A Tale of Four Packages

Jack Dongarra
University of Tennessee
Oak Ridge National Laboratory
University of Manchester

In collaboration with Ahmad Abdelfattah, Mark Gates, Azzam Haidar, Jakub Kurzak, Piotr Luszczek, Stan Tomov, Asim YarKhan, and more at ICL and many more internationally
Outline for the Talk

• Its about dense linear algebra software and packages
  • LAPACK and ScaLAPACK in the ‘90s & ‘00s
  • PLASMA in the ‘10s
  • MAGMA in the ‘10s
  • SLATE in the ‘20s
DLA Solvers

• We are interested in developing Dense Linear Algebra Solvers
• Retool LAPACK and ScaLAPACK for multicore and hybrid architectures
  • These are two very successful packages
  • They have transitioned to vendors, both hardware and software, which provide optimized versions.
Dense Linear Algebra

- **Common Operations**

\[ Ax = b; \quad \min_{x} \| Ax - b \|; \quad Ax = \lambda x \]

- A major source of large dense linear systems is problems involving the solution of boundary integral equations.
  - The price one pays for replacing three dimensions with two is that what started as a sparse problem in \( O(n^3) \) variables is replaced by a dense problem in \( O(n^2) \).

- Dense systems of linear equations are found in numerous other applications, including:
  - Electronic structures;
  - Maxwell equations;
  - Plasma containment;
  - Airplane wing design;
  - Radar cross-section studies;
  - Flow around ships and other off-shore constructions;
  - Diffusion of solid bodies in a liquid;

### 50 Years Evolving SW and Alg

**Tracking Hardware Developments**

<table>
<thead>
<tr>
<th>Software/Algorithms follow hardware evolution in time</th>
</tr>
</thead>
</table>
| **EISPACK (1970s)**  
(Translation of Algol to F66) | ![Image] | Rely on  
- Fortran, but row oriented |
| **LINPACK (1980s)**  
(Vector operations) | ![Image] | Rely on  
- Level-1 BLAS operations  
- Column oriented |
| **LAPACK (1990s)**  
(Blocking, cache friendly) | ![Image] | Rely on  
- Level-3 BLAS operations |
| **ScalAPACK (2000s)**  
(Distributed Memory) | ![Image] | Rely on  
- PBLAS for Message Passing |
| **PLASMA & MAGMA (2010s)**  
New Algorithms  
(many-core friendly & GPU) | ![Image] | Rely on  
- DAG/scheduler  
- block data layout  
- some extra kernels |
| **SLATE (2020s)** | ![Image] | Distributed Memory  
Rely on  
- C++  
- Tasking DAG scheduling  
- Tiling, but tiles can come from anywhere  
- Batched Dispatch |
• 1974: Effort to standardize Basic Linear Algebra Subprograms
  • Basic LA vector operations
  • Referred to now as Level 1 BLAS
    • Dot product, 2-norm, $\alpha^*x+y$, $\alpha^*x$, etc.

• 1975: LINPACK Project started
  • Effort to produce portable, efficient linear algebra software for dense matrix computations.

• 1976: Vector computers in use for HPC

• 1977: DEC VAX system in common use
The Standard LU Factorization LINPACK
1970’s HPC of the Day: Vector Architecture

Main points
- Fortran was the language, implied column orientation
- Factorization column (zero) mostly sequential due to memory bottleneck
- Level 1 BLAS
- Divide pivot row has little parallelism
- OK on machines with excess memory bandwidth, but
- Too much data movement per step
1984 - 1990

• “Attack of the Killer Micros”, Brooks @ SC90
• Cache based & SMP machines
• Blocked partitioned algorithms was the way to go
  • Reduce data movement; today’s buzzword “Communication avoiding”
• Level 2 BLAS standard published (mat-vec ops)
• Level 3 BLAS standardization started (mat-mat ops)
- LAPACK Published
- ScaLAPACK started
LAPACK Software
Jointly with UTK and UCB and Many Other Contributors

