P1673R3: A free function linear algebra interface based on the BLAS

Mark Hoemmen & coauthors

mhoemmen@stellarscience.com

LEWG meeting, 22 June 2021

P1673(R3) current status

- Reviewed by SG6, SG14, LEWGI, & LEWG
- Targeting IS
- Depends on P0009 (+ P2299 changes)
- Implementation
  - https://github.com/kokkos/stdBLAS
  - Design like coauthors' existing libraries (e.g., kokkos-kernels)
  - Builds on decades of coauthors' implementation experience
- Wording reviewed by wordsmith / coauthor Dan Sunderland
- New draft: https://github.com/ORNL/cpp-proposals-pub/blob/master/D1673/blas_interface.md

Why does P1673 belong in the Standard Library?

- Linear algebra algorithms are like `sort`
  - Obvious algorithms are slow and inaccurate
  - Fastest call for hardware-specific tuning
- Core functionality for many applications
  - At least as useful as the “mathematical functions” already in the Standard Library
- Builds on decades of existing practice
  - Including an actual standard (BLAS)…
  - …and implementations from many vendors

Design summary

- Algorithms working on views of data
  - Use mdspan (P0009) to view multidimensional arrays
  - Otherwise, like existing standard algorithms
- Algorithms, not containers
  - No matrix/vector operator arithmetic (no "C = A * B")
  - Express matrix properties like symmetry as algorithms' assumptions about data, not as a class hierarchy
- Generic algorithms for any element types
  - Integers, short or extended floats, fixed-point, polynomials, crazy custom math types, …
  - Can mix precisions in the same algorithm

Linear algebra has layers

Image credit: Lali Masriera (Barcelona, Catalunya)
https://en.wikipedia.org/wiki/File:Cortando_cebolla.jpg

Abstraction layers of linear algebra

- Layer -1: Multidimensional arrays, iteration, …
- Layer 0: Computational kernels
  - Vector-vector ops: dot, norm, vector sum, apply plane rotation, …
  - Matrix-vector ops: matrix-vector multiply, triangular solve, outer product update, …
  - Matrix-matrix ops: matrix-matrix multiply, triangular solve with multiple vectors, low-rank / symmetric update, …
- Layer 1: Solve low-level math problems
  - Linear systems $Ax = b$ (& determinants etc.)
  - Least-squares problems $\min_x \|Ax - b\|$
  - Eigenvalue & singular value problems $Ax = \lambda x$
- Layer 2: Solve higher-level math problems
  - Nonlinear system of partial differential equations
  - Solve huge problem by projecting onto small Layer 1

Abstraction layers of linear algebra

P0009, P1684, Parallelism TS v2, …

- Layer -1: Multidimensional arrays, iteration, …
- Layer 0: Computational kernels
  - Vector-vector ops: dot, norm, vector sum, apply plane rotation, …
  - Matrix-vector ops: matrix-vector multiply, triangular solve, outer product update, …
  - Matrix-matrix ops: matrix-matrix multiply, triangular solve with multiple vectors, low-rank / symmetric update, …

