Next Generation BLAS (BLAS G2)

Mark Gates\textsuperscript{1}, Jim Demmel\textsuperscript{2}, Greg Henry\textsuperscript{3}, Xiaoye Sherry Li\textsuperscript{4}, Jason Riedy\textsuperscript{5}, Peter Tang\textsuperscript{3}

\textsuperscript{1} U. of Tennessee, \textsuperscript{2} U. of California at Berkeley, \textsuperscript{3} Intel, \textsuperscript{4} Lawrence Berkeley National Laboratory (LBNL), \textsuperscript{5} Georgia Tech

http://icl.utk.edu/bblas/siam-cse19/
Classical BLAS

- **Level 1 BLAS: vector operations (1979)**
  - Examples: saxpy, daxpy, caxpy, zaxpy
  - 50 routines

- **Level 2 BLAS: matrix-vector operations (1988)**
  - Examples: sgemv, dgemv, cgemv, zgemv
  - 66 routines

- **Level 3 BLAS: matrix-matrix operations (1990)**
  - Examples: sgemm, dgemm, cgemm, zgemm
  - 30 routines

- **Total 146 routines**
XBLAS (2001)

- Supports mixed precision and internal extended precision
  - BLAS_zgemm_c_c_x
    - C matrix is “z” double-complex
    - A matrix is “c” single-complex
    - B matrix is “c” single-complex
    - “x” is internal extended precision (> 80 bit)
- Adds ~490 routines
BLAS G2 Goals

- Mixed & extended precision
  - Support distributed computation by having higher precision outputs
- Support more data types
  - half precision, double-double, fixed point (quantized integer), ...
- Reproducibility
- More flexible layout (column & row strides)
- Better error handling
- Fix inf & nan propagation
- Easy high-level interface (`blas::gemm`)
- Extensible, uniform low-level naming scheme (`BLAS_gemm_r64_repro3`)
- Reasonably optimized reference implementation (todo)
- Orthogonal to batch
Mixed precision

- XBLAS allowed mixtures of single/double or real/complex (not both):
  - single
  - single-complex
  - double
  - double-complex

- We extend this to other datatypes:
  - half, double-double, ...
  - Can support any combination found to be useful
Extended precision

- XBLAS didn’t specify exact internal extended precision
- We propose using double-double, and adding inputs & outputs
  - \( x = x_{hi} + x_{lo} \)
  - New ops in IEEE 754 for efficiency (e.g., augmentedAddition twoSum)
- Non-interleaved
  - Two arrays: one high order, one low order
  - Low bits as workspace to throw away
- Interleaved
  - One array of double-double structs
  - Suited for C++ templating (e.g., QD library)
  - Pass as non-interleaved using column & row strides
Extended precision

• XBLAS didn’t specify exact internal extended precision
• We propose using double-double, and adding inputs & outputs
  • \( x = x_{hi} + x_{lo} \)
  • New ops in IEEE 754 for efficiency (e.g., augmentedAddition twoSum)
• Non-interleaved
  • Two arrays: one high order, one low order
  • Low bits as workspace to throw away
• Interleaved
  • One array of double-double structs
  • Suited for C++ templating (e.g., QD library)
  • Pass as non-interleaved using column & row strides
Out-of-place operations

- XBLAS does not give access to internal extended precision
- Parallel reduction needs local output in extended precision, to do reduction
  - blas::gemv( ..., A, x, y )
    \[ y = Ax + y \]
  - blas::gemv_out( ..., A, x, y, y_out )
    \[ y_out = Ax + y \]
Out-of-place operations

- XBLAS does not give access to internal extended precision
- Parallel reduction needs local output in extended precision, to do reduction
  - blas::gemv( ..., A, x, y )
    \[ y = Ax + y \]
  - blas::gemv_out( ..., A, x, y, y_out )
    \[ y_{\text{out}} = Ax + y \]
Out-of-place operations

