SLATE: Design of a Modern Distributed and Accelerated Dense Linear Algebra Library

**Mark Gates, Jakub Kurzak, Ali Charara,
Asim YarKhan, Jack Dongarra**

SC19 — Nov 19, 2019

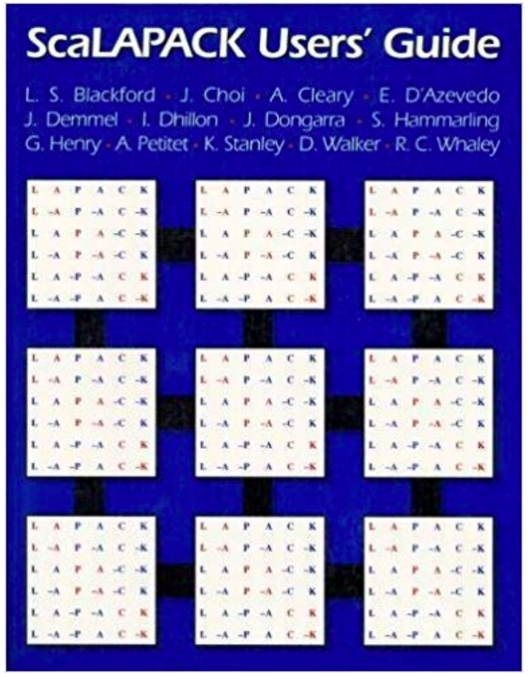
ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Yesterday's HPC

ScaLAPACK

- First released Feb 1995, 25 years old
- Lacks dynamic scheduling, look-ahead panels, communication avoiding algorithms, ...
- Can't be adequately retrofitted for accelerators
- Written in archaic language (Fortran 77)



SGI Origin 2000 (ASCI Blue Mountain, 1998)

- 6,144 MIPS R10000
- 3 Tflop/s

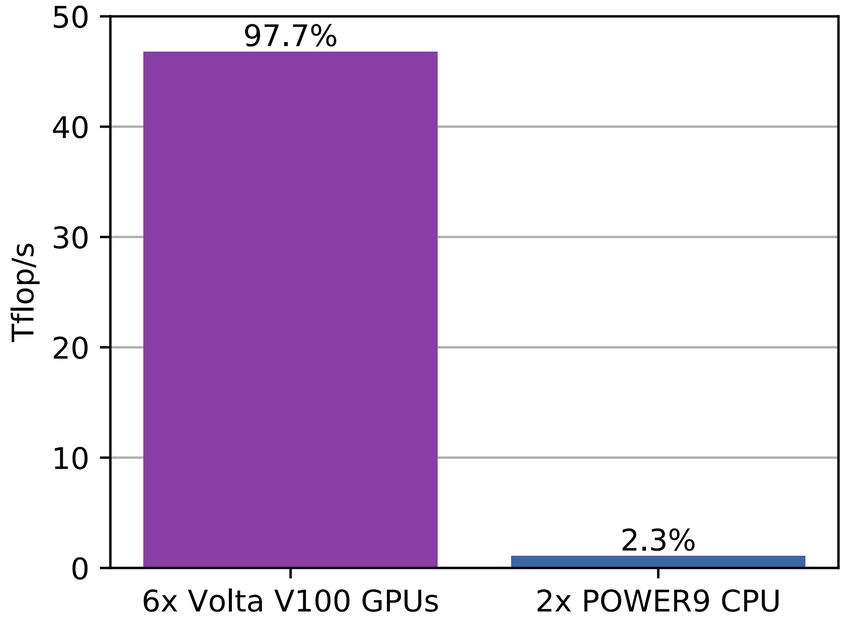
ASCI Blue Mountain

Today's HPC

Summit

- 2x IBM POWER9 CPUs + 6x NVIDIA V100 GPUs per node
- 4608 nodes, 405,504 CPU cores
- 27,648 GPUs, 2,322,432 GPU SMs
- 200 Pflop/s (148 Pflop/s HPL)

CPUs keep GPUs busy



Upcoming systems

- Perlmutter: AMD CPUs + NVIDIA GPUs
- Aurora: Intel CPUs + Intel GPUs
- Frontier: AMD CPUs + AMD GPUs
- Fugaku: ARM CPUs

Overview of SLATE

Goals

Coverage

C++ Usage

SLATE's Goals

Target modern HPC hardware

- Multicore processors, multiple accelerators per node

Achieve portable high performance

- Rely on MPI, OpenMP, vendor-optimized BLAS, LAPACK

Scalability

- 2D block cyclic distribution, arbitrary distribution, dynamic scheduling, communication overlapping

Assure maintainability

- C++ templating and other features to minimize codebase

Ease transition from ScaLAPACK

- Natively support ScaLAPACK 2D block-cyclic layout, backwards compatible API

SLATE's Coverage

Goal is to match or exceed ScaLAPACK's coverage

- Parallel BLAS: gemm, hemm, herk, her2k, trmm, trsm
- Matrix norms: one, inf, max, Frobenius
- Linear systems: LU, Cholesky, symmetric indefinite (block Aasen) †
- Mixed precision: LU †, Cholesky †
- Matrix inverse: LU, Cholesky *
- Least squares: QR, LQ
- Singular Value Decomposition (SVD) (vectors forthcoming)
- Symmetric / Hermitian eigenvalues (vectors forthcoming)
- Generalized Symmetric / Hermitian eigenvalues (2020)
- Non-symmetric eigenvalues (2020) †

† Not in ScaLAPACK

* Standard caveat: solve Ax = b using gesv, etc., rather than inverting and multiplying $x = A^{-1} b$

Why C++?

Growing acceptance & demand in applications

Benefits from std library, better library and compiler support

Simpler, less error-prone interfaces:

- `pdgemm( transA, transB, m, n, k, alpha,`
  `A, ia, ja, descA,`
  `B, ib, jb, descB,`
  `beta,`
  `C, ic, jc, descC,`
  `transA_len, transB_len )`

  19 arguments + possibly 2 hidden arguments

- `slate::gemm( alpha, A, B, beta, C )`

  5 arguments

C++ Usage

Templates for precisions

- Code is precision independent
- Currently instantiate for 4 precisions (float, double, complex, complex-double)
- Other precisions (half, double-double) just need BLAS

Shared pointers

- Sub-matrices share data with parent
- Reference counted, deletes when all copies go out-of-scope

Exceptions

- Avoid silently ignoring errors

Containers

- std::vector, std::map, std::list, std::tuple, etc.

Matrix Format

Tile

Matrix storage: map of tiles

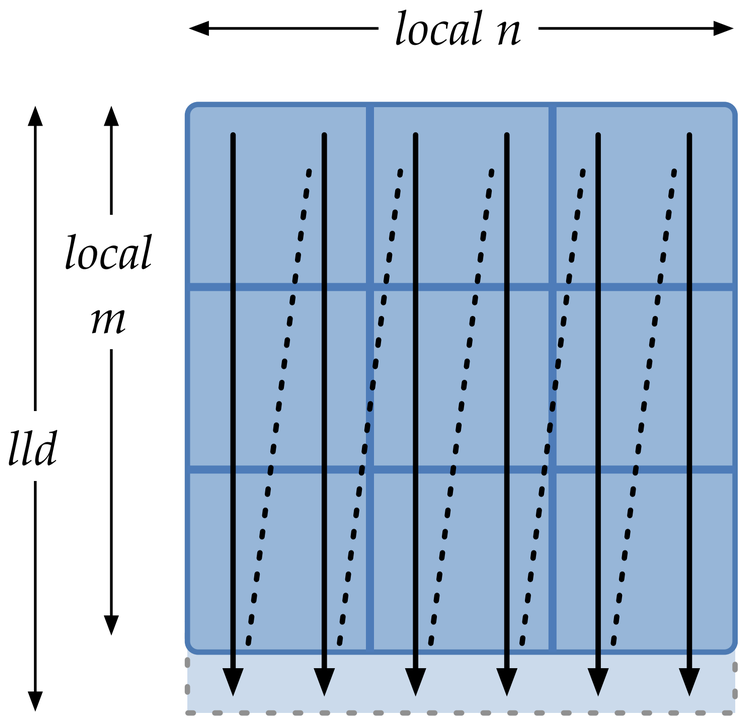
Shallow copy semantics

Matrix hierarchy

Tile Format

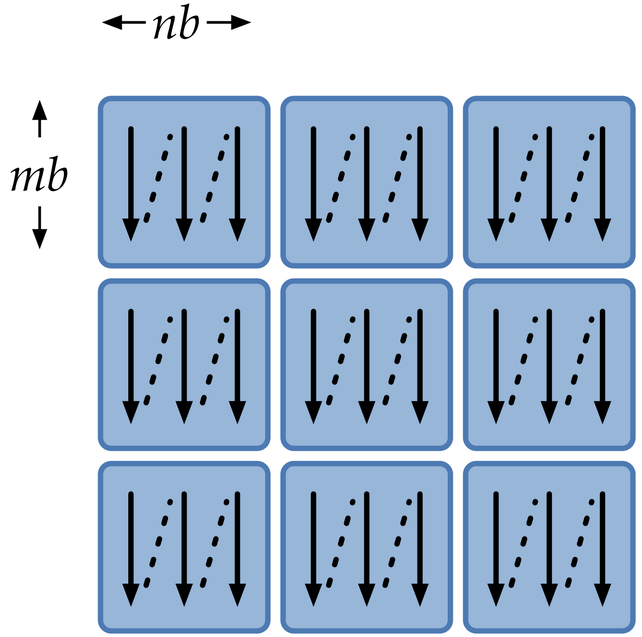
ScaLAPACK format

- 2D block cyclic with stride (*lld*)



OR

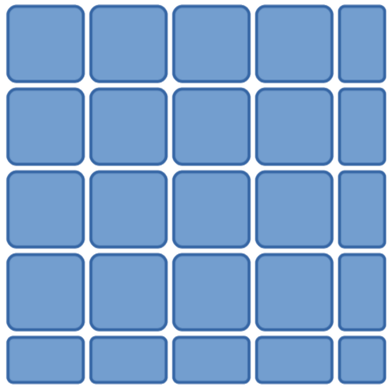
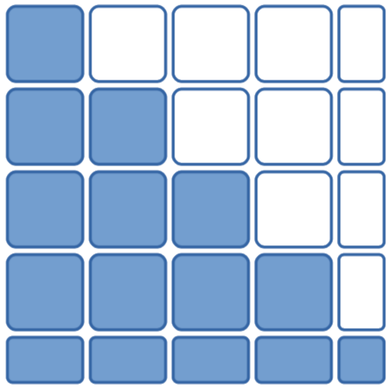
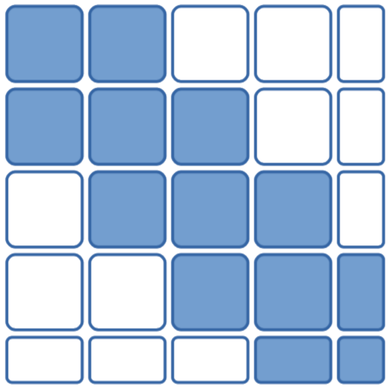
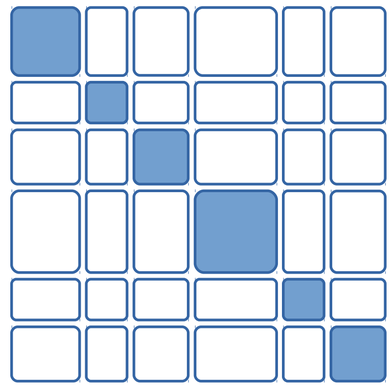
Individually allocated tiles



