SAND2019-8470PE

*B*asic*L*inear*A*lgebra*S*ubprogram for C++

- **P1674r0: Evolving a Standard C++ Linear Algebra Library from the BLAS**
  - Extensive design justification and background for existing BLAS
- **P1673R0: A free function linear algebra interface based on the BLAS**
  - The design proposal for BLAS functions in the C++ standard

This is not much talking about P1385R1: **A proposal to add linear algebra support to the C++ standard library**

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and 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-NA-0003525.

Authors
•Mark Hoemmen (mhoemme@sandia.gov) (Sandia National Laboratories)
•David Hollman (dshollm@sandia.gov) (Sandia National Laboratories)
•Christian Trott (crtrott@sandia.gov) (Sandia National Laboratories)
•Daniel Sunderland (dsunder@sandia.gov) (Sandia National Laboratories)
•Nevin Liber (nliber@anl.gov) (Argonne National Laboratory)
•Siva Rajamanickam (srajama@sandia.gov) (Sandia National Laboratories)
•Li-Ta Lo (ollie@lanl.gov) (Los Alamos National Laboratory)
•Graham Lopez (lopezmg@ornl.gov) (Oak Ridge National Laboratories)
•Peter Caday (peter.caday@intel.com) (Intel)
•Sarah Knepper (sarah.knepper@intel.com) (Intel)
•Piotr Luszczek (luszczek@icl.utk.edu) (University of Tennessee)
•Timothy Costa (tcosta@nvidia.com) (NVIDIA)
Contributors
•Chip Freitag (chip.freitag@amd.com) (AMD)
•Bryce Lelbach (blelbach@nvidia.com) (NVIDIA)
•Srinath Vadlamani (Srinath.Vadlamani@arm.com) (ARM)
•Rene Vanoostrum (Rene.Vanoostrum@amd.com) (AMD)

Outline

- Why should BLAS be in the C++ standard?
  - Existing practice?
  - What are the issues with that practice?
- A proposal for function based BLAS
  - General design principal.
  - Analogy to std::algorithms

Why should BLAS be in the standard?

- C++ applications in "important application areas" (see [P0939R0](http://wg21.link/p0939r0)) have depended on linear algebra for a long time.
- Linear algebra is like `sort`: obvious algorithms are slow, and the fastest implementations call for hardware-specific tuning.
- Dense linear algebra is core functionality for most of linear algebra, and can also serve as a building block for tensor operations.
- The C++ Standard Library includes plenty of "mathematical functions." Linear algebra operations like matrix-matrix multiply are at least as broadly useful.
- The set of linear algebra operations in this proposal are derived from a well-established, standard set of algorithms that has changed very little in decades. It is one of the strongest possible examples of standardizing existing practice that anyone could bring to C++.
- This proposal follows in the footsteps of many recent successful incorporations of existing standards into C++, including the UTC and TAI standard definitions from the International Telecommunications Union, the time zone database standard from the International Assigned Numbers Authority, and the ongoing effort to integrate the ISO unicode standard.

Why should BLAS be in the standard?

"Directions for ISO C++" (P0939R0) offers the following in support of adding linear algebra to the C++ Standard Library:

- P0939R0 calls out "Support for demanding applications in important application areas, such as medical, finance, automotive, and games (e.g., key libraries...)" as an area of general concern that "we should not ignore." All of these areas depend on linear algebra.
- "Is my proposal essential for some important application domain?" Many large and small private companies, science and engineering laboratories, and academics in many different fields all depend on linear algebra.
- "We need better support for modern hardware": Modern hardware spends many of its cycles in linear algebra. For decades, hardware vendors, some represented at WG21 meetings, have provided and continue to provide features specifically to accelerate linear algebra operations. For example, SIMD (single instruction multiple data) is a feature added to processors to speed up matrix and vector operations. P0214R9, a C++ SIMD library, was voted into the C++20 draft. Several large computer system vendors offer optimized linear algebra libraries based on or closely resembling the BLAS; these include AMD's BLIS, ARM's Performance Libraries, Cray's LibSci, Intel's Math Kernel Library (MKL), IBM's Engineering and Scientific Subroutine Library (ESSL), and NVIDIA's cuBLAS.

Existing Standard

- Most widely used is BLAS
- Standardization effort started in 1995
  - Based on 40 years of linear algebra work (see P1417R0)
- Fortran Library
- Many hardware vendors ship their own optimized variants of the standard
  - Intel: MKL, NVIDIA: CuBLAS, IBM: ESSL, …

Major Problems with the existing standard

- It is Fortran
- Limited Scalar type support
- No Layouts, compile dimensions or inlining
- No path to integrate executors.

Problems with the existing standard: *It is Fortran*

- To use BLAS one typically does:
  - Figure out how to link against it
    - Depending on library this is non trivial due to options such as threading model, integer types (32bit or 64bit), compiler version etc.
    - Intel has a website to generate link line: https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor
  - Write external C function declarations to interface to the Fortran functions if not provided by vendors
    - SUBROUTINE dgemm(TRANSA,TRANSB,M,N,K,ALPHA,A,LDA,B,LDB,BETA,C,LDC)
    - extern "C" void **F77_BLAS_MANGLE**(dgemm,DGEMM)(const char*, const char*,int*,int*, int*, const double* ,const double* ,int*, const double*, int*, const double*, double*, int*);
    - F77_BLAS_MANGLE: deal with random underscores and capitalization depending on compiler used …
    - Hope that no other library introduced the same functions ….

