SAND2019-11113C

P1673: A proposal for a C++ Standard linear algebra library

Mark Hoemmen (mhoemme@sandia.gov)

CppCon, SG14 meeting, 18 Sep 2019

SAND2019-XXXX C (UUR)

U.S. DEPARTMENT OF ENERGY NNSA National Nuclear Security Administration

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.

P1673 is a C++ BLAS binding

Sandia National Laboratories

- Basic Linear Algebra Subroutines
  - > 40 years' practice; 2002 Standard w/ Fortran & C interfaces
  - "…standard building blocks for performing basic vector and matrix operations" (netlib.org/blas/)
  - Many optimized implementations
- Free functions taking mdspan (P0009) or mdarray (P1684)
  - mdspan : matrix algorithms :: iterators/ranges : existing std algorithms
- P1673 wraps & extends BLAS
  - Same functions, but more supported element types, layouts, etc.
  - Impls. can call BLAS if it supports matrix element type & layout
  - Impls. can optimize more cases, like tiny compile-time dimensions
- SG14 voted in Cologne to move P1673 to LEWG

Key features of P1673

Sandia National Laboratories

- Extensible
  - Collection of useful low-level standard algorithms
  - Is not & need not be a total solution to linear algebra
  - If you like operator* etc., can build that on top of this
  - Can grow, just like BLAS did (BLAS 1, 2, 3, Standard)
- Standardizes existing practice
  - Many C++ linear algebra libraries use & wrap BLAS
- Low effort for vendors to optimize P1673
  - Most already have an optimized BLAS library; just wrap it!
  - Vendor coauthors: AMD, ARM, NVIDIA, & Intel
- Parallelizable & customizable via ExecutionPolicy
- Strategy for vectorization / SIMD (batched + layout)

Builds on several WG21 papers

Sandia National Laboratories

- P1674: Design justification (please read first!)
  - Evolution of a C++ linear algebra library from “raw” BLAS
  - Explains common practice in many higher-level libraries & apps
- P1417: History of linear algebra libraries
- P0009 (mdspan) & P1684 (mdarray)
  - Multidimensional arrays w/ customizable layout & access
  - Can mix compile-time & run-time dimensions
  - Views (mdspan) & containers (mdarray)
  - P0009 at revision >= 9 & in LWG review
- P1673: Linear algebra proposal itself (don’t start here!)

What's linear algebra?

We focus on building blocks for computations, not math

Levels of linear algebra

Sandia National Laboratories

- Level -1: Data structures & iteration for matrices, vectors, …
- Level 0: Computational kernels (mostly + and *, some /)
  - Vector-vector ops: dot, norm, vector sum
  - Matrix-vector ops: matrix-vector multiply, triangular solve
  - Matrix-matrix ops: matrix-matrix mult, tri solve, low-rank update
- Level 1: Solve low-level math problems
  - Linear systems $Ax = b$ (& determinants etc.)
  - Least-squares problems $\min_x \|Ax - b\|$
  - Eigenvalue & singular value problems $Ax = \lambda x$
- Level 2: Solve higher-level math problems
  - Nonlinear / time-dependent system of partial differential equations
  - Approximate a huge problem by projecting onto a small Level 1

Levels of linear algebra

Sandia National Laboratories

P0009, P1684, Parallelism TS v2, …

- Level -1: Data structures & iteration for matrices, vectors, …
- Level 0: Computational kernels (mostly + and *, some /)
  - Vector-vector ops: dot, norm, vector sum
  - Matrix-vector ops: matrix-vector multiply, triangular solve
  - Matrix-matrix ops: matrix-matrix mult, tri solve, low-rank update