Matrix Storage

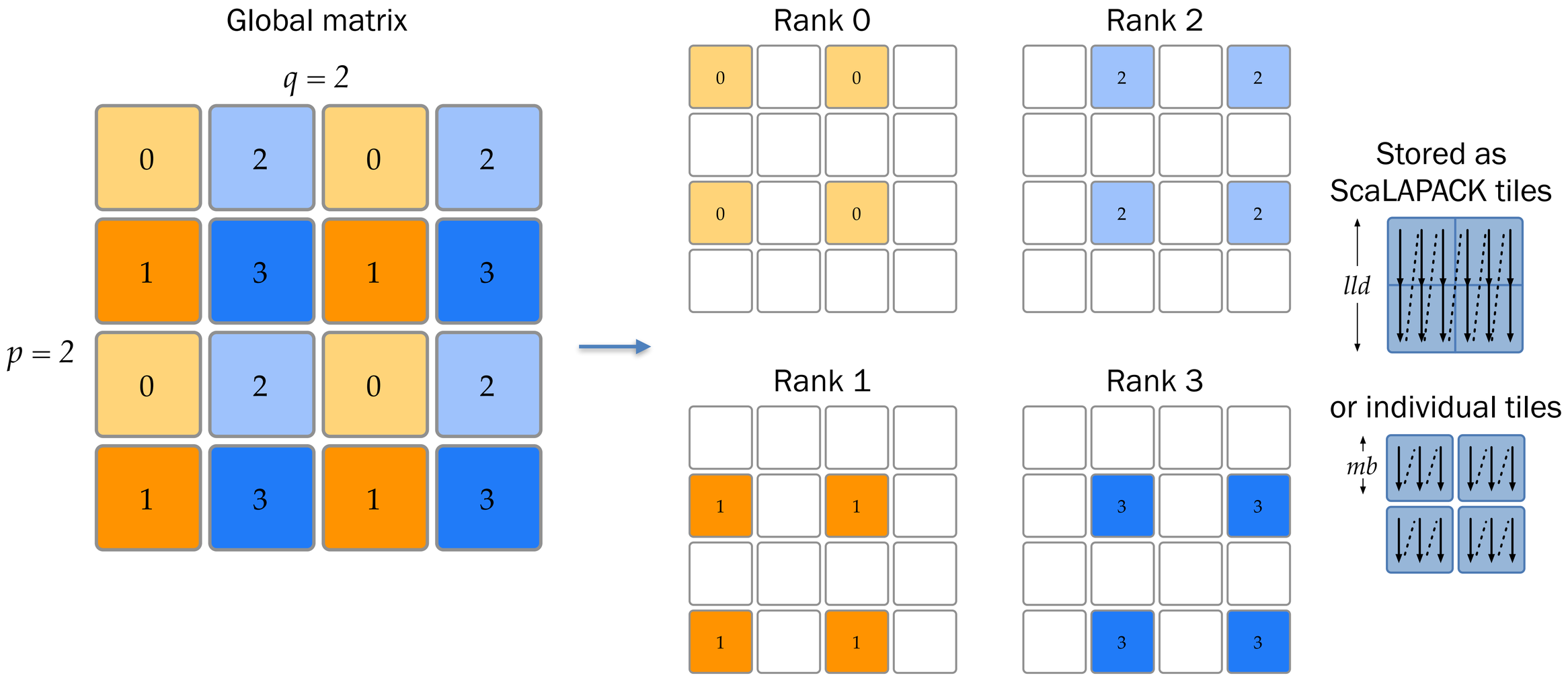
Map from tile indices { i, j } & device id to Tile data

- Tiles individually allocated
- Global addressing
- No wasted space for symmetric, triangular, band matrices
- Rectangular tiles

Distributed Matrix Storage

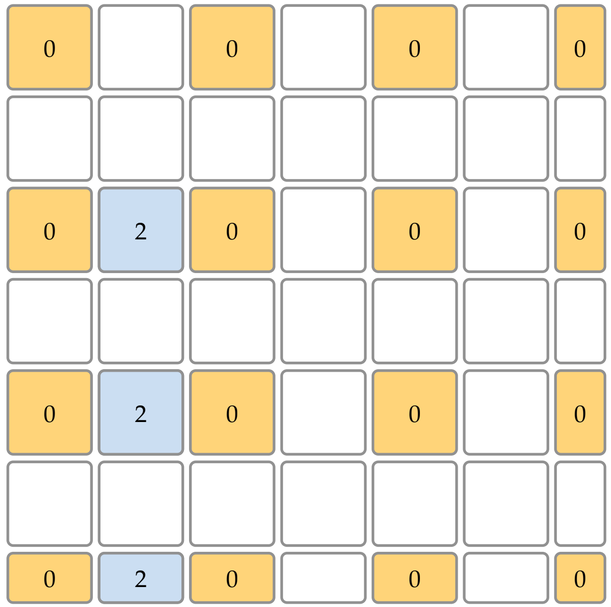
Nodes store local tiles in map (here, 2D block cyclic)



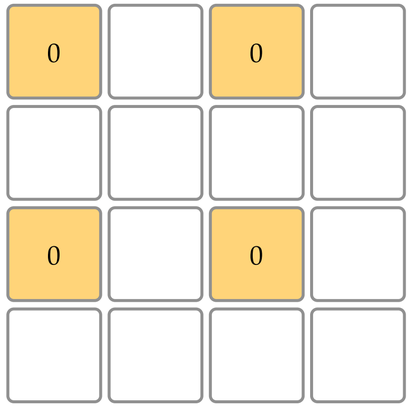
Distributed Matrix Storage

Replicate remote tiles in map

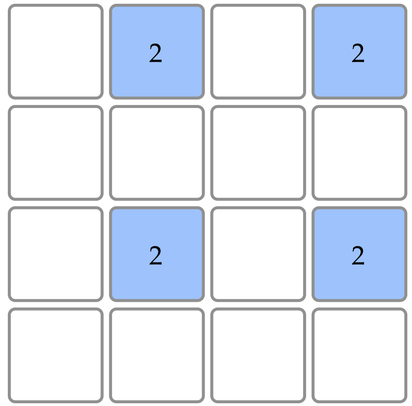
Rank 0, with copies of tiles from Rank 2



