How to grade the accuracy of an implementation of the BLAS

Short update on Exception Handling

Jim Demmel, Xiaoye Li, Julien Langou,
Weslley Pereira, Mark Gates, Cindy Rubio Gonzalez

Motivation

- Many new BLAS implementations
  - Use of low-precision accelerators
  - Including integer arithmetic accelerators
    - [1] “DGEMM on Integer Matrix Multiplication Unit”, Ootomo, Ozaki, Yokota, 2024
    - Yesterday’s talk by Devangi Parikh and Greg Henry
  - Use of Strassen-like algorithms
- What can/should a BLAS implementation guarantee?
- Can we design tests that cannot be “gamed”, even if tests are public but BLAS source code is not (proprietary)?
- Approach: “Reverse engineer” the underlying algorithm

Possible Grades for GEMM: For data in "some range," GEMM satisfies:

- Bound 1 (norm-wise): Grade = "C"
  - $\| fl(A * B) - (A * B) \| \leq f(n)\varepsilon \| A \| \| B \|$
  - Can be satisfied by Strassen-like algorithms, enough for many backward error analyses (D., Dumitriu, Holtz, [2,3], Ballard et al [6], D., Higham [7])
- Bound 2 (component-wise): Grade = "A"
  - $\forall i, j$: $|fl(A * B)(i,j) - (A * B)(i,j)| \leq f(n)\varepsilon(|A| * |B|)(i,j)$
  - Must do O(n^3) flops, so only classical matmul, not Strassen-like (Miller [4])
  - Invariant under diagonal scaling $A \rightarrow D_1 * A * D_3$, $B \rightarrow D_3^{-1} * B * D_2$
- Bound 3 (mixed) : Grade = "B"
  - $\forall i, j$: $|fl(A * B)(i,j) - (A * B)(i,j)| \leq f(n)\varepsilon \| A(i,:) \| \| B(:,j) \|$
  - "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

Test 1: Strassen vs classical

- Test 1a - Gamable
  - A = randn(n, n), B = randn(n, n), set a few randomly chosen rows of A and columns of B to zero, so corresponding rows and columns of $A * B$ are zero
  - Strassen will not compute these as zero (w.h.p.)
  - But easy to game by scaling with D_1,D_2 so $\| (D_1 A)(i,:) \| = \| (B D_2)(:,j) \| = 1$ to attain Bound 3 for Strassen, can detect and fix zero rows and columns of $A * B$
- Test 1b – Not Gamable
  - 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 $A * B$ are exactly sums of zeros
  - Strassen will not compute all these entries of $A * B$ as zero (w.h.p), not gamable.
  - Can determine size of base case n_0 for Strassen, when switch to $O(n_0^3)$ algorithm

Test 2: Given $O(n^3)$ matmul, distinguish standard floating point from arithmetic as in [1]

- Problem with [1]: zeros out tiny entries, even if these would be multiplied by large entries in other matrix
  - Creates error in i-th row of A proportional to $\varepsilon \parallel A(i,:) \parallel$, ditto for j-th col of B
  - Can only satisfy Bound 3, unless slower for matrices with big number ranges
- Test 2a - Gamable
  - Scale $A \rightarrow A * D_3,\ B \rightarrow D_3^{-1} * B$ where D_3 has large range of diagonal values
  - Either much slower, or less accurate than floating point result
  - Gamable, by prescaling each $A(:,i)$ and $B(i,:)$ with D_3 to have nearly equal norms
- Test 2b – Not Gamable
  - Take scaled 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 $A * B$ should be accurate in standard floating point, not [1]

Test 3: Given Strassen-like matmul, distinguish standard floating point from arithmetic as in [1]

- Much trickier: so far
  - Depends on "Strassen-like" algorithm (we know how to do classical Strassen)
  - Depends on base case size n_0: when recurrence shifts to $O(n_0^3)$ algorithm
- Test3a – Gamable
  - Choose $n = 2^k$, scale A, B with $D_3(i,i) = 2^{(2(i \bmod 2) - 1)m} = 2^{\pm m}$, m large
  - Strassen will maintain scaling pattern on each recursive call (not down to 1)
  - Accurate (Bound 2) if floating point used, not [1]
- Test 3b – Not Gamable
  - Various tricks needed, eg modify last row (col) of A (B) to avoid being able to unscale with D_3

What about $f(n)$?

- Actually $f(m, n, k)$, where $A^{m\ x\ n}$ and $B^{n\ x\ k}$
- 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 Bounds 1, 2 and 3

Future work and open questions

- Test for possible uses of different algorithm/arithmetic depending on problem sizes?
- Should we test for use of higher internal precision than used for inputs and outputs? Ex: TPUs, talk by Parikh/Henry
- Other BLAS3
  - TRSM might be trickier: Is $T = [I, T_{12}; 0, I]$ enough, i.e. reduction to matmul?
- BLAS2 and BLAS1
  - Easier: no Strassen
- Correct propagation of Infs and NaNs …

Update on Exception Handling for BLAS + LAPACK

- Presented in BLIS Retreat 2021, progress since then
- [5] “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, (CORRECTNESS’22), Nov 2022
(longer version at arxiv.org/abs/2207.09281, ~90 pages)
- 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

“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: 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 BLAS Exception Handling (1/3): 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 (2/3): 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, NaN; 0, NaN], b1 = [1;0] ⇒ x1 = [1;0];
    - **NaNs do not propagate**; TRSV checks for trailing 0s in b, ignores cols of U
  - U2 = [1, NaN, 1; 0, 1, 1; 0, 0, 1], b2 = [2;1;1] ⇒ 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**) ⇒ x = [NaN; 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*NaN = 0 is ok
  - If NaN means “anything”, 0 * NaN = 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

What about the Sparse BLAS? (3/3)

- Similar issues, and more
- SPMV: y = A*x
  - If x(i) = 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

References (1/2)

1. “DGEMM on integer matrix multiplication unit”, H. Ootomo, K. Ozaki, R. Yokota, Intern. J. HPC Apps., vol. 0(0) 1-17, 2024, arxiv.org/pdf/2306.11975.pdf
2. “Fast Matrix Multiplication is Stable,” J. Demmel, I. Dumitriu, O. Holtz, Num. Math, v. 106, n. 2, 2007, arxiv.org/abs/math/0603207
3. “Fast Linear Algebra is Stable,” J. Demmel, I. Dumitriu, O. Holtz, Num. Math., v. 108, n. 1, 2007, arxiv.org/abs/math/0612264
4. “Computational complexity and numerical stability,” W. Miller, SIAM J. Comput. vol. 4, n. 2, 1975

References (2/2)

5. “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, Proc. Intern. Workshop on Software Correctness for HPC Applications (CORRECTNESS’22), Nov 2022
(longer version at arxiv.org/abs/2207.09281, ~90 pages)
6. “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
7. “Stability of Block Algorithms with Fast Level-3 BLAS”, J. Demmel, N. Higham, ACM TOMS, v. 18, n. 3, 1992