• First release in February 1992
• Current: LAPACK Version 3.9.0 (Nov, 2019) ~2M LoC
• LICENSE: Mod-BSD, freely-available software package - Thus, it can be included in commercial software packages (and has been). We only ask that proper credit be given to the authors.
• Public GitHub repository
• 4 Precisions: single, double, complex, double complex
  • Considering 16-bit floating point version
• Multi-OS *nix, macOS, Windows
• Multi-build support (Make and Cmake)
• Reference BLAS and CBLAS
• LAPACKE: Standard C language APIs for LAPACK
• Prebuilt Libraries for Windows
• Extensive test suite
• Forum and User support: http://icl.cs.utk.edu/lapack-forum/
• Goal: bug free library – Since 2009, 165 bugs reported, only 11 pending correction
### LAPACK Functionality

<table>
<thead>
<tr>
<th>Type of Problem</th>
<th>Acronyms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear systems of equations</td>
<td>SV</td>
</tr>
<tr>
<td>Linear least squares problems</td>
<td>LS</td>
</tr>
<tr>
<td>Linear equality-constrained least squares problems</td>
<td>LSE</td>
</tr>
<tr>
<td>General linear model problem</td>
<td>GLM</td>
</tr>
<tr>
<td>Symmetric eigenproblems</td>
<td>EV</td>
</tr>
<tr>
<td>Nonsymmetric eigenproblems</td>
<td>EV</td>
</tr>
<tr>
<td>Singular value decomposition</td>
<td>SVD</td>
</tr>
<tr>
<td>Generalized symmetric definite eigenproblems</td>
<td>GV</td>
</tr>
<tr>
<td>Generalized nonsymmetric eigenproblems</td>
<td>GG</td>
</tr>
<tr>
<td>Generalized (or quotient) singular value decomposition</td>
<td>GG</td>
</tr>
</tbody>
</table>
ScaLAPACK

- Library of software dealing with dense & banded routines
- Distributed Memory - Message Passing
  - When project started MPI didn’t exist
- MIMD Computers and Networks of Workstations, Clusters of SMPs
- Data layout critical for performance

- Relies on LAPACK / BLAS and BLACS / MPI
- Includes PBLAS (Parallel BLAS)
ScaLAPACK Programming Style

- **SPMD Fortran 77 using an object based design**
- **Built on various modules**
  - **PBLAS** Interprocessor communication & computation
    - **BLAS**
    - **BLACS**
      - Targeted PVM, IBM SP, CRI T3, Intel, TMC
      - MPI when standardized
      - Provides right level of abstraction.
- **Object based - Array descriptor**
  - Contains information required to establish mapping between a global array entry and its corresponding process and memory location.
  - Provides a flexible framework to easily specify additional data distributions or matrix types.
  - Currently dense, banded, & out-of-core
- **Using the concept of context**
Performance Issues with ScaLAPACK

• The major problem with ScaLAPACK is the lack of overlap of computation and communication.
  • No overlap, resulting in performance issues

• Each phase done separately, bulk synchronous.
  • Computation phase then a communication phase.
  • All (most) processes compute then a communication phase (broadcast)
  • This is how the PBLAS operate.

• Need a “new” interface which allows computation and communication to take place simultaneously, in an asynchronous fashion.
OpenMP in LAPACK and ScaLAPACK

• LAPACK and ScaLAPACK, in general, don’t use OpenMP directly, just in the BLAS kernels
  • So to some extent the BLAS may be implemented using OpenMP
• There is an exception – one of the newer routines
  • 2-stage eigenvalue routines for the bulge chasing.
  • LAPACK has OpenMP in the "bulge chasing" stage
  • for 2-stage symmetric and hermitian eigensolver.
• The routines are:
  • real: {s,d}sytrd_sb2st
  • complex: {c,z}hetrd_hb2st
• It uses tasking.
  • If tasks were not used then only a single core cache would be used.
  • With tasks, the caches are combined and data reuse increases.
• And all of this is in Fortran.
Since LAPACK and ScaLAPACK

• A lot has changed
  • OpenMP
  • Manycore and accelerators
  • Use a different set of ideas to provide efficient use of underlying hardware
    • PLASMA/DPLASMA
    • MAGMA
PLASMA

• PLASMA is a dense linear algebra library
  • For shared-memory multi-core processors.
  • Algorithms are expressed as sequential kernels acting on tiles of data
  • Runtime takes sequential kernels (tasks), uses task-superscalar scheduling, and exposes parallelism
  • Linear algebra for OpenMP
    - dataflow scheduling
    - tile matrix layout
    - tile algorithms

A. Buttari, J. Langou, J. Kurzak, J. Dongarra,
A class of parallel tiled linear algebra algorithms for multicore architectures,
Tile Algorithms

