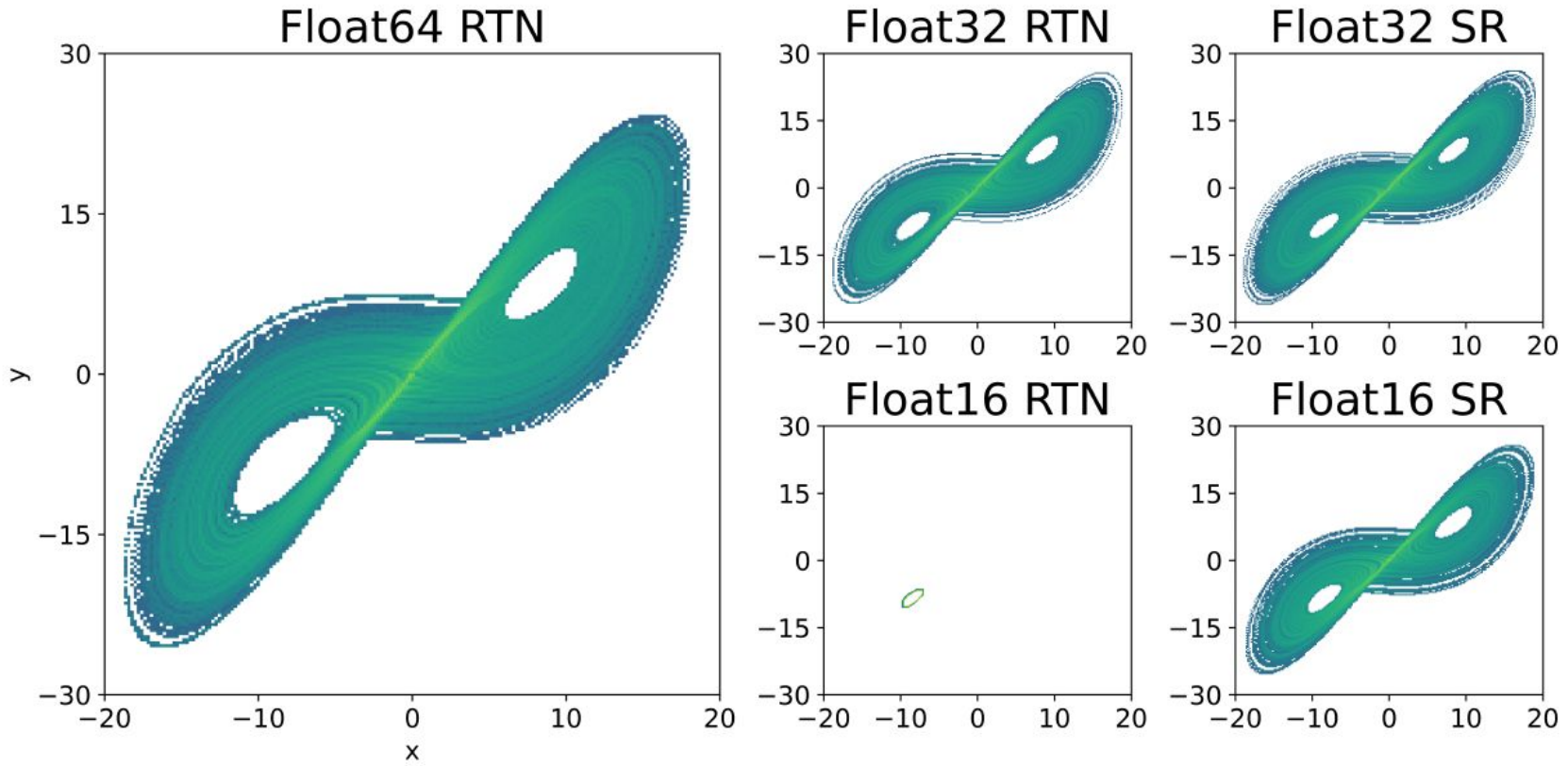


Figure 3.1: Our simulation of the Lorenz system at different floating-point precisions. Brighter colors indicate higher point density. Orbits are plotted on a 200×200 pixel grid, except for Float16 RTN, which is plotted on a 25×25 grid for legibility.