- XBLAS does not give access to internal extended precision
- Parallel reduction needs local output in extended precision, to do reduction
  - `blas::gemv( ..., A, x, y )
    \[y = Ax + y\]
  - `blas::gemv_out( ..., A, x, y, y_out )
    \[y_{out} = Ax + y\]
Out-of-place operations

- XBLAS does not give access to internal extended precision
- Parallel reduction needs local output in extended precision, to do reduction
  - blas::gemv(..., A, x, y)
    \[ y = Ax + y \]
  - blas::gemv_out(..., A, x, y, y_out)
    \[ y\_out = Ax + y \]

\[
\begin{array}{ccc}
A_1 & x_1 & \Rightarrow \quad y_1 \\
A_2 & x_2 & \Rightarrow \quad y_2 \\
\end{array}
\quad \Rightarrow \quad \begin{array}{c}
y_1 + y_2 \\
\Rightarrow \quad y \\
\end{array}
\]
New data types

- **BLAS had only 4 precisions:** s, d, c, z
- **Proposed low-level interface** specifies real or complex (R or C), and precision
  - `BLAS_gemm_r16`  `BLAS_gemm_c16`
  - `BLAS_gemm_r32` (sgemm)  `BLAS_gemm_c32` (cgemm)
  - `BLAS_gemm_r64` (dgemm)  `BLAS_gemm_c64` (zgemm)
  - `BLAS_gemm_r64x2`  `BLAS_gemm_c64x2` (double-double)
- **What to do with bfloat16 (Google TPU, ...)?**
  - 8-bit exponent (same as single) and 8-bit precision (bfloat16)
  - 5-bit exponent and 11-bit precision (IEEE 16-bit)
- **Fixed point (quantized integer)**
  - `BLAS_gemm_qi32`
Reproducibility

- Floating point is not associative!
  - \( x + x + (-x) + (-x) \) for \( x > \frac{1}{2} \) overflow
  - Result can be 0, +\( \text{inf} \), –\( \text{inf} \), or \( \text{nan} \), depending on summation order
  - An application could obtain different results with different number of threads
  - 2-norm of large set of numbers
    - \( \text{ssq} = \sum_{i=1}^{n} x_i^2 \)
    - May lose accuracy if \( \sum_{i=1}^{k} x_i^2 \gg x_{k+1}^2 \)
    - Better to sum in chunks, then sum chunks (recent fix in (Sca)LAPACK)

- Vendors (Intel MKL, NVIDIA cuBLAS, ...) have some reproducibility guarantees
  - Conditional (e.g., data aligned nicely, same # threads, ...)
Reproducibility

• Desire: bitwise identical results, regardless of hardware or summation order
  • ReproK is reproducible accumulator with $K \geq 3$ bins
  • Each bin is 2 floats
  • For reductions using C_in and C_out, we restrict alpha, beta to \{ ±1, 0 \}
  • alpha, beta, A, B do not need to be reproducible accumulators

• Some routines are naturally reproducible, if implemented consistently
  • rot, scal, axpy, ger, ...
Data layout

• Propose both column and row strides
  • Eliminates column-major vs. row-major issue
  • Useful for tensor products
    • 2D slice of 3D tensor may need both strides
  • Use for interleaved & non-interleaved formats
Error checking

• Check dimensions, but return error instead of calling xerbla
  • int BLAS_gemm_r64( ... );
  • In C++, throw exception

• Eliminate use of xerbla
  • xerbla’s Fortran string interface inconvenient for C/C++ users
  • xerbla’s default behavior is halt, instead of modern exception handling
  • Only one instance of xerbla when using multiple libraries

• Wrapper library if xerbla’s *halt* behavior is desired for debugging
Inf & nan propagation

• Some BLAS predates IEEE 754 spec (1985) that introduced inf and nan
• Consistent inf & nan propagation — unlike current implementations
• What is nan?
  • Missing data (as in R)
  • Unknown value (finite)
  • Unknown value (finite or infinite)
  • Unknown value (finite, infinite, or invalid) ⇐ our proposal
Inf & nan propagation

