ARGONNE | ATPESC 2026 | Extreme-Scale Computing

Verificarlo: Debugging and optimising floating-point calculations

Yohan Chatelain

camh

Krembil Centre for Neuroinformatics

What are the root causes of these industrial accidents?

Floating-point calculations!

Ariane 5

$370 millions (1996)

Overflow

[Wikipedia] [ESA]

Vancouver Stock Exchange

Index drop -50% (1983)

Rounding error

[Insider]

Intel Pentium Bug

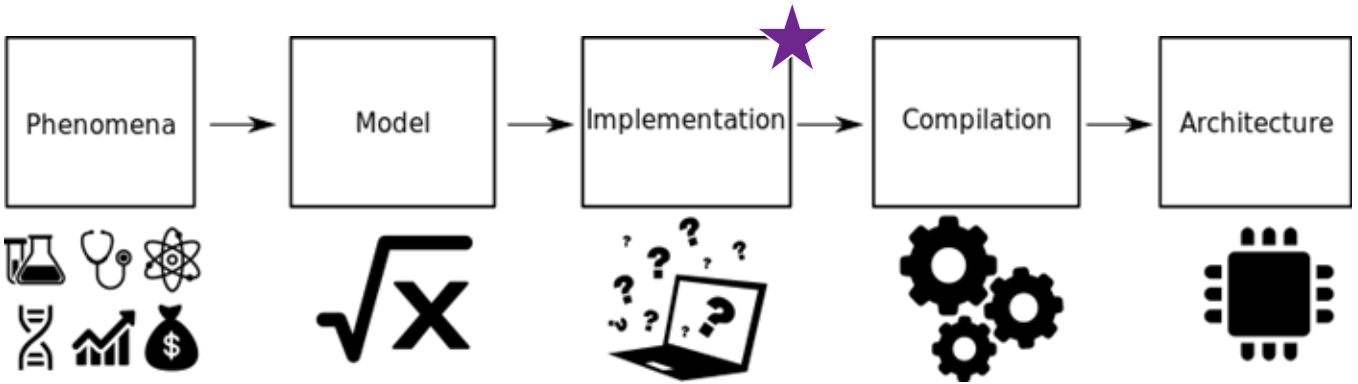
$475 millions (1994)

Bug in floating-point division

[Wikipedia]

For more stories on this subject:
Forum on Risks to the Public in Computers and Related Systems
ACM Committee on Computers and Public Policy.

To build a numerically robust scientific pipeline is complex



Numerical stability is a recurring topic in HPC

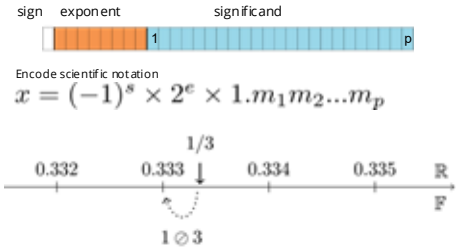
- Numerical errors at every step
- Large and complex codes
- Long-lifespan code (10~30 years)
- Hardware and software updates
- Architectural evolution cycle
- Highly heterogeneous architectures

How does numerical variability arise?

Floating point arithmetic (encoding real number in computer) has **limited precision,** is **not associative** and **has deterministic rounding**. This leads to:

- Rounding-error, catastrophic cancellation, overflow, underflow
- Snowball effect: errors accumulate
- OS, software and hardware updates amplify this snowball effect
- Ill-conditioned problems are sensitive to these perturbations

IEEE-754 standard floating-point model

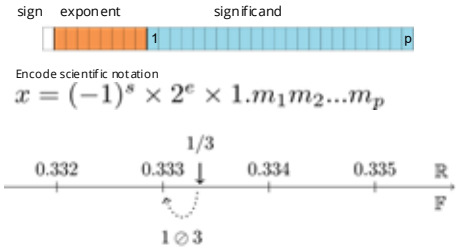


How does numerical variability arise?

Floating point arithmetic (encoding real number in computer) has **limited precision,** is **not associative** and **has deterministic rounding**. This leads to:

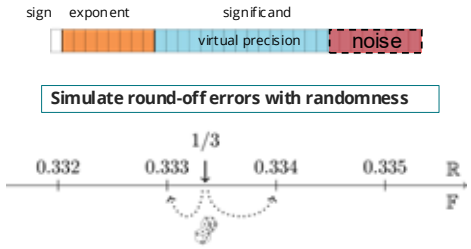
- Rounding-error, catastrophic cancellation, overflow, underflow
- Snowball effect: errors accumulate
- OS, software and hardware updates amplify this snowball effect
- Ill-conditioned problems are sensitive to these perturbations

IEEE-754 standard floating-point model



Monte Carlo Arithmetic (MCA) [Parker97]

part of Stochastic Arithmetic family



Metrology

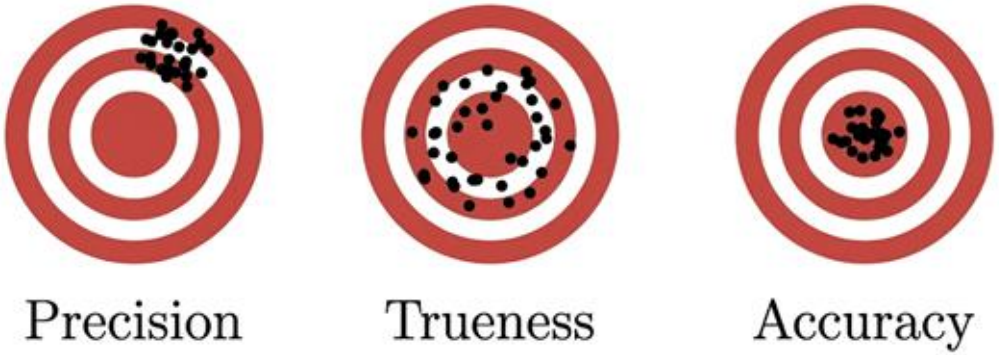


FIGURE 2.3 – Illustration of the concepts of precision, trueness, and accuracy in the context of numerical computation. Each black point represents the result of an execution. These points coincide if the execution is deterministic. Accuracy is a strong property that requires both precision – that the empirical results are not too far from each other – and trueness – that the empirical results is centered on the theoretical result.