Lorenz Simulation

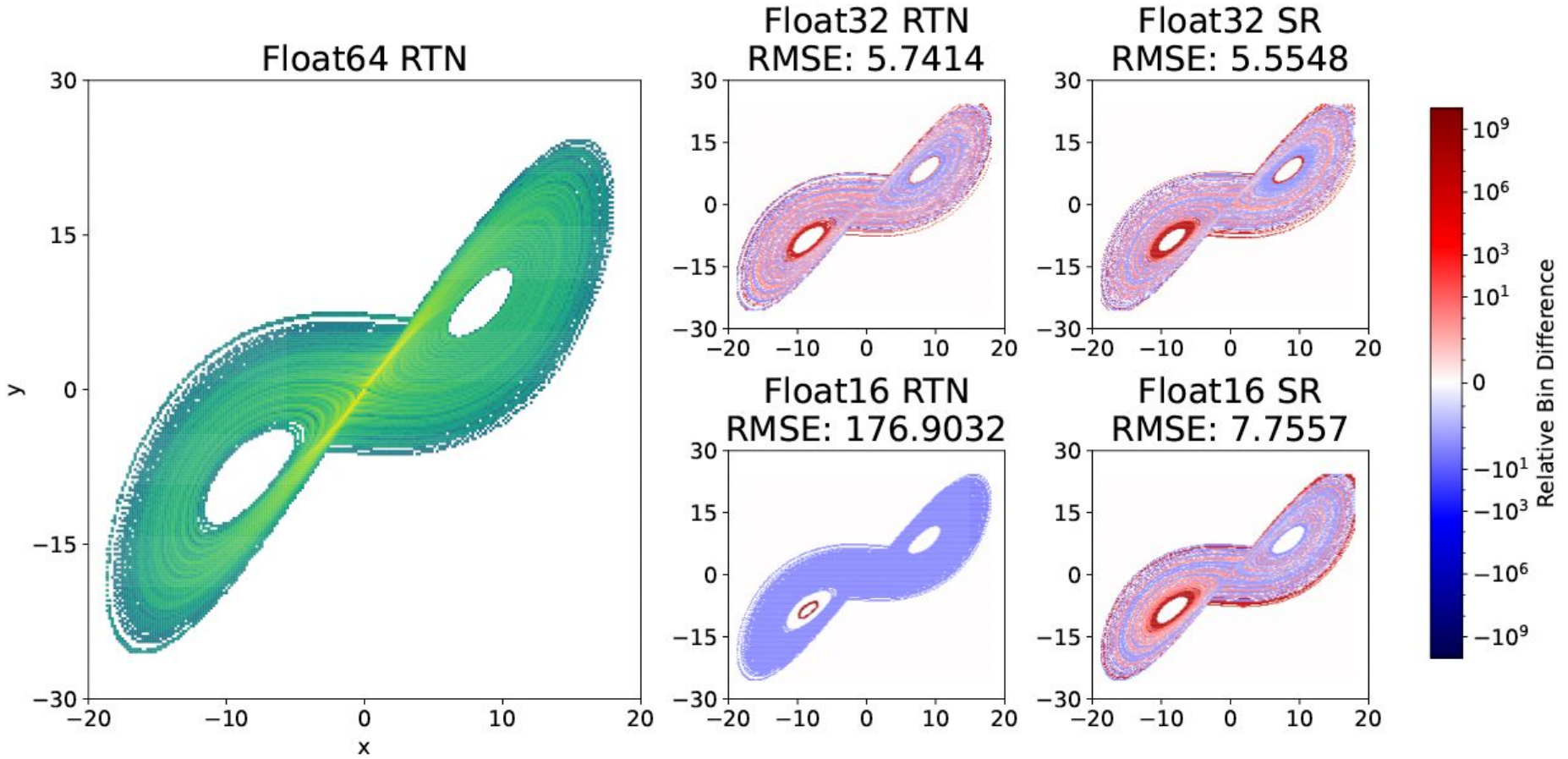


Figure 3.2: Root mean squared error (RMSE) of the pixel-wise differences between Lorenz system simulations across different floating-point formats.

Shallow Water Simulation

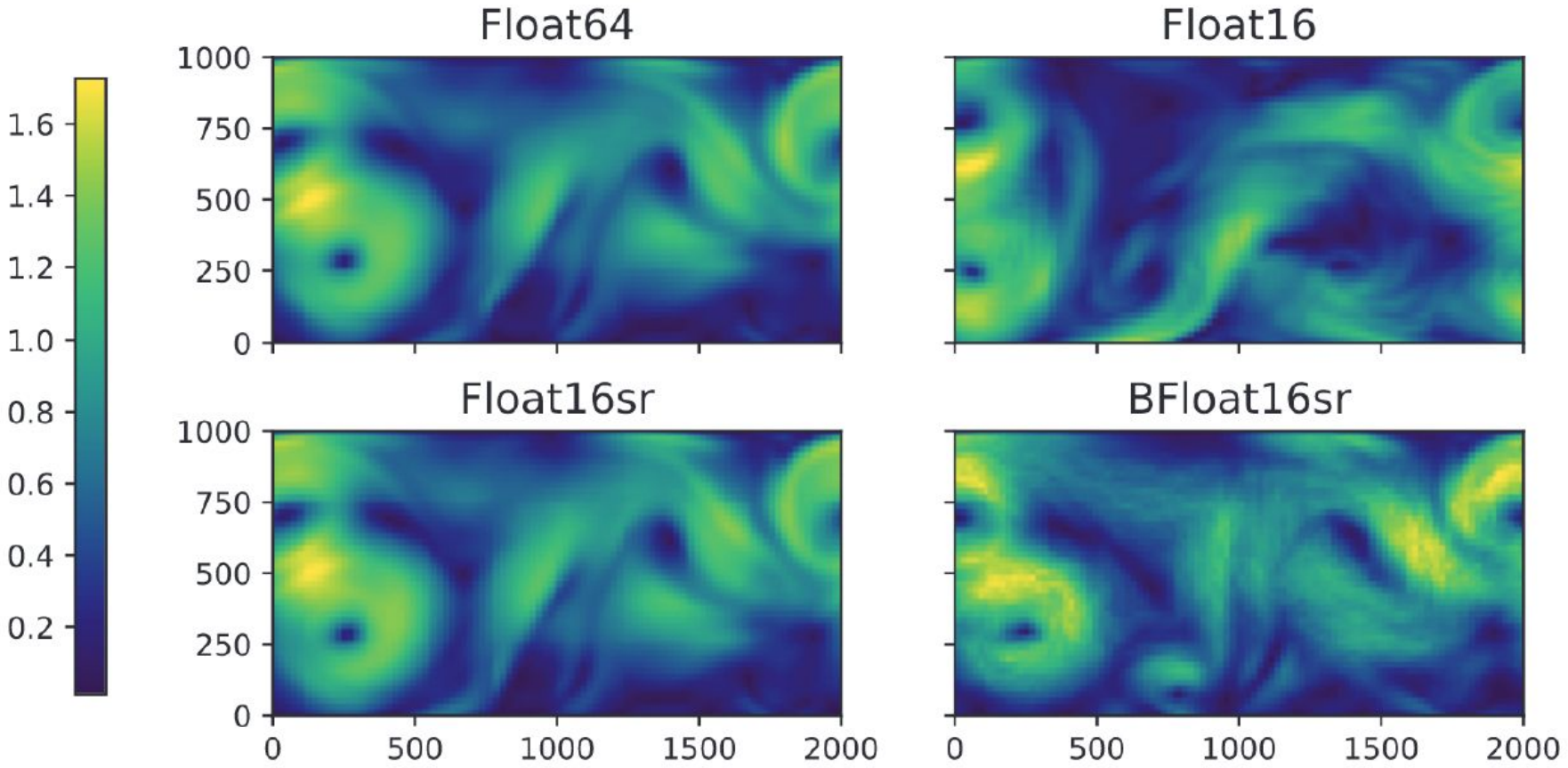


Figure 3.3: The shallow water model integrated at different precision levels by Paxton. A snapshot of the flow speed (m s^{-1}), initiated from the same initial condition, after 50 days.

