Outline

Availability

Routines

Code

Testers

Methodology

MAGMA 2.3

Availability

- http://icl.utk.edu/magma/ download, documentation, forum
- https://bitbucket.org/icl/magma Mercurial repo

Support

- Linux, macOS, Windows
- CUDA >= 5.0; recommend latest CUDA
- CUDA architecture >= 2.0 (Fermi, Kepler, Maxwell, Pascal, Volta)
- BLAS & LAPACK: Intel MKL, OpenBLAS, macOS Accelerate, ...

May be pre-installed on supercomputers

```
titan-ext1> module avail magma
----------- /sw/xk6/modulefiles -----------
magma/1.3                magma/1.6.2(default)
```

Installation options

1. **Makefile**
   - Edit make.inc for compilers and flags (see make.inc examples)
   - `magma>` **`make && make install`**
2. **CMake**
   - `magma>` **`mkdir build && cd build`**
   - `magma/build>` **`cmake ..`** or **`ccmake ..`**
   - Adjust settings, esp. LAPACK_LIBRARIES and GPU_TARGET
   - `magma/build>` **`make && make install`**
3. **Spack**
   - Part of xSDK
   - **`spack install magma`**
   - Caveat: nvcc is picky about its host compiler (gcc, ...)

Outline

Availability

Routines

Code

Testers

Methodology

MAGMA Overview

Hybrid LAPACK-style functions

- Matrix factorizations: LU, Cholesky, QR, eigenvalue, SVD, ...
- Solve linear systems and linear least squares, ...
- Nearly all are synchronous:
  return on CPU when computation is finished

GPU BLAS and auxiliary functions

- Matrix-vector multiply, matrix norms, transpose (in-place and out-of-place), ...
- Most are asynchronous:
  return immediately on CPU; computation proceeds on GPU

Wrappers around CUDA and cuBLAS

- BLAS routines (gemm, symm, symv, ...)
- Copy host ⇔ device, queue (stream) support, GPU malloc & free, ...

Naming example

magma_ or **magmablas_** prefix

magma_zgesv_gpu

Precision (1–2 characters)

- **S**ingle, **D**ouble, single **C**omplex, "**Z**" double complex, **I**nteger
  Mixed precision (**DS** and **ZC**)

Matrix type (2 characters)

- **GE**neral **SY**mmetric **HE**rmetian **PO**sitive definite
  ORthogonal **UN**itary **TR**iangular

Operation (2–3+ characters)

- **SV** solve
  TRF triangular factorization
  EV eigenvalue problem
  GV generalized eigenvalue problem
  etc.

_gpu suffix for interface

Naming example

magma_ or **magmablas_** prefix

magma_zgesv_gpu

Precision (1–2 characters)

- **S**ingle, **D**ouble, single **C**omplex, "**Z**" double complex, **I**nteger
  Mixed precision (**DS** and **ZC**)

Matrix type (2 characters)

- **GE**neral **SY**mmetric **HE**rmetian **PO**sitive definite
  ORthogonal **UN**itary **TR**iangular

Operation (2–3+ characters)

- **SV** solve
  TRF triangular factorization
  EV eigenvalue problem
  GV generalized eigenvalue problem
  etc.

_gpu suffix for interface

Naming example

magma_ or **magmablas_** prefix

Precision (1–2 characters)

- **S**ingle, **D**ouble, single **C**omplex, "**Z**" double complex, **I**nteger
  Mixed precision (**DS** and **ZC**)

Matrix type (2 characters)

- **GE**neral **SY**mmetric **HE**rmetian **PO**sitive definite
  ORthogonal **UN**itary **TR**iangular

Operation (2–3+ characters)

- **SV** solve
  TRF triangular factorization
  EV eigenvalue problem
  GV generalized eigenvalue problem
  etc.

_gpu suffix for interface

magma_zgesv_gpu

Naming example

magma_ or **magmablas_** prefix

magma_zgesv_gpu

Precision (1–2 characters)

- **S**ingle, **D**ouble, single **C**omplex, "**Z**" double complex, **I**nteger
  Mixed precision (**DS** and **ZC**)

Matrix type (2 characters)

- **GE**neral **SY**mmetric **HE**rmetian **PO**sitive definite
  ORthogonal **UN**itary **TR**iangular

Operation (2–3+ characters)

- **SV** solve
  TRF triangular factorization
  EV eigenvalue problem
  GV generalized eigenvalue problem
  etc.

_gpu suffix for interface

Naming example

magma_ or **magmablas_** prefix

magma_zgesv_gpu

Precision (1–2 characters)

- **S**ingle, **D**ouble, single **C**omplex, "**Z**" double complex, **I**nteger
  Mixed precision (**DS** and **ZC**)

Matrix type (2 characters)

- **GE**neral **SY**mmetric **HE**rmetian **PO**sitive definite
  ORthogonal **UN**itary **TR**iangular

Operation (2–3+ characters)

- **SV** solve
  TRF triangular factorization
  EV eigenvalue problem
  GV generalized eigenvalue problem
  etc.

_gpu suffix for interface

Naming example

magma_ or **magmablas_** prefix

magma_zgesv_gpu

Precision (1–2 characters)

- **S**ingle, **D**ouble, single **C**omplex, "**Z**" double complex, **I**nteger
  Mixed precision (**DS** and **ZC**)

Matrix type (2 characters)

- **GE**neral **SY**mmetric **HE**rmetian **PO**sitive definite
  ORthogonal **UN**itary **TR**iangular

Operation (2–3+ characters)

- **SV** solve
  TRF triangular factorization
  EV eigenvalue problem
  GV generalized eigenvalue problem
  etc.

_gpu suffix for interface

Linear solvers

Solve linear system: $AX = B$

Solve linear least squares: **minimize** $\|AX - B\|_2$

Type	Routine	Mixed precision routine	Interface CPU	Interface GPU
General	dgesv	dsgesv	✓	✓
Positive definite	dposv	dsposv	✓	✓
Symmetric	dsysv		✓	
Hermitian	zhesv		✓	
Least squares	dgels		✓	✓

Selected routines; complete documentation at http://icl.utk.edu/magma/

Eigenvalue / singular value problems

Eigenvalue problem: $Ax = \lambda x$

Generalized eigenvalue problem: $Ax = \lambda Bx$ (and variants)

Singular value decomposition: $A = U\Sigma V^H$

Matrix type	Operation	Routine	Interface CPU	Interface GPU
General	SVD	dgesvd, dgesdd	✓	
General non-symmetric	Eigenvalue	dgeev	✓	
Symmetric	Eigenvalue	dsyevd / zheevd	✓	✓
Symmetric	Generalized	dsygvd / zhegvd	✓	

Additional variants; complete documentation at http://icl.utk.edu/magma/
Fastest are divide-and-conquer (gesdd, syevd) and 2-stage versions.