P1673

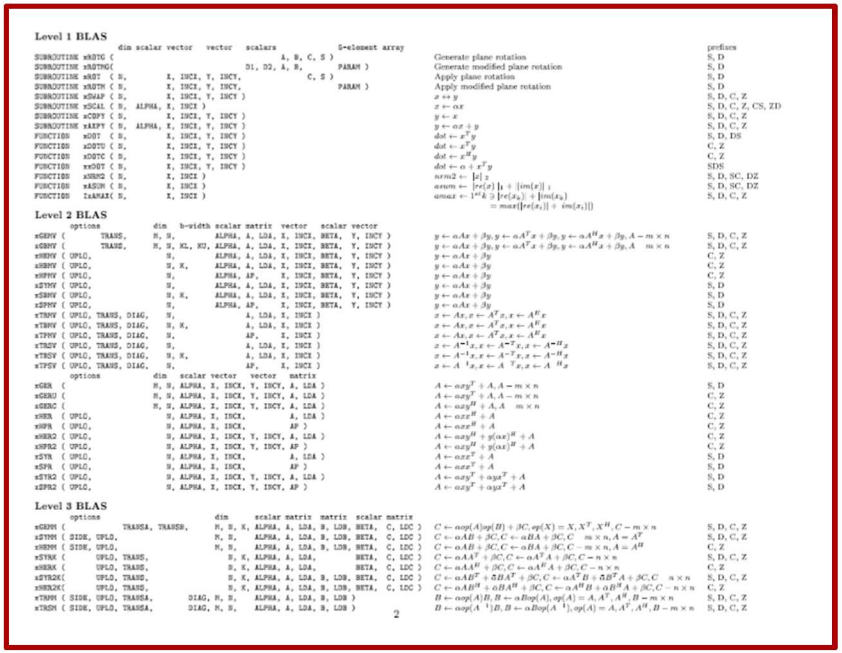
- Level 1: Solve low-level math problems
  - Linear systems $Ax = b$ (& determinants etc.)
  - Least-squares problems $\min_x \|Ax - b\|$
  - Eigenvalue & singular value problems $Ax = \lambda x$

other C++ libraries (no standard yet)

- Level 2: Solve higher-level math problems
  - Nonlinear / time-dependent system of partial differential equations
  - Approximate a huge problem by projecting onto a small Level 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
  - Higher level ➔ more data reuse
- Many vendor impl's, e.g.,
  - AMD, ARM, IBM, Intel, NVIDIA



Level 1 BLAS

	dim	scalar	vector	vector	scalars			5-element array		
SUBROUTINE xROTG (							A, B, C, S)		Generate plane rotation	S, D
SUBROUTINE xROTMG(						D1, D2, A, B,		PARAM)	Generate modified plane rotation	S, D
SUBROUTINE xROT (	N,		X, INCX,	Y, INCY,			C, S)		Apply plane rotation	S, D
SUBROUTINE xROTM (	N,		X, INCX,	Y, INCY,				PARAM)	Apply modified plane rotation	S, D
SUBROUTINE xSWAP (	N,		X, INCX,	Y, INCY)					$x \leftrightarrow y$	S, D, C, Z
SUBROUTINE xSCAL (	N,	ALPHA,	X, INCX)						$x \leftarrow \alpha x$	S, D, C, Z, CS, ZD
SUBROUTINE xCOPY (	N,		X, INCX,	Y, INCY)					$y \leftarrow x$	S, D, C, Z
SUBROUTINE xAXPY (	N,	ALPHA,	X, INCX,	Y, INCY)					$y \leftarrow \alpha x + y$	S, D, C, Z
FUNCTION xDOT (	N,		X, INCX,	Y, INCY)					$dot \leftarrow x^T y$	S, D, DS
FUNCTION xDOTU (	N,		X, INCX,	Y, INCY)					$dot \leftarrow x^T y$	C, Z
FUNCTION xDOTC (	N,		X, INCX,	Y, INCY)					$dot \leftarrow x^H y$	C, Z
FUNCTION xxDOT (	N,		X, INCX,	Y, INCY)					$dot \leftarrow \alpha + x^T y$	SDS
FUNCTION xNRM2 (	N,		X, INCX)						$nrm2 \leftarrow \|x\|_2$	S, D, SC, DZ
FUNCTION xASUM (	N,		X, INCX)						$asum \leftarrow \|re(x)\|_1 + \|im(x)\|_1$	S, D, SC, DZ
FUNCTION IxAMAX(	N,		X, INCX)						$amax \leftarrow 1^{st} k \ni \|re(x_k)\| + \|im(x_k)\| = \max(\|re(x_i)\| + \|im(x_i)\|)$	S, D, C, Z