- Accuracy and trueness are hard to measure without ground truths
- Precision does not require a reference

Linear 2x2 system with Cramer's rule

```c
/* Cramer's rule for a 2x2 system. */
double det = a11 * a22 - a12 * a21;
double x0  = (b1 * a22 - a12 * b2) / det;
double x1  = (a11 * b2 - b1 * a21) / det;
```

Ill-conditioned system

- Solves Ax=b using Cramer's rule

$$\begin{pmatrix} 0.2161 & 0.1441 \\ 1.2969 & 0.8648 \end{pmatrix} x = \begin{pmatrix} 0.1440 \\ 0.8642 \end{pmatrix}$$

- Exact solution is **x=(2 -2)**
- Condition number
  ~2.5x10^8 → 28 bits
- 1000 MCA repetitions per precision

Significant digits

$$s = -\log_2\left(\frac{\hat{\sigma}}{|\hat{\mu}|}\right)$$

- Signal to noise ratio magnitude
- Rigorous extension in [Sohier21]

binary32 (p=24) → x=(1.50000000000000000 -1.50000000000000000)
binary64 (p=53) → x=(1.99999999583666366 -1.99999999722444244)

Is increasing precision always enough?

Muller's sequence:

- To which value converge this sequence?

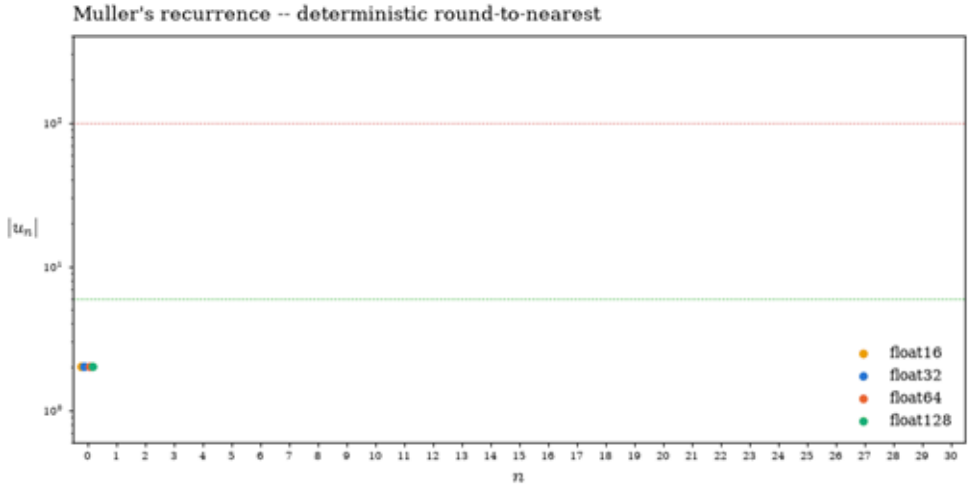
$$u_n = \begin{cases} u_0 = 2 \\ u_1 = -4 \\ u_{n+1} = 111 - \dfrac{1130}{u_n} + \dfrac{3000}{u_n u_{n-1}} \end{cases}$$

Is increasing precision always enough?

Muller's sequence:

- To which value converge this sequence?

$$u_n = \begin{cases} u_0 = 2 \\ u_1 = -4 \\ u_{n+1} = 111 - \dfrac{1130}{u_n} + \dfrac{3000}{u_n u_{n-1}} \end{cases}$$

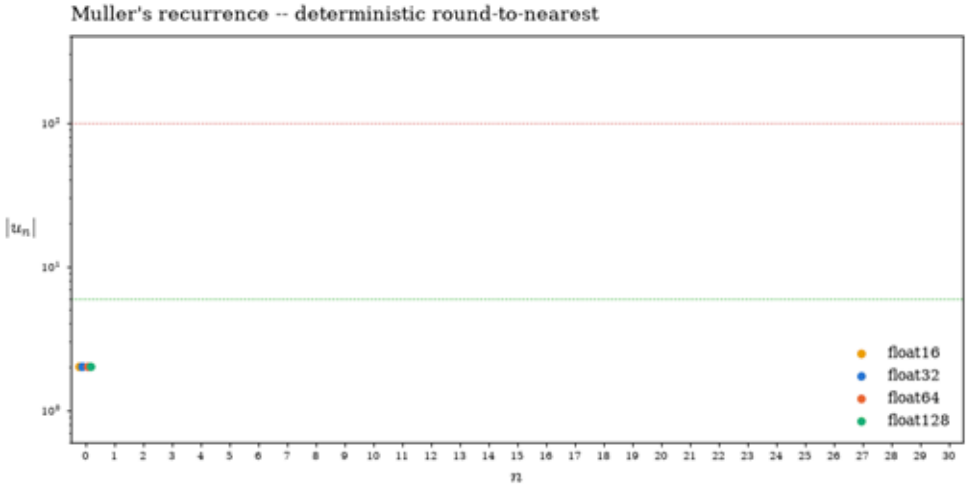


Is increasing precision always enough? NO

Muller's sequence:

- To which value converge this sequence?
  - Exact value is **6** ...
  - but it converges to **100** for any **finite precision arithmetic**

$$u_n = \begin{cases} u_0 = 2 \\ u_1 = -4 \\ u_{n+1} = 111 - \dfrac{1130}{u_n} + \dfrac{3000}{u_n u_{n-1}} \end{cases}$$



The recurrence converges to 6 in exact arithmetic. In floating-point arithmetic, however, rounding errors perturb the computed trajectory toward the spurious limit 100, one of the other roots of the fixed-point polynomial:

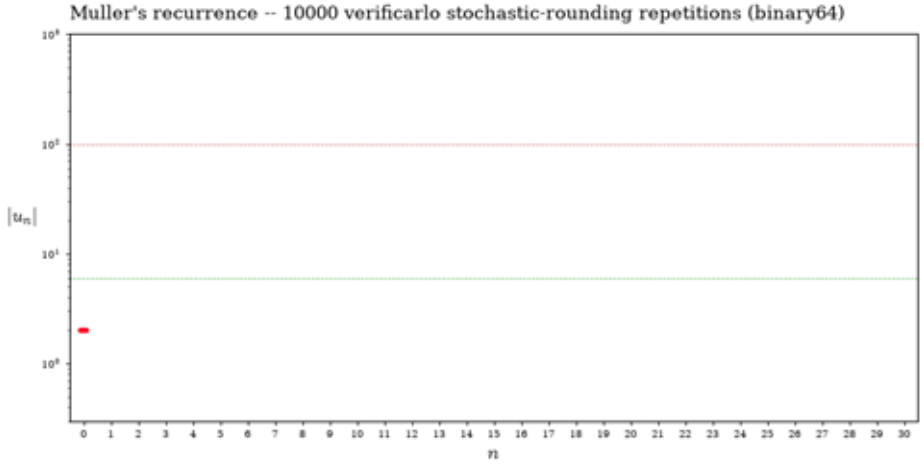
$$t^3 - 111t^2 + 1130t - 3000 = (t-5)(t-6)(t-100)$$

