LASs - Linear Algebra Routines on OmpSs

1.0.0

Generated by Doxygen 1.8.11

Contents

Chapter 1

File Index

1.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 2

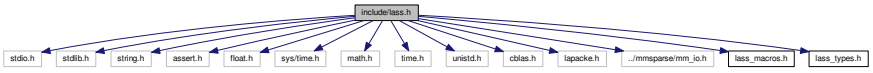
File Documentation

2.1 include/lass.h File Reference

LASs header definition.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <float.h>
#include <sys/time.h>
#include <math.h>
#include <time.h>
#include <unistd.h>
#include <cblas.h>
#include <lapacke.h>
#include "../mmsparse/mm_io.h"
#include "lass_macros.h"
#include "lass_types.h"
```

Include dependency graph for lass.h:



This graph shows which files directly or indirectly include this file:

Functions

- int ddss_dgemm (enum DDSS_TRANS TRANS_A, enum DDSS_TRANS TRANS_B, int M, int N, int K, double ALPHA, double ∗A, int LDA, double ∗B, int LDB, double BETA, double ∗C, int LDC)

- int ddss_dsymm (enum DDSS_SIDE SIDE, enum DDSS_UPLO UPLO, int M, int N, double ALPHA, double *A, int LDA, double *B, int LDB, double BETA, double *C, int LDC)
- int ddss_dtrsm (enum DDSS_SIDE SIDE, enum DDSS_UPLO UPLO, enum DDSS_TRANS TRANS_A, enum DDSS_DIAG DIAG, int M, int N, const double ALPHA, double *A, int LDA, double *B, int LDB)
- int ddss_dtrmm (enum DDSS_SIDE SIDE, enum DDSS_UPLO UPLO, enum DDSS_TRANS TRANS_A, enum DDSS_DIAG DIAG, int M, int N, const double ALPHA, double *A, int LDA, double *B, int LDB)
- int ddss_dsyrk (enum DDSS_UPLO UPLO, enum DDSS_TRANS TRANS_A, int N, int K, const double AL↩ PHA, double *A, int LDA, const double BETA, double *C, int LDC)
- int ddss_dsyr2k (enum DDSS_UPLO UPLO, enum DDSS_TRANS TRANS, int N, int K, const double ALPHA, double *A, int LDA, double *B, int LDB, const double BETA, double *C, int LDC)
- int ddss_dnpgetrf (int M, int N, double *A, int LDA)
- int ddss_dnpgesv (int N, int NRHS, double *A, int LDA, double *B, int LDB)
- int ddss_dtpgetrf (int M, int N, double *A, int LDA, int *ipiv)
- int ddss_dtpgesv (int N, int NRHS, double *A, int LDA, int *IPIV, double *B, int LDB)
- int ddss_dpotrf (enum DDSS_UPLO UPLO, int N, double *A, int LDA)
- int ddss_dposv (enum DDSS_UPLO UPLO, int N, int NRHS, double *A, int LDA, double *B, int LDB)
- enum LASS_RETURN kdgemm (enum DDSS_TRANS TRANS_A, enum DDSS_TRANS TRANS_B, int M, int N, int K, const double ALPHA, double *A, int LDA, double *B, int LDB, const double BETA, double *C, int LDC)
- enum LASS_RETURN kdsymm (enum DDSS_SIDE SIDE, enum DDSS_UPLO UPLO, int M, int N, const double ALPHA, double *A, int LDA, double *B, int LDB, const double BETA, double *C, int LDC)
- enum LASS_RETURN kdtrsm (enum DDSS_SIDE SIDE, enum DDSS_UPLO UPLO, enum DDSS_TRANS TRANS_A, enum DDSS_DIAG DIAG, int M, int N, const double ALPHA, double *A, int LDA, double *B, int LDB)
- enum LASS_RETURN kdtrmm (enum DDSS_SIDE SIDE, enum DDSS_UPLO UPLO, enum DDSS_TRANS TRANS_A, enum DDSS_DIAG DIAG, int M, int N, const double ALPHA, double *A, int LDA, double *B, int LDB)
- enum LASS_RETURN kdsyrk (enum DDSS_UPLO UPLO, enum DDSS_TRANS TRANS_A, int N, int K, const double ALPHA, double *A, int LDA, const double BETA, double *C, int LDC)
- enum LASS_RETURN kdsyr2k (enum DDSS_UPLO UPLO, enum DDSS_TRANS TRANS, int N, int K, const double ALPHA, double *A, int LDA, double *B, int LDB, const double BETA, double *C, int LDC)
- enum LASS_RETURN kdnpgetrf (int M, int N, double *A, int LDA)
- enum LASS_RETURN kdnpgesv (int N, int NRHS, double *A, int LDA, double *B, int LDB)
- enum LASS_RETURN kdtpgetrf (int M, int N, double *A, int LDA, int *ipiv)
- enum LASS_RETURN kdtpgesv (int N, int NRHS, double *A, int LDA, int *IPIV, double *B, int LDB)
- enum LASS_RETURN kdpotrf (enum DDSS_UPLO UPLO, int N, double *A, int LDA)
- enum LASS_RETURN kdposv (enum DDSS_UPLO UPLO, int N, int NRHS, double *A, int LDA, double *B, int LDB)
- enum LASS_RETURN dnpgetrf (int M, int N, double *A, int LDA)
- int ddss_tile_size (int M, int MT)
- void ddss_dflat2tiled (int M, int N, double *A, int LDA, int MT, int NT, double(*TILE_A)[NT][TILE_SIZE *TI↩ LE_SIZE])
- void ddss_dsymtiled2flat (int M, int N, double *A, int LDA, int MT, int NT, double(*TILE_A)[NT][TILE_SIZE *TILE_SIZE], enum DDSS_UPLO UPLO)
- void ddss_dgather_tile (int M, int N, double *A, int LDA, double *TILE_A, int MID, int NID)
- void ddss_dtiled2flat (int M, int N, double *A, int LDA, int MT, int NT, double(*TILE_A)[NT][TILE_SIZE *TI↩ LE_SIZE])
- void ddss_dtiled2flat_nb (int M, int N, double *A, int LDA, int MT, int NT, double(*TILE_A)[NT][TILE_SIZE *TILE_SIZE])
- void ddss_dsymflat2tiled (int M, int N, double *A, int LDA, int MT, int NT, double(*TILE_A)[NT][TILE_SIZE *TILE_SIZE], enum DDSS_UPLO UPLO)
- void ddss_dsymtiled2flat_nb (int M, int N, double *A, int LDA, int MT, int NT, double(*TILE_A)[NT][TILE_SIZE *TILE_SIZE], enum DDSS_UPLO UPLO)
- void ddss_dscatter_tile (int M, int N, double *A, int LDA, double *TILE_A, int MID, int NID)
- int **dsss_dgtsv** (int N, double *DL, double *D, double *DU, double *RHS)

- int **dsss_dspmv** (int M, int N, double ALPHA, const double *VAL_A, const int *ROW_PTR_A, const int *COL_IND_A, const double *X, double BETA, double *Y)
- enum LASS_RETURN **kdgtsv** (int N, double *DL, double *D, double *DU, double *RHS)
- enum LASS_RETURN **kdspmv** (int M, int N, double ALPHA, const double *VAL_A, const int *ROW_PTR_A, const int *COL_IND_A, const double *X, double BETA, double *Y)
- void **dgtsvseq** (int N, double *DL, double *D, double *DU, double *RHS)
- void dspmvseq (int M, int N, double ALPHA, const double *VAL_A, const int *ROW_PTR_A, const int *COL_IND_A, const double *X, double BETA, double *Y)

2.1.1 Detailed Description

LASs header definition.

LASs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es

Date

2017-09-07

2.1.2 Function Documentation

2.1.2.1 void ddss_dflat2tiled (int *M*, int *N*, double * *A*, int *LDA*, int *MT*, int *NT*, double(*) *TILE_A[NT][TILE_SIZE*TILE_SIZE]*)

ddss_dflat2tiled: Performs the change of the data layout from flat layout to tiled layout according to row-major order.

Parameters

in	*M*	int. M specifies the number of rows of the flat matrix.
in	*N*	int. N specifies the number of columns of the flat matrix.
in	*A*	double *. A is a pointer to the flat matrix.
in	*LDA*	int. LDA specifies the number of columns (row-major order) of matrix A.
in	*MT*	int. MT specifies the number of rows of the matrix TILE_A.
in	*NT*	int. NT specifies the number of columns of the matrix TILE_A.
in,out	*TILE_A*	double *. TILE_A is a pointer to the tile matrix.

See also

ddss_dgather_tile
ddss_tile_size

Definition at line 68 of file ddss_flat2tiled.c.

References ddss_dgather_tile().

Referenced by kdgemm(), kdnpgesv(), kdnpgetrf(), kdposv(), kdsymm(), kdsyr2k(), kdsyrk(), kdtpgesv(), kdtpgetrf(), kdtrmm(), and kdtrsm().

```
70 {
71
72     // Local variables
73     int m, n;
74
75     for ( m = 0; m < MT; m++ )
76     {
77         for ( n = 0; n < NT; n++ )
78         {
79             #pragma oss task inout(TILE_A[m][n]) \
80             label( dflat2tiled )
81             {
82                 ddss_dgather_tile( M, N, &A[m * TILE_SIZE * N + n * TILE_SIZE],
83                     LDA, TILE_A[m][n], m, n );
84             }
85         }
86     }
87
88 }
```

2.1.2.2 void ddss_dgather_tile (int *M*, int *N*, double * *A*, int *LDA*, double * *TILE_A*, int *MID*, int *NID*)

ddss_dgather_tile: Performs the copy of a tile from the flat matrix A to the tile matrix TILE_A for the MT, NT tile.

Parameters

in	*M*	int. M specifies the number of rows of the flat matrix.
in	*N*	int. N specifies the number of columns of the flat matrix.
in	*A*	double *. A is a pointer to the flat matrix.
in	*LDA*	int. LDA specifies the number of columns (row-major order) of matrix A.
in,out	*TILE_A*	double *. TILE_A is a pointer to the tile matrix.
in	*MID*	int. MID specifies the row id of the tile.
in	*NID*	int. NID specifies the column id of the tile.

See also

ddss_tile_size
ddss_dflat2tiled

Definition at line 238 of file ddss_flat2tiled.c.

References ddss_tile_size().

Referenced by ddss_dflat2tiled(), and ddss_dsymflat2tiled().

```
240 {
241
242     //Local variables
243     int i, j;
244     int tile_size_m, tile_size_n;
245
246     tile_size_m = ddss_tile_size( M, MID );
247     tile_size_n = ddss_tile_size( N, NID );
248
249     for ( i = 0; i < tile_size_m; i++ )
250     {
251         for ( j = 0; j < tile_size_n; j++ )
252         {
253             TILE_A[i * tile_size_n + j] = A[i * LDA + j];
254         }
255     }
256
257 }
```

2.1.2.3 int ddss_dgemm (enum DDSS_TRANS *TRANS_A,* enum DDSS_TRANS *TRANS_B,* int *M,* int *N,* int *K,* double *ALPHA,* double * *A,* int *LDA,* double * *B,* int *LDB,* double *BETA,* double * *C,* int *LDC*)

Performs the matrix-matrix operation:

```
C = ALPHA * op( A ) * op( B ) + BETA * C
```

where op(X) is one of:

```
op( X ) = X      or
op( X ) = X**T
```

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

in	*TRANS_A*	enum DDSS_TRANS. TRANS_A specifies the form of op(A) to be used in the matrix multiplication as follows: • NoTrans: op(A) = A. • Trans: op(A) = A**T.
in	*TRANS_B*	enum DDSS_TRANS. TRANS_B specifies the form of op(B) to be used in the matrix multiplication as follows: • NoTrans: op(B) = B. • Trans: op(B) = B**T.
in	*M*	int. M specifies the number of rows of the matrix A and the number of rows of the matrix C. M must be greater than zero.
in	*N*	int. N specifies the number of columns of the matrix B and the number of columns of the matrix C. N must be greater than zero.
in	*K*	int. K specifies the number of columns of the matrix A and the number of rows of the matrix B. K must be greater than zero.
in	*ALPHA*	double.
in	*A*	double *. A is a pointer to a matrix of dimension Ma (rows) by Ka (columns), where Ma is M and Ka is K when TRANS_A = NoTrans, and Ma is K and Ka is M otherwise.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). When TRANS_A = NoTrans then LDA must be at least max(1, K), otherwise LDA must be at least max(1, M).
in	*B*	double *. B is a pointer to a matrix of dimension Kb (rows) by Nb (columns), where Kb is K and Nb is N when TRANS_B = NoTrans, and Kb is N and Nb is K otherwise.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). When TRANS_B = NoTrans then LDB must be at least max(1, N), otherwise LDB must be at least max(1, K).
in	*BETA*	double.
in,out	*C*	double *. C is a pointer to a matrix of dimension M by N. On exit, C is overwritten by the M by N matrix (ALPHA*op(A)*op(B) + BETA*C).
in	*LDC*	int. LDC specifies the number of columns of C (row-major order). LDC must be at least max(1, N).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

kdgemm

Definition at line 128 of file ddss_dgemm.c.

References kdgemm().

```
133 {
134
135     // Local variables
136     int An, Bn;
137
138     // Argument checking
139     if ( ( TRANS_A != NoTrans ) && ( TRANS_A != Trans ) )
140     {
141         fprintf( stderr, "Illegal value of TRANS_A, in ddss_dgemm code\n" );
142         return NoSuccess;
143     }
144
145     if ( ( TRANS_B != NoTrans ) && ( TRANS_B != Trans ) )
146     {
147         fprintf( stderr, "Illegal value of TRANS_B, in ddss_dgemm code\n" );
148         return NoSuccess;
149     }
150
151     if ( M < 0 )
152     {
153         fprintf( stderr, "Illegal value of M, in ddss_dgemm code\n" );
154         return NoSuccess;
155     }
156
157     if ( N < 0 )
158     {
159         fprintf( stderr, "Illegal value of N, in ddss_dgemm code\n" );
160         return NoSuccess;
161     }
162
163     if ( K < 0 )
164     {
165         fprintf( stderr, "Illegal value of K, in ddss_dgemm code\n" );
166         return NoSuccess;
167     }
168
169     if ( TRANS_A == NoTrans )
170     {
171         An = K;
172     }
173     else
174     {
175         An = M;
176     }
177
178     if ( LDA < MAX( 1, An ) )
179     {
180         fprintf( stderr, "Illegal value of LDA, in ddss_dgemm code\n" );
181         return NoSuccess;
182     }
183
184     if ( TRANS_B == NoTrans )
185     {
186         Bn = N;
187     }
188     else
189     {
190         Bn = K;
191     }
192
193     if ( LDB < MAX( 1, Bn ) )
194     {
195         fprintf( stderr, "Illegal value of LDB, in ddss_dgemm code\n" );
196         return NoSuccess;
```

```
197     }
198
199     if ( LDC < MAX( 1, N ) )
200     {
201         fprintf( stderr, "Illegal value of LDC, in ddss_dgemm code\n" );
202         return NoSuccess;
203     }
204
205     // Quick return
206     if ( M == 0 || N == 0 || ( ( ALPHA == 0.0 || K == 0 ) && BETA == 1.0 ) )
207     {
208         return Success;
209     }
210
211     return kdgemm( TRANS_A, TRANS_B, M, N, K,
212                    (const double) ALPHA, A, LDA,
213                                          B, LDB,
214                    (const double)  BETA, C, LDC );
215
216 }
```

2.1.2.4 int ddss_dnpgesv (int *N*, int *NRHS*, double * *A*, int *LDA*, double * *B*, int *LDB*)

Solves a system of linear equations A X = B, where A is a N-by-N general matrix and X and B are N-by-NRHS matrices. The matrix A is factorized using the LU descomposition without pivoting. The matrix A is descomposed as:

```
A = L * U
```

where L is a lower triangular matrix with unit diagonal elements and U is an upper triangular matrix.

Parameters

in	*N*	int. N specifies the order of the square matrix A. N >= 0.
in	*NRHS*	int. NRHS specifies the number of right-hand-sides (number of columns of B). NRHS >= 0.
in,out	*A*	double *. A is a pointer to a regular matrix of dimension N-by-LDA. On exit, if return value is Success, the matrix A is overwriten by the factors L and U. The unit diagonal elements of L are not stored.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).
in,out	*B*	double *. B is a pointer to a matrix of dimension N by NRHS, which stores the right-hand-sides of the systems of linear equations. (row-major order). On exit, if return value is Success, the matrix B is overwriten by the solution matrix X.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, NRHS).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

kdnpgesv

Definition at line 84 of file ddss_dnpgesv.c.

References kdnpgesv().

```
87 {
88
89     // Argument checking
90     if ( N < 0 )
91     {
92         fprintf( stderr, "Illegal value of N, in ddss_dnpgesv code\n" );
93         return NoSuccess;
94     }
95
96     if ( NRHS < 0 )
97     {
98         fprintf( stderr, "Illegal value of NRHS, in ddss_dnpgesv code\n" );
99         return NoSuccess;
100     }
101
102     if ( LDA < MAX( 1, N ) )
103     {
104         fprintf( stderr, "Illegal value of LDA, in ddss_dnpgesv code\n" );
105         return NoSuccess;
106     }
107
108     if ( LDB < MAX( 1, NRHS ) )
109     {
110         fprintf( stderr, "Illegal value of LDB, in ddss_dnpgesv code\n" );
111         return NoSuccess;
112     }
113
114     // Quick return
115     if ( MAX( N, 0 ) == 0 || MAX( NRHS, 0 ) == 0 )
116     {
117         return Success;
118     }
119
120     return kdnpgesv( N, NRHS, A, LDA, B, LDB );
121
122 }
```

2.1.2.5 int ddss_dnpgetrf (int *M,* int *N,* double * *A,* int *LDA*)

Performs the LU factorization without pivoting of a general M-by-N matrix A:

```
A = L * U
```

where L is a lower triangular (lower trapezoidal if M > N) matrix with unit diagonal elements and U is an upper triangular (upper trapezoidal if M < N) matrix.

Parameters

in	*M*	int. M specifies the number of rows of the matrix A. M >= 0.
in	*N*	int. N specifies the number of columns of the matrix A. N >= 0.
in,out	*A*	double *. A is a pointer to a regular matrix of dimension M-by-N. On exit, if return value is Success, the matrix A is overwriten by the factors L and U. The unit diagonal elements of L are not stored.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

knpdgetrf

Definition at line 70 of file ddss_dnpgetrf.c.

References kdnpgetrf().

```
71 {
72
73     // Argument checking
74     if ( M < 0 )
75     {
76         fprintf( stderr, "Illegal value of M, in ddss_dnpgetrf code\n" );
77         return NoSuccess;
78     }
79
80     if ( N < 0 )
81     {
82         fprintf( stderr, "Illegal value of N, in ddss_dnpgetrf code\n" );
83         return NoSuccess;
84     }
85
86     if ( LDA < MAX( 1, N ) )
87     {
88         fprintf( stderr, "Illegal value of LDA, in ddss_dnpgetrf code\n" );
89         return NoSuccess;
90     }
91
92     // Qick return
93     if ( MIN( M, N ) == 0 )
94     {
95         return Success;
96     }
97
98     return kdnpgetrf( M, N, A, LDA );
99
100 }
```

2.1.2.6 int ddss_dposv (enum DDSS_UPLO *UPLO*, int *N*, int *NRHS*, double * *A*, int *LDA*, double * *B*, int *LDB*)

Solves a system of linear equations A X = B, where A is a m-by-m simmetric positive definite matrix and X and B are m-by-nrhs matrices. The matrix A is descomposed as:

```
A = L \times L^T
or
A = U^T \times U
```

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

Parameters

in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form of A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.
in	*N*	int. N specifies the order of the square matrix A. N >= 0.
in	*NRHS*	int. NRHS specifies the number of right-hand-sides (number of columns of B). NRHS >= 0.
in,out	*A*	double *. A is a pointer to a positive definite matrix of dimension N by LDA. On exit, if return value is Success, the matrix A is overwriten by the factor U or L.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).
in,out	*B*	double *. B is a pointer to a matrix of dimension N by NRHS, which stores the right-hand-sides of the systems of linear equations. (row-major order). On exit, if return value is Success, the matrix B is overwriten by the solution matrix X.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, NRHS).

Return values

Success	sucessful exit
NoSuccess	unsucessful exit

See also

kdposv

Definition at line 92 of file ddss_dposv.c.

References kdposv().

```
96 {
97
98     // Argument checking
99     if ( ( UPLO != Upper ) && ( UPLO != Lower ) )
100     {
101         fprintf( stderr, "Illegal value of UPLO, in ddss_dposv code\n" );
102         return NoSuccess;
103     }
104
105     if ( N < 0 )
106     {
107         fprintf( stderr, "Illegal value of N, in ddss_dposv code\n" );
108         return NoSuccess;
109     }
110
111     if ( NRHS < 0 )
112     {
113         fprintf( stderr, "Illegal value of NRHS, in ddss_dposv code\n" );
114         return NoSuccess;
115     }
116
117     if ( LDA < MAX( 1, N ) )
118     {
119         fprintf( stderr, "Illegal value of LDA, in ddss_dposv code\n" );
120         return NoSuccess;
121     }
122
123     if ( LDB < MAX( 1, NRHS ) )
124     {
125         fprintf( stderr, "Illegal value of LDB, in ddss_dposv code\n" );
126         return NoSuccess;
127     }
128
129     // Quick return
130     if ( MAX( N, 0 ) == 0 || MAX( NRHS, 0 ) == 0 )
131     {
132         return Success;
133     }
134
135     return kdposv( UPLO, N, NRHS, A, LDA, B, LDB );
136
137 }
```

2.1.2.7 int ddss_dpotrf (enum DDSS_UPLO *UPLO*, int *N*, double * *A*, int *LDA*)

Performs the Cholesky factorization of a symmetric positive definite matrix A:

```
A = L \times L^T
or
A = U^T \times U
```

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

Parameters

in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form of A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.
in	*N*	int. N specifies the order of the square matrix A. N >= 0.
in,out	*A*	double *. A is a pointer to a positive definite matrix of dimension N by LDA. On exit, if return value is Success, the matrix A is overwriten by the factor U or L.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

kdpotrf

Definition at line 74 of file ddss_dpotrf.c.

References kdpotrf().

```
75 {
76
77     // Argument checking
78     if ( ( UPLO != Upper ) && ( UPLO != Lower ) )
79     {
80         fprintf( stderr, "Illegal value of UPLO, in ddss_dportf code\n" );
81         return NoSuccess;
82     }
83
84     if ( N < 0 )
85     {
86         fprintf( stderr, "Illegal value of N, in ddss_dportf code\n" );
87         return NoSuccess;
88     }
89
90     if ( LDA < MAX( 1, N ) )
91     {
92         fprintf( stderr, "Illegal value of LDA, in ddss_dportf code\n" );
93         return NoSuccess;
94     }
95
96     // Quick return
97     if ( MAX( N, 0 ) == 0 )
98     {
99         return Success;
100     }
101
102     return kdpotrf( UPLO, N, A, LDA );
103
104 }
```

2.1.2.8 void ddss_dscatter_tile (int *M*, int *N*, double * *A*, int *LDA*, double * *TILE_A*, int *MID*, int *NID*)

ddss_dscatter_tile: Performs the copy of a tile from the tile matrix TILE_A to the flat matrix A for the MT, NT tile.

Parameters

in	*M*	int. M specifies the number of rows of the flat matrix.
in	*N*	int. N specifies the number of columns of the flat matrix.
in,out	*A*	double ∗. A is a pointer to the flat matrix.
in	*LDA*	int. LDA specifies the number of columns (row-major order) of matrix A.
in	*TILE_A*	double ∗. TILE_A is a pointer to the tile matrix.
in	*MID*	int. MID specifies the row id of the tile.
in	*NID*	int. NID specifies the column id of the tile.

See also

ddss_tile_size
ddss_dtiled2flat

Definition at line 414 of file ddss_tiled2flat.c.

References ddss_tile_size().

Referenced by ddss_dsymtiled2flat(), ddss_dsymtiled2flat_nb(), ddss_dtiled2flat(), and ddss_dtiled2flat_nb().

```
416 {
417
418     //Local variables
419     int i, j;
420     int tile_size_m, tile_size_n;
421
422     tile_size_m = ddss_tile_size( M, MID );
423     tile_size_n = ddss_tile_size( N, NID );
424
425     for ( i = 0; i < tile_size_m; i++ )
426     {
427         for ( j = 0; j < tile_size_n; j++ )
428         {
429             A[i * LDA + j] = TILE_A[i * tile_size_n + j];
430         }
431     }
432
433 }
```

2.1.2.9 void ddss_dsymflat2tiled (int *M*, int *N*, double ∗ *A*, int *LDA*, int *MT*, int *NT*, double(∗) *TILE_A[NT][TILE_SIZE ∗TILE_SIZE]*, enum DDSS_UPLO *UPLO*)

ddss_dsymflat2tiled: Performs the change of the data layout from flat layout to tiled layout for symmetric matrices according to row-major order.

Parameters

in	*M*	int. M specifies the number of rows of the flat matrix.
in	*N*	int. N specifies the number of columns of the flat matrix.
in	*A*	double ∗. A is a pointer to the flat matrix.
in	*LDA*	int. LDA specifies the number of columns (row-major order) of matrix A.
in	*MT*	int. MT specifies the number of rows of the matrix TILE_A.
in	*NT*	int. NT specifies the number of columns of the matrix TILE_A.
in,out	*TILE_A*	double ∗. TILE_A is a pointer to the tile matrix.
in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form of A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.

See also

ddss_dgather_tile
ddss_tile_size

Definition at line 147 of file ddss_flat2tiled.c.

References ddss_dgather_tile().

Referenced by kdposv(), kdpotrf(), kdsymm(), kdsyr2k(), kdsyrk(), kdtrmm(), and kdtrsm().

```
149 {
150
151     // Local variables
152     int m, n;
153
154     if ( UPLO == Upper )
155     {
156         for ( m = 0; m < MT; m++ )
157         {
158             for ( n = NT-1; n >= m; n-- )
159             {
160                 #pragma oss task inout(TILE_A[m][n]) \
161                 label( dsymmflat2tiled )
162                 {
163                     ddss_dgather_tile( M, N,
164                         &A[m * TILE_SIZE * N + n * TILE_SIZE],
165                         LDA, TILE_A[m][n], m, n );
166                 }
167             }
168         }
169     }
170     else if ( UPLO == Lower )
171     {
172         for ( m = 0; m < MT; m++ )
173         {
174             for ( n = 0; n <= m; n++ )
175             {
176                 #pragma oss task inout(TILE_A[m][n]) \
177                 label( dsymmflat2tiled )
178                 {
179                     ddss_dgather_tile( M, N,
180                         &A[m * TILE_SIZE * N + n * TILE_SIZE],
181                         LDA, TILE_A[m][n], m, n );
182                 }
183             }
184         }
185     }
186
187 }
```

2.1.2.10 int ddss_dsymm (enum DDSS_SIDE *SIDE*, enum DDSS_UPLO *UPLO*, int *M*, int *N*, double *ALPHA*, double * *A*, int *LDA*, double * *B*, int *LDB*, double *BETA*, double * *C*, int *LDC*)

Performs one of the matrix-matrix operations:

```
C = ALPHA * A * B + BETA * C
```

or

```
C = ALPHA * B * A + BETA * C
```

where op(X) is one of:

```
op( X ) = X      or
op( X ) = X**T
```

ALPHA and BETA are scalars, A is a symmetric matrix, and B and C are M by N matrices.

Parameters

in	*SIDE*	enum DDSS_SIDE. UPLO specifies the position of the symmetric A matrix in the operation: • Left: C = ALPHA ∗ A ∗ B + BETA ∗ C • Right: C = ALPHA ∗ B ∗ A + BETA ∗ C
in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form of A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.
in	*M*	int. M specifies the number of rows of the matrix C. M must be equal or greater than zero.
in	*N*	int. N specifies the number of columns of the matrix C. N must be equal or greater than zero.
in	*ALPHA*	double.
in	*A*	double ∗. A is a pointer to a matrix of dimension Ma (rows) by Na (columns), where Ma is M and Na is M when SIDE = Left, and Ma is N and Na is N when SIDE = Right
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, Na).
in	*B*	double ∗. B is a pointer to a matrix of dimension M by N.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, N).
in	*BETA*	double.
in,out	*C*	double ∗. C is a pointer to a matrix of dimension M by N. On exit, C is overwritten by the M by N matrix.
in	*LDC*	int. LDC specifies the number of columns of C (row-major order). LDC must be at least max(1, N).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

kdsymm

Definition at line 120 of file ddss_dsymm.c.

References kdsymm().

