MAGMA Library

version 0.1

S. Tomov J. Dongarra V. Volkov J. Demmel

```
-- MAGMA (version 0.1) --
   Univ. of Tennessee, Knoxville
   Univ. of California, Berkeley
   Univ. of Colorado, Denver
   June 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)
     J. Kurzak   (UT Knoxville)
     J. Langou   (UC Denver)
     S. Tomov    (UT Knoxville)
     V. Volkov   (UC Berkeley)
```

Contents

Chapter 1

The MAGMA Library

Major chip manufacturers are developing next-generation microprocessor designs that are heterogeneous/hybrid in nature, integrating homogeneous x86-based multicore CPU components and GPU components. The **MAGMA** (*Matrix Algebra on GPU and Multicore Architectures*) project's goal is to develop innovative linear algebra algorithms and to incorporate them into a library that is

- **similar to LAPACK in functionality, data storage, and interface**

  but targeting the

- **next-generation of highly parallel, and heterogeneous processors**.

This will allow scientists to effortlessly port any of their LAPACK-relying software components and to take advantage of the new architectures.

MAGMA is designed to run on homogeneous x86-based multicores and take advantage of GPU components (if available). This is achieved by developing a class of **multi-level blocking** algorithms that split the computation into tasks of varying granularity (e.g. large for available GPUs) and dynamically scheduling their execution.

The transition from small tasks (of small block size) to large tasks is done in a recursive fashion where the intermediate for the transition tasks are executed in parallel using dynamic scheduling. The new algorithms, when run on just homogeneous x86-based multicores, outperform vendor implementations (e.g. MKL) in LAPACK accuracy and data layout (no block data-layouts). Adding a GPU increases the performance proportionally to the GPU's computational characteristics. These results are for the one-sided matrix factorizations – LU, QR, and Cholesky. Work on the two-sided factorizations, e.g. Hessenberg reduction, shows more drastic performance improvements (significantly exceeding

an order of magnitude) when comparing homogeneous multicores to hybrid multicores+GPUs. The main reason for these performance improvements is mainly due to the fact that the two-sided factorizations have bandwidth limitations that can not be overcome using just homogeneous multicores.

In addition to standard accuracy algorithms (LAPACK compliant accuracy), we develop algorithms within MAGMA that would allow a user-defined trade-off between accuracy and speed. These algorithms are based on mixed-precision arithmetic and take advantage of GPU's still much higher single *vs* double precision arithmetic.

MAGMA version 0.1 is a release intended for a single GPU – see the specifications in Section 2.1. MAGMA (version 0.1) includes the 3 one-sided matrix factorizations: LU, QR, and Cholesky. There are functions for single and double precision arithmetic and for each of them 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.

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

This is an alpha release. It is based on the **hybridization** of the corresponding LAPACK algorithms [5, 7] and GPU motivated algorithmic innovations [7, 1]. The accuracy is as in LAPACK. The performance relies on the performance of the CPU BLAS and GPU BLAS used. Upcoming releases will incorporate

- Multi-level blocking;
- User defined choice on running using available GPUs or just stand alone homogeneous multicores;
- Auto-tuning [3];
- Optimized GPU BLAS [7, 3];
- New routines allowing for trade-off between performance and accuracy (including mixed precision) [1, 7];
- Routines for optimized use of the multicore host [5, 6];
- Two-sided factorizations and eigen-problem solvers [6].
- New DLA developments on *communication-optimal* algorithms [2];

A reference performance is given in Chapter 3.

