PLASMA

2.3.0

Generated by Doxygen 1.6.3

Tue Nov 30 18:44:57 2010

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Data Type Index

2.1 Data Types List

Here are the data types with brief descriptions:

Chapter 3

Module Documentation

3.1 Simple Interface - Double Complex

Functions/Subroutines

- int PLASMA_zcgels (PLASMA_enum trans, int M, int N, int NRHS, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗B, int LDB, PLASMA_Complex64_t ∗X, int LDX, int ∗ITER)
- int PLASMA_zcgesv (int N, int NRHS, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗B, int LDB, PLASMA_Complex64_t ∗X, int LDX, int ∗ITER)
- int PLASMA_zcposv (PLASMA_enum uplo, int N, int NRHS, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗B, int LDB, PLASMA_Complex64_t ∗X, int LDX, int ∗ITER)
- int PLASMA_zcungesv (PLASMA_enum trans, int N, int NRHS, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗B, int LDB, PLASMA_Complex64_t ∗X, int LDX, int ∗ITER)
- int PLASMA_zgelqf (int M, int N, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗T)
- int PLASMA_zgelqs (int M, int N, int NRHS, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗T, PLASMA_Complex64_t ∗B, int LDB)
- int PLASMA_zgels (PLASMA_enum trans, int M, int N, int NRHS, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗T, PLASMA_Complex64_t ∗B, int LDB)
- int PLASMA_zgemm (PLASMA_enum transA, PLASMA_enum transB, int M, int N, int K, PLASMA_Complex64_t alpha, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗B, int LDB, PLASMA_Complex64_t beta, PLASMA_Complex64_t ∗C, int LDC)
- int PLASMA_zgeqrf (int M, int N, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗T)
- int PLASMA_zgeqrs (int M, int N, int NRHS, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗T, PLASMA_Complex64_t ∗B, int LDB)
- int PLASMA_zgesv (int N, int NRHS, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗L, int ∗IPIV, PLASMA_Complex64_t ∗B, int LDB)
- int PLASMA_zgetrf (int M, int N, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗L, int ∗IPIV)
- int PLASMA_zgetrs (PLASMA_enum trans, int N, int NRHS, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗L, int ∗IPIV, PLASMA_Complex64_t ∗B, int LDB)
- int PLASMA_zhemm (PLASMA_enum side, PLASMA_enum uplo, int M, int N, PLASMA_Complex64_t alpha, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗B, int LDB, PLASMA_Complex64_t beta, PLASMA_Complex64_t ∗C, int LDC)

- int PLASMA_zher2k (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, PLASMA_Complex64_t alpha, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗B, int LDB, double beta, PLASMA_Complex64_t ∗C, int LDC)
- int PLASMA_zherk (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, double alpha, PLASMA_Complex64_t ∗A, int LDA, double beta, PLASMA_Complex64_t ∗C, int LDC)
- double PLASMA_zlange (PLASMA_enum norm, int M, int N, PLASMA_Complex64_t ∗A, int LDA, double ∗work)
- double PLASMA_zlanhe (PLASMA_enum norm, PLASMA_enum uplo, int N, PLASMA_Complex64_t ∗A, int LDA, double ∗work)
- double PLASMA_zlansy (PLASMA_enum norm, PLASMA_enum uplo, int N, PLASMA_Complex64_t ∗A, int LDA, double ∗work)
- int PLASMA_zlauum (PLASMA_enum uplo, int N, PLASMA_Complex64_t ∗A, int LDA)
- int PLASMA_zplghe (double bump, int N, PLASMA_Complex64_t ∗A, int LDA, unsigned long long int seed)
- int PLASMA_zplgsy (PLASMA_Complex64_t bump, int N, PLASMA_Complex64_t ∗A, int LDA, unsigned long long int seed)
- int PLASMA_zplrnt (int M, int N, PLASMA_Complex64_t ∗A, int LDA, unsigned long long int seed)
- int PLASMA_zposv (PLASMA_enum uplo, int N, int NRHS, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗B, int LDB)
- int PLASMA_zpotrf (PLASMA_enum uplo, int N, PLASMA_Complex64_t ∗A, int LDA)
- int PLASMA_zpotri (PLASMA_enum uplo, int N, PLASMA_Complex64_t ∗A, int LDA)
- int PLASMA_zpotrs (PLASMA_enum uplo, int N, int NRHS, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗B, int LDB)
- int PLASMA_zsymm (PLASMA_enum side, PLASMA_enum uplo, int M, int N, PLASMA_Complex64_t alpha, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗B, int LDB, PLASMA_Complex64_t beta, PLASMA_Complex64_t ∗C, int LDC)
- int PLASMA_zsyr2k (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, PLASMA_Complex64_t alpha, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗B, int LDB, PLASMA_Complex64_t beta, PLASMA_Complex64_t ∗C, int LDC)
- int PLASMA_zsyrk (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, PLASMA_Complex64_t alpha, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t beta, PLASMA_Complex64_t ∗C, int LDC)
- int PLASMA_ztrmm (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, int N, int NRHS, PLASMA_Complex64_t alpha, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗B, int LDB)
- int PLASMA_ztrsm (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, int N, int NRHS, PLASMA_Complex64_t alpha, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗B, int LDB)
- int PLASMA_ztrsmpl (int N, int NRHS, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗L, int ∗IPIV, PLASMA_Complex64_t ∗B, int LDB)
- int PLASMA_ztrtri (PLASMA_enum uplo, PLASMA_enum diag, int N, PLASMA_Complex64_t ∗A, int LDA)
- int PLASMA_zunglq (int M, int N, int K, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗T, PLASMA_Complex64_t ∗B, int LDB)
- int PLASMA_zungqr (int M, int N, int K, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗T, PLASMA_Complex64_t ∗Q, int LDQ)
- int PLASMA_zunmlq (PLASMA_enum side, PLASMA_enum trans, int M, int N, int K, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗T, PLASMA_Complex64_t ∗B, int LDB)
- int PLASMA_zunmqr (PLASMA_enum side, PLASMA_enum trans, int M, int N, int K, PLASMA_Complex64_t ∗A, int LDA, PLASMA_Complex64_t ∗T, PLASMA_Complex64_t ∗B, int LDB)

- int PLASMA_zLapack_to_Tile (PLASMA_Complex64_t *Af77, int LDA, PLASMA_desc *A)
- int PLASMA_zTile_to_Lapack (PLASMA_desc *A, PLASMA_Complex64_t *Af77, int LDA)

3.1.1 Detailed Description

This is the group of double complex functions using the simple user interface.

3.1.2 Function/Subroutine Documentation

3.1.2.1 int PLASMA_zcgels (PLASMA_enum *trans*, int *M*, int *N*, int *NRHS*, PLASMA_Complex64_t * *A*, int *LDA*, PLASMA_Complex64_t * *B*, int *LDB*, PLASMA_Complex64_t * *X*, int *LDX*, int * *ITER*)

PLASMA_zcgels - Solves overdetermined or underdetermined linear systems involving an M-by-N matrix A using the QR or the LQ factorization of A. It is assumed that A has full rank. The following options are provided:

trans = PlasmaNoTrans and M >= N: find the least squares solution of an overdetermined system, i.e., solve the least squares problem: minimize || B - A*X ||.

trans = PlasmaNoTrans and M < N: find the minimum norm solution of an underdetermined system A * X = B.

Several right hand side vectors B and solution vectors X can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

PLASMA_zcgels first attempts to factorize the matrix in COMPLEX and use this factorization within an iterative refinement procedure to produce a solution with COMPLEX*16 normwise backward error quality (see below). If the approach fails the method switches to a COMPLEX*16 factorization and solve.

The iterative refinement is not going to be a winning strategy if the ratio COMPLEX performance over COMPLEX*16 performance is too small. A reasonable strategy should take the number of right-hand sides and the size of the matrix into account. This might be done with a call to ILAENV in the future. Up to now, we always try iterative refinement.

The iterative refinement process is stopped if ITER > ITERMAX or for all the RHS we have: RNRM < N*XNRM*ANRM*EPS*BWDMAX where:

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

Actually, in its current state (PLASMA 2.1.0), the test is slightly relaxed.

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

We follow Bjorck's algorithm proposed in "Iterative Refinement of Linear Least Squares solutions I", BIT, 7:257-278, 1967.

Parameters

← ***trans*** Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaConjTrans: the linear system involves A**H. Currently only PlasmaNoTrans is supported.

← ***M*** The number of rows of the matrix A. M >= 0.

← ***N*** The number of columns of the matrix A. N >= 0.

← ***NRHS*** The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

← ***A*** The M-by-N matrix A. This matrix is not modified.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

← ***B*** The M-by-NRHS matrix B of right hand side vectors, stored columnwise. Not modified.

← ***LDB*** The leading dimension of the array B. LDB >= MAX(1,M,N).

→ ***X*** If return value = 0, the solution vectors, stored columnwise. if M >= N, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if M < N, rows 1 to N of B contain the minimum norm solution vectors;

← ***LDX*** The leading dimension of the array B. LDB >= MAX(1,M,N).

→ ***ITER*** The number of the current iteration in the iterative refinement process

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_zcgels_Tile
PLASMA_zcgels_Tile_Async
PLASMA_dsgels
PLASMA_zgels

3.1.2.2 int PLASMA_zcgesv (int *N*, int *NRHS*, PLASMA_Complex64_t ∗ *A*, int *LDA*, PLASMA_Complex64_t ∗ *B*, int *LDB*, PLASMA_Complex64_t ∗ *X*, int *LDX*, int ∗ *ITER*)

PLASMA_zcgesv - Computes the solution to a system of linear equations A ∗ X = B, where A is an N-by-N matrix and X and B are N-by-NRHS matrices.

PLASMA_zcgesv first attempts to factorize the matrix in COMPLEX and use this factorization within an iterative refinement procedure to produce a solution with COMPLEX∗16 normwise backward error quality (see below). If the approach fails the method switches to a COMPLEX∗16 factorization and solve.

The iterative refinement is not going to be a winning strategy if the ratio COMPLEX performance over COMPLEX∗16 performance is too small. A reasonable strategy should take the number of right-hand sides and the size of the matrix into account. This might be done with a call to ILAENV in the future. Up to now, we always try iterative refinement.

The iterative refinement process is stopped if ITER > ITERMAX or for all the RHS we have: RNRM < N∗XNRM∗ANRM∗EPS∗BWDMAX where:

- ITER is the number of the current iteration in the iterative refinement process
- RNRM is the infinity-norm of the residual

- XNRM is the infinity-norm of the solution
- ANRM is the infinity-operator-norm of the matrix A
- EPS is the machine epsilon returned by DLAMCH('Epsilon').

Actually, in its current state (PLASMA 2.1.0), the test is slightly relaxed.

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

Parameters

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

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

← ***A*** The N-by-N coefficient matrix A. This matrix is not modified.

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

← ***B*** The N-by-NRHS matrix of right hand side matrix B.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

→ ***X*** If return value = 0, the N-by-NRHS solution matrix X.

← ***LDX*** The leading dimension of the array B. LDX >= max(1,N).

→ ***ITER*** The number of the current iteration in the iterative refinement process

Returns

Return values

PLASMA_SUCCESS successful exit

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

>0 if i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

PLASMA_zcgesv_Tile
PLASMA_zcgesv_Tile_Async
PLASMA_dsgesv
PLASMA_zgesv

3.1.2.3 int PLASMA_zcposv (PLASMA_enum *uplo*, int *N*, int *NRHS*, PLASMA_Complex64_t ∗ *A*, int *LDA*, PLASMA_Complex64_t ∗ *B*, int *LDB*, PLASMA_Complex64_t ∗ *X*, int *LDX*, int ∗ *ITER*)

PLASMA_zcposv - Computes the solution to a system of linear equations A ∗ X = B, where A is an N-by-N symmetric positive definite (or Hermitian positive definite in the complex case) matrix and X and B are N-by-NRHS matrices. The Cholesky decomposition is used to factor A as

A = U∗∗H ∗ U, if uplo = PlasmaUpper, or A = L ∗ L∗∗H, if uplo = PlasmaLower,

where U is an upper triangular matrix and L is a lower triangular matrix. The factored form of A is then used to solve the system of equations A ∗ X = B.

PLASMA_zcposv first attempts to factorize the matrix in COMPLEX and use this factorization within an iterative refinement procedure to produce a solution with COMPLEX∗16 normwise backward error quality (see below). If the approach fails the method switches to a COMPLEX∗16 factorization and solve.

The iterative refinement is not going to be a winning strategy if the ratio COMPLEX performance over COMPLEX∗16 performance is too small. A reasonable strategy should take the number of right-hand sides and the size of the matrix into account. This might be done with a call to ILAENV in the future. Up to now, we always try iterative refinement.

The iterative refinement process is stopped if ITER > ITERMAX or for all the RHS we have: RNRM < N∗XNRM∗ANRM∗EPS∗BWDMAX where:

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

Actually, in its current state (PLASMA 2.1.0), the test is slightly relaxed.

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

Parameters

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

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

← ***A*** The N-by-N symmetric positive definite (or Hermitian) coefficient matrix A. If uplo = PlasmaUpper, 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. This matrix is not modified.

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

← ***B*** The N-by-NRHS matrix of right hand side matrix B.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

→ ***X*** If return value = 0, the N-by-NRHS solution matrix X.

← ***LDX*** The leading dimension of the array B. LDX >= max(1,N).

→ ***ITER*** The number of the current iteration in the iterative refinement process

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

>***0*** if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_zcposv_Tile
PLASMA_zcposv_Tile_Async
PLASMA_dsposv
PLASMA_zposv

3.1.2.4 int PLASMA_zcungesv (PLASMA_enum *trans*, int *N*, int *NRHS*, PLASMA_Complex64_t ∗ *A*, int *LDA*, PLASMA_Complex64_t ∗ *B*, int *LDB*, PLASMA_Complex64_t ∗ *X*, int *LDX*, int ∗ *ITER*)

PLASMA_zcungesv - Solves overdetermined or underdetermined linear systems involving an M-by-N matrix A using the QR or the LQ factorization of A. It is assumed that A has full rank. The following options are provided:

trans = PlasmaNoTrans and M >= N: find the least squares solution of an overdetermined system, i.e., solve the least squares problem: minimize || B - A∗X ||.

trans = PlasmaNoTrans and M < N: find the minimum norm solution of an underdetermined system A ∗ X = B.

Several right hand side vectors B and solution vectors X can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

PLASMA_zcungesv first attempts to factorize the matrix in COMPLEX and use this factorization within an iterative refinement procedure to produce a solution with COMPLEX∗16 normwise backward error quality (see below). If the approach fails the method switches to a COMPLEX∗16 factorization and solve.

The iterative refinement is not going to be a winning strategy if the ratio COMPLEX performance over COMPLEX∗16 performance is too small. A reasonable strategy should take the number of right-hand sides and the size of the matrix into account. This might be done with a call to ILAENV in the future. Up to now, we always try iterative refinement.

The iterative refinement process is stopped if ITER > ITERMAX or for all the RHS we have: RNRM < N∗XNRM∗ANRM∗EPS∗BWDMAX where:

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

Actually, in its current state (PLASMA 2.1.0), the test is slightly relaxed.

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

We follow Bjorck's algorithm proposed in "Iterative Refinement of Linear Least Squares solutions I", BIT, 7:257-278, 1967.4

Parameters

← ***trans*** Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaConjTrans: the linear system involves A∗∗H. Currently only PlasmaNoTrans is supported.

← ***M*** The number of rows of the matrix A. M >= 0.

$\leftarrow$ ***N*** The number of columns of the matrix A. N >= 0.

$\leftarrow$ ***NRHS*** The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

$\leftarrow$ ***A*** The M-by-N matrix A. This matrix is not modified.

$\leftarrow$ ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

$\leftarrow$ ***B*** The M-by-NRHS matrix B of right hand side vectors, stored columnwise. Not modified.

$\leftarrow$ ***LDB*** The leading dimension of the array B. LDB >= MAX(1,M,N).

$\rightarrow$ ***X*** If return value = 0, the solution vectors, stored columnwise. if M >= N, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if M < N, rows 1 to N of B contain the minimum norm solution vectors;

$\leftarrow$ ***LDX*** The leading dimension of the array B. LDB >= MAX(1,M,N).

$\rightarrow$ ***ITER*** The number of the current iteration in the iterative refinement process

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_zcungesv_Tile
PLASMA_zcungesv_Tile_Async
PLASMA_dsungesv
PLASMA_zgels

3.1.2.5 int PLASMA_zgelqf (int *M*, int *N*, PLASMA_Complex64_t ∗ *A*, int *LDA*, PLASMA_Complex64_t ∗ *T*)

PLASMA_zgelqf - Computes the tile LQ factorization of a complex M-by-N matrix A: A = L ∗ Q.

Parameters

$\leftarrow$ ***M*** The number of rows of the matrix A. M >= 0.

$\leftarrow$ ***N*** The number of columns of the matrix A. N >= 0.

$\leftrightarrow$ ***A*** On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the m-by-min(M,N) lower trapezoidal matrix L (L is lower triangular if M <= N); the elements above the diagonal represent the unitary matrix Q as a product of elementary reflectors, stored by tiles.

$\leftarrow$ ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

$\rightarrow$ ***T*** On exit, auxiliary factorization data, required by PLASMA_zgelqs to solve the system of equations.

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_zgelqf_Tile
PLASMA_zgelqf_Tile_Async
PLASMA_cgelqf
PLASMA_dgelqf
PLASMA_sgelqf
PLASMA_zgelqs

3.1.2.6 int PLASMA_zgelqs (int *M*, int *N*, int *NRHS*, PLASMA_Complex64_t $*$ *A*, int *LDA*, PLASMA_Complex64_t $*$ *T*, PLASMA_Complex64_t $*$ *B*, int *LDB*)

PLASMA_zgelqs - Compute a minimum-norm solution min || A∗X - B || using the LQ factorization A = L∗Q computed by PLASMA_zgelqf.

Parameters

← ***M*** The number of rows of the matrix A. M >= 0.

← ***N*** The number of columns of the matrix A. N >= M >= 0.

← ***NRHS*** The number of columns of B. NRHS >= 0.

← ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_zgelqf.

← ***LDA*** The leading dimension of the array A. LDA >= M.

← ***T*** Auxiliary factorization data, computed by PLASMA_zgelqf.

↔ ***B*** On entry, the M-by-NRHS right hand side matrix B. On exit, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= N.

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_zgelqs_Tile
PLASMA_zgelqs_Tile_Async
PLASMA_cgelqs
PLASMA_dgelqs
PLASMA_sgelqs
PLASMA_zgelqf

3.1.2.7 int PLASMA_zgels (PLASMA_enum *trans*, int *M*, int *N*, int *NRHS*, PLASMA_Complex64_t * *A*, int *LDA*, PLASMA_Complex64_t * *T*, PLASMA_Complex64_t * *B*, int *LDB*)

PLASMA_zgels - solves overdetermined or underdetermined linear systems involving an M-by-N matrix A using the QR or the LQ factorization of A. It is assumed that A has full rank. The following options are provided:

trans = PlasmaNoTrans and M >= N: find the least squares solution of an overdetermined system, i.e., solve the least squares problem: minimize || B - A*X ||.

trans = PlasmaNoTrans and M < N: find the minimum norm solution of an underdetermined system A * X = B.

Several right hand side vectors B and solution vectors X can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

Parameters

← ***trans*** Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaConjTrans: the linear system involves A**H. Currently only PlasmaNoTrans is supported.

← ***M*** The number of rows of the matrix A. M >= 0.

← ***N*** The number of columns of the matrix A. N >= 0.

← ***NRHS*** The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

↔ ***A*** On entry, the M-by-N matrix A. On exit, if M >= N, A is overwritten by details of its QR factorization as returned by PLASMA_zgeqrf; if M < N, A is overwritten by details of its LQ factorization as returned by PLASMA_zgelqf.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

→ ***T*** On exit, auxiliary factorization data.

↔ ***B*** On entry, the M-by-NRHS matrix B of right hand side vectors, stored columnwise; On exit, if return value = 0, B is overwritten by the solution vectors, stored columnwise: if M >= N, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if M < N, rows 1 to N of B contain the minimum norm solution vectors;

← ***LDB*** The leading dimension of the array B. LDB >= MAX(1,M,N).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_zgels_Tile
PLASMA_zgels_Tile_Async
PLASMA_cgels
PLASMA_dgels
PLASMA_sgels

3.1.2.8 int PLASMA_zgemm (PLASMA_enum *transA*, PLASMA_enum *transB*, int *M*, int *N*, int *K*, PLASMA_Complex64_t *alpha*, PLASMA_Complex64_t * *A*, int *LDA*, PLASMA_Complex64_t * *B*, int *LDB*, PLASMA_Complex64_t *beta*, PLASMA_Complex64_t * *C*, int *LDC*)

PLASMA_zgemm - Performs one of the matrix-matrix operations

$$C = \alpha[op(A) \times op(B)] + \beta C$$

,

where op(X) is one of

op(X) = X or op(X) = X' or op(X) = conjg(X')

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

Parameters

← ***transA*** Specifies whether the matrix A is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is transposed; = PlasmaConjTrans: A is conjugate transposed.

← ***transB*** Specifies whether the matrix B is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: B is not transposed; = PlasmaTrans: B is transposed; = PlasmaConjTrans: B is conjugate transposed.

← ***M*** M specifies the number of rows of the matrix op(A) and of the matrix C. M >= 0.

← ***N*** N specifies the number of columns of the matrix op(B) and of the matrix C. N >= 0.

← ***K*** K specifies the number of columns of the matrix op(A) and the number of rows of the matrix op(B). K >= 0.

← ***alpha*** alpha specifies the scalar alpha

← ***A*** A is a LDA-by-ka matrix, where ka is K when transA = PlasmaNoTrans, and is M otherwise.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

← ***B*** B is a LDB-by-kb matrix, where kb is N when transB = PlasmaNoTrans, and is K otherwise.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

← ***beta*** beta specifies the scalar beta

↔ ***C*** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N matrix (alpha*op(A)*op(B) + beta*C)

← ***LDC*** The leading dimension of the array C. LDC >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zgemm_Tile
PLASMA_cgemm
PLASMA_dgemm
PLASMA_sgemm

3.1.2.9 int PLASMA_zgeqrf (int *M*, int *N*, PLASMA_Complex64_t * *A*, int *LDA*, PLASMA_Complex64_t * *T*)

PLASMA_zgeqrf - Computes the tile QR factorization of a complex M-by-N matrix A: A = Q * R.

Parameters

← ***M*** The number of rows of the matrix A. M >= 0.

← ***N*** The number of columns of the matrix A. N >= 0.

↔ ***A*** 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 represent the unitary matrix Q as a product of elementary reflectors stored by tiles.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

→ ***T*** On exit, auxiliary factorization data, required by PLASMA_zgeqrs to solve the system of equations.

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_zgeqrf_Tile
PLASMA_zgeqrf_Tile_Async
PLASMA_cgeqrf
PLASMA_dgeqrf
PLASMA_sgeqrf
PLASMA_zgeqrs

3.1.2.10 int PLASMA_zgeqrs (int *M*, int *N*, int *NRHS*, PLASMA_Complex64_t * *A*, int *LDA*, PLASMA_Complex64_t * *T*, PLASMA_Complex64_t * *B*, int *LDB*)

PLASMA_zgeqrs - Compute a minimum-norm solution min || A*X - B || using the RQ factorization A = R*Q computed by PLASMA_zgeqrf.

Parameters

← ***M*** The number of rows of the matrix A. M >= 0.

← ***N*** The number of columns of the matrix A. N >= M >= 0.

← ***NRHS*** The number of columns of B. NRHS >= 0.

↔ ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_zgeqrf.

← ***LDA*** The leading dimension of the array A. LDA >= M.

← ***T*** Auxiliary factorization data, computed by PLASMA_zgeqrf.

↔ ***B*** On entry, the m-by-nrhs right hand side matrix B. On exit, the n-by-nrhs solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

$<$*0* if -i, the i-th argument had an illegal value

See also

PLASMA_zgeqrs_Tile
PLASMA_zgeqrs_Tile_Async
PLASMA_cgeqrs
PLASMA_dgeqrs
PLASMA_sgeqrs
PLASMA_zgeqrf

3.1.2.11 int PLASMA_zgesv (int *N*, int *NRHS*, PLASMA_Complex64_t $*$ *A*, int *LDA*, PLASMA_Complex64_t $*$ *L*, int $*$ *IPIV*, PLASMA_Complex64_t $*$ *B*, int *LDB*)

PLASMA_zgesv - Computes the solution to a system of linear equations $A * X = B$, where A is an N-by-N matrix and X and B are N-by-NRHS matrices. The tile LU decomposition with partial tile pivoting and row interchanges is used to factor A. The factored form of A is then used to solve the system of equations $A * X = B$.

Parameters

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

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

$\leftrightarrow$ ***A*** On entry, the N-by-N coefficient matrix A. On exit, the tile L and U factors from the factorization (not equivalent to LAPACK).

$\leftarrow$ ***LDA*** The leading dimension of the array A. LDA >= max(1,N).

$\rightarrow$ ***L*** On exit, auxiliary factorization data, related to the tile L factor, necessary to solve the system of equations.

$\rightarrow$ ***IPIV*** On exit, the pivot indices that define the permutations (not equivalent to LAPACK).

$\leftrightarrow$ ***B*** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

$\leftarrow$ ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

$<$*0* if -i, the i-th argument had an illegal value

$>$*0* if i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

PLASMA_zgesv_Tile
PLASMA_zgesv_Tile_Async
PLASMA_cgesv
PLASMA_dgesv
PLASMA_sgesv
PLASMA_zcgesv

3.1.2.12 int PLASMA_zgetrf (int *M*, int *N*, PLASMA_Complex64_t * *A*, int *LDA*, PLASMA_Complex64_t * *L*, int * *IPIV*)

PLASMA_zgetrf - Computes an LU factorization of a general M-by-N matrix A using the tile LU algorithm with partial tile pivoting with row interchanges.

Parameters

← ***M*** The number of rows of the matrix A. M >= 0.

← ***N*** The number of columns of the matrix A. N >= 0.

↔ ***A*** On entry, the M-by-N matrix to be factored. On exit, the tile factors L and U from the factorization.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

→ ***L*** On exit, auxiliary factorization data, related to the tile L factor, required by PLASMA_zgetrs to solve the system of equations.

→ ***IPIV*** The pivot indices that define the permutations (not equivalent to LAPACK).

Returns

Return values

PLASMA_SUCCESS successful exit

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

>0 if 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.

See also

PLASMA_zgetrf_Tile
PLASMA_zgetrf_Tile_Async
PLASMA_cgetrf
PLASMA_dgetrf
PLASMA_sgetrf
PLASMA_zgetrs

3.1.2.13 int PLASMA_zgetrs (PLASMA_enum *trans*, int *N*, int *NRHS*, PLASMA_Complex64_t * *A*, int *LDA*, PLASMA_Complex64_t * *L*, int * *IPIV*, PLASMA_Complex64_t * *B*, int *LDB*)

PLASMA_zgetrs - Solves a system of linear equations A * X = B, with a general N-by-N matrix A using the tile LU factorization computed by PLASMA_zgetrf.

Parameters

← ***trans*** Intended to specify the the form of the system of equations: = PlasmaNoTrans: A ∗ X = B (No transpose) = PlasmaTrans: A∗∗T ∗ X = B (Transpose) = PlasmaConjTrans: A∗∗H ∗ X = B (Conjugate transpose) Currently only PlasmaNoTrans is supported.

← ***N*** The order of the matrix A. N >= 0.

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

← ***A*** The tile factors L and U from the factorization, computed by PLASMA_zgetrf.

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

← ***L*** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_zgetrf.

← ***IPIV*** The pivot indices from PLASMA_zgetrf (not equivalent to LAPACK).

↔ ***B*** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, the solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

Returns

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

See also

PLASMA_zgetrs_Tile
PLASMA_zgetrs_Tile_Async
PLASMA_cgetrs
PLASMA_dgetrs
PLASMA_sgetrs
PLASMA_zgetrf

3.1.2.14 int PLASMA_zhemm (PLASMA_enum *side*, PLASMA_enum *uplo*, int *M*, int *N*, PLASMA_Complex64_t *alpha*, PLASMA_Complex64_t ∗ *A*, int *LDA*, PLASMA_Complex64_t ∗ *B*, int *LDB*, PLASMA_Complex64_t *beta*, PLASMA_Complex64_t ∗ *C*, int *LDC*)

PLASMA_zhemm - Performs one of the matrix-matrix operations

$$C = \alpha \times A \times B + \beta \times C$$

or

$$C = \alpha \times B \times A + \beta \times C$$

where alpha and beta are scalars, A is an hermitian matrix and B and C are m by n matrices.

Parameters

← ***side*** Specifies whether the hermitian matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

← ***uplo*** Specifies whether the upper or lower triangular part of the hermitian matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the hermitian matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the hermitian matrix A is to be referenced.

← ***M*** Specifies the number of rows of the matrix C. M >= 0.

← ***N*** Specifies the number of columns of the matrix C. N >= 0.

← ***alpha*** Specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,ka).

← ***B*** B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,M).

← ***beta*** Specifies the scalar beta.

↔ ***C*** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

← ***LDC*** The leading dimension of the array C. LDC >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zhemm_Tile
PLASMA_chemm
PLASMA_dhemm
PLASMA_shemm

3.1.2.15 int PLASMA_zher2k (PLASMA_enum *uplo*, PLASMA_enum *trans*, int *N*, int *K*, PLASMA_Complex64_t *alpha*, PLASMA_Complex64_t ∗ *A*, int *LDA*, PLASMA_Complex64_t ∗ *B*, int *LDB*, double *beta*, PLASMA_Complex64_t ∗ *C*, int *LDC*)

PLASMA_zher2k - Performs one of the hermitian rank 2k operations

$$C = \alpha[op(A) \times conjg(op(B)')] + conjg(\alpha)[op(B) \times conjg(op(A)')] + \beta C$$

, or

$$C = \alpha[conjg(op(A)') \times op(B)] + conjg(\alpha)[conjg(op(B)') \times op(A)] + \beta C$$

,

where op(X) is one of

op(X) = X or op(X) = conjg(X')

where alpha and beta are real scalars, C is an n-by-n symmetric matrix and A and B are an n-by-k matrices the first case and k-by-n matrices in the second case.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← ***trans*** Specifies whether the matrix A is transposed or conjugate transposed: = PlasmaNoTrans:

$$C = \alpha[op(A) \times conjg(op(B)')] + conjg(\alpha)[op(B) \times conjg(op(A)')] + \beta C$$

= PlasmaConjTrans:

$$C = \alpha[conjg(op(A)') \times op(B)] + conjg(\alpha)[conjg(op(B)') \times op(A)] + \beta C$$

← ***N*** N specifies the order of the matrix C. N must be at least zero.

← ***K*** K specifies the number of columns of the A and B matrices with trans = PlasmaNoTrans. K specifies the number of rows of the A and B matrices with trans = PlasmaTrans.

← ***alpha*** alpha specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.

← ***LDA*** The leading dimension of the array A. LDA must be at least max(1, N), otherwise LDA must be at least max(1, K).

← ***B*** B is a LDB-by-kb matrix, where kb is K when trans = PlasmaNoTrans, and is N otherwise.

← ***LDB*** The leading dimension of the array B. LDB must be at least max(1, N), otherwise LDB must be at least max(1, K).

← ***beta*** beta specifies the scalar beta.

↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

← ***LDC*** The leading dimension of the array C. LDC >= max(1, N).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zher2k_Tile
PLASMA_cher2k
PLASMA_dher2k
PLASMA_sher2k

3.1.2.16 int PLASMA_zherk (PLASMA_enum *uplo*, PLASMA_enum *trans*, int *N*, int *K*, double *alpha*, PLASMA_Complex64_t ∗ *A*, int *LDA*, double *beta*, PLASMA_Complex64_t ∗ *C*, int *LDC*)

PLASMA_zherk - Performs one of the hermitian rank k operations

$$C = \alpha[op(A) \times conjg(op(A)')] + \beta C$$

,

where op(X) is one of

op(X) = X or op(X) = conjg(X')

where alpha and beta are real scalars, C is an n-by-n hermitian matrix and A is an n-by-k matrix in the first case and a k-by-n matrix in the second case.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← ***trans*** Specifies whether the matrix A is transposed or conjugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaConjTrans: A is conjugate transposed.

← ***N*** N specifies the order of the matrix C. N must be at least zero.

← ***K*** K specifies the number of columns of the matrix op(A).

← ***alpha*** alpha specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.

← ***LDA*** The leading dimension of the array A. LDA must be at least max(1, N), otherwise LDA must be at least max(1, K).

← ***beta*** beta specifies the scalar beta

↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

← ***LDC*** The leading dimension of the array C. LDC >= max(1, N).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zherk_Tile
PLASMA_cherk
PLASMA_dherk
PLASMA_sherk

3.1.2.17 double PLASMA_zlange (PLASMA_enum *norm*, int *M*, int *N*, PLASMA_Complex64_t ∗ *A*, int *LDA*, double ∗ *work*)

PLASMA_zlange returns the value

zlange = (max(abs(A(i,j))), NORM = PlasmaMaxNorm ((norm1(A), NORM = PlasmaOneNorm ((normI(A), NORM = PlasmaInfNorm ((normF(A), NORM = PlasmaFrobeniusNorm

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

← ***norm*** = PlasmaMaxNorm: Max norm = PlasmaOneNorm: One norm = PlasmaInfNorm: Infinity norm = PlasmaFrobeniusNorm: Frobenius norm

← ***M*** The number of rows of the matrix A. M >= 0. When M = 0, the returned value is set to zero.

← ***N*** The number of columns of the matrix A. N >= 0. When N = 0, the returned value is set to zero.

← ***A*** The M-by-N matrix A.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

← ***work*** double precision array of dimension (MAX(1,LWORK)), where LWORK >= M when NORM = PlasmaInfNorm; otherwise, WORK is not referenced.

Returns

Return values

the norm described above.

See also

PLASMA_zlange_Tile
PLASMA_zlange_Tile_Async
PLASMA_clange
PLASMA_dlange
PLASMA_slange

3.1.2.18 double PLASMA_zlanhe (PLASMA_enum *norm*, PLASMA_enum *uplo*, int *N*, PLASMA_Complex64_t ∗ *A*, int *LDA*, double ∗ *work*)

PLASMA_zlanhe returns the value

zlanhe = (max(abs(A(i,j))), NORM = PlasmaMaxNorm ((norm1(A), NORM = PlasmaOneNorm ((normI(A), NORM = PlasmaInfNorm ((normF(A), NORM = PlasmaFrobeniusNorm

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

← ***norm*** = PlasmaMaxNorm: Max norm = PlasmaOneNorm: One norm = PlasmaInfNorm: Infinity norm = PlasmaFrobeniusNorm: Frobenius norm

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***N*** The number of columns/rows of the matrix A. N >= 0. When N = 0, the returned value is set to zero.

← ***A*** The N-by-N matrix A.

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

← ***work*** double precision array of dimension PLASMA_SIZE is PLASMA_STATIC_SCHEDULING is used, and NULL otherwise.

Returns

Return values

the norm described above.

See also

PLASMA_zlanhe_Tile
PLASMA_zlanhe_Tile_Async
PLASMA_clanhe
PLASMA_dlanhe
PLASMA_slanhe

3.1.2.19 double PLASMA_zlansy (PLASMA_enum *norm*, PLASMA_enum *uplo*, int *N*, PLASMA_Complex64_t ∗ *A*, int *LDA*, double ∗ *work*)

PLASMA_zlansy returns the value

zlansy = (max(abs(A(i,j))), NORM = PlasmaMaxNorm ((norm1(A), NORM = PlasmaOneNorm ((normI(A), NORM = PlasmaInfNorm ((normF(A), NORM = PlasmaFrobeniusNorm

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

← ***norm*** = PlasmaMaxNorm: Max norm = PlasmaOneNorm: One norm = PlasmaInfNorm: Infinity norm = PlasmaFrobeniusNorm: Frobenius norm

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***N*** The number of columns/rows of the matrix A. N >= 0. When N = 0, the returned value is set to zero.

← ***A*** The N-by-N matrix A.

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

← ***work*** double precision array of dimension PLASMA_SIZE is PLASMA_STATIC_SCHEDULING is used, and NULL otherwise.

Returns

Return values

the norm described above.

See also

PLASMA_zlansy_Tile

PLASMA_zlansy_Tile_Async
PLASMA_clansy
PLASMA_dlansy
PLASMA_slansy

3.1.2.20 int PLASMA_zLapack_to_Tile (PLASMA_Complex64_t ∗ *Af77*, int *LDA*, PLASMA_desc ∗ *A*)

PLASMA_zLapack_to_Tile - Conversion from LAPACK layout to tile layout.

Parameters

← ***Af77*** LAPACK matrix.

← ***LDA*** The leading dimension of the matrix Af77.

↔ ***A*** Descriptor of the PLASMA matrix in tile layout. If PLASMA_TRANSLATION_MODE is set to PLASMA_INPLACE, A->mat is not used and set to Af77 when returns, else if PLASMA_-TRANSLATION_MODE is set to PLASMA_OUTOFPLACE, A->mat has to be allocated before.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zLapack_to_Tile_Async
PLASMA_zTile_to_Lapack
PLASMA_cLapack_to_Tile
PLASMA_dLapack_to_Tile
PLASMA_sLapack_to_Tile

3.1.2.21 int PLASMA_zlauum (PLASMA_enum *uplo*, int *N*, PLASMA_Complex64_t ∗ *A*, int *LDA*)

PLASMA_zlauum - Computes the product U ∗ U' or L' ∗ L, where the triangular factor U or L is stored in the upper or lower triangular part of the array A.

If UPLO = 'U' or 'u' then the upper triangle of the result is stored, overwriting the factor U in A. If UPLO = 'L' or 'l' then the lower triangle of the result is stored, overwriting the factor L in A.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***N*** The order of the triangular factor U or L. N >= 0.

↔ ***A*** On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U ∗ U'; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L' ∗ L.

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

Returns

Return values

PLASMA_SUCCESS successful exit

$<$***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_zlauum_Tile
PLASMA_zlauum_Tile_Async
PLASMA_clauum
PLASMA_dlauum
PLASMA_slauum
PLASMA_zpotri

3.1.2.22 int PLASMA_zplghe (double *bump*, int *N*, PLASMA_Complex64_t ∗ *A*, int *LDA*, unsigned long long int *seed*)

PLASMA_zplghe - Generate a random hermitian matrix by tiles.

Parameters

← ***bump*** The value to add to the diagonal to be sure to have a positive definite matrix.

← ***N*** The order of the matrix A. N >= 0.

→ ***A*** On exit, The random hermitian matrix A generated.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

$<$***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_zplghe_Tile
PLASMA_zplghe_Tile_Async
PLASMA_cplghe
PLASMA_dplghe
PLASMA_splghe
PLASMA_zplrnt
PLASMA_zplgsy

3.1.2.23 int PLASMA_zplgsy (PLASMA_Complex64_t *bump*, int *N*, PLASMA_Complex64_t ∗ *A*, int *LDA*, unsigned long long int *seed*)

PLASMA_zplgsy - Generate a random hermitian matrix by tiles.

Parameters

← ***bump*** The value to add to the diagonal to be sure to have a positive definite matrix.

← ***N*** The order of the matrix A. N >= 0.

→ ***A*** On exit, The random hermitian matrix A generated.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_zplgsy_Tile
PLASMA_zplgsy_Tile_Async
PLASMA_cplgsy
PLASMA_dplgsy
PLASMA_splgsy
PLASMA_zplrnt
PLASMA_zplgsy

3.1.2.24 int PLASMA_zplrnt (int *M*, int *N*, PLASMA_Complex64_t ∗ *A*, int *LDA*, unsigned long long int *seed*)

PLASMA_zplrnt - Generate a random matrix by tiles.

Parameters

← ***M*** The number of rows of A.

← ***N*** The order of the matrix A. N >= 0.

→ ***A*** On exit, The random matrix A generated.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_zplrnt_Tile
PLASMA_zplrnt_Tile_Async
PLASMA_cplrnt
PLASMA_dplrnt
PLASMA_splrnt
PLASMA_zplghe
PLASMA_zplgsy

3.1.2.25 int PLASMA_zposv (PLASMA_enum *uplo*, int *N*, int *NRHS*, PLASMA_Complex64_t ∗ *A*, int *LDA*, PLASMA_Complex64_t ∗ *B*, int *LDB*)

PLASMA_zposv - Computes the solution to a system of linear equations A ∗ X = B, where A is an N-by-N symmetric positive definite (or Hermitian positive definite in the complex case) matrix and X and B are N-by-NRHS matrices. The Cholesky decomposition is used to factor A as

$$A = \{_{L\times L^H, if uplo = PlasmaLower}^{U^H \times U, if uplo = PlasmaUpper}$$

where U is an upper triangular matrix and L is a lower triangular matrix. The factored form of A is then used to solve the system of equations A ∗ X = B.

Parameters

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

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

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

↔ ***A*** On entry, the symmetric positive definite (or Hermitian) matrix A. If uplo = PlasmaUpper, 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 return value = 0, the factor U or L from the Cholesky factorization A = U∗∗H∗U or A = L∗L∗∗H.

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

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

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

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_zposv_Tile
PLASMA_zposv_Tile_Async
PLASMA_cposv
PLASMA_dposv
PLASMA_sposv

3.1.2.26 int PLASMA_zpotrf (PLASMA_enum *uplo*, int *N*, PLASMA_Complex64_t ∗ *A*, int *LDA*)

PLASMA_zpotrf - Computes the Cholesky factorization of a symmetric positive definite (or Hermitian positive definite in the complex case) matrix A. The factorization has the form

$$A = \{ {}^{U^H \times U, if uplo = PlasmaUpper}_{L \times L^H, if uplo = PlasmaLower}$$

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

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***N*** The order of the matrix A. N >= 0.

↔ ***A*** On entry, the symmetric positive definite (or Hermitian) matrix A. If uplo = PlasmaUpper, 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 return value = 0, the factor U or L from the Cholesky factorization A = U∗∗H∗U or A = L∗L∗∗H.

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

Returns

Return values

PLASMA_SUCCESS successful exit

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

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_zpotrf_Tile
PLASMA_zpotrf_Tile_Async
PLASMA_cpotrf
PLASMA_dpotrf
PLASMA_spotrf
PLASMA_zpotrs

3.1.2.27 int PLASMA_zpotri (PLASMA_enum *uplo*, int *N*, PLASMA_Complex64_t ∗ *A*, int *LDA*)

PLASMA_zpotri - Computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization A = U∗∗H∗U or A = L∗L∗∗H computed by PLASMA_zpotrf.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***N*** The order of the matrix A. N >= 0.

↔ ***A*** On entry, the triangular factor U or L from the Cholesky factorization A = U∗∗H∗U or A = L∗L∗∗H, as computed by PLASMA_zpotrf. On exit, the upper or lower triangle of the (Hermitian) inverse of A, overwriting the input factor U or L.

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

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

>***0*** if i, the (i,i) element of the factor U or L is zero, and the inverse could not be computed.

See also

PLASMA_zpotri_Tile
PLASMA_zpotri_Tile_Async
PLASMA_cpotri
PLASMA_dpotri
PLASMA_spotri
PLASMA_zpotrf

3.1.2.28 int PLASMA_zpotrs (PLASMA_enum *uplo*, int *N*, int *NRHS*, PLASMA_Complex64_t ∗ *A*, int *LDA*, PLASMA_Complex64_t ∗ *B*, int *LDB*)

PLASMA_zpotrs - Solves a system of linear equations A ∗ X = B with a symmetric positive definite (or Hermitian positive definite in the complex case) matrix A using the Cholesky factorization A = U∗∗H∗U or A = L∗L∗∗H computed by PLASMA_zpotrf.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***N*** The order of the matrix A. N >= 0.

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

← ***A*** The triangular factor U or L from the Cholesky factorization A = U∗∗H∗U or A = L∗L∗∗H, computed by PLASMA_zpotrf.

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

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_zpotrs_Tile
PLASMA_zpotrs_Tile_Async
PLASMA_cpotrs
PLASMA_dpotrs
PLASMA_spotrs
PLASMA_zpotrf

3.1.2.29 int PLASMA_zsymm (PLASMA_enum *side*, PLASMA_enum *uplo*, int *M*, int *N*, PLASMA_Complex64_t *alpha*, PLASMA_Complex64_t * *A*, int *LDA*, PLASMA_Complex64_t * *B*, int *LDB*, PLASMA_Complex64_t *beta*, PLASMA_Complex64_t * *C*, int *LDC*)

PLASMA_zsymm - Performs one of the matrix-matrix operations

$$C = \alpha \times A \times B + \beta \times C$$

or

$$C = \alpha \times B \times A + \beta \times C$$

where alpha and beta are scalars, A is an symmetric matrix and B and C are m by n matrices.

Parameters

← ***side*** Specifies whether the symmetric matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

← ***uplo*** Specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the symmetric matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the symmetric matrix A is to be referenced.

← ***M*** Specifies the number of rows of the matrix C. M >= 0.

← ***N*** Specifies the number of columns of the matrix C. N >= 0.

← ***alpha*** Specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,ka).

← ***B*** B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,M).

← ***beta*** Specifies the scalar beta.

↔ ***C*** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

← ***LDC*** The leading dimension of the array C. LDC >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zsymm_Tile
PLASMA_csymm
PLASMA_dsymm
PLASMA_ssymm

3.1.2.30 int PLASMA_zsyr2k (PLASMA_enum *uplo*, PLASMA_enum *trans*, int *N*, int *K*, PLASMA_Complex64_t *alpha*, PLASMA_Complex64_t $*$ *A*, int *LDA*, PLASMA_Complex64_t $*$ *B*, int *LDB*, PLASMA_Complex64_t *beta*, PLASMA_Complex64_t $*$ *C*, int *LDC*)

PLASMA_zsyr2k - Performs one of the symmetric rank 2k operations

$$C = \alpha[op(A) \times conjg(op(B)')] + \alpha[op(B) \times conjg(op(A)')] + \beta C$$

, or

$$C = \alpha[conjg(op(A)') \times op(B)] + \alpha[conjg(op(B)') \times op(A)] + \beta C$$

,

where op(X) is one of

op(X) = X or op(X) = conjg(X')

where alpha and beta are real scalars, C is an n-by-n symmetric matrix and A and B are an n-by-k matrices the first case and k-by-n matrices in the second case.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← ***trans*** Specifies whether the matrix A is transposed or conjugate transposed: = PlasmaNoTrans:

$$C = \alpha[op(A) \times conjg(op(B)')] + \alpha[op(B) \times conjg(op(A)')] + \beta C$$

= PlasmaTrans:

$$C = \alpha[conjg(op(A)') \times op(B)] + \alpha[conjg(op(B)') \times op(A)] + \beta C$$

← ***N*** N specifies the order of the matrix C. N must be at least zero.

← ***K*** K specifies the number of columns of the A and B matrices with trans = PlasmaNoTrans. K specifies the number of rows of the A and B matrices with trans = PlasmaTrans.

← ***alpha*** alpha specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.

← ***LDA*** The leading dimension of the array A. LDA must be at least max(1, N), otherwise LDA must be at least max(1, K).

← ***B*** B is a LDB-by-kb matrix, where kb is K when trans = PlasmaNoTrans, and is N otherwise.

← ***LDB*** The leading dimension of the array B. LDB must be at least max(1, N), otherwise LDB must be at least max(1, K).

← ***beta*** beta specifies the scalar beta.

↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

← ***LDC*** The leading dimension of the array C. LDC >= max(1, N).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zsyr2k_Tile
PLASMA_csyr2k
PLASMA_dsyr2k
PLASMA_ssyr2k

3.1.2.31 int PLASMA_zsyrk (PLASMA_enum *uplo*, PLASMA_enum *trans*, int *N*, int *K*, PLASMA_Complex64_t *alpha*, PLASMA_Complex64_t ∗ *A*, int *LDA*, PLASMA_Complex64_t *beta*, PLASMA_Complex64_t ∗ *C*, int *LDC*)

PLASMA_zsyrk - Performs one of the hermitian rank k operations

$$C = \alpha[op(A) \times conjg(op(A)')] + \beta C$$

,

where op(X) is one of

op(X) = X or op(X) = conjg(X')

where alpha and beta are real scalars, C is an n-by-n hermitian matrix and A is an n-by-k matrix in the first case and a k-by-n matrix in the second case.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← ***trans*** Specifies whether the matrix A is transposed or conjugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans : A is transposed.

← ***N*** N specifies the order of the matrix C. N must be at least zero.

← ***K*** K specifies the number of columns of the matrix op(A).

← ***alpha*** alpha specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.

← ***LDA*** The leading dimension of the array A. LDA must be at least max(1, N), otherwise LDA must be at least max(1, K).

← ***beta*** beta specifies the scalar beta

↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

← ***LDC*** The leading dimension of the array C. LDC >= max(1, N).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zsyrk_Tile
PLASMA_csyrk
PLASMA_dsyrk
PLASMA_ssyrk

3.1.2.32 int PLASMA_zTile_to_Lapack (PLASMA_desc ∗ *A*, PLASMA_Complex64_t ∗ *Af77*, int *LDA*)

PLASMA_Tile_to_Lapack - Conversion from tile layout to LAPACK layout.

