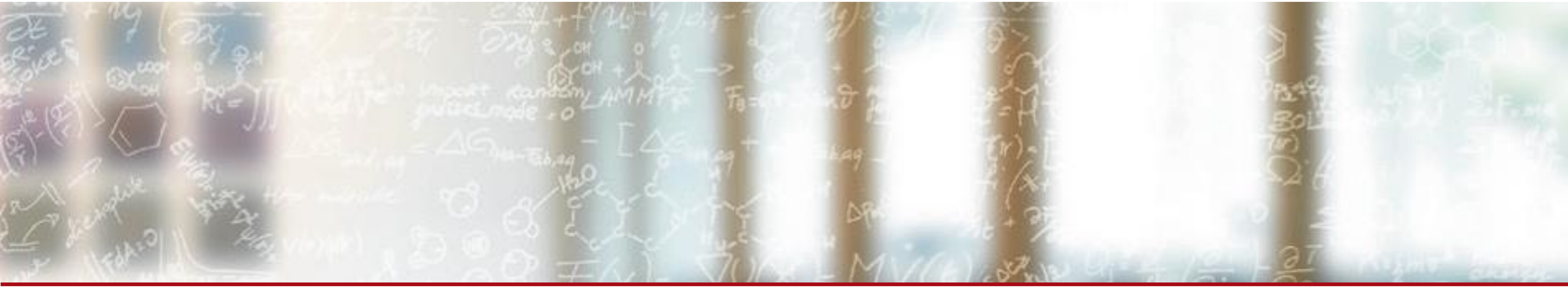
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



std::BLAS

C++ course
Raffaele Solcà, CSCS
September 02, 2024

28.9 Basic linear algebra algorithms

https://eel.is/c++draft/linalg

- What is covered?
  - BLAS Level 1
  - BLAS Level 2
  - BLAS Level 3

- What is not covered?
  - LAPACK
  - PBLAS
  - ScaLAPACK

Example: GEMM

- CBLAS API:
  - cblas_gemm (layout, transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc);

Provided by mdspan

$$C \leftarrow \beta C + \alpha \, \mathrm{Op}(A) \, \mathrm{Op}(B)$$

Also provided by extensions of mdspan

- C++ API
  - std::linalg::matrix_product(A, B, C);

$$C \leftarrow AB$$

  - std::linalg::matrix_product(A, B, E, C);

$$C \leftarrow E + AB$$

mdspan: new features

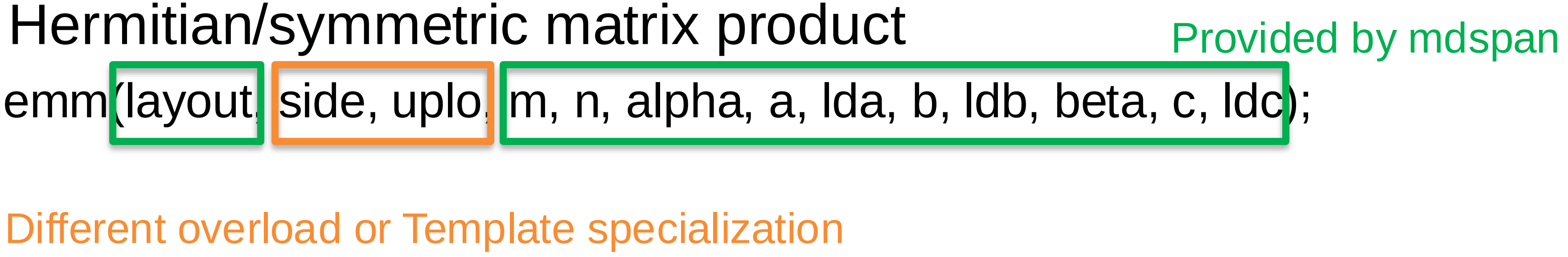
- Some addition to std::mdspan
  - `std::linalg::conjugated(A);`
  - `std::linalg::transposed(A);`
  - `std::linalg::conjugate_transposed(A);`

  - `std::linalg::scaled(alpha, A);`
    - Note: difference of `std::linalg::scaled` v.s. `std::linalg::scale`

  - They return a read-only mdspan with a different type, layout and accessor. No operations performed.

- C++ API covers all the cblas GEMM cases, e.g.:
  - `matrix_product(scaled(alpha, transposed(A)), B, scaled(beta, C), C);`
- and even more (mixed precision, …).

Can we get rid of all extra parameters?

- Consider Hermitian/symmetric matrix product
  - 
    cblas_hemm(layout, side, uplo, m, n, alpha, a, lda, b, ldb, beta, c, ldc);

    Provided by mdspan

    Different overload or Template specialization
- C++ API:
  - Left:
    hermitian_matrix_product(A, t, B, C);
    hermitian_matrix_product(A, t, B, E, C);
  - Right:
    hermitian_matrix_product(A, B, t, C);
    hermitian_matrix_product(A, B, t, E, C);

There is still “diag”

Need different name

Provided by mdspan

- Triangular solver
  - cblas_trsm(layout, side, uplo, trans, diag, m, n, alpha, a, lda, b, ldb);

Different overload or Template specialization

- C++ API:
  - Left:
    triangular_matrix_matrix_left_solve(A, t, d, B)
    triangular_matrix_matrix_left_solve(A, t, d, B, divide)
  - Right:
    triangular_matrix_matrix_right_solve(A, t, d, B)
    triangular_matrix_matrix_right_solve(A, t, d, B, divide)

Warning:
upper/lower_triangle_t refers to the input matrix after any transpose() operation has been applied.

