Use of C++ in Computational Science Libraries and Applications

Organizers:

Piotr Luszczek, Mark Hoemmen, Heike Jagode, Damien Genet

- MS22
  - Interfacing Dense Linear Algebra Libraries in C++, **Piotr Luszczek**, UTK
  - Kokkos Libraries and Applications, **Christian Trott**, Sandia
  - Benchmarking Modern C++ Abstraction Penalty, **Marcin Zalewski, Andrew Lumsdaine**, U. WA
  - Modern C++ in Computation Science, **David Hollman**, Sandia
- MS57
  - Data Flow Graph Programming for High-performance Scientific Computing in C++, **Edward Valeev**, V Tech (travel issues)
  - The Simulation Development Environment (SDE): A C++ Framework for Reusable Computational Chemistry, **Ryan Richard**, Ames Lab (travel issues)
  - Automated Fortran-C++ Bindings for Scientific Applications, **Seth Johnson**, ORNL

Interfacing Dense Linear Algebra Libraries in C++

Piotr Luszczek

(A view from the Foothills of Smokey Mountains)

Monday February 25, 2019

History and Overview

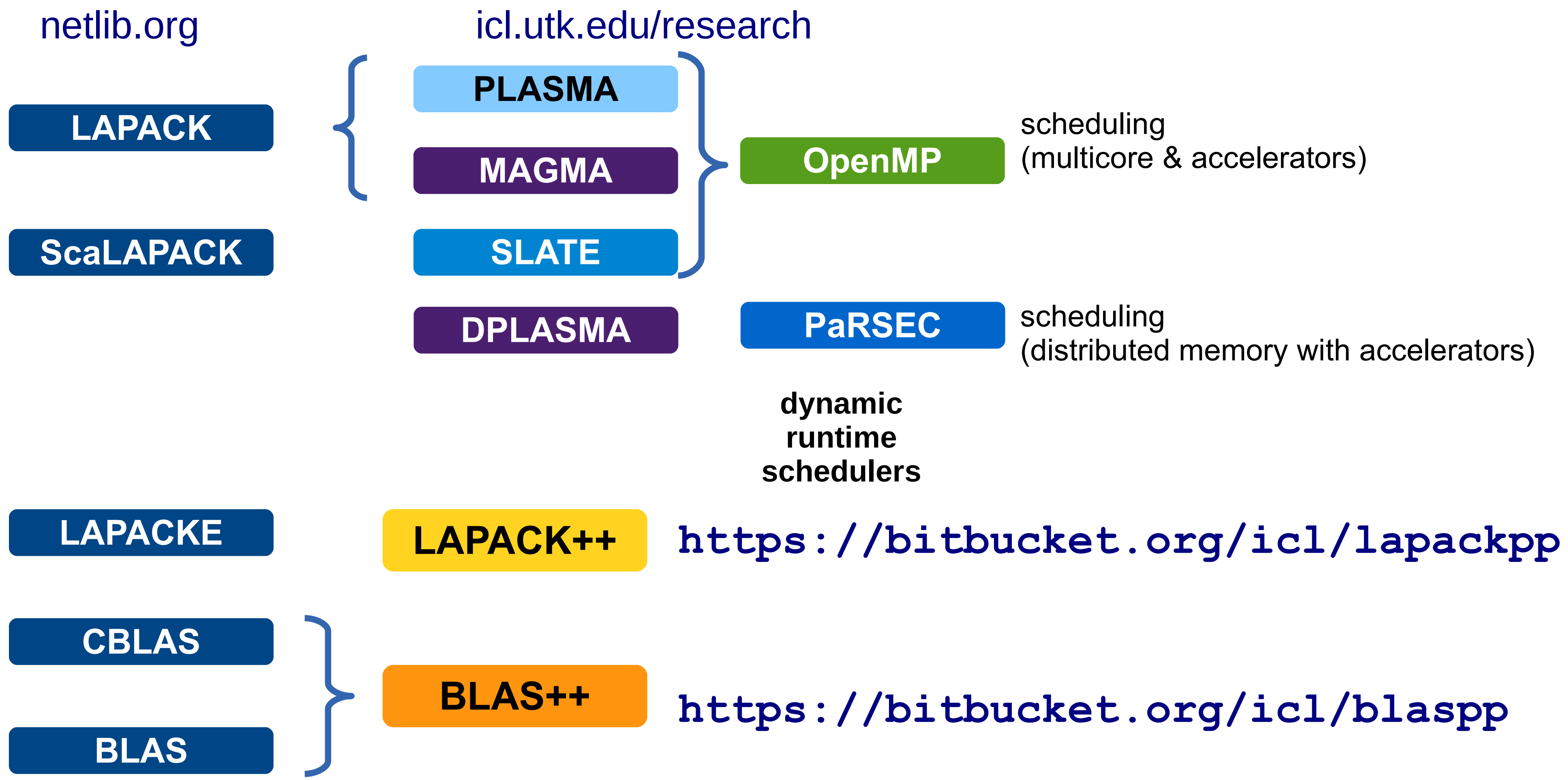
History of C++ for Numerical Linear Algebra