```
125 {
126
127     // Local variables
128     int nA;
129
130     // Argument checking
131     if ( ( SIDE != Left ) && ( SIDE != Right ) )
132     {
133         fprintf( stderr, "Illegal value of SIDE, in ddss_dsymm code\n" );
134         return NoSuccess;
135     }
136
137     if ( ( UPLO != Upper ) && ( UPLO != Lower ) )
138     {
139         fprintf( stderr, "Illegal value of UPLO, in ddss_dsymm code\n" );
```

```
140         return NoSuccess;
141     }
142
143     if ( M < 0 )
144     {
145         fprintf( stderr, "Illegal value of M, in ddss_dsymm code\n" );
146         return NoSuccess;
147     }
148
149     if ( N < 0 )
150     {
151         fprintf( stderr, "Illegal value of N, in ddss_dsymm code\n" );
152         return NoSuccess;
153     }
154
155     // Quick return
156     if ( M == 0 || N == 0 || ( ALPHA == 0.0 && BETA == 1.0 ) )
157     {
158         return Success;
159     }
160
161     if ( SIDE == Left )
162     {
163         nA = M;
164     }
165     else
166     {
167         nA = N;
168     }
169
170     if ( LDA < MAX( 1, nA ) )
171     {
172         fprintf( stderr, "Illegal value of LDA, in ddss_dsymm code\n" );
173         return NoSuccess;
174     }
175
176     if ( LDB < MAX( 1, N ) )
177     {
178         fprintf( stderr, "Illegal value of LDB, in ddss_dsymm code\n" );
179         return NoSuccess;
180     }
181
182     if ( LDC < MAX( 1, N ) )
183     {
184         fprintf( stderr, "Illegal value of LDC, in ddss_dsymm code\n" );
185         return NoSuccess;
186     }
187
188     return kdsymm( SIDE, UPLO, M, N,
189                    ( const double ) ALPHA, A, LDA,
190                                             B, LDB,
191                    ( const double )  BETA, C, LDC );
192
193 }
```

2.1.2.11 void ddss_dsymtiled2flat (int *M*, int *N*, double * *A*, int *LDA*, int *MT*, int *NT*, double(*) *TILE_A[NT][TILE_SIZE *TILE_SIZE]*, enum DDSS_UPLO *UPLO*)

ddss_dsymtiled2flat: Performs the change of the data layout from tile layout to flat layout for symmetric matrices according to row-major order.

Parameters

in	*M*	int. M specifies the number of rows of the flat matrix.
in	*N*	int. N specifies the number of columns of the flat matrix.
in,out	*A*	double *. A is a pointer to the flat matrix.
in	*LDA*	int. LDA specifies the number of columns (row-major order) of matrix A.
in	*MT*	int. MT specifies the number of rows of the matrix TILE_A.
in	*NT*	int. NT specifies the number of columns of the matrix TILE_A.
in	*TILE_A*	double *. TILE_A is a pointer to the tile matrix.

Parameters

in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form of A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.

See also

ddss_dscatter_tile
ddss_tile_size

Definition at line 223 of file ddss_tiled2flat.c.

References ddss_dscatter_tile().

Referenced by kdpotrf(), kdsyr2k(), and kdsyrk().

```
225 {
226
227     // Local variables
228     int m, n;
229
230     if ( UPLO == Upper )
231     {
232         for ( m = 0; m < MT; m++ )
233         {
234             for ( n = NT-1; n >= m; n-- )
235             {
236                 #pragma oss task inout(TILE_A[m][n]) \
237                 label( dsymmtiled2flat )
238                 {
239                     ddss_dscatter_tile( M, N,
240                         &A[m * TILE_SIZE * N + n * TILE_SIZE], LDA,
241                         TILE_A[m][n], m, n );
242                 }
243             }
244         }
245     }
246     else if ( UPLO == Lower )
247     {
248         for ( m = 0; m < MT; m++ )
249         {
250             for ( n = 0; n <= m; n++ )
251             {
252                 #pragma oss task inout(TILE_A[m][n]) \
253                 label( dsymmtiled2flat )
254                 {
255                     ddss_dscatter_tile( M, N,
256                         &A[m * TILE_SIZE * N + n * TILE_SIZE], LDA,
257                         TILE_A[m][n], m, n );
258                 }
259             }
260         }
261     }
262
263     #pragma oss taskwait
264 }
```

2.1.2.12 void ddss_dsymtiled2flat_nb (int *M*, int *N*, double * *A*, int *LDA*, int *MT*, int *NT*, double(*) *TILE_A[NT][TILE_SIZE *TILE_SIZE]*, enum DDSS_UPLO *UPLO*)

ddss_dsymtiled2flat_nb: Performs the change of the data layout from tile layout to flat layout for symmetric matrices according to row-major order in a non-blocking execution mode.

Parameters

in	*M*	int. M specifies the number of rows of the flat matrix.
in	*N*	int. N specifies the number of columns of the flat matrix.
in,out	*A*	double *. A is a pointer to the flat matrix.
in	*LDA*	int. LDA specifies the number of columns (row-major order) of matrix A.
in	*MT*	int. MT specifies the number of rows of the matrix TILE_A.
in	*NT*	int. NT specifies the number of columns of the matrix TILE_A.
in	*TILE_A*	double *. TILE_A is a pointer to the tile matrix.
in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form of A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.

See also

ddss_dscatter_tile
ddss_tile_size

Definition at line 324 of file ddss_tiled2flat.c.

References ddss_dscatter_tile().

Referenced by kdposv().

```
326 {
327
328     // Local variables
329     int m, n;
330
331     if ( UPLO == Upper )
332     {
333         for ( m = 0; m < MT; m++ )
334         {
335             for ( n = NT-1; n >= m; n-- )
336             {
337                 #pragma oss task inout(TILE_A[m][n]) \
338                 label( dsymmtiled2flat_nb )
339                 {
340                     ddss_dscatter_tile( M, N,
341                         &A[m * TILE_SIZE * N + n * TILE_SIZE], LDA,
342                         TILE_A[m][n], m, n );
343                 }
344             }
345         }
346     }
347     else if ( UPLO == Lower )
348     {
349         for ( m = 0; m < MT; m++ )
350         {
351             for ( n = 0; n <= m; n++ )
352             {
353                 #pragma oss task inout(TILE_A[m][n]) \
354                 label( dsymmtiled2flat_nb )
355                 {
356                     ddss_dscatter_tile( M, N,
357                         &A[m * TILE_SIZE * N + n * TILE_SIZE], LDA,
358                         TILE_A[m][n], m, n );
359                 }
360             }
361         }
362     }
363
364 }
```

2.1.2.13 int ddss_dsyr2k (enum DDSS_UPLO *UPLO*, enum DDSS_TRANS *TRANS*, int *N*, int *K*, const double *ALPHA*, double * *A*, int *LDA*, double * *B*, int *LDB*, const double *BETA*, double * *C*, int *LDC*)

Performs one of the symmetric rank 2k operations:

```
C = ALPHA * A * B**T + ALPHA * B * A**T + BETA * C or
C = ALPHA * A**T * B + ALPHA * B**T * A + BETA * C
```

ALPHA and BETA are scalars, C is an N by N symmetric matrix and A and B are N by K matrices in the first case and K by N matrices in the second case.

Parameters

in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form in which C is stored: • Lower: Lower triangular part of C is stored. The upper traingular part is not referenced. • Upper: Upper triangular part of C is stored. The lower triangular part is not referenced.
in	*TRANS*	enum DDSS_TRANS. TRANS specifies the operation to be performed as follows: • NoTrans: C = ALPHA * A * B**T + ALPHA * B * A**T + BETA * C • Trans: C = ALPHA * A**T * B + ALPHA * B**T * A + BETA * C
in	*N*	int. N specifies the order of matrix C. N must be at least zero.
in	*K*	int. With TRANS = NoTrans, K specifies the number of columns of the matrices A and B, and with TRANS = Trans, K specifies the number of rows of the matrices A and B. K must be at least zero.
in	*ALPHA*	double. ALPHA specifies the scalar alpha.
in	*A*	double *. A is a pointer to a matrix of dimension Na (rows) by Ka (columns), where Na is N and Ka is K when TRANS = NoTrans, and Na is K and Ka is N otherwise.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). When TRANS = NoTrans then LDA must be at least max(1, K), otherwise LDA must be at least max(1, N).
in	*B*	double *. B is a pointer to a matrix of dimension Nb (rows) by Kb (columns), where Nb is N and Kb is K when TRANS = NoTrans, and Nb is K and Kb is N otherwise.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). When TRANS = NoTrans then LDB must be at least max(1, K), otherwise LDB must be at least max(1, N).
in	*BETA*	double. BETA specifies the scalar beta.
in,out	*C*	double *. C is a pointer to a matrix of dimension N by N. When UPLO = Uppper the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of C is overwritten by the upper triangular part of the updated solution matrix C. When UPLO = Lower the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of C is overwritten by the lower triangular part of the updated solution matrix C.
in	*LDC*	int. LDC specifies the number of columns of C (row-major order). LDC must be at least max(1, N).

See also

kdsyr2k

Definition at line 114 of file ddss_dsyr2k.c.

References kdsyr2k().

```
119 {
120     // Local variables
121     int nA, nB;
122
123     // Argument checking
124     if ( ( UPLO != Upper ) && ( UPLO != Lower ) )
125     {
126         fprintf( stderr, "Illegal value of UPLO, in ddss_dsyr2k code\n" );
127         return NoSuccess;
128     }
129
130     if ( ( TRANS != NoTrans ) && ( TRANS != Trans ) )
131     {
132         fprintf( stderr, "Illegal value of TRANS, in ddss_dsyr2k code\n" );
133         return NoSuccess;
134     }
135
136     if ( N < 0 )
137     {
138         fprintf( stderr, "Illegal value of N, in ddss_dsyr2k code\n" );
139         return NoSuccess;
140     }
141
142     if ( K < 0 )
143     {
144         fprintf( stderr, "Illegal value of K, in ddss_dsyr2k code\n" );
145         return NoSuccess;
146     }
147
148     if ( TRANS == NoTrans )
149     {
150         nA = nB = K;
151     }
152     else
153     {
154         nA = nB = N;
155     }
156
157     if ( LDA < MAX( 1, nA ) )
158     {
159         fprintf( stderr, "Illegal value of LDA, in ddss_dsyr2k code\n" );
160         return NoSuccess;
161     }
162
163     if ( LDB < MAX( 1, nB ) )
164     {
165         fprintf( stderr, "Illegal value of LDB, in ddss_dsyr2k code\n" );
166         return NoSuccess;
167     }
168
169     if ( LDC < MAX( 1, N ) )
170     {
171         fprintf( stderr, "Illegal value of LDC, in ddss_dsyr2k code\n" );
172         return NoSuccess;
173     }
174
175     // Quick return
176      if ( N == 0 || ( ( ALPHA == 0.0 || K == 0 ) && BETA == 1.0 ) )
177      {
178          return Success;
179      }
180
181     return kdsyr2k( UPLO, TRANS,
182                     N, K,
183                     ALPHA, A, LDA,
184                            B, LDB,
185                     BETA,  C, LDC );
186
187 }
```

2.1.2.14 int ddss_dsyrk (enum DDSS_UPLO *UPLO*, enum DDSS_TRANS *TRANS_A*, int *N*, int *K*, const double *ALPHA*, double * *A*, int *LDA*, const double *BETA*, double * *C*, int *LDC*)

Performs one of the symmetric rank k operations:

```
C = ALPHA * A * op( A ) + BETA * C  or
C = ALPHA * op( A ) * A + BETA * C
```

where op(X) is:

```
op( X ) = X**T
```

ALPHA and BETA are scalars, C is an N by N symmetric matrix and A is an N by K matrix in the first case and a K by N matrix in the second case.

Parameters

in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form in which C is stored: • Lower: Lower triangular part of C is stored. The upper traingular part is not referenced. • Upper: Upper triangular part of C is stored. The lower triangular part is not referenced.
in	*TRANS_A*	enum DDSS_TRANS. TRANS_A specifies the operation to be performed as follows: • NoTrans: C = ALPHA ∗ A ∗ A**T + BETA ∗ C • Trans: C = ALPHA ∗ A**T ∗ A + BETA ∗ C
in	*N*	int. N specifies the order of matrix C. N must be at least zero.
in	*K*	int. With TRANS_A = NoTrans, K specifies the number of columns of the matrix A, and with TRANS_A = Trans, K specifies the number of rows of the matrix A. K must be at least zero.
in	*ALPHA*	double. ALPHA specifies the scalar alpha.
in	*A*	double ∗. A is a pointer to a matrix of dimension Na (rows) by Ka (columns), where Na is N and Ka is K when TRANS_A = NoTrans, and Na is K and Ka is N otherwise.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). When TRANS_A = NoTrans then LDA must be at least max(1, K), otherwise LDA must be at least max(1, N).
in	*BETA*	double. BETA specifies the scalar beta.
in,out	*C*	double ∗. C is a pointer to a matrix of dimension N by N. When UPLO = Uppper the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of C is overwritten by the upper triangular part of the updated solution matrix C. When UPLO = Lower the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of C is overwritten by the lower triangular part of the updated solution matrix C.
in	*LDC*	int. LDC specifies the number of columns of C (row-major order). LDC must be at least max(1, N).

See also

kdsyrk

Definition at line 106 of file ddss_dsyrk.c.

References kdsyrk().

```
110 {
111
112     // Local variables
113     int nA;
114
115     // Argument checking
116     if ( ( UPLO != Upper ) && ( UPLO != Lower ) )
117     {
118         fprintf( stderr, "Illegal value of UPLO, in ddss_dsyrk code\n" );
119         return NoSuccess;
120     }
121
122     if ( ( TRANS_A != NoTrans ) && ( TRANS_A != Trans ) )
123     {
124         fprintf( stderr, "Illegal value of TRANS_A, in ddss_dsyrk code\n" );
125         return NoSuccess;
126     }
127
128     if ( N < 0 )
129     {
130         fprintf( stderr, "Illegal value of N, in ddss_dsyrk code\n" );
131         return NoSuccess;
132     }
133
134     if ( K < 0 )
135     {
136         fprintf( stderr, "Illegal value of K, in ddss_dsyrk code\n" );
137         return NoSuccess;
138     }
139
140     if ( TRANS_A == NoTrans )
141     {
142         nA = K;
143     }
144     else
145     {
146         nA = N;
147     }
148
149     if ( LDA < MAX( 1, nA ) )
150     {
151         fprintf( stderr, "Illegal value of LDA, in ddss_dsyrk code\n" );
152         return NoSuccess;
153     }
154
155     if ( LDC < MAX( 1, N ) )
156     {
157         fprintf( stderr, "Illegal value of LDC, in ddss_dsyrk code\n" );
158         return NoSuccess;
159     }
160
161     // Quick return
162      if ( N == 0 || ( ( ALPHA == 0.0 || K == 0 ) && BETA == 1.0 ) )
163      {
164          return Success;
165      }
166
167     return kdsyrk( UPLO, TRANS_A,
168                    N, K,
169                    ALPHA, A, LDA,
170                    BETA,  C, LDC );
171
172 }
```

2.1.2.15 void ddss_dtiled2flat (int *M*, int *N*, double * *A*, int *LDA*, int *MT*, int *NT*, double(*) *TILE_A[NT][TILE_SIZE*TILE_SIZE]*)

ddss_dtiled2flat: Performs the change of the data layout from tile layout to flat layout according to row-major order.

Parameters

in	*M*	int. M specifies the number of rows of the flat matrix.
in	*N*	int. N specifies the number of columns of the flat matrix.
in,out	*A*	double *. A is a pointer to the flat matrix.
in	*LDA*	int. LDA specifies the number of columns (row-major order) of matrix A.
in	*MT*	int. MT specifies the number of rows of the matrix TILE_A.

Parameters

in	*NT*	int. NT specifies the number of columns of the matrix TILE_A.
in	*TILE↩ _A*	double *. TILE_A is a pointer to the tile matrix.

See also

ddss_dscatter_tile
ddss_tile_size

Definition at line 68 of file ddss_tiled2flat.c.

References ddss_dscatter_tile().

Referenced by kdgemm(), kdnpgesv(), kdnpgetrf(), kdposv(), kdsymm(), kdtpgesv(), kdtpgetrf(), kdtrmm(), and kdtrsm().

```
70 {
71
72     // Local variables
73     int m, n;
74
75     for ( m = 0; m < MT; m++ )
76     {
77         for ( n = 0; n < NT; n++ )
78         {
79             #pragma oss task inout(TILE_A[m][n]) \
80             label( dtiled2flat)
81             {
82                 ddss_dscatter_tile( M, N,
83                     &A[m * TILE_SIZE * N + n * TILE_SIZE], LDA,
84                     TILE_A[m][n], m, n );
85             }
86         }
87     }
88
89     #pragma oss taskwait
90
91 }
```

2.1.2.16 void ddss_dtiled2flat_nb (int *M*, int *N*, double * *A*, int *LDA*, int *MT*, int *NT*, double(*) *TILE_A[NT][TILE_SIZE *TILE_SIZE]*)

ddss_dtiled2flat_nb: Performs the change of the data layout from tile layout to flat layout according to row-major order in a non-blocking execution mode.

Parameters

in	*M*	int. M specifies the number of rows of the flat matrix.
in	*N*	int. N specifies the number of columns of the flat matrix.
in,out	*A*	double *. A is a pointer to the flat matrix.
in	*LDA*	int. LDA specifies the number of columns (row-major order) of matrix A.
in	*MT*	int. MT specifies the number of rows of the matrix TILE_A.
in	*NT*	int. NT specifies the number of columns of the matrix TILE_A.
in	*TILE↩ _A*	double *. TILE_A is a pointer to the tile matrix.

See also

ddss_dscatter_tile
ddss_tile_size

Definition at line 142 of file ddss_tiled2flat.c.

References ddss_dscatter_tile().

Referenced by kdnpgesv().

```
144 {
145
146     // Local variables
147     int m, n;
148
149     for ( m = 0; m < MT; m++ )
150     {
151         for ( n = 0; n < NT; n++ )
152         {
153             #pragma oss task inout(TILE_A[m][n]) \
154             label( dtiled2flat)
155             {
156                 ddss_dscatter_tile( M, N,
157                     &A[m * TILE_SIZE * N + n * TILE_SIZE], LDA,
158                     TILE_A[m][n], m, n );
159             }
160         }
161     }
162
163 }
```

2.1.2.17 int ddss_dtpgesv (int *N*, int *NRHS*, double * *A*, int *LDA*, int * *IPIV*, double * *B*, int *LDB*)

Solves a system of linear equations A X = B, where A is a N-by-N general matrix and X and B are N-by-NRHS matrices. The matrix A is factorized using the LU descomposition with tiled-pivoting. The matrix A is descomposed as:

```
A = P * L * U
```

where P is a permutation matrix, L is a lower triangular matrix with unit diagonal elements and U is an upper triangular matrix.

Parameters

in	*N*	int. N specifies the order of the square matrix A. N >= 0.

NRHS int. NRHS specifies the number of right-hand-sides (number of columns of B). NRHS >= 0.

Parameters

in,out	*A*	double *. A is a pointer to a regular matrix of dimension N-by-LDA. On exit, if return value is Success, the matrix A is overwriten by the factors L and U. The unit diagonal elements of L are not stored.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).
out	*IPIV*	int *. ipiv is a pointer to an array of dimesion at least max(1, min (M, N)). ipiv(i) = j, 1 <= i <= min(M, N) implies that rows i and j have been interchanged.

Parameters

in,out	*B*	double ∗. B is a pointer to a matrix of dimension N by NRHS, which stores the right-hand-sides of the systems of linear equations. (row-major order). On exit, if return value is Success, the matrix B is overwriten by the solution matrix X.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, NRHS).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

kdtpgesv

Definition at line 89 of file ddss_dtpgesv.c.

References kdtpgesv().

```
93 {
94
95     // Argument checking
96     if ( N < 0 )
97     {
98         fprintf( stderr, "Illegal value of N, in ddss_dtpgesv code\n" );
99         return NoSuccess;
100     }
101
102     if ( NRHS < 0 )
103     {
104         fprintf( stderr, "Illegal value of NRHS, in ddss_dtpgesv code\n" );
105         return NoSuccess;
106     }
107
108     if ( LDA < MAX( 1, N ) )
109     {
110         fprintf( stderr, "Illegal value of LDA, in ddss_dtpgesv code\n" );
111         return NoSuccess;
112     }
113
114     if ( LDB < MAX( 1, NRHS ) )
115     {
116         fprintf( stderr, "Illegal value of LDB, in ddss_dtpgesv code\n" );
117         return NoSuccess;
118     }
119
120     // Quick return
121     if ( MAX( N, 0 ) == 0 || MAX( NRHS, 0 ) == 0 )
122     {
123         return Success;
124     }
125
126     return kdtpgesv( N, NRHS, A, LDA, IPIV, B, LDB );
127
128 }
```

2.1.2.18 int ddss_dtpgetrf (int *M*, int *N*, double ∗ *A*, int *LDA*, int ∗ *IPIV*)

Performs the LU factorization with tiled pivoting (row interchanges) of a general M-by-N matrix A:

```
A = P * L * U
```

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

Parameters

in	*M*	int. M specifies the number of rows of the matrix A. M >= 0.
in	*N*	int. N specifies the number of columns of the matrix A. N >= 0.
in,out	*A*	double *. A is a pointer to a regular matrix of dimension M-by-N. On exit, if return value is Success, the matrix A is overwriten by the factors L and U. The unit diagonal elements of L are not stored.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).
out	*I↩ PIV*	int *. ipiv is a pointer to an array of dimesion at least max(1, min (M, N)). ipiv(i) = j, 1 <= i <= min(M, N) implies that rows i and j have been interchanged.

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

kdtpgetrf

Definition at line 77 of file ddss_dtpgetrf.c.

References kdtpgetrf().

```
78 {
79
80     // Argument checking
81     if ( M < 0 )
82     {
83         fprintf( stderr, "Illegal value of M, in ddss_dtpgetrf code\n" );
84         return NoSuccess;
85     }
86
87     if ( N < 0 )
88     {
89         fprintf( stderr, "Illegal value of N, in ddss_dtpgetrf code\n" );
90         return NoSuccess;
91     }
92
93     if ( LDA < MAX( 1, N ) )
94     {
95         fprintf( stderr, "Illegal value of LDA, in ddss_dtpgetrf code\n" );
96         return NoSuccess;
97     }
98
99     // Qick return
100     if ( MIN( M, N ) == 0 )
101     {
102         return Success;
103     }
104
105     return kdtpgetrf( M, N, A, LDA, IPIV );
106
107 }
```

2.1.2.19 int ddss_dtrmm (enum DDSS_SIDE *SIDE*, enum DDSS_UPLO *UPLO*, enum DDSS_TRANS *TRANS_A*, enum DDSS_DIAG *DIAG*, int *M*, int *N*, const double *ALPHA*, double * *A*, int *LDA*, double * *B*, int *LDB*)

Performs one of the matrix-matrix operations:

```
B = ALPHA * op( A ) * B, or B = ALPHA * B * op( A )
```

where op(A) is one of:

```
op( A ) = A      or
op( A ) = A**T
```

ALPHA is a scalar, B is a M by N matrix and A is a unit, or non-unit, upper or lower triangular matrix.

Parameters

in	*SIDE*	enum DDSS_SIDE. SIDE specifies the position of the triangular A matrix in the operations: • Left: B = ALPHA ∗ op(A) ∗ B. • Right: B = ALPHA ∗ B ∗ op(A).
in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form in which A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.
in	*TRANS↩ _A*	enum DDSS_TRANS. TRANS_A specifies the form of op(A) to be used: • NoTrans: op(A) = A. • Trans: op(A) = A∗∗T.
in	*DIAG*	enum DDSS_DIAG. DIAG specifies whether or not A is unit triangular as follows: • Unit: A is assumed to be unit triangular. • NonUnit: A is not assumed to be unit triangular.
in	*M*	int. M specifies the number of rows of B. M must be at least zero.
in	*N*	int. N specifies the number of columns of B. N must be at least zero.
in	*ALPHA*	double. ALPHA specifies the scalar alpha.
in	*A*	double ∗. A is a pointer to a matrix of dimension K by K, where K is M when SIDE = Left and is N otherwise. When UPLO = Uppper the strictly lower triangular part of A is not referenced and when UPLO = Lower the strictly upper triangular part of A is not referenced. Note that when DIAG = Unit, the diagonal elements of A are not referenced either, but are assumed to be unity.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). When SIDE = Left then LDA must be at least max(1, M), otherwise LDA must be at least max(1, N).
in,out	*B*	double ∗. B is a pointer to a matrix of dimension M by N. On exit the matrix B is overwritten by the transformed matrix.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, N).

See also

kdtrmm

Definition at line 113 of file ddss_dtrmm.c.

References kdtrmm().

```
118 {
119
120     // Local variables
121     int nA;
122
123     // Argument checking
124     if ( ( SIDE != Left ) && ( SIDE != Right ) )
125     {
126         fprintf( stderr, "Illegal value of SIDE, in ddss_dtrmm code\n" );
127         return NoSuccess;
```

```
128     }
129
130     if ( ( UPLO != Upper ) && ( UPLO != Lower ) )
131     {
132         fprintf( stderr, "Illegal value of UPLO, in ddss_dtrmm code\n" );
133         return NoSuccess;
134     }
135
136     if ( ( TRANS_A != NoTrans ) && ( TRANS_A != Trans ) )
137     {
138         fprintf( stderr, "Illegal value of TRANS_A, in ddss_dtrmm code\n" );
139         return NoSuccess;
140     }
141
142     if ( ( DIAG != Unit ) && ( DIAG != NonUnit ) )
143     {
144         fprintf( stderr, "Illegal value of DIAG, in ddss_dtrmm code\n" );
145         return NoSuccess;
146     }
147
148     if ( M < 0 )
149     {
150         fprintf( stderr, "Illegal value of M, in ddss_dtrmm code\n" );
151         return NoSuccess;
152     }
153
154     if ( N < 0 )
155     {
156         fprintf( stderr, "Illegal value of N, in ddss_dtrmm code\n" );
157         return NoSuccess;
158     }
159
160     if ( SIDE == Left )
161     {
162         nA = M;
163     }
164     else
165     {
166         nA = N;
167     }
168
169     if ( LDA < MAX( 1, nA ) )
170     {
171         fprintf( stderr, "Illegal value of LDA, in ddss_dtrmm code\n" );
172         return NoSuccess;
173     }
174
175     if ( LDB < MAX( 1, N ) )
176     {
177         fprintf( stderr, "Illegal value of LDB, in ddss_dtrmm code\n" );
178         return NoSuccess;
179     }
180
181     // Quick return
182     if ( M == 0 || N == 0 )
183     {
184         return Success;
185     }
186
187     return kdtrmm( SIDE, UPLO,
188                    TRANS_A, DIAG,
189                    M, N,
190                    ALPHA, A, LDA,
191                           B, LDB );
192
193 }
```

2.1.2.20 int ddss_dtrsm (enum DDSS_SIDE *SIDE*, enum DDSS_UPLO *UPLO*, enum DDSS_TRANS *TRANS_A*, enum DDSS_DIAG *DIAG*, int *M*, int *N*, const double *ALPHA*, double * *A*, int *LDA*, double * *B*, int *LDB*)

Solves one of the matrix equations:

```
op( A ) * X = ALPHA * B, or X * op( A ) = ALPHA * B
```

where op(A) is one of:

```
op( A ) = A      or
op( A ) = A**T
```

ALPHA is a scalar, X and B are M by N matrices, A is a unit, or non-unit, upper or lower triangular matrix. The matrix X is overwritten on B

Parameters

in	*SIDE*	enum DDSS_SIDE. SIDE specifies the position of the triangular A matrix in the operations: • Left: op(A) ∗ X = ALPHA ∗ B. • Right: X ∗ op(A) = ALPHA ∗ B.
in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form of A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.
in	*TRANS_A*	enum DDSS_TRANS. TRANS_A specifies the form of op(A) to be used: • NoTrans: op(A) = A. • Trans: op(A) = A∗∗T.
in	*DIAG*	enum DDSS_DIAG. DIAG specifies whether or not A is unit triangular as follows: • Unit: A is assumed to be unit triangular. • NonUnit: A is not assumed to be unit triangular.
in	*M*	int. M specifies the number of rows of B. M must be at least zero.
in	*N*	int. N specifies the number of columns of B. N must be at least zero.
in	*ALPHA*	double. ALPHA specifies the scalar alpha.
in	*A*	double ∗. A is a pointer to a matrix of dimension K by K, where K is M when SIDE = Left and is N otherwise. When UPLO = Uppper the strictly lower triangular part of A is not referenced and when UPLO = Lower the strictly upper triangular part of A is not referenced. Note that when DIAG = Unit, the diagonal elements of A are not referenced either, but are assumed to be unity.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). When SIDE = Left then LDA must be at least max(1, M), otherwise LDA must be at least max(1, N).
in,out	*B*	double ∗. B is a pointer to a matrix of dimension M by N. On exit the matrix B is overwritten by the solution matrix X.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, N).

See also

kdtrsm

Definition at line 113 of file ddss_dtrsm.c.

References kdtrsm().