LAPACK Algorithm

= \text{cho}( )

= \text{trsm} \newline

= \text{herk}
Tile Algorithms

- Decompose large operations into many small operations on tiles
- Track dependencies between tiles
- Parallelism implicit in task graph

LAPACK Algorithm

\[
\begin{align*}
A & = \text{cho}( ) \\
A'B' & = \text{trsm} \\
A'B'C' & = \text{herk}
\end{align*}
\]

Tile Algorithm

\[
\begin{align*}
A & = \text{cho}( ) \\
A'B' & = \text{trsm} \\
A'B'C' & = \text{trsm} \\
A'B'C' & = \text{herk} \\
A'B'C' & = \text{gemm} \\
A'B'C' & = \text{herk}
\end{align*}
\]
Execution trace

- LAPACK-style fork-join leave cores idle

24 cores
Matrix is 8000 x 8000, tile size is 400 x 400.
Execution trace

- PLASMA squeezes out idle time

24 cores
Matrix is 8000 x 8000, tile size is 400 x 400.
QUARK Runtime system for PLASMA

- PLASMA needed a way to express the DAGs
- Initial dataflow execution engine in PLASMA
  - For each task inserted, data is marked R, W, RW
  - Future tasks accessing data create a dependency
  - The dependencies form an implicit task-DAG
- QUARK uses superscalar execution
  - Creates a list of tasks and data accessed
  - Tracks data dependencies
  - Launches out-of-order parallel task execution
  - Uses a window of active tasks to limit memory usage
- QUARK allows task-priorities, task-locality, multi-threaded tasks, task-sequence cancellation, incremental runtime-dependencies and other execution ideas...
PLASMA: Original Design  (Discontinued with Version 2.8)

- PLASMA Distribution
  - PLASMA
    - core BLAS
      - CBLAS
      - LAPACK
      - (C)LAPACK
    - BLAS
  - QUARK
    - POSIX Threads
    - WinThreads
    - hwloc
Dynamic Scheduling, OpenMP, GNU GCC

May 2008  |  OpenMP 3.0  
April 2009  |  GCC 4.4  
July 2013  |  OpenMP 4.0  
April 2014  |  GCC 4.9  
Nov. 2015  |  OpenMP 4.5  
April 2016  |  GCC 6.1  
Nov. 2018  |  OpenMP 5.0  
May 2019  |  GCC 9.1  
Nov. 2019  |  OpenMP 5.1 preview  

#pragma omp task
#pragma omp task depend
#pragma omp task priority
#pragma omp task affinity(A) detach(hndl)
PLASMA: From QUARK to OpenMP

- OpenMP 4.0 adopted task superscalar scheduling (2013)
  - OpenMP 4.5 added task priorities (2015)
- QUARK was phased out in favor of the standard OpenMP runtime
  - Compiler support removed the need to pack/unpack arguments

- All this:
  ```c
  void CORE_dpotrf_quark(Quark *quark)
  {
      PLASMA_enum uplo;
      int n;
      double *A;
      int lda;
      PLASMA_sequence *sequence;
      PLASMA_request *request;
      int *info;
      quark_unpack_args_7(quark, uplo, n, A, lda, sequence, request, *info);
      info = LAPACK_dpotrf_work(
          LAPACK_COL_MAJOR,
          lapack_const_uplo),
          n, A, lda);
      if (sequence->status == PLASMA_SUCCESS && info != 0)
          plasma_sequence_flush(quark, sequence, request, info+info);
  }
  
  #pragma omp task depend(inout:A(k, k)[0:nb*nb])
  LAPACK_dpotrf_work(
      LAPACK_COL_MAJOR,
      'L', nb, A(k, k), nb);
  ```

- Replaced by this:
PLASMA: From QUARK to OpenMP

• PLASMA tile algorithms map well from QUARK to OpenMP
  • QUARK task insertion maps directly to OpenMP task pragmas

• Task priorities allow tasks on the critical path to be prioritized
  • Tile algorithm tasks are sequentially presented to the runtime
  • Critical path tasks may not be exposed to the runtime early
  • Algorithms need to present-unroll tasks in the right order