- 1993: LAPACK++
- 1993: ScaLAPACK++
- 1994..: a large number of libraries were create for linear algebra in C++
  - http://www.netlib.org/utk/people/JackDongarra/la-sw.html
- ..Few libraries are still maintained
- MPI C++ binding rejected
- MPI C++ binding created
- MPI C++ binding abandoned
- More history: C++ committee document P1417
- Guy Davidson and Bob Steagall, *Towards Standardization of Linear Algebra*
  - See later summary of P1385

wg21.link/p1417

wg21.link/p1385

Major C++ Efforts in Dense Linear Algebra



BLAS++ and LAPACK++

C++ Software and API for Dense Linear Algebra

- BLAS++
  - https://bitbucket.org/icl/blaspp
- Batched BLAS++
  - https://bitbucket.org/icl/bblaspp
- LAPACK++
  - https://bitbucket.org/icl/lapackpp
- SLATE
  - http://www.icl.utk.edu/publications/series/swans

- ScaLAPACK++
  - LAWN 61 netlib.org/lapack/lawnspdf/lawn61.pdf
  - DOI: 10.1109/SPLC.1993.365563 · IEEE Xplore, Proceedings of the Conference: Scalable Parallel Libraries Conference, 1993 (25 years ago)
  - Microsoft HPC Pack
- PBLAS++
  - “in progress”
- BLACS++
  - “in progress”

BLAS++ Design Principles

- Lessons learned
  - BLAS classic: F..77
  - BLAST F..90 interface
  - CBLAS
    - Netlib
    - ATLAS
    - MKL
      - MKL_DIRECT_CALL
  - BLAS G2
    - gemm_r64()

- namespace blas {
  template <typename FloatType>
  gemm(..);
  }
- Stateless interface
  - No more XERBLA()
- Error propagation
  - blas_error_if(invalid_input)
    - Must be cheap if there are no errors to keep production runs free from overheads:
      - Pipeline stalls due to branch instructions
      - Stack unwinding code when exceptions are desired

LAPACK++ design principles

- LAPACK Wrappers
  - LAPACKE
  - LAPACK++
    - Used by ScaLAPACK++
- LAPACK Translations
  - CLAPACK
    - ATLAS
    - Netlib

- SLATE’s LAPACK++ wrapper
  - Do not underestimate the volume of LAPACK code
    - Nearly 400 source files!
    - Function declarations: 12k lines
    - Wrapper declarations: 10k lines
  - Automation is your friend
- Functionality
  - Name mangling
  - FLOP counts
  - Data types

