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

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 ans 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. 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 (single);
- 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

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 >= 0.

    N       (input) INTEGER
            The number of columns of the matrix A.  N >= 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 >= max(1,M).

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

    WORK    (workspace/output) REAL array, dimension >= 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 >= 0.

    N       (input) INTEGER
            The number of columns of the matrix A.  N >= 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 >= 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 >= 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 >= 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) . . . 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 >= 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 >= 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_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 >= 0.

    N       (input) INTEGER
            The number of columns of the matrix A.  N >= 0.

    A       (input/output) REAL array on the GPU, dimension (LDA,N) where
            LDA >= 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 >= max(1,M).

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

    WORK    (workspace/output) REAL array, dimension >= 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.
```

1.1.5 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 >= 0.

    N       (input) INTEGER
            The number of columns of the matrix A.  N >= 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 >= 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 >= 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 A is in pinned memory,
            e.g. allocated using cudaMallocHost.

    LWORK   (input) INTEGER
            The dimension of the array WORK.  LWORK >= (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) . . . 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).
```

1.1.6 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 >= 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 >= 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.
```

1.1.7 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 >= 0.

    N       (input) INTEGER
            The number of columns of the matrix A.  N >= 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 >= max(1,M).

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

    WORK    (workspace/output) DOUBLE array, dimension >= 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.
```

1.1.8 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 >= 0.

    N       (input) INTEGER
            The number of columns of the matrix A.  N >= 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 >= 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 >= 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 >= 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) . . . 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).
```

1.1.9 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 >= 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 >= 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.
```

1.1.10 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 >= 0.

    N       (input) INTEGER
            The number of columns of the matrix A.  N >= 0.

    A       (input/output) DOUBLE array on the GPU, dimension (LDA,N) where
            LDA >= 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 >= max(1,M).

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

    WORK    (workspace/output) DOUBLE array, dimension >= 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.
```

1.1.11 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 >= 0.

    N       (input) INTEGER
            The number of columns of the matrix A.  N >= 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 >= 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 >= 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 A is in pinned memory,
            e.g. allocated using cudaMallocHost.

    LWORK   (input) INTEGER
            The dimension of the array WORK.  LWORK >= (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) . . . 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).
```

1.1.12 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 >= 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 >= 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.
```

1.1.13 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 >= 0.

    N       (input) INTEGER
            The number of columns of the matrix A.  N >= 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 >= max(1,M).

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

    WORK    (workspace/output) COMPLEX array, dimension >= 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.
```

1.1.14 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 >= 0.

    N       (input) INTEGER
            The number of columns of the matrix A.  N >= 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 >= 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 >= 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 >= 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) . . . 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).
```

1.1.15 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 >= 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 >= 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.
```

1.1.16 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 >= 0.

    N       (input) INTEGER
            The number of columns of the matrix A.  N >= 0.

    A       (input/output) COMPLEX array on the GPU, dimension (LDA,N) where
            LDA >= 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 >= max(1,M).

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

    WORK    (workspace/output) COMPLEX array, dimension >= 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.
```

1.1.17 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 >= 0.

    N       (input) INTEGER
            The number of columns of the matrix A.  N >= 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 >= 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 >= 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 A is in pinned memory,
            e.g. allocated using cudaMallocHost.

    LWORK   (input) INTEGER
            The dimension of the array WORK.  LWORK >= (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) . . . 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).
```

1.1.18 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 >= 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 >= 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.
```

1.1.19 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 >= 0.

    N       (input) INTEGER
            The number of columns of the matrix A.  N >= 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 >= max(1,M).

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

    WORK    (workspace/output) DOUBLE COMPLEX array, dimension >= 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.
```

1.1.20 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 >= 0.

    N       (input) INTEGER
            The number of columns of the matrix A.  N >= 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 >= 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 >= 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 >= 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) . . . 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).
```

1.1.21 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 >= 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 >= 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.
```

1.1.22 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 >= 0.

    N       (input) INTEGER
            The number of columns of the matrix A.  N >= 0.

    A       (input/output) DOUBLE COMPLEX array on the GPU, dimension (LDA,N) where
            LDA >= 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 >= max(1,M).

