

# MAGMA Library

version 0.2

---

S. Tomov   R. Nath   P. Du   J. Dongarra

-- MAGMA (version 0.2) --  
Univ. of Tennessee, Knoxville  
Univ. of California, Berkeley  
Univ. of Colorado, Denver  
November 2009

MAGMA project homepage:  
<http://icl.cs.utk.edu/magma/>

MAGMA project collaborators:  
M. Baboulin (U Coimbra, Portugal)  
J. Demmel (UC Berkeley)  
J. Dongarra (UT Knoxville)  
P. Du (UT Knoxville)  
J. Kurzak (UT Knoxville)  
H. Ltaief (UT Knoxville)  
P. Luszczek (UT Knoxville)  
J. Langou (UC Denver)  
R. Nath (UT Knoxville)  
S. Tomov (UT Knoxville)  
V. Volkov (UC Berkeley)

# Contents

<b>1</b>	<b>The MAGMA Library</b>	<b>5</b>
1.1	One-sided matrix factorizations . . . . .	7
1.1.1	Function <b>magma_sgetrf</b> . . . . .	8
1.1.2	Function <b>magma_sgeqrf</b> . . . . .	9
1.1.3	Function <b>magma_spotrf</b> . . . . .	10
1.1.4	Function <b>magma_sgeqlf</b> . . . . .	11
1.1.5	Function <b>magma_sgelqf</b> . . . . .	12
1.1.6	Function <b>magma_sgetrf_gpu</b> . . . . .	13
1.1.7	Function <b>magma_sgeqrf_gpu</b> . . . . .	14
1.1.8	Function <b>magma_spotrf_gpu</b> . . . . .	15
1.1.9	Function <b>magma_dgetrf</b> . . . . .	16
1.1.10	Function <b>magma_dgeqrf</b> . . . . .	17
1.1.11	Function <b>magma_dpotrf</b> . . . . .	18
1.1.12	Function <b>magma_dgeqlf</b> . . . . .	19
1.1.13	Function <b>magma_dgelqf</b> . . . . .	20
1.1.14	Function <b>magma_dgetrf_gpu</b> . . . . .	21
1.1.15	Function <b>magma_dgeqrf_gpu</b> . . . . .	22
1.1.16	Function <b>magma_dpotrf_gpu</b> . . . . .	23
1.1.17	Function <b>magma_cgetrf</b> . . . . .	24
1.1.18	Function <b>magma_cgeqrf</b> . . . . .	25
1.1.19	Function <b>magma_cpotrf</b> . . . . .	26
1.1.20	Function <b>magma_cgetrf_gpu</b> . . . . .	27
1.1.21	Function <b>magma_cgeqrf_gpu</b> . . . . .	28
1.1.22	Function <b>magma_cpotrf_gpu</b> . . . . .	29
1.1.23	Function <b>magma_zgetrf</b> . . . . .	30
1.1.24	Function <b>magma_zgeqrf</b> . . . . .	31
1.1.25	Function <b>magma_zpotrf</b> . . . . .	32
1.1.26	Function <b>magma_zgetrf_gpu</b> . . . . .	33
1.1.27	Function <b>magma_zgeqrf_gpu</b> . . . . .	34
1.1.28	Function <b>magma_zpotrf_gpu</b> . . . . .	35
1.2	Linear solvers . . . . .	36
1.2.1	Function <b>magma_sgetrs_gpu</b> . . . . .	37
1.2.2	Function <b>magma_sgeqrs_gpu</b> . . . . .	38
1.2.3	Function <b>magma_spotrs_gpu</b> . . . . .	39

1.2.4	Function <b>magma_dgetrs_gpu</b> . . . . .	40
1.2.5	Function <b>magma_dgeqrs_gpu</b> . . . . .	41
1.2.6	Function <b>magma_dpotrs_gpu</b> . . . . .	42
1.2.7	Function <b>magma_dsgeev_gpu</b> . . . . .	43
1.2.8	Function <b>magma_dsgeqrsv_gpu</b> . . . . .	45
1.2.9	Function <b>magma_dsposv_gpu</b> . . . . .	48
1.3	Two-sided matrix factorizations . . . . .	50
1.3.1	Function <b>magma_sgehrd</b> . . . . .	51
1.3.2	Function <b>magma_dgehrd</b> . . . . .	53
<b>2</b>	<b>The MAGMA BLAS Library</b>	<b>55</b>
2.1	Matrix-Matrix Multiplication . . . . .	56
2.1.1	Function <b>magmablas_sgemm</b> . . . . .	57
2.1.2	Function <b>magmablas_dgemm</b> . . . . .	59
2.2	Matrix-Vector Multiplication . . . . .	61
2.2.1	Function <b>magma_smemv</b> . . . . .	62
2.2.2	Function <b>magma_dgemv</b> . . . . .	63
2.3	Symmetric Matrix-Vector multiplication . . . . .	64
2.3.1	Function <b>magmablas_ssylv</b> . . . . .	65
2.3.2	Function <b>magmablas_dsylv</b> . . . . .	67
2.4	Triangular Matrix Solvers . . . . .	69
2.4.1	Function <b>magma_strsm</b> . . . . .	70
2.4.2	Function <b>magma_dtrsm</b> . . . . .	72
<b>3</b>	<b>Use</b>	<b>74</b>
3.1	Hardware specifications . . . . .	74
3.2	Software specifications . . . . .	74
3.3	Examples and testing . . . . .	75
<b>4</b>	<b>Performance</b>	<b>76</b>
4.1	Single precision one-sided factorizations . . . . .	77
4.2	Double precision one-sided factorizations . . . . .	78
4.3	Single complex one-sided factorizations . . . . .	79
4.4	Double complex one-sided factorizations . . . . .	80
4.5	LU-based linear solvers . . . . .	81
4.6	QR-based least squares solvers . . . . .	82
4.7	Cholesky-based linear solvers . . . . .	83
	<b>Acknowledgments</b>	<b>83</b>
	<b>Bibliography</b>	<b>84</b>

# Chapter 1

## The MAGMA Library

The goal of the *Matrix Algebra on GPU and Multicore Architectures* (MAGMA) project is to create a new generation of linear algebra libraries that achieve the fastest possible time to an accurate solution on hybrid/heterogeneous architectures, starting with current multicore+multiGPU systems. To address the complex challenges stemming from these systems' heterogeneity, massive parallelism, and gap in compute power vs CPU-GPU communication speeds, MAGMA's research is based on the idea that optimal software solutions will themselves have to hybridize, combining the strengths of different algorithms within a single framework. Building on this idea, the goal is to design linear algebra algorithms and frameworks for hybrid multicore and multiGPU systems that can enable applications to fully exploit the power that each of the hybrid components offers.

Designed to be similar to LAPACK in functionality, data storage, and interface, the MAGMA library will allow scientists to effortlessly port their LAPACK-relying software components and to take advantage of the new hybrid architectures.

**MAGMA version 0.2** is a release intended for a single GPU – see the specifications in Section 3.1. MAGMA (version 0.2) includes the one-sided matrix factorizations and solvers based on them, including mixed-precision iterative refinement solvers. The factorizations are provided in all 4 precisions – single, double, single complex, and double complex. For each function there are 2 LAPACK-style interfaces. The first one, referred to as **CPU interface**, takes the input and produces the result in the CPU's memory. The second, referred to as **GPU interface**, takes the input and produces the result in the GPU's memory. Work is in progress on the two-sided factorizations and eigen-solvers based on them. Included is the reduction to upper Hessenberg form in single and double precision. Included is also MAGMA BLAS, a complementary to CUBLAS subset of CUDA BLAS that are crucial for the performance of MAGMA routines. MAGMA uses standard data layout (column major) and

can be used as a complement to LAPACK to accelerate the functions currently provided.

The algorithm names are derived by the corresponding LAPACK names, prefixed by `magma_`, and for the case of the GPU interface suffixed by `_gpu`.

MAGMA version 0.1 included the LU, QR, and Cholesky factorizations in real arithmetic (single and double) for both CPU and GPU interfaces [5]. The following list gives the additions that are now available in MAGMA version 0.2:

- Complex arithmetic (single and double) LU, QR, and Cholesky factorizations for both CPU and GPU interfaces;
- LQ and QL factorizations in real arithmetic (both single and double precision);
- Linear solvers based on LU, QR, and Cholesky in real arithmetic (single and double);
- Mixed-precision, iterative refinement solvers based on LU, QR, and Cholesky in real arithmetic;
- Reduction to upper Hessenberg form in real arithmetic (single and double)
- MAGMA BLAS in real arithmetic (single and double), including `gemm` and `trsm`.

A reference performance is given in Chapter 4.

## 1.1 One-sided matrix factorizations

We use hybrid algorithms where the computation is split between the GPU and the CPU. In general for the one-sided factorizations, the panels are factored on the CPU and the trailing sub-matrix updates on the GPU. Look-ahead techniques are used to overlap the CPU and GPU work (and some communications). Figure 1.1 illustrates this by quantifying the CPU-GPU overlap for the case of QR in single precision arithmetic.

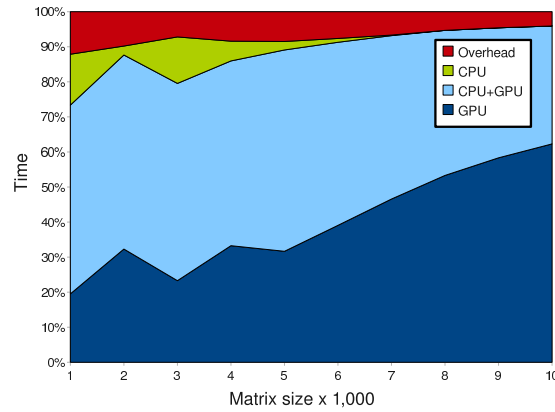


Figure 1.1: Time breakdown for the hybrid QR from MAGMA in single precision arithmetic on GTX280 GPU and Intel Xeon 2.33GHz CPU.

In both the CPU and GPU interfaces the matrix to be factored resides on the GPU memory, and CPU-GPU transfers are associated only with the panels. The resulting matrix is accumulated (on the CPU or GPU according to the interface) along the computation, as a byproduct of the algorithm, *vs* sending the the entire matrix when needed. In the CPU interface, the original transfer of the matrix to the GPU is overlapped with the factorization of the first panel. In this sense the CPU and GPU interfaces, although similar, are not derivatives of each other as they have different communication patterns.

Besides this common approach, the different algorithms have specific optimizations motivated by the GPU architecture [4, 7].

### 1.1.1 Function magma\_sgetrf

```
int magma_sgetrf(int *m, int *n, float *a, int *lda,
                int *ipiv, float *work, float *da, int *info)
```

SGETRF computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.

The factorization has the form

$$A = P * L * U$$

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if  $m > n$ ), and U is upper triangular (upper trapezoidal if  $m < n$ ).

This is the right-looking Level 3 BLAS version of the algorithm.

**M** (input) INTEGER  
The number of rows of the matrix A.  $M \geq 0$ .

**N** (input) INTEGER  
The number of columns of the matrix A.  $N \geq 0$ .

**A** (input/output) REAL array, dimension (LDA,N)  
On entry, the M-by-N matrix to be factored.  
On exit, the factors L and U from the factorization  $A = P*L*U$ ; the unit diagonal elements of L are not stored.  
Higher performance is achieved if A is in pinned memory, e.g. allocated using cudaMallocHost.

**LDA** (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1,M)$ .

**IPIV** (output) INTEGER array, dimension (min(M,N))  
The pivot indices; for  $1 \leq i \leq \min(M,N)$ , row i of the matrix was interchanged with row IPIV(i).

**WORK** (workspace/output) REAL array, dimension  $\geq N*NB$ ,  
where NB can be obtained through magma\_get\_sgetrf\_nb(M).  
Higher performance is achieved if WORK is in pinned memory, e.g. allocated using cudaMallocHost.

**DA** (workspace) REAL array on the GPU, dimension  $(\max(M, N) + k1)^2 + (M + k2)*NB + 2*NB^2$ ,  
where NB can be obtained through magma\_get\_sgetrf\_nb(M).  
 $k1 < 32$  and  $k2 < 32$  are such that  $(\max(M, N) + k1)\%32==0$  and  $(M+k2)\%32==0$ .

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value  
> 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.



### 1.1.2 Function magma\_sgeqrf

```
int magma_sgeqrf(int *m, int *n, float *a, int *lda, float *tau,
                float *work, int *lwork, float *da, int *info )
```

SGEQRF computes a QR factorization of a real M-by-N matrix A:  $A = Q * R$ .

**M** (input) INTEGER  
The number of rows of the matrix A.  $M \geq 0$ .

**N** (input) INTEGER  
The number of columns of the matrix A.  $N \geq 0$ .

**A** (input/output) REAL array, dimension (LDA,N)  
On entry, the M-by-N matrix A.  
On exit, the elements on and above the diagonal of the array contain the  $\min(M,N)$ -by-N upper trapezoidal matrix R (R is upper triangular if  $m \geq n$ ); the elements below the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of  $\min(m,n)$  elementary reflectors.  
Higher performance is achieved if A is in pinned memory, e.g. allocated using cudaMallocHost.

**LDA** (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1,M)$ .

**TAU** (output) REAL array, dimension (min(M,N))  
The scalar factors of the elementary reflectors.

**WORK** (workspace/output) REAL array, dimension (MAX(1,LWORK))  
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.  
Higher performance is achieved if WORK is in pinned memory, e.g. allocated using cudaMallocHost.

**LWORK** (input) INTEGER  
The dimension of the array WORK.  $LWORK \geq N * NB$ , where NB can be obtained through magma\_get\_sgeqrf\_nb(M).  
  
If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued.

**DA** (workspace) REAL array on the GPU, dimension  $N * (M + NB)$ , where NB can be obtained through magma\_get\_sgeqrf\_nb(M). (size to be reduced in upcoming versions).

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value

The matrix Q is represented as a product of elementary reflectors

$Q = H(1) H(2) \dots H(k)$ , where  $k = \min(m,n)$ .

Each H(i) has the form

$H(i) = I - \tau * v * v'$

where tau is a real scalar, and v is a real vector with  $v(1:i-1) = 0$  and  $v(i) = 1$ ;  $v(i+1:m)$  is stored on exit in  $A(i+1:m,i)$ , and tau in TAU(i).

### 1.1.3 Function magma\_spotrf

```
int magma_spotrf(char *uplo, int *n, float *a, int *lda, float *work,
                int *info)
```

SPOTRF computes the Cholesky factorization of a real symmetric positive definite matrix A.

The factorization has the form

$A = U^{*T} * U$ , if UPLO = 'U', or

$A = L * L^{*T}$ , if UPLO = 'L',

where U is an upper triangular matrix and L is lower triangular.

This is the block version of the algorithm, calling Level 3 BLAS.

UPLO (input) CHARACTER\*1  
 = 'U': Upper triangle of A is stored;  
 = 'L': Lower triangle of A is stored.

N (input) INTEGER  
 The order of the matrix A.  $N \geq 0$ .

A (input/output) REAL array, dimension (LDA,N)  
 On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization  $A = U^{*T} * U$  or  $A = L * L^{*T}$ .  
 Higher performance is achieved if A is in pinned memory, e.g. allocated using cudaMallocHost.

LDA (input) INTEGER  
 The leading dimension of the array A.  $LDA \geq \max(1, N)$ .

WORK (workspace) REAL array on the GPU, dimension (N, N)  
 (size to be reduced in upcoming versions).

INFO (output) INTEGER  
 = 0: successful exit  
 < 0: if INFO = -i, the i-th argument had an illegal value  
 > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

### 1.1.4 Function magma\_sgeqlf

```
int magma_sgeqlf(int *m, int *n, float *a, int *lda,
                float *tau, float *work, int *lwork, float *da, int *info)
```

SGEQLF computes a QL factorization of a real M-by-N matrix A:  $A = Q * L$ .

**M** (input) INTEGER  
The number of rows of the matrix A.  $M \geq 0$ .

**N** (input) INTEGER  
The number of columns of the matrix A.  $N \geq 0$ .

**A** (input/output) REAL array, dimension (LDA,N)  
On entry, the M-by-N matrix A.  
On exit, if  $m \geq n$ , the lower triangle of the subarray  $A(m-n+1:m,1:n)$  contains the N-by-N lower triangular matrix L; if  $m < n$ , the elements on and below the (n-m)-th superdiagonal contain the M-by-N lower trapezoidal matrix L; the remaining elements, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors.  
Higher performance is achieved if A is in pinned memory, e.g. allocated using cudaMallocHost.

**LDA** (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1,M)$ .

**TAU** (output) REAL array, dimension (min(M,N))  
The scalar factors of the elementary reflectors.

**WORK** (workspace/output) REAL array, dimension (MAX(1,LWORK))  
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.  
Higher performance is achieved if WORK is in pinned memory, e.g. allocated using cudaMallocHost.

**LWORK** (input) INTEGER  
The dimension of the array WORK.  $LWORK \geq \max(1,N)$ .  
For optimum performance  $LWORK \geq N * NB$ , where NB is the optimal blocksize. If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued.

**DA** (workspace) REAL array on the GPU, dimension  $N * (M + NB)$ , where NB can be obtained through magma\_get\_sgeqlf\_nb(M). (size to be reduced in upcoming versions).

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value

The matrix Q is represented as a product of elementary reflectors

$$Q = H(k) \dots H(2) H(1), \text{ where } k = \min(m,n).$$

Each  $H(i)$  has the form

$$H(i) = I - \tau * v * v'$$

where  $\tau$  is a real scalar, and  $v$  is a real vector with

$v(m-k+i+1:m) = 0$  and  $v(m-k+i) = 1$ ;  $v(1:m-k+i-1)$  is stored on exit in  $A(1:m-k+i-1, n-k+i)$ , and  $\tau$  in TAU(i).