Parameters

← ***A*** Descriptor of the PLASMA matrix in tile layout.

↔ ***Af77*** LAPACK matrix. If PLASMA_TRANSLATION_MODE is set to PLASMA_INPLACE, Af77 has to be A->mat, else if PLASMA_TRANSLATION_MODE is set to PLASMA_-OUTOFPLACE, Af77 has to be allocated before.

← ***LDA*** The leading dimension of the matrix Af77.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zTile_to_Lapack_Async
PLASMA_zLapack_to_Tile
PLASMA_cTile_to_Lapack
PLASMA_dTile_to_Lapack
PLASMA_sTile_to_Lapack

3.1.2.33 int PLASMA_ztrmm (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, int *N*, int *NRHS*, PLASMA_Complex64_t *alpha*, PLASMA_Complex64_t ∗ *A*, int *LDA*, PLASMA_Complex64_t ∗ *B*, int *LDB*)

PLASMA_ztrmm - Computes B = alpha∗op(A)∗B or B = alpha∗B∗op(A).

Parameters

← ***side*** Specifies whether A appears on the left or on the right of X: = PlasmaLeft: A∗X = B = PlasmaRight: X∗A = B

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***transA*** Specifies whether the matrix A is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaConjTrans: A is conjugate transposed.

← ***diag*** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A us unit.

← ***N*** The order of the matrix A. N >= 0.

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

← ***A*** The triangular matrix A. If uplo = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If uplo = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If diag = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.

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

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_ztrmm_Tile
PLASMA_ztrmm_Tile_Async
PLASMA_ctrmm
PLASMA_dtrmm
PLASMA_strmm

3.1.2.34 int PLASMA_ztrsm (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, int *N*, int *NRHS*, PLASMA_Complex64_t *alpha*, PLASMA_Complex64_t ∗ *A*, int *LDA*, PLASMA_Complex64_t ∗ *B*, int *LDB*)

PLASMA_ztrsm - Computes triangular solve A∗X = B or X∗A = B.

Parameters

← ***side*** Specifies whether A appears on the left or on the right of X: = PlasmaLeft: A∗X = B = PlasmaRight: X∗A = B

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***transA*** Specifies whether the matrix A is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaConjTrans: A is conjugate transposed.

← ***diag*** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A us unit.

← ***N*** The order of the matrix A. N >= 0.

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

← ***A*** The triangular matrix A. If uplo = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If uplo = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If diag = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.

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

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_ztrsm_Tile
PLASMA_ztrsm_Tile_Async
PLASMA_ctrsm
PLASMA_dtrsm
PLASMA_strsm

3.1.2.35 int PLASMA_ztrsmpl (int *N*, int *NRHS*, PLASMA_Complex64_t ∗ *A*, int *LDA*, PLASMA_Complex64_t ∗ *L*, int ∗ *IPIV*, PLASMA_Complex64_t ∗ *B*, int *LDB*)

PLASMA_ztrsmpl - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix.

Parameters

← ***N*** The order of the matrix A. N >= 0.

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

← ***A*** The tile factor L from the factorization, computed by PLASMA_zgetrf.

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

← ***L*** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_zgetrf.

← ***IPIV*** The pivot indices from PLASMA_zgetrf (not equivalent to LAPACK).

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

$<$***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_ztrsmpl_Tile
PLASMA_ztrsmpl_Tile_Async
PLASMA_ctrsmpl
PLASMA_dtrsmpl
PLASMA_strsmpl
PLASMA_zgetrf

3.1.2.36 int PLASMA_ztrtri (PLASMA_enum *uplo*, PLASMA_enum *diag*, int *N*, PLASMA_Complex64_t $*$ *A*, int *LDA*)

PLASMA_ztrtri - Computes the inverse of a complex upper or lower triangular matrix A.

Parameters

$\leftarrow$ ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

$\leftarrow$ ***diag*** = PlasmaNonUnit: A is non-unit triangular; = PlasmaUnit: A is unit triangular.

$\leftarrow$ ***N*** The order of the matrix A. N >= 0.

$\leftrightarrow$ ***A*** On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1. On exit, the (triangular) inverse of the original matrix, in the same storage format.

$\leftarrow$ ***LDA*** The leading dimension of the array A. LDA >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

$<$***0*** if -i, the i-th argument had an illegal value

$>$***0*** if i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

See also

PLASMA_ztrtri_Tile
PLASMA_ztrtri_Tile_Async
PLASMA_ctrtri
PLASMA_dtrtri
PLASMA_strtri
PLASMA_zpotri

3.1.2.37 int PLASMA_zunglq (int *M*, int *N*, int *K*, PLASMA_Complex64_t * *A*, int *LDA*, PLASMA_Complex64_t * *T*, PLASMA_Complex64_t * *B*, int *LDB*)

PLASMA_zunglq - Generates an M-by-N matrix Q with orthonormal rows, which is defined as the first M rows of a product of the elementary reflectors returned by PLASMA_zgelqf.

Parameters

← ***M*** The number of rows of the matrix Q. M >= 0.

← ***N*** The number of columns of the matrix Q. N >= M.

← ***K*** The number of rows of elementary tile reflectors whose product defines the matrix Q. M >= K >= 0.

← ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_zgelqf.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

← ***T*** Auxiliary factorization data, computed by PLASMA_zgelqf.

→ ***B*** On exit, the M-by-N matrix Q.

← ***LDA*** The leading dimension of the array B. LDB >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

PLASMA_SUCCESS <0 if -i, the i-th argument had an illegal value

See also

PLASMA_zunglq_Tile
PLASMA_zunglq_Tile_Async
PLASMA_cunglq
PLASMA_dunglq
PLASMA_sunglq
PLASMA_zgelqf

3.1.2.38 int PLASMA_zungqr (int *M*, int *N*, int *K*, PLASMA_Complex64_t * *A*, int *LDA*, PLASMA_Complex64_t * *T*, PLASMA_Complex64_t * *Q*, int *LDQ*)

PLASMA_zungqr - Generates an M-by-N matrix Q with orthonormal columns, which is defined as the first N columns of a product of the elementary reflectors returned by PLASMA_zgeqrf.

Parameters

← ***M*** The number of rows of the matrix Q. M >= 0.

← ***N*** The number of columns of the matrix Q. N >= M.

← ***K*** The number of columns of elementary tile reflectors whose product defines the matrix Q. M >= K >= 0.

← ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_zgeqrf.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

← ***T*** Auxiliary factorization data, computed by PLASMA_zgeqrf.

→ ***Q*** On exit, the M-by-N matrix Q.

← ***LDQ*** The leading dimension of the array Q. LDQ >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_zungqr_Tile
PLASMA_zungqr_Tile_Async
PLASMA_cungqr
PLASMA_dungqr
PLASMA_sungqr
PLASMA_zgeqrf

3.1.2.39 int PLASMA_zunmlq (PLASMA_enum *side*, PLASMA_enum *trans*, int *M*, int *N*, int *K*, PLASMA_Complex64_t ∗ *A*, int *LDA*, PLASMA_Complex64_t ∗ *T*, PLASMA_Complex64_t ∗ *B*, int *LDB*)

PLASMA_zunmlq - overwrites the general M-by-N matrix C with Q∗C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_zgelqf. Q is of order M.

Parameters

← ***side*** Intended usage: = PlasmaLeft: apply Q or Q∗∗H from the left; = PlasmaRight: apply Q or Q∗∗H from the right. Currently only PlasmaLeft is supported.

← ***trans*** Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaConjTrans: conjugate transpose, apply Q∗∗H. Currently only PlasmaConjTrans is supported.

← ***M*** The number of rows of the matrix C. M >= 0.

← ***N*** The number of columns of the matrix C. N >= 0.

← ***K*** The number of rows of elementary tile reflectors whose product defines the matrix Q. M >= K >= 0.

← ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_zgelqf.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,K).

← ***T*** Auxiliary factorization data, computed by PLASMA_zgelqf.

↔ ***B*** On entry, the M-by-N matrix B. On exit, B is overwritten by Q∗B or Q∗∗H∗B.

← ***LDB*** The leading dimension of the array C. LDC >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_zunmlq_Tile
PLASMA_zunmlq_Tile_Async
PLASMA_cunmlq
PLASMA_dunmlq
PLASMA_sunmlq
PLASMA_zgelqf

3.1.2.40 int PLASMA_zunmqr (PLASMA_enum *side*, PLASMA_enum *trans*, int *M*, int *N*, int *K*, PLASMA_Complex64_t ∗ *A*, int *LDA*, PLASMA_Complex64_t ∗ *T*, PLASMA_Complex64_t ∗ *B*, int *LDB*)

PLASMA_zunmqr - overwrites the general M-by-N matrix C with Q∗C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_zgeqrf. Q is of order M.

Parameters

← ***side*** Intended usage: = PlasmaLeft: apply Q or Q∗∗H from the left; = PlasmaRight: apply Q or Q∗∗H from the right. Currently only PlasmaLeft is supported.

← ***trans*** Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaConjTrans: conjugate transpose, apply Q∗∗H. Currently only PlasmaConjTrans is supported.

← ***M*** The number of rows of the matrix C. M >= 0.

← ***N*** The number of columns of the matrix C. N >= 0.

← ***K*** The number of columns of elementary tile reflectors whose product defines the matrix Q. M >= K >= 0.

← ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_zgeqrf.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M);

← ***T*** Auxiliary factorization data, computed by PLASMA_zgeqrf.

↔ ***B*** On entry, the M-by-N matrix B. On exit, B is overwritten by Q∗B or Q∗∗H∗B.

← ***LDB*** The leading dimension of the array C. LDC >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_zunmqr_Tile
PLASMA_zunmqr_Tile_Async
PLASMA_cunmqr
PLASMA_dunmqr
PLASMA_sunmqr
PLASMA_zgeqrf

3.2 Simple Interface - Single Complex

Functions/Subroutines

- int PLASMA_cgelqf (int M, int N, PLASMA_Complex32_t ∗A, int LDA, PLASMA_Complex32_t ∗T)
- int PLASMA_cgelqs (int M, int N, int NRHS, PLASMA_Complex32_t ∗A, int LDA, PLASMA_-Complex32_t ∗T, PLASMA_Complex32_t ∗B, int LDB)
- int PLASMA_cgels (PLASMA_enum trans, int M, int N, int NRHS, PLASMA_Complex32_t ∗A, int LDA, PLASMA_Complex32_t ∗T, PLASMA_Complex32_t ∗B, int LDB)
- int PLASMA_cgemm (PLASMA_enum transA, PLASMA_enum transB, int M, int N, int K, PLASMA_Complex32_t alpha, PLASMA_Complex32_t ∗A, int LDA, PLASMA_Complex32_t ∗B, int LDB, PLASMA_Complex32_t beta, PLASMA_Complex32_t ∗C, int LDC)
- int PLASMA_cgeqrf (int M, int N, PLASMA_Complex32_t ∗A, int LDA, PLASMA_Complex32_t ∗T)
- int PLASMA_cgeqrs (int M, int N, int NRHS, PLASMA_Complex32_t ∗A, int LDA, PLASMA_-Complex32_t ∗T, PLASMA_Complex32_t ∗B, int LDB)
- int PLASMA_cgesv (int N, int NRHS, PLASMA_Complex32_t ∗A, int LDA, PLASMA_-Complex32_t ∗L, int ∗IPIV, PLASMA_Complex32_t ∗B, int LDB)
- int PLASMA_cgetrf (int M, int N, PLASMA_Complex32_t ∗A, int LDA, PLASMA_Complex32_t ∗L, int ∗IPIV)
- int PLASMA_cgetrs (PLASMA_enum trans, int N, int NRHS, PLASMA_Complex32_t ∗A, int LDA, PLASMA_Complex32_t ∗L, int ∗IPIV, PLASMA_Complex32_t ∗B, int LDB)
- int PLASMA_chemm (PLASMA_enum side, PLASMA_enum uplo, int M, int N, PLASMA_-Complex32_t alpha, PLASMA_Complex32_t ∗A, int LDA, PLASMA_Complex32_t ∗B, int LDB, PLASMA_Complex32_t beta, PLASMA_Complex32_t ∗C, int LDC)
- int PLASMA_cher2k (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, PLASMA_-Complex32_t alpha, PLASMA_Complex32_t ∗A, int LDA, PLASMA_Complex32_t ∗B, int LDB, float beta, PLASMA_Complex32_t ∗C, int LDC)
- int PLASMA_cherk (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, float alpha, PLASMA_Complex32_t ∗A, int LDA, float beta, PLASMA_Complex32_t ∗C, int LDC)
- float PLASMA_clange (PLASMA_enum norm, int M, int N, PLASMA_Complex32_t ∗A, int LDA, float ∗work)
- float PLASMA_clanhe (PLASMA_enum norm, PLASMA_enum uplo, int N, PLASMA_-Complex32_t ∗A, int LDA, float ∗work)
- float PLASMA_clansy (PLASMA_enum norm, PLASMA_enum uplo, int N, PLASMA_-Complex32_t ∗A, int LDA, float ∗work)
- int PLASMA_clauum (PLASMA_enum uplo, int N, PLASMA_Complex32_t ∗A, int LDA)
- int PLASMA_cplghe (float bump, int N, PLASMA_Complex32_t ∗A, int LDA, unsigned long long int seed)
- int PLASMA_cplgsy (PLASMA_Complex32_t bump, int N, PLASMA_Complex32_t ∗A, int LDA, unsigned long long int seed)
- int PLASMA_cplrnt (int M, int N, PLASMA_Complex32_t ∗A, int LDA, unsigned long long int seed)
- int PLASMA_cposv (PLASMA_enum uplo, int N, int NRHS, PLASMA_Complex32_t ∗A, int LDA, PLASMA_Complex32_t ∗B, int LDB)
- int PLASMA_cpotrf (PLASMA_enum uplo, int N, PLASMA_Complex32_t ∗A, int LDA)
- int PLASMA_cpotri (PLASMA_enum uplo, int N, PLASMA_Complex32_t ∗A, int LDA)
- int PLASMA_cpotrs (PLASMA_enum uplo, int N, int NRHS, PLASMA_Complex32_t ∗A, int LDA, PLASMA_Complex32_t ∗B, int LDB)

- int PLASMA_csymm (PLASMA_enum side, PLASMA_enum uplo, int M, int N, PLASMA_Complex32_t alpha, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *B, int LDB, PLASMA_Complex32_t beta, PLASMA_Complex32_t *C, int LDC)
- int PLASMA_csyr2k (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, PLASMA_Complex32_t alpha, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *B, int LDB, PLASMA_Complex32_t beta, PLASMA_Complex32_t *C, int LDC)
- int PLASMA_csyrk (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, PLASMA_Complex32_t alpha, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t beta, PLASMA_Complex32_t *C, int LDC)
- int PLASMA_ctrmm (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, int N, int NRHS, PLASMA_Complex32_t alpha, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *B, int LDB)
- int PLASMA_ctrsm (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, int N, int NRHS, PLASMA_Complex32_t alpha, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *B, int LDB)
- int PLASMA_ctrsmpl (int N, int NRHS, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *L, int *IPIV, PLASMA_Complex32_t *B, int LDB)
- int PLASMA_ctrtri (PLASMA_enum uplo, PLASMA_enum diag, int N, PLASMA_Complex32_t *A, int LDA)
- int PLASMA_cunglq (int M, int N, int K, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *T, PLASMA_Complex32_t *B, int LDB)
- int PLASMA_cungqr (int M, int N, int K, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *T, PLASMA_Complex32_t *Q, int LDQ)
- int PLASMA_cunmlq (PLASMA_enum side, PLASMA_enum trans, int M, int N, int K, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *T, PLASMA_Complex32_t *B, int LDB)
- int PLASMA_cunmqr (PLASMA_enum side, PLASMA_enum trans, int M, int N, int K, PLASMA_Complex32_t *A, int LDA, PLASMA_Complex32_t *T, PLASMA_Complex32_t *B, int LDB)
- int PLASMA_cLapack_to_Tile (PLASMA_Complex32_t *Af77, int LDA, PLASMA_desc *A)
- int PLASMA_cTile_to_Lapack (PLASMA_desc *A, PLASMA_Complex32_t *Af77, int LDA)

3.2.1 Detailed Description

This is the group of single complex functions using the simple user interface.

3.2.2 Function/Subroutine Documentation

3.2.2.1 int PLASMA_cgelqf (int *M*, int *N*, PLASMA_Complex32_t * *A*, int *LDA*, PLASMA_Complex32_t * *T*)

PLASMA_cgelqf - Computes the tile LQ factorization of a complex M-by-N matrix A: A = L * Q.

Parameters

← ***M*** The number of rows of the matrix A. M >= 0.

← ***N*** The number of columns of the matrix A. N >= 0.

↔ ***A*** On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the m-by-min(M,N) lower trapezoidal matrix L (L is lower triangular if M <= N); the elements above the diagonal represent the unitary matrix Q as a product of elementary reflectors, stored by tiles.

$\leftarrow$ ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

$\rightarrow$ ***T*** On exit, auxiliary factorization data, required by PLASMA_cgelqs to solve the system of equations.

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_cgelqf_Tile
PLASMA_cgelqf_Tile_Async
PLASMA_cgelqf
PLASMA_dgelqf
PLASMA_sgelqf
PLASMA_cgelqs

3.2.2.2 int PLASMA_cgelqs (int *M*, int *N*, int *NRHS*, PLASMA_Complex32_t $*$ *A*, int *LDA*, PLASMA_Complex32_t $*$ *T*, PLASMA_Complex32_t $*$ *B*, int *LDB*)

PLASMA_cgelqs - Compute a minimum-norm solution min || A∗X - B || using the LQ factorization A = L∗Q computed by PLASMA_cgelqf.

Parameters

$\leftarrow$ ***M*** The number of rows of the matrix A. M >= 0.

$\leftarrow$ ***N*** The number of columns of the matrix A. N >= M >= 0.

$\leftarrow$ ***NRHS*** The number of columns of B. NRHS >= 0.

$\leftarrow$ ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_cgelqf.

$\leftarrow$ ***LDA*** The leading dimension of the array A. LDA >= M.

$\leftarrow$ ***T*** Auxiliary factorization data, computed by PLASMA_cgelqf.

$\leftrightarrow$ ***B*** On entry, the M-by-NRHS right hand side matrix B. On exit, the N-by-NRHS solution matrix X.

$\leftarrow$ ***LDB*** The leading dimension of the array B. LDB >= N.

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_cgelqs_Tile

PLASMA_cgelqs_Tile_Async
PLASMA_cgelqs
PLASMA_dgelqs
PLASMA_sgelqs
PLASMA_cgelqf

3.2.2.3 int PLASMA_cgels (PLASMA_enum *trans*, int *M*, int *N*, int *NRHS*, PLASMA_Complex32_t ∗ *A*, int *LDA*, PLASMA_Complex32_t ∗ *T*, PLASMA_Complex32_t ∗ *B*, int *LDB*)

PLASMA_cgels - solves overdetermined or underdetermined linear systems involving an M-by-N matrix A using the QR or the LQ factorization of A. It is assumed that A has full rank. The following options are provided:

trans = PlasmaNoTrans and M >= N: find the least squares solution of an overdetermined system, i.e., solve the least squares problem: minimize || B - A∗X ||.

trans = PlasmaNoTrans and M < N: find the minimum norm solution of an underdetermined system A ∗ X = B.

Several right hand side vectors B and solution vectors X can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

Parameters

← ***trans*** Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaConjTrans: the linear system involves A\∗\∗H. Currently only PlasmaNoTrans is supported.

← ***M*** The number of rows of the matrix A. M >= 0.

← ***N*** The number of columns of the matrix A. N >= 0.

← ***NRHS*** The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

↔ ***A*** On entry, the M-by-N matrix A. On exit, if M >= N, A is overwritten by details of its QR factorization as returned by PLASMA_cgeqrf; if M < N, A is overwritten by details of its LQ factorization as returned by PLASMA_cgelqf.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

→ ***T*** On exit, auxiliary factorization data.

↔ ***B*** On entry, the M-by-NRHS matrix B of right hand side vectors, stored columnwise; On exit, if return value = 0, B is overwritten by the solution vectors, stored columnwise: if M >= N, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if M < N, rows 1 to N of B contain the minimum norm solution vectors;

← ***LDB*** The leading dimension of the array B. LDB >= MAX(1,M,N).

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_cgels_Tile
PLASMA_cgels_Tile_Async
PLASMA_cgels
PLASMA_dgels
PLASMA_sgels

3.2.2.4 int PLASMA_cgemm (PLASMA_enum *transA*, PLASMA_enum *transB*, int *M*, int *N*, int *K*, PLASMA_Complex32_t *alpha*, PLASMA_Complex32_t $*$ *A*, int *LDA*, PLASMA_Complex32_t $*$ *B*, int *LDB*, PLASMA_Complex32_t *beta*, PLASMA_Complex32_t $*$ *C*, int *LDC*)

PLASMA_cgemm - Performs one of the matrix-matrix operations

$$C = \alpha[op(A) \times op(B)] + \beta C$$

,

where op(X) is one of

op(X) = X or op(X) = X' or op(X) = conjfg(X')

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

Parameters

← ***transA*** Specifies whether the matrix A is transposed, not transposed or conjfugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is transposed; = PlasmaConjTrans: A is conjfugate transposed.

← ***transB*** Specifies whether the matrix B is transposed, not transposed or conjfugate transposed: = PlasmaNoTrans: B is not transposed; = PlasmaTrans: B is transposed; = PlasmaConjTrans: B is conjfugate transposed.

← ***M*** M specifies the number of rows of the matrix op(A) and of the matrix C. M >= 0.

← ***N*** N specifies the number of columns of the matrix op(B) and of the matrix C. N >= 0.

← ***K*** K specifies the number of columns of the matrix op(A) and the number of rows of the matrix op(B). K >= 0.

← ***alpha*** alpha specifies the scalar alpha

← ***A*** A is a LDA-by-ka matrix, where ka is K when transA = PlasmaNoTrans, and is M otherwise.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

← ***B*** B is a LDB-by-kb matrix, where kb is N when transB = PlasmaNoTrans, and is K otherwise.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

← ***beta*** beta specifies the scalar beta

↔ ***C*** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N matrix (alpha∗op(A)∗op(B) + beta∗C)

← ***LDC*** The leading dimension of the array C. LDC >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_cgemm_Tile
PLASMA_cgemm
PLASMA_dgemm
PLASMA_sgemm

3.2.2.5 int PLASMA_cgeqrf (int *M*, int *N*, PLASMA_Complex32_t $*$ *A*, int *LDA*, PLASMA_Complex32_t $*$ *T*)

PLASMA_cgeqrf - Computes the tile QR factorization of a complex M-by-N matrix A: A = Q $*$ R.

Parameters

$\leftarrow$ ***M*** The number of rows of the matrix A. M >= 0.

$\leftarrow$ ***N*** The number of columns of the matrix A. N >= 0.

$\leftrightarrow$ ***A*** 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 represent the unitary matrix Q as a product of elementary reflectors stored by tiles.

$\leftarrow$ ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

$\rightarrow$ ***T*** On exit, auxiliary factorization data, required by PLASMA_cgeqrs to solve the system of equations.

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_cgeqrf_Tile
PLASMA_cgeqrf_Tile_Async
PLASMA_cgeqrf
PLASMA_dgeqrf
PLASMA_sgeqrf
PLASMA_cgeqrs

3.2.2.6 int PLASMA_cgeqrs (int *M*, int *N*, int *NRHS*, PLASMA_Complex32_t $*$ *A*, int *LDA*, PLASMA_Complex32_t $*$ *T*, PLASMA_Complex32_t $*$ *B*, int *LDB*)

PLASMA_cgeqrs - Compute a minimum-norm solution min || A$*$X - B || using the RQ factorization A = R$*$Q computed by PLASMA_cgeqrf.

Parameters

$\leftarrow$ ***M*** The number of rows of the matrix A. M >= 0.

$\leftarrow$ ***N*** The number of columns of the matrix A. N >= M >= 0.

$\leftarrow$ ***NRHS*** The number of columns of B. NRHS >= 0.

$\leftrightarrow$ ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_cgeqrf.

$\leftarrow$ ***LDA*** The leading dimension of the array A. LDA >= M.

$\leftarrow$ ***T*** Auxiliary factorization data, computed by PLASMA_cgeqrf.

$\leftrightarrow$ ***B*** On entry, the m-by-nrhs right hand side matrix B. On exit, the n-by-nrhs solution matrix X.

$\leftarrow$ ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_cgeqrs_Tile
PLASMA_cgeqrs_Tile_Async
PLASMA_cgeqrs
PLASMA_dgeqrs
PLASMA_sgeqrs
PLASMA_cgeqrf

3.2.2.7 int PLASMA_cgesv (int *N*, int *NRHS*, PLASMA_Complex32_t $*$ *A*, int *LDA*, PLASMA_Complex32_t $*$ *L*, int $*$ *IPIV*, PLASMA_Complex32_t $*$ *B*, int *LDB*)

PLASMA_cgesv - Computes the solution to a system of linear equations A $*$ X = B, where A is an N-by-N matrix and X and B are N-by-NRHS matrices. The tile LU decomposition with partial tile pivoting and row interchanges is used to factor A. The factored form of A is then used to solve the system of equations A $*$ X = B.

Parameters

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

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

$\leftrightarrow$ ***A*** On entry, the N-by-N coefficient matrix A. On exit, the tile L and U factors from the factorization (not equivalent to LAPACK).

$\leftarrow$ ***LDA*** The leading dimension of the array A. LDA >= max(1,N).

$\rightarrow$ ***L*** On exit, auxiliary factorization data, related to the tile L factor, necessary to solve the system of equations.

$\rightarrow$ ***IPIV*** On exit, the pivot indices that define the permutations (not equivalent to LAPACK).

$\leftrightarrow$ ***B*** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

$\leftarrow$ ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

$<$***0*** if -i, the i-th argument had an illegal value

$>$***0*** if i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

PLASMA_cgesv_Tile
PLASMA_cgesv_Tile_Async
PLASMA_cgesv
PLASMA_dgesv
PLASMA_sgesv
PLASMA_ccgesv

3.2.2.8 int PLASMA_cgetrf (int *M*, int *N*, PLASMA_Complex32_t $*$ *A*, int *LDA*, PLASMA_Complex32_t $*$ *L*, int $*$ *IPIV*)

PLASMA_cgetrf - Computes an LU factorization of a general M-by-N matrix A using the tile LU algorithm with partial tile pivoting with row interchanges.

Parameters

$\leftarrow$ ***M*** The number of rows of the matrix A. M >= 0.

$\leftarrow$ ***N*** The number of columns of the matrix A. N >= 0.

$\leftrightarrow$ ***A*** On entry, the M-by-N matrix to be factored. On exit, the tile factors L and U from the factorization.

$\leftarrow$ ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

$\rightarrow$ ***L*** On exit, auxiliary factorization data, related to the tile L factor, required by PLASMA_cgetrs to solve the system of equations.

$\rightarrow$ ***IPIV*** The pivot indices that define the permutations (not equivalent to LAPACK).

Returns

Return values

PLASMA_SUCCESS successful exit

$<$***0*** if -i, the i-th argument had an illegal value

$>$***0*** if 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.

See also

PLASMA_cgetrf_Tile
PLASMA_cgetrf_Tile_Async
PLASMA_cgetrf
PLASMA_dgetrf
PLASMA_sgetrf
PLASMA_cgetrs

3.2.2.9 int PLASMA_cgetrs (PLASMA_enum *trans*, int *N*, int *NRHS*, PLASMA_Complex32_t ∗ *A*, int *LDA*, PLASMA_Complex32_t ∗ *L*, int ∗ *IPIV*, PLASMA_Complex32_t ∗ *B*, int *LDB*)

PLASMA_cgetrs - Solves a system of linear equations A ∗ X = B, with a general N-by-N matrix A using the tile LU factorization computed by PLASMA_cgetrf.

Parameters

← ***trans*** Intended to specify the the form of the system of equations: = PlasmaNoTrans: A ∗ X = B (No transpose) = PlasmaTrans: A∗∗T ∗ X = B (Transpose) = PlasmaConjTrans: A\∗\∗H ∗ X = B (Conjugate transpose) Currently only PlasmaNoTrans is supported.

← ***N*** The order of the matrix A. N >= 0.

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

← ***A*** The tile factors L and U from the factorization, computed by PLASMA_cgetrf.

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

← ***L*** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_cgetrf.

← ***IPIV*** The pivot indices from PLASMA_cgetrf (not equivalent to LAPACK).

↔ ***B*** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, the solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

Returns

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

See also

PLASMA_cgetrs_Tile
PLASMA_cgetrs_Tile_Async
PLASMA_cgetrs
PLASMA_dgetrs
PLASMA_sgetrs
PLASMA_cgetrf

3.2.2.10 int PLASMA_chemm (PLASMA_enum *side*, PLASMA_enum *uplo*, int *M*, int *N*, PLASMA_Complex32_t *alpha*, PLASMA_Complex32_t ∗ *A*, int *LDA*, PLASMA_Complex32_t ∗ *B*, int *LDB*, PLASMA_Complex32_t *beta*, PLASMA_Complex32_t ∗ *C*, int *LDC*)

PLASMA_chemm - Performs one of the matrix-matrix operations

$$C = \alpha \times A \times B + \beta \times C$$

or

$$C = \alpha \times B \times A + \beta \times C$$

where alpha and beta are scalars, A is an hermitian matrix and B and C are m by n matrices.

Parameters

← ***side*** Specifies whether the hermitian matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

← ***uplo*** Specifies whether the upper or lower triangular part of the hermitian matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the hermitian matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the hermitian matrix A is to be referenced.

← ***M*** Specifies the number of rows of the matrix C. M >= 0.

← ***N*** Specifies the number of columns of the matrix C. N >= 0.

← ***alpha*** Specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,ka).

← ***B*** B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,M).

← ***beta*** Specifies the scalar beta.

↔ ***C*** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

← ***LDC*** The leading dimension of the array C. LDC >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_chemm_Tile
PLASMA_chemm
PLASMA_dhemm
PLASMA_shemm

3.2.2.11 int PLASMA_cher2k (PLASMA_enum *uplo*, PLASMA_enum *trans*, int *N*, int *K*, PLASMA_Complex32_t *alpha*, PLASMA_Complex32_t $*$ *A*, int *LDA*, PLASMA_Complex32_t $*$ *B*, int *LDB*, float *beta*, PLASMA_Complex32_t $*$ *C*, int *LDC*)

PLASMA_cher2k - Performs one of the hermitian rank 2k operations

$$C = \alpha[op(A) \times conjfg(op(B)')] + conjfg(\alpha)[op(B) \times conjfg(op(A)')] + \beta C$$

, or

$$C = \alpha[conjfg(op(A)') \times op(B)] + conjfg(\alpha)[conjfg(op(B)') \times op(A)] + \beta C$$

,

where op(X) is one of

op(X) = X or op(X) = conjfg(X')

where alpha and beta are real scalars, C is an n-by-n symmetric matrix and A and B are an n-by-k matrices the first case and k-by-n matrices in the second case.

Parameters

- ← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.
- ← ***trans*** Specifies whether the matrix A is transposed or conjfugate transposed: = PlasmaNoTrans:

  $$C = \alpha[op(A) \times conjfg(op(B)')] + conjfg(\alpha)[op(B) \times conjfg(op(A)')] + \beta C$$

  = PlasmaConjTrans:

  $$C = \alpha[conjfg(op(A)') \times op(B)] + conjfg(\alpha)[conjfg(op(B)') \times op(A)] + \beta C$$

- ← ***N*** N specifies the order of the matrix C. N must be at least zero.
- ← ***K*** K specifies the number of columns of the A and B matrices with trans = PlasmaNoTrans. K specifies the number of rows of the A and B matrices with trans = PlasmaTrans.
- ← ***alpha*** alpha specifies the scalar alpha.
- ← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.
- ← ***LDA*** The leading dimension of the array A. LDA must be at least max(1, N), otherwise LDA must be at least max(1, K).
- ← ***B*** B is a LDB-by-kb matrix, where kb is K when trans = PlasmaNoTrans, and is N otherwise.
- ← ***LDB*** The leading dimension of the array B. LDB must be at least max(1, N), otherwise LDB must be at least max(1, K).
- ← ***beta*** beta specifies the scalar beta.
- ↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.
- ← ***LDC*** The leading dimension of the array C. LDC >= max(1, N).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_cher2k_Tile
PLASMA_cher2k
PLASMA_dher2k
PLASMA_sher2k

3.2.2.12 int PLASMA_cherk (PLASMA_enum *uplo*, PLASMA_enum *trans*, int *N*, int *K*, float *alpha*, PLASMA_Complex32_t ∗ *A*, int *LDA*, float *beta*, PLASMA_Complex32_t ∗ *C*, int *LDC*)

PLASMA_cherk - Performs one of the hermitian rank k operations

$$C = \alpha[op(A) \times conjfg(op(A)')] + \beta C$$

,

where op(X) is one of

op(X) = X or op(X) = conjfg(X')

where alpha and beta are real scalars, C is an n-by-n hermitian matrix and A is an n-by-k matrix in the first case and a k-by-n matrix in the second case.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← ***trans*** Specifies whether the matrix A is transposed or conjfugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaConjTrans: A is conjfugate transposed.

← ***N*** N specifies the order of the matrix C. N must be at least zero.

← ***K*** K specifies the number of columns of the matrix op(A).

← ***alpha*** alpha specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.

← ***LDA*** The leading dimension of the array A. LDA must be at least max(1, N), otherwise LDA must be at least max(1, K).

← ***beta*** beta specifies the scalar beta

↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

← ***LDC*** The leading dimension of the array C. LDC >= max(1, N).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_cherk_Tile
PLASMA_cherk
PLASMA_dherk
PLASMA_sherk

3.2.2.13 float PLASMA_clange (PLASMA_enum *norm*, int *M*, int *N*, PLASMA_Complex32_t ∗ *A*, int *LDA*, float ∗ *work*)

PLASMA_clange returns the value

clange = (max(abs(A(i,j))), NORM = PlasmaMaxNorm ((norm1(A), NORM = PlasmaOneNorm ((normI(A), NORM = PlasmaInfNorm ((normF(A), NORM = PlasmaFrobeniusNorm

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

- ← ***norm*** = PlasmaMaxNorm: Max norm = PlasmaOneNorm: One norm = PlasmaInfNorm: Infinity norm = PlasmaFrobeniusNorm: Frobenius norm
- ← ***M*** The number of rows of the matrix A. M >= 0. When M = 0, the returned value is set to zero.
- ← ***N*** The number of columns of the matrix A. N >= 0. When N = 0, the returned value is set to zero.
- ← ***A*** The M-by-N matrix A.
- ← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).
- ← ***work*** float precision array of dimension (MAX(1,LWORK)), where LWORK >= M when NORM = PlasmaInfNorm; otherwise, WORK is not referenced.

Returns

Return values

the norm described above.

See also

PLASMA_clange_Tile
PLASMA_clange_Tile_Async
PLASMA_clange
PLASMA_dlange
PLASMA_slange

3.2.2.14 float PLASMA_clanhe (PLASMA_enum *norm*, PLASMA_enum *uplo*, int *N*, PLASMA_Complex32_t ∗ *A*, int *LDA*, float ∗ *work*)

PLASMA_clanhe returns the value

clanhe = (max(abs(A(i,j))), NORM = PlasmaMaxNorm ((norm1(A), NORM = PlasmaOneNorm ((normI(A), NORM = PlasmaInfNorm ((normF(A), NORM = PlasmaFrobeniusNorm

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

- ← ***norm*** = PlasmaMaxNorm: Max norm = PlasmaOneNorm: One norm = PlasmaInfNorm: Infinity norm = PlasmaFrobeniusNorm: Frobenius norm
- ← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.
- ← ***N*** The number of columns/rows of the matrix A. N >= 0. When N = 0, the returned value is set to zero.
- ← ***A*** The N-by-N matrix A.

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

← ***work*** float precision array of dimension PLASMA_SIZE is PLASMA_STATIC_SCHEDULING is used, and NULL otherwise.

Returns

Return values

the norm described above.

See also

PLASMA_clanhe_Tile
PLASMA_clanhe_Tile_Async
PLASMA_clanhe
PLASMA_dlanhe
PLASMA_slanhe

3.2.2.15 float PLASMA_clansy (PLASMA_enum *norm*, PLASMA_enum *uplo*, int *N*, PLASMA_Complex32_t ∗ *A*, int *LDA*, float ∗ *work*)

PLASMA_clansy returns the value

clansy = (max(abs(A(i,j))), NORM = PlasmaMaxNorm ((norm1(A), NORM = PlasmaOneNorm ((normI(A), NORM = PlasmaInfNorm ((normF(A), NORM = PlasmaFrobeniusNorm

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

← ***norm*** = PlasmaMaxNorm: Max norm = PlasmaOneNorm: One norm = PlasmaInfNorm: Infinity norm = PlasmaFrobeniusNorm: Frobenius norm

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***N*** The number of columns/rows of the matrix A. N >= 0. When N = 0, the returned value is set to zero.

← ***A*** The N-by-N matrix A.

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

← ***work*** float precision array of dimension PLASMA_SIZE is PLASMA_STATIC_SCHEDULING is used, and NULL otherwise.

Returns

Return values

the norm described above.

See also

PLASMA_clansy_Tile

PLASMA_clansy_Tile_Async
PLASMA_clansy
PLASMA_dlansy
PLASMA_slansy

3.2.2.16 int PLASMA_cLapack_to_Tile (PLASMA_Complex32_t ∗ *Af77*, int *LDA*, PLASMA_desc ∗ *A*)

PLASMA_cLapack_to_Tile - Conversion from LAPACK layout to tile layout.

Parameters

← ***Af77*** LAPACK matrix.

← ***LDA*** The leading dimension of the matrix Af77.

↔ ***A*** Descriptor of the PLASMA matrix in tile layout. If PLASMA_TRANSLATION_MODE is set to PLASMA_INPLACE, A->mat is not used and set to Af77 when returns, else if PLASMA_-TRANSLATION_MODE is set to PLASMA_OUTOFPLACE, A->mat has to be allocated before.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_cLapack_to_Tile_Async
PLASMA_cTile_to_Lapack
PLASMA_cLapack_to_Tile
PLASMA_dLapack_to_Tile
PLASMA_sLapack_to_Tile

3.2.2.17 int PLASMA_clauum (PLASMA_enum *uplo*, int *N*, PLASMA_Complex32_t ∗ *A*, int *LDA*)

PLASMA_clauum - Computes the product U ∗ U' or L' ∗ L, where the triangular factor U or L is stored in the upper or lower triangular part of the array A.

If UPLO = 'U' or 'u' then the upper triangle of the result is stored, overwriting the factor U in A. If UPLO = 'L' or 'l' then the lower triangle of the result is stored, overwriting the factor L in A.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***N*** The order of the triangular factor U or L. N >= 0.

↔ ***A*** On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U ∗ U'; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L' ∗ L.

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

Returns

Return values

PLASMA_SUCCESS successful exit

$<$*0* if -i, the i-th argument had an illegal value

See also

PLASMA_clauum_Tile
PLASMA_clauum_Tile_Async
PLASMA_clauum
PLASMA_dlauum
PLASMA_slauum
PLASMA_cpotri

3.2.2.18 int PLASMA_cplghe (float *bump*, int *N*, PLASMA_Complex32_t ∗ *A*, int *LDA*, unsigned long long int *seed*)

PLASMA_cplghe - Generate a random hermitian matrix by tiles.

Parameters

← ***bump*** The value to add to the diagonal to be sure to have a positive definite matrix.

← ***N*** The order of the matrix A. N >= 0.

→ ***A*** On exit, The random hermitian matrix A generated.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

$<$*0* if -i, the i-th argument had an illegal value

See also

PLASMA_cplghe_Tile
PLASMA_cplghe_Tile_Async
PLASMA_cplghe
PLASMA_dplghe
PLASMA_splghe
PLASMA_cplrnt
PLASMA_cplgsy

3.2.2.19 int PLASMA_cplgsy (PLASMA_Complex32_t *bump*, int *N*, PLASMA_Complex32_t ∗ *A*, int *LDA*, unsigned long long int *seed*)

PLASMA_cplgsy - Generate a random hermitian matrix by tiles.

Parameters

← ***bump*** The value to add to the diagonal to be sure to have a positive definite matrix.

← ***N*** The order of the matrix A. N >= 0.

→ ***A*** On exit, The random hermitian matrix A generated.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_cplgsy_Tile
PLASMA_cplgsy_Tile_Async
PLASMA_cplgsy
PLASMA_dplgsy
PLASMA_splgsy
PLASMA_cplrnt
PLASMA_cplgsy

3.2.2.20 int PLASMA_cplrnt (int *M*, int *N*, PLASMA_Complex32_t * *A*, int *LDA*, unsigned long long int *seed*)

PLASMA_cplrnt - Generate a random matrix by tiles.

Parameters

← ***M*** The number of rows of A.

← ***N*** The order of the matrix A. N >= 0.

→ ***A*** On exit, The random matrix A generated.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_cplrnt_Tile
PLASMA_cplrnt_Tile_Async
PLASMA_cplrnt
PLASMA_dplrnt
PLASMA_splrnt
PLASMA_cplghe
PLASMA_cplgsy

3.2.2.21 int PLASMA_cposv (PLASMA_enum *uplo*, int *N*, int *NRHS*, PLASMA_Complex32_t * *A*, int *LDA*, PLASMA_Complex32_t * *B*, int *LDB*)

PLASMA_cposv - Computes the solution to a system of linear equations A * X = B, where A is an N-by-N symmetric positive definite (or Hermitian positive definite in the complex case) matrix and X and B are N-by-NRHS matrices. The Cholesky decomposition is used to factor A as

$$A = \{^{U^H \times U, if uplo = PlasmaUpper}_{L \times L^H, if uplo = PlasmaLower}$$

where U is an upper triangular matrix and L is a lower triangular matrix. The factored form of A is then used to solve the system of equations A * X = B.

Parameters

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

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

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

↔ ***A*** On entry, the symmetric positive definite (or Hermitian) matrix A. If uplo = PlasmaUpper, 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 return value = 0, the factor U or L from the Cholesky factorization A = U**H*U or A = L*L**H.

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

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

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

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_cposv_Tile
PLASMA_cposv_Tile_Async
PLASMA_cposv
PLASMA_dposv
PLASMA_sposv

3.2.2.22 int PLASMA_cpotrf (PLASMA_enum *uplo*, int *N*, PLASMA_Complex32_t ∗ *A*, int *LDA*)

PLASMA_cpotrf - Computes the Cholesky factorization of a symmetric positive definite (or Hermitian positive definite in the complex case) matrix A. The factorization has the form

$$A = \{^{U^H \times U, if uplo = PlasmaUpper}_{L \times L^H, if uplo = PlasmaLower}$$

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

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***N*** The order of the matrix A. N >= 0.

↔ ***A*** On entry, the symmetric positive definite (or Hermitian) matrix A. If uplo = PlasmaUpper, 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 return value = 0, the factor U or L from the Cholesky factorization A = U\∗\∗H∗U or A = L∗L\∗\∗H.

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

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

>***0*** if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_cpotrf_Tile
PLASMA_cpotrf_Tile_Async
PLASMA_cpotrf
PLASMA_dpotrf
PLASMA_spotrf
PLASMA_cpotrs

3.2.2.23 int PLASMA_cpotri (PLASMA_enum *uplo*, int *N*, PLASMA_Complex32_t ∗ *A*, int *LDA*)

PLASMA_cpotri - Computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization A = U\∗\∗H∗U or A = L∗L\∗\∗H computed by PLASMA_cpotrf.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***N*** The order of the matrix A. N >= 0.

↔ ***A*** On entry, the triangular factor U or L from the Cholesky factorization A = U**H*U or A = L*L**H, as computed by PLASMA_cpotrf. On exit, the upper or lower triangle of the (Hermitian) inverse of A, overwriting the input factor U or L.

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

Returns

Return values

PLASMA_SUCCESS successful exit

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

>0 if i, the (i,i) element of the factor U or L is zero, and the inverse could not be computed.

See also

PLASMA_cpotri_Tile
PLASMA_cpotri_Tile_Async
PLASMA_cpotri
PLASMA_dpotri
PLASMA_spotri
PLASMA_cpotrf

3.2.2.24 int PLASMA_cpotrs (PLASMA_enum *uplo*, int *N*, int *NRHS*, PLASMA_Complex32_t * *A*, int *LDA*, PLASMA_Complex32_t * *B*, int *LDB*)

PLASMA_cpotrs - Solves a system of linear equations A * X = B with a symmetric positive definite (or Hermitian positive definite in the complex case) matrix A using the Cholesky factorization A = U**H*U or A = L*L**H computed by PLASMA_cpotrf.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***N*** The order of the matrix A. N >= 0.

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

← ***A*** The triangular factor U or L from the Cholesky factorization A = U**H*U or A = L*L**H, computed by PLASMA_cpotrf.

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

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_cpotrs_Tile
PLASMA_cpotrs_Tile_Async
PLASMA_cpotrs
PLASMA_dpotrs
PLASMA_spotrs
PLASMA_cpotrf

3.2.2.25 int PLASMA_csymm (PLASMA_enum *side*, PLASMA_enum *uplo*, int *M*, int *N*, PLASMA_Complex32_t *alpha*, PLASMA_Complex32_t ∗ *A*, int *LDA*, PLASMA_Complex32_t ∗ *B*, int *LDB*, PLASMA_Complex32_t *beta*, PLASMA_Complex32_t ∗ *C*, int *LDC*)

PLASMA_csymm - Performs one of the matrix-matrix operations

$$C = \alpha \times A \times B + \beta \times C$$

or

$$C = \alpha \times B \times A + \beta \times C$$

where alpha and beta are scalars, A is an symmetric matrix and B and C are m by n matrices.

Parameters

← ***side*** Specifies whether the symmetric matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

← ***uplo*** Specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the symmetric matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the symmetric matrix A is to be referenced.

← ***M*** Specifies the number of rows of the matrix C. M >= 0.

← ***N*** Specifies the number of columns of the matrix C. N >= 0.

← ***alpha*** Specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,ka).

← ***B*** B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,M).

← ***beta*** Specifies the scalar beta.

↔ ***C*** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

← ***LDC*** The leading dimension of the array C. LDC >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_csymm_Tile
PLASMA_csymm
PLASMA_dsymm
PLASMA_ssymm

3.2.2.26 int PLASMA_csyr2k (PLASMA_enum *uplo*, PLASMA_enum *trans*, int *N*, int *K*, PLASMA_Complex32_t *alpha*, PLASMA_Complex32_t * *A*, int *LDA*, PLASMA_Complex32_t * *B*, int *LDB*, PLASMA_Complex32_t *beta*, PLASMA_Complex32_t * *C*, int *LDC*)

PLASMA_csyr2k - Performs one of the symmetric rank 2k operations

$$C = \alpha[op(A) \times conjfg(op(B)')] + \alpha[op(B) \times conjfg(op(A)')] + \beta C$$

, or

$$C = \alpha[conjfg(op(A)') \times op(B)] + \alpha[conjfg(op(B)') \times op(A)] + \beta C$$

,

where op(X) is one of

op(X) = X or op(X) = conjfg(X')

where alpha and beta are real scalars, C is an n-by-n symmetric matrix and A and B are an n-by-k matrices the first case and k-by-n matrices in the second case.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← ***trans*** Specifies whether the matrix A is transposed or conjfugate transposed: = PlasmaNoTrans:

$$C = \alpha[op(A) \times conjfg(op(B)')] + \alpha[op(B) \times conjfg(op(A)')] + \beta C$$

= PlasmaTrans:

$$C = \alpha[conjfg(op(A)') \times op(B)] + \alpha[conjfg(op(B)') \times op(A)] + \beta C$$

← ***N*** N specifies the order of the matrix C. N must be at least zero.

← ***K*** K specifies the number of columns of the A and B matrices with trans = PlasmaNoTrans. K specifies the number of rows of the A and B matrices with trans = PlasmaTrans.

← ***alpha*** alpha specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.

← ***LDA*** The leading dimension of the array A. LDA must be at least max(1, N), otherwise LDA must be at least max(1, K).

← ***B*** B is a LDB-by-kb matrix, where kb is K when trans = PlasmaNoTrans, and is N otherwise.

← ***LDB*** The leading dimension of the array B. LDB must be at least max(1, N), otherwise LDB must be at least max(1, K).

← ***beta*** beta specifies the scalar beta.

↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

← ***LDC*** The leading dimension of the array C. LDC >= max(1, N).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_csyr2k_Tile
PLASMA_csyr2k
PLASMA_dsyr2k
PLASMA_ssyr2k

3.2.2.27 int PLASMA_csyrk (PLASMA_enum *uplo*, PLASMA_enum *trans*, int *N*, int *K*, PLASMA_Complex32_t *alpha*, PLASMA_Complex32_t ∗ *A*, int *LDA*, PLASMA_Complex32_t *beta*, PLASMA_Complex32_t ∗ *C*, int *LDC*)

PLASMA_csyrk - Performs one of the hermitian rank k operations

$$C = \alpha[op(A) \times conjfg(op(A)')] + \beta C$$

,

where op(X) is one of

op(X) = X or op(X) = conjfg(X')

where alpha and beta are real scalars, C is an n-by-n hermitian matrix and A is an n-by-k matrix in the first case and a k-by-n matrix in the second case.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← ***trans*** Specifies whether the matrix A is transposed or conjfugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans : A is transposed.

← ***N*** N specifies the order of the matrix C. N must be at least zero.

← ***K*** K specifies the number of columns of the matrix op(A).

← ***alpha*** alpha specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.

← ***LDA*** The leading dimension of the array A. LDA must be at least max(1, N), otherwise LDA must be at least max(1, K).

← ***beta*** beta specifies the scalar beta

↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

← ***LDC*** The leading dimension of the array C. LDC >= max(1, N).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_csyrk_Tile
PLASMA_csyrk
PLASMA_dsyrk
PLASMA_ssyrk

3.2.2.28 int PLASMA_cTile_to_Lapack (PLASMA_desc ∗ *A*, PLASMA_Complex32_t ∗ *Af77*, int *LDA*)

PLASMA_Tile_to_Lapack - Conversion from tile layout to LAPACK layout.

Parameters

← ***A*** Descriptor of the PLASMA matrix in tile layout.

↔ ***Af77*** LAPACK matrix. If PLASMA_TRANSLATION_MODE is set to PLASMA_INPLACE, Af77 has to be A->mat, else if PLASMA_TRANSLATION_MODE is set to PLASMA_-OUTOFPLACE, Af77 has to be allocated before.

← ***LDA*** The leading dimension of the matrix Af77.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_cTile_to_Lapack_Async
PLASMA_cLapack_to_Tile
PLASMA_cTile_to_Lapack
PLASMA_dTile_to_Lapack
PLASMA_sTile_to_Lapack

3.2.2.29 int PLASMA_ctrmm (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, int *N*, int *NRHS*, PLASMA_Complex32_t *alpha*, PLASMA_Complex32_t ∗ *A*, int *LDA*, PLASMA_Complex32_t ∗ *B*, int *LDB*)

PLASMA_ctrmm - Computes B = alpha∗op(A)∗B or B = alpha∗B∗op(A).

Parameters

← ***side*** Specifies whether A appears on the left or on the right of X: = PlasmaLeft: A∗X = B = PlasmaRight: X∗A = B

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***transA*** Specifies whether the matrix A is transposed, not transposed or conjfugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaConjTrans: A is conjfugate transposed.

← ***diag*** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A us unit.

← ***N*** The order of the matrix A. N >= 0.

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

← ***A*** The triangular matrix A. If uplo = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If uplo = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If diag = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.

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

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_ctrmm_Tile
PLASMA_ctrmm_Tile_Async
PLASMA_ctrmm
PLASMA_dtrmm
PLASMA_strmm

3.2.2.30 int PLASMA_ctrsm (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, int *N*, int *NRHS*, PLASMA_Complex32_t *alpha*, PLASMA_Complex32_t ∗ *A*, int *LDA*, PLASMA_Complex32_t ∗ *B*, int *LDB*)

PLASMA_ctrsm - Computes triangular solve A∗X = B or X∗A = B.

Parameters

← ***side*** Specifies whether A appears on the left or on the right of X: = PlasmaLeft: A∗X = B = PlasmaRight: X∗A = B

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***transA*** Specifies whether the matrix A is transposed, not transposed or conjfugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaConjTrans: A is conjfugate transposed.

← ***diag*** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A us unit.

← ***N*** The order of the matrix A. N >= 0.

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

← ***A*** The triangular matrix A. If uplo = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If uplo = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If diag = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.

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

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_ctrsm_Tile
PLASMA_ctrsm_Tile_Async
PLASMA_ctrsm
PLASMA_dtrsm
PLASMA_strsm

3.2.2.31 int PLASMA_ctrsmpl (int *N*, int *NRHS*, PLASMA_Complex32_t ∗ *A*, int *LDA*, PLASMA_Complex32_t ∗ *L*, int ∗ *IPIV*, PLASMA_Complex32_t ∗ *B*, int *LDB*)

PLASMA_ctrsmpl - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix.

Parameters

← ***N*** The order of the matrix A. N >= 0.

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

← ***A*** The tile factor L from the factorization, computed by PLASMA_cgetrf.

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

← ***L*** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_cgetrf.

← ***IPIV*** The pivot indices from PLASMA_cgetrf (not equivalent to LAPACK).

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_ctrsmpl_Tile
PLASMA_ctrsmpl_Tile_Async
PLASMA_ctrsmpl
PLASMA_dtrsmpl
PLASMA_strsmpl
PLASMA_cgetrf

3.2.2.32 int PLASMA_ctrtri (PLASMA_enum *uplo*, PLASMA_enum *diag*, int *N*, PLASMA_Complex32_t ∗ *A*, int *LDA*)

PLASMA_ctrtri - Computes the inverse of a complex upper or lower triangular matrix A.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***diag*** = PlasmaNonUnit: A is non-unit triangular; = PlasmaUnit: A is unit triangular.

← ***N*** The order of the matrix A. N >= 0.

↔ ***A*** On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1. On exit, the (triangular) inverse of the original matrix, in the same storage format.

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

Returns

Return values

PLASMA_SUCCESS successful exit

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

>0 if i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

See also

PLASMA_ctrtri_Tile
PLASMA_ctrtri_Tile_Async
PLASMA_ctrtri
PLASMA_dtrtri
PLASMA_strtri
PLASMA_cpotri

3.2.2.33 int PLASMA_cunglq (int *M*, int *N*, int *K*, PLASMA_Complex32_t ∗ *A*, int *LDA*, PLASMA_Complex32_t ∗ *T*, PLASMA_Complex32_t ∗ *B*, int *LDB*)

PLASMA_cunglq - Generates an M-by-N matrix Q with orthonormal rows, which is defined as the first M rows of a product of the elementary reflectors returned by PLASMA_cgelqf.

Parameters

← ***M*** The number of rows of the matrix Q. M >= 0.

← ***N*** The number of columns of the matrix Q. N >= M.

← ***K*** The number of rows of elementary tile reflectors whose product defines the matrix Q. M >= K >= 0.

← ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_cgelqf.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

← ***T*** Auxiliary factorization data, computed by PLASMA_cgelqf.

→ ***B*** On exit, the M-by-N matrix Q.

← ***LDA*** The leading dimension of the array B. LDB >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

PLASMA_SUCCESS <0 if -i, the i-th argument had an illegal value

See also

PLASMA_cunglq_Tile
PLASMA_cunglq_Tile_Async
PLASMA_cunglq
PLASMA_dunglq
PLASMA_sunglq
PLASMA_cgelqf

3.2.2.34 int PLASMA_cungqr (int *M*, int *N*, int *K*, PLASMA_Complex32_t ∗ *A*, int *LDA*, PLASMA_Complex32_t ∗ *T*, PLASMA_Complex32_t ∗ *Q*, int *LDQ*)

PLASMA_cungqr - Generates an M-by-N matrix Q with orthonormal columns, which is defined as the first N columns of a product of the elementary reflectors returned by PLASMA_cgeqrf.

Parameters

← ***M*** The number of rows of the matrix Q. M >= 0.

← ***N*** The number of columns of the matrix Q. N >= M.

← ***K*** The number of columns of elementary tile reflectors whose product defines the matrix Q. M >= K >= 0.

← ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_cgeqrf.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

← ***T*** Auxiliary factorization data, computed by PLASMA_cgeqrf.

→ ***Q*** On exit, the M-by-N matrix Q.

← ***LDQ*** The leading dimension of the array Q. LDQ >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_cungqr_Tile
PLASMA_cungqr_Tile_Async
PLASMA_cungqr
PLASMA_dungqr
PLASMA_sungqr
PLASMA_cgeqrf

3.2.2.35 int PLASMA_cunmlq (PLASMA_enum *side*, PLASMA_enum *trans*, int *M*, int *N*, int *K*, PLASMA_Complex32_t ∗ *A*, int *LDA*, PLASMA_Complex32_t ∗ *T*, PLASMA_Complex32_t ∗ *B*, int *LDB*)

PLASMA_cunmlq - overwrites the general M-by-N matrix C with Q∗C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_cgelqf. Q is of order M.

Parameters

← ***side*** Intended usage: = PlasmaLeft: apply Q or Q\∗\∗H from the left; = PlasmaRight: apply Q or Q\∗\∗H from the right. Currently only PlasmaLeft is supported.

← ***trans*** Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaConjTrans: conjfugate transpose, apply Q\∗\∗H. Currently only PlasmaConjTrans is supported.

← ***M*** The number of rows of the matrix C. M >= 0.

← ***N*** The number of columns of the matrix C. N >= 0.

← ***K*** The number of rows of elementary tile reflectors whose product defines the matrix Q. M >= K >= 0.

← ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_cgelqf.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,K).

← ***T*** Auxiliary factorization data, computed by PLASMA_cgelqf.

↔ ***B*** On entry, the M-by-N matrix B. On exit, B is overwritten by Q∗B or Q\∗\∗H∗B.

← ***LDB*** The leading dimension of the array C. LDC >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_cunmlq_Tile
PLASMA_cunmlq_Tile_Async
PLASMA_cunmlq
PLASMA_dunmlq
PLASMA_sunmlq
PLASMA_cgelqf

3.2.2.36 int PLASMA_cunmqr (PLASMA_enum *side*, PLASMA_enum *trans*, int *M*, int *N*, int *K*, PLASMA_Complex32_t $*$ *A*, int *LDA*, PLASMA_Complex32_t $*$ *T*, PLASMA_Complex32_t $*$ *B*, int *LDB*)

PLASMA_cunmqr - overwrites the general M-by-N matrix C with Q$*$C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_cgeqrf. Q is of order M.

Parameters

← ***side*** Intended usage: = PlasmaLeft: apply Q or Q\$*$\$*$H from the left; = PlasmaRight: apply Q or Q\$*$\$*$H from the right. Currently only PlasmaLeft is supported.

← ***trans*** Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaConjTrans: conjfugate transpose, apply Q\$*$\$*$H. Currently only PlasmaConjTrans is supported.

← ***M*** The number of rows of the matrix C. M >= 0.

← ***N*** The number of columns of the matrix C. N >= 0.

← ***K*** The number of columns of elementary tile reflectors whose product defines the matrix Q. M >= K >= 0.

← ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_cgeqrf.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M);

← ***T*** Auxiliary factorization data, computed by PLASMA_cgeqrf.

↔ ***B*** On entry, the M-by-N matrix B. On exit, B is overwritten by Q$*$B or Q\$*$\$*H*$B.

← ***LDB*** The leading dimension of the array C. LDC >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_cunmqr_Tile
PLASMA_cunmqr_Tile_Async
PLASMA_cunmqr
PLASMA_dunmqr
PLASMA_sunmqr
PLASMA_cgeqrf

3.3 Simple Interface - Double Real

Functions/Subroutines

- int PLASMA_dgelqf (int M, int N, double ∗A, int LDA, double ∗T)
- int PLASMA_dgelqs (int M, int N, int NRHS, double ∗A, int LDA, double ∗T, double ∗B, int LDB)
- int PLASMA_dgels (PLASMA_enum trans, int M, int N, int NRHS, double ∗A, int LDA, double ∗T, double ∗B, int LDB)
- int PLASMA_dgemm (PLASMA_enum transA, PLASMA_enum transB, int M, int N, int K, double alpha, double ∗A, int LDA, double ∗B, int LDB, double beta, double ∗C, int LDC)
- int PLASMA_dgeqrf (int M, int N, double ∗A, int LDA, double ∗T)
- int PLASMA_dgeqrs (int M, int N, int NRHS, double ∗A, int LDA, double ∗T, double ∗B, int LDB)
- int PLASMA_dgesv (int N, int NRHS, double ∗A, int LDA, double ∗L, int ∗IPIV, double ∗B, int LDB)
- int PLASMA_dgetrf (int M, int N, double ∗A, int LDA, double ∗L, int ∗IPIV)
- int PLASMA_dgetrs (PLASMA_enum trans, int N, int NRHS, double ∗A, int LDA, double ∗L, int ∗IPIV, double ∗B, int LDB)
- double PLASMA_dlange (PLASMA_enum norm, int M, int N, double ∗A, int LDA, double ∗work)
- double PLASMA_dlansy (PLASMA_enum norm, PLASMA_enum uplo, int N, double ∗A, int LDA, double ∗work)
- int PLASMA_dlauum (PLASMA_enum uplo, int N, double ∗A, int LDA)
- int PLASMA_dorglq (int M, int N, int K, double ∗A, int LDA, double ∗T, double ∗B, int LDB)
- int PLASMA_dorgqr (int M, int N, int K, double ∗A, int LDA, double ∗T, double ∗Q, int LDQ)
- int PLASMA_dormlq (PLASMA_enum side, PLASMA_enum trans, int M, int N, int K, double ∗A, int LDA, double ∗T, double ∗B, int LDB)
- int PLASMA_dormqr (PLASMA_enum side, PLASMA_enum trans, int M, int N, int K, double ∗A, int LDA, double ∗T, double ∗B, int LDB)
- int PLASMA_dplgsy (double bump, int N, double ∗A, int LDA, unsigned long long int seed)
- int PLASMA_dplrnt (int M, int N, double ∗A, int LDA, unsigned long long int seed)
- int PLASMA_dposv (PLASMA_enum uplo, int N, int NRHS, double ∗A, int LDA, double ∗B, int LDB)
- int PLASMA_dpotrf (PLASMA_enum uplo, int N, double ∗A, int LDA)
- int PLASMA_dpotri (PLASMA_enum uplo, int N, double ∗A, int LDA)
- int PLASMA_dpotrs (PLASMA_enum uplo, int N, int NRHS, double ∗A, int LDA, double ∗B, int LDB)
- int PLASMA_dsgesv (int N, int NRHS, double ∗A, int LDA, double ∗B, int LDB, double ∗X, int LDX, int ∗ITER)
- int PLASMA_dsposv (PLASMA_enum uplo, int N, int NRHS, double ∗A, int LDA, double ∗B, int LDB, double ∗X, int LDX, int ∗ITER)
- int PLASMA_dsungesv (PLASMA_enum trans, int N, int NRHS, double ∗A, int LDA, double ∗B, int LDB, double ∗X, int LDX, int ∗ITER)
- int PLASMA_dsymm (PLASMA_enum side, PLASMA_enum uplo, int M, int N, double alpha, double ∗A, int LDA, double ∗B, int LDB, double beta, double ∗C, int LDC)
- int PLASMA_dsyr2k (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, double alpha, double ∗A, int LDA, double ∗B, int LDB, double beta, double ∗C, int LDC)
- int PLASMA_dsyrk (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, double alpha, double ∗A, int LDA, double beta, double ∗C, int LDC)
- int PLASMA_dtrmm (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, int N, int NRHS, double alpha, double ∗A, int LDA, double ∗B, int LDB)
- int PLASMA_dtrsm (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, int N, int NRHS, double alpha, double ∗A, int LDA, double ∗B, int LDB)

- int PLASMA_dtrsmpl (int N, int NRHS, double ∗A, int LDA, double ∗L, int ∗IPIV, double ∗B, int LDB)
- int PLASMA_dtrtri (PLASMA_enum uplo, PLASMA_enum diag, int N, double ∗A, int LDA)
- int PLASMA_dLapack_to_Tile (double ∗Af77, int LDA, PLASMA_desc ∗A)
- int PLASMA_dTile_to_Lapack (PLASMA_desc ∗A, double ∗Af77, int LDA)

3.3.1 Detailed Description

This is the group of double real functions using the simple user interface.

3.3.2 Function/Subroutine Documentation

3.3.2.1 int PLASMA_dgelqf (int *M*, int *N*, double ∗ *A*, int *LDA*, double ∗ *T*)

PLASMA_dgelqf - Computes the tile LQ factorization of a complex M-by-N matrix A: A = L ∗ Q.

Parameters

← ***M*** The number of rows of the matrix A. M >= 0.

← ***N*** The number of columns of the matrix A. N >= 0.

↔ ***A*** On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the m-by-min(M,N) lower trapezoidal matrix L (L is lower triangular if M <= N); the elements above the diagonal represent the unitary matrix Q as a product of elementary reflectors, stored by tiles.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

→ ***T*** On exit, auxiliary factorization data, required by PLASMA_dgelqs to solve the system of equations.

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_dgelqf_Tile
PLASMA_dgelqf_Tile_Async
PLASMA_cgelqf
PLASMA_dgelqf
PLASMA_sgelqf
PLASMA_dgelqs

3.3.2.2 int PLASMA_dgelqs (int *M*, int *N*, int *NRHS*, double ∗ *A*, int *LDA*, double ∗ *T*, double ∗ *B*, int *LDB*)

PLASMA_dgelqs - Compute a minimum-norm solution min || A∗X - B || using the LQ factorization A = L∗Q computed by PLASMA_dgelqf.

Parameters

← ***M*** The number of rows of the matrix A. M >= 0.

← ***N*** The number of columns of the matrix A. N >= M >= 0.

← ***NRHS*** The number of columns of B. NRHS >= 0.

← ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_dgelqf.

← ***LDA*** The leading dimension of the array A. LDA >= M.

← ***T*** Auxiliary factorization data, computed by PLASMA_dgelqf.

↔ ***B*** On entry, the M-by-NRHS right hand side matrix B. On exit, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= N.

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_dgelqs_Tile
PLASMA_dgelqs_Tile_Async
PLASMA_cgelqs
PLASMA_dgelqs
PLASMA_sgelqs
PLASMA_dgelqf

3.3.2.3 int PLASMA_dgels (PLASMA_enum *trans*, int *M*, int *N*, int *NRHS*, double ∗ *A*, int *LDA*, double ∗ *T*, double ∗ *B*, int *LDB*)

PLASMA_dgels - solves overdetermined or underdetermined linear systems involving an M-by-N matrix A using the QR or the LQ factorization of A. It is assumed that A has full rank. The following options are provided:

trans = PlasmaNoTrans and M >= N: find the least squares solution of an overdetermined system, i.e., solve the least squares problem: minimize || B - A∗X ||.

trans = PlasmaNoTrans and M < N: find the minimum norm solution of an underdetermined system A ∗ X = B.

Several right hand side vectors B and solution vectors X can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

Parameters

← ***trans*** Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaTrans: the linear system involves A\∗\∗T. Currently only PlasmaNoTrans is supported.

← ***M*** The number of rows of the matrix A. M >= 0.

← ***N*** The number of columns of the matrix A. N >= 0.

← ***NRHS*** The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

↔ ***A*** On entry, the M-by-N matrix A. On exit, if M >= N, A is overwritten by details of its QR factorization as returned by PLASMA_dgeqrf; if M < N, A is overwritten by details of its LQ factorization as returned by PLASMA_dgelqf.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

→ ***T*** On exit, auxiliary factorization data.

↔ ***B*** On entry, the M-by-NRHS matrix B of right hand side vectors, stored columnwise; On exit, if return value = 0, B is overwritten by the solution vectors, stored columnwise: if M >= N, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if M < N, rows 1 to N of B contain the minimum norm solution vectors;

← ***LDB*** The leading dimension of the array B. LDB >= MAX(1,M,N).

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_dgels_Tile
PLASMA_dgels_Tile_Async
PLASMA_cgels
PLASMA_dgels
PLASMA_sgels

3.3.2.4 int PLASMA_dgemm (PLASMA_enum *transA*, PLASMA_enum *transB*, int *M*, int *N*, int *K*, double *alpha*, double ∗ *A*, int *LDA*, double ∗ *B*, int *LDB*, double *beta*, double ∗ *C*, int *LDC*)

PLASMA_dgemm - Performs one of the matrix-matrix operations

$$C = \alpha[op(A) \times op(B)] + \beta C$$

,

where op(X) is one of

op(X) = X or op(X) = X' or op(X) = g(X')

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

Parameters

← ***transA*** Specifies whether the matrix A is transposed, not transposed or ugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is transposed; = PlasmaTrans: A is ugate transposed.

← ***transB*** Specifies whether the matrix B is transposed, not transposed or ugate transposed: = PlasmaNoTrans: B is not transposed; = PlasmaTrans: B is transposed; = PlasmaTrans: B is ugate transposed.

← ***M*** M specifies the number of rows of the matrix op(A) and of the matrix C. M >= 0.

← ***N*** N specifies the number of columns of the matrix op(B) and of the matrix C. N >= 0.

← ***K*** K specifies the number of columns of the matrix op(A) and the number of rows of the matrix op(B). K >= 0.

← ***alpha*** alpha specifies the scalar alpha

← ***A*** A is a LDA-by-ka matrix, where ka is K when transA = PlasmaNoTrans, and is M otherwise.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

← ***B*** B is a LDB-by-kb matrix, where kb is N when transB = PlasmaNoTrans, and is K otherwise.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

← ***beta*** beta specifies the scalar beta

↔ ***C*** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N matrix (alpha∗op(A)∗op(B) + beta∗C)

← ***LDC*** The leading dimension of the array C. LDC >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dgemm_Tile
PLASMA_cgemm
PLASMA_dgemm
PLASMA_sgemm

3.3.2.5 int PLASMA_dgeqrf (int *M*, int *N*, double ∗ *A*, int *LDA*, double ∗ *T*)

PLASMA_dgeqrf - Computes the tile QR factorization of a complex M-by-N matrix A: A = Q ∗ R.

Parameters

← ***M*** The number of rows of the matrix A. M >= 0.

← ***N*** The number of columns of the matrix A. N >= 0.

↔ ***A*** 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 represent the unitary matrix Q as a product of elementary reflectors stored by tiles.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

→ ***T*** On exit, auxiliary factorization data, required by PLASMA_dgeqrs to solve the system of equations.

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_dgeqrf_Tile
PLASMA_dgeqrf_Tile_Async
PLASMA_cgeqrf
PLASMA_dgeqrf
PLASMA_sgeqrf
PLASMA_dgeqrs

3.3.2.6 int PLASMA_dgeqrs (int *M*, int *N*, int *NRHS*, double ∗ *A*, int *LDA*, double ∗ *T*, double ∗ *B*, int *LDB*)

PLASMA_dgeqrs - Compute a minimum-norm solution min || A∗X - B || using the RQ factorization A = R∗Q computed by PLASMA_dgeqrf.

Parameters

← ***M*** The number of rows of the matrix A. M >= 0.

← ***N*** The number of columns of the matrix A. N >= M >= 0.

← ***NRHS*** The number of columns of B. NRHS >= 0.

↔ ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_dgeqrf.

← ***LDA*** The leading dimension of the array A. LDA >= M.

← ***T*** Auxiliary factorization data, computed by PLASMA_dgeqrf.

↔ ***B*** On entry, the m-by-nrhs right hand side matrix B. On exit, the n-by-nrhs solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_dgeqrs_Tile
PLASMA_dgeqrs_Tile_Async
PLASMA_cgeqrs
PLASMA_dgeqrs
PLASMA_sgeqrs
PLASMA_dgeqrf

3.3.2.7 int PLASMA_dgesv (int *N*, int *NRHS*, double ∗ *A*, int *LDA*, double ∗ *L*, int ∗ *IPIV*, double ∗ *B*, int *LDB*)

PLASMA_dgesv - Computes the solution to a system of linear equations A ∗ X = B, where A is an N-by-N matrix and X and B are N-by-NRHS matrices. The tile LU decomposition with partial tile pivoting and row interchanges is used to factor A. The factored form of A is then used to solve the system of equations A ∗ X = B.

Parameters

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

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

↔ ***A*** On entry, the N-by-N coefficient matrix A. On exit, the tile L and U factors from the factorization (not equivalent to LAPACK).

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

→ ***L*** On exit, auxiliary factorization data, related to the tile L factor, necessary to solve the system of equations.

→ ***IPIV*** On exit, the pivot indices that define the permutations (not equivalent to LAPACK).

↔ ***B*** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

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

>0 if i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

PLASMA_dgesv_Tile
PLASMA_dgesv_Tile_Async
PLASMA_cgesv
PLASMA_dgesv
PLASMA_sgesv
PLASMA_dcgesv

3.3.2.8 int PLASMA_dgetrf (int *M*, int *N*, double ∗ *A*, int *LDA*, double ∗ *L*, int ∗ *IPIV*)

PLASMA_dgetrf - Computes an LU factorization of a general M-by-N matrix A using the tile LU algorithm with partial tile pivoting with row interchanges.

Parameters

← ***M*** The number of rows of the matrix A. M >= 0.

← ***N*** The number of columns of the matrix A. N >= 0.

↔ ***A*** On entry, the M-by-N matrix to be factored. On exit, the tile factors L and U from the factorization.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

→ ***L*** On exit, auxiliary factorization data, related to the tile L factor, required by PLASMA_dgetrs to solve the system of equations.

→ ***IPIV*** The pivot indices that define the permutations (not equivalent to LAPACK).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

>***0*** if 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.

See also

PLASMA_dgetrf_Tile
PLASMA_dgetrf_Tile_Async
PLASMA_cgetrf
PLASMA_dgetrf
PLASMA_sgetrf
PLASMA_dgetrs

3.3.2.9 int PLASMA_dgetrs (PLASMA_enum *trans*, int *N*, int *NRHS*, double ∗ *A*, int *LDA*, double ∗ *L*, int ∗ *IPIV*, double ∗ *B*, int *LDB*)

PLASMA_dgetrs - Solves a system of linear equations A ∗ X = B, with a general N-by-N matrix A using the tile LU factorization computed by PLASMA_dgetrf.

Parameters

← ***trans*** Intended to specify the the form of the system of equations: = PlasmaNoTrans: A ∗ X = B (No transpose) = PlasmaTrans: A∗∗T ∗ X = B (Transpose) = PlasmaTrans: A\∗\∗T ∗ X = B (Conjugate transpose) Currently only PlasmaNoTrans is supported.

← ***N*** The order of the matrix A. N >= 0.

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

← ***A*** The tile factors L and U from the factorization, computed by PLASMA_dgetrf.

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

← ***L*** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_dgetrf.

← ***IPIV*** The pivot indices from PLASMA_dgetrf (not equivalent to LAPACK).

↔ ***B*** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, the solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

Returns

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

See also

PLASMA_dgetrs_Tile
PLASMA_dgetrs_Tile_Async
PLASMA_cgetrs
PLASMA_dgetrs
PLASMA_sgetrs
PLASMA_dgetrf

3.3.2.10 double PLASMA_dlange (PLASMA_enum *norm*, int *M*, int *N*, double * *A*, int *LDA*, double * *work*)

PLASMA_dlange returns the value

dlange = (max(abs(A(i,j))), NORM = PlasmaMaxNorm ((norm1(A), NORM = PlasmaOneNorm ((normI(A), NORM = PlasmaInfNorm ((normF(A), NORM = PlasmaFrobeniusNorm

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

← ***norm*** = PlasmaMaxNorm: Max norm = PlasmaOneNorm: One norm = PlasmaInfNorm: Infinity norm = PlasmaFrobeniusNorm: Frobenius norm

← ***M*** The number of rows of the matrix A. M >= 0. When M = 0, the returned value is set to zero.

← ***N*** The number of columns of the matrix A. N >= 0. When N = 0, the returned value is set to zero.

← ***A*** The M-by-N matrix A.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

← ***work*** double precision array of dimension (MAX(1,LWORK)), where LWORK >= M when NORM = PlasmaInfNorm; otherwise, WORK is not referenced.

Returns

Return values

the norm described above.

See also

PLASMA_dlange_Tile
PLASMA_dlange_Tile_Async
PLASMA_clange
PLASMA_dlange
PLASMA_slange

3.3.2.11 double PLASMA_dlansy (PLASMA_enum *norm*, PLASMA_enum *uplo*, int *N*, double ∗ *A*, int *LDA*, double ∗ *work*)

PLASMA_dlansy returns the value

dlansy = (max(abs(A(i,j))), NORM = PlasmaMaxNorm ((norm1(A), NORM = PlasmaOneNorm ((normI(A), NORM = PlasmaInfNorm ((normF(A), NORM = PlasmaFrobeniusNorm

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

← ***norm*** = PlasmaMaxNorm: Max norm = PlasmaOneNorm: One norm = PlasmaInfNorm: Infinity norm = PlasmaFrobeniusNorm: Frobenius norm

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***N*** The number of columns/rows of the matrix A. N >= 0. When N = 0, the returned value is set to zero.

← ***A*** The N-by-N matrix A.

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

← ***work*** double precision array of dimension PLASMA_SIZE is PLASMA_STATIC_SCHEDULING is used, and NULL otherwise.

Returns

Return values

the norm described above.

See also

PLASMA_dlansy_Tile
PLASMA_dlansy_Tile_Async
PLASMA_clansy
PLASMA_dlansy
PLASMA_slansy

3.3.2.12 int PLASMA_dLapack_to_Tile (double ∗ *Af77*, int *LDA*, PLASMA_desc ∗ *A*)

PLASMA_dLapack_to_Tile - Conversion from LAPACK layout to tile layout.

Parameters

← ***Af77*** LAPACK matrix.

← ***LDA*** The leading dimension of the matrix Af77.

↔ ***A*** Descriptor of the PLASMA matrix in tile layout. If PLASMA_TRANSLATION_MODE is set to PLASMA_INPLACE, A->mat is not used and set to Af77 when returns, else if PLASMA_-TRANSLATION_MODE is set to PLASMA_OUTOFPLACE, A->mat has to be allocated before.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dLapack_to_Tile_Async
PLASMA_dTile_to_Lapack
PLASMA_cLapack_to_Tile
PLASMA_dLapack_to_Tile
PLASMA_sLapack_to_Tile

3.3.2.13 int PLASMA_dlauum (PLASMA_enum *uplo*, int *N*, double ∗ *A*, int *LDA*)

PLASMA_dlauum - Computes the product U ∗ U' or L' ∗ L, where the triangular factor U or L is stored in the upper or lower triangular part of the array A.

If UPLO = 'U' or 'u' then the upper triangle of the result is stored, overwriting the factor U in A. If UPLO = 'L' or 'l' then the lower triangle of the result is stored, overwriting the factor L in A.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***N*** The order of the triangular factor U or L. N >= 0.

↔ ***A*** On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U ∗ U'; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L' ∗ L.

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

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_dlauum_Tile
PLASMA_dlauum_Tile_Async
PLASMA_clauum
PLASMA_dlauum
PLASMA_slauum
PLASMA_dpotri

3.3.2.14 int PLASMA_dorglq (int *M*, int *N*, int *K*, double ∗ *A*, int *LDA*, double ∗ *T*, double ∗ *B*, int *LDB*)

PLASMA_dorglq - Generates an M-by-N matrix Q with orthonormal rows, which is defined as the first M rows of a product of the elementary reflectors returned by PLASMA_dgelqf.

Parameters

← ***M*** The number of rows of the matrix Q. M >= 0.

← ***N*** The number of columns of the matrix Q. N >= M.

← ***K*** The number of rows of elementary tile reflectors whose product defines the matrix Q. M >= K >= 0.

← ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_dgelqf.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

← ***T*** Auxiliary factorization data, computed by PLASMA_dgelqf.

→ ***B*** On exit, the M-by-N matrix Q.

← ***LDA*** The leading dimension of the array B. LDB >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

PLASMA_SUCCESS <0 if -i, the i-th argument had an illegal value

See also

PLASMA_dorglq_Tile
PLASMA_dorglq_Tile_Async
PLASMA_cunglq
PLASMA_dunglq
PLASMA_sunglq
PLASMA_dgelqf

3.3.2.15 int PLASMA_dorgqr (int *M*, int *N*, int *K*, double ∗ *A*, int *LDA*, double ∗ *T*, double ∗ *Q*, int *LDQ*)

PLASMA_dorgqr - Generates an M-by-N matrix Q with orthonormal columns, which is defined as the first N columns of a product of the elementary reflectors returned by PLASMA_dgeqrf.

Parameters

← ***M*** The number of rows of the matrix Q. M >= 0.

← ***N*** The number of columns of the matrix Q. N >= M.

← ***K*** The number of columns of elementary tile reflectors whose product defines the matrix Q. M >= K >= 0.

← ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_dgeqrf.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

← ***T*** Auxiliary factorization data, computed by PLASMA_dgeqrf.

→ ***Q*** On exit, the M-by-N matrix Q.

← ***LDQ*** The leading dimension of the array Q. LDQ >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_dorgqr_Tile
PLASMA_dorgqr_Tile_Async
PLASMA_cungqr
PLASMA_dungqr
PLASMA_sungqr
PLASMA_dgeqrf

3.3.2.16 int PLASMA_dormlq (PLASMA_enum *side*, PLASMA_enum *trans*, int *M*, int *N*, int *K*, double ∗ *A*, int *LDA*, double ∗ *T*, double ∗ *B*, int *LDB*)

PLASMA_dormlq - overwrites the general M-by-N matrix C with Q∗C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_dgelqf. Q is of order M.

Parameters

← ***side*** Intended usage: = PlasmaLeft: apply Q or Q\∗\∗T from the left; = PlasmaRight: apply Q or Q\∗\∗T from the right. Currently only PlasmaLeft is supported.

← ***trans*** Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaTrans: ugate transpose, apply Q\∗\∗T. Currently only PlasmaTrans is supported.

← ***M*** The number of rows of the matrix C. M >= 0.

← ***N*** The number of columns of the matrix C. N >= 0.

← ***K*** The number of rows of elementary tile reflectors whose product defines the matrix Q. M >= K >= 0.

← ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_dgelqf.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,K).

← ***T*** Auxiliary factorization data, computed by PLASMA_dgelqf.

↔ ***B*** On entry, the M-by-N matrix B. On exit, B is overwritten by Q∗B or Q\∗\∗T∗B.

← ***LDB*** The leading dimension of the array C. LDC >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_dormlq_Tile
PLASMA_dormlq_Tile_Async
PLASMA_cunmlq
PLASMA_dunmlq
PLASMA_sunmlq
PLASMA_dgelqf

3.3.2.17 int PLASMA_dormqr (PLASMA_enum *side*, PLASMA_enum *trans*, int *M*, int *N*, int *K*, double ∗ *A*, int *LDA*, double ∗ *T*, double ∗ *B*, int *LDB*)

PLASMA_dormqr - overwrites the general M-by-N matrix C with Q∗C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_dgeqrf. Q is of order M.

Parameters

← ***side*** Intended usage: = PlasmaLeft: apply Q or Q\∗\∗T from the left; = PlasmaRight: apply Q or Q\∗\∗T from the right. Currently only PlasmaLeft is supported.

← ***trans*** Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaTrans: ugate transpose, apply Q\∗\∗T. Currently only PlasmaTrans is supported.

← ***M*** The number of rows of the matrix C. M >= 0.

← ***N*** The number of columns of the matrix C. N >= 0.

← ***K*** The number of columns of elementary tile reflectors whose product defines the matrix Q. M >= K >= 0.

← ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_dgeqrf.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M);

← ***T*** Auxiliary factorization data, computed by PLASMA_dgeqrf.

↔ ***B*** On entry, the M-by-N matrix B. On exit, B is overwritten by Q∗B or Q\∗\∗T∗B.

← ***LDB*** The leading dimension of the array C. LDC >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_dormqr_Tile
PLASMA_dormqr_Tile_Async
PLASMA_cunmqr
PLASMA_dunmqr
PLASMA_sunmqr
PLASMA_dgeqrf

3.3.2.18 int PLASMA_dplgsy (double *bump*, int *N*, double ∗ *A*, int *LDA*, unsigned long long int *seed*)

PLASMA_dplgsy - Generate a random hermitian matrix by tiles.

Parameters

← ***bump*** The value to add to the diagonal to be sure to have a positive definite matrix.

← ***N*** The order of the matrix A. N >= 0.

→ ***A*** On exit, The random hermitian matrix A generated.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_dplgsy_Tile
PLASMA_dplgsy_Tile_Async
PLASMA_cplgsy
PLASMA_dplgsy
PLASMA_splgsy
PLASMA_dplrnt
PLASMA_dplgsy

3.3.2.19 int PLASMA_dplrnt (int *M*, int *N*, double ∗ *A*, int *LDA*, unsigned long long int *seed*)

PLASMA_dplrnt - Generate a random matrix by tiles.

Parameters

← ***M*** The number of rows of A.

← ***N*** The order of the matrix A. N >= 0.

→ ***A*** On exit, The random matrix A generated.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_dplrnt_Tile
PLASMA_dplrnt_Tile_Async
PLASMA_cplrnt
PLASMA_dplrnt
PLASMA_splrnt
PLASMA_dplgsy
PLASMA_dplgsy

3.3.2.20 int PLASMA_dposv (PLASMA_enum *uplo*, int *N*, int *NRHS*, double ∗ *A*, int *LDA*, double ∗ *B*, int *LDB*)

PLASMA_dposv - Computes the solution to a system of linear equations A ∗ X = B, where A is an N-by-N symmetric positive definite (or Hermitian positive definite in the complex case) matrix and X and B are N-by-NRHS matrices. The Cholesky decomposition is used to factor A as

$$A = \{_{L \times L^H, if uplo = PlasmaLower}^{U^H \times U, if uplo = PlasmaUpper}$$

where U is an upper triangular matrix and L is a lower triangular matrix. The factored form of A is then used to solve the system of equations A ∗ X = B.

Parameters

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

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

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

↔ ***A*** On entry, the symmetric positive definite (or Hermitian) matrix A. If uplo = PlasmaUpper, 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 return value = 0, the factor U or L from the Cholesky factorization A = U\∗\∗T∗U or A = L∗L\∗\∗T.

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

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

>***0*** if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_dposv_Tile
PLASMA_dposv_Tile_Async
PLASMA_cposv
PLASMA_dposv
PLASMA_sposv

3.3.2.21 int PLASMA_dpotrf (PLASMA_enum *uplo*, int *N*, double ∗ *A*, int *LDA*)

PLASMA_dpotrf - Computes the Cholesky factorization of a symmetric positive definite (or Hermitian positive definite in the complex case) matrix A. The factorization has the form

$$A = \{_{L \times L^H, if uplo = PlasmaLower}^{U^H \times U, if uplo = PlasmaUpper}$$

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

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***N*** The order of the matrix A. N >= 0.

↔ ***A*** On entry, the symmetric positive definite (or Hermitian) matrix A. If uplo = PlasmaUpper, 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 return value = 0, the factor U or L from the Cholesky factorization A = U\∗\∗T∗U or A = L∗L\∗\∗T.

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

Returns

Return values

PLASMA_SUCCESS successful exit

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

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_dpotrf_Tile
PLASMA_dpotrf_Tile_Async
PLASMA_cpotrf
PLASMA_dpotrf
PLASMA_spotrf
PLASMA_dpotrs

3.3.2.22 int PLASMA_dpotri (PLASMA_enum *uplo*, int *N*, double ∗ *A*, int *LDA*)

PLASMA_dpotri - Computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization A = U\∗\∗T∗U or A = L∗L\∗\∗T computed by PLASMA_dpotrf.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***N*** The order of the matrix A. N >= 0.

↔ ***A*** On entry, the triangular factor U or L from the Cholesky factorization A = U\∗\∗T∗U or A = L∗L\∗\∗T, as computed by PLASMA_dpotrf. On exit, the upper or lower triangle of the (Hermitian) inverse of A, overwriting the input factor U or L.

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

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

>***0*** if i, the (i,i) element of the factor U or L is zero, and the inverse could not be computed.

See also

PLASMA_dpotri_Tile
PLASMA_dpotri_Tile_Async
PLASMA_cpotri
PLASMA_dpotri
PLASMA_spotri
PLASMA_dpotrf

3.3.2.23 int PLASMA_dpotrs (PLASMA_enum *uplo*, int *N*, int *NRHS*, double ∗ *A*, int *LDA*, double ∗ *B*, int *LDB*)

PLASMA_dpotrs - Solves a system of linear equations A ∗ X = B with a symmetric positive definite (or Hermitian positive definite in the complex case) matrix A using the Cholesky factorization A = U\∗\∗T∗U or A = L∗L\∗\∗T computed by PLASMA_dpotrf.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***N*** The order of the matrix A. N >= 0.

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

← ***A*** The triangular factor U or L from the Cholesky factorization A = U\∗\∗T∗U or A = L∗L\∗\∗T, computed by PLASMA_dpotrf.

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

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_dpotrs_Tile
PLASMA_dpotrs_Tile_Async
PLASMA_cpotrs
PLASMA_dpotrs
PLASMA_spotrs
PLASMA_dpotrf

3.3.2.24 int PLASMA_dsgesv (int *N*, int *NRHS*, double ∗ *A*, int *LDA*, double ∗ *B*, int *LDB*, double ∗ *X*, int *LDX*, int ∗ *ITER*)

PLASMA_dsgesv - Computes the solution to a system of linear equations A ∗ X = B, where A is an N-by-N matrix and X and B are N-by-NRHS matrices.

PLASMA_dsgesv first attempts to factorize the matrix in COMPLEX and use this factorization within an iterative refinement procedure to produce a solution with COMPLEX∗16 normwise backward error quality (see below). If the approach fails the method switches to a COMPLEX∗16 factorization and solve.

The iterative refinement is not going to be a winning strategy if the ratio COMPLEX performance over COMPLEX∗16 performance is too small. A reasonable strategy should take the number of right-hand sides and the size of the matrix into account. This might be done with a call to ILAENV in the future. Up to now, we always try iterative refinement.

The iterative refinement process is stopped if ITER > ITERMAX or for all the RHS we have: RNRM < N∗XNRM∗ANRM∗EPS∗BWDMAX where:

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

Actually, in its current state (PLASMA 2.1.0), the test is slightly relaxed.

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

Parameters

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

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

← ***A*** The N-by-N coefficient matrix A. This matrix is not modified.

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

← ***B*** The N-by-NRHS matrix of right hand side matrix B.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

→ ***X*** If return value = 0, the N-by-NRHS solution matrix X.

← ***LDX*** The leading dimension of the array B. LDX >= max(1,N).

→ ***ITER*** The number of the current iteration in the iterative refinement process

Returns

Return values

PLASMA_SUCCESS successful exit

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

>0 if i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

PLASMA_dsgesv_Tile
PLASMA_dsgesv_Tile_Async
PLASMA_dsgesv
PLASMA_dgesv

3.3.2.25 int PLASMA_dsposv (PLASMA_enum *uplo*, int *N*, int *NRHS*, double ∗ *A*, int *LDA*, double ∗ *B*, int *LDB*, double ∗ *X*, int *LDX*, int ∗ *ITER*)

PLASMA_dsposv - Computes the solution to a system of linear equations A ∗ X = B, where A is an N-by-N symmetric positive definite (or Hermitian positive definite in the complex case) matrix and X and B are N-by-NRHS matrices. The Cholesky decomposition is used to factor A as

A = U∗∗H ∗ U, if uplo = PlasmaUpper, or A = L ∗ L∗∗H, if uplo = PlasmaLower,

where U is an upper triangular matrix and L is a lower triangular matrix. The factored form of A is then used to solve the system of equations A ∗ X = B.

PLASMA_dsposv first attempts to factorize the matrix in COMPLEX and use this factorization within an iterative refinement procedure to produce a solution with COMPLEX∗16 normwise backward error quality (see below). If the approach fails the method switches to a COMPLEX∗16 factorization and solve.

The iterative refinement is not going to be a winning strategy if the ratio COMPLEX performance over COMPLEX∗16 performance is too small. A reasonable strategy should take the number of right-hand sides and the size of the matrix into account. This might be done with a call to ILAENV in the future. Up to now, we always try iterative refinement.

The iterative refinement process is stopped if ITER > ITERMAX or for all the RHS we have: RNRM < N∗XNRM∗ANRM∗EPS∗BWDMAX where:

- ITER is the number of the current iteration in the iterative refinement process
- RNRM is the infinity-norm of the residual
- XNRM is the infinity-norm of the solution
- ANRM is the infinity-operator-norm of the matrix A

- EPS is the machine epsilon returned by DLAMCH('Epsilon').

Actually, in its current state (PLASMA 2.1.0), the test is slightly relaxed.

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

Parameters

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

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

← ***A*** The N-by-N symmetric positive definite (or Hermitian) coefficient matrix A. If uplo = PlasmaUpper, 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. This matrix is not modified.

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

← ***B*** The N-by-NRHS matrix of right hand side matrix B.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

→ ***X*** If return value = 0, the N-by-NRHS solution matrix X.

← ***LDX*** The leading dimension of the array B. LDX >= max(1,N).

→ ***ITER*** The number of the current iteration in the iterative refinement process

Returns

Return values

PLASMA_SUCCESS successful exit

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

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_dsposv_Tile
PLASMA_dsposv_Tile_Async
PLASMA_dsposv
PLASMA_zposv

3.3.2.26 int PLASMA_dsungesv (PLASMA_enum *trans*, int *N*, int *NRHS*, double ∗ *A*, int *LDA*, double ∗ *B*, int *LDB*, double ∗ *X*, int *LDX*, int ∗ *ITER*)

PLASMA_dsungesv - Solves overdetermined or underdetermined linear systems involving an M-by-N matrix A using the QR or the LQ factorization of A. It is assumed that A has full rank. The following options are provided:

trans = PlasmaNoTrans and M >= N: find the least squares solution of an overdetermined system, i.e., solve the least squares problem: minimize || B - A∗X ||.

trans = PlasmaNoTrans and M < N: find the minimum norm solution of an underdetermined system A ∗ X = B.

Several right hand side vectors B and solution vectors X can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

PLASMA_dsungesv first attempts to factorize the matrix in COMPLEX and use this factorization within an iterative refinement procedure to produce a solution with COMPLEX∗16 normwise backward error quality (see below). If the approach fails the method switches to a COMPLEX∗16 factorization and solve.

The iterative refinement is not going to be a winning strategy if the ratio COMPLEX performance over COMPLEX∗16 performance is too small. A reasonable strategy should take the number of right-hand sides and the size of the matrix into account. This might be done with a call to ILAENV in the future. Up to now, we always try iterative refinement.

The iterative refinement process is stopped if ITER > ITERMAX or for all the RHS we have: RNRM < N∗XNRM∗ANRM∗EPS∗BWDMAX where:

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

Actually, in its current state (PLASMA 2.1.0), the test is slightly relaxed.

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

We follow Bjorck's algorithm proposed in "Iterative Refinement of Linear Least Squares solutions I", BIT, 7:257-278, 1967.4

Parameters

← ***trans*** Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaTrans: the linear system involves A∗∗H. Currently only PlasmaNoTrans is supported.

← ***M*** The number of rows of the matrix A. M >= 0.

← ***N*** The number of columns of the matrix A. N >= 0.

← ***NRHS*** The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

← ***A*** The M-by-N matrix A. This matrix is not modified.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

← ***B*** The M-by-NRHS matrix B of right hand side vectors, stored columnwise. Not modified.

← ***LDB*** The leading dimension of the array B. LDB >= MAX(1,M,N).

→ ***X*** If return value = 0, the solution vectors, stored columnwise. if M >= N, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if M < N, rows 1 to N of B contain the minimum norm solution vectors;

← ***LDX*** The leading dimension of the array B. LDB >= MAX(1,M,N).

→ ***ITER*** The number of the current iteration in the iterative refinement process

Returns

Return values

PLASMA_SUCCESS successful exit

$<$***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_dsungesv_Tile
PLASMA_dsungesv_Tile_Async
PLASMA_dsungesv
PLASMA_zgels

3.3.2.27 int PLASMA_dsymm (PLASMA_enum *side*, PLASMA_enum *uplo*, int *M*, int *N*, double *alpha*, double $*$ *A*, int *LDA*, double $*$ *B*, int *LDB*, double *beta*, double $*$ *C*, int *LDC*)

PLASMA_dsymm - Performs one of the matrix-matrix operations

$$C = \alpha \times A \times B + \beta \times C$$

or

$$C = \alpha \times B \times A + \beta \times C$$

where alpha and beta are scalars, A is an symmetric matrix and B and C are m by n matrices.

Parameters

$\leftarrow$ ***side*** Specifies whether the symmetric matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

$\leftarrow$ ***uplo*** Specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the symmetric matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the symmetric matrix A is to be referenced.

$\leftarrow$ ***M*** Specifies the number of rows of the matrix C. M >= 0.

$\leftarrow$ ***N*** Specifies the number of columns of the matrix C. N >= 0.

$\leftarrow$ ***alpha*** Specifies the scalar alpha.

$\leftarrow$ ***A*** A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.

$\leftarrow$ ***LDA*** The leading dimension of the array A. LDA >= max(1,ka).

$\leftarrow$ ***B*** B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.

$\leftarrow$ ***LDB*** The leading dimension of the array B. LDB >= max(1,M).

$\leftarrow$ ***beta*** Specifies the scalar beta.