Heat Diffusion in a Soil Column

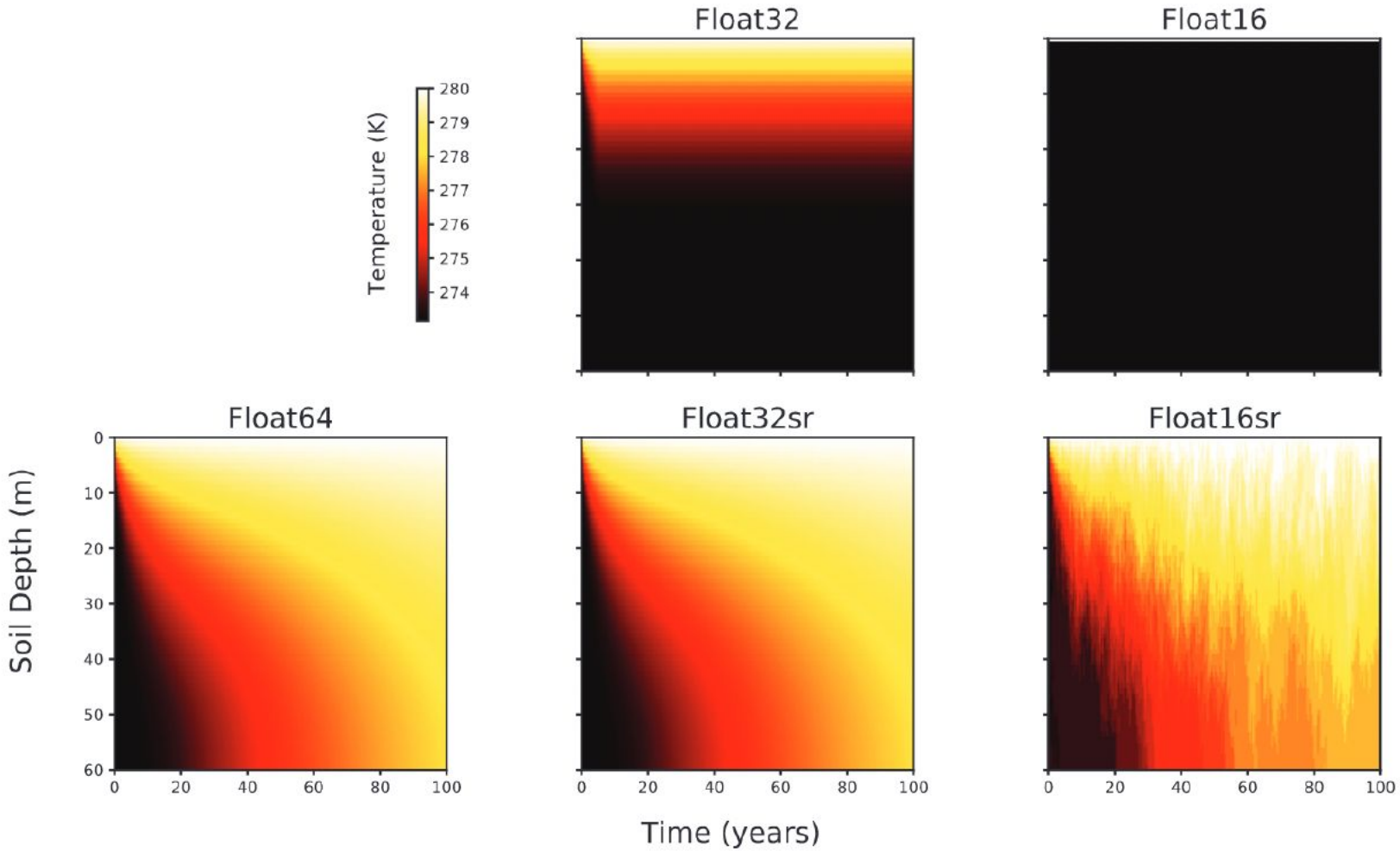


Figure 3.4: Heat diffusion in a soil column with different number formats and rounding modes

Implementation

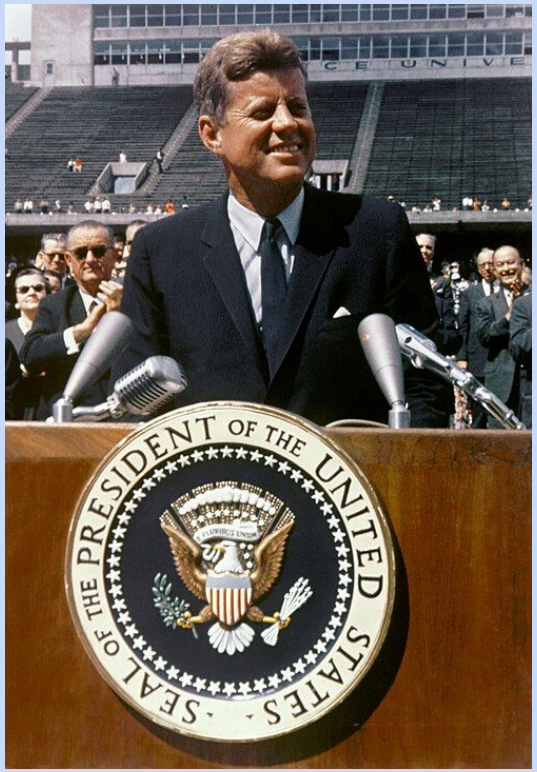
- Stochastic Rounding in DaCe
- DaCe Type Change Pass
- Error Benchmarking in NPBench

Bringing Stochastic Rounding to DaCe

- Stochastic rounding works on toy problems
- Literature begs the question, what about complex simulations?
- SPCL is collaborating with MPI-M to bring DaCe to the ICON Climate Model
- Can we add SR to DaCe to perform ICON procedures at low precision with little increase in error in shorter time?

Goals

- Must be performant and easy to use
- Replicate existing experiments in DaCe
- Benchmark the performance and error
- Apply stochastic rounding to an ICON procedure



“We choose to [add stochastic rounding] not because it is easy but because it is hard”
— JFK

wikipedia.org

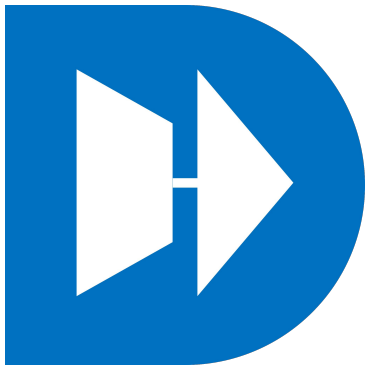
Implementation in DaCe

- Introduced two new types:

  `dace::float32sr`

  `dace::float16sr`

- Added CUDA support for the new types
  - Surprisingly, no feature parity with single and half precision
- Unit tests for operation behaviour and statistical properties



Typical Stochastic Rounding Implementation

Algorithm 1 Stochastic Rounding by Perturbation and Truncation

```
1: function StochasticRound(double)
2:     rand ← RNG32()
3:     mask ← (1 ≪ 29) − 1                ▷ Mask bits lost to rounding
4:     rand ← rand & mask
5:     double ← double + rand             ▷ Perturb bits lost to rounding
6:     double ← double & ∼mask            ▷ Truncate surplus bits
7:     return FloatCast(double)
8: end function
```

- SR(2.3) to the nearest integer: generate a random value [0, 1.0), add it to 2.3, truncate the decimal part

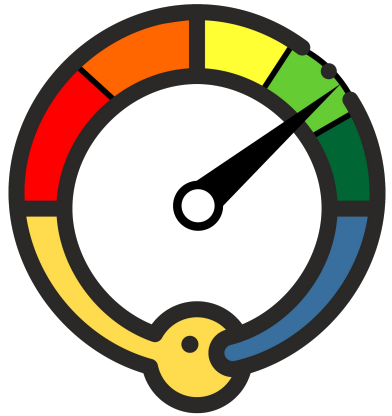
Deviations From The Typical Stochastic Rounding Implementation

- Difference: consume random values from a large circular array populated at initialization
- Alternative ideas
  - A queue of rands populated by a background thread
  - Random rounding
  - Using input as a source of randomness