Computational routines

Computational routines solve one part of problem

Matrix type	Operation	Routine	Interface CPU	Interface GPU
General	LU	dgetrf	✓	✓
	Solve (given LU)	dgetrs		✓
	Inverse	dgetri		✓
SPD	Cholesky	dpotrf	✓	✓
	Solve (given LL^T)	dpotrs		✓
	Inverse	dpotri	✓	✓
General	QR	dgeqrf	✓	✓
	Generate Q	dorgqr / zungqr	✓	✓
	Multiply by Q	dormqr / zunmqr	✓	✓

Selected routines; complete documentation at http://icl.utk.edu/magma/

BLAS and auxiliary routines

Category	**Operation**	**Routine** (all GPU interface)
Level 1 BLAS	$y = \alpha x + y$	daxpy
	$r = x^{\mathrm{T}} y$	ddot
Level 2 BLAS	$y = \alpha Ax + \beta y$, general A	dgemv
	$y = \alpha Ax + \beta y$, symmetric A	dsymv
Level 3 BLAS	$C = \alpha AB + \beta C$	dgemm
	$C = \alpha AB + \beta C$, symmetric A	dsymm
	$C = \alpha AA^T + \beta C$, symmetric C	dsyrk
Auxiliary	$\|A\|_1$, $\|A\|_{\mathrm{inf}}$, $\|A\|_{\mathrm{fro}}$, $\|A\|_{\max}$	dlange (norm, general A)
		dlansy (norm, symmetric A)
	$B = A^{\mathrm{T}}$ (out-of-place)	dtranspose
	$A = A^{\mathrm{T}}$ (in-place, square)	dtranspose_inplace

Selected routines; complete documentation at http://icl.utk.edu/magma/

Outline

Availability

Routines

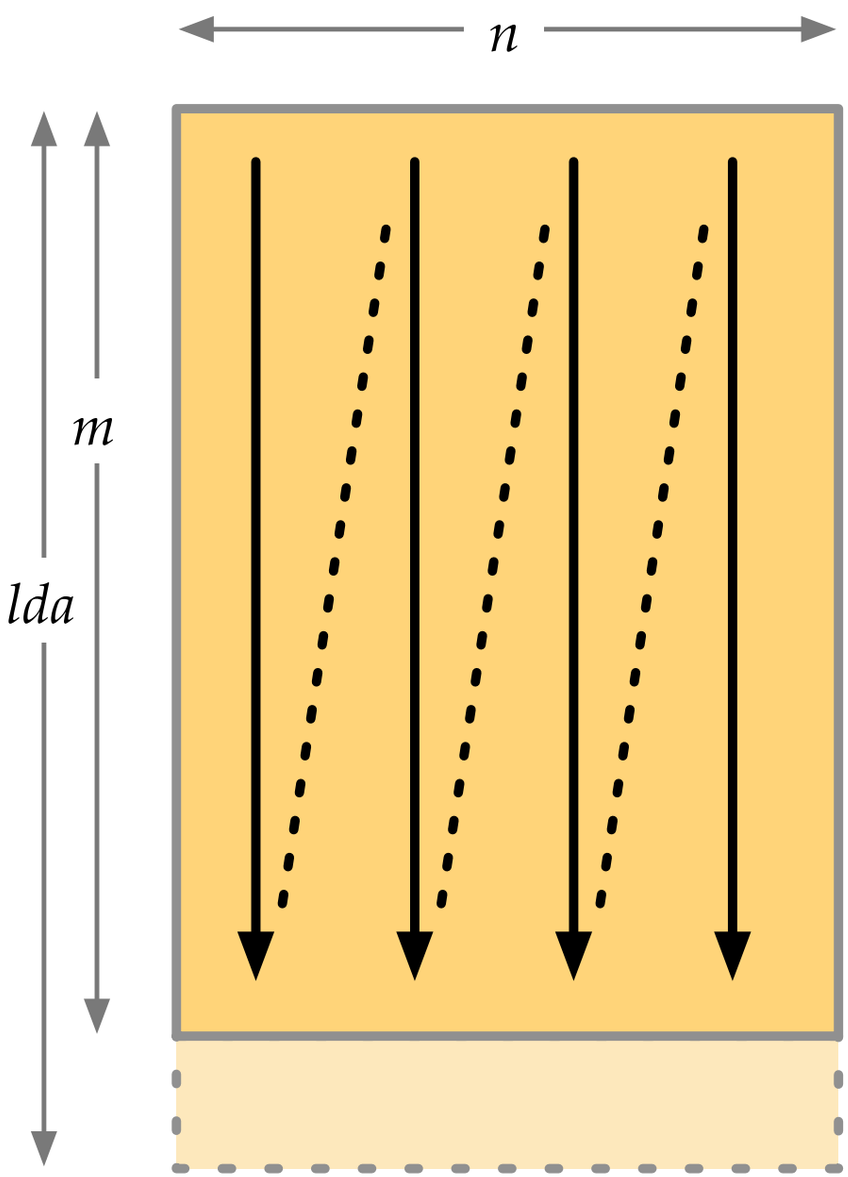
Code

Testers

Methodology

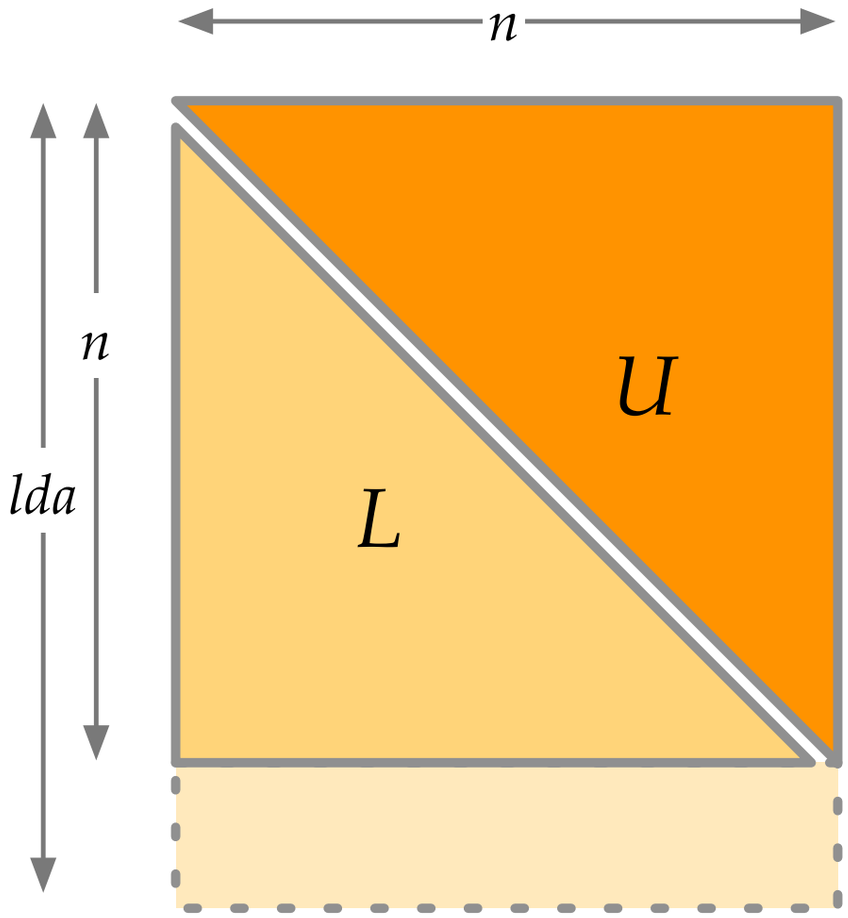
Matrix layout

General m-by-n matrix, LAPACK column-major layout



Symmetric / Hermitian / Triangular n-by-n matrix

- uplo = Lower or Upper
- Entries in opposite triangle ignored



Simple example

Solve $AX = B$

- **D**ouble precision, **GE**neral matrix, **S**ol**V**e (**DGESV**)