$\leftrightarrow$ ***C*** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

$\leftarrow$ ***LDC*** The leading dimension of the array C. LDC >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dsymm_Tile
PLASMA_csymm
PLASMA_dsymm
PLASMA_ssymm

3.3.2.28 int PLASMA_dsyr2k (PLASMA_enum *uplo*, PLASMA_enum *trans*, int *N*, int *K*, double *alpha*, double $*$ *A*, int *LDA*, double $*$ *B*, int *LDB*, double *beta*, double $*$ *C*, int *LDC*)

PLASMA_dsyr2k - Performs one of the symmetric rank 2k operations

$$C = \alpha[op(A) \times g(op(B)')] + \alpha[op(B) \times g(op(A)')] + \beta C$$

, or

$$C = \alpha[g(op(A)') \times op(B)] + \alpha[g(op(B)') \times op(A)] + \beta C$$

,

where op(X) is one of

op(X) = X or op(X) = g(X')

where alpha and beta are real scalars, C is an n-by-n symmetric matrix and A and B are an n-by-k matrices the first case and k-by-n matrices in the second case.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← ***trans*** Specifies whether the matrix A is transposed or ugate transposed: = PlasmaNoTrans:

$$C = \alpha[op(A) \times g(op(B)')] + \alpha[op(B) \times g(op(A)')] + \beta C$$

= PlasmaTrans:

$$C = \alpha[g(op(A)') \times op(B)] + \alpha[g(op(B)') \times op(A)] + \beta C$$

← ***N*** N specifies the order of the matrix C. N must be at least zero.

← ***K*** K specifies the number of columns of the A and B matrices with trans = PlasmaNoTrans. K specifies the number of rows of the A and B matrices with trans = PlasmaTrans.

← ***alpha*** alpha specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.

← ***LDA*** The leading dimension of the array A. LDA must be at least max(1, N), otherwise LDA must be at least max(1, K).

← ***B*** B is a LDB-by-kb matrix, where kb is K when trans = PlasmaNoTrans, and is N otherwise.

← ***LDB*** The leading dimension of the array B. LDB must be at least max(1, N), otherwise LDB must be at least max(1, K).

← ***beta*** beta specifies the scalar beta.

↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

← ***LDC*** The leading dimension of the array C. LDC >= max(1, N).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dsyr2k_Tile
PLASMA_csyr2k
PLASMA_dsyr2k
PLASMA_ssyr2k

3.3.2.29 int PLASMA_dsyrk (PLASMA_enum *uplo*, PLASMA_enum *trans*, int *N*, int *K*, double *alpha*, double ∗ *A*, int *LDA*, double *beta*, double ∗ *C*, int *LDC*)

PLASMA_dsyrk - Performs one of the hermitian rank k operations

$$C = \alpha[op(A) \times g(op(A)')] + \beta C$$

,

where op(X) is one of

op(X) = X or op(X) = g(X')

where alpha and beta are real scalars, C is an n-by-n hermitian matrix and A is an n-by-k matrix in the first case and a k-by-n matrix in the second case.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← ***trans*** Specifies whether the matrix A is transposed or ugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans : A is transposed.

← ***N*** N specifies the order of the matrix C. N must be at least zero.

← ***K*** K specifies the number of columns of the matrix op(A).

← ***alpha*** alpha specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.

← ***LDA*** The leading dimension of the array A. LDA must be at least max(1, N), otherwise LDA must be at least max(1, K).

← ***beta*** beta specifies the scalar beta

↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

← ***LDC*** The leading dimension of the array C. LDC >= max(1, N).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dsyrk_Tile
PLASMA_csyrk
PLASMA_dsyrk
PLASMA_ssyrk

3.3.2.30 int PLASMA_dTile_to_Lapack (PLASMA_desc ∗ *A*, double ∗ *Af77*, int *LDA*)

PLASMA_Tile_to_Lapack - Conversion from tile layout to LAPACK layout.

Parameters

← ***A*** Descriptor of the PLASMA matrix in tile layout.

↔ ***Af77*** LAPACK matrix. If PLASMA_TRANSLATION_MODE is set to PLASMA_INPLACE, Af77 has to be A->mat, else if PLASMA_TRANSLATION_MODE is set to PLASMA_-OUTOFPLACE, Af77 has to be allocated before.

← ***LDA*** The leading dimension of the matrix Af77.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dTile_to_Lapack_Async
PLASMA_dLapack_to_Tile
PLASMA_cTile_to_Lapack
PLASMA_dTile_to_Lapack
PLASMA_sTile_to_Lapack

3.3.2.31 int PLASMA_dtrmm (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, int *N*, int *NRHS*, double *alpha*, double ∗ *A*, int *LDA*, double ∗ *B*, int *LDB*)

PLASMA_dtrmm - Computes B = alpha∗op(A)∗B or B = alpha∗B∗op(A).

Parameters

← ***side*** Specifies whether A appears on the left or on the right of X: = PlasmaLeft: A∗X = B = PlasmaRight: X∗A = B

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***transA*** Specifies whether the matrix A is transposed, not transposed or ugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaTrans: A is ugate transposed.

← ***diag*** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A us unit.

← ***N*** The order of the matrix A. N >= 0.

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

← ***A*** The triangular matrix A. If uplo = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If uplo = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If diag = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.

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

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_dtrmm_Tile
PLASMA_dtrmm_Tile_Async
PLASMA_ctrmm
PLASMA_dtrmm
PLASMA_strmm

3.3.2.32 int PLASMA_dtrsm (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, int *N*, int *NRHS*, double *alpha*, double ∗ *A*, int *LDA*, double ∗ *B*, int *LDB*)

PLASMA_dtrsm - Computes triangular solve A∗X = B or X∗A = B.

Parameters

← ***side*** Specifies whether A appears on the left or on the right of X: = PlasmaLeft: A∗X = B = PlasmaRight: X∗A = B

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***transA*** Specifies whether the matrix A is transposed, not transposed or ugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaTrans: A is ugate transposed.

← ***diag*** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A us unit.

← ***N*** The order of the matrix A. N >= 0.

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

← ***A*** The triangular matrix A. If uplo = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If uplo = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If diag = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.

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

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_dtrsm_Tile
PLASMA_dtrsm_Tile_Async
PLASMA_ctrsm
PLASMA_dtrsm
PLASMA_strsm

3.3.2.33 int PLASMA_dtrsmpl (int *N*, int *NRHS*, double ∗ *A*, int *LDA*, double ∗ *L*, int ∗ *IPIV*, double ∗ *B*, int *LDB*)

PLASMA_dtrsmpl - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix.

Parameters

← ***N*** The order of the matrix A. N >= 0.

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

← ***A*** The tile factor L from the factorization, computed by PLASMA_dgetrf.

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

← ***L*** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_dgetrf.

← ***IPIV*** The pivot indices from PLASMA_dgetrf (not equivalent to LAPACK).

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_dtrsmpl_Tile
PLASMA_dtrsmpl_Tile_Async
PLASMA_ctrsmpl
PLASMA_dtrsmpl
PLASMA_strsmpl
PLASMA_dgetrf

3.3.2.34 int PLASMA_dtrtri (PLASMA_enum *uplo*, PLASMA_enum *diag*, int *N*, double ∗ *A*, int *LDA*)

PLASMA_dtrtri - Computes the inverse of a complex upper or lower triangular matrix A.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***diag*** = PlasmaNonUnit: A is non-unit triangular; = PlasmaUnit: A is unit triangular.

← ***N*** The order of the matrix A. N >= 0.

↔ ***A*** On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1. On exit, the (triangular) inverse of the original matrix, in the same storage format.

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

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

>***0*** if i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

See also

PLASMA_dtrtri_Tile
PLASMA_dtrtri_Tile_Async
PLASMA_ctrtri
PLASMA_dtrtri
PLASMA_strtri
PLASMA_dpotri

3.4 Simple Interface - Single Real

Functions/Subroutines

- int PLASMA_sgelqf (int M, int N, float *A, int LDA, float *T)
- int PLASMA_sgelqs (int M, int N, int NRHS, float *A, int LDA, float *T, float *B, int LDB)
- int PLASMA_sgels (PLASMA_enum trans, int M, int N, int NRHS, float *A, int LDA, float *T, float *B, int LDB)
- int PLASMA_sgemm (PLASMA_enum transA, PLASMA_enum transB, int M, int N, int K, float alpha, float *A, int LDA, float *B, int LDB, float beta, float *C, int LDC)
- int PLASMA_sgeqrf (int M, int N, float *A, int LDA, float *T)
- int PLASMA_sgeqrs (int M, int N, int NRHS, float *A, int LDA, float *T, float *B, int LDB)
- int PLASMA_sgesv (int N, int NRHS, float *A, int LDA, float *L, int *IPIV, float *B, int LDB)
- int PLASMA_sgetrf (int M, int N, float *A, int LDA, float *L, int *IPIV)
- int PLASMA_sgetrs (PLASMA_enum trans, int N, int NRHS, float *A, int LDA, float *L, int *IPIV, float *B, int LDB)
- float PLASMA_slange (PLASMA_enum norm, int M, int N, float *A, int LDA, float *work)
- float PLASMA_slansy (PLASMA_enum norm, PLASMA_enum uplo, int N, float *A, int LDA, float *work)
- int PLASMA_slauum (PLASMA_enum uplo, int N, float *A, int LDA)
- int PLASMA_sorglq (int M, int N, int K, float *A, int LDA, float *T, float *B, int LDB)
- int PLASMA_sorgqr (int M, int N, int K, float *A, int LDA, float *T, float *Q, int LDQ)
- int PLASMA_sormlq (PLASMA_enum side, PLASMA_enum trans, int M, int N, int K, float *A, int LDA, float *T, float *B, int LDB)
- int PLASMA_sormqr (PLASMA_enum side, PLASMA_enum trans, int M, int N, int K, float *A, int LDA, float *T, float *B, int LDB)
- int PLASMA_splgsy (float bump, int N, float *A, int LDA, unsigned long long int seed)
- int PLASMA_splrnt (int M, int N, float *A, int LDA, unsigned long long int seed)
- int PLASMA_sposv (PLASMA_enum uplo, int N, int NRHS, float *A, int LDA, float *B, int LDB)
- int PLASMA_spotrf (PLASMA_enum uplo, int N, float *A, int LDA)
- int PLASMA_spotri (PLASMA_enum uplo, int N, float *A, int LDA)
- int PLASMA_spotrs (PLASMA_enum uplo, int N, int NRHS, float *A, int LDA, float *B, int LDB)
- int PLASMA_ssymm (PLASMA_enum side, PLASMA_enum uplo, int M, int N, float alpha, float *A, int LDA, float *B, int LDB, float beta, float *C, int LDC)
- int PLASMA_ssyr2k (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, float alpha, float *A, int LDA, float *B, int LDB, float beta, float *C, int LDC)
- int PLASMA_ssyrk (PLASMA_enum uplo, PLASMA_enum trans, int N, int K, float alpha, float *A, int LDA, float beta, float *C, int LDC)
- int PLASMA_strmm (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, int N, int NRHS, float alpha, float *A, int LDA, float *B, int LDB)
- int PLASMA_strsm (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, int N, int NRHS, float alpha, float *A, int LDA, float *B, int LDB)
- int PLASMA_strsmpl (int N, int NRHS, float *A, int LDA, float *L, int *IPIV, float *B, int LDB)
- int PLASMA_strtri (PLASMA_enum uplo, PLASMA_enum diag, int N, float *A, int LDA)
- int PLASMA_sLapack_to_Tile (float *Af77, int LDA, PLASMA_desc *A)
- int PLASMA_sTile_to_Lapack (PLASMA_desc *A, float *Af77, int LDA)

3.4.1 Detailed Description

This is the group of single real functions using the simple user interface.

3.4.2 Function/Subroutine Documentation

3.4.2.1 int PLASMA_sgelqf (int *M*, int *N*, float ∗ *A*, int *LDA*, float ∗ *T*)

PLASMA_sgelqf - Computes the tile LQ factorization of a complex M-by-N matrix A: A = L ∗ Q.

Parameters

← ***M*** The number of rows of the matrix A. M >= 0.

← ***N*** The number of columns of the matrix A. N >= 0.

↔ ***A*** On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the m-by-min(M,N) lower trapezoidal matrix L (L is lower triangular if M <= N); the elements above the diagonal represent the unitary matrix Q as a product of elementary reflectors, stored by tiles.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

→ ***T*** On exit, auxiliary factorization data, required by PLASMA_sgelqs to solve the system of equations.

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_sgelqf_Tile
PLASMA_sgelqf_Tile_Async
PLASMA_cgelqf
PLASMA_dgelqf
PLASMA_sgelqf
PLASMA_sgelqs

3.4.2.2 int PLASMA_sgelqs (int *M*, int *N*, int *NRHS*, float ∗ *A*, int *LDA*, float ∗ *T*, float ∗ *B*, int *LDB*)

PLASMA_sgelqs - Compute a minimum-norm solution min || A∗X - B || using the LQ factorization A = L∗Q computed by PLASMA_sgelqf.

Parameters

← ***M*** The number of rows of the matrix A. M >= 0.

← ***N*** The number of columns of the matrix A. N >= M >= 0.

← ***NRHS*** The number of columns of B. NRHS >= 0.

← ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_sgelqf.

← ***LDA*** The leading dimension of the array A. LDA >= M.

← ***T*** Auxiliary factorization data, computed by PLASMA_sgelqf.

↔ ***B*** On entry, the M-by-NRHS right hand side matrix B. On exit, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= N.

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_sgelqs_Tile
PLASMA_sgelqs_Tile_Async
PLASMA_cgelqs
PLASMA_dgelqs
PLASMA_sgelqs
PLASMA_sgelqf

3.4.2.3 int PLASMA_sgels (PLASMA_enum *trans*, int *M*, int *N*, int *NRHS*, float ∗ *A*, int *LDA*, float ∗ *T*, float ∗ *B*, int *LDB*)

PLASMA_sgels - solves overdetermined or underdetermined linear systems involving an M-by-N matrix A using the QR or the LQ factorization of A. It is assumed that A has full rank. The following options are provided:

trans = PlasmaNoTrans and M >= N: find the least squares solution of an overdetermined system, i.e., solve the least squares problem: minimize || B - A∗X ||.

trans = PlasmaNoTrans and M < N: find the minimum norm solution of an underdetermined system A ∗ X = B.

Several right hand side vectors B and solution vectors X can be handled in a single call; they are stored as the columns of the M-by-NRHS right hand side matrix B and the N-by-NRHS solution matrix X.

Parameters

← ***trans*** Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaTrans: the linear system involves A\∗\∗T. Currently only PlasmaNoTrans is supported.

← ***M*** The number of rows of the matrix A. M >= 0.

← ***N*** The number of columns of the matrix A. N >= 0.

← ***NRHS*** The number of right hand sides, i.e., the number of columns of the matrices B and X. NRHS >= 0.

↔ ***A*** On entry, the M-by-N matrix A. On exit, if M >= N, A is overwritten by details of its QR factorization as returned by PLASMA_sgeqrf; if M < N, A is overwritten by details of its LQ factorization as returned by PLASMA_sgelqf.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

→ ***T*** On exit, auxiliary factorization data.

↔ ***B*** On entry, the M-by-NRHS matrix B of right hand side vectors, stored columnwise; On exit, if return value = 0, B is overwritten by the solution vectors, stored columnwise: if M >= N, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if M < N, rows 1 to N of B contain the minimum norm solution vectors;

← ***LDB*** The leading dimension of the array B. LDB >= MAX(1,M,N).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_sgels_Tile
PLASMA_sgels_Tile_Async
PLASMA_cgels
PLASMA_dgels
PLASMA_sgels

3.4.2.4 int PLASMA_sgemm (PLASMA_enum *transA*, PLASMA_enum *transB*, int *M*, int *N*, int *K*, float *alpha*, float ∗ *A*, int *LDA*, float ∗ *B*, int *LDB*, float *beta*, float ∗ *C*, int *LDC*)

PLASMA_sgemm - Performs one of the matrix-matrix operations

$$C = \alpha[op(A) \times op(B)] + \beta C$$

,

where op(X) is one of

op(X) = X or op(X) = X' or op(X) = g(X')

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

Parameters

← ***transA*** Specifies whether the matrix A is transposed, not transposed or ugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is transposed; = PlasmaTrans: A is ugate transposed.

← ***transB*** Specifies whether the matrix B is transposed, not transposed or ugate transposed: = PlasmaNoTrans: B is not transposed; = PlasmaTrans: B is transposed; = PlasmaTrans: B is ugate transposed.

← ***M*** M specifies the number of rows of the matrix op(A) and of the matrix C. M >= 0.

← ***N*** N specifies the number of columns of the matrix op(B) and of the matrix C. N >= 0.

← ***K*** K specifies the number of columns of the matrix op(A) and the number of rows of the matrix op(B). K >= 0.

← ***alpha*** alpha specifies the scalar alpha

← ***A*** A is a LDA-by-ka matrix, where ka is K when transA = PlasmaNoTrans, and is M otherwise.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

← ***B*** B is a LDB-by-kb matrix, where kb is N when transB = PlasmaNoTrans, and is K otherwise.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

← ***beta*** beta specifies the scalar beta

↔ ***C*** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N matrix (alpha∗op(A)∗op(B) + beta∗C)

← ***LDC*** The leading dimension of the array C. LDC >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_sgemm_Tile
PLASMA_cgemm
PLASMA_dgemm
PLASMA_sgemm

3.4.2.5 int PLASMA_sgeqrf (int *M*, int *N*, float ∗ *A*, int *LDA*, float ∗ *T*)

PLASMA_sgeqrf - Computes the tile QR factorization of a complex M-by-N matrix A: A = Q ∗ R.

Parameters

← ***M*** The number of rows of the matrix A. M >= 0.

← ***N*** The number of columns of the matrix A. N >= 0.

↔ ***A*** 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 represent the unitary matrix Q as a product of elementary reflectors stored by tiles.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

→ ***T*** On exit, auxiliary factorization data, required by PLASMA_sgeqrs to solve the system of equations.

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_sgeqrf_Tile
PLASMA_sgeqrf_Tile_Async
PLASMA_cgeqrf
PLASMA_dgeqrf
PLASMA_sgeqrf
PLASMA_sgeqrs

3.4.2.6 int PLASMA_sgeqrs (int *M*, int *N*, int *NRHS*, float ∗ *A*, int *LDA*, float ∗ *T*, float ∗ *B*, int *LDB*)

PLASMA_sgeqrs - Compute a minimum-norm solution min || A∗X - B || using the RQ factorization A = R∗Q computed by PLASMA_sgeqrf.

Parameters

← ***M*** The number of rows of the matrix A. M >= 0.

← ***N*** The number of columns of the matrix A. N >= M >= 0.

← ***NRHS*** The number of columns of B. NRHS >= 0.

↔ ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_sgeqrf.

← ***LDA*** The leading dimension of the array A. LDA >= M.

← ***T*** Auxiliary factorization data, computed by PLASMA_sgeqrf.

↔ ***B*** On entry, the m-by-nrhs right hand side matrix B. On exit, the n-by-nrhs solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_sgeqrs_Tile
PLASMA_sgeqrs_Tile_Async
PLASMA_cgeqrs
PLASMA_dgeqrs
PLASMA_sgeqrs
PLASMA_sgeqrf

3.4.2.7 int PLASMA_sgesv (int *N*, int *NRHS*, float ∗ *A*, int *LDA*, float ∗ *L*, int ∗ *IPIV*, float ∗ *B*, int *LDB*)

PLASMA_sgesv - Computes the solution to a system of linear equations A ∗ X = B, where A is an N-by-N matrix and X and B are N-by-NRHS matrices. The tile LU decomposition with partial tile pivoting and row interchanges is used to factor A. The factored form of A is then used to solve the system of equations A ∗ X = B.

Parameters

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

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

↔ ***A*** On entry, the N-by-N coefficient matrix A. On exit, the tile L and U factors from the factorization (not equivalent to LAPACK).

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

$\rightarrow$ ***L*** On exit, auxiliary factorization data, related to the tile L factor, necessary to solve the system of equations.

$\rightarrow$ ***IPIV*** On exit, the pivot indices that define the permutations (not equivalent to LAPACK).

$\leftrightarrow$ ***B*** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

$\leftarrow$ ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

>***0*** if i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

PLASMA_sgesv_Tile
PLASMA_sgesv_Tile_Async
PLASMA_cgesv
PLASMA_dgesv
PLASMA_sgesv
PLASMA_scgesv

3.4.2.8 int PLASMA_sgetrf (int *M*, int *N*, float ∗ *A*, int *LDA*, float ∗ *L*, int ∗ *IPIV*)

PLASMA_sgetrf - Computes an LU factorization of a general M-by-N matrix A using the tile LU algorithm with partial tile pivoting with row interchanges.

Parameters

$\leftarrow$ ***M*** The number of rows of the matrix A. M >= 0.

$\leftarrow$ ***N*** The number of columns of the matrix A. N >= 0.

$\leftrightarrow$ ***A*** On entry, the M-by-N matrix to be factored. On exit, the tile factors L and U from the factorization.

$\leftarrow$ ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

$\rightarrow$ ***L*** On exit, auxiliary factorization data, related to the tile L factor, required by PLASMA_sgetrs to solve the system of equations.

$\rightarrow$ ***IPIV*** The pivot indices that define the permutations (not equivalent to LAPACK).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

$>$***0*** if 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.

See also

PLASMA_sgetrf_Tile
PLASMA_sgetrf_Tile_Async
PLASMA_cgetrf
PLASMA_dgetrf
PLASMA_sgetrf
PLASMA_sgetrs

3.4.2.9 int PLASMA_sgetrs (PLASMA_enum *trans*, int *N*, int *NRHS*, float ∗ *A*, int *LDA*, float ∗ *L*, int ∗ *IPIV*, float ∗ *B*, int *LDB*)

PLASMA_sgetrs - Solves a system of linear equations A ∗ X = B, with a general N-by-N matrix A using the tile LU factorization computed by PLASMA_sgetrf.

Parameters

← ***trans*** Intended to specify the the form of the system of equations: = PlasmaNoTrans: A ∗ X = B (No transpose) = PlasmaTrans: A∗∗T ∗ X = B (Transpose) = PlasmaTrans: A\∗\∗T ∗ X = B (Conjugate transpose) Currently only PlasmaNoTrans is supported.

← ***N*** The order of the matrix A. N >= 0.

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

← ***A*** The tile factors L and U from the factorization, computed by PLASMA_sgetrf.

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

← ***L*** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_sgetrf.

← ***IPIV*** The pivot indices from PLASMA_sgetrf (not equivalent to LAPACK).

↔ ***B*** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, the solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

Returns

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

See also

PLASMA_sgetrs_Tile
PLASMA_sgetrs_Tile_Async
PLASMA_cgetrs
PLASMA_dgetrs
PLASMA_sgetrs
PLASMA_sgetrf

3.4.2.10 float PLASMA_slange (PLASMA_enum *norm*, int *M*, int *N*, float ∗ *A*, int *LDA*, float ∗ *work*)

PLASMA_slange returns the value

slange = (max(abs(A(i,j))), NORM = PlasmaMaxNorm ((norm1(A), NORM = PlasmaOneNorm ((normI(A), NORM = PlasmaInfNorm ((normF(A), NORM = PlasmaFrobeniusNorm

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

← ***norm*** = PlasmaMaxNorm: Max norm = PlasmaOneNorm: One norm = PlasmaInfNorm: Infinity norm = PlasmaFrobeniusNorm: Frobenius norm

← ***M*** The number of rows of the matrix A. M >= 0. When M = 0, the returned value is set to zero.

← ***N*** The number of columns of the matrix A. N >= 0. When N = 0, the returned value is set to zero.

← ***A*** The M-by-N matrix A.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

← ***work*** float precision array of dimension (MAX(1,LWORK)), where LWORK >= M when NORM = PlasmaInfNorm; otherwise, WORK is not referenced.

Returns

Return values

the norm described above.

See also

PLASMA_slange_Tile
PLASMA_slange_Tile_Async
PLASMA_clange
PLASMA_dlange
PLASMA_slange

3.4.2.11 float PLASMA_slansy (PLASMA_enum *norm*, PLASMA_enum *uplo*, int *N*, float ∗ *A*, int *LDA*, float ∗ *work*)

PLASMA_slansy returns the value

slansy = (max(abs(A(i,j))), NORM = PlasmaMaxNorm ((norm1(A), NORM = PlasmaOneNorm ((normI(A), NORM = PlasmaInfNorm ((normF(A), NORM = PlasmaFrobeniusNorm

where norm1 denotes the one norm of a matrix (maximum column sum), normI denotes the infinity norm of a matrix (maximum row sum) and normF denotes the Frobenius norm of a matrix (square root of sum of squares). Note that max(abs(A(i,j))) is not a consistent matrix norm.

Parameters

← ***norm*** = PlasmaMaxNorm: Max norm = PlasmaOneNorm: One norm = PlasmaInfNorm: Infinity norm = PlasmaFrobeniusNorm: Frobenius norm

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***N*** The number of columns/rows of the matrix A. N >= 0. When N = 0, the returned value is set to zero.

← ***A*** The N-by-N matrix A.

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

← ***work*** float precision array of dimension PLASMA_SIZE is PLASMA_STATIC_SCHEDULING is used, and NULL otherwise.

Returns

Return values

the norm described above.

See also

PLASMA_slansy_Tile
PLASMA_slansy_Tile_Async
PLASMA_clansy
PLASMA_dlansy
PLASMA_slansy

3.4.2.12 int PLASMA_sLapack_to_Tile (float ∗ *Af77*, int *LDA*, PLASMA_desc ∗ *A*)

PLASMA_sLapack_to_Tile - Conversion from LAPACK layout to tile layout.

Parameters

← ***Af77*** LAPACK matrix.

← ***LDA*** The leading dimension of the matrix Af77.

↔ ***A*** Descriptor of the PLASMA matrix in tile layout. If PLASMA_TRANSLATION_MODE is set to PLASMA_INPLACE, A->mat is not used and set to Af77 when returns, else if PLASMA_TRANSLATION_MODE is set to PLASMA_OUTOFPLACE, A->mat has to be allocated before.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_sLapack_to_Tile_Async
PLASMA_sTile_to_Lapack
PLASMA_cLapack_to_Tile
PLASMA_dLapack_to_Tile
PLASMA_sLapack_to_Tile

3.4.2.13 int PLASMA_slauum (PLASMA_enum *uplo*, int *N*, float ∗ *A*, int *LDA*)

PLASMA_slauum - Computes the product U ∗ U' or L' ∗ L, where the triangular factor U or L is stored in the upper or lower triangular part of the array A.

If UPLO = 'U' or 'u' then the upper triangle of the result is stored, overwriting the factor U in A. If UPLO = 'L' or 'l' then the lower triangle of the result is stored, overwriting the factor L in A.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***N*** The order of the triangular factor U or L. N >= 0.

↔ ***A*** On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U ∗ U'; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L' ∗ L.

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

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_slauum_Tile
PLASMA_slauum_Tile_Async
PLASMA_clauum
PLASMA_dlauum
PLASMA_slauum
PLASMA_spotri

3.4.2.14 int PLASMA_sorglq (int *M*, int *N*, int *K*, float ∗ *A*, int *LDA*, float ∗ *T*, float ∗ *B*, int *LDB*)

PLASMA_sorglq - Generates an M-by-N matrix Q with orthonormal rows, which is defined as the first M rows of a product of the elementary reflectors returned by PLASMA_sgelqf.

Parameters

← ***M*** The number of rows of the matrix Q. M >= 0.

← ***N*** The number of columns of the matrix Q. N >= M.

← ***K*** The number of rows of elementary tile reflectors whose product defines the matrix Q. M >= K >= 0.

← ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_sgelqf.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

← ***T*** Auxiliary factorization data, computed by PLASMA_sgelqf.

→ ***B*** On exit, the M-by-N matrix Q.

← ***LDA*** The leading dimension of the array B. LDB >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

PLASMA_SUCCESS <0 if -i, the i-th argument had an illegal value

See also

PLASMA_sorglq_Tile
PLASMA_sorglq_Tile_Async
PLASMA_cunglq
PLASMA_dunglq
PLASMA_sunglq
PLASMA_sgelqf

3.4.2.15 int PLASMA_sorgqr (int *M*, int *N*, int *K*, float ∗ *A*, int *LDA*, float ∗ *T*, float ∗ *Q*, int *LDQ*)

PLASMA_sorgqr - Generates an M-by-N matrix Q with orthonormal columns, which is defined as the first N columns of a product of the elementary reflectors returned by PLASMA_sgeqrf.

Parameters

← ***M*** The number of rows of the matrix Q. M >= 0.

← ***N*** The number of columns of the matrix Q. N >= M.

← ***K*** The number of columns of elementary tile reflectors whose product defines the matrix Q. M >= K >= 0.

← ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_sgeqrf.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

← ***T*** Auxiliary factorization data, computed by PLASMA_sgeqrf.

→ ***Q*** On exit, the M-by-N matrix Q.

← ***LDQ*** The leading dimension of the array Q. LDQ >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_sorgqr_Tile
PLASMA_sorgqr_Tile_Async
PLASMA_cungqr
PLASMA_dungqr
PLASMA_sungqr
PLASMA_sgeqrf

3.4.2.16 int PLASMA_sormlq (PLASMA_enum *side*, PLASMA_enum *trans*, int *M*, int *N*, int *K*, float ∗ *A*, int *LDA*, float ∗ *T*, float ∗ *B*, int *LDB*)

PLASMA_sormlq - overwrites the general M-by-N matrix C with Q∗C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_sgelqf. Q is of order M.

Parameters

← ***side*** Intended usage: = PlasmaLeft: apply Q or Q\∗\∗T from the left; = PlasmaRight: apply Q or Q\∗\∗T from the right. Currently only PlasmaLeft is supported.

← ***trans*** Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaTrans: ugate transpose, apply Q\∗\∗T. Currently only PlasmaTrans is supported.

← ***M*** The number of rows of the matrix C. M >= 0.

← ***N*** The number of columns of the matrix C. N >= 0.

← ***K*** The number of rows of elementary tile reflectors whose product defines the matrix Q. M >= K >= 0.

← ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_sgelqf.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,K).

← ***T*** Auxiliary factorization data, computed by PLASMA_sgelqf.

↔ ***B*** On entry, the M-by-N matrix B. On exit, B is overwritten by Q∗B or Q\∗\∗T∗B.

← ***LDB*** The leading dimension of the array C. LDC >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_sormlq_Tile
PLASMA_sormlq_Tile_Async
PLASMA_cunmlq
PLASMA_dunmlq
PLASMA_sunmlq
PLASMA_sgelqf

3.4.2.17 int PLASMA_sormqr (PLASMA_enum *side*, PLASMA_enum *trans*, int *M*, int *N*, int *K*, float ∗ *A*, int *LDA*, float ∗ *T*, float ∗ *B*, int *LDB*)

PLASMA_sormqr - overwrites the general M-by-N matrix C with Q∗C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_sgeqrf. Q is of order M.

Parameters

← ***side*** Intended usage: = PlasmaLeft: apply Q or Q\∗\∗T from the left; = PlasmaRight: apply Q or Q\∗\∗T from the right. Currently only PlasmaLeft is supported.

← ***trans*** Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaTrans: ugate transpose, apply Q**T. Currently only PlasmaTrans is supported.

← ***M*** The number of rows of the matrix C. M >= 0.

← ***N*** The number of columns of the matrix C. N >= 0.

← ***K*** The number of columns of elementary tile reflectors whose product defines the matrix Q. M >= K >= 0.

← ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_sgeqrf.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M);

← ***T*** Auxiliary factorization data, computed by PLASMA_sgeqrf.

↔ ***B*** On entry, the M-by-N matrix B. On exit, B is overwritten by Q∗B or Q\∗\∗T∗B.

← ***LDB*** The leading dimension of the array C. LDC >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_sormqr_Tile
PLASMA_sormqr_Tile_Async
PLASMA_cunmqr
PLASMA_dunmqr
PLASMA_sunmqr
PLASMA_sgeqrf

3.4.2.18 int PLASMA_splgsy (float *bump*, int *N*, float ∗ *A*, int *LDA*, unsigned long long int *seed*)

PLASMA_splgsy - Generate a random hermitian matrix by tiles.

Parameters

← ***bump*** The value to add to the diagonal to be sure to have a positive definite matrix.

← ***N*** The order of the matrix A. N >= 0.

→ ***A*** On exit, The random hermitian matrix A generated.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_splgsy_Tile
PLASMA_splgsy_Tile_Async
PLASMA_cplgsy
PLASMA_dplgsy
PLASMA_splgsy
PLASMA_splrnt
PLASMA_splgsy

3.4.2.19 int PLASMA_splrnt (int *M*, int *N*, float ∗ *A*, int *LDA*, unsigned long long int *seed*)

PLASMA_splrnt - Generate a random matrix by tiles.

Parameters

← ***M*** The number of rows of A.

← ***N*** The order of the matrix A. N >= 0.

→ ***A*** On exit, The random matrix A generated.

← ***LDA*** The leading dimension of the array A. LDA >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_splrnt_Tile
PLASMA_splrnt_Tile_Async
PLASMA_cplrnt
PLASMA_dplrnt
PLASMA_splrnt
PLASMA_splgsy
PLASMA_splgsy

3.4.2.20 int PLASMA_sposv (PLASMA_enum *uplo*, int *N*, int *NRHS*, float ∗ *A*, int *LDA*, float ∗ *B*, int *LDB*)

PLASMA_sposv - Computes the solution to a system of linear equations A ∗ X = B, where A is an N-by-N symmetric positive definite (or Hermitian positive definite in the complex case) matrix and X and B are N-by-NRHS matrices. The Cholesky decomposition is used to factor A as

$$A = \{_{L \times L^H, if uplo = PlasmaLower}^{U^H \times U, if uplo = PlasmaUpper}$$

where U is an upper triangular matrix and L is a lower triangular matrix. The factored form of A is then used to solve the system of equations A ∗ X = B.

Parameters

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

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

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

↔ ***A*** On entry, the symmetric positive definite (or Hermitian) matrix A. If uplo = PlasmaUpper, 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 return value = 0, the factor U or L from the Cholesky factorization A = U**T*U or A = L*L**T.

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

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

>***0*** if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_sposv_Tile
PLASMA_sposv_Tile_Async
PLASMA_cposv
PLASMA_dposv
PLASMA_sposv

3.4.2.21 int PLASMA_spotrf (PLASMA_enum *uplo*, int *N*, float ∗ *A*, int *LDA*)

PLASMA_spotrf - Computes the Cholesky factorization of a symmetric positive definite (or Hermitian positive definite in the complex case) matrix A. The factorization has the form

$$A = \{^{U^H \times U, if uplo = PlasmaUpper}_{L \times L^H, if uplo = PlasmaLower}$$

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

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***N*** The order of the matrix A. N >= 0.

↔ ***A*** On entry, the symmetric positive definite (or Hermitian) matrix A. If uplo = PlasmaUpper, 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 return value = 0, the factor U or L from the Cholesky factorization A = U**T*U or A = L*L**T.

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

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

>***0*** if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_spotrf_Tile
PLASMA_spotrf_Tile_Async
PLASMA_cpotrf
PLASMA_dpotrf
PLASMA_spotrf
PLASMA_spotrs

3.4.2.22 int PLASMA_spotri (PLASMA_enum *uplo*, int *N*, float * *A*, int *LDA*)

PLASMA_spotri - Computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization A = U**T*U or A = L*L**T computed by PLASMA_spotrf.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***N*** The order of the matrix A. N >= 0.

↔ ***A*** On entry, the triangular factor U or L from the Cholesky factorization A = U**T*U or A = L*L**T, as computed by PLASMA_spotrf. On exit, the upper or lower triangle of the (Hermitian) inverse of A, overwriting the input factor U or L.

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

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

>***0*** if i, the (i,i) element of the factor U or L is zero, and the inverse could not be computed.

See also

PLASMA_spotri_Tile
PLASMA_spotri_Tile_Async
PLASMA_cpotri
PLASMA_dpotri
PLASMA_spotri
PLASMA_spotrf

3.4.2.23 int PLASMA_spotrs (PLASMA_enum *uplo*, int *N*, int *NRHS*, float ∗ *A*, int *LDA*, float ∗ *B*, int *LDB*)

PLASMA_spotrs - Solves a system of linear equations A ∗ X = B with a symmetric positive definite (or Hermitian positive definite in the complex case) matrix A using the Cholesky factorization A = U\∗\∗T∗U or A = L∗L\∗\∗T computed by PLASMA_spotrf.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***N*** The order of the matrix A. N >= 0.

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

← ***A*** The triangular factor U or L from the Cholesky factorization A = U\∗\∗T∗U or A = L∗L\∗\∗T, computed by PLASMA_spotrf.

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

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_spotrs_Tile
PLASMA_spotrs_Tile_Async
PLASMA_cpotrs
PLASMA_dpotrs
PLASMA_spotrs
PLASMA_spotrf

3.4.2.24 int PLASMA_ssymm (PLASMA_enum *side*, PLASMA_enum *uplo*, int *M*, int *N*, float *alpha*, float ∗ *A*, int *LDA*, float ∗ *B*, int *LDB*, float *beta*, float ∗ *C*, int *LDC*)

PLASMA_ssymm - Performs one of the matrix-matrix operations

$$C = \alpha \times A \times B + \beta \times C$$

or

$$C = \alpha \times B \times A + \beta \times C$$

where alpha and beta are scalars, A is an symmetric matrix and B and C are m by n matrices.

Parameters

$\leftarrow$ ***side*** Specifies whether the symmetric matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

$\leftarrow$ ***uplo*** Specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the symmetric matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the symmetric matrix A is to be referenced.

$\leftarrow$ ***M*** Specifies the number of rows of the matrix C. M >= 0.

$\leftarrow$ ***N*** Specifies the number of columns of the matrix C. N >= 0.

$\leftarrow$ ***alpha*** Specifies the scalar alpha.

$\leftarrow$ ***A*** A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.

$\leftarrow$ ***LDA*** The leading dimension of the array A. LDA >= max(1,ka).

$\leftarrow$ ***B*** B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.

$\leftarrow$ ***LDB*** The leading dimension of the array B. LDB >= max(1,M).

$\leftarrow$ ***beta*** Specifies the scalar beta.

$\leftrightarrow$ ***C*** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

$\leftarrow$ ***LDC*** The leading dimension of the array C. LDC >= max(1,M).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_ssymm_Tile
PLASMA_csymm
PLASMA_dsymm
PLASMA_ssymm

3.4.2.25 int PLASMA_ssyr2k (PLASMA_enum *uplo*, PLASMA_enum *trans*, int *N*, int *K*, float *alpha*, float ∗ *A*, int *LDA*, float ∗ *B*, int *LDB*, float *beta*, float ∗ *C*, int *LDC*)

PLASMA_ssyr2k - Performs one of the symmetric rank 2k operations

$$C = \alpha[op(A) \times g(op(B)')] + \alpha[op(B) \times g(op(A)')] + \beta C$$

, or

$$C = \alpha[g(op(A)') \times op(B)] + \alpha[g(op(B)') \times op(A)] + \beta C$$

,

where op(X) is one of

op(X) = X or op(X) = g(X')

where alpha and beta are real scalars, C is an n-by-n symmetric matrix and A and B are an n-by-k matrices the first case and k-by-n matrices in the second case.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← ***trans*** Specifies whether the matrix A is transposed or ugate transposed: = PlasmaNoTrans:

$$C = \alpha[op(A) \times g(op(B)')] + \alpha[op(B) \times g(op(A)')] + \beta C$$

= PlasmaTrans:

$$C = \alpha[g(op(A)') \times op(B)] + \alpha[g(op(B)') \times op(A)] + \beta C$$

← ***N*** N specifies the order of the matrix C. N must be at least zero.

← ***K*** K specifies the number of columns of the A and B matrices with trans = PlasmaNoTrans. K specifies the number of rows of the A and B matrices with trans = PlasmaTrans.

← ***alpha*** alpha specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.

← ***LDA*** The leading dimension of the array A. LDA must be at least max(1, N), otherwise LDA must be at least max(1, K).

← ***B*** B is a LDB-by-kb matrix, where kb is K when trans = PlasmaNoTrans, and is N otherwise.

← ***LDB*** The leading dimension of the array B. LDB must be at least max(1, N), otherwise LDB must be at least max(1, K).

← ***beta*** beta specifies the scalar beta.

↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

← ***LDC*** The leading dimension of the array C. LDC >= max(1, N).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_ssyr2k_Tile
PLASMA_csyr2k
PLASMA_dsyr2k
PLASMA_ssyr2k

3.4.2.26 int PLASMA_ssyrk (PLASMA_enum *uplo*, PLASMA_enum *trans*, int *N*, int *K*, float *alpha*, float * *A*, int *LDA*, float *beta*, float * *C*, int *LDC*)

PLASMA_ssyrk - Performs one of the hermitian rank k operations

$$C = \alpha[op(A) \times g(op(A)')] + \beta C$$

,

where op(X) is one of

op(X) = X or op(X) = g(X')

where alpha and beta are real scalars, C is an n-by-n hermitian matrix and A is an n-by-k matrix in the first case and a k-by-n matrix in the second case.

Parameters

- ← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.
- ← ***trans*** Specifies whether the matrix A is transposed or ugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans : A is transposed.
- ← ***N*** N specifies the order of the matrix C. N must be at least zero.
- ← ***K*** K specifies the number of columns of the matrix op(A).
- ← ***alpha*** alpha specifies the scalar alpha.
- ← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.
- ← ***LDA*** The leading dimension of the array A. LDA must be at least max(1, N), otherwise LDA must be at least max(1, K).
- ← ***beta*** beta specifies the scalar beta
- ↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.
- ← ***LDC*** The leading dimension of the array C. LDC >= max(1, N).

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_ssyrk_Tile
PLASMA_csyrk
PLASMA_dsyrk
PLASMA_ssyrk

3.4.2.27 int PLASMA_sTile_to_Lapack (PLASMA_desc * *A*, float * *Af77*, int *LDA*)

PLASMA_Tile_to_Lapack - Conversion from tile layout to LAPACK layout.

Parameters

- ← ***A*** Descriptor of the PLASMA matrix in tile layout.

↔ ***Af77*** LAPACK matrix. If PLASMA_TRANSLATION_MODE is set to PLASMA_INPLACE, Af77 has to be A->mat, else if PLASMA_TRANSLATION_MODE is set to PLASMA_-OUTOFPLACE, Af77 has to be allocated before.

← ***LDA*** The leading dimension of the matrix Af77.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_sTile_to_Lapack_Async
PLASMA_sLapack_to_Tile
PLASMA_cTile_to_Lapack
PLASMA_dTile_to_Lapack
PLASMA_sTile_to_Lapack

3.4.2.28 int PLASMA_strmm (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, int *N*, int *NRHS*, float *alpha*, float ∗ *A*, int *LDA*, float ∗ *B*, int *LDB*)

PLASMA_strmm - Computes B = alpha∗op(A)∗B or B = alpha∗B∗op(A).

Parameters

← ***side*** Specifies whether A appears on the left or on the right of X: = PlasmaLeft: A∗X = B = PlasmaRight: X∗A = B

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***transA*** Specifies whether the matrix A is transposed, not transposed or ugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaTrans: A is ugate transposed.

← ***diag*** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A us unit.

← ***N*** The order of the matrix A. N >= 0.

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

← ***A*** The triangular matrix A. If uplo = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If uplo = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If diag = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.

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

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

$<$***0*** if -i, the i-th argument had an illegal value

See also

PLASMA_strmm_Tile
PLASMA_strmm_Tile_Async
PLASMA_ctrmm
PLASMA_dtrmm
PLASMA_strmm

3.4.2.29 int PLASMA_strsm (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, int *N*, int *NRHS*, float *alpha*, float ∗ *A*, int *LDA*, float ∗ *B*, int *LDB*)

PLASMA_strsm - Computes triangular solve A∗X = B or X∗A = B.

Parameters

← ***side*** Specifies whether A appears on the left or on the right of X: = PlasmaLeft: A∗X = B = PlasmaRight: X∗A = B

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***transA*** Specifies whether the matrix A is transposed, not transposed or ugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaTrans: A is ugate transposed.

← ***diag*** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A us unit.

← ***N*** The order of the matrix A. N >= 0.

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

← ***A*** The triangular matrix A. If uplo = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If uplo = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If diag = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.

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

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_strsm_Tile
PLASMA_strsm_Tile_Async
PLASMA_ctrsm
PLASMA_dtrsm
PLASMA_strsm

3.4.2.30 int PLASMA_strsmpl (int *N*, int *NRHS*, float ∗ *A*, int *LDA*, float ∗ *L*, int ∗ *IPIV*, float ∗ *B*, int *LDB*)

PLASMA_strsmpl - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix.

Parameters

← ***N*** The order of the matrix A. N >= 0.

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

← ***A*** The tile factor L from the factorization, computed by PLASMA_sgetrf.

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

← ***L*** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_sgetrf.

← ***IPIV*** The pivot indices from PLASMA_sgetrf (not equivalent to LAPACK).

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

← ***LDB*** The leading dimension of the array B. LDB >= max(1,N).

Returns

Return values

PLASMA_SUCCESS successful exit

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

See also

PLASMA_strsmpl_Tile
PLASMA_strsmpl_Tile_Async
PLASMA_ctrsmpl
PLASMA_dtrsmpl
PLASMA_strsmpl
PLASMA_sgetrf

3.4.2.31 int PLASMA_strtri (PLASMA_enum *uplo*, PLASMA_enum *diag*, int *N*, float ∗ *A*, int *LDA*)

PLASMA_strtri - Computes the inverse of a complex upper or lower triangular matrix A.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***diag*** = PlasmaNonUnit: A is non-unit triangular; = PlasmaUnit: A is unit triangular.

← ***N*** The order of the matrix A. N >= 0.

↔ ***A*** On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1. On exit, the (triangular) inverse of the original matrix, in the same storage format.

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

Returns

Return values

PLASMA_SUCCESS successful exit

<***0*** if -i, the i-th argument had an illegal value

>***0*** if i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

See also

PLASMA_strtri_Tile
PLASMA_strtri_Tile_Async
PLASMA_ctrtri
PLASMA_dtrtri
PLASMA_strtri
PLASMA_spotri

3.5 Advanced Interface: Synchronous - Double Complex

Functions/Subroutines

- int PLASMA_zcgels_Tile (PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_desc ∗X, int ∗ITER)
- int PLASMA_zcgesv_Tile (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_desc ∗B, PLASMA_desc ∗X, int ∗ITER)
- int PLASMA_zcposv_Tile (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_desc ∗X, int ∗ITER)
- int PLASMA_zcungesv_Tile (PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_desc ∗X, int ∗ITER)
- int PLASMA_zgelqf_Tile (PLASMA_desc ∗A, PLASMA_desc ∗T)
- int PLASMA_zgelqs_Tile (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B)
- int PLASMA_zgels_Tile (PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B)
- int PLASMA_zgemm_Tile (PLASMA_enum transA, PLASMA_enum transB, PLASMA_-Complex64_t alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_Complex64_t beta, PLASMA_desc ∗C)
- int PLASMA_zgeqrf_Tile (PLASMA_desc ∗A, PLASMA_desc ∗T)
- int PLASMA_zgeqrs_Tile (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B)
- int PLASMA_zgesv_Tile (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_desc ∗B)
- int PLASMA_zgetrf_Tile (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV)
- int PLASMA_zgetrs_Tile (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_desc ∗B)
- int PLASMA_zhemm_Tile (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex64_t alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_Complex64_t beta, PLASMA_desc ∗C)
- int PLASMA_zher2k_Tile (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex64_t alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, double beta, PLASMA_desc ∗C)
- int PLASMA_zherk_Tile (PLASMA_enum uplo, PLASMA_enum trans, double alpha, PLASMA_-desc ∗A, double beta, PLASMA_desc ∗C)
- double PLASMA_zlange_Tile (PLASMA_enum norm, PLASMA_desc ∗A, double ∗work)
- double PLASMA_zlanhe_Tile (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc ∗A, double ∗work)
- double PLASMA_zlansy_Tile (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc ∗A, double ∗work)
- int PLASMA_zlauum_Tile (PLASMA_enum uplo, PLASMA_desc ∗A)
- int PLASMA_zplghe_Tile (double bump, PLASMA_desc ∗A, unsigned long long int seed)
- int PLASMA_zplgsy_Tile (PLASMA_Complex64_t bump, PLASMA_desc ∗A, unsigned long long int seed)
- int PLASMA_zplrnt_Tile (PLASMA_desc ∗A, unsigned long long int seed)
- int PLASMA_zposv_Tile (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_desc ∗B)
- int PLASMA_zpotrf_Tile (PLASMA_enum uplo, PLASMA_desc ∗A)
- int PLASMA_zpotri_Tile (PLASMA_enum uplo, PLASMA_desc ∗A)
- int PLASMA_zpotrs_Tile (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_desc ∗B)
- int PLASMA_zsymm_Tile (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex64_t alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_Complex64_t beta, PLASMA_desc ∗C)
- int PLASMA_zsyr2k_Tile (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex64_t alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_Complex64_t beta, PLASMA_desc ∗C)
- int PLASMA_zsyrk_Tile (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex64_t alpha, PLASMA_desc ∗A, PLASMA_Complex64_t beta, PLASMA_desc ∗C)

- int PLASMA_ztrmm_Tile (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, PLASMA_Complex64_t alpha, PLASMA_desc *A, PLASMA_desc *B)
- int PLASMA_ztrsm_Tile (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, PLASMA_Complex64_t alpha, PLASMA_desc *A, PLASMA_desc *B)
- int PLASMA_ztrsmpl_Tile (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B)
- int PLASMA_ztrtri_Tile (PLASMA_enum uplo, PLASMA_enum diag, PLASMA_desc *A)
- int PLASMA_zunglq_Tile (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int PLASMA_zungqr_Tile (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *Q)
- int PLASMA_zunmlq_Tile (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int PLASMA_zunmqr_Tile (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)

3.5.1 Detailed Description

This is the group of double complex functions using the advanced synchronous user interface.