### 1.1.5 Function magma\_sgelqf

```
int magma_sgelqf(int *m, int *n, float *a, int *lda, float *tau,
                float *work, int *lwork, float *da, int *info)
```

SGELQF computes an LQ factorization of a real M-by-N matrix A:  $A = L * Q$ .

**M** (input) INTEGER  
The number of rows of the matrix A.  $M \geq 0$ .

**N** (input) INTEGER  
The number of columns of the matrix A.  $N \geq 0$ .

**A** (input/output) REAL array, dimension (LDA,N)  
On entry, the M-by-N matrix A.  
On exit, the elements on and below the diagonal of the array contain the m-by-min(m,n) lower trapezoidal matrix L (L is lower triangular if  $m \leq n$ ); the elements above the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors.  
Higher performance is achieved if A is in pinned memory, e.g. allocated using cudaMallocHost.

**LDA** (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1,M)$ .

**TAU** (output) REAL array, dimension (min(M,N))  
The scalar factors of the elementary reflectors.

**WORK** (workspace/output) REAL array, dimension (MAX(1,LWORK))  
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.  
Higher performance is achieved if WORK is in pinned memory, e.g. allocated using cudaMallocHost.

**LWORK** (input) INTEGER  
The dimension of the array WORK.  $LWORK \geq \max(1,M)$ .  
For optimum performance  $LWORK \geq M * NB$ , where NB is the optimal blocksize. If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message is issued.

**DA** (workspace) REAL array on the GPU, dimension  $M * (N + NB)$ , where NB can be obtained through magma\_get\_sgeqrf\_nb(M).  
(size to be reduced in upcoming versions).

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value

The matrix Q is represented as a product of elementary reflectors

$$Q = H(k) \dots H(2) H(1), \text{ where } k = \min(m,n).$$

Each H(i) has the form

$$H(i) = I - \tau * v * v'$$

where tau is a real scalar, and v is a real vector with

$v(1:i-1) = 0$  and  $v(i) = 1$ ;  $v(i+1:n)$  is stored on exit in  $A(i,i+1:n)$ , and tau in TAU(i).

### 1.1.6 Function magma\_sgetrf\_gpu

```
int magma_sgetrf_gpu(int *m, int *n, float *a, int *lda,
                    int *ipiv, float *work, int *info)
```

SGETRF computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.

The factorization has the form

$$A = P * L * U$$

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if  $m > n$ ), and U is upper triangular (upper trapezoidal if  $m < n$ ).

This is the right-looking Level 3 BLAS version of the algorithm.

**M** (input) INTEGER  
The number of rows of the matrix A.  $M \geq 0$ .

**N** (input) INTEGER  
The number of columns of the matrix A.  $N \geq 0$ .

**A** (input/output) REAL array on the GPU, dimension (LDA,N) where  $LDA \geq \max(M, N) + k_1$ ,  $k_1 < 32$  such that  $(\max(M, N) + k_1) \% 32 == 0$ .  
The memory pointed by A should be at least  $(\max(M, N) + k_1)^2 + (M + k_2) * NB + 2 * NB^2$  where  $k_2 < 32$  such that  $(M + k_2) \% 32 == 0$ .  
  
On entry, the M-by-N matrix to be factored.  
On exit, the factors L and U from the factorization  $A = P * L * U$ ; the unit diagonal elements of L are not stored.  
The rest of A is considered work space and is changed.

**LDA** (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1, M)$ .

**IPIV** (output) INTEGER array, dimension (min(M,N))  
The pivot indices; for  $1 \leq i \leq \min(M, N)$ , row i of the matrix was interchanged with row IPIV(i).

**WORK** (workspace/output) REAL array, dimension  $\geq N * NB$ , where NB can be obtained through `magma_get_sgetrf_nb(M)`.  
Higher performance is achieved if WORK is in pinned memory, e.g. allocated using `cudaMallocHost`.

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value  
> 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

### 1.1.7 Function magma\_sgeqrf\_gpu

```
int magma_sgeqrf_gpu(int *m, int *n, float *a, int *lda, float *tau,
                    float *work, int *lwork, float *dwork, int *info )
```

SGEQRF computes a QR factorization of a real M-by-N matrix A:  $A = Q * R$ .

**M** (input) INTEGER  
The number of rows of the matrix A.  $M \geq 0$ .

**N** (input) INTEGER  
The number of columns of the matrix A.  $N \geq 0$ .

**A** (input/output) REAL array on the GPU, dimension (LDA,N)  
On entry, the M-by-N matrix A.  
On exit, the elements on and above the diagonal of the array contain the  $\min(M,N)$ -by-N upper trapezoidal matrix R (R is upper triangular if  $m \geq n$ ); the elements below the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of  $\min(m,n)$  elementary reflectors.

**LDA** (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1,M)$ .

**TAU** (output) REAL array, dimension (min(M,N))  
The scalar factors of the elementary reflectors (see Further Details).

**WORK** (workspace/output) REAL array, dimension (MAX(1,LWORK))  
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.  
Higher performance is achieved if WORK is in pinned memory, e.g. allocated using cudaMallocHost.

**LWORK** (input) INTEGER  
The dimension of the array WORK.  $LWORK \geq (M+N)*NB$ , where NB can be obtained through magma\_get\_sgeqrf\_nb(M).  
  
If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued.

**DWORK** (workspace) REAL array on the GPU, dimension N\*NB,  
where NB can be obtained through magma\_get\_sgeqrf\_nb(M).

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value

The matrix Q is represented as a product of elementary reflectors

$$Q = H(1) H(2) \dots H(k), \text{ where } k = \min(m,n).$$

Each H(i) has the form

$$H(i) = I - \tau * v * v'$$

where tau is a real scalar, and v is a real vector with  $v(1:i-1) = 0$  and  $v(i) = 1$ ;  $v(i+1:m)$  is stored on exit in  $A(i+1:m,i)$ , and tau in TAU(i).

### 1.1.8 Function magma\_spotrf\_gpu

```
int magma_spotrf_gpu(char *uplo, int *n, float *a, int *lda,
                    float *work, int *info)
```

SPOTRF computes the Cholesky factorization of a real symmetric positive definite matrix A.

The factorization has the form

$A = U^{*T} * U$ , if UPLO = 'U', or

$A = L * L^{*T}$ , if UPLO = 'L',

where U is an upper triangular matrix and L is lower triangular.

This is the block version of the algorithm, calling Level 3 BLAS.

UPLO (input) CHARACTER\*1  
 = 'U': Upper triangle of A is stored;  
 = 'L': Lower triangle of A is stored.

N (input) INTEGER  
 The order of the matrix A.  $N \geq 0$ .

A (input/output) REAL array on the GPU, dimension (LDA,N)  
 On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization  $A = U^{*T} * U$  or  $A = L * L^{*T}$ .

LDA (input) INTEGER  
 The leading dimension of the array A.  $LDA \geq \max(1, N)$ .

WORK (workspace) REAL array, dimension at least (nb, nb)  
 where nb can be obtained through `magma_get_spotrf_nb(*n)`  
 Work array allocated with `cudaMallocHost`.

INFO (output) INTEGER  
 = 0: successful exit  
 < 0: if INFO = -i, the i-th argument had an illegal value  
 > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

### 1.1.9 Function magma\_dgetrf

```
int magma_dgetrf(int *m, int *n, double *a, int *lda,
                int *ipiv, double *work, double *da, int *info)
```

DGETRF computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.

The factorization has the form

$$A = P * L * U$$

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if  $m > n$ ), and U is upper triangular (upper trapezoidal if  $m < n$ ).

This is the right-looking Level 3 BLAS version of the algorithm.

**M** (input) INTEGER  
The number of rows of the matrix A.  $M \geq 0$ .

**N** (input) INTEGER  
The number of columns of the matrix A.  $N \geq 0$ .

**A** (input/output) DOUBLE array, dimension (LDA,N)  
On entry, the M-by-N matrix to be factored.  
On exit, the factors L and U from the factorization  $A = P*L*U$ ; the unit diagonal elements of L are not stored.  
Higher performance is achieved if A is in pinned memory, e.g. allocated using cudaMallocHost.

**LDA** (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1,M)$ .

**IPIV** (output) INTEGER array, dimension (min(M,N))  
The pivot indices; for  $1 \leq i \leq \min(M,N)$ , row i of the matrix was interchanged with row IPIV(i).

**WORK** (workspace/output) DOUBLE array, dimension  $\geq N*NB$ ,  
where NB can be obtained through magma\_get\_sgetrf\_nb(M).  
Higher performance is achieved if WORK is in pinned memory, e.g. allocated using cudaMallocHost.

**DA** (workspace) DOUBLE array on the GPU, dimension  $(\max(M, N) + k1)^2 + (M + k2)*NB + 2*NB^2$ ,  
where NB can be obtained through magma\_get\_sgetrf\_nb(M).  
 $k1 < 32$  and  $k2 < 32$  are such that  $(\max(M, N) + k1)\%32==0$  and  $(M+k2)\%32==0$ .

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value  
> 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.



### 1.1.10 Function magma\_dgeqrf

```
int magma_dgeqrf(int *m, int *n, double *a, int *lda, double *tau,
                 double *work, int *lwork, double *da, int *info )
```

DGEQRF computes a QR factorization of a real M-by-N matrix A:  $A = Q * R$ .

**M** (input) INTEGER  
The number of rows of the matrix A.  $M \geq 0$ .

**N** (input) INTEGER  
The number of columns of the matrix A.  $N \geq 0$ .

**A** (input/output) DOUBLE array, dimension (LDA,N)  
On entry, the M-by-N matrix A.  
On exit, the elements on and above the diagonal of the array contain the  $\min(M,N)$ -by-N upper trapezoidal matrix R (R is upper triangular if  $m \geq n$ ); the elements below the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of  $\min(m,n)$  elementary reflectors.  
Higher performance is achieved if A is in pinned memory, e.g. allocated using cudaMallocHost.

**LDA** (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1,M)$ .

**TAU** (output) DOUBLE array, dimension (min(M,N))  
The scalar factors of the elementary reflectors.

**WORK** (workspace/output) DOUBLE array, dimension (MAX(1,LWORK))  
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.  
Higher performance is achieved if WORK is in pinned memory, e.g. allocated using cudaMallocHost.

**LWORK** (input) INTEGER  
The dimension of the array WORK.  $LWORK \geq N * NB$ , where NB can be obtained through magma\_get\_dgeqrf\_nb(M).  
  
If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued.

**DA** (workspace) DOUBLE array on the GPU, dimension  $N * (M + NB)$ , where NB can be obtained through magma\_get\_dgeqrf\_nb(M). (size to be reduced in upcoming versions).

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value

The matrix Q is represented as a product of elementary reflectors

$Q = H(1) H(2) \dots H(k)$ , where  $k = \min(m,n)$ .

Each H(i) has the form

$H(i) = I - \tau * v * v'$

where tau is a real scalar, and v is a real vector with  $v(1:i-1) = 0$  and  $v(i) = 1$ ;  $v(i+1:m)$  is stored on exit in  $A(i+1:m,i)$ , and tau in TAU(i).

### 1.1.11 Function magma\_dpotrf

```
int magma_dpotrf(char *uplo, int *n, double *a, int *lda, double *work,
                int *info)
```

DPOTRF computes the Cholesky factorization of a real symmetric positive definite matrix A.

The factorization has the form

$A = U^{*T} * U$ , if UPLO = 'U', or

$A = L * L^{*T}$ , if UPLO = 'L',

where U is an upper triangular matrix and L is lower triangular.

This is the block version of the algorithm, calling Level 3 BLAS.

UPLO (input) CHARACTER\*1

= 'U': Upper triangle of A is stored;

= 'L': Lower triangle of A is stored.

N (input) INTEGER

The order of the matrix A.  $N \geq 0$ .

A (input/output) DOUBLE array, dimension (LDA,N)

On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization  $A = U^{*T} * U$  or  $A = L * L^{*T}$ .

Higher performance is achieved if A is in pinned memory, e.g. allocated using cudaMallocHost.

LDA (input) INTEGER

The leading dimension of the array A.  $LDA \geq \max(1, N)$ .

WORK (workspace) DOUBLE array on the GPU, dimension (N, N)

(size to be reduced in upcoming versions).

INFO (output) INTEGER

= 0: successful exit

< 0: if INFO = -i, the i-th argument had an illegal value

> 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

### 1.1.12 Function magma\_dgeqlf

```
int magma_dgeqlf(int *m, int *n, double *a, int *lda,
                 double *tau, double *work, int *lwork, double *da, int *info)
```

DGEQLF computes a QL factorization of a real M-by-N matrix A:  $A = Q * L$ .

**M** (input) INTEGER  
The number of rows of the matrix A.  $M \geq 0$ .

**N** (input) INTEGER  
The number of columns of the matrix A.  $N \geq 0$ .

**A** (input/output) DOUBLE REAL array, dimension (LDA,N)  
On entry, the M-by-N matrix A.  
On exit, if  $m \geq n$ , the lower triangle of the subarray  $A(m-n+1:m,1:n)$  contains the N-by-N lower triangular matrix L; if  $m < n$ , the elements on and below the (n-m)-th superdiagonal contain the M-by-N lower trapezoidal matrix L; the remaining elements, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors.  
Higher performance is achieved if A is in pinned memory, e.g. allocated using cudaMallocHost.

**LDA** (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1,M)$ .

**TAU** (output) DOUBLE REAL array, dimension (min(M,N))  
The scalar factors of the elementary reflectors.

**WORK** (workspace/output) DOUBLE REAL array, dimension (MAX(1,LWORK))  
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.  
Higher performance is achieved if WORK is in pinned memory, e.g. allocated using cudaMallocHost.

**LWORK** (input) INTEGER  
The dimension of the array WORK.  $LWORK \geq \max(1,N)$ .  
For optimum performance  $LWORK \geq N * NB$ , where NB is the optimal blocksize. If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued.

**DA** (workspace) DOUBLE REAL array on the GPU, dimension  $N * (M + NB)$ , where NB can be obtained through magma\_get\_dgeqlf\_nb(M).  
(size to be reduced in upcoming versions).

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value

The matrix Q is represented as a product of elementary reflectors

$Q = H(k) \dots H(2) H(1)$ , where  $k = \min(m,n)$ .

Each  $H(i)$  has the form

$H(i) = I - \tau * v * v'$

where tau is a real scalar, and v is a real vector with

$v(m-k+i+1:m) = 0$  and  $v(m-k+i) = 1$ ;  $v(1:m-k+i-1)$  is stored on exit in  $A(1:m-k+i-1, n-k+i)$ , and tau in TAU(i).

### 1.1.13 Function magma\_dgelqf

```
int magma_dgelqf(int *m, int *n, double *a, int *lda, double *tau,
                double *work, int *lwork, double *da, int *info)
```

DGELQF computes an LQ factorization of a real M-by-N matrix A:  $A = L * Q$ .

**M** (input) INTEGER  
The number of rows of the matrix A.  $M \geq 0$ .

**N** (input) INTEGER  
The number of columns of the matrix A.  $N \geq 0$ .

**A** (input/output) DOUBLE REAL array, dimension (LDA,N)  
On entry, the M-by-N matrix A.  
On exit, the elements on and below the diagonal of the array contain the m-by-min(m,n) lower trapezoidal matrix L (L is lower triangular if  $m \leq n$ ); the elements above the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors.  
Higher performance is achieved if A is in pinned memory, e.g. allocated using cudaMallocHost.

**LDA** (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1,M)$ .

**TAU** (output) DOUBLE REAL array, dimension (min(M,N))  
The scalar factors of the elementary reflectors.

**WORK** (workspace/output) DOUBLE REAL array, dimension (MAX(1,LWORK))  
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.  
Higher performance is achieved if WORK is in pinned memory, e.g. allocated using cudaMallocHost.

**LWORK** (input) INTEGER  
The dimension of the array WORK.  $LWORK \geq \max(1,M)$ .  
For optimum performance  $LWORK \geq M * NB$ , where NB is the optimal blocksize. If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message is issued.

**DA** (workspace) DOUBLE REAL array on the GPU, dimension  $M * (N + NB)$ , where NB can be obtained through magma\_get\_dgeqrf\_nb(M).  
(size to be reduced in upcoming versions).

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value

The matrix Q is represented as a product of elementary reflectors

$Q = H(k) \dots H(2) H(1)$ , where  $k = \min(m,n)$ .

Each H(i) has the form

$H(i) = I - \tau * v * v'$

where tau is a real scalar, and v is a real vector with

$v(1:i-1) = 0$  and  $v(i) = 1$ ;  $v(i+1:n)$  is stored on exit in  $A(i,i+1:n)$ ,

and tau in TAU(i).

### 1.1.14 Function magma\_dgetrf\_gpu

```
int magma_dgetrf_gpu(int *m, int *n, double *a, int *lda,
                    int *ipiv, double *work, int *info)
```

DGETRF computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.

The factorization has the form

$$A = P * L * U$$

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if  $m > n$ ), and U is upper triangular (upper trapezoidal if  $m < n$ ).

This is the right-looking Level 3 BLAS version of the algorithm.

**M** (input) INTEGER  
The number of rows of the matrix A.  $M \geq 0$ .

**N** (input) INTEGER  
The number of columns of the matrix A.  $N \geq 0$ .

**A** (input/output) DOUBLE array on the GPU, dimension (LDA,N) where  $LDA \geq \max(M, N) + k1$ ,  $k1 < 32$  such that  $(\max(M, N) + k1) \% 32 == 0$ .  
The memory pointed by A should be at least  $(\max(M, N) + k1)^2 + (M + k2) * NB + 2 * NB^2$  where  $k2 < 32$  such that  $(M + k2) \% 32 == 0$ .  
  