Level 2 BLAS

	options	dim	b-width	scalar	matrix	vector	scalar	vector		
xGEMV (	TRANS,	M, N,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
xGBMV (	TRANS,	M, N, KL, KU,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A \quad m \times n$	S, D, C, Z
xHEMV (	UPLO,	N,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xHBMV (	UPLO,	N, K,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xHPMV (	UPLO,	N,		ALPHA,	AP,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xSYMV (	UPLO,	N,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xSBMV (	UPLO,	N, K,		ALPHA,	A, LDA,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xSPMV (	UPLO,	N,		ALPHA,	AP,	X, INCX,	BETA,	Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xTRMV (	UPLO, TRANS, DIAG,	N,			A, LDA,	X, INCX)			$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTBMV (	UPLO, TRANS, DIAG,	N, K,			A, LDA,	X, INCX)			$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTPMV (	UPLO, TRANS, DIAG,	N,			AP,	X, INCX)			$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTRSV (	UPLO, TRANS, DIAG,	N,			A, LDA,	X, INCX)			$x \leftarrow A^{-1} x, x \leftarrow A^{-T} x, x \leftarrow A^{-H} x$	S, D, C, Z
xTBSV (	UPLO, TRANS, DIAG,	N, K,			A, LDA,	X, INCX)			$x \leftarrow A^{-1} x, x \leftarrow A^{-T} x, x \leftarrow A^{-H} x$	S, D, C, Z
xTPSV (	UPLO, TRANS, DIAG,	N,			AP,	X, INCX)			$x \leftarrow A^{-1} x, x \leftarrow A^{-T} x, x \leftarrow A^{-H} x$	S, D, C, Z

	options	dim	scalar	vector	vector	matrix		
xGER (		M, N,	ALPHA, X, INCX,	Y, INCY,	A, LDA)		$A \leftarrow \alpha xy^T + A, A - m \times n$	S, D
xGERU (		M, N,	ALPHA, X, INCX,	Y, INCY,	A, LDA)		$A \leftarrow \alpha xy^T + A, A - m \times n$	C, Z
xGERC (		M, N,	ALPHA, X, INCX,	Y, INCY,	A, LDA)		$A \leftarrow \alpha xy^H + A, A \quad m \times n$	C, Z
xHER (	UPLO,	N,	ALPHA, X, INCX,		A, LDA)		$A \leftarrow \alpha xx^H + A$	C, Z
xHPR (	UPLO,	N,	ALPHA, X, INCX,		AP)		$A \leftarrow \alpha xx^H + A$	C, Z
xHER2 (	UPLO,	N,	ALPHA, X, INCX,	Y, INCY,	A, LDA)		$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$	C, Z
xHPR2 (	UPLO,	N,	ALPHA, X, INCX,	Y, INCY,	AP)		$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$	C, Z
xSYR (	UPLO,	N,	ALPHA, X, INCX,		A, LDA)		$A \leftarrow \alpha xx^T + A$	S, D
xSPR (	UPLO,	N,	ALPHA, X, INCX,		AP)		$A \leftarrow \alpha xx^T + A$	S, D
xSYR2 (	UPLO,	N,	ALPHA, X, INCX,	Y, INCY,	A, LDA)		$A \leftarrow \alpha xy^T + \alpha yx^T + A$	S, D
xSPR2 (	UPLO,	N,	ALPHA, X, INCX,	Y, INCY,	AP)		$A \leftarrow \alpha xy^T + \alpha yx^T + A$	S, D

