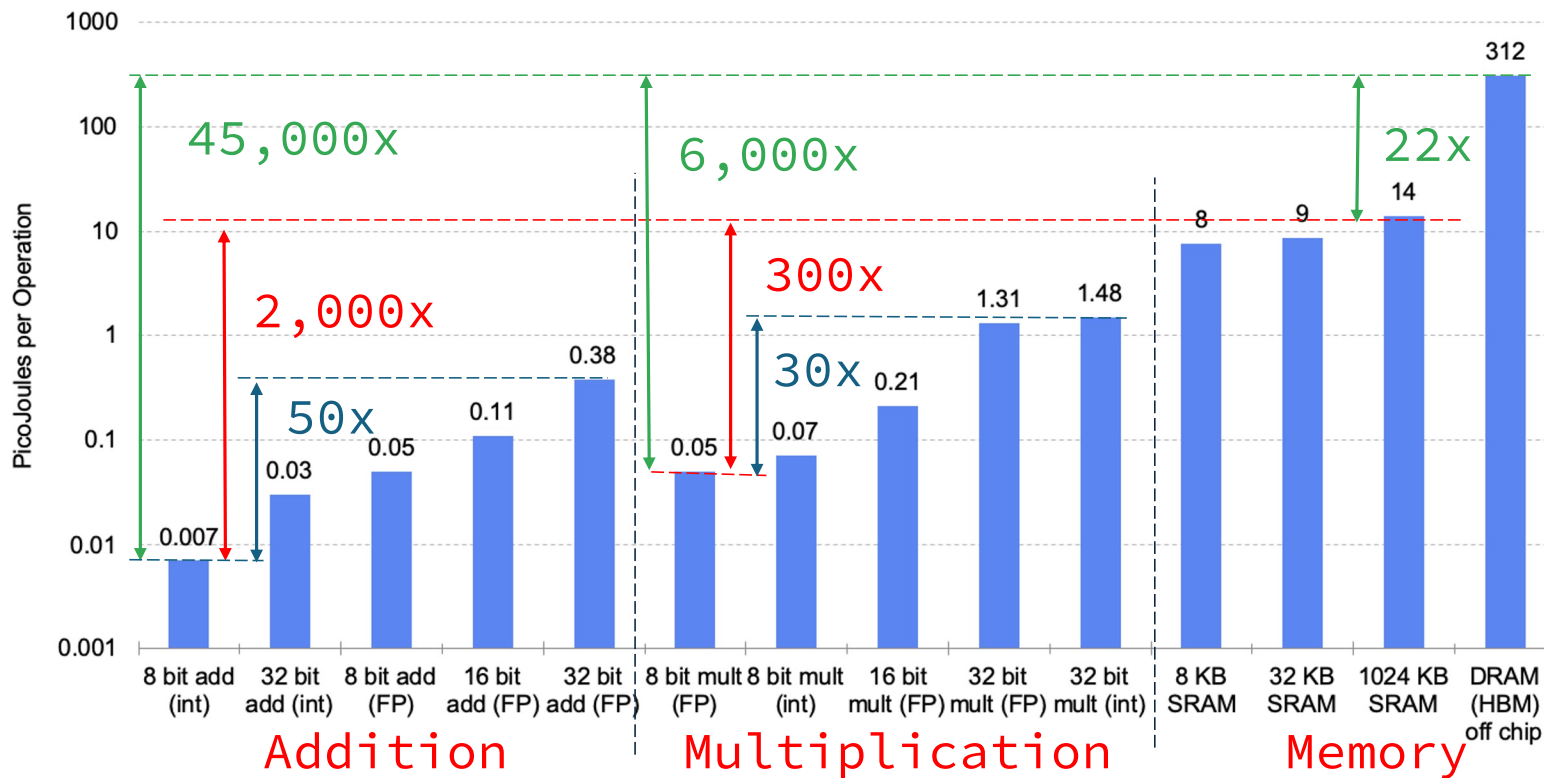


Portability with Low and Mixed Precision Arithmetic

Jim Demmel, UC Berkeley
and many others...

Chips limited by Power, Not Transistors (7 nm)

Slide from Dave Patterson



In addition to smaller data using less energy per op, they use less on-chip memory capacity and less off-chip memory bandwidth

Outline

- Summary of the draft P3109 Standard for Floating Point for Machine Learning (FP4ML)
 - Public version open for comment [1]
- Grading the BLAS
 - What accuracy guarantees should BLAS implementations, using low-precision accelerators, or Strassen-like algorithms, promise?
 - How can we verify promises made for proprietary code?
 - Early version adopted by Nvidia [2]
- Consistent exception handling
 - What behavior should libraries guarantee when exceptions occur?
 - How can we verify these promises?
 - Found bugs dating back to ISAMAX in BLAS1 [3] 😞

Summary of the draft P3109 Standard for Floating Point for Machine Learning (FP4ML)

- Motivation
 - As in the days before IEEE754, many companies are inventing their own different low precision arithmetics
 - OCP (Nvidia, Intel, ARM, Google, AMD, Meta) [4]
 - AGQ (AMD, Graphcore, Qualcomm) [5]
 - TSL (Tesla Dojo) [6]
 - P3109 convened in 2023, with many industry participants, to try to find a common design, to enable portability, compatibility and reuse of ML hardware, software and training data
 - Stakeholders include system developers, vendors and users of ML across many disciplines and industries, including medical, telecom, e-commerce, fleet management, automotive, robotics, security
- Interim reports for public comment released in Sep 2023, Oct 2024, July 2025 and Jan 2026 [1] – getting close to done!