On entry, the M-by-N matrix to be factored.  
On exit, the factors L and U from the factorization  $A = P * L * U$ ; the unit diagonal elements of L are not stored.  
The rest of A is considered work space and is changed.

**LDA** (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1, M)$ .

**IPIV** (output) INTEGER array, dimension (min(M,N))  
The pivot indices; for  $1 \leq i \leq \min(M, N)$ , row i of the matrix was interchanged with row IPIV(i).

**WORK** (workspace/output) DOUBLE array, dimension  $\geq N * NB$ , where NB can be obtained through `magma_get_dgetrf_nb(M)`.  
Higher performance is achieved if WORK is in pinned memory, e.g. allocated using `cudaMallocHost`.

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value  
> 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

### 1.1.15 Function magma\_dgeqrf\_gpu

```
int magma_dgeqrf_gpu(int *m, int *n, double *a, int *lda, double *tau,
                    double *work, int *lwork, double *dwork, int *info )
```

DGEQRF computes a QR factorization of a real M-by-N matrix A:  $A = Q * R$ .

**M** (input) INTEGER  
The number of rows of the matrix A.  $M \geq 0$ .

**N** (input) INTEGER  
The number of columns of the matrix A.  $N \geq 0$ .

**A** (input/output) DOUBLE array on the GPU, dimension (LDA,N)  
On entry, the M-by-N matrix A.  
On exit, the elements on and above the diagonal of the array contain the  $\min(M,N)$ -by-N upper trapezoidal matrix R (R is upper triangular if  $m \geq n$ ); the elements below the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of  $\min(m,n)$  elementary reflectors.

**LDA** (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1,M)$ .

**TAU** (output) DOUBLE array, dimension ( $\min(M,N)$ )  
The scalar factors of the elementary reflectors.

**WORK** (workspace/output) DOUBLE array, dimension ( $\max(1,LWORK)$ )  
On exit, if  $INFO = 0$ ,  $WORK(1)$  returns the optimal LWORK.  
Higher performance is achieved if WORK is in pinned memory, e.g. allocated using `cudaMallocHost`.

**LWORK** (input) INTEGER  
The dimension of the array WORK.  $LWORK \geq (M+N)*NB$ , where NB can be obtained through `magma_get_dgeqrf_nb(M)`.  
  
If  $LWORK = -1$ , then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued.

**DWORK** (workspace) DOUBLE array on the GPU, dimension  $N*NB$ , where NB can be obtained through `magma_get_dgeqrf_nb(M)`.

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if  $INFO = -i$ , the i-th argument had an illegal value

The matrix Q is represented as a product of elementary reflectors

$$Q = H(1) H(2) \dots H(k), \text{ where } k = \min(m,n).$$

Each  $H(i)$  has the form

$$H(i) = I - \tau * v * v'$$

where  $\tau$  is a real scalar, and  $v$  is a real vector with  $v(1:i-1) = 0$  and  $v(i) = 1$ ;  $v(i+1:m)$  is stored on exit in  $A(i+1:m,i)$ , and  $\tau$  in  $TAU(i)$ .

### 1.1.16 Function magma\_dpotrf\_gpu

```
int magma_dpotrf_gpu(char *uplo, int *n, double *a, int *lda, double *work,
                    int *info)
```

DPOTRF computes the Cholesky factorization of a real symmetric positive definite matrix A.

The factorization has the form

$A = U^{**T} * U$ , if UPLO = 'U', or

$A = L * L^{**T}$ , if UPLO = 'L',

where U is an upper triangular matrix and L is lower triangular.

This is the block version of the algorithm, calling Level 3 BLAS.

UPLO (input) CHARACTER\*1

= 'U': Upper triangle of A is stored;

= 'L': Lower triangle of A is stored.

N (input) INTEGER

The order of the matrix A.  $N \geq 0$ .

A (input/output) DOUBLE array on the GPU, dimension (LDA,N)

On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization  $A = U^{**T} * U$  or  $A = L * L^{**T}$ .

LDA (input) INTEGER

The leading dimension of the array A.  $LDA \geq \max(1, N)$ .

WORK (workspace) DOUBLE array, dimension at least (nb, nb)

where nb can be obtained through `magma_get_dpotrf_nb(*n)`  
Work array allocated with `cudaMallocHost`.

INFO (output) INTEGER

= 0: successful exit

< 0: if INFO = -i, the i-th argument had an illegal value

> 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

### 1.1.17 Function magma\_cgetrf

```
int magma_cgetrf(int *m, int *n, float2 *a, int *lda,
                int *ipiv, float2 *work, float2 *da, int *info)
```

CGETRF computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.

The factorization has the form

$$A = P * L * U$$

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if  $m > n$ ), and U is upper triangular (upper trapezoidal if  $m < n$ ).

This is the right-looking Level 3 BLAS version of the algorithm.

**M** (input) INTEGER  
The number of rows of the matrix A.  $M \geq 0$ .

**N** (input) INTEGER  
The number of columns of the matrix A.  $N \geq 0$ .

**A** (input/output) COMPLEX array, dimension (LDA,N)  
On entry, the M-by-N matrix to be factored.  
On exit, the factors L and U from the factorization  $A = P*L*U$ ; the unit diagonal elements of L are not stored.  
Higher performance is achieved if A is in pinned memory, e.g. allocated using cudaMallocHost.

**LDA** (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1, M)$ .

**IPIV** (output) INTEGER array, dimension (min(M,N))  
The pivot indices; for  $1 \leq i \leq \min(M, N)$ , row i of the matrix was interchanged with row IPIV(i).

**WORK** (workspace/output) COMPLEX array, dimension  $\geq N*NB$ ,  
where NB can be obtained through magma\_get\_cgetrf\_nb(M).  
Higher performance is achieved if WORK is in pinned memory, e.g. allocated using cudaMallocHost.

**DA** (workspace) COMPLEX array on the GPU, dimension  $(\max(M, N) + k1)^2 + (M + k2)*NB + 2*NB^2$ ,  
where NB can be obtained through magma\_get\_cgetrf\_nb(M).  
 $k1 < 32$  and  $k2 < 32$  are such that  $(\max(M, N) + k1)\%32==0$  and  $(M+k2)\%32==0$ .

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value  
> 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.



### 1.1.18 Function magma\_cgeqrf

```
int magma_cgeqrf(int *m, int *n, float2 *a, int *lda, float2 *tau,
                float2 *work, int *lwork, float2 *da, int *info )
```

CGEQRF computes a QR factorization of a complex M-by-N matrix A:  $A = Q * R$ .

**M** (input) INTEGER  
The number of rows of the matrix A.  $M \geq 0$ .

**N** (input) INTEGER  
The number of columns of the matrix A.  $N \geq 0$ .

**A** (input/output) COMPLEX array, dimension (LDA,N)  
On entry, the M-by-N matrix A.  
On exit, the elements on and above the diagonal of the array contain the  $\min(M,N)$ -by-N upper trapezoidal matrix R (R is upper triangular if  $m \geq n$ ); the elements below the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of  $\min(m,n)$  elementary reflectors.  
Higher performance is achieved if A is in pinned memory, e.g. allocated using cudaMallocHost.

**LDA** (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1,M)$ .

**TAU** (output) COMPLEX array, dimension (min(M,N))  
The scalar factors of the elementary reflectors.

**WORK** (workspace/output) COMPLEX array, dimension (MAX(1,LWORK))  
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.  
Higher performance is achieved if WORK is in pinned memory, e.g. allocated using cudaMallocHost.

**LWORK** (input) INTEGER  
The dimension of the array WORK.  $LWORK \geq N * NB$ , where NB can be obtained through magma\_get\_cgeqrf\_nb(M).  
  
If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued.

**DA** (workspace) COMPLEX array on the GPU, dimension  $N * (M + NB)$ , where NB can be obtained through magma\_get\_cgeqrf\_nb(M). (size to be reduced in upcoming versions).

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value

The matrix Q is represented as a product of elementary reflectors

$Q = H(1) H(2) \dots H(k)$ , where  $k = \min(m,n)$ .

Each  $H(i)$  has the form

$H(i) = I - \tau * v * v'$

where tau is a complex scalar, and v is a complex vector with  $v(1:i-1) = 0$  and  $v(i) = 1$ ;  $v(i+1:m)$  is stored on exit in  $A(i+1:m,i)$ , and tau in TAU(i).

### 1.1.19 Function magma\_cpotrf

```
int magma_cpotrf(char *uplo, int *n, float2 *a, int *lda, float2 *work,
                int *info)
```

CPOTRF computes the Cholesky factorization of a complex Hermitian positive definite matrix A.

The factorization has the form

$A = U^{*T} * U$ , if UPLO = 'U', or

$A = L * L^{*T}$ , if UPLO = 'L',

where U is an upper triangular matrix and L is lower triangular.

This is the block version of the algorithm, calling Level 3 BLAS.

UPLO (input) CHARACTER\*1

= 'U': Upper triangle of A is stored;

= 'L': Lower triangle of A is stored.

N (input) INTEGER

The order of the matrix A.  $N \geq 0$ .

A (input/output) COMPLEX array, dimension (LDA,N)

On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization  $A = U^{*T} * U$  or  $A = L * L^{*T}$ .

Higher performance is achieved if A is in pinned memory, e.g. allocated using cudaMallocHost.

LDA (input) INTEGER

The leading dimension of the array A.  $LDA \geq \max(1, N)$ .

WORK (workspace) COMPLEX array on the GPU, dimension (N, N)

(size to be reduced in upcoming versions).

INFO (output) INTEGER

= 0: successful exit

< 0: if INFO = -i, the i-th argument had an illegal value

> 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

### 1.1.20 Function magma\_cgetrf\_gpu

```
int magma_cgetrf_gpu(int *m, int *n, float2 *a, int *lda,
                    int *ipiv, float2 *work, int *info)
```

CGETRF computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.

The factorization has the form

$$A = P * L * U$$

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if  $m > n$ ), and U is upper triangular (upper trapezoidal if  $m < n$ ).

This is the right-looking Level 3 BLAS version of the algorithm.

**M** (input) INTEGER  
The number of rows of the matrix A.  $M \geq 0$ .

**N** (input) INTEGER  
The number of columns of the matrix A.  $N \geq 0$ .

**A** (input/output) COMPLEX array on the GPU, dimension (LDA,N) where  $LDA \geq \max(M, N) + k_1$ ,  $k_1 < 32$  such that  $(\max(M, N) + k_1) \% 32 == 0$ .  
The memory pointed by A should be at least  $(\max(M, N) + k_1)^2 + (M + k_2) * NB + 2 * NB^2$  where  $k_2 < 32$  such that  $(M + k_2) \% 32 == 0$ .  
  
On entry, the M-by-N matrix to be factored.  
On exit, the factors L and U from the factorization  $A = P * L * U$ ; the unit diagonal elements of L are not stored.  
The rest of A is considered work space and is changed.

**LDA** (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1, M)$ .

**IPIV** (output) INTEGER array, dimension (min(M,N))  
The pivot indices; for  $1 \leq i \leq \min(M, N)$ , row i of the matrix was interchanged with row IPIV(i).

**WORK** (workspace/output) COMPLEX array, dimension  $\geq N * NB$ , where NB can be obtained through `magma_get_cgetrf_nb(M)`.  
Higher performance is achieved if WORK is in pinned memory, e.g. allocated using `cudaMallocHost`.

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value  
> 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

### 1.1.21 Function magma\_cgeqrf\_gpu

```
int magma_cgeqrf_gpu(int *m, int *n, float2 *a, int *lda, float2 *tau,
                    float2 *work, int *lwork, float2 *dwork, int *info )
```

CGEQRF computes a QR factorization of a complex M-by-N matrix A:  $A = Q * R$ .

**M** (input) INTEGER  
The number of rows of the matrix A.  $M \geq 0$ .

**N** (input) INTEGER  
The number of columns of the matrix A.  $N \geq 0$ .

**A** (input/output) COMPLEX array on the GPU, dimension (LDA,N)  
On entry, the M-by-N matrix A.  
On exit, the elements on and above the diagonal of the array contain the  $\min(M,N)$ -by-N upper trapezoidal matrix R (R is upper triangular if  $m \geq n$ ); the elements below the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of  $\min(m,n)$  elementary reflectors.

**LDA** (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1,M)$ .

**TAU** (output) COMPLEX array, dimension (min(M,N))  
The scalar factors of the elementary reflectors (see Further Details).

**WORK** (workspace/output) COMPLEX array, dimension (MAX(1,LWORK))  
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.  
Higher performance is achieved if WORK is in pinned memory, e.g. allocated using cudaMallocHost.

**LWORK** (input) INTEGER  
The dimension of the array WORK.  $LWORK \geq (M+N)*NB$ , where NB can be obtained through magma\_get\_cgeqrf\_nb(M).  
  
If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued.

**DWORK** (workspace) COMPLEX array on the GPU, dimension N\*NB,  
where NB can be obtained through magma\_get\_cgeqrf\_nb(M).

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value

The matrix Q is represented as a product of elementary reflectors

$Q = H(1) H(2) \dots H(k)$ , where  $k = \min(m,n)$ .

Each H(i) has the form

$H(i) = I - \tau * v * v'$

where tau is a complex scalar, and v is a complex vector with  $v(1:i-1) = 0$  and  $v(i) = 1$ ;  $v(i+1:m)$  is stored on exit in  $A(i+1:m,i)$ , and tau in TAU(i).

### 1.1.22 Function magma\_cpotrf\_gpu

```
int magma_cpotrf_gpu(char *uplo, int *n, float2 *a, int *lda,
                    float2 *work, int *info)
```

CPOTRF computes the Cholesky factorization of a complex Hermitian positive definite matrix A.

The factorization has the form

$A = U^{*T} * U$ , if UPLO = 'U', or

$A = L * L^{*T}$ , if UPLO = 'L',

where U is an upper triangular matrix and L is lower triangular.

This is the block version of the algorithm, calling Level 3 BLAS.

UPLO (input) CHARACTER\*1  
 = 'U': Upper triangle of A is stored;  
 = 'L': Lower triangle of A is stored.

N (input) INTEGER  
 The order of the matrix A.  $N \geq 0$ .

A (input/output) COMPLEX array on the GPU, dimension (LDA,N)  
 On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization  $A = U^{*T} * U$  or  $A = L * L^{*T}$ .

LDA (input) INTEGER  
 The leading dimension of the array A.  $LDA \geq \max(1, N)$ .

WORK (workspace) COMPLEX array, dimension at least (nb, nb)  
 where nb can be obtained through `magma_get_cpotrf_nb(*n)`  
 Work array allocated with `cudaMallocHost`.

INFO (output) INTEGER  
 = 0: successful exit  
 < 0: if INFO = -i, the i-th argument had an illegal value  
 > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

### 1.1.23 Function magma\_zgetrf

```
int magma_zgetrf(int *m, int *n, double2 *a, int *lda,
                int *ipiv, double2 *work, double2 *da, int *info)
```

ZGETRF computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.

The factorization has the form

$$A = P * L * U$$

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if  $m > n$ ), and U is upper triangular (upper trapezoidal if  $m < n$ ).

This is the right-looking Level 3 BLAS version of the algorithm.

**M** (input) INTEGER  
The number of rows of the matrix A.  $M \geq 0$ .

**N** (input) INTEGER  
The number of columns of the matrix A.  $N \geq 0$ .

**A** (input/output) DOUBLE COMPLEX array, dimension (LDA,N)  
On entry, the M-by-N matrix to be factored.  
On exit, the factors L and U from the factorization  $A = P*L*U$ ; the unit diagonal elements of L are not stored.  
Higher performance is achieved if A is in pinned memory, e.g. allocated using cudaMallocHost.

**LDA** (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1,M)$ .

**IPIV** (output) INTEGER array, dimension (min(M,N))  
The pivot indices; for  $1 \leq i \leq \min(M,N)$ , row i of the matrix was interchanged with row IPIV(i).

**WORK** (workspace/output) DOUBLE COMPLEX array, dimension  $\geq N*NB$ , where NB can be obtained through magma\_get\_cgetrf\_nb(M).  
Higher performance is achieved if WORK is in pinned memory, e.g. allocated using cudaMallocHost.

**DA** (workspace) DOUBLE COMPLEX array on the GPU, dimension  $(\max(M, N) + k1)^2 + (M + k2)*NB + 2*NB^2$ , where NB can be obtained through magma\_get\_cgetrf\_nb(M).  
 $k1 < 32$  and  $k2 < 32$  are such that  $(\max(M, N) + k1)\%32==0$  and  $(M+k2)\%32==0$ .

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value  
> 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

### 1.1.24 Function magma\_zgeqrf

```
int magma_zgeqrf(int *m, int *n, double2 *a, int *lda, double2 *tau,
                double2 *work, int *lwork, double2 *da, int *info )
```

ZGEQRF computes a QR factorization of a complex M-by-N matrix A:  $A = Q * R$ .

**M** (input) INTEGER  
The number of rows of the matrix A.  $M \geq 0$ .

**N** (input) INTEGER  
The number of columns of the matrix A.  $N \geq 0$ .

**A** (input/output) DOUBLE COMPLEX array, dimension (LDA,N)  
On entry, the M-by-N matrix A.  
On exit, the elements on and above the diagonal of the array contain the min(M,N)-by-N upper trapezoidal matrix R (R is upper triangular if  $m \geq n$ ); the elements below the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of min(m,n) elementary reflectors.  
Higher performance is achieved if A is in pinned memory, e.g. allocated using cudaMallocHost.