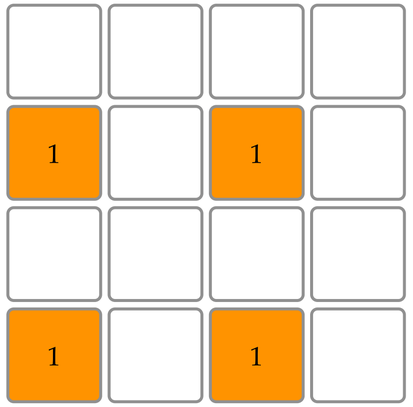
Rank 0



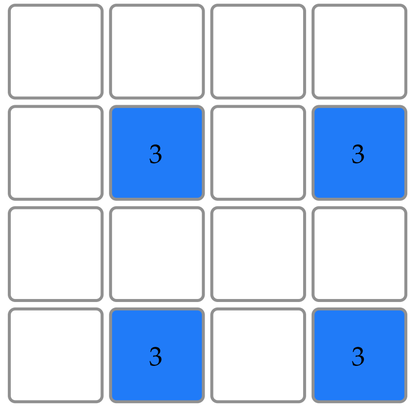
Rank 2



Rank 1



Rank 3

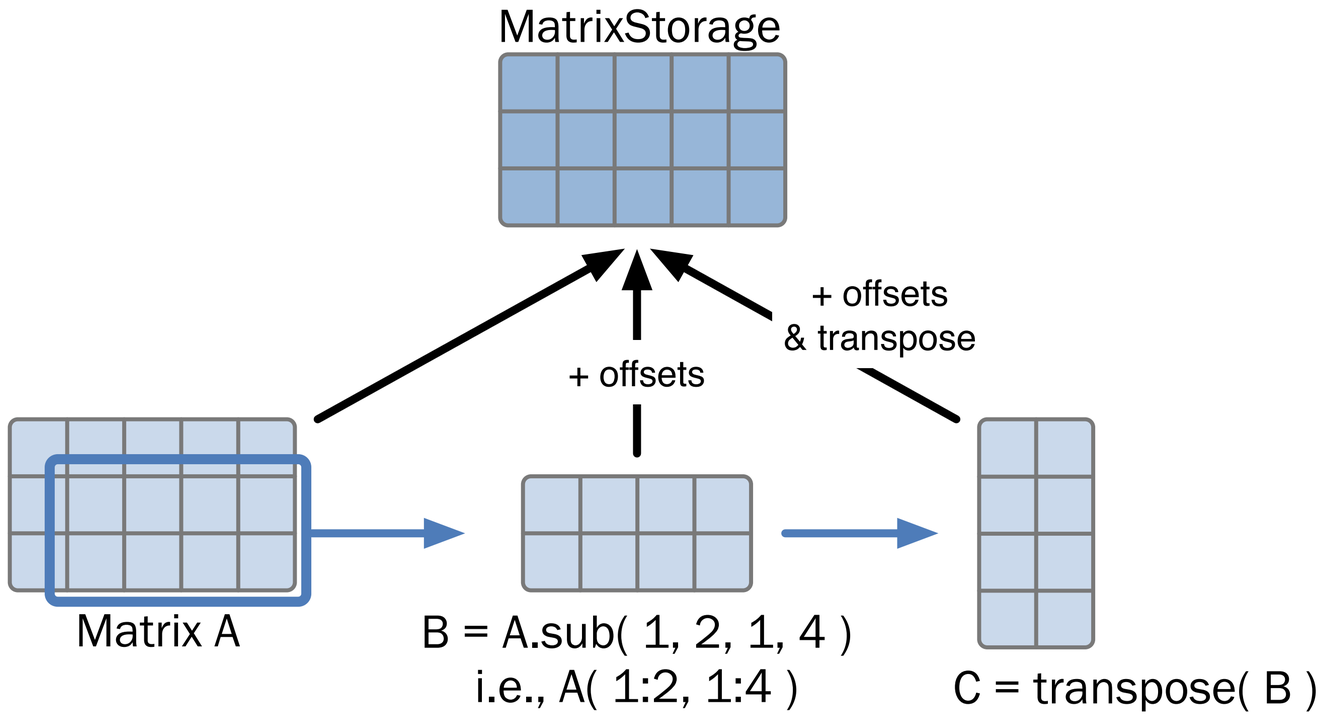


Shallow copy semantics

Matrix is a view onto a map of tiles, using a shared pointer

Matrices and tiles generally use *shallow copies*, not deep copies

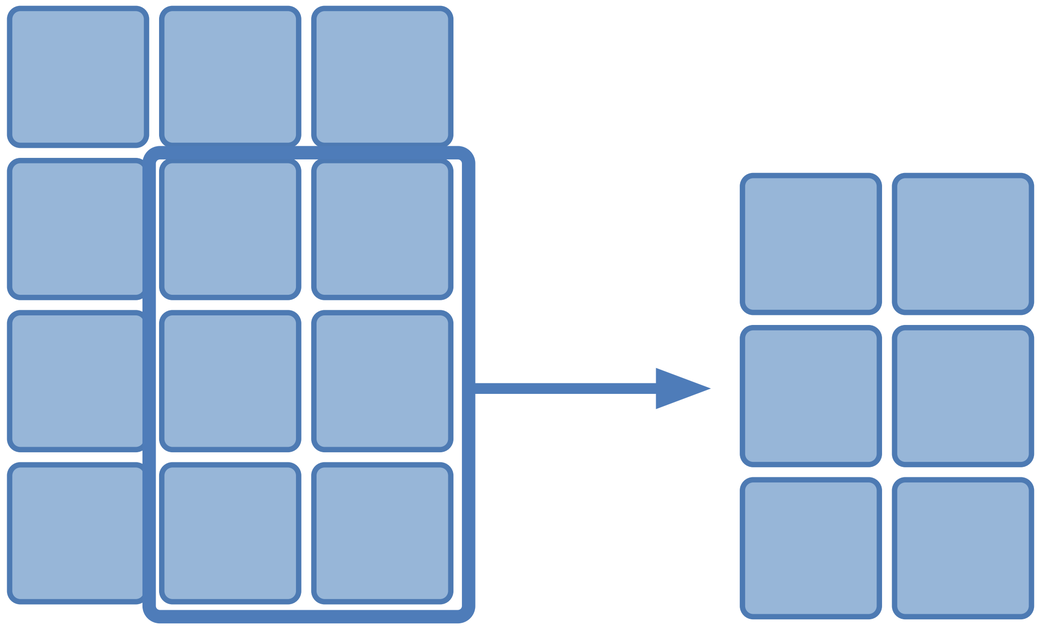
- A, B, C all view a subset of the same data



Sub-matrix

Sub-matrices based on tile indices

- A.sub(i1, i2, j1, j2) is A(i1 : i2, j1 : j2) inclusive



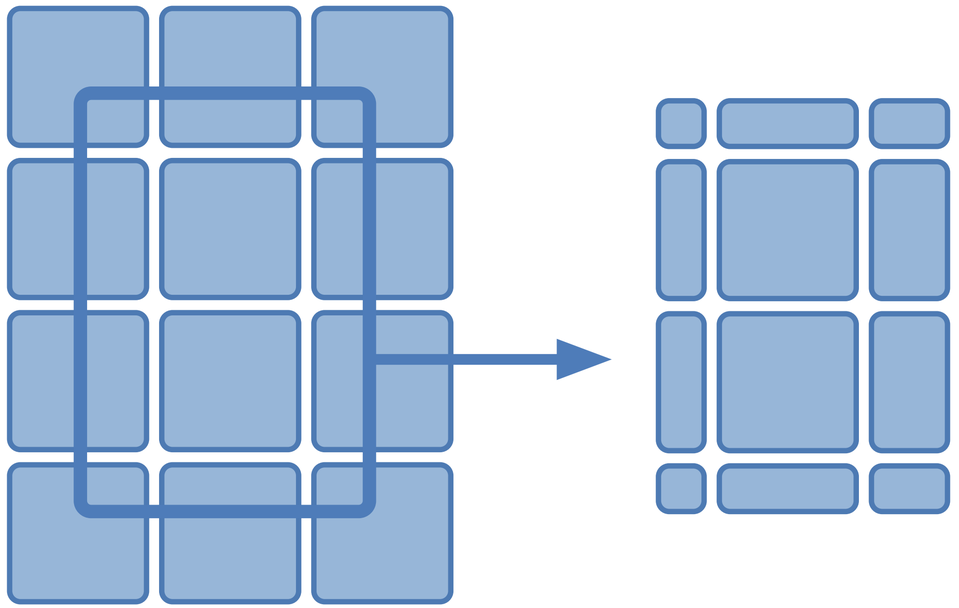
A.sub(1, A.mt()-1, 1, A.nt()-1) ⇒ trailing matrix

Shallow copy semantics!

Sliced Matrix

Slicing uses row & column indices, instead of tile indices. Can slice part of a tile.

- A.slice(row1, row2, col1, col2) is A(row1 : row2, col1 : col2), inclusive



A.slice(i1, i2, j1, j2) ⇒ arbitrary region

Shallow copy semantics!

Less efficient than A.sub

Matrix Class Hierarchy

Properties

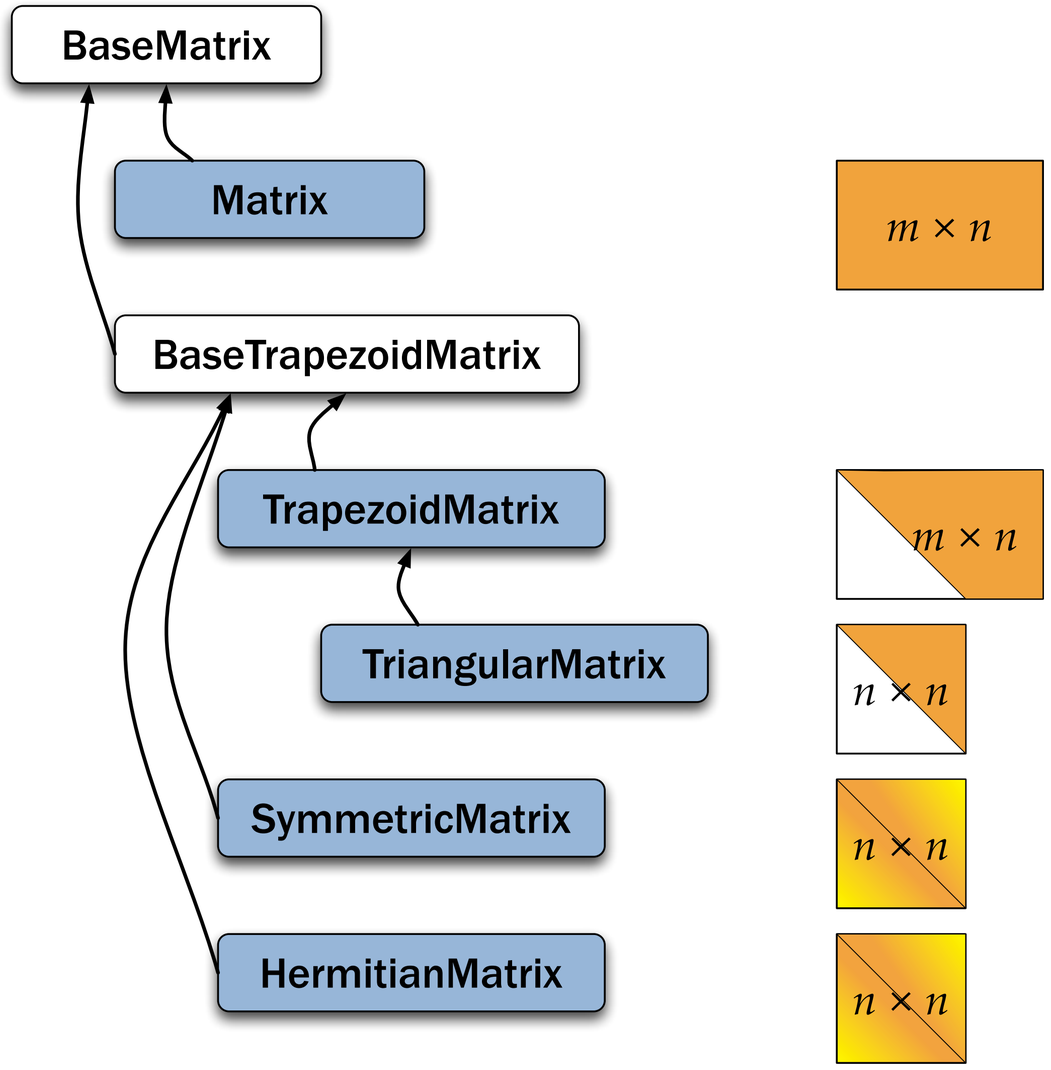
- Dimensions
- Transpose Operation
- Upper or Lower storage

Matrix handles transposition

- AT(i, j) returns A(j, i), where AT = transpose(A)

Algorithms implement 1 or 2 cases

- gemm has only 1 case (NoTrans–NoTrans) instead of 4 real or 9 complex cases
- trsm has only 2 cases (Left Lower & Upper) instead of 8 cases
- Cholesky has only 1 case (Lower), instead of 2 cases
- Other cases mapped by transposing