Is increasing precision always enough? NO

Muller's sequence:

- To which value converge this sequence?
  - Exact value is **6** ...
  - but it converges to **100** for any **finite precision arithmetic**
  - using MCA helps visualising the **bifurcation**

$$u_n = \begin{cases} u_0 = 2 \\ u_1 = -4 \\ u_{n+1} = 111 - \dfrac{1130}{u_n} + \dfrac{3000}{u_n u_{n-1}} \end{cases}$$



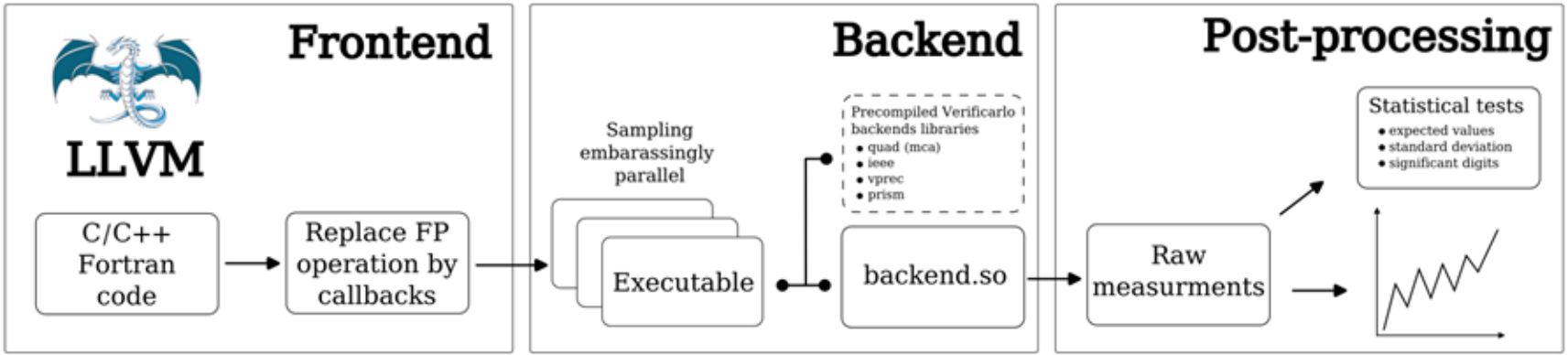
The recurrence converges to 6 in exact arithmetic. In floating-point arithmetic, however, rounding errors perturb the computed trajectory toward the spurious limit 100, one of the other roots of the fixed-point polynomial:

$$t^3 - 111t^2 + 1130t - 3000 = (t-5)(t-6)(t-100)$$

Verificarlo

Verificarlo architecture

github.com/verificarlo/verificarlo

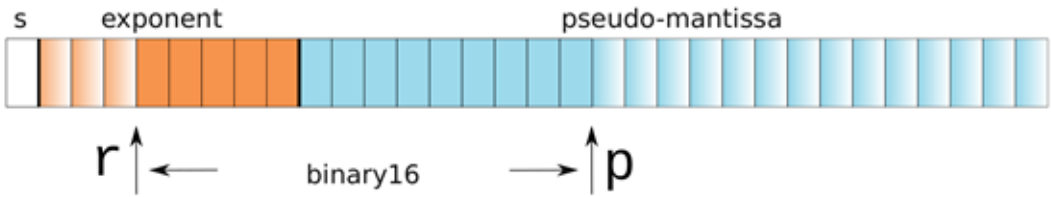


veriTracer PyTracer
significantdigits

- Compiler based on clang (C,C++) / flang (Fortran)
- Replaces floating-point operations with generic function calls
- Instrumentation occurs before code generation (catching compiler optimizations effects)
- Selection of alternative floating-point models (backend) at runtime
- Linux / x86-64 / AArch64 architectures

[Denis15, Chatelain18, Chatelain19, Sohier21, Pepe26.1]

VPREC: Evaluating reduced-precision dynamically



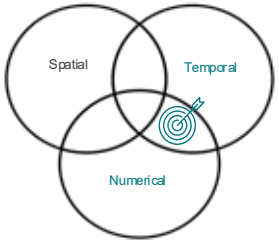
By setting $r = 5$ and $p = 10$, VPREC simulates a *binary*16 embedded inside a *binary*32

VPREC (Virtual PRECision)

- Emulates reduced precision in software (faithful rounding)
- Fine granularity (exponent & mantissa bits)
- Low overhead (3~15x)
- Integrated into Verificarlo and Verrou
- Allows testing new formats dynamically (without recompilation)

[Chatelain19]

Coupling VPREC with temporal analysis



Strategies for finding the best configuration, combining temporal analysis and reduced precision

k	x_{k+1}	s_k^{10}	s_k^2
0	0.0690266447076745	0.11	0.37
1	0.1230846130203958	0.21	0.70
2	0.1985746566605835	0.43	1.43
3	0.2732703639721015	0.84	2.79
4	0.3119369815109966	1.79	5.95
5	0.3181822938100336	3.40	11.3
6	0.3183098350392471	6.79	22.6
7	0.3183098861837825	13.6	45.2
8	0.3183098861837907	15.6	51.8
9	0.3183098861837907	15.6	51.8

```
double newton(double x0) {
  double x_k;
  double x_k1=x0;
  double b=PI;
  do {
    x_k  = x_k1;
    x_k1 = x_k*(2-b*x_k);
  } while (fabs((x_k1-x_k)/x_k)
                    >= 1e-15);
  return x_k1;
}
```

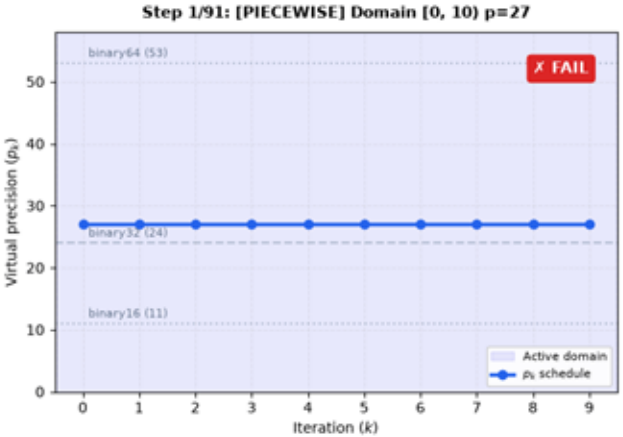
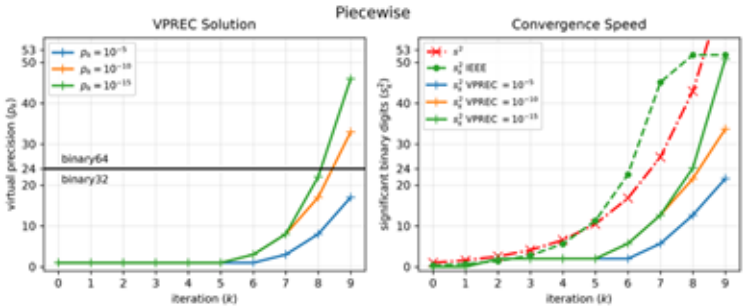