```
118 {
119
120     // Local variables
121     int nA;
122
123     // Argument checking
124     if ( ( SIDE != Left ) && ( SIDE != Right ) )
125     {
126         fprintf( stderr, "Illegal value of SIDE, in ddss_dtrsm code\n" );
127         return NoSuccess;
```

```
128     }
129
130     if ( ( UPLO != Upper ) && ( UPLO != Lower ) )
131     {
132         fprintf( stderr, "Illegal value of UPLO, in ddss_dtrsm code\n" );
133         return NoSuccess;
134     }
135
136     if ( ( TRANS_A != NoTrans ) && ( TRANS_A != Trans ) )
137     {
138         fprintf( stderr, "Illegal value of TRANS_A, in ddss_dtrsm code\n" );
139         return NoSuccess;
140     }
141
142     if ( ( DIAG != Unit ) && ( DIAG != NonUnit ) )
143     {
144         fprintf( stderr, "Illegal value of DIAG, in ddss_dtrsm code\n" );
145         return NoSuccess;
146     }
147
148     if ( M < 0 )
149     {
150         fprintf( stderr, "Illegal value of M, in ddss_dtrsm code\n" );
151         return NoSuccess;
152     }
153
154     if ( N < 0 )
155     {
156         fprintf( stderr, "Illegal value of N, in ddss_dtrsm code\n" );
157         return NoSuccess;
158     }
159
160     if ( SIDE == Left )
161     {
162         nA = M;
163     }
164     else
165     {
166         nA = N;
167     }
168
169     if ( LDA < MAX( 1, nA ) )
170     {
171         fprintf( stderr, "Illegal value of LDA, in ddss_dtrsm code\n" );
172         return NoSuccess;
173     }
174
175     if ( LDB < MAX( 1, N ) )
176     {
177         fprintf( stderr, "Illegal value of LDB, in ddss_dtrsm code\n" );
178         return NoSuccess;
179     }
180
181     // Quick return
182     if ( M == 0 || N == 0 )
183     {
184         return Success;
185     }
186
187     return kdtrsm( SIDE, UPLO,
188                    TRANS_A, DIAG,
189                    M, N,
190                    ALPHA, A, LDA,
191                           B, LDB );
192
193 }
```

2.1.2.21 int ddss_tile_size (int *M*, int *MT*)

tile_size: Computes the size of the tile passed as parameter.

Parameters

in	*M*	int. M specifies the size (rows of columns) of the matrix.
in	*MT*	int. MT specifies the id of the tile.

Return values

int	size of the tile passed as parameter.

See also

ddss_gather_tile

Definition at line 52 of file ddss_tile.c.

Referenced by ddss_dgather_tile(), ddss_dscatter_tile(), kdgemm(), kdnpgesv(), kdnpgetrf(), kdposv(), kdpotrf(), kdsymm(), kdsyr2k(), kdsyrk(), kdtpgesv(), kdtpgetrf(), kdtrmm(), and kdtrsm().

```
53 {
54
55     if ( M - ( MT * TILE_SIZE ) > TILE_SIZE )
56         return TILE_SIZE;
57     else
58         return M - ( MT * TILE_SIZE );
59
60 }
```

2.1.2.22 enum LASS_RETURN dnpgetrf (int *M*, int *N*, double * *A*, int *LDA*)

Performs the LU factorization without pivoting of a general M-by-N matrix A:

```
A = L * U
```

where L is a lower triangular (lower trapezoidal if M > N) matrix with unit diagonal elements and U is an upper triangular (upper trapezoidal if M < N) matrix.

Parameters

`in`	*M*	int. M specifies the number of rows of the matrix A. M >= 0.
`in`	*N*	int. N specifies the number of columns of the matrix A. N >= 0.
`in,out`	*A*	double *. A is a pointer to a regular matrix of dimension M-by-N. On exit, if return value is Success, the matrix A is overwriten by the factors L and U. The unit diagonal elements of L are not stored.
`in`	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

kdnpgetrf

Definition at line 70 of file dnpgetrf.c.

Referenced by kdnpgesv(), and kdnpgetrf().

```
71 {
72
73     // Local variable
74     double alpha;
75     double sfmin;
76     int i, j, k;
77     int info = 0;
78
79     // Argument checking
80     if ( M < 0 )
81     {
82         fprintf( stderr, "Illegal value of M, in dnpgetrf code\n" );
83         return NoSuccess;
84     }
85
86     if ( N < 0 )
87     {
88         fprintf( stderr, "Illegal value of N, in dnpgetrf code\n" );
89         return NoSuccess;
90     }
91
92     if ( LDA < MAX( 1, N ) )
93     {
94         fprintf( stderr, "Illegal value of LDA, in dnpgetrf code\n" );
95         return NoSuccess;
96     }
97
98     // Qick return
99     if ( MIN( M, N ) == 0 )
100     {
101         return Success;
102     }
103
104     /****************
105     --DNPGETRF tile--
106     ****************/
107
108     // Minimum value in double precision
109     sfmin = LAPACKE_dlamch_work( 'S' );
110     k = MIN( M, N );
111
112     for ( i = 0; i < k; i++ )
113     {
114         alpha = A[i * LDA + i];
115         if ( alpha != ( double )0.0 )
116         {
117             // Compute elements from J+1 to M of the J-th column
118             if ( i < M )
119             {
120                 if ( fabs( alpha ) > fabs( sfmin ) )
121                 {
122                     alpha = 1.0 / alpha;
123                     cblas_dscal( M - i - 1, alpha,
124                         &( A[( i + 1 ) * LDA + i] ), LDA );
125                 }
126                 else
127                 {
128                     for( j= i + 1; j < M; j++ )
129                     {
130                         A[j * LDA + i] = A[j * LDA + i] / alpha;
131                     }
132                 }
133             }
134         }
135         else if ( info == 0 )
136         {
137             info = i;
138         }
139         if ( i < k )
140         {
141             cblas_dger( CblasRowMajor,
142                         M - i - 1, N - i - 1,
143                         -1.0,
144                         &A[( i + 1 ) * LDA + i], LDA,
145                         &A[ i * LDA + ( i + 1 )], 1,
146                         &A[( i + 1 ) * LDA + ( i + 1 ) ], LDA );
147         }
148     }
149
150     return Success;
151
152 }
```

2.1.2.23 void dspmvseq (int *M,* int *N,* double *ALPHA,* const double * *VAL_A,* const int * *ROW_PTR_A,* const int * *COL_IND_A,* const double * *X,* double *BETA,* double * *Y*)

Performs the sparse matrix-vector operation:

```
Y = ALPHA * A * X + BETA * Y
```

where ALPHA and BETA are scalars, X and Y are vectors, and A is a sparse matrix stored in thhe next three vectors VAL_A, COL_IND_A and ROW_PTR_A according to CSR format.

Parameters

in	*M*	int. M specifies the number of rows of A.
in	*N*	int. N specifies the number of columns of A.
in	*ALPHA*	double.
in	*VAL_A*	double *. VAL_A is a pointer to a vector which specifies the non-zero values of the original matrix A.
in	*COL_IND_A*	unsigned int *. COL_IND_A is a pointer to a vector which specifies the column of each value of A in the original matrix.
in	*ROW_PTR_A*	unsigned int *. ROW_PTR_A is a pointer to a vector which specifies the number of values that A contains in each row of the original matrix.
in	*X*	double *. X is a pointer to a vector of dimension N.
in	*BETA*	double .
in,out	*Y*	double *. Y is a pointer to a vector of dimension M. On exit, Y is overwritten by the sparse matrix-vector result.

Return values

Success	sucessful exit
NoSuccess	unsucessful exit

Definition at line 84 of file dspmv.c.

```
90 {
91
92     // Local variables
93     double svalue = 0.0, value = 0.0;
94     int max = 0, col = 0;
95     int i, x;
96
97     for ( x = 0; x < M; x++ )
98     {
99         svalue = 0.0;
100         max = ROW_PTR_A[x + 1] - ROW_PTR_A[x];
101
102         for( i = 0; i < max; i++ )
103         {
104             value = VAL_A[ROW_PTR_A[x] + i];
105             col = COL_IND_A[ROW_PTR_A[x] + i];
106             svalue += value * X[col] * ALPHA;
107
108
109         }
110
111         Y[x] = svalue + Y[x] * BETA;
112
113     }
114
115 }
```

2.1.2.24 enum LASS_RETURN kdgemm (enum DDSS_TRANS *TRANS_A*, enum DDSS_TRANS *TRANS_B*, int *M*, int *N*, int *K*, const double *ALPHA*, double * *A*, int *LDA*, double * *B*, int *LDB*, const double *BETA*, double * *C*, int *LDC*)

Performs the matrix-matrix operation:

```
C = ALPHA * op( A ) * op( B ) + BETA * C
```

where op(X) is one of:

```
op( X ) = X      or
op( X ) = X**T
```

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

in	*TRANS_A*	enum DDSS_TRANS. TRANS_A specifies the form of op(A) to be used in the matrix multiplication as follows: • NoTrans: op(A) = A. • Trans: op(A) = A**T.
in	*TRANS_B*	enum DDSS_TRANS. TRANS_B specifies the form of op(B) to be used in the matrix multiplication as follows: • NoTrans: op(B) = B. • Trans: op(B) = B**T.
in	*M*	int. M specifies the number of rows of the matrix A and of the matrix C. M must be greater than zero.
in	*N*	int. N specifies the number of columns of the matrix B and the number of columns of the matrix C. N must be greater than zero.
in	*K*	int. K specifies the number of columns of the matrix A and the number of rows of the matrix B. K must be greater than zero.
in	*ALPHA*	double.
in	*A*	double *. A is a pointer to a matrix of dimension Ma (rows) by Ka (columns), where Ma is M and Ka is K when TRANS_A = NoTrans, and Ma is K and Ka is M otherwise.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). When TRANS_A = NoTrans then LDA must be at least max(1, K), otherwise LDA must be at least max(1, M).
in	*B*	double *. B is a pointer to a matrix of dimension Kb (rows) by Nb (columns), where Kb is K and Nb is N when TRANS_B = NoTrans, and Kb is N and Nb is K otherwise.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). When TRANS_B = NoTrans then LDB must be at least max(1, N), otherwise LDB must be at least max(1, K).
in	*BETA*	double.
in,out	*C*	double *. C is a pointer to a matrix of dimension LDC by N. On exit, C is overwritten by the M by N matrix (ALPHA*op(A)*op(B) + BETA*C).
in	*LDC*	int. LDC specifies the number of columns of C (row-major order). LDC must be at least max(1, M)

Return values

Success	sucessful exit
NoSuccess	unsucessful exit

See also

ddss_dgemm
ddss_tile
ddss_flat2tiled
ddss_tiled2flat

Definition at line 131 of file kdgemm.c.

References ddss_dflat2tiled(), ddss_dtiled2flat(), and ddss_tile_size().

Referenced by ddss_dgemm().

```
136 {
137
138     // Local variables
139     int mt, kt, nt;
140     int mi, ki, ni;
141     int Am, An;
142     int Bm, Bn;
143     int tile_size_m;
144     int tile_size_n;
145     int tile_size_k;
146     int k_check;
147     double betat;
148
149     // Number of tiles
150     if ( M % TILE_SIZE == 0 )
151     {
152         mt = M / TILE_SIZE;
153     }
154     else
155     {
156         mt = ( M / TILE_SIZE ) + 1;
157     }
158
159     if ( K % TILE_SIZE == 0 )
160     {
161         kt = K / TILE_SIZE;
162     }
163     else
164     {
165         kt = ( K / TILE_SIZE ) + 1;
166     }
167
168     if ( N % TILE_SIZE == 0 )
169     {
170         nt = N / TILE_SIZE;
171     }
172     else
173     {
174         nt = ( N / TILE_SIZE ) + 1;
175     }
176
177     /***************************
178     --Tile matrices declaration--
179     ***************************/
180
181     if ( TRANS_A == NoTrans )
182     {
183         Am = mt;
184         An = kt;
185     }
186     else
187     {
188         Am = kt;
189         An = mt;
190     }
191
192     if ( TRANS_B == NoTrans )
```

```
193     {
194         Bm = kt;
195         Bn = nt;
196     }
197     else
198     {
199         Bm = nt;
200         Bn = kt;
201     }
202 
203     /***************************
204     --Tile matrices allocation--
205     ***************************/
206 
207     double (*TILE_A)[An][TILE_SIZE * TILE_SIZE] = malloc ( Am * An *
208         TILE_SIZE * TILE_SIZE * sizeof( double ) );
209 
210     if ( TILE_A == NULL)
211     {
212         fprintf( stderr, "Failure in kdgemm for matrix TILE_A\n" );
213         return NoSuccess;
214     }
215 
216     double (*TILE_B)[Bn][TILE_SIZE * TILE_SIZE] = malloc ( Bm * Bn *
217         TILE_SIZE * TILE_SIZE * sizeof( double ) );
218 
219     if ( TILE_B == NULL)
220     {
221         fprintf( stderr, "Failure in kdgemm for matrix TILE_B\n" );
222         return NoSuccess;
223     }
224 
225     double (*TILE_C)[nt][TILE_SIZE * TILE_SIZE] = malloc ( mt * nt *
226         TILE_SIZE * TILE_SIZE * sizeof( double ) );
227 
228     if ( TILE_C == NULL)
229     {
230         fprintf( stderr, "Failure in kdgemm for matrix TILE_C\n" );
231         return NoSuccess;
232     }
233 
234     /*****************************************
235     --From flat data layout to tiled data layout--
236     *****************************************/
237 
238     // From flat matrix A to tile matrix TILE_A
239     if ( TRANS_A == NoTrans )
240     {
241         ddss_dflat2tiled( M, K, A, LDA, mt, kt, TILE_A );
242     }
243     else
244     {
245         ddss_dflat2tiled( K, M, A, LDA, kt, mt, TILE_A );
246     }
247 
248     // From flat matrix B to tile matrix TILE_B
249     if ( TRANS_B == NoTrans )
250     {
251         ddss_dflat2tiled( K, N, B, LDB, kt, nt, TILE_B );
252     }
253     else
254     {
255         ddss_dflat2tiled( N, K, B, LDB, nt, kt, TILE_B );
256     }
257 
258     // From flat matrix C to tile matrix TILE_C
259     ddss_dflat2tiled( M, N, C, LDC, mt, nt, TILE_C );
260 
261     /*************
262     --DGEMM tile--
263     *************/
264 
265     for ( mi = 0; mi < mt; mi++ )
266     {
267         tile_size_m = ddss_tile_size( M, mi );
268         for ( ni = 0; ni < nt; ni++ )
269         {
270             tile_size_n = ddss_tile_size( N, ni );
271             if ( TRANS_A == NoTrans )
272             {
273                 k_check = K;
274             }
275             else
276             {
277                 k_check = M;
278             }
279             // --Scale on C ( C = BETA * C )--
```

```
280             if ( ( ALPHA == 0.0 ) || ( k_check == 0 ) )
281             {
282                 #pragma oss task inout( TILE_C[mi][ni] ) \
283                     shared( TILE_A, TILE_B, TILE_C ) \
284                     firstprivate( mi, ni ) \
285                     no_copy_deps
286                 cblas_dgemm( CblasRowMajor,
287                              ( CBLAS_TRANSPOSE ) TRANS_A,
288                              ( CBLAS_TRANSPOSE ) TRANS_B,
289                              tile_size_m,
290                              tile_size_n,
291                                        0,
292                              ALPHA,   TILE_A[0][0], 1,
293                                       TILE_B[0][0], 1,
294                               BETA, TILE_C[mi][ni], tile_size_n );
295             }
296             else if ( TRANS_A == NoTrans )
297             {
298                 // --TRANS_A = NoTrans & TRANS_B = NoTrans--
299                 if ( TRANS_B == NoTrans )
300                 {
301                     for ( ki = 0; ki < kt; ki++ )
302                     {
303                         tile_size_k = ddss_tile_size( K, ki );
304                         if (ki == 0)
305                         {
306                             betat = BETA;
307                         }
308                         else
309                         {
310                             betat = 1.0;
311                         }
312
313                         #pragma oss task in( TILE_A[mi][ki] ) \
314                             in( TILE_B[ki][ni] ) \
315                             inout( TILE_C[mi][ni] ) \
316                             shared( TILE_A, TILE_B, TILE_C ) \
317                             firstprivate( mi, ni, ki, betat ) \
318                             no_copy_deps
319                         cblas_dgemm( CblasRowMajor,
320                                      ( CBLAS_TRANSPOSE ) TRANS_A,
321                                      ( CBLAS_TRANSPOSE ) TRANS_B,
322                                      tile_size_m,
323                                      tile_size_n,
324                                      tile_size_k,
325                                      ALPHA, TILE_A[mi][ki], tile_size_k,
326                                             TILE_B[ki][ni], tile_size_n,
327                                      betat, TILE_C[mi][ni], tile_size_n );
328                     }
329                 }
330                 // --TRANS_A = NoTrans & TRANS_B = Trans--
331                 else
332                 {
333                     for ( ki = 0; ki < kt; ki++ )
334                     {
335                         tile_size_k = ddss_tile_size( K, ki );
336                         if (ki == 0)
337                         {
338                             betat = BETA;
339                         }
340                         else
341                         {
342                             betat = 1.0;
343                         }
344
345                         #pragma oss task in( TILE_A[mi][ki] ) \
346                             in( TILE_B[ni][ki] ) \
347                             inout( TILE_C[mi][ni] ) \
348                             shared( TILE_A, TILE_B, TILE_C ) \
349                             firstprivate( mi, ni, ki, betat )
350                         cblas_dgemm( CblasRowMajor,
351                                      ( CBLAS_TRANSPOSE ) TRANS_A,
352                                      ( CBLAS_TRANSPOSE ) TRANS_B,
353                                      tile_size_m,
354                                      tile_size_n,
355                                      tile_size_k,
356                                      ALPHA,  TILE_A[mi][ki], tile_size_k,
357                                              TILE_B[ni][ki], tile_size_k,
358                                      betat,  TILE_C[mi][ni], tile_size_n );
359                     }
360                 }
361             }
362             else
363             {
364                 // --TRANS_A = Trans & TRANS_B = NoTrans--
365                 if ( TRANS_B == NoTrans )
366                 {
```

```
367                     for ( ki = 0; ki < kt; ki++ )
368                     {
369                         tile_size_k = ddss_tile_size( K, ki );
370                         if (ki == 0)
371                         {
372                             betat = BETA;
373                         }
374                         else
375                         {
376                             betat = 1.0;
377                         }
378 
379                         #pragma oss task in( TILE_A[ki][mi] ) \
380                             in( TILE_B[ki][ni] ) \
381                             inout( TILE_C[mi][ni] ) \
382                             shared( TILE_A, TILE_B, TILE_C ) \
383                             firstprivate( mi, ni, ki, betat )
384                         cblas_dgemm( CblasRowMajor,
385                                     ( CBLAS_TRANSPOSE ) TRANS_A,
386                                     ( CBLAS_TRANSPOSE ) TRANS_B,
387                                     tile_size_m,
388                                     tile_size_n,
389                                     tile_size_k,
390                                     ALPHA,  TILE_A[ki][mi], tile_size_m,
391                                             TILE_B[ki][ni], tile_size_n,
392                                     betat,  TILE_C[mi][ni], tile_size_n );
393                     }
394                 }
395                 // --TRANS_A = Trans & TRANS_B = Trans--
396                 else
397                 {
398                     for ( ki = 0; ki < kt; ki++ )
399                     {
400                         tile_size_k = ddss_tile_size( K, ki );
401                         if (ki == 0)
402                         {
403                             betat = BETA;
404                         }
405                         else
406                         {
407                             betat = 1.0;
408                         }
409 
410                         #pragma oss task in( TILE_A[ki][mi] ) \
411                             in( TILE_B[ni][ki] ) \
412                             inout( TILE_C[mi][ni] ) \
413                             shared( TILE_A, TILE_B, TILE_C ) \
414                             firstprivate( mi, ni, ki, betat )
415                         cblas_dgemm( CblasRowMajor,
416                                     ( CBLAS_TRANSPOSE ) TRANS_A,
417                                     ( CBLAS_TRANSPOSE ) TRANS_B,
418                                     tile_size_m,
419                                     tile_size_n,
420                                     tile_size_k,
421                                     ALPHA,  TILE_A[ki][mi], tile_size_m,
422                                             TILE_B[ni][ki], tile_size_k,
423                                     betat,  TILE_C[mi][ni], tile_size_n );
424                     }
425                 }
426             }
427         }
428     }
429 
430     /*********************************************
431     // --From tiled data layout to flat data layout--
432     *********************************************/
433 
434     // From tile matrix TILE_C to flat matrix C
435     ddss_dtiled2flat( M, N, C, LDC, mt, nt, TILE_C );
436 
437     // --Tile matrices free--
438     free( TILE_A );
439     free( TILE_B );
440     free( TILE_C );
441 
442     return Success;
443 
444 }
```

2.1.2.25 enum LASS_RETURN kdnpgesv (int *N*, int *NRHS*, double * *A*, int *LDA*, double * *B*, int *LDB*)

Solves a system of linear equations A X = B, where A is a N-by-N general matrix and X and B are N-by-NRHS matrices. The matrix A is factorized using the LU descomposition without pivoting. The matrix A is descomposed as:

```
A = L * U
```

where L is a lower triangular matrix with unit diagonal elements and U is an upper triangular matrix.

Parameters

in	*N*	int. N specifies the order of the square matrix A. N >= 0.

NRHS int. NRHS specifies the number of right-hand-sides (number of columns of B). NRHS >= 0.

Parameters

in,out	*A*	double *. A is a pointer to a regular matrix of dimension N-by-LDA. On exit, if return value is Success, the matrix A is overwriten by the factors L and U. The unit diagonal elements of L are not stored.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).
in,out	*B*	double *. B is a pointer to a matrix of dimension N by NRHS, which stores the right-hand-sides of the systems of linear equations. (row-major order). On exit, if return value is Success, the matrix B is overwriten by the solution matrix X.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, NRHS).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

dnpgesv
ddss_tile
ddss_flat2tiled
ddss_tiled2flat

Definition at line 86 of file kdnpgesv.c.

References ddss_dflat2tiled(), ddss_dtiled2flat(), ddss_dtiled2flat_nb(), ddss_tile_size(), and dnpgetrf().

Referenced by ddss_dnpgesv().

```
89 {
90
91     // Local variables
92     int nt, nrhst;
93     int mi, mmi, ki, ni, nni;
94     int tile_size_m;
95     int tile_size_mm;
96     int tile_size_n;
97     int tile_size_nn;
98     int tile_size_k;
99
100     // Number of tiles
101     if ( N % TILE_SIZE == 0 )
102     {
103         nt = N / TILE_SIZE;
104     }
```

```
105     else
106     {
107         nt = ( N / TILE_SIZE ) + 1;
108     }
109
110     if ( NRHS % TILE_SIZE == 0 )
111     {
112         nrhst = NRHS / TILE_SIZE;
113     }
114     else
115     {
116         nrhst = ( NRHS / TILE_SIZE ) + 1;
117     }
118
119     /***************************
120     --Tile matrices allocation--
121     ***************************/
122
123     double (*TILE_A)[nt][TILE_SIZE * TILE_SIZE] = malloc ( nt * nt *
124         TILE_SIZE * TILE_SIZE * sizeof( double ) );
125
126     if ( TILE_A == NULL )
127     {
128         fprintf( stderr, "Failure in kdnpgesv for matrix TILE_A\n" );
129         return NoSuccess;
130     }
131
132     double ( *TILE_B )[nrhst][TILE_SIZE * TILE_SIZE] = malloc(
133              nt * nrhst * TILE_SIZE * TILE_SIZE * sizeof( double ) );
134
135     if ( TILE_B == NULL )
136     {
137         fprintf( stderr, "Failure in kdnpgesv for matrix TILE_B\n" );
138         return NoSuccess;
139     }
140
141     /***********************************************
142     // --From flat data layout to tiled data layout--
143     ***********************************************/
144
145     // From flat matrix A to tile matrix TILE_A
146     ddss_dflat2tiled( N, N, A, LDA, nt, nt, TILE_A );
147
148     // From flat matrix B to tile matrix TILE_B
149     ddss_dflat2tiled( N, NRHS, B, LDB, nt, nrhst, TILE_B );
150
151     /****************
152     --DNPGESV tile--
153     ****************/
154
155     // LU descomposition without pivoting
156     for ( ki = 0; ki < nt; ki++ )
157     {
158         tile_size_k = ddss_tile_size( N, ki );
159
160         #if defined(LASs_WITH_MKL)
161
162         #pragma oss task inout( TILE_A[ki][ki] ) \
163                 shared( TILE_A ) \
164                 firstprivate( ki ) \
165                 label( mkl_dgetrfnpi )
166         LAPACKE_mkl_dgetrfnpi( CblasRowMajor,
167                                tile_size_k, tile_size_k,
168                                tile_size_k,
169                                TILE_A[ki][ki], tile_size_k );
170
171         #else
172
173         #pragma oss task inout( TILE_A[ki][ki] ) \
174                 shared( TILE_A ) \
175                 firstprivate( ki ) \
176                 label( dnpgetrf )
177         dnpgetrf( tile_size_k, tile_size_k,
178                   TILE_A[ki][ki],
179                   tile_size_k );
180
181         #endif
182
183         for ( mi = ki + 1; mi < nt; mi++ )
184         {
185             tile_size_m = ddss_tile_size( N, mi );
186
187             #pragma oss task in( TILE_A[ki][ki] ) \
188                     inout(TILE_A[mi][ki]) \
189                     shared( TILE_A ) \
190                     firstprivate( mi, ki ) \
191                     label( dnpgetrf_dtrsm_below )
```

```
192             cblas_dtrsm( CblasRowMajor,
193                          ( CBLAS_SIDE ) Right, ( CBLAS_UPLO ) Upper,
194                          ( CBLAS_TRANSPOSE ) NoTrans, ( CBLAS_DIAG ) NonUnit,
195                          tile_size_m, tile_size_k,
196                          1.0, TILE_A[ki][ki], tile_size_k,
197                               TILE_A[mi][ki], tile_size_k );
198         }
199         for ( ni = ki + 1; ni < nt; ni++ )
200         {
201             tile_size_n = ddss_tile_size( N, ni );
202
203             #pragma oss task in( TILE_A[ki][ki] ) \
204                     inout(TILE_A[ki][ni]) \
205                     shared( TILE_A ) \
206                     firstprivate( ni, ki ) \
207                     label( dnpgetrf_dtrsm_right )
208             cblas_dtrsm( CblasRowMajor,
209                          ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Lower,
210                          ( CBLAS_TRANSPOSE ) NoTrans, ( CBLAS_DIAG ) Unit,
211                          tile_size_k, tile_size_n,
212                          1.0, TILE_A[ki][ki], tile_size_k,
213                               TILE_A[ki][ni], tile_size_n );
214
215             for ( mmi = ki + 1; mmi < nt; mmi++ )
216             {
217                 tile_size_mm = ddss_tile_size( N, mmi );
218
219                 #pragma oss task in( TILE_A[mmi][ki] ) \
220                         in( TILE_A[ki][ni] ) \
221                         inout( TILE_A[mmi][ni] ) \
222                         shared( TILE_A ) \
223                         firstprivate( ni, mmi, ki ) \
224                         label( dnpgetrf_dgemm )
225                 cblas_dgemm( CblasRowMajor,
226                              ( CBLAS_TRANSPOSE ) NoTrans,
227                              ( CBLAS_TRANSPOSE ) NoTrans,
228                              tile_size_mm,
229                              tile_size_n,
230                              TILE_SIZE,
231                              -1.0, TILE_A[mmi][ki], tile_size_k,
232                                    TILE_A[ki][ni],  tile_size_n,
233                               1.0, TILE_A[mmi][ni], tile_size_n );
234             }
235         }
236     }
237
238     /*******************************************
239     --From tiled data layout to flat data layout--
240     *******************************************/
241     ddss_dtiled2flat_nb( N, N, A, LDA, nt, nt, TILE_A );
242
243     // Triandular solve
244     // --SIDE = Left & UPLO = Lower & TRANS_A = NoTrans & Unit--
245     for ( ki = 0; ki < nt; ki++ )
246     {
247         tile_size_k = ddss_tile_size( N, ki );
248
249         for ( ni = 0; ni < nrhst; ni++ )
250         {
251             tile_size_n = ddss_tile_size( NRHS, ni );
252
253             #pragma oss task in( TILE_A[ki][ki] ) \
254                 inout(  TILE_B[ki][ni] ) \
255                 shared( TILE_A, TILE_B ) \
256                 firstprivate( ki, ni ) \
257                 label( dtrsm_dtrsm1 )
258             cblas_dtrsm( CblasRowMajor,
259                          ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Lower,
260                          ( CBLAS_TRANSPOSE ) NoTrans,
261                          ( CBLAS_DIAG ) Unit,
262                          tile_size_k,
263                          tile_size_n,
264                          1.0, TILE_A[ki][ki], tile_size_k,
265                               TILE_B[ki][ni], tile_size_n );
266         }
267         for ( nni = 0; nni < nrhst; nni++ )
268         {
269             tile_size_nn = ddss_tile_size( NRHS, nni );
270
271             for ( mi = ki + 1; mi < nt; mi++ )
272             {
273
274                 #pragma oss task in( TILE_A[mi][ki] ) \
275                     in( TILE_B[ki][nni] ) \
276                     inout( TILE_B[mi][nni] ) \
277                     shared( TILE_A, TILE_B ) \
278                     firstprivate( ki, mi, nni ) \
```

