Low- and Mixed-Precision Floating-Point Hardware: Algorithms, Error analysis, and Standardisation

Mantas Mikaitis

School of Computer Science, University of Leeds, Leeds, UK

Mathematics Seminar Series, School of Computing and Mathematical Sciences,
University of Leicester, Leicester, UK
12 March, 2025
Slides available: `https://mmikaitis.github.io`

IEEE 754: Most important standard in computing history

Standard for binary and decimal fixed-precision arithmetic

Defines subsets of reals, their encoding in memory, conversion and arithmetic behaviour, rounding, exception handling, and more. **Concept of correct rounding**.

Released in 1985, revised in

- 2008
- 2019 (active)
- 2029 (work in progress)

We are participating in IEEE 754-2029

- Fortnightly meetings, discussion on the mailing list, thoroughly reading the 2019 revision and raising issues.
- Working group: international, many members work in computing industry.

Floating-point arithmetic, main tools

A floating-point system $\mathbb{F} \subset \mathbb{R}$ is described with $\beta, t, e_{min}, e_{max}$ with elements

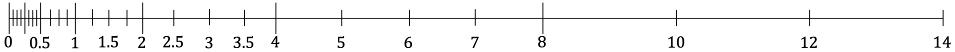
$$x = \pm m \times \beta^{e-t+1}.$$

Virtually all computers have $\beta = 2$ (binary FP).

Here t is precision, $e_{min} \leq e \leq e_{max}$ an exponent, $m \leq \beta^t - 1$ a significand $(m,\ t,\ e \in \mathbb{Z})$.

Toy FP system

Below: the positive numbers in $F(\beta = 2, t = 3, e_{min} = -2, e_{max} = 3)$.



Floating-point arithmetic, main tools

Standard model [Higham, 2002]

Given $x, y \in \mathbb{R}$ that lie in the range of $\mathbb{F}$ it can be shown that

$$\mathrm{fl}(x \ \mathrm{op}\ y) = (x \ \mathrm{op}\ y)(1+\delta), \quad |\delta| \le u,$$

where $u = 2^{-t}$, $\mathrm{op} \in \{+, -, \times, \div\}$ and **round-to-nearest** mode.

Building error bounds: small example

Rounding errors δ accumulate. For example, consider computing $s = x_1 y_1 + x_2 y_2 + x_3 y_3$.

We compute $\widehat{s}$ with

$$\begin{aligned}\widehat{s} &= \Big(\big(x_1 y_1 (1+\delta_1) + x_2 y_2 (1+\delta_2)\big)(1+\delta_3) + x_3 y_3 (1+\delta_4)\Big)(1+\delta_5) \\ &= x_1 y_1 (1+\delta_1)(1+\delta_3)(1+\delta_5) + x_2 y_2 (1+\delta_2)(1+\delta_3)(1+\delta_5) + x_3 y_3 (1+\delta_4)(1+\delta_5).\end{aligned}$$

Therefore we compute a solution for the inputs *perturbed at most by* $\prod_{i=1}^{n}(1+\delta_i)$.

Worst case backward error bound

$\prod_{i=1}^{n}(1+\delta_i) = 1 + \theta_n, \quad |\theta_n| \leq \gamma_n$, with $\gamma_n = \frac{nu}{1-nu}$ and assuming $nu < 1$.

To simplify, we say worst-case error growth is $\mathcal{O}(nu)$.

IEEE floating-point standardisation work: IEEE P3109

Standard for Arithmetic Formats for Machine Learning

New IEEE standard for computer arithmetic for AI is in progress. We are also actively participating in it: meeting fortnightly.

Standardises:

- Small floating-point subsets of reals,
- encoding in 8-bit words,
- rounding behaviour,
- arithmetic operations needed in AI workloads,
- conversion to/from 754,
- exception handling.

Interim report available: `http://bit.ly/42gPWcy`

IEEE floating-point standardisation work: IEEE P3109

Current draft outlines these key aspects:

- Defines formats as binary$K$pP with $K = 8$ and $P = \{1, 2, 3, 4, 5, 6, 7\}$.
- No -0 (would have been 0x80)
- Only one NaN (0x80) - note IEEE 754 numbers have many bit patterns for NaNs.
- $\pm\infty$ 0x7F and 0xFF.
- Saturation mode: on overflow, return maximum finite value.

Format	minSubnormal	maxSubnormal	minNormal	maxNormal
binary8p1	N/A	N/A	1×2^{-62}	1×2^{63}
binary8p2	1×2^{-32}	1×2^{-32}	1×2^{-31}	1×2^{31}
binary8p3	1×2^{-17}	$3/2 \times 2^{-16}$	1×2^{-15}	$3/2 \times 2^{15}$
binary8p4	1×2^{-10}	$7/4 \times 2^{-8}$	1×2^{-7}	$7/4 \times 2^{7}$
binary8p5	1×2^{-7}	$15/8 \times 2^{-4}$	1×2^{-3}	$15/8 \times 2^{3}$
binary8p6	1×2^{-6}	$31/16 \times 2^{-2}$	1×2^{-1}	$31/16 \times 2^{1}$
binary8p7	1×2^{-6}	$63/32 \times 2^{-1}$	1×2^{0}	$63/32 \times 2^{0}$

We can write down all numbers in a particular binary8pP set on one page. $\rightarrow$

IEEE P3109: format on one page (screenshot from the report)