Level 3 BLAS

	options		dim		scalar	matrix	matrix	scalar	matrix		
xGEMM (		TRANSA, TRANSB,	M, N, K,		ALPHA, A, LDA, B, LDB,		BETA,	C, LDC)	$C \leftarrow \alpha op(A) op(B) + \beta C, op(X) = X, X^T, X^H, C - m \times n$	S, D, C, Z
xSYMM (	SIDE, UPLO,		M, N,		ALPHA, A, LDA, B, LDB,		BETA,	C, LDC)	$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C, C \quad m \times n, A = A^T$	S, D, C, Z
xHEMM (	SIDE, UPLO,		M, N,		ALPHA, A, LDA, B, LDB,		BETA,	C, LDC)	$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C, C - m \times n, A = A^H$	C, Z
xSYRK (	UPLO, TRANS,			N, K,	ALPHA, A, LDA,		BETA,	C, LDC)	$C \leftarrow \alpha AA^T + \beta C, C \leftarrow \alpha A^T A + \beta C, C - n \times n$	S, D, C, Z
xHERK (	UPLO, TRANS,			N, K,	ALPHA, A, LDA,		BETA,	C, LDC)	$C \leftarrow \alpha AA^H + \beta C, C \leftarrow \alpha A^H A + \beta C, C - n \times n$	C, Z
xSYR2K(	UPLO, TRANS,			N, K,	ALPHA, A, LDA, B, LDB,		BETA,	C, LDC)	$C \leftarrow \alpha AB^T + \beta BA^T + \beta C, C \leftarrow \alpha A^T B + \beta B^T A + \beta C, C \quad n \times n$	S, D, C, Z
xHER2K(	UPLO, TRANS,			N, K,	ALPHA, A, LDA, B, LDB,		BETA,	C, LDC)	$C \leftarrow \alpha AB^H + \alpha BA^H + \beta C, C \leftarrow \alpha A^H B + \alpha B^H A + \beta C, C - n \times n$	C, Z
xTRMM (	SIDE, UPLO, TRANSA,	DIAG, M, N,			ALPHA, A, LDA, B, LDB)				$B \leftarrow \alpha op(A) B, B \leftarrow \alpha B op(A), op(A) = A, A^T, A^H, B - m \times n$	S, D, C, Z
xTRSM (	SIDE, UPLO, TRANSA,	DIAG, M, N,			ALPHA, A, LDA, B, LDB)				$B \leftarrow \alpha op(A^{-1}) B, B \leftarrow \alpha B op(A^{-1}), op(A) = A, A^T, A^H, B - m \times n$	S, D, C, Z

2

(Fortran) BLAS quick reference:
http://www.netlib.org/blas
(See also Jack Dongarra's oral history)

BLAS 1-3 coevolved w/ computers

Seymour Cray w/ Cray 1, circa 1976 (when LINPACK funding started)

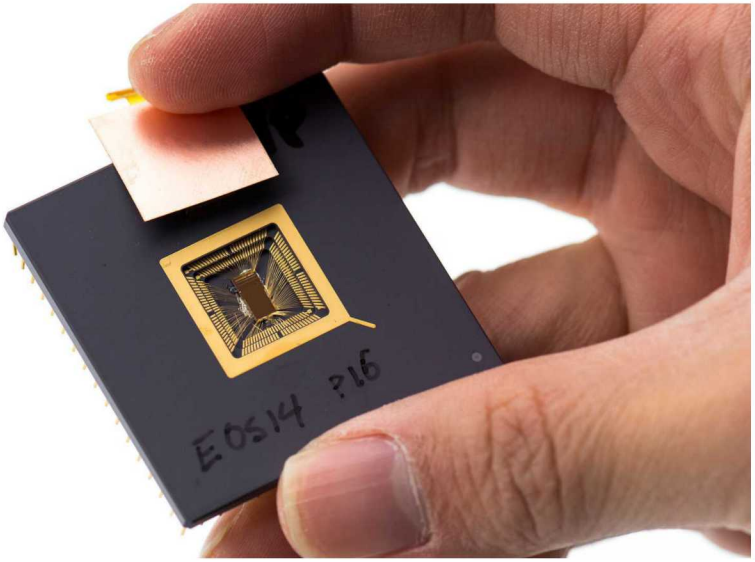
Vector (Cray, NEC)

- Low flop/byte ratio
- Favor long, dependence-free loops & regular data access
- BLAS 1 target