```
279                     label( dtrsm_dgemm1 )
280                 cblas_dgemm( CblasRowMajor,
281                              CblasNoTrans, CblasNoTrans,
282                              tile_size_k,
283                              tile_size_nn,
284                              tile_size_k,
285                              -1.0,  TILE_A[mi][ki],  tile_size_k,
286                                     TILE_B[ki][nni], tile_size_nn,
287                               1.0, TILE_B[mi][nni], tile_size_nn );
288             }
289         }
290     }
291
292     // Triandular solve
293     // --SIDE = Left & UPLO = Upper & TRANS_A = NoTrans & NonUnit--
294     for ( ki = nt - 1; ki >= 0; ki-- )
295     {
296         tile_size_k = ddss_tile_size( N, ki );
297
298         for ( ni = 0; ni < nrhst; ni++ )
299         {
300             tile_size_n = ddss_tile_size( NRHS, ni );
301
302             #pragma oss task in( TILE_A[ki][ki] ) \
303                 inout( TILE_B[ki][ni] ) \
304                 shared( TILE_A, TILE_B ) \
305                 firstprivate( ki, ni ) \
306                 label( dtrsmi_dtrsm2 )
307             cblas_dtrsm( CblasRowMajor,
308                          ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Upper,
309                          ( CBLAS_TRANSPOSE ) NoTrans,
310                          ( CBLAS_DIAG ) NonUnit,
311                          tile_size_k,
312                          tile_size_n,
313                          1.0, TILE_A[ki][ki], tile_size_k,
314                               TILE_B[ki][ni], tile_size_n );
315         }
316
317         for ( mi = ki - 1; mi >= 0; mi-- )
318         {
319             tile_size_m = ddss_tile_size( N, mi );
320
321             for ( nni = 0; nni < nrhst; nni++ )
322             {
323                 tile_size_nn = ddss_tile_size( NRHS, nni );
324
325                 #pragma oss task in( TILE_A[mi][ki] ) \
326                     in( TILE_B[ki][nni] ) \
327                     inout( TILE_B[mi][nni] ) \
328                     shared( TILE_A, TILE_B ) \
329                     firstprivate( ki, mi, nni ) \
330                     label( dtrsm_dgemm2 )
331                 cblas_dgemm( CblasRowMajor,
332                              CblasNoTrans, CblasNoTrans,
333                              tile_size_m,
334                              tile_size_nn,
335                              tile_size_k,
336                              -1.0, TILE_A[mi][ki],  tile_size_k,
337                                    TILE_B[ki][nni], tile_size_nn,
338                               1.0, TILE_B[mi][nni], tile_size_nn );
339             }
340         }
341     }
342
343     /*******************************************
344     --From tiled data layout to flat data layout--
345     *******************************************/
346     ddss_dtiled2flat( N, NRHS, B, LDB, nt, nrhst, TILE_B );
347
348     // --Tile A and B matrices free--
349     free( TILE_A );
350     free( TILE_B );
351
352     return Success;
353
354 }
```

2.1.2.26 enum LASS_RETURN kdnpgetrf (int *M*, int *N*, double * *A*, int *LDA*)

Performs the LU factorization without pivoting of a general M-by-N matrix A:

```
A = L * U
```

where L is a lower triangular (lower trapezoidal if M $>$ N) matrix with unit diagonal elements and U is an upper triangular (upper trapezoidal if M $<$ N) matrix.

Parameters

in	*M*	int. M specifies the number of rows of the matrix A. M >= 0.
in	*N*	int. N specifies the number of columns of the matrix A. N >= 0.
in,out	*A*	double $*$. A is a pointer to a regular matrix of dimension M-by-N. On exit, if return value is Success, the matrix A is overwriten by the factors L and U. The unit diagonal elements of L are not stored.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

dnpgetrf
tune_dnpgetrf
smart_dnpgetrf
ddss_tile
ddss_flat2tiled
ddss_tiled2flat

Definition at line 74 of file kdnpgetrf.c.

References ddss_dflat2tiled(), ddss_dtiled2flat(), ddss_tile_size(), and dnpgetrf().

Referenced by ddss_dnpgetrf().

```
75 {
76
77     // Local variables
78     int mt, nt;
79     int mi, mmi, ki, ni, nni, li;
80     int tile_size_m;
81     int tile_size_mm;
82     int tile_size_n;
83     int tile_size_nn;
84     int tile_size_k;
85     int tile_size_km;
86     int tile_size_kn;
87
88     // Number of tiles
89     if ( M % TILE_SIZE == 0 )
90     {
91         mt = M / TILE_SIZE;
92     }
93     else
94     {
95         mt = ( M / TILE_SIZE ) + 1;
96     }
97
98     if ( N % TILE_SIZE == 0 )
99     {
100         nt = N / TILE_SIZE;
101     }
102     else
103     {
104         nt = ( N / TILE_SIZE ) + 1;
105     }
106
```

```
107     /**************************
108     --Tile A matrix allocation--
109     **************************/
110
111     double (*TILE_A)[nt][TILE_SIZE * TILE_SIZE] = malloc ( mt * nt *
112         TILE_SIZE * TILE_SIZE * sizeof( double ) );
113
114     if ( TILE_A == NULL )
115     {
116         fprintf( stderr, "Failure in kdnpgetrf for matrix TILE_A\n" );
117         return NoSuccess;
118     }
119
120     /*******************************************
121     --From flat data layout to tiled data layout--
122     *******************************************/
123
124     ddss_dflat2tiled( M, N, A, LDA, mt, nt, TILE_A );
125
126     /****************
127     --DNPGETRF tile--
128     ****************/
129
130     for ( ki = 0; ki < MIN( mt, nt ); ki++ )
131     {
132         tile_size_km = ddss_tile_size( M, ki );
133         tile_size_kn = ddss_tile_size( N, ki );
134
135         #if defined(LASs_WITH_MKL)
136
137         tile_size_k = MIN( tile_size_km, tile_size_kn );
138
139         #pragma oss task inout( TILE_A[ki][ki] ) \
140                 shared( TILE_A ) \
141                 firstprivate( ki ) \
142                 priority( nt ) \
143                 label( mkl_dgetrfnpi )
144         LAPACKE_mkl_dgetrfnpi( CblasRowMajor,
145                                tile_size_km, tile_size_kn,
146                                tile_size_k,
147                                TILE_A[ki][ki], tile_size_kn );
148
149         #else
150
151         #pragma oss task inout( TILE_A[ki][ki] ) \
152                 shared( TILE_A ) \
153                 firstprivate( ki ) \
154                 priority( nt ) \
155                 label( dnpgetrf )
156         dnpgetrf( tile_size_km, tile_size_kn,
157                   TILE_A[ki][ki],
158                   tile_size_kn );
159
160         #endif
161
162         for ( mi = ki + 1; mi < mt; mi++ )
163         {
164             tile_size_m = ddss_tile_size( M, mi );
165
166             #pragma oss task in( TILE_A[ki][ki] ) \
167                     inout(TILE_A[mi][ki]) \
168                     shared( TILE_A ) \
169                     firstprivate( mi, ki ) \
170                     priority( nt ) \
171                     label( dtrsm_below )
172             cblas_dtrsm( CblasRowMajor,
173                          ( CBLAS_SIDE ) Right, ( CBLAS_UPLO ) Upper,
174                          ( CBLAS_TRANSPOSE ) NoTrans, ( CBLAS_DIAG ) NonUnit,
175                          tile_size_m, tile_size_kn,
176                          1.0, TILE_A[ki][ki], tile_size_kn,
177                               TILE_A[mi][ki], tile_size_kn );
178         }
179         for ( ni = ki + 1; ni < nt; ni++ )
180         {
181             tile_size_n = ddss_tile_size( N, ni );
182
183             #pragma oss task in( TILE_A[ki][ki] ) \
184                     inout(TILE_A[ki][ni]) \
185                     shared( TILE_A ) \
186                     firstprivate( ni, ki ) \
187                     priority( nt ) \
188                     label( dtrsm_right )
189             cblas_dtrsm( CblasRowMajor,
190                          ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Lower,
191                          ( CBLAS_TRANSPOSE ) NoTrans, ( CBLAS_DIAG ) Unit,
192                          tile_size_km, tile_size_n,
193                          1.0, TILE_A[ki][ki], tile_size_kn,
```

```
194                                 TILE_A[ki][ni], tile_size_n );
195         }
196
197         for ( nni = ki + 1; nni < nt; nni++ )
198         {
199             tile_size_nn = ddss_tile_size( N, nni );
200
201             for ( mmi = ki + 1; mmi < mt; mmi++ )
202             {
203                 tile_size_mm = ddss_tile_size( M, mmi );
204
205                 #pragma oss task in( TILE_A[mmi][ki] ) \
206                     in( TILE_A[ki][nni] ) \
207                     inout( TILE_A[mmi][nni] ) \
208                     shared( TILE_A ) \
209                     firstprivate( nni, mmi, ki ) \
210                     priority( nt - nni ) \
211                     label( dgemm )
212                 cblas_dgemm( CblasRowMajor,
213                              ( CBLAS_TRANSPOSE ) NoTrans,
214                              ( CBLAS_TRANSPOSE ) NoTrans,
215                              tile_size_mm,
216                              tile_size_nn,
217                              TILE_SIZE,
218                              -1.0, TILE_A[mmi][ki],  tile_size_kn,
219                                    TILE_A[ki][nni],  tile_size_nn,
220                               1.0, TILE_A[mmi][nni], tile_size_nn );
221             }
222         }
223     }
224
225     // --From tile data layout to flat data layout--
226     ddss_dtiled2flat( M, N, A, LDA, mt, nt, TILE_A );
227
228     // --Tile A matrix free--
229     free( TILE_A );
230
231     return Success;
232
233 }
```

2.1.2.27 enum LASS_RETURN kdposv (enum DDSS_UPLO *UPLO*, int *N*, int *NRHS*, double * *A*, int *LDA*, double * *B*, int *LDB*)

Solves a system of linear equations A X = B, where A is a M-by-M simmetric positive definite matrix and X and B are M-by-NRHS matrices. The matrix A is descomposed as:

```
A = L \times L^T
or
A = U^T \times U
```

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

Parameters

in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form of A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.
in	*N*	int. N specifies the order of the square matrix A. N >= 0.
in	*NRHS*	int. NRHS specifies the number of right-hand-sides (number of columns of B). NRHS >= 0.
in,out	*A*	double *. A is a pointer to a positive definite matrix of dimension N by LDA. On exit, if return value is Success, the matrix A is overwriten by the factor U or L.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).
in,out	*B*	double *. B is a pointer to a matrix of dimension N by NRHS, which stores the right-hand-sides of the systems of linear equations. (row-major order). On exit, if return value is Success, the matrix B is overwriten by the solution matrix X.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, NRHS).

Return values

Success	sucessful exit
NoSuccess	unsucessful exit

See also

ddss_dpotrf
ddss_tile
ddss_symflat2tiled
ddss_symtiled2flat

Definition at line 95 of file kdposv.c.

References ddss_dflat2tiled(), ddss_dsymflat2tiled(), ddss_dsymtiled2flat_nb(), ddss_dtiled2flat(), and ddss_tile↩
_size().

Referenced by ddss_dposv().

```
99 {
100
101     // Local variables
102     int nt, nrhst;
103     int mi, mmi, ki, ni, nni;
104     int Am, An;
105     int Bm, Bn;
106     int tile_size_m;
107     int tile_size_mm;
108     int tile_size_n;
109     int tile_size_nn;
110     int tile_size_k;
111
112     // Number of tiles
113     if ( N % TILE_SIZE == 0 )
114     {
115         nt = N / TILE_SIZE;
116     }
117     else
118     {
119         nt = ( N / TILE_SIZE ) + 1;
120     }
121     if ( NRHS % TILE_SIZE == 0 )
122     {
123         nrhst = NRHS / TILE_SIZE;
124     }
125     else
126     {
127         nrhst = ( NRHS / TILE_SIZE ) + 1;
128     }
129
130
131     /*****************************
132     --Tile A matrix declaration--
133     *****************************/
134
135     Am = nt;
136     An = nt;
137
138     Bm = nt;
139     Bn = nrhst;
140
141     /*****************************
142     --Tile matrices allocation--
143     *****************************/
144
145     double (*TILE_A)[An][TILE_SIZE * TILE_SIZE] = malloc ( Am * An *
146         TILE_SIZE * TILE_SIZE * sizeof( double ) );
147
148     if ( TILE_A == NULL)
149     {
150         fprintf( stderr, "Failure in kdposv for matrix TILE_A\n" );
151         return NoSuccess;
152     }
153
```

```
154     double ( *TILE_B )[Bn][TILE_SIZE * TILE_SIZE] = malloc(
155              Bm * Bn * TILE_SIZE * TILE_SIZE * sizeof( double ) );
156
157     if ( TILE_B == NULL )
158     {
159         fprintf( stderr, "Failure in kdposv for matrix TILE_B\n" );
160         return NoSuccess;
161     }
162
163     /*********************************************
164     // --From flat data layout to tiled data layout--
165     *********************************************/
166
167     // From flat matrix A to tile matrix TILE_A
168     ddss_dsymflat2tiled( N, N, A, LDA, nt, nt, TILE_A, UPLO );
169
170     // From flat matrix B to tile matrix TILE_B
171     ddss_dflat2tiled( N, NRHS, B, LDB, nt, nrhst, TILE_B );
172
173     /**************
174     --DPOSV tile--
175     **************/
176
177     if ( UPLO == Lower )
178     {
179         // Cholesky descomposition
180         // --UPLO = Lower--
181         for ( ki = 0; ki < nt; ki++ )
182         {
183             tile_size_k = ddss_tile_size( N, ki );
184
185             #pragma oss task inout( TILE_A[ki][ki] ) \
186                 firstprivate( ki ) \
187                 label( dpotrf )
188             LAPACKE_dpotrf_work( LAPACK_ROW_MAJOR,
189                                  'L',
190                                  tile_size_k,
191                                  TILE_A[ki][ki], tile_size_k );
192
193             for ( mi = ki + 1; mi < nt; mi++ )
194             {
195                 tile_size_m = ddss_tile_size( N, mi );
196
197                 #pragma oss task in( TILE_A[ki][ki] ) \
198                     inout(TILE_A[mi][ki]) \
199                     firstprivate( TILE_A ) \
200                     firstprivate( mi, ki ) \
201                     label( dpotrf_dtrsm )
202                 cblas_dtrsm( CblasRowMajor,
203                              ( CBLAS_SIDE ) Right, ( CBLAS_UPLO ) Lower,
204                              ( CBLAS_TRANSPOSE ) Trans, ( CBLAS_DIAG ) NonUnit,
205                              tile_size_m, TILE_SIZE,
206                              1.0, TILE_A[ki][ki], TILE_SIZE,
207                                   TILE_A[mi][ki], TILE_SIZE );
208             }
209             for ( mmi = ki + 1; mmi < nt; mmi++ )
210             {
211                 tile_size_mm = ddss_tile_size( N, mmi );
212
213                 #pragma oss task in( TILE_A[mmi][ki] ) \
214                     inout( TILE_A[mmi][mmi] ) \
215                     firstprivate( mmi, ki ) \
216                     label( dpotrf_dsyrk )
217                 cblas_dsyrk( CblasRowMajor,
218                              ( CBLAS_UPLO ) Lower, ( CBLAS_TRANSPOSE ) NoTrans,
219                              tile_size_mm, TILE_SIZE,
220                              -1.0,  TILE_A[mmi][ki], TILE_SIZE,
221                               1.0, TILE_A[mmi][mmi], tile_size_mm );
222
223                 for ( ni = ki + 1; ni < mmi; ni++ )
224                 {
225
226                     #pragma oss task in( TILE_A[mmi][ki] ) \
227                         in( TILE_A[ni][ki] ) \
228                         inout( TILE_A[mmi][ni] ) \
229                         firstprivate( ni, mmi, ki ) \
230                         label( dpotrf_dgemm )
231                     cblas_dgemm( CblasRowMajor,
232                                  ( CBLAS_TRANSPOSE ) NoTrans,
233                                  ( CBLAS_TRANSPOSE ) Trans,
234                                  TILE_SIZE,
235                                  TILE_SIZE,
236                                  TILE_SIZE,
237                                  -1.0, TILE_A[mmi][ki], TILE_SIZE,
238                                        TILE_A[ni][ki], TILE_SIZE,
239                                   1.0, TILE_A[mmi][ni], TILE_SIZE );
240                 }
```

```
241             }
242         }
243
244         /*******************************************
245         --From tiled data layout to flat data layout--
246         *******************************************/
247         ddss_dsymtiled2flat_nb( N, N, A, LDA, nt, nt, TILE_A, UPLO );
248
249         // Triangular solve
250         // --SIDE = Left & UPLO = Lower & TRANS_A = NoTrans--
251         for ( ki = 0; ki < nt; ki++ )
252         {
253             tile_size_k = ddss_tile_size( N, ki );
254
255             for ( ni = 0; ni < nrhst; ni++ )
256             {
257                 tile_size_n = ddss_tile_size( NRHS, ni );
258
259                 #pragma oss task in( TILE_A[ki][ki] ) \
260                     inout(  TILE_B[ki][ni] ) \
261                     shared( TILE_A, TILE_B ) \
262                     firstprivate( ki, ni ) \
263                     label( dtrsm_dtrsm1 )
264                 cblas_dtrsm( CblasRowMajor,
265                              ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Lower,
266                              ( CBLAS_TRANSPOSE ) NoTrans,
267                              ( CBLAS_DIAG ) NonUnit,
268                              tile_size_k,
269                              tile_size_n,
270                              1.0, TILE_A[ki][ki], tile_size_k,
271                                   TILE_B[ki][ni], tile_size_n );
272             }
273             for ( nni = 0; nni < nrhst; nni++ )
274             {
275                 tile_size_nn = ddss_tile_size( NRHS, nni );
276
277                 for ( mi = ki + 1; mi < nt; mi++ )
278                 {
279
280                     #pragma oss task in( TILE_A[mi][ki] ) \
281                         in( TILE_B[ki][nni] ) \
282                         inout( TILE_B[mi][nni] ) \
283                         shared( TILE_A, TILE_B ) \
284                         firstprivate( ki, mi, nni ) \
285                         label( dtrsm_dgemm1 )
286                     cblas_dgemm( CblasRowMajor,
287                                  CblasNoTrans, CblasNoTrans,
288                                  tile_size_k,
289                                  tile_size_nn,
290                                  tile_size_k,
291                                  -1.0,  TILE_A[mi][ki],  tile_size_k,
292                                         TILE_B[ki][nni], tile_size_nn,
293                                   1.0, TILE_B[mi][nni], tile_size_nn );
294                 }
295             }
296         }
297
298         // Triangular solve
299         // --SIDE = Left & UPLO = Lower & TRANS_A = Trans--
300         for ( ki = 0; ki < nt; ki++ )
301         {
302             tile_size_k = ddss_tile_size( N, ki );
303
304             for ( ni = 0; ni < nrhst; ni++ )
305             {
306                 tile_size_n = ddss_tile_size( NRHS, ni );
307
308                 #pragma oss task in( TILE_A[ki][ki] ) \
309                     inout( TILE_B[ki][ni] ) \
310                     shared( TILE_A, TILE_B ) \
311                     firstprivate( ki, ni ) \
312                     label( dtrsm_dtrsm2 )
313                 cblas_dtrsm( CblasRowMajor,
314                              ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Lower,
315                              ( CBLAS_TRANSPOSE ) Trans,
316                              ( CBLAS_DIAG ) NonUnit,
317                              tile_size_k,
318                              tile_size_n,
319                              1.0,   TILE_A[ki][ki], tile_size_k,
320                                     TILE_B[ki][ni], tile_size_n );
321             }
322
323             for ( nni = 0; nni < nrhst; nni++ )
324             {
325                 tile_size_nn = ddss_tile_size( NRHS, nni );
326
327                 for ( mi = ki - 1; mi >= 0; mi-- )
```

```
328                 {
329                     tile_size_m = ddss_tile_size( N, mi );
330 
331                     #pragma oss task in( TILE_A[ki][mi] ) \
332                         in( TILE_B[ki][nni] ) \
333                         inout(  TILE_B[mi][nni] ) \
334                         shared( TILE_A, TILE_B ) \
335                         firstprivate( ki, mi, nni ) \
336                         label( dtrsm_dgemm2 )
337                     cblas_dgemm( CblasRowMajor,
338                                  CblasTrans, CblasNoTrans,
339                                  tile_size_m,
340                                  tile_size_nn,
341                                  tile_size_k,
342                                  -1.0,  TILE_A[ki][mi],  tile_size_m,
343                                         TILE_B[ki][nni], tile_size_nn,
344                                   1.0, TILE_B[mi][nni], tile_size_nn );
345                 }
346             }
347         }
348     }
349     else
350     {
351         // Cholesky descomposition
352         // --UPLO = Upper--
353         for ( ki = 0; ki < nt; ki++ )
354         {
355             tile_size_k = ddss_tile_size( N, ki );
356 
357             #pragma oss task inout( TILE_A[ki][ki] ) \
358                 shared( TILE_A ) \
359                 firstprivate( ki ) \
360                 label( dpotrf )
361             LAPACKE_dpotrf_work( LAPACK_ROW_MAJOR,
362                                  'U',
363                                  tile_size_k, TILE_A[ki][ki],
364                                  tile_size_k );
365 
366             for ( mi = ki + 1; mi < nt; mi++ )
367             {
368                 tile_size_m = ddss_tile_size( N, mi );
369 
370                 #pragma oss task in( TILE_A[ki][ki] ) \
371                     inout(TILE_A[ki][mi]) \
372                     shared( TILE_A ) \
373                     firstprivate( mi, ki ) \
374                     label( dpotrf_dtrsm )
375                 cblas_dtrsm( CblasRowMajor,
376                              ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Upper,
377                              ( CBLAS_TRANSPOSE ) Trans, ( CBLAS_DIAG ) NonUnit,
378                              TILE_SIZE, tile_size_m,
379                              1.0, TILE_A[ki][ki], TILE_SIZE,
380                                   TILE_A[ki][mi], tile_size_m );
381             }
382             for ( mmi = ki + 1; mmi < nt; mmi++ )
383             {
384                 tile_size_mm = ddss_tile_size( N, mmi );
385 
386                 #pragma oss task in( TILE_A[ki][mmi] ) \
387                     inout( TILE_A[mmi][mmi] ) \
388                     shared( TILE_A ) \
389                     firstprivate( mmi, ki ) \
390                     label( dpotrf_dsyrk )
391                 cblas_dsyrk( CblasRowMajor,
392                              ( CBLAS_UPLO ) Upper, ( CBLAS_TRANSPOSE ) Trans,
393                              tile_size_mm, TILE_SIZE,
394                              -1.0,  TILE_A[ki][mmi], tile_size_mm,
395                               1.0, TILE_A[mmi][mmi], tile_size_mm );
396 
397                 for ( ni = ki + 1; ni < mmi; ni++ )
398                 {
399                     #pragma oss task in( TILE_A[ki][ni] ) \
400                         in( TILE_A[ki][mmi] ) \
401                         inout( TILE_A[ni][mmi] ) \
402                         shared( TILE_A ) \
403                         firstprivate( ni, mmi, ki ) \
404                         label( dpotrf_dgemm )
405                     cblas_dgemm( CblasRowMajor,
406                                  ( CBLAS_TRANSPOSE ) Trans,
407                                  ( CBLAS_TRANSPOSE ) NoTrans,
408                                  TILE_SIZE,
409                                  TILE_SIZE,
410                                  TILE_SIZE,
411                                  -1.0, TILE_A[ki][ni],  TILE_SIZE,
412                                        TILE_A[ki][mmi], TILE_SIZE,
413                                   1.0, TILE_A[ni][mmi], TILE_SIZE );
414                 }
```

```
415             }
416         }
417         /*****************************************
418         --From tiled data layout to flat data layout--
419          *****************************************/
420         ddss_dsymtiled2flat_nb( N, N, A, LDA, nt, nt, TILE_A, UPLO );
421         // Triangular solve
422         // --SIDE = Left & UPLO = Upper & TRANS_A = Trans--
423         for ( ki = 0; ki < nt; ki++ )
424         {
425             tile_size_k = ddss_tile_size( N, ki );
426
427             for ( ni = 0; ni < nrhst; ni++ )
428             {
429                 tile_size_n = ddss_tile_size( NRHS, ni );
430
431                 #pragma oss task in( TILE_A[ki][ki] ) \
432                     inout( TILE_B[ki][ni] ) \
433                     shared( TILE_A, TILE_B ) \
434                     firstprivate( ki, ni ) \
435                     label( dtrsm_dtrsm1 )
436                 cblas_dtrsm( CblasRowMajor,
437                              ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Lower,
438                              ( CBLAS_TRANSPOSE ) Trans,
439                              ( CBLAS_DIAG ) NonUnit,
440                              tile_size_k,
441                              tile_size_n,
442                              1.0, TILE_A[ki][ki], tile_size_k,
443                                   TILE_B[ki][ni], tile_size_n );
444             }
445
446             for ( mi = ki + 1; mi < nt; mi++ )
447             {
448                 tile_size_m = ddss_tile_size( N, mi );
449
450                 for ( nni = 0; nni < nrhst; nni++ )
451                 {
452                     tile_size_nn = ddss_tile_size( NRHS, nni );
453
454                     #pragma oss task in( TILE_A[ki][mi]) \
455                         in( TILE_B[ki][nni] ) \
456                         inout(  TILE_B[mi][nni] ) \
457                         shared( TILE_A, TILE_B ) \
458                         firstprivate( ki, mi, nni ) \
459                         label( dtrsm_dgemm1 )
460                     cblas_dgemm( CblasRowMajor,
461                                  CblasTrans, CblasNoTrans,
462                                  tile_size_m,
463                                  tile_size_nn,
464                                  tile_size_k,
465                                  -1.0,  TILE_A[ki][mi],  tile_size_m,
466                                         TILE_B[ki][nni], tile_size_nn,
467                                   1.0, TILE_B[mi][nni], tile_size_nn );
468                 }
469             }
470         }
471         // Triangular solve
472         // --SIDE = Left & UPLO = Upper & TRANS_A = NoTrans--
473         for ( ki = 0; ki < nt; ki++ )
474         {
475             tile_size_k = ddss_tile_size( N, ki );
476
477             for ( ni = 0; ni < nrhst; ni++ )
478             {
479                 tile_size_n = ddss_tile_size( NRHS, ni );
480
481                 #pragma oss task in( TILE_A[ki][ki] ) \
482                     inout( TILE_B[ki][ni] ) \
483                     shared( TILE_A, TILE_B ) \
484                     firstprivate( ki, ni ) \
485                     label( dtrsm_dtrsm2 )
486                 cblas_dtrsm( CblasRowMajor,
487                              ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Lower,
488                              ( CBLAS_TRANSPOSE ) NoTrans,
489                              ( CBLAS_DIAG ) NonUnit,
490                              tile_size_k,
491                              tile_size_n,
492                              1.0, TILE_A[ki][ki], tile_size_k,
493                                   TILE_B[ki][ni], tile_size_n );
494             }
495
496             for ( nni = 0; nni < nrhst; nni++ )
497             {
498                 tile_size_nn = ddss_tile_size( NRHS, nni );
499
500                 for ( mi = ki - 1; mi >= 0; mi-- )
501                 {
```

```
502                     tile_size_m = ddss_tile_size( N, mi );
503
504                     #pragma oss task in( TILE_A[mi][ki] ) \
505                         in( TILE_B[ki][nni] ) \
506                         inout( TILE_B[mi][nni] ) \
507                         shared( TILE_A, TILE_B ) \
508                         firstprivate( ki, mi, nni ) \
509                         label( dtrsm_dgemm2 )
510                     cblas_dgemm( CblasRowMajor,
511                                  CblasNoTrans, CblasNoTrans,
512                                  tile_size_m,
513                                  tile_size_nn,
514                                  tile_size_k,
515                                  -1.0,  TILE_A[mi][ki],  tile_size_k,
516                                         TILE_B[ki][nni], tile_size_nn,
517                                   1.0, TILE_B[mi][nni], tile_size_nn );
518                 }
519             }
520         }
521     }
522
523     /*****************************************
524     --From tiled data layout to flat data layout--
525     *****************************************/
526     ddss_dtiled2flat( N, NRHS, B, LDB, nt, nrhst, TILE_B );
527
528     // --Tile A and B matrices free--
529     free( TILE_A );
530     free( TILE_B );
531
532     return Success;
533
534 }
```

2.1.2.28 enum LASS_RETURN kdpotrf (enum DDSS_UPLO *UPLO*, int *N*, double * *A*, int *LDA*)

Performs the Cholesky factorization of a symmetric positive definite matrix A:

```
A = L \times L^T
or
A = U^T \times U
```

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

Parameters

in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form of A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.
in	*N*	int. N specifies the order of the square matrix A. N >= 0.
in,out	*A*	double *. A is a pointer to a positive definite matrix of dimension N by LDA. On exit, if return value is Success, the matrix A is overwriten by the factor U or L.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).

Return values

Success	sucessful exit
NoSuccess	unsucessful exit

See also

ddss_dpotrf
ddss_tile
ddss_symflat2tiled
ddss_symtiled2flat

Definition at line 78 of file kdpotrf.c.

References ddss_dsymflat2tiled(), ddss_dsymtiled2flat(), and ddss_tile_size().

Referenced by ddss_dpotrf().