C.4 Value Table: P4, emin = −7, emax = 7

0x00 = 0_0000_000 = 0.0	0x40 = 0_1000_000 = +0b1.000×2^0 = 1.0	0x80 = 1_0000_000 = NaN	0xc0 = 1_1000_000 = −0b1.000×2^0 = -1.0
0x01 = 0_0000_001 = +0b0.001×2^-7 = 0.0009765625	0x41 = 0_1000_001 = +0b1.001×2^0 = 1.125	0x81 = 1_0000_001 = −0b0.001×2^-7 = -0.0009765625	0xc1 = 1_1000_001 = −0b1.001×2^0 = -1.125
0x02 = 0_0000_010 = +0b0.010×2^-7 = 0.001953125	0x42 = 0_1000_010 = +0b1.010×2^0 = 1.25	0x82 = 1_0000_010 = −0b0.010×2^-7 = -0.001953125	0xc2 = 1_1000_010 = −0b1.010×2^0 = -1.25
0x03 = 0_0000_011 = +0b0.011×2^-7 = 0.0029296875	0x43 = 0_1000_011 = +0b1.011×2^0 = 1.375	0x83 = 1_0000_011 = −0b0.011×2^-7 = -0.0029296875	0xc3 = 1_1000_011 = −0b1.011×2^0 = -1.375
0x04 = 0_0000_100 = +0b0.100×2^-7 = 0.00390625	0x44 = 0_1000_100 = +0b1.100×2^0 = 1.5	0x84 = 1_0000_100 = −0b0.100×2^-7 = -0.00390625	0xc4 = 1_1000_100 = −0b1.100×2^0 = -1.5
0x05 = 0_0000_101 = +0b0.101×2^-7 = 0.0048828125	0x45 = 0_1000_101 = +0b1.101×2^0 = 1.625	0x85 = 1_0000_101 = −0b0.101×2^-7 = -0.0048828125	0xc5 = 1_1000_101 = −0b1.101×2^0 = -1.625
0x06 = 0_0000_110 = +0b0.110×2^-7 = 0.005859375	0x46 = 0_1000_110 = +0b1.110×2^0 = 1.75	0x86 = 1_0000_110 = −0b0.110×2^-7 = -0.005859375	0xc6 = 1_1000_110 = −0b1.110×2^0 = -1.75
0x07 = 0_0000_111 = +0b0.111×2^-7 = 0.0068359375	0x47 = 0_1000_111 = +0b1.111×2^0 = 1.875	0x87 = 1_0000_111 = −0b0.111×2^-7 = -0.0068359375	0xc7 = 1_1000_111 = −0b1.111×2^0 = -1.875
0x08 = 0_0001_000 = +0b1.000×2^-7 = 0.0078125	0x48 = 0_1001_000 = +0b1.000×2^1 = 2.0	0x88 = 1_0001_000 = −0b1.000×2^-7 = -0.0078125	0xc8 = 1_1001_000 = −0b1.000×2^1 = -2.0
0x09 = 0_0001_001 = +0b1.001×2^-7 = 0.0087890625	0x49 = 0_1001_001 = +0b1.001×2^1 = 2.25	0x89 = 1_0001_001 = −0b1.001×2^-7 = -0.0087890625	0xc9 = 1_1001_001 = −0b1.001×2^1 = -2.25
0x0a = 0_0001_010 = +0b1.010×2^-7 = 0.009765625	0x4a = 0_1001_010 = +0b1.010×2^1 = 2.5	0x8a = 1_0001_010 = −0b1.010×2^-7 = -0.009765625	0xca = 1_1001_010 = −0b1.010×2^1 = -2.5
0x0b = 0_0001_011 = +0b1.011×2^-7 = 0.0107421875	0x4b = 0_1001_011 = +0b1.011×2^1 = 2.75	0x8b = 1_0001_011 = −0b1.011×2^-7 = -0.0107421875	0xcb = 1_1001_011 = −0b1.011×2^1 = -2.75
0x0c = 0_0001_100 = +0b1.100×2^-7 = 0.01171875	0x4c = 0_1001_100 = +0b1.100×2^1 = 3.0	0x8c = 1_0001_100 = −0b1.100×2^-7 = -0.01171875	0xcc = 1_1001_100 = −0b1.100×2^1 = -3.0
0x0d = 0_0001_101 = +0b1.101×2^-7 = 0.0126953125	0x4d = 0_1001_101 = +0b1.101×2^1 = 3.25	0x8d = 1_0001_101 = −0b1.101×2^-7 = -0.0126953125	0xcd = 1_1001_101 = −0b1.101×2^1 = -3.25
0x0e = 0_0001_110 = +0b1.110×2^-7 = 0.013671875	0x4e = 0_1001_110 = +0b1.110×2^1 = 3.5	0x8e = 1_0001_110 = −0b1.110×2^-7 = -0.013671875	0xce = 1_1001_110 = −0b1.110×2^1 = -3.5
0x0f = 0_0001_111 = +0b1.111×2^-7 = 0.0146484375	0x4f = 0_1001_111 = +0b1.111×2^1 = 3.75	0x8f = 1_0001_111 = −0b1.111×2^-7 = -0.0146484375	0xcf = 1_1001_111 = −0b1.111×2^1 = -3.75
0x10 = 0_0010_000 = +0b1.000×2^-6 = 0.015625	0x50 = 0_1010_000 = +0b1.000×2^2 = 4.0	0x90 = 1_0010_000 = −0b1.000×2^-6 = -0.015625	0xd0 = 1_1010_000 = −0b1.000×2^2 = -4.0
0x11 = 0_0010_001 = +0b1.001×2^-6 = 0.017578125	0x51 = 0_1010_001 = +0b1.001×2^2 = 4.5	0x91 = 1_0010_001 = −0b1.001×2^-6 = -0.017578125	0xd1 = 1_1010_001 = −0b1.001×2^2 = -4.5
0x12 = 0_0010_010 = +0b1.010×2^-6 = 0.01953125	0x52 = 0_1010_010 = +0b1.010×2^2 = 5.0	0x92 = 1_0010_010 = −0b1.010×2^-6 = -0.01953125	0xd2 = 1_1010_010 = −0b1.010×2^2 = -5.0
0x13 = 0_0010_011 = +0b1.011×2^-6 = 0.021484375	0x53 = 0_1010_011 = +0b1.011×2^2 = 5.5	0x93 = 1_0010_011 = −0b1.011×2^-6 = -0.021484375	0xd3 = 1_1010_011 = −0b1.011×2^2 = -5.5
0x14 = 0_0010_100 = +0b1.100×2^-6 = 0.0234375	0x54 = 0_1010_100 = +0b1.100×2^2 = 6.0	0x94 = 1_0010_100 = −0b1.100×2^-6 = -0.0234375	0xd4 = 1_1010_100 = −0b1.100×2^2 = -6.0
