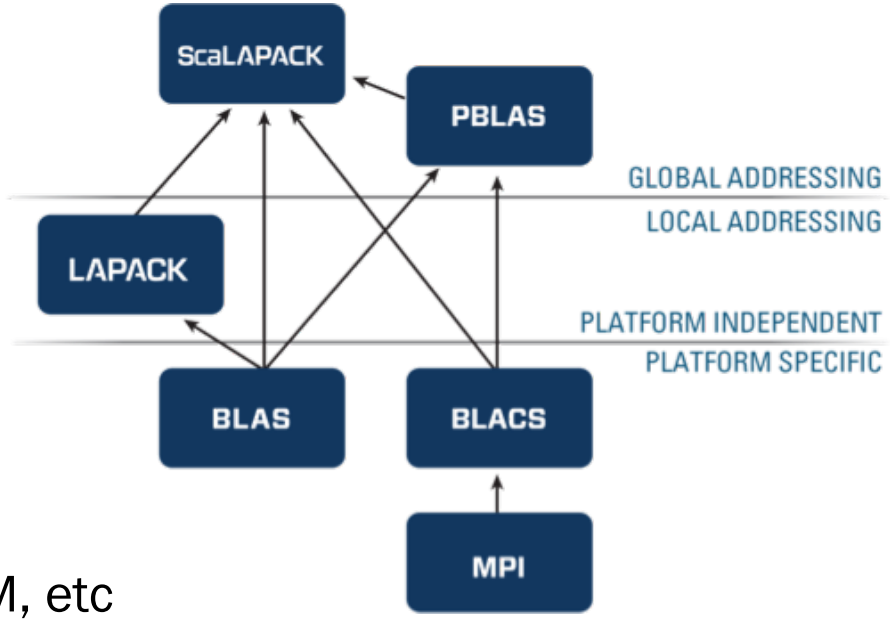
SLATE: Multilevel Tasking in Dense Linear Algebra Libraries

Asim YarKhan, Mark Gates,
Piotr Luszczek, Jakub Kurzak,
Jack Dongarra

THE UNIVERSITY OF
TENNESSEE
KNOXVILLE

ScaLAPACK

- Robust
  - 20 years of testing
- Asymptotically scalable
  - Excellent CPU performance
- Widespread adoption
  - Including vendors like Intel, Cray, IBM, etc
- BUT: GPU and accelerator support missing
  - No easy path to get there



Exascale Computing Project

- DOE ECP Project
  - Getting ready for exascale
  - Exascale around 2021
- Summit at ORNL
  - 4,608 nodes
  - 27,000 NVIDIA Volta GPUs with more than 9,000 IBM Power9 CPUs
  - Each node
    - 2 Power 9 CPUs
      - 2.3 % of the DP-FLOPS
    - 6 Nvidia V100 GPUs
      - 97.7 % of the DP-FLOPS
- GPUs provide almost all of the FLOPS

SLATE Objectives

- Functionality
  - ScaLAPACK replacement
  - Communication avoiding, randomized, …
- Target
  - DOE Exascale, multi-{GPU,core,memory}
- Portability
  - Use standard components, technologies, runtimes
- Scalability
  - Tens of thousands of nodes, limiting bandwidth
- Performance
  - Asymptotic performance toward (80-90%) machine peak
- Sustainability
  - Maintain code, extend algorithms, new architectures