1.1 Function magma_sgetrf

```
int
magma_sgetrf(int *m, int *n, float *a, int *lda,
             int *ipiv, float *work, float *da, int *info)
{
/*  -- MAGMA (version 0.1) --
       Univ. of Tennessee, Knoxville
       Univ. of California, Berkeley
       Univ. of Colorado, Denver
       June 2009

    Purpose
    =======

    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.

    Arguments
    =========

    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.

=================================================================== */
```

1.2 Function magma_sgeqrf

```
int
magma_sgeqrf(int *m, int *n, float *a, int  *lda,  float  *tau,
             float *work, int *lwork, float *da, int *info )
{
/*  -- MAGMA (version 0.1) --
       Univ. of Tennessee, Knoxville
       Univ. of California, Berkeley
       Univ. of Colorado, Denver
       June 2009

    Purpose
    =======

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

    Arguments
    =========

    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 (see Further
            Details).

            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 (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 >= 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

Further Details
===============

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).

=================================================================== */
```

1.3 Function magma_spotrf

```
int
magma_spotrf(char *uplo, int *n, float *a, int *lda, float *work,
             int *info)
{
/*  -- MAGMA (version 0.1) --
       Univ. of Tennessee, Knoxville
       Univ. of California, Berkeley
       Univ. of Colorado, Denver
       June 2009

    Purpose
    =======

    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.

    Arguments
    =========

    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.

=============================================================== */
```

1.4 Function magma_sgetrf_gpu

```
int
magma_sgetrf_gpu(int *m, int *n, float *a, int *lda,
                 int *ipiv, float *work, int *info)
{
/*  -- MAGMA (version 0.1) --
       Univ. of Tennessee, Knoxville
       Univ. of California, Berkeley
       Univ. of Colorado, Denver
       June 2009

    Purpose
    =======

    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.

    Arguments
    =========

    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.

=================================================================== */
```

1.5 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 )
{
/*  -- MAGMA (version 0.1) --
       Univ. of Tennessee, Knoxville
       Univ. of California, Berkeley
       Univ. of Colorado, Denver
       June 2009

    Purpose
    =======

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

    Arguments
    =========

    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 (see Further
            Details).

    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

Further Details
===============

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).

=================================================================== */
```

1.6 Function magma_spotrf_gpu

```
int
magma_spotrf_gpu(char *uplo, int *n, float *a, int *lda,
                 float *work, int *info)
{
/*  -- MAGMA (version 0.1) --
       Univ. of Tennessee, Knoxville
       Univ. of California, Berkeley
       Univ. of Colorado, Denver
       June 2009

    Purpose
    =======

    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.

    Arguments
    =========

    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.

    ================================================================== */
```

1.7 Function magma_dgetrf

```
int
magma_dgetrf(int *m, int *n, double *a, int *lda,
             int *ipiv, double *work, double *da, int *info)
{
/*  -- MAGMA (version 0.1) --
       Univ. of Tennessee, Knoxville
       Univ. of California, Berkeley
       Univ. of Colorado, Denver
       June 2009

    Purpose
    =======

    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.

    Arguments
    =========

    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.

=================================================================== */
```

1.8 Function magma_dgeqrf

```
int
magma_dgeqrf(int *m, int *n, double *a, int  *lda,  double  *tau,
             double *work, int *lwork, double *da, int *info )
{
/*  -- MAGMA (version 0.1) --
       Univ. of Tennessee, Knoxville
       Univ. of California, Berkeley
       Univ. of Colorado, Denver
       June 2009

    Purpose
    =======

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

    Arguments
    =========

    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 (see Further
            Details).

            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 (see Further
            Details).
```

```
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

Further Details
===============

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).

=================================================================== */
```

1.9 Function magma_dpotrf

```
int
magma_dpotrf(char *uplo, int *n, double *a, int *lda, double *work,
             int *info)
{
/*  -- MAGMA (version 0.1) --
       Univ. of Tennessee, Knoxville
       Univ. of California, Berkeley
       Univ. of Colorado, Denver
       June 2009

    Purpose
    =======

    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.

    Arguments
    =========

    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.

=============================================================== */
```

1.10 Function magma_dgetrf_gpu

```
int
magma_dgetrf_gpu(int *m, int *n, double *a, int *lda,
                 int *ipiv, double *work, int *info)
{
/*  -- MAGMA (version 0.1) --
       Univ. of Tennessee, Knoxville
       Univ. of California, Berkeley
       Univ. of Colorado, Denver
       June 2009

    Purpose
    =======

    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.

    Arguments
    =========

    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.