Tile Algorithms

Tile algorithms on CPU vs. GPU

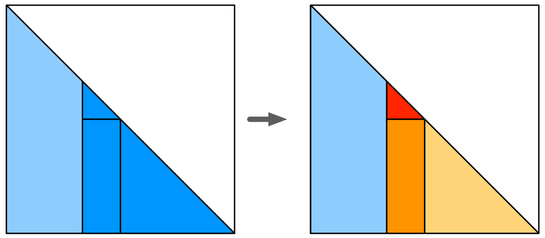
Tasks and Dependencies

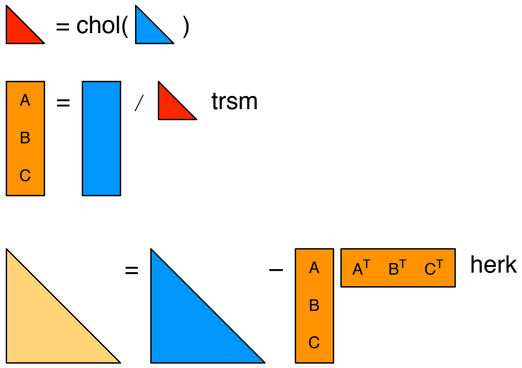
Tile Algorithms

Decompose large operations into many small operations on tiles

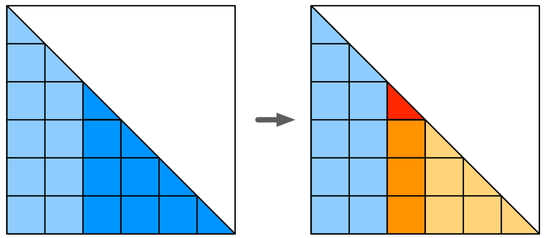
Track dependencies between tiles

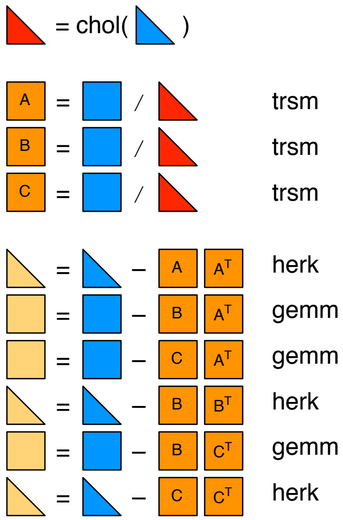
LAPACK Algorithm



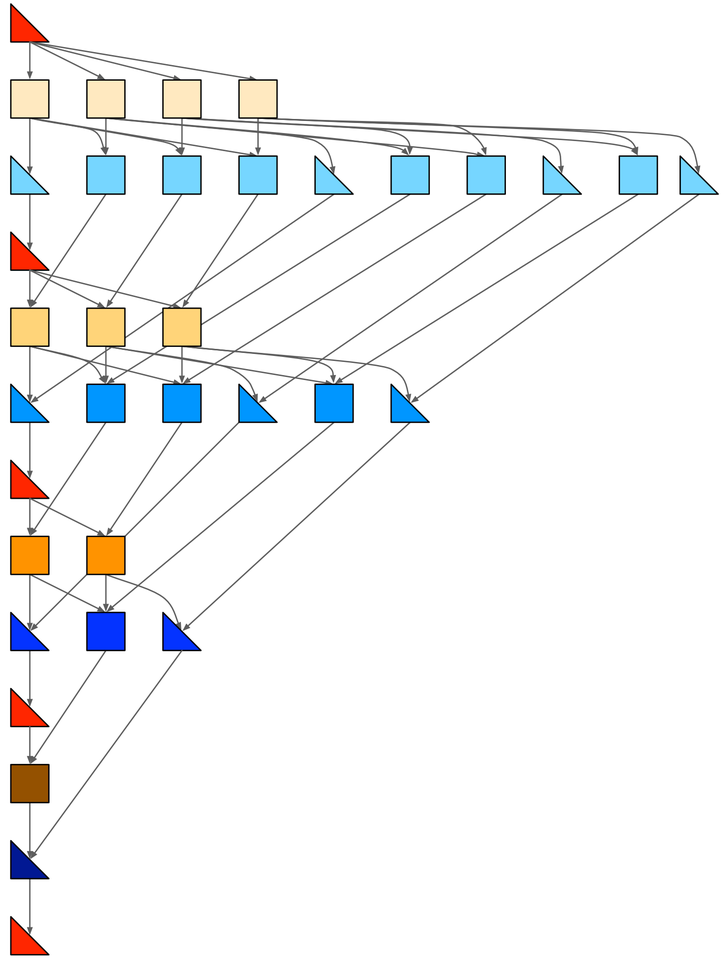


Tile Algorithm





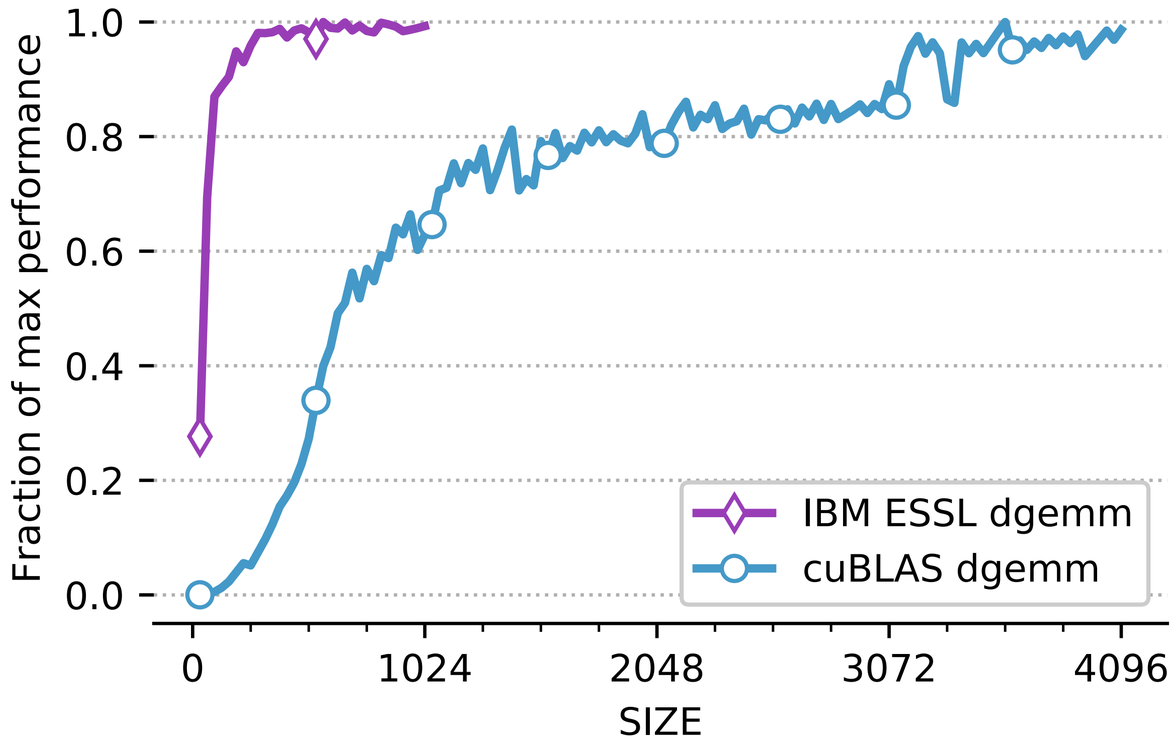
Task Graph (DAG)



Tile Algorithms

On CPUs, vendor BLAS (here, ESSL) quickly reaches peak performance (90% at nb ≥ 160)

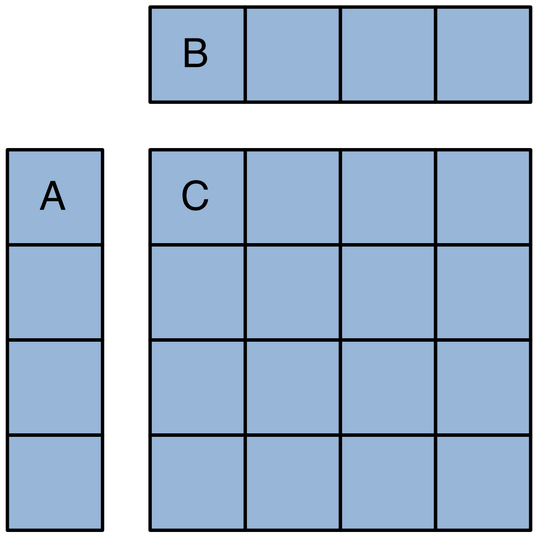
On GPUs, cuBLAS requires large size for peak performance (90% at nb ≥ 3136)