```
79 {
80
81     // Local variables
82     int nt;
83     int mi, mmi, ki, ni;
84     int Am;
85     int An;
86     int tile_size_m;
87     int tile_size_mm;
88     int tile_size_n;
89     int tile_size_k;
90     int k_check;
91     double betat;
92
93     // Number of tiles
94     if ( N % TILE_SIZE == 0 )
95     {
96         nt = N / TILE_SIZE;
97     }
98     else
99     {
100         nt = ( N / TILE_SIZE ) + 1;
101     }
102
103     /****************************
104     --Tile A matrix declaration--
105     ****************************/
106
107     Am = nt;
108     An = nt;
109
110     /***************************
111     --Tile A matrix allocation--
112     ***************************/
113
114     double (*TILE_A)[An][TILE_SIZE * TILE_SIZE] = malloc ( Am * An *
115         TILE_SIZE * TILE_SIZE * sizeof( double ) );
116
117     if ( TILE_A == NULL)
118     {
119         fprintf( stderr, "Failure in kdpotrf for matrix TILE_A\n" );
120         return NoSuccess;
121     }
122
123     /***********************************************
124     // --From flat data layout to tiled data layout--
125     ***********************************************/
126
127     ddss_dsymflat2tiled( N, N, A, LDA, nt, nt, TILE_A, UPLO );
128
129     /**************
130     --DPOTRF tile--
131     **************/
132
133     if ( UPLO == Lower )
134     {
135         // --UPLO = Lower--
136         for ( ki = 0; ki < nt; ki++ )
137         {
138             tile_size_k = ddss_tile_size( N, ki );
139
140             #pragma oss task inout( TILE_A[ki][ki] ) \
141                 firstprivate( ki ) \
142                 label( dpotrf )
143             LAPACKE_dpotrf_work( LAPACK_ROW_MAJOR,
144                                  'L',
145                                  tile_size_k, TILE_A[ki][ki], tile_size_k );
```

```
146
147             for ( mi = ki + 1; mi < nt; mi++ )
148             {
149                 tile_size_m = ddss_tile_size( N, mi );
150
151                 #pragma oss task in( TILE_A[ki][ki] ) \
152                     inout(TILE_A[mi][ki]) \
153                     firstprivate( mi, ki ) \
154                     label( dtrsm )
155                 cblas_dtrsm( CblasRowMajor,
156                              ( CBLAS_SIDE ) Right, ( CBLAS_UPLO ) Lower,
157                              ( CBLAS_TRANSPOSE ) Trans, ( CBLAS_DIAG ) NonUnit,
158                              tile_size_m, TILE_SIZE,
159                              1.0, TILE_A[ki][ki], TILE_SIZE,
160                                   TILE_A[mi][ki], TILE_SIZE );
161             }
162             for ( mmi = ki + 1; mmi < nt; mmi++ )
163             {
164                 tile_size_mm = ddss_tile_size( N, mmi );
165
166                 #pragma oss task in( TILE_A[mmi][ki] ) \
167                     inout( TILE_A[mmi][mmi] ) \
168                     firstprivate( mmi, ki ) \
169                     label( dsyrk )
170                 cblas_dsyrk( CblasRowMajor,
171                              ( CBLAS_UPLO ) Lower, ( CBLAS_TRANSPOSE ) NoTrans,
172                              tile_size_mm, TILE_SIZE,
173                              -1.0,  TILE_A[mmi][ki], TILE_SIZE,
174                               1.0, TILE_A[mmi][mmi], tile_size_mm );
175
176                 for ( ni = ki + 1; ni < mmi; ni++ )
177                 {
178
179                     #pragma oss task in( TILE_A[mmi][ki] ) \
180                         in( TILE_A[ni][ki] ) \
181                         inout( TILE_A[mmi][ni] ) \
182                         firstprivate( ni, mmi, ki ) \
183                         label( dgemm )
184                         cblas_dgemm( CblasRowMajor,
185                                      ( CBLAS_TRANSPOSE ) NoTrans,
186                                      ( CBLAS_TRANSPOSE ) Trans,
187                                      TILE_SIZE,
188                                      TILE_SIZE,
189                                      TILE_SIZE,
190                                      -1.0, TILE_A[mmi][ki], TILE_SIZE,
191                                            TILE_A[ni][ki], TILE_SIZE,
192                                       1.0, TILE_A[mmi][ni], TILE_SIZE );
193                 }
194             }
195         }
196     }
197     else
198     {
199         // --UPLO = Upper--
200         for ( ki = 0; ki < nt; ki++ )
201         {
202             tile_size_k = ddss_tile_size( N, ki );
203
204             #pragma oss task inout( TILE_A[ki][ki] ) \
205                 shared( TILE_A ) \
206                 firstprivate( ki ) \
207                 label( dpotrf )
208             LAPACKE_dpotrf_work( LAPACK_ROW_MAJOR,
209                                  'U',
210                                  tile_size_k, TILE_A[ki][ki],
211                                  tile_size_k );
212
213             for ( mi = ki + 1; mi < nt; mi++ )
214             {
215                 tile_size_m = ddss_tile_size( N, mi );
216
217                 #pragma oss task in( TILE_A[ki][ki] ) \
218                     inout(TILE_A[ki][mi]) \
219                     shared( TILE_A ) \
220                     firstprivate( mi, ki ) \
221                     label( dtrsm )
222                 cblas_dtrsm( CblasRowMajor,
223                              ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Upper,
224                              ( CBLAS_TRANSPOSE ) Trans, ( CBLAS_DIAG ) NonUnit,
225                              TILE_SIZE, tile_size_m,
226                              1.0, TILE_A[ki][ki], TILE_SIZE,
227                                   TILE_A[ki][mi], tile_size_m );
228             }
229             for ( mmi = ki + 1; mmi < nt; mmi++ )
230             {
231                 tile_size_mm = ddss_tile_size( N, mmi );
232
```

```
233                 #pragma oss task in( TILE_A[ki][mmi] ) \
234                     inout( TILE_A[mmi][mmi] ) \
235                     shared( TILE_A ) \
236                     firstprivate( mmi, ki ) \
237                     label( dsyrk )
238                 cblas_dsyrk( CblasRowMajor,
239                              ( CBLAS_UPLO ) Upper, ( CBLAS_TRANSPOSE ) Trans,
240                              tile_size_mm, TILE_SIZE,
241                              -1.0,  TILE_A[ki][mmi], tile_size_mm,
242                               1.0, TILE_A[mmi][mmi], tile_size_mm );
243
244                 for ( ni = ki + 1; ni < mmi; ni++ )
245                 {
246                     #pragma oss task in( TILE_A[ki][ni] ) \
247                         in( TILE_A[ki][mmi] ) \
248                         inout( TILE_A[ni][mmi] ) \
249                         shared( TILE_A ) \
250                         firstprivate( ni, mmi, ki ) \
251                         label( dgemm )
252                         cblas_dgemm( CblasRowMajor,
253                                      ( CBLAS_TRANSPOSE ) Trans,
254                                      ( CBLAS_TRANSPOSE ) NoTrans,
255                                      TILE_SIZE,
256                                      TILE_SIZE,
257                                      TILE_SIZE,
258                                      -1.0, TILE_A[ki][ni],  TILE_SIZE,
259                                            TILE_A[ki][mmi], TILE_SIZE,
260                                       1.0, TILE_A[ni][mmi], TILE_SIZE );
261                 }
262             }
263         }
264     }
265
266     // --From tile data layout to flat data layout--
267     ddss_dsymtiled2flat( N, N, A, LDA, nt, nt, TILE_A, UPLO );
268
269     // --Tile A matrix free--
270     free( TILE_A );
271
272     return Success;
273
274 }
```

2.1.2.29 enum LASS_RETURN kdsymm (enum DDSS_SIDE *SIDE*, enum DDSS_UPLO *UPLO*, int *M*, int *N*, const double *ALPHA*, double * *A*, int *LDA*, double * *B*, int *LDB*, const double *BETA*, double * *C*, int *LDC*)

Performs one of the matrix-matrix operations:

```
C = ALPHA * A * B + BETA * C
```

or

```
C = ALPHA * B * A + BETA * C
```

where op(X) is one of:

```
op( X ) = X      or
op( X ) = X**T
```

ALPHA and BETA are scalars, A is a symmetric matrix, and B and C are M by N matrices.

Parameters

in	*SIDE*	enum DDSS_SIDE. UPLO specifies the position of the symmetric A matrix in the operation: • Left: C = ALPHA ∗ A ∗ B + BETA ∗ C • Right: C = ALPHA ∗ B ∗ A + BETA ∗ C
in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form of A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.
in	*M*	int. M specifies the number of rows of the matrix C. M must be equal or greater than zero.
in	*N*	int. N specifies the number of columns of the matrix C. N must be equal or greater than zero.
in	*ALPHA*	double.
in	*A*	double ∗. A is a pointer to a matrix of dimension Ma (rows) by Na (columns), where Ma is M and Na is M when SIDE = Left, and Ma is N and Na is N when SIDE = Right
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, Na).
in	*B*	double ∗. B is a pointer to a matrix of dimension M by N.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, N).
in	*BETA*	double.
in,out	*C*	double ∗. C is a pointer to a matrix of dimension M by N. On exit, C is overwritten by the M by N matrix.
in	*LDC*	int. LDC specifies the number of columns of C (row-major order). LDC must be at least max(1, N).

Return values

Success	sucessful exit
NoSuccess	unsucessful exit

See also

ddss_dgemm
ddss_tile
ddss_flat2tiled
ddss_symflat2tiled
ddss_tiled2flat
ddss_symtiled2flat

Definition at line 125 of file kdsymm.c.

References ddss_dflat2tiled(), ddss_dsymflat2tiled(), ddss_dtiled2flat(), and ddss_tile_size().

Referenced by ddss_dsymm().

```
130 {
131
132     // Local variables
133     int mt, kt, nt;
```

```
134     int mi, ki, ni;
135     int Am, An;
136     int Amt, Ant;
137     int tile_size_m;
138     int tile_size_n;
139     int tile_size_k;
140     int k_check;
141     double betat;
142 
143     // Number of tiles
144     if ( M % TILE_SIZE == 0 )
145     {
146         mt = M / TILE_SIZE;
147     }
148     else
149     {
150         mt = ( M / TILE_SIZE ) + 1;
151     }
152 
153     if ( N % TILE_SIZE == 0 )
154     {
155         nt = N / TILE_SIZE;
156     }
157     else
158     {
159         nt = ( N / TILE_SIZE ) + 1;
160     }
161 
162     if ( SIDE == Left )
163     {
164         if ( M % TILE_SIZE == 0 )
165         {
166             kt = M / TILE_SIZE;
167         }
168         else
169         {
170             kt = ( M / TILE_SIZE ) + 1;
171         }
172     }
173     else
174     {
175         if ( N % TILE_SIZE == 0 )
176         {
177             kt = N / TILE_SIZE;
178         }
179         else
180         {
181             kt = ( N / TILE_SIZE ) + 1;
182         }
183     }
184 
185     /****************************
186     --Tile matrices declaration--
187     ****************************/
188 
189     if ( SIDE == Left )
190     {
191         Am = M;
192         An = M;
193         Amt = mt;
194         Ant = mt;
195     }
196     else
197     {
198         Am = N;
199         An = N;
200         Amt = nt;
201         Ant = nt;
202     }
203 
204     /****************************
205     --Tile matrices allocation--
206     ****************************/
207 
208     double (*TILE_A)[Ant][TILE_SIZE * TILE_SIZE] = malloc ( Amt * Ant *
209         TILE_SIZE * TILE_SIZE * sizeof( double ) );
210 
211     if ( TILE_A == NULL)
212     {
213         fprintf( stderr, "Failure in kdsymm for matrix TILE_A\n" );
214         return NoSuccess;
215     }
216 
217     double (*TILE_B)[nt][TILE_SIZE * TILE_SIZE] = malloc ( mt * nt *
218         TILE_SIZE * TILE_SIZE * sizeof( double ) );
219 
220     if ( TILE_B == NULL)
```

```
221     {
222         fprintf( stderr, "Failure in kdsymm for matrix TILE_B\n" );
223         return NoSuccess;
224     }
225
226     double (*TILE_C)[nt][TILE_SIZE * TILE_SIZE] = malloc ( mt * nt *
227         TILE_SIZE * TILE_SIZE * sizeof( double ) );
228
229     if ( TILE_C == NULL)
230     {
231         fprintf( stderr, "Failure in kdsymm for matrix TILE_C\n" );
232         return NoSuccess;
233     }
234
235     /***********************************************
236     // --From flat data layout to tiled data layout--
237     ***********************************************/
238
239     // From flat and symmetric matrix A to tile matrix TILE_A
240     ddss_dsymflat2tiled( Am, An, A, LDA, Amt, Ant, TILE_A, UPLO );
241
242     // From flat matrix B to tile matrix TILE_B
243     ddss_dflat2tiled( M, N, B, LDB, mt, nt, TILE_B );
244
245     // From flat matrix C to tile matrix TILE_C
246     ddss_dflat2tiled( M, N, C, LDC, mt, nt, TILE_C );
247
248     /*************
249     --DSYMM tile--
250     *************/
251
252     for ( mi = 0; mi < mt; mi++ )
253     {
254         tile_size_m  = ddss_tile_size( M, mi );
255         for ( ni = 0; ni < nt; ni++ )
256         {
257             tile_size_n  = ddss_tile_size( N, ni );
258             if ( SIDE == Left )
259             {
260                 // --SIDE = Left & UPLO = Lower--
261                 if ( UPLO == Lower)
262                 {
263                     for ( ki = 0; ki < kt; ki++ )
264                     {
265                         if (ki == 0)
266                         {
267                             betat = BETA;
268                         }
269                         else
270                         {
271                             betat = 1.0;
272                         }
273                         tile_size_k  = ddss_tile_size( M, ki );
274
275                         if ( ki < mi )
276                         {
277                             #pragma oss task in( TILE_A[mi][ki] ) \
278                                 in( TILE_B[ki][ni] ) \
279                                 inout( TILE_C[mi][ni] ) \
280                                 shared( TILE_A, TILE_B, TILE_C ) \
281                                 firstprivate( mi, ni, ki, betat )
282                             cblas_dgemm( CblasRowMajor,
283                                         ( CBLAS_TRANSPOSE ) NoTrans,
284                                         ( CBLAS_TRANSPOSE ) NoTrans,
285                                         tile_size_m,
286                                         tile_size_n,
287                                         tile_size_k,
288                                         ALPHA, TILE_A[mi][ki], tile_size_k,
289                                                TILE_B[ki][ni], tile_size_n,
290                                         betat, TILE_C[mi][ni], tile_size_n );
291                         }
292                         else
293                         {
294                             if ( ki == mi )
295                             {
296                                 #pragma oss task in( TILE_A[ki][ki] ) \
297                                     in( TILE_B[ki][ni] ) \
298                                     inout( TILE_C[mi][ni] ) \
299                                     shared( TILE_A, TILE_B, TILE_C ) \
300                                     firstprivate( mi, ni, ki, betat )
301                                 cblas_dsymm( CblasRowMajor,
302                                             ( CBLAS_SIDE ) SIDE,
303                                             ( CBLAS_UPLO ) UPLO,
304                                             tile_size_m, tile_size_n,
305                                             ALPHA, TILE_A[ki][ki], tile_size_k,
306                                                    TILE_B[ki][ni], tile_size_n,
307                                             betat, TILE_C[mi][ni],
```

```
308                                         tile_size_n );
309                         }
310                         else
311                         {
312                             #pragma oss task in( TILE_A[ki][mi] ) \
313                                 in( TILE_B[ki][ni] ) \
314                                 inout( TILE_C[mi][ni] ) \
315                                 shared( TILE_A, TILE_B, TILE_C ) \
316                                 firstprivate( mi, ni, ki, betat )
317                             cblas_dgemm( CblasRowMajor,
318                                         ( CBLAS_TRANSPOSE ) Trans,
319                                         ( CBLAS_TRANSPOSE ) NoTrans,
320                                         tile_size_m,
321                                         tile_size_n,
322                                         tile_size_k,
323                                         ALPHA, TILE_A[ki][mi], tile_size_m,
324                                                TILE_B[ki][ni], tile_size_n,
325                                         betat, TILE_C[mi][ni],
326                                         tile_size_n );
327                         }
328                     }
329                 }
330             }
331             // --SIDE = Left & UPLO = Upper--
332             else
333             {
334                 for ( ki = 0; ki < kt; ki++ )
335                 {
336                     if (ki == 0)
337                     {
338                         betat = BETA;
339                     }
340                     else
341                     {
342                         betat = 1.0;
343                     }
344                     tile_size_k  = ddss_tile_size( M, ki );
345 
346                     if ( ki < mi )
347                     {
348                         #pragma oss task in( TILE_A[ki][mi] ) \
349                             in( TILE_B[ki][ni] ) \
350                             inout( TILE_C[mi][ni] ) \
351                             shared( TILE_A, TILE_B, TILE_C ) \
352                             firstprivate( mi, ni, ki, betat )
353                         cblas_dgemm( CblasRowMajor,
354                                     ( CBLAS_TRANSPOSE ) Trans,
355                                     ( CBLAS_TRANSPOSE ) NoTrans,
356                                     tile_size_m,
357                                     tile_size_n,
358                                     tile_size_k,
359                                     ALPHA, TILE_A[ki][mi], tile_size_m,
360                                            TILE_B[ki][ni], tile_size_n,
361                                     betat, TILE_C[mi][ni], tile_size_n );
362                     }
363                     else
364                     {
365                         if ( ki == mi )
366                         {
367                             #pragma oss task in( TILE_A[ki][ki] ) \
368                                 in( TILE_B[ki][ni] ) \
369                                 inout( TILE_C[mi][ni] ) \
370                                 shared( TILE_A, TILE_B, TILE_C ) \
371                                 firstprivate( mi, ni, ki, betat )
372                             cblas_dsymm( CblasRowMajor,
373                                         ( CBLAS_SIDE ) SIDE,
374                                         ( CBLAS_UPLO ) UPLO,
375                                         tile_size_m, tile_size_n,
376                                         ALPHA, TILE_A[ki][ki], tile_size_k,
377                                                TILE_B[ki][ni], tile_size_n,
378                                         betat, TILE_C[mi][ni],
379                                         tile_size_n );
380                         }
381                         else
382                         {
383                             #pragma oss task in( TILE_A[mi][ki] ) \
384                                 in( TILE_B[ki][ni] ) \
385                                 inout( TILE_C[mi][ni] ) \
386                                 shared( TILE_A, TILE_B, TILE_C ) \
387                                 firstprivate( mi, ni, ki, betat )
388                             cblas_dgemm( CblasRowMajor,
389                                         ( CBLAS_TRANSPOSE ) NoTrans,
390                                         ( CBLAS_TRANSPOSE ) NoTrans,
391                                         tile_size_m,
392                                         tile_size_n,
393                                         tile_size_k,
394                                         ALPHA, TILE_A[mi][ki], tile_size_k,
```

```
395                                                 TILE_B[ki][ni], tile_size_n,
396                                             betat, TILE_C[mi][ni],
397                                             tile_size_n );
398                             }
399                         }
400                     }
401                 }
402             }
403             else
404             {
405                 // --SIDE = Right & UPLO = Lower--
406                 if ( UPLO == Lower)
407                 {
408                     for ( ki = 0; ki < kt; ki++ )
409                     {
410                         if (ki == 0)
411                         {
412                             betat = BETA;
413                         }
414                         else
415                         {
416                             betat = 1.0;
417                         }
418                         tile_size_k  = ddss_tile_size( N, ki );
419 
420                         if ( ki < ni )
421                         {
422                             #pragma oss task in( TILE_B[mi][ki] ) \
423                                 in( TILE_A[ni][ki] ) \
424                                 inout( TILE_C[mi][ni] ) \
425                                 shared( TILE_A, TILE_B, TILE_C ) \
426                                 firstprivate( mi, ni, ki, betat )
427                             cblas_dgemm( CblasRowMajor,
428                                         ( CBLAS_TRANSPOSE ) NoTrans,
429                                         ( CBLAS_TRANSPOSE ) Trans,
430                                         tile_size_m,
431                                         tile_size_n,
432                                         tile_size_k,
433                                         ALPHA, TILE_B[mi][ki], tile_size_k,
434                                                TILE_A[ni][ki], tile_size_k,
435                                         betat, TILE_C[mi][ni], tile_size_n );
436                         }
437                         else
438                         {
439                             if ( ki == ni )
440                             {
441                                 #pragma oss task in( TILE_A[ki][ki] ) \
442                                     in( TILE_B[mi][ki] ) \
443                                     inout( TILE_C[mi][ni] ) \
444                                     shared( TILE_A, TILE_B, TILE_C ) \
445                                     firstprivate( mi, ni, ki, betat )
446                                 cblas_dsymm( CblasRowMajor,
447                                             ( CBLAS_SIDE ) SIDE,
448                                             ( CBLAS_UPLO ) UPLO,
449                                             tile_size_m, tile_size_n,
450                                             ALPHA, TILE_A[ki][ki], tile_size_k,
451                                                    TILE_B[mi][ki], tile_size_k,
452                                             betat, TILE_C[mi][ni],
453                                             tile_size_n );
454                             }
455                             else
456                             {
457                                 #pragma oss task in( TILE_B[mi][ki] ) \
458                                     in( TILE_A[ki][ni] ) \
459                                     inout( TILE_C[mi][ni] ) \
460                                     shared( TILE_A, TILE_B, TILE_C ) \
461                                     firstprivate( mi, ni, ki, betat )
462                                 cblas_dgemm( CblasRowMajor,
463                                             ( CBLAS_TRANSPOSE ) NoTrans,
464                                             ( CBLAS_TRANSPOSE ) NoTrans,
465                                             tile_size_m,
466                                             tile_size_n,
467                                             tile_size_k,
468                                             ALPHA, TILE_B[mi][ki], tile_size_k,
469                                                    TILE_A[ki][ni], tile_size_n,
470                                             betat, TILE_C[mi][ni],
471                                             tile_size_n );
472                             }
473                         }
474                     }
475                 }
476                 // --SIDE = Right & UPLO = Upper--
477                 else
478                 {
479                     for ( ki = 0; ki < kt; ki++ )
480                     {
481                         if (ki == 0)
```

```
482                         {
483                             betat = BETA;
484                         }
485                         else
486                         {
487                             betat = 1.0;
488                         }
489                         tile_size_k  = ddss_tile_size( N, ki );
490 
491                         if ( ki < ni )
492                         {
493                             #pragma oss task in( TILE_B[mi][ki] ) \
494                                 in( TILE_A[ki][ni] ) \
495                                 inout( TILE_C[mi][ni] ) \
496                                 shared( TILE_A, TILE_B, TILE_C ) \
497                                 firstprivate( mi, ni, ki, betat )
498                             cblas_dgemm( CblasRowMajor,
499                                         ( CBLAS_TRANSPOSE ) NoTrans,
500                                         ( CBLAS_TRANSPOSE ) NoTrans,
501                                         tile_size_m,
502                                         tile_size_n,
503                                         tile_size_k,
504                                         ALPHA, TILE_B[mi][ki], tile_size_k,
505                                                TILE_A[ki][ni], tile_size_n,
506                                         betat, TILE_C[mi][ni], tile_size_n );
507                         }
508                         else
509                         {
510                             if ( ki == ni )
511                             {
512                                 #pragma oss task in( TILE_A[ki][ki] ) \
513                                     in( TILE_B[ki][ni] ) \
514                                     inout( TILE_C[mi][ni] ) \
515                                     shared( TILE_A, TILE_B, TILE_C ) \
516                                     firstprivate( mi, ni, ki, betat )
517                                 cblas_dsymm( CblasRowMajor,
518                                             ( CBLAS_SIDE ) SIDE,
519                                             ( CBLAS_UPLO ) UPLO,
520                                             tile_size_m, tile_size_n,
521                                             ALPHA, TILE_A[ki][ki], tile_size_k,
522                                                    TILE_B[mi][ki], tile_size_k,
523                                             betat, TILE_C[mi][ni],
524                                             tile_size_n );
525                             }
526                             else
527                             {
528                                 #pragma oss task in( TILE_B[mi][ki] ) \
529                                     in( TILE_A[ni][ki] ) \
530                                     inout( TILE_C[mi][ni] ) \
531                                     shared( TILE_A, TILE_B, TILE_C ) \
532                                     firstprivate( mi, ni, ki, betat )
533                                 cblas_dgemm( CblasRowMajor,
534                                             ( CBLAS_TRANSPOSE ) NoTrans,
535                                             ( CBLAS_TRANSPOSE ) Trans,
536                                             tile_size_m,
537                                             tile_size_n,
538                                             tile_size_k,
539                                             ALPHA, TILE_B[mi][ki], tile_size_k,
540                                                    TILE_A[ni][ki], tile_size_k,
541                                             betat, TILE_C[mi][ni],
542                                             tile_size_n );
543                             }
544                         }
545                     }
546                 }
547             }
548         }
549     }
550 
551     // From tile matrix TILE_C to flat matrix C
552     ddss_dtiled2flat( M, N, C, LDC, mt, nt, TILE_C );
553 
554     // --Tile matrices free--
555     free( TILE_A );
556     free( TILE_B );
557     free( TILE_C );
558 
559     return Success;
560 
561 }
```

2.1.2.30 enum LASS_RETURN kdsyr2k (enum DDSS_UPLO *UPLO*, enum DDSS_TRANS *TRANS*, int *N*, int *K*, const double *ALPHA*, double * *A*, int *LDA*, double * *B*, int *LDB*, const double *BETA*, double * *C*, int *LDC*)

Performs one of the symmetric rank 2k operations:

```
C = ALPHA * A * B**T + ALPHA * B * A**T + BETA * C or
C = ALPHA * A**T * B + ALPHA * B**T * A + BETA * C
```

ALPHA and BETA are scalars, C is an N by N symmetric matrix and A and B are N by K matrices in the first case and K by N matrices in the second case.

Parameters

in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form in which C is stored: • Lower: Lower triangular part of C is stored. The upper traingular part is not referenced. • Upper: Upper triangular part of C is stored. The lower triangular part is not referenced.
in	*TRANS*	enum DDSS_TRANS. TRANS specifies the operation to be performed as follows: • NoTrans: C = ALPHA * A * B**T + ALPHA * B * A**T + BETA * C • Trans: C = ALPHA * A**T * B + ALPHA * B**T * A + BETA * C
in	*N*	int. N specifies the order of matrix C. N must be at least zero.
in	*K*	int. With TRANS = NoTrans, K specifies the number of columns of the matrices A and B, and with TRANS = Trans, K specifies the number of rows of the matrices A and B. K must be at least zero.
in	*ALPHA*	double. ALPHA specifies the scalar alpha.
in	*A*	double *. A is a pointer to a matrix of dimension Na (rows) by Ka (columns), where Na is N and Ka is K when TRANS = NoTrans, and Na is K and Ka is N otherwise.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). When TRANS = NoTrans then LDA must be at least max(1, K), otherwise LDA must be at least max(1, N).
in	*B*	double *. B is a pointer to a matrix of dimension Nb (rows) by Kb (columns), where Nb is N and Kb is K when TRANS = NoTrans, and Nb is K and Kb is N otherwise.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). When TRANS = NoTrans then LDB must be at least max(1, K), otherwise LDB must be at least max(1, N).
in	*BETA*	double. BETA specifies the scalar beta.
in,out	*C*	double *. C is a pointer to a matrix of dimension N by N. When UPLO = Uppper the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of C is overwritten by the upper triangular part of the updated solution matrix C. When UPLO = Lower the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of C is overwritten by the lower triangular part of the updated solution matrix C.
in	*LDC*	int. LDC specifies the number of columns of C (row-major order). LDC must be at least max(1, N).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

ddss_tile
ddss_flat2tiled
ddss_dsymflat2tiled
ddss_dsymtiled2flat

Definition at line 124 of file kdsyr2k.c.

References ddss_dflat2tiled(), ddss_dsymflat2tiled(), ddss_dsymtiled2flat(), and ddss_tile_size().

Referenced by ddss_dsyr2k().