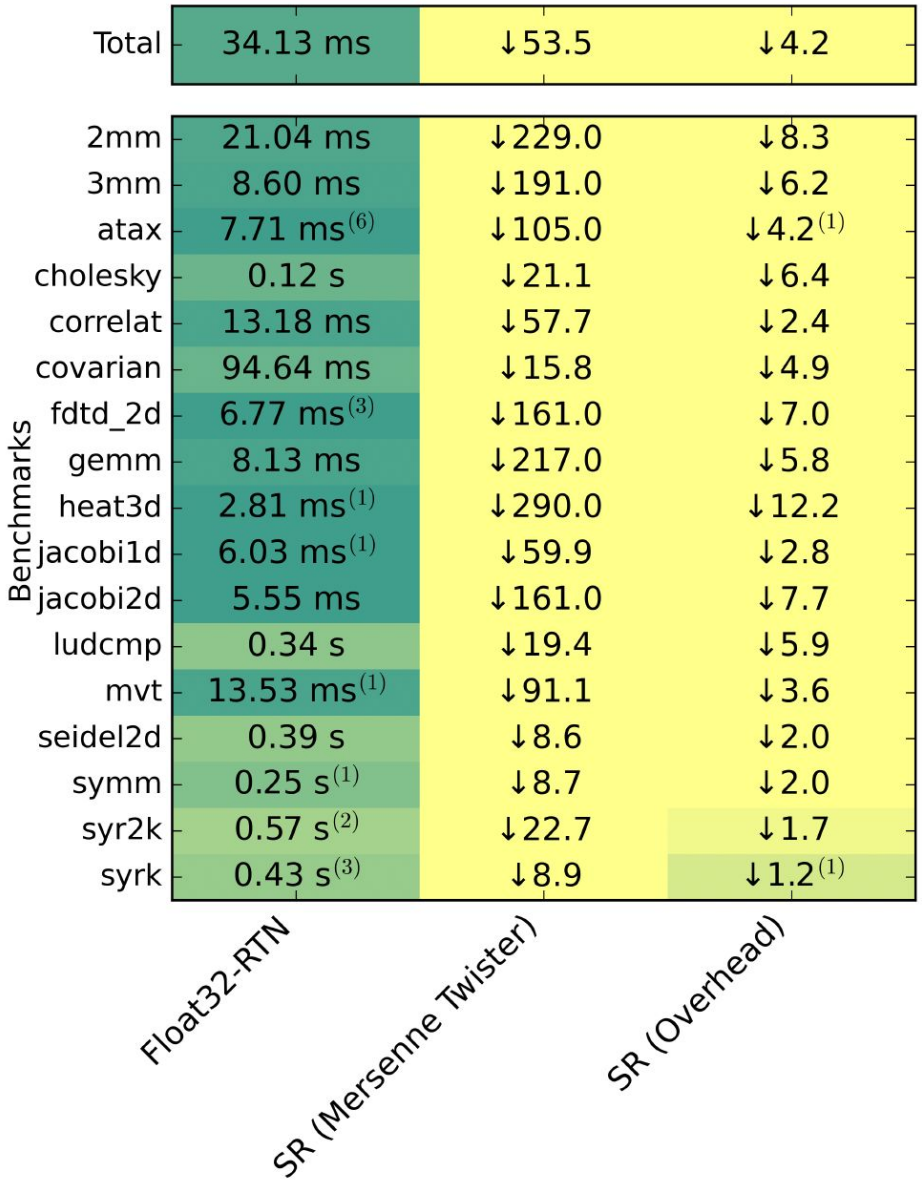
Interstellar (2014)

DaCe Stochastic Rounding Performance

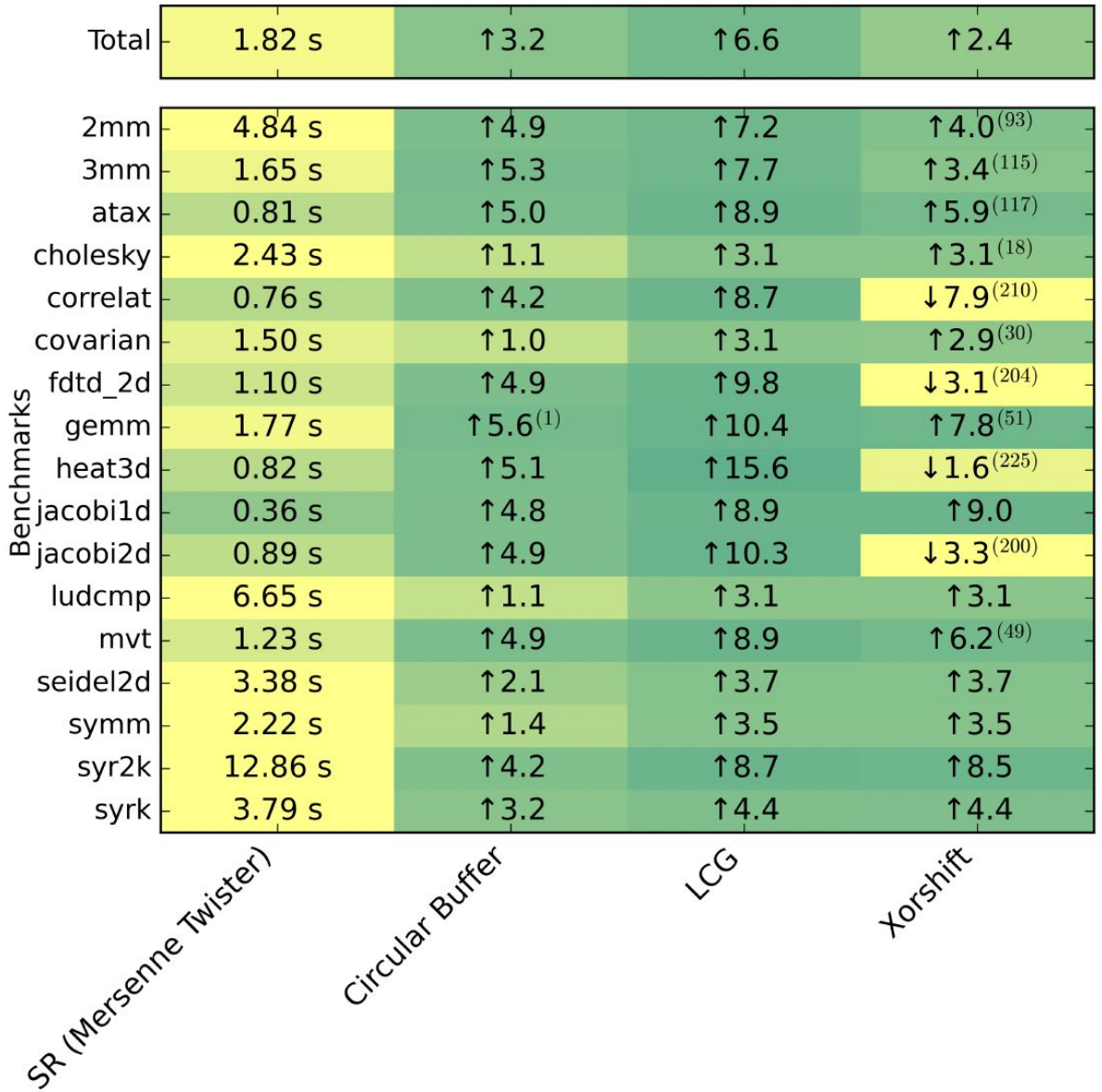
- From literature, we should expect a ~5× performance hit
- We leverage NPBench to measure the performance and to check compatibility with Python/Numpy language features
- Add a NPBench test suite for measuring error
- Initial tests showed > 10× performance hit
  - > 50% of the cost was incurred by the RNG
- We would like to improve upon that