BLAS: ‘L’, ‘T’
CBLAS: CblasLower, CblasTrans
C++: transpose(A), upper_triangle

Summarizing

- matrix (pointer, size, leading dimension, transpose, conjugate_transpose): mdspan
- scaling factors: `linalg::scaled`
- side: overload or different function name
- uplo and diag:

  -
    ```
    struct upper_triangle_t;
    inline constexpr upper_triangle_t upper_triangle;
    struct lower_triangle_t;
    inline constexpr lower_triangle_t lower_triangle;

    struct implicit_unit_diagonal_t;
    inline constexpr implicit_unit_diagonal_t implicit_unit_diagonal;
    struct explicit_diagonal_t;
    inline constexpr explicit_diagonal_t explicit_diagonal;
    ```

BLAS v.s. std::linalg

- BLAS API has only runtime parameters
- C++ API:
  - matrix sizes and leading dimensions can be decided at compile time or at runtime,
  - scaling factors at runtime,
  - BLAS side, uplo, trans and diag only at compile time.

Beneficial for small sizes.

Want possibility to decide at runtime?
=> requires multiple instantiations of the code, therefore larger libraries / executables.

Massive template usage might lead to longer compilation time.

First example

- A simple implementation of Cholesky.

https://github.com/eth-cscs/cpp-course-2024/blob/main/stdBLAS/examples/cholesky.cpp#L57-L109

Quick look to reference GEMM

https://github.com/kokkos/stdBLAS/blob/main/include/experimental/__p1673_bits/blas3_matrix_product.hpp#L655-L732

Performance

- Don’t expect any performance from the reference implementation.
  - An attempt to include a BLAS call to gemm has be commented out.
    - Hopefully calls to BLAS will become available.
    - However not all the calls can be mapped to BLAS. Be aware of performance penalties.

- Reference implementations are bad. Do not even include basic optimizations such as loop reordering.

- Reference implementations are not pre-compiled. Compiling your code with **no optimizations** (e.g. for debugging) means really bad performance.

Some experiments with GEMM

https://github.com/eth-cscs/cpp-course-2024/blob/main/stdBLAS/examples/gemm.cpp#L528-L594

- Reference GEMM

https://github.com/eth-cscs/cpp-course-2024/blob/main/stdBLAS/examples/gemm.cpp#L59-L62

- Loop reordering 3 cases. Best inner loop:
  - Column major:
    - NN: i, NT: i, TN: k, TT: no
  - Row major:
    - NN: j, NT: k, TN: j, TT: no

https://github.com/eth-cscs/cpp-course-2024/blob/main/stdBLAS/examples/gemm.cpp#L64-L190

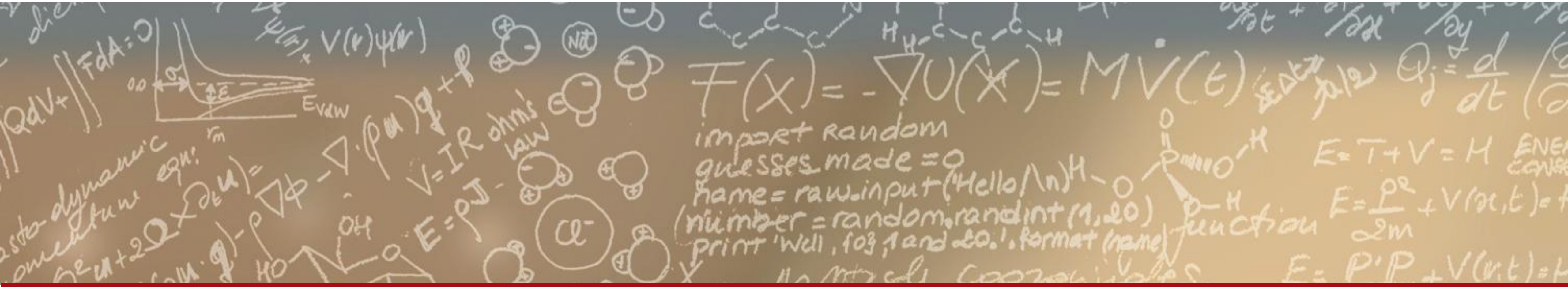
- Calling BLAS

https://github.com/eth-cscs/cpp-course-2024/blob/main/stdBLAS/examples/gemm.cpp#L330-L376

Summary

- Performance not on par with expectations.
- Compilation without optimizations: 100x – 250x slower.
- Adding basic optimizations such as loop re-ordering is not straightforward.
  - Different layout (or transpose) might need a different variant.

- The standard is not finalized yet.
  - Fixed BLAS3 rank-k reference implementation (was completely wrong).
  - Found errors in the accepted standard wording.
  - Some proposals to refine the interface are still open:
    - https://isocpp.org/files/papers/P3371R0.html BLAS3 rank-k and rank-2k API (breaking) change
    - https://isocpp.org/files/papers/P3050R1.html Optimize conjugated for non-complex types
    - https://isocpp.org/files/papers/P3222R0.html Introduce layout-[left, right]-padded



Thank you for your attention.