Design space for P3109 (1/5)

- Storage width K = # of bits/word ($K \geq 3$, only $K < 16$ of interest)
- Precision P = # of significand bits, including implicit leading bit ($P \geq 1$)
- Signedness $\sigma \in \{\text{Unsigned, Signed}\} = \{u, s\}$
 - Unsigned $\Rightarrow P \leq K$, Signed $\Rightarrow P \leq K-1$
- Domain $\delta \in \{\text{Finite, Extended (includes } \infty)\} = \{f, e\}$

Notation: Binary $KpP\sigma\delta$, eg. Binary4p1uf, Binary10p5se, ...

- Rounding modes: many
- Saturation modes: how to handle overflow
- Operations, including BlockScaling

Design space for P3109 (2/5)

- If Signed ($\sigma = s$)
 - If Extended ($\delta = e$): Includes \pm Finites, one 0, one NaN, $\pm\infty$ (**se**)
 - If Finite ($\delta = f$): Includes \pm Finites, one 0, one NaN (**sf**)
- If Unsigned ($\sigma = u$)
 - If Extended ($\delta = e$): Includes +Finites, one 0, one NaN, $+\infty$ (**ue**)
 - If Finite ($\delta = f$): Includes +Finites, one 0, one NaN (**uf**)
- Always one 0, one NaN (so $1/0 = \text{NaN}$, else $1/(1/(-\infty)) = +\infty$)
- Subnormals included (gradients behave better) – SUPERnormals?
- NaN replaces -0
- No “exceptions,” just rules for dealing with $\pm\infty$ and NaN

Design space for P3109 (3/5)

- Rounding modes
 - NearestTiesToEven (required)
 - NearestTiesToAway, TowardPositive, TowardNegative, TowardZero
 - ToOdd (fast, no double rounding when done at precision+2)
 - Stochastic{A,B,C} (number of random bits not specified)
 - Try to round up/down with probability proportional to distance to opposite value, different approximations [7]
- Saturation Modes: how to handle overflow
 - Ovflnf – as in IEEE754
 - SatPropagate – clamp to largest finite number, propagate $\pm\infty$
 - SatFinite – clamp all values to largest finite number

Design space for P3109 (4/5)

- Math Operations

- Sign: Negate, Abs, CopySign

- Arithmetic:

- $1/x$, \sqrt{x} , $1/\sqrt{x}$

- $x + y$, $x - y$, $x \cdot y$, x / y , $\sqrt{x^2 + y^2}$

- FMA ($x \cdot y + z$), FAA ($x + y + z$)

- Elementary functions:

- e^x , 2^x , $\log(x)$, $\log_2(x)$, $\log(1 + e^x)$, $\sin(x)$, $\cosh(x)$, ...

- See [1] for complete list

- κ – approximation

- $\kappa = 0$ means correctly rounded

- $\kappa = 1$ means can be at most 1 floating point number farther away

- $\kappa = 2$ means can be at most 2 floating point numbers farther away...

Design space for P3109 (5/5)

- Block operations
 - Block = $[s, x_1, \dots, x_n]$, represents $[s \cdot x_1, \dots, s \cdot x_n]$
 - Can add blocks, sum-reduce blocks, compute dot products, etc.
- Other Operations (see [1])
 - Querying format, rounding properties, eg `BitwidthOf()`, `MaxFiniteOf()`, ...
 - Conversions between formats (including IEEE754)

Summary of P3109

- Summary
 - Large design space, many more formats, etc. than IEEE754
 - Vendors **not** expected to implement all options, but document which they do
 - Compromise between desire to have custom formats for different ML applications, and portability
 - Lots of work on formalizing specification, tools for validation
 - Recent email about improving description of emax
 - See Appendix C of [1] for summary of differences with OCP, AGQ, TSL (eg -0, multiple NaNs, no $-\infty$)
- What about IEEE754?
 - Meeting again, as required by IEEE every 10 years
 - Likely to add RoundToOdd as new recommended rounding mode
 - Lots of email about exception handling with parallelism

LoFloat – simulation tool for low precision arithmetic

- Efficient simulator for large design space of low precision arithmetics, including all of P3109 [8]
- C++ interface to instantiate types, PyTorch interface for mixed-precision inference experiments
- Shortcuts available for popular types (like 3109 types, OCP, TSL Dojo, etc)
- Store custom floats as short bitstrings or as hardware-native float.
- LoFloat supports MicroScaling, mixed precision BLAS (varying accumulation precision), custom exception handling (eg: P3109 NaN different from 754)
- Can generate stress tests to validate mixed precision BLAS and accumulation precision for hardware accelerator.

Motivation for Grading the BLAS

- Many new BLAS implementations
 - Use of low-precision accelerators
 - Including integer arithmetic accelerators [2]
 - Use of Strassen-like algorithms
- What can/should a BLAS implementation guarantee?
- How does this impact higher level algorithms, eg in LAPACK?
- Can we design tests that cannot be “gamed”, even if tests are public but BLAS source code is not (proprietary)?
- Initial approach: “Reverse engineer” the underlying algorithm
 - But hard to foresee whatever an LLM may imagine (or hallucinate) tomorrow
- Under some assumptions (eg algorithm not chosen randomly at each call 😊) we believe our proposed tests are reliable

Possible Grades for GEMM:

For data in “some range,” GEMM satisfies:

- Bound 1 (norm-wise): Grade = “C”
 - $\| fl(A \cdot B) - (A \cdot B) \| \leq f(n)\varepsilon \| A \| \| B \|$
 - Can be satisfied by Strassen-like algorithms [9,10,11], enough for many backward error analyses [12]
- Bound 2 (component-wise): Grade = “A”
 - $\forall i, j: |fl(A \cdot B)(i, j) - (A \cdot B)(i, j)| \leq f(n)\varepsilon (|A| \cdot |B|)(i, j)$
 - Must do $O(n^3)$ flops, so only classical matmul, not Strassen-like [13]
 - Invariant under diagonal scaling $A \rightarrow D_1 \cdot A \cdot D_3, B \rightarrow D_3^{-1} \cdot B \cdot D_2$
- Bound 3 (mixed) : Grade = “B”
 - $\forall i, j: |fl(A \cdot B)(i, j) - (A \cdot B)(i, j)| \leq f(n)\varepsilon \| A(i, :) \| \| B(: , j) \|$
 - “Between” Bounds 1 and 2; invariant under diagonal scaling with D_1 and D_2 , not D_3
- Ideally, any BLAS implementation should publish what bounds it satisfies
- “A+” possible (high relative accuracy), but expensive, will not test for it

Test 1: Strassen vs classical

- Test 1a - Gameable

- $A = \text{randn}(n, n)$, $B = \text{randn}(n, n)$, set a few randomly chosen rows of A and columns of B to zero, so corresponding rows and columns of $A \cdot B$ are zero
- Strassen will not compute these as zero (w.h.p. from roundoff)
- But easy to game by scaling with D_1, D_2 so $\| (D_1 A)(i, :) \| = \| (B D_2)(:, j) \| = 1$ to attain grade="B" for Strassen, can detect and fix zero rows and columns of $A \cdot B$

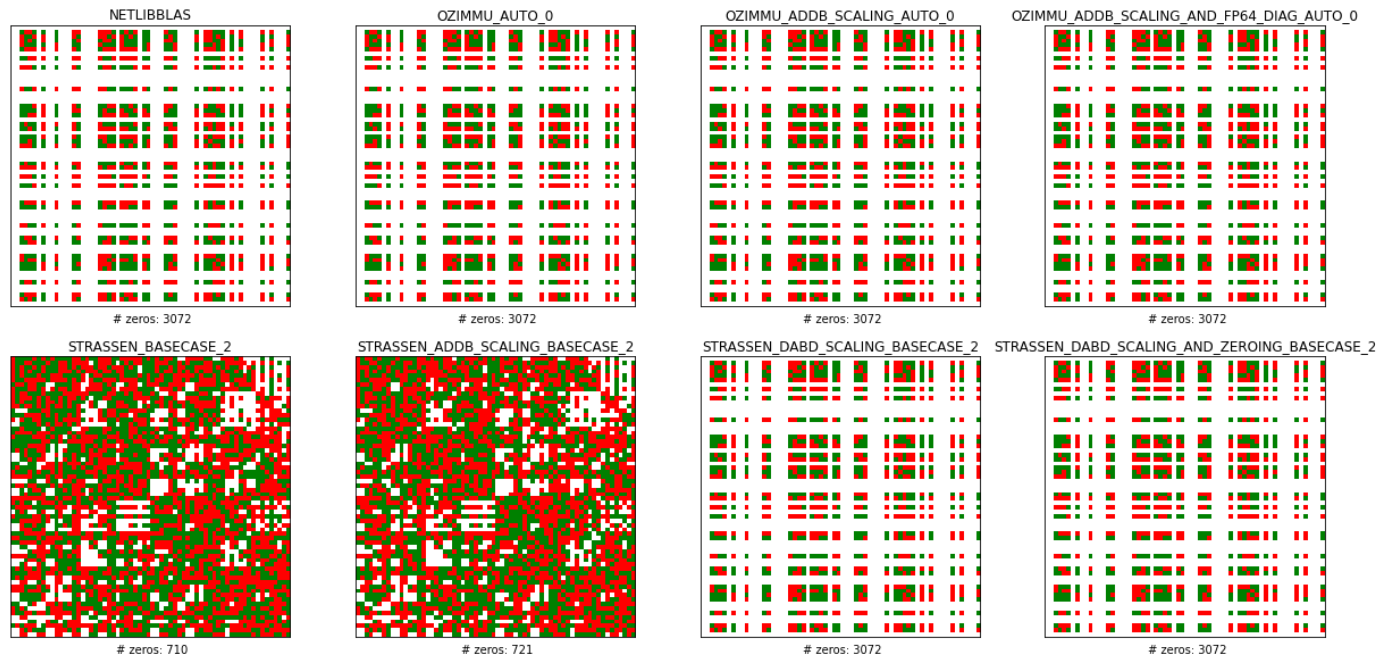
- Test 1b - ~~Not~~ Gameable

- Pick some random rows of A , make ~50% randomly sparse, pick equally many random columns of B , make complementarily sparse to a sparse row of A , so corresponding random entries of $A \cdot B$ are exactly sums of zeros
- Strassen will not compute all these entries of $A \cdot B$ as zero (w.h.p, from roundoff), so not gameable.
- Can determine size of base case n_0 for Strassen, when switch to $O(n_0^3)$ algorithm
- Conjecture: Works for Strassen-like algorithms in general
- Set tiny entries of product to 0; errors could result from Strassen or $O(n^3)$ with emulation

Results of Test 1a (see [24] for emulation code)