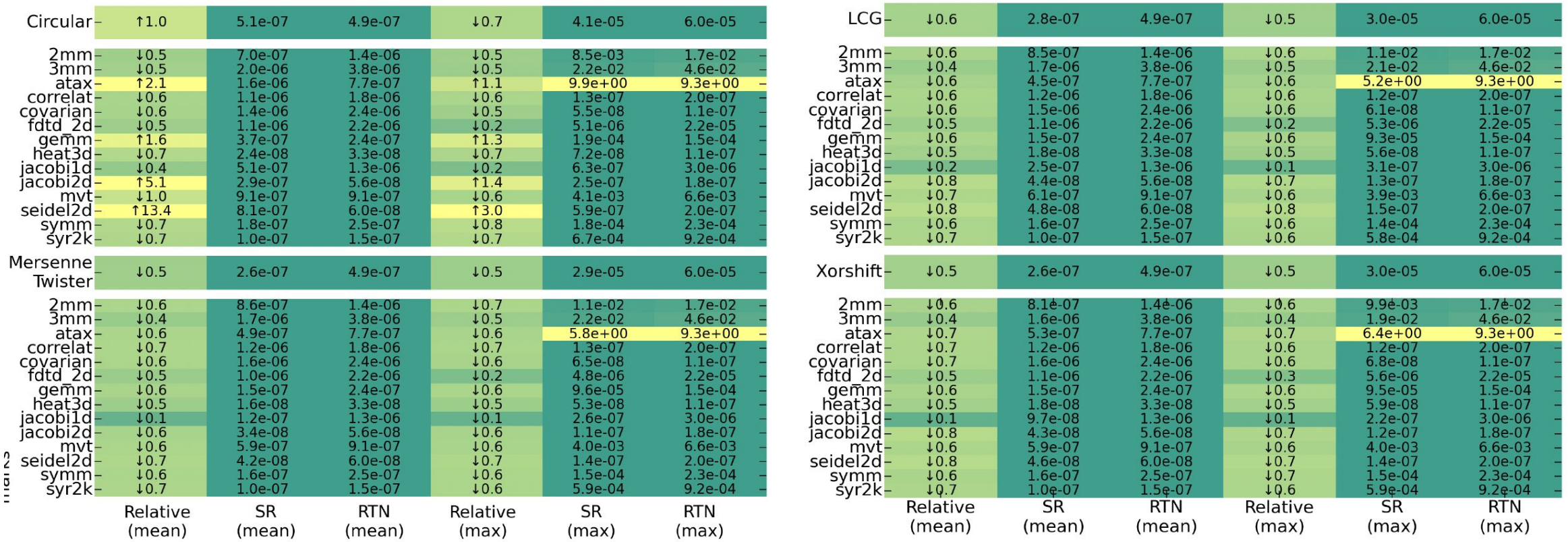
NPBench Performance



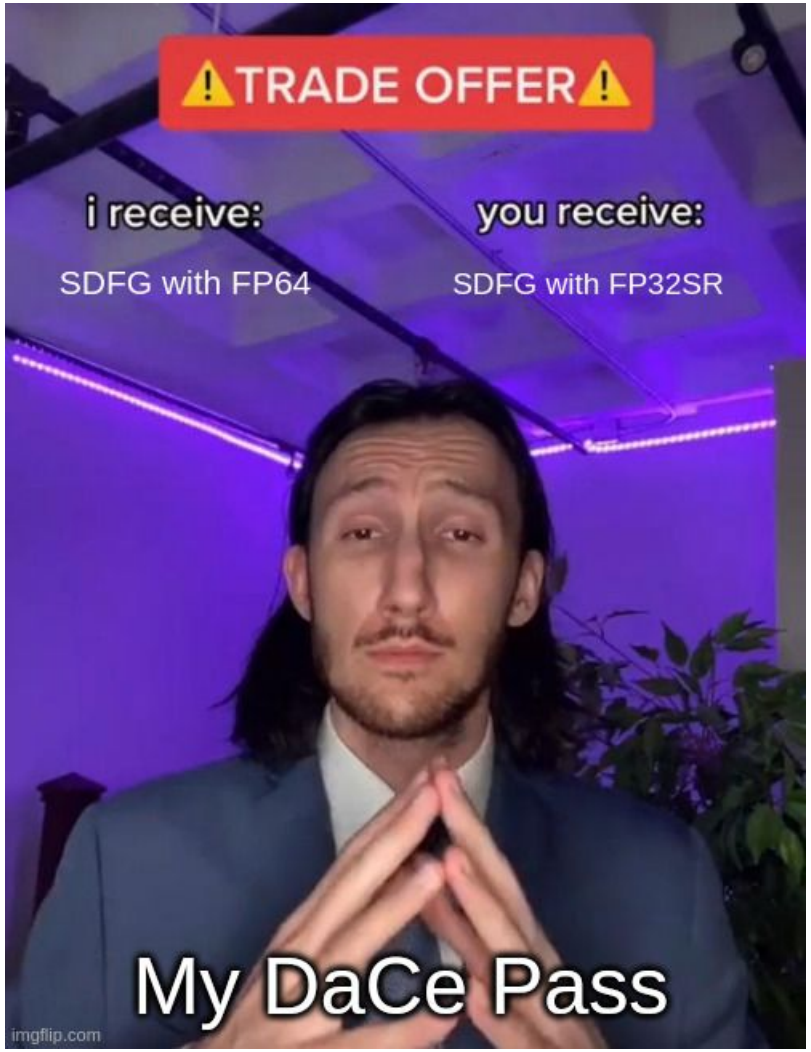
NPBench Performance



NPBench Performance

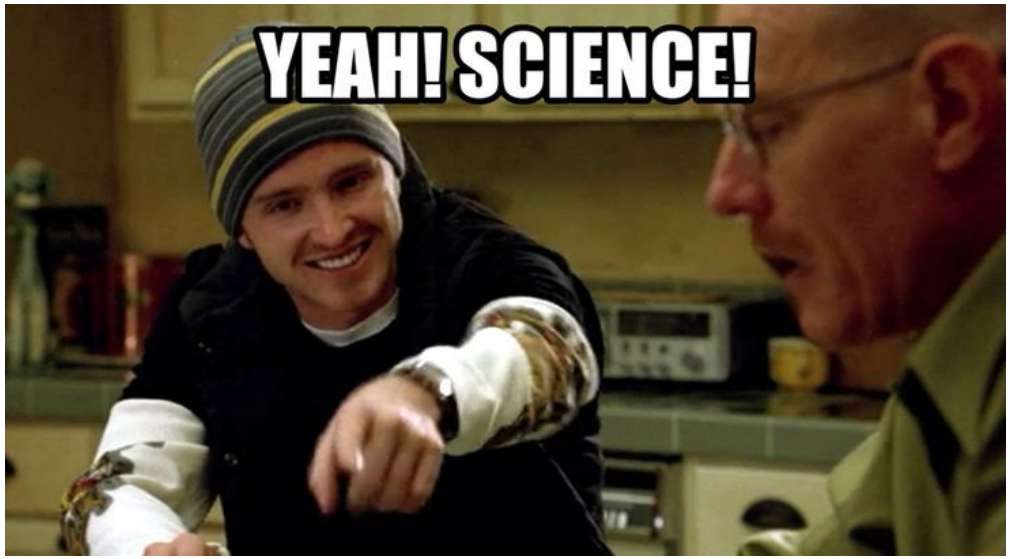


Type Change DaCe Pass



- Previously we applied SR types manually
  - Not sustainable for large SDFGs like VT
- Created a DaCe Pass to swap arbitrary simple data types