Software for Linear Algebra Targeting Exascale

Prior Projects

LAPACK

ScaLAPACK

PLASMA

QUARK

MAGMA

HPL

DPLASMA

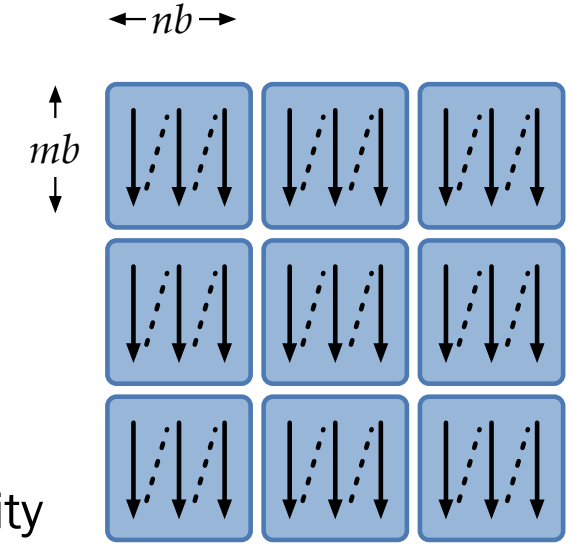
PaRSEC

BATCHED BLAS

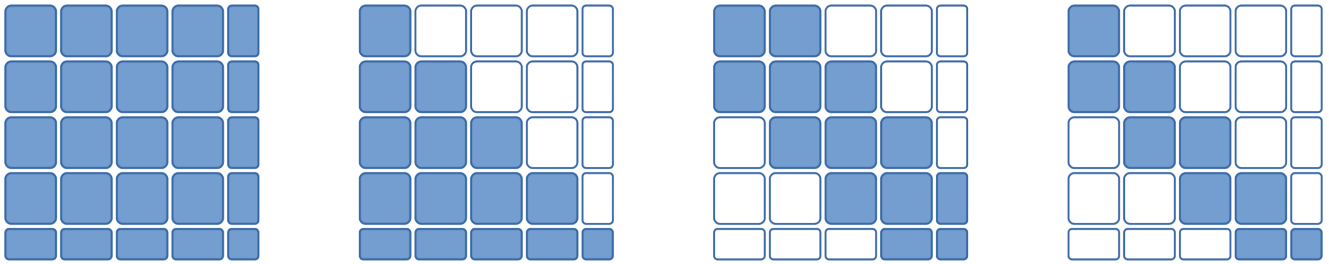
SLATE Design Principles

- Data Tiling
- Multi-level Task Scheduling
- Batched Execution

Data Tiling

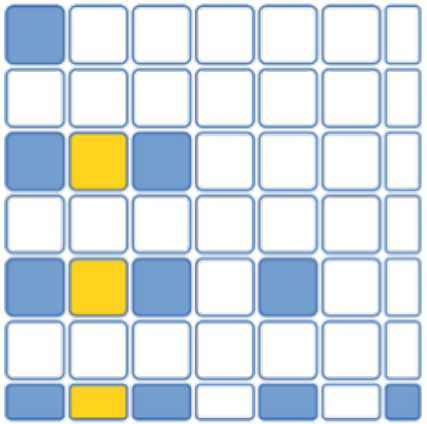


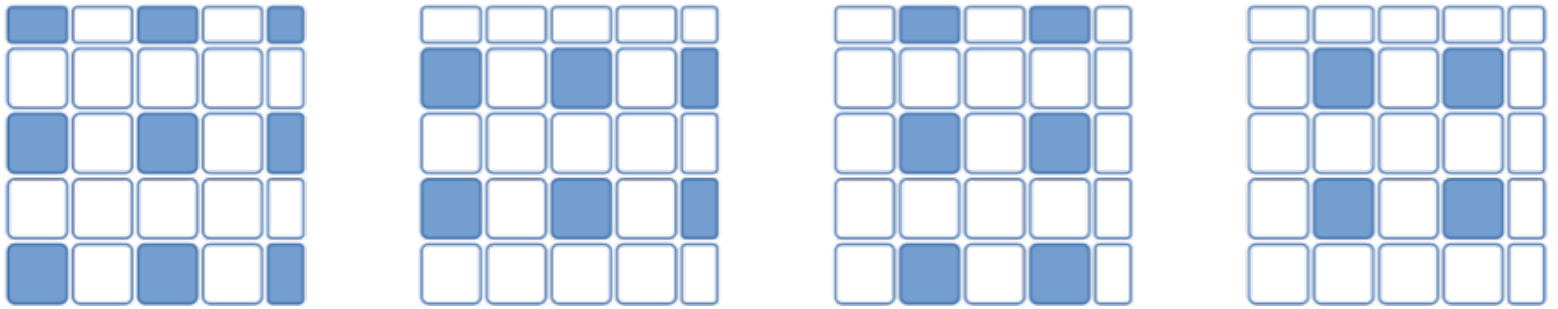
- Matrix is a map of tiles
  - Can be individually allocated
  - Can have a stride
  - Enables ScaLAPACK 2D block cyclic compatibility
- Map from tile indices { i, j, device } to Tile
  - Global addressing of tiles
  - Distributed & GPU support is straightforward
  - No wasted memory for symmetric and triangular matrices
  - Accommodates band & block-sparse matrices



Data Tiling in Distributed Memory

- Map from tile indices { i, j, device } to Tile
  - Map exists on all nodes
  - Local storage for tiles at each location
  - 2D block cyclic by default
  - Any mapping possible
  - Remote data is attached to map placeholder

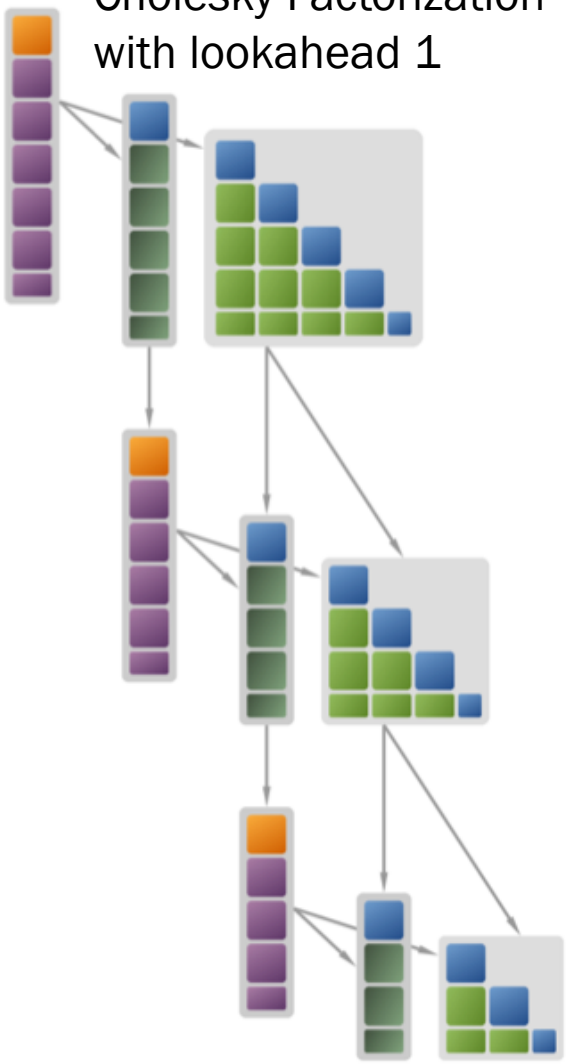




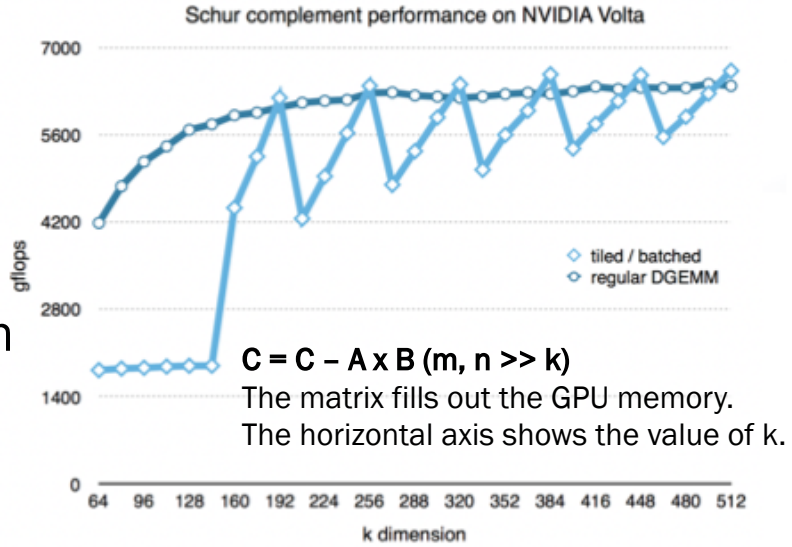
Multi-Level Tasking

- Data-dependent
  - Top level algorithmic scheduling
  - High level large blocks (dependency arrows)
    omp_task_depend inout(column[k])
  - Dependency on tile-column (not actual data)
  - Enables lookahead
- Nested parallel
  - Lower level parallel tasks
  - Without dependencies, nested tasks
    omp_task
  - Performance via batching

Cholesky Factorization
with lookahead 1



Batched Execution



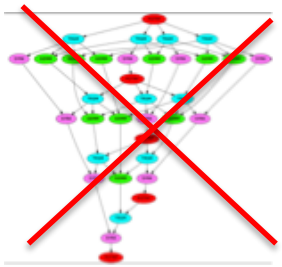
C = C – A x B (m, n >> k)
The matrix fills out the GPU memory.
The horizontal axis shows the value of k.