```
129 {
130
131     // Local variables
132     int nt, kt;
133     int mi, ni, ki;
134     int Am, An;
135     int Bm, Bn;
136     int Cm, Cn;
137     int tile_size_m;
138     int tile_size_n;
139     int tile_size_k;
140     int ka;
141     int kb;
142     double beta;
143
144     // Number of tiles
145     if ( N % TILE_SIZE == 0 )
146     {
147         nt = N / TILE_SIZE;
148     }
149     else
150     {
151         nt = ( N / TILE_SIZE ) + 1;
152     }
153
154     if ( K % TILE_SIZE == 0 )
155     {
156         kt = K / TILE_SIZE;
157     }
158     else
159     {
160         kt = ( K / TILE_SIZE ) + 1;
161     }
162
163     /*****************************
164      --Tile matrices declaration--
165      *****************************/
166
167     if ( TRANS == NoTrans )
168     {
169         Am = Bm = nt;
170         An = Bn = kt;
171         ka = kb = N;
172     }
173     else
174     {
175         Am = Bm = kt;
176         An = Bn = nt;
177         ka = kb = K;
178     }
179
180     Cm = Cn = nt;
181
182     /***************************
183      --Tile matrices allocation--
184      ***************************/
185
186     double ( *TILE_A )[An][TILE_SIZE * TILE_SIZE] = malloc(
187             Am * An * TILE_SIZE * TILE_SIZE * sizeof( double ) );
188
189     if ( TILE_A == NULL )
190     {
191         fprintf( stderr, "Failure in ddss_dtile_alloc for matrix TILE_A\n" );
192         return NoSuccess;
193     }
194
195     double ( *TILE_B )[Bn][TILE_SIZE * TILE_SIZE] = malloc(
```

```
196             Bm * Bn * TILE_SIZE * TILE_SIZE * sizeof( double ) );
197
198     if ( TILE_B == NULL )
199     {
200         fprintf( stderr, "Failure in ddss_dtile_alloc for matrix TILE_B\n" );
201         return NoSuccess;
202     }
203
204     double ( *TILE_C )[Cn][TILE_SIZE * TILE_SIZE] = malloc(
205             Cm * Cn * TILE_SIZE * TILE_SIZE * sizeof( double ) );
206
207     if ( TILE_C == NULL )
208     {
209         fprintf( stderr, "Failure in ddss_dtile_alloc for matrix TILE_C\n" );
210         return NoSuccess;
211     }
212
213     /*****************************************
214      --From flat data layout to tiled data layout--
215      *****************************************/
216
217     // From flat matrix A to tile matrix TILE_A
218     ddss_dflat2tiled( ka, LDA, A, LDA, Am, An, TILE_A );
219
220     // From flat matrix B to tile matrix TILE_B
221     ddss_dflat2tiled( kb, LDB, B, LDB, Bm, Bn, TILE_B );
222
223     // From flat matrix C to tile matrix TILE_C
224     ddss_dsymflat2tiled( N, N, C, LDC, Cm, Cn, TILE_C, UPLO );
225
226     /*************
227      --DSYRK tile--
228      *************/
229
230     // --TRANS = NoTrans & UPLO = Upper--
231     if ( TRANS == NoTrans )
232     {
233         if ( UPLO == Upper )
234         {
235             for ( mi = 0; mi < nt; mi++ )
236             {
237                 tile_size_m = ddss_tile_size( N, mi );
238                 for ( ni = 0; ni < kt; ni++ )
239                 {
240                     tile_size_n = ddss_tile_size( K, ni );
241
242                     if ( ni == 0 )
243                     {
244                         beta = BETA;
245                     }
246                     else
247                     {
248                         beta = 1.0;
249                     }
250
251                     #pragma oss task in( TILE_A[mi][ni] ) \
252                         in( TILE_B[mi][ni] ) \
253                         inout( TILE_C[mi][mi] ) \
254                         shared( TILE_A, TILE_B, TILE_C ) \
255                         firstprivate( mi,ni ) \
256                         label( dsyr2k )
257                     cblas_dsyr2k( CblasRowMajor,
258                                 ( CBLAS_UPLO ) UPLO, ( CBLAS_TRANSPOSE ) TRANS,
259                                 tile_size_m,
260                                 tile_size_n,
261                                 ALPHA, TILE_A[mi][ni], tile_size_n,
262                                        TILE_B[mi][ni], tile_size_n,
263                                 beta,  TILE_C[mi][mi], tile_size_m );
264
265                     for ( ki = mi + 1; ki < nt; ki++ )
266                     {
267                         tile_size_k = ddss_tile_size( N, ki );
268
269                         #pragma oss task in( TILE_A[mi][ni] ) \
270                             in( TILE_A[ki][ni] ) \
271                             in( TILE_B[mi][ni] ) \
272                             in( TILE_B[ki][ni] ) \
273                             inout( TILE_C[mi][ki] ) \
274                             shared( TILE_A, TILE_B, TILE_C ) \
275                             firstprivate( mi,ni,ki ) \
276                             label( dgemm )
277                         {
278                             cblas_dgemm( CblasRowMajor,
279                                         CblasNoTrans, CblasTrans,
280                                         tile_size_m,
281                                         tile_size_k,
282                                         tile_size_n,
```

```
283                                     ALPHA, TILE_A[mi][ni], tile_size_n,
284                                            TILE_B[ki][ni], tile_size_n,
285                                     beta,  TILE_C[mi][ki], tile_size_k );
286
287                             cblas_dgemm( CblasRowMajor,
288                                     CblasNoTrans, CblasTrans,
289                                     tile_size_m,
290                                     tile_size_k,
291                                     tile_size_n,
292                                     ALPHA, TILE_B[mi][ni], tile_size_n,
293                                            TILE_A[ki][ni], tile_size_n,
294                                     1.0,   TILE_C[mi][ki], tile_size_k );
295                         }
296                     }
297
298                 }
299             }
300         }
301         // --TRANS = NoTrans & UPLO = Lower--
302         else
303         {
304             for ( mi = 0; mi < nt; mi++ )
305             {
306                 tile_size_m = ddss_tile_size( N, mi );
307                 for ( ni = 0; ni < kt; ni++ )
308                 {
309                     tile_size_n = ddss_tile_size( K, ni );
310
311                     if ( ni == 0 )
312                     {
313                         beta = BETA;
314                     }
315                     else
316                     {
317                         beta = 1.0;
318                     }
319
320                     #pragma oss task in( TILE_A[mi][ni] ) \
321                         in( TILE_B[mi][ni] ) \
322                         inout( TILE_C[mi][mi] ) \
323                         shared( TILE_A, TILE_B, TILE_C ) \
324                         firstprivate( mi,ni ) \
325                         label( dsyr2k )
326                     cblas_dsyr2k( CblasRowMajor,
327                                 ( CBLAS_UPLO ) UPLO, ( CBLAS_TRANSPOSE ) TRANS,
328                                 tile_size_m,
329                                 tile_size_n,
330                                 ALPHA, TILE_A[mi][ni], tile_size_n,
331                                        TILE_B[mi][ni], tile_size_n,
332                                 beta,  TILE_C[mi][mi], tile_size_m );
333
334                     for ( ki = mi + 1; ki < nt; ki++ )
335                     {
336                         tile_size_k = ddss_tile_size( N, ki );
337
338                         #pragma oss task in( TILE_A[ki][ni] ) \
339                             in( TILE_A[mi][ni] ) \
340                             in( TILE_B[ki][ni] ) \
341                             in( TILE_B[mi][ni] ) \
342                             inout( TILE_C[ki][mi] ) \
343                             shared( TILE_A, TILE_B, TILE_C ) \
344                             firstprivate( mi,ni,ki ) \
345                             label( dgemm )
346                         {
347                             cblas_dgemm( CblasRowMajor,
348                                     CblasNoTrans, CblasTrans,
349                                     tile_size_k,
350                                     tile_size_m,
351                                     tile_size_n,
352                                     ALPHA, TILE_A[ki][ni], tile_size_n,
353                                            TILE_B[mi][ni], tile_size_n,
354                                     beta,  TILE_C[ki][mi], tile_size_m );
355
356                             cblas_dgemm( CblasRowMajor,
357                                     CblasNoTrans, CblasTrans,
358                                     tile_size_k,
359                                     tile_size_m,
360                                     tile_size_n,
361                                     ALPHA, TILE_B[ki][ni], tile_size_n,
362                                            TILE_A[mi][ni], tile_size_n,
363                                     1.0,   TILE_C[ki][mi], tile_size_m );
364                         }
365                     }
366
367                 }
368             }
369         }
```

```
370     }
371     // --TRANS = Trans & UPLO = Upper--
372     else
373     {
374         if ( UPLO == Upper )
375         {
376             for ( mi = 0; mi < kt; mi++ )
377             {
378                 tile_size_m = ddss_tile_size( K, mi );
379                 for ( ni = 0; ni < nt; ni++ )
380                 {
381                     tile_size_n = ddss_tile_size( N, ni );
382 
383                     if ( mi == 0 )
384                     {
385                         beta = BETA;
386                     }
387                     else
388                     {
389                         beta = 1.0;
390                     }
391 
392                     #pragma oss task in( TILE_A[mi][ni] ) \
393                         in( TILE_B[mi][ni] ) \
394                         inout( TILE_C[ni][ni] ) \
395                         shared( TILE_A, TILE_B, TILE_C ) \
396                         firstprivate( mi,ni ) \
397                         label( dsyr2k )
398                     cblas_dsyr2k( CblasRowMajor,
399                                   ( CBLAS_UPLO ) UPLO, ( CBLAS_TRANSPOSE ) TRANS,
400                                   tile_size_n,
401                                   tile_size_m,
402                                   ALPHA, TILE_A[mi][ni], tile_size_n,
403                                          TILE_B[mi][ni], tile_size_n,
404                                   beta,  TILE_C[ni][ni], tile_size_n );
405 
406                     for ( ki = ni + 1; ki < nt; ki++ )
407                     {
408                         tile_size_k = ddss_tile_size( N, ki );
409 
410                         #pragma oss task in( TILE_A[mi][ni] ) \
411                             in( TILE_A[mi][ki] ) \
412                             in( TILE_B[mi][ni] ) \
413                             in( TILE_B[mi][ki] ) \
414                             inout( TILE_C[ni][ki] ) \
415                             shared( TILE_A, TILE_B, TILE_C ) \
416                             firstprivate( mi,ni,ki ) \
417                             label( dgemm )
418                         {
419                             cblas_dgemm( CblasRowMajor,
420                                          CblasTrans, CblasNoTrans,
421                                          tile_size_n,
422                                          tile_size_k,
423                                          tile_size_m,
424                                          ALPHA, TILE_A[mi][ni], tile_size_n,
425                                                 TILE_B[mi][ki], tile_size_k,
426                                          beta,  TILE_C[ni][ki], tile_size_k );
427 
428                             cblas_dgemm( CblasRowMajor,
429                                          CblasTrans, CblasNoTrans,
430                                          tile_size_n,
431                                          tile_size_k,
432                                          tile_size_m,
433                                          ALPHA, TILE_B[mi][ni], tile_size_n,
434                                                 TILE_A[mi][ki], tile_size_k,
435                                          1.0,   TILE_C[ni][ki], tile_size_k );
436                         }
437                     }
438                 }
439             }
440         }
441         // --TRANS = Trans & UPLO = Lower--
442         else
443         {
444             for ( mi = 0; mi < kt; mi++ )
445             {
446                 tile_size_m = ddss_tile_size( K, mi );
447                 for ( ni = 0; ni < nt; ni++ )
448                 {
449                     tile_size_n = ddss_tile_size( N, ni );
450 
451                     if ( mi == 0 )
452                     {
453                         beta = BETA;
454                     }
455                     else
456                     {
```

```
457                         beta = 1.0;
458                     }
459
460                     #pragma oss task in( TILE_A[mi][ni] ) \
461                         in( TILE_B[mi][ni] ) \
462                         inout( TILE_C[ni][ni] ) \
463                         shared( TILE_A, TILE_B, TILE_C ) \
464                         firstprivate( mi,ni ) \
465                         label( dsyr2k )
466                     cblas_dsyr2k( CblasRowMajor,
467                                 ( CBLAS_UPLO ) UPLO, ( CBLAS_TRANSPOSE ) TRANS,
468                                 tile_size_n,
469                                 tile_size_m,
470                                 ALPHA, TILE_A[mi][ni], tile_size_n,
471                                        TILE_B[mi][ni], tile_size_n,
472                                 beta,  TILE_C[ni][ni], tile_size_n );
473
474                     for ( ki = ni + 1; ki < nt; ki++ )
475                     {
476                         tile_size_k = ddss_tile_size( N, ki );
477
478                         #pragma oss task in( TILE_A[mi][ki] ) \
479                             in( TILE_A[mi][ni] ) \
480                             in( TILE_B[mi][ki] ) \
481                             in( TILE_B[mi][ni] ) \
482                             inout( TILE_C[ki][ni] ) \
483                             shared( TILE_A, TILE_B, TILE_C ) \
484                             firstprivate( mi,ni,ki ) \
485                             label( dgemm )
486                         {
487                             cblas_dgemm( CblasRowMajor,
488                                         CblasTrans, CblasNoTrans,
489                                         tile_size_k,
490                                         tile_size_n,
491                                         tile_size_m,
492                                         ALPHA, TILE_A[mi][ki], tile_size_k,
493                                                TILE_B[mi][ni], tile_size_n,
494                                         beta,  TILE_C[ki][ni], tile_size_n );
495
496                             cblas_dgemm( CblasRowMajor,
497                                         CblasTrans, CblasNoTrans,
498                                         tile_size_k,
499                                         tile_size_n,
500                                         tile_size_m,
501                                         ALPHA, TILE_B[mi][ki], tile_size_k,
502                                                TILE_A[mi][ni], tile_size_n,
503                                         1.0,   TILE_C[ki][ni], tile_size_n );
504                         }
505                     }
506                 }
507             }
508         }
509     }
510
511     /*****************************************
512      --From tiled data layout to flat data layout--
513      *****************************************/
514
515     ddss_dsymtiled2flat( N, N, C, LDC, Cm, Cn, TILE_C, UPLO );
516
517     // --Tile matrices free--
518     free( TILE_A );
519     free( TILE_B );
520     free( TILE_C );
521
522     return Success;
523
524 }
```

2.1.2.31 enum LASS_RETURN kdsyrk (enum DDSS_UPLO *UPLO*, enum DDSS_TRANS *TRANS_A*, int *N*, int *K*, const double *ALPHA*, double * *A*, int *LDA*, const double *BETA*, double * *C*, int *LDC*)

Performs one of the symmetric rank k operations:

```
C = ALPHA * A * op( A ) + BETA * C  or
C = ALPHA * op( A ) * A + BETA * C
```

where op(X) is:

```
op( X ) = X**T
```

ALPHA and BETA are scalars, C is an N by N symmetric matrix and A is an N by K matrix in the first case and a K by N matrix in the second case.

Parameters

in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form in which C is stored: • Lower: Lower triangular part of C is stored. The upper traingular part is not referenced. • Upper: Upper triangular part of C is stored. The lower triangular part is not referenced.
in	*TRANS_A*	enum DDSS_TRANS. TRANS_A specifies the operation to be performed as follows: • NoTrans: C = ALPHA ∗ A ∗ A**T + BETA ∗ C • Trans: C = ALPHA ∗ A**T ∗ A + BETA ∗ C
in	*N*	int. N specifies the order of matrix C. N must be at least zero.
in	*K*	int. With TRANS_A = NoTrans, K specifies the number of columns of the matrix A, and with TRANS_A = Trans, K specifies the number of rows of the matrix A. K must be at least zero.
in	*ALPHA*	double. ALPHA specifies the scalar alpha.
in	*A*	double ∗. A is a pointer to a matrix of dimension Na (rows) by Ka (columns), where Na is N and Ka is K when TRANS_A = NoTrans, and Na is K and Ka is N otherwise.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). When TRANS_A = NoTrans then LDA must be at least max(1, K), otherwise LDA must be at least max(1, N).
in	*BETA*	double. BETA specifies the scalar beta.
in,out	*C*	double ∗. C is a pointer to a matrix of dimension N by N. When UPLO = Uppper the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of C is overwritten by the upper triangular part of the updated solution matrix C. When UPLO = Lower the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of C is overwritten by the lower triangular part of the updated solution matrix C.
in	*LDC*	int. LDC specifies the number of columns of C (row-major order). LDC must be at least max(1, N).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

ddss_tile
ddss_flat2tiled
ddss_dsymflat2tiled
ddss_dsymtiled2flat

Definition at line 117 of file kdsyrk.c.

References ddss_dflat2tiled(), ddss_dsymflat2tiled(), ddss_dsymtiled2flat(), and ddss_tile_size().

Referenced by ddss_dsyrk().

```
121 {
122
123     // Local variables
124     int nt, kt;
125     int mi, ni, ki;
126     int Am, An;
127     int Cm, Cn;
128     int tile_size_m;
129     int tile_size_n;
130     int tile_size_k;
131     int ka;
132     double beta;
133
134     // Number of tiles
135     if ( N % TILE_SIZE == 0 )
136     {
137         nt = N / TILE_SIZE;
138     }
139     else
140     {
141         nt = ( N / TILE_SIZE ) + 1;
142     }
143
144     if ( K % TILE_SIZE == 0 )
145     {
146         kt = K / TILE_SIZE;
147     }
148     else
149     {
150         kt = ( K / TILE_SIZE ) + 1;
151     }
152
153     /*****************************
154      --Tile matrices declaration--
155      *****************************/
156
157     if ( TRANS_A == NoTrans )
158     {
159         Am = nt;
160         An = kt;
161         ka = N;
162     }
163     else
164     {
165         Am = kt;
166         An = nt;
167         ka = K;
168     }
169
170     Cm = Cn = nt;
171
172     /***************************
173      --Tile matrices allocation--
174      ***************************/
175
176     double ( *TILE_A )[An][TILE_SIZE * TILE_SIZE] = malloc(
177             Am * An * TILE_SIZE * TILE_SIZE * sizeof( double ) );
178
179     if ( TILE_A == NULL )
180     {
181         fprintf( stderr, "Failure in ddss_dtile_alloc for matrix TILE_A\n" );
182         return NoSuccess;
183     }
184
185     double ( *TILE_C )[Cn][TILE_SIZE * TILE_SIZE] = malloc(
186             Cm * Cn * TILE_SIZE * TILE_SIZE * sizeof( double ) );
187
188     if ( TILE_C == NULL )
189     {
190         fprintf( stderr, "Failure in ddss_dtile_alloc for matrix TILE_C\n" );
191         return NoSuccess;
192     }
193
194     /*****************************************
195      --From flat data layout to tiled data layout--
196      *****************************************/
197
198     // From flat matrix A to tile matrix TILE_A
199     ddss_dflat2tiled( ka, LDA, A, LDA, Am, An, TILE_A );
200
201     // From flat matrix C to tile matrix TILE_C
202     ddss_dsymflat2tiled( N, N, C, LDC, Cm, Cn, TILE_C, UPLO );
203
204     /*************
205      --DSYRK tile--
206      *************/
207
```

```
208     // --TRANS_A = NoTrans & UPLO = Upper--
209     if ( TRANS_A == NoTrans )
210     {
211         if ( UPLO == Upper )
212         {
213             for ( mi = 0; mi < nt; mi++ )
214             {
215                 tile_size_m = ddss_tile_size( N, mi );
216                 for ( ni = 0; ni < kt; ni++ )
217                 {
218                     tile_size_n = ddss_tile_size( K, ni );
219 
220                     if ( ni == 0 )
221                     {
222                         beta = BETA;
223                     }
224                     else
225                     {
226                         beta = 1.0;
227                     }
228 
229                     #pragma oss task in( TILE_A[mi][ni] ) \
230                         inout( TILE_C[mi][mi] ) \
231                         shared( TILE_A, TILE_C ) \
232                         firstprivate( mi, ni ) \
233                         label( dsyrk )
234                     cblas_dsyrk( CblasRowMajor,
235                                  ( CBLAS_UPLO ) UPLO,
236                                  ( CBLAS_TRANSPOSE ) TRANS_A,
237                                  tile_size_m,
238                                  tile_size_n,
239                                  ALPHA, TILE_A[mi][ni], tile_size_n,
240                                  beta,  TILE_C[mi][mi], tile_size_m );
241 
242                     for ( ki = mi + 1; ki < nt; ki++ )
243                     {
244                         tile_size_k = ddss_tile_size( N, ki );
245 
246                         #pragma oss task in( TILE_A[mi][ni] ) \
247                             in( TILE_A[ki][ni] ) \
248                             inout( TILE_C[mi][ki] ) \
249                             shared( TILE_A, TILE_C ) \
250                             firstprivate( mi, ni, ki ) \
251                             label( dgemm )
252                         cblas_dgemm( CblasRowMajor,
253                                      CblasNoTrans, CblasTrans,
254                                      tile_size_m,
255                                      tile_size_k,
256                                      tile_size_n,
257                                      ALPHA, TILE_A[mi][ni], tile_size_n,
258                                             TILE_A[ki][ni], tile_size_n,
259                                      beta,  TILE_C[mi][ki], tile_size_k );
260                     }
261 
262                 }
263             }
264         }
265         // --TRANS_A = NoTrans & UPLO = Lower--
266         else
267         {
268             for ( mi = 0; mi < nt; mi++ )
269             {
270                 tile_size_m = ddss_tile_size( N, mi );
271                 for ( ni = 0; ni < kt; ni++ )
272                 {
273                     tile_size_n = ddss_tile_size( K, ni );
274 
275                     if ( ni == 0 )
276                     {
277                         beta = BETA;
278                     }
279                     else
280                     {
281                         beta = 1.0;
282                     }
283 
284                     #pragma oss task in( TILE_A[mi][ni] ) \
285                         inout( TILE_C[mi][mi] ) \
286                         shared( TILE_A, TILE_C ) \
287                         firstprivate( mi, ni ) \
288                         label( dsyrk )
289                     cblas_dsyrk( CblasRowMajor,
290                                  ( CBLAS_UPLO ) UPLO,
291                                  ( CBLAS_TRANSPOSE ) TRANS_A,
292                                  tile_size_m,
293                                  tile_size_n,
294                                  ALPHA, TILE_A[mi][ni], tile_size_n,
```

```
295                                 beta,  TILE_C[mi][mi], tile_size_m );
296
297                     for ( ki = mi + 1; ki < nt; ki++ )
298                     {
299                         tile_size_k = ddss_tile_size( N, ki );
300
301                         #pragma oss task in( TILE_A[ki][ni] ) \
302                             in( TILE_A[mi][ni] ) \
303                             inout( TILE_C[ki][mi] ) \
304                             shared( TILE_A, TILE_C ) \
305                             firstprivate( mi, ni, ki ) \
306                             label( dgemm )
307                         cblas_dgemm( CblasRowMajor,
308                                      CblasNoTrans, CblasTrans,
309                                      tile_size_k,
310                                      tile_size_m,
311                                      tile_size_n,
312                                      ALPHA, TILE_A[ki][ni], tile_size_n,
313                                             TILE_A[mi][ni], tile_size_n,
314                                      beta,  TILE_C[ki][mi], tile_size_m );
315                     }
316
317                 }
318             }
319         }
320     }
321     // --TRANS_A = Trans & UPLO = Upper--
322     else
323     {
324         if ( UPLO == Upper )
325         {
326             for ( mi = 0; mi < kt; mi++ )
327             {
328                 tile_size_m = ddss_tile_size( K, mi );
329                 for ( ni = 0; ni < nt; ni++ )
330                 {
331                     tile_size_n = ddss_tile_size( N, ni );
332
333                     if ( mi == 0 )
334                     {
335                         beta = BETA;
336                     }
337                     else
338                     {
339                         beta = 1.0;
340                     }
341
342                     #pragma oss task in( TILE_A[mi][ni] ) \
343                         inout( TILE_C[ni][ni] ) \
344                         shared( TILE_A, TILE_C ) \
345                         firstprivate( mi, ni ) \
346                         label( dsyrk )
347                     cblas_dsyrk( CblasRowMajor,
348                                  ( CBLAS_UPLO ) UPLO,
349                                  ( CBLAS_TRANSPOSE ) TRANS_A,
350                                  tile_size_n,
351                                  tile_size_m,
352                                  ALPHA, TILE_A[mi][ni], tile_size_n,
353                                  beta,  TILE_C[ni][ni], tile_size_n );
354
355                     for ( ki = ni + 1; ki < nt; ki++ )
356                     {
357                         tile_size_k = ddss_tile_size( N, ki );
358
359                         #pragma oss task in( TILE_A[mi][ni] ) \
360                             in( TILE_A[mi][ki] ) \
361                             inout( TILE_C[ni][ki] ) \
362                             shared( TILE_A, TILE_C ) \
363                             firstprivate( mi, ni, ki ) \
364                             label( dgemm )
365                         cblas_dgemm( CblasRowMajor,
366                                      CblasTrans, CblasNoTrans,
367                                      tile_size_n,
368                                      tile_size_k,
369                                      tile_size_m,
370                                      ALPHA, TILE_A[mi][ni], tile_size_n,
371                                             TILE_A[mi][ki], tile_size_k,
372                                      beta,  TILE_C[ni][ki], tile_size_k );
373                     }
374                 }
375             }
376         }
377         // --TRANS_A = Trans & UPLO = Lower--
378         else
379         {
380             for ( mi = 0; mi < kt; mi++ )
381             {
```

```
382                 tile_size_m = ddss_tile_size( K, mi );
383                 for ( ni = 0; ni < nt; ni++ )
384                 {
385                     tile_size_n = ddss_tile_size( N, ni );
386 
387                     if ( mi == 0 )
388                     {
389                         beta = BETA;
390                     }
391                     else
392                     {
393                         beta = 1.0;
394                     }
395 
396                     #pragma oss task in( TILE_A[mi][ni] ) \
397                         inout( TILE_C[ni][ni] ) \
398                         shared( TILE_A, TILE_C ) \
399                         firstprivate( mi, ni ) \
400                         label( dsyrk )
401                     cblas_dsyrk( CblasRowMajor,
402                                  ( CBLAS_UPLO ) UPLO,
403                                  ( CBLAS_TRANSPOSE ) TRANS_A,
404                                  tile_size_n,
405                                  tile_size_m,
406                                  ALPHA, TILE_A[mi][ni], tile_size_n,
407                                  beta,  TILE_C[ni][ni], tile_size_n );
408 
409                     for ( ki = ni + 1; ki < nt; ki++ )
410                     {
411                         tile_size_k = ddss_tile_size( N, ki );
412 
413                         #pragma oss task in( TILE_A[mi][ki] ) \
414                             in( TILE_A[mi][ni] ) \
415                             inout( TILE_C[ki][ni] ) \
416                             shared( TILE_A, TILE_C ) \
417                             firstprivate( mi, ni, ki ) \
418                             label( dgemm )
419                         cblas_dgemm( CblasRowMajor,
420                                      CblasTrans, CblasNoTrans,
421                                      tile_size_k,
422                                      tile_size_n,
423                                      tile_size_m,
424                                      ALPHA, TILE_A[mi][ki], tile_size_k,
425                                             TILE_A[mi][ni], tile_size_n,
426                                      beta,  TILE_C[ki][ni], tile_size_n );
427                     }
428                 }
429             }
430         }
431     }
432 
433     /*****************************************
434      --From tiled data layout to flat data layout--
435      *****************************************/
436 
437     ddss_dsymtiled2flat( N, N, C, LDC, Cm, Cn, TILE_C, UPLO );
438 
439     // --Tile matrices free--
440     free( TILE_A );
441     free( TILE_C );
442 
443     return Success;
444 
445 }
```

2.1.2.32 enum LASS_RETURN kdtpgesv (int *N*, int *NRHS*, double * *A*, int *LDA*, int * *IPIV*, double * *B*, int *LDB*)

Solves a system of linear equations A X = B, where A is a N-by-N general matrix and X and B are N-by-NRHS matrices. The matrix A is factorized using the LU descomposition with tiled-pivoting. The matrix A is descomposed as:

```
A = P * L * U
```

where P is a permutation matrix, L is a lower triangular matrix with unit diagonal elements and U is an upper triangular matrix.

Parameters

in	*N*	int. N specifies the order of the square matrix A. N >= 0.

NRHS int. NRHS specifies the number of right-hand-sides (number of columns of B). NRHS >= 0.

Parameters

in,out	*A*	double *. A is a pointer to a regular matrix of dimension N-by-LDA. On exit, if return value is Success, the matrix A is overwriten by the factors L and U. The unit diagonal elements of L are not stored.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).
out	*IPIV*	int *. ipiv is a pointer to an array of dimesion at least max(1, min (M, N)). ipiv(i) = j, 1 <= i <= min(M, N) implies that rows i and j have been interchanged.
in,out	*B*	double *. B is a pointer to a matrix of dimension N by NRHS, which stores the right-hand-sides of the systems of linear equations. (row-major order). On exit, if return value is Success, the matrix B is overwriten by the solution matrix X.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, NRHS).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

ddss_tile
ddss_flat2tiled
ddss_tiled2flat

Definition at line 91 of file kdtpgesv.c.

References ddss_dflat2tiled(), ddss_dtiled2flat(), and ddss_tile_size().

Referenced by ddss_dtpgesv().