• A few features are require attention to match with OpenMP, i.e...
  • QUARK has thread-data-affinity hinting, now in OpenMP 5
  • QUARK has multi-threaded tasks often called gang-tasks;
    • These are tasks that take multiple-threads which all work on a common activity like the panel.
  • QUARK tasks can be locked to threads or thread-masks (set of threads)

PLASMA with OpenMP

- PLASMA version 17 switched to OpenMP
  - Transition led to redesigning some algorithms; notably LU factorization
- OpenMP is supported by industry and community
  - Optimized implementations
  - New features (e.g. target offload to accelerators)
  - High adoption in HPC community
  - Allows interoperability with other OpenMP software


PLASMA – QR Factorization

**Tile QR**
- great for square matrices
- great for multicore processors

**TSQR / CAQR**
- great for tall and skinny matrices
- great for distributed memory
PLASMA – Algorithms – SVD/EVP (symmetric)

reduction to band
parallel & cache efficient
tile algorithm

band reduction
parallel & cache efficient
a flavor of communication avoiding

divide and conquer
task-based
dataflow
**PLASMA OpenMP Cholesky Performance**

Double precision Cholesky factorization

Intel Xeon E5-2650 v3 (Haswell), 2.3GHz, 20 cores

---

PLASMA Cholesky factorization using OpenMP

Intel Xeon E5-2650 v3 (Haswell) 2.3GHz 20 cores

Tiles of size 224 x 224, matrix of size 20 x 20 tiles (4480 x 4480)

- Red = potrf
- Orange = trsm
- Purple = syrk
- Green = gemm
PLASMA OpenMP Cholesky Inversion Trace

PLASMA Cholesky inversion using OpenMP
Intel Xeon E5-2650 v3 (Haswell) 2.3GHz 20 cores
tiles of size 224 x 224, matrix of size 13 x 13 tiles (2912 x 2912)

plasma_dpotrf(uplo, n, pA, lda);
plasma_dlauum(uplo, n, pA, lda);
plasma_dtrtri(uplo, diag, n, pA, lda);
#pragma omp parallel
#pragma omp master
{
    plasma_omp_zge2desc(pA, lda, A, sequence, &request);

    plasma_omp_dpotrf(uplo, A, sequence, &request);
    plasma_omp_zlauum(uplo, A, sequence, &request);
    plasma_omp_ztrtri(uplo, diag, A, sequence, &request);

    plasma_omp_zdesc2ge(A, pA, lda, sequence, &request);
}
PLASMA OpenMP Cholesky Inversion DAG

PLASMA Cholesky inversion using OpenMP
Intel Xeon E5-2650 v3 (Haswell) 2.3GHz 20 cores
tiles of size 224 x 224, matrix of size 13 x 13 tiles (2912 x 2912)
Define standard API for batched BLAS and LAPACK in collaboration with Intel/Nvidia/other users

• Fixed size: most of BLAS and LAPACK released
• Variable size: most of BLAS released
• Variable size: LAPACK in the branch
• Native GPU algorithms (Cholesky, LU, QR) in the branch
• Tiled algorithm using batched routines on tile or LAPACK data layout in the branch

• Framework for Deep Neural Network kernels
• CPU, KNL and GPU routines
• FP16 routines in progress
**Batched Computations**

- **Non-batched computation**
- **loop over the matrices one by one** and compute using multithread (note that, since matrices are of small sizes there is not enough work for all the cores). So we expect low performance as well as threads contention might also affect the performance

```c
for (i=0; i<batchcount; i++)
    dgemm(...)
```
Batched Computations

- **Batched computation**
- Distribute all the matrices over the available resources by assigning a matrix to each group of core/TB to operate on it independently
  - For very small matrices, assign a matrix/core (CPU) or per TB for GPU
  - For medium size a matrix go to a team of cores (CPU) or many TB’s (GPU)
  - For large size switch to multithreads classical 1 matrix per round.

```
Batched_dgemm(...)
```

Based on the kernel design that decide the number of TB or threads (GPU/CPU) and through the Nvidia/OpenMP scheduler.