3.5.2 Function/Subroutine Documentation

3.5.2.1 int PLASMA_zcgels_Tile (PLASMA_enum *trans*, PLASMA_desc * *A*, PLASMA_desc * *T*, PLASMA_desc * *B*, PLASMA_desc * *X*, int * *ITER*)

PLASMA_zcgels_Tile - Solves overdetermined or underdetermined linear system of equations using the tile QR or the tile LQ factorization and mixed-precision iterative refinement. Tile equivalent of PLASMA_zcgesv(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***trans*** Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaConjTrans: the linear system involves A**H. Currently only PlasmaNoTrans is supported.

↔ ***A***
- If the iterative refinement converged, A is not modified;
- otherwise, it fell back to double precision solution, and on exit the M-by-N matrix A contains: if M >= N, A is overwritten by details of its QR factorization as returned by PLASMA_zgeqrf; if M < N, A is overwritten by details of its LQ factorization as returned by PLASMA_zgelqf.

→ ***T*** On exit:
- if the iterative refinement converged, T is not modified;
- otherwise, it fell back to double precision solution, and then T is an auxiliary factorization data.

↔ ***B*** On entry, the M-by-NRHS matrix B of right hand side vectors, stored columnwise; On exit, if return value = 0, B is overwritten by the solution vectors, stored columnwise: if M >= N, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if M < N, rows 1 to N of B contain the minimum norm solution vectors;

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zcgels
PLASMA_zcgels_Tile_Async
PLASMA_dsgels_Tile
PLASMA_zgels_Tile

3.5.2.2 int PLASMA_zcgesv_Tile (PLASMA_desc $*$ *A*, PLASMA_desc $*$ *L*, int $*$ *IPIV*, PLASMA_desc $*$ *B*, PLASMA_desc $*$ *X*, int $*$ *ITER*)

PLASMA_zcgesv_Tile - Solves a system of linear equations using the tile LU factorization and mixed-precision iterative refinement. Tile equivalent of PLASMA_zcgesv(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

$\leftrightarrow$ ***A*** On entry, the N-by-N coefficient matrix A.

- If the iterative refinement converged, A is not modified;
- otherwise, it fell back to double precision solution, and then A contains the tile L and U factors from the factorization (not equivalent to LAPACK).

$\rightarrow$ ***L*** On exit:

- if the iterative refinement converged, L is not modified;
- otherwise, it fell back to double precision solution, and then L is an auxiliary factorization data, related to the tile L factor, necessary to solve the system of equations (not equivalent to LAPACK).

$\rightarrow$ ***IPIV*** On exit, the pivot indices that define the permutations (not equivalent to LAPACK).

$\leftrightarrow$ ***B*** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

PLASMA_zcgesv
PLASMA_zcgesv_Tile_Async
PLASMA_dsgesv_Tile
PLASMA_zgesv_Tile

3.5.2.3 int PLASMA_zcposv_Tile (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_desc ∗ *X*, int ∗ *ITER*)

PLASMA_zcposv_Tile - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization and mixed-precision iterative refinement. Tile equivalent of PLASMA_zcposv(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

↔ ***A*** On entry, the N-by-N symmetric positive definite (or Hermitian) coefficient matrix A. If uplo = PlasmaUpper, 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.

- If the iterative refinement converged, A is not modified;
- otherwise, it falled backed to double precision solution,

↔ ***B*** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_zcposv
PLASMA_zcposv_Tile_Async
PLASMA_dsposv_Tile
PLASMA_zposv_Tile

3.5.2.4 int PLASMA_zcungesv_Tile (PLASMA_enum *trans*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_desc ∗ *X*, int ∗ *ITER*)

PLASMA_zcungesv_Tile - Solves symmetric linear system of equations using the tile QR or the tile LQ factorization and mixed-precision iterative refinement. Tile equivalent of PLASMA_zcungesv(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***trans*** Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaConjTrans: the linear system involves A∗∗H. Currently only PlasmaNoTrans is supported.

↔ ***A***
- If the iterative refinement converged, A is not modified;

- otherwise, it fell back to double precision solution, and on exit the M-by-N matrix A contains: if M >= N, A is overwritten by details of its QR factorization as returned by PLASMA_zgeqrf; if M < N, A is overwritten by details of its LQ factorization as returned by PLASMA_zgelqf.

$\rightarrow$ ***T*** On exit:

- if the iterative refinement converged, T is not modified;
- otherwise, it fell back to double precision solution, and then T is an auxiliary factorization data.

$\leftrightarrow$ ***B*** On entry, the M-by-NRHS matrix B of right hand side vectors, stored columnwise; On exit, if return value = 0, B is overwritten by the solution vectors, stored columnwise: if M >= N, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if M < N, rows 1 to N of B contain the minimum norm solution vectors;

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zcungesv
PLASMA_zcungesv_Tile_Async
PLASMA_dsungesv_Tile
PLASMA_zgels_Tile

3.5.2.5 int PLASMA_zgelqf_Tile (PLASMA_desc $*$ *A*, PLASMA_desc $*$ *T*)

PLASMA_zgelqf_Tile - Computes the tile LQ factorization of a matrix. Tile equivalent of PLASMA_-zgelqf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

$\leftrightarrow$ ***A*** On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the m-by-min(M,N) lower trapezoidal matrix L (L is lower triangular if M <= N); the elements above the diagonal represent the unitary matrix Q as a product of elementary reflectors, stored by tiles.

$\rightarrow$ ***T*** On exit, auxiliary factorization data, required by PLASMA_zgelqs to solve the system of equations.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zgelqf

PLASMA_zgelqf_Tile_Async
PLASMA_cgelqf_Tile
PLASMA_dgelqf_Tile
PLASMA_sgelqf_Tile
PLASMA_zgelqs_Tile

3.5.2.6 int PLASMA_zgelqs_Tile (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*)

PLASMA_zgelqs_Tile - Computes a minimum-norm solution using previously computed LQ factorization. Tile equivalent of PLASMA_zgelqs(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_zgelqf.

← ***T*** Auxiliary factorization data, computed by PLASMA_zgelqf.

↔ ***B*** On entry, the M-by-NRHS right hand side matrix B. On exit, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zgelqs
PLASMA_zgelqs_Tile_Async
PLASMA_cgelqs_Tile
PLASMA_dgelqs_Tile
PLASMA_sgelqs_Tile
PLASMA_zgelqf_Tile

3.5.2.7 int PLASMA_zgels_Tile (PLASMA_enum *trans*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*)

PLASMA_zgels_Tile - Solves overdetermined or underdetermined linear system of equations using the tile QR or the tile LQ factorization. Tile equivalent of PLASMA_zgels(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***trans*** Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaConjTrans: the linear system involves A∗∗H. Currently only PlasmaNoTrans is supported.

↔ ***A*** On entry, the M-by-N matrix A. On exit, if M >= N, A is overwritten by details of its QR factorization as returned by PLASMA_zgeqrf; if M < N, A is overwritten by details of its LQ factorization as returned by PLASMA_zgelqf.

→ ***T*** On exit, auxiliary factorization data.

↔ ***B*** On entry, the M-by-NRHS matrix B of right hand side vectors, stored columnwise; On exit, if return value = 0, B is overwritten by the solution vectors, stored columnwise: if M >= N, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if M < N, rows 1 to N of B contain the minimum norm solution vectors;

Returns

PLASMA_SUCCESS successful exit

See also

PLASMA_zgels
PLASMA_zgels_Tile_Async
PLASMA_cgels_Tile
PLASMA_dgels_Tile
PLASMA_sgels_Tile

3.5.2.8 int PLASMA_zgemm_Tile (PLASMA_enum *transA*, PLASMA_enum *transB*, PLASMA_Complex64_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_Complex64_t *beta*, PLASMA_desc ∗ *C*)

PLASMA_zgemm_Tile - Performs matrix multiplication. Tile equivalent of PLASMA_zgemm(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***transA*** Specifies whether the matrix A is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is transposed; = PlasmaConjTrans: A is conjugate transposed.

← ***transB*** Specifies whether the matrix B is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: B is not transposed; = PlasmaTrans: B is transposed; = PlasmaConjTrans: B is conjugate transposed.

← ***alpha*** alpha specifies the scalar alpha

← ***A*** A is a LDA-by-ka matrix, where ka is K when transA = PlasmaNoTrans, and is M otherwise.

← ***B*** B is a LDB-by-kb matrix, where kb is N when transB = PlasmaNoTrans, and is K otherwise.

← ***beta*** beta specifies the scalar beta

↔ ***C*** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N matrix (alpha∗op(A)∗op(B) + beta∗C)

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zgemm
PLASMA_zgemm_Tile_Async
PLASMA_cgemm_Tile
PLASMA_dgemm_Tile
PLASMA_sgemm_Tile

3.5.2.9 int PLASMA_zgeqrf_Tile (PLASMA_desc * *A*, PLASMA_desc * *T*)

PLASMA_zgeqrf_Tile - Computes the tile QR factorization of a matrix. Tile equivalent of PLASMA_zgeqrf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ ***A*** 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 represent the unitary matrix Q as a product of elementary reflectors stored by tiles.
- → ***T*** On exit, auxiliary factorization data, required by PLASMA_zgeqrs to solve the system of equations.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zgeqrf
PLASMA_zgeqrf_Tile_Async
PLASMA_cgeqrf_Tile
PLASMA_dgeqrf_Tile
PLASMA_sgeqrf_Tile
PLASMA_zgeqrs_Tile

3.5.2.10 int PLASMA_zgeqrs_Tile (PLASMA_desc * *A*, PLASMA_desc * *T*, PLASMA_desc * *B*)

PLASMA_zgeqrs_Tile - Computes a minimum-norm solution using the tile QR factorization. Tile equivalent of PLASMA_zgetrf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ↔ ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_zgeqrf.
- ← ***T*** Auxiliary factorization data, computed by PLASMA_zgeqrf.
- ↔ ***B*** On entry, the m-by-nrhs right hand side matrix B. On exit, the n-by-nrhs solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zgeqrs

PLASMA_zgeqrs_Tile_Async
PLASMA_cgeqrs_Tile
PLASMA_dgeqrs_Tile
PLASMA_sgeqrs_Tile
PLASMA_zgeqrf_Tile

3.5.2.11 int PLASMA_zgesv_Tile (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*, PLASMA_desc ∗ *B*)

PLASMA_zgesv_Tile - Solves a system of linear equations using the tile LU factorization. Tile equivalent of PLASMA_zgetrf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

↔ ***A*** On entry, the N-by-N coefficient matrix A. On exit, the tile L and U factors from the factorization (not equivalent to LAPACK).

↔ ***L*** On exit, auxiliary factorization data, related to the tile L factor, necessary to solve the system of equations.

→ ***IPIV*** On exit, the pivot indices that define the permutations (not equivalent to LAPACK).

↔ ***B*** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

PLASMA_zgesv
PLASMA_zgesv_Tile_Async
PLASMA_cgesv_Tile
PLASMA_dgesv_Tile
PLASMA_sgesv_Tile
PLASMA_zcgesv_Tile

3.5.2.12 int PLASMA_zgetrf_Tile (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*)

PLASMA_zgetrf_Tile - Computes the tile LU factorization of a matrix. Tile equivalent of PLASMA_zgetrf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

↔ ***A*** On entry, the M-by-N matrix to be factored. On exit, the tile factors L and U from the factorization.

$\rightarrow$ ***L*** On exit, auxiliary factorization data, related to the tile L factor, required by PLASMA_zgetrs to solve the system of equations.

$\rightarrow$ ***IPIV*** The pivot indices that define the permutations (not equivalent to LAPACK).

Returns

Return values

PLASMA_SUCCESS successful exit

$>$***0*** if 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.

See also

PLASMA_zgetrf
PLASMA_zgetrf_Tile_Async
PLASMA_cgetrf_Tile
PLASMA_dgetrf_Tile
PLASMA_sgetrf_Tile
PLASMA_zgetrs_Tile

3.5.2.13 int PLASMA_zgetrs_Tile (PLASMA_desc $*$ *A*, PLASMA_desc $*$ *L*, int $*$ *IPIV*, PLASMA_desc $*$ *B*)

PLASMA_zgetrs_Tile - Solves a system of linear equations using previously computed LU factorization. Tile equivalent of PLASMA_zgetrs(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

$\leftarrow$ ***A*** The tile factors L and U from the factorization, computed by PLASMA_zgetrf.

$\leftarrow$ ***L*** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_zgetrf.

$\leftarrow$ ***IPIV*** The pivot indices from PLASMA_zgetrf (not equivalent to LAPACK).

$\leftrightarrow$ ***B*** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, the solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zgetrs
PLASMA_zgetrs_Tile_Async
PLASMA_cgetrs_Tile
PLASMA_dgetrs_Tile
PLASMA_sgetrs_Tile
PLASMA_zgetrf_Tile

3.5.2.14 int PLASMA_zhemm_Tile (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_Complex64_t *alpha*, PLASMA_desc * *A*, PLASMA_desc * *B*, PLASMA_Complex64_t *beta*, PLASMA_desc * *C*)

PLASMA_zhemm_Tile - Performs Hermitian matrix multiplication. Tile equivalent of PLASMA_-zhemm(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***side*** Specifies whether the hermitian matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

← ***uplo*** Specifies whether the upper or lower triangular part of the hermitian matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the hermitian matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the hermitian matrix A is to be referenced.

← ***alpha*** Specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.

← ***B*** B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.

← ***beta*** Specifies the scalar beta.

↔ ***C*** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zhemm
PLASMA_zhemm_Tile_Async
PLASMA_chemm_Tile
PLASMA_dhemm_Tile
PLASMA_shemm_Tile

3.5.2.15 int PLASMA_zher2k_Tile (PLASMA_enum *uplo*, PLASMA_enum *trans*, PLASMA_Complex64_t *alpha*, PLASMA_desc * *A*, PLASMA_desc * *B*, double *beta*, PLASMA_desc * *C*)

PLASMA_zher2k_Tile - Performs hermitian rank k update. Tile equivalent of PLASMA_zher2k(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← ***trans*** Specifies whether the matrix A is transposed or conjugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaConjTrans: A is conjugate transposed.

← ***alpha*** alpha specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.

← ***beta*** beta specifies the scalar beta

↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zher2k_Tile
PLASMA_cher2k
PLASMA_dher2k
PLASMA_sher2k

3.5.2.16 int PLASMA_zherk_Tile (PLASMA_enum *uplo*, PLASMA_enum *trans*, double *alpha*, PLASMA_desc ∗ *A*, double *beta*, PLASMA_desc ∗ *C*)

PLASMA_zherk_Tile - Performs hermitian rank k update. Tile equivalent of PLASMA_zherk(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← ***trans*** Specifies whether the matrix A is transposed or conjugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaConjTrans: A is conjugate transposed.

← ***alpha*** alpha specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.

← ***beta*** beta specifies the scalar beta

↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zherk_Tile
PLASMA_cherk
PLASMA_dherk
PLASMA_sherk

3.5.2.17 double PLASMA_zlange_Tile (PLASMA_enum *norm*, PLASMA_desc ∗ *A*, double ∗ *work*)

PLASMA_zlange_Tile - Tile equivalent of PLASMA_zlange(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U ∗ U'; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L' ∗ L.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zlange
PLASMA_zlange_Tile_Async
PLASMA_clange_Tile
PLASMA_dlange_Tile
PLASMA_slange_Tile

3.5.2.18 double PLASMA_zlanhe_Tile (PLASMA_enum *norm*, PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, double ∗ *work*)

PLASMA_zlanhe_Tile - Tile equivalent of PLASMA_zlanhe(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U ∗ U'; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L' ∗ L.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zlanhe
PLASMA_zlanhe_Tile_Async
PLASMA_clanhe_Tile
PLASMA_dlanhe_Tile
PLASMA_slanhe_Tile

3.5.2.19 double PLASMA_zlansy_Tile (PLASMA_enum *norm*, PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, double ∗ *work*)

PLASMA_zlansy_Tile - Tile equivalent of PLASMA_zlansy(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U ∗ U'; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L' ∗ L.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zlansy
PLASMA_zlansy_Tile_Async
PLASMA_clansy_Tile
PLASMA_dlansy_Tile
PLASMA_slansy_Tile

3.5.2.20 int PLASMA_zlauum_Tile (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*)

PLASMA_zlauum_Tile - Computes the product U ∗ U' or L' ∗ L, where the triangular factor U or L is stored in the upper or lower triangular part of the array A. Tile equivalent of PLASMA_zlauum(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U ∗ U'; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L' ∗ L.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zlauum
PLASMA_zlauum_Tile_Async
PLASMA_clauum_Tile
PLASMA_dlauum_Tile
PLASMA_slauum_Tile
PLASMA_zpotri_Tile

3.5.2.21 int PLASMA_zplghe_Tile (double *bump*, PLASMA_desc $*$ *A*, unsigned long long int *seed*)

PLASMA_zplghe_Tile - Generate a random hermitian matrix by tiles. Tile equivalent of PLASMA_-zplghe(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

$\leftarrow$ ***A*** On exit, The random hermitian matrix A generated.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zplghe
PLASMA_zplghe_Tile_Async
PLASMA_cplghe_Tile
PLASMA_dplghe_Tile
PLASMA_splghe_Tile
PLASMA_zplrnt_Tile
PLASMA_zplgsy_Tile

3.5.2.22 int PLASMA_zplgsy_Tile (PLASMA_Complex64_t *bump*, PLASMA_desc $*$ *A*, unsigned long long int *seed*)

PLASMA_zplgsy_Tile - Generate a random hermitian matrix by tiles. Tile equivalent of PLASMA_-zplgsy(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

$\leftarrow$ ***A*** On exit, The random hermitian matrix A generated.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zplgsy
PLASMA_zplgsy_Tile_Async
PLASMA_cplgsy_Tile
PLASMA_dplgsy_Tile
PLASMA_splgsy_Tile
PLASMA_zplrnt_Tile
PLASMA_zplgsy_Tile

3.5.2.23 int PLASMA_zplrnt_Tile (PLASMA_desc ∗ *A*, unsigned long long int *seed*)

PLASMA_zplrnt_Tile - Generate a random matrix by tiles. Tile equivalent of PLASMA_zplrnt(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***A*** On exit, The random matrix A generated.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zplrnt
PLASMA_zplrnt_Tile_Async
PLASMA_cplrnt_Tile
PLASMA_dplrnt_Tile
PLASMA_splrnt_Tile
PLASMA_zplghe_Tile
PLASMA_zplgsy_Tile

3.5.2.24 int PLASMA_zposv_Tile (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*)

PLASMA_zposv_Tile - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization. Tile equivalent of PLASMA_zposv(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

↔ ***A*** On entry, the symmetric positive definite (or Hermitian) matrix A. If uplo = PlasmaUpper, 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 return value = 0, the factor U or L from the Cholesky factorization A = U∗∗H∗U or A = L∗L∗∗H.

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_zposv
PLASMA_zposv_Tile_Async
PLASMA_cposv_Tile
PLASMA_dposv_Tile
PLASMA_sposv_Tile

3.5.2.25 int PLASMA_zpotrf_Tile (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*)

PLASMA_zpotrf_Tile - Computes the Cholesky factorization of a symmetric positive definite or Hermitian positive definite matrix. Tile equivalent of PLASMA_zpotrf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** On entry, the symmetric positive definite (or Hermitian) matrix A. If uplo = PlasmaUpper, 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 return value = 0, the factor U or L from the Cholesky factorization A = U∗∗H∗U or A = L∗L∗∗H.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_zpotrf
PLASMA_zpotrf_Tile_Async
PLASMA_cpotrf_Tile
PLASMA_dpotrf_Tile
PLASMA_spotrf_Tile
PLASMA_zpotrs_Tile

3.5.2.26 int PLASMA_zpotri_Tile (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*)

PLASMA_zpotri_Tile - Computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization A = U∗∗H∗U or A = L∗L∗∗H computed by PLASMA_zpotrf. Tile equivalent of PLASMA_zpotri(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** On entry, the triangular factor U or L from the Cholesky factorization A = U∗∗H∗U or A = L∗L∗∗H, as computed by PLASMA_zpotrf. On exit, the upper or lower triangle of the (Hermitian) inverse of A, overwriting the input factor U or L.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_zpotri
PLASMA_zpotri_Tile_Async
PLASMA_cpotri_Tile
PLASMA_dpotri_Tile
PLASMA_spotri_Tile
PLASMA_zpotrf_Tile

3.5.2.27 int PLASMA_zpotrs_Tile (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*)

PLASMA_zpotrs_Tile - Solves a system of linear equations using previously computed Cholesky factorization. Tile equivalent of PLASMA_zpotrs(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** The triangular factor U or L from the Cholesky factorization A = U**H*U or A = L*L**H, computed by PLASMA_zpotrf.

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zpotrs
PLASMA_zpotrs_Tile_Async
PLASMA_cpotrs_Tile
PLASMA_dpotrs_Tile
PLASMA_spotrs_Tile
PLASMA_zpotrf_Tile

3.5.2.28 int PLASMA_zsymm_Tile (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_Complex64_t *alpha*, PLASMA_desc $*$ *A*, PLASMA_desc $*$ *B*, PLASMA_Complex64_t *beta*, PLASMA_desc $*$ *C*)

PLASMA_zsymm_Tile - Performs symmetric matrix multiplication. Tile equivalent of PLASMA_-zsymm(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***side*** Specifies whether the symmetric matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

← ***uplo*** Specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the symmetric matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the symmetric matrix A is to be referenced.

← ***alpha*** Specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.

← ***B*** B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.

← ***beta*** Specifies the scalar beta.

↔ ***C*** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zsymm
PLASMA_zsymm_Tile_Async
PLASMA_csymm_Tile
PLASMA_dsymm_Tile
PLASMA_ssymm_Tile

3.5.2.29 int PLASMA_zsyr2k_Tile (PLASMA_enum *uplo*, PLASMA_enum *trans*, PLASMA_Complex64_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_Complex64_t *beta*, PLASMA_desc ∗ *C*)

PLASMA_zsyr2k_Tile - Performs symmetric rank k update. Tile equivalent of PLASMA_zsyr2k(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← ***trans*** Specifies whether the matrix A is transposed or conjugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is conjugate transposed.

← ***alpha*** alpha specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.

← ***beta*** beta specifies the scalar beta

↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zsyr2k_Tile
PLASMA_csyr2k
PLASMA_dsyr2k
PLASMA_ssyr2k

3.5.2.30 int PLASMA_zsyrk_Tile (PLASMA_enum *uplo*, PLASMA_enum *trans*, PLASMA_Complex64_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_Complex64_t *beta*, PLASMA_desc ∗ *C*)

PLASMA_zsyrk_Tile - Performs rank k update. Tile equivalent of PLASMA_zsyrk(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← ***trans*** Specifies whether the matrix A is transposed or conjugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is transposed.

← ***alpha*** alpha specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.

← ***beta*** beta specifies the scalar beta

↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zsyrk_Tile
PLASMA_csyrk
PLASMA_dsyrk
PLASMA_ssyrk

3.5.2.31 int PLASMA_ztrmm_Tile (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, PLASMA_Complex64_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*)

PLASMA_ztrmm_Tile - Computes triangular solve. Tile equivalent of PLASMA_ztrmm(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***side*** Specifies whether A appears on the left or on the right of X: = PlasmaLeft: A∗X = B = PlasmaRight: X∗A = B

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***transA*** Specifies whether the matrix A is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaConjTrans: A is conjugate transposed.

← ***diag*** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A us unit.

← ***A*** The triangular matrix A. If uplo = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If uplo = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If diag = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_ztrmm
PLASMA_ztrmm_Tile_Async
PLASMA_ctrmm_Tile
PLASMA_dtrmm_Tile
PLASMA_strmm_Tile

3.5.2.32 int PLASMA_ztrsm_Tile (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, PLASMA_Complex64_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*)

PLASMA_ztrsm_Tile - Computes triangular solve. Tile equivalent of PLASMA_ztrsm(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***side*** Specifies whether A appears on the left or on the right of X: = PlasmaLeft: A∗X = B = PlasmaRight: X∗A = B

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***transA*** Specifies whether the matrix A is transposed, not transposed or conjugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaConjTrans: A is conjugate transposed.

← ***diag*** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A us unit.

← ***A*** The triangular matrix A. If uplo = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If uplo = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If diag = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_ztrsm
PLASMA_ztrsm_Tile_Async
PLASMA_ctrsm_Tile
PLASMA_dtrsm_Tile
PLASMA_strsm_Tile

3.5.2.33 int PLASMA_ztrsmpl_Tile (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*, PLASMA_desc ∗ *B*)

PLASMA_ztrsmpl_Tile - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***A*** The tile factor L from the factorization, computed by PLASMA_zgetrf.

← ***L*** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_zgetrf.

← ***IPIV*** The pivot indices from PLASMA_zgetrf (not equivalent to LAPACK).

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_ztrsmpl
PLASMA_ztrsmpl_Tile_Async
PLASMA_ctrsmpl_Tile
PLASMA_dtrsmpl_Tile
PLASMA_strsmpl_Tile
PLASMA_zgetrf_Tile

3.5.2.34 int PLASMA_ztrtri_Tile (PLASMA_enum *uplo*, PLASMA_enum *diag*, PLASMA_desc ∗ *A*)

PLASMA_ztrtri_Tile - Computes the inverse of a complex upper or lower triangular matrix A. Tile equivalent of PLASMA_ztrtri(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***diag*** = PlasmaNonUnit: A is non-unit triangular; = PlasmaUnit: A us unit triangular.

← ***A*** On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1. On exit, the (triangular) inverse of the original matrix, in the same storage format.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

See also

PLASMA_ztrtri
PLASMA_ztrtri_Tile_Async
PLASMA_ctrtri_Tile
PLASMA_dtrtri_Tile
PLASMA_strtri_Tile
PLASMA_zpotri_Tile

3.5.2.35 int PLASMA_zunglq_Tile (PLASMA_desc * *A*, PLASMA_desc * *T*, PLASMA_desc * *B*)

PLASMA_zunglq_Tile - Generates an M-by-N matrix Q with orthonormal rows, which is defined as the first M rows of a product of the elementary reflectors returned by PLASMA_zgelqf. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_zgelqf.

← ***T*** Auxiliary factorization data, computed by PLASMA_zgelqf.

→ ***B*** On exit, the M-by-N matrix Q.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zunglq
PLASMA_zunglq_Tile_Async
PLASMA_cunglq_Tile
PLASMA_dunglq_Tile
PLASMA_sunglq_Tile
PLASMA_zgelqf_Tile

3.5.2.36 int PLASMA_zungqr_Tile (PLASMA_desc * *A*, PLASMA_desc * *T*, PLASMA_desc * *Q*)

PLASMA_zungqr_Tile - Generates an M-by-N matrix Q with orthonormal columns, which is defined as the first N columns of a product of the elementary reflectors returned by PLASMA_zgeqrf. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_zgeqrf.

← ***T*** Auxiliary factorization data, computed by PLASMA_zgeqrf.

→ ***Q*** On exit, the M-by-N matrix Q.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zungqr
PLASMA_zungqr_Tile_Async
PLASMA_cungqr_Tile
PLASMA_dungqr_Tile
PLASMA_sungqr_Tile
PLASMA_zgeqrf_Tile

3.5.2.37 int PLASMA_zunmlq_Tile (PLASMA_enum *side*, PLASMA_enum *trans*, PLASMA_desc * *A*, PLASMA_desc * *T*, PLASMA_desc * *B*)

PLASMA_zunmlq_Tile - overwrites the general M-by-N matrix C with Q*C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_-zgelqf_Tile Q is of order M. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***side*** Intended usage: = PlasmaLeft: apply Q or Q**H from the left; = PlasmaRight: apply Q or Q**H from the right. Currently only PlasmaLeft is supported.

← ***trans*** Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaConjTrans: conjugate transpose, apply Q**H. Currently only PlasmaConjTrans is supported.

← ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_zgelqf.

← ***T*** Auxiliary factorization data, computed by PLASMA_zgelqf.

↔ ***B*** On entry, the M-by-N matrix B. On exit, B is overwritten by Q*B or Q**H*B.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zunmlq
PLASMA_zunmlq_Tile_Async
PLASMA_cunmlq_Tile
PLASMA_dunmlq_Tile
PLASMA_sunmlq_Tile
PLASMA_zgelqf_Tile

3.5.2.38 int PLASMA_zunmqr_Tile (PLASMA_enum *side*, PLASMA_enum *trans*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*)

PLASMA_zunmqr_Tile - overwrites the general M-by-N matrix C with Q∗C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_-zgeqrf_Tile Q is of order M. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***side*** Intended usage: = PlasmaLeft: apply Q or Q∗∗H from the left; = PlasmaRight: apply Q or Q∗∗H from the right. Currently only PlasmaLeft is supported.

← ***trans*** Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaConjTrans: conjugate transpose, apply Q∗∗H. Currently only PlasmaConjTrans is supported.

← ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_zgeqrf.

← ***T*** Auxiliary factorization data, computed by PLASMA_zgeqrf.

↔ ***B*** On entry, the M-by-N matrix B. On exit, B is overwritten by Q∗B or Q∗∗H∗B.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_zunmqr
PLASMA_zunmqr_Tile_Async
PLASMA_cunmqr_Tile
PLASMA_dunmqr_Tile
PLASMA_sunmqr_Tile
PLASMA_zgeqrf_Tile

3.6 Advanced Interface: Synchronous - Single Complex

Functions/Subroutines

- int PLASMA_cgelqf_Tile (PLASMA_desc *A, PLASMA_desc *T)
- int PLASMA_cgelqs_Tile (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int PLASMA_cgels_Tile (PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int PLASMA_cgemm_Tile (PLASMA_enum transA, PLASMA_enum transB, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_Complex32_t beta, PLASMA_desc *C)
- int PLASMA_cgeqrf_Tile (PLASMA_desc *A, PLASMA_desc *T)
- int PLASMA_cgeqrs_Tile (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int PLASMA_cgesv_Tile (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B)
- int PLASMA_cgetrf_Tile (PLASMA_desc *A, PLASMA_desc *L, int *IPIV)
- int PLASMA_cgetrs_Tile (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B)
- int PLASMA_chemm_Tile (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_Complex32_t beta, PLASMA_desc *C)
- int PLASMA_cher2k_Tile (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B, float beta, PLASMA_desc *C)
- int PLASMA_cherk_Tile (PLASMA_enum uplo, PLASMA_enum trans, float alpha, PLASMA_desc *A, float beta, PLASMA_desc *C)
- float PLASMA_clange_Tile (PLASMA_enum norm, PLASMA_desc *A, float *work)
- float PLASMA_clanhe_Tile (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc *A, float *work)
- float PLASMA_clansy_Tile (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc *A, float *work)
- int PLASMA_clauum_Tile (PLASMA_enum uplo, PLASMA_desc *A)
- int PLASMA_cplghe_Tile (float bump, PLASMA_desc *A, unsigned long long int seed)
- int PLASMA_cplgsy_Tile (PLASMA_Complex32_t bump, PLASMA_desc *A, unsigned long long int seed)
- int PLASMA_cplrnt_Tile (PLASMA_desc *A, unsigned long long int seed)
- int PLASMA_cposv_Tile (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B)
- int PLASMA_cpotrf_Tile (PLASMA_enum uplo, PLASMA_desc *A)
- int PLASMA_cpotri_Tile (PLASMA_enum uplo, PLASMA_desc *A)
- int PLASMA_cpotrs_Tile (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B)
- int PLASMA_csymm_Tile (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_Complex32_t beta, PLASMA_desc *C)
- int PLASMA_csyr2k_Tile (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_Complex32_t beta, PLASMA_desc *C)
- int PLASMA_csyrk_Tile (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_Complex32_t beta, PLASMA_desc *C)
- int PLASMA_ctrmm_Tile (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B)
- int PLASMA_ctrsm_Tile (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B)
- int PLASMA_ctrsmpl_Tile (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B)
- int PLASMA_ctrtri_Tile (PLASMA_enum uplo, PLASMA_enum diag, PLASMA_desc *A)
- int PLASMA_cunglq_Tile (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B)
- int PLASMA_cungqr_Tile (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *Q)

- int PLASMA_cunmlq_Tile (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B)
- int PLASMA_cunmqr_Tile (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B)

3.6.1 Detailed Description

This is the group of single complex functions using the advanced synchronous interface.

3.6.2 Function/Subroutine Documentation

3.6.2.1 int PLASMA_cgelqf_Tile (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*)

PLASMA_cgelqf_Tile - Computes the tile LQ factorization of a matrix. Tile equivalent of PLASMA_cgelqf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

↔ ***A*** On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the m-by-min(M,N) lower trapezoidal matrix L (L is lower triangular if M <= N); the elements above the diagonal represent the unitary matrix Q as a product of elementary reflectors, stored by tiles.

→ ***T*** On exit, auxiliary factorization data, required by PLASMA_cgelqs to solve the system of equations.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_cgelqf
PLASMA_cgelqf_Tile_Async
PLASMA_cgelqf_Tile
PLASMA_dgelqf_Tile
PLASMA_sgelqf_Tile
PLASMA_cgelqs_Tile

3.6.2.2 int PLASMA_cgelqs_Tile (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*)

PLASMA_cgelqs_Tile - Computes a minimum-norm solution using previously computed LQ factorization. Tile equivalent of PLASMA_cgelqs(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_cgelqf.

← ***T*** Auxiliary factorization data, computed by PLASMA_cgelqf.

↔ ***B*** On entry, the M-by-NRHS right hand side matrix B. On exit, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_cgelqs
PLASMA_cgelqs_Tile_Async
PLASMA_cgelqs_Tile
PLASMA_dgelqs_Tile
PLASMA_sgelqs_Tile
PLASMA_cgelqf_Tile

3.6.2.3 int PLASMA_cgels_Tile (PLASMA_enum *trans*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*)

PLASMA_cgels_Tile - Solves overdetermined or underdetermined linear system of equations using the tile QR or the tile LQ factorization. Tile equivalent of PLASMA_cgels(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***trans*** Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaConjTrans: the linear system involves A\∗\∗H. Currently only PlasmaNoTrans is supported.

↔ ***A*** On entry, the M-by-N matrix A. On exit, if M >= N, A is overwritten by details of its QR factorization as returned by PLASMA_cgeqrf; if M < N, A is overwritten by details of its LQ factorization as returned by PLASMA_cgelqf.

→ ***T*** On exit, auxiliary factorization data.

↔ ***B*** On entry, the M-by-NRHS matrix B of right hand side vectors, stored columnwise; On exit, if return value = 0, B is overwritten by the solution vectors, stored columnwise: if M >= N, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if M < N, rows 1 to N of B contain the minimum norm solution vectors;

Returns

PLASMA_SUCCESS successful exit

See also

PLASMA_cgels
PLASMA_cgels_Tile_Async
PLASMA_cgels_Tile
PLASMA_dgels_Tile
PLASMA_sgels_Tile

3.6.2.4 int PLASMA_cgemm_Tile (PLASMA_enum *transA*, PLASMA_enum *transB*, PLASMA_Complex32_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_Complex32_t *beta*, PLASMA_desc ∗ *C*)

PLASMA_cgemm_Tile - Performs matrix multiplication. Tile equivalent of PLASMA_cgemm(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***transA*** Specifies whether the matrix A is transposed, not transposed or conjfugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is transposed; = PlasmaConjTrans: A is conjfugate transposed.

← ***transB*** Specifies whether the matrix B is transposed, not transposed or conjfugate transposed: = PlasmaNoTrans: B is not transposed; = PlasmaTrans: B is transposed; = PlasmaConjTrans: B is conjfugate transposed.

← ***alpha*** alpha specifies the scalar alpha

← ***A*** A is a LDA-by-ka matrix, where ka is K when transA = PlasmaNoTrans, and is M otherwise.

← ***B*** B is a LDB-by-kb matrix, where kb is N when transB = PlasmaNoTrans, and is K otherwise.

← ***beta*** beta specifies the scalar beta

↔ ***C*** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N matrix (alpha∗op(A)∗op(B) + beta∗C)

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_cgemm
PLASMA_cgemm_Tile_Async
PLASMA_cgemm_Tile
PLASMA_dgemm_Tile
PLASMA_sgemm_Tile

3.6.2.5 int PLASMA_cgeqrf_Tile (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*)

PLASMA_cgeqrf_Tile - Computes the tile QR factorization of a matrix. Tile equivalent of PLASMA_-cgeqrf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

↔ ***A*** 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 represent the unitary matrix Q as a product of elementary reflectors stored by tiles.

→ ***T*** On exit, auxiliary factorization data, required by PLASMA_cgeqrs to solve the system of equations.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_cgeqrf
PLASMA_cgeqrf_Tile_Async
PLASMA_cgeqrf_Tile
PLASMA_dgeqrf_Tile
PLASMA_sgeqrf_Tile
PLASMA_cgeqrs_Tile

3.6.2.6 int PLASMA_cgeqrs_Tile (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*)

PLASMA_cgeqrs_Tile - Computes a minimum-norm solution using the tile QR factorization. Tile equivalent of PLASMA_cgetrf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

↔ ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_cgeqrf.

← ***T*** Auxiliary factorization data, computed by PLASMA_cgeqrf.

↔ ***B*** On entry, the m-by-nrhs right hand side matrix B. On exit, the n-by-nrhs solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_cgeqrs
PLASMA_cgeqrs_Tile_Async
PLASMA_cgeqrs_Tile
PLASMA_dgeqrs_Tile
PLASMA_sgeqrs_Tile
PLASMA_cgeqrf_Tile

3.6.2.7 int PLASMA_cgesv_Tile (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*, PLASMA_desc ∗ *B*)

PLASMA_cgesv_Tile - Solves a system of linear equations using the tile LU factorization. Tile equivalent of PLASMA_cgetrf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

↔ ***A*** On entry, the N-by-N coefficient matrix A. On exit, the tile L and U factors from the factorization (not equivalent to LAPACK).

↔ ***L*** On exit, auxiliary factorization data, related to the tile L factor, necessary to solve the system of equations.

→ ***IPIV*** On exit, the pivot indices that define the permutations (not equivalent to LAPACK).

↔ ***B*** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

PLASMA_cgesv
PLASMA_cgesv_Tile_Async
PLASMA_cgesv_Tile
PLASMA_dgesv_Tile
PLASMA_sgesv_Tile
PLASMA_ccgesv_Tile

3.6.2.8 int PLASMA_cgetrf_Tile (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*)

PLASMA_cgetrf_Tile - Computes the tile LU factorization of a matrix. Tile equivalent of PLASMA_cgetrf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

↔ ***A*** On entry, the M-by-N matrix to be factored. On exit, the tile factors L and U from the factorization.

→ ***L*** On exit, auxiliary factorization data, related to the tile L factor, required by PLASMA_cgetrs to solve the system of equations.

→ ***IPIV*** The pivot indices that define the permutations (not equivalent to LAPACK).

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if 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.

See also

PLASMA_cgetrf
PLASMA_cgetrf_Tile_Async
PLASMA_cgetrf_Tile
PLASMA_dgetrf_Tile
PLASMA_sgetrf_Tile
PLASMA_cgetrs_Tile

3.6.2.9 int PLASMA_cgetrs_Tile (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*, PLASMA_desc ∗ *B*)

PLASMA_cgetrs_Tile - Solves a system of linear equations using previously computed LU factorization. Tile equivalent of PLASMA_cgetrs(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***A*** The tile factors L and U from the factorization, computed by PLASMA_cgetrf.
← ***L*** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_cgetrf.
← ***IPIV*** The pivot indices from PLASMA_cgetrf (not equivalent to LAPACK).
↔ ***B*** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, the solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_cgetrs
PLASMA_cgetrs_Tile_Async
PLASMA_cgetrs_Tile
PLASMA_dgetrs_Tile
PLASMA_sgetrs_Tile
PLASMA_cgetrf_Tile

3.6.2.10 int PLASMA_chemm_Tile (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_Complex32_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_Complex32_t *beta*, PLASMA_desc ∗ *C*)

PLASMA_chemm_Tile - Performs Hermitian matrix multiplication. Tile equivalent of PLASMA_-chemm(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***side*** Specifies whether the hermitian matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

← ***uplo*** Specifies whether the upper or lower triangular part of the hermitian matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the hermitian matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the hermitian matrix A is to be referenced.

← ***alpha*** Specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.

← ***B*** B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.

← ***beta*** Specifies the scalar beta.

↔ ***C*** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_chemm
PLASMA_chemm_Tile_Async
PLASMA_chemm_Tile
PLASMA_dhemm_Tile
PLASMA_shemm_Tile

3.6.2.11 int PLASMA_cher2k_Tile (PLASMA_enum *uplo*, PLASMA_enum *trans*, PLASMA_Complex32_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, float *beta*, PLASMA_desc ∗ *C*)

PLASMA_cher2k_Tile - Performs hermitian rank k update. Tile equivalent of PLASMA_cher2k(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← ***trans*** Specifies whether the matrix A is transposed or conjfugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaConjTrans: A is conjfugate transposed.

← ***alpha*** alpha specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.

← ***beta*** beta specifies the scalar beta

↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_cher2k_Tile
PLASMA_cher2k
PLASMA_dher2k
PLASMA_sher2k

3.6.2.12 int PLASMA_cherk_Tile (PLASMA_enum *uplo*, PLASMA_enum *trans*, float *alpha*, PLASMA_desc $*$ *A*, float *beta*, PLASMA_desc $*$ *C*)

PLASMA_cherk_Tile - Performs hermitian rank k update. Tile equivalent of PLASMA_cherk(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- $\leftarrow$ ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.
- $\leftarrow$ ***trans*** Specifies whether the matrix A is transposed or conjfugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaConjTrans: A is conjfugate transposed.
- $\leftarrow$ ***alpha*** alpha specifies the scalar alpha.
- $\leftarrow$ ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.
- $\leftarrow$ ***beta*** beta specifies the scalar beta
- $\leftrightarrow$ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_cherk_Tile
PLASMA_cherk
PLASMA_dherk
PLASMA_sherk

3.6.2.13 float PLASMA_clange_Tile (PLASMA_enum *norm*, PLASMA_desc $*$ *A*, float $*$ *work*)

PLASMA_clange_Tile - Tile equivalent of PLASMA_clange(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- $\leftarrow$ ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U ∗ U'; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L' ∗ L.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_clange
PLASMA_clange_Tile_Async
PLASMA_clange_Tile
PLASMA_dlange_Tile
PLASMA_slange_Tile

3.6.2.14 float PLASMA_clanhe_Tile (PLASMA_enum *norm*, PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, float ∗ *work*)

PLASMA_clanhe_Tile - Tile equivalent of PLASMA_clanhe(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U ∗ U'; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L' ∗ L.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_clanhe
PLASMA_clanhe_Tile_Async
PLASMA_clanhe_Tile
PLASMA_dlanhe_Tile
PLASMA_slanhe_Tile

3.6.2.15 float PLASMA_clansy_Tile (PLASMA_enum *norm*, PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, float ∗ *work*)

PLASMA_clansy_Tile - Tile equivalent of PLASMA_clansy(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U ∗ U'; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L' ∗ L.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_clansy
PLASMA_clansy_Tile_Async
PLASMA_clansy_Tile
PLASMA_dlansy_Tile
PLASMA_slansy_Tile

3.6.2.16 int PLASMA_clauum_Tile (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*)

PLASMA_clauum_Tile - Computes the product U ∗ U' or L' ∗ L, where the triangular factor U or L is stored in the upper or lower triangular part of the array A. Tile equivalent of PLASMA_clauum(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U ∗ U'; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L' ∗ L.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_clauum
PLASMA_clauum_Tile_Async
PLASMA_clauum_Tile
PLASMA_dlauum_Tile
PLASMA_slauum_Tile
PLASMA_cpotri_Tile

3.6.2.17 int PLASMA_cplghe_Tile (float *bump*, PLASMA_desc ∗ *A*, unsigned long long int *seed*)

PLASMA_cplghe_Tile - Generate a random hermitian matrix by tiles. Tile equivalent of PLASMA_-cplghe(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***A*** On exit, The random hermitian matrix A generated.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_cplghe
PLASMA_cplghe_Tile_Async
PLASMA_cplghe_Tile
PLASMA_dplghe_Tile
PLASMA_splghe_Tile
PLASMA_cplrnt_Tile
PLASMA_cplgsy_Tile

3.6.2.18 int PLASMA_cplgsy_Tile (PLASMA_Complex32_t *bump*, PLASMA_desc ∗ *A*, unsigned long long int *seed*)

PLASMA_cplgsy_Tile - Generate a random hermitian matrix by tiles. Tile equivalent of PLASMA_-cplgsy(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***A*** On exit, The random hermitian matrix A generated.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_cplgsy
PLASMA_cplgsy_Tile_Async
PLASMA_cplgsy_Tile
PLASMA_dplgsy_Tile
PLASMA_splgsy_Tile
PLASMA_cplrnt_Tile
PLASMA_cplgsy_Tile

3.6.2.19 int PLASMA_cplrnt_Tile (PLASMA_desc $*$ *A*, unsigned long long int *seed*)

PLASMA_cplrnt_Tile - Generate a random matrix by tiles. Tile equivalent of PLASMA_cplrnt(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

$\leftarrow$ ***A*** On exit, The random matrix A generated.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_cplrnt
PLASMA_cplrnt_Tile_Async
PLASMA_cplrnt_Tile
PLASMA_dplrnt_Tile
PLASMA_splrnt_Tile
PLASMA_cplghe_Tile
PLASMA_cplgsy_Tile

3.6.2.20 int PLASMA_cposv_Tile (PLASMA_enum *uplo*, PLASMA_desc $*$ *A*, PLASMA_desc $*$ *B*)

PLASMA_cposv_Tile - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization. Tile equivalent of PLASMA_cposv(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

$\leftarrow$ ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

$\leftrightarrow$ ***A*** On entry, the symmetric positive definite (or Hermitian) matrix A. If uplo = PlasmaUpper, 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 return value = 0, the factor U or L from the Cholesky factorization A = U**H*U or A = L*L**H.

$\leftrightarrow$ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_cposv
PLASMA_cposv_Tile_Async
PLASMA_cposv_Tile
PLASMA_dposv_Tile
PLASMA_sposv_Tile

3.6.2.21 int PLASMA_cpotrf_Tile (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*)

PLASMA_cpotrf_Tile - Computes the Cholesky factorization of a symmetric positive definite or Hermitian positive definite matrix. Tile equivalent of PLASMA_cpotrf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** On entry, the symmetric positive definite (or Hermitian) matrix A. If uplo = PlasmaUpper, 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 return value = 0, the factor U or L from the Cholesky factorization A = U\∗\∗H∗U or A = L∗L\∗\∗H.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_cpotrf
PLASMA_cpotrf_Tile_Async
PLASMA_cpotrf_Tile
PLASMA_dpotrf_Tile
PLASMA_spotrf_Tile
PLASMA_cpotrs_Tile

3.6.2.22 int PLASMA_cpotri_Tile (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*)

PLASMA_cpotri_Tile - Computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization A = U\∗\∗H∗U or A = L∗L\∗\∗H computed by PLASMA_cpotrf. Tile equivalent of PLASMA_cpotri(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** On entry, the triangular factor U or L from the Cholesky factorization A = U**H*U or A = L*L**H, as computed by PLASMA_cpotrf. On exit, the upper or lower triangle of the (Hermitian) inverse of A, overwriting the input factor U or L.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_cpotri
PLASMA_cpotri_Tile_Async
PLASMA_cpotri_Tile
PLASMA_dpotri_Tile
PLASMA_spotri_Tile
PLASMA_cpotrf_Tile

3.6.2.23 int PLASMA_cpotrs_Tile (PLASMA_enum *uplo*, PLASMA_desc * *A*, PLASMA_desc * *B*)

PLASMA_cpotrs_Tile - Solves a system of linear equations using previously computed Cholesky factorization. Tile equivalent of PLASMA_cpotrs(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** The triangular factor U or L from the Cholesky factorization A = U**H*U or A = L*L**H, computed by PLASMA_cpotrf.

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_cpotrs
PLASMA_cpotrs_Tile_Async
PLASMA_cpotrs_Tile
PLASMA_dpotrs_Tile
PLASMA_spotrs_Tile
PLASMA_cpotrf_Tile

3.6.2.24 int PLASMA_csymm_Tile (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_Complex32_t *alpha*, PLASMA_desc * *A*, PLASMA_desc * *B*, PLASMA_Complex32_t *beta*, PLASMA_desc * *C*)

PLASMA_csymm_Tile - Performs symmetric matrix multiplication. Tile equivalent of PLASMA_csymm(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***side*** Specifies whether the symmetric matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

← ***uplo*** Specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the symmetric matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the symmetric matrix A is to be referenced.

