A Proposal for a Next-Generation BLAS

Jim Demmel, UC Berkeley
Greg Henry, Intel
Xiaoye Li, LBL
Jason Riedy, Georgia Tech
Peter Tang, Intel

bit.ly/Batch-BLAS-2017

Outline

- Goals of Next Gen BLAS
- Proposed Naming Scheme
- Reproducible BLAS
- Mixed/Extended Precision BLAS (nee XBLAS)
- Batch BLAS
- Fixed Point BLAS
- Error handling
- Questions for audience

Goals of Next Gen BLAS (1/2)

- One interface to accommodate current and future needs for BLAS
  - Standard, mixed and new precisions, batched, reproducible, and combinations
  - Support current and future versions of Sca/LAPACK (eg reproducible, batched)
  - Would be wrapper around existing BLAS when semantics match

Goals of Next Gen BLAS (2/2)

- Straightforward but detailed naming scheme
  - "F77" level, makes semantics clear
  - Overloading can greatly decrease the number and lengths of names in higher level languages
- We only recommend implementing, or optimizing, the (small) useful subset
  - This subset may grow or shrink over time
- Not all details settled
  - Comments and discussion welcome!

Proposed Naming Scheme (1/3)

- BLAS_<blasFunction>_<typeSequence>[_<other>]
  - blasFunction ∈ {gemm, symv, dot, rotg, gemm_batch…}
  - typeSequence = one or more types
    - type = <mathType><length>[<multiplier>]
    - mathType ∈{R,C}, possibly {I, QI, …}
    - length ∈{8,16,32,64,80, ...}
      - Examples: R32 (old S), C64 (old Z)
    - [optional] multiplier:
      - x2 … R64x2 is double-double
      - Repro3 … R64Repro3 is reproducible accumulator of 3 bins

Proposed Naming Scheme (2/3)

- BLAS_<blasFunction>_<typeSequence>[_<other>]
  - blasFunction∈{gemm, symv, dot, rotg, gemm_batch…}
  - typeSequence = one or more types
  - How many types?
    - 1 type
      - All arguments the same (excluding N, LDA etc), usual case
      - Ex: BLAS_GEMM_R64: usual DGEMM, C = ALPHA*A*B + BETA*C
    - #arrays types
      - All array types specified
      - Use max length for internal precision (default)
      - Scalar types inferred, to be “max” mathType and length, ok for XBLAS
      - Ex: BLAS_GEMM_R32R32R64: A and B are R32; C, ALPHA and BETA are R64
    - #arrays + #scalars types
      - All array and scalar types specified; rare!
      - Ex: BLAS_ROTG_C32C32R32C32(CA,CB,C,S) is CROTG(CA,CB,C,S), CROT too
    - Inferable, so BLAS_GEMM enough

Proposed Naming Scheme (3/3)

- BLAS_<blasFunction>_<typeSequence>[_<other>]
  - blasFunction ∈{gemm, symv, dot, rotg, gemm_batch…}
  - typeSequence = one or more types, eg C32, R64x2
  - [optional] other, could be
    - <length><multiplier>, to indicate internal precision
      - Ex: _64x2, for double-double, used for XBLAS
    - _Repro3, to indicate reproducible summation, with a 3 bin reproducible accumulator
    - Combinations, extensions possible

Examples

- BLAS_GEMM_R64(…, ALPHA, A,…,B,.., BETA, C,…)
  - Usual DGEMM, R64 inferable by overloading
- BLAS_GEMM_R64_64x2(…, ALPHA, A,…,B,.., BETA, C,…)
  - Use double-double internally, _R64 inferable, not _64x2
- BLAS_GEMM_OUT_R64(…, ALPHA,A,…,B,…,BETA, C_in, C_out,…)
  - separate input and output C matrices, _R64 inferable
- BLAS_GEMM_OUT_R64R64R64R64x2(…, ALPHA_hi, ALPHA_lo, A,…, B,…, BETA_hi, BETA_lo,C_in, C_out_hi, C_out_lo,…)
  - C_out is double-double, so the scalars ALPHA, BETA are too, and the internal precision
  - Code can optimize if ALPHA_lo=0 or BETA_lo=0, else specify types of ALPHA and BETA to be R64
  - C_out_hi and C_out_lo each has its own leading dimension

Examples

- BLAS_GEMM_R64(…, ALPHA, A,…,B,.., BETA, C,…)
  - Usual DGEMM, R64 inferable by overloading
- BLAS_GEMM_R64_64x2(…, ALPHA, A,…,B,.., BETA, C,…)
  - Use double-double internally, _R64 inferable, not _64x2
- BLAS_GEMM_OUT_R64(…, ALPHA,A,…,B,…,BETA, C_in, C_out,…)
  - separate input and output C matrices, _R64 inferable
- BLAS_GEMM_OUT_R64R64R64R64x2(…, ALPHA_hi, ALPHA_lo, A,…, B,…, BETA_hi, BETA_lo,C_in, C_out_hi, C_out_lo,…)
  - C_out is double-double, so the scalars ALPHA, BETA are too, and the internal precision
  - Code can optimize if ALPHA_lo=0 or BETA_lo=0, else specify types of ALPHA and BETA to be R64
  - C_out_hi and C_out_lo each has its own leading dimension

