SAND2019-5665C

Title:

Evolving a Standard C++ Linear Algebra Library from the BLAS

Abstract:

Many different applications depend on linear algebra. This includes machine learning, data mining, web search, statistics, computer graphics, medical imaging, geolocation and mapping, and physics-based simulations. Good implementations of seemingly trivial linear algebra algorithms can perform asymptotically better than bad ones, and can get much more accurate results. This is why authors (not just us!) have proposed adding linear algebra to the C++ Standard Library.
Linear algebra builds on well over 40 years of well-accepted libraries, on which science and engineering applications depend. In fact, there is an actual multiple-language standard for a set of fundamental operations, called the Basic Linear Algebra Subprograms (BLAS). The BLAS Standard defines a small set of hooks that vendors or experts in computer architecture can optimize. Application developers or linear algebra algorithm experts can then build on these hooks to get good performance for a variety of algorithms.
The only thing missing from the BLAS is a modern C++ interface. Nevertheless, its wide availability and many highly optimized implementations make it a good starting point for developing a standard C++ library. In this talk, we will walk through the design iterations of a typical C++ linear algebra library that builds on the BLAS. This process will highlight challenges that BLAS users currently face. It will also show features C++ developers need that the BLAS and its derivatives (like LAPACK) do not provide. We will both argue for inclusion of linear algebra in the C++ Standard Library, and give our views about how simple that library could be and still be useful.

Talk outline:

```
I. What do we mean by "linear algebra"?
   A. Operations on matrices & vectors
   B. This includes both
      1. Products and other "computer science things"
      2. Factorizations & other "numerical analysis things"
II. Why put linear algebra into C++ standard library?
   A. Linear algebra widely used, & growing interest
   B. Codify decades of practice in C++ & other languages
   C. Like sort: Obvious algorithms asymptotically slower
      than the right algorithms
   D. Many opportunities for vendors to add value
III. What is the BLAS and why use it?
   A. BLAS is a standard with a long history (if time allows,
      summarize BLAS development history, and tie to development of
      cache-optimal algorithms)
   B. BLAS has high-performance implementations
   C. Writing a high-performance implementation is nontrivial
   D. Most public C++ linear algebra libraries rely on BLAS for
      performance anyway
IV. Design iterations of a BLAS-based C++ linear algebra library
   A. Motivating use case: Portable high-performance C++ application
      that needs dense matrix-matrix products
      1. Na\"ive triply nested loop is slow
      2. Want minimal external library dependencies
      3. Iterative development process -- group challenges in the
         order they are likely to arise
   B. Start: Use the BLAS from C++
      1. Challenges
         a. Find optimized BLAS implementation
         b. Link to the library
         c. Call BLAS (Fortran or C ABI) from C++
      2. Success criteria
         a. Successfully link to BLAS
         b. Extern "C" declarations of BLAS functions
         c. Portably correct with respect to BLAS ABI
   C. Next: Type safety and genericity
      1. Challenges
         a. Type safety
            i. Correct pointer types
            ii. Complex number ABI issues
            iii. Passing scalar arguments by pointer
         b. Genericity: function names and interfaces
            i. BLAS function names encode type
            ii. Want functions overloaded for different types
            iii. Real vs. complex interface differences
         c. Genericity: BLAS only supports 4 types, but
            C++ developers want more (e.g., short floats)
            i. Much of BLAS only uses plus and times, so those
               functions should work for integers (etc.) too
            ii. Dot products and norms: Modest mathematical
                ambiguity for more "interesting" types
```