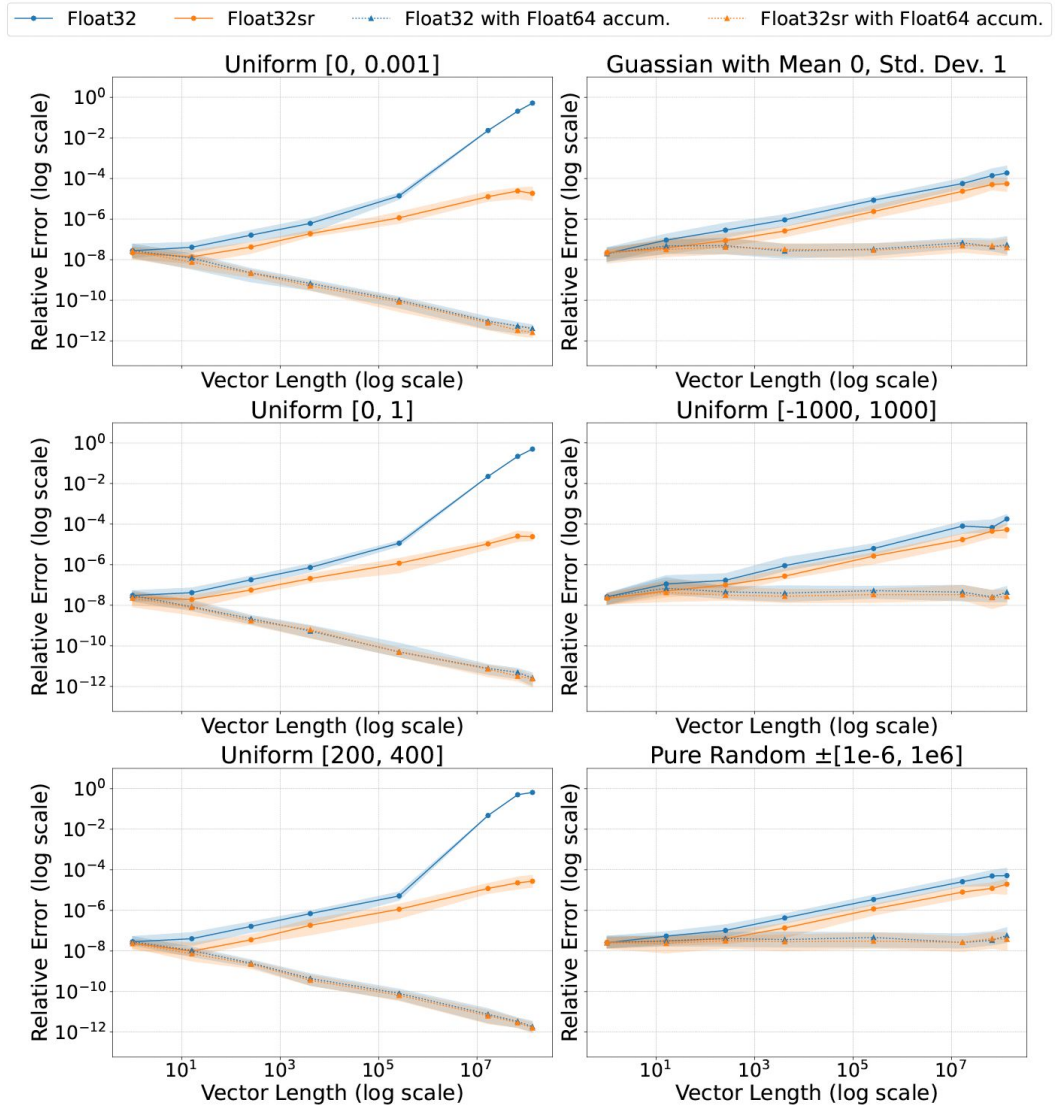
Experiments



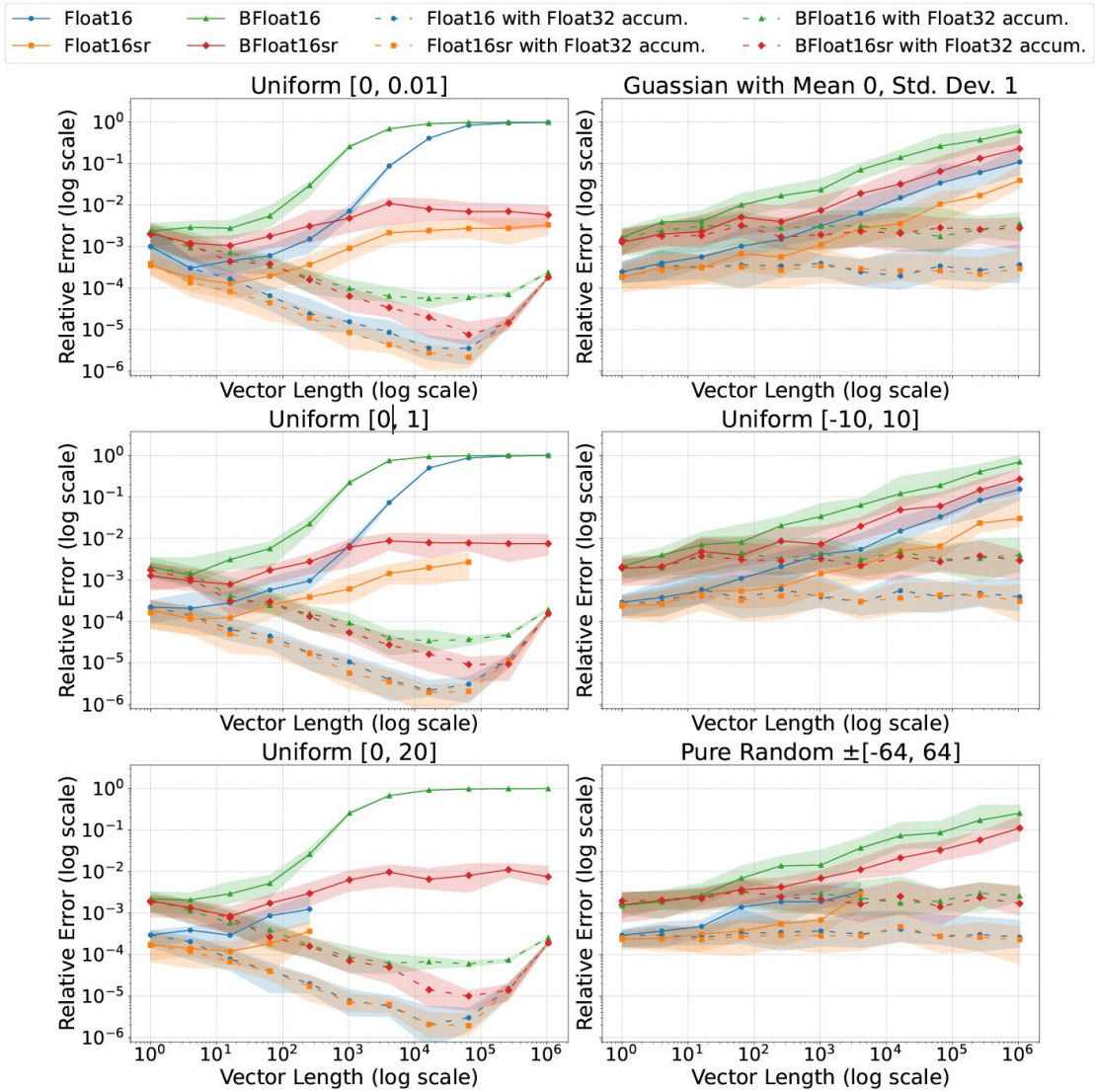
- Dot Product
- GEMM
- ICON

Breaking Bad (2008)

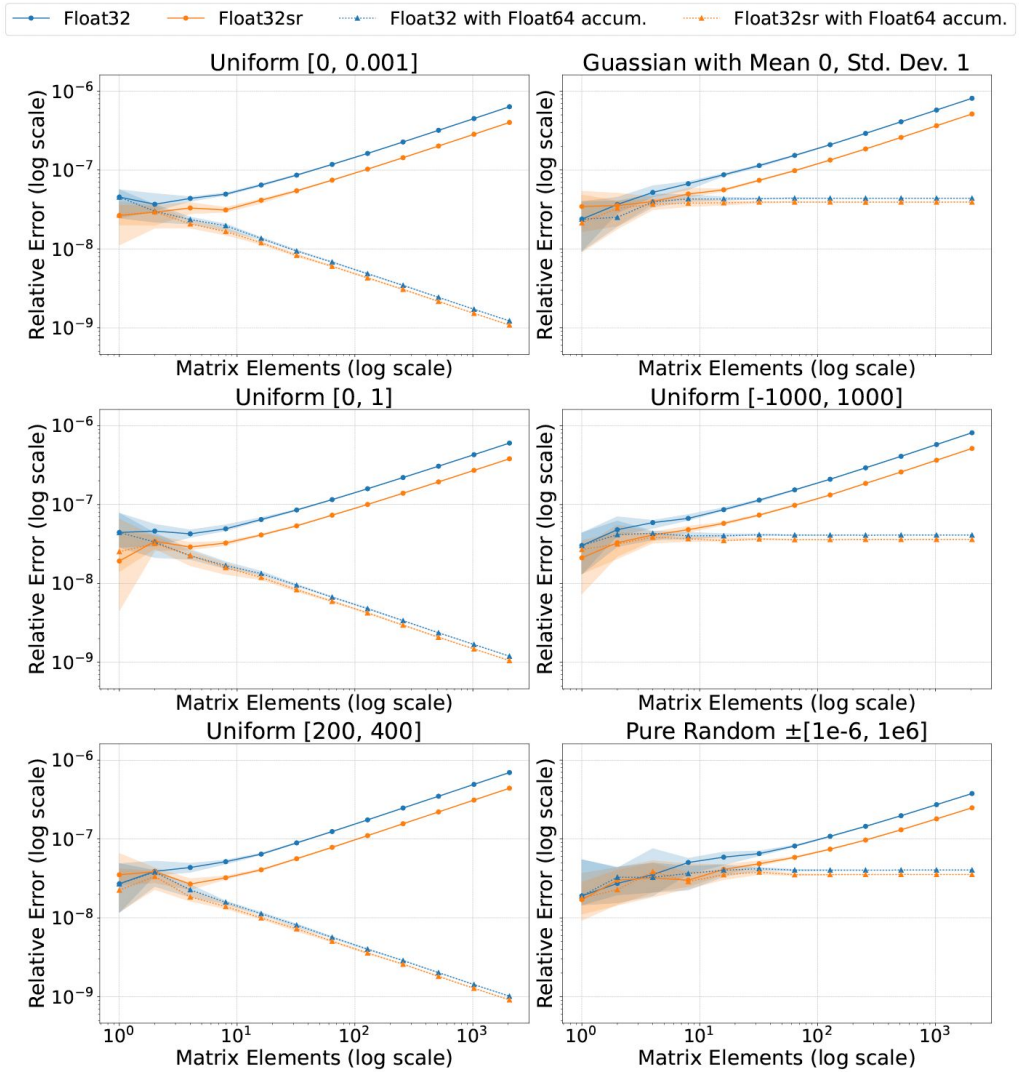
Dot Product


Float32
Float32sr
Float32 with Float64 accum.
Float32sr with Float64 accum.
Uniform [0, 0.001]
Guassian with Mean 0, Std. Dev. 1
Uniform [0, 1]
Uniform [-1000, 1000]
Uniform [200, 400]
Pure Random ±[1e-6, 1e6]
Relative Error (log scale)
Vector Length (log scale)