P1673

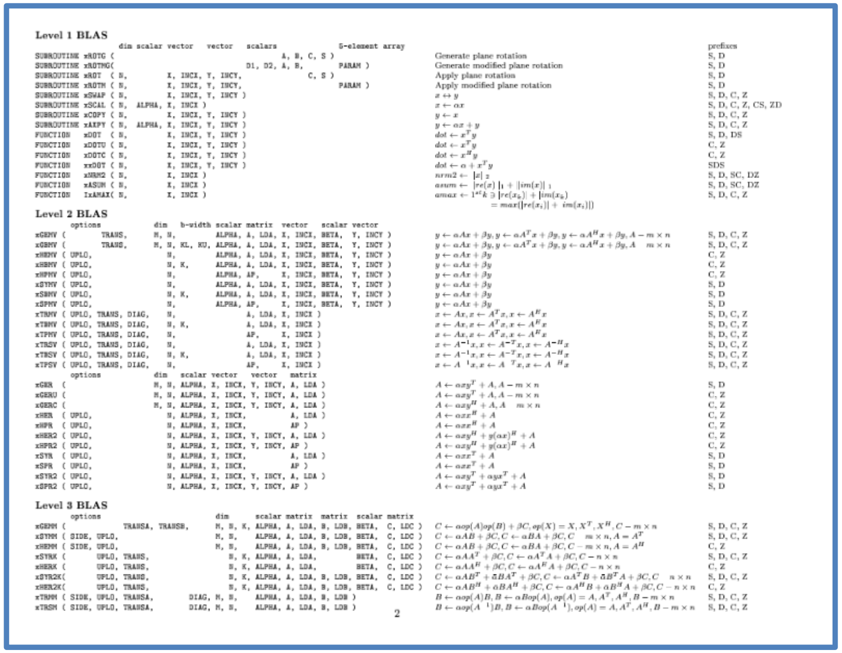
- Layer 1: Solve low-level math problems
  - Linear systems $Ax = b$ (& determinants etc.)
  - Least-squares problems $\min_x \|Ax - b\|$
  - Eigenvalue & singular value problems $Ax = \lambda x$

other C++ libraries (no standard yet)

- Layer 2: Solve higher-level math problems
  - Nonlinear system of partial differential equations
  - Solve huge problem by projecting onto small Layer 1

Basic Linear Algebra Subprograms

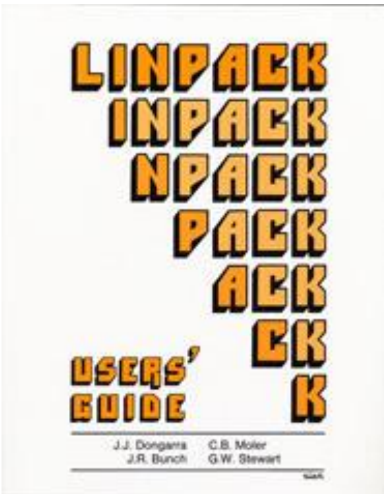
- Standard published 2002
  - 1995-99 meetings
  - Fortran & C interfaces
  - Dense matrix & vector ops
- Developed in levels (1,2,3):
  - Vector-vector (BLAS 1): 1979
  - Matrix-vector (BLAS 2): 1988
  - Matrix-matrix (BLAS 3): 1990
  - Level ➔ more data reuse
- Implementations by many system vendors, e.g.,
  - AMD, ARM, IBM, Intel, NVIDIA, Xilinx (FPGA)
- Open-source impl.s too



(Fortran) BLAS quick reference:
http://www.netlib.org/blas

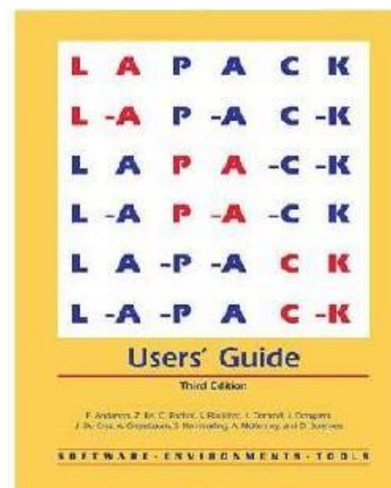
See also Jack Dongarra interview:
http://history.siam.org/oralhistories/dongarra.htm

BLAS codesigned w/ algorithms



- LINPACK library: 1979
  - General dense, symmetric, & banded
  - “Layer 1” algorithms (linear systems & linear least squares)
  - Designed to use BLAS (1), for good performance on many different computers

- LAPACK: 1990
  - Combines functionality of LINPACK + EISPACK ({eigen,singular}value problems)
  - “Coreleased” w/ BLAS 3; common authors
  - Algorithms w/ optimal data reuse (*)
  - BLAS 3 was designed for those algorithms



(*) I’m simplifying: see our paper, “Communication lower bounds & optimal algorithms for numerical linear algebra” (Acta Numerica 2014).