**LDA** (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1,M)$ .

**TAU** (output) DOUBLE COMPLEX array, dimension (min(M,N))  
The scalar factors of the elementary reflectors.

**WORK** (workspace/output) DOUBLE COMPLEX array, dimension (MAX(1,LWORK))  
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.  
Higher performance is achieved if WORK is in pinned memory, e.g. allocated using cudaMallocHost.

**LWORK** (input) INTEGER  
The dimension of the array WORK.  $LWORK \geq N * NB$ , where NB can be obtained through magma\_get\_zgeqrf\_nb(M).  
  
If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued.

**DA** (workspace) DOUBLE COMPLEX array on the GPU, dimension  $N * (M + NB)$ , where NB can be obtained through magma\_get\_zgeqrf\_nb(M). (size to be reduced in upcoming versions).

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value

The matrix Q is represented as a product of elementary reflectors

$Q = H(1) H(2) \dots H(k)$ , where  $k = \min(m,n)$ .

Each H(i) has the form

$H(i) = I - \tau * v * v'$

where tau is a complex scalar, and v is a complex vector with  $v(1:i-1) = 0$  and  $v(i) = 1$ ;  $v(i+1:m)$  is stored on exit in  $A(i+1:m,i)$ , and tau in TAU(i).

### 1.1.25 Function magma\_zpotrf

```
int magma_zpotrf(char *uplo, int *n, double2 *a, int *lda, double2 *work,
                int *info)
```

ZPOTRF computes the Cholesky factorization of a complex Hermitian positive definite matrix A.

The factorization has the form

$A = U^{*T} * U$ , if UPLO = 'U', or

$A = L * L^{*T}$ , if UPLO = 'L',

where U is an upper triangular matrix and L is lower triangular.

This is the block version of the algorithm, calling Level 3 BLAS.

UPLO (input) CHARACTER\*1

= 'U': Upper triangle of A is stored;

= 'L': Lower triangle of A is stored.

N (input) INTEGER

The order of the matrix A.  $N \geq 0$ .

A (input/output) DOUBLE COMPLEX array, dimension (LDA,N)

On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization  $A = U^{*T} * U$  or  $A = L * L^{*T}$ .

Higher performance is achieved if A is in pinned memory, e.g. allocated using cudaMallocHost.

LDA (input) INTEGER

The leading dimension of the array A.  $LDA \geq \max(1, N)$ .

WORK (workspace) DOUBLE COMPLEX array on the GPU, dimension (N, N)

(size to be reduced in upcoming versions).

INFO (output) INTEGER

= 0: successful exit

< 0: if INFO = -i, the i-th argument had an illegal value

> 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.



### 1.1.26 Function magma\_zgetrf\_gpu

```
int magma_zgetrf_gpu(int *m, int *n, double2 *a, int *lda,
                    int *ipiv, double2 *work, int *info)
```

ZGETRF computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.

The factorization has the form

$$A = P * L * U$$

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if  $m > n$ ), and U is upper triangular (upper trapezoidal if  $m < n$ ).

This is the right-looking Level 3 BLAS version of the algorithm.

**M** (input) INTEGER  
The number of rows of the matrix A.  $M \geq 0$ .

**N** (input) INTEGER  
The number of columns of the matrix A.  $N \geq 0$ .

**A** (input/output) DOUBLE COMPLEX array on the GPU, dimension (LDA,N) where  $LDA \geq \max(M, N) + k_1$ ,  $k_1 < 32$  such that  $(\max(M, N) + k_1) \% 32 == 0$ .  
The memory pointed by A should be at least  $(\max(M, N) + k_1)^2 + (M + k_2) * NB + 2 * NB^2$  where  $k_2 < 32$  such that  $(M + k_2) \% 32 == 0$ .  
  
On entry, the M-by-N matrix to be factored.  
On exit, the factors L and U from the factorization  $A = P * L * U$ ; the unit diagonal elements of L are not stored.  
The rest of A is considered work space and is changed.

**LDA** (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1, M)$ .

**IPIV** (output) INTEGER array, dimension (min(M,N))  
The pivot indices; for  $1 \leq i \leq \min(M, N)$ , row i of the matrix was interchanged with row IPIV(i).

**WORK** (workspace/output) DOUBLE COMPLEX array, dimension  $\geq N * NB$ , where NB can be obtained through `magma_get_zgetrf_nb(M)`.  
Higher performance is achieved if WORK is in pinned memory, e.g. allocated using `cudaMallocHost`.

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value  
> 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

### 1.1.27 Function magma\_zgeqrf\_gpu

```
int magma_zgeqrf_gpu(int *m, int *n, double2 *a, int *lda, double2 *tau,
                    double2 *work, int *lwork, double2 *dwork, int *info )
```

ZGEQRF computes a QR factorization of a complex M-by-N matrix A:  $A = Q * R$ .

**M** (input) INTEGER  
The number of rows of the matrix A.  $M \geq 0$ .

**N** (input) INTEGER  
The number of columns of the matrix A.  $N \geq 0$ .

**A** (input/output) DOUBLE COMPLEX array on the GPU, dimension (LDA,N)  
On entry, the M-by-N matrix A.  
On exit, the elements on and above the diagonal of the array contain the  $\min(M,N)$ -by-N upper trapezoidal matrix R (R is upper triangular if  $m \geq n$ ); the elements below the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of  $\min(m,n)$  elementary reflectors.

**LDA** (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1,M)$ .

**TAU** (output) DOUBLE COMPLEX array, dimension (min(M,N))  
The scalar factors of the elementary reflectors.

**WORK** (workspace/output) DOUBLE COMPLEX array, dimension (MAX(1,LWORK))  
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.  
Higher performance is achieved if WORK is in pinned memory, e.g. allocated using cudaMallocHost.

**LWORK** (input) INTEGER  
The dimension of the array WORK.  $LWORK \geq (M+N)*NB$ , where NB can be obtained through magma\_get\_zgeqrf\_nb(M).  
  
If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued.

**DWORK** (workspace) DOUBLE COMPLEX array on the GPU, dimension N\*NB, where NB can be obtained through magma\_get\_zgeqrf\_nb(M).

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value

The matrix Q is represented as a product of elementary reflectors

$$Q = H(1) H(2) \dots H(k), \text{ where } k = \min(m,n).$$

Each H(i) has the form

$$H(i) = I - \tau * v * v'$$

where tau is a complex scalar, and v is a complex vector with  $v(1:i-1) = 0$  and  $v(i) = 1$ ;  $v(i+1:m)$  is stored on exit in  $A(i+1:m,i)$ , and tau in TAU(i).

### 1.1.28 Function magma\_zpotrf\_gpu

```
int magma_zpotrf_gpu(char *uplo, int *n, double2 *a, int *lda, double2 *work,
                    int *info)
```

ZPOTRF computes the Cholesky factorization of a complex Hermitian positive definite matrix A.

The factorization has the form

$A = U^{*T} * U$ , if UPLO = 'U', or

$A = L * L^{*T}$ , if UPLO = 'L',

where U is an upper triangular matrix and L is lower triangular.

This is the block version of the algorithm, calling Level 3 BLAS.

UPLO (input) CHARACTER\*1  
 = 'U': Upper triangle of A is stored;  
 = 'L': Lower triangle of A is stored.

N (input) INTEGER  
 The order of the matrix A.  $N \geq 0$ .

A (input/output) DOUBLE COMPLEX array on the GPU, dimension (LDA,N)  
 On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if INFO = 0, the factor U or L from the Cholesky factorization  $A = U^{*T} * U$  or  $A = L * L^{*T}$ .

LDA (input) INTEGER  
 The leading dimension of the array A.  $LDA \geq \max(1, N)$ .

WORK (workspace) DOUBLE COMPLEX array, dimension at least (nb, nb) where nb can be obtained through `magma_get_zpotrf_nb(*n)`  
 Work array allocated with `cudaMallocHost`.

INFO (output) INTEGER  
 = 0: successful exit  
 < 0: if INFO = -i, the i-th argument had an illegal value  
 > 0: if INFO = i, the leading minor of order i is not positive definite, and the factorization could not be completed.

## 1.2 Linear solvers

Provided are solvers in real arithmetic, both single and double precision, based on the LU, QR, and Cholesky factorizations. To solve

$$Ax = b$$

the matrix  $A$  is first factored, and second, the resulting factors are used in solving the original problem. A general recommendation is to use LU for general  $n \times n$  matrices, Cholesky for symmetric and positive definite  $n \times n$  matrices, and QR for solving least squares problems

$$\min ||Ax - b||$$

for general  $m \times n$ ,  $m \geq n$  matrices.

The solvers are provided in the more critical GPU interface. Indeed, if a user has used the CPU interface for the matrix factorization, it would be often faster to do the solving step directly on the CPU using LAPACK with optimized CPU BLAS – the solving step is a bandwidth limited (Level 2 BLAS) operation and currently CPU's bus is of higher bandwidth than the bandwidth of the CPU-GPU connection (i.e. the time to solve on the CPU would be faster than just transferring the entire matrix to the GPU).

Although the solution step has  $O(n) \times$  less floating point operations than the factorization, it is still very important to optimize it. Solving a triangular system of equations can be very slow because the computation is bandwidth limited and naturally not parallel. Various approaches have been proposed in the past. We use an approach where diagonal blocks of  $A$  are explicitly inverted and used in a block algorithm. This results in a numerically stable algorithm, especially when used with triangular matrices coming from numerically stable factorization algorithms (e.g. as in LAPACK and as implemented here in MAGMA), of high performance, e.g. often exceeding  $100 \times$  the performance of the corresponding CUBLAS implementations.

To take advantage of the fact that GPU's single precision is currently of much higher performance than the double precision (theoretically  $\approx 10 \times$ ), MAGMA version 0.2 provides a second set of solvers, based on the mixed precision iterative refinement technique. The solvers are based again on correspondingly the LU, QR, and Cholesky factorizations, and are designed to solve linear problems in double precision accuracy but at a speed that is characteristic for the much faster single precision computations. The idea is to use single precision for the bulk of the computation, namely the factorization step, and then use that factorization as a preconditioner in a simple iterative refinement process in double precision arithmetic. This often results in the desired high performance and high accuracy solvers.

### 1.2.1 Function magma\_sgetrs\_gpu

```
int magma_sgetrs_gpu(char *trans , int n, int nrhs, float *a , int lda,
                    int *ipiv, float *b, int ldb, int *info, float *hwork)
```

Solves a system of linear equations

$A * X = B$  or  $A' * X = B$

with a general N-by-N matrix A using the LU factorization computed by SGETRF\_GPU.

TRANS (input) CHARACTER\*1

Specifies the form of the system of equations:

= 'N':  $A * X = B$  (No transpose)

= 'T':  $A' * X = B$  (Transpose)

= 'C':  $A' * X = B$  (Conjugate transpose = Transpose)

N (input) INTEGER

The order of the matrix A.  $N \geq 0$ .

NRHS (input) INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B.  $NRHS \geq 0$ .

A (input) REAL array on the GPU, dimension (LDA,N)

The factors L and U from the factorization  $A = P * L * U$  as computed by SGETRF\_GPU.

LDA (input) INTEGER

The leading dimension of the array A.  $LDA \geq \max(1, N)$ .

IPIV (input) INTEGER array, dimension (N)

The pivot indices from SGETRF; for  $1 \leq i \leq N$ , row i of the matrix was interchanged with row IPIV(i).

B (input/output) REAL array on the GPU, dimension (LDB,NRHS)

On entry, the right hand side matrix B.

On exit, the solution matrix X.

LDB (input) INTEGER

The leading dimension of the array B.  $LDB \geq \max(1, N)$ .

INFO (output) INTEGER

= 0: successful exit

< 0: if INFO = -i, the i-th argument had an illegal value

HWORK (workspace) REAL array, dimension N\*NRHS

## 1.2.2 Function magma\_sgeqrs\_gpu

```
int magma_sgeqrs_gpu(int *m, int *n, int *nrhs,
                    float *a, int *lda, float *tau, float *c, int *ldc,
                    float *work, int *lwork, float *td, int *info)
```

Solves the least squares problem

$\min || A \cdot X - C ||$

using the QR factorization  $A = Q \cdot R$  computed by SGEQRF\_GPU2.

**M** (input) INTEGER  
The number of rows of the matrix A.  $M \geq 0$ .

**N** (input) INTEGER  
The number of columns of the matrix A.  $M \geq N \geq 0$ .

**NRHS** (input) INTEGER  
The number of columns of the matrix C.  $NRHS \geq 0$ .

**A** (input) REAL array on the GPU, dimension (LDA,N)  
The i-th column must contain the vector which defines the elementary reflector  $H(i)$ , for  $i = 1, 2, \dots, n$ , as returned by SGEQRF\_GPU2 in the first n columns of its array argument A.

**LDA** (input) INTEGER  
The leading dimension of the array A,  $LDA \geq M$ .

**TAU** (input) REAL array, dimension (N)  
TAU(i) must contain the scalar factor of the elementary reflector  $H(i)$ , as returned by MAGMA\_SGEQRF\_GPU2.

**C** (input/output) REAL array on the GPU, dimension (LDC,NRHS)  
On entry, the M-by-NRHS matrix C.  
On exit, the N-by-NRHS solution matrix X.

**LDC** (input) INTEGER  
The leading dimension of the array C.  $LDC \geq M$ .

**WORK** (workspace/output) REAL array, dimension (LWORK)  
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

**LWORK** (input) INTEGER  
The dimension of the array WORK,  $LWORK \geq \max(1, NRHS)$ .  
For optimum performance  $LWORK \geq (M - N + NB + 2 * NRHS) * NB$ , where NB is the blocksize given by magma\_get\_sgeqrf\_nb( M ).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array.

**TD** (input) REAL array that is the output (the 9th argument) of magma\_sgeqrf\_gpu2.

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value

### 1.2.3 Function magma\_spotrs\_gpu

```
int magma_spotrs_gpu(char *UPLO, int N , int NRHS, float *A , int LDA,
                    float *B, int LDB, int *INFO)
```

Solves a system of linear equations  $A \cdot X = B$  with a symmetric positive definite matrix  $A$  using the Cholesky factorization  $A = U \cdot U^T$  or  $A = L \cdot L^T$  computed by SPOTRF\_GPU.

UPLO (input) CHARACTER\*1  
 = 'U': Upper triangle of  $A$  is stored;  
 = 'L': Lower triangle of  $A$  is stored.

N (input) INTEGER  
 The order of the matrix  $A$ .  $N \geq 0$ .

NRHS (input) INTEGER  
 The number of right hand sides, i.e., the number of columns of the matrix  $B$ .  $NRHS \geq 0$ .

A (input) REAL array on the GPU, dimension (LDA,N)  
 The triangular factor  $U$  or  $L$  from the Cholesky factorization  $A = U \cdot U^T$  or  $A = L \cdot L^T$ , as computed by SPOTRF.

LDA (input) INTEGER  
 The leading dimension of the array  $A$ .  $LDA \geq \max(1, N)$ .

B (input/output) REAL array on the GPU, dimension (LDB,NRHS)  
 On entry, the right hand side matrix  $B$ .  
 On exit, the solution matrix  $X$ .

LDB (input) INTEGER  
 The leading dimension of the array  $B$ .  $LDB \geq \max(1, N)$ .

INFO (output) INTEGER  
 = 0: successful exit  
 < 0: if  $INFO = -i$ , the  $i$ -th argument had an illegal value

### 1.2.4 Function magma\_dgetrs\_gpu

```
int magma_dgetrs_gpu(char *trans , int n, int nrhs, double *a , int lda,
                    int *ipiv, double *b, int ldb, int *info, double *hwork)
```

Solves a system of linear equations

$A * X = B$  or  $A' * X = B$

with a general N-by-N matrix A using the LU factorization computed by SGETRF\_GPU.

TRANS (input) CHARACTER\*1

Specifies the form of the system of equations:

= 'N':  $A * X = B$  (No transpose)

= 'T':  $A' * X = B$  (Transpose)

= 'C':  $A' * X = B$  (Conjugate transpose = Transpose)

N (input) INTEGER

The order of the matrix A.  $N \geq 0$ .

NRHS (input) INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B.  $NRHS \geq 0$ .

A (input) DOUBLE array on the GPU, dimension (LDA,N)

The factors L and U from the factorization  $A = P * L * U$  as computed by SGETRF\_GPU.

LDA (input) INTEGER

The leading dimension of the array A.  $LDA \geq \max(1, N)$ .

IPIV (input) INTEGER array, dimension (N)

The pivot indices from SGETRF; for  $1 \leq i \leq N$ , row i of the matrix was interchanged with row IPIV(i).

B (input/output) DOUBLE array on the GPU, dimension (LDB, NRHS)

On entry, the right hand side matrix B.

On exit, the solution matrix X.

LDB (input) INTEGER

The leading dimension of the array B.  $LDB \geq \max(1, N)$ .

INFO (output) INTEGER

= 0: successful exit

< 0: if INFO = -i, the i-th argument had an illegal value

HWORK (workspace) DOUBLE array, dimension N\*NRHS



### 1.2.5 Function magma\_dgeqrs\_gpu

```
int magma_dgeqrs_gpu(int *m, int *n, int *nrhs,
                    double *a, int *lda, double *tau, double *c, int *ldc,
                    double *work, int *lwork, double *td, int *info)
```

Solves the least squares problem

$\min || A \cdot X - C ||$

using the QR factorization  $A = Q \cdot R$  computed by SGEQRF\_GPU2.