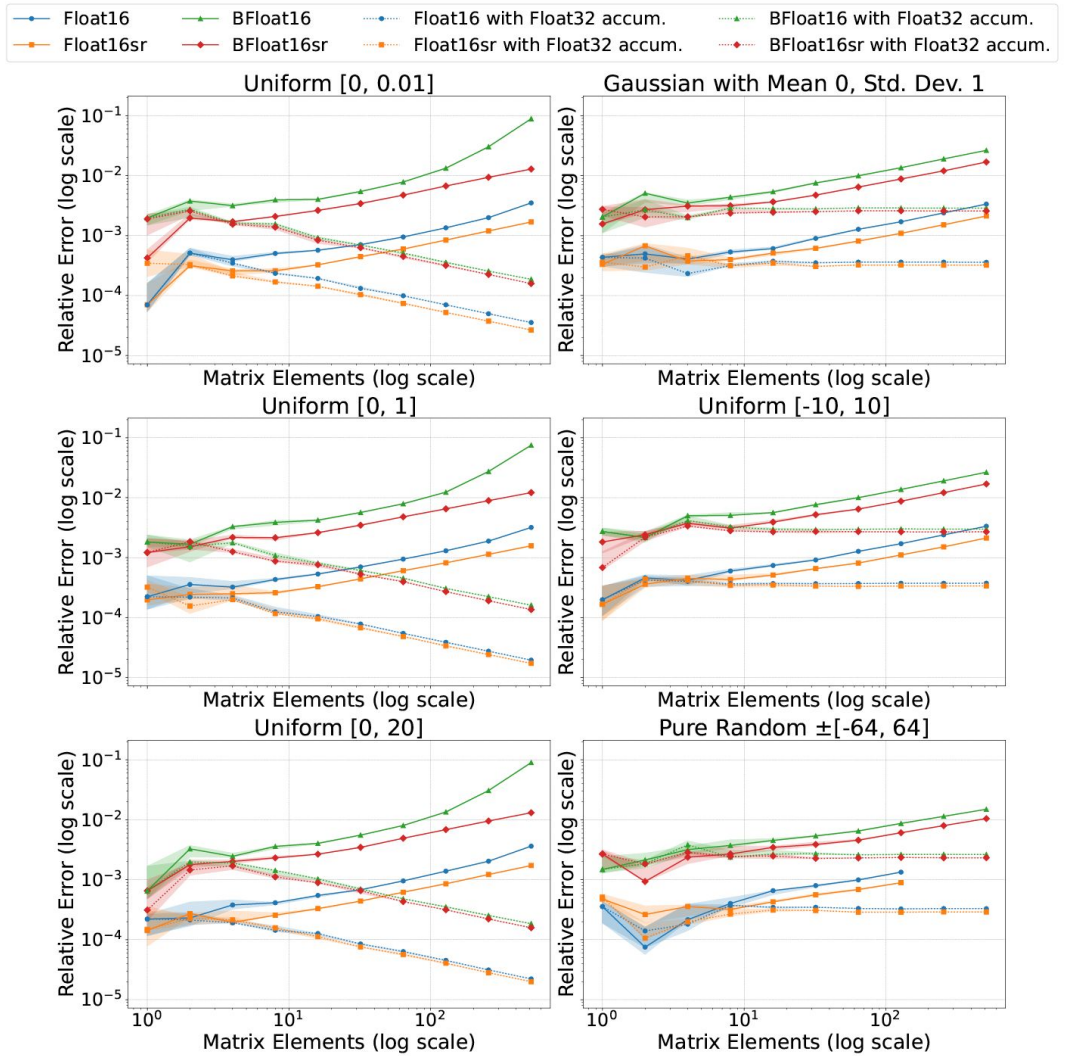
Dot Product


Float16
Float16sr
BFloat16
BFloat16sr
Float16 with Float32 accum.
Float16sr with Float32 accum.
BFloat16 with Float32 accum.
BFloat16sr with Float32 accum.
Uniform [0, 0.01]
Guassian with Mean 0, Std. Dev. 1
Uniform [0, 1]
Uniform [-10, 10]
Uniform [0, 20]
Pure Random ±[-64, 64]
Relative Error (log scale)
Vector Length (log scale)

General Matrix Multiply (GEMM)



General Matrix Multiply (GEMM)



Integration with ICON

- SPCL has been working to bring DaCe to ICON
- github.com/spcl/icon-artifacts contains full SDFGs of ICON dynamical core procedures
- Of interest is the Velocity Tendencies procedure
  - Large example that validates and spans many DaCe features

Integration with Velocity Tendencies Procedure

- Ran and validated VT with double precision
- Created an updated validation script to measure error on an element by element basis
- Ran VT at single precision with RTN and SR
    - We would like FP32SR to produce results much closer to FP64 than what FP32 can produce

ICON Velocity Tendencies

Array	Method	Mean Abs Error	Median Abs Diff	Q75	Max Abs Error	Mean Rel Error
ddt_vn_apc_pc	SR	1.70×10^{-10}	1.16×10^{-10}	2.33×10^{-10}	2.79×10^{-9}	1.533×10^{-2}
	RTN	1.45×10^{-10}	8.73×10^{-11}	2.33×10^{-10}	2.10×10^{-9}	5.815×10^{-3}
vn_ie	SR	7.71×10^{-7}	4.77×10^{-7}	9.54×10^{-7}	7.63×10^{-6}	8.475×10^{-8}
	RTN	7.59×10^{-7}	4.77×10^{-7}	9.54×10^{-7}	3.81×10^{-6}	8.384×10^{-8}
vt	SR	8.08×10^{-7}	4.77×10^{-7}	9.54×10^{-7}	7.63×10^{-6}	2.038×10^{-2}
	RTN	7.56×10^{-7}	4.77×10^{-7}	9.54×10^{-7}	3.81×10^{-6}	2.368×10^{-2}
w_concorr_c	SR	2.73×10^{-15}	4.44×10^{-16}	1.78×10^{-15}	2.27×10^{-13}	2.572×10^{-3}
	RTN	2.39×10^{-15}	4.44×10^{-16}	1.78×10^{-15}	1.14×10^{-13}	1.952×10^{-3}
z_kin_hor_e_1	SR	1.81×10^{-5}	7.63×10^{-6}	3.05×10^{-5}	3.05×10^{-4}	1.056×10^{-7}
	RTN	1.65×10^{-5}	7.63×10^{-6}	3.05×10^{-5}	2.44×10^{-4}	9.640×10^{-8}
z_w_concorr_me_1	SR	3.86×10^{-15}	4.44×10^{-16}	3.55×10^{-15}	6.82×10^{-13}	6.396×10^{-3}
	RTN	3.51×10^{-15}	4.44×10^{-16}	1.78×10^{-15}	4.55×10^{-13}	1.458×10^{-4}

Conclusion

- Contributions
- Future Work

Contributions

- New DaCe Pass for modifying SDFG types
- New reasonably performant SR data type for DaCe
  - Complete with unit and integration tests
- NPBench suite for benchmarking SR performance and error
- Initial investigation showing limitation of SR for complex simulations

Future Work: Precision & Tools

- Explore SR benefits in half-precision and lower formats in greater detail
  - SR could offer immediate accuracy benefits for applications transitioning to half-precision or lower
  - In particular FP16, BFloat16, MiniFloat, TensorFloat-32
- Extend DaCe to fully support low-precision formats to avoid hybrid setups

Future Work: SR Behavior & Applications

- Test different RNG buffer sizes and cycling strategies
    - Investigate why buffering performs so poorly
- Evaluate advanced RNGs such as Permuted Congruential Generator for SR effectiveness
- Investigate why SR underperforms in ICON velocity tendencies
    - Run over a greater time horizon or loop it
- Experiment with additional ICON modules