Test-1A, Signed Sparsity Plots for dim=64

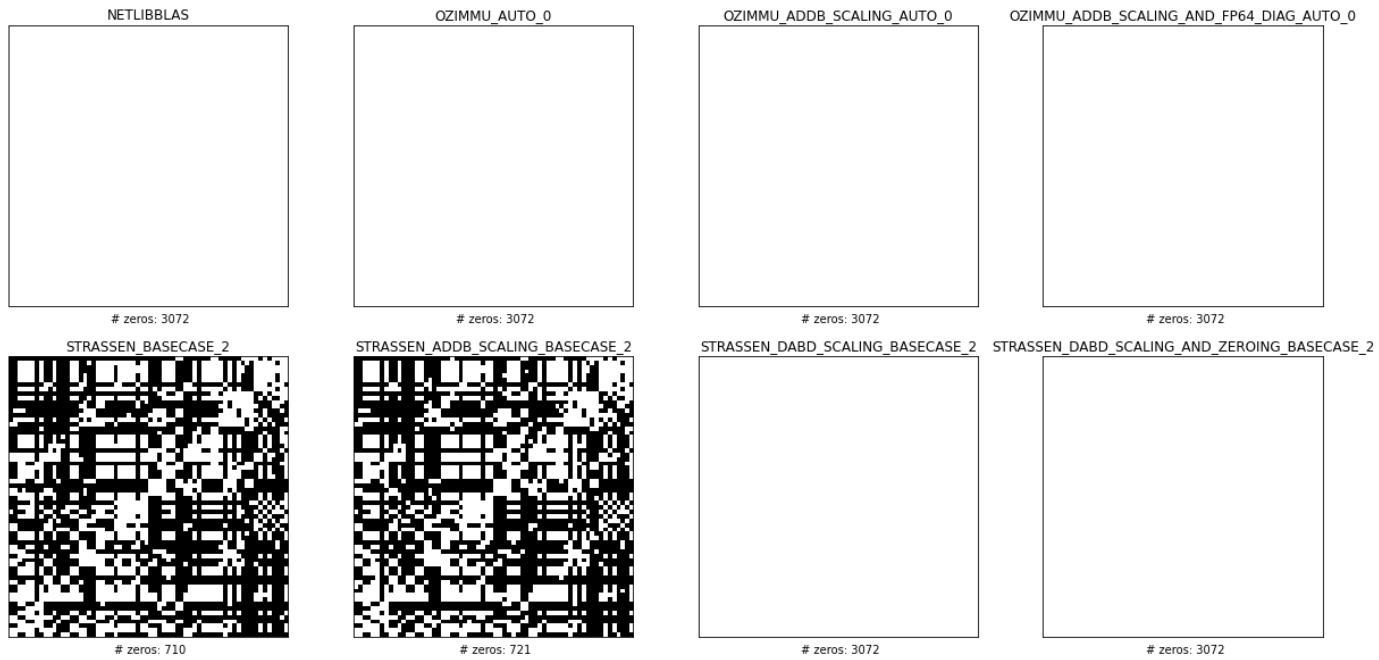
White: Zero, Green: Positive, Red: Negative



Results of Test 1a

Test-1A, Sign Comparison to Reference for dim=64

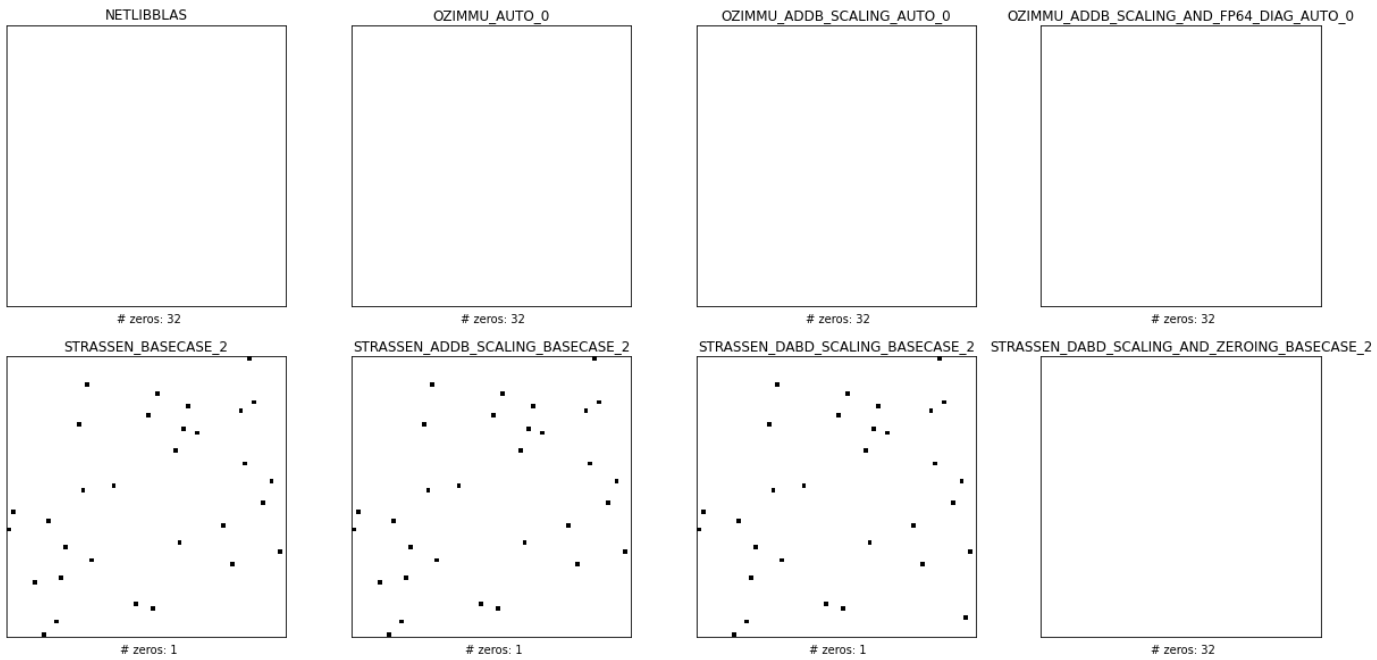
Black Squares Have Incorrect Sign wrt Reference