Table: (*left*) Convergence speed of Newton-Raphson for the computation of the inverse of π using the IEEE-754 `binary64` format (*right*). Highlighted digits in red are non significant

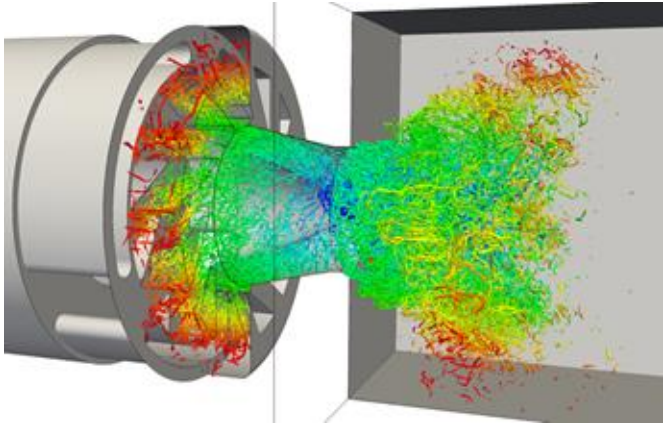


[Chatelain19]

Exploration on an industrial use case: YALES2

Solver for two-phase combustion
- From primary atomization to pollutant prediction

Deflated Preconditioned Conjugate Gradient
- Fine grid is geometrically projected on a coarse grid
- CG is applied on the coarse grid and on the fine grid



A-DEF2 using preconditioner M^{-1} and deflation matrix W.

- In blue the coarse grid and green the fine grid
- deflated part = blue, entire application = green + blue

Require : A, b, M^{-1}, W
initialization
for k = 0 , 1 , . . . until required convergence **do**

$$\alpha_{k+1} = \frac{r_k^T w_k}{p_k^T A p_k}$$

$$x_{k+1} = x_k + \alpha_{k+1} p_k$$

$$r_{k+1} = r_k - \alpha_{k+1} A p_k$$

Solve $\hat{A} d_{k+1} = W^T (A M^{-1} - I) r_{k+1}$

$$w_{k+1} = M^{-1} r_{k+1} - W d_{k+1}$$

$$\beta_{k+1} = \frac{r_k^T w_{k+1}}{r_k^T w_k}$$

$$p_{k+1} = w_{k+1} + \beta_{k+1} p_k$$

end for

CERFACS
CENTRE EUROPÉEN DE RECHERCHE ET DE FORMATION AVANCÉE EN CALCUL SCIENTIFIQUE

Mixed-precision improves performance

Reduced precision helps to:

- Speed up calculations
- Reduce memory footprint
- Reduce the volume of exchanged communications (MPI)

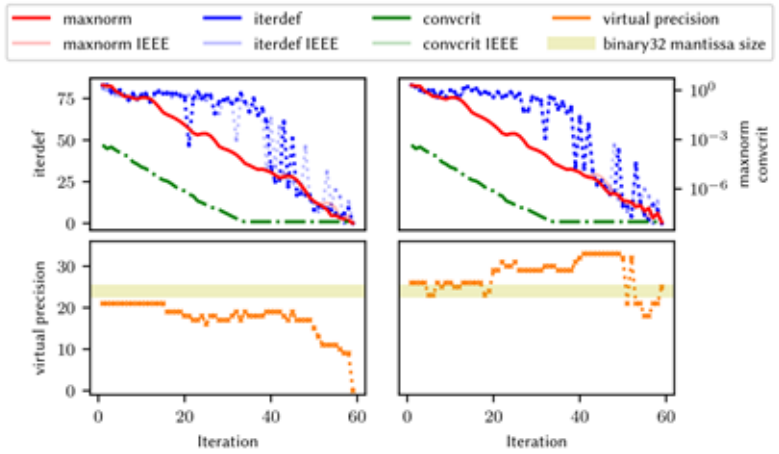
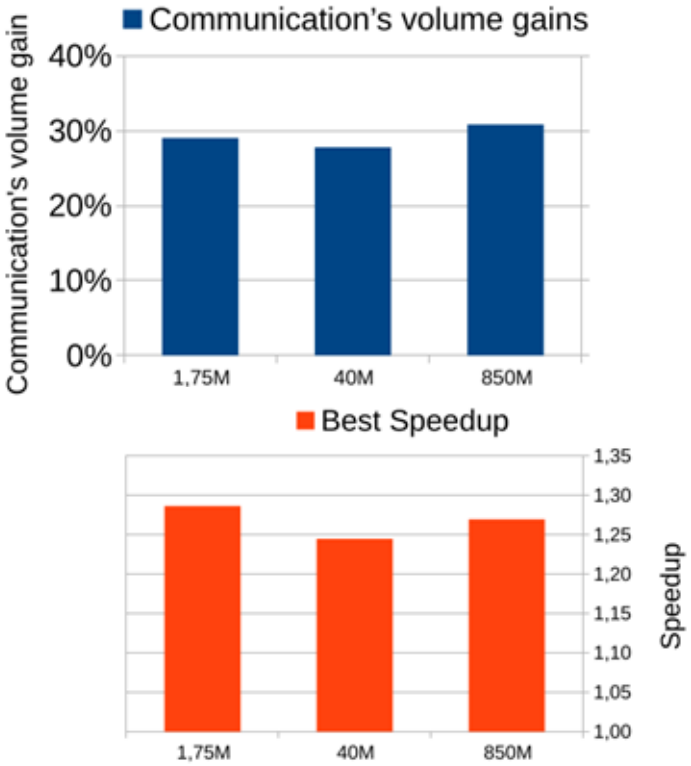


Fig. 5. Adaptive precision searching on YALES2's DPCG with the deflated part (*left*) and the entire code (*right*). On both plots, we can see that our reduced precision solution follows the reference IEEE convergence profile.