Traditional LAPACK call

Complete codes available at bit.ly/magma-tutorial

```
// tutorial0_lapack.cc
#include "magma_lapack.h"

int main( int argc, char** argv )
{
    int n = 100, nrhs = 10;
    int lda = n, ldx = n;
    double *A = new double[ lda*n ];
    double *X = new double[ ldx*nrhs ];
    int* ipiv = new int[ n ];

    // ... fill in A and X with your data
    // A[ i + j*lda ] = A_ij
    // X[ i + j*ldx ] = X_ij

    // solve AX = B where B is in X
    int info;
    lapackf77_dgesv( &n, &nrhs,
                     A, &lda, ipiv,
                     X, &ldx, &info );
    if (info != 0) {
        throw std::exception();
    }

    // ... use result in X

    delete[] A;
    delete[] X;
    delete[] ipiv;
}
```

Simple example

MAGMA CPU interface

- Input & output matrices in CPU host memory

Add MAGMA init & finalize

MAGMA call direct replacement for LAPACK call

```
// tutorial1_cpu_interface.cc
#include "magma_v2.h"

int main( int argc, char** argv )
{
    magma_init();

    int n = 100, nrhs = 10;
    int lda = n, ldx = n;
    double *A = new double[ lda*n ];
    double *X = new double[ ldx*nrhs ];
    int* ipiv = new int[ n ];

    // ... fill in A and X with your data
    // A[ i + j*lda ] = A_ij
    // X[ i + j*ldx ] = X_ij

    // solve AX = B where B is in X
    int info;
    magma_dgesv( n, nrhs,
                 A, lda, ipiv,
                 X, ldx, &info );
    if (info != 0) {
        throw std::exception();
    }

    // ... use result in X

    delete[] A;
    delete[] X;
    delete[] ipiv;

    magma_finalize();
}
```

Simple example

MAGMA GPU interface

- Add **_gpu** suffix
- Input & output matrices in GPU device memory (“d” prefix on variables)
- ipiv still in CPU memory
- Set GPU stride (ldda) to multiple of 32 for better performance
- roundup returns ceil(n / 32) * 32

MAGMA malloc & free

- Type-safe wrappers around cudaMalloc & cudaFree

```
// tutorial2_gpu_interface.cc
int main( int argc, char** argv )
{
    magma_init();

    int n = 100, nrhs = 10;
    int ldda = magma_roundup( n, 32 );
    int lddx = magma_roundup( n, 32 );
    int* ipiv = new int[ n ];

    double *dA, *dX;
    magma_dmalloc( &dA, ldda*n );
    magma_dmalloc( &dX, lddx*nrhs );
    assert( dA != nullptr );
    assert( dX != nullptr );

    // ... fill in dA and dX (on GPU)

    // solve AX = B where B is in X
    int info;
    magma_dgesv_gpu( n, nrhs,
                     dA, ldda, ipiv,
                     dX, lddx, &info );
    if (info != 0) {
        throw std::exception();
    }

    // ... use result in dX

    magma_free( dA );
    magma_free( dX );
    delete[] ipiv;

    magma_finalize();
}
```

BLAS example

Matrix multiply $C = -AB + C$

- **D**ouble-precision, **GE**neral **M**atrix **M**ultiply (**DGEMM**)

Asynchronous

- BLAS take queue and are async
- Return to CPU immediately
- Queue wraps CUDA stream and cuBLAS handle
- Can create queue from existing CUDA stream and cuBLAS handle, if required

```
// tutorial3_blas.cc
int main( int argc, char** argv )
{
    // ... setup matrices on GPU:
    // m-by-k matrix dA,
    // k-by-n matrix dB,
    // m-by-n matrix dC.

    int device;
    magma_queue_t queue;
    magma_getdevice( &device );
    magma_queue_create( device, &queue );

    // C = -A B + C
    magma_dgemm( MagmaNoTrans,
                 MagmaNoTrans, m, n, k,
                 -1.0, dA, ldda,
                       dB, lddb,
                  1.0, dC, lddc, queue );

    // ... do concurrent work on CPU

    // wait for gemm to finish
    magma_queue_sync( queue );

    // ... use result in dC

    magma_queue_destroy( queue );

    // ... cleanup
}
```

Copy example

Copy data host ⇔ device

- setmatrix (host to device)
- getmatrix (device to host)
- copymatrix (device to device)
- setvector (host to device)
- getvector (device to host)
- copyvector (device to device)

Default is synchronous

- Return when transfer is done

Strides (lda, ldda) can differ on CPU and GPU

- Set GPU stride (ldda) to multiple of 32 for better performance

```
// tutorial4_copy.cc
int main( int argc, char** argv )
{
    // ... setup A, X in CPU memory;
    // dA, dX in GPU device memory

    int device;
    magma_queue_t queue;
    magma_getdevice( &device );
    magma_queue_create( device, &queue );

    // copy A, X to dA, dX
    magma_dsetmatrix( n, n,
                      A, lda,
                     dA, ldda, queue );
    magma_dsetmatrix( n, nrhs,
                      X, ldx,
                     dX, lddx, queue );

    // ... solve AX = B

    // copy result dX to X
    magma_dgetmatrix( n, nrhs,
                     dX, lddx,
                      X, ldx, queue );

    // ... use result in X

    magma_queue_destroy( queue );

    // ... cleanup
}
```

Async copy

Add _async suffix

Use pinned CPU memory

- Page locked, so DMA can access it
- Better performance
- Required by CUDA for async behavior
- But pinned memory is limited resource, and expensive to allocate

Overlap:

- Sending data (host to device)
- Getting data (device to host)
- Host computation
- Device computation

```
// tutorial5_copy_async.cc
int main( int argc, char** argv )
{
    // ... setup dA, dX, queue

    // allocate A, X in pinned CPU memory
    double *A, *X;
    magma_dmalloc_pinned( &A, lda*n );
    magma_dmalloc_pinned( &X, ldx*nrhs );

    // ... fill in A and X

    // copy A, X to dA, dX, then wait
    magma_dsetmatrix_async( n, n,
            A, lda, dA, ldda, queue );
    magma_dsetmatrix_async( n, nrhs,
            X, ldx, dX, lddx, queue );
    magma_queue_sync( queue );

    // ... solve AX = B

    // copy result dX to X, then wait
    magma_dgetmatrix_async( n, nrhs,
            dX, ldx, X, lddx, queue );
    magma_queue_sync( queue );

    // ... use result in X

    magma_free_pinned( A );
    magma_free_pinned( X );

    // ... cleanup
}
```