Results of Test 1b

Test-1B, Sign Comparison to Reference for dim=64

Black Squares Have Incorrect Sign wrt Reference



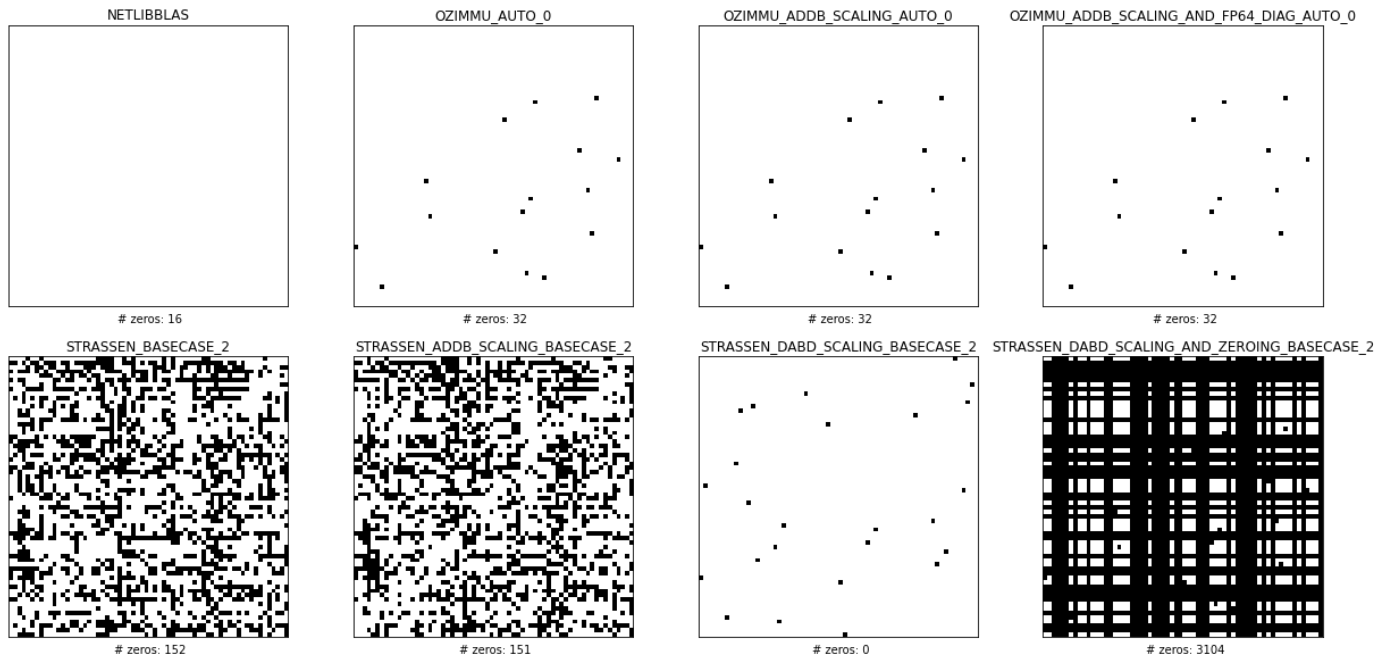
Test 1: Grade "A" or not

- Test 1c – Not Gameable (so far)
 - Strengthen Test 1b so only an $O(n^3)$ algorithm using conventional floating point can pass
 - In some random subset of sparsified rows/columns of A and B , modify them:
 - For some random k , set $A(i, k) = B(k, j) = x$, so $C(i, j) = x^2$, where $x = O(UN^{1/2})$
 - Set other nonzero $A(i, *)$ and $B(*, j) = O(OV^{1/2})$
 - Both Strassen and $O(n^3)$ with emulation will get x^2 wrong
 - Diagonal scaling $D_1 \cdot A \cdot D_3$ and $D_3^{-1} \cdot B \cdot D_2$ doesn't help

Results of Test 1c

Test-1C, Sign Comparison to Reference for dim=64

Black Squares Have Incorrect Sign wrt Reference



Test 2: Grade “A” or not

- More testing is more reliable
- Test 2a - Gameable
 - Let $x = (\text{rand}(1,n)+1)$, D_3 be diagonal within entries from UN to OV
 - Let A have rows $x \cdot D_3$, B have columns $D_3^{-1}x^T$, so each $(A \cdot B)_{i,j} = xx^T$
 - Strassen or $O(n^3)$ with emulation will be inaccurate
 - Gameable, by prescaling each $A(:, i)$ and $B(i, :)$ with D_3 to have nearly equal norms (can improve accuracy in general [11])
- Test 2b – ~~Not~~ Gameable
 - Take A and B , circularly shift i -th row of A (col of B) right (down) by i
 - Each row and col of A and B has same norm so diagonal scaling pointless
 - All diagonal entries of $A \cdot B$ should be accurate only using $O(n^3)$ with standard floating point
 - Gameable: recompute diagonal entries only, using standard floating point