[Chatelain19]

Python code instrumentation

verificarlo + NumPy SciPy scikit learn PyTorch → Fuzzy docker

github.com/verificarlo/fuzzy

Université Concordia University

[Kiar20,Pepe26]

PRISM : Probabilistic Rounding with Instruction Set Management

- Low-overhead implementation of Stochastic Rounding (SR) [Fasi21]
- Vectorized implementation (based on Google Highway)
- Selects "best" implementation at runtime
  - x86-64: SSE4.2, AVX, AVX2, AVX-512, ...
- Uses direct instrumentation for better performance
  - No backend change possible at runtime (VFC_BACKENDS)
  - Virtual precision selectable

→ Makes SR + Pytorch tractable on CPU

Fuzzy PyTorch docker

github.com/big-data-lab-team/fuzzy-pytorch

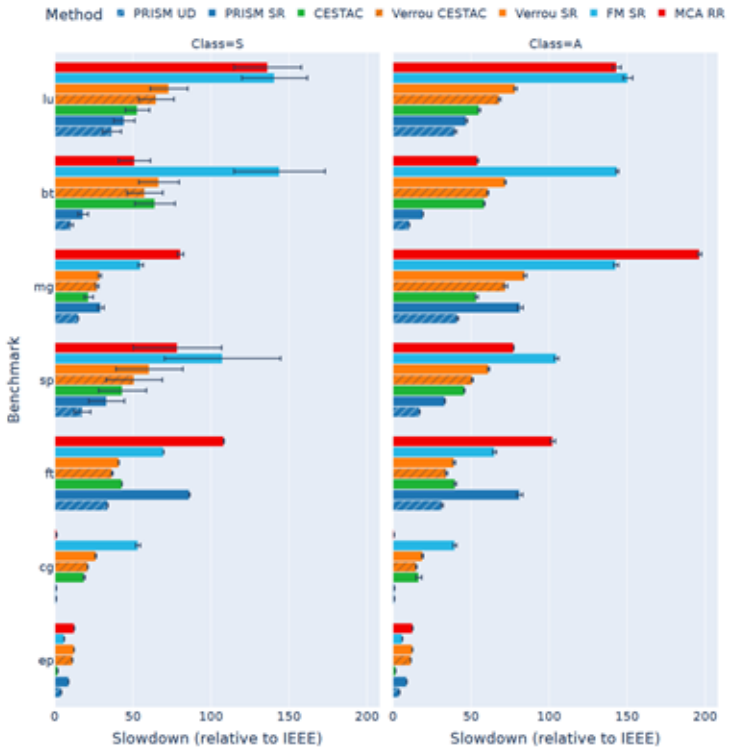
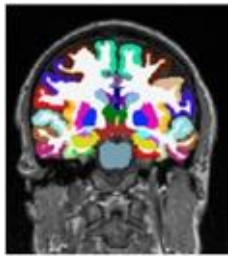


Figure 2: Comparison of slowdowns across numerical variability analysis tools for NAS Parallel Benchmarks on dataset S and A, relative to the IEEE binary64 baseline.

PRISM: SR 50x faster than SOTA tool

MNIST [LeCun98]
Inference execution time for 10,000 images (single-thread)



FastSurfer [Henschel20]
U-net type CNN
Complete brain segmentation
Execution time per slice

WavLM

WavLM [Chen22]
Speech recognition
Parkinson's detection

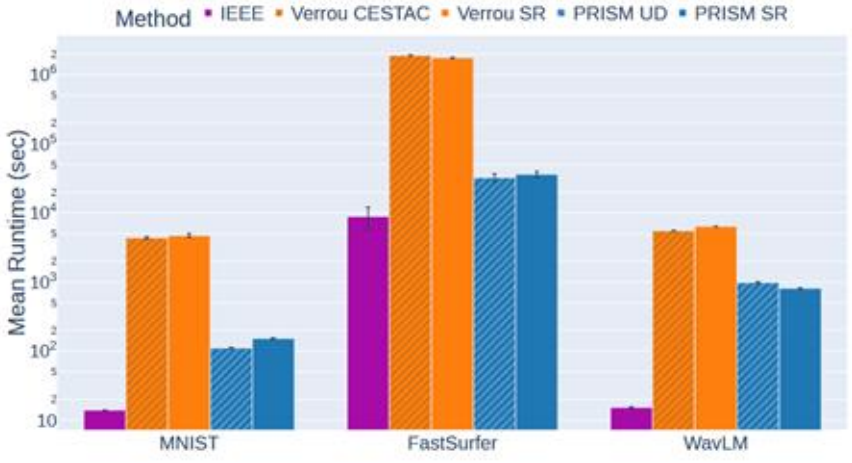
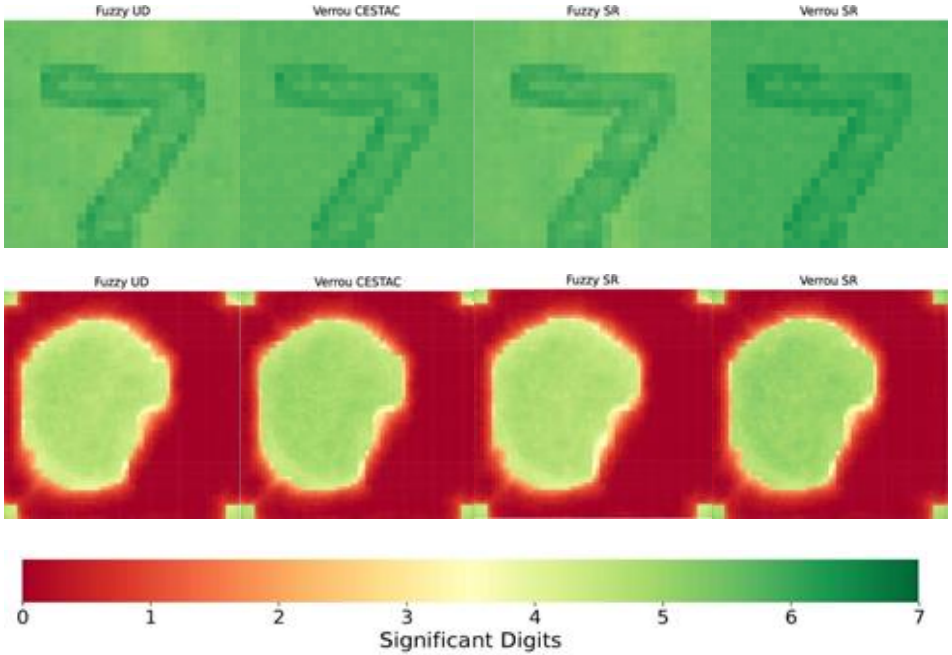


Figure 3: Comparison of instrumentation runtimes across DL models

Comparison of the numerical quality of embeddings



Test cases:

- MNIST: First convolutional layer. (Top)
- FastSurfer: Output of the 2nd decoder block. (Bottom)