```
95 {
96
97     // Local variables
98     int i;
99     int nt, nrhst;
100     int ni, mi, nni, ki, nrhsi;
101     int tile_size_n;
102     int tile_size_m;
103     int tile_size_nn;
104     int tile_size_nrhs;
105     int tile_size_k;
106     int local_ipiv_size;
107     int current_position;
108     enum LASS_RETURN ret;
109     int *ipiv_local;
110
111     // Number of tiles
112     if ( N % TILE_SIZE == 0 )
113     {
114         nt = N / TILE_SIZE;
115     }
116     else
```

```
117     {
118         nt = ( N / TILE_SIZE ) + 1;
119     }
120
121     if ( NRHS % TILE_SIZE == 0 )
122     {
123         nrhst = NRHS / TILE_SIZE;
124     }
125     else
126     {
127         nrhst = ( NRHS / TILE_SIZE ) + 1;
128     }
129
130     /***************************
131     --Tile matrices allocation--
132     ***************************/
133
134     double ( *TILE_A )[nt][TILE_SIZE * TILE_SIZE] = malloc ( nt * nt *
135         TILE_SIZE * TILE_SIZE * sizeof( double ) );
136
137     if ( TILE_A == NULL )
138     {
139         fprintf( stderr, "Failure in kdtpgesv for matrix TILE_A\n" );
140         return NoSuccess;
141     }
142
143     double ( *TILE_B )[nrhst][TILE_SIZE * TILE_SIZE] = malloc(
144              nt * nrhst * TILE_SIZE * TILE_SIZE * sizeof( double ) );
145
146     if ( TILE_B == NULL )
147     {
148         fprintf( stderr, "Failure in kdtpgesv for matrix TILE_B\n" );
149         return NoSuccess;
150     }
151
152     ipiv_local = ( int* ) malloc( N * sizeof( int ) );
153
154     if ( ipiv_local == NULL )
155     {
156         fprintf( stderr, "Failure in kdtpgesv for ipiv_local\n" );
157         return NoSuccess;
158     }
159
160     /***********************************************
161     // --From flat data layout to tiled data layout--
162     ***********************************************/
163
164     // From flat matrix A to tile matrix TILE_A
165     ddss_dflat2tiled( N, N, A, LDA, nt, nt, TILE_A );
166
167     // From flat matrix B to tile matrix TILE_B
168     ddss_dflat2tiled( N, NRHS, B, LDB, nt, nrhst, TILE_B );
169
170     /****************
171     --DGETRF tile--
172     ****************/
173
174     current_position = 0;
175     i = 0;
176     for ( ki = 0; ki < nt; ki++ )
177     {
178         tile_size_k = ddss_tile_size( N, ki );
179
180         #pragma oss task inout( TILE_A[ki][ki] ) \
181             inout( ipiv_local ) \
182             inout( current_position ) \
183             out( local_ipiv_size ) \
184             shared( TILE_A ) \
185             firstprivate( ki, i ) \
186             label( dgetrf )
187         {
188             LAPACKE_dgetrf( CblasRowMajor,
189                             tile_size_k, tile_size_k,
190                             TILE_A[ki][ki], tile_size_k,
191                             ipiv_local + current_position );
192
193             // Update the global ipiv array
194             local_ipiv_size = tile_size_k;
195             for ( i = 0; i < local_ipiv_size; i++ )
196             {
197                 ( IPIV + current_position )[i] =
198                         ( ipiv_local + current_position )[i] + current_position;
199             }
200             current_position += local_ipiv_size;
201         }
202         for ( mi = ki + 1; mi < nt; mi++ )
203         {
```

```
204             tile_size_m = ddss_tile_size( N, mi );
205
206             #pragma oss task in( TILE_A[ki][ki] ) \
207                 inout(TILE_A[mi][ki]) \
208                 shared( TILE_A ) \
209                 firstprivate( mi, ki ) \
210                 label( dtrsm_below )
211             cblas_dtrsm( CblasRowMajor,
212                          ( CBLAS_SIDE ) Right, ( CBLAS_UPLO ) Upper,
213                          ( CBLAS_TRANSPOSE ) NoTrans, ( CBLAS_DIAG ) NonUnit,
214                          tile_size_m, tile_size_k,
215                          1.0, TILE_A[ki][ki], tile_size_k,
216                               TILE_A[mi][ki], tile_size_k );
217         }
218         for ( ni = 0; ni < ki; ni++ )
219         {
220             tile_size_n = ddss_tile_size( N, ni );
221             // Swap the tiles to the left of A[ki][ki]
222             #pragma oss task in ( ipiv_local ) \
223                 in( current_position ) \
224                 in( local_ipiv_size ) \
225                 inout( TILE_A[ki][ni] ) \
226                 shared( TILE_A ) \
227                 firstprivate( ki, ni ) \
228                 label( dlaswp_left )
229             LAPACKE_dlaswp( CblasRowMajor,
230                             tile_size_n, TILE_A[ki][ni], tile_size_n,
231                             1, local_ipiv_size,
232                             ipiv_local + current_position - local_ipiv_size,
233                             1 );
234
235         }
236         for ( nrhsi = 0; nrhsi < nrhst; nrhsi++ )
237         {
238             tile_size_nrhs = ddss_tile_size( NRHS, nrhsi );
239             // Swap on the B[ki][nrhsi] tile
240             #pragma oss task in ( ipiv_local ) \
241                 in( current_position ) \
242                 in( local_ipiv_size ) \
243                 inout( TILE_B[ki][nrhsi] ) \
244                 shared( TILE_B ) \
245                 firstprivate( ki, nrhsi ) \
246                 label( dlaswp_on_B )
247             LAPACKE_dlaswp( CblasRowMajor,
248                             tile_size_nrhs, TILE_B[ki][nrhsi], tile_size_nrhs,
249                             1, local_ipiv_size,
250                             ipiv_local + current_position - local_ipiv_size,
251                             1 );
252
253         }
254         for ( ni = ki + 1; ni < nt; ni++ )
255         {
256             tile_size_n = ddss_tile_size( N, ni );
257             // Swap the tiles to the right of A[ki][ki]
258             #pragma oss task in ( ipiv_local ) \
259                 in( current_position ) \
260                 in( local_ipiv_size ) \
261                 inout( TILE_A[ki][ni] ) \
262                 shared( TILE_A ) \
263                 firstprivate( ki, ni ) \
264                 label( dlaswp_right )
265             LAPACKE_dlaswp( CblasRowMajor,
266                             tile_size_n, TILE_A[ki][ni], tile_size_n,
267                             1, local_ipiv_size,
268                             ipiv_local + current_position - local_ipiv_size,
269                             1 );
270
271             #pragma oss task in( TILE_A[ki][ki] ) \
272                 inout(TILE_A[ki][ni]) \
273                 shared( TILE_A ) \
274                 firstprivate( ni, ki ) \
275                 label( dtrsm_right )
276             cblas_dtrsm( CblasRowMajor,
277                          ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Lower,
278                          ( CBLAS_TRANSPOSE ) NoTrans, ( CBLAS_DIAG ) Unit,
279                          tile_size_k, tile_size_n,
280                          1.0, TILE_A[ki][ki], tile_size_k,
281                               TILE_A[ki][ni], tile_size_n );
282
283             for ( nni = ki + 1; nni < nt; nni++ )
284             {
285                 tile_size_nn = ddss_tile_size( N, nni );
286
287                 #pragma oss task in( TILE_A[nni][ki] ) \
288                     in( TILE_A[ki][ni] ) \
289                     inout( TILE_A[nni][ni] ) \
290                     shared( TILE_A ) \
```

```
291                     firstprivate( ni, nni, ki ) \
292                     label( dgemm )
293                 cblas_dgemm( CblasRowMajor,
294                              ( CBLAS_TRANSPOSE ) NoTrans,
295                              ( CBLAS_TRANSPOSE ) NoTrans,
296                              tile_size_nn,
297                              tile_size_n,
298                              TILE_SIZE,
299                              -1.0, TILE_A[nni][ki], tile_size_k,
300                                    TILE_A[ki][ni],  tile_size_n,
301                               1.0, TILE_A[nni][ni], tile_size_n );
302             }
303         }
304     }
305
306     // --From tile data layout to flat data layout--
307     ddss_dtiled2flat( N, N, A, LDA, nt, nt, TILE_A );
308
309     // Triandular solve
310     // --SIDE = Left & UPLO = Lower & TRANS_A = NoTrans & Unit--
311     for ( ki = 0; ki < nt; ki++ )
312     {
313         tile_size_k = ddss_tile_size( N, ki );
314
315         for ( ni = 0; ni < nrhst; ni++ )
316         {
317             tile_size_n = ddss_tile_size( NRHS, ni );
318
319             #pragma oss task in( TILE_A[ki][ki] ) \
320                 inout(  TILE_B[ki][ni] ) \
321                 shared( TILE_A, TILE_B ) \
322                 firstprivate( ki, ni ) \
323                 label( dtrsm_dtrsm1 )
324             cblas_dtrsm( CblasRowMajor,
325                          ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Lower,
326                          ( CBLAS_TRANSPOSE ) NoTrans,
327                          ( CBLAS_DIAG ) Unit,
328                          tile_size_k,
329                          tile_size_n,
330                          1.0, TILE_A[ki][ki], tile_size_k,
331                               TILE_B[ki][ni], tile_size_n );
332         }
333         for ( nni = 0; nni < nrhst; nni++ )
334         {
335             tile_size_nn = ddss_tile_size( NRHS, nni );
336
337             for ( mi = ki + 1; mi < nt; mi++ )
338             {
339
340                 #pragma oss task in( TILE_A[mi][ki] ) \
341                     in( TILE_B[ki][nni] ) \
342                     inout( TILE_B[mi][nni] ) \
343                     shared( TILE_A, TILE_B ) \
344                     firstprivate( ki, mi, nni ) \
345                     label( dtrsm_dgemm1 )
346                 cblas_dgemm( CblasRowMajor,
347                              CblasNoTrans, CblasNoTrans,
348                              tile_size_k,
349                              tile_size_nn,
350                              tile_size_k,
351                              -1.0,  TILE_A[mi][ki],  tile_size_k,
352                                     TILE_B[ki][nni], tile_size_nn,
353                               1.0, TILE_B[mi][nni], tile_size_nn );
354             }
355         }
356     }
357
358     // Triandular solve
359     // --SIDE = Left & UPLO = Upper & TRANS_A = NoTrans & NonUnit--
360     for ( ki = nt - 1; ki >= 0; ki-- )
361     {
362         tile_size_k = ddss_tile_size( N, ki );
363
364         for ( ni = 0; ni < nrhst; ni++ )
365         {
366             tile_size_n = ddss_tile_size( NRHS, ni );
367
368             #pragma oss task in( TILE_A[ki][ki] ) \
369                 inout( TILE_B[ki][ni] ) \
370                 shared( TILE_A, TILE_B ) \
371                 firstprivate( ki, ni ) \
372                 label( dtrsmi_dtrsm2 )
373             cblas_dtrsm( CblasRowMajor,
374                          ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Upper,
375                          ( CBLAS_TRANSPOSE ) NoTrans,
376                          ( CBLAS_DIAG ) NonUnit,
377                          tile_size_k,
```

```
378                         tile_size_n,
379                         1.0, TILE_A[ki][ki], tile_size_k,
380                              TILE_B[ki][ni], tile_size_n );
381         }
382
383         for ( mi = ki - 1; mi >= 0; mi-- )
384         {
385             tile_size_m = ddss_tile_size( N, mi );
386
387             for ( nni = 0; nni < nrhst; nni++ )
388             {
389                 tile_size_nn = ddss_tile_size( NRHS, nni );
390
391                 #pragma oss task in( TILE_A[mi][ki] ) \
392                     in( TILE_B[ki][nni] ) \
393                     inout( TILE_B[mi][nni] ) \
394                     shared( TILE_A, TILE_B ) \
395                     firstprivate( ki, mi, nni ) \
396                     label( dtrsm_dgemm2 )
397                 cblas_dgemm( CblasRowMajor,
398                              CblasNoTrans, CblasNoTrans,
399                              tile_size_m,
400                              tile_size_nn,
401                              tile_size_k,
402                              -1.0, TILE_A[mi][ki],  tile_size_k,
403                                    TILE_B[ki][nni], tile_size_nn,
404                               1.0, TILE_B[mi][nni], tile_size_nn );
405             }
406         }
407     }
408
409     /*****************************************
410     --From tiled data layout to flat data layout--
411     *****************************************/
412     ddss_dtiled2flat( N, NRHS, B, LDB, nt, nrhst, TILE_B );
413
414     // --Free--
415     free( TILE_A );
416     free( TILE_B );
417     free( ipiv_local );
418
419     return Success;
420
421 }
```

2.1.2.33 enum LASS_RETURN kdtpgetrf (int *M*, int *N*, double * *A*, int *LDA*, int * *IPIV*)

Performs the LU factorization with tiled pivoting (row interchanges) of a general M-by-N matrix A:

```
A = P * L * U
```

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

Parameters

in	*M*	int. M specifies the number of rows of the matrix A. M >= 0.
in	*N*	int. N specifies the number of columns of the matrix A. N >= 0.
in,out	*A*	double *. A is a pointer to a regular matrix of dimension M-by-N. On exit, if return value is Success, the matrix A is overwriten by the factors L and U. The unit diagonal elements of L are not stored.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).
out	*IPIV*	int *. ipiv is a pointer to an array of dimesion at least max(1, min (M, N)). ipiv(i) = j, 1 <= i <= min(M, N) implies that rows i and j have been interchanged.

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

ddss_tile
ddss_flat2tiled
ddss_tiled2flat

Definition at line 80 of file kdtpgetrf.c.

References ddss_dflat2tiled(), ddss_dtiled2flat(), and ddss_tile_size().

Referenced by ddss_dtpgetrf().

```
81 {
82
83     // Local variables
84     int i;
85     int mt, nt;
86     int mi, mmi, ki, ni;
87     int tile_size_m;
88     int tile_size_mm;
89     int tile_size_n;
90     int tile_size_km;
91     int tile_size_kn;
92     int local_ipiv_size;
93     int current_position;
94     enum LASS_RETURN ret;
95     int *ipiv_local;
96
97     // Number of tiles
98     if ( M % TILE_SIZE == 0 )
99     {
100         mt = M / TILE_SIZE;
101     }
102     else
103     {
104         mt = ( M / TILE_SIZE ) + 1;
105     }
106
107     if ( N % TILE_SIZE == 0 )
108     {
109         nt = N / TILE_SIZE;
110     }
111     else
112     {
113         nt = ( N / TILE_SIZE ) + 1;
114     }
115
116     /***************************
117     --Tile A matrix allocation--
118     ***************************/
119
120     double ( *TILE_A )[nt][TILE_SIZE * TILE_SIZE] = malloc ( mt * nt *
121         TILE_SIZE * TILE_SIZE * sizeof( double ) );
122
123     if ( TILE_A == NULL )
124     {
125         fprintf( stderr, "Failure in kdnpgetrf for matrix TILE_A\n" );
126         return NoSuccess;
127     }
128
129     ipiv_local = ( int* ) malloc( MIN( M, N ) * sizeof( int ) );
130
131     if ( ipiv_local == NULL )
132     {
133         fprintf( stderr, "Failure in kdnpgetrf for ipiv_local\n" );
134         return NoSuccess;
135     }
136
137     /*********************************************
138     // --From flat data layout to tiled data layout--
```

```
139     ***********************************************/
140
141     ddss_dflat2tiled( M, N, A, LDA, mt, nt, TILE_A );
142
143     /****************
144     --DGETRF tile--
145     ****************/
146
147     current_position = 0;
148     i = 0;
149     for ( ki = 0; ki < MIN( mt, nt ); ki++ )
150     {
151         tile_size_km = ddss_tile_size( M, ki );
152         tile_size_kn = ddss_tile_size( N, ki );
153
154         #pragma oss task inout( TILE_A[ki][ki] ) \
155             inout( ipiv_local ) \
156             inout( current_position ) \
157             out( local_ipiv_size ) \
158             shared( TILE_A ) \
159             firstprivate( ki, i ) \
160             label( dgetrf )
161         {
162             LAPACKE_dgetrf( CblasRowMajor,
163                             tile_size_km, tile_size_kn,
164                             TILE_A[ki][ki], tile_size_kn,
165                             ipiv_local + current_position );
166
167             // Update the global ipiv array
168             local_ipiv_size = MIN( tile_size_km, tile_size_kn );
169             for ( i = 0; i < local_ipiv_size; i++ )
170             {
171                 ( IPIV + current_position )[i] =
172                         ( ipiv_local + current_position )[i] + current_position;
173             }
174             current_position += local_ipiv_size;
175         }
176         for ( mi = ki + 1; mi < mt; mi++ )
177         {
178             tile_size_m = ddss_tile_size( M, mi );
179
180             #pragma oss task in( TILE_A[ki][ki] ) \
181                 inout( TILE_A[mi][ki] ) \
182                 shared( TILE_A ) \
183                 firstprivate( mi, ki ) \
184                 label( dtrsm_below )
185             cblas_dtrsm( CblasRowMajor,
186                          ( CBLAS_SIDE ) Right, ( CBLAS_UPLO ) Upper,
187                          ( CBLAS_TRANSPOSE ) NoTrans, ( CBLAS_DIAG ) NonUnit,
188                          tile_size_m, tile_size_kn,
189                          1.0, TILE_A[ki][ki], tile_size_kn,
190                               TILE_A[mi][ki], tile_size_kn );
191         }
192         for ( ni = 0; ni < ki; ni++ )
193         {
194             tile_size_n = ddss_tile_size( N, ni );
195             // Swap the tiles to the left of A[ki][ki]
196             #pragma oss task in( ipiv_local ) \
197                 in( current_position ) \
198                 in( local_ipiv_size ) \
199                 inout( TILE_A[ki][ni] ) \
200                 shared( TILE_A ) \
201                 firstprivate( ki, ni ) \
202                 label( dlaswp_left )
203             LAPACKE_dlaswp( CblasRowMajor,
204                             tile_size_n, TILE_A[ki][ni], tile_size_n,
205                             1, local_ipiv_size,
206                             ipiv_local + current_position - local_ipiv_size,
207                             1 );
208
209         }
210         for ( ni = ki + 1; ni < nt; ni++ )
211         {
212             tile_size_n = ddss_tile_size( N, ni );
213             // Swap the tiles to the right of A[ki][ki]
214             #pragma oss task in( ipiv_local ) \
215                 in( current_position ) \
216                 in( local_ipiv_size ) \
217                 inout( TILE_A[ki][ni] ) \
218                 shared( TILE_A ) \
219                 firstprivate( ki, ni ) \
220                 label( dlaswp_right )
221             LAPACKE_dlaswp( CblasRowMajor,
222                             tile_size_n, TILE_A[ki][ni], tile_size_n,
223                             1, local_ipiv_size,
224                             ipiv_local + current_position - local_ipiv_size,
225                             1 );
```

```
226
227             #pragma oss task in( TILE_A[ki][ki] ) \
228                 inout( TILE_A[ki][ni] ) \
229                 shared( TILE_A ) \
230                 firstprivate( ni, ki ) \
231                 label( dtrsm_right )
232             cblas_dtrsm( CblasRowMajor,
233                         ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Lower,
234                         ( CBLAS_TRANSPOSE ) NoTrans, ( CBLAS_DIAG ) Unit,
235                         tile_size_km, tile_size_n,
236                         1.0, TILE_A[ki][ki], tile_size_kn,
237                              TILE_A[ki][ni], tile_size_n );
238
239             for ( mmi = ki + 1; mmi < mt; mmi++ )
240             {
241                 tile_size_mm = ddss_tile_size( M, mmi );
242
243                 #pragma oss task in( TILE_A[mmi][ki] ) \
244                     in( TILE_A[ki][ni] ) \
245                     inout( TILE_A[mmi][ni] ) \
246                     shared( TILE_A ) \
247                     firstprivate( ni, mmi, ki ) \
248                     label( dgemm )
249                 cblas_dgemm( CblasRowMajor,
250                              ( CBLAS_TRANSPOSE ) NoTrans,
251                              ( CBLAS_TRANSPOSE ) NoTrans,
252                              tile_size_mm,
253                              tile_size_n,
254                              TILE_SIZE,
255                              -1.0, TILE_A[mmi][ki], tile_size_kn,
256                                    TILE_A[ki][ni],  tile_size_n,
257                               1.0, TILE_A[mmi][ni], tile_size_n );
258             }
259         }
260     }
261
262     // --From tile data layout to flat data layout--
263     ddss_dtiled2flat( M, N, A, LDA, mt, nt, TILE_A );
264
265     // --Free--
266     free( TILE_A );
267     free( ipiv_local );
268
269     return Success;
270
271 }
```

2.1.2.34 enum LASS_RETURN kdtrmm (enum DDSS_SIDE *SIDE*, enum DDSS_UPLO *UPLO*, enum DDSS_TRANS *TRANS_A*, enum DDSS_DIAG *DIAG*, int *M*, int *N*, const double *ALPHA*, double * *A*, int *LDA*, double * *B*, int *LDB*)

Performs one of the matrix-matrix operations:

```
B = ALPHA * op( A ) * B, or B = ALPHA * B * op( A )
```

where op(A) is one of:

```
op( A ) = A      or
op( A ) = A**T
```

ALPHA is a scalar, B is a M by N matrix and A is a unit, or non-unit, upper or lower triangular matrix.

Parameters

in	*SIDE*	enum DDSS_SIDE. SIDE specifies the position of the triangular A matrix in the operations: • Left: B = ALPHA * op(A) * B. • Right: B = ALPHA * B * op(A).

Parameters

in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form in which A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.
in	*TRANS_A*	enum DDSS_TRANS. TRANS_A specifies the form of op(A) to be used: • NoTrans: op(A) = A. • Trans: op(A) = A**T.
in	*DIAG*	enum DDSS_DIAG. DIAG specifies whether or not A is unit triangular as follows: • Unit: A is assumed to be unit triangular. • NonUnit: A is not assumed to be unit triangular.
in	*M*	int. M specifies the number of rows of B. M must be at least zero.
in	*N*	int. N specifies the number of columns of B. N must be at least zero.
in	*ALPHA*	double. ALPHA specifies the scalar alpha.
in	*A*	double *. A is a pointer to a matrix of dimension K by K, where K is M when SIDE = Left and is N otherwise. When UPLO = Uppper the strictly lower triangular part of A is not referenced and when UPLO = Lower the strictly upper triangular part of A is not referenced. Note that when DIAG = Unit, the diagonal elements of A are not referenced either, but are assumed to be unity.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). When SIDE = Left then LDA must be at least max(1, M), otherwise LDA must be at least max(1, N).
in,out	*B*	double *. B is a pointer to a matrix of dimension M by N. On exit the matrix B is overwritten by the transformed matrix.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, N).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

ddss_tile
ddss_flat2tiled
ddss_dsymflat2tiled
ddss_tiled2flat

Definition at line 123 of file kdtrmm.c.

References ddss_dflat2tiled(), ddss_dsymflat2tiled(), ddss_dtiled2flat(), and ddss_tile_size().

Referenced by ddss_dtrmm().