Test 2: Grade “A” or not

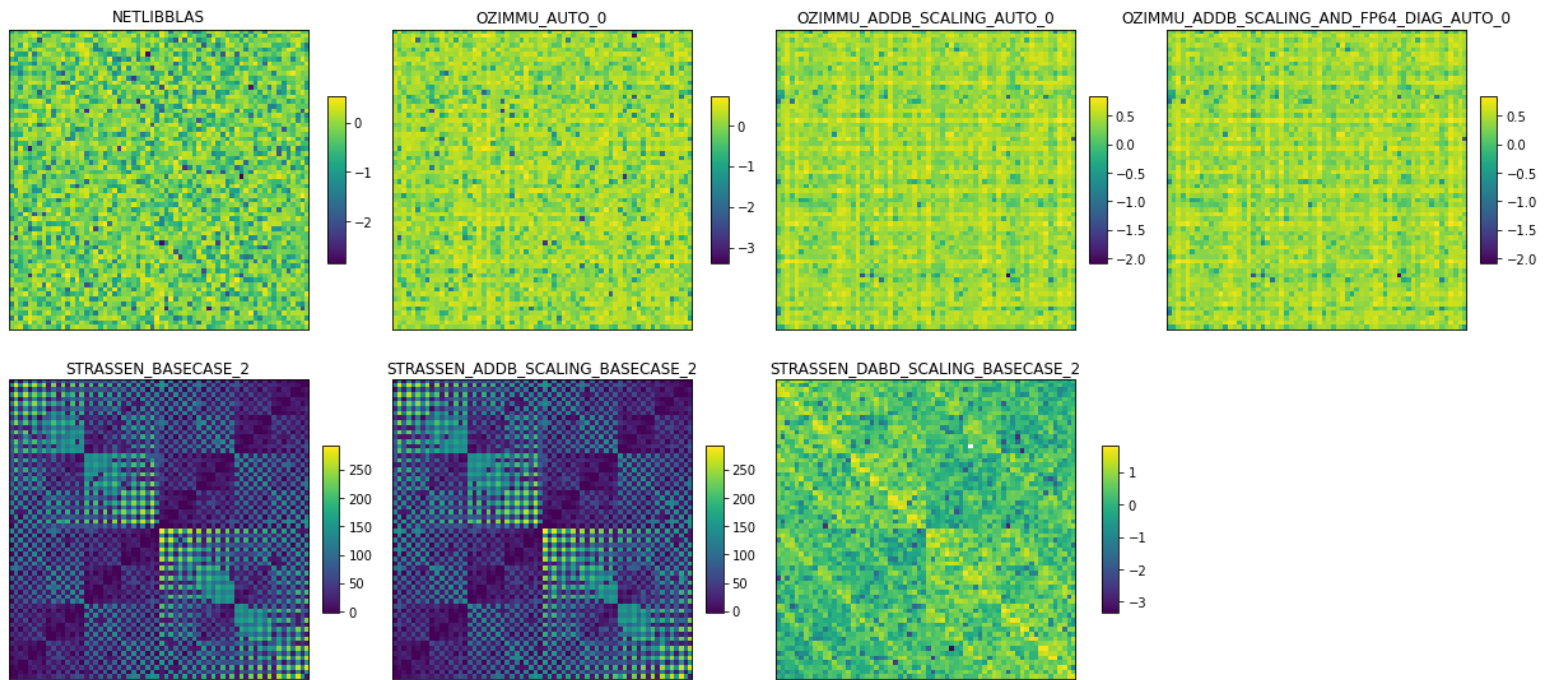
- Test 2c – Not Gameable (so far)
 - Take A and B from Test 2b, randomly permute rows of A and columns of B
 - Too expensive to search for and recompute all tiny entries accurately
 - Gaming suggestions welcome!

Test 3: Deciding between grades “B” and “C”

- Once we know the grade is not “A”, need to decide between “B” and “C”
- Let $A = \text{rand}(n,n)+1$, $B = \text{rand}(n,n)+1$, so $A \cdot B$ always accurate
- Let D_1, D_2 be diagonal with random entries from $UN^{1/2}$ to $OV^{1/2}$
- Check whether $(D_1 \cdot A) \cdot (B \cdot D_2)$ as accurate as $D_1 \cdot (A \cdot B) \cdot D_2$
- If so, grade = “B”, else “C”

Results of Test 3

Test-3, log plots of componentwise $f(n)$ factors for $\text{dim} = 64$



Test 4: One more possible test for Strassen

- Take any previously suggested A, B . Let P_1, P_2 be random permutations
- Compare $P_1 \cdot (A \cdot B) \cdot P_2$ and $(P_1 \cdot A) \cdot (B \cdot P_2)$
- If not always bitwise identical, then Strassen, else $O(n^3)$
- Possible flaw 1: floating point summation not necessarily bitwise reproducible, especially in parallel [14,15,16]
- Possible flaw 2: If choosing scale factors for emulation depend on a whole block of the matrix (subset of rows and columns), permuting will change scale factors

What about $f(n)$ factor in error bounds?

- Actually $f(m, n, k)$, where $A^{m \times n}$ and $B^{n \times k}$
- Depends on algorithm and choice of norm
- Non-Strassen, floating point, $m = k = 1$ (dot products)
 - Worst case: depends on summation order and input values
 - From $O(\log_2 n)$ (binary tree) to $O(n)$ (linear)
 - Average case (e.g. random input values)
 - Expect $O(n)$ to drop to $O(\sqrt{n})$
- $f(n)$ graded separately from Grades “A”, “B” and “C”
- Possible to get different grade for different problem sizes, if different algorithms used for different sizes

What about solving $T \cdot X = B$ (TRSM)?

- Let \hat{X} be computed solution, and $E = T \cdot \hat{X} - B$
- Bound 1 (norm-wise): Grade = “C” [19]
 - $\|E\| \leq f(n)\varepsilon \|T\| \|\hat{X}\|$
- Bound 2 (component-wise): Grade = “A” [19]
 - $\forall i, j \quad |E(i, j)| \leq f(n)\varepsilon (|T| \cdot |\hat{X}|)(i, j)$
- Bound 3 (mixed) : Grade = “B”
 - $\forall i, j \quad |E(i, j)| \leq f(n)\varepsilon \|T(i, :)\| \|\hat{X}(:, j)\|$
- Need to confirm which (public) implementations satisfy these bounds

What about solving spd $A \cdot x = b$ with tiled Cholesky?