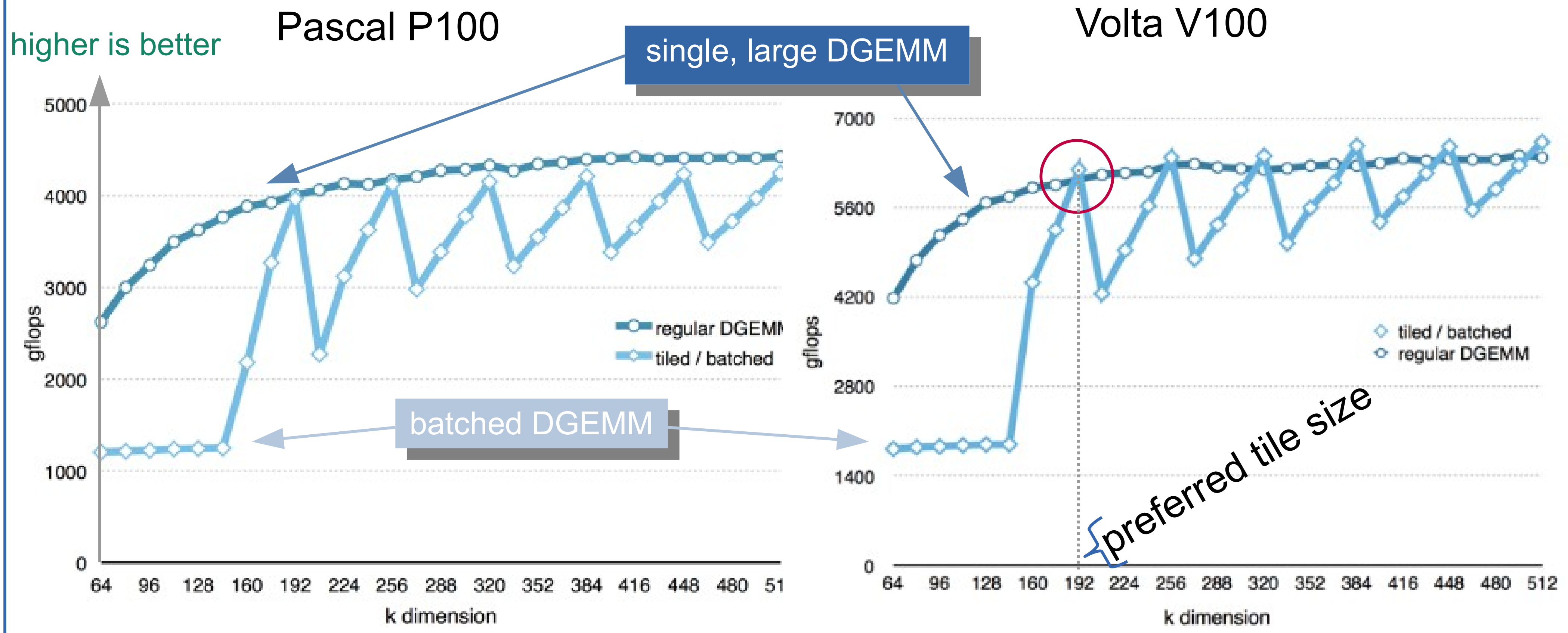
Batched Interface

Batched BLAS++ Design Principles

- Lessons learned
  - AMD hipBLAS and rocBLAS
  - NVIDIA cuBlasBatched
    - DGETRI → cublasDmatinvBatched()
  - MKL Batched
    - Group interface
    - Packed GEMM
  - MAGMA Batched
    - Variable size interface

- namespace blas {
  - namespace batch {
    template <typename FloatType>
    gemm(vector<..> const);
    }
- }
- Fixed and variable sizes inside a batch
  - Uses std::vector::size() to detect
- Error checking more complicated
  - No errors (production runs)
  - One error (all successful)
  - One error code for each matrix in batch