High percentage of the resources is used.
Accelerators to Enhance Performance
We Have Seen This Before

- Floating Point Systems FPS-164/MAX Supercomputer (1976)
- Intel Math Co-processor (1980)
- Weitek Math Co-processor (1981)
Today Many HPC Systems …

- Use a hybrid architecture design
  - Think standard multicore chips and accelerators (GPUs)
- Successive generations become more integrated
- AMD’s Radeon Instinct Mi100 GPU
- Nvidia’s Ampere GPU
- Intel’s Xe Ponte Vecchio GPU

<table>
<thead>
<tr>
<th>Data Center GPU Name</th>
<th>NVIDIA Tesla V100</th>
<th>NVIDIA A100</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU Codename</td>
<td>GV100</td>
<td>GA100</td>
</tr>
<tr>
<td>GPU Architecture</td>
<td>NVIDIA Volta</td>
<td>NVIDIA Ampere</td>
</tr>
<tr>
<td>SMs</td>
<td>80</td>
<td>108</td>
</tr>
<tr>
<td>GPU Boost Clock</td>
<td>1530 MHz</td>
<td>1410 MHz</td>
</tr>
<tr>
<td>Peak FP16 Tensor Core TFLOPS$^1$</td>
<td>125</td>
<td>312</td>
</tr>
<tr>
<td>Peak Bfloat16 Tensor Core TFLOPS$^1$</td>
<td>NA</td>
<td>312</td>
</tr>
<tr>
<td>Peak TF32 Tensor TFLOPS$^1$</td>
<td>NA</td>
<td>156</td>
</tr>
<tr>
<td>Peak FP64 Tensor TFLOPS$^1$</td>
<td>NA</td>
<td>19.5</td>
</tr>
<tr>
<td>Peak INT8 Tensor TOPS$^1$</td>
<td>NA</td>
<td>624</td>
</tr>
<tr>
<td>Peak FP16 TFLOPS$^1$</td>
<td>31.4</td>
<td>78</td>
</tr>
<tr>
<td>Peak Bfloat16 TFLOPS$^1$</td>
<td>NA</td>
<td>39</td>
</tr>
<tr>
<td>Peak FP32 TFLOPS$^1$</td>
<td>15.7</td>
<td>19.5</td>
</tr>
<tr>
<td>Peak FP64 TFLOPS$^1$</td>
<td>7.8</td>
<td>9.7</td>
</tr>
<tr>
<td>Peak INT8 TOPS$^1$</td>
<td>15.7</td>
<td>19.5</td>
</tr>
<tr>
<td>Memory Interface</td>
<td>4096-bit HBM2</td>
<td>5120-bit HBM2</td>
</tr>
<tr>
<td>Memory Size</td>
<td>32 GB / 16 GB</td>
<td>40 GB</td>
</tr>
<tr>
<td>Memory Data Rate</td>
<td>877.5 MHz DDR</td>
<td>1215 MHz DDR</td>
</tr>
<tr>
<td>Memory Bandwidth</td>
<td>900 GB/sec</td>
<td>1.6 TB/sec</td>
</tr>
<tr>
<td>L2 Cache Size</td>
<td>6144 KB</td>
<td>40960 KB</td>
</tr>
<tr>
<td>Shared Memory Size / SM</td>
<td>Configurable up to 96 KB</td>
<td>Configurable up to 164 KB</td>
</tr>
</tbody>
</table>

1. Peak rates are based on GPU Boost Clock
MAGMA
Provides highly optimized LA for GPUs
Designed for single node with multiple GPUs
Research vehicle for LA on new architectures

for architectures in
{ CPUs + Nvidia GPUs (CUDA),
  CPUs + AMD GPUs (HIP & OpenCL),
  CPUs + Intel Xeon Phi,
  manycore (native: GPU or KNL/CPU),
  embedded systems, combinations }

for precisions in
{ s, d, c, z,
  half-precision (FP16),
  mixed, … } } ”

for interfaces
{ heterogeneous CPU/GPU, native, … } } ”

• LAPACK
• BLAS
• Batched LAPACK
• Batched BLAS
• Sparse
• Tensors
• MAGMA-DNN
• Templates
• …

How to design for performance and energy efficiency
Programming model: BLAS tasks + scheduling