=================================================================== */
```

1.11 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 )
{
/*  -- MAGMA (version 0.1) --
       Univ. of Tennessee, Knoxville
       Univ. of California, Berkeley
       Univ. of Colorado, Denver
       June 2009

    Purpose
    =======

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

    Arguments
    =========

    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 (see Further
            Details).

    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 (see Further
            Details).

    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

Further Details
===============

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).

=================================================================== */
```

1.12 Function magma_dpotrf_gpu

```
int
magma_dpotrf_gpu(char *uplo, int *n, double *a, int *lda, double *work,
                 int *info)
{
/*  -- MAGMA (version 0.1) --
       Univ. of Tennessee, Knoxville
       Univ. of California, Berkeley
       Univ. of Colorado, Denver
       June 2009

    Purpose
    =======

    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.

    Arguments
    =========

    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.

    ================================================================== */
```

Chapter 2

Use

2.1 Hardware specifications

MAGMA version 0.1 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 [4]. 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.

2.2 Software specifications

MAGMA version 0.1 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 2.3). 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

2.3 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 3

Performance

Here we give the reference performance results using MAGMA version 0.1 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 [3]. 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.

3.1 Single precision

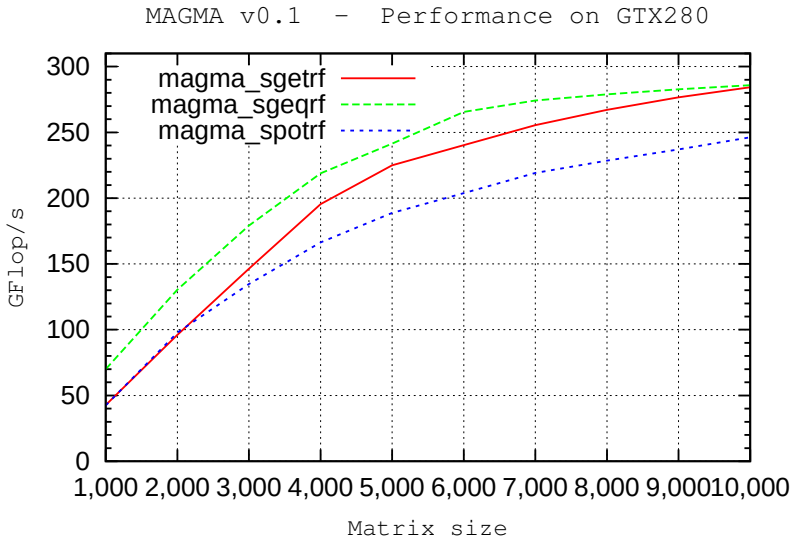

Figure 3.1: Performance of the **CPU interface** one-sided factorizations.

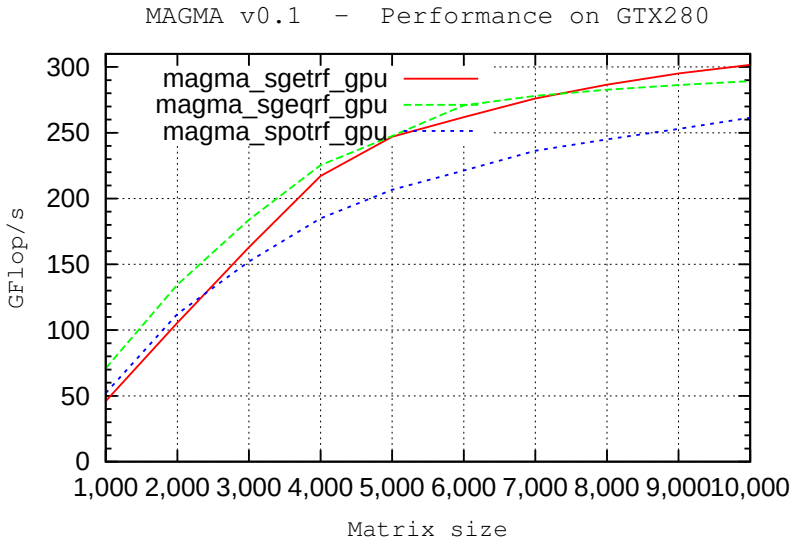

Figure 3.2: Performance of the **GPU interface** one-sided factorizations.