Schur Complement Performance and GEMM Efficiency

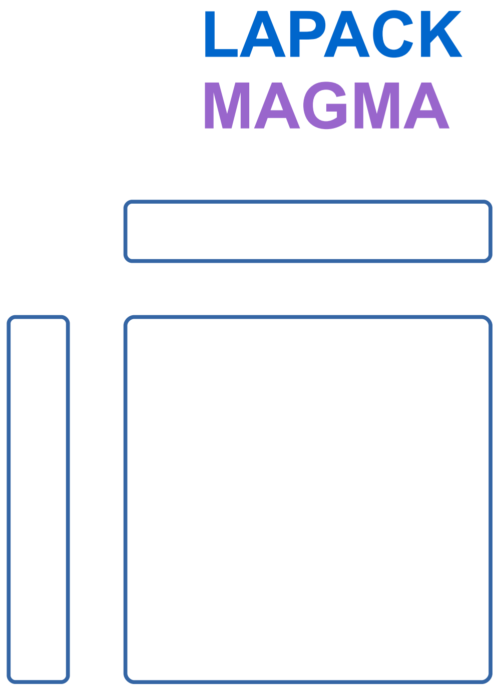


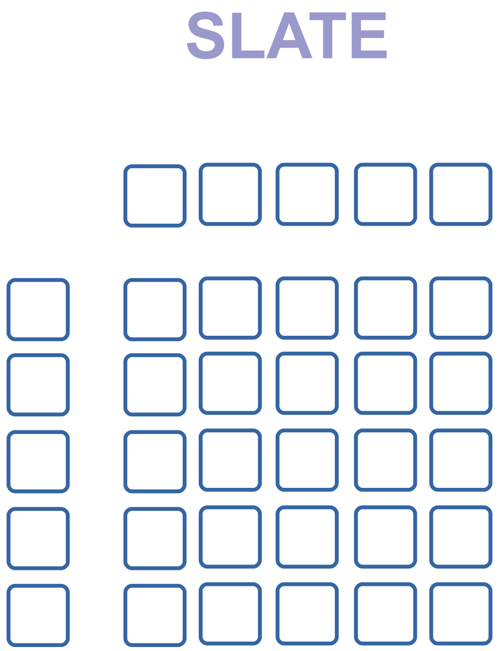
C = C – A × B with small k, i.e., the DGEMM called in LU factorization

The matrix fills out the GPU memory. The X axis shows the k dimension.