- Tiled data + tasks
  - Good for algorithm design and parallelism
  - But GPU performance needs large tiles?
  - Often needs tiles of 1024+
- Solution = batching
  - Group or batch a number of operations and dispatch simultaneously
  - Get performance for smaller tile sizes (192, ...)
- Batching
  - GPUs: CUBLAS Batch GEMM
  - CPUs: Batch GEMM or OpenMP tasking

`targets`

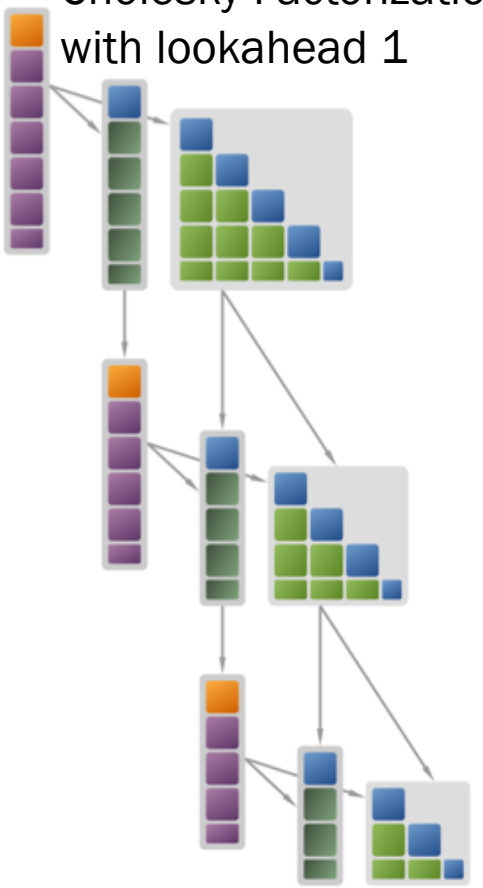
- `Target::Host`
- `Target::HostTask`
- `Target::HostNest`
- `Target::HostBatch`
- `Target::Devices`

Data Dependent Tasking



- Panel/block-column dependencies
  - Compared to full tile-based dataflow
  - O(n) dependencies rather than $O(n^3)$ dependencies
  - Block-column allows pivoting
- Task priorities
  - Prioritize panel factorization
  - Priorities 1 (or more) lookahead block-columns of the trailing submatrix
  - **omp task depend(inout:column[k]) priority(P)**
  - Lookahead improves performance but needs to be limited to limit buffer usage (distributed memory) Diminishing returns from lookahead=2,3…
  - Update usually runs at a lower priority

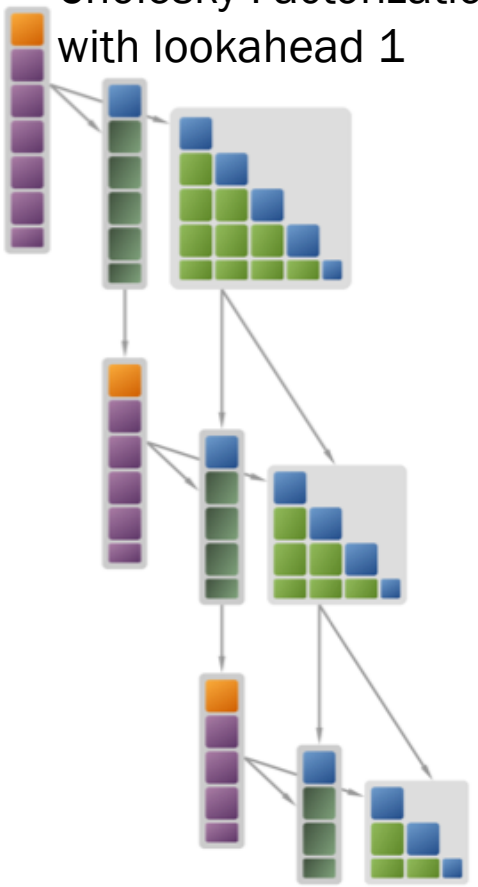
Cholesky Factorization with lookahead 1



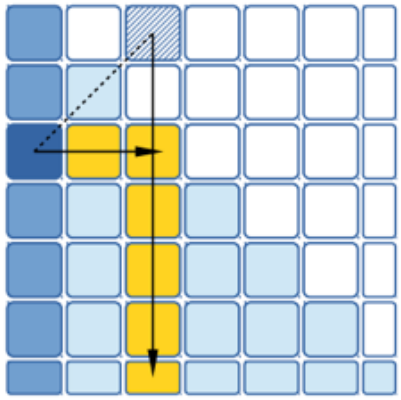
Nested Tasks and Device Support

- Multiple targets
  - HostTask, Device, HostNest, HostBatch
  - At the high level data-dependency
- Target=Device
  - At the lower nested-task level
  - **omp task** for each device (GPU)
  - Tiles-with-work are batched (depends on batch API)
  - CUDA: Use batch cublas (**cublasGemmBatched**, …)
- Target=HostTask
  - At the lower nested-task level
  - **omp task** for each tile (gemm, ..)