Outline

Availability

Routines

Code

Testers

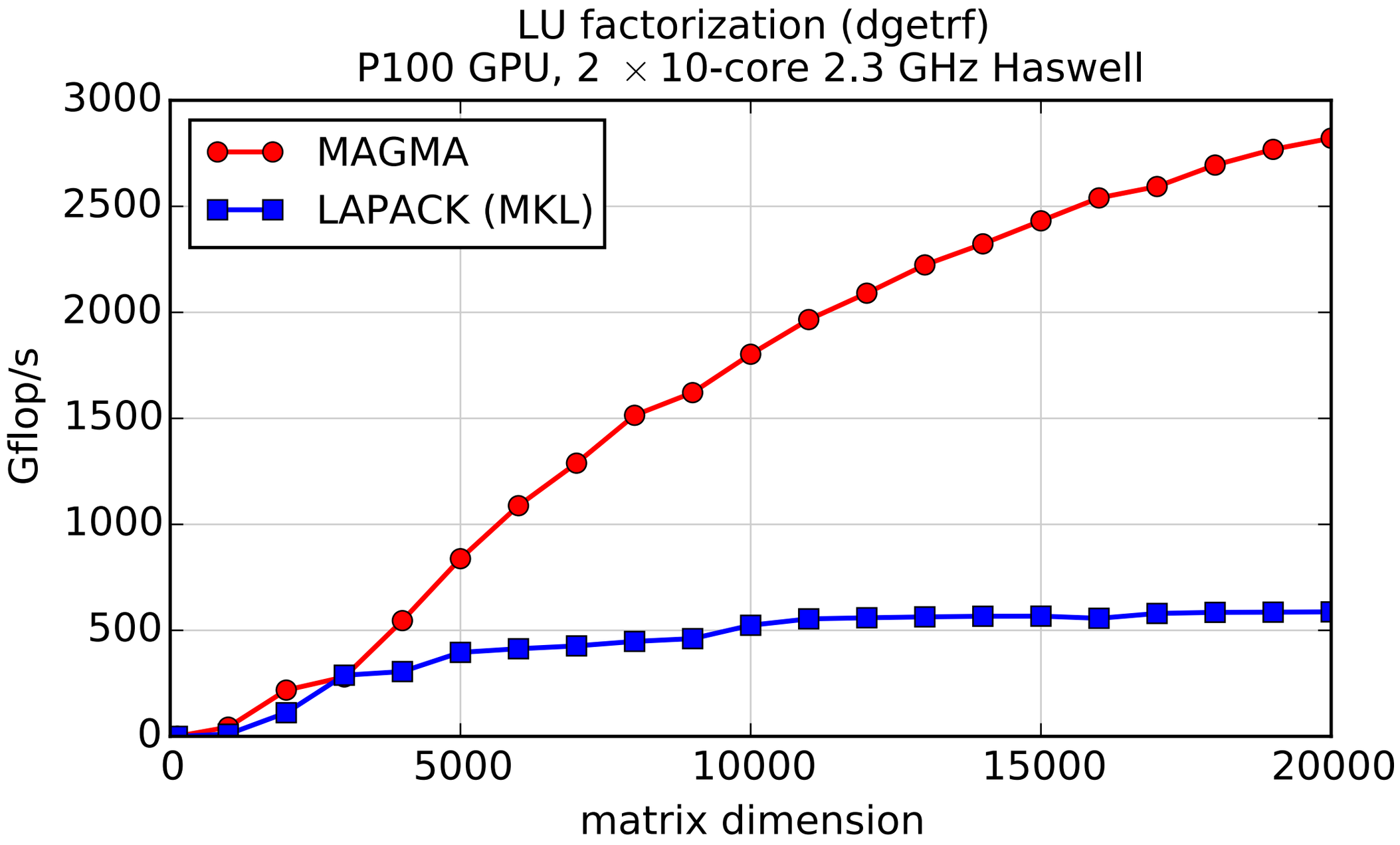
Methodology

Testers: LU factorization (dgetrf)

```
magm> cd testing
magma/testing> ./testing_dgetrf -n 123 -n 1000:20000:1000 --lapack --check
% MAGMA 2.2.0  compiled for CUDA capability >= 6.0, 32-bit magma_int_t, 64-bit pointer.
% CUDA runtime 8000, driver 9000. OpenMP threads 20. MKL 2017.0.1, MKL threads 20.
% device 0: Tesla P100-PCIE-16GB, 1328.5 MHz clock, 16276.2 MiB memory, capability 6.0

%   M     N   CPU Gflop/s (sec)   GPU Gflop/s (sec)   |PA-LU|/(N*|A|)
%=========================================================================
  123   123       0.20 (   0.01)      0.40 (   0.00)   3.59e-18   ok       # warmup run
 1000  1000      10.40 (   0.06)     43.50 (   0.02)   2.76e-18   ok
 2000  2000     111.64 (   0.05)    218.26 (   0.02)   2.68e-18   ok
 3000  3000     288.38 (   0.06)    280.28 (   0.06)   2.65e-18   ok
 4000  4000     305.58 (   0.14)    545.90 (   0.08)   2.81e-18   ok
 5000  5000     396.16 (   0.21)    838.09 (   0.10)   2.71e-18   ok
 6000  6000     413.37 (   0.35)   1088.14 (   0.13)   2.71e-18   ok
 7000  7000     426.71 (   0.54)   1288.60 (   0.18)   2.67e-18   ok
 8000  8000     447.85 (   0.76)   1514.43 (   0.23)   2.66e-18   ok
 9000  9000     461.05 (   1.05)   1621.29 (   0.30)   2.87e-18   ok
10000 10000     524.06 (   1.27)   1802.39 (   0.37)   2.84e-18   ok
11000 11000     554.16 (   1.60)   1965.85 (   0.45)   2.84e-18   ok
12000 12000     559.33 (   2.06)   2090.42 (   0.55)   2.82e-18   ok
13000 13000     563.56 (   2.60)   2223.62 (   0.66)   2.80e-18   ok
14000 14000     566.58 (   3.23)   2323.04 (   0.79)   2.78e-18   ok
15000 15000     567.17 (   3.97)   2431.59 (   0.93)   2.77e-18   ok
16000 16000     556.86 (   4.90)   2539.66 (   1.08)   2.79e-18   ok
17000 17000     579.82 (   5.65)   2593.40 (   1.26)   2.75e-18   ok
18000 18000     584.93 (   6.65)   2694.57 (   1.44)   2.76e-18   ok
19000 19000     585.78 (   7.81)   2768.67 (   1.65)   2.75e-18   ok
20000 20000     587.08 (   9.08)   2821.48 (   1.89)   2.74e-18   ok
```

Testers: LU factorization (dgetrf)