**M** (input) INTEGER  
The number of rows of the matrix A.  $M \geq 0$ .

**N** (input) INTEGER  
The number of columns of the matrix A.  $M \geq N \geq 0$ .

**NRHS** (input) INTEGER  
The number of columns of the matrix C.  $NRHS \geq 0$ .

**A** (input) DOUBLE array on the GPU, dimension (LDA,N)  
The i-th column must contain the vector which defines the elementary reflector  $H(i)$ , for  $i = 1, 2, \dots, n$ , as returned by SGEQRF\_GPU2 in the first n columns of its array argument A.

**LDA** (input) INTEGER  
The leading dimension of the array A,  $LDA \geq M$ .

**TAU** (input) DOUBLE array, dimension (N)  
TAU(i) must contain the scalar factor of the elementary reflector  $H(i)$ , as returned by MAGMA\_DGEQRF\_GPU2.

**C** (input/output) DOUBLE array on the GPU, dimension (LDC,NRHS)  
On entry, the M-by-NRHS matrix C.  
On exit, the N-by-NRHS solution matrix X.

**LDC** (input) INTEGER  
The leading dimension of the array C.  $LDC \geq M$ .

**WORK** (workspace/output) DOUBLE array, dimension (LWORK)  
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

**LWORK** (input) INTEGER  
The dimension of the array WORK,  $LWORK \geq \max(1, NRHS)$ .  
For optimum performance  $LWORK \geq (M - N + NB + 2 * NRHS) * NB$ , where NB is the blocksize given by magma\_get\_sgeqrf\_nb( M ).

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array.

**TD** (input) DOUBLE array that is the output (the 9th argument) of magma\_dgeqrf\_gpu2.

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value

## 1.2.6 Function magma\_dpotrs\_gpu

```
int magma_dpotrs_gpu(char *UPLO, int N , int NRHS, double *A , int LDA,
                    double *B, int LDB, int *INFO)
```

Solves a system of linear equations  $A \cdot X = B$  with a symmetric positive definite matrix  $A$  using the Cholesky factorization  $A = U^* \cdot U$  or  $A = L \cdot L^* \cdot T$  computed by SPOTRF\_GPU.

UPLO    (input) CHARACTER\*1  
          = 'U': Upper triangle of  $A$  is stored;  
          = 'L': Lower triangle of  $A$  is stored.

N        (input) INTEGER  
          The order of the matrix  $A$ .  $N \geq 0$ .

NRHS    (input) INTEGER  
          The number of right hand sides, i.e., the number of columns of the matrix  $B$ .  $NRHS \geq 0$ .

A        (input) DOUBLE array on the GPU, dimension (LDA,N)  
          The triangular factor  $U$  or  $L$  from the Cholesky factorization  $A = U^* \cdot U$  or  $A = L \cdot L^* \cdot T$ , as computed by SPOTRF.

LDA      (input) INTEGER  
          The leading dimension of the array  $A$ .  $LDA \geq \max(1, N)$ .

B        (input/output) DOUBLE array on the GPU, dimension (LDB, NRHS)  
          On entry, the right hand side matrix  $B$ .  
          On exit, the solution matrix  $X$ .

LDB      (input) INTEGER  
          The leading dimension of the array  $B$ .  $LDB \geq \max(1, N)$ .

INFO     (output) INTEGER  
          = 0: successful exit  
          < 0: if  $INFO = -i$ , the  $i$ -th argument had an illegal value

### 1.2.7 Function magma\_dsgesv\_gpu

```
int magma_dsgesv_gpu(int N, int NRHS, double *A, int LDA, int *IPIV, double *B,
                    int LDB, double *X, int LDX, double *WORK, float *SWORK,
                    int *ITER, int *INFO, float *H_SWORK, double *H_WORK,
                    int *DIPIV)
```

Computes the solution to a real system of linear equations

$A * X = B$ ,

where A is an N-by-N matrix and X and B are N-by-NRHS matrices.

DSGESV first attempts to factorize the matrix in SINGLE PRECISION and use this factorization within an iterative refinement procedure to produce a solution with DOUBLE PRECISION normwise backward error quality (see below). If the approach fails the method switches to a DOUBLE PRECISION factorization and solve.

The iterative refinement is not going to be a winning strategy if the ratio SINGLE PRECISION performance over DOUBLE PRECISION performance is too small. A reasonable strategy should take the number of right-hand sides and the size of the matrix into account. This might be done with a call to ILAENV in the future. Up to now, we always try iterative refinement.

The iterative refinement process is stopped if

$ITER > ITERMAX$

or for all the RHS we have:

$RNRM < \sqrt{N} * XNRM * ANRM * EPS * BWDMAX$

where

- o ITER is the number of the current iteration in the iterative refinement process
- o RNRM is the infinity-norm of the residual
- o XNRM is the infinity-norm of the solution
- o ANRM is the infinity-operator-norm of the matrix A
- o EPS is the machine epsilon returned by DLAMCH('Epsilon')

The value ITERMAX and BWDMAX are fixed to 30 and 1.0D+00 respectively.

- N (input) INTEGER  
The number of linear equations, i.e., the order of the matrix A.  $N \geq 0$ .
- NRHS (input) INTEGER  
The number of right hand sides, i.e., the number of columns of the matrix B.  $NRHS \geq 0$ .
- A (input or input/output) DOUBLE PRECISION array, dimension (LDA,N)  
On entry, the N-by-N coefficient matrix A.  
On exit, if iterative refinement has been successfully used (INFO.EQ.0 and ITER.GE.0, see description below), A is unchanged. If double precision factorization has been used (INFO.EQ.0 and ITER.LT.0, see description below), then the array A contains the factors L and U from the factorization  $A = P * L * U$ ; the unit diagonal elements of L are not stored.
- LDA (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1, N)$ .
- IPIV (output) INTEGER array, dimension (N)  
The pivot indices that define the permutation matrix P;

row  $i$  of the matrix was interchanged with row  $\text{IPIV}(i)$ .  
 Corresponds either to the single precision factorization  
 (if  $\text{INFO.EQ.0}$  and  $\text{ITER.GE.0}$ ) or the double precision  
 factorization (if  $\text{INFO.EQ.0}$  and  $\text{ITER.LT.0}$ ).

**B** (input) DOUBLE PRECISION array, dimension (LDB,NRHS)  
 The N-by-NRHS right hand side matrix B.

**LDB** (input) INTEGER  
 The leading dimension of the array B.  $\text{LDB} \geq \max(1, N)$ .

**X** (output) DOUBLE PRECISION array, dimension (LDX,NRHS)  
 If  $\text{INFO} = 0$ , the N-by-NRHS solution matrix X.

**LDX** (input) INTEGER  
 The leading dimension of the array X.  $\text{LDX} \geq \max(1, N)$ .

**WORK** (workspace) DOUBLE PRECISION array, dimension (N\*NRHS)  
 This array is used to hold the residual vectors.

**SWORK** (workspace) REAL array, dimension (N\*(N+NRHS))  
 This array is used to store the single precision matrix and the  
 right-hand sides or solutions in single precision.

**ITER** (output) INTEGER  
 < 0: iterative refinement has failed, double precision  
 factorization has been performed  
 -1 : the routine fell back to full precision for  
 implementation- or machine-specific reasons  
 -2 : narrowing the precision induced an overflow,  
 the routine fell back to full precision  
 -3 : failure of SGETRF  
 -31: stop the iterative refinement after the 30th  
 iterations  
 > 0: iterative refinement has been successfully used.  
 Returns the number of iterations

**INFO** (output) INTEGER  
 = 0: successful exit  
 < 0: if  $\text{INFO} = -i$ , the  $i$ -th argument had an illegal value  
 > 0: if  $\text{INFO} = i$ ,  $U(i,i)$  computed in DOUBLE PRECISION is  
 exactly zero. The factorization has been completed,  
 but the factor U is exactly singular, so the solution  
 could not be computed.

**H\_SWORK** (workspace) REAL array, dimension at least (nb, nb)  
 where nb can be obtained through `magma_get_sgetrf_nb(*n)`  
 Work array allocated with `cudaMallocHost`.

**H\_WORK** (workspace) DOUBLE array, dimension at least (nb, nb)  
 where nb can be obtained through `magma_get_dgetrf_nb(*n)`  
 Work array allocated with `cudaMallocHost`.

**DIPIV** (output) INTEGER array on the GPU, dimension (min(M,N))  
 The pivot indices; for  $1 \leq i \leq \min(M, N)$ , row  $i$  of the  
 matrix was moved to row  $\text{IPIV}(i)$ .

## 1.2.8 Function magma\_dsgeqrsv\_gpu

```
int magma_dsgeqrsv_gpu(int M, int N, int NRHS, double *A, int LDA, double *B,
                      int LDB, double *X, int LDX, double *WORK, float *SWORK,
                      int *ITER, int *INFO, float *tau, int lwork, float *h_work,
                      float *d_work, double *tau_d, int lwork_d, double *h_work_d,
                      double *d_work_d)
```

DSGEQRSV solves the least squares problem

$$\min || A * X - B ||,$$

where A is an M-by-N matrix and X and B are M-by-NRHS matrices.

DSGEQRSV first attempts to factorize the matrix in SINGLE PRECISION and use this factorization within an iterative refinement procedure to produce a solution with DOUBLE PRECISION norm-wise backward error quality (see below). If the approach fails the method switches to a DOUBLE PRECISION factorization and solve.

The iterative refinement is not going to be a winning strategy if the ratio SINGLE PRECISION performance over DOUBLE PRECISION performance is too small. A reasonable strategy should take the number of right-hand sides and the size of the matrix into account. This might be done with a call to ILAENV in the future. Up to now, we always try iterative refinement.

The iterative refinement process is stopped if

$$ITER > ITERMAX$$

or for all the RHS we have:

$$RNRM < \sqrt{N} * XNRM * ANRM * EPS * BWDMAX$$

where

- o ITER is the number of the current iteration in the iterative refinement process
- o RNRM is the infinity-norm of the residual
- o XNRM is the infinity-norm of the solution
- o ANRM is the infinity-operator-norm of the matrix A
- o EPS is the machine epsilon returned by DLAMCH('Epsilon')

The value ITERMAX and BWDMAX are fixed to 30 and 1.0D+00 respectively.

- M (input) INTEGER  
The number of rows of the matrix A. M >= 0.
- N (input) INTEGER  
The number of columns of the matrix A. M >= N >= 0.
- NRHS (input) INTEGER  
The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.
- A (input or input/output) DOUBLE PRECISION array, dimension (LDA,N)  
On entry, the M-by-N coefficient matrix A.  
On exit, if iterative refinement has been successfully used (INFO.EQ.0 and ITER.GE.0, see description below), A is unchanged. If double precision factorization has been used (INFO.EQ.0 and ITER.LT.0, see description below), then the array A contains the QR factorization of A as returned by function DGEQRF\_GPU2.
- LDA (input) INTEGER

The leading dimension of the array A.  $LDA \geq \max(1, M)$ .

**B** (input) DOUBLE PRECISION array, dimension (LDB, NRHS)  
The M-by-NRHS right hand side matrix B.

**LDB** (input) INTEGER  
The leading dimension of the array B.  $LDB \geq \max(1, M)$ .

**X** (output) DOUBLE PRECISION array, dimension (LDX, NRHS)  
If INFO = 0, the N-by-NRHS solution matrix X.

**LDX** (input) INTEGER  
The leading dimension of the array X.  $LDX \geq \max(1, N)$ .

**WORK** (workspace) DOUBLE PRECISION array, dimension (N\*NRHS)  
This array is used to hold the residual vectors.

**SWORK** (workspace) REAL array, dimension (M\*(N+NRHS))  
This array is used to store the single precision matrix and the right-hand sides or solutions in single precision.

**ITER** (output) INTEGER  
 < 0: iterative refinement has failed, double precision factorization has been performed  
   -1 : the routine fell back to full precision for implementation- or machine-specific reasons  
   -2 : narrowing the precision induced an overflow, the routine fell back to full precision  
   -3 : failure of SGETRF  
  -31: stop the iterative refinement after the 30th iterations  
 > 0: iterative refinement has been successfully used.  
 Returns the number of iterations

**INFO** (output) INTEGER  
 = 0: successful exit  
 < 0: if INFO = -i, the i-th argument had an illegal value

**TAU** (output) REAL array, dimension (N)  
On exit, TAU(i) contains the scalar factor of the elementary reflector H(i), as returned by magma\_sgeqrf\_gpu2.

**LWORK** (input) INTEGER  
The dimension of the array H\_WORK.  $LWORK \geq (M+N+NB)*NB$ , where NB can be obtained through magma\_get\_sgeqrf\_nb(M).

**H\_WORK** (workspace/output) REAL array, dimension (MAX(1, LWORK))  
Higher performance is achieved if H\_WORK is in pinned memory, e.g. allocated using cudaMallocHost.

**D\_WORK** (workspace/output) REAL array on the GPU, dimension  $2*N*NB$ , where NB can be obtained through magma\_get\_sgeqrf\_nb(M). It starts with  $NB*NB$  blocks that store the triangular T matrices, followed by the  $NB*NB$  blocks of the diagonal inverses for the R matrix.

**TAU\_D** (output) DOUBLE REAL array, dimension (N)

On exit, if the matrix had to be factored in double precision, TAU(i) contains the scalar factor of the elementary reflector H(i), as returned by magma\_dgeqrf\_gpu2.

LWORK\_D (input) INTEGER

The dimension of the array H\_WORK\_D. LWORK\_D  $\geq (M+N+NB)*NB$ , where NB can be obtained through magma\_get\_dgeqrf\_nb(M).

H\_WORK\_D (workspace/output) DOUBLE REAL array, dimension (MAX(1,LWORK\_D))

This memory is unattached if the iterative refinement worked, otherwise it is used as workspace to factor the matrix in double precision. Higher performance is achieved if H\_WORK\_D is in pinned memory, e.g. allocated using cudaMallocHost.

D\_WORK\_D (workspace/output) DOUBLE REAL array on the GPU, dimension  $2*N*NB$ ,

where NB can be obtained through magma\_get\_dgeqrf\_nb(M). This memory is unattached if the iterative refinement worked, otherwise it is used as workspace to factor the matrix in double precision. It starts with  $NB*NB$  blocks that store the triangular T matrices, followed by the  $NB*NB$  blocks of the diagonal inverses for the R matrix.

## 1.2.9 Function magma\_dsposv\_gpu

```
int magma_dsposv_gpu(char UPLO, int N, int NRHS, double *A, int LDA, double *B,
                    int LDB, double *X, int LDX, double *WORK, float *SWORK,
                    int *ITER, int *INFO, float *H_SWORK, double *H_WORK)
```

DSPOSV computes the solution to a real system of linear equations

$$A * X = B,$$

where A is an N-by-N symmetric positive definite matrix and X and B are N-by-NRHS matrices.

DSPOSV first attempts to factorize the matrix in SINGLE PRECISION and use this factorization within an iterative refinement procedure to produce a solution with DOUBLE PRECISION norm-wise backward error quality (see below). If the approach fails the method switches to a DOUBLE PRECISION factorization and solve.

The iterative refinement is not going to be a winning strategy if the ratio SINGLE PRECISION performance over DOUBLE PRECISION performance is too small. A reasonable strategy should take the number of right-hand sides and the size of the matrix into account. This might be done with a call to ILAENV in the future. Up to now, we always try iterative refinement.

The iterative refinement process is stopped if

$$ITER > ITERMAX$$

or for all the RHS we have:

$$RNRM < \sqrt{N} * XNRM * ANRM * EPS * BWDMAX$$

where

- o ITER is the number of the current iteration in the iterative refinement process
- o RNRM is the infinity-norm of the residual
- o XNRM is the infinity-norm of the solution
- o ANRM is the infinity-operator-norm of the matrix A
- o EPS is the machine epsilon returned by DLAMCH('Epsilon')

The value ITERMAX and BWDMAX are fixed to 30 and 1.0D+00 respectively.

UPLO (input) CHARACTER

= 'U': Upper triangle of A is stored;  
 = 'L': Lower triangle of A is stored.

N (input) INTEGER

The number of linear equations, i.e., the order of the matrix A. N >= 0.

NRHS (input) INTEGER

The number of right hand sides, i.e., the number of columns of the matrix B. NRHS >= 0.

A (input or input/output) DOUBLE PRECISION array,  
 dimension (LDA,N)

On entry, the symmetric matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.



On exit, if iterative refinement has been successfully used (INFO.EQ.0 and ITER.GE.0, see description below), then A is unchanged, if double precision factorization has been used (INFO.EQ.0 and ITER.LT.0, see description below), then the array A contains the factor U or L from the Cholesky factorization  $A = U^*T^*U$  or  $A = L^*L^*T$ .

LDA (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1, N)$ .

B (input) DOUBLE PRECISION array, dimension (LDB, NRHS)  
The N-by-NRHS right hand side matrix B.

LDB (input) INTEGER  
The leading dimension of the array B.  $LDB \geq \max(1, N)$ .

X (output) DOUBLE PRECISION array, dimension (LDX, NRHS)  
If INFO = 0, the N-by-NRHS solution matrix X.

LDX (input) INTEGER  
The leading dimension of the array X.  $LDX \geq \max(1, N)$ .

WORK (workspace) DOUBLE PRECISION array, dimension (N\*NRHS)  
This array is used to hold the residual vectors.

SWORK (workspace) REAL array, dimension (N\*(N+NRHS))  
This array is used to use the single precision matrix and the right-hand sides or solutions in single precision.

ITER (output) INTEGER  
< 0: iterative refinement has failed, double precision factorization has been performed  
-1 : the routine fell back to full precision for implementation- or machine-specific reasons  
-2 : narrowing the precision induced an overflow, the routine fell back to full precision  
-3 : failure of SPOTRF  
-31: stop the iterative refinement after the 30th iterations  
> 0: iterative refinement has been successfully used.  
Returns the number of iterations

INFO (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value  
> 0: if INFO = i, the leading minor of order i of (DOUBLE PRECISION) A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

H\_SWORK (workspace) REAL array, dimension at least (nb, nb)  
where nb can be obtained through `magma_get_spotrf_nb(*n)`  
Work array allocated with `cudaMallocHost`.