0x15 = 0_0010_101 = +0b1.101×2^-6 = 0.025390625	0x55 = 0_1010_101 = +0b1.101×2^2 = 6.5	0x95 = 1_0010_101 = −0b1.101×2^-6 = -0.025390625	0xd5 = 1_1010_101 = −0b1.101×2^2 = -6.5
0x16 = 0_0010_110 = +0b1.110×2^-6 = 0.02734375	0x56 = 0_1010_110 = +0b1.110×2^2 = 7.0	0x96 = 1_0010_110 = −0b1.110×2^-6 = -0.02734375	0xd6 = 1_1010_110 = −0b1.110×2^2 = -7.0
0x17 = 0_0010_111 = +0b1.111×2^-6 = 0.029296875	0x57 = 0_1010_111 = +0b1.111×2^2 = 7.5	0x97 = 1_0010_111 = −0b1.111×2^-6 = -0.029296875	0xd7 = 1_1010_111 = −0b1.111×2^2 = -7.5
0x18 = 0_0011_000 = +0b1.000×2^-5 = 0.03125	0x58 = 0_1011_000 = +0b1.000×2^3 = 8.0	0x98 = 1_0011_000 = −0b1.000×2^-5 = -0.03125	0xd8 = 1_1011_000 = −0b1.000×2^3 = -8.0
0x19 = 0_0011_001 = +0b1.001×2^-5 = 0.03515625	0x59 = 0_1011_001 = +0b1.001×2^3 = 9.0	0x99 = 1_0011_001 = −0b1.001×2^-5 = -0.03515625	0xd9 = 1_1011_001 = −0b1.001×2^3 = -9.0
0x1a = 0_0011_010 = +0b1.010×2^-5 = 0.0390625	0x5a = 0_1011_010 = +0b1.010×2^3 = 10.0	0x9a = 1_0011_010 = −0b1.010×2^-5 = -0.0390625	0xda = 1_1011_010 = −0b1.010×2^3 = -10.0
0x1b = 0_0011_011 = +0b1.011×2^-5 = 0.04296875	0x5b = 0_1011_011 = +0b1.011×2^3 = 11.0	0x9b = 1_0011_011 = −0b1.011×2^-5 = -0.04296875	0xdb = 1_1011_011 = −0b1.011×2^3 = -11.0
0x1c = 0_0011_100 = +0b1.100×2^-5 = 0.046875	0x5c = 0_1011_100 = +0b1.100×2^3 = 12.0	0x9c = 1_0011_100 = −0b1.100×2^-5 = -0.046875	0xdc = 1_1011_100 = −0b1.100×2^3 = -12.0
0x1d = 0_0011_101 = +0b1.101×2^-5 = 0.05078125	0x5d = 0_1011_101 = +0b1.101×2^3 = 13.0	0x9d = 1_0011_101 = −0b1.101×2^-5 = -0.05078125	0xdd = 1_1011_101 = −0b1.101×2^3 = -13.0
0x1e = 0_0011_110 = +0b1.110×2^-5 = 0.0546875	0x5e = 0_1011_110 = +0b1.110×2^3 = 14.0	0x9e = 1_0011_110 = −0b1.110×2^-5 = -0.0546875	0xde = 1_1011_110 = −0b1.110×2^3 = -14.0
0x1f = 0_0011_111 = +0b1.111×2^-5 = 0.05859375	0x5f = 0_1011_111 = +0b1.111×2^3 = 15.0	0x9f = 1_0011_111 = −0b1.111×2^-5 = -0.05859375	0xdf = 1_1011_111 = −0b1.111×2^3 = -15.0
0x20 = 0_0100_000 = +0b1.000×2^-4 = 0.0625	0x60 = 0_1100_000 = +0b1.000×2^4 = 16.0	0xa0 = 1_0100_000 = −0b1.000×2^-4 = -0.0625	0xe0 = 1_1100_000 = −0b1.000×2^4 = -16.0
0x21 = 0_0100_001 = +0b1.001×2^-4 = 0.0703125	0x61 = 0_1100_001 = +0b1.001×2^4 = 18.0	0xa1 = 1_0100_001 = −0b1.001×2^-4 = -0.0703125	0xe1 = 1_1100_001 = −0b1.001×2^4 = -18.0
0x22 = 0_0100_010 = +0b1.010×2^-4 = 0.078125	0x62 = 0_1100_010 = +0b1.010×2^4 = 20.0	0xa2 = 1_0100_010 = −0b1.010×2^-4 = -0.078125	0xe2 = 1_1100_010 = −0b1.010×2^4 = -20.0
0x23 = 0_0100_011 = +0b1.011×2^-4 = 0.0859375	0x63 = 0_1100_011 = +0b1.011×2^4 = 22.0	0xa3 = 1_0100_011 = −0b1.011×2^-4 = -0.0859375	0xe3 = 1_1100_011 = −0b1.011×2^4 = -22.0
0x24 = 0_0100_100 = +0b1.100×2^-4 = 0.09375	0x64 = 0_1100_100 = +0b1.100×2^4 = 24.0	0xa4 = 1_0100_100 = −0b1.100×2^-4 = -0.09375	0xe4 = 1_1100_100 = −0b1.100×2^4 = -24.0
0x25 = 0_0100_101 = +0b1.101×2^-4 = 0.1015625	0x65 = 0_1100_101 = +0b1.101×2^4 = 26.0	0xa5 = 1_0100_101 = −0b1.101×2^-4 = -0.1015625	0xe5 = 1_1100_101 = −0b1.101×2^4 = -26.0
0x26 = 0_0100_110 = +0b1.110×2^-4 = 0.109375	0x66 = 0_1100_110 = +0b1.110×2^4 = 28.0	0xa6 = 1_0100_110 = −0b1.110×2^-4 = -0.109375	0xe6 = 1_1100_110 = −0b1.110×2^4 = -28.0
0x27 = 0_0100_111 = +0b1.111×2^-4 = 0.1171875	0x67 = 0_1100_111 = +0b1.111×2^4 = 30.0	0xa7 = 1_0100_111 = −0b1.111×2^-4 = -0.1171875	0xe7 = 1_1100_111 = −0b1.111×2^4 = -30.0
0x28 = 0_0101_000 = +0b1.000×2^-3 = 0.125	0x68 = 0_1101_000 = +0b1.000×2^5 = 32.0	0xa8 = 1_0101_000 = −0b1.000×2^-3 = -0.125	0xe8 = 1_1101_000 = −0b1.000×2^5 = -32.0
0x29 = 0_0101_001 = +0b1.001×2^-3 = 0.140625	0x69 = 0_1101_001 = +0b1.001×2^5 = 36.0	0xa9 = 1_0101_001 = −0b1.001×2^-3 = -0.140625	0xe9 = 1_1101_001 = −0b1.001×2^5 = -36.0