SLATE: Building on Top of BLAS++, LAPACK++ and Batched BLAS++

Data Storage Comparison



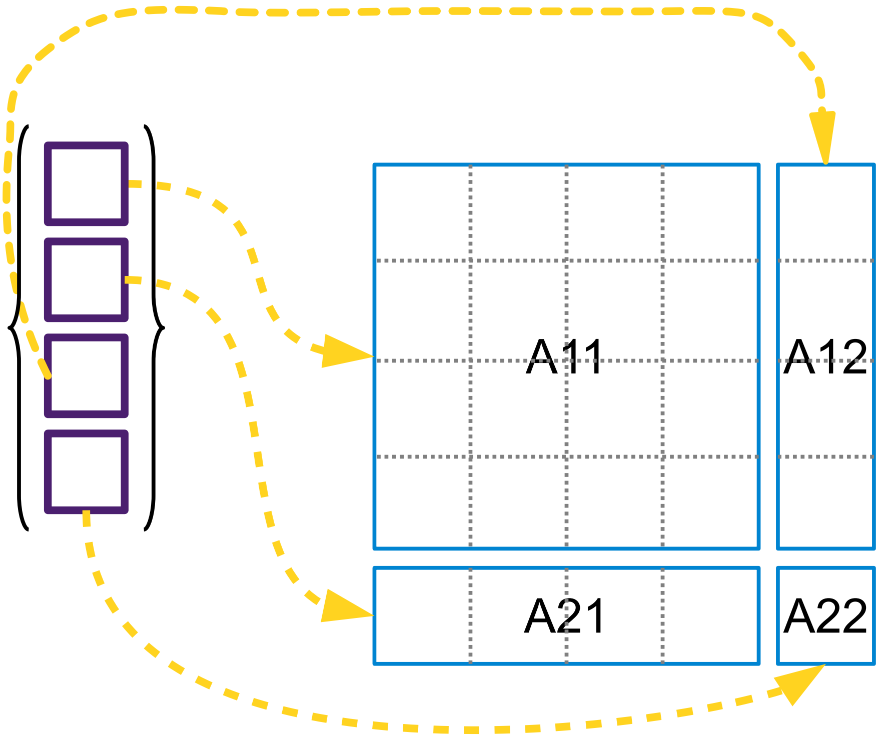


Schur complement: **C** ← **C – A × B**

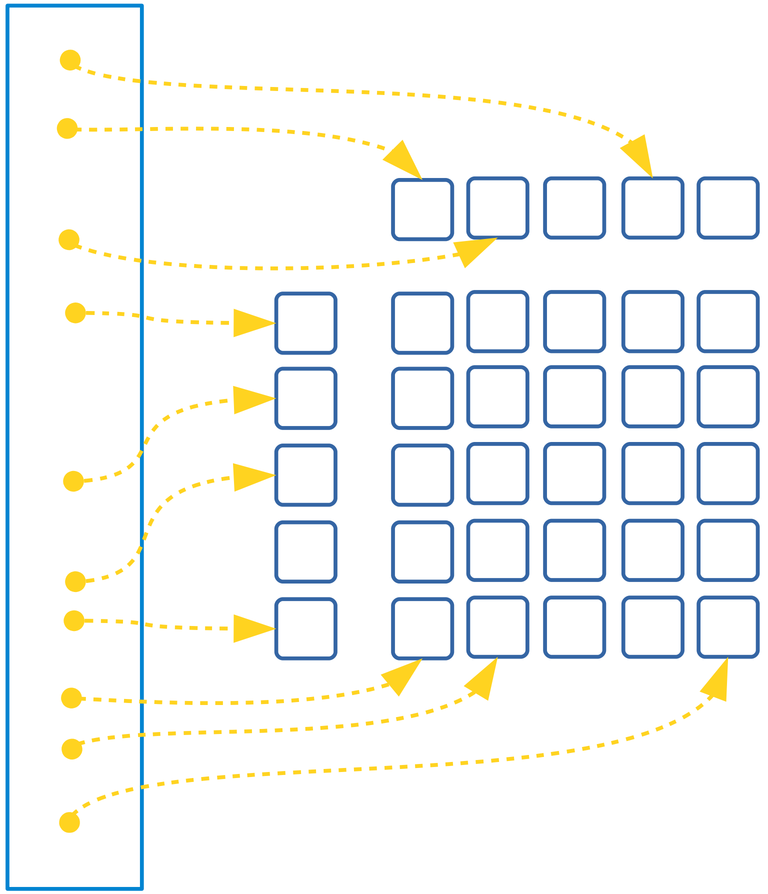
$$A_{22} \leftarrow A_{22} - A_{21} A_{11}^{-1} A_{12}$$

PLASMA (C) and SLATE(C++) Data Storage Comparison

PLASMA Descriptor

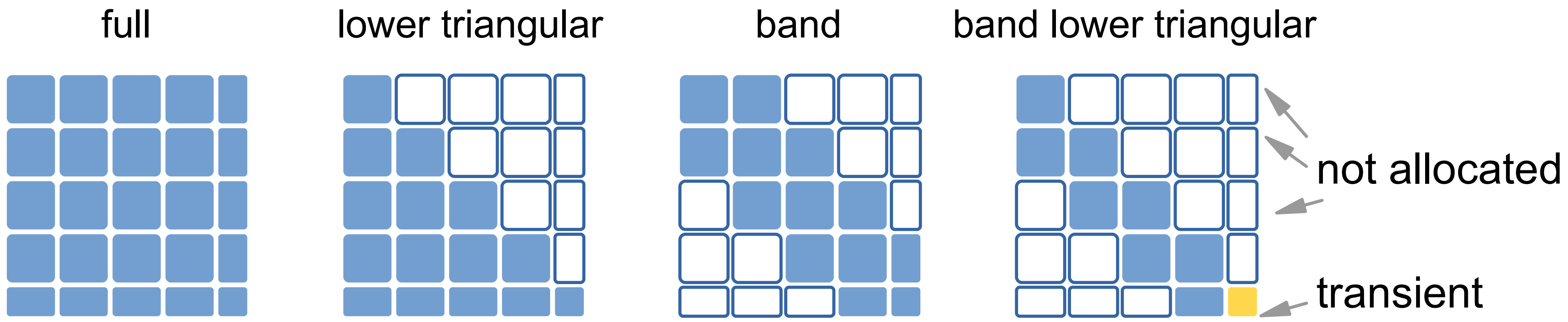


SLATE Tile Map<>



This layout allows in-place translation.