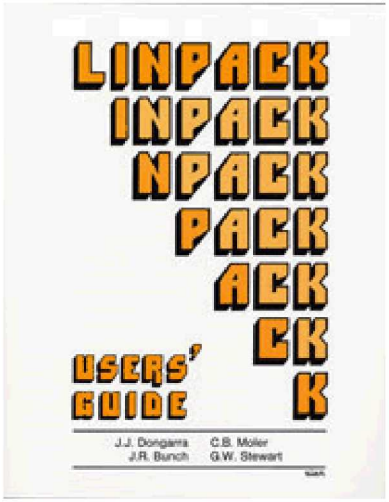
Cache based

- "Killer micros" (see 1991 NYT article)
- High flop/byte ratio
- Favor data reuse
- BLAS 3 target

Yunsup Lee holding RISC V prototype, 2013

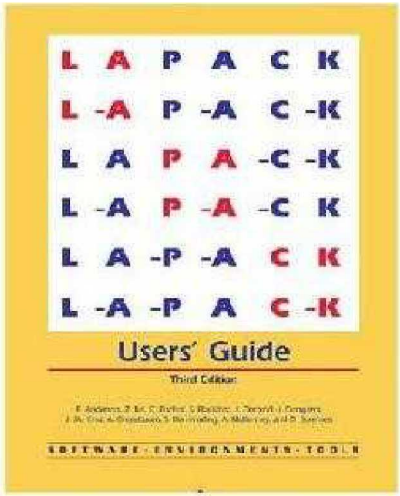


BLAS codesigned w/ algorithms



- LINPACK library: 1979
  - General dense, symmetric, & banded
  - Linear systems (LU, Cholesky)
  - Linear least squares (QR)
  - 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, w/ common authors
  - Algorithms that better exploit data reuse
  - BLAS 3 designed for those algorithms



Imitate BLAS' std-ization approach

Sandia National Laboratories

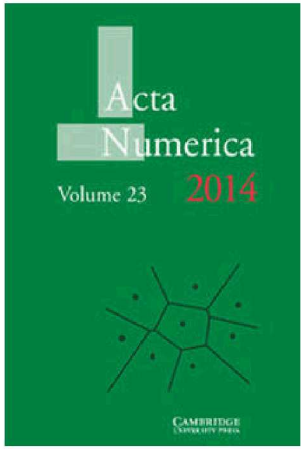
- Standardize in layers, bottom up
  - Lowest: Multidimensional arrays, SIMD types & ops
  - Lower: Fundamental algebra of matrices & vectors
  - Higher: Linear systems, least squares, eigenvalue problems, …
- Layer by developers' expertise
  - BLAS: hardware / performance; LAPACK: numerical analysis
  - Kazushige Goto (talented BLAS optimizer) is self-taught [1]
  - Std Lib implementers not typically trained in numerical analysis
- Layer for performance portability
  - Design higher levels to spend most time in lower levels
  - Multiple implementations add more value at lower levels
  - Vendors can tune lower levels for specific hardware

BLAS designed for optimal algs.

Sandia National Laboratories

- BLAS designed for LAPACK's algorithms
  - Will future algorithms need different fundamental operations?
  - Would a bottom-up approach risk overspecializing the lowest level?

- OK: LAPACK-like algorithms are optimal
  - Last 2 decades: Many theoretical results on algorithm optimality (matrix & tensor), so unlikely to need radical interface changes
  - Lowest level useful, whether or not matrix factorization algorithms change radically
  - Participation by more communities (e.g., embedded, graphics, machine learning) will reduce overspecification risk



See "Communication lower bounds & optimal algorithms for numerical linear algebra," Ballard, Carson, Demmel, Hoemmen, Knight, & Schwarz, 2014

SG14: Don’t rely on inlining

Sandia National Laboratories

- P1832: Avoid relying on inlining for decent performance
  - Debug builds shouldn’t be too slow
  - Reduce build times & sizes (debug symbols) (Big deal for us too!)
    - e.g., build failures due to compiler / linker running out of memory
- P1673 advantages
  - Design encourages vendors to precompile complex optimized code
  - matrix_product: if constexpr “Can I call BLAS?”, else easy loops
  - Minimize dependence on inlining in inner loops
    - Indexing: typical mdspan inline depth 2-3
    - Success w/ inlining in production code (see P3HPC submission to SC19)
    - Standardizing mdspan makes special-case inlining possible, like std::vector::operator[]
    - Avoid expression templates (free functions only)