Reproducibility

- Outline
  - Recall definition and design goals
    - Algorithm sketch, error bounds
  - Summarize Design Space for Interface
    - Questions for audience
  - Making Sca/LAPACK reproducible
- Paper with details, software at bebop.cs.berkeley.edu/reproblas
  - Joint work with Hong Diep Nguyen, Peter Ahrens

Defining reproducibility (1/2)

- Get bitwise identical answers on any computer, no matter what hardware resources are available, or how they are scheduled, for any size and ordering of inputs, that would get identical results in exact arithmetic.
- Assumptions
  - Sum at most 2^{33} singles or 2^{64} doubles
  - Limited subset of IEEE 754 available
  - If double-double used for any intermediate results, must always be same algorithm

Defining reproducibility (2/2)

- Other design goals
  - Accuracy at least as good as conventional, and tunable
  - Handle exceptions reproducibly
  - One read-only pass over summands
  - One reduction
  - Use as little memory as possible, to enable tiling BLAS

Algorithm sketch

- Take floating point exponent range, divide it into intervals of some fixed width (eg w=40 bits)
- For each summand
  - Break its mantissa into intervals
  - Sum all bits in each interval exactly (call the accumulator for an interval a *bin*; each bin represented by 2 floats)
  - Only keep the topmost k bins, where k chosen by user
    - Ex: Repro3 means k=3
- After summing, round bins to 1 float
- Call k bins (2k floats) a *reproducible accumulator*
- See 100+ page report for details

Error bound for Reproducible Sum

- Notation
  - S = exact sum of $x_1,\ldots,x_n$
  - S_{repro} = computed sum
  - k = #bins, w = bin width, ε = machine epsilon
- $|S - S_{repro}| \le n*2^{(1-k)w} *\max |x_i| + 7*\varepsilon*|S|$
- Ex: for double precision, $\varepsilon = 2^{-53}$, w=40, k=3

  $|S - S_{repro}| \le n*2^{-80} *\max |x_i| + 7*2^{-53}*|S|$
- May be 10^8 times smaller than usual bound, depending on cancellation

Reproducible GEMM (1/3)

- C_out = ALPHA*A*B + BETA*C_in
- Which variables can be Repro*k*?
  - No algorithms (or use cases) for multiplying a reproducible accumulator by a general scalar, i.e. other than {±1,0}
    - Only C_out and/or C_in may be Repro*k*
    - If C_in is Repro*k*, then BETA ∈{±1,0}
    - Needed to support parallel reductions (ScaLAPACK)

Reproducible GEMM (2/3)

- C_out = ALPHA*A*B + BETA*C_in
- Dealing with general ALPHA and BETA
- Where to put parentheses in ALPHA*A*B?
  - (ALPHA*A)*B or A*(ALPHA*B): up to user, may require workspace for As=ALPHA*A or Bs=ALPHA*B, which we want to avoid if possible
  - Σ_k (ALPHA*(A(i,k)*B(k,j)) – doubles #multiplies
  - ALPHA*(A*B) – may lose reproducibility if C_in or C_out are Repro*k*, i.e. part of reduction

Reproducible GEMM (3/3)

- C_out = ALPHA*A*B + BETA*C_in
- Interface #1 (*recommended*)
  - If C_in and C_out are both floats, ALPHA and BETA can be general
    - Simplest for reproducible LAPACK
  - If either C_in or C_out are Repro*k*, ALPHA & BETA restricted to {±1,0}
- Interface #2 (*more restrictive*)
  - C_in and C_out can be floats or Repro*k*
  - ALPHA and BETA restricted to {±1,0}
- Interface #3 (*most restrictive*)
  - C_in and C_out must both be float or both be Repro*k*
  - ALPHA and BETA restricted to {±1,0}
  - Still enough for reproducible Sca/LAPACK, with some small algorithmic changes, no interface changes or extra workspace (in LAPACK/SRC)
- Comments welcome!