• **iamax**: index of max abs. value
  • Returns index of max value, *unless first value is nan, then it returns 1*
  • `iamax([nan, 0, 2])` ⇒ index 1 (value `nan`)
  • `iamax([0, nan, 2])` ⇒ index 3 (value 2); should be index 2 (value `nan`)
  • Vendor libraries differ from reference & from each other!
  • **We propose returning index of first nan, if any, else index of max value**
Inf & nan propagation

• **trsv**: triangular solve $\text{op}(A) = b$ with 0 in RHS $b$
  
  • Upper triangular case ignores entries in $U$ — nan not propagated!

  $$ x = U^{-1}b = \begin{bmatrix} 2 & \text{nan} \\ \text{nan} & \end{bmatrix}^{-1} \begin{bmatrix} 6 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} 3 \\ 0 \end{bmatrix} $$

  • Lower triangular, transposed case does not ignore entries in $L$ — nan propagated

  $$ x = L^{-T}b = \begin{bmatrix} 2 & \text{nan} \\ \text{nan} & \end{bmatrix}^{-T} \begin{bmatrix} 6 \\ 0 \end{bmatrix} \Rightarrow \begin{bmatrix} \text{nan} \\ \text{nan} \end{bmatrix} $$

  • Vendor libraries differ from reference!

  • **We propose always propagating nan and inf (*)**
Inf & nan propagation

• **ger: rank one update,** \( A = xy^T \)
  - 0 in x (0*nan = nan) versus
    0 in y (0*nan = 0)

\[
\begin{bmatrix}
1 \\
0
\end{bmatrix}
\begin{bmatrix}
3 & \text{nan}
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
3 & \text{nan}
\end{bmatrix}
\]

\[
\begin{bmatrix}
\text{nan} \\
1 \\
0
\end{bmatrix}
\begin{bmatrix}
3 & 0
\end{bmatrix}
\Rightarrow
\begin{bmatrix}
\text{nan} & 0
\end{bmatrix}
\]

• Vendor libraries differ from reference!

• Again, we propose always propagating nan and inf (*)
Inf & nan propagation

• **gemm**: matrix-matrix multiply
  
  • \( \alpha \cdot AB + 0 \cdot C: \quad C = \alpha \cdot AB \), ignores input \( C \) \hspace{1cm} (documented)
  
  • \( 0 \cdot AB + \beta \cdot C: \quad C = \beta \cdot C \), ignores inputs \( A, B \) \hspace{1cm} (not documented)
  
  • \( 0 \cdot AB + 0 \cdot C: \quad C = 0 \), ignores inputs \( A, B, C \) \hspace{1cm} (not documented)

• Vendor libraries follow reference.

• (*) **We propose that scalars = 0 elide argument (current practice), while matrix entries = 0 always propagate nan and inf.**
C++ interface

- Extension of BLAS++ library ([https://bitbucket.org/icl/blaspp](https://bitbucket.org/icl/blaspp))
- Overloaded functions
  - `blas::gemm( ..., double* A, double* B, double* C )` \( \Rightarrow \) `dgemm`
  - `blas::gemm( ..., float* A, float* B, double* C )` \( \Rightarrow \) mixed precision
  - `blas::gemm( ..., double* A, std::complex<double>* B, std::complex<double>* C )` \( \Rightarrow \) mixed real/complex

  - Here, omitting details like col-stride, row-stride
  - Compiler determines correct function based on datatype
  - Type-independent names enables use in templated code
Harmonizing real & complex

- C++ templates need one name to call
- Map complex names (herk) to real names (syrk) for real arithmetic
- ConjTrans = Trans in real