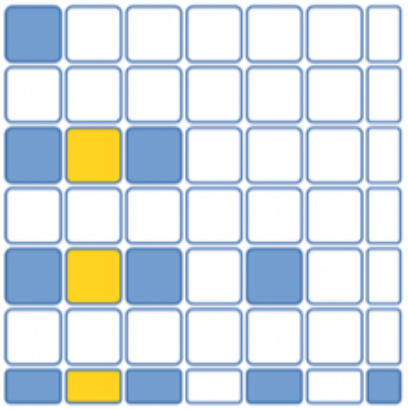
Cholesky Factorization
with lookahead 1



Distributed Tile Support



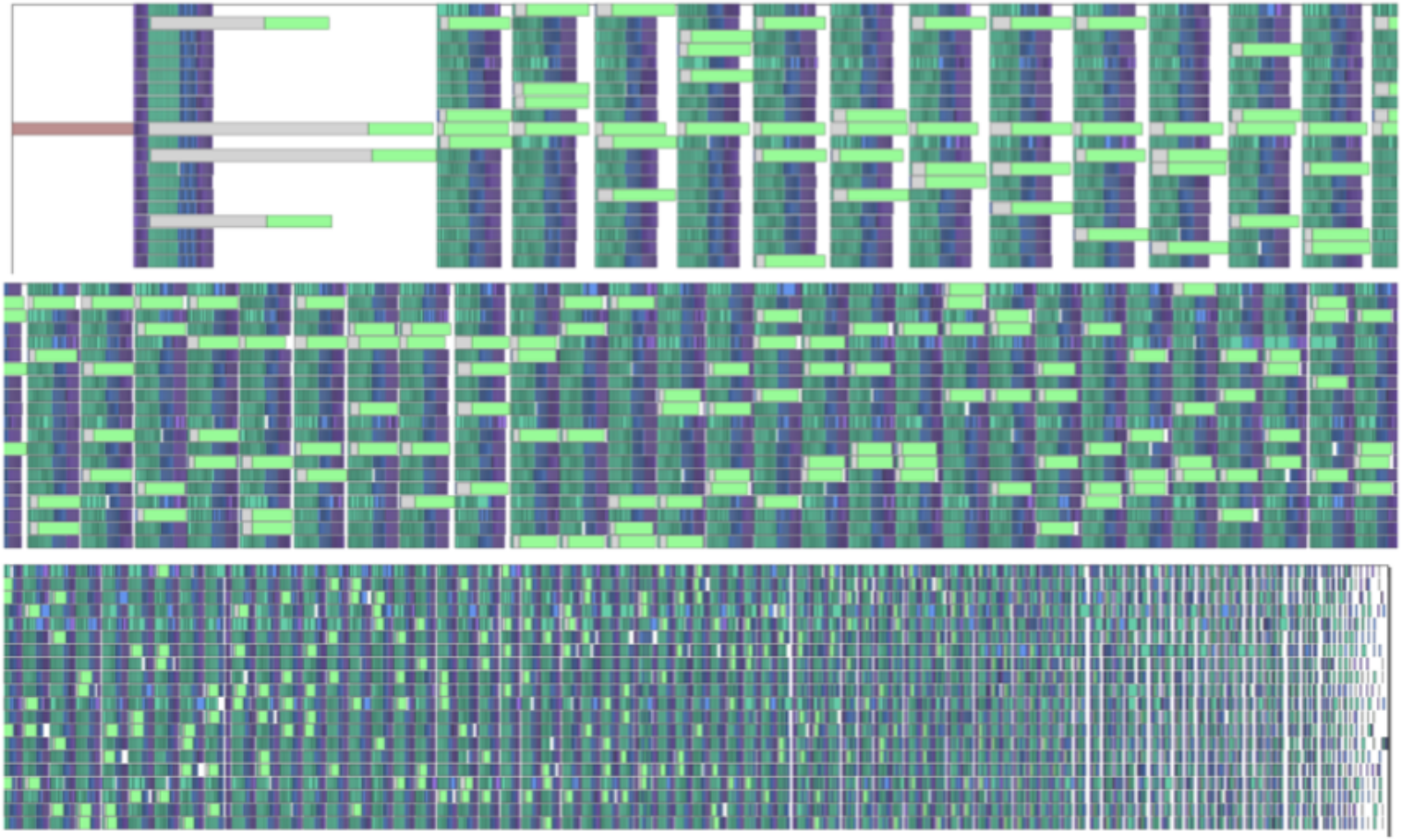
- Tiles are pushed to submatrices
  - Explicit dataflow tileSend(tile, range1, range2)
  - From ranges and distribution, build the ranks
  - On receive, attach the tile-buffers to the map (tile indices { i, j, device } to Tile)
  - Mark transient tile-buffers with a lifespan



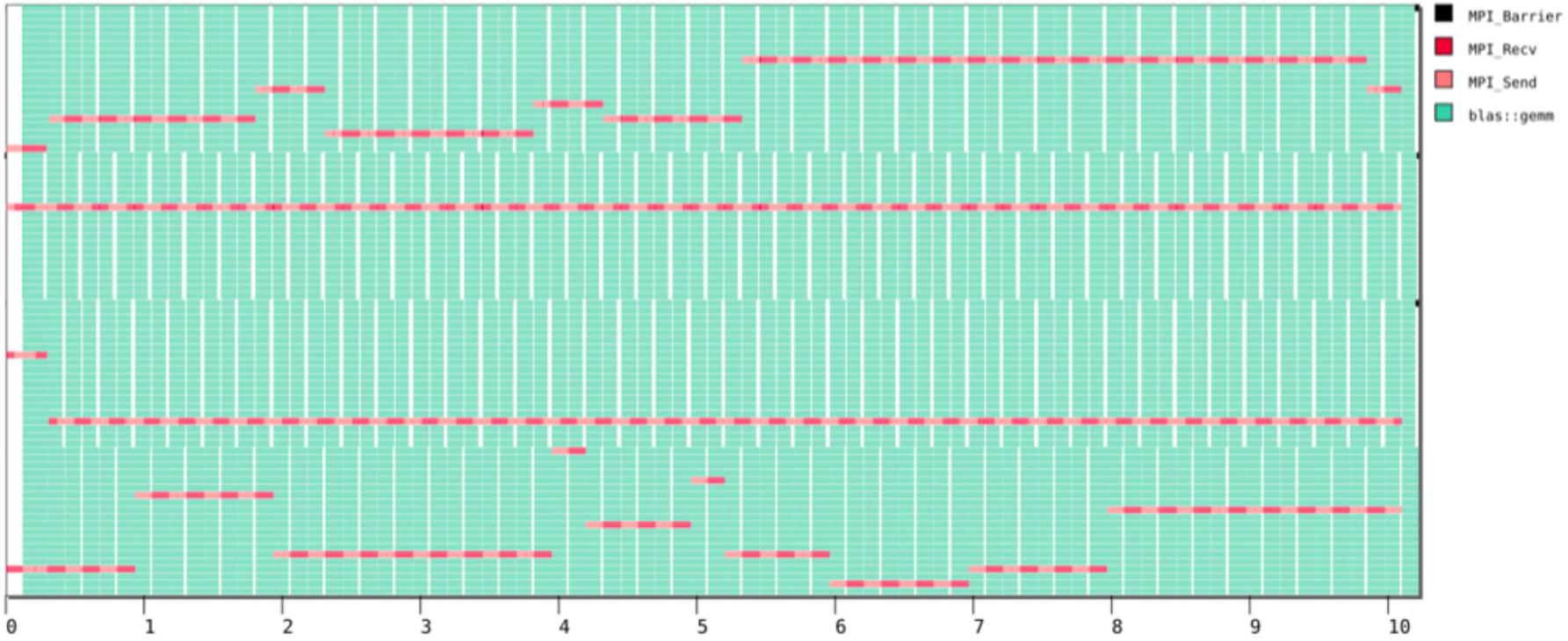
- Implementation
  - Communication is multicast send and receive
  - Requires MPI_THREAD_MULTIPLE
  - Communication contained in **OMP Tasks**
  - Tried: non-blocking collectives; comm_group_create
  - Current: Home grown N-dim (2-dim) hypercube of point-to-point non-blocking sends and receives (followed by wait)