Reproducible Sca/LAPACK

- Only address sources of nonreproducibility in BLAS
  - Ex: If compilers use FMA differently, not our problem
- Proposed Interface #1 allows us to replace BLAS calls
- What about Interfaces #2,3 with ALPHA,BETA $\in\{\pm 1,0\}$?
  - Goal: Don't change interfaces, including workspace needed
  - Ex: Replacing C = ALPHA*A*B+BETA*C by Ctmp = A*B, C = ALPHA*Ctmp + BETA*C requires Ctmp
- Used parser (thanks to Ben Mehne) to extract all BLAS calls and their ALPHA, BETA arguments in Sca/LAPACK
  - Few did not have ALPHA,BETA $\in\{\pm 1,0\}$
  - Most handled by simple transformations, like replacing y = 1*A*x+BETA*y by y = BETA*y, y = 1*A*x+1*y
  - A few were more complicated
    - Ex: replace GEMM by C=(BETA/ALPHA)*C, C = A*B+C, C = ALPHA*C
  - Some interface changes needed in LAPACK/TESTING, not SRC
  - Details in report on webpage, or in (very detailed!) spreadsheets
- ScaLAPACK simpler than LAPACK (less functionality)

Mixed/Extended Precision BLAS (nee XBLAS)

- Proposed naming scheme supports mixed precision
  - Ex: BLAS_GEMM_R32R64R64 : A is 32 bit, B and C are 64 bit (ALPHA & BETA too)
- Output C_out may be higher precision than inputs
  - Useful for distributed memory algorithms
  - Missing functionality in current XBLAS
  - Ex: BLAS_GEMM_OUT_R64R64R64R64x2: A, B, C_in are 64 bit, C_out is double-double
- Mostly XBLAS is used by LAPACK in 28 cases of matrix-vector product, for iterative refinement
  - Hundreds of cases unused
  - Not all cases should be implemented/tuned
- R64x2 outputs can be “normalized”:
  - (double) (high+low) = high, i.e. low can be ignored

Batch GEMM

- Using the same naming scheme, only implement useful subset
- Batching using groups, as in MKL
  - Inputs divided into groups (# = group_count)
  - Group i contains size_per_group(i) sets of inputs/outputs
  - Group shares common values of TRANSA, TRANSB, M, N, K, LDA, LDB, LDC, ALPHA, BETA
  - If group_count = 1, standard approach
- Alternate data structures possible (eg cuBLAS, Kokkos) - Beyond scope
- BLAS_GEMM_Batch_R64 (

  TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB, BETA, C, LDA,

  group_count, size_per_group)
  - Dimension of green arrays is group_count
  - Dimension of blue arrays is Σ_i size_per_group(i), contain pointers to matrices
- Ex: group_count = 2
  - Group 1: TRANSA = TRANSB=T, M=N=N=LDA=LDB=LDC=5, ALPHA=BETA=1.0 with 3 matrices.
  - Group 2: TRANSA = TRANSB=N, M=N=K=LDA=LDB=LDC=4, ALPHA=BETA=3.0 with 2 matrices.
  - There are 3+2 = 5 matrix pointers in A, B, C

Fixed Point – Quantized Integer (QI)

- A QI is encoded by an integer pair (I, Q) which represents the value I/2^Q.
- Ex: BLAS_GEMM_QI32(… ALPHA, ALPHAQ, A, AQ, …, B, BQ, BETA, BETAQ, C, CQ, …)
  - If (ALPHA,ALPHAQ) = (100,8), the value represented is 100/2^8 = 0.390625
  - All entries of the A matrix share a single Q value, so AQ is one integer.
- Use integer arithmetic
  - Ex: (ALPHA,ALPHAQ) = (BETA,BETAQ) = (1,0), AQ = BQ = CQ = 8
  - C = A*B + C, all matrices with 8 bits to the right of the binary point
  - The integer value of Σ_k A(i,k)*B(k,j) is right shifted 8 bits and and rounded to an integer before adding to C(i,j).
- We can also use [_<other>] to specify specific rounding methods
  - Default: truncation / round-toward-zero ?
  - Use extra internal width, eg BLAS_GEMM_QI32_64(…) uses 32-bit inputs & outputs and 64 bits internally

Error Handling (1/2)
XERBLA and Functional Returns

- We propose deprecating XERBLA.
- Each BLAS subroutine has an integer return value for parameter checking like INFO in LAPACK
  - 0 means success
  - -K means the Kth parameter was an illegal input
  - +K could be used for other “error” conditions
    - Ex: Zero on diagonal in TRSV, NaN checking
- Old BLAS functions, eg DOT, will now be subroutines with the answer as a final new parameter

Error handling (2/2)
Consistent Exception Handling

- Guaranteed by reproducible summation
  - Always (or never) get same ±Inf or NaN
- Should we target consistent exception handling in general?
- Ex: Reference TRSV propagates NaNs differently in solving U*x=b with TRANS='N' and L^T*x=b with L=U^T and TRANS='T'
- Ex: Reference ISAMAX([0,NaN,2]) returns 3, and ISAMAX([NaN,0,2]) returns 1

Discussion/Questions for Audience

- Lots of details omitted, see posted document at
  - bit.ly/Batch-BLAS-2017
- Which of 3 interfaces for Reproducible BLAS to use?
- Which batch functionality should we include?
- Which fixed point formats should we include?
- All BLAS are integer functions returning an error flag, like INFO?
- Target consistent exception handling? Report other errors?

- Is recommended subset of BLAS to be implemented ok?
- Add any matrix formats, like RFP = Rectangular Full Packed?

“One string to rule them all”

Backup Slides

RFP = Rectangular Full Packed