- GEMM, SYRK and TRSM used as in LAPACK; BLAS2, BLAS1 unchanged
- Let $A = D \cdot H \cdot D$, D diagonal with $D_{ii} = A_{ii}^{1/2}$, $H_{ii} = 1$
 - Possible for $\text{cond}(H) \ll \text{cond}(A)$ [20,21] if $\max_i A_{ii} \gg \min_i A_{ii}$
- Let \hat{x} = computed solution, \hat{L} = Cholesky factor, and $E = \hat{L} \cdot \hat{L}^T - A$
- Bound 1 (norm-wise): Grade = “C” [19]
 - $\|E\| \leq f(n)\varepsilon \|A\|$, $\|x - \hat{x}\| / \|x\| = O(\varepsilon) \text{cond}(A)$
- Bound 2 (component-wise): Grade = “A” [17,18]
 - $\forall i, j \quad |E(i, j)| \leq f(n)\varepsilon (|\hat{L}| \cdot |\hat{L}^T|)(i, j)$, $\|D(x - \hat{x})\| / \|Dx\| = O(\varepsilon) \text{cond}(H)$
- Bound 3 (mixed) : Grade = “B”
 - $\forall i, j \quad |E(i, j)| \leq f(n)\varepsilon D_{ii}D_{jj}$, $\|D(x - \hat{x})\| / \|Dx\| = O(\varepsilon) \text{cond}(H)$

Future work and open questions

- Grading of various BLAS implementations underway
- Devise tests for other BLAS3
- Test for possible uses of different algorithms/arithmetics depending on problem sizes?
- Should we test for use of higher internal precision than used for inputs and outputs?
- Is it worth testing BLAS2 and BLAS1?
- Correct propagation of Infs and NaNs ...

Exception Handling for BLAS + LAPACK

- Summary of detailed list of existing “bugs”, proposed fixes, in [3,22]
- NSF/DOE proposal submitted to Correctness call
 - Plan to form working group of stakeholders (both users and providers) to advise on next steps
- Exception handling under active discussion in 754 and P3109 floating point standards committees
- Main question for this meeting: How should emulation deals with exceptions, either on input or internally generated?

“Consistent” Exception Handling Goals for BLAS and LAPACK

- If NaNs or Infs are inputs, or created while running
 1. The program will still terminate
 - Undecidable in general, we refer to constructs that can fail if a NaN appears, but are assumed to terminate otherwise, like
repeat ... until (error < tolerance)
 2. Either
 - NaNs and Infs propagate to the output in some way (either in a floating point output, or “flag”) so they are not “lost,” or
 - They are dealt with explicitly by the programmer, or
 - There are some simple, well-documented, “user-approved” cases where they do not propagate (ex: GEMM: $C = 0*A*B + 0*C$)
 3. For LAPACK, provide reporting (using INFO and more)
 - No changes to BLAS interfaces
 - Satisfy user and DOE requests for LAPACK

Inconsistent Language Exception Handling: Across programming languages/compiler

- Use of FMA or not
- Reorganizing expressions or not (eg $a+b+c+d$)
 - If $a=b=-c=-d = .75*OV$, could get Inf, -Inf, NaN or 0 (correct)
- Complex arithmetic in C and Fortran
 - $(Inf + 0*i)*(Inf + Inf*i) = NaN + NaN*i$ (Fortran) vs $Inf + Inf*i$ (C)
 - C standard provides 30+ lines of code for their multiply
- Min and Max were not associative, fixed in 754-2019, maybe not in languages
- How to make consistent
 - Outside our scope, live with it
 - Ok to propagate Inf or NaN

Inconsistent BLAS Exception Handling (1/4): ISAMAX: return index i of largest |A(i)|

- Code:

```
isamax = 1, smax = abs(A(1))
for i = 2:n
    if (abs(A(i)) .gt. smax) isamax = i, smax = abs(A(i))
```
- Inconsistency:
 - `isamax([0, NaN, 2]) = 3`
 - `isamax([NaN, 0, 2]) = 1`
- How to make consistent:
 - Point to NaN, if one exists, or (first) largest number?
 - We recommend NaN
- ICAMAX: even worse
 - `ICAMAX([OV + i*OV, Inf + i*0]) = 1`
 - Can get wrong answer with all finite inputs
- Challenge: this (inconsistent) behavior is a standard!

Inconsistent BLAS Exception Handling (2/4): TRSV: Solve $T^*x = b$, T triangular

- T can be upper (U) or lower (L), general or “unit” ($T(i,i)=1$)
- Inconsistency:
 - $U1 = [1, \text{NaN}; 0, \text{NaN}]$, $b1 = [1; 0] \Rightarrow x1 = [1; 0]$;
 - **NaNs do not propagate**; TRSV checks for trailing 0s in b , ignores cols of U
 - $U2 = [1, \text{NaN}, 1; 0, 1, 1; 0, 0, 1]$, $b2 = [2; 1; 1] \Rightarrow x2 = [1; 0; 1]$
 - **NaNs do not propagate**; TRSV checks for 0s in x , does not multiply by them
 - $L = U1^T$, $b = b1$; solve $L^T x = b$ (**same as 1st example**) $\Rightarrow x = [\text{NaN}; \text{NaN}]$
 - **TRSV does not check for zeros in this case**
- How to make consistent: Depends on what NaN means
 - If NaN means some finite number, $0 * \text{NaN} = 0$ is ok
 - If NaN means “anything”, $0 * \text{NaN} = \text{NaN}$ (IEEE 754 rules)
 - We choose latter
- Challenge: this (inconsistent) behavior is a standard!
 - And potentially much faster, $O(n)$ vs $O(n^2)$, sometimes

Inconsistent BLAS Exception Handling (3/4): Other examples, automatic tools

- Other examples (some fixed) include GER, NRM2, TRSM, SYR, SYR2, SYRK, SYR2K, SPR, SPR2, ... [22]
- Can we automate discovery of such bugs?
 - J. Vanover, C. Rubio Gonzalez, UC Davis, have such a tool, EXCVATE [23]
 - “Spoofing”: works with binary code as input, inserts Infs and NaNs as result of (some) instructions, tests if they propagate, uses SMT solver to generate an input that generates them
 - New example: GBMV, analogous to SpMV...