Cholesky Trace: 4 GPUs

- 20 cores and four GPUs factoring a 112, 000 × 112, 000 matrix with the tile size of 512 × 512. The lookahead is one. The matrix is distributed to the GPUs in a 1D block cyclic fashion.

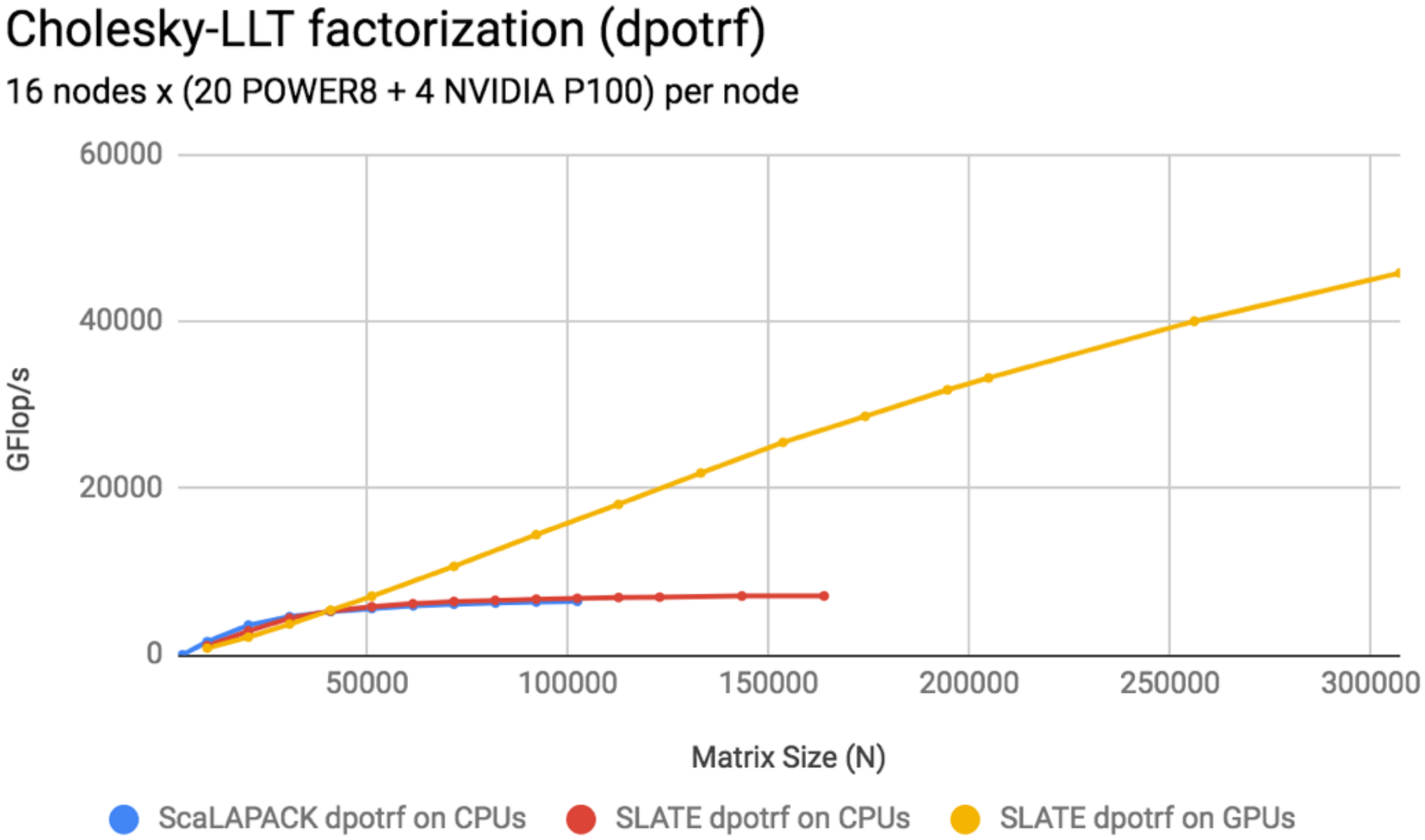


DGEMM Trace: 4 nodes



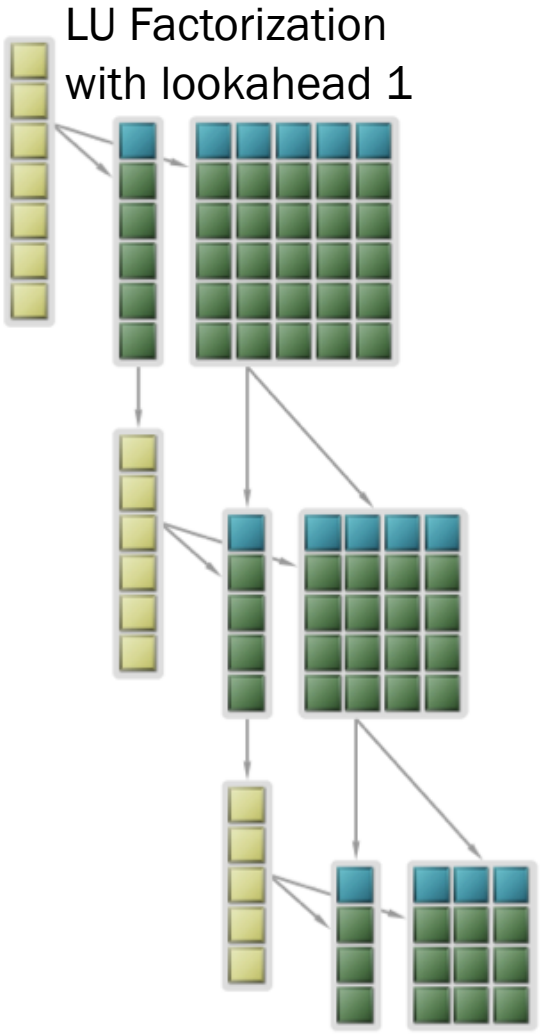
Performance: Cholesky

Cherry Picked Results



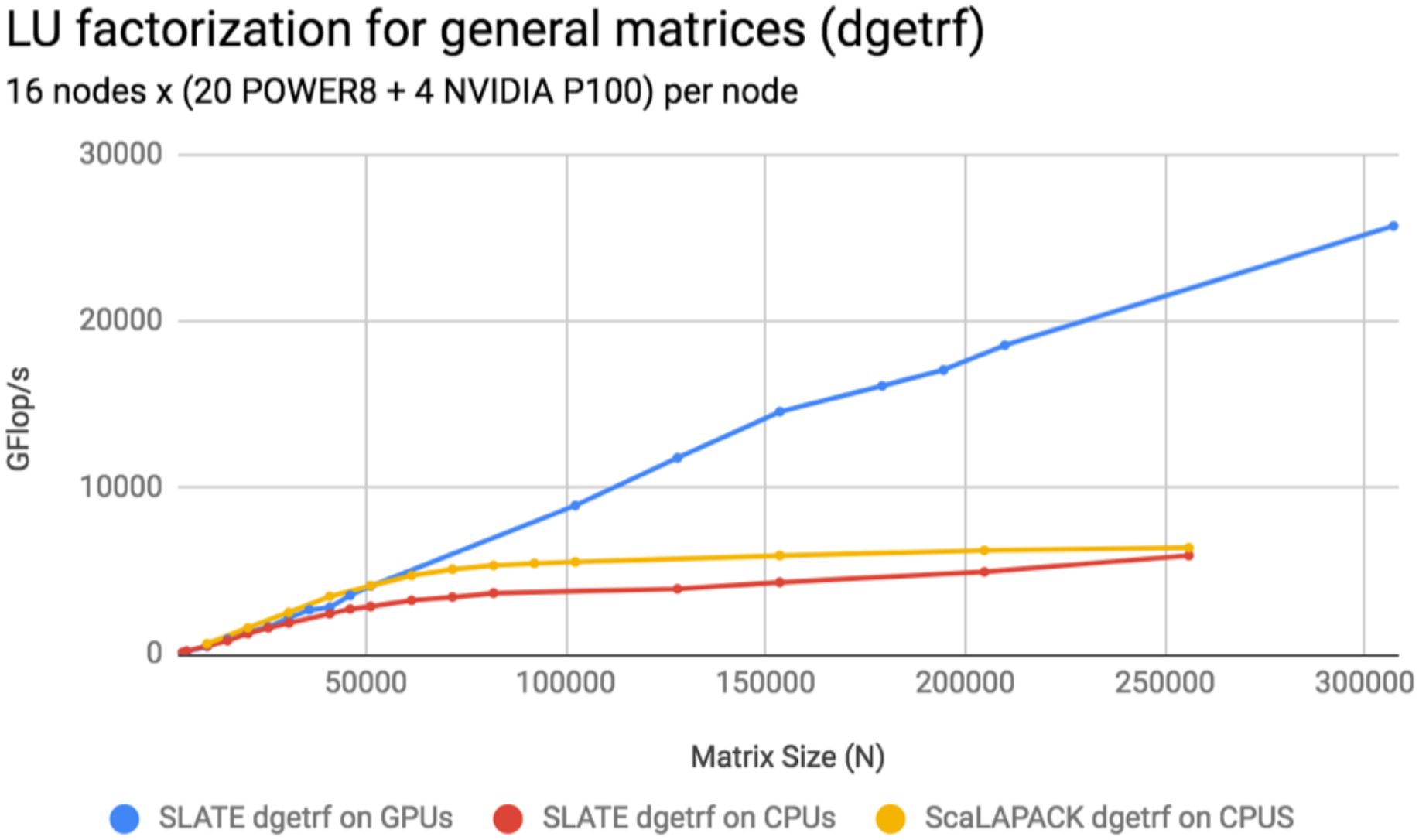
Scheduling LU

- LU Factorization
  - Gaussian elimination; partial row pivoting, forward and backward substitution
  - Fully distributed (2D block cyclic included)
- Challenges
  - Application of row pivoting
    Complex, synchronous panel factorization
    Communication overhead of left/right swaps
  - GPU layout translation because row swapping operation is extremely inefficient in column major