<table>
<thead>
<tr>
<th></th>
<th>real</th>
<th>complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C += AA^T$</td>
<td>syrk</td>
<td>syrk</td>
</tr>
<tr>
<td></td>
<td>$\Rightarrow$ syrk_r(32</td>
<td>64)</td>
</tr>
<tr>
<td>$C += AA^H$</td>
<td>herk</td>
<td>herk</td>
</tr>
<tr>
<td></td>
<td>$\Rightarrow$ syrk_r(32</td>
<td>64)</td>
</tr>
</tbody>
</table>
Harmonizing real & complex

- C++ templates need *one name* to call
- Map complex names (herk) to real names (syrk) for real arithmetic
- ConjTrans = Trans in real

\[
\begin{array}{|c|c|}
\hline
\text{real} & \text{complex} \\
\hline
C += A \ A^T & \Rightarrow \text{syrk}_r(32|64) \\
C += A \ A^H & \Rightarrow \text{syrk}_r(32|64) \\
\hline
\end{array}
\]

```cpp
// Cholesky, overly simplified
template <typename FloatType>
int potrf(Uplo uplo, int64_t n, FloatType* A, int64_t lda)
{
    for (j = 0; j < n; j += nb) {
        blas::herk( ...);
        lapack::potf2( ...);
        blas::gemm( ...);
        blas::trsm( ...);
    }
    return info;
}
```
Harmonizing real & complex

- C++ templates need *one name* to call
- Map complex names (herk) to real names (syrk) for real arithmetic
- ConjTrans = Trans in real

<table>
<thead>
<tr>
<th></th>
<th>real</th>
<th>complex</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C += AA^T$</td>
<td>syrk</td>
<td>syrk</td>
</tr>
<tr>
<td></td>
<td>$\Rightarrow$ syrk_r(32</td>
<td>64)</td>
</tr>
<tr>
<td>$C += AA^H$</td>
<td>herk</td>
<td>herk</td>
</tr>
<tr>
<td></td>
<td>$\Rightarrow$ syrk_r(32</td>
<td>64)</td>
</tr>
</tbody>
</table>
C++ interface

- enums
  - enum class Op { NoTrans, Trans, ConjTrans }
  - enum class Uplo { Lower, Upper }

- C++ exceptions
  - Currently, BLAS++ can be compiled with exceptions or asserts

- Extended precision supports both:
  - Non-interleaved struct-of-arrays:  std::pair< double*, double* > A
  - Interleaved array-of-structs: DoubleDouble* A
Library interface (low-level)

- Examples:
  - \texttt{BLAS\_gemm\_r64} = \texttt{dgemm}
  - \texttt{BLAS\_gemm\_c64} = \texttt{zgemm}
  - \texttt{BLAS\_gemm\_r64c64c64x2} = double * complex + (complex double-double)
  - \texttt{BLAS\_gemm\_r64\_repro3} = \texttt{dgemm}, reproducible (internal)

- \texttt{BLAS\_<function\>_\_<typeSeq>[_\_<precision>[_\_<mult>][_\_<suppl>]]( ... )}
  - \texttt{<function>} := dot | gemm | gemm\_out | gemm\_batch | ...
  - \texttt{<typeSeq>} := <type> | <typeSeq>
  - \texttt{<type>} := (r | c | ...) <precision> [<mult>]
  - \texttt{precision} := 8 | 16 | 32 | 64 | ...
  - \texttt{mult} := x2 | repro3 | ...
  - \texttt{suppl} := repro3 | ...
Feedback

• Extended abstract (4 page)
  • https://tinyurl.com/yb7m7lov

• Proposal (48 page)
  • http://goo.gl/D1UKnw

• BLAS++ and LAPACK++ library
  • Implement C++ wrappers for standard BLAS; to be extended for BLAS G2
  • https://bitbucket.org/icl/blaspp
  • https://bitbucket.org/icl/lapackpp
ICL is hiring!

- Projects include
  - SLATE — ScaLAPACK replacement for distributed dense linear algebra
  - CEED — tensor algebra, batched operations
  - PEEKS — Krylov methods
  - Distributed FFT

- [www.icl.utk.edu/jobs](http://www.icl.utk.edu/jobs)