P1673 design lesson: Layer!

- Standardize in layers
  - Multidim. arrays (P0009)
  - Computations (BLAS, P1673)
  - Then actual math algorithms
- Layer by developers' expertise
  - System vendors write P1673
  - So mathematicians can math
- Layer for performance portability
  - BLAS has architecture-specific optimizations
  - So math algorithms can have portable implementations

Layering improves longevity, just like the Dobos torte
(Image credit: Wikipedia)

P1673 works on views

- BLAS and P1673 both work on views
  - matrix_product(A, B, C)
  - Not "C = A * B"
- Higher-level algorithms depend on different steps viewing the "same matrix" in different ways, e.g.,
  - Rectangular submatrices during LU factorization
  - Upper and lower triangles during linear system solve
- Most interesting operations are not elementwise
  - e.g., matrix-matrix multiply
  - Expression templates won't help avoid allocation
  - P1673 has many more algorithms than {+, *} (C++ != APL)
- "C = A * B" hinders optimizing larger problems

Goldilocks & the 3 problem sizes

- Small
  - ≤ 4x4? Fits in a register?
  - Cheap copying favors op arithmetic (C = A * B)
- Large
  - Need 64-bit dimensions? Won't fit in memory?
  - Specialized algorithms and data structures (my field)
  - Which often have medium-sized subproblems
- Medium (BLAS)
  - Like sort: enough work that algorithms matter
  - Avoid {allocate, copy}; work on views
  - "Large" methods don't pay
- P1673: Medium to small
  - Remove BLAS' error checking (narrow contract)
  - Exploit mdspan's compile-time dimensions
  - Can add batched overloads

P1673 is extensible

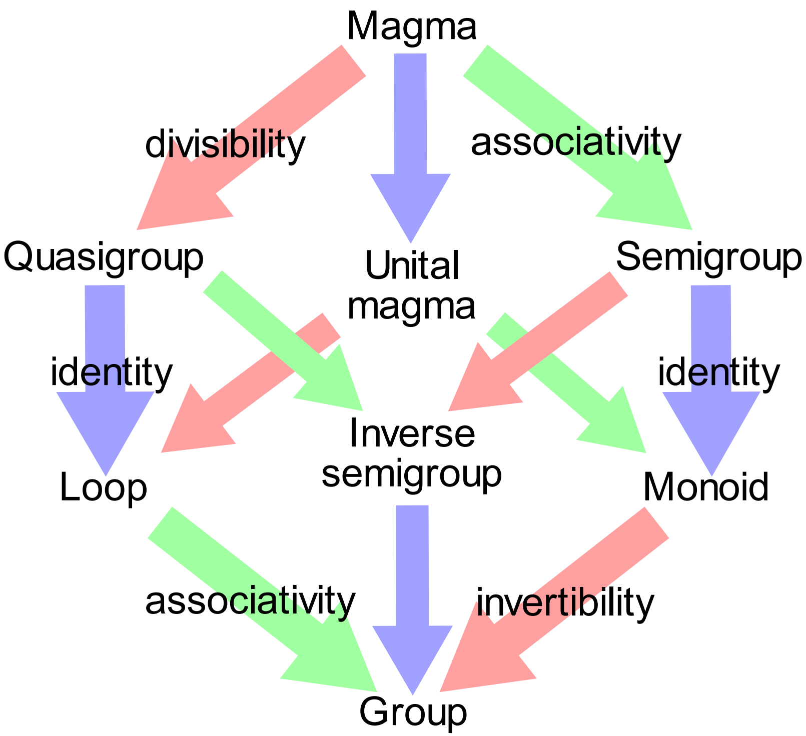
- Algorithms are independent of `mdspan` layouts and accessors, so easy to add more
  - Tiled or recursive layouts for better locality
  - Accessors for heterogeneous computing
- `ExecutionPolicy&&` overloads
  - Control parallelism (reordering assumptions)
  - Extra context: scheduling queue, scratch space
- Easy to add "batched" overloads
  - Batched: Solve many same-size problems at once
  - Best way to vectorize & parallelize small problems
  - Easy with `mdspan`: extra extent + different layout