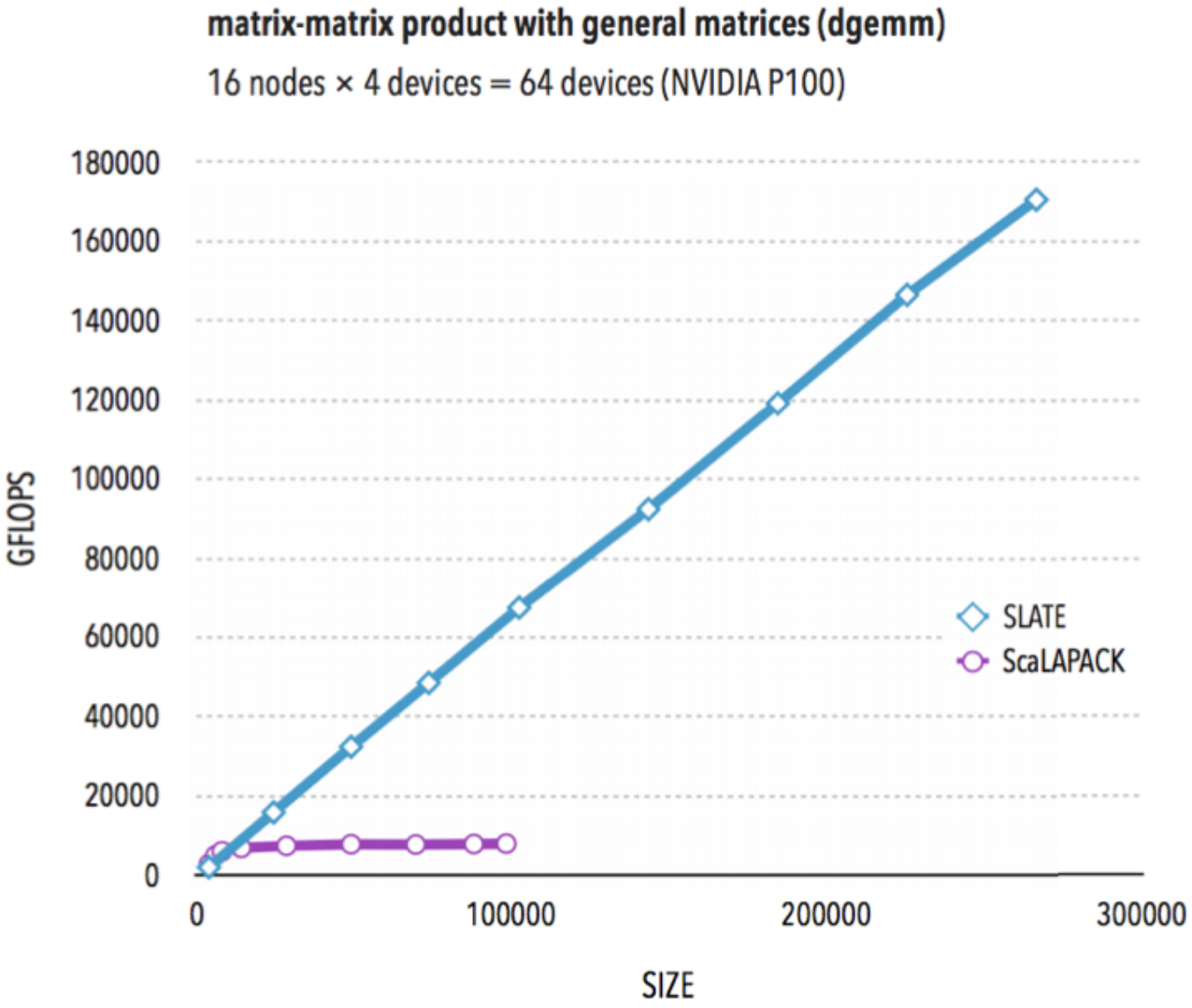
Kurzak, J, et. al. “Linear Systems Performance Report,” SLATE Working Notes, no. 8, ICL-UT-18-08: Innovative Computing Laboratory, University of Tennessee, 2018

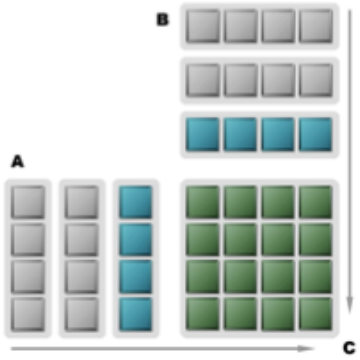
Performance: LU



PBLAS-Level 3: GEMM

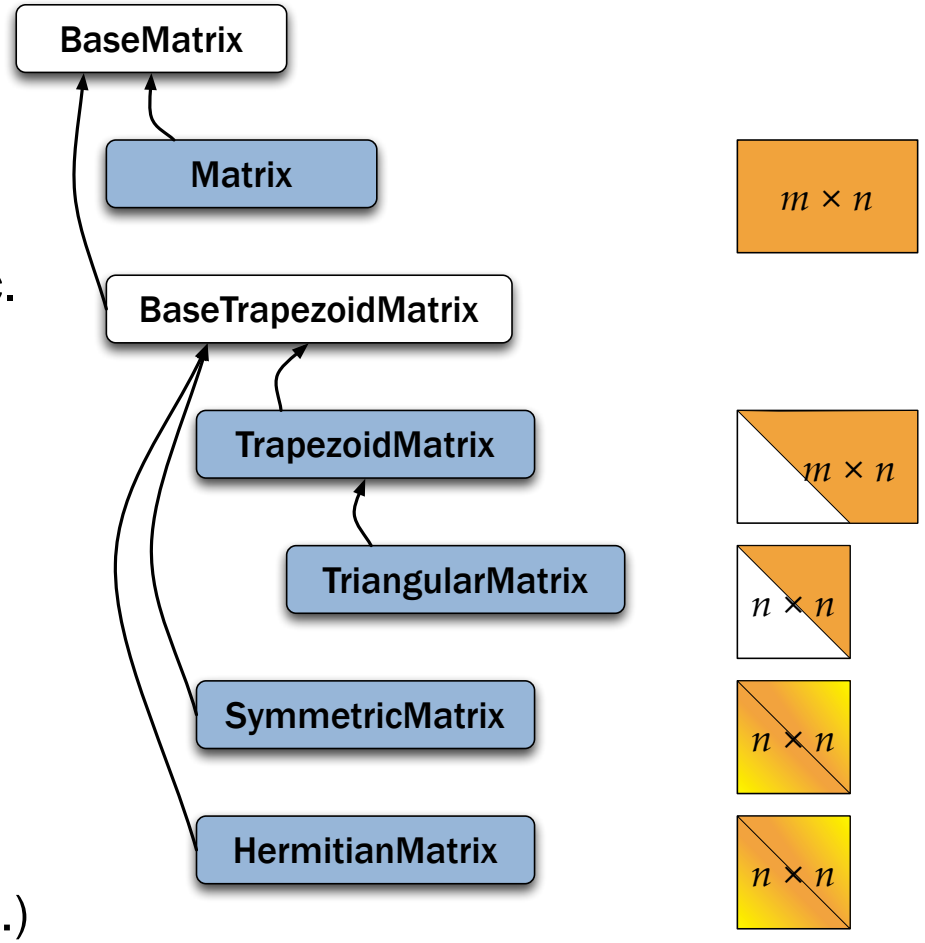
Cherry Picked Results





SLATE Tools

- Modern Language
  - C++11 and up
  - templating, containers
  - smart pointers, move semantic, etc.
- Object Oriented Design
  - Matrix class, Tile class, etc.
  - inheritance, polymorphism, etc.
- Standards
  - MPI 3 and up
  - OpenMP 4 and up
  - possibly other runtimes (PaRSEC, ...)