```
128 {
129
130     // Local variables
```

```
131     int k;
132     int mt, nt;
133     int ki, ni, mi, nni, mmi;
134     int Am, An;
135     int Bm, Bn;
136     int tile_size_m;
137     int tile_size_n;
138     int tile_size_k;
139
140     // Number of tiles
141     if ( M % TILE_SIZE == 0 )
142     {
143         mt = M / TILE_SIZE;
144     }
145     else
146     {
147         mt = ( M / TILE_SIZE ) + 1;
148     }
149
150     if ( N % TILE_SIZE == 0 )
151     {
152         nt = N / TILE_SIZE;
153     }
154     else
155     {
156         nt = ( N / TILE_SIZE ) + 1;
157     }
158
159     /****************************
160      --Tile matrices declaration--
161      ****************************/
162
163     if ( SIDE == Left )
164     {
165         Am = An = mt;
166         k = M;
167     }
168     else
169     {
170         Am = An = nt;
171         k = N;
172     }
173
174     Bm = mt;
175     Bn = nt;
176
177     /****************************
178      --Tile matrices allocation--
179      ****************************/
180
181     double ( *TILE_A )[An][TILE_SIZE * TILE_SIZE] = malloc(
182             Am * An * TILE_SIZE * TILE_SIZE * sizeof( double ) );
183
184     if ( TILE_A == NULL )
185     {
186         fprintf( stderr, "Failure in ddss_dtile_alloc for matrix TILE_A\n" );
187         return NoSuccess;
188     }
189
190     double ( *TILE_B )[Bn][TILE_SIZE * TILE_SIZE] = malloc(
191             Bm * Bn * TILE_SIZE * TILE_SIZE * sizeof( double ) );
192
193     if ( TILE_B == NULL )
194     {
195         fprintf( stderr, "Failure in ddss_dtile_alloc for matrix TILE_B\n" );
196         return NoSuccess;
197     }
198
199     /*******************************************
200      --From flat data layout to tiled data layout--
201      *******************************************/
202
203     // From flat matrix A to tile matrix TILE_A
204     ddss_dsymflat2tiled( k, k, A, LDA, Am, An, TILE_A, UPLO );
205
206     // From flat matrix B to tile matrix TILE_B
207     ddss_dflat2tiled( M, N, B, LDB, Bm, Bn, TILE_B );
208
209     /*************
210      --DTRMM tile--
211      *************/
212
213     if ( SIDE == Left )
214     {
215         if ( UPLO == Upper )
216         {
217             // --SIDE = Left & UPLO = Upper & TRANS_A = NoTrans--
```

```
218             if ( TRANS_A == NoTrans )
219             {
220                 for ( ki = 0; ki < mt; ki++ )
221                 {
222                     tile_size_k = ddss_tile_size( M, ki );
223 
224                     for ( ni = 0; ni < nt; ni++ )
225                     {
226                         tile_size_n = ddss_tile_size( N, ni );
227 
228                         #pragma oss task in( TILE_A[ki][ki] ) \
229                                 inout( TILE_B[ki][ni] ) \
230                                 shared( TILE_A, TILE_B ) \
231                                 firstprivate( ki, ni ) \
232                                 label( dtrmm )
233                         cblas_dtrmm( CblasRowMajor,
234                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
235                                      ( CBLAS_TRANSPOSE ) TRANS_A,
236                                      ( CBLAS_DIAG ) DIAG,
237                                      tile_size_k,
238                                      tile_size_n,
239                                      ALPHA, TILE_A[ki][ki], tile_size_k,
240                                             TILE_B[ki][ni], tile_size_n );
241 
242                         for ( mi = ki + 1; mi < mt; mi++ )
243                         {
244                             tile_size_m = ddss_tile_size( M, mi );
245 
246                             #pragma oss task in( TILE_A[ki][mi] ) \
247                                     in( TILE_B[mi][ni] ) \
248                                     inout( TILE_B[ki][ni] ) \
249                                     shared( TILE_A, TILE_B ) \
250                                     firstprivate( ki, mi, ni ) \
251                                     label( dgemm )
252                             cblas_dgemm( CblasRowMajor,
253                                          CblasNoTrans, CblasNoTrans,
254                                          tile_size_k,
255                                          tile_size_n,
256                                          tile_size_m,
257                                          ALPHA, TILE_A[ki][mi],  tile_size_m,
258                                                 TILE_B[mi][ni],  tile_size_n,
259                                          1.0,   TILE_B[ki][ni],  tile_size_n );
260                         }
261                     }
262                 }
263             }
264             // --SIDE = Left & UPLO = Upper & TRANS_A = Trans--
265             else
266             {
267                 for ( ki = mt-1; ki > -1; ki-- )
268                 {
269                     tile_size_k = ddss_tile_size( M, ki );
270 
271                     for ( ni = 0; ni < nt; ni++ )
272                     {
273                         tile_size_n = ddss_tile_size( N, ni );
274 
275                         #pragma oss task in( TILE_A[ki][ki] ) \
276                                 inout( TILE_B[ki][ni] ) \
277                                 shared( TILE_A, TILE_B ) \
278                                 firstprivate( ki, ni ) \
279                                 label( dtrmm )
280                         cblas_dtrmm( CblasRowMajor,
281                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
282                                      ( CBLAS_TRANSPOSE ) TRANS_A,
283                                      ( CBLAS_DIAG ) DIAG,
284                                      tile_size_k,
285                                      tile_size_n,
286                                      ALPHA, TILE_A[ki][ki], tile_size_k,
287                                             TILE_B[ki][ni], tile_size_n );
288 
289                         for ( mi = 0; mi < ki; mi++ )
290                         {
291                             tile_size_m = ddss_tile_size( M, mi );
292 
293                             #pragma oss task in( TILE_A[mi][ki] ) \
294                                     in( TILE_B[mi][ni] ) \
295                                     inout( TILE_B[ki][ni] ) \
296                                     shared( TILE_A, TILE_B ) \
297                                     firstprivate( ki, mi, ni ) \
298                                     label( dgemm )
299                             cblas_dgemm( CblasRowMajor,
300                                          CblasTrans, CblasNoTrans,
301                                          tile_size_k,
302                                          tile_size_n,
303                                          tile_size_m,
304                                          ALPHA, TILE_A[mi][ki],  tile_size_k,
```

```
305                                             TILE_B[mi][ni],  tile_size_n,
306                                     1.0,    TILE_B[ki][ni],  tile_size_n );
307                         }
308                     }
309                 }
310
311             }
312         }
313         // --SIDE = Left & UPLO = Lower & TRANS_A = NoTrans--
314         else
315         {
316             if ( TRANS_A == NoTrans )
317             {
318                 for ( ki = mt-1; ki > -1; ki-- )
319                 {
320                     tile_size_k = ddss_tile_size( M, ki );
321
322                     for ( ni = 0; ni < nt; ni++ )
323                     {
324                         tile_size_n = ddss_tile_size( N, ni );
325
326                         #pragma oss task in( TILE_A[ki][ki] ) \
327                                 inout( TILE_B[ki][ni] ) \
328                                 shared( TILE_A, TILE_B ) \
329                                 firstprivate( ki, ni ) \
330                                 label( dtrmm )
331                         cblas_dtrmm( CblasRowMajor,
332                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
333                                      ( CBLAS_TRANSPOSE ) TRANS_A,
334                                      ( CBLAS_DIAG ) DIAG,
335                                      tile_size_k,
336                                      tile_size_n,
337                                      ALPHA, TILE_A[ki][ki], tile_size_k,
338                                             TILE_B[ki][ni], tile_size_n );
339
340                         for ( mi = 0; mi < ki; mi++ )
341                         {
342                             tile_size_m = ddss_tile_size( M, mi );
343
344                             #pragma oss task in( TILE_A[ki][mi] ) \
345                                     in( TILE_B[mi][ni] ) \
346                                     inout( TILE_B[ki][ni] ) \
347                                     shared( TILE_A, TILE_B ) \
348                                     firstprivate( ki, mi, ni ) \
349                                     label( dgemm )
350                             cblas_dgemm( CblasRowMajor,
351                                          CblasNoTrans, CblasNoTrans,
352                                          tile_size_k,
353                                          tile_size_n,
354                                          tile_size_m,
355                                          ALPHA, TILE_A[ki][mi],  tile_size_m,
356                                                 TILE_B[mi][ni],  tile_size_n,
357                                          1.0,   TILE_B[ki][ni],  tile_size_n );
358                         }
359                     }
360                 }
361
362             }
363             // --SIDE = Left & UPLO = Lower & TRANS_A = Trans--
364             else
365             {
366                 for ( ki = 0; ki < mt; ki++ )
367                 {
368                     tile_size_k = ddss_tile_size( M, ki );
369
370                     for ( ni = 0; ni < nt; ni++ )
371                     {
372                         tile_size_n = ddss_tile_size( N, ni );
373
374                         #pragma oss task in( TILE_A[ki][ki] ) \
375                                 inout( TILE_B[ki][ni] ) \
376                                 shared( TILE_A, TILE_B ) \
377                                 firstprivate( ki, ni ) \
378                                 label( dtrmm )
379                         cblas_dtrmm( CblasRowMajor,
380                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
381                                      ( CBLAS_TRANSPOSE ) TRANS_A,
382                                      ( CBLAS_DIAG ) DIAG,
383                                      tile_size_k,
384                                      tile_size_n,
385                                      ALPHA, TILE_A[ki][ki], tile_size_k,
386                                             TILE_B[ki][ni], tile_size_n );
387
388                         for ( mi = ki + 1; mi < mt; mi++ )
389                         {
390                             tile_size_m = ddss_tile_size( M, mi );
391
```

```
392                             #pragma oss task in( TILE_A[mi][ki] ) \
393                                     in( TILE_B[mi][ni] ) \
394                                     inout( TILE_B[ki][ni] ) \
395                                     shared( TILE_A, TILE_B ) \
396                                     firstprivate( ki, mi, ni ) \
397                                     label( dgemm )
398                             cblas_dgemm( CblasRowMajor,
399                                          CblasTrans, CblasNoTrans,
400                                          tile_size_k,
401                                          tile_size_n,
402                                          tile_size_m,
403                                          ALPHA, TILE_A[mi][ki],  tile_size_k,
404                                                 TILE_B[mi][ni],  tile_size_n,
405                                          1.0,   TILE_B[ki][ni],  tile_size_n );
406                         }
407                     }
408                 }
409             }
410         }
411     }
412     // --SIDE = Right & UPLO = Upper & TRANS_A = NoTrans--
413     else
414     {
415         if ( UPLO == Upper )
416         {
417             if ( TRANS_A == NoTrans )
418             {
419                 for ( ki = nt-1; ki > -1; ki-- )
420                 {
421                     tile_size_k = ddss_tile_size( N, ki );
422 
423                     for ( ni = 0; ni < mt; ni++ )
424                     {
425                         tile_size_n = ddss_tile_size( M, ni );
426 
427                         #pragma oss task in( TILE_A[ki][ki] ) \
428                                 inout( TILE_B[ni][ki] ) \
429                                 shared( TILE_A, TILE_B ) \
430                                 firstprivate( ki, ni ) \
431                                 label( dtrmm )
432                         cblas_dtrmm( CblasRowMajor,
433                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
434                                      ( CBLAS_TRANSPOSE ) TRANS_A,
435                                      ( CBLAS_DIAG ) DIAG,
436                                      tile_size_n,
437                                      tile_size_k,
438                                      ALPHA, TILE_A[ki][ki], tile_size_k,
439                                             TILE_B[ni][ki], tile_size_k );
440 
441                         for ( mi = 0; mi < ki; mi++ )
442                         {
443                             tile_size_m = ddss_tile_size( N, mi );
444 
445                             #pragma oss task in( TILE_B[ni][mi] ) \
446                                     in( TILE_A[mi][ki] ) \
447                                     inout( TILE_B[ni][ki] ) \
448                                     shared( TILE_A, TILE_B ) \
449                                     firstprivate( ki, mi, ni ) \
450                                     label( dgemm )
451                             cblas_dgemm( CblasRowMajor,
452                                          CblasNoTrans, CblasNoTrans,
453                                          tile_size_n,
454                                          tile_size_k,
455                                          tile_size_m,
456                                          ALPHA, TILE_B[ni][mi],  tile_size_m,
457                                                 TILE_A[mi][ki],  tile_size_k,
458                                          1.0,   TILE_B[ni][ki],  tile_size_k );
459                         }
460                     }
461                 }
462             }
463             // --SIDE = Right & UPLO = Upper & TRANS_A = Trans--
464             else
465             {
466                 for ( ki = 0; ki < nt; ki++ )
467                 {
468                     tile_size_k = ddss_tile_size( N, ki );
469 
470                     for ( ni = 0; ni < mt; ni++ )
471                     {
472                         tile_size_n = ddss_tile_size( M, ni );
473 
474                         #pragma oss task in( TILE_A[ki][ki] ) \
475                                 inout( TILE_B[ni][ki] ) \
476                                 shared( TILE_A, TILE_B ) \
477                                 firstprivate( ki, ni ) \
478                                 label( dtrmm )
```

```
479                         cblas_dtrmm( CblasRowMajor,
480                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
481                                      ( CBLAS_TRANSPOSE ) TRANS_A,
482                                      ( CBLAS_DIAG ) DIAG,
483                                      tile_size_n,
484                                      tile_size_k,
485                                      ALPHA, TILE_A[ki][ki], tile_size_k,
486                                             TILE_B[ni][ki], tile_size_k );
487
488                         for ( mi = ki+1; mi < nt; mi++ )
489                         {
490                             tile_size_m = ddss_tile_size( N, mi );
491
492                             #pragma oss task in( TILE_B[ni][mi] ) \
493                                     in( TILE_A[ki][mi] ) \
494                                     inout( TILE_B[ni][ki] ) \
495                                     shared( TILE_A, TILE_B ) \
496                                     firstprivate( ki, mi, ni ) \
497                                     label( dgemm )
498                             cblas_dgemm( CblasRowMajor,
499                                          CblasNoTrans, CblasTrans,
500                                          tile_size_n,
501                                          tile_size_k,
502                                          tile_size_m,
503                                          ALPHA, TILE_B[ni][mi],  tile_size_m,
504                                                 TILE_A[ki][mi],  tile_size_m,
505                                          1.0,   TILE_B[ni][ki],  tile_size_k );
506                         }
507                     }
508                 }
509             }
510         }
511         // --SIDE = Right & UPLO = Lower & TRANS_A = NoTrans--
512         else
513         {
514             if ( TRANS_A == NoTrans )
515             {
516                 for ( ki = 0; ki < nt; ki++ )
517                 {
518                     tile_size_k = ddss_tile_size( N, ki );
519
520                     for ( ni = 0; ni < mt; ni++ )
521                     {
522                         tile_size_n = ddss_tile_size( M, ni );
523
524                         #pragma oss task in( TILE_A[ki][ki] ) \
525                                 inout( TILE_B[ni][ki] ) \
526                                 shared( TILE_A, TILE_B ) \
527                                 firstprivate( ki, ni ) \
528                                 label( dtrmm )
529                         cblas_dtrmm( CblasRowMajor,
530                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
531                                      ( CBLAS_TRANSPOSE ) TRANS_A,
532                                      ( CBLAS_DIAG ) DIAG,
533                                      tile_size_n,
534                                      tile_size_k,
535                                      ALPHA, TILE_A[ki][ki], tile_size_k,
536                                             TILE_B[ni][ki], tile_size_k );
537
538                         for ( mi = ki+1; mi < nt; mi++ )
539                         {
540                             tile_size_m = ddss_tile_size( N, mi );
541
542                             #pragma oss task in( TILE_B[ni][mi] ) \
543                                     in( TILE_A[mi][ki] ) \
544                                     inout( TILE_B[ni][ki] ) \
545                                     shared( TILE_A, TILE_B ) \
546                                     firstprivate( ki, mi, ni ) \
547                                     label( dgemm )
548                             cblas_dgemm( CblasRowMajor,
549                                          CblasNoTrans, CblasNoTrans,
550                                          tile_size_n,
551                                          tile_size_k,
552                                          tile_size_m,
553                                          ALPHA, TILE_B[ni][mi],  tile_size_m,
554                                                 TILE_A[mi][ki],  tile_size_k,
555                                          1.0,   TILE_B[ni][ki],  tile_size_k );
556                         }
557                     }
558                 }
559             }
560             // --SIDE = Right & UPLO = Lower & TRANS_A = Trans--
561             else
562             {
563                 for ( ki = nt-1; ki > -1; ki-- )
564                 {
565                     tile_size_k = ddss_tile_size( N, ki );
```

```
566
567                     for ( ni = 0; ni < mt; ni++ )
568                     {
569                         tile_size_n = ddss_tile_size( M, ni );
570
571                         #pragma oss task in( TILE_A[ki][ki] ) \
572                                 inout( TILE_B[ni][ki] ) \
573                                 shared( TILE_A, TILE_B ) \
574                                 firstprivate( ki, ni ) \
575                                 label( dtrmm )
576                         cblas_dtrmm( CblasRowMajor,
577                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
578                                      ( CBLAS_TRANSPOSE ) TRANS_A,
579                                      ( CBLAS_DIAG ) DIAG,
580                                      tile_size_n,
581                                      tile_size_k,
582                                      ALPHA, TILE_A[ki][ki], tile_size_k,
583                                             TILE_B[ni][ki], tile_size_k );
584
585                         for ( mi = 0; mi < ki; mi++ )
586                         {
587                             tile_size_m = ddss_tile_size( N, mi );
588
589                             #pragma oss task in( TILE_B[ni][mi] ) \
590                                     in( TILE_A[ki][mi] ) \
591                                     inout( TILE_B[ni][ki] ) \
592                                     shared( TILE_A, TILE_B ) \
593                                     firstprivate( ki, mi, ni ) \
594                                     label( dgemm )
595                             cblas_dgemm( CblasRowMajor,
596                                          CblasNoTrans, CblasTrans,
597                                          tile_size_n,
598                                          tile_size_k,
599                                          tile_size_m,
600                                          ALPHA, TILE_B[ni][mi],  tile_size_m,
601                                                 TILE_A[ki][mi],  tile_size_m,
602                                          1.0,   TILE_B[ni][ki],  tile_size_k );
603                         }
604                     }
605                 }
606             }
607         }
608     }
609
610     /*****************************************
611      --From tiled data layout to flat data layout--
612      *****************************************/
613
614     ddss_dtiled2flat( M, N, B, LDB, Bm, Bn, TILE_B );
615
616     // --Tile matrices free--
617     free( TILE_A );
618     free( TILE_B );
619
620     return Success;
621
622 }
```

2.1.2.35 enum LASS_RETURN kdtrsm (enum DDSS_SIDE *SIDE*, enum DDSS_UPLO *UPLO*, enum DDSS_TRANS *TRANS_A*, enum DDSS_DIAG *DIAG*, int *M*, int *N*, const double *ALPHA*, double * *A*, int *LDA*, double * *B*, int *LDB*)

Solves one of the matrix equations:

```
op( A ) * X = ALPHA * B, or X * op( A ) = ALPHA * B
```

where op(A) is one of:

```
op( A ) = A      or
op( A ) = A**T
```

ALPHA is a scalar, X and B are M by N matrices, A is a unit, or non-unit, upper or lower triangular matrix. The matrix X is overwritten on B

Parameters

in	*SIDE*	enum DDSS_SIDE. SIDE specifies the position of the triangular A matrix in the operations: • Left: op(A) ∗ X = ALPHA ∗ B. • Right: X ∗ op(A) = ALPHA ∗ B.
in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form of A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.
in	*TRANS↩ _A*	enum DDSS_TRANS. TRANS_A specifies the form of op(A) to be used: • NoTrans: op(A) = A. • Trans: op(A) = A∗∗T.
in	*DIAG*	enum DDSS_DIAG. DIAG specifies whether or not A is unit triangular as follows: • Unit: A is assumed to be unit triangular. • NonUnit: A is not assumed to be unit triangular.
in	*M*	int. M specifies the number of rows of B. M must be at least zero.
in	*N*	int. N specifies the number of columns of B. N must be at least zero.
in	*ALPHA*	double. ALPHA specifies the scalar alpha.
in	*A*	double ∗. A is a pointer to a matrix of dimension K by K, where K is M when SIDE = Left and is N otherwise. When UPLO = Uppper the strictly lower triangular part of A is not referenced and when UPLO = Lower the strictly upper triangular part of A is not referenced. Note that when DIAG = Unit, the diagonal elements of A are not referenced either, but are assumed to be unity.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). When SIDE = Left then LDA must be at least max(1, M), otherwise LDA must be at least max(1, N).
in,out	*B*	double ∗. B is a pointer to a matrix of dimension M by N. On exit the matrix B is overwritten by the solution matrix X.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, N).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

ddss_tile
ddss_flat2tiled
ddss_dsymflat2tiled
ddss_tiled2flat

Definition at line 123 of file kdtrsm.c.

References ddss_dflat2tiled(), ddss_dsymflat2tiled(), ddss_dtiled2flat(), and ddss_tile_size().

Referenced by ddss_dtrsm().

```
128 {
129
130     // Local variables
131     int k;
132     int mt, nt;
133     int ki, ni, mi, nni, mmi;
134     int Am, An;
135     int Bm, Bn;
136     int tile_size_m;
137     int tile_size_mm;
138     int tile_size_n;
139     int tile_size_nn;
140     int tile_size_k;
141     double alpha;
142
143     // Number of tiles
144     if ( M % TILE_SIZE == 0 )
145     {
146         mt = M / TILE_SIZE;
147     }
148     else
149     {
150         mt = ( M / TILE_SIZE ) + 1;
151     }
152
153     if ( N % TILE_SIZE == 0 )
154     {
155         nt = N / TILE_SIZE;
156     }
157     else
158     {
159         nt = ( N / TILE_SIZE ) + 1;
160     }
161
162     /*****************************
163      --Tile matrices declaration--
164      *****************************/
165
166     if ( SIDE == Left )
167     {
168         Am = An = mt;
169         k = M;
170     }
171     else
172     {
173         Am = An = nt;
174         k = N;
175     }
176
177     Bm = mt;
178     Bn = nt;
179
180     /***************************
181      --Tile matrices allocation--
182      ***************************/
183
184     double ( *TILE_A )[An][TILE_SIZE * TILE_SIZE] = malloc(
185               Am * An * TILE_SIZE * TILE_SIZE * sizeof( double ) );
186
187     if ( TILE_A == NULL )
188     {
189         fprintf( stderr, "Failure in ddss_dtile_alloc for matrix TILE_A\n" );
190         return NoSuccess;
191     }
192
193     double ( *TILE_B )[Bn][TILE_SIZE * TILE_SIZE] = malloc(
194               Bm * Bn * TILE_SIZE * TILE_SIZE * sizeof( double ) );
195
196     if ( TILE_B == NULL )
197     {
198         fprintf( stderr, "Failure in ddss_dtile_alloc for matrix TILE_B\n" );
199         return NoSuccess;
200     }
201
202     /*****************************************
203      --From flat data layout to tiled data layout--
204      *****************************************/
205
206     // From flat matrix A to tile matrix TILE_A
207     ddss_dsymflat2tiled( k, k, A, LDA, Am, An, TILE_A, UPLO );
208
209     // From flat matrix B to tile matrix TILE_B
210     ddss_dflat2tiled( M, N, B, LDB, Bm, Bn, TILE_B );
211
212     /*************
213      --DTRSM tile--
214      *************/
```

```
215
216     if ( SIDE == Left )
217     {
218         if ( UPLO == Upper )
219         {
220             // --SIDE = Left & UPLO = Upper & TRANS_A = NoTrans--
221             if ( TRANS_A == NoTrans )
222             {
223                 for ( ki = mt - 1; ki >= 0; ki-- )
224                 {
225                     tile_size_k = ddss_tile_size( M, ki );
226                     if ( ki == mt - 1 )
227                     {
228                         alpha = ALPHA;
229                     }
230                     else
231                     {
232                         alpha = 1.0;
233                     }
234
235                     for ( ni = 0; ni < nt; ni++ )
236                     {
237                         tile_size_n = ddss_tile_size( N, ni );
238
239                         #pragma oss task in( TILE_A[ki][ki] ) \
240                             inout( TILE_B[ki][ni] ) \
241                             shared( TILE_A, TILE_B ) \
242                             firstprivate( ki, ni ) \
243                             label( dtrsm )
244                         cblas_dtrsm( CblasRowMajor,
245                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
246                                      ( CBLAS_TRANSPOSE ) TRANS_A,
247                                      ( CBLAS_DIAG ) DIAG,
248                                      tile_size_k,
249                                      tile_size_n,
250                                      alpha, TILE_A[ki][ki], tile_size_k,
251                                             TILE_B[ki][ni], tile_size_n );
252                     }
253
254                     for ( mi = ki - 1; mi >= 0; mi-- )
255                     {
256                         tile_size_m = ddss_tile_size( M, mi );
257                         for ( nni = 0; nni < nt; nni++ )
258                         {
259                             tile_size_nn = ddss_tile_size( N, nni );
260
261                             #pragma oss task in( TILE_A[mi][ki] ) \
262                                 in( TILE_B[ki][nni] ) \
263                                 inout( TILE_B[mi][nni] ) \
264                                 shared( TILE_A, TILE_B ) \
265                                 firstprivate( ki, mi, nni ) \
266                                 label( dgemm )
267                             cblas_dgemm( CblasRowMajor,
268                                          CblasNoTrans, CblasNoTrans,
269                                          tile_size_m,
270                                          tile_size_nn,
271                                          tile_size_k,
272                                          -1.0,  TILE_A[mi][ki],  tile_size_k,
273                                                 TILE_B[ki][nni], tile_size_nn,
274                                          alpha, TILE_B[mi][nni], tile_size_nn );
275                         }
276                     }
277                 }
278             }
279
280             // --SIDE = Left & UPLO = Upper & TRANS_A = Trans--
281             else
282             {
283                 for ( ki = 0; ki < mt; ki++ )
284                 {
285                     tile_size_k = ddss_tile_size( M, ki );
286                     if ( ki == 0 )
287                     {
288                         alpha = ALPHA;
289                     }
290                     else
291                     {
292                         alpha = 1.0;
293                     }
294
295                     for ( ni = 0; ni < nt; ni++ )
296                     {
297                         tile_size_n = ddss_tile_size( N, ni );
298
299                         #pragma oss task in( TILE_A[ki][ki] ) \
300                             inout( TILE_B[ki][ni] ) \
301                             shared( TILE_A, TILE_B ) \
```

```
302                             firstprivate( ki, ni ) \
303                             label( dtrsm )
304                         cblas_dtrsm( CblasRowMajor,
305                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
306                                      ( CBLAS_TRANSPOSE ) TRANS_A,
307                                      ( CBLAS_DIAG ) DIAG,
308                                      tile_size_k,
309                                      tile_size_n,
310                                      alpha, TILE_A[ki][ki], tile_size_k,
311                                             TILE_B[ki][ni], tile_size_n );
312                     }
313
314                     for ( mi = ki + 1; mi < mt; mi++ )
315                     {
316                         tile_size_m = ddss_tile_size( M, mi );
317                         for ( nni = 0; nni < nt; nni++ )
318                         {
319                             tile_size_nn = ddss_tile_size( N, nni );
320
321                             #pragma oss task in( TILE_A[ki][mi]) \
322                                 in( TILE_B[ki][nni] ) \
323                                 inout(  TILE_B[mi][nni] ) \
324                                 shared( TILE_A, TILE_B ) \
325                                 firstprivate( ki, mi, nni ) \
326                                 label( dgemm )
327                             cblas_dgemm( CblasRowMajor,
328                                          CblasTrans, CblasNoTrans,
329                                          tile_size_m,
330                                          tile_size_nn,
331                                          tile_size_k,
332                                          -1.0,  TILE_A[ki][mi],  tile_size_m,
333                                                 TILE_B[ki][nni], tile_size_nn,
334                                          alpha, TILE_B[mi][nni], tile_size_nn );
335                         }
336                     }
337                 }
338             }
339         }
340
341         // --SIDE = Left & UPLO = Lower & TRANS_A = NoTrans--
342         else
343         {
344             if ( TRANS_A == NoTrans )
345             {
346                 for ( ki = 0; ki < mt; ki++ )
347                 {
348                     tile_size_k = ddss_tile_size( M, ki );
349                     if ( ki == 0 )
350                     {
351                         alpha = ALPHA;
352                     }
353                     else
354                     {
355                         alpha = 1.0;
356                     }
357
358                     for ( ni = 0; ni < nt; ni++ )
359                     {
360                         tile_size_n = ddss_tile_size( N, ni );
361
362                         #pragma oss task in( TILE_A[ki][ki] ) \
363                             inout(  TILE_B[ki][ni] ) \
364                             shared( TILE_A, TILE_B ) \
365                             firstprivate( ki, ni ) \
366                             label( dtrsm )
367                         cblas_dtrsm( CblasRowMajor,
368                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
369                                      ( CBLAS_TRANSPOSE ) TRANS_A,
370                                      ( CBLAS_DIAG ) DIAG,
371                                      tile_size_k,
372                                      tile_size_n,
373                                      alpha, TILE_A[ki][ki], tile_size_k,
374                                             TILE_B[ki][ni], tile_size_n );
375
376                     }
377                     for ( nni = 0; nni < nt; nni++ )
378                     {
379                         tile_size_nn = ddss_tile_size( N, nni );
380
381                         for ( mi = ki + 1; mi < mt; mi++ )
382                         {
383
384                             tile_size_m = ddss_tile_size( M, mi );
385
386                             #pragma oss task in( TILE_A[mi][ki] ) \
387                                 in( TILE_B[ki][nni] ) \
388                                 inout( TILE_B[mi][nni] ) \
```

```
389                                 shared( TILE_A, TILE_B ) \
390                                 firstprivate( ki, mi, nni ) \
391                                 label( dgemm )
392                             cblas_dgemm( CblasRowMajor,
393                                         CblasNoTrans, CblasNoTrans,
394                                         tile_size_m,
395                                         tile_size_nn,
396                                         tile_size_k,
397                                         -1.0,   TILE_A[mi][ki],  tile_size_k,
398                                                TILE_B[ki][nni], tile_size_nn,
399                                         alpha, TILE_B[mi][nni], tile_size_nn );
400                         }
401                     }
402                 }
403             }
404
405             // --SIDE = Left & UPLO = Lower & TRANS_A = Trans--
406             else
407             {
408                 for ( ki = mt - 1; ki >= 0; ki-- )
409                 {
410                     tile_size_k = ddss_tile_size( M, ki );
411                     if ( ki == mt - 1 )
412                     {
413                         alpha = ALPHA;
414                     }
415                     else
416                     {
417                         alpha = 1.0;
418                     }
419
420                     for ( ni = 0; ni < nt; ni++ )
421                     {
422                         tile_size_n = ddss_tile_size( N, ni );
423
424                         #pragma oss task in( TILE_A[ki][ki] ) \
425                             inout( TILE_B[ki][ni] ) \
426                             shared( TILE_A, TILE_B ) \
427                             firstprivate( ki, ni ) \
428                             label( dtrsm )
429                         cblas_dtrsm( CblasRowMajor,
430                                     ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
431                                     ( CBLAS_TRANSPOSE ) TRANS_A,
432                                     ( CBLAS_DIAG ) DIAG,
433                                     tile_size_k,
434                                     tile_size_n,
435                                     alpha, TILE_A[ki][ki], tile_size_k,
436                                            TILE_B[ki][ni], tile_size_n );
437                     }
438
439                     for ( mi = ki - 1; mi >= 0; mi-- )
440                     {
441                         tile_size_m = ddss_tile_size( M, mi );
442                         for ( nni = 0; nni < nt; nni++ )
443                         {
444                             tile_size_nn = ddss_tile_size( N, nni );
445
446                             #pragma oss task in( TILE_A[ki][mi] ) \
447                                 in( TILE_B[ki][nni] ) \
448                                 inout(  TILE_B[mi][nni] ) \
449                                 shared( TILE_A, TILE_B ) \
450                                 firstprivate( ki, mi, nni ) \
451                                 label( dgemm )
452                             cblas_dgemm( CblasRowMajor,
453                                         CblasTrans, CblasNoTrans,
454                                         tile_size_m,
455                                         tile_size_nn,
456                                         tile_size_k,
457                                         -1.0,  TILE_A[ki][mi],  tile_size_m,
458                                                TILE_B[ki][nni], tile_size_nn,
459                                         alpha, TILE_B[mi][nni], tile_size_nn );
460                         }
461                     }
462                 }
463             }
464         }
465     }
466
467     // --SIDE = Right & UPLO = Upper & TRANS_A = NoTrans--
468     else
469     {
470         if ( UPLO == Upper )
471         {
472             if ( TRANS_A == NoTrans )
473             {
474                 for ( ki = 0; ki < nt; ki++ )
475                 {
```

```
476                     tile_size_k = ddss_tile_size( N, ki );
477                     if ( ki == 0 )
478                     {
479                         alpha = ALPHA;
480                     }
481                     else
482                     {