H\_WORK (workspace) DOUBLE array, dimension at least (nb, nb)  
where nb can be obtained through `magma_get_dpotrf_nb(*n)`  
Work array allocated with `cudaMallocHost`.

### 1.3 Two-sided matrix factorizations

As the one-sided matrix factorizations are the bases for various linear solvers, the two-sided matrix factorizations are the bases for eigen-solvers, and therefore form an important class of dense linear algebra routines. The two-sided factorizations have been traditionally more difficult to handle/accelerate on current architectures. The reason is that the two-sided factorizations involve large matrix-vector products which are memory bound, and as the gap between compute and communication power increases exponentially, these memory bound operations become an increasingly more difficult to handle bottleneck. GPUs though offer an attractive possibility to accelerate them. Indeed, having a high bandwidth (e.g.  $10\times$  larger than current CPU bus bandwidths), GPUs can accelerate matrix-vector products significantly ( $10$  to  $30\times$ ). Having this important component, it's just a matter on how to organize the computation into a hybrid fashion. An approach similar to the one for one-sided factorizations is used (see Figure 1.2 for an illustration of the performance to be expected and [6] for further detail).

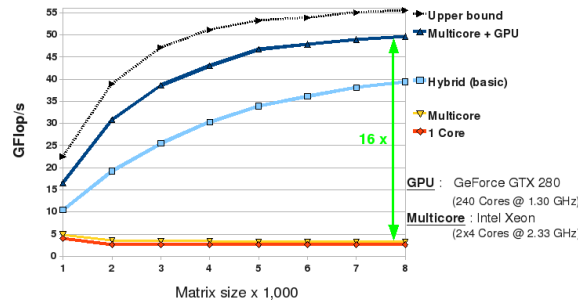


Figure 1.2: Performance (in double precision) for the hybrid reduction to upper Hessenberg form

This release provides implementations for the reduction to upper Hessenberg form in single and double precision real arithmetic. Various highly optimized matrix-vector multiplications have been developed as needed for the other two-sided factorizations. Combining these kernels with the hybrid approach from the Hessenberg reduction, indicates the rest of the two-sided factorizations can be similarly developed to yield large performance improvements (*vs* algorithms for just homogeneous multicore architectures).

### 1.3.1 Function magma\_sgehrd

```
int magma_sgehrd(int *n, int *ilo, int *ihi, float *a, int *lda,
                float *tau, float *work, int *lwork, float *da, int *info)
```

DGEHRD reduces a real general matrix A to upper Hessenberg form H by an orthogonal similarity transformation:  $Q' * A * Q = H$ .

**N** (input) INTEGER  
The order of the matrix A.  $N \geq 0$ .

**ILO** (input) INTEGER  
**IHI** (input) INTEGER  
It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to DGEBAL; otherwise they should be set to 1 and N respectively. See Further Details.  
 $1 \leq ILO \leq IHI \leq N$ , if  $N > 0$ ;  $ILO=1$  and  $IHI=0$ , if  $N=0$ .

**A** (input/output) SINGLE PRECISION array, dimension (LDA,N)  
On entry, the N-by-N general matrix to be reduced.  
On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the elements below the first subdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors. See Further Details.

**LDA** (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1, N)$ .

**TAU** (output) SINGLE PRECISION array, dimension (N-1)  
The scalar factors of the elementary reflectors (see Further Details). Elements 1:ILO-1 and IHI:N-1 of TAU are set to zero.

**WORK** (workspace/output) SINGLE PRECISION array, dimension (LWORK)  
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

**LWORK** (input) INTEGER  
The length of the array WORK.  $LWORK \geq \max(1, N)$ .  
For optimum performance  $LWORK \geq N \cdot NB$ , where NB is the optimal blocksize.  
  
If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

**DA** (workspace) SINGLE array on the GPU, dimension  $N \cdot N + 2 \cdot N \cdot NB + NB \cdot NB$ ,  
where NB can be obtained through magma\_get\_sgehrd\_nb(N).

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value.

Further Details  
=====

The matrix  $Q$  is represented as a product of  $(ihi-ilo)$  elementary reflectors

$$Q = H(ilo) H(ilo+1) \dots H(ihi-1).$$

Each  $H(i)$  has the form

$$H(i) = I - \tau * v * v'$$

where  $\tau$  is a real scalar, and  $v$  is a real vector with  $v(1:i) = 0$ ,  $v(i+1) = 1$  and  $v(ihi+1:n) = 0$ ;  $v(i+2:ihi)$  is stored on exit in  $A(i+2:ihi,i)$ , and  $\tau$  in  $TAU(i)$ .

The contents of  $A$  are illustrated by the following example, with  $n = 7$ ,  $ilo = 2$  and  $ihi = 6$ :

on entry,	on exit,
( a a a a a a a )	( a a h h h h a )
( a a a a a a a )	( a h h h h h a )
( a a a a a a a )	( h h h h h h )
( a a a a a a a )	( v2 h h h h h )
( a a a a a a a )	( v2 v3 h h h h )
( a a a a a a a )	( v2 v3 v4 h h h )
( a )	( a )

where  $a$  denotes an element of the original matrix  $A$ ,  $h$  denotes a modified element of the upper Hessenberg matrix  $H$ , and  $v_i$  denotes an element of the vector defining  $H(i)$ .

This implementation follows the algorithm and notations described in

S. Tomov and J. Dongarra, "Accelerating the reduction to upper Hessenberg form through hybrid GPU-based computing," University of Tennessee Computer Science Technical Report, UT-CS-09-642 (also LAPACK Working Note 219), May 24, 2009.

### 1.3.2 Function magma\_dgehrd

```
int magma_dgehrd(int *n, int *ilo, int *ihi, double *a, int *lda,
                 double *tau, double *work, int *lwork, double *da, int *info)
```

DGEHRD reduces a real general matrix A to upper Hessenberg form H by an orthogonal similarity transformation:  $Q' * A * Q = H$ .

**N** (input) INTEGER  
The order of the matrix A.  $N \geq 0$ .

**ILO** (input) INTEGER  
**IHI** (input) INTEGER  
It is assumed that A is already upper triangular in rows and columns 1:ILO-1 and IHI+1:N. ILO and IHI are normally set by a previous call to DGEBAL; otherwise they should be set to 1 and N respectively. See Further Details.  
 $1 \leq ILO \leq IHI \leq N$ , if  $N > 0$ ;  $ILO=1$  and  $IHI=0$ , if  $N=0$ .

**A** (input/output) DOUBLE PRECISION array, dimension (LDA,N)  
On entry, the N-by-N general matrix to be reduced.  
On exit, the upper triangle and the first subdiagonal of A are overwritten with the upper Hessenberg matrix H, and the elements below the first subdiagonal, with the array TAU, represent the orthogonal matrix Q as a product of elementary reflectors. See Further Details.

**LDA** (input) INTEGER  
The leading dimension of the array A.  $LDA \geq \max(1, N)$ .

**TAU** (output) DOUBLE PRECISION array, dimension (N-1)  
The scalar factors of the elementary reflectors (see Further Details). Elements 1:ILO-1 and IHI:N-1 of TAU are set to zero.

**WORK** (workspace/output) DOUBLE PRECISION array, dimension (LWORK)  
On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

**LWORK** (input) INTEGER  
The length of the array WORK.  $LWORK \geq \max(1, N)$ .  
For optimum performance  $LWORK \geq N \cdot NB$ , where NB is the optimal blocksize.  
  
If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

**DA** (workspace) DOUBLE array on the GPU, dimension  $N \cdot N + 2 \cdot N \cdot NB + NB \cdot NB$ ,  
where NB can be obtained through magma\_get\_dgehrd\_nb(N).

**INFO** (output) INTEGER  
= 0: successful exit  
< 0: if INFO = -i, the i-th argument had an illegal value.

Further Details  
=====

The matrix  $Q$  is represented as a product of  $(ihi-ilo)$  elementary reflectors

$$Q = H(ilo) H(ilo+1) \dots H(ihi-1).$$

Each  $H(i)$  has the form

$$H(i) = I - \tau * v * v'$$

where  $\tau$  is a real scalar, and  $v$  is a real vector with  $v(1:i) = 0$ ,  $v(i+1) = 1$  and  $v(ihi+1:n) = 0$ ;  $v(i+2:ihi)$  is stored on exit in  $A(i+2:ihi,i)$ , and  $\tau$  in  $TAU(i)$ .

The contents of  $A$  are illustrated by the following example, with  $n = 7$ ,  $ilo = 2$  and  $ihi = 6$ :

on entry,	on exit,
( a   a   a   a   a   a   a )	( a   a   h   h   h   h   a )
(   a   a   a   a   a   a )	(   a   h   h   h   h   a )
(     a   a   a   a   a   a )	(     h   h   h   h   h   h )
(     a   a   a   a   a   a )	(     v2   h   h   h   h   h )
(     a   a   a   a   a   a )	(     v2   v3   h   h   h   h )
(     a   a   a   a   a   a )	(     v2   v3   v4   h   h   h )
(                                a )	(                                a )

where  $a$  denotes an element of the original matrix  $A$ ,  $h$  denotes a modified element of the upper Hessenberg matrix  $H$ , and  $v_i$  denotes an element of the vector defining  $H(i)$ .

This implementation follows the algorithm and notations described in

S. Tomov and J. Dongarra, "Accelerating the reduction to upper Hessenberg form through hybrid GPU-based computing," University of Tennessee Computer Science Technical Report, UT-CS-09-642 (also LAPACK Working Note 219), May 24, 2009.

## Chapter 2

# The MAGMA BLAS Library

The MAGMA BLAS Library is a subset of CUDA BLAS. It is meant as a complementary to the the CUBLAS Library provided by NVIDIA. Included are only certain kernels that are crucial for the performance of MAGMA routines. Although originally meant as an internal to MAGMA library, some of its routines are general enough and are given a user interface to be used externally.

Provided are the matrix-matrix multiplication routines (gemm), matrix-vector multiplication routines, and triangular matrix solvers in real arithmetic (both single and double).

Two main techniques distinguishing the MAGMA BLAS Library are:

1. **Auto-tuners** – both the *Beauty and the Beast* behind MAGMA – an elegant and very practical solution for easy maintenance and performance portability, while often being a brute force, empirically-based exhaustive search that would find and set automatically the best performing algorithms/kernels for a specific hardware configuration.
2. **Pointer redirecting** – a set of GPU specific optimization techniques that allow to easily remove performance oscillations associated with problem sizes not divisible by certain block sizes.

See [1, 2] for further detail.

## 2.1 Matrix-Matrix Multiplication



### 2.1.1 Function `magmablas_sgemm`

```
void magmablas_sgemm(char TRANSA, char TRANSB, int m, int n, int k, float alpha,
                    const float *A, int lda, const float *B, int ldb, float beta,
                    float *C, int ldc)
```

SGEMM performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$ ,

where  $\text{op}(X)$  is one of

$\text{op}(X) = X$  or  $\text{op}(X) = X'$ ,

alpha and beta are scalars, and A, B and C are matrices, with  $\text{op}(A)$  an m by k matrix,  $\text{op}(B)$  a k by n matrix and C an m by n matrix.

TRANSA - CHARACTER\*1.

On entry, TRANSA specifies the form of  $\text{op}(A)$  to be used in the matrix multiplication as follows:

TRANSA = 'N' or 'n',  $\text{op}(A) = A$ .

TRANSA = 'T' or 't',  $\text{op}(A) = A'$ .

TRANSA = 'C' or 'c',  $\text{op}(A) = A'$ .

Unchanged on exit.

TRANSB - CHARACTER\*1.

On entry, TRANSB specifies the form of  $\text{op}(B)$  to be used in the matrix multiplication as follows:

TRANSB = 'N' or 'n',  $\text{op}(B) = B$ .

TRANSB = 'T' or 't',  $\text{op}(B) = B'$ .

TRANSB = 'C' or 'c',  $\text{op}(B) = B'$ .

Unchanged on exit.

M - INTEGER.

On entry, M specifies the number of rows of the matrix  $\text{op}(A)$  and of the matrix C. M must be at least zero.

Unchanged on exit.

N - INTEGER.

On entry, N specifies the number of columns of the matrix  $\text{op}(B)$  and the number of columns of the matrix C. N must be at least zero.

Unchanged on exit.

K - INTEGER.

On entry, K specifies the number of columns of the matrix  $\text{op}(A)$  and the number of rows of the matrix  $\text{op}(B)$ . K must be at least zero.

Unchanged on exit.

ALPHA - SINGLE PRECISION.

On entry, ALPHA specifies the scalar alpha.

Unchanged on exit.

A - SINGLE PRECISION array of DIMENSION (LDA, ka), where ka is k when TRANSA = 'N' or 'n', and is m otherwise. Before entry with TRANSA = 'N' or 'n', the leading m by k part of the array A must contain the matrix A, otherwise the leading k by m part of the array A must contain the matrix A.

Unchanged on exit.

LDA - INTEGER.

On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When TRANSA = 'N' or 'n' then

LDA must be at least  $\max(1, m)$ , otherwise LDA must be at least  $\max(1, k)$ .  
 Unchanged on exit.

- B - SINGLE PRECISION array of DIMENSION ( LDB, kb ), where kb is n when TRANSB = 'N' or 'n', and is k otherwise.  
 Before entry with TRANSB = 'N' or 'n', the leading k by n part of the array B must contain the matrix B, otherwise the leading n by k part of the array B must contain the matrix B.  
 Unchanged on exit.
- LDB - INTEGER.  
 On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When TRANSB = 'N' or 'n' then LDB must be at least  $\max(1, k)$ , otherwise LDB must be at least  $\max(1, n)$ .  
 Unchanged on exit.
- BETA - SINGLE PRECISION.  
 On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input.  
 Unchanged on exit.
- C - SINGLE PRECISION array of DIMENSION ( LDC, n ).  
 Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry.  
 On exit, the array C is overwritten by the m by n matrix  $(\alpha * \text{op}(A) * \text{op}(B) + \text{beta} * C)$ .
- LDC - INTEGER.  
 On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least  $\max(1, m)$ .  
 Unchanged on exit.

## 2.1.2 Function `magmablas_dgemm`

```
void magmablas_dgemm(char TRANSA, char TRANSB, int m, int n, int k, double alpha,
                    const double *A, int lda, const double *B, int ldb,
                    double beta, double *C, int ldc)
```

DGEMM performs one of the matrix-matrix operations

$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$ ,

where  $\text{op}(X)$  is one of

$\text{op}(X) = X$  or  $\text{op}(X) = X'$ ,

$\alpha$  and  $\beta$  are scalars, and  $A$ ,  $B$  and  $C$  are matrices, with  $\text{op}(A)$  an  $m$  by  $k$  matrix,  $\text{op}(B)$  a  $k$  by  $n$  matrix and  $C$  an  $m$  by  $n$  matrix.

TRANSA - CHARACTER\*1.

On entry, TRANSA specifies the form of  $\text{op}(A)$  to be used in the matrix multiplication as follows:

TRANSA = 'N' or 'n',  $\text{op}(A) = A$ .

TRANSA = 'T' or 't',  $\text{op}(A) = A'$ .

TRANSA = 'C' or 'c',  $\text{op}(A) = A'$ .

Unchanged on exit.

TRANSB - CHARACTER\*1.

On entry, TRANSB specifies the form of  $\text{op}(B)$  to be used in the matrix multiplication as follows:

TRANSB = 'N' or 'n',  $\text{op}(B) = B$ .

TRANSB = 'T' or 't',  $\text{op}(B) = B'$ .

TRANSB = 'C' or 'c',  $\text{op}(B) = B'$ .

Unchanged on exit.

M - INTEGER.

On entry, M specifies the number of rows of the matrix  $\text{op}(A)$  and of the matrix  $C$ . M must be at least zero. Unchanged on exit.

N - INTEGER.

On entry, N specifies the number of columns of the matrix  $\text{op}(B)$  and the number of columns of the matrix  $C$ . N must be at least zero. Unchanged on exit.

K - INTEGER.

On entry, K specifies the number of columns of the matrix  $\text{op}(A)$  and the number of rows of the matrix  $\text{op}(B)$ . K must be at least zero. Unchanged on exit.

ALPHA - DOUBLE PRECISION.

On entry, ALPHA specifies the scalar  $\alpha$ . Unchanged on exit.

A - DOUBLE PRECISION array of DIMENSION (LDA, ka), where ka is k when TRANSA = 'N' or 'n', and is m otherwise. Before entry with TRANSA = 'N' or 'n', the leading m by k part of the array A must contain the matrix A, otherwise the leading k by m part of the array A must contain the matrix A. Unchanged on exit.

- LDA - INTEGER.  
On entry, LDA specifies the first dimension of A as declared in the calling (sub) program. When `TRANSA = 'N'` or `'n'` then LDA must be at least `max( 1, m )`, otherwise LDA must be at least `max( 1, k )`.  
Unchanged on exit.
- B - DOUBLE PRECISION array of DIMENSION ( LDB, kb ), where kb is n when `TRANSB = 'N'` or `'n'`, and is k otherwise.  
Before entry with `TRANSB = 'N'` or `'n'`, the leading k by n part of the array B must contain the matrix B, otherwise the leading n by k part of the array B must contain the matrix B.  
Unchanged on exit.
- LDB - INTEGER.  
On entry, LDB specifies the first dimension of B as declared in the calling (sub) program. When `TRANSB = 'N'` or `'n'` then LDB must be at least `max( 1, k )`, otherwise LDB must be at least `max( 1, n )`.  
Unchanged on exit.
- BETA - DOUBLE PRECISION.  
On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input.  
Unchanged on exit.
- C - DOUBLE PRECISION array of DIMENSION ( LDC, n ).  
Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry.  
On exit, the array C is overwritten by the m by n matrix  $( \alpha * \text{op}( A ) * \text{op}( B ) + \text{beta} * C )$ .
- LDC - INTEGER.  
On entry, LDC specifies the first dimension of C as declared in the calling (sub) program. LDC must be at least `max( 1, m )`.  
Unchanged on exit.