Execution trace with hybrid task scheduling
MAGMA on Nvidia GPUs

PERFORMANCE & ENERGY EFFICIENCY

MAGMA 2.5.3 LU factorization in double precision arithmetic

<table>
<thead>
<tr>
<th>GPU Type</th>
<th>Architecture</th>
<th>Clock Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel Xeon E5-2650 v3 (Haswell)</td>
<td>2x10 cores @ 2.30 GHz</td>
</tr>
<tr>
<td>K40</td>
<td>NVIDIA Kepler GPU</td>
<td>15 MP x 192 @ 0.88 GHz</td>
</tr>
<tr>
<td>P100</td>
<td>NVIDIA Pascal GPU</td>
<td>56 MP x 64 @ 1.19 GHz</td>
</tr>
<tr>
<td>V100</td>
<td>NVIDIA Volta GPU</td>
<td>80 MP x 64 @ 1.38 GHz</td>
</tr>
<tr>
<td>A100</td>
<td>NVIDIA Ampere GPU</td>
<td>108 MP x 64 @ 1.41 GHz</td>
</tr>
</tbody>
</table>

Energy efficiency (under ~ the same power draw)

Matrix size N x N

Performance GFLOP/s

Energy efficiency

GFLOPs / Watt
Scheduling of computational tasks

- Main scheduling mechanism in MAGMA is data flow driven using streams
- MAGMA research has explored use of OpenMP scheduling (similar to approach in PLASMA)


Other uses of OpenMP in MAGMA

- Batched LA on CPUs uses OpenMP
- Divide & Conquer for Hermitian or real symmetric matrices use OpenMP (in a hybrid algorithm that runs Divide & Conquer on the CPU)
- MAGMA Sparse uses OpenMP in incomplete LU; data-format preparations, transformations, and initializations, etc.
• As part of DOE’s ECP we are working on a package to fit within the architectures for Exascale systems.

• The research within LAPACK, ScaLAPACK, PLASMA and MAGMA will go into a new package called SLATE
  • Software for Linear Algebra Targeting Exascale
Software for Linear Algebra Targeting Exascale (SLATE)
Focused on Dense Linear Algebra Problems

- Linear systems of equations \( Ax = b \)
- Linear least squares \( \min || b - Ax ||_2 \)
- Singular value decomposition (SVD) \( A = U\Sigma V^T \)
- Eigenvalue value problems (EVP) \( Ax = \lambda x \)

- Dense (square, rectangular)
- Band
**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

- **Flexibility**
  - Users can construct new routines from well-designed parts
SLATE design

• Modern C++ replacement for ScaLAPACK
  ▪ Code templated for precision
  ▪ Backwards compatibility layer
• Flexible
  ▪ Non-uniform block sizes
  ▪ Arbitrary distributions; default 2D block-cyclic
• Standards based
  ▪ MPI for distributed communication
  ▪ OpenMP 4.5 tasks for shared memory parallelism
  ▪ Includes GPU support, currently using cuBLAS
  ▪ S,D,C,Z,H (float16) precisions
• Developed from scratch as ECP project
• LAPACK/ScaLAPACK calling sequences mapping to SLATE
Leverage emerging programming frameworks for scheduling tasks to large scale machines with multicores, accelerators and complex memory systems. Perhaps plug into different run-time systems

- Runtime provides ...
  - Dynamic task scheduling
    - Mutithreading
    - Accelerator offload
  - Accelerator memory management
    - Basically a cache model with LRU policy
  - Communication hiding
    - Asynchronous message passing
    - Asynchronous PCI DMAs (host-device)
- Investigating PaRSEC (UTK), StarPU (INRIA), Kokkos (SNL), Legion (Stanford),...
## Coverage

### Basic linear algebra \( (C = AB, \ldots) \)

<table>
<thead>
<tr>
<th>Feature</th>
<th>ScaLAPACK</th>
<th>SLATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1 PBLAS</td>
<td>✓</td>
<td>✗ (use Level 3)</td>
</tr>
<tr>
<td>Level 2 PBLAS</td>
<td>✓</td>
<td>✗ (use Level 3)</td>
</tr>
<tr>
<td>Level 3 PBLAS</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Matrix norms</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Test matrix generation</td>
<td>✓</td>
<td>✓ (new)</td>
</tr>
</tbody>
</table>

### Least squares \( (Ax \equiv b) \)

<table>
<thead>
<tr>
<th>Feature</th>
<th>ScaLAPACK</th>
<th>SLATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>QR</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LQ</td>
<td>✓</td>
<td>✓ (new)</td>
</tr>
<tr>
<td>Least squares solver</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