← ***alpha*** Specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.

← ***B*** B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.

← ***beta*** Specifies the scalar beta.

↔ ***C*** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_csymm
PLASMA_csymm_Tile_Async
PLASMA_csymm_Tile
PLASMA_dsymm_Tile
PLASMA_ssymm_Tile

3.6.2.25 int PLASMA_csyr2k_Tile (PLASMA_enum *uplo*, PLASMA_enum *trans*, PLASMA_Complex32_t *alpha*, PLASMA_desc * *A*, PLASMA_desc * *B*, PLASMA_Complex32_t *beta*, PLASMA_desc * *C*)

PLASMA_csyr2k_Tile - Performs symmetric rank k update. Tile equivalent of PLASMA_csyr2k(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← ***trans*** Specifies whether the matrix A is transposed or conjfugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is conjfugate transposed.

← ***alpha*** alpha specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.

← ***beta*** beta specifies the scalar beta

↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_csyr2k_Tile
PLASMA_csyr2k
PLASMA_dsyr2k
PLASMA_ssyr2k

3.6.2.26 int PLASMA_csyrk_Tile (PLASMA_enum *uplo*, PLASMA_enum *trans*, PLASMA_Complex32_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_Complex32_t *beta*, PLASMA_desc ∗ *C*)

PLASMA_csyrk_Tile - Performs rank k update. Tile equivalent of PLASMA_csyrk(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← ***trans*** Specifies whether the matrix A is transposed or conjfugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is transposed.

← ***alpha*** alpha specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.

← ***beta*** beta specifies the scalar beta

↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_csyrk_Tile
PLASMA_csyrk
PLASMA_dsyrk
PLASMA_ssyrk

3.6.2.27 int PLASMA_ctrmm_Tile (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, PLASMA_Complex32_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*)

PLASMA_ctrmm_Tile - Computes triangular solve. Tile equivalent of PLASMA_ctrmm(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***side*** Specifies whether A appears on the left or on the right of X: = PlasmaLeft: A∗X = B = PlasmaRight: X∗A = B

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***transA*** Specifies whether the matrix A is transposed, not transposed or conjfugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaConjTrans: A is conjfugate transposed.

← ***diag*** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A us unit.

← ***A*** The triangular matrix A. If uplo = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If uplo = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If diag = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_ctrmm
PLASMA_ctrmm_Tile_Async
PLASMA_ctrmm_Tile
PLASMA_dtrmm_Tile
PLASMA_strmm_Tile

3.6.2.28 int PLASMA_ctrsm_Tile (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, PLASMA_Complex32_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*)

PLASMA_ctrsm_Tile - Computes triangular solve. Tile equivalent of PLASMA_ctrsm(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***side*** Specifies whether A appears on the left or on the right of X: = PlasmaLeft: A∗X = B = PlasmaRight: X∗A = B

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***transA*** Specifies whether the matrix A is transposed, not transposed or conjfugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaConjTrans: A is conjfugate transposed.

← ***diag*** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A us unit.

← ***A*** The triangular matrix A. If uplo = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If uplo = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If diag = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_ctrsm
PLASMA_ctrsm_Tile_Async
PLASMA_ctrsm_Tile
PLASMA_dtrsm_Tile
PLASMA_strsm_Tile

3.6.2.29 int PLASMA_ctrsmpl_Tile (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*, PLASMA_desc ∗ *B*)

PLASMA_ctrsmpl_Tile - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***A*** The tile factor L from the factorization, computed by PLASMA_cgetrf.

← ***L*** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_cgetrf.

← ***IPIV*** The pivot indices from PLASMA_cgetrf (not equivalent to LAPACK).

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_ctrsmpl
PLASMA_ctrsmpl_Tile_Async
PLASMA_ctrsmpl_Tile
PLASMA_dtrsmpl_Tile
PLASMA_strsmpl_Tile
PLASMA_cgetrf_Tile

3.6.2.30 int PLASMA_ctrtri_Tile (PLASMA_enum *uplo*, PLASMA_enum *diag*, PLASMA_desc ∗ *A*)

PLASMA_ctrtri_Tile - Computes the inverse of a complex upper or lower triangular matrix A. Tile equivalent of PLASMA_ctrtri(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***diag*** = PlasmaNonUnit: A is non-unit triangular; = PlasmaUnit: A us unit triangular.

← ***A*** On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1. On exit, the (triangular) inverse of the original matrix, in the same storage format.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

See also

PLASMA_ctrtri
PLASMA_ctrtri_Tile_Async
PLASMA_ctrtri_Tile
PLASMA_dtrtri_Tile
PLASMA_strtri_Tile
PLASMA_cpotri_Tile

3.6.2.31 int PLASMA_cunglq_Tile (PLASMA_desc $*$ *A*, PLASMA_desc $*$ *T*, PLASMA_desc $*$ *B*)

PLASMA_cunglq_Tile - Generates an M-by-N matrix Q with orthonormal rows, which is defined as the first M rows of a product of the elementary reflectors returned by PLASMA_cgelqf. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

$\leftarrow$ ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_cgelqf.

$\leftarrow$ ***T*** Auxiliary factorization data, computed by PLASMA_cgelqf.

$\rightarrow$ ***B*** On exit, the M-by-N matrix Q.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_cunglq
PLASMA_cunglq_Tile_Async
PLASMA_cunglq_Tile
PLASMA_dunglq_Tile
PLASMA_sunglq_Tile
PLASMA_cgelqf_Tile

3.6.2.32 int PLASMA_cungqr_Tile (PLASMA_desc $*$ *A*, PLASMA_desc $*$ *T*, PLASMA_desc $*$ *Q*)

PLASMA_cungqr_Tile - Generates an M-by-N matrix Q with orthonormal columns, which is defined as the first N columns of a product of the elementary reflectors returned by PLASMA_cgeqrf. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

$\leftarrow$ ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_cgeqrf.

$\leftarrow$ ***T*** Auxiliary factorization data, computed by PLASMA_cgeqrf.

$\rightarrow$ ***Q*** On exit, the M-by-N matrix Q.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_cungqr
PLASMA_cungqr_Tile_Async
PLASMA_cungqr_Tile
PLASMA_dungqr_Tile
PLASMA_sungqr_Tile
PLASMA_cgeqrf_Tile

3.6.2.33 int PLASMA_cunmlq_Tile (PLASMA_enum *side*, PLASMA_enum *trans*, PLASMA_desc * *A*, PLASMA_desc * *T*, PLASMA_desc * *B*)

PLASMA_cunmlq_Tile - overwrites the general M-by-N matrix C with Q*C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_-cgelqf_Tile Q is of order M. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***side*** Intended usage: = PlasmaLeft: apply Q or Q**H from the left; = PlasmaRight: apply Q or Q**H from the right. Currently only PlasmaLeft is supported.

← ***trans*** Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaConjTrans: conjfugate transpose, apply Q**H. Currently only PlasmaConjTrans is supported.

← ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_cgelqf.

← ***T*** Auxiliary factorization data, computed by PLASMA_cgelqf.

↔ ***B*** On entry, the M-by-N matrix B. On exit, B is overwritten by Q*B or Q**H*B.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_cunmlq
PLASMA_cunmlq_Tile_Async
PLASMA_cunmlq_Tile
PLASMA_dunmlq_Tile
PLASMA_sunmlq_Tile
PLASMA_cgelqf_Tile

3.6.2.34 int PLASMA_cunmqr_Tile (PLASMA_enum *side*, PLASMA_enum *trans*, PLASMA_desc * *A*, PLASMA_desc * *T*, PLASMA_desc * *B*)

PLASMA_cunmqr_Tile - overwrites the general M-by-N matrix C with Q*C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_-cgeqrf_Tile Q is of order M. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***side*** Intended usage: = PlasmaLeft: apply Q or Q**H from the left; = PlasmaRight: apply Q or Q**H from the right. Currently only PlasmaLeft is supported.

← ***trans*** Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaConjTrans: conjfugate transpose, apply Q**H. Currently only PlasmaConjTrans is supported.

← ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_cgeqrf.

← ***T*** Auxiliary factorization data, computed by PLASMA_cgeqrf.

↔ ***B*** On entry, the M-by-N matrix B. On exit, B is overwritten by Q*B or Q**H*B.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_cunmqr
PLASMA_cunmqr_Tile_Async
PLASMA_cunmqr_Tile
PLASMA_dunmqr_Tile
PLASMA_sunmqr_Tile
PLASMA_cgeqrf_Tile

3.7 Advanced Interface: Synchronous - Double Real

Functions/Subroutines

- int PLASMA_dgelqf_Tile (PLASMA_desc ∗A, PLASMA_desc ∗T)
- int PLASMA_dgelqs_Tile (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B)
- int PLASMA_dgels_Tile (PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B)
- int PLASMA_dgemm_Tile (PLASMA_enum transA, PLASMA_enum transB, double alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, double beta, PLASMA_desc ∗C)
- int PLASMA_dgeqrf_Tile (PLASMA_desc ∗A, PLASMA_desc ∗T)
- int PLASMA_dgeqrs_Tile (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B)
- int PLASMA_dgesv_Tile (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_desc ∗B)
- int PLASMA_dgetrf_Tile (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV)
- int PLASMA_dgetrs_Tile (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_desc ∗B)
- double PLASMA_dlange_Tile (PLASMA_enum norm, PLASMA_desc ∗A, double ∗work)
- double PLASMA_dlansy_Tile (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc ∗A, double ∗work)
- int PLASMA_dlauum_Tile (PLASMA_enum uplo, PLASMA_desc ∗A)
- int PLASMA_dorglq_Tile (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B)
- int PLASMA_dorgqr_Tile (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗Q)
- int PLASMA_dormlq_Tile (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B)
- int PLASMA_dormqr_Tile (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B)
- int PLASMA_dplgsy_Tile (double bump, PLASMA_desc ∗A, unsigned long long int seed)
- int PLASMA_dplrnt_Tile (PLASMA_desc ∗A, unsigned long long int seed)
- int PLASMA_dposv_Tile (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_desc ∗B)
- int PLASMA_dpotrf_Tile (PLASMA_enum uplo, PLASMA_desc ∗A)
- int PLASMA_dpotri_Tile (PLASMA_enum uplo, PLASMA_desc ∗A)
- int PLASMA_dpotrs_Tile (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_desc ∗B)
- int PLASMA_dsgesv_Tile (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_desc ∗B, PLASMA_desc ∗X, int ∗ITER)
- int PLASMA_dsposv_Tile (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_desc ∗X, int ∗ITER)
- int PLASMA_dsungesv_Tile (PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_desc ∗X, int ∗ITER)
- int PLASMA_dsymm_Tile (PLASMA_enum side, PLASMA_enum uplo, double alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, double beta, PLASMA_desc ∗C)
- int PLASMA_dsyr2k_Tile (PLASMA_enum uplo, PLASMA_enum trans, double alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, double beta, PLASMA_desc ∗C)
- int PLASMA_dsyrk_Tile (PLASMA_enum uplo, PLASMA_enum trans, double alpha, PLASMA_-desc ∗A, double beta, PLASMA_desc ∗C)
- int PLASMA_dtrmm_Tile (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, double alpha, PLASMA_desc ∗A, PLASMA_desc ∗B)
- int PLASMA_dtrsm_Tile (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, double alpha, PLASMA_desc ∗A, PLASMA_desc ∗B)
- int PLASMA_dtrsmpl_Tile (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_desc ∗B)
- int PLASMA_dtrtri_Tile (PLASMA_enum uplo, PLASMA_enum diag, PLASMA_desc ∗A)

3.7.1 Detailed Description

This is the group of double real functions using the advanced synchronous interface.

3.7.2 Function/Subroutine Documentation

3.7.2.1 int PLASMA_dgelqf_Tile (PLASMA_desc $*$ *A*, PLASMA_desc $*$ *T*)

PLASMA_dgelqf_Tile - Computes the tile LQ factorization of a matrix. Tile equivalent of PLASMA_dgelqf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

$\leftrightarrow$ ***A*** On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the m-by-min(M,N) lower trapezoidal matrix L (L is lower triangular if M <= N); the elements above the diagonal represent the unitary matrix Q as a product of elementary reflectors, stored by tiles.

$\rightarrow$ ***T*** On exit, auxiliary factorization data, required by PLASMA_dgelqs to solve the system of equations.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dgelqf
PLASMA_dgelqf_Tile_Async
PLASMA_cgelqf_Tile
PLASMA_dgelqf_Tile
PLASMA_sgelqf_Tile
PLASMA_dgelqs_Tile

3.7.2.2 int PLASMA_dgelqs_Tile (PLASMA_desc $*$ *A*, PLASMA_desc $*$ *T*, PLASMA_desc $*$ *B*)

PLASMA_dgelqs_Tile - Computes a minimum-norm solution using previously computed LQ factorization. Tile equivalent of PLASMA_dgelqs(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

$\leftarrow$ ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_dgelqf.

$\leftarrow$ ***T*** Auxiliary factorization data, computed by PLASMA_dgelqf.

$\leftrightarrow$ ***B*** On entry, the M-by-NRHS right hand side matrix B. On exit, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dgelqs
PLASMA_dgelqs_Tile_Async
PLASMA_cgelqs_Tile
PLASMA_dgelqs_Tile
PLASMA_sgelqs_Tile
PLASMA_dgelqf_Tile

3.7.2.3 int PLASMA_dgels_Tile (PLASMA_enum *trans*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*)

PLASMA_dgels_Tile - Solves overdetermined or underdetermined linear system of equations using the tile QR or the tile LQ factorization. Tile equivalent of PLASMA_dgels(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***trans*** Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaTrans: the linear system involves A\∗\∗T. Currently only PlasmaNoTrans is supported.

↔ ***A*** On entry, the M-by-N matrix A. On exit, if M >= N, A is overwritten by details of its QR factorization as returned by PLASMA_dgeqrf; if M < N, A is overwritten by details of its LQ factorization as returned by PLASMA_dgelqf.

→ ***T*** On exit, auxiliary factorization data.

↔ ***B*** On entry, the M-by-NRHS matrix B of right hand side vectors, stored columnwise; On exit, if return value = 0, B is overwritten by the solution vectors, stored columnwise: if M >= N, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if M < N, rows 1 to N of B contain the minimum norm solution vectors;

Returns

PLASMA_SUCCESS successful exit

See also

PLASMA_dgels
PLASMA_dgels_Tile_Async
PLASMA_cgels_Tile
PLASMA_dgels_Tile
PLASMA_sgels_Tile

3.7.2.4 int PLASMA_dgemm_Tile (PLASMA_enum *transA*, PLASMA_enum *transB*, double *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, double *beta*, PLASMA_desc ∗ *C*)

PLASMA_dgemm_Tile - Performs matrix multiplication. Tile equivalent of PLASMA_dgemm(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***transA*** Specifies whether the matrix A is transposed, not transposed or ugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is transposed; = PlasmaTrans: A is ugate transposed.

← ***transB*** Specifies whether the matrix B is transposed, not transposed or ugate transposed: = PlasmaNoTrans: B is not transposed; = PlasmaTrans: B is transposed; = PlasmaTrans: B is ugate transposed.

← ***alpha*** alpha specifies the scalar alpha

← ***A*** A is a LDA-by-ka matrix, where ka is K when transA = PlasmaNoTrans, and is M otherwise.

← ***B*** B is a LDB-by-kb matrix, where kb is N when transB = PlasmaNoTrans, and is K otherwise.

← ***beta*** beta specifies the scalar beta

↔ ***C*** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N matrix (alpha∗op(A)∗op(B) + beta∗C)

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dgemm
PLASMA_dgemm_Tile_Async
PLASMA_cgemm_Tile
PLASMA_dgemm_Tile
PLASMA_sgemm_Tile

3.7.2.5 int PLASMA_dgeqrf_Tile (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*)

PLASMA_dgeqrf_Tile - Computes the tile QR factorization of a matrix. Tile equivalent of PLASMA_dgeqrf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

↔ ***A*** 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 represent the unitary matrix Q as a product of elementary reflectors stored by tiles.

→ ***T*** On exit, auxiliary factorization data, required by PLASMA_dgeqrs to solve the system of equations.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dgeqrf
PLASMA_dgeqrf_Tile_Async
PLASMA_cgeqrf_Tile
PLASMA_dgeqrf_Tile
PLASMA_sgeqrf_Tile
PLASMA_dgeqrs_Tile

3.7.2.6 int PLASMA_dgeqrs_Tile (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*)

PLASMA_dgeqrs_Tile - Computes a minimum-norm solution using the tile QR factorization. Tile equivalent of PLASMA_dgetrf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

↔ ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_dgeqrf.

← ***T*** Auxiliary factorization data, computed by PLASMA_dgeqrf.

↔ ***B*** On entry, the m-by-nrhs right hand side matrix B. On exit, the n-by-nrhs solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dgeqrs
PLASMA_dgeqrs_Tile_Async
PLASMA_cgeqrs_Tile
PLASMA_dgeqrs_Tile
PLASMA_sgeqrs_Tile
PLASMA_dgeqrf_Tile

3.7.2.7 int PLASMA_dgesv_Tile (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*, PLASMA_desc ∗ *B*)

PLASMA_dgesv_Tile - Solves a system of linear equations using the tile LU factorization. Tile equivalent of PLASMA_dgetrf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

↔ ***A*** On entry, the N-by-N coefficient matrix A. On exit, the tile L and U factors from the factorization (not equivalent to LAPACK).

↔ ***L*** On exit, auxiliary factorization data, related to the tile L factor, necessary to solve the system of equations.

→ ***IPIV*** On exit, the pivot indices that define the permutations (not equivalent to LAPACK).

↔ ***B*** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

$>$***0*** if i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

PLASMA_dgesv
PLASMA_dgesv_Tile_Async
PLASMA_cgesv_Tile
PLASMA_dgesv_Tile
PLASMA_sgesv_Tile
PLASMA_dcgesv_Tile

3.7.2.8 int PLASMA_dgetrf_Tile (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*)

PLASMA_dgetrf_Tile - Computes the tile LU factorization of a matrix. Tile equivalent of PLASMA_dgetrf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

↔ ***A*** On entry, the M-by-N matrix to be factored. On exit, the tile factors L and U from the factorization.

→ ***L*** On exit, auxiliary factorization data, related to the tile L factor, required by PLASMA_dgetrs to solve the system of equations.

→ ***IPIV*** The pivot indices that define the permutations (not equivalent to LAPACK).

Returns

Return values

PLASMA_SUCCESS successful exit

$>$***0*** if 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.

See also

PLASMA_dgetrf
PLASMA_dgetrf_Tile_Async
PLASMA_cgetrf_Tile
PLASMA_dgetrf_Tile
PLASMA_sgetrf_Tile
PLASMA_dgetrs_Tile

3.7.2.9 int PLASMA_dgetrs_Tile (PLASMA_desc $*$ *A*, PLASMA_desc $*$ *L*, int $*$ *IPIV*, PLASMA_desc $*$ *B*)

PLASMA_dgetrs_Tile - Solves a system of linear equations using previously computed LU factorization. Tile equivalent of PLASMA_dgetrs(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

$\leftarrow$ ***A*** The tile factors L and U from the factorization, computed by PLASMA_dgetrf.

$\leftarrow$ ***L*** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_dgetrf.

$\leftarrow$ ***IPIV*** The pivot indices from PLASMA_dgetrf (not equivalent to LAPACK).

$\leftrightarrow$ ***B*** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, the solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dgetrs
PLASMA_dgetrs_Tile_Async
PLASMA_cgetrs_Tile
PLASMA_dgetrs_Tile
PLASMA_sgetrs_Tile
PLASMA_dgetrf_Tile

3.7.2.10 double PLASMA_dlange_Tile (PLASMA_enum *norm*, PLASMA_desc $*$ *A*, double $*$ *work*)

PLASMA_dlange_Tile - Tile equivalent of PLASMA_dlange(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

$\leftarrow$ ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

$\leftarrow$ ***A*** On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U $*$ U'; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L' $*$ L.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dlange

PLASMA_dlange_Tile_Async
PLASMA_clange_Tile
PLASMA_dlange_Tile
PLASMA_slange_Tile

3.7.2.11 double PLASMA_dlansy_Tile (PLASMA_enum *norm*, PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, double ∗ *work*)

PLASMA_dlansy_Tile - Tile equivalent of PLASMA_dlansy(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U ∗ U'; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L' ∗ L.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dlansy
PLASMA_dlansy_Tile_Async
PLASMA_clansy_Tile
PLASMA_dlansy_Tile
PLASMA_slansy_Tile

3.7.2.12 int PLASMA_dlauum_Tile (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*)

PLASMA_dlauum_Tile - Computes the product U ∗ U' or L' ∗ L, where the triangular factor U or L is stored in the upper or lower triangular part of the array A. Tile equivalent of PLASMA_dlauum(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U ∗ U'; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L' ∗ L.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dlauum
PLASMA_dlauum_Tile_Async
PLASMA_clauum_Tile
PLASMA_dlauum_Tile
PLASMA_slauum_Tile
PLASMA_dpotri_Tile

3.7.2.13 int PLASMA_dorglq_Tile (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*)

PLASMA_dorglq_Tile - Generates an M-by-N matrix Q with orthonormal rows, which is defined as the first M rows of a product of the elementary reflectors returned by PLASMA_dgelqf. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_dgelqf.

← ***T*** Auxiliary factorization data, computed by PLASMA_dgelqf.

→ ***B*** On exit, the M-by-N matrix Q.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dorglq
PLASMA_dorglq_Tile_Async
PLASMA_cunglq_Tile
PLASMA_dunglq_Tile
PLASMA_sunglq_Tile
PLASMA_dgelqf_Tile

3.7.2.14 int PLASMA_dorgqr_Tile (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *Q*)

PLASMA_dorgqr_Tile - Generates an M-by-N matrix Q with orthonormal columns, which is defined as the first N columns of a product of the elementary reflectors returned by PLASMA_dgeqrf. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_dgeqrf.

← ***T*** Auxiliary factorization data, computed by PLASMA_dgeqrf.

→ ***Q*** On exit, the M-by-N matrix Q.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dorgqr
PLASMA_dorgqr_Tile_Async
PLASMA_cungqr_Tile
PLASMA_dungqr_Tile
PLASMA_sungqr_Tile
PLASMA_dgeqrf_Tile

3.7.2.15 int PLASMA_dormlq_Tile (PLASMA_enum *side*, PLASMA_enum *trans*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*)

PLASMA_dormlq_Tile - overwrites the general M-by-N matrix C with Q∗C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_-dgelqf_Tile Q is of order M. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***side*** Intended usage: = PlasmaLeft: apply Q or Q\∗\∗T from the left; = PlasmaRight: apply Q or Q\∗\∗T from the right. Currently only PlasmaLeft is supported.

← ***trans*** Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaTrans: ugate transpose, apply Q\∗\∗T. Currently only PlasmaTrans is supported.

← ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_dgelqf.

← ***T*** Auxiliary factorization data, computed by PLASMA_dgelqf.

↔ ***B*** On entry, the M-by-N matrix B. On exit, B is overwritten by Q∗B or Q\∗\∗T∗B.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dormlq
PLASMA_dormlq_Tile_Async
PLASMA_cunmlq_Tile
PLASMA_dunmlq_Tile
PLASMA_sunmlq_Tile
PLASMA_dgelqf_Tile

3.7.2.16 int PLASMA_dormqr_Tile (PLASMA_enum *side*, PLASMA_enum *trans*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*)

PLASMA_dormqr_Tile - overwrites the general M-by-N matrix C with Q∗C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_-dgeqrf_Tile Q is of order M. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***side*** Intended usage: = PlasmaLeft: apply Q or Q**T from the left; = PlasmaRight: apply Q or Q**T from the right. Currently only PlasmaLeft is supported.

← ***trans*** Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaTrans: ugate transpose, apply Q**T. Currently only PlasmaTrans is supported.

← ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_dgeqrf.

← ***T*** Auxiliary factorization data, computed by PLASMA_dgeqrf.

↔ ***B*** On entry, the M-by-N matrix B. On exit, B is overwritten by Q∗B or Q**T∗B.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dormqr
PLASMA_dormqr_Tile_Async
PLASMA_cunmqr_Tile
PLASMA_dunmqr_Tile
PLASMA_sunmqr_Tile
PLASMA_dgeqrf_Tile

3.7.2.17 int PLASMA_dplgsy_Tile (double *bump*, PLASMA_desc ∗ *A*, unsigned long long int *seed*)

PLASMA_dplgsy_Tile - Generate a random hermitian matrix by tiles. Tile equivalent of PLASMA_dplgsy(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***A*** On exit, The random hermitian matrix A generated.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dplgsy
PLASMA_dplgsy_Tile_Async
PLASMA_cplgsy_Tile
PLASMA_dplgsy_Tile
PLASMA_splgsy_Tile
PLASMA_dplrnt_Tile
PLASMA_dplgsy_Tile

3.7.2.18 int PLASMA_dplrnt_Tile (PLASMA_desc ∗ *A*, unsigned long long int *seed*)

PLASMA_dplrnt_Tile - Generate a random matrix by tiles. Tile equivalent of PLASMA_dplrnt(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***A*** On exit, The random matrix A generated.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dplrnt
PLASMA_dplrnt_Tile_Async
PLASMA_cplrnt_Tile
PLASMA_dplrnt_Tile
PLASMA_splrnt_Tile
PLASMA_dplgsy_Tile
PLASMA_dplgsy_Tile

3.7.2.19 int PLASMA_dposv_Tile (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*)

PLASMA_dposv_Tile - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization. Tile equivalent of PLASMA_dposv(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

↔ ***A*** On entry, the symmetric positive definite (or Hermitian) matrix A. If uplo = PlasmaUpper, 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 return value = 0, the factor U or L from the Cholesky factorization A = U\∗\∗T∗U or A = L∗L\∗\∗T.

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_dposv
PLASMA_dposv_Tile_Async
PLASMA_cposv_Tile
PLASMA_dposv_Tile
PLASMA_sposv_Tile

3.7.2.20 int PLASMA_dpotrf_Tile (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*)

PLASMA_dpotrf_Tile - Computes the Cholesky factorization of a symmetric positive definite or Hermitian positive definite matrix. Tile equivalent of PLASMA_dpotrf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** On entry, the symmetric positive definite (or Hermitian) matrix A. If uplo = PlasmaUpper, 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 return value = 0, the factor U or L from the Cholesky factorization A = U\∗\∗T∗U or A = L∗L\∗\∗T.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_dpotrf
PLASMA_dpotrf_Tile_Async
PLASMA_cpotrf_Tile
PLASMA_dpotrf_Tile
PLASMA_spotrf_Tile
PLASMA_dpotrs_Tile

3.7.2.21 int PLASMA_dpotri_Tile (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*)

PLASMA_dpotri_Tile - Computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization A = U\∗\∗T∗U or A = L∗L\∗\∗T computed by PLASMA_dpotrf. Tile equivalent of PLASMA_dpotri(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** On entry, the triangular factor U or L from the Cholesky factorization A = U**T*U or A = L*L**T, as computed by PLASMA_dpotrf. On exit, the upper or lower triangle of the (Hermitian) inverse of A, overwriting the input factor U or L.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_dpotri
PLASMA_dpotri_Tile_Async
PLASMA_cpotri_Tile
PLASMA_dpotri_Tile
PLASMA_spotri_Tile
PLASMA_dpotrf_Tile

3.7.2.22 int PLASMA_dpotrs_Tile (PLASMA_enum *uplo*, PLASMA_desc * *A*, PLASMA_desc * *B*)

PLASMA_dpotrs_Tile - Solves a system of linear equations using previously computed Cholesky factorization. Tile equivalent of PLASMA_dpotrs(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** The triangular factor U or L from the Cholesky factorization A = U**T*U or A = L*L**T, computed by PLASMA_dpotrf.

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dpotrs
PLASMA_dpotrs_Tile_Async
PLASMA_cpotrs_Tile
PLASMA_dpotrs_Tile
PLASMA_spotrs_Tile
PLASMA_dpotrf_Tile

3.7.2.23 int PLASMA_dsgesv_Tile (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*, PLASMA_desc ∗ *B*, PLASMA_desc ∗ *X*, int ∗ *ITER*)

PLASMA_dsgesv_Tile - Solves a system of linear equations using the tile LU factorization and mixed-precision iterative refinement. Tile equivalent of PLASMA_dsgesv(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

↔ ***A*** On entry, the N-by-N coefficient matrix A.

- If the iterative refinement converged, A is not modified;
- otherwise, it fell back to double precision solution, and then A contains the tile L and U factors from the factorization (not equivalent to LAPACK).

→ ***L*** On exit:

- if the iterative refinement converged, L is not modified;
- otherwise, it fell back to double precision solution, and then L is an auxiliary factorization data, related to the tile L factor, necessary to solve the system of equations (not equivalent to LAPACK).

→ ***IPIV*** On exit, the pivot indices that define the permutations (not equivalent to LAPACK).

↔ ***B*** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

>***0*** if i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

PLASMA_dsgesv
PLASMA_dsgesv_Tile_Async
PLASMA_dsgesv_Tile
PLASMA_dgesv_Tile

3.7.2.24 int PLASMA_dsposv_Tile (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_desc ∗ *X*, int ∗ *ITER*)

PLASMA_dsposv_Tile - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization and mixed-precision iterative refinement. Tile equivalent of PLASMA_dsposv(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

↔ ***A*** On entry, the N-by-N symmetric positive definite (or Hermitian) coefficient matrix A. If uplo = PlasmaUpper, 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.

- If the iterative refinement converged, A is not modified;
- otherwise, it falled backed to double precision solution,

↔ ***B*** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

$>$***0*** if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_dsposv
PLASMA_dsposv_Tile_Async
PLASMA_dsposv_Tile
PLASMA_zposv_Tile

3.7.2.25 int PLASMA_dsungesv_Tile (PLASMA_enum *trans*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_desc ∗ *X*, int ∗ *ITER*)

PLASMA_dsungesv_Tile - Solves symmetric linear system of equations using the tile QR or the tile LQ factorization and mixed-precision iterative refinement. Tile equivalent of PLASMA_dsungesv(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***trans*** Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaTrans: the linear system involves A∗∗H. Currently only PlasmaNoTrans is supported.

↔ ***A***
- If the iterative refinement converged, A is not modified;
- otherwise, it fell back to double precision solution, and on exit the M-by-N matrix A contains: if M >= N, A is overwritten by details of its QR factorization as returned by PLASMA_zgeqrf; if M < N, A is overwritten by details of its LQ factorization as returned by PLASMA_zgelqf.

→ ***T*** On exit:

- if the iterative refinement converged, T is not modified;
- otherwise, it fell back to double precision solution, and then T is an auxiliary factorization data.

↔ ***B*** On entry, the M-by-NRHS matrix B of right hand side vectors, stored columnwise; On exit, if return value = 0, B is overwritten by the solution vectors, stored columnwise: if M >= N, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if M < N, rows 1 to N of B contain the minimum norm solution vectors;

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dsungesv
PLASMA_dsungesv_Tile_Async
PLASMA_dsungesv_Tile
PLASMA_zgels_Tile

3.7.2.26 int PLASMA_dsymm_Tile (PLASMA_enum *side*, PLASMA_enum *uplo*, double *alpha*, PLASMA_desc $*$ *A*, PLASMA_desc $*$ *B*, double *beta*, PLASMA_desc $*$ *C*)

PLASMA_dsymm_Tile - Performs symmetric matrix multiplication. Tile equivalent of PLASMA_dsymm(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

$\leftarrow$ ***side*** Specifies whether the symmetric matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

$\leftarrow$ ***uplo*** Specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the symmetric matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the symmetric matrix A is to be referenced.

$\leftarrow$ ***alpha*** Specifies the scalar alpha.

$\leftarrow$ ***A*** A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.

$\leftarrow$ ***B*** B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.

$\leftarrow$ ***beta*** Specifies the scalar beta.

$\leftrightarrow$ ***C*** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dsymm
PLASMA_dsymm_Tile_Async
PLASMA_csymm_Tile
PLASMA_dsymm_Tile
PLASMA_ssymm_Tile

3.7.2.27 int PLASMA_dsyr2k_Tile (PLASMA_enum *uplo*, PLASMA_enum *trans*, double *alpha*, PLASMA_desc * *A*, PLASMA_desc * *B*, double *beta*, PLASMA_desc * *C*)

PLASMA_dsyr2k_Tile - Performs symmetric rank k update. Tile equivalent of PLASMA_dsyr2k(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.
- ← ***trans*** Specifies whether the matrix A is transposed or ugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is ugate transposed.
- ← ***alpha*** alpha specifies the scalar alpha.
- ← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.
- ← ***beta*** beta specifies the scalar beta
- ↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dsyr2k_Tile
PLASMA_csyr2k
PLASMA_dsyr2k
PLASMA_ssyr2k

3.7.2.28 int PLASMA_dsyrk_Tile (PLASMA_enum *uplo*, PLASMA_enum *trans*, double *alpha*, PLASMA_desc * *A*, double *beta*, PLASMA_desc * *C*)

PLASMA_dsyrk_Tile - Performs rank k update. Tile equivalent of PLASMA_dsyrk(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.
- ← ***trans*** Specifies whether the matrix A is transposed or ugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is transposed.
- ← ***alpha*** alpha specifies the scalar alpha.
- ← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.
- ← ***beta*** beta specifies the scalar beta
- ↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dsyrk_Tile
PLASMA_csyrk
PLASMA_dsyrk
PLASMA_ssyrk

3.7.2.29 int PLASMA_dtrmm_Tile (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, double *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*)

PLASMA_dtrmm_Tile - Computes triangular solve. Tile equivalent of PLASMA_dtrmm(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***side*** Specifies whether A appears on the left or on the right of X: = PlasmaLeft: A∗X = B = PlasmaRight: X∗A = B

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***transA*** Specifies whether the matrix A is transposed, not transposed or ugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaTrans: A is ugate transposed.

← ***diag*** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A us unit.

← ***A*** The triangular matrix A. If uplo = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If uplo = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If diag = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dtrmm

PLASMA_dtrmm_Tile_Async
PLASMA_ctrmm_Tile
PLASMA_dtrmm_Tile
PLASMA_strmm_Tile

3.7.2.30 int PLASMA_dtrsm_Tile (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, double *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*)

PLASMA_dtrsm_Tile - Computes triangular solve. Tile equivalent of PLASMA_dtrsm(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***side*** Specifies whether A appears on the left or on the right of X: = PlasmaLeft: A∗X = B = PlasmaRight: X∗A = B

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***transA*** Specifies whether the matrix A is transposed, not transposed or ugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaTrans: A is ugate transposed.

← ***diag*** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A us unit.

← ***A*** The triangular matrix A. If uplo = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If uplo = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If diag = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dtrsm
PLASMA_dtrsm_Tile_Async
PLASMA_ctrsm_Tile
PLASMA_dtrsm_Tile
PLASMA_strsm_Tile

3.7.2.31 int PLASMA_dtrsmpl_Tile (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*, PLASMA_desc ∗ *B*)

PLASMA_dtrsmpl_Tile - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***A*** The tile factor L from the factorization, computed by PLASMA_dgetrf.

← ***L*** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_dgetrf.

← ***IPIV*** The pivot indices from PLASMA_dgetrf (not equivalent to LAPACK).

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_dtrsmpl
PLASMA_dtrsmpl_Tile_Async
PLASMA_ctrsmpl_Tile
PLASMA_dtrsmpl_Tile
PLASMA_strsmpl_Tile
PLASMA_dgetrf_Tile

3.7.2.32 int PLASMA_dtrtri_Tile (PLASMA_enum *uplo*, PLASMA_enum *diag*, PLASMA_desc ∗ *A*)

PLASMA_dtrtri_Tile - Computes the inverse of a complex upper or lower triangular matrix A. Tile equivalent of PLASMA_dtrtri(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***diag*** = PlasmaNonUnit: A is non-unit triangular; = PlasmaUnit: A us unit triangular.

← ***A*** On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1. On exit, the (triangular) inverse of the original matrix, in the same storage format.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

See also

PLASMA_dtrtri

PLASMA_dtrtri_Tile_Async
PLASMA_ctrtri_Tile
PLASMA_dtrtri_Tile
PLASMA_strtri_Tile
PLASMA_dpotri_Tile

3.8 Advanced Interface: Synchronous - Single Real

Functions/Subroutines

- int PLASMA_sgelqf_Tile (PLASMA_desc ∗A, PLASMA_desc ∗T)
- int PLASMA_sgelqs_Tile (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B)
- int PLASMA_sgels_Tile (PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B)
- int PLASMA_sgemm_Tile (PLASMA_enum transA, PLASMA_enum transB, float alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, float beta, PLASMA_desc ∗C)
- int PLASMA_sgeqrf_Tile (PLASMA_desc ∗A, PLASMA_desc ∗T)
- int PLASMA_sgeqrs_Tile (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B)
- int PLASMA_sgesv_Tile (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_desc ∗B)
- int PLASMA_sgetrf_Tile (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV)
- int PLASMA_sgetrs_Tile (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_desc ∗B)
- float PLASMA_slange_Tile (PLASMA_enum norm, PLASMA_desc ∗A, float ∗work)
- float PLASMA_slansy_Tile (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc ∗A, float ∗work)
- int PLASMA_slauum_Tile (PLASMA_enum uplo, PLASMA_desc ∗A)
- int PLASMA_sorglq_Tile (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B)
- int PLASMA_sorgqr_Tile (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗Q)
- int PLASMA_sormlq_Tile (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B)
- int PLASMA_sormqr_Tile (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B)
- int PLASMA_splgsy_Tile (float bump, PLASMA_desc ∗A, unsigned long long int seed)
- int PLASMA_splrnt_Tile (PLASMA_desc ∗A, unsigned long long int seed)
- int PLASMA_sposv_Tile (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_desc ∗B)
- int PLASMA_spotrf_Tile (PLASMA_enum uplo, PLASMA_desc ∗A)
- int PLASMA_spotri_Tile (PLASMA_enum uplo, PLASMA_desc ∗A)
- int PLASMA_spotrs_Tile (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_desc ∗B)
- int PLASMA_ssymm_Tile (PLASMA_enum side, PLASMA_enum uplo, float alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, float beta, PLASMA_desc ∗C)
- int PLASMA_ssyr2k_Tile (PLASMA_enum uplo, PLASMA_enum trans, float alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, float beta, PLASMA_desc ∗C)
- int PLASMA_ssyrk_Tile (PLASMA_enum uplo, PLASMA_enum trans, float alpha, PLASMA_desc ∗A, float beta, PLASMA_desc ∗C)
- int PLASMA_strmm_Tile (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, float alpha, PLASMA_desc ∗A, PLASMA_desc ∗B)
- int PLASMA_strsm_Tile (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, float alpha, PLASMA_desc ∗A, PLASMA_desc ∗B)
- int PLASMA_strsmpl_Tile (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_desc ∗B)
- int PLASMA_strtri_Tile (PLASMA_enum uplo, PLASMA_enum diag, PLASMA_desc ∗A)

3.8.1 Detailed Description

This is the group of single real functions using the advanced synchronous interface.

3.8.2 Function/Subroutine Documentation

3.8.2.1 int PLASMA_sgelqf_Tile (PLASMA_desc $*$ *A*, PLASMA_desc $*$ *T*)

PLASMA_sgelqf_Tile - Computes the tile LQ factorization of a matrix. Tile equivalent of PLASMA_-sgelqf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

$\leftrightarrow$ ***A*** On entry, the M-by-N matrix A. On exit, the elements on and below the diagonal of the array contain the m-by-min(M,N) lower trapezoidal matrix L (L is lower triangular if M <= N); the elements above the diagonal represent the unitary matrix Q as a product of elementary reflectors, stored by tiles.

$\rightarrow$ ***T*** On exit, auxiliary factorization data, required by PLASMA_sgelqs to solve the system of equations.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_sgelqf
PLASMA_sgelqf_Tile_Async
PLASMA_cgelqf_Tile
PLASMA_dgelqf_Tile
PLASMA_sgelqf_Tile
PLASMA_sgelqs_Tile

3.8.2.2 int PLASMA_sgelqs_Tile (PLASMA_desc $*$ *A*, PLASMA_desc $*$ *T*, PLASMA_desc $*$ *B*)

PLASMA_sgelqs_Tile - Computes a minimum-norm solution using previously computed LQ factorization. Tile equivalent of PLASMA_sgelqs(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

$\leftarrow$ ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_sgelqf.

$\leftarrow$ ***T*** Auxiliary factorization data, computed by PLASMA_sgelqf.

$\leftrightarrow$ ***B*** On entry, the M-by-NRHS right hand side matrix B. On exit, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_sgelqs
PLASMA_sgelqs_Tile_Async
PLASMA_cgelqs_Tile
PLASMA_dgelqs_Tile
PLASMA_sgelqs_Tile
PLASMA_sgelqf_Tile

3.8.2.3 int PLASMA_sgels_Tile (PLASMA_enum *trans*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*)

PLASMA_sgels_Tile - Solves overdetermined or underdetermined linear system of equations using the tile QR or the tile LQ factorization. Tile equivalent of PLASMA_sgels(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← ***trans*** Intended usage: = PlasmaNoTrans: the linear system involves A; = PlasmaTrans: the linear system involves A\∗\∗T. Currently only PlasmaNoTrans is supported.
- ↔ ***A*** On entry, the M-by-N matrix A. On exit, if M >= N, A is overwritten by details of its QR factorization as returned by PLASMA_sgeqrf; if M < N, A is overwritten by details of its LQ factorization as returned by PLASMA_sgelqf.
- → ***T*** On exit, auxiliary factorization data.
- ↔ ***B*** On entry, the M-by-NRHS matrix B of right hand side vectors, stored columnwise; On exit, if return value = 0, B is overwritten by the solution vectors, stored columnwise: if M >= N, rows 1 to N of B contain the least squares solution vectors; the residual sum of squares for the solution in each column is given by the sum of squares of the modulus of elements N+1 to M in that column; if M < N, rows 1 to N of B contain the minimum norm solution vectors;

Returns

PLASMA_SUCCESS successful exit

See also

PLASMA_sgels
PLASMA_sgels_Tile_Async
PLASMA_cgels_Tile
PLASMA_dgels_Tile
PLASMA_sgels_Tile

3.8.2.4 int PLASMA_sgemm_Tile (PLASMA_enum *transA*, PLASMA_enum *transB*, float *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, float *beta*, PLASMA_desc ∗ *C*)

PLASMA_sgemm_Tile - Performs matrix multiplication. Tile equivalent of PLASMA_sgemm(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

- ← ***transA*** Specifies whether the matrix A is transposed, not transposed or ugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is transposed; = PlasmaTrans: A is ugate transposed.

← ***transB*** Specifies whether the matrix B is transposed, not transposed or ugate transposed: = PlasmaNoTrans: B is not transposed; = PlasmaTrans: B is transposed; = PlasmaTrans: B is ugate transposed.

← ***alpha*** alpha specifies the scalar alpha

← ***A*** A is a LDA-by-ka matrix, where ka is K when transA = PlasmaNoTrans, and is M otherwise.

← ***B*** B is a LDB-by-kb matrix, where kb is N when transB = PlasmaNoTrans, and is K otherwise.

← ***beta*** beta specifies the scalar beta

↔ ***C*** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N matrix (alpha∗op(A)∗op(B) + beta∗C)

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_sgemm
PLASMA_sgemm_Tile_Async
PLASMA_cgemm_Tile
PLASMA_dgemm_Tile
PLASMA_sgemm_Tile

3.8.2.5 int PLASMA_sgeqrf_Tile (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*)

PLASMA_sgeqrf_Tile - Computes the tile QR factorization of a matrix. Tile equivalent of PLASMA_sgeqrf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

↔ ***A*** 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 represent the unitary matrix Q as a product of elementary reflectors stored by tiles.

→ ***T*** On exit, auxiliary factorization data, required by PLASMA_sgeqrs to solve the system of equations.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_sgeqrf
PLASMA_sgeqrf_Tile_Async
PLASMA_cgeqrf_Tile
PLASMA_dgeqrf_Tile
PLASMA_sgeqrf_Tile
PLASMA_sgeqrs_Tile

3.8.2.6 int PLASMA_sgeqrs_Tile (PLASMA_desc * *A*, PLASMA_desc * *T*, PLASMA_desc * *B*)

PLASMA_sgeqrs_Tile - Computes a minimum-norm solution using the tile QR factorization. Tile equivalent of PLASMA_sgetrf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

↔ ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_sgeqrf.

← ***T*** Auxiliary factorization data, computed by PLASMA_sgeqrf.

↔ ***B*** On entry, the m-by-nrhs right hand side matrix B. On exit, the n-by-nrhs solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_sgeqrs
PLASMA_sgeqrs_Tile_Async
PLASMA_cgeqrs_Tile
PLASMA_dgeqrs_Tile
PLASMA_sgeqrs_Tile
PLASMA_sgeqrf_Tile

3.8.2.7 int PLASMA_sgesv_Tile (PLASMA_desc * *A*, PLASMA_desc * *L*, int * *IPIV*, PLASMA_desc * *B*)

PLASMA_sgesv_Tile - Solves a system of linear equations using the tile LU factorization. Tile equivalent of PLASMA_sgetrf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

↔ ***A*** On entry, the N-by-N coefficient matrix A. On exit, the tile L and U factors from the factorization (not equivalent to LAPACK).

↔ ***L*** On exit, auxiliary factorization data, related to the tile L factor, necessary to solve the system of equations.

→ ***IPIV*** On exit, the pivot indices that define the permutations (not equivalent to LAPACK).

↔ ***B*** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

See also

PLASMA_sgesv
PLASMA_sgesv_Tile_Async
PLASMA_cgesv_Tile
PLASMA_dgesv_Tile
PLASMA_sgesv_Tile
PLASMA_scgesv_Tile

3.8.2.8 int PLASMA_sgetrf_Tile (PLASMA_desc $*$ *A*, PLASMA_desc $*$ *L*, int $*$ *IPIV*)

PLASMA_sgetrf_Tile - Computes the tile LU factorization of a matrix. Tile equivalent of PLASMA_sgetrf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

$\leftrightarrow$ ***A*** On entry, the M-by-N matrix to be factored. On exit, the tile factors L and U from the factorization.

$\rightarrow$ ***L*** On exit, auxiliary factorization data, related to the tile L factor, required by PLASMA_sgetrs to solve the system of equations.

$\rightarrow$ ***IPIV*** The pivot indices that define the permutations (not equivalent to LAPACK).

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if 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.

See also

PLASMA_sgetrf
PLASMA_sgetrf_Tile_Async
PLASMA_cgetrf_Tile
PLASMA_dgetrf_Tile
PLASMA_sgetrf_Tile
PLASMA_sgetrs_Tile

3.8.2.9 int PLASMA_sgetrs_Tile (PLASMA_desc $*$ *A*, PLASMA_desc $*$ *L*, int $*$ *IPIV*, PLASMA_desc $*$ *B*)

PLASMA_sgetrs_Tile - Solves a system of linear equations using previously computed LU factorization. Tile equivalent of PLASMA_sgetrs(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***A*** The tile factors L and U from the factorization, computed by PLASMA_sgetrf.

← ***L*** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_sgetrf.

← ***IPIV*** The pivot indices from PLASMA_sgetrf (not equivalent to LAPACK).

↔ ***B*** On entry, the N-by-NRHS matrix of right hand side matrix B. On exit, the solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_sgetrs
PLASMA_sgetrs_Tile_Async
PLASMA_cgetrs_Tile
PLASMA_dgetrs_Tile
PLASMA_sgetrs_Tile
PLASMA_sgetrf_Tile

3.8.2.10 float PLASMA_slange_Tile (PLASMA_enum *norm*, PLASMA_desc ∗ *A*, float ∗ *work*)

PLASMA_slange_Tile - Tile equivalent of PLASMA_slange(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U ∗ U'; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L' ∗ L.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_slange
PLASMA_slange_Tile_Async
PLASMA_clange_Tile
PLASMA_dlange_Tile
PLASMA_slange_Tile

3.8.2.11 float PLASMA_slansy_Tile (PLASMA_enum *norm*, PLASMA_enum *uplo*, PLASMA_desc $*$ *A*, float $*$ *work*)

PLASMA_slansy_Tile - Tile equivalent of PLASMA_slansy(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

$\leftarrow$ ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

$\leftarrow$ ***A*** On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U $*$ U'; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L' $*$ L.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_slansy
PLASMA_slansy_Tile_Async
PLASMA_clansy_Tile
PLASMA_dlansy_Tile
PLASMA_slansy_Tile

3.8.2.12 int PLASMA_slauum_Tile (PLASMA_enum *uplo*, PLASMA_desc $*$ *A*)

PLASMA_slauum_Tile - Computes the product U $*$ U' or L' $*$ L, where the triangular factor U or L is stored in the upper or lower triangular part of the array A. Tile equivalent of PLASMA_slauum(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

$\leftarrow$ ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

$\leftarrow$ ***A*** On entry, the triangular factor U or L. On exit, if UPLO = 'U', the upper triangle of A is overwritten with the upper triangle of the product U $*$ U'; if UPLO = 'L', the lower triangle of A is overwritten with the lower triangle of the product L' $*$ L.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_slauum
PLASMA_slauum_Tile_Async

PLASMA_clauum_Tile
PLASMA_dlauum_Tile
PLASMA_slauum_Tile
PLASMA_spotri_Tile

3.8.2.13 int PLASMA_sorglq_Tile (PLASMA_desc $*$ *A*, PLASMA_desc $*$ *T*, PLASMA_desc $*$ *B*)

PLASMA_sorglq_Tile - Generates an M-by-N matrix Q with orthonormal rows, which is defined as the first M rows of a product of the elementary reflectors returned by PLASMA_sgelqf. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

$\leftarrow$ ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_sgelqf.