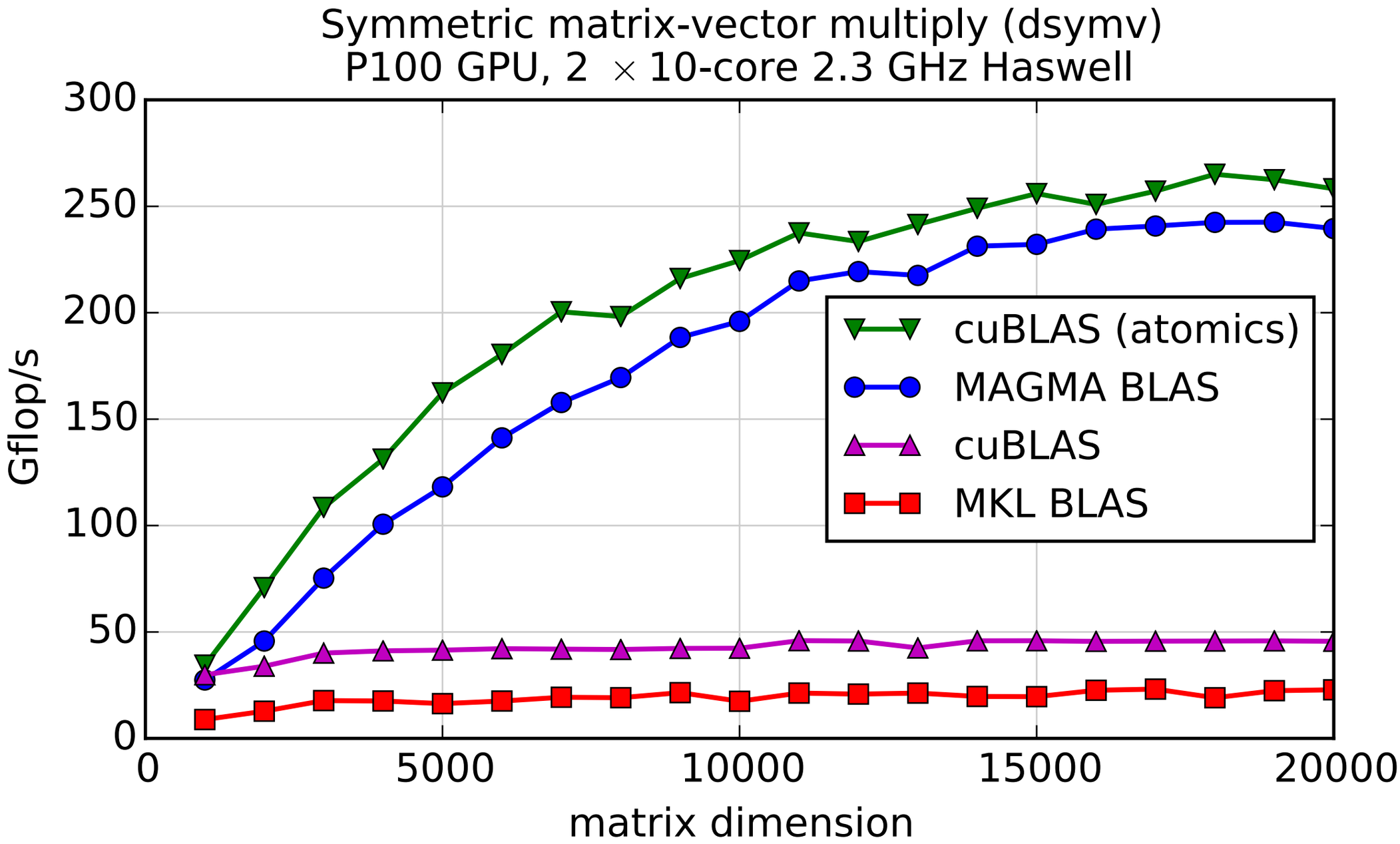


Testers: symmetric matrix-vector multiply (dsymv)

```
# (abbreviated output)
magma> cd testing
magma/testing> ./testing_dsymv -n 123 -n 1000:20000:1000 --lapack --check
% MAGMA 2.2.0  compiled for CUDA capability >= 6.0, 32-bit magma_int_t, 64-bit pointer.
% CUDA runtime 8000, driver 9000. OpenMP threads 20. MKL 2017.0.1, MKL threads 20.
% device 0: Tesla P100-PCIE-16GB, 1328.5 MHz clock, 16276.2 MiB memory, capability 6.0

% uplo = Lower
%   N     MAGMA   Atomics    cuBLAS       CPU   error
%       Gflop/s   Gflop/s   Gflop/s   Gflop/s
%=============================================================
  123      0.76      0.76      0.51      0.58   ok    # warmup run
 1000     27.44     34.41     29.88      8.86   ok
 2000     45.74     70.83     33.91     12.78   ok
 3000     75.30    108.51     40.09     17.76   ok
 4000    100.64    131.23     41.13     17.57   ok
 5000    118.17    162.35     41.46     16.33   ok
 6000    141.21    180.43     42.16     17.55   ok
 7000    157.81    200.44     41.94     19.32   ok
 8000    169.54    198.21     41.78     19.12   ok
 9000    188.40    216.07     42.28     21.50   ok
10000    195.92    224.50     42.36     17.44   ok
11000    214.93    237.51     45.91     21.30   ok
12000    219.33    233.44     45.76     20.81   ok
13000    217.52    241.45     42.49     21.29   ok
14000    231.26    249.06     45.84     19.71   ok
15000    232.12    255.98     45.87     19.60   ok
16000    239.26    250.89     45.58     22.61   ok
17000    240.74    257.13     45.69     23.15   ok
18000    242.45    265.05     45.75     19.09   ok
19000    242.53    262.48     45.81     22.42   ok
20000    239.53    258.24     45.63     22.83   ok
```

Testers: symmetric matrix-vector multiply (dsymv)



Test everything: run_tests.py

Python script to run:

- All testers
- All possible options (left/right, lower/upper, ...)
- Various size ranges (small, medium, large; square, tall, wide)

Occasionally, tests fail innocuously

- E.g., error = 1.1e-15 > tol = 1e-15

Some experimental routines are known to fail

- E.g., gegqr_gpu, geqr2x_gpu
- See magma/BUGS.txt

Running ALL tests can take > 24 hours

Test everything: run_tests.py

```
magma/testing> python ./run_tests.py *trsm --xsmall --small > trsm.txt
testing_strsm -SL -L -DN -c       ok  # left, lower, non-unit, [no-trans]
testing_dtrsm -SL -L -DN -c       ok
testing_ctrsm -SL -L -DN -c       ok
testing_ztrsm -SL -L -DN -c       ok
testing_strsm -SL -L -DU -c       ok  # left, lower, unit, [no-trans]
testing_dtrsm -SL -L -DU -c       ok
testing_ctrsm -SL -L -DU -c       ok
testing_ztrsm -SL -L -DU -c       ok
testing_strsm -SL -L -C -DN -c    ok  # left, lower, non-unit, conj-trans
testing_dtrsm -SL -L -C -DN -c    ok
testing_ctrsm -SL -L -C -DN -c    ok
testing_ztrsm -SL -L -C -DN -c    ok
...
testing_strsm -SR -U -T -DU -c    ok  # right, upper, unit, trans
testing_dtrsm -SR -U -T -DU -c    ok
testing_ctrsm -SR -U -T -DU -c    ok
testing_ztrsm -SR -U -T -DU -c    ok

****************************************************************************************************
summary
****************************************************************************************************
 6240 tests in 192 commands passed
   96 tests failed accuracy test
    0 errors detected (crashes, CUDA errors, etc.)
routines with failures:
    testing_ctrsm --ngpu 2 -SL -L -C -DN -c
    testing_ctrsm --ngpu 2 -SL -L -C -DU -c
...
```