Addressing LEWG R2 feedback

- Introduce std::linalg namespace
  - Use it to replace linalg_ prefix
- Remove _view suffix
  - Consider it reserved for ranges
  - conjugate_view -> conjugated, etc.
- Investigate "concept-ification"
  - As in P1813R0 for numeric algorithms like reduce
  - Added discussion to R3; see following slides
- Investigate support for GPU memory
- Wording & other improvements

P1813R0 concept-ification

- Define concepts that describe generalizations of a (math) group
- Use them to constrain elements of algorithms' range parameters
- e.g., **reduce** requires <u>associative</u> op+
  - (a + b) + c == a + (b + c)



Algebraic structures between magmas & groups (Image credit: Wikipedia)

Associativity is too strict

- There are many useful arithmetic systems with nonassociative operator+
  - With infinity: If 10 is Inf, 3+8-7 is either Inf or 4
  - Saturating: If 10 is max, 3+8-7 is either 3 or 4
  - With rounding (e.g., floating point)
- Concepts are “always”; users accept “usually”
  - Linear algebra has many examples of algorithms (e.g., matrix factorizations) that don’t succeed for all (run-time) input

Generalizing associativity is not useful for our algorithms

- P1813R0's most general concept: <u>magma</u>
  - Set M with binary operation * such that
  - if x and y are in M, then x * y is in M ("closed")
- Sounds general, but already too specific
  - Assumes only one set (i.e., type) M
    - No mixed precision or expression templates
    - Example: y(i) = alpha * x(i) + y(i) (could have 5 types!)
  - Assumes binary operation is closed
    - What if it could throw or otherwise fail?
    - e.g., rational with bounded number of digits; signaling NaN

Need adverbs >> adjectives

- Constraints are adjectives
  - Properties of ranges' element types
  - Useful to us mainly to describe what compiles, or how to write a custom element type that works
- We need adverbs
  - Permissions that users give an algorithm
  - Regardless of properties of ranges' element types
  - C++ Standard already has GENERALIZED_SUM

Support for discrete GPUs

- GPU: Graphics processing unit
- Algorithm w/ the right ExecutionPolicy could access memory that C++ ordinarily could not
  - "Inaccessible memory" not Standard C++
  - Straightforward extension in practice
  - Long history outside GPUs (PGAS)
- Asynchronous execution of algorithms
  - Different interface, esp. for return value
  - LEWG asked us to consider return by pointer

Return scalars by value, for now

- Earlier reviews (SG6, SG14, LEWGI) insisted that we return scalar results by value, like **reduce**
  - Original design had output (by ref) arguments
  - Made batched overloads more natural
- Just writing to a pointer isn't enough to express asynchronous execution
  - How do we know if the value is ready?
  - Async algorithms need to track data flow too
- Standard interface isn't settled yet
  - Senders / receivers model (P0443, P1897, P2300) would permit composing asynchronous work

P0009 & P2299 changes

- Recent P0009 (mdspan) design changes
  - e.g., P2299's CTAD improvements
  - basic_mdspan -> mdspan
- P1673R3 predates those changes
  - Next revision will incorporate them
  - Draft: https://github.com/ORNL/cpp-proposals-pub/blob/master/D1673/blas_interface.md

Summary

- Our proposal P1673 is a C++ BLAS interface
- BLAS: established standard with a long history
- P1674 explains how we built up our library proposal P1673 from BLAS + mdspan (P0009)
- Please also read P1417 for historical context
- Thanks especially to:
  - Daisy Hollman
  - Alicia Klinvex
  - Nevin Liber
  - Christian Trott
- & to my employer, Stellar Science