What about the Sparse BLAS? (4/4)

- Similar issues, and more
- SPMV: $y = A*x$
 - If $x(i) = \text{Inf}$ or NaN , and $A(j,i) = 0$ not stored, then ignore $A(j,i)*x(i)$
 - But what if register blocking introduces an explicit zero (eg an optimization in OSKI)?
 - Possible way forward: have a “paranoid” version that handles exceptions consistently, and a “reckless” version that is just as fast as possible

Collaborators

- P3109 and 754: See webpages for long lists of other committee members
- LoFloat: Sudhanva Kulkarni, Laura Grigori, David Chen
- Grading the BLAS: Mark Gates, Greg Henry, Igor Kozachenko, Julien Langou, Xiaoye Li, Wesley Pereira, Jason Riedy, Jackson Vanover
- Exception Handling: Jack Dongarra, Mark Gates, Greg Henry, Julien Langou, Xiaoye Li, Piotr Luszczek, Wesley Pereira, Jason Riedy, Cindy Rubio-Gonzalez, Jackson Vanover

References (1/5)

1. “IEEE Working Group P3109 Interim Report on Binary Floating-point Formats for Machine Learning,” ver. 3.2, 5 Jan 2026, <https://github.com/P3109/Public>
2. “Guaranteed DGEMM Accuracy While Using Reduced Precision Tensor Cores Through Extensions of the Ozaki Scheme,” A. Schwartz et al, Proc. SC Asia 2026, 10.1145/3773656.3773670
3. “Proposed Exception Handling for the BLAS and LAPACK,” J. Demmel et.al, Proc. Workshop on Software Correctness for HPC Applications, Nov 2022
4. “OCP 8-bit floating point specification (OFP8)”, P. Micikevisius et. al., <https://www.opencompute.org/documents/ocp-8-bit-floating-point-specification-ofp8-revision-1-0-2023-12-01-pdf-1>
5. “8-bit numerical formats for deep neural networks”, B. Nouné et. al., <https://arxiv.org/abs/2206.02915>

References (2/5)

6. “Tesla Dojo Technology: A guide to Tesla’s configurable floating point formats and arithmetic”, 2023,
https://web.archive.org/web/20230503235751/https://tesla-cdn.thron.com/static/MXMU3S_tesla-dojo-technology_1WDVZN.pdf
7. “On Stochastic Rounding with few random bits”, A. Fitzgibbon and S. Felix, ARITH 2025
8. LoFloat: <https://github.com/SudhanvaKulkarni123/LoFloat>
9. “Fast Matrix Multiplication is Stable,” J. Demmel, I. Dumitriu, O. Holtz, R. Kleinberg, Num. Math, v. 106, n. 2, 2007,
arxiv.org/abs/math/0603207

References (3/5)

10. “Fast Linear Algebra is Stable,” J. Demmel, I. Dumitriu, O. Holtz, Num. Math., v. 108, n. 1, 2007, arxiv.org/abs/math/0612264
11. “Improving the numerical stability of fast matrix multiplication”, G. Ballard, A. Benson, A. Druinsky, B. Lipshitz, O. Schwartz, SIAM J. Mat. Anal. Appl., v. 37, n. 4, 2016
12. “Stability of Block Algorithms with Fast Level-3 BLAS”, J. Demmel, N. Higham, ACM TOMS, v. 18, n. 3, 1992
13. “Computational complexity and numerical stability,” W. Miller, SIAM J. Comput. vol. 4, n. 2, 1975

References (4/5)

14. “Reproducible BLAS: Make Addition Associative Again!”, J. Demmel, J. Riedy, W. Ahrens, SIAM News, Oct 1, 2018
15. “A New IEEE 754 Standard for Floating-Point Arithmetic in an Ever-Changing World”, J. Demmel, J. Riedy, SIAM News, July 6, 2021
16. “Algorithms for Efficient Reproducible Floating Point Summation,” J. Demmel, W. Ahrens, J. D. Nguyen, ACM TOMS, v. 46, July 2020
17. “On floating point errors in Cholesky,” J. Demmel, LAPACK Working Note #14, 1989
18. “Accuracy and Stability of Numerical Algorithms”, N. Higham, 2002, SIAM

References (5/5)

19. “Stability of block algorithms with fast level-3 BLAS”, J. Demmel, N. Higham, ACM TOMS, v. 18, n. 3, 1992
20. “Condition numbers and equilibration of matrices”, A. Van Der Sluis, Num. Math, v. 14, 1969
21. “Nearly optimal block Jacobi preconditioning”, J. Demmel, SIAM J. Mat. Anal. Appl., v. 44, 2023
22. “Proposed Consistent Exception Handling for the BLAS and LAPACK”, J. Demmel, J. Dongarra, M. Gates, G. Henry, J. Langou, X. S. Li, P. Luszczek, W. Pereira, J. Riedy, C. Rubio-Gonzalez; arxiv.org/abs/2207.09281, ~90 pages
23. “EXCVATE: Spoofing Exceptions and Solving Constraints to Test Exception Handling in Numerical Libraries,” J. Vanover, X. S. Li, J. Demmel, C. Rubio-Gonzalez, ARITH 2025, **Best Paper Award**
24. ”DGEMM on Integer Matrix Multiplication Unit”, H. Ootomo, K. Ozaki, R. Yokota, [arxiv2306:11975v4](https://arxiv.org/abs/2306.11975v4), Mar 2024