Outline

Availability

Routines

Code

Testers

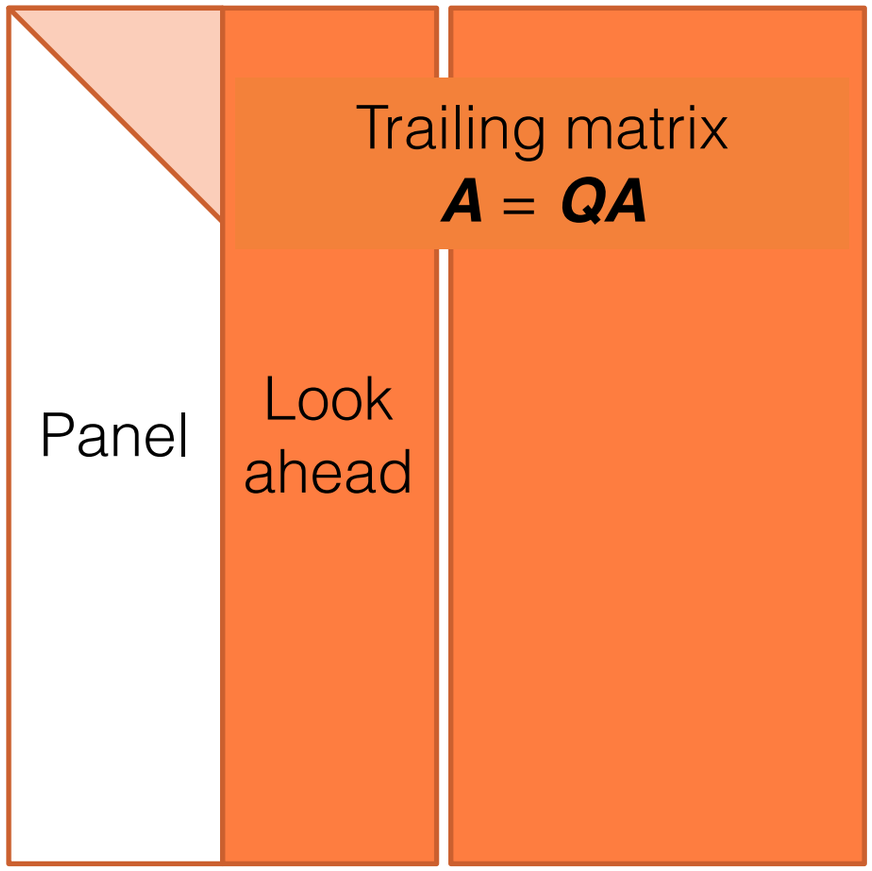
Methodology

One-sided factorizations

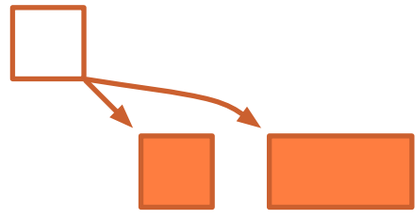
LU, Cholesky, QR factorizations for solving linear systems

Level 2 BLAS on CPU

Level 3 BLAS on GPU



DAG

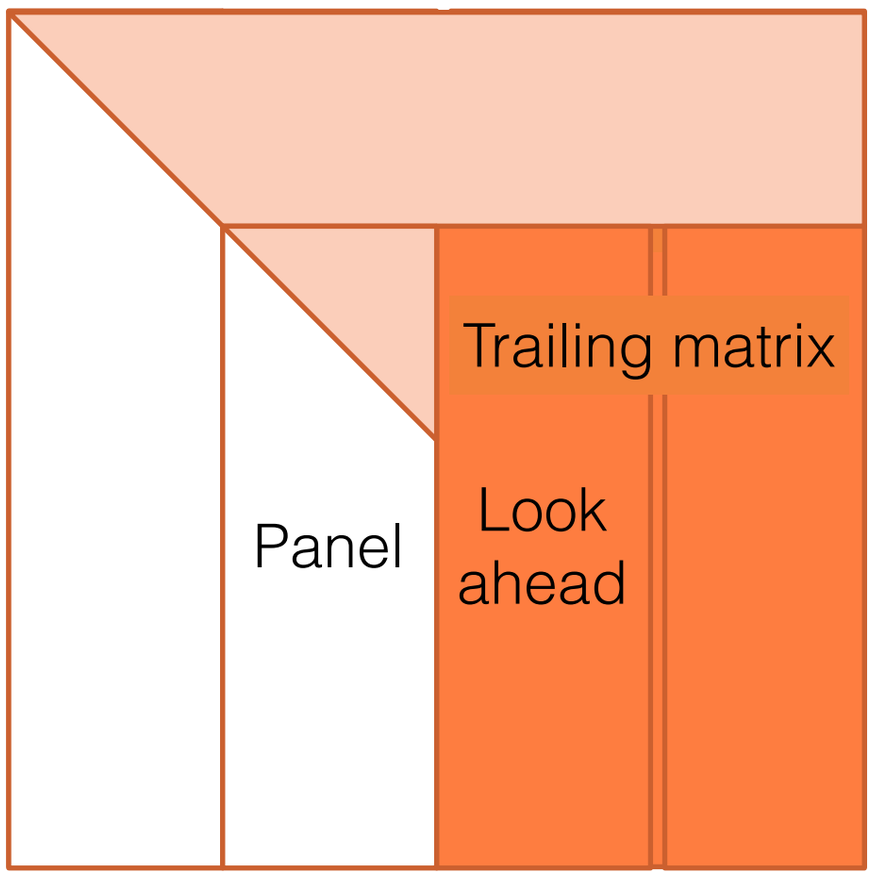


One-sided factorizations

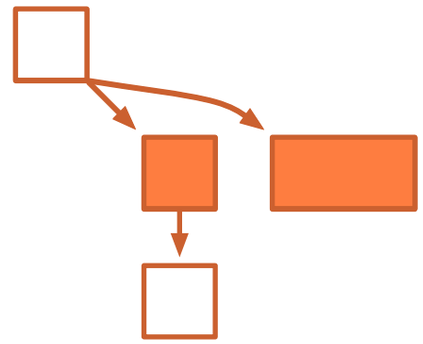
LU, Cholesky, QR factorizations for solving linear systems

Level 2 BLAS on CPU

Level 3 BLAS on GPU



DAG

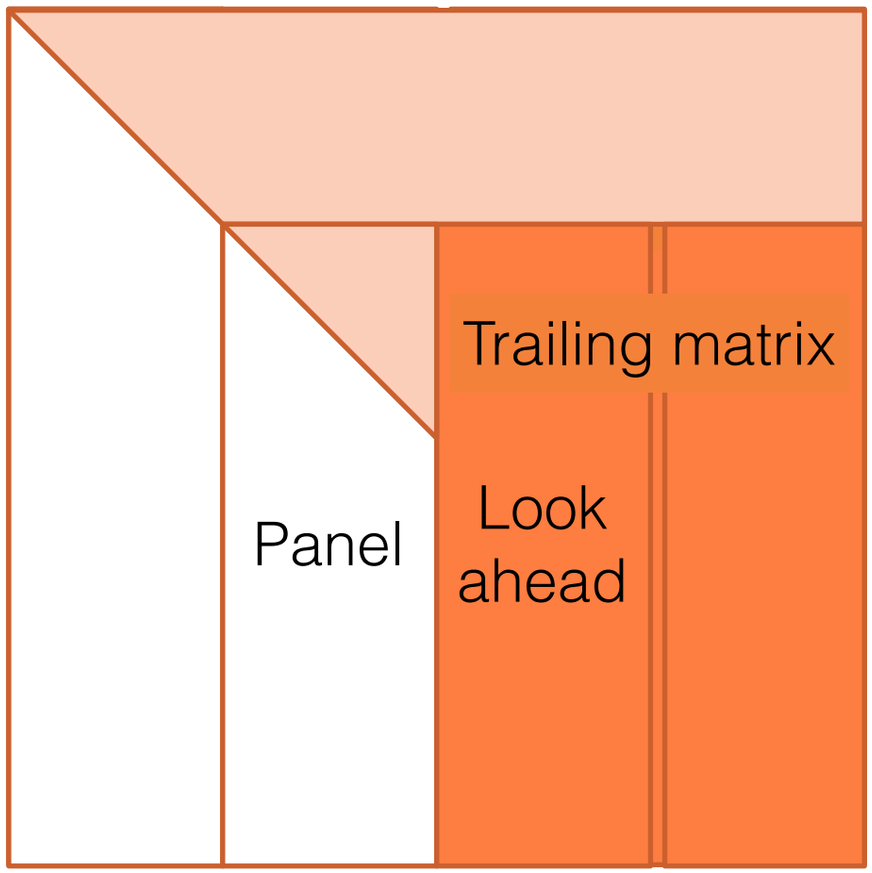


One-sided factorizations

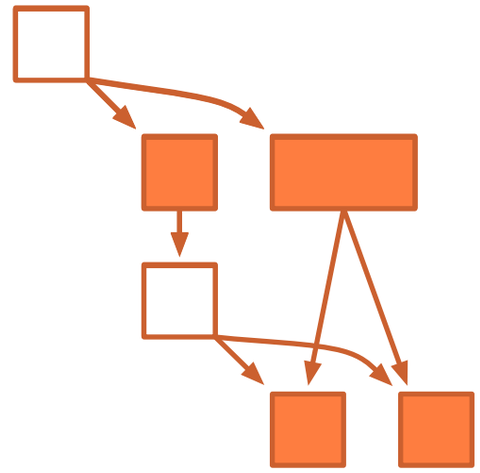
LU, Cholesky, QR factorizations for solving linear systems