More SG14 feedback

Sandia National Laboratories

- Users must be able to change back-end if slow
  - P1673: Customize via ExecutionPolicy overloads
  - Compare to classic C++ linear algebra technique: “Engine”
- Sometimes need to optimize for tiny problems
  - Batched: Expose parallelism / vectorization over multiple problems
    - P1673 option: Pack as extra md{span,array} dimension
    - mdspan Layout for contiguous / coalesced access
    - e.g., w/ LayoutLeft, A_batch(p,i,j), A_batch(p+1,i,j) contiguous
    - mdspan Accessor for any needed decorations (e.g., restrict, alignment)
  - Micro (“µBLAS”): solve one problem at a time
    - Typical game developer use cases: {4x4,3x3} matrices, {3,4}x1 vectors
    - P1673: mdarray (P1684) interface, but see Questions

Q: Tiny interface (µBLAS)

Sandia National Laboratories

- P1673 imitates existing standard algorithms
  - mdspan : matrix algorithms :: iterators/ranges : std algorithms
  - Iterate over data structures; don't return deep copies
  - Matches BLAS: No dynamic allocation ➔ don't "return a new matrix"
- What about tiny vectors & matrices (e.g., fit in void*)?
  - mdspan must store at least a pointer (it's a view)
  - P1673 approach: Pass mdarray by reference (input & output)
    - Goal: Uniform interface for problems of all sizes
    - Assumes compiler can optimize mdarray<T, extents<N>> like fixed_size_simd<T, N> (or we could simply take *simd types)
    - SIMD TS passes input arguments by reference, but returns by value
- Q: Do we need a different interface for tiny objects?
  - If so, can we defer this to future SIMD TS addition?

Q: Thin BLAS

Sandia National Laboratories

- P1673: BLAS 1 overlaps w/ standard algorithms
  - We plan to add range adapters for rank-1 md{span,array}
    - Separate paper add-on to P0009
    - Would make transform, reduce, copy, max_element, etc. work
  - But, users like traditional names, & vendors can add value (e.g., accuracy, reproducibility) to dot, nrm2, …
  - Rank-N? Iterators not the best idiom for multidimensional arrays
    - Compilers good at optimizing nested for loops with integer indices
    - Multi-D iterators have more state; compilers would need "retraining"
    - Multi-D iterators would delay standardization w/out new functionality
- Q: How much of the BLAS 1 belongs in C++?

Summary

Sandia National Laboratories

- P1673 key features
  - Standardizes 40 years of BLAS & 25+ years of C++ BLAS interfaces
  - Vendors have low effort to optimize P1673: Wrap existing BLAS
  - Not a total solution to linear algebra; can grow, just like BLAS did
  - Parallel & customizable via ExecutionPolicy, like std algorithms
  - SIMD solution: Batched + custom Layout + custom Accessor
- Questions for SG14
  - Need different interface for tiny problems (µBLAS)?
  - BLAS 1 has useful names & functionality, but overlaps with existing standard algorithms; how much overlap should we permit?

Extra slides

Sandia National Laboratories

Problems with current BLAS

Sandia National Laboratories

- It's not C++, it's Fortran
  - Fortran name mangling & ABI not portable & vary widely
  - C BLAS exists but not universally available
- Not templated; only works w/ 4 element types
  - float, double, complex<float>, complex<double>
  - Assumes Fortran & C++ complex have same bit representation
  - C BLAS takes void* for complex values & arrays
  - Machine learning wants half precision, etc.
- Data layout & storage restricted
  - Fortran BLAS only takes column-major arrays
- No way to interface w/ C++ executors
- Hard to use: takes a bazillion integers & raw pointers
  - Matlab exists because BLAS & LINPACK were hard to use