0x2a = 0_0101_010 = +0b1.010×2^-3 = 0.15625	0x6a = 0_1101_010 = +0b1.010×2^5 = 40.0	0xaa = 1_0101_010 = −0b1.010×2^-3 = -0.15625	0xea = 1_1101_010 = −0b1.010×2^5 = -40.0
0x2b = 0_0101_011 = +0b1.011×2^-3 = 0.171875	0x6b = 0_1101_011 = +0b1.011×2^5 = 44.0	0xab = 1_0101_011 = −0b1.011×2^-3 = -0.171875	0xeb = 1_1101_011 = −0b1.011×2^5 = -44.0
0x2c = 0_0101_100 = +0b1.100×2^-3 = 0.1875	0x6c = 0_1101_100 = +0b1.100×2^5 = 48.0	0xac = 1_0101_100 = −0b1.100×2^-3 = -0.1875	0xec = 1_1101_100 = −0b1.100×2^5 = -48.0
0x2d = 0_0101_101 = +0b1.101×2^-3 = 0.203125	0x6d = 0_1101_101 = +0b1.101×2^5 = 52.0	0xad = 1_0101_101 = −0b1.101×2^-3 = -0.203125	0xed = 1_1101_101 = −0b1.101×2^5 = -52.0
0x2e = 0_0101_110 = +0b1.110×2^-3 = 0.21875	0x6e = 0_1101_110 = +0b1.110×2^5 = 56.0	0xae = 1_0101_110 = −0b1.110×2^-3 = -0.21875	0xee = 1_1101_110 = −0b1.110×2^5 = -56.0
0x2f = 0_0101_111 = +0b1.111×2^-3 = 0.234375	0x6f = 0_1101_111 = +0b1.111×2^5 = 60.0	0xaf = 1_0101_111 = −0b1.111×2^-3 = -0.234375	0xef = 1_1101_111 = −0b1.111×2^5 = -60.0
0x30 = 0_0110_000 = +0b1.000×2^-2 = 0.25	0x70 = 0_1110_000 = +0b1.000×2^6 = 64.0	0xb0 = 1_0110_000 = −0b1.000×2^-2 = -0.25	0xf0 = 1_1110_000 = −0b1.000×2^6 = -64.0
0x31 = 0_0110_001 = +0b1.001×2^-2 = 0.28125	0x71 = 0_1110_001 = +0b1.001×2^6 = 72.0	0xb1 = 1_0110_001 = −0b1.001×2^-2 = -0.28125	0xf1 = 1_1110_001 = −0b1.001×2^6 = -72.0
0x32 = 0_0110_010 = +0b1.010×2^-2 = 0.3125	0x72 = 0_1110_010 = +0b1.010×2^6 = 80.0	0xb2 = 1_0110_010 = −0b1.010×2^-2 = -0.3125	0xf2 = 1_1110_010 = −0b1.010×2^6 = -80.0
0x33 = 0_0110_011 = +0b1.011×2^-2 = 0.34375	0x73 = 0_1110_011 = +0b1.011×2^6 = 88.0	0xb3 = 1_0110_011 = −0b1.011×2^-2 = -0.34375	0xf3 = 1_1110_011 = −0b1.011×2^6 = -88.0
0x34 = 0_0110_100 = +0b1.100×2^-2 = 0.375	0x74 = 0_1110_100 = +0b1.100×2^6 = 96.0	0xb4 = 1_0110_100 = −0b1.100×2^-2 = -0.375	0xf4 = 1_1110_100 = −0b1.100×2^6 = -96.0
0x35 = 0_0110_101 = +0b1.101×2^-2 = 0.40625	0x75 = 0_1110_101 = +0b1.101×2^6 = 104.0	0xb5 = 1_0110_101 = −0b1.101×2^-2 = -0.40625	0xf5 = 1_1110_101 = −0b1.101×2^6 = -104.0
0x36 = 0_0110_110 = +0b1.110×2^-2 = 0.4375	0x76 = 0_1110_110 = +0b1.110×2^6 = 112.0	0xb6 = 1_0110_110 = −0b1.110×2^-2 = -0.4375	0xf6 = 1_1110_110 = −0b1.110×2^6 = -112.0
0x37 = 0_0110_111 = +0b1.111×2^-2 = 0.46875	0x77 = 0_1110_111 = +0b1.111×2^6 = 120.0	0xb7 = 1_0110_111 = −0b1.111×2^-2 = -0.46875	0xf7 = 1_1110_111 = −0b1.111×2^6 = -120.0
0x38 = 0_0111_000 = +0b1.000×2^-1 = 0.5	0x78 = 0_1111_000 = +0b1.000×2^7 = 128.0	0xb8 = 1_0111_000 = −0b1.000×2^-1 = -0.5	0xf8 = 1_1111_000 = −0b1.000×2^7 = -128.0
0x39 = 0_0111_001 = +0b1.001×2^-1 = 0.5625	0x79 = 0_1111_001 = +0b1.001×2^7 = 144.0	0xb9 = 1_0111_001 = −0b1.001×2^-1 = -0.5625	0xf9 = 1_1111_001 = −0b1.001×2^7 = -144.0
0x3a = 0_0111_010 = +0b1.010×2^-1 = 0.625	0x7a = 0_1111_010 = +0b1.010×2^7 = 160.0	0xba = 1_0111_010 = −0b1.010×2^-1 = -0.625	0xfa = 1_1111_010 = −0b1.010×2^7 = -160.0
0x3b = 0_0111_011 = +0b1.011×2^-1 = 0.6875	0x7b = 0_1111_011 = +0b1.011×2^7 = 176.0	0xbb = 1_0111_011 = −0b1.011×2^-1 = -0.6875	0xfb = 1_1111_011 = −0b1.011×2^7 = -176.0
0x3c = 0_0111_100 = +0b1.100×2^-1 = 0.75	0x7c = 0_1111_100 = +0b1.100×2^7 = 192.0	0xbc = 1_0111_100 = −0b1.100×2^-1 = -0.75	0xfc = 1_1111_100 = −0b1.100×2^7 = -192.0
0x3d = 0_0111_101 = +0b1.101×2^-1 = 0.8125	0x7d = 0_1111_101 = +0b1.101×2^7 = 208.0	0xbd = 1_0111_101 = −0b1.101×2^-1 = -0.8125	0xfd = 1_1111_101 = −0b1.101×2^7 = -208.0
0x3e = 0_0111_110 = +0b1.110×2^-1 = 0.875	0x7e = 0_1111_110 = +0b1.110×2^7 = 224.0	0xbe = 1_0111_110 = −0b1.110×2^-1 = -0.875	0xfe = 1_1111_110 = −0b1.110×2^7 = -224.0
0x3f = 0_0111_111 = +0b1.111×2^-1 = 0.9375	0x7f = 0_1111_111 = +Inf	0xbf = 1_0111_111 = −0b1.111×2^-1 = -0.9375	0xff = 1_1111_111 = −Inf