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

```
            d. What if BLAS library not available?
         2. Success criteria
            a. Generic and type-safe C++ library, using raw pointers
            b. Fall-back C++ implementation
               i. For unsupported types, or
               ii. if BLAS library not available, or
               iii. to test correctness of Fortran ABI assumptions
                    or of BLAS itself.
      D. Next: C++ data structures & encapsulation
         1. Challenges
            a. Need C++ data structures for matrices & vectors
            b. BLAS functions take many, unencapsulated arguments
               (C++ Core Guidelines I.23, I.24)
         2. Solutions
            a. Take matrices and vectors as mdspan (P0009)
            b. scaled_view to encapsulate scaling factors
            c. transpose_view to encapsulate transpose arguments
            d. mdspan accessor to encapsulate different data layouts
      E. Next: Composition of parallelism
         1. Challenges
            a. Conflicts between BLAS implementation's internal
               parallelism and user's parallelism, even if user does not
               call BLAS from parallel code (e.g., thread pools that
               fight over resources)
            b. User wants to specify execution policy for linear algebra
               operations, just like for C++ standard algorithms
            c. Hierarchical parallelism (calling BLAS from parallel code)
         2. Solutions
            a. Constraint on BLAS implementation and user's parallel
               programming model
            b. Imitate C++ >= 17 algorithms by taking optional execution
               policy argument (also a hook for hierarchical parallelism)
      F. Next: Optimize for tiny matrices and vectors
         1. What does "tiny" mean?
            a. Anything from "cheaper to pass by value" to "fits in cache"
            b. Consider compile-time sizes
         2. Important use case for many fields
         3. Challenge: BLAS interface not designed for tiny data structures
            a. Can't inline through extern "C" interface
            b. Mandatory error checking
            c. Dimensions and other arguments all run time
         4. Solutions
            a. mdspan-based interface
               i. Specialize for compile-time dimensions
               ii. Specialize for transpose, etc. arguments
               iii. Relax error handling requirements
            b. Containers vs. views
               i. mdspan stores a pointer
               ii. overhead, esp. for short types
               iii. mdarray proposal
            c. "Batch" interface
               i. Do many small operations in one call
               ii. Expose more levels of parallelism
               iii. mdspan helpful
               iv. Different hardware vendors have implementations
   V. Taking stock: What do we have, what's missing, and why
      A. We have
         1. Modern C++ interface to the BLAS
         2. Hooks for hierarchical parallelism
         3. Hooks to optimize for tiny matrices
      B. We also have a suggestion for a standard C++ library
         that builds on long industry practice
      C. We don't yet have
         1. LAPACK or related functionality
            a. LAPACK is an implementation, not a standard
            b. Need BLAS-like components for an LAPACK-like library anyway
            c. Standardization can proceed in phases
            d. True genericity of complicated numerical algorithms over
               many number types is hard
            e. Even though LAPACK has "generic" design
         2. Freedom from external BLAS library
            a. Optimizations are nontrivial, but well understood for
               common computer architectures
            b. Opens door to algorithmic optimizations that may affect
               rounding error (e.g., Strassen)
         3. Overloaded arithmetic operators
            a. Developers in some fields (e.g., graphics) expect them
            b. C++ libraries can and do implement operators using BLAS or
               any "named function" interface
            c. What should operator* mean for matrices?
               (C++ Core Guidelines C.167)
```

      d. Invasive to unrelated code (ADL)
   4. Optimizations that fuse multiple operations into one
      a. Why fuse operations?
         i. Maximize data reuse
         ii. Amortize parallel launch overhead
      b. C++ technique: expression templates
         i. Over 20 years of C++ prior art
         ii. Often used by C++ linear algebra libraries
      c. Why not expression templates?
         i. "auto" and dangling references problem
         ii. Fusing operations not always a win
         iii. Encourages overloading arithmetic operators
            - Invasive to unrelated code (ADL)
            - What should operator* mean for matrices?
         iv. Poor precedent in Standard Library
            - valarray permits but does not mandate them
              (proposals to add them were rejected)
         v. Existing features could cover many use cases
            - mdspan accessors could be used for expression
              templates, analogously to David Vandevoorde's
              valarray proposal, but without any more risk of
              dangling references than mdspan itself admits
            - Ranges (as they evolve to support e.g., execution
              policies)
      d. Tension with orthogonality / single responsibility
         i. C++ Core Guidelines F.2
         ii. BLAS matrix multiply (C = beta*C + alpha*A*B)
             combines addition and multiplication
         iii. Context: this is exactly what LAPACK's LU
              factorization needs for trailing matrix update
         iv. BLAS' unexpected beta=0 overwrite rules result in
             part from functions having multiple responsibilities;
             our design permits harmless aliasing to avoid these
D. Extra credit challenges
   1. Optimizations for types not supported by BLAS
   2. Does mixing precisions, real and complex, etc.
      do the right thing?
   3. Some kinds of number types (e.g., fixed point)
      may call for a different linear algebra interface