## 2.2 Matrix-Vector Multiplication

### 2.2.1 Function `magma_smemv`

### 2.2.2 Function magma\_dgemv

## **2.3 Symmetric Matrix-Vector multiplication**



### 2.3.1 Function `magmablas_ssymv`

```
int magmablas_ssymv(char uplo, int m, float alpha, float *A, int lda, float *X,
                    int incx, float beta, float *Y, int incy)
```

SSYMV performs the matrix-vector operation

$y := \alpha A x + \beta y$ ,

where  $\alpha$  and  $\beta$  are scalars,  $x$  and  $y$  are  $n$  element vectors and  $A$  is an  $n$  by  $n$  symmetric matrix.

UPLO CHARACTER\*1.

On entry, UPLO specifies whether the upper or lower triangular part of the array  $A$  is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of  $A$  is to be referenced.

UPLO = 'L' or 'l' Only the lower triangular part of  $A$  is to be referenced.

Unchanged on exit.

N INTEGER.

On entry,  $N$  specifies the order of the matrix  $A$ .

$N$  must be at least zero.

Unchanged on exit.

ALPHA REAL.

On entry, ALPHA specifies the scalar  $\alpha$ .

Unchanged on exit.

A REAL array of DIMENSION ( LDA,  $n$  ).

Before entry with UPLO = 'U' or 'u', the leading  $n$  by  $n$  upper triangular part of the array  $A$  must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of  $A$  is not referenced.

Before entry with UPLO = 'L' or 'l', the leading  $n$  by  $n$  lower triangular part of the array  $A$  must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of  $A$  is not referenced.

Unchanged on exit.

LDA INTEGER.

On entry, LDA specifies the first dimension of  $A$  as declared in the calling (sub) program. LDA must be at least  $\max(1, n)$ .

Unchanged on exit.

X REAL array of dimension at least  $(1 + (n - 1) * \text{abs}(\text{INCX}))$ .

Before entry, the incremented array  $X$  must contain the  $n$  element vector  $x$ .

Unchanged on exit.

INCX INTEGER.

On entry, INCX specifies the increment for the elements of  $X$ . INCX must not be zero.

Unchanged on exit.

BETA REAL.

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then Y need not be set on input. Unchanged on exit.

Y        REAL array of dimension at least  
          $(1 + (n - 1) * \text{abs}(\text{ INCY }))$ .  
Before entry, the incremented array Y must contain the n element vector y. On exit, Y is overwritten by the updated vector y.

INCY     INTEGER.  
On entry, INCY specifies the increment for the elements of Y. INCY must not be zero. Unchanged on exit.

### 2.3.2 Function `magmablas_dsymv`

```
int magmablas_dsymv(char uplo, int m, double alpha, double *A, int lda,
                   double *X, int incx, double beta, double *Y, int incy)
```

DSYMV performs the matrix-vector operation

$y := \alpha A x + \beta y$ ,

where  $\alpha$  and  $\beta$  are scalars,  $x$  and  $y$  are  $n$  element vectors and  $A$  is an  $n$  by  $n$  symmetric matrix.

UPLO CHARACTER\*1.

On entry, UPLO specifies whether the upper or lower triangular part of the array  $A$  is to be referenced as follows:

UPLO = 'U' or 'u' Only the upper triangular part of  $A$  is to be referenced.  
 UPLO = 'L' or 'l' Only the lower triangular part of  $A$  is to be referenced.

Unchanged on exit.

N INTEGER.

On entry,  $N$  specifies the order of the matrix  $A$ .

$N$  must be at least zero.

Unchanged on exit.

ALPHA DOUBLE PRECISION.

On entry, ALPHA specifies the scalar  $\alpha$ .

Unchanged on exit.

A DOUBLE PRECISION array of DIMENSION ( LDA,  $n$  ).

Before entry with UPLO = 'U' or 'u', the leading  $n$  by  $n$  upper triangular part of the array  $A$  must contain the upper triangular part of the symmetric matrix and the strictly lower triangular part of  $A$  is not referenced.

Before entry with UPLO = 'L' or 'l', the leading  $n$  by  $n$  lower triangular part of the array  $A$  must contain the lower triangular part of the symmetric matrix and the strictly upper triangular part of  $A$  is not referenced.

Unchanged on exit.

LDA INTEGER.

On entry, LDA specifies the first dimension of  $A$  as declared in the calling (sub) program. LDA must be at least  $\max(1, n)$ .

Unchanged on exit.

X DOUBLE PRECISION array of dimension at least

$(1 + (n - 1) * \text{abs}(\text{INCX}))$ .

Before entry, the incremented array  $X$  must contain the  $n$  element vector  $x$ .

Unchanged on exit.

INCX INTEGER.

On entry, INCX specifies the increment for the elements of  $X$ . INCX must not be zero.

Unchanged on exit.

BETA      DOUBLE PRECISION.  
On entry, BETA specifies the scalar beta. When BETA is  
supplied as zero then Y need not be set on input.  
Unchanged on exit.

Y          DOUBLE PRECISION array of dimension at least  
          ( 1 + ( n - 1 ) \* abs( INCY ) ).  
Before entry, the incremented array Y must contain the n  
element vector y. On exit, Y is overwritten by the updated  
vector y.

INCY      INTEGER.  
On entry, INCY specifies the increment for the elements of  
Y. INCY must not be zero.  
Unchanged on exit.

## 2.4 Triangular Matrix Solvers

### 2.4.1 Function magma\_strsm

```
int magmablas_strsm(char side, char uplo, char tran, char diag, int M, int N,
                    float alpha, float* A, int lda, float* b, int ldb)
```

STRSM solves one of the matrix equations on GPU

$\text{op}(A) * X = \alpha * B$ , or  $X * \text{op}(A) = \alpha * B$ ,  
 where  $\alpha$  is a scalar,  $X$  and  $B$  are  $m$  by  $n$  matrices,  $A$  is a unit, or  
 non-unit, upper or lower triangular matrix and  $\text{op}(A)$  is one of  
 $\text{op}(A) = A$  or  $\text{op}(A) = A'$ .

The solution matrix  $X$  overwrites  $B$ .

To extract more parallelism and performance, the diagonal blocks of  $A$   
 of size  $32 \times 32$  are explicitly inverted. When  $M$  or  $N$  is not a multiple of  
 the current blocking size (32 for now), cublasStrsm is called instead.

**side** CHARACTER\*1.  
 On entry, side specifies whether  $\text{op}(A)$  appears on the left  
 or right of  $X$  as follows:  
     side = 'L' or 'l'     $\text{op}(A) * X = \alpha * B$ .  
     side = 'R' or 'r'     $X * \text{op}(A) = \alpha * B$ .  
 Unchanged on exit.

**uplo** CHARACTER\*1.  
 On entry, uplo specifies whether the matrix  $A$  is an upper or  
 lower triangular matrix as follows:  
     uplo = 'U' or 'u'     $A$  is an upper triangular matrix.  
     uplo = 'L' or 'l'     $A$  is a lower triangular matrix.  
 Unchanged on exit.

**tran** CHARACTER\*1.  
 On entry, tran specifies the form of  $\text{op}(A)$  to be used in  
 the matrix multiplication as follows:  
     tran = 'N' or 'n'     $\text{op}(A) = A$ .  
     tran = 'T' or 't'     $\text{op}(A) = A'$ .  
     tran = 'C' or 'c'     $\text{op}(A) = A'$ .  
 Unchanged on exit.

**diag** CHARACTER\*1.  
 On entry, diag specifies whether or not  $A$  is unit triangular  
 as follows:  
     diag = 'U' or 'u'     $A$  is assumed to be unit triangular.  
     diag = 'N' or 'n'     $A$  is not assumed to be unit triangular.  
 Unchanged on exit.

**m** INTEGER.  
 On entry, m specifies the number of rows of  $B$ . m must be at  
 least zero.  
 Unchanged on exit.

**n** INTEGER.  
 On entry, n specifies the number of columns of  $B$ . n must be  
 at least zero.  
 Unchanged on exit.

**alpha** REAL.  
 On entry, alpha specifies the scalar alpha. When alpha is

zero then A is not referenced and B need not be set before entry.

Unchanged on exit.

A REAL array of DIMENSION ( lda, k ), where k is m when side = 'L' or 'l' and is n when side = 'R' or 'r'. Before entry with uplo = 'U' or 'u', the leading k by k upper triangular part of the array A must contain the upper triangular matrix and the strictly lower triangular part of A is not referenced. Before entry with uplo = 'L' or 'l', the leading k by k lower triangular part of the array A must contain the lower triangular matrix and the strictly upper triangular part of A is not referenced. Note that when diag = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity. Unchanged on exit.

lda INTEGER.  
On entry, lda specifies the first dimension of A as declared in the calling (sub) program. When side = 'L' or 'l' then lda must be at least max( 1, m ), when side = 'R' or 'r' then lda must be at least max( 1, n ).  
Unchanged on exit.

b REAL array of DIMENSION ( ldb, n ).  
Before entry, the leading m by n part of the array B must contain the right-hand side matrix B, and on exit is overwritten by the solution matrix X.

ldb INTEGER.  
On entry, ldb specifies the first dimension of B as declared in the calling (sub) program. ldb must be at least max( 1, m ).  
Unchanged on exit.

DTRSM solves one of the matrix equations on GPU  
 $\text{op}(A) * X = \alpha * B$ , or  $X * \text{op}(A) = \alpha * B$ ,  
 where  $\alpha$  is a scalar,  $X$  and  $B$  are  $m$  by  $n$  matrices,  $A$  is a unit, or  
 non-unit, upper or lower triangular matrix and  $\text{op}(A)$  is one of  
 $\text{op}(A) = A$  or  $\text{op}(A) = A'$ .  
 The matrix  $X$  is overwritten on  $B$ .

To extract more parallelism and performance, the diagonal blocks of  $A$  of size  $32 \times 32$  are explicitly inverted. When  $M$  or  $N$  is not a multiple of the current blocking size (32 for now), `cublasStrsm` is called instead.

```

side      CHARACTER*1.
On entry, side specifies whether op( A ) appears on the left
or right of X as follows:
    side = 'L' or 'l'    op( A ) * X = alpha * B.
    side = 'R' or 'r'    X * op( A ) = alpha * B.
Unchanged on exit.

```

```
uplo      CHARACTER*1.
On entry, uplo specifies whether the matrix A is an upper or
lower triangular matrix as follows:
    uplo = 'U' or 'u'   A is an upper triangular matrix.
    uplo = 'L' or 'l'   A is a lower triangular matrix.
Unchanged on exit.
```

```
tran      CHARACTER*1.  
On entry, tran specifies the form of op( A ) to be used in  
the matrix multiplication as follows:  
    tran = 'N' or 'n'   op( A ) = A.  
    tran = 'T' or 't'   op( A ) = A'.  
    tran = 'C' or 'c'   op( A ) = A'.  
Unchanged on exit.
```

```
diag      CHARACTER*1.
On entry, diag specifies whether or not A is unit triangular
as follows:
    diag = 'U' or 'u'   A is assumed to be unit triangular.
    diag = 'N' or 'n'   A is not assumed to be unit
                        triangular.

Unchanged on exit.
```

```

m      INTEGER.
      On entry, m specifies the number of rows of B. m must be at
      least zero.
      Unchanged on exit.

```

n            INTEGER.  
On entry, n specifies the number of columns of B. n must be  
at least zero.  
Unchanged on exit.

alpha DOUBLE PRECISION.  
On entry, alpha specifies the scalar alpha. When alpha is



zero then A is not referenced and B need not be set before entry.

Unchanged on exit.

- A      DOUBLE PRECISION array of DIMENSION ( lda, k ), where k is m when side = 'L' or 'l' and is n when side = 'R' or 'r'. Before entry with uplo = 'U' or 'u', the leading k by k upper triangular part of the array A must contain the upper triangular matrix and the dtrictly lower triangular part of A is not referenced. Before entry with uplo = 'L' or 'l', the leading k by k lower triangular part of the array A must contain the lower triangular matrix and the dtrictly upper triangular part of A is not referenced. Note that when diag = 'U' or 'u', the diagonal elements of A are not referenced either, but are assumed to be unity. Unchanged on exit.
- lda     INTEGER.  
On entry, lda specifies the first dimension of A as declared in the calling (sub) program. When side = 'L' or 'l' then lda must be at least max( 1, m ), when side = 'R' or 'r' then lda must be at least max( 1, n ).  
Unchanged on exit.
- b      DOUBLE PRECISION array of DIMENSION ( ldb, n ).  
Before entry, the leading m by n part of the array B must contain the right-hand side matrix B, and on exit is overwritten by the solution matrix X.
- ldb     INTEGER.  
On entry, ldb specifies the first dimension of B as declared in the calling (sub) program. ldb must be at least max( 1, m ).  
Unchanged on exit.

# Chapter 3

## Use

### 3.1 Hardware specifications

MAGMA version 0.2 is intended for a single CUDA enabled NVIDIA GPU and it's host. CUDA enabled GPUs are for example the GeForce 8 Series, the Tesla GPUs, and some Quadro GPUs [3]. MAGMA's double precision routines can be used on CUDA enabled GPUs that support double precision arithmetic. These are for example the GeForce 200 Series and the Tesla solutions. The host can be any shared memory multiprocessor for which LAPACK is suitable. One host core is required and multiple can be used through multicore LAPACK implementation.

### 3.2 Software specifications

MAGMA version 0.2 is a Linux release that requires

- the CUDA driver and CUDA toolkit <sup>1</sup>;
- CPU BLAS and LAPACK.

MAGMA users do not have to know CUDA in order to use the library. A testing directory gives examples on how to use every function (see Section 3.3). Applications can use the CPU interface without any significant change to the application – LAPACK calls have to be prefixed with `magma_` and a workspace argument (for the GPU memory) has to be added (shown in the examples).

---

<sup>1</sup>freely available from NVIDIA  
[http://www.nvidia.com/object/cuda\\_get.html](http://www.nvidia.com/object/cuda_get.html)

### 3.3 Examples and testing

Directory `magma/testing` has drivers that test and show how to use every function of this distribution. Below is an example showing the output of the `sgetrf_gpu` driver.

```
> ./testing_sgetrf_gpu
Using device 0: GeForce GTX 280
```

Usage:

```
testing_sgetrf_gpu -N 1024
```

N	CPU GFlop/s	GPU GFlop/s	PA-LU   / (  A  *N)
1024	34.36	45.92	1.861593e-09
2048	52.85	105.63	1.722339e-09
3072	64.19	162.94	1.411851e-09
4032	80.45	225.07	1.384447e-09
5184	86.43	260.74	1.346531e-09
6016	91.01	277.32	1.341646e-09
7040	95.62	292.25	1.322844e-09
8064	99.71	302.55	1.388576e-09
9088	101.09	310.33	1.554564e-09
10112	103.65	316.23	1.660534e-09

Performance and accuracy for particular values of the matrix size can also be tested. Note that performance is slower for matrix sizes that are not divisible by the block size of the corresponding algorithm. The block sizes will be auto-tuned in future releases. Currently, the user can change them through file `get_nb.cpp` to manually tune the performance for specific hardware and software settings. Some of the performance issues related to matrix sizes not divisible by the block size are addressed on BLAS level (thorough MAGMA BLAS). It still needs improvements though:

```
> ./testing_sgetrf_gpu -N 5000
Using device 0: GeForce GTX 280
```

N	CPU GFlop/s	GPU GFlop/s	PA-LU   / (  A  *N)
5000	86.58	230.94	1.340746e-09

# Chapter 4

## Performance

Here we give the reference performance results using MAGMA version 0.2 in the following hardware and software configuration:

**GPU:** NVIDIA GeForce GTX 280;

**CPU:** Intel Xeon dual socket quad-core @ 2.33 GHz;

**GPU BLAS:** CUBLAS 2.3;

**CPU BLAS:** MKL 10.0;

**Compiler:** gcc 4.1.2;

**Tuning:** Hand tuned (and hard coded).

Note that this release is hand tuned for this particular configuration. Different configurations may require different tuning in which case there would be a negative impact on the performance. Future releases will be auto-tuned using an empirically-based approach [1]. A handle to user tuning is given in file

`testing/get_nb.cpp`

through functions

`magma_get_{function name}_nb`

which, based on a matrix size, return a block size to be used by the corresponding function. Optimal sizes (for the functions in this distribution) would be a multiple of 32.