Problems with the existing standard: *Limited Scalar Type Support*

- Scalar types are not handled via overloads:
  - ***d***gemm => double
  - ***s***gemm => float
  - ***c***gemm => complex<float>
  - ***z***gemm => complex<double>

- No way to handle other scalar types such as are proposed now in p1468
- No way to handle mixed precision
- For complex we rely on same bit representation of Fortran complex numbers and std::complex …

Problems with the existing standard: *No Layouts, compile time dimensions or inlining*

- BLAS expects the pointers for a matrix to be layed out as column major
  - double A[N][M]; is row major
  - Sometimes “trick” the BLAS function via its “transpose” parameters
  - Using submatrices from tensors doesn’t work generally
- The functions take runtime dimensions (via pointers even) and can’t be inlined
  - This makes it unsuitable for small linear algebra operations such as used by gaming

Problems with the existing standard: *No path for integration of executors*

- One can link against a threaded BLAS library or a scalar one
  - You can’t decide at runtime which to use
- Since Fortran will never know about executors no way to interface this in the future
- With hardware being more heterogeneous and more hierarchical (think Sockets, Cores, Threads, SIMD lanes) control over which resources execute a BLAS function is critical
- Current problem: C++ std::thread and the typical OpenMP runtime of Fortran BLAS get in each others way
  - In HPC: don’t use std::thread but OpenMP directives in the application!

To Operator Overload Or Not

- Operator overloading as proposed in … feels more natural
- But it is hard to express all operations we desire:
  - E.g. outer vs inner product: what does A*B do?
- The data types themselves would need to contain executors in the future (and executor adaptors)

BUT: There is no conflict!

Operator overloading can be implemented on top of a functional interface!

Why functional interface?

- Some function are needed anyway (not enough logical operator choices)
- Most widely used interfaces are function based (the ones provided by vendors)
- Can mirror C++ standard parallel algorithms for controlling execution resources
- Like algorithms we can make the algorithm itself a customization point: no need to define the entire machinery behind it
- Could serve as low level layer below an operator based interface

Our proposal in a nutshell

BLAS

```
void dgemv(const char* trans_A, int* M, int* N, const double* alpha,
           const double* A, int* LDA, const double* x, int* incx,
           const double* beta, double* y, int* incy);
dgemv('N', 4, M, 3.0, A, 4, x, 1, 0.0, y, 1);
```

p1673

```
mdspan<float,4,  dynamic_extent> A(…);
mdspan<double, dynamic_extent> x(…),y(…);
// y = 3.0 * A * x;
matrix_vector_product(par, scaled_view(3.0, A), x, y);
```

- ***New type of algorithms***
  - Like algorithms take (optional) execution policies
  - mdspan (or mdarray) instead of iterators to represent matrices and vectors
- ***Scalar scaling parameters and conjugation***
  - functions returning basic_mdspan with special accessors
- ***Transposition of matrices***
  - Functions returning basic_mdspan with special layout

Addressing the major issues of the BLAS standard

- It is Fortran
  - => This is not
- Limited Scalar type support
  - => Templated on all input/output arguments
- No Layouts, compile dimensions or inlining
  - => mdspan provides layouts and compile time dimensions
  - => It is C++ template functions and thus could be inlined
- No path to integrate executors
  - => Follow whatever happens with parallel algorithms

Comparison with P1385

P1673

- Algorithms Analog
- Function Based
- Relative large common user API surface
  - Most of the common Linear Algebra capabilities
- Algorithms are customization points
- Both Small (compile time sizes) and Large operations well supported

P1385

- Math Objects with operator overloading
- Relative small common user API surface
  - Only most fundamental Linear Algebra capabilities
- Large amount of interlocking customization points
- Both Small (compile time sizes) and Large operations well supported

All the algorithms

- givens_rotation_setup
- givens_rotation_apply
- linalg_swap
- scale
- linalg_copy
- linalg_add
- dot
- vector_norm2
- vector_abs_sum
- vector_idx_abs_max
- [symmetric_|hermitian_|triangular_]matrix_vector_product
- triangular_matrix_vector_solve
- [symmetric_|hermitian_]matrix_rank_[1/2]_update
- [symmetric_|hermitian_]matrix_product
- [symmetric_|hermitian_]matrix_rank_2k_update
- triangular_matrix_vector_solve

Helperfunctions etc.

- transposed_view
- scaled_view
- conjugated_view
- layout_blas_general
- layout_blas_packed

The Future: Executors and Batched

- Introduce Executors as well as senders and receivers the same way as in algorithms
  - Control where to execute and potentially merge operations

```
dot(matrix_vector_product (par_unseq.on(CPU),  par_unseq.on(GPU_Exec), A,x) , y);
```

- Batched Operations could be supported by simply increasing the rank of arguments
  - Should cover a decent chunk of basic functions for ML

```
mdspan<float, dynamic_extent, 4, 4> A(…);
mdspan<double, dynamic_extent, 4> x(…),y(…);
// y = 3.0 * A * x;
matrix_vector_product(par, scaled_view(3.0, A), x, y);
```