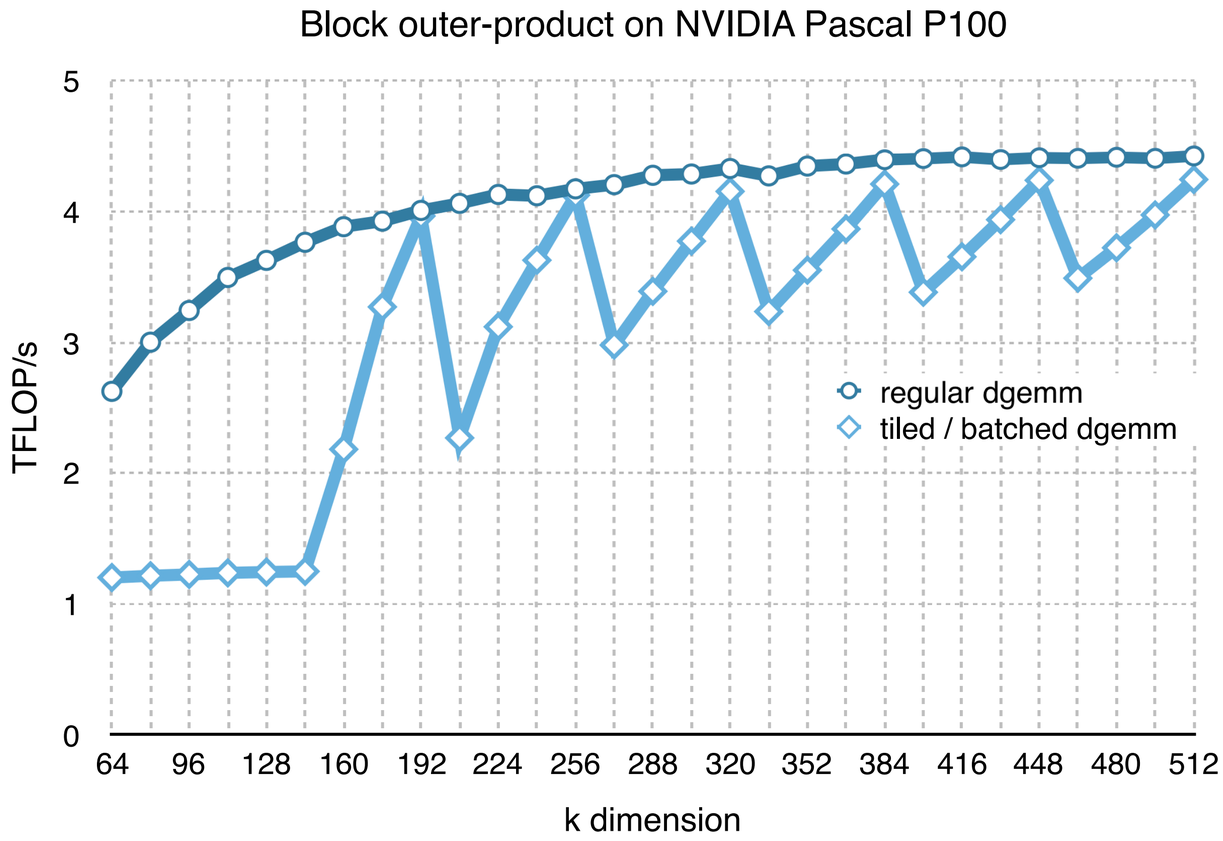


Results on IBM Power8 CPU and NVIDIA P100 GPU

Tile Algorithms

Block outer product using batched gemm matches peak at “nice” tile sizes



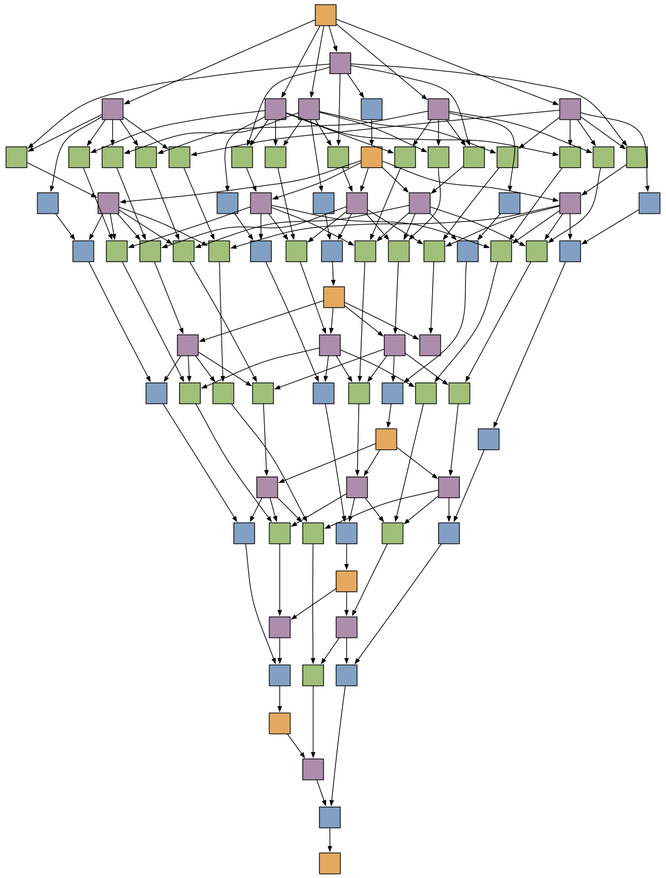


m = n = 40,000

Tasks and dependencies

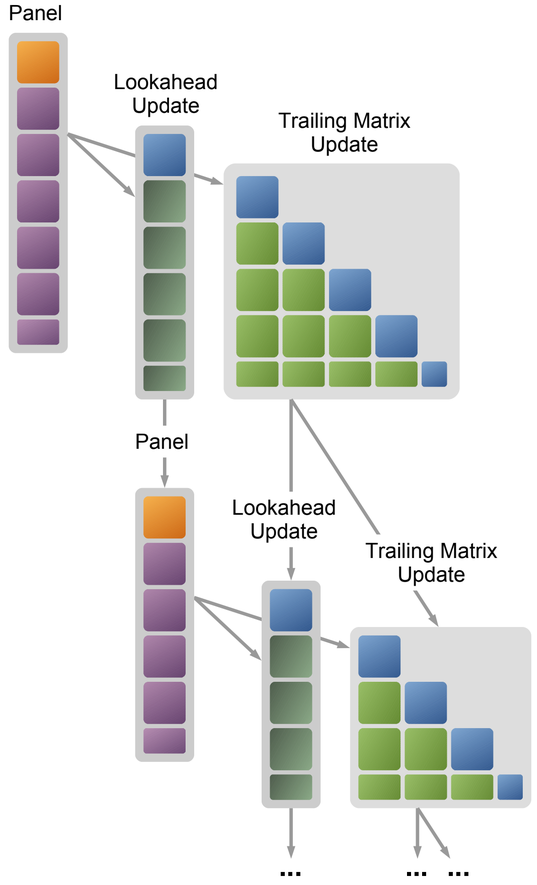
Traditional tile-by-tile data flow

- $O(n^3)$ tasks and dependencies



SLATE uses large tasks

- O(n) tasks and dependencies



Communication

MPI communication

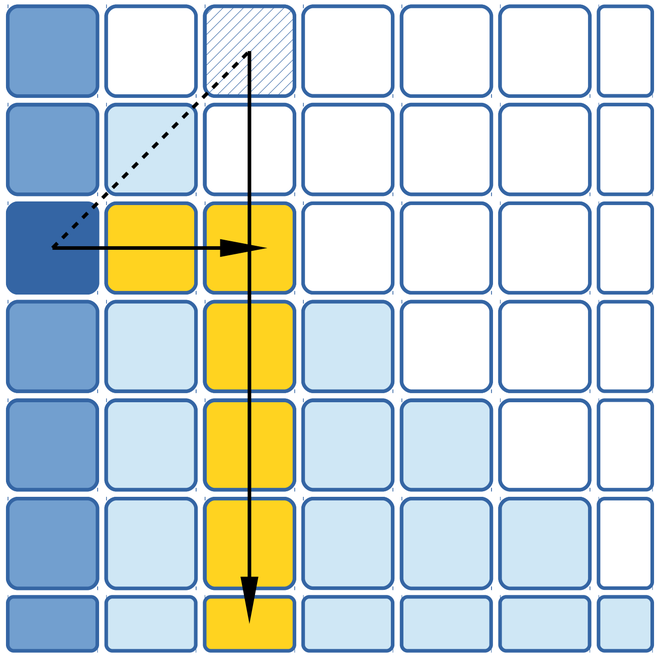
CPU ⟺ GPU communication

Message passing communication

Broadcast tile to area of matrix it will update

Matrix figures out which nodes that area covers

- Broadcast implemented by point-to-point communication in hypercube, rather than building expensive MPI communicators



Node-level memory consistency

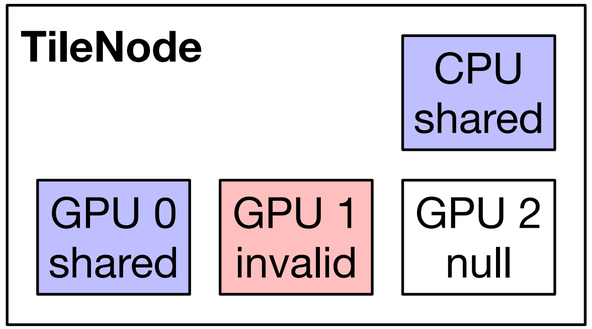
Tiles can be allocated in CPU or multiple GPU memories

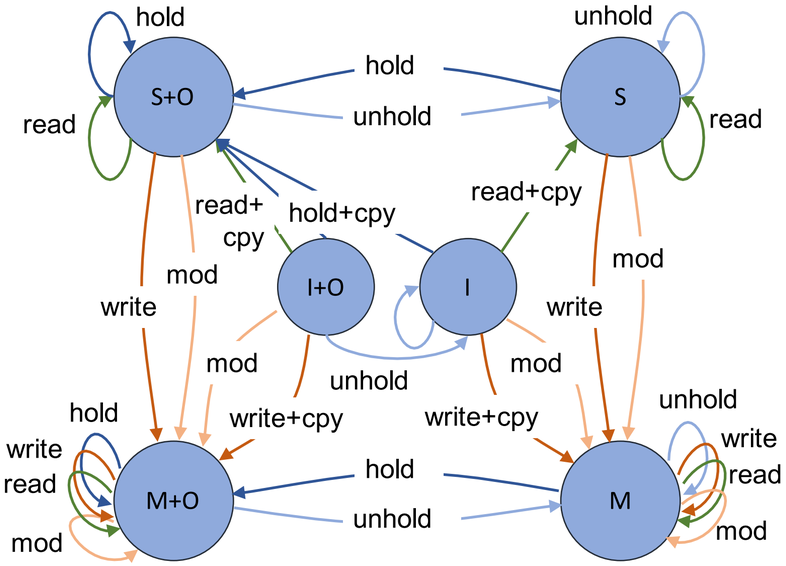
Memory consistency inspired by MOSI cache coherency model

- Modified — data is valid, others are invalid
- Shared — data is valid, others are invalid or shared
- Invalid — data is invalid
- OnHold — flag to prevent purging tile; orthogonal to MSI state

API

- tileGetForWriting(tile or set of tiles, device)
- tileGetForReading(tile or set of tiles, device)
- tileGetAndHold(tile or set of tiles, device)





SLATE API Layers

Driver

bitbucket.org/icl/slate/src/default/src/posv.cc

- Solve entire problem: Ax = b, eigenvalues, SVD, ...

Computational

bitbucket.org/icl/slate/src/default/src/potrf.cc

bitbucket.org/icl/slate/src/default/src/trsm.cc

- Compute one piece of problem
  - factor A = LU, A = LL^H, A = QR (getrf, potrf, geqrf, ...)
  - multiply C = αAB + βC (gemm, ...)

Internal

bitbucket.org/icl/slate/src/default/src/internal/internal_gemm.cc

- Big tasks: LU panel, trailing matrix update (block outer product)
- Generally composed of independent tasks, can be done as batch

Tile

bitbucket.org/icl/slate/src/default/include/slate/Tile_blas.hh

- On CPU, call BLAS++ and LAPACK++ wrappers around vendor BLAS and LAPACK

Early results on Summitdev

16 nodes, 2x Power8 + 4x Pascal P100 per node

CPU peak performance: 9. Tflop/s double, 18. Tflop/s single

GPU peak performance: 348. Tflop/s double, 696. Tflop/s single

Run SLATE 1 MPI rank per node (changed in later results)

Run ScaLAPACK 1 MPI rank per core

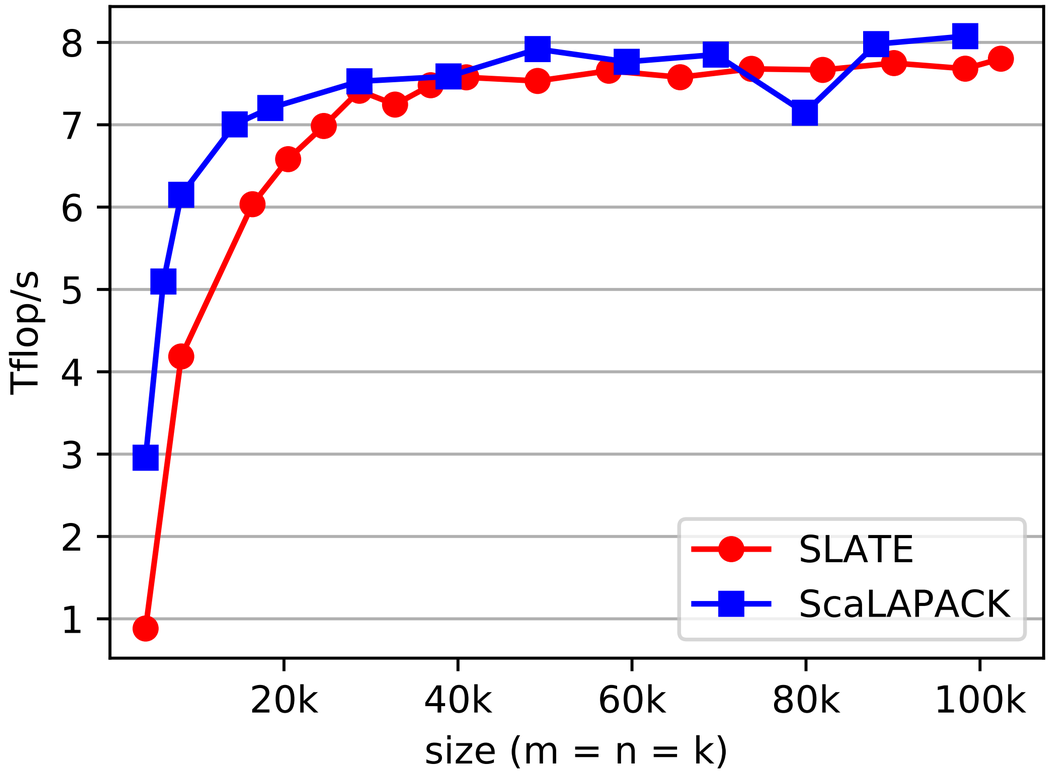
ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Summitdev — matrix multiply, double (dgemm)