Results:

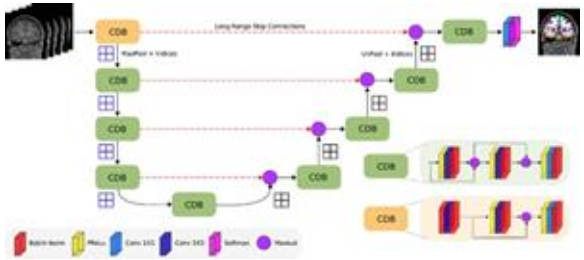
- MNIST: stable embeddings
- FastSurfer: Low quality in the background of the embeddings but stable final outputs

Origin of instability (FastSurfer):

- Localized outside the brain region
- Caused by unstable indices generated by max-pooling
- Post-processing masks the instability in the background

Conclusion:

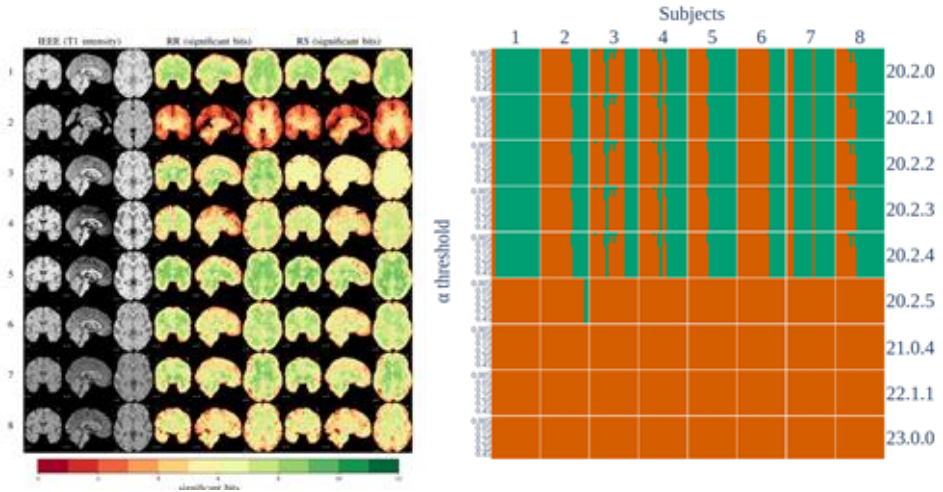
- Consistent results between Fuzzy PyTorch and Verrou.



[Pepe26.2]

Ensuring numerical quality through software updates

To use numerical variability to quantify acceptable bound on results:
- Proof-of-concept on neuroimaging (fMRIPrep)
- Generalizable to other domains



Automatic detection of a **significant change** introduced with the LTS 20.2.5 version

Université de Montréal | Stanford University | fMRIPrep a robust preprocessing pipeline for task-based and resting-state fMRI data

[Chatelain24]

How to identify numerical instabilities?

veriTracer

JavaIDE, Damien Thenot's internship

github.com/verificarlo/verificarlo/tree/veritracer

[Chatelain18]

PyTracer

github.com/big-data-lab-team/pytracer

[Chatelain22]

To trace precisely the numerical instability

Same function, different behavior

- Trace of Simpson's integral (abinit)
- 1 color = 1 call-site path (CSP)
- Compensated version improves precision (Dot2, [Ogita02])
- 1 CSP has error's reentrance

Fine-grained level visualization

- facial recognition example (scipy)
- randomized SVD with segvd
- precision drops for smaller values
- potential for dimensionality reduction

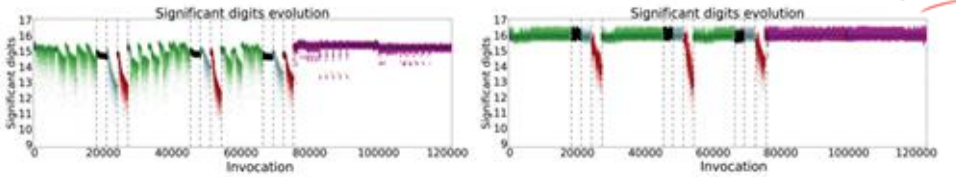
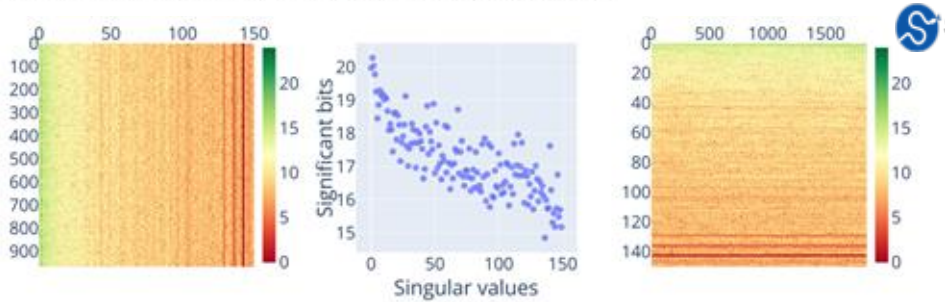


Figure 8: Original

Figure 9: Compensated

Figures show profiles produced by VeriTracer of `simp_gen`. Several accuracy losses present in the original version (fig. 8) are improved by the compensated algorithm Dot2 (fig. 9). Each color maps one of the 31 distinct CSPs. Dot2 improved 30 out of 31 CSPs.Vertical dashes separate the main CSPs.



(a) Left matrix U of the SVD decomposition

(b) Singular values Σ of the SVD decomposition

(c) Right matrix V of the SVD decomposition

Fig. 6. Precision of the `randomized_svd` function in the `face_recognition` example using the `gesvd` SVD method and RR. Figs. 6(a), 6(b), and 6(c) show respectively the values U, Σ and V. Result precision is lower for smaller singular values and this loss of precision translates to singular vectors. It would be interesting to investigate the use of numerical precision for dimensionality reduction.