SLATE Matrix Storage



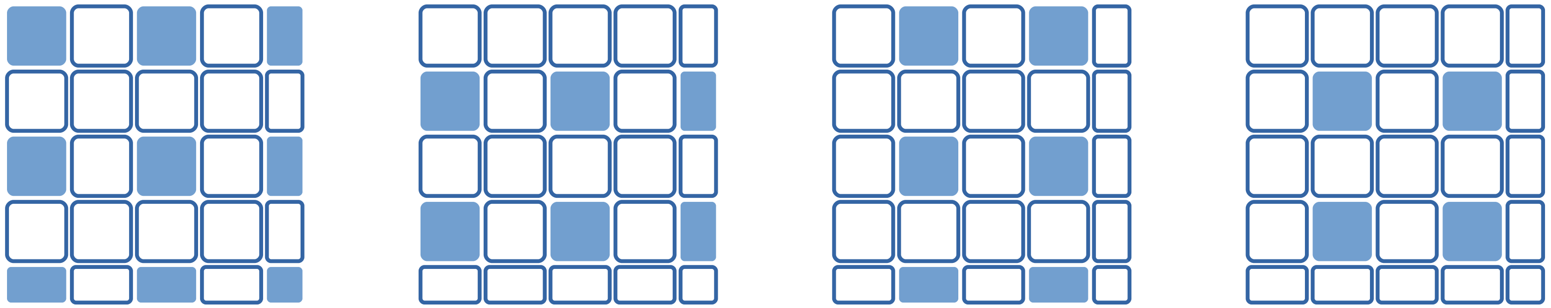
```
std::map<std::tuple<int64_t, int64_t, int>, Tile<FloatType>*> *tiles_;
```

While in the PLASMA library the matrix is also stored in tiles, the tiles are laid out contiguously in memory.

In contrast, in SLATE, the tiles are individually allocated, with no correlation of their locations in the matrix to their addresses in memory.

- collection of tiles
- **individually allocated**
- only allocate what is needed
- accommodates: symmetric, triangular, band, …

SLATE Distributed Matrix



```
std::map<std::tuple<int64_t, int64_t, int>, Tile<FloatType>*> *tiles_;
```

- distributed matrix
- global indexing of tiles
- only allocate the local part
- any distribution is possible (2D block cyclic by default)

The same structure, used for single node representation, naturally supports distributed memory representation.

Looking Forward

Future Future: Linear Algebra in C++ Std Lib: P1385

`wg21.link/p1385`

- Target release: C++23
- Scope
  - Matrices and vectors with rank and dimension
  - Additions and subtraction, also negation
  - Engine for storage definition and access: fixed size (FS), dynamic size (DS), fixed-capacity (FC), dynamically re-sizable (DR)
- Quotes:
  - Several types of decomposition are often performed in solving least-squares problems.
  - Eigen-decompositions are decompositions performed upon a symmetric matrix
- Tensors in C++26?
- “Concurrency and Parallelism” possible in implementations (as in game development)
  - See N4454 from 2015 and N4184 from 2014 with SIMD Types
- No slicing, tiling, submatrices, views? But “Transpose Engine” is mentioned.
- Still digesting the content...

Response to P1417: Historical Lessons for C++ Lin. Alg. Std.

`wg21.link/p1417`

- These my highlights from Mark Hoemmen’s slides to the C++ committee
  - Layering for performance portability and spreading developer expertise
  - BLAS: Fortran-based, templating-by-naming, column-major data layout, long param. Lists
  - Lowest layer: multi-dim array + SIMD (zero-overhead, compiler support needed, storage)
  - mdspan: dimensions, strides, views, access, slices, memory spaces
  - 1D is inefficient (data locality, expressivness
  - Typing dilemma: vector, matrix, tensor
  - Related effort: C++ Graphics library proposal
  - Related standard documents: P0009 (multi-dim. arrays), N4744 (SIMD), P1385 (proposal), P1417 (detailed proposal response)

MS and Posters of Interest at SIAM CSE

MS	Title	Date/time
22	C++ in CSE p. 1	9:45pm, 203
57	C++ in CSE p. 2	2:15pm, 203
117	Batched BLAS p. 1	Tue, 9:45
151	Batched BLAS p. 2	Tue, 2:15
197	Task-based Lin. Alg.1	Wed, 9:45
231	Task-based Lin. Alg.2	Wed, 2:15
398	C++ Library for SIMD	Fri, 12:45

Sess.	Title	Date/time
PP103	Software Productivity	Tue, 4:50
PP103	Sustainable Linear Algebra	Tue, 4:50
PP103	CSE Software Ecosystems	Tue, 4:50