Level 2 BLAS on CPU

Level 3 BLAS on GPU



DAG

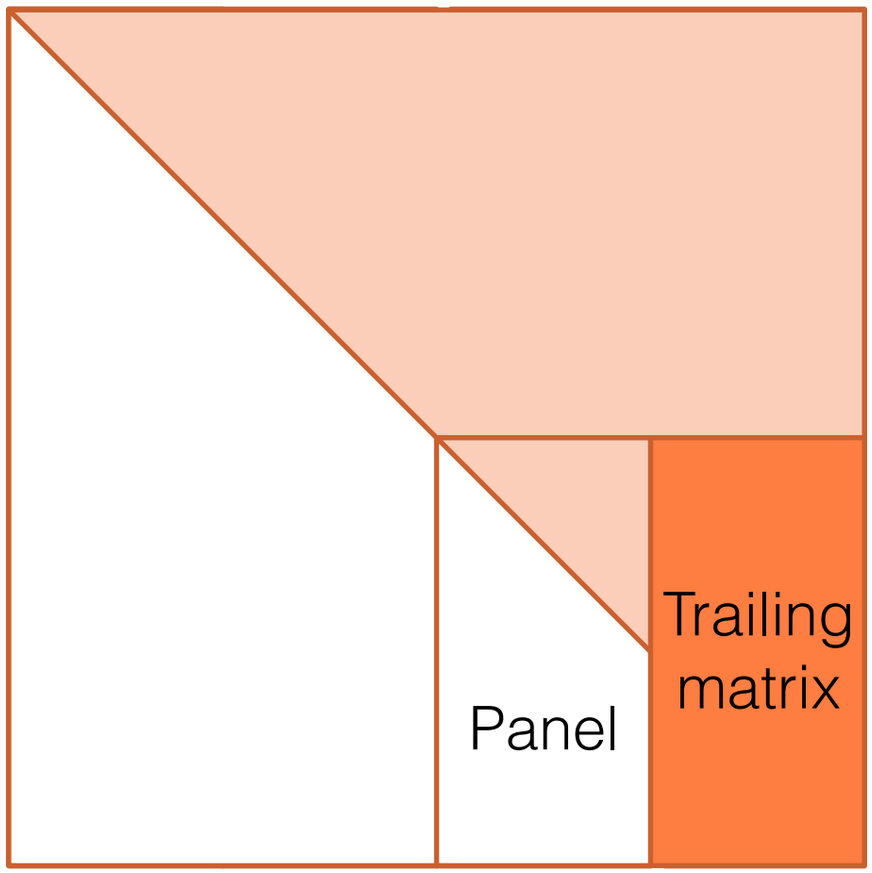


One-sided factorizations

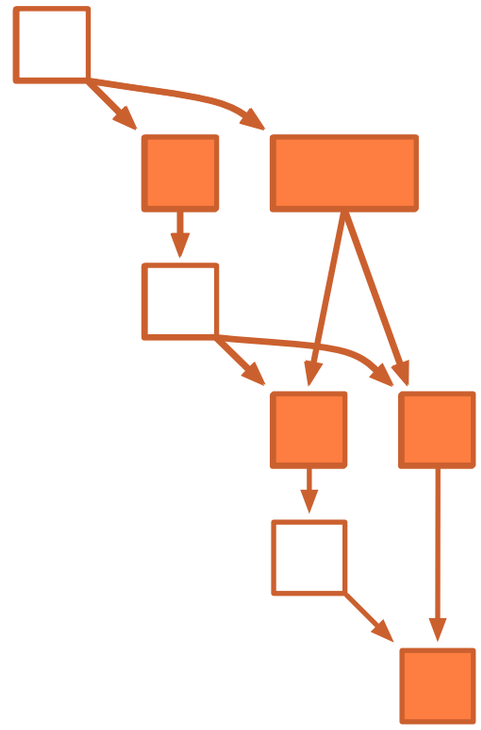
LU, Cholesky, QR factorizations for solving linear systems

Level 2 BLAS on CPU

Level 3 BLAS on GPU



DAG

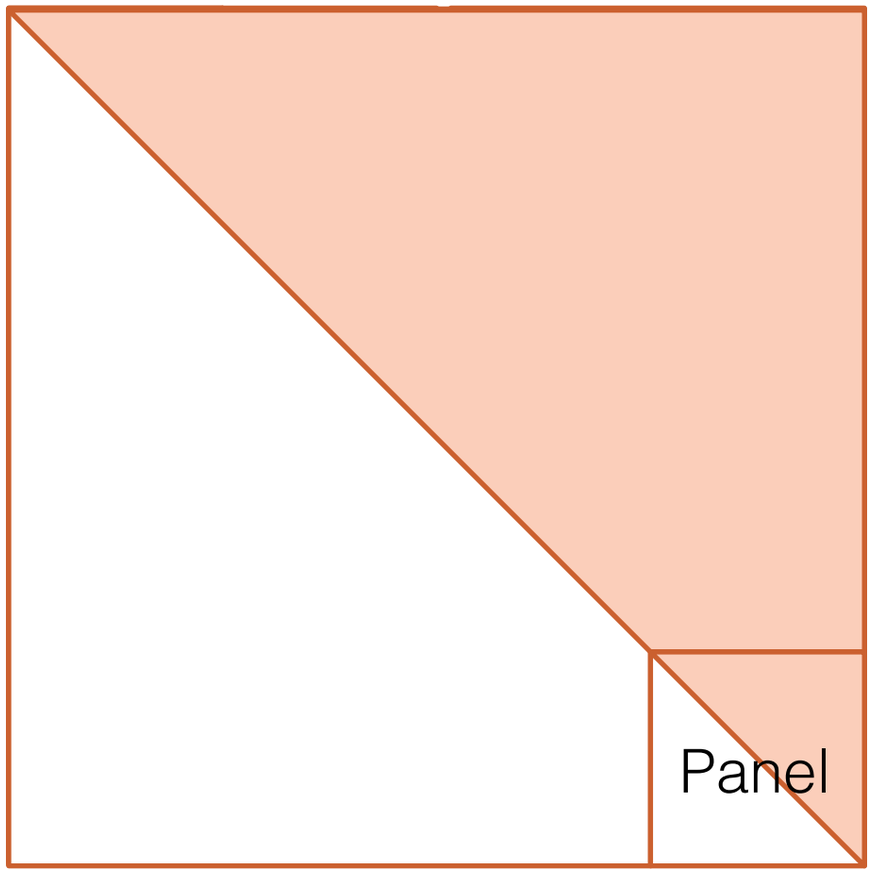


One-sided factorizations

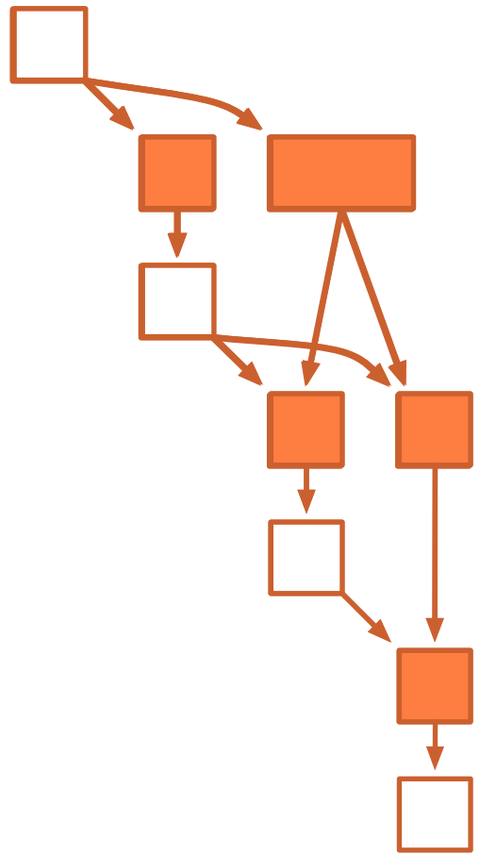
LU, Cholesky, QR factorizations for solving linear systems