Alternate procedural approaches

Sandia National Laboratories

- Outside WG21 & C++ Standard Library
- Add C++ binding to BLAS Standard?
  - (+) Meeting minutes show they planned a C++ interface [2]
  - (o) C++ & Java excluded: “time constraints” & possible Java changes
  - (+) CBLAS exists & has support from multiple vendors
  - (-) Invading small (< 20) focused committee, inactive nearly 20 years
  - (-) WG21 linear algebra interests may differ a lot from theirs
  - (-) May reject C++ binding if too semantically different (compare: MPI rejected original C++ interface; result satisfied few & was deprecated)
- Form our own standard?
  - Experience from Batched BLAS & GraphBLAS: this process more useful for organizing research collaborations, than for agreeing on a standard interface

Fundamental or batteries included?

Sandia National Laboratories

- See [3,4]: Fundamental-ness influences WG21 prioritization
- Lowest level of linear algebra: Clearly fundamental, including
  - Multidimensional arrays: P0009, mdspan, at LWG level
  - SIMD types & ops: Voted into Parallelism TS 2
- What makes them fundamental?
  - Need / would benefit from compiler support
  - Vocabulary
  - Multiple use cases

Multidimensional arrays: mdspan

Sandia National Laboratories

- P0009, currently in LWG wording review
- Meant as a zero-overhead abstraction
  - Like Fortran arrays, complete w/ slices
  - Can mix compile- & run-time dimensions
  - Compiler could optimize indexing
- Polymorphic data layout
  - Needed for performance with tensors & batched small dense
  - Performance portability (tune layout for architecture)
  - Compatibility with different libraries in other languages
- Polymorphic storage & access
  - Path forward for heterogeneous computing (memory spaces)

Is mdspan a “matrix”?

Sandia National Laboratories

- Useful array features
  - Encapsulates BLAS arguments like dimensions & strides
  - Slices, like Matlab or NumPy
  - Efficient array access that compilers could optimize
- mdspan is a view [views]
  - Can’t create (allocate) matrix
  - But see mdarray (P1684)
- Not a matrix; usable by it
  - mdspan lacks mathematical structure, but so does BLAS
  - Users responsible for math

“Room above” mdspan:

- Owning tensor / mat / vec classes
- Dispatch to mdspan kernels
- Build user-friendly expressions atop mdspan & kernels

HPC's priorities

Sandia National Laboratories

- May not align 100% w/ graphics, etc., but we think…
- …we have compatible interests, esp. w/ layered approach
- Lower-level primitives would address most of our concerns
  - e.g., free functions for matrix & vector algebra that take mdspan
- Higher-level expressions
  - Expression templates are not a priority for us, but…
  - …we do not oppose them & think they are complementary

Conclusions

Sandia National Laboratories

- If we want to standardize a C++ linear algebra library,
- we should learn from BLAS Standard's successes:
  - Standardize in layers, from the bottom up
  - Layer by expertise: hardware / performance vs. numerical analysis
- Identify fundamentals for any linear algebra / tensor library
  - SIMD types & ops: Voted into N4744 (Parallelism TS 2)
  - Multidimensional arrays: P0009, in LWG wording review
  - Fundamental algebra of matrices & vectors: This week :-D
- Thanks to P1417 coauthors & all of you!

References

- [1] John Markoff, “Writing the Fastest Code, by Hand, for Fun: A Human Computer Keeps Speeding Up Chips,” *New York Times*, Nov. 28, 2005.
- [2] BLAST Forum MINUTES, NIST, Washington, D.C., Oct. 8-9, 1998. Available online: http://www.netlib.org/blas/blast-forum/blast-forum-minutes.oct8-9.98.html [last accessed Feb. 14, 2019].
- [3] Guy Davidson, “Batteries not included: what should go in the C++ standard library?”, *World of hatcat*, Feb. 16, 2018. Available online: https://hatcat.com/?p=16 [last accessed Feb. 10, 2019].
- [4] Titus Winters, “What Should Go Into the C++ Standard Library,” *Abseil Blog*, Feb. 27, 2018. Available online: https://abseil.io/blog/20180227-what-should-go-stdlib [last accessed Feb. 10, 2019].