3.2 Double precision

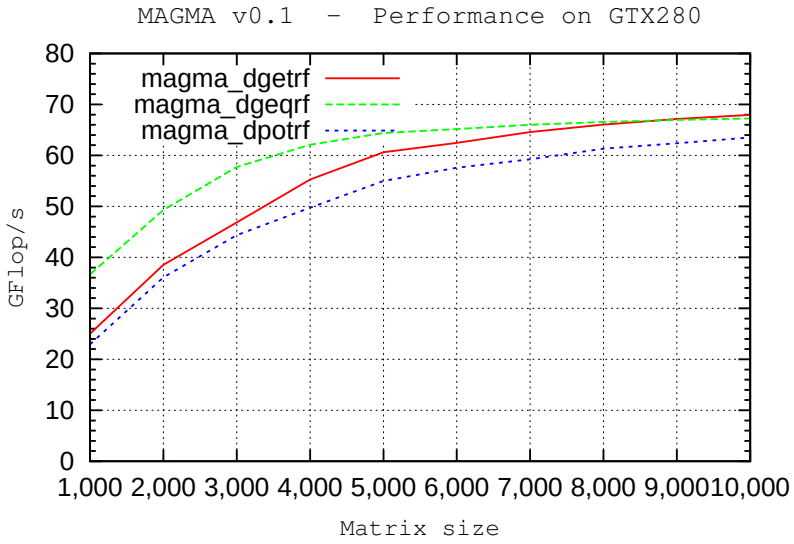

Figure 3.3: Performance of the **CPU interface** one-sided factorizations.

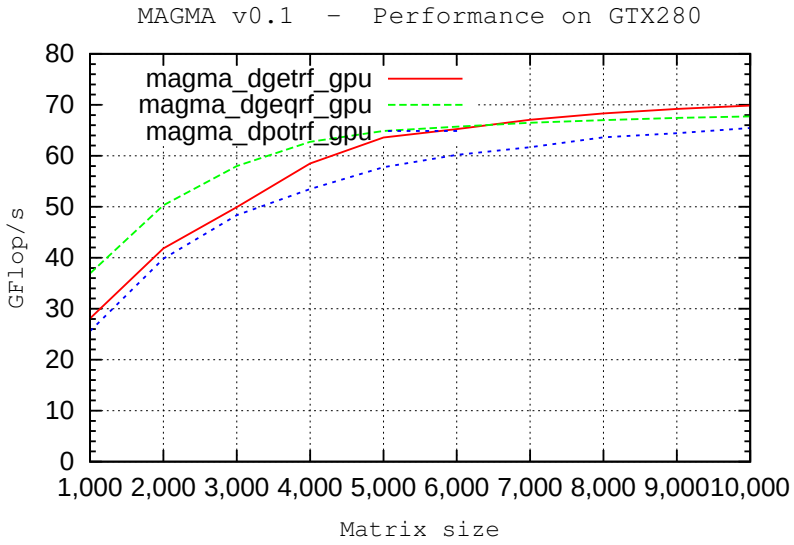

Figure 3.4: Performance of the **GPU interface** one-sided factorizations.

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] M. Baboulin, J. Demmel, J. Dongarra, S. Tomov, and V. Volkov, *Enhancing the performance of dense linear algebra solvers on GPUs [in the MAGMA project]*, Poster at supercomputing 08, November 18, 2009.

[2] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, *Communication-optimal parallel and sequential qr and lu factorizations*, UC Berkeley EECS Technical Report EECS-2008-89, submitted to SIAM J. Sci. Comp., 2008.

[3] Yinan Li, Jack Dongarra, and Stanimire Tomov, *A note on auto-tuning GEMM for GPUs.*, Lecture Notes in Computer Science, vol. 5544, Springer, 2009.

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

[5] 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.

[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.