Coding with SLATE: LU solve

https://bitbucket.org/icl/slate-tutorial

SLATE tutorial by Mark Gates

```
int main( int argc, char** argv )
{
    MPI_Init( &argc, &argv );
    MPI_Comm_size( MPI_COMM_WORLD, &mpi_size );
    int64_t n=1000, nrhs=200, nb=100, p=2, q=1;
    assert( mpi_size == p*q );
    slate::Matrix<double> A( n, n,    nb, p, q, MPI_COMM_WORLD );
    slate::Matrix<double> B( n, nrhs, nb, p, q, MPI_COMM_WORLD );
    A.insertLocalTiles(); // SLATE function
    B.insertLocalTiles(); // SLATE function
    random_matrix( A );   // local function
    random_matrix( B );   // local function
    slate::Pivots pivots;

    printf( "gesv\n" );
    slate::gesv( A, pivots, B );
    printf( "done\n" );
    MPI_Finalize();
}
```

Compatibility APIs (in Progress)

- ScaLAPACK libslate_scalapack_api.so
  - Can intercept ScaLAPACK library calls (3 Fortran name-mangling)
  - Uses ScaLAPACK 2D block-cyclic distribution directly
  - Can SLATE-enable / GPU-enable existing code
- LAPACK libslate_lapack_api.so
  - Parameter level compatibility
  - In source code replace dgemm () with slate_dgemm()
  - Uses LAPACK layout directly
  - SLATE / GPU enable code
- Caveats
  - Large problem sizes. Appropriate {Sca}LAPACK nb. Fall back to vendor.

Current Status

- Least squares
  - QR factorization (CAQR panel)
  - GELS
- Linear systems
  - LLT (Cholesky) factor, solve (potrf, potrs, posv)
  - LU factor, solve (getrf, getrs, gesv)
  - LDLT (Aasen's) (problems: symmetric pivoting prevents GPU acceleration)
- Parallel Norms
  - Norms (ge, tr, sy, he, gb); complexities due to tiles;
- Parallel BLAS
  - Parallel gemm, symm, syrk, syr2k, trmm, trsm, ...

C++ templated datatypes: float, double, complex<float>, complex<double>

SLATE WORKING NOTES

Gates, M., A. Charara, J. Kurzak, A. YarKhan, I. Yamazaki, and J. Dongarra, "**Least Squares Performance Report**," *SLATE Working Notes*, no. 9, ICL-UT-18-10: Innovative Computing Laboratory, University of Tennessee, December 2018.
GOOGLE SCHOLAR · BIBTEX · TAGGED · XML
DOWNLOAD (1.76 MB)

Kurzak, J., M. Gates, I. Yamazaki, A. Charara, A. YarKhan, J. Finney, G. Ragghianti, P. Luszczek, and J. Dongarra, "**Linear Systems Performance Report**," *SLATE Working Notes*, no. 8, ICL-UT-18-08: Innovative Computing Laboratory, University of Tennessee, September 2018.
GOOGLE SCHOLAR · BIBTEX · TAGGED · XML
DOWNLOAD (1.64 MB)

Abdelfattah, A., M. Gates, J. Kurzak, P. Luszczek, and J. Dongarra, "**Implementation of the C++ API for Batch BLAS**," *SLATE Working Notes*, no. 7, ICL-UT-18-04: Innovative Computing Laboratory, University of Tennessee, June 2018.
GOOGLE SCHOLAR · BIBTEX · TAGGED · XML
DOWNLOAD (1.07 MB)

Kurzak, J., M. Gates, A. YarKhan, I. Yamazaki, P. Luszczek, J. Finney, and J. Dongarra, "**Parallel Norms Performance Report**," *SLATE Working Notes*, no. 6, ICL-UT-18-06: Innovative Computing Laboratory, University of Tennessee, June 2018.
GOOGLE SCHOLAR · BIBTEX · TAGGED · XML
DOWNLOAD (1.13 MB)

Kurzak, J., M. Gates, A. YarKhan, I. Yamazaki, P. Wu, P. Luszczek, J. Finney, and J. Dongarra, "**Parallel BLAS Performance Report**," *SLATE Working Notes*, no. 5, ICL-UT-18-01: University of Tennessee, April 2018.
GOOGLE SCHOLAR · BIBTEX · TAGGED · XML
DOWNLOAD (4.39 MB)

Abdelfattah, A., K. Arturov, C. Cecka, J. Dongarra, C. Freitag, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, et al., "**C++ API for Batch BLAS**," *SLATE Working Notes*, no. 4, ICL-UT-17-12: University of Tennessee, December 2017.
GOOGLE SCHOLAR · BIBTEX · TAGGED · XML
DOWNLOAD (1.89 MB)

Kurzak, J., P. Wu, M. Gates, I. Yamazaki, P. Luszczek, G. Ragghianti, and J. Dongarra, "**Designing SLATE: Software for Linear Algebra Targeting Exascale**," *SLATE Working Notes*, no. 3, ICL-UT-17-06: Innovative Computing Laboratory, University of Tennessee, October 2017.
GOOGLE SCHOLAR · BIBTEX · TAGGED · XML

Work In Progress / TODO

Directions

- Communication-avoiding
- Randomization
- Low-rank compressed tiles
- Arbitrary distributions
- Mixed precision

Issues

- Consistency protocol (MOSI)
- Matrix iterators
- Batch blaspp integration
- Polymorphic tiles
- Error handling

Performance Engineering

- Parallel BLAS
  - slow syrk and syr2k
- Parallel norms
  - unoptimized GPU kernels
  - neglected CPU infinity norm
- Linear systems
  - slow pivoting in LU
  - LDLT performance calamity
- Least squares

SLATE

- **SLATE website:**
  https://icl.utk.edu/slate/
- SLATE repository:
  https://bitbucket.org/icl/slate
- Google Groups Mailing list:
  Search for “SLATE User”

- **C++ API for BLAS**:
  https://bitbucket.org/icl/blaspp
- **C++ API for LAPACK**:
  https://bitbucket.org/icl/lapackpp