Level 2 BLAS on CPU

Level 3 BLAS on GPU



DAG

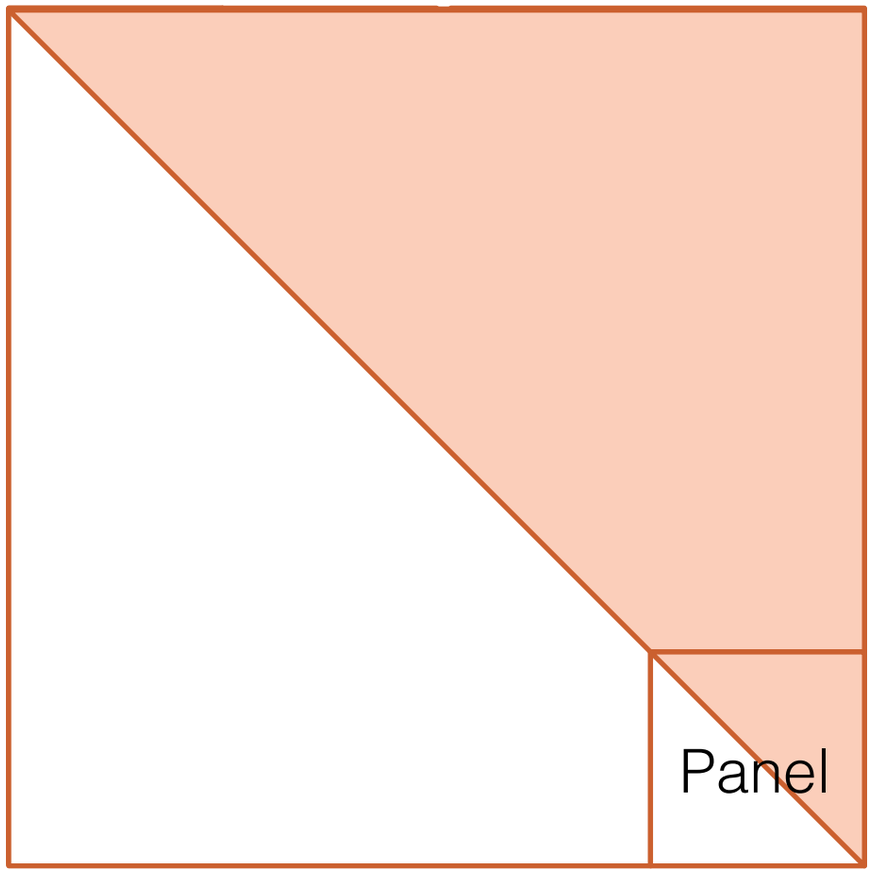


One-sided factorizations

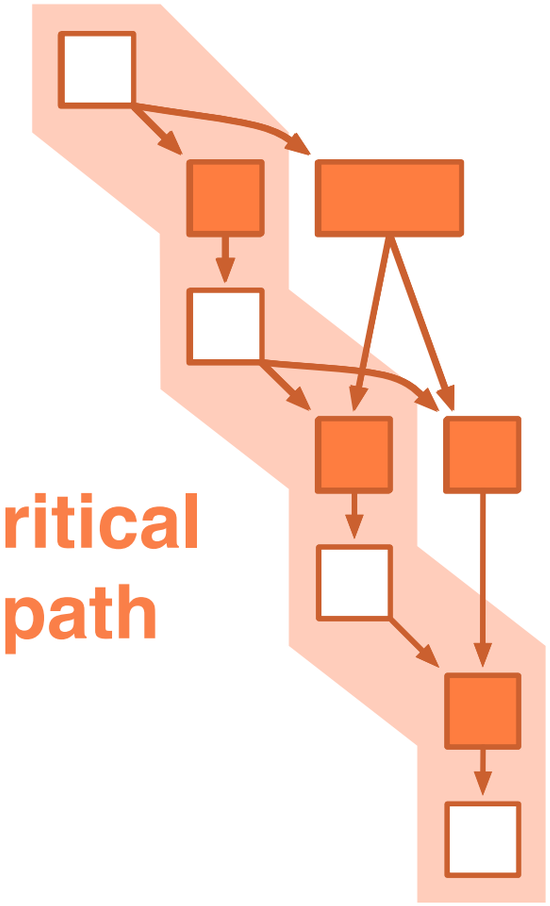
LU, Cholesky, QR factorizations for solving linear systems

Level 2 BLAS on CPU

Level 3 BLAS on GPU



DAG

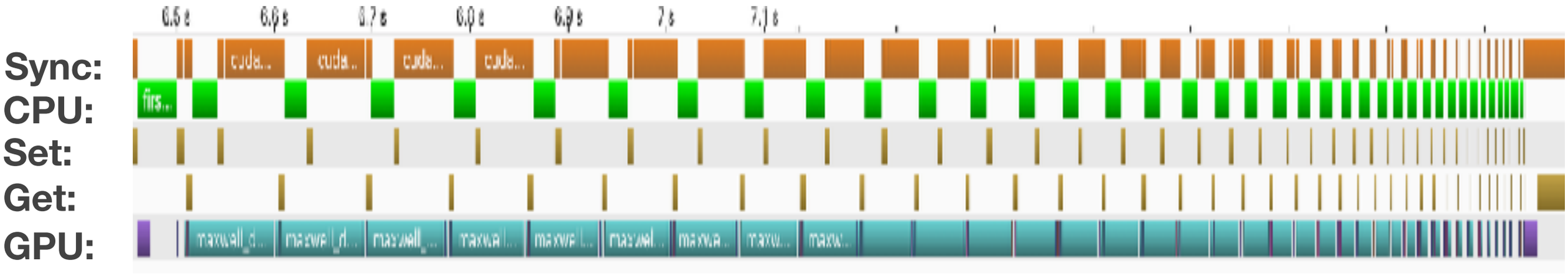


critical path

Execution trace

Panels on CPU **(green)** and **set/get communication (brown)** overlapped with **trailing matrix updates (teal)** on GPU

Goal to keep GPU busy all the time; CPU may idle



LU factorization (dgetrf), n = 20000
P100 GPU, 2 × 10-core 2.3 GHz Haswell

Optimization: for LU, we transpose matrix on GPU so row-swaps are fast

Two-sided factorizations

Hessenberg, tridiagonal, bidiagonal factorizations for eigenvalue and singular value problems

Level 2 BLAS on CPU

Level 2 BLAS on GPU

Level 3 BLAS on GPU