Open Compute Project Floating-Point Standards

Two standards by AMD, Arm, Google, Intel, Meta, and NVIDIA:

- OCP 8-bit Floating Point Specification (released 2023)
- OCP Microscaling Formats (MX) Specification (released 2023)

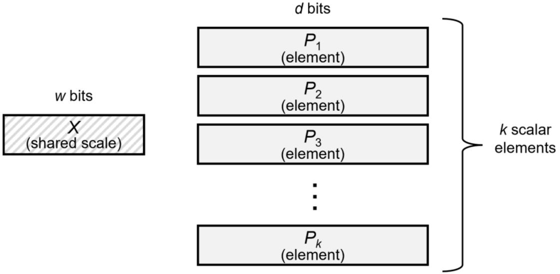
Key points:

- 8-, 6-, and 4-bit formats.
- Retains -0.
- NaNs are removed in some formats, to increase the range.
- Scaled floating-point: common scale factor (2^X) for vectors of FP or integer numbers.

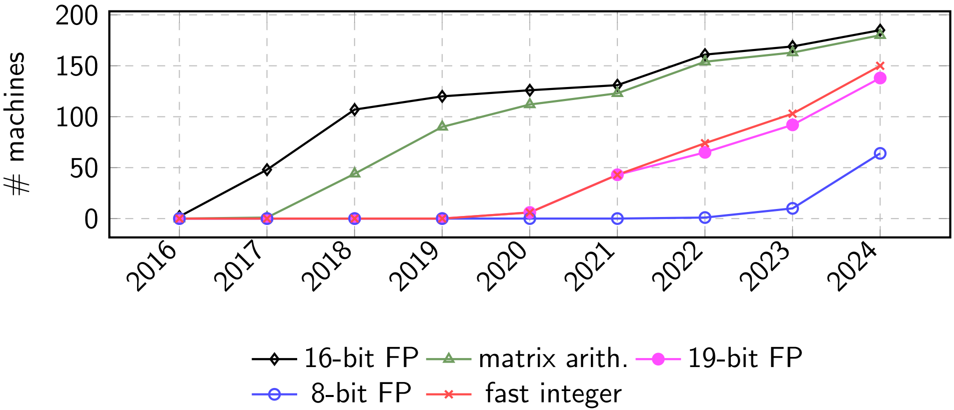
Key takeaway

OCP is different from IEEE P3109 in terms of defining subsets of reals (formats). For the foreseeable future, two 8-bit standards will be driving the progress in hardware.

Open Compute Project Floating-Point Standards (screenshot from the MX standard)



Using the TOP500 to anticipate where HPC hardware is going



Devices counted: P100, V100, A100, H100, MI210, MI250X, MI300X, Intel Data Center GPU, from `https://www.top500.org`. With NVIDIA Blackwell 4/6-bit FP will appear.

Research direction 1: Mixed-precision matrix multipliers

Architecture	Input format	Accumulation format
NVIDIA PTX ISA 8.5	fp8-E5M2	binary32
	fp8-E4M3	binary32
	binary16	binary16
	binary16	binary32
	bfloat16	binary32
	19-bit FP	binary32
AMD MI300 ISA	fp8-E5M2	binary32
	fp8-E4M3	binary32
	binary16	binary32
	bfloat16	binary32
	19-bit FP	binary32

Research direction 1: Using low precision floating-point matrix arith.

Mary and Mikaitis [2024]; Fasi et al. [2023]; Higham et al. [2019].

Approach 1: Use 8-bit FP directly (scale to avoid overflow)

$$C = \Lambda^{-1}\Big(\mathrm{fl}(\Lambda A)\mathrm{fl}(BM)\Big)M^{-1}$$

Approach 2: Use multiple 8-bit FP directly

$$A^{(i)} = \mathrm{fl}\left(\left(\Lambda A - \sum_{k=0}^{i-1} u^k A^{(k)}\right)/u^i\right)$$

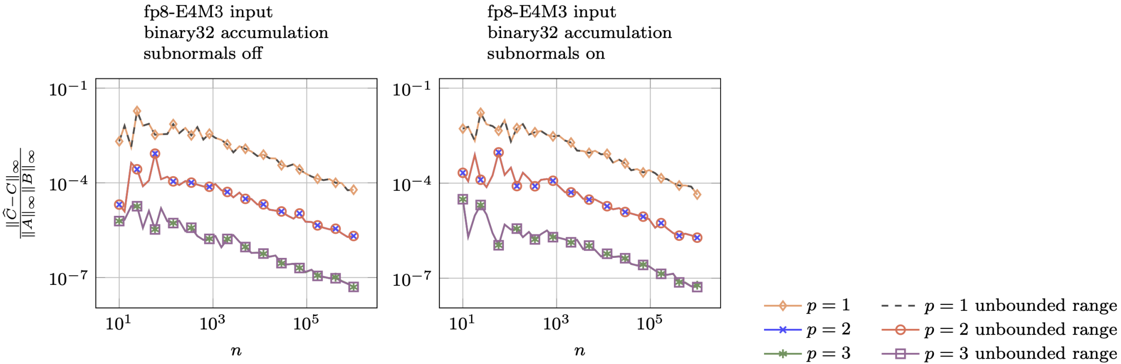
and we approximate $C = AB$ as

$$C \approx \Lambda^{-1}\left(\sum_{i+j<p} u^{i+j} A^{(i)} B^{(j)}\right) M^{-1}.$$

Research direction 1: Using low precision floating-point matrix arith.

T. Mary and M. Mikaitis [2024].

Matrix multiply; data in $[-10^{10}, -10^{-10}] \cup [10^{-10}, 10^{10}]$.



Research direction 1: Using low precision integer matrix arith.