Conclusion

Three takeaways

- Numerical error is *measurable*:
  - stochastic arithmetic turns "do I trust this result?" into an observable quantity, without needing a ground truth
- More precision is not a silver bullet:
  - Muller's sequence converges to the wrong value at *every* finite precision
  - ill-conditioning has to be diagnosed and not out-precisioned
- The same instrumentation answers three different questions:
  - **is it right?** (SR)
  - **how little precision do I need?** (VPREC)
  - **where does it break?** (VeriTracer, PyTracer)

What the toolchain gives you today

- One compilation, several numerical models selected at *runtime*
  - no recompile to switch backend or virtual precision
- Full-stack coverage:
  - C/C++/Fortran (Verificarlo)
  - Python & libm (Fuzzy)
  - PyTorch (PRISM / Fuzzy PyTorch)
- Overheads compatible with real workloads:
  - VPREC 3-15×
  - PRISM ~50× faster than state-of-the-art SR
  - SR on AI models is now tractable on CPU

Demonstrated end-to-end

- YALES2 → mixed precision cuts time, memory footprint, and communication volume on a production combustion solver
- FastSurfer → instability localized to max-pooling index selection, outside the brain mask; final segmentation stable
- fMRIPrep → automatic detection of a significant numerical change introduced at LTS 20.2.5

verificarlo

Pablo de Oliveira Castro

Eric Petit

Yohan Chatelain

UVSQ université PARIS-SACLAY

intel

Université Concordia University

camh Krembil Centre for Neuroinformatics

github.com/verificarlo/verificarlo

cnrs cea Université Perpignan Via Domitia CERFACS Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique Stanford University Université de Montréal University of Washington

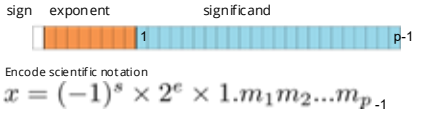
Bibliography

➜[Parker97] Parker, D. S. (1997). *Monte Carlo arithmetic: exploiting randomness in floating-point arithmetic*. University of California (Los Angeles). Computer Science Department.
➜[Sohier21] Sohier, D., Castro, P. D. O., Févotte, F., Lathuilière, B., Petit, E., & Jamond, O. (2021). Confidence intervals for stochastic arithmetic. *ACM Transactions on Mathematical Software (TOMS)*, *47*(2), 1-33.
➜[Denis15] Denis, C., Castro, P. D. O., & Petit, E. (2016, July). Verificarlo: Checking Floating Point Accuracy through Monte Carlo Arithmetic. In *2016 IEEE 23nd Symposium on Computer Arithmetic (ARITH)* (pp. 55-62). IEEE.
➜[Chatelain18] Chatelain, Y., Castro, P. D. O., Petit, E., Defour, D., Bieder, J., & Torrent, M. (2018, June). VeriTracer: Context-enriched tracer for floating-point arithmetic analysis. In *2018 IEEE 25th Symposium on Computer Arithmetic (ARITH)* (pp. 61-68). IEEE.
➜[Chatelain19] Chatelain, Y., Petit, E., de Oliveira Castro, P., Lartigue, G., & Defour, D. (2019, August). Automatic exploration of reduced floating-point representations in iterative methods. In *European conference on parallel processing* (pp. 481-494). Cham: Springer International Publishing.
➜[Pepe26.1] Gonzalez-Pepe, I., Akhaddar, H., Glatard, T., & Chatelain, Y. (2026). Fuzzy PyTorch: Rapid Numerical Variability Evaluation for Deep Learning Models. *arXiv preprint arXiv:2605.25991*.
➜[Chatelain22] Chatelain, Y., Sao Young, N. Y., Kiar, G., & Glatard, T. (2022). Pytracer: Automatically profiling numerical instabilities in python. *IEEE Transactions on Computers*, *72*(6), 1792-1803.
➜[Ogita02] Ogita, T., Rump, S. M., & Oishi, S. I. (2005). Accurate sum and dot product. *SIAM Journal on Scientific Computing*, *26*(6), 1955-1988.
➜[Fasi21] Fasi, M., & Mikaitis, M. (2021). Algorithms for stochastically rounded elementary arithmetic operations in IEEE 754 floating-point arithmetic. *IEEE Transactions on Emerging Topics in Computing*, *9*(3), 1451-1466.
➜[LeCun98] LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278-2324.
➜[Henschel20] Henschel, L., Conjeti, S., Estrada, S., Diers, K., Fischl, B., & Reuter, M. (2020). Fastsurfer-a fast and accurate deep learning based neuroimaging pipeline. *NeuroImage*, *219*, 117012.
➜[Chen22] Chen, S., Wang, C., Chen, Z., Wu, Y., Liu, S., Chen, Z., ... & Wei, F. (2022). Wavlm: Large-scale self-supervised pre-training for full stack speech processing. *IEEE Journal of Selected Topics in Signal Processing*, *16*(6), 1505-1518.
➜[Pepe26.2] Gonzalez-Pepe, I., Sivakolunthu, V., Fortin, J., Chatelain, Y., & Glatard, T. (2026). Conservative & Aggressive NaNs Accelerate U-Nets for Neuroimaging. *arXiv preprint arXiv:2601.17180*.
➜[Chatelain24] Chatelain, Y., Tetrel, L., Markiewicz, C. J., Goncalves, M., Kiar, G., Esteban, O., ... & Glatard, T. (2024). A numerical variability approach to results stability tests and its application to neuroimaging. *IEEE Transactions on Computers*, *74*(1), 200-209.

Supplementary

IEEE-754 floating-point model

IEEE-754 standard floating-point model



IEEE 754 Binary Interchange Formats

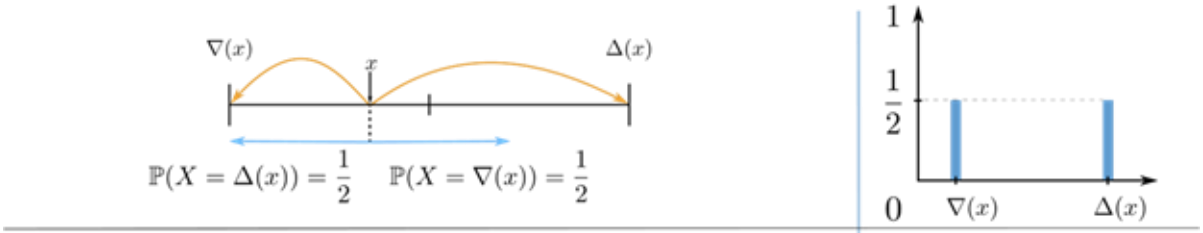
Format	Total bits	Sign	Exponent	Fraction	Precision p	Bias	Decimal digits
binary16 (half)	16	1	5	10	11	15	≈ 3.31
binary32 (single)	32	1	8	23	24	127	≈ 7.22
binary64 (double)	64	1	11	52	53	1023	≈ 15.95
binary128 (quadruple)	128	1	15	112	113	16383	≈ 34.02