$\leftarrow$ ***T*** Auxiliary factorization data, computed by PLASMA_sgelqf.

$\rightarrow$ ***B*** On exit, the M-by-N matrix Q.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_sorglq
PLASMA_sorglq_Tile_Async
PLASMA_cunglq_Tile
PLASMA_dunglq_Tile
PLASMA_sunglq_Tile
PLASMA_sgelqf_Tile

3.8.2.14 int PLASMA_sorgqr_Tile (PLASMA_desc $*$ *A*, PLASMA_desc $*$ *T*, PLASMA_desc $*$ *Q*)

PLASMA_sorgqr_Tile - Generates an M-by-N matrix Q with orthonormal columns, which is defined as the first N columns of a product of the elementary reflectors returned by PLASMA_sgeqrf. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

$\leftarrow$ ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_sgeqrf.

$\leftarrow$ ***T*** Auxiliary factorization data, computed by PLASMA_sgeqrf.

$\rightarrow$ ***Q*** On exit, the M-by-N matrix Q.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_sorgqr
PLASMA_sorgqr_Tile_Async
PLASMA_cungqr_Tile
PLASMA_dungqr_Tile
PLASMA_sungqr_Tile
PLASMA_sgeqrf_Tile

3.8.2.15 int PLASMA_sormlq_Tile (PLASMA_enum *side*, PLASMA_enum *trans*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*)

PLASMA_sormlq_Tile - overwrites the general M-by-N matrix C with Q∗C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_-sgelqf_Tile Q is of order M. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***side*** Intended usage: = PlasmaLeft: apply Q or Q\∗\∗T from the left; = PlasmaRight: apply Q or Q\∗\∗T from the right. Currently only PlasmaLeft is supported.

← ***trans*** Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaTrans: ugate transpose, apply Q\∗\∗T. Currently only PlasmaTrans is supported.

← ***A*** Details of the LQ factorization of the original matrix A as returned by PLASMA_sgelqf.

← ***T*** Auxiliary factorization data, computed by PLASMA_sgelqf.

↔ ***B*** On entry, the M-by-N matrix B. On exit, B is overwritten by Q∗B or Q\∗\∗T∗B.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_sormlq
PLASMA_sormlq_Tile_Async
PLASMA_cunmlq_Tile
PLASMA_dunmlq_Tile
PLASMA_sunmlq_Tile
PLASMA_sgelqf_Tile

3.8.2.16 int PLASMA_sormqr_Tile (PLASMA_enum *side*, PLASMA_enum *trans*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*)

PLASMA_sormqr_Tile - overwrites the general M-by-N matrix C with Q∗C, where Q is an orthogonal matrix (unitary in the complex case) defined as the product of elementary reflectors returned by PLASMA_-sgeqrf_Tile Q is of order M. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***side*** Intended usage: = PlasmaLeft: apply Q or Q**T from the left; = PlasmaRight: apply Q or Q**T from the right. Currently only PlasmaLeft is supported.

← ***trans*** Intended usage: = PlasmaNoTrans: no transpose, apply Q; = PlasmaTrans: ugate transpose, apply Q**T. Currently only PlasmaTrans is supported.

← ***A*** Details of the QR factorization of the original matrix A as returned by PLASMA_sgeqrf.

← ***T*** Auxiliary factorization data, computed by PLASMA_sgeqrf.

↔ ***B*** On entry, the M-by-N matrix B. On exit, B is overwritten by Q*B or Q**T*B.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_sormqr
PLASMA_sormqr_Tile_Async
PLASMA_cunmqr_Tile
PLASMA_dunmqr_Tile
PLASMA_sunmqr_Tile
PLASMA_sgeqrf_Tile

3.8.2.17 int PLASMA_splgsy_Tile (float *bump*, PLASMA_desc * *A*, unsigned long long int *seed*)

PLASMA_splgsy_Tile - Generate a random hermitian matrix by tiles. Tile equivalent of PLASMA_-splgsy(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***A*** On exit, The random hermitian matrix A generated.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_splgsy
PLASMA_splgsy_Tile_Async
PLASMA_cplgsy_Tile
PLASMA_dplgsy_Tile
PLASMA_splgsy_Tile
PLASMA_splrnt_Tile
PLASMA_splgsy_Tile

3.8.2.18 int PLASMA_splrnt_Tile (PLASMA_desc * *A*, unsigned long long int *seed*)

PLASMA_splrnt_Tile - Generate a random matrix by tiles. Tile equivalent of PLASMA_splrnt(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***A*** On exit, The random matrix A generated.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_splrnt
PLASMA_splrnt_Tile_Async
PLASMA_cplrnt_Tile
PLASMA_dplrnt_Tile
PLASMA_splrnt_Tile
PLASMA_splgsy_Tile
PLASMA_splgsy_Tile

3.8.2.19 int PLASMA_sposv_Tile (PLASMA_enum *uplo*, PLASMA_desc * *A*, PLASMA_desc * *B*)

PLASMA_sposv_Tile - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization. Tile equivalent of PLASMA_sposv(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

↔ ***A*** On entry, the symmetric positive definite (or Hermitian) matrix A. If uplo = PlasmaUpper, 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 return value = 0, the factor U or L from the Cholesky factorization A = U**T*U or A = L*L**T.

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_sposv
PLASMA_sposv_Tile_Async
PLASMA_cposv_Tile
PLASMA_dposv_Tile
PLASMA_sposv_Tile

3.8.2.20 int PLASMA_spotrf_Tile (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*)

PLASMA_spotrf_Tile - Computes the Cholesky factorization of a symmetric positive definite or Hermitian positive definite matrix. Tile equivalent of PLASMA_spotrf(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** On entry, the symmetric positive definite (or Hermitian) matrix A. If uplo = PlasmaUpper, 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 return value = 0, the factor U or L from the Cholesky factorization A = U\∗\∗T∗U or A = L∗L\∗\∗T.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_spotrf
PLASMA_spotrf_Tile_Async
PLASMA_cpotrf_Tile
PLASMA_dpotrf_Tile
PLASMA_spotrf_Tile
PLASMA_spotrs_Tile

3.8.2.21 int PLASMA_spotri_Tile (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*)

PLASMA_spotri_Tile - Computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization A = U\∗\∗T∗U or A = L∗L\∗\∗T computed by PLASMA_spotrf. Tile equivalent of PLASMA_spotri(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** On entry, the triangular factor U or L from the Cholesky factorization A = U\∗\∗T∗U or A = L∗L\∗\∗T, as computed by PLASMA_spotrf. On exit, the upper or lower triangle of the (Hermitian) inverse of A, overwriting the input factor U or L.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

See also

PLASMA_spotri
PLASMA_spotri_Tile_Async
PLASMA_cpotri_Tile
PLASMA_dpotri_Tile
PLASMA_spotri_Tile
PLASMA_spotrf_Tile

3.8.2.22 int PLASMA_spotrs_Tile (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*)

PLASMA_spotrs_Tile - Solves a system of linear equations using previously computed Cholesky factorization. Tile equivalent of PLASMA_spotrs(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***A*** The triangular factor U or L from the Cholesky factorization A = U\∗\∗T∗U or A = L∗L\∗\∗T, computed by PLASMA_spotrf.

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_spotrs
PLASMA_spotrs_Tile_Async
PLASMA_cpotrs_Tile
PLASMA_dpotrs_Tile
PLASMA_spotrs_Tile
PLASMA_spotrf_Tile

3.8.2.23 int PLASMA_ssymm_Tile (PLASMA_enum *side*, PLASMA_enum *uplo*, float *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, float *beta*, PLASMA_desc ∗ *C*)

PLASMA_ssymm_Tile - Performs symmetric matrix multiplication. Tile equivalent of PLASMA_-ssymm(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***side*** Specifies whether the symmetric matrix A appears on the left or right in the operation as follows: = PlasmaLeft:

$$C = \alpha \times A \times B + \beta \times C$$

= PlasmaRight:

$$C = \alpha \times B \times A + \beta \times C$$

← ***uplo*** Specifies whether the upper or lower triangular part of the symmetric matrix A is to be referenced as follows: = PlasmaLower: Only the lower triangular part of the symmetric matrix A is to be referenced. = PlasmaUpper: Only the upper triangular part of the symmetric matrix A is to be referenced.

← ***alpha*** Specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is M when side = PlasmaLeft, and is N otherwise. Only the uplo triangular part is referenced.

← ***B*** B is a LDB-by-N matrix, where the leading M-by-N part of the array B must contain the matrix B.

← ***beta*** Specifies the scalar beta.

↔ ***C*** C is a LDC-by-N matrix. On exit, the array is overwritten by the M by N updated matrix.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_ssymm
PLASMA_ssymm_Tile_Async
PLASMA_csymm_Tile
PLASMA_dsymm_Tile
PLASMA_ssymm_Tile

3.8.2.24 int PLASMA_ssyr2k_Tile (PLASMA_enum *uplo*, PLASMA_enum *trans*, float *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, float *beta*, PLASMA_desc ∗ *C*)

PLASMA_ssyr2k_Tile - Performs symmetric rank k update. Tile equivalent of PLASMA_ssyr2k(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← ***trans*** Specifies whether the matrix A is transposed or ugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is ugate transposed.

← ***alpha*** alpha specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.

← ***beta*** beta specifies the scalar beta

↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_ssyr2k_Tile
PLASMA_csyr2k
PLASMA_dsyr2k
PLASMA_ssyr2k

3.8.2.25 int PLASMA_ssyrk_Tile (PLASMA_enum *uplo*, PLASMA_enum *trans*, float *alpha*, PLASMA_desc ∗ *A*, float *beta*, PLASMA_desc ∗ *C*)

PLASMA_ssyrk_Tile - Performs rank k update. Tile equivalent of PLASMA_ssyrk(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of C is stored; = PlasmaLower: Lower triangle of C is stored.

← ***trans*** Specifies whether the matrix A is transposed or ugate transposed: = PlasmaNoTrans: A is not transposed; = PlasmaTrans: A is transposed.

← ***alpha*** alpha specifies the scalar alpha.

← ***A*** A is a LDA-by-ka matrix, where ka is K when trans = PlasmaNoTrans, and is N otherwise.

← ***beta*** beta specifies the scalar beta

↔ ***C*** C is a LDC-by-N matrix. On exit, the array uplo part of the matrix is overwritten by the uplo part of the updated matrix.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_ssyrk_Tile
PLASMA_csyrk
PLASMA_dsyrk
PLASMA_ssyrk

3.8.2.26 int PLASMA_strmm_Tile (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, float *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*)

PLASMA_strmm_Tile - Computes triangular solve. Tile equivalent of PLASMA_strmm(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***side*** Specifies whether A appears on the left or on the right of X: = PlasmaLeft: A∗X = B = PlasmaRight: X∗A = B

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***transA*** Specifies whether the matrix A is transposed, not transposed or ugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaTrans: A is ugate transposed.

← ***diag*** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A us unit.

← ***A*** The triangular matrix A. If uplo = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If uplo = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If diag = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_strmm
PLASMA_strmm_Tile_Async
PLASMA_ctrmm_Tile
PLASMA_dtrmm_Tile
PLASMA_strmm_Tile

3.8.2.27 int PLASMA_strsm_Tile (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, float *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*)

PLASMA_strsm_Tile - Computes triangular solve. Tile equivalent of PLASMA_strsm(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***side*** Specifies whether A appears on the left or on the right of X: = PlasmaLeft: A∗X = B = PlasmaRight: X∗A = B

← ***uplo*** Specifies whether the matrix A is upper triangular or lower triangular: = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***transA*** Specifies whether the matrix A is transposed, not transposed or ugate transposed: = PlasmaNoTrans: A is transposed; = PlasmaTrans: A is not transposed; = PlasmaTrans: A is ugate transposed.

← ***diag*** Specifies whether or not A is unit triangular: = PlasmaNonUnit: A is non unit; = PlasmaUnit: A us unit.

← ***A*** The triangular matrix A. If uplo = PlasmaUpper, the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If uplo = PlasmaLower, the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If diag = PlasmaUnit, the diagonal elements of A are also not referenced and are assumed to be 1.

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_strsm
PLASMA_strsm_Tile_Async
PLASMA_ctrsm_Tile
PLASMA_dtrsm_Tile
PLASMA_strsm_Tile

3.8.2.28 int PLASMA_strsmpl_Tile (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*, PLASMA_desc ∗ *B*)

PLASMA_strsmpl_Tile - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***A*** The tile factor L from the factorization, computed by PLASMA_sgetrf.

← ***L*** Auxiliary factorization data, related to the tile L factor, computed by PLASMA_sgetrf.

← ***IPIV*** The pivot indices from PLASMA_sgetrf (not equivalent to LAPACK).

↔ ***B*** On entry, the N-by-NRHS right hand side matrix B. On exit, if return value = 0, the N-by-NRHS solution matrix X.

Returns

Return values

PLASMA_SUCCESS successful exit

See also

PLASMA_strsmpl
PLASMA_strsmpl_Tile_Async
PLASMA_ctrsmpl_Tile
PLASMA_dtrsmpl_Tile
PLASMA_strsmpl_Tile
PLASMA_sgetrf_Tile

3.8.2.29 int PLASMA_strtri_Tile (PLASMA_enum *uplo*, PLASMA_enum *diag*, PLASMA_desc * *A*)

PLASMA_strtri_Tile - Computes the inverse of a complex upper or lower triangular matrix A. Tile equivalent of PLASMA_strtri(). Operates on matrices stored by tiles. All matrices are passed through descriptors. All dimensions are taken from the descriptors.

Parameters

← ***uplo*** = PlasmaUpper: Upper triangle of A is stored; = PlasmaLower: Lower triangle of A is stored.

← ***diag*** = PlasmaNonUnit: A is non-unit triangular; = PlasmaUnit: A us unit triangular.

← ***A*** On entry, the triangular matrix A. If UPLO = 'U', the leading N-by-N upper triangular part of the array A contains the upper triangular matrix, and the strictly lower triangular part of A is not referenced. If UPLO = 'L', the leading N-by-N lower triangular part of the array A contains the lower triangular matrix, and the strictly upper triangular part of A is not referenced. If DIAG = 'U', the diagonal elements of A are also not referenced and are assumed to be 1. On exit, the (triangular) inverse of the original matrix, in the same storage format.

Returns

Return values

PLASMA_SUCCESS successful exit

>0 if i, A(i,i) is exactly zero. The triangular matrix is singular and its inverse can not be computed.

See also

PLASMA_strtri
PLASMA_strtri_Tile_Async
PLASMA_ctrtri_Tile
PLASMA_dtrtri_Tile
PLASMA_strtri_Tile
PLASMA_spotri_Tile

3.9 Advanced Interface: Asynchronous - Double Complex

Functions/Subroutines

- int PLASMA_zcgels_Tile_Async (PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_desc ∗X, int ∗ITER, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zcgesv_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_desc ∗B, PLASMA_desc ∗X, int ∗ITER, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zcposv_Tile_Async (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_desc ∗X, int ∗ITER, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zcungesv_Tile_Async (PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_desc ∗X, int ∗ITER, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zgelqf_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zgelqs_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zgels_Tile_Async (PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zgemm_Tile_Async (PLASMA_enum transA, PLASMA_enum transB, PLASMA_Complex64_t alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_Complex64_t beta, PLASMA_desc ∗C, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zgeqrf_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zgeqrs_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zgesv_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zgetrf_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zgetrs_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zhemm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex64_t alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_Complex64_t beta, PLASMA_desc ∗C, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zher2k_Tile_Async (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex64_t alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, double beta, PLASMA_desc ∗C, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zherk_Tile_Async (PLASMA_enum uplo, PLASMA_enum trans, double alpha, PLASMA_desc ∗A, double beta, PLASMA_desc ∗C, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zlange_Tile_Async (PLASMA_enum norm, PLASMA_desc ∗A, double ∗work, double ∗value, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zlanhe_Tile_Async (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc ∗A, double ∗work, double ∗value, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zlansy_Tile_Async (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc ∗A, double ∗work, double ∗value, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zlauum_Tile_Async (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_sequence ∗sequence, PLASMA_request ∗request)

- int PLASMA_zplghe_Tile_Async (double bump, PLASMA_desc ∗A, unsigned long long int seed, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zplgsy_Tile_Async (PLASMA_Complex64_t bump, PLASMA_desc ∗A, unsigned long long int seed, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zplrnt_Tile_Async (PLASMA_desc ∗A, unsigned long long int seed, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zposv_Tile_Async (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zpotrf_Tile_Async (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zpotri_Tile_Async (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zpotrs_Tile_Async (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zsymm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex64_t alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_Complex64_t beta, PLASMA_desc ∗C, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zsyr2k_Tile_Async (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex64_t alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_Complex64_t beta, PLASMA_desc ∗C, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zsyrk_Tile_Async (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex64_t alpha, PLASMA_desc ∗A, PLASMA_Complex64_t beta, PLASMA_desc ∗C, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_ztrmm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, PLASMA_Complex64_t alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_ztrsm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, PLASMA_Complex64_t alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_ztrsmpl_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_ztrtri_Tile_Async (PLASMA_enum uplo, PLASMA_enum diag, PLASMA_desc ∗A, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zunglq_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zungqr_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗Q, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zunmlq_Tile_Async (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zunmqr_Tile_Async (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zLapack_to_Tile_Async (PLASMA_Complex64_t ∗Af77, int LDA, PLASMA_desc ∗A, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_zTile_to_Lapack_Async (PLASMA_desc ∗A, PLASMA_Complex64_t ∗Af77, int LDA, PLASMA_sequence ∗sequence, PLASMA_request ∗request)

3.9.1 Detailed Description

This is the group of double complex functions using the advanced asynchronous interface.

3.9.2 Function/Subroutine Documentation

3.9.2.1 int PLASMA_zcgels_Tile_Async (PLASMA_enum *trans*, PLASMA_desc * *A*, PLASMA_desc * *T*, PLASMA_desc * *B*, PLASMA_desc * *X*, int * *ITER*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_zcgels_Tile_Async - Solves overdetermined or underdetermined linear system of equations using the tile QR or the tile LQ factorization and mixed-precision iterative refinement. Non-blocking equivalent of PLASMA_zcgels_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zcgels
PLASMA_zcgels_Tile
PLASMA_dsgels_Tile_Async
PLASMA_zgels_Tile_Async

3.9.2.2 int PLASMA_zcgesv_Tile_Async (PLASMA_desc * *A*, PLASMA_desc * *L*, int * *IPIV*, PLASMA_desc * *B*, PLASMA_desc * *X*, int * *ITER*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_zcgesv_Tile_Async - Solves a system of linear equations using the tile LU factorization and mixed-precision iterative refinement. Non-blocking equivalent of PLASMA_zcgesv_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zcgesv
PLASMA_zcgesv_Tile
PLASMA_dsgesv_Tile_Async
PLASMA_zgesv_Tile_Async

3.9.2.3 int PLASMA_zcposv_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc * *A*, PLASMA_desc * *B*, PLASMA_desc * *X*, int * *ITER*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_zcposv_Tile_Async - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization and mixed-precision iterative refinement. Non-blocking equivalent of PLASMA_zcposv_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zcposv
PLASMA_zcposv_Tile
PLASMA_dsposv_Tile_Async
PLASMA_zposv_Tile_Async

3.9.2.4 int PLASMA_zcungesv_Tile_Async (PLASMA_enum *trans*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_desc ∗ *X*, int ∗ *ITER*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_zcungesv_Tile_Async - Solves symmetric linear system of equations using the tile QR or the tile LQ factorization and mixed-precision iterative refinement. Non-blocking equivalent of PLASMA_-zcungesv_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zcungesv
PLASMA_zcungesv_Tile
PLASMA_dsungesv_Tile_Async
PLASMA_zgels_Tile_Async

3.9.2.5 int PLASMA_zgelqf_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_zgelqf_Tile_Async - Computes the tile LQ factorization of a matrix. Non-blocking equivalent of PLASMA_zgelqf_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zgelqf
PLASMA_zgelqf_Tile
PLASMA_cgelqf_Tile_Async

PLASMA_dgelqf_Tile_Async
PLASMA_sgelqf_Tile_Async
PLASMA_zgelqs_Tile_Async

3.9.2.6 int PLASMA_zgelqs_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_zgelqs_Tile_Async - Computes a minimum-norm solution using previously computed LQ factorization. Non-blocking equivalent of PLASMA_zgelqs_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zgelqs
PLASMA_zgelqs_Tile
PLASMA_cgelqs_Tile_Async
PLASMA_dgelqs_Tile_Async
PLASMA_sgelqs_Tile_Async
PLASMA_zgelqf_Tile_Async

3.9.2.7 int PLASMA_zgels_Tile_Async (PLASMA_enum *trans*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_zgels_Tile_Async - Solves overdetermined or underdetermined linear system of equations using the tile QR or the tile LQ factorization. Non-blocking equivalent of PLASMA_zgels_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zgels
PLASMA_zgels_Tile
PLASMA_cgels_Tile_Async
PLASMA_dgels_Tile_Async
PLASMA_sgels_Tile_Async

3.9.2.8 int PLASMA_zgemm_Tile_Async (PLASMA_enum *transA*, PLASMA_enum *transB*, PLASMA_Complex64_t *alpha*, PLASMA_desc * *A*, PLASMA_desc * *B*, PLASMA_Complex64_t *beta*, PLASMA_desc * *C*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_zgemm_Tile_Async - Performs matrix multiplication. Non-blocking equivalent of PLASMA_-zgemm_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zgemm
PLASMA_zgemm_Tile
PLASMA_cgemm_Tile_Async
PLASMA_dgemm_Tile_Async
PLASMA_sgemm_Tile_Async

3.9.2.9 int PLASMA_zgeqrf_Tile_Async (PLASMA_desc * *A*, PLASMA_desc * *T*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_zgeqrf_Tile_Async - Computes the tile QR factorization of a matrix. Non-blocking equivalent of PLASMA_zgeqrf_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zgeqrf
PLASMA_zgeqrf_Tile
PLASMA_cgeqrf_Tile_Async
PLASMA_dgeqrf_Tile_Async
PLASMA_sgeqrf_Tile_Async
PLASMA_zgeqrs_Tile_Async

3.9.2.10 int PLASMA_zgeqrs_Tile_Async (PLASMA_desc * *A*, PLASMA_desc * *T*, PLASMA_desc * *B*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_zgeqrs_Tile_Async - Computes a minimum-norm solution using the tile QR factorization. Non-blocking equivalent of PLASMA_zgeqrs_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zgeqrs
PLASMA_zgeqrs_Tile
PLASMA_cgeqrs_Tile_Async
PLASMA_dgeqrs_Tile_Async
PLASMA_sgeqrs_Tile_Async
PLASMA_zgeqrf_Tile_Async

3.9.2.11 int PLASMA_zgesv_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_zgesv_Tile_Async - Solves a system of linear equations using the tile LU factorization. Non-blocking equivalent of PLASMA_zgesv_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zgesv
PLASMA_zgesv_Tile
PLASMA_cgesv_Tile_Async
PLASMA_dgesv_Tile_Async
PLASMA_sgesv_Tile_Async
PLASMA_zcgesv_Tile_Async

3.9.2.12 int PLASMA_zgetrf_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_zgetrf_Tile_Async - Computes the tile LU factorization of a matrix. Non-blocking equivalent of PLASMA_zgetrf_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zgetrf

PLASMA_zgetrf_Tile
PLASMA_cgetrf_Tile_Async
PLASMA_dgetrf_Tile_Async
PLASMA_sgetrf_Tile_Async
PLASMA_zgetrs_Tile_Async

3.9.2.13 int PLASMA_zgetrs_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_zgetrs_Tile_Async - Solves a system of linear equations using previously computed LU factorization. Non-blocking equivalent of PLASMA_zgetrs_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zgetrs
PLASMA_zgetrs_Tile
PLASMA_cgetrs_Tile_Async
PLASMA_dgetrs_Tile_Async
PLASMA_sgetrs_Tile_Async
PLASMA_zgetrf_Tile_Async

3.9.2.14 int PLASMA_zhemm_Tile_Async (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_Complex64_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_Complex64_t *beta*, PLASMA_desc ∗ *C*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_zhemm_Tile_Async - Performs Hermitian matrix multiplication. Non-blocking equivalent of PLASMA_zhemm_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zhemm
PLASMA_zhemm_Tile
PLASMA_chemm_Tile_Async
PLASMA_dhemm_Tile_Async
PLASMA_shemm_Tile_Async

3.9.2.15 int PLASMA_zher2k_Tile_Async (PLASMA_enum *uplo*, PLASMA_enum *trans*, PLASMA_Complex64_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, double *beta*, PLASMA_desc ∗ *C*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_zher2k_Tile_Async - Performs Hermitian rank-k update. Non-blocking equivalent of PLASMA_zher2k_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zher2k
PLASMA_zher2k_Tile
PLASMA_cher2k_Tile_Async
PLASMA_dher2k_Tile_Async
PLASMA_sher2k_Tile_Async

3.9.2.16 int PLASMA_zherk_Tile_Async (PLASMA_enum *uplo*, PLASMA_enum *trans*, double *alpha*, PLASMA_desc ∗ *A*, double *beta*, PLASMA_desc ∗ *C*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_zherk_Tile_Async - Performs Hermitian rank-k update. Non-blocking equivalent of PLASMA_zherk_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zherk
PLASMA_zherk_Tile
PLASMA_cherk_Tile_Async
PLASMA_dherk_Tile_Async
PLASMA_sherk_Tile_Async

3.9.2.17 int PLASMA_zlange_Tile_Async (PLASMA_enum *norm*, PLASMA_desc ∗ *A*, double ∗ *work*, double ∗ *value*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_zlange_Tile_Async - Non-blocking equivalent of PLASMA_zlange_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zlange
PLASMA_zlange_Tile
PLASMA_clange_Tile_Async
PLASMA_dlange_Tile_Async
PLASMA_slange_Tile_Async

3.9.2.18 int PLASMA_zlanhe_Tile_Async (PLASMA_enum *norm*, PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, double ∗ *work*, double ∗ *value*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_zlanhe_Tile_Async - Non-blocking equivalent of PLASMA_zlanhe_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zlanhe
PLASMA_zlanhe_Tile
PLASMA_clanhe_Tile_Async
PLASMA_dlanhe_Tile_Async
PLASMA_slanhe_Tile_Async

3.9.2.19 int PLASMA_zlansy_Tile_Async (PLASMA_enum *norm*, PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, double ∗ *work*, double ∗ *value*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_zlansy_Tile_Async - Non-blocking equivalent of PLASMA_zlansy_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zlansy
PLASMA_zlansy_Tile
PLASMA_clansy_Tile_Async
PLASMA_dlansy_Tile_Async
PLASMA_slansy_Tile_Async

3.9.2.20 int PLASMA_zLapack_to_Tile_Async (PLASMA_Complex64_t ∗ *Af77*, int *LDA*, PLASMA_desc ∗ *A*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_zLapack_to_Tile_Async - Conversion from LAPACK layout to tile layout. Non-blocking equivalent of PLASMA_zLapack_to_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zTile_to_Lapack_Async
PLASMA_zLapack_to_Tile
PLASMA_cLapack_to_Tile_Async
PLASMA_dLapack_to_Tile_Async
PLASMA_sLapack_to_Tile_Async

3.9.2.21 int PLASMA_zlauum_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_zlauum_Tile_Async - Computes the product U ∗ U' or L' ∗ L, where the triangular factor U or L is stored in the upper or lower triangular part of the array A. Non-blocking equivalent of PLASMA_zlauum_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zlauum
PLASMA_zlauum_Tile
PLASMA_clauum_Tile_Async
PLASMA_dlauum_Tile_Async
PLASMA_slauum_Tile_Async
PLASMA_zpotri_Tile_Async

3.9.2.22 int PLASMA_zplghe_Tile_Async (double *bump*, PLASMA_desc ∗ *A*, unsigned long long int *seed*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_zplghe_Tile_Async - Generate a random hermitian matrix by tiles. Non-blocking equivalent of PLASMA_zplghe_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zplghe
PLASMA_zplghe_Tile
PLASMA_cplghe_Tile_Async
PLASMA_dplghe_Tile_Async
PLASMA_splghe_Tile_Async
PLASMA_zplghe_Tile_Async
PLASMA_zplgsy_Tile_Async

3.9.2.23 int PLASMA_zplgsy_Tile_Async (PLASMA_Complex64_t *bump*, PLASMA_desc ∗ *A*, unsigned long long int *seed*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_zplgsy_Tile_Async - Generate a random hermitian matrix by tiles. Non-blocking equivalent of PLASMA_zplgsy_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zplgsy
PLASMA_zplgsy_Tile
PLASMA_cplgsy_Tile_Async
PLASMA_dplgsy_Tile_Async
PLASMA_splgsy_Tile_Async
PLASMA_zplgsy_Tile_Async
PLASMA_zplgsy_Tile_Async

3.9.2.24 int PLASMA_zplrnt_Tile_Async (PLASMA_desc ∗ *A*, unsigned long long int *seed*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_zplrnt_Tile_Async - Generate a random matrix by tiles. Non-blocking equivalent of PLASMA_zplrnt_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zplrnt
PLASMA_zplrnt_Tile
PLASMA_cplrnt_Tile_Async
PLASMA_dplrnt_Tile_Async
PLASMA_splrnt_Tile_Async
PLASMA_zplghe_Tile_Async
PLASMA_zplgsy_Tile_Async

3.9.2.25 int PLASMA_zposv_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc $*$ *A*, PLASMA_desc $*$ *B*, PLASMA_sequence $*$ *sequence*, PLASMA_request $*$ *request*)

PLASMA_zposv_Tile_Async - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization. Non-blocking equivalent of PLASMA_zposv_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

$\leftarrow$ ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

$\rightarrow$ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zposv
PLASMA_zposv_Tile
PLASMA_cposv_Tile_Async
PLASMA_dposv_Tile_Async
PLASMA_sposv_Tile_Async

3.9.2.26 int PLASMA_zpotrf_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc $*$ *A*, PLASMA_sequence $*$ *sequence*, PLASMA_request $*$ *request*)

PLASMA_zpotrf_Tile_Async - Computes the Cholesky factorization of a symmetric positive definite or Hermitian positive definite matrix. Non-blocking equivalent of PLASMA_zpotrf_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

$\leftarrow$ ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

$\rightarrow$ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zpotrf
PLASMA_zpotrf_Tile
PLASMA_cpotrf_Tile_Async
PLASMA_dpotrf_Tile_Async
PLASMA_spotrf_Tile_Async
PLASMA_zpotrs_Tile_Async

3.9.2.27 int PLASMA_zpotri_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc $*$ *A*, PLASMA_sequence $*$ *sequence*, PLASMA_request $*$ *request*)

PLASMA_zpotri_Tile_Async - Computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization A = U**H*U or A = L*L**H computed by PLASMA_zpotrf. Non-blocking equivalent of PLASMA_zpotri_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zpotri
PLASMA_zpotri_Tile
PLASMA_cpotri_Tile_Async
PLASMA_dpotri_Tile_Async
PLASMA_spotri_Tile_Async
PLASMA_zpotrf_Tile_Async

3.9.2.28 int PLASMA_zpotrs_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc $*$ *A*, PLASMA_desc $*$ *B*, PLASMA_sequence $*$ *sequence*, PLASMA_request $*$ *request*)

PLASMA_zpotrs_Tile_Async - Solves a system of linear equations using previously computed Cholesky factorization. Non-blocking equivalent of PLASMA_zpotrs_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zpotrs
PLASMA_zpotrs_Tile
PLASMA_cpotrs_Tile_Async
PLASMA_dpotrs_Tile_Async
PLASMA_spotrs_Tile_Async
PLASMA_zpotrf_Tile_Async

3.9.2.29 int PLASMA_zsymm_Tile_Async (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_Complex64_t *alpha*, PLASMA_desc $*$ *A*, PLASMA_desc $*$ *B*, PLASMA_Complex64_t *beta*, PLASMA_desc $*$ *C*, PLASMA_sequence $*$ *sequence*, PLASMA_request $*$ *request*)

PLASMA_zsymm_Tile_Async - Performs symmetric matrix multiplication. Non-blocking equivalent of PLASMA_zsymm_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zsymm
PLASMA_zsymm_Tile
PLASMA_csymm_Tile_Async
PLASMA_dsymm_Tile_Async
PLASMA_ssymm_Tile_Async

3.9.2.30 int PLASMA_zsyr2k_Tile_Async (PLASMA_enum *uplo*, PLASMA_enum *trans*, PLASMA_Complex64_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_Complex64_t *beta*, PLASMA_desc ∗ *C*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_zsyr2k_Tile_Async - Performs symmetric rank-k update. Non-blocking equivalent of PLASMA_zsyr2k_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zsyr2k
PLASMA_zsyr2k_Tile
PLASMA_csyr2k_Tile_Async
PLASMA_dsyr2k_Tile_Async
PLASMA_ssyr2k_Tile_Async

3.9.2.31 int PLASMA_zsyrk_Tile_Async (PLASMA_enum *uplo*, PLASMA_enum *trans*, PLASMA_Complex64_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_Complex64_t *beta*, PLASMA_desc ∗ *C*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_zsyrk_Tile_Async - Performs rank-k update. Non-blocking equivalent of PLASMA_zsyrk_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zsyrk

PLASMA_zsyrk_Tile
PLASMA_csyrk_Tile_Async
PLASMA_dsyrk_Tile_Async
PLASMA_ssyrk_Tile_Async

3.9.2.32 int PLASMA_zTile_to_Lapack_Async (PLASMA_desc ∗ *A*, PLASMA_Complex64_t ∗ *Af77*, int *LDA*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_zTile_to_Lapack_Async - Conversion from LAPACK layout to tile layout. Non-blocking equivalent of PLASMA_zTile_to_Lapack(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zLapack_to_Tile_Async
PLASMA_zTile_to_Lapack
PLASMA_cTile_to_Lapack_Async
PLASMA_dTile_to_Lapack_Async
PLASMA_sTile_to_Lapack_Async

3.9.2.33 int PLASMA_ztrmm_Tile_Async (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, PLASMA_Complex64_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_ztrmm_Tile_Async - Performs triangular matrix multiplication. Non-blocking equivalent of PLASMA_ztrmm_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_ztrmm
PLASMA_ztrmm_Tile
PLASMA_ctrmm_Tile_Async
PLASMA_dtrmm_Tile_Async
PLASMA_strmm_Tile_Async

3.9.2.34 int PLASMA_ztrsm_Tile_Async (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, PLASMA_Complex64_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_ztrsm_Tile_Async - Computes triangular solve. Non-blocking equivalent of PLASMA_ztrsm_-Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_ztrsm
PLASMA_ztrsm_Tile
PLASMA_ctrsm_Tile_Async
PLASMA_dtrsm_Tile_Async
PLASMA_strsm_Tile_Async

3.9.2.35 int PLASMA_ztrsmpl_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_ztrsmpl_Tile - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix. Non-blocking equivalent of PLASMA_ztrsmpl_Tile(). Returns control to the user thread before worker threads finish the computation to allow for pipelined execution of diferent routines.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_ztrsmpl
PLASMA_ztrsmpl_Tile
PLASMA_ctrsmpl_Tile_Async
PLASMA_dtrsmpl_Tile_Async
PLASMA_strsmpl_Tile_Async
PLASMA_zgetrf_Tile_Async

3.9.2.36 int PLASMA_ztrtri_Tile_Async (PLASMA_enum *uplo*, PLASMA_enum *diag*, PLASMA_desc ∗ *A*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_ztrtri_Tile_Async - Computes the inverse of a complex upper or lower triangular matrix A. Non-blocking equivalent of PLASMA_ztrtri_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_ztrtri
PLASMA_ztrtri_Tile
PLASMA_ctrtri_Tile_Async
PLASMA_dtrtri_Tile_Async
PLASMA_strtri_Tile_Async
PLASMA_zpotri_Tile_Async

3.9.2.37 int PLASMA_zunglq_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

Non-blocking equivalent of PLASMA_zunglq_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zunglq
PLASMA_zunglq_Tile
PLASMA_cunglq_Tile_Async
PLASMA_dunglq_Tile_Async
PLASMA_sunglq_Tile_Async
PLASMA_zgelqf_Tile_Async

3.9.2.38 int PLASMA_zungqr_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *Q*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

Non-blocking equivalent of PLASMA_zungqr_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zungqr
PLASMA_zungqr_Tile

PLASMA_cungqr_Tile_Async
PLASMA_dungqr_Tile_Async
PLASMA_sungqr_Tile_Async
PLASMA_zgeqrf_Tile_Async

3.9.2.39 int PLASMA_zunmlq_Tile_Async (PLASMA_enum *side*, PLASMA_enum *trans*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

Non-blocking equivalent of PLASMA_zunmlq_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zunmlq
PLASMA_zunmlq_Tile
PLASMA_cunmlq_Tile_Async
PLASMA_dunmlq_Tile_Async
PLASMA_sunmlq_Tile_Async
PLASMA_zgelqf_Tile_Async

3.9.2.40 int PLASMA_zunmqr_Tile_Async (PLASMA_enum *side*, PLASMA_enum *trans*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

Non-blocking equivalent of PLASMA_zunmqr_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_zunmqr
PLASMA_zunmqr_Tile
PLASMA_cunmqr_Tile_Async
PLASMA_dunmqr_Tile_Async
PLASMA_sunmqr_Tile_Async
PLASMA_zgeqrf_Tile_Async

3.10 Advanced Interface: Asynchronous - Single Complex

Functions/Subroutines

- int PLASMA_cgelqf_Tile_Async (PLASMA_desc *A, PLASMA_desc *T, PLASMA_sequence *sequence, PLASMA_request *request)
- int PLASMA_cgelqs_Tile_Async (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int PLASMA_cgels_Tile_Async (PLASMA_enum trans, PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int PLASMA_cgemm_Tile_Async (PLASMA_enum transA, PLASMA_enum transB, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_Complex32_t beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int PLASMA_cgeqrf_Tile_Async (PLASMA_desc *A, PLASMA_desc *T, PLASMA_sequence *sequence, PLASMA_request *request)
- int PLASMA_cgeqrs_Tile_Async (PLASMA_desc *A, PLASMA_desc *T, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int PLASMA_cgesv_Tile_Async (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int PLASMA_cgetrf_Tile_Async (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_sequence *sequence, PLASMA_request *request)
- int PLASMA_cgetrs_Tile_Async (PLASMA_desc *A, PLASMA_desc *L, int *IPIV, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int PLASMA_chemm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B, PLASMA_Complex32_t beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int PLASMA_cher2k_Tile_Async (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex32_t alpha, PLASMA_desc *A, PLASMA_desc *B, float beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int PLASMA_cherk_Tile_Async (PLASMA_enum uplo, PLASMA_enum trans, float alpha, PLASMA_desc *A, float beta, PLASMA_desc *C, PLASMA_sequence *sequence, PLASMA_request *request)
- int PLASMA_clange_Tile_Async (PLASMA_enum norm, PLASMA_desc *A, float *work, float *value, PLASMA_sequence *sequence, PLASMA_request *request)
- int PLASMA_clanhe_Tile_Async (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc *A, float *work, float *value, PLASMA_sequence *sequence, PLASMA_request *request)
- int PLASMA_clansy_Tile_Async (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc *A, float *work, float *value, PLASMA_sequence *sequence, PLASMA_request *request)
- int PLASMA_clauum_Tile_Async (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)
- int PLASMA_cplghe_Tile_Async (float bump, PLASMA_desc *A, unsigned long long int seed, PLASMA_sequence *sequence, PLASMA_request *request)
- int PLASMA_cplgsy_Tile_Async (PLASMA_Complex32_t bump, PLASMA_desc *A, unsigned long long int seed, PLASMA_sequence *sequence, PLASMA_request *request)
- int PLASMA_cplrnt_Tile_Async (PLASMA_desc *A, unsigned long long int seed, PLASMA_sequence *sequence, PLASMA_request *request)
- int PLASMA_cposv_Tile_Async (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_desc *B, PLASMA_sequence *sequence, PLASMA_request *request)
- int PLASMA_cpotrf_Tile_Async (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)
- int PLASMA_cpotri_Tile_Async (PLASMA_enum uplo, PLASMA_desc *A, PLASMA_sequence *sequence, PLASMA_request *request)

- int PLASMA_cpotrs_Tile_Async (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_csymm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_Complex32_t alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_Complex32_t beta, PLASMA_desc ∗C, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_csyr2k_Tile_Async (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex32_t alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_Complex32_t beta, PLASMA_desc ∗C, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_csyrk_Tile_Async (PLASMA_enum uplo, PLASMA_enum trans, PLASMA_Complex32_t alpha, PLASMA_desc ∗A, PLASMA_Complex32_t beta, PLASMA_desc ∗C, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_ctrmm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, PLASMA_Complex32_t alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_ctrsm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, PLASMA_Complex32_t alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_ctrsmpl_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_ctrtri_Tile_Async (PLASMA_enum uplo, PLASMA_enum diag, PLASMA_desc ∗A, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_cunglq_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_cungqr_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗Q, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_cunmlq_Tile_Async (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_cunmqr_Tile_Async (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_cLapack_to_Tile_Async (PLASMA_Complex32_t ∗Af77, int LDA, PLASMA_desc ∗A, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_cTile_to_Lapack_Async (PLASMA_desc ∗A, PLASMA_Complex32_t ∗Af77, int LDA, PLASMA_sequence ∗sequence, PLASMA_request ∗request)

3.10.1 Detailed Description

This is the group of single complex functions using the advanced asynchronous interface.