A. Abdelfattah, J. Dongarra, M. Fasi, M. Mikaitis, and F. Tisseur, [2025, in prep.].

Ootomo et al. [2024] discovered algorithms for simulating FP matrix multiply with integer matrix multiply. Small example split:

$$\underset{A^T}{\begin{bmatrix} 2^0 \cdot 1.1001000 \\ 2^3 \cdot 1.0000000 \\ -2^1 \cdot 1.1101100 \end{bmatrix}} \Rightarrow 2^4 \cdot \underset{\text{Block fixed-point}}{\begin{bmatrix} \cancel{1}.\underline{000}\ \underline{110}\ \underline{010}\ \underline{000} \\ \cancel{1}.\underline{100}\ \underline{000}\ \underline{000}\ \underline{000} \\ -\cancel{1}.\underline{001}\ \underline{110}\ \underline{110}\ \underline{000} \end{bmatrix}} \Rightarrow 2^1 \cdot \underset{A^T_{(1)}}{\begin{bmatrix} 000 \\ 100 \\ -001 \end{bmatrix}} + 2^{-2} \cdot \underset{A^T_{(2)}}{\begin{bmatrix} 110 \\ 000 \\ -110 \end{bmatrix}} + 2^{-5} \cdot \underset{A^T_{(3)}}{\begin{bmatrix} 010 \\ 000 \\ -110 \end{bmatrix}} + 2^{-8} \cdot \underset{A^T_{(4)}}{\begin{bmatrix} 000 \\ 000 \\ 000 \end{bmatrix}}$$

$$\underset{B}{\begin{bmatrix} 2^0 \cdot 1.0110001 \\ -2^2 \cdot 1.1110100 \\ 2^1 \cdot 1.1101000 \end{bmatrix}} \Rightarrow 2^3 \cdot \underset{\text{Block fixed-point}}{\begin{bmatrix} \cancel{1}.\underline{001}\ \underline{011}\ \underline{000}\ \underline{100} \\ -\cancel{1}.\underline{111}\ \underline{101}\ \underline{000}\ \underline{000} \\ \cancel{1}.\underline{011}\ \underline{101}\ \underline{000}\ \underline{000} \end{bmatrix}} \Rightarrow 2^0 \cdot \underset{B^{(1)}}{\begin{bmatrix} 001 \\ -111 \\ 011 \end{bmatrix}} + 2^{-3} \cdot \underset{B^{(2)}}{\begin{bmatrix} 011 \\ -101 \\ 101 \end{bmatrix}} + 2^{-6} \cdot \underset{B^{(3)}}{\begin{bmatrix} 000 \\ 000 \\ 000 \end{bmatrix}} + 2^{-9} \cdot \underset{B^{(4)}}{\begin{bmatrix} 100 \\ 000 \\ 000 \end{bmatrix}}$$

Research direction 1: Using low precision integer matrix arith.

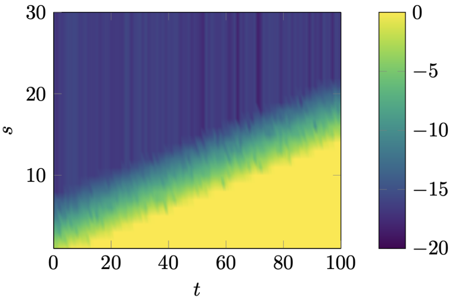
A. Abdelfattah, J. Dongarra, M. Fasi, M. Mikaitis, and F. Tisseur, [2025, in prep.].

We are developing theoretical and experimental analysis of the algorithms. Analysis is general, to cover any future hardware changes.

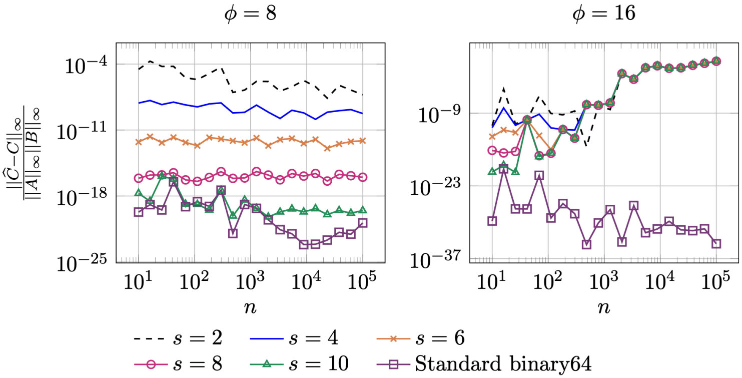
Here s is the number of splits into 8-bit chunks; colour denotes forward error of (pow 10) $\frac{|a^T b - c|}{|c|}$, where

$$a = \begin{bmatrix} 2^{-t}x \\ 1 \end{bmatrix}, \qquad b = \begin{bmatrix} 2^{t}y \\ 1 \end{bmatrix},$$

and x, y are drawn from a uniform distribution and c is a reference solution.



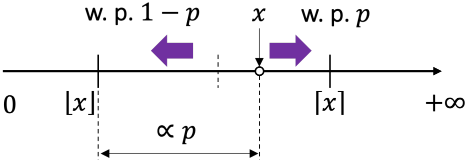
$$A_{ij}, B_{ij} = \text{uniform}(-0.5, 0.5) \times e^{\phi \times \text{normal}(0,1)}.$$



Research direction 2: Stochastic rounding

With **stochastic rounding** (**SR**), we are not rounding a number to the same direction, but to either direction with probability.

Given some x and FP neighbours $\lfloor x \rfloor$, $\lceil x \rceil$, we round to $\lceil x \rceil$ with prob. p and $\lfloor x \rfloor$ with $p - 1$.



Mode 1 SR (nearness): $p = \frac{x - \lfloor x \rfloor}{\lceil x \rceil - \lfloor x \rfloor}$ **Mode 2 SR**: $p = 0.5$

Mode 1

With **Mode 1 SR** we round x depending on its distances to the nearest two FP numbers, **cancelling out errors of different signs**.

Research direction 2: Rounding error analysis with SR

Standard error model for SR

With SR we replace u by $2u$ since it can round to the second nearest neighbour in $\mathbb{F}$.

Rounding error analysis

Worst-case error analysis determines the **upper bounds of errors**, while probabilistic error analysis describes **more realistic bounds**.