Format	Smallest positive subnormal	Smallest positive normal	Largest finite value
binary16	$2^{-24} \approx 5.9605 \times 10^{-8}$	$2^{-14} \approx 6.1035 \times 10^{-5}$	65504
binary32	$2^{-149} \approx 1.4013 \times 10^{-45}$	$2^{-126} \approx 1.1755 \times 10^{-38}$	$\approx 3.4028 \times 10^{38}$
binary64	$2^{-1074} \approx 4.9407 \times 10^{-324}$	$2^{-1022} \approx 2.2251 \times 10^{-308}$	$\approx 1.7977 \times 10^{308}$
binary128	$2^{-16494} \approx 6.4752 \times 10^{-4966}$	$2^{-16382} \approx 3.3621 \times 10^{-4932}$	$\approx 1.1897 \times 10^{4932}$

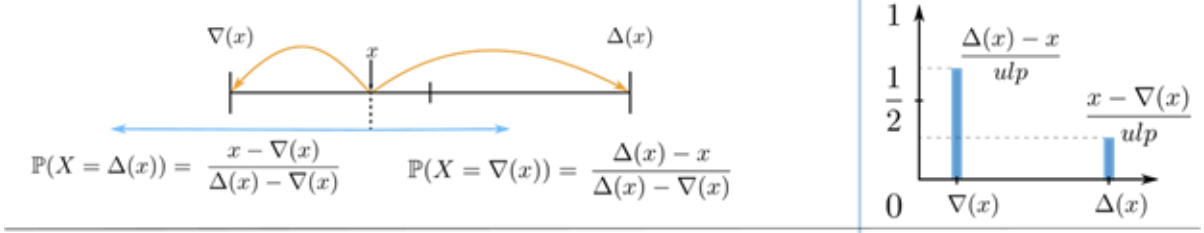
For normal numbers, p includes the implicit leading significand bit. Exponent field 0 encodes zeros and subnormals; the all-ones exponent encodes infinities and NaNs.

Stochastic rounding methods

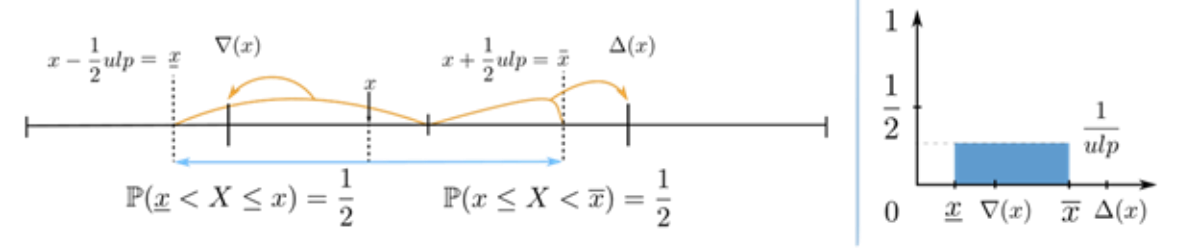
CESTAC



SR-nearess



MCA



Google's Highway Library

- C++ library with a DSL to express vector instructions
- Static mode with code generation for target architectures
- Dynamic mode with automatic selection of the best architecture at runtime
- Wide range of supported architectures and ISAs:
  - Arvmv7+ : Neon and SSE
  - IBM Z : z14, z15
  - POWER : PPC8, PPC9, PPC10
  - RISC-V : RVV
  - WebAssembly : WASM
  - x86 : SSE2, SSSE3, SSE4, AVX2, AVX3 (AVX-512F), AVX3_DL, AVX3_ZEN4, AVX3_SPR

Very good compromise between performance, portability, and modularity!

```
template <class D, class V = hn::VFromD<D>, typename T = hn::TFromD<D>>
HWY_FLATTEN void twosum(const D d, const V a, const V b, V &sigma, V &tau) {
  dbg::debug_msg("\n[twosum] START");
  dbg::debug_vec(d, "[twosum] a", a);
  dbg::debug_vec(d, "[twosum] b", b);

  sigma = hn::Add(a, b);
  const auto a_p = hn::Sub(sigma, b);
  const auto b_p = hn::Sub(sigma, a_p);
  const auto d_a = hn::Sub(a, a_p);
  const auto d_b = hn::Sub(b, b_p);
  tau = hn::Add(d_a, d_b);
  tau = hn::IfThenElseZero(hn::IsFinite(sigma), tau);

  dbg::debug_vec(d, "[twosum] sigma", sigma);
  dbg::debug_vec(d, "[twosum] tau", tau);
  dbg::debug_msg("[twosum] END\n");
}
```

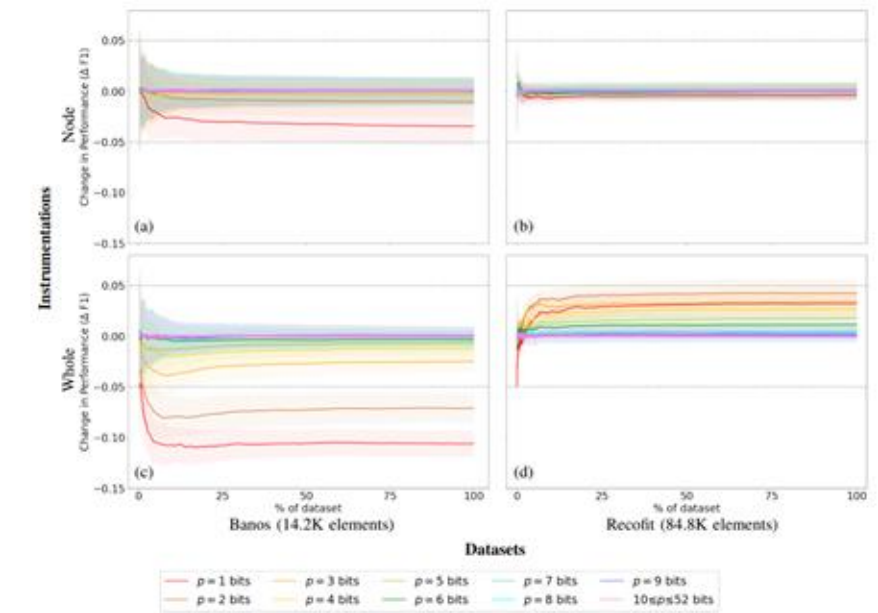


Fig. 1: Difference between the F1 scores obtained at reduced and double precision ($p = 52$ bits) with 3.0 MB of memory and 5 trees. Negative values indicate that the instrumented implementation performed worse than the original. Color shades represent the standard deviation in the corresponding Mondrian Forest. The grey lines at -0.05, 0.05 are for reference only. The 7 ordering of data points tested in Banos performed similarly.

Reducing numerical precision preserves classification accuracy in Mondrian Forests
Marc Vicuna, Martin Khannouz, Gregory Kiar, **Yohan Chatelain**, Tristan Glatard
2021 IEEE International Conference on Big Data (Big Data)