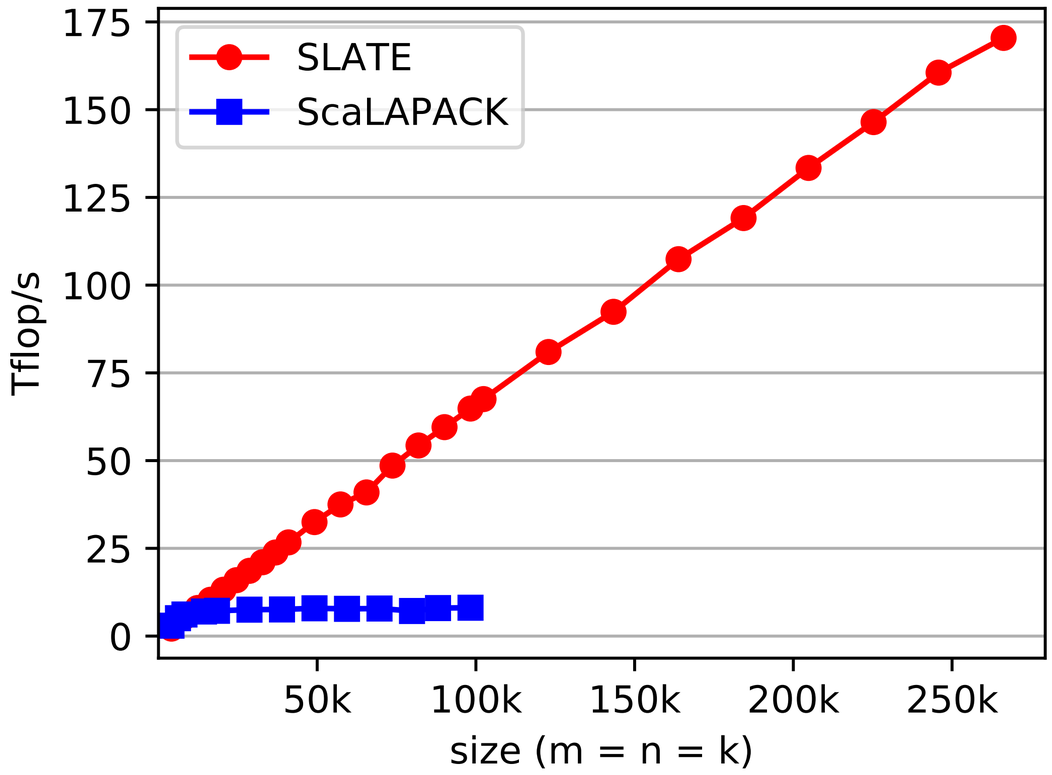
CPUs only

16 nodes x 2 sockets x 10 cores = 320 cores (IBM Power8)



CPUs + GPU accelerators

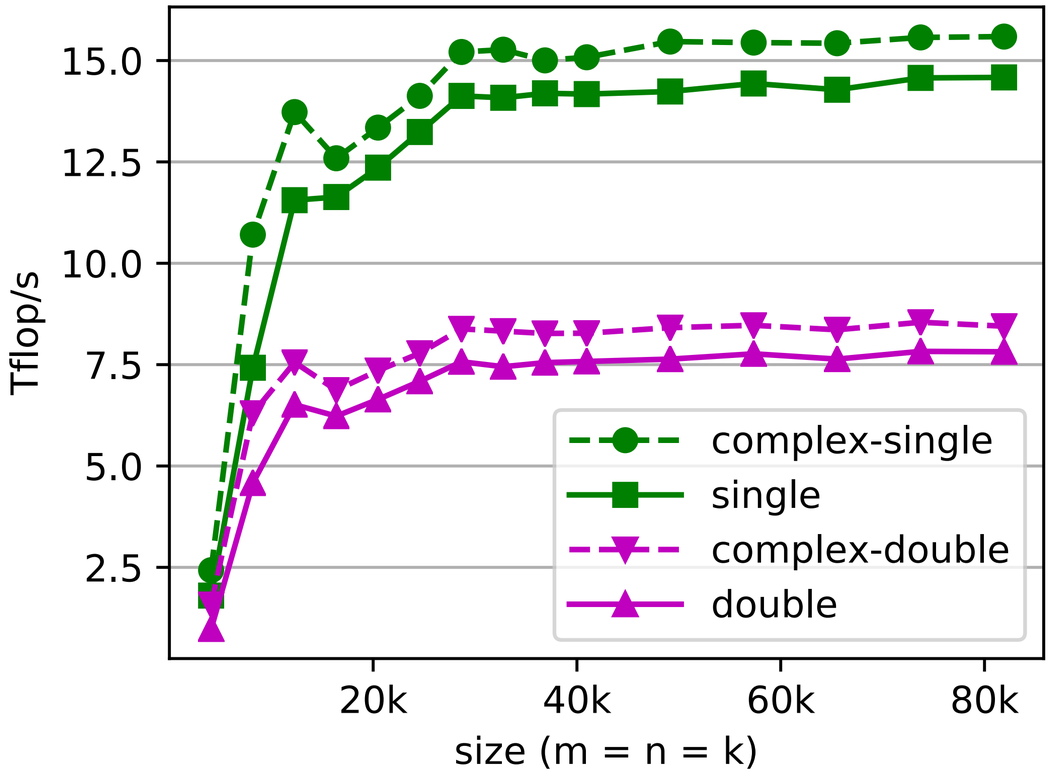
16 nodes x 4 devices = 64 devices (NVIDIA P100)



Summitdev — matrix multiply, all precisions

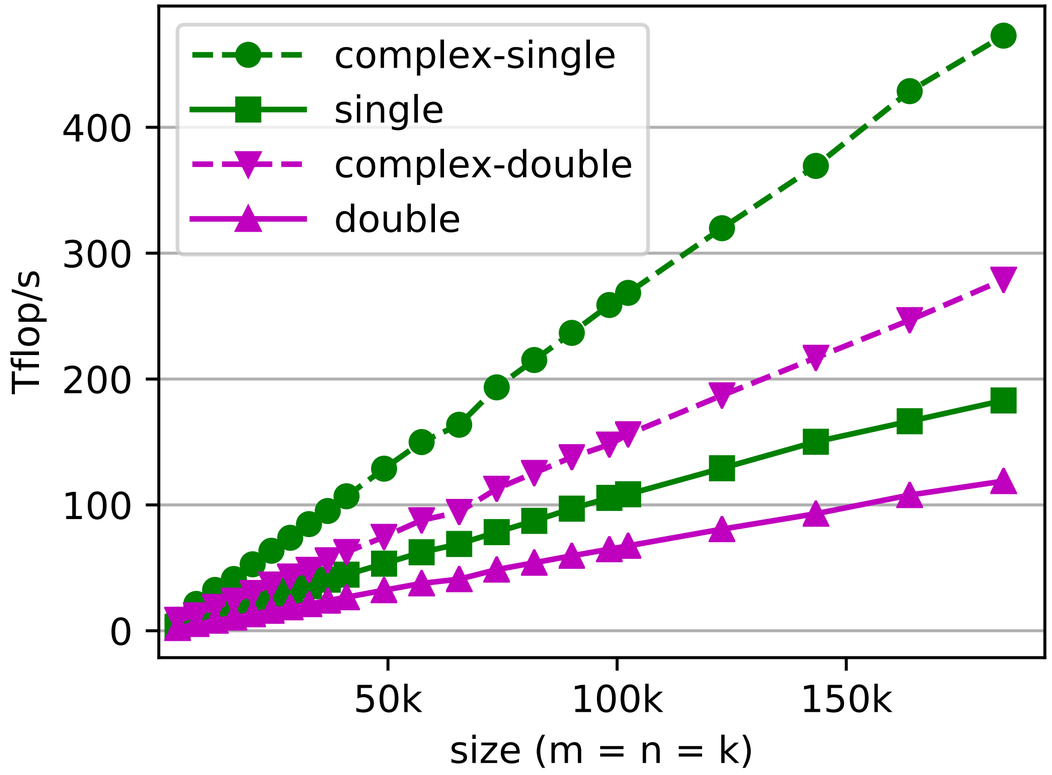
CPUs only

16 nodes x 2 sockets x 10 cores = 320 cores (IBM Power8)



CPUs + GPU accelerators

16 nodes x 4 devices = 64 devices (NVIDIA P100)

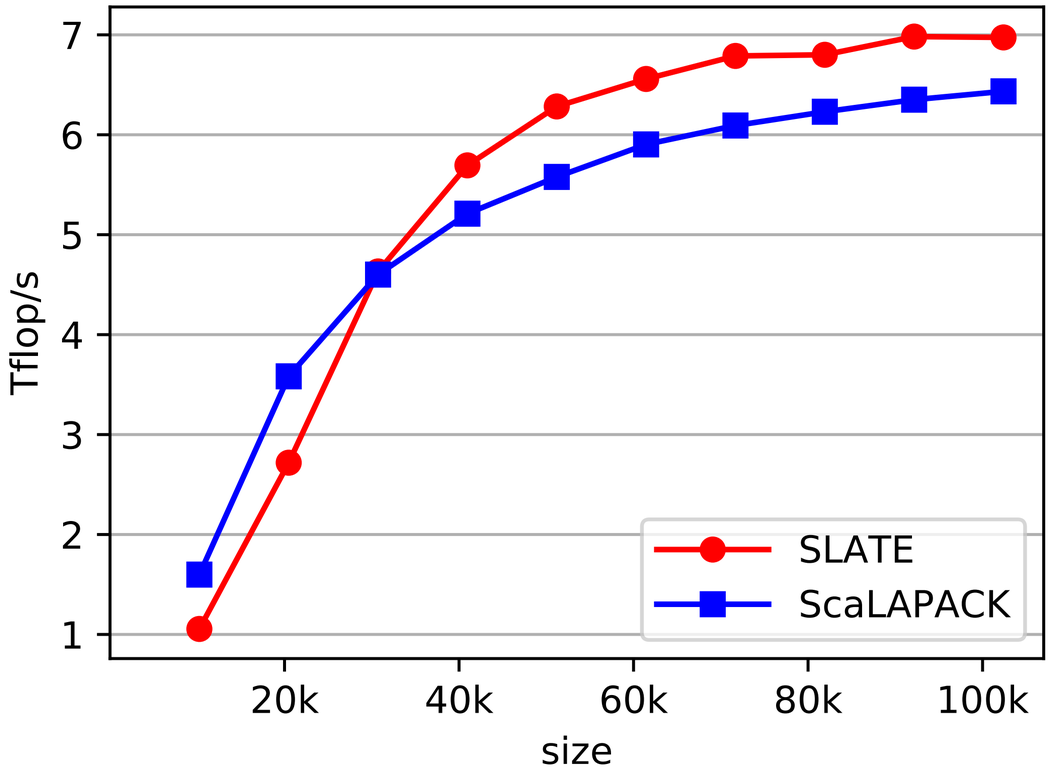


ICL THE UNIVERSITY OF TENNESSEE KNOXVILLE

Summitdev — Cholesky factorization, double (dpotrf)