3.10.2 Function/Subroutine Documentation

3.10.2.1 int PLASMA_cgelqf_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_cgelqf_Tile_Async - Computes the tile LQ factorization of a matrix. Non-blocking equivalent of PLASMA_cgelqf_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cgelqf
PLASMA_cgelqf_Tile
PLASMA_cgelqf_Tile_Async
PLASMA_dgelqf_Tile_Async
PLASMA_sgelqf_Tile_Async
PLASMA_cgelqs_Tile_Async

3.10.2.2 int PLASMA_cgelqs_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_cgelqs_Tile_Async - Computes a minimum-norm solution using previously computed LQ factorization. Non-blocking equivalent of PLASMA_cgelqs_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cgelqs
PLASMA_cgelqs_Tile
PLASMA_cgelqs_Tile_Async
PLASMA_dgelqs_Tile_Async
PLASMA_sgelqs_Tile_Async
PLASMA_cgelqf_Tile_Async

3.10.2.3 int PLASMA_cgels_Tile_Async (PLASMA_enum *trans*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_cgels_Tile_Async - Solves overdetermined or underdetermined linear system of equations using the tile QR or the tile LQ factorization. Non-blocking equivalent of PLASMA_cgels_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cgels
PLASMA_cgels_Tile
PLASMA_cgels_Tile_Async
PLASMA_dgels_Tile_Async
PLASMA_sgels_Tile_Async

3.10.2.4 int PLASMA_cgemm_Tile_Async (PLASMA_enum *transA*, PLASMA_enum *transB*, PLASMA_Complex32_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_Complex32_t *beta*, PLASMA_desc ∗ *C*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_cgemm_Tile_Async - Performs matrix multiplication. Non-blocking equivalent of PLASMA_-cgemm_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cgemm
PLASMA_cgemm_Tile
PLASMA_cgemm_Tile_Async
PLASMA_dgemm_Tile_Async
PLASMA_sgemm_Tile_Async

3.10.2.5 int PLASMA_cgeqrf_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_cgeqrf_Tile_Async - Computes the tile QR factorization of a matrix. Non-blocking equivalent of PLASMA_cgeqrf_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cgeqrf
PLASMA_cgeqrf_Tile
PLASMA_cgeqrf_Tile_Async
PLASMA_dgeqrf_Tile_Async
PLASMA_sgeqrf_Tile_Async
PLASMA_cgeqrs_Tile_Async

3.10.2.6 int PLASMA_cgeqrs_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_cgeqrs_Tile_Async - Computes a minimum-norm solution using the tile QR factorization. Non-blocking equivalent of PLASMA_cgeqrs_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cgeqrs
PLASMA_cgeqrs_Tile
PLASMA_cgeqrs_Tile_Async
PLASMA_dgeqrs_Tile_Async
PLASMA_sgeqrs_Tile_Async
PLASMA_cgeqrf_Tile_Async

3.10.2.7 int PLASMA_cgesv_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_cgesv_Tile_Async - Solves a system of linear equations using the tile LU factorization. Non-blocking equivalent of PLASMA_cgesv_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cgesv
PLASMA_cgesv_Tile
PLASMA_cgesv_Tile_Async
PLASMA_dgesv_Tile_Async
PLASMA_sgesv_Tile_Async
PLASMA_ccgesv_Tile_Async

3.10.2.8 int PLASMA_cgetrf_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_cgetrf_Tile_Async - Computes the tile LU factorization of a matrix. Non-blocking equivalent of PLASMA_cgetrf_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cgetrf

PLASMA_cgetrf_Tile
PLASMA_cgetrf_Tile_Async
PLASMA_dgetrf_Tile_Async
PLASMA_sgetrf_Tile_Async
PLASMA_cgetrs_Tile_Async

3.10.2.9 int PLASMA_cgetrs_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_cgetrs_Tile_Async - Solves a system of linear equations using previously computed LU factorization. Non-blocking equivalent of PLASMA_cgetrs_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cgetrs
PLASMA_cgetrs_Tile
PLASMA_cgetrs_Tile_Async
PLASMA_dgetrs_Tile_Async
PLASMA_sgetrs_Tile_Async
PLASMA_cgetrf_Tile_Async

3.10.2.10 int PLASMA_chemm_Tile_Async (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_Complex32_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_Complex32_t *beta*, PLASMA_desc ∗ *C*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_chemm_Tile_Async - Performs Hermitian matrix multiplication. Non-blocking equivalent of PLASMA_chemm_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_chemm
PLASMA_chemm_Tile
PLASMA_chemm_Tile_Async
PLASMA_dhemm_Tile_Async
PLASMA_shemm_Tile_Async

3.10.2.11 int PLASMA_cher2k_Tile_Async (PLASMA_enum *uplo*, PLASMA_enum *trans*, PLASMA_Complex32_t *alpha*, PLASMA_desc * *A*, PLASMA_desc * *B*, float *beta*, PLASMA_desc * *C*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_cher2k_Tile_Async - Performs Hermitian rank-k update. Non-blocking equivalent of PLASMA_cher2k_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cher2k
PLASMA_cher2k_Tile
PLASMA_cher2k_Tile_Async
PLASMA_dher2k_Tile_Async
PLASMA_sher2k_Tile_Async

3.10.2.12 int PLASMA_cherk_Tile_Async (PLASMA_enum *uplo*, PLASMA_enum *trans*, float *alpha*, PLASMA_desc * *A*, float *beta*, PLASMA_desc * *C*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_cherk_Tile_Async - Performs Hermitian rank-k update. Non-blocking equivalent of PLASMA_cherk_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cherk
PLASMA_cherk_Tile
PLASMA_cherk_Tile_Async
PLASMA_dherk_Tile_Async
PLASMA_sherk_Tile_Async

3.10.2.13 int PLASMA_clange_Tile_Async (PLASMA_enum *norm*, PLASMA_desc * *A*, float * *work*, float * *value*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_clange_Tile_Async - Non-blocking equivalent of PLASMA_clange_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_clange
PLASMA_clange_Tile
PLASMA_clange_Tile_Async
PLASMA_dlange_Tile_Async
PLASMA_slange_Tile_Async

3.10.2.14 int PLASMA_clanhe_Tile_Async (PLASMA_enum *norm*, PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, float ∗ *work*, float ∗ *value*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_clanhe_Tile_Async - Non-blocking equivalent of PLASMA_clanhe_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_clanhe
PLASMA_clanhe_Tile
PLASMA_clanhe_Tile_Async
PLASMA_dlanhe_Tile_Async
PLASMA_slanhe_Tile_Async

3.10.2.15 int PLASMA_clansy_Tile_Async (PLASMA_enum *norm*, PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, float ∗ *work*, float ∗ *value*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_clansy_Tile_Async - Non-blocking equivalent of PLASMA_clansy_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_clansy
PLASMA_clansy_Tile
PLASMA_clansy_Tile_Async
PLASMA_dlansy_Tile_Async
PLASMA_slansy_Tile_Async

3.10.2.16 int PLASMA_cLapack_to_Tile_Async (PLASMA_Complex32_t ∗ *Af77*, int *LDA*, PLASMA_desc ∗ *A*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_cLapack_to_Tile_Async - Conversion from LAPACK layout to tile layout. Non-blocking equivalent of PLASMA_cLapack_to_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cTile_to_Lapack_Async
PLASMA_cLapack_to_Tile
PLASMA_cLapack_to_Tile_Async
PLASMA_dLapack_to_Tile_Async
PLASMA_sLapack_to_Tile_Async

3.10.2.17 int PLASMA_clauum_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_clauum_Tile_Async - Computes the product U ∗ U' or L' ∗ L, where the triangular factor U or L is stored in the upper or lower triangular part of the array A. Non-blocking equivalent of PLASMA_clauum_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_clauum
PLASMA_clauum_Tile
PLASMA_clauum_Tile_Async
PLASMA_dlauum_Tile_Async
PLASMA_slauum_Tile_Async
PLASMA_cpotri_Tile_Async

3.10.2.18 int PLASMA_cplghe_Tile_Async (float *bump*, PLASMA_desc ∗ *A*, unsigned long long int *seed*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_cplghe_Tile_Async - Generate a random hermitian matrix by tiles. Non-blocking equivalent of PLASMA_cplghe_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cplghe
PLASMA_cplghe_Tile
PLASMA_cplghe_Tile_Async
PLASMA_dplghe_Tile_Async
PLASMA_splghe_Tile_Async
PLASMA_cplghe_Tile_Async
PLASMA_cplgsy_Tile_Async

3.10.2.19 int PLASMA_cplgsy_Tile_Async (PLASMA_Complex32_t *bump*, PLASMA_desc ∗ *A*, unsigned long long int *seed*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_cplgsy_Tile_Async - Generate a random hermitian matrix by tiles. Non-blocking equivalent of PLASMA_cplgsy_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cplgsy
PLASMA_cplgsy_Tile
PLASMA_cplgsy_Tile_Async
PLASMA_dplgsy_Tile_Async
PLASMA_splgsy_Tile_Async
PLASMA_cplgsy_Tile_Async
PLASMA_cplgsy_Tile_Async

3.10.2.20 int PLASMA_cplrnt_Tile_Async (PLASMA_desc ∗ *A*, unsigned long long int *seed*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_cplrnt_Tile_Async - Generate a random matrix by tiles. Non-blocking equivalent of PLASMA_cplrnt_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cplrnt
PLASMA_cplrnt_Tile
PLASMA_cplrnt_Tile_Async
PLASMA_dplrnt_Tile_Async
PLASMA_splrnt_Tile_Async
PLASMA_cplghe_Tile_Async
PLASMA_cplgsy_Tile_Async

3.10.2.21 int PLASMA_cposv_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_cposv_Tile_Async - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization. Non-blocking equivalent of PLASMA_cposv_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cposv
PLASMA_cposv_Tile
PLASMA_cposv_Tile_Async
PLASMA_dposv_Tile_Async
PLASMA_sposv_Tile_Async

3.10.2.22 int PLASMA_cpotrf_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_cpotrf_Tile_Async - Computes the Cholesky factorization of a symmetric positive definite or Hermitian positive definite matrix. Non-blocking equivalent of PLASMA_cpotrf_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cpotrf
PLASMA_cpotrf_Tile
PLASMA_cpotrf_Tile_Async
PLASMA_dpotrf_Tile_Async
PLASMA_spotrf_Tile_Async
PLASMA_cpotrs_Tile_Async

3.10.2.23 int PLASMA_cpotri_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc * *A*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_cpotri_Tile_Async - Computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization A = U**H*U or A = L*L**H computed by PLASMA_cpotrf. Non-blocking equivalent of PLASMA_cpotri_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cpotri
PLASMA_cpotri_Tile
PLASMA_cpotri_Tile_Async
PLASMA_dpotri_Tile_Async
PLASMA_spotri_Tile_Async
PLASMA_cpotrf_Tile_Async

3.10.2.24 int PLASMA_cpotrs_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc * *A*, PLASMA_desc * *B*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_cpotrs_Tile_Async - Solves a system of linear equations using previously computed Cholesky factorization. Non-blocking equivalent of PLASMA_cpotrs_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cpotrs
PLASMA_cpotrs_Tile
PLASMA_cpotrs_Tile_Async
PLASMA_dpotrs_Tile_Async
PLASMA_spotrs_Tile_Async
PLASMA_cpotrf_Tile_Async

3.10.2.25 int PLASMA_csymm_Tile_Async (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_Complex32_t *alpha*, PLASMA_desc * *A*, PLASMA_desc * *B*, PLASMA_Complex32_t *beta*, PLASMA_desc * *C*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_csymm_Tile_Async - Performs symmetric matrix multiplication. Non-blocking equivalent of PLASMA_csymm_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_csymm
PLASMA_csymm_Tile
PLASMA_csymm_Tile_Async
PLASMA_dsymm_Tile_Async
PLASMA_ssymm_Tile_Async

3.10.2.26 int PLASMA_csyr2k_Tile_Async (PLASMA_enum *uplo*, PLASMA_enum *trans*, PLASMA_Complex32_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_Complex32_t *beta*, PLASMA_desc ∗ *C*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_csyr2k_Tile_Async - Performs symmetric rank-k update. Non-blocking equivalent of PLASMA_csyr2k_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_csyr2k
PLASMA_csyr2k_Tile
PLASMA_csyr2k_Tile_Async
PLASMA_dsyr2k_Tile_Async
PLASMA_ssyr2k_Tile_Async

3.10.2.27 int PLASMA_csyrk_Tile_Async (PLASMA_enum *uplo*, PLASMA_enum *trans*, PLASMA_Complex32_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_Complex32_t *beta*, PLASMA_desc ∗ *C*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_csyrk_Tile_Async - Performs rank-k update. Non-blocking equivalent of PLASMA_csyrk_-Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_csyrk

PLASMA_csyrk_Tile
PLASMA_csyrk_Tile_Async
PLASMA_dsyrk_Tile_Async
PLASMA_ssyrk_Tile_Async

3.10.2.28 int PLASMA_cTile_to_Lapack_Async (PLASMA_desc ∗ *A*, PLASMA_Complex32_t ∗ *Af77*, int *LDA*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_cTile_to_Lapack_Async - Conversion from LAPACK layout to tile layout. Non-blocking equivalent of PLASMA_cTile_to_Lapack(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cLapack_to_Tile_Async
PLASMA_cTile_to_Lapack
PLASMA_cTile_to_Lapack_Async
PLASMA_dTile_to_Lapack_Async
PLASMA_sTile_to_Lapack_Async

3.10.2.29 int PLASMA_ctrmm_Tile_Async (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, PLASMA_Complex32_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_ctrmm_Tile_Async - Performs triangular matrix multiplication. Non-blocking equivalent of PLASMA_ctrmm_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_ctrmm
PLASMA_ctrmm_Tile
PLASMA_ctrmm_Tile_Async
PLASMA_dtrmm_Tile_Async
PLASMA_strmm_Tile_Async

3.10.2.30 int PLASMA_ctrsm_Tile_Async (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, PLASMA_Complex32_t *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_ctrsm_Tile_Async - Computes triangular solve. Non-blocking equivalent of PLASMA_ctrsm_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_ctrsm
PLASMA_ctrsm_Tile
PLASMA_ctrsm_Tile_Async
PLASMA_dtrsm_Tile_Async
PLASMA_strsm_Tile_Async

3.10.2.31 int PLASMA_ctrsmpl_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_ctrsmpl_Tile - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix. Non-blocking equivalent of PLASMA_ctrsmpl_Tile(). Returns control to the user thread before worker threads finish the computation to allow for pipelined execution of diferent routines.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_ctrsmpl
PLASMA_ctrsmpl_Tile
PLASMA_ctrsmpl_Tile_Async
PLASMA_dtrsmpl_Tile_Async
PLASMA_strsmpl_Tile_Async
PLASMA_cgetrf_Tile_Async

3.10.2.32 int PLASMA_ctrtri_Tile_Async (PLASMA_enum *uplo*, PLASMA_enum *diag*, PLASMA_desc ∗ *A*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_ctrtri_Tile_Async - Computes the inverse of a complex upper or lower triangular matrix A. Non-blocking equivalent of PLASMA_ctrtri_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_ctrtri
PLASMA_ctrtri_Tile
PLASMA_ctrtri_Tile_Async
PLASMA_dtrtri_Tile_Async
PLASMA_strtri_Tile_Async
PLASMA_cpotri_Tile_Async

3.10.2.33 int PLASMA_cunglq_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

Non-blocking equivalent of PLASMA_cunglq_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cunglq
PLASMA_cunglq_Tile
PLASMA_cunglq_Tile_Async
PLASMA_dunglq_Tile_Async
PLASMA_sunglq_Tile_Async
PLASMA_cgelqf_Tile_Async

3.10.2.34 int PLASMA_cungqr_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *Q*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

Non-blocking equivalent of PLASMA_cungqr_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cungqr
PLASMA_cungqr_Tile

PLASMA_cungqr_Tile_Async
PLASMA_dungqr_Tile_Async
PLASMA_sungqr_Tile_Async
PLASMA_cgeqrf_Tile_Async

3.10.2.35 int PLASMA_cunmlq_Tile_Async (PLASMA_enum *side*, PLASMA_enum *trans*, PLASMA_desc $*$ *A*, PLASMA_desc $*$ *T*, PLASMA_desc $*$ *B*, PLASMA_sequence $*$ *sequence*, PLASMA_request $*$ *request*)

Non-blocking equivalent of PLASMA_cunmlq_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cunmlq
PLASMA_cunmlq_Tile
PLASMA_cunmlq_Tile_Async
PLASMA_dunmlq_Tile_Async
PLASMA_sunmlq_Tile_Async
PLASMA_cgelqf_Tile_Async

3.10.2.36 int PLASMA_cunmqr_Tile_Async (PLASMA_enum *side*, PLASMA_enum *trans*, PLASMA_desc $*$ *A*, PLASMA_desc $*$ *T*, PLASMA_desc $*$ *B*, PLASMA_sequence $*$ *sequence*, PLASMA_request $*$ *request*)

Non-blocking equivalent of PLASMA_cunmqr_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_cunmqr
PLASMA_cunmqr_Tile
PLASMA_cunmqr_Tile_Async
PLASMA_dunmqr_Tile_Async
PLASMA_sunmqr_Tile_Async
PLASMA_cgeqrf_Tile_Async

3.11 Advanced Interface: Asynchronous - Double Real

Functions/Subroutines

- int PLASMA_dgelqf_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dgelqs_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dgels_Tile_Async (PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dgemm_Tile_Async (PLASMA_enum transA, PLASMA_enum transB, double alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, double beta, PLASMA_desc ∗C, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dgeqrf_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dgeqrs_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dgesv_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dgetrf_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dgetrs_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dlange_Tile_Async (PLASMA_enum norm, PLASMA_desc ∗A, double ∗work, double ∗value, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dlansy_Tile_Async (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc ∗A, double ∗work, double ∗value, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dlauum_Tile_Async (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dorglq_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dorgqr_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗Q, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dormlq_Tile_Async (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dormqr_Tile_Async (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dplgsy_Tile_Async (double bump, PLASMA_desc ∗A, unsigned long long int seed, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dplrnt_Tile_Async (PLASMA_desc ∗A, unsigned long long int seed, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dposv_Tile_Async (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dpotrf_Tile_Async (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dpotri_Tile_Async (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dpotrs_Tile_Async (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)

- int PLASMA_dsgesv_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_desc ∗B, PLASMA_desc ∗X, int ∗ITER, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dsposv_Tile_Async (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_desc ∗X, int ∗ITER, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dsungesv_Tile_Async (PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_desc ∗X, int ∗ITER, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dsymm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, double alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, double beta, PLASMA_desc ∗C, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dsyr2k_Tile_Async (PLASMA_enum uplo, PLASMA_enum trans, double alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, double beta, PLASMA_desc ∗C, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dsyrk_Tile_Async (PLASMA_enum uplo, PLASMA_enum trans, double alpha, PLASMA_desc ∗A, double beta, PLASMA_desc ∗C, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dtrmm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, double alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dtrsm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, double alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dtrsmpl_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dtrtri_Tile_Async (PLASMA_enum uplo, PLASMA_enum diag, PLASMA_desc ∗A, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dLapack_to_Tile_Async (double ∗Af77, int LDA, PLASMA_desc ∗A, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_dTile_to_Lapack_Async (PLASMA_desc ∗A, double ∗Af77, int LDA, PLASMA_sequence ∗sequence, PLASMA_request ∗request)

3.11.1 Detailed Description

This is the group of double real functions using the advanced asynchronous interface.

3.11.2 Function/Subroutine Documentation

3.11.2.1 int PLASMA_dgelqf_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_dgelqf_Tile_Async - Computes the tile LQ factorization of a matrix. Non-blocking equivalent of PLASMA_dgelqf_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dgelqf
PLASMA_dgelqf_Tile
PLASMA_cgelqf_Tile_Async
PLASMA_dgelqf_Tile_Async
PLASMA_sgelqf_Tile_Async
PLASMA_dgelqs_Tile_Async

3.11.2.2 int PLASMA_dgelqs_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_dgelqs_Tile_Async - Computes a minimum-norm solution using previously computed LQ factorization. Non-blocking equivalent of PLASMA_dgelqs_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dgelqs
PLASMA_dgelqs_Tile
PLASMA_cgelqs_Tile_Async
PLASMA_dgelqs_Tile_Async
PLASMA_sgelqs_Tile_Async
PLASMA_dgelqf_Tile_Async

3.11.2.3 int PLASMA_dgels_Tile_Async (PLASMA_enum *trans*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_dgels_Tile_Async - Solves overdetermined or underdetermined linear system of equations using the tile QR or the tile LQ factorization. Non-blocking equivalent of PLASMA_dgels_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dgels
PLASMA_dgels_Tile
PLASMA_cgels_Tile_Async
PLASMA_dgels_Tile_Async
PLASMA_sgels_Tile_Async

3.11.2.4 int PLASMA_dgemm_Tile_Async (PLASMA_enum *transA*, PLASMA_enum *transB*, double *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, double *beta*, PLASMA_desc ∗ *C*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_dgemm_Tile_Async - Performs matrix multiplication. Non-blocking equivalent of PLASMA_dgemm_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dgemm
PLASMA_dgemm_Tile
PLASMA_cgemm_Tile_Async
PLASMA_dgemm_Tile_Async
PLASMA_sgemm_Tile_Async

3.11.2.5 int PLASMA_dgeqrf_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_dgeqrf_Tile_Async - Computes the tile QR factorization of a matrix. Non-blocking equivalent of PLASMA_dgeqrf_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dgeqrf
PLASMA_dgeqrf_Tile
PLASMA_cgeqrf_Tile_Async
PLASMA_dgeqrf_Tile_Async
PLASMA_sgeqrf_Tile_Async
PLASMA_dgeqrs_Tile_Async

3.11.2.6 int PLASMA_dgeqrs_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_dgeqrs_Tile_Async - Computes a minimum-norm solution using the tile QR factorization. Non-blocking equivalent of PLASMA_dgeqrs_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dgeqrs
PLASMA_dgeqrs_Tile
PLASMA_cgeqrs_Tile_Async
PLASMA_dgeqrs_Tile_Async
PLASMA_sgeqrs_Tile_Async
PLASMA_dgeqrf_Tile_Async

3.11.2.7 int PLASMA_dgesv_Tile_Async (PLASMA_desc $*$ *A*, PLASMA_desc $*$ *L*, int $*$ *IPIV*, PLASMA_desc $*$ *B*, PLASMA_sequence $*$ *sequence*, PLASMA_request $*$ *request*)

PLASMA_dgesv_Tile_Async - Solves a system of linear equations using the tile LU factorization. Non-blocking equivalent of PLASMA_dgesv_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dgesv
PLASMA_dgesv_Tile
PLASMA_cgesv_Tile_Async
PLASMA_dgesv_Tile_Async
PLASMA_sgesv_Tile_Async
PLASMA_dcgesv_Tile_Async

3.11.2.8 int PLASMA_dgetrf_Tile_Async (PLASMA_desc $*$ *A*, PLASMA_desc $*$ *L*, int $*$ *IPIV*, PLASMA_sequence $*$ *sequence*, PLASMA_request $*$ *request*)

PLASMA_dgetrf_Tile_Async - Computes the tile LU factorization of a matrix. Non-blocking equivalent of PLASMA_dgetrf_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dgetrf

PLASMA_dgetrf_Tile
PLASMA_cgetrf_Tile_Async
PLASMA_dgetrf_Tile_Async
PLASMA_sgetrf_Tile_Async
PLASMA_dgetrs_Tile_Async

3.11.2.9 int PLASMA_dgetrs_Tile_Async (PLASMA_desc * *A*, PLASMA_desc * *L*, int * *IPIV*, PLASMA_desc * *B*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_dgetrs_Tile_Async - Solves a system of linear equations using previously computed LU factorization. Non-blocking equivalent of PLASMA_dgetrs_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dgetrs
PLASMA_dgetrs_Tile
PLASMA_cgetrs_Tile_Async
PLASMA_dgetrs_Tile_Async
PLASMA_sgetrs_Tile_Async
PLASMA_dgetrf_Tile_Async

3.11.2.10 int PLASMA_dlange_Tile_Async (PLASMA_enum *norm*, PLASMA_desc * *A*, double * *work*, double * *value*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_dlange_Tile_Async - Non-blocking equivalent of PLASMA_dlange_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dlange
PLASMA_dlange_Tile
PLASMA_clange_Tile_Async
PLASMA_dlange_Tile_Async
PLASMA_slange_Tile_Async

3.11.2.11 int PLASMA_dlansy_Tile_Async (PLASMA_enum *norm*, PLASMA_enum *uplo*, PLASMA_desc * *A*, double * *work*, double * *value*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_dlansy_Tile_Async - Non-blocking equivalent of PLASMA_dlansy_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dlansy
PLASMA_dlansy_Tile
PLASMA_clansy_Tile_Async
PLASMA_dlansy_Tile_Async
PLASMA_slansy_Tile_Async

3.11.2.12 int PLASMA_dLapack_to_Tile_Async (double * *Af77*, int *LDA*, PLASMA_desc * *A*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_dLapack_to_Tile_Async - Conversion from LAPACK layout to tile layout. Non-blocking equivalent of PLASMA_dLapack_to_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dTile_to_Lapack_Async
PLASMA_dLapack_to_Tile
PLASMA_cLapack_to_Tile_Async
PLASMA_dLapack_to_Tile_Async
PLASMA_sLapack_to_Tile_Async

3.11.2.13 int PLASMA_dlauum_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc * *A*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_dlauum_Tile_Async - Computes the product U * U' or L' * L, where the triangular factor U or L is stored in the upper or lower triangular part of the array A. Non-blocking equivalent of PLASMA_dlauum_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dlauum
PLASMA_dlauum_Tile
PLASMA_clauum_Tile_Async
PLASMA_dlauum_Tile_Async
PLASMA_slauum_Tile_Async
PLASMA_dpotri_Tile_Async

3.11.2.14 int PLASMA_dorglq_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

Non-blocking equivalent of PLASMA_dorglq_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dorglq
PLASMA_dorglq_Tile
PLASMA_cunglq_Tile_Async
PLASMA_dunglq_Tile_Async
PLASMA_sunglq_Tile_Async
PLASMA_dgelqf_Tile_Async

3.11.2.15 int PLASMA_dorgqr_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *Q*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

Non-blocking equivalent of PLASMA_dorgqr_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dorgqr
PLASMA_dorgqr_Tile
PLASMA_cungqr_Tile_Async
PLASMA_dungqr_Tile_Async
PLASMA_sungqr_Tile_Async
PLASMA_dgeqrf_Tile_Async

3.11.2.16 int PLASMA_dormlq_Tile_Async (PLASMA_enum *side*, PLASMA_enum *trans*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

Non-blocking equivalent of PLASMA_dormlq_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dormlq
PLASMA_dormlq_Tile
PLASMA_cunmlq_Tile_Async
PLASMA_dunmlq_Tile_Async
PLASMA_sunmlq_Tile_Async
PLASMA_dgelqf_Tile_Async

3.11.2.17 int PLASMA_dormqr_Tile_Async (PLASMA_enum *side*, PLASMA_enum *trans*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

Non-blocking equivalent of PLASMA_dormqr_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dormqr
PLASMA_dormqr_Tile
PLASMA_cunmqr_Tile_Async
PLASMA_dunmqr_Tile_Async
PLASMA_sunmqr_Tile_Async
PLASMA_dgeqrf_Tile_Async

3.11.2.18 int PLASMA_dplgsy_Tile_Async (double *bump*, PLASMA_desc ∗ *A*, unsigned long long int *seed*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_dplgsy_Tile_Async - Generate a random hermitian matrix by tiles. Non-blocking equivalent of PLASMA_dplgsy_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dplgsy
PLASMA_dplgsy_Tile
PLASMA_cplgsy_Tile_Async
PLASMA_dplgsy_Tile_Async
PLASMA_splgsy_Tile_Async
PLASMA_dplgsy_Tile_Async
PLASMA_dplgsy_Tile_Async

3.11.2.19 int PLASMA_dplrnt_Tile_Async (PLASMA_desc ∗ *A*, unsigned long long int *seed*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_dplrnt_Tile_Async - Generate a random matrix by tiles. Non-blocking equivalent of PLASMA_dplrnt_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dplrnt
PLASMA_dplrnt_Tile
PLASMA_cplrnt_Tile_Async
PLASMA_dplrnt_Tile_Async
PLASMA_splrnt_Tile_Async
PLASMA_dplgsy_Tile_Async
PLASMA_dplgsy_Tile_Async

3.11.2.20 int PLASMA_dposv_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_dposv_Tile_Async - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization. Non-blocking equivalent of PLASMA_dposv_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dposv
PLASMA_dposv_Tile
PLASMA_cposv_Tile_Async
PLASMA_dposv_Tile_Async
PLASMA_sposv_Tile_Async

3.11.2.21 int PLASMA_dpotrf_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_dpotrf_Tile_Async - Computes the Cholesky factorization of a symmetric positive definite or Hermitian positive definite matrix. Non-blocking equivalent of PLASMA_dpotrf_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dpotrf
PLASMA_dpotrf_Tile
PLASMA_cpotrf_Tile_Async
PLASMA_dpotrf_Tile_Async
PLASMA_spotrf_Tile_Async
PLASMA_dpotrs_Tile_Async

3.11.2.22 int PLASMA_dpotri_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_dpotri_Tile_Async - Computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization A = U\∗\∗T∗U or A = L∗L\∗\∗T computed by PLASMA_dpotrf. Non-blocking equivalent of PLASMA_dpotri_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dpotri
PLASMA_dpotri_Tile
PLASMA_cpotri_Tile_Async
PLASMA_dpotri_Tile_Async
PLASMA_spotri_Tile_Async
PLASMA_dpotrf_Tile_Async

3.11.2.23 int PLASMA_dpotrs_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_dpotrs_Tile_Async - Solves a system of linear equations using previously computed Cholesky factorization. Non-blocking equivalent of PLASMA_dpotrs_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dpotrs
PLASMA_dpotrs_Tile
PLASMA_cpotrs_Tile_Async
PLASMA_dpotrs_Tile_Async
PLASMA_spotrs_Tile_Async
PLASMA_dpotrf_Tile_Async

3.11.2.24 int PLASMA_dsgesv_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*, PLASMA_desc ∗ *B*, PLASMA_desc ∗ *X*, int ∗ *ITER*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_dsgesv_Tile_Async - Solves a system of linear equations using the tile LU factorization and mixed-precision iterative refinement. Non-blocking equivalent of PLASMA_dsgesv_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dsgesv
PLASMA_dsgesv_Tile
PLASMA_dsgesv_Tile_Async
PLASMA_dgesv_Tile_Async

3.11.2.25 int PLASMA_dsposv_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_desc ∗ *X*, int ∗ *ITER*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_dsposv_Tile_Async - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization and mixed-precision iterative refinement. Non-blocking equivalent of PLASMA_dsposv_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dposv
PLASMA_dposv_Tile
PLASMA_dposv_Tile_Async
PLASMA_zposv_Tile_Async

3.11.2.26 int PLASMA_dsungesv_Tile_Async (PLASMA_enum *trans*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_desc ∗ *X*, int ∗ *ITER*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_dsungesv_Tile_Async - Solves symmetric linear system of equations using the tile QR or the tile LQ factorization and mixed-precision iterative refinement. Non-blocking equivalent of PLASMA_dsungesv_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dsungesv
PLASMA_dsungesv_Tile
PLASMA_dsungesv_Tile_Async
PLASMA_zgels_Tile_Async

3.11.2.27 int PLASMA_dsymm_Tile_Async (PLASMA_enum *side*, PLASMA_enum *uplo*, double *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, double *beta*, PLASMA_desc ∗ *C*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_dsymm_Tile_Async - Performs symmetric matrix multiplication. Non-blocking equivalent of PLASMA_dsymm_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dsymm
PLASMA_dsymm_Tile

PLASMA_csymm_Tile_Async
PLASMA_dsymm_Tile_Async
PLASMA_ssymm_Tile_Async

3.11.2.28 int PLASMA_dsyr2k_Tile_Async (PLASMA_enum *uplo*, PLASMA_enum *trans*, double *alpha*, PLASMA_desc $*$ *A*, PLASMA_desc $*$ *B*, double *beta*, PLASMA_desc $*$ *C*, PLASMA_sequence $*$ *sequence*, PLASMA_request $*$ *request*)

PLASMA_dsyr2k_Tile_Async - Performs symmetric rank-k update. Non-blocking equivalent of PLASMA_dsyr2k_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dsyr2k
PLASMA_dsyr2k_Tile
PLASMA_csyr2k_Tile_Async
PLASMA_dsyr2k_Tile_Async
PLASMA_ssyr2k_Tile_Async

3.11.2.29 int PLASMA_dsyrk_Tile_Async (PLASMA_enum *uplo*, PLASMA_enum *trans*, double *alpha*, PLASMA_desc $*$ *A*, double *beta*, PLASMA_desc $*$ *C*, PLASMA_sequence $*$ *sequence*, PLASMA_request $*$ *request*)

PLASMA_dsyrk_Tile_Async - Performs rank-k update. Non-blocking equivalent of PLASMA_dsyrk_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dsyrk
PLASMA_dsyrk_Tile
PLASMA_csyrk_Tile_Async
PLASMA_dsyrk_Tile_Async
PLASMA_ssyrk_Tile_Async

3.11.2.30 int PLASMA_dTile_to_Lapack_Async (PLASMA_desc $*$ *A*, double $*$ *Af77*, int *LDA*, PLASMA_sequence $*$ *sequence*, PLASMA_request $*$ *request*)

PLASMA_dTile_to_Lapack_Async - Conversion from LAPACK layout to tile layout. Non-blocking equivalent of PLASMA_dTile_to_Lapack(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dLapack_to_Tile_Async
PLASMA_dTile_to_Lapack
PLASMA_cTile_to_Lapack_Async
PLASMA_dTile_to_Lapack_Async
PLASMA_sTile_to_Lapack_Async

3.11.2.31 int PLASMA_dtrmm_Tile_Async (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, double *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_dtrmm_Tile_Async - Performs triangular matrix multiplication. Non-blocking equivalent of PLASMA_dtrmm_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dtrmm
PLASMA_dtrmm_Tile
PLASMA_ctrmm_Tile_Async
PLASMA_dtrmm_Tile_Async
PLASMA_strmm_Tile_Async

3.11.2.32 int PLASMA_dtrsm_Tile_Async (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, double *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_dtrsm_Tile_Async - Computes triangular solve. Non-blocking equivalent of PLASMA_dtrsm_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dtrsm
PLASMA_dtrsm_Tile

PLASMA_ctrsm_Tile_Async
PLASMA_dtrsm_Tile_Async
PLASMA_strsm_Tile_Async

3.11.2.33 int PLASMA_dtrsmpl_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_dtrsmpl_Tile - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix. Non-blocking equivalent of PLASMA_dtrsmpl_Tile(). Returns control to the user thread before worker threads finish the computation to allow for pipelined execution of diferent routines.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dtrsmpl
PLASMA_dtrsmpl_Tile
PLASMA_ctrsmpl_Tile_Async
PLASMA_dtrsmpl_Tile_Async
PLASMA_strsmpl_Tile_Async
PLASMA_dgetrf_Tile_Async

3.11.2.34 int PLASMA_dtrtri_Tile_Async (PLASMA_enum *uplo*, PLASMA_enum *diag*, PLASMA_desc ∗ *A*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_dtrtri_Tile_Async - Computes the inverse of a complex upper or lower triangular matrix A. Non-blocking equivalent of PLASMA_dtrtri_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_dtrtri
PLASMA_dtrtri_Tile
PLASMA_ctrtri_Tile_Async
PLASMA_dtrtri_Tile_Async
PLASMA_strtri_Tile_Async
PLASMA_dpotri_Tile_Async

3.12 Advanced Interface: Asynchronous - Single Real

Functions/Subroutines

- int PLASMA_sgelqf_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_sgelqs_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_sgels_Tile_Async (PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_sgemm_Tile_Async (PLASMA_enum transA, PLASMA_enum transB, float alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, float beta, PLASMA_desc ∗C, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_sgeqrf_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_sgeqrs_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_sgesv_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_sgetrf_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_sgetrs_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_slange_Tile_Async (PLASMA_enum norm, PLASMA_desc ∗A, float ∗work, float ∗value, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_slansy_Tile_Async (PLASMA_enum norm, PLASMA_enum uplo, PLASMA_desc ∗A, float ∗work, float ∗value, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_slauum_Tile_Async (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_sorglq_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_sorgqr_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗Q, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_sormlq_Tile_Async (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_sormqr_Tile_Async (PLASMA_enum side, PLASMA_enum trans, PLASMA_desc ∗A, PLASMA_desc ∗T, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_splgsy_Tile_Async (float bump, PLASMA_desc ∗A, unsigned long long int seed, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_splrnt_Tile_Async (PLASMA_desc ∗A, unsigned long long int seed, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_sposv_Tile_Async (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_spotrf_Tile_Async (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_spotri_Tile_Async (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_spotrs_Tile_Async (PLASMA_enum uplo, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)

- int PLASMA_ssymm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, float alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, float beta, PLASMA_desc ∗C, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_ssyr2k_Tile_Async (PLASMA_enum uplo, PLASMA_enum trans, float alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, float beta, PLASMA_desc ∗C, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_ssyrk_Tile_Async (PLASMA_enum uplo, PLASMA_enum trans, float alpha, PLASMA_desc ∗A, float beta, PLASMA_desc ∗C, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_strmm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, float alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_strsm_Tile_Async (PLASMA_enum side, PLASMA_enum uplo, PLASMA_enum transA, PLASMA_enum diag, float alpha, PLASMA_desc ∗A, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_strsmpl_Tile_Async (PLASMA_desc ∗A, PLASMA_desc ∗L, int ∗IPIV, PLASMA_desc ∗B, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_strtri_Tile_Async (PLASMA_enum uplo, PLASMA_enum diag, PLASMA_desc ∗A, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_sLapack_to_Tile_Async (float ∗Af77, int LDA, PLASMA_desc ∗A, PLASMA_sequence ∗sequence, PLASMA_request ∗request)
- int PLASMA_sTile_to_Lapack_Async (PLASMA_desc ∗A, float ∗Af77, int LDA, PLASMA_sequence ∗sequence, PLASMA_request ∗request)

3.12.1 Detailed Description

This is the group of single real functions using the advanced asynchronous interface.

3.12.2 Function/Subroutine Documentation

3.12.2.1 int PLASMA_sgelqf_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_sgelqf_Tile_Async - Computes the tile LQ factorization of a matrix. Non-blocking equivalent of PLASMA_sgelqf_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_sgelqf
PLASMA_sgelqf_Tile
PLASMA_cgelqf_Tile_Async
PLASMA_dgelqf_Tile_Async
PLASMA_sgelqf_Tile_Async
PLASMA_sgelqs_Tile_Async

3.12.2.2 int PLASMA_sgelqs_Tile_Async (PLASMA_desc * *A*, PLASMA_desc * *T*, PLASMA_desc * *B*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_sgelqs_Tile_Async - Computes a minimum-norm solution using previously computed LQ factorization. Non-blocking equivalent of PLASMA_sgelqs_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_sgelqs
PLASMA_sgelqs_Tile
PLASMA_cgelqs_Tile_Async
PLASMA_dgelqs_Tile_Async
PLASMA_sgelqs_Tile_Async
PLASMA_sgelqf_Tile_Async

3.12.2.3 int PLASMA_sgels_Tile_Async (PLASMA_enum *trans*, PLASMA_desc * *A*, PLASMA_desc * *T*, PLASMA_desc * *B*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_sgels_Tile_Async - Solves overdetermined or underdetermined linear system of equations using the tile QR or the tile LQ factorization. Non-blocking equivalent of PLASMA_sgels_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_sgels
PLASMA_sgels_Tile
PLASMA_cgels_Tile_Async
PLASMA_dgels_Tile_Async
PLASMA_sgels_Tile_Async

3.12.2.4 int PLASMA_sgemm_Tile_Async (PLASMA_enum *transA*, PLASMA_enum *transB*, float *alpha*, PLASMA_desc * *A*, PLASMA_desc * *B*, float *beta*, PLASMA_desc * *C*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_sgemm_Tile_Async - Performs matrix multiplication. Non-blocking equivalent of PLASMA_sgemm_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_sgemm
PLASMA_sgemm_Tile
PLASMA_cgemm_Tile_Async
PLASMA_dgemm_Tile_Async
PLASMA_sgemm_Tile_Async

3.12.2.5 int PLASMA_sgeqrf_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_sgeqrf_Tile_Async - Computes the tile QR factorization of a matrix. Non-blocking equivalent of PLASMA_sgeqrf_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_sgeqrf
PLASMA_sgeqrf_Tile
PLASMA_cgeqrf_Tile_Async
PLASMA_dgeqrf_Tile_Async
PLASMA_sgeqrf_Tile_Async
PLASMA_sgeqrs_Tile_Async

3.12.2.6 int PLASMA_sgeqrs_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_sgeqrs_Tile_Async - Computes a minimum-norm solution using the tile QR factorization. Non-blocking equivalent of PLASMA_sgeqrs_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_sgeqrs

PLASMA_sgeqrs_Tile
PLASMA_cgeqrs_Tile_Async
PLASMA_dgeqrs_Tile_Async
PLASMA_sgeqrs_Tile_Async
PLASMA_sgeqrf_Tile_Async

3.12.2.7 int PLASMA_sgesv_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_sgesv_Tile_Async - Solves a system of linear equations using the tile LU factorization. Non-blocking equivalent of PLASMA_sgesv_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_sgesv
PLASMA_sgesv_Tile
PLASMA_cgesv_Tile_Async
PLASMA_dgesv_Tile_Async
PLASMA_sgesv_Tile_Async
PLASMA_scgesv_Tile_Async

3.12.2.8 int PLASMA_sgetrf_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_sgetrf_Tile_Async - Computes the tile LU factorization of a matrix. Non-blocking equivalent of PLASMA_sgetrf_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_sgetrf
PLASMA_sgetrf_Tile
PLASMA_cgetrf_Tile_Async
PLASMA_dgetrf_Tile_Async
PLASMA_sgetrf_Tile_Async
PLASMA_sgetrs_Tile_Async

3.12.2.9 int PLASMA_sgetrs_Tile_Async (PLASMA_desc * *A*, PLASMA_desc * *L*, int * *IPIV*, PLASMA_desc * *B*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_sgetrs_Tile_Async - Solves a system of linear equations using previously computed LU factorization. Non-blocking equivalent of PLASMA_sgetrs_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_sgetrs
PLASMA_sgetrs_Tile
PLASMA_cgetrs_Tile_Async
PLASMA_dgetrs_Tile_Async
PLASMA_sgetrs_Tile_Async
PLASMA_sgetrf_Tile_Async

3.12.2.10 int PLASMA_slange_Tile_Async (PLASMA_enum *norm*, PLASMA_desc * *A*, float * *work*, float * *value*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_slange_Tile_Async - Non-blocking equivalent of PLASMA_slange_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_slange
PLASMA_slange_Tile
PLASMA_clange_Tile_Async
PLASMA_dlange_Tile_Async
PLASMA_slange_Tile_Async

3.12.2.11 int PLASMA_slansy_Tile_Async (PLASMA_enum *norm*, PLASMA_enum *uplo*, PLASMA_desc * *A*, float * *work*, float * *value*, PLASMA_sequence * *sequence*, PLASMA_request * *request*)

PLASMA_slansy_Tile_Async - Non-blocking equivalent of PLASMA_slansy_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_slansy
PLASMA_slansy_Tile
PLASMA_clansy_Tile_Async
PLASMA_dlansy_Tile_Async
PLASMA_slansy_Tile_Async

3.12.2.12 int PLASMA_sLapack_to_Tile_Async (float ∗ *Af77*, int *LDA*, PLASMA_desc ∗ *A*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_sLapack_to_Tile_Async - Conversion from LAPACK layout to tile layout. Non-blocking equivalent of PLASMA_sLapack_to_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_sTile_to_Lapack_Async
PLASMA_sLapack_to_Tile
PLASMA_cLapack_to_Tile_Async
PLASMA_dLapack_to_Tile_Async
PLASMA_sLapack_to_Tile_Async

3.12.2.13 int PLASMA_slauum_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_slauum_Tile_Async - Computes the product U ∗ U' or L' ∗ L, where the triangular factor U or L is stored in the upper or lower triangular part of the array A. Non-blocking equivalent of PLASMA_slauum_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_slauum
PLASMA_slauum_Tile
PLASMA_clauum_Tile_Async
PLASMA_dlauum_Tile_Async
PLASMA_slauum_Tile_Async
PLASMA_spotri_Tile_Async

3.12.2.14 int PLASMA_sorglq_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

Non-blocking equivalent of PLASMA_sorglq_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_sorglq
PLASMA_sorglq_Tile
PLASMA_cunglq_Tile_Async
PLASMA_dunglq_Tile_Async
PLASMA_sunglq_Tile_Async
PLASMA_sgelqf_Tile_Async

3.12.2.15 int PLASMA_sorgqr_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *Q*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

Non-blocking equivalent of PLASMA_sorgqr_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_sorgqr
PLASMA_sorgqr_Tile
PLASMA_cungqr_Tile_Async
PLASMA_dungqr_Tile_Async
PLASMA_sungqr_Tile_Async
PLASMA_sgeqrf_Tile_Async

3.12.2.16 int PLASMA_sormlq_Tile_Async (PLASMA_enum *side*, PLASMA_enum *trans*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

Non-blocking equivalent of PLASMA_sormlq_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_sormlq
PLASMA_sormlq_Tile
PLASMA_cunmlq_Tile_Async
PLASMA_dunmlq_Tile_Async
PLASMA_sunmlq_Tile_Async
PLASMA_sgelqf_Tile_Async

3.12.2.17 int PLASMA_sormqr_Tile_Async (PLASMA_enum *side*, PLASMA_enum *trans*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *T*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

Non-blocking equivalent of PLASMA_sormqr_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_sormqr
PLASMA_sormqr_Tile
PLASMA_cunmqr_Tile_Async
PLASMA_dunmqr_Tile_Async
PLASMA_sunmqr_Tile_Async
PLASMA_sgeqrf_Tile_Async

3.12.2.18 int PLASMA_splgsy_Tile_Async (float *bump*, PLASMA_desc ∗ *A*, unsigned long long int *seed*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_splgsy_Tile_Async - Generate a random hermitian matrix by tiles. Non-blocking equivalent of PLASMA_splgsy_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_splgsy
PLASMA_splgsy_Tile
PLASMA_cplgsy_Tile_Async
PLASMA_dplgsy_Tile_Async

PLASMA_splgsy_Tile_Async
PLASMA_splgsy_Tile_Async
PLASMA_splgsy_Tile_Async

3.12.2.19 int PLASMA_splrnt_Tile_Async (PLASMA_desc ∗ *A*, unsigned long long int *seed*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_splrnt_Tile_Async - Generate a random matrix by tiles. Non-blocking equivalent of PLASMA_splrnt_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_splrnt
PLASMA_splrnt_Tile
PLASMA_cplrnt_Tile_Async
PLASMA_dplrnt_Tile_Async
PLASMA_splrnt_Tile_Async
PLASMA_splgsy_Tile_Async
PLASMA_splgsy_Tile_Async

3.12.2.20 int PLASMA_sposv_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_sposv_Tile_Async - Solves a symmetric positive definite or Hermitian positive definite system of linear equations using the Cholesky factorization. Non-blocking equivalent of PLASMA_sposv_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_sposv
PLASMA_sposv_Tile
PLASMA_cposv_Tile_Async
PLASMA_dposv_Tile_Async
PLASMA_sposv_Tile_Async

3.12.2.21 int PLASMA_spotrf_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_spotrf_Tile_Async - Computes the Cholesky factorization of a symmetric positive definite or Hermitian positive definite matrix. Non-blocking equivalent of PLASMA_spotrf_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_spotrf
PLASMA_spotrf_Tile
PLASMA_cpotrf_Tile_Async
PLASMA_dpotrf_Tile_Async
PLASMA_spotrf_Tile_Async
PLASMA_spotrs_Tile_Async

3.12.2.22 int PLASMA_spotri_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_spotri_Tile_Async - Computes the inverse of a complex Hermitian positive definite matrix A using the Cholesky factorization A = U\∗\∗T∗U or A = L∗L\∗\∗T computed by PLASMA_spotrf. Non-blocking equivalent of PLASMA_spotri_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_spotri
PLASMA_spotri_Tile
PLASMA_cpotri_Tile_Async
PLASMA_dpotri_Tile_Async
PLASMA_spotri_Tile_Async
PLASMA_spotrf_Tile_Async

3.12.2.23 int PLASMA_spotrs_Tile_Async (PLASMA_enum *uplo*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_spotrs_Tile_Async - Solves a system of linear equations using previously computed Cholesky factorization. Non-blocking equivalent of PLASMA_spotrs_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_spotrs
PLASMA_spotrs_Tile
PLASMA_cpotrs_Tile_Async
PLASMA_dpotrs_Tile_Async
PLASMA_spotrs_Tile_Async
PLASMA_spotrf_Tile_Async

3.12.2.24 int PLASMA_ssymm_Tile_Async (PLASMA_enum *side*, PLASMA_enum *uplo*, float *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, float *beta*, PLASMA_desc ∗ *C*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_ssymm_Tile_Async - Performs symmetric matrix multiplication. Non-blocking equivalent of PLASMA_ssymm_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_ssymm
PLASMA_ssymm_Tile
PLASMA_csymm_Tile_Async
PLASMA_dsymm_Tile_Async
PLASMA_ssymm_Tile_Async

3.12.2.25 int PLASMA_ssyr2k_Tile_Async (PLASMA_enum *uplo*, PLASMA_enum *trans*, float *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, float *beta*, PLASMA_desc ∗ *C*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_ssyr2k_Tile_Async - Performs symmetric rank-k update. Non-blocking equivalent of PLASMA_ssyr2k_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_ssyr2k
PLASMA_ssyr2k_Tile
PLASMA_csyr2k_Tile_Async
PLASMA_dsyr2k_Tile_Async
PLASMA_ssyr2k_Tile_Async

3.12.2.26 int PLASMA_ssyrk_Tile_Async (PLASMA_enum *uplo*, PLASMA_enum *trans*, float *alpha*, PLASMA_desc ∗ *A*, float *beta*, PLASMA_desc ∗ *C*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_ssyrk_Tile_Async - Performs rank-k update. Non-blocking equivalent of PLASMA_ssyrk_-Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_ssyrk
PLASMA_ssyrk_Tile
PLASMA_csyrk_Tile_Async
PLASMA_dsyrk_Tile_Async
PLASMA_ssyrk_Tile_Async

3.12.2.27 int PLASMA_sTile_to_Lapack_Async (PLASMA_desc ∗ *A*, float ∗ *Af77*, int *LDA*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_sTile_to_Lapack_Async - Conversion from LAPACK layout to tile layout. Non-blocking equivalent of PLASMA_sTile_to_Lapack(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_sLapack_to_Tile_Async
PLASMA_sTile_to_Lapack
PLASMA_cTile_to_Lapack_Async
PLASMA_dTile_to_Lapack_Async
PLASMA_sTile_to_Lapack_Async

3.12.2.28 int PLASMA_strmm_Tile_Async (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, float *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_strmm_Tile_Async - Performs triangular matrix multiplication. Non-blocking equivalent of PLASMA_strmm_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_strmm
PLASMA_strmm_Tile
PLASMA_ctrmm_Tile_Async
PLASMA_dtrmm_Tile_Async
PLASMA_strmm_Tile_Async

3.12.2.29 int PLASMA_strsm_Tile_Async (PLASMA_enum *side*, PLASMA_enum *uplo*, PLASMA_enum *transA*, PLASMA_enum *diag*, float *alpha*, PLASMA_desc ∗ *A*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_strsm_Tile_Async - Computes triangular solve. Non-blocking equivalent of PLASMA_strsm_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_strsm
PLASMA_strsm_Tile
PLASMA_ctrsm_Tile_Async
PLASMA_dtrsm_Tile_Async
PLASMA_strsm_Tile_Async

3.12.2.30 int PLASMA_strsmpl_Tile_Async (PLASMA_desc ∗ *A*, PLASMA_desc ∗ *L*, int ∗ *IPIV*, PLASMA_desc ∗ *B*, PLASMA_sequence ∗ *sequence*, PLASMA_request ∗ *request*)

PLASMA_strsmpl_Tile - Performs the forward substitution step of solving a system of linear equations after the tile LU factorization of the matrix. Non-blocking equivalent of PLASMA_strsmpl_Tile(). Returns control to the user thread before worker threads finish the computation to allow for pipelined execution of diferent routines.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_strsmpl
PLASMA_strsmpl_Tile
PLASMA_ctrsmpl_Tile_Async
PLASMA_dtrsmpl_Tile_Async
PLASMA_strsmpl_Tile_Async
PLASMA_sgetrf_Tile_Async

3.12.2.31 int PLASMA_strtri_Tile_Async (PLASMA_enum *uplo*, PLASMA_enum *diag*, PLASMA_desc $*$ *A*, PLASMA_sequence $*$ *sequence*, PLASMA_request $*$ *request*)

PLASMA_strtri_Tile_Async - Computes the inverse of a complex upper or lower triangular matrix A. Non-blocking equivalent of PLASMA_strtri_Tile(). May return before the computation is finished. Allows for pipelining of operations ar runtime.

Parameters

← ***sequence*** Identifies the sequence of function calls that this call belongs to (for completion checks and exception handling purposes).

→ ***request*** Identifies this function call (for exception handling purposes).

See also

PLASMA_strtri
PLASMA_strtri_Tile
PLASMA_ctrtri_Tile_Async
PLASMA_dtrtri_Tile_Async
PLASMA_strtri_Tile_Async
PLASMA_spotri_Tile_Async

Chapter 4

Data Type Documentation

4.1 plasma_context_map_struct Type Reference

`#include <context.h>`

Data Fields

- pthread_t **thread_id**
- plasma_context_t * **context**

4.1.1 Detailed Description

Threads contexts map

4.2 plasma_context_struct Type Reference

```
#include <context.h>
```

Data Fields

- PLASMA_bool **initialized**
- int **world_size**
- int **group_size**
- int **thread_bind** [CONTEXT_THREADS_MAX]
- int **thread_rank** [CONTEXT_THREADS_MAX]
- pthread_attr_t **thread_attr**
- pthread_t **thread_id** [CONTEXT_THREADS_MAX]
- pthread_mutex_t **action_mutex**
- pthread_cond_t **action_condt**
- volatile PLASMA_enum **action**
- void(∗ **parallel_func_ptr**)(struct plasma_context_struct ∗)
- unsigned char **args_buff** [ARGS_BUFF_SIZE]
- PLASMA_bool **errors_enabled**
- PLASMA_bool **warnings_enabled**
- PLASMA_bool **autotuning_enabled**
- PLASMA_bool **dynamic_section**
- PLASMA_enum **scheduling**
- PLASMA_enum **householder**
- PLASMA_enum **translation**
- int **nb**
- int **ib**
- int **nbnbsize**
- int **ibnbsize**
- int **info**
- int **iinfo**
- int **rhblock**
- int volatile **barrier_id**
- int volatile **barrier_nblocked_thrds**
- pthread_mutex_t **barrier_synclock**
- pthread_cond_t **barrier_synccond**
- int **ss_ld**
- volatile int **ss_abort**
- volatile int ∗ **ss_progress**
- Quark ∗ **quark**

4.2.1 Detailed Description

PLASMA context

4.3 primedec Type Reference

Data Fields

- int **p**
- int **e**
- int **pe**

4.4 pthread_cond_s Type Reference

Data Fields

- HANDLE **hSem**
- HANDLE **hEvt**
- CRITICAL_SECTION **cs**
- int **waitCount**

4.5 pthread_s Type Reference

Data Fields

- HANDLE **hThread**
- unsigned int **uThId**

Index