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

    WORK    (workspace/output) DOUBLE COMPLEX array, dimension >= 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.
```

1.1.23 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 >= 0.

    N       (input) INTEGER
            The number of columns of the matrix A.  N >= 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 >= 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 >= 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 A is in pinned memory,
            e.g. allocated using cudaMallocHost.

    LWORK   (input) INTEGER
            The dimension of the array WORK.  LWORK >= (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) . . . 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).
```

1.1.24 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 >= 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 >= 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

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 >= 0.

    NRHS    (input) INTEGER
            The number of right hand sides, i.e., the number of columns
            of the matrix B.  NRHS >= 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 >= max(1,N).

    IPIV    (input) INTEGER array, dimension (N)
            The pivot indices from SGETRF; for 1<=i<=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 >= 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*X - C ||
    using the QR factorization A = Q*R computed by SGEQRF_GPU2.

    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 columns of the matrix C. NRHS >= 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,...,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 >= 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 >= 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 >= max(1,NRHS).
            For optimum performance LWORK >= (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*X = B with a symmetric
    positive definite matrix A using the Cholesky factorization
    A = U**T*U or A = L*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 >= 0.

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

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

    LDA     (input) INTEGER
            The leading dimension of the array A.  LDA >= 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 >= 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 >= 0.

    NRHS    (input) INTEGER
            The number of right hand sides, i.e., the number of columns
            of the matrix B.  NRHS >= 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 >= max(1,N).

    IPIV    (input) INTEGER array, dimension (N)
            The pivot indices from SGETRF; for 1<=i<=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 >= 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*X - C ||
    using the QR factorization A = Q*R computed by SGEQRF_GPU2.

    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 columns of the matrix C. NRHS >= 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,...,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 >= 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 >= 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 >= max(1,NRHS).
            For optimum performance LWORK >= (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*X = B with a symmetric
    positive definite matrix A using the Cholesky factorization
    A = U**T*U or A = L*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 >= 0.

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

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

    LDA     (input) INTEGER
            The leading dimension of the array A.  LDA >= 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 >= max(1,N).

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

1.3 Two-sided matrix factorizations

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 >= 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 <= ILO <= IHI <= 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 >= 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 >= max(1,N).
            For optimum performance LWORK >= 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 by XERBLA.

    DA      (workspace)  SINGLE array on the GPU, dimension
            N*N + 2*N*NB + NB*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) . . . 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 vi 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 >= 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 <= ILO <= IHI <= 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 >= 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 >= max(1,N).
            For optimum performance LWORK >= 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 by XERBLA.

    DA      (workspace)  DOUBLE array on the GPU, dimension
            N*N + 2*N*NB + NB*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) . . . 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 vi 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

2.1 Matrix-metrix multiplication

2.2 Matrix-vector multiplication

2.3 Matrix-vector multiplication

2.4 Triangular matrix solvers

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 [2]. 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 [1];
- 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).

[1] freely available from NVIDIA
http://www.nvidia.com/object/cuda_get.html

3.3 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 driver.

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

Usage:
  testing_sgetrf -N 1024

  N    CPU GFlop/s    GPU GFlop/s    ||PA-LU|| / (||A||*N)
==========================================================
 1024     33.26          42.77         1.861593e-09
 2048     52.29          96.06         1.722339e-09
 3072     64.03         146.33         1.411851e-09
 4032     80.60         195.44         1.371482e-09
 5184     86.65         224.92         1.332554e-09
 6016     91.66         240.33         1.331916e-09
 7040     96.02         255.51         1.306940e-09
 8064     99.88         267.17         1.391934e-09
 9088    101.18         276.59         1.549758e-09
10112    104.38         284.30         1.661756e-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. The issue for matrix sizes not divisible by the block size will be addressed in future MAGMA releases (currently due to CUBLAS being slower for those cases).

```
> ./testing_sgetrf -N 1026
Using device 0: GeForce GTX 280


  N    CPU GFlop/s    GPU GFlop/s    ||PA-LU|| / (||A||*N)
==========================================================
 1026     32.93          41.09         1.834303e-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 2.1;

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

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

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

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

Acknowledgments

This work is supported by Microsoft, the U.S. National Science Foundation, and the U.S. Department of Energy. We thank NVIDIA and NVIDIA's Professor Partnership Program for their hardware donations.

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] NVIDIA, *NVIDIA CUDA Programming Guide*, 6/07/2008, Version 2.0.