## 4.1 Single precision one-sided factorizations

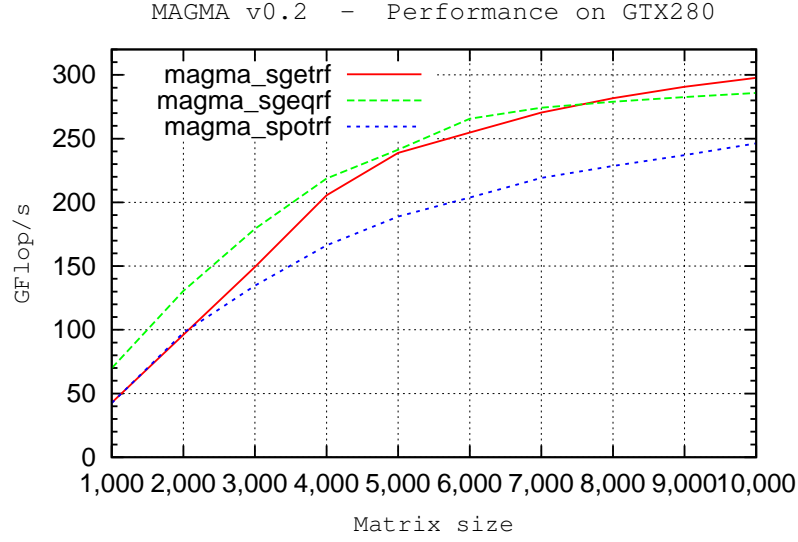


Figure 4.1: Performance of the **CPU interface** one-sided factorizations.

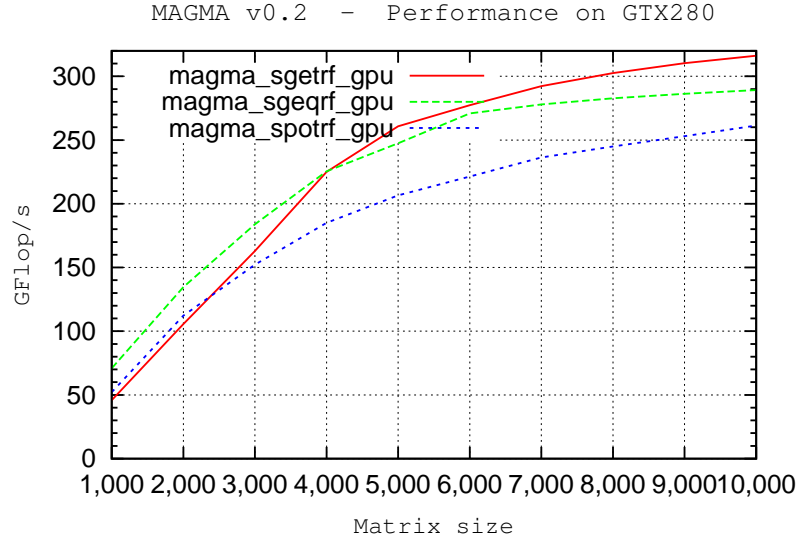


Figure 4.2: Performance of the **GPU interface** one-sided factorizations.

## 4.2 Double precision one-sided factorizations

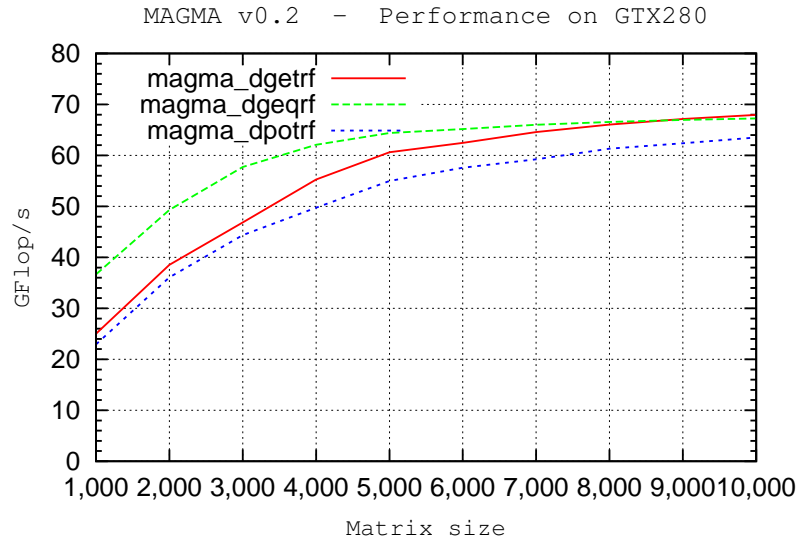


Figure 4.3: Performance of the **CPU interface** one-sided factorizations.

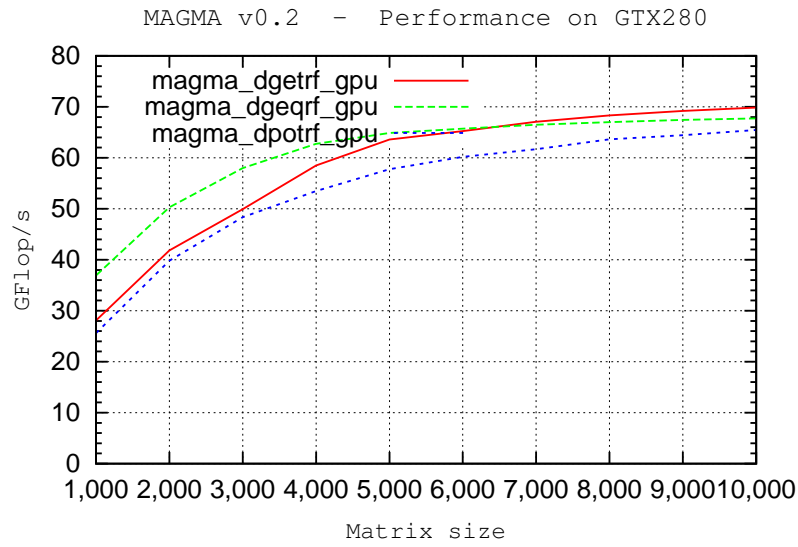


Figure 4.4: Performance of the **GPU interface** one-sided factorizations.

### 4.3 Single complex one-sided factorizations

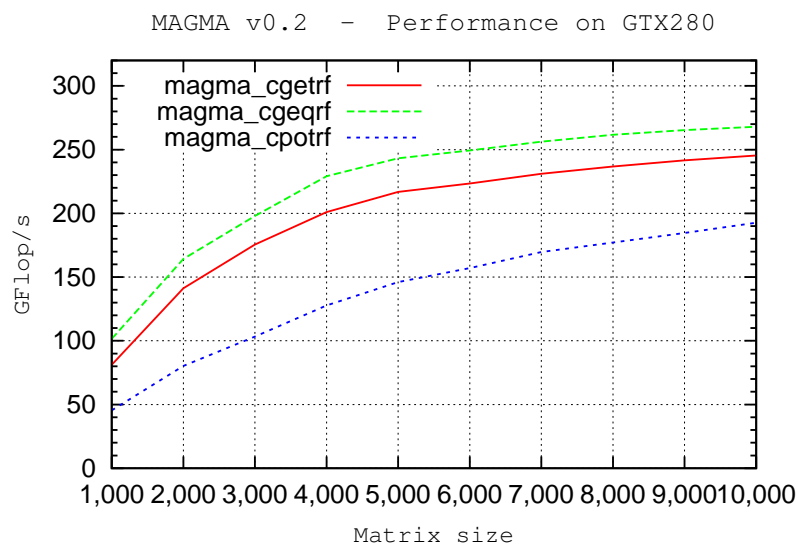


Figure 4.5: Performance of the **CPU interface** one-sided factorizations.

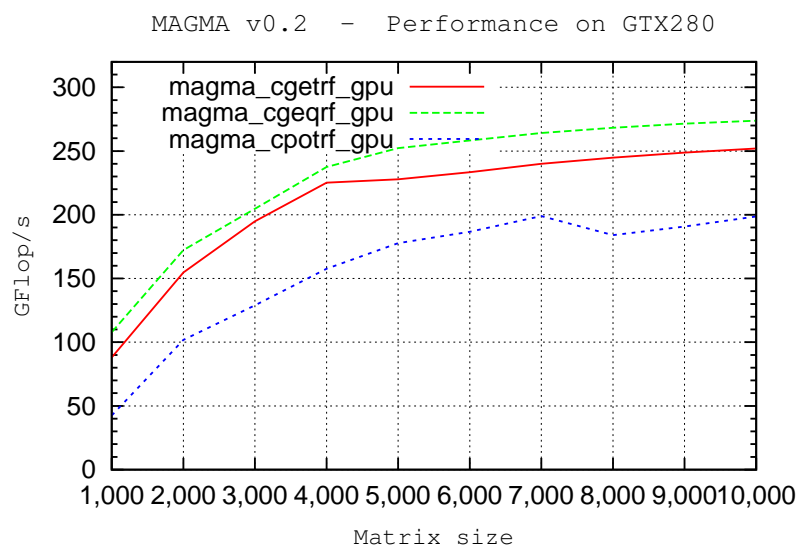


Figure 4.6: Performance of the **GPU interface** one-sided factorizations.

## 4.4 Double complex one-sided factorizations

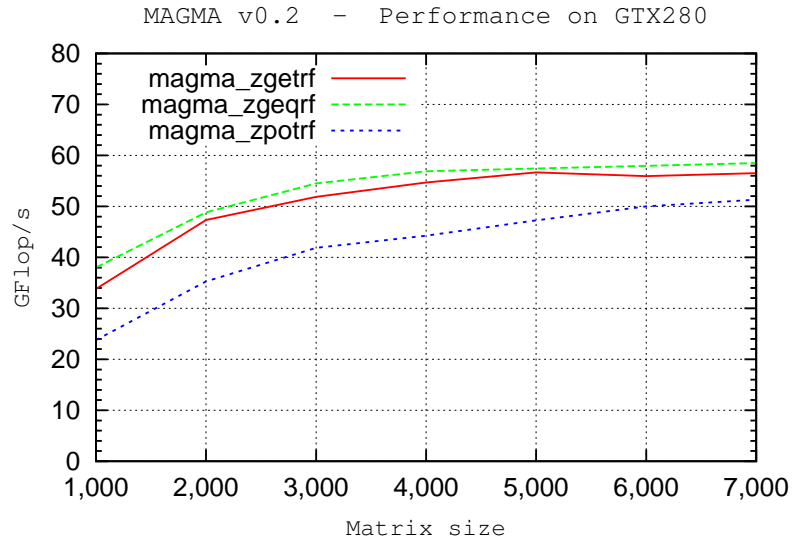


Figure 4.7: Performance of the **CPU interface** one-sided factorizations.

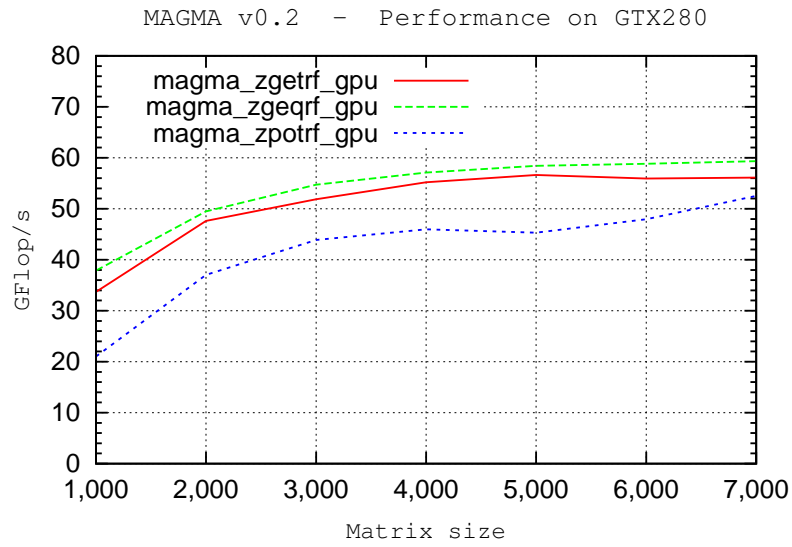


Figure 4.8: Performance of the **GPU interface** one-sided factorizations.



## 4.5 LU-based linear solvers

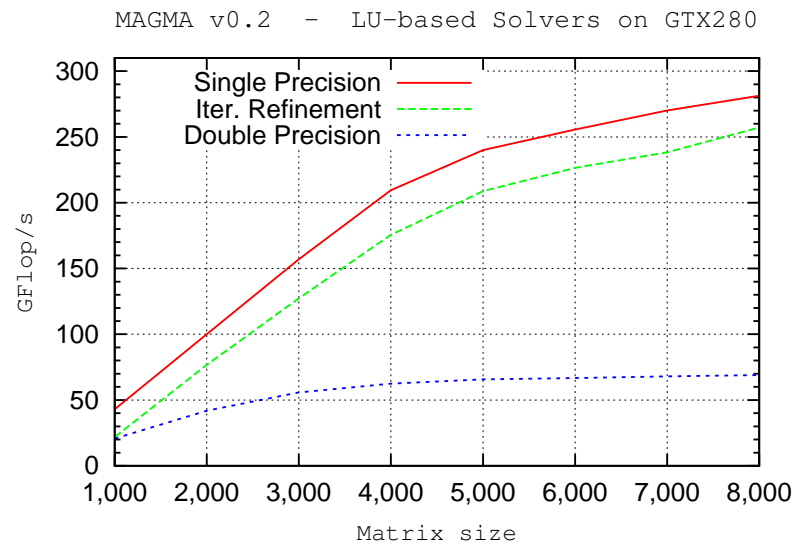


Figure 4.9: Performance of LU-based linear solvers.

## 4.6 QR-based least squares solvers

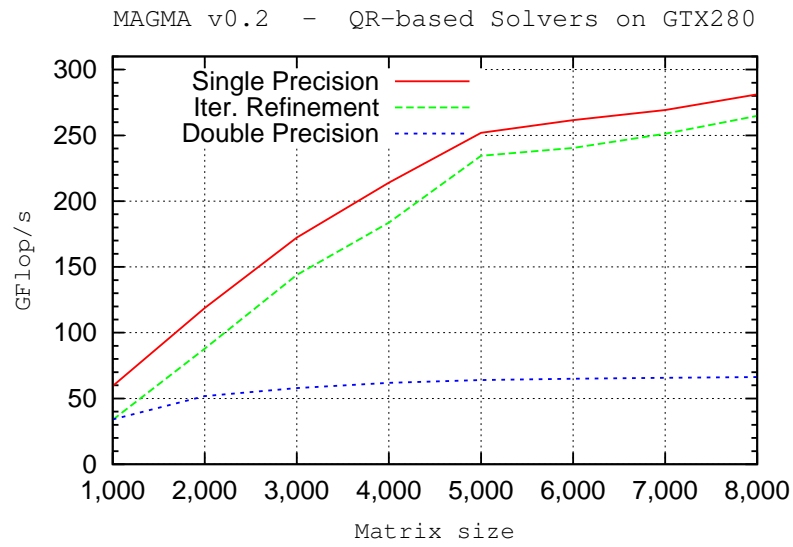


Figure 4.10: Performance of QR-based least squares solvers.

## 4.7 Cholesky-based linear solvers

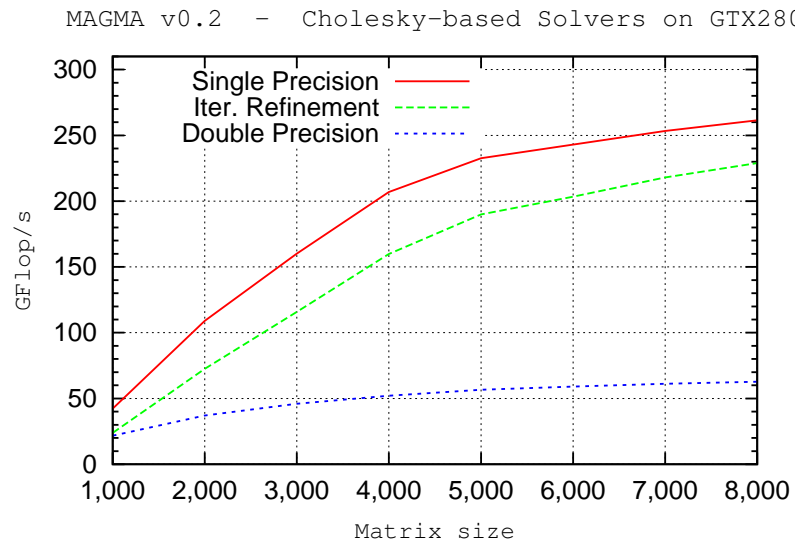


Figure 4.11: Performance of Cholesky-based linear solvers.

# Acknowledgments

This work is supported by Microsoft, NVIDIA, the U.S. National Science Foundation, and the U.S. Department of Energy.

# Bibliography

- [1] Yinan Li, Jack Dongarra, and Stanimire Tomov, *A note on auto-tuning GEMM for GPUs.*, Lecture Notes in Computer Science, vol. 5544, Springer, 2009.
- [2] R. Nath, J. Dongarra, S. Tomov, H. Ltaief, and P. Du, *Numerical Linear Algebra on Hybrid Architectures: Recent Developments in the MAGMA Project*, <http://icl.cs.utk.edu/projectsfiles/magma/pubs/MAGMA-BLAS-SC09.pdf>, 09/2009, Poster Supercomputing '09.
- [3] NVIDIA, *NVIDIA CUDA Programming Guide*, 6/07/2008, Version 2.0.
- [4] S. Tomov, J. Dongarra, and M. Baboulin, *Towards dense linear algebra for hybrid GPU accelerated manycore systems.*, Tech. report, LAPACK Working Note 210, October 2008.
- [5] S. Tomov, J. Dongarra, V. Volkov, and J. Demmel, *MAGMA Library, version 0.1*, <http://icl.cs.utk.edu/magma>, 08/2009.
- [6] Stanimire Tomov and Jack Dongarra, *Accelerating the reduction to upper Hessenberg form through hybrid GPU-based computing.*, Tech. Report 219, LAPACK Working Note, May 2009.
- [7] V. Volkov and J. Demmel, *Benchmarking GPUs to tune dense linear algebra*, SC '08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing (Piscataway, NJ, USA), IEEE Press, 2008, pp. 1–11.