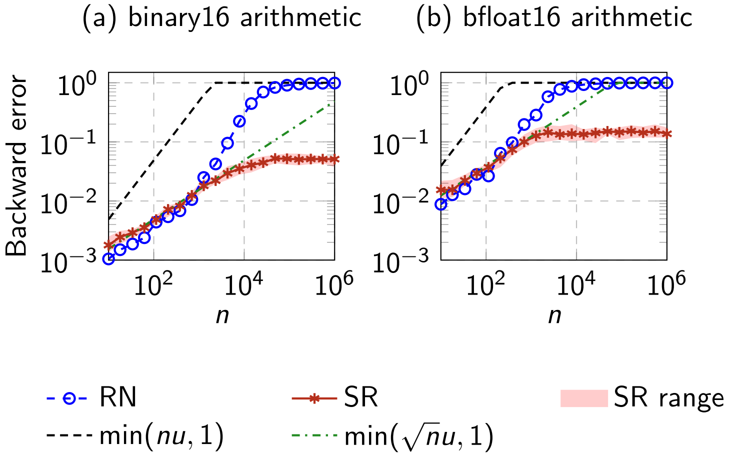
- Worst-case b-err bound with **RN**: $\frac{nu}{1-nu}$.
- Probabilistic bound with **RN**: $\lambda\sqrt{n}u + \mathcal{O}(u^2)$ w. p. $1 - 2ne^{-\lambda^2/2}$. Requires an assumption that δ_i are *mean independent zero-mean* quantities—do not always hold [Connolly, Higham, Mary, 2021].

Wilkinson rule of thumb

$\mathcal{O}(\sqrt{n}u)$ error growth is a rule of thumb with **RN**, but always holds with **SR**.

Research direction 2: Example error growth with SR in mat-vec prod

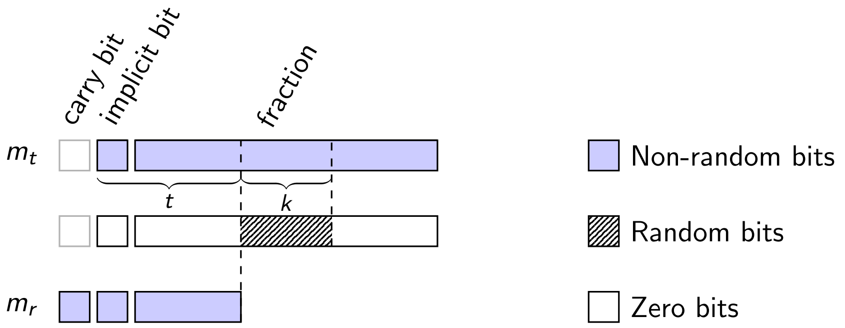
Backward error in $y = Ax$ where $A \in \mathbb{R}^{100\times n}$ with entries from uniform dist over $[0, 10^{-3}]$ and $x \in \mathbb{R}^n$ over $[0, 1]$: $\max_i \frac{|\widehat{y}-y|_i}{(|A||x|)_i}$.



Research direction 2: Consider implementation of SR

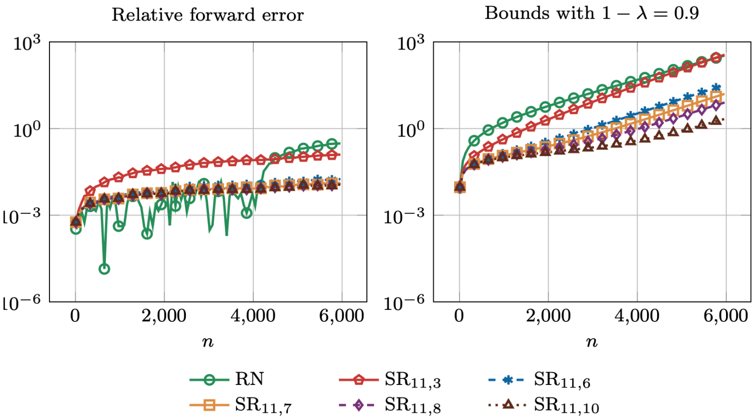
Take m_t to be a high precision unrounded significand from an operation.

Take t to be source precision and k the precision of random numbers.



E.-M. El Arar et al. [2024]: we derived a new bound $\mathcal{O}(\sqrt{n}u_p + nu_{p+r})$

Error when adding n random values in $(0,1)$ in 16-bit floating point:



Research direction 3: Determining numerical features by testing hardware

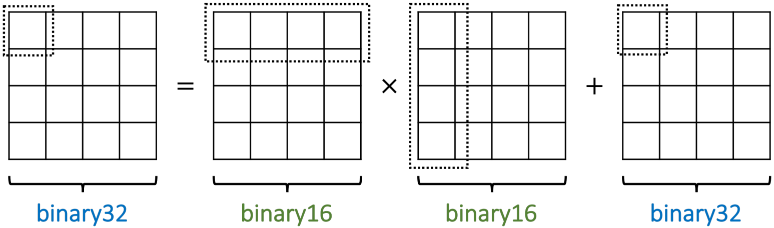
Given a small matrix multiplier not necessarily using IEEE 754 $+, \times$:

$$\begin{array}{ccccccc} D & = & C & + & A & \times & B, \\ \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}_{\substack{\text{binary16 or} \\ \text{binary32}}} & = & \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}_{\substack{\text{binary16 or} \\ \text{binary32}}} & + & \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}_{\text{binary16}} & \times & \underbrace{\begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix}}_{\text{binary16}} \end{array}$$

We are building theory and tools that allow to *report the internal precision, order of computations, rounding mode, and more*. Fasi, Higham, Pranesh, Mikaitis [2021].

We reported bit level differences between NVIDIA V100 and A100, not documented.

Research direction 3: Mixed-precision matrix multipliers on GPUs



$$a_{11} \times b_{11} + a_{12} \times b_{21} + a_{13} \times b_{31} + a_{14} \times b_{41}$$

- We are used to **normalize-round after each op**.
- In hardware it is not necessarily the case.
- Normalize at the end? Savings in circuit area and latency.
- Round or drop bits? Savings.
- Accumulate in higher precision? Can do with 1-bit granularity.

Research direction 3: Why test the mathematical hardware

IEEE 754 does not define strict rules on dot products:

> *"Implementations may associate in any order or evaluate in any wider format."*

- Implementations might differ.
- Not documented in detail by vendors.
- Massive throughput means we are using MMAs in other areas that traditionally use standardized FP.
- Can be said we are back to pre-IEEE-754 with mixed-precision.
- Eventually need to standardise (IEEE working group **P3109**).

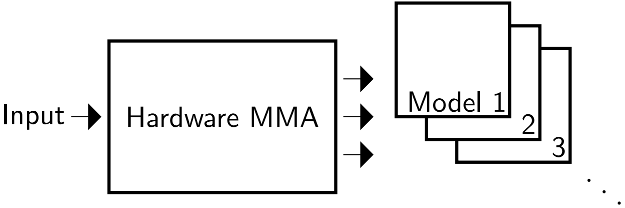
For now we create tests that determine features and differences between architectures.