### Linear systems \( (Ax = b) \)

<table>
<thead>
<tr>
<th>Feature</th>
<th>ScaLAPACK</th>
<th>SLATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>LU (partial pivoting)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LU, band (pp)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LU (non-pivoting)</td>
<td>✗</td>
<td>✓ (new)</td>
</tr>
<tr>
<td>Cholesky</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Cholesky, band</td>
<td>✓</td>
<td>✓ (new)</td>
</tr>
<tr>
<td>Symmetric Indefinite (block Aasen)</td>
<td>✓</td>
<td>✓ (CPU only)</td>
</tr>
<tr>
<td>Mixed precision (single-double)</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Inverses (LU, Cholesky)</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Condition estimate</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

### SVD, eigenvalues \( (A = UΣV^H, Ax = λx) \)

<table>
<thead>
<tr>
<th>Feature</th>
<th>ScaLAPACK</th>
<th>SLATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Singular value decomposition (SVD)</td>
<td>✓</td>
<td>✓ values (new)</td>
</tr>
<tr>
<td>Symmetric eigenvalues</td>
<td>✓</td>
<td>✓ values (new)</td>
</tr>
<tr>
<td>Generalized symmetric eigenvalues</td>
<td>✓</td>
<td>✓ values (new)</td>
</tr>
<tr>
<td>Polar decomposition (QDWH)</td>
<td>✗</td>
<td>✓ (new)</td>
</tr>
<tr>
<td>Non-symmetric eigenvalues</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Hessenberg reduction</td>
<td>✓</td>
<td>2021</td>
</tr>
<tr>
<td>Hessenberg eigen solver</td>
<td>• real only</td>
<td>2022</td>
</tr>
<tr>
<td>Back-transform</td>
<td>• complex only</td>
<td>2021</td>
</tr>
</tbody>
</table>
Milestones

• **Completed**
  - Hermitian eigenvalues & SVD — 2-stage reductions (“bulge chasing”)
  - Performance improvements (BLAS, norms, Cholesky, QR)
  - Developers’ Guide
  - Generalized Hermitian eigenvalues
  - Simplified C++ API (lu_factor instead of getrf)

• **Upcoming**
  - Performance improvements for LU, Cholesky
  - Port to AMD (HIP) and Intel (oneAPI, OpenMP offload)
  - Performance improvements for QR, eigenvalues, SVD
  - Divide-and-conquer for eigenvalues
Tasks and dependencies

• PLASMA tile-by-tile data flow
  • $O(n^3)$ tasks and dependencies

• SLATE aggregates tiles into large tasks
  • $O(n)$ tasks and dependencies
CPU and GPU Targets

• SLATE algorithms templated for target: CPU Host or GPU Devices
  • One high-level Cholesky code can call different low-level kernels (CPU or GPU)
• Today, user can specify target
• In future, default will be GPU Devices if available, else CPU Host, perhaps based on matrix size

```c
// Default on GPU, if available, else CPU.
slate::chol_factor( A );

// User-specified target.
slate::chol_factor( A, {{ Option::Target, Target::Devices }} );
slate::chol_factor( A, {{ Option::Target, Target::Host    }} );
```
GPU support

- Currently, SLATE directly uses CUDA and cuBLAS
- Plan to add portability layer
  - Support HIP/ROCm, OpenMP offload, or SYCL
  - Primarily rely on vendor BLAS (cuBLAS, hipBLAS, MKL, ...)
    - BLAS++ library as portability layer
  - SLATE has few custom kernels to implement in CUDA / HIP / OpenMP offload / SYCL
    - Batched transpose, batched matrix norm, ...
SLATE Features

- **Runtime interface**
  - Use OpenMP
  - Would like to plug into other systems
    - PaRSEC, Legion, Darma, StarPU, ...
    - Statically scheduled across nodes; dynamically schedule within node

- **Tiled Algorithms**
  - Runtime scheduling based on dataflow
  - Runtime dependency tracking
    - Plug into the different runtime systems

- **Data distribution as in ScaLAPACK**
  - Given the layout and arrangement of processes communication is understood

- **Task based parallelism inspired by PLASMA**
  - High level DAG enables overlap of computation and communication

- **Ability to use accelerators as in MAGMA**
  - Hybrid computing using the runtime system
Conclusions

- Many changes in the past 50 years...
  - Hardware, Languages, Standards, Algorithms, and Applications

- Standards (both defacto and official) and licensing are important in wide spread adoption of libraries.

- As numerical library developers we have tracked the advances and have taken advantage of these changes to enhance the software base.