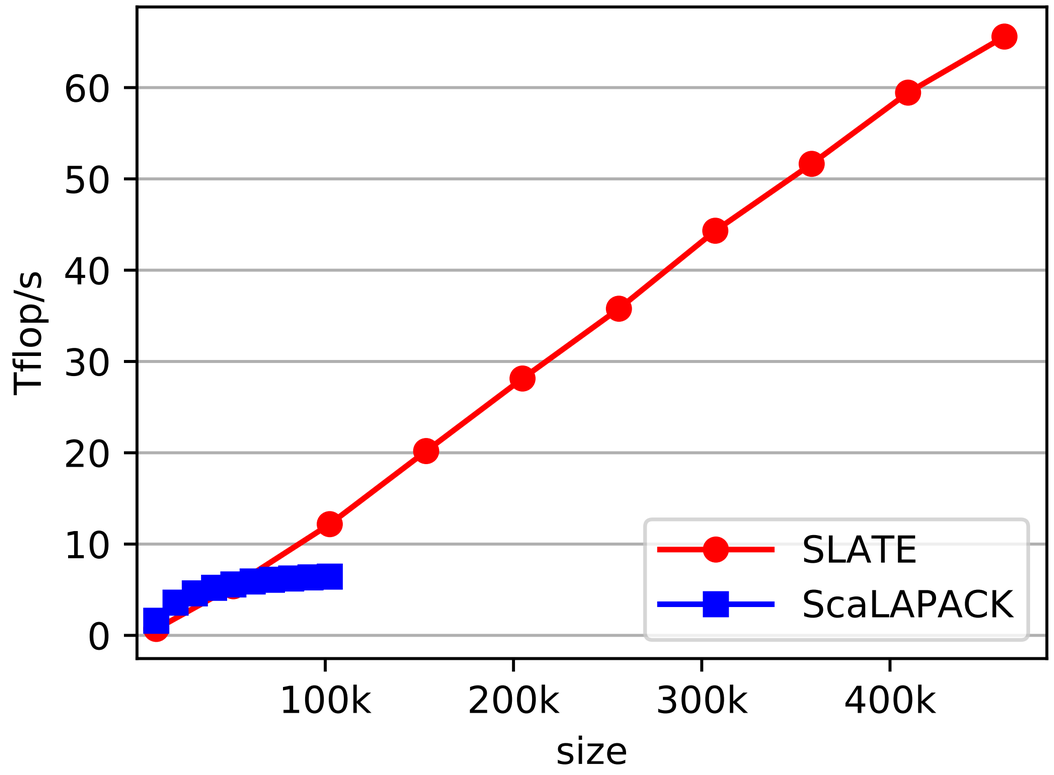
CPUs only

16 nodes x 2 sockets x 10 cores = 320 cores (IBM Power8)



CPUs + GPU accelerators

16 nodes x 4 devices = 64 devices (NVIDIA P100)



ICL THE UNIVERSITY OF TENNESSEE KNOXVILLE

Recent results on Summit

16 nodes, 2x Power9 + 6x Volta V100 per node

CPU peak performance: 17. Tflop/s double, 33. Tflop/s single

GPU peak performance: 765. Tflop/s double, 1540. Tflop/s single

Run SLATE 2 MPI ranks per node

Run ScaLAPACK 1 MPI rank per core

1 MPI rank per socket (2 per node), instead of 1 MPI rank per node

- Eliminates threads accessing cross-socket NUMA memory
- Better utilizes dual-rail InfiniBand network

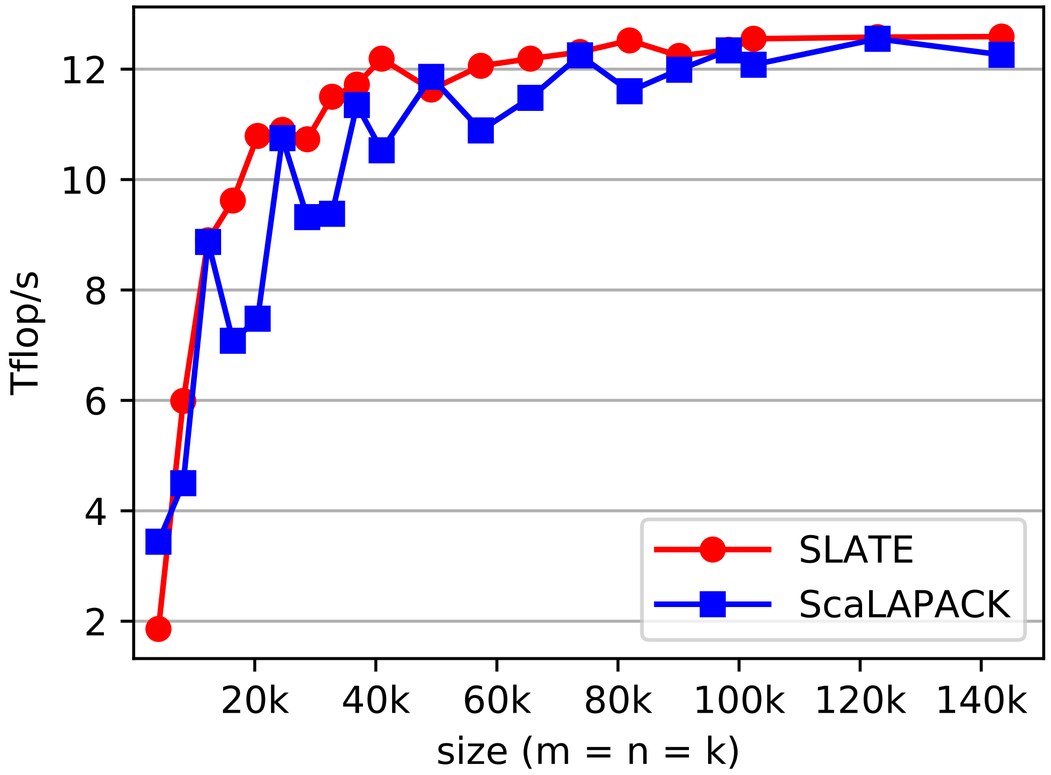
ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

Summit — matrix multiply, double (dgemm)

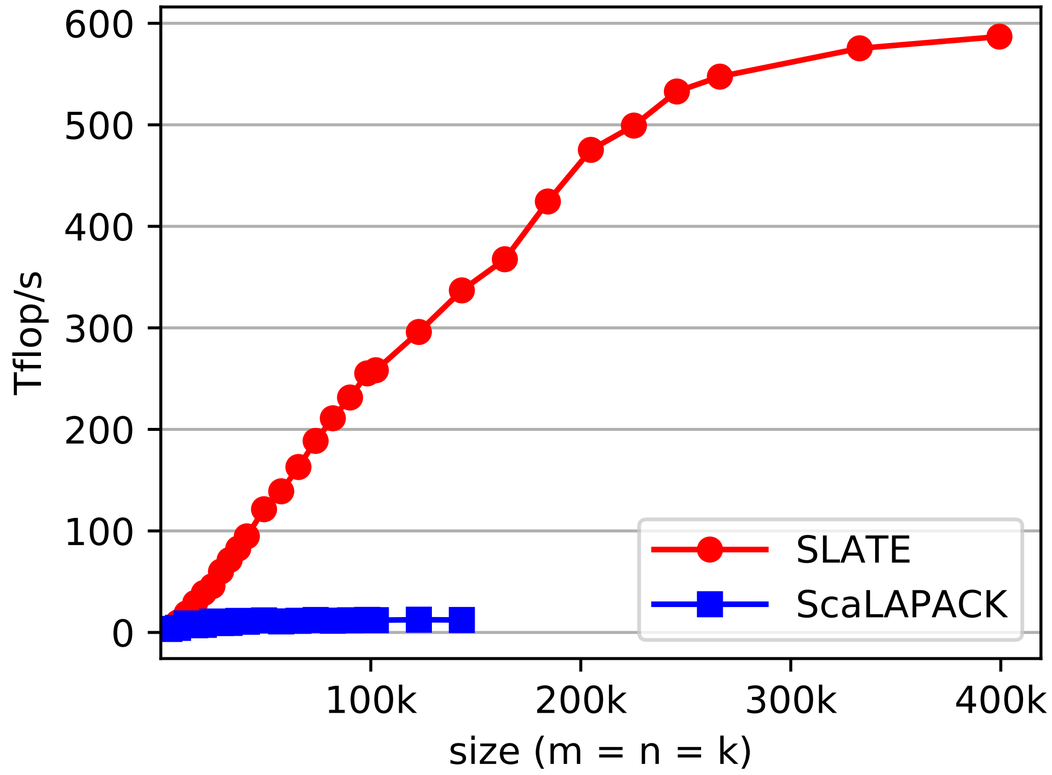
CPUs only

16 nodes x 2 sockets x 21 cores = 672 cores (IBM Power9)



CPUs + GPU accelerators

16 nodes x 2 sockets x 3 devices = 96 devices (NVIDIA V100)

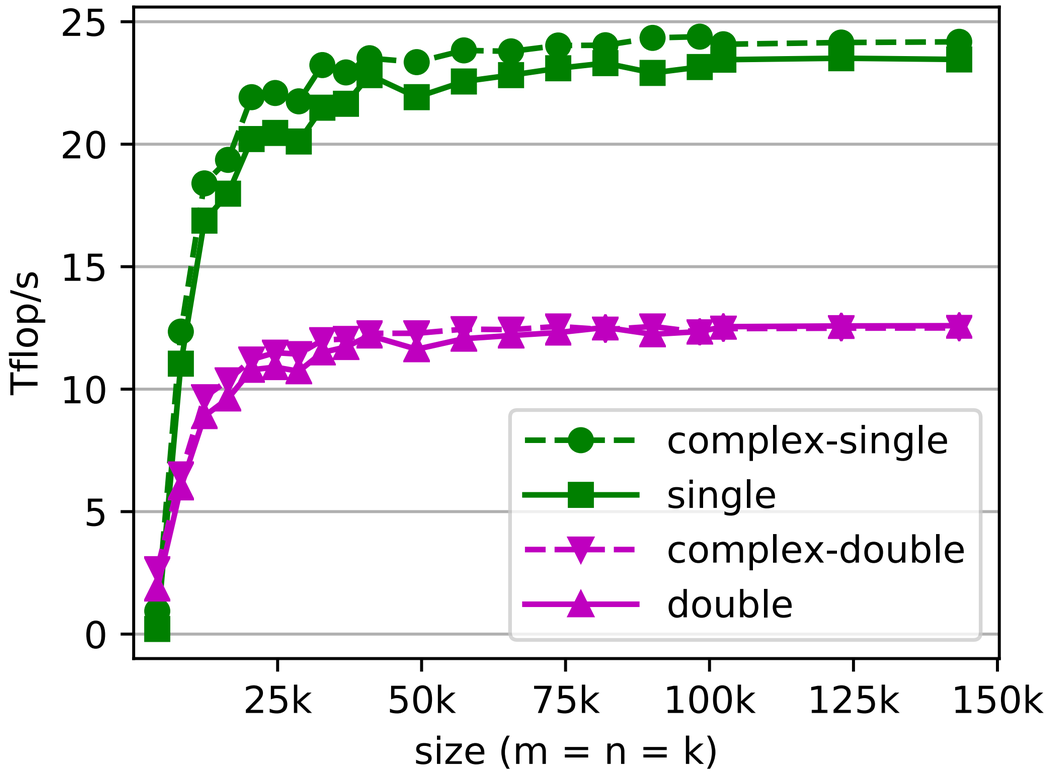


ICL THE UNIVERSITY OF TENNESSEE KNOXVILLE

Summit — matrix multiply, all precisions

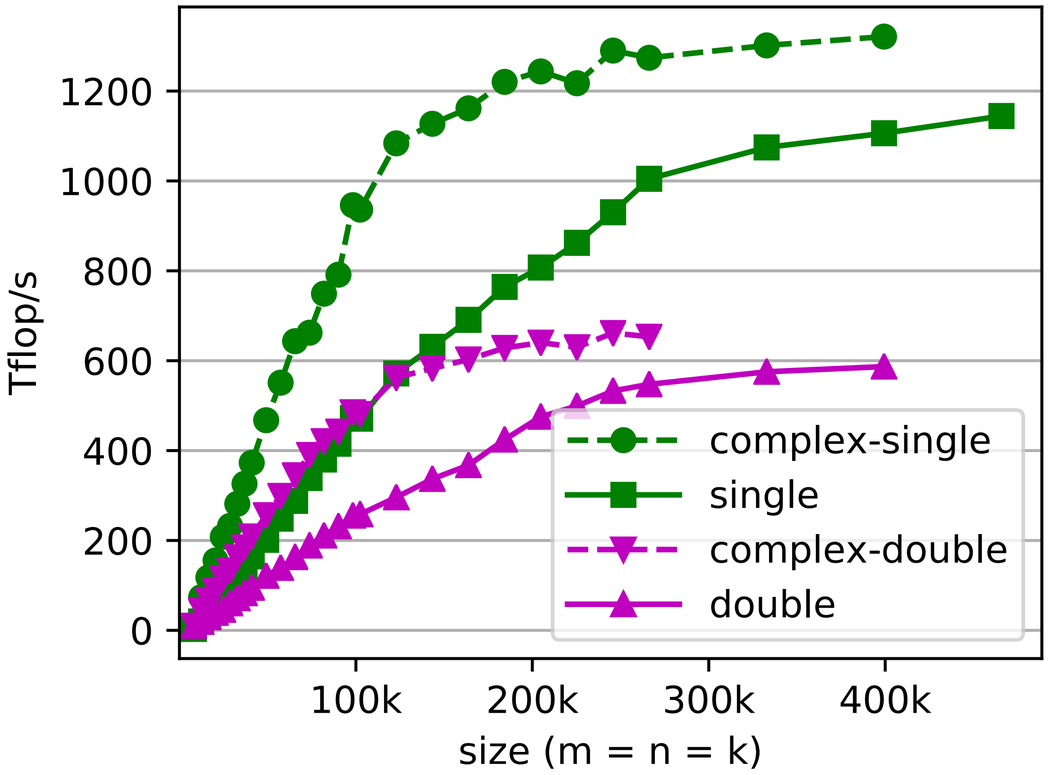
CPUs only

16 nodes x 2 sockets x 21 cores = 672 cores (IBM Power9)



CPUs + GPU accelerators

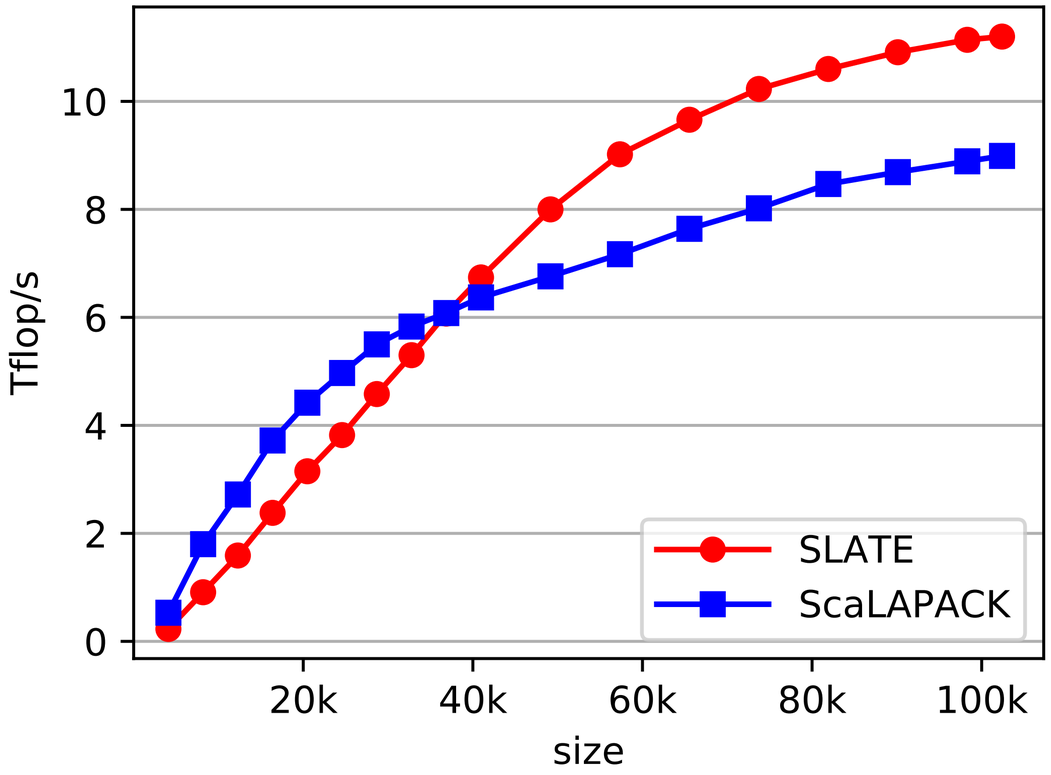
16 nodes x 2 sockets x 3 devices = 96 devices (NVIDIA V100)



Summit — Cholesky factorization, double (dpotrf)

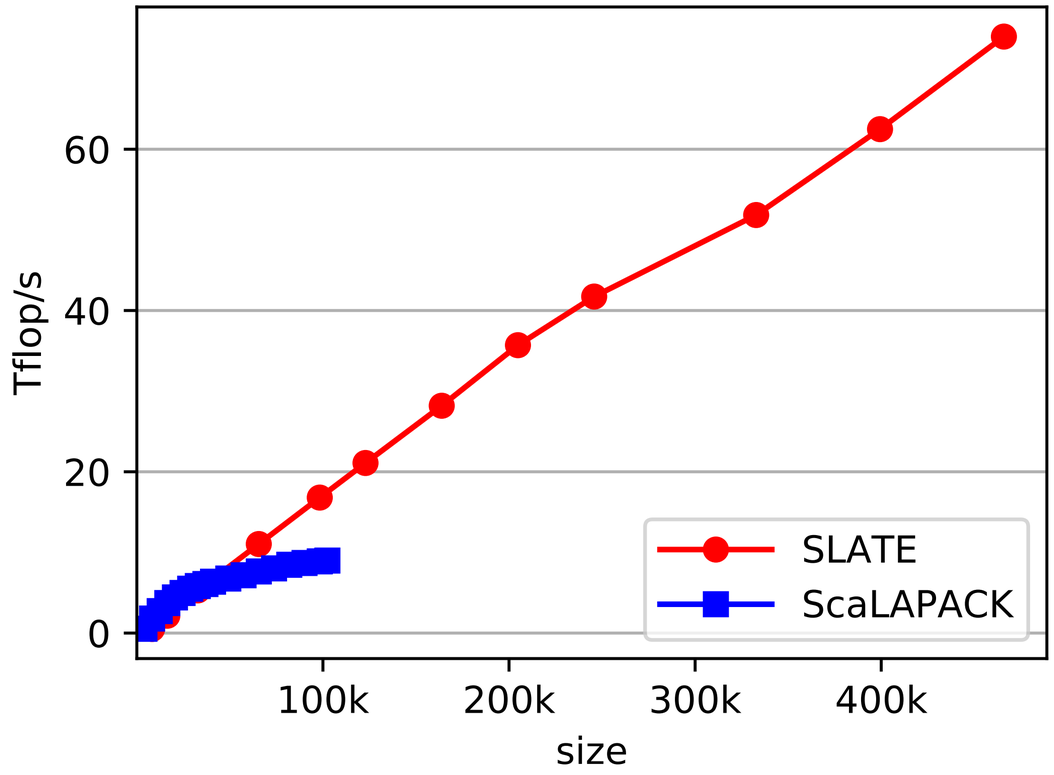
CPUs only

16 nodes x 2 sockets x 21 cores = 672 cores (IBM Power9)



CPUs + GPU accelerators

16 nodes x 2 sockets x 3 devices = 96 devices (NVIDIA V100)



Future

Less cryptic names, such as:

- cholesky_factor(A), cholesky_solve(A, B), lu_factor(A), lu_solve(A, B)

Overloaded names

multiply(A, B, C)

- General Matrix ⇒ gemm
- Symmetric Matrix ⇒ symm
- Hermitian Matrix ⇒ hemm

solve(A, B)

- Triangular A ⇒ triangular solve (trsm)
- Symmetric A ⇒ Cholesky (posv); fall back LDL^T or LU?
- General & square A ⇒ LU (gesv)
- Rectangular A ⇒ Least squares (gels)

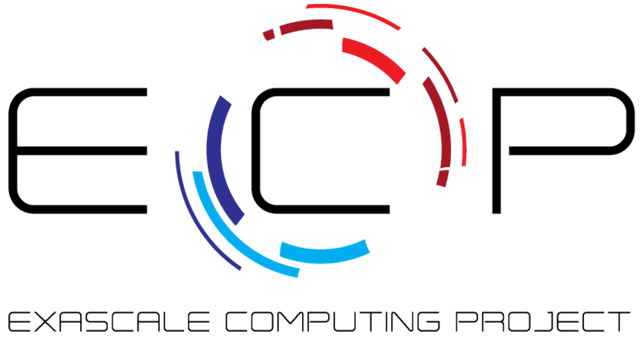
Availability

http://icl.utk.edu/slate/

- Papers and SLATE Working Notes (SWANs)
- https://bitbucket.org/icl/slate/ SLATE repo
- https://bitbucket.org/icl/blaspp/ BLAS++
- https://bitbucket.org/icl/lapackpp/ LAPACK++
- Mercurial repo (transitioning to git)
- Issue tracking
- Pull requests for user contributions
- Modified BSD License

SLATE user email list

- slate-user@icl.utk.edu



This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

ICL is hiring!

Projects include

- SLATE — distributed dense linear algebra
- CEED — tensor algebra, batched operations
- PEEKS — Krylov methods
- heFFTe — distributed FFT
- PAPI — performance measurement and modeling
- ParSEC — distributed tasking for exascale

www.icl.utk.edu/jobs

ICL INNOVATIVE COMPUTING LABORATORY

THE UNIVERSITY OF TENNESSEE KNOXVILLE

THE UNIVERSITY OF
TENNESSEE
KNOXVILLE