Research direction 3: how do we test mathematical hardware

Technique goes back to software called `Paranoia` from the 1980s.

MMAs more complicated than $+, \times, \div$

Find floating-point **inputs that will yield different outputs on different hardware**.



Research direction 3: our main findings from testing

$$a_{11} \times b_{11} + a_{12} \times b_{21} + a_{13} \times b_{31} + a_{14} \times b_{41}$$

1-bit difference in accumulators of V100 and T4/A100.

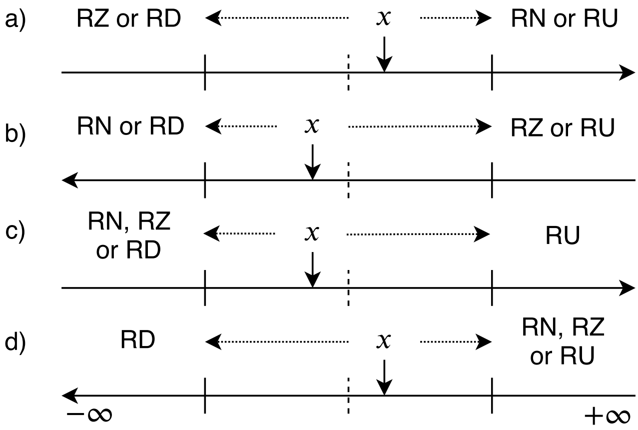
Normalization at the end, not at intermediate adds.

Rounding is not to nearest.

Monotonicity: increase one input, do not change order, dot product decreases. See [Mikaitis, 2024].

No fixed order.

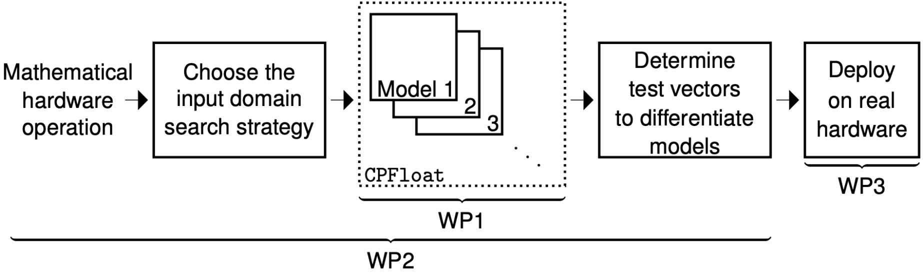
Consider determining the rounding direction:



Research direction 3: Looking for numerical features by testing hardware

3-year EPSRC-funded project 2025–2028.

Collaboration with Argonne National Lab and Intel (US).



WP4: Develop automated open-source software to report hardware features.

Research tool development: custom precision simulators

- Various packages available: `chop`, `FLOATP`, `QPyTorch`.
- Usual approach is to perform ops in binary32/64 HW.
- Round down to sub-32-bit precision: careful with double rounding.
- We believe ours is most customizable and fastest: `CPFloat` [Fasi & Mikaitis, 2023].
- Can be used in MATLAB, Octave or C.

Research tool development: example with CPFloat in MATLAB/Octave

```
>> options.format = 'binary16';
>> [~,options] = cpfloat(0, options)
options = struct with fields:
       format: 'binary16'
       params: [11 15]
    subnormal: 1
        round: 5
         flip: 0
            p: 0.5000
       explim: 1
>> cpfloat(pi, options)
ans =
    3.1406
>> options.params(1) = options.params(1) + 1;
>> cpfloat(pi, options)
ans =
    3.1426
```

Summary

- We are building theory and tools to explain computer arithmetic hardware and analyse algorithms.
- People have been there before: pre-IEEE-754 1985.
- More complicated now: vector and matrix hardware.
- Standardisation will impact future low-precision hardware.

Slides and more info at `http://mmikaitis.github.io`

Leeds Mathematical Software and Hardware Lab

Informal group, within Scientific Computation group, in the School of Computing, Univ. Leeds.

Massimiliano Fasi
Lecturer

Research and teaching

Mantas Mikaitis
Lecturer

Research and teaching

- Focusing on computer arithmetic, numerical linear algebra, high-performance computing.
- Working with IEEE P3109 and IEEE 754-2029.
- Serving on PC committees of ARITH.
- Planning MSc module on computer arithmetic.
- PhD studentships available.

References I

N. J. Higham
Accuracy and Stability of Numerical Algorithms. 2nd edition
SIAM. 2002

T. Mary and M. Mikaitis
Error Analysis of Matrix Multiplication with Narrow Range Floating-Point Arithmetic
hal-04671474. Aug. 2024

M. Fasi, N. J. Higham, F. Lopez, T. Mary, and M. Mikaitis
Matrix Multiplication in Multiword Arithmetic: Error Analysis and Application to GPU Tensor Cores
SIAM J. Sci. Comput., 45:1. Feb. 2023.

N. J. Higham, S. Pranesh, and M. Zounon
Squeezing a Matrix into Half Precision, with an Application to Solving Linear Systems
SIAM J. Sci. Comput., 41:4. Aug. 2019.

References II

A. Abdelfattah, J. Dongarra, M. Fasi, M. Mikaitis, and F. Tisseur
Error Analysis of Floating-Point Matrix Multiplication Computed via Low-Precision Integer Arithmetic
In preparation. 2025.

H. Ootomo, K. Ozaki, and R. Yokota
DGEMM on integer matrix multiplication unit
Int. J. High Perform. Comput. Appl., 38:4. Mar. 2024.

M. P. Connolly, N. J. Higham, T. Mary
Stochastic rounding and its probabilistic backward error analysis.
SIAM J. Sci. Comput., 43. 2021.

E.-M. El Arar, M. Fasi, S.-I. Filip, and M. Mikaitis
Probabilistic Error Analysis of Limited-Precision Stochastic Rounding
hal-04665809. Jul. 2024.

References III

- M. Fasi, N. J. Higham, S. Pranesh, and M. Mikaitis
  Numerical Behavior of NVIDIA Tensor Cores
  PeerJ Comput. Sci., 7. 2021
- M. Mikaitis
  Monotonicity of Multi-Term Floating-Point Adders
  IEEE Trans. Comput., 73. 2024.
- M. Fasi and M. Mikaitis
  CPFloat: A C library for Simulating Low-Precision Arithmetic
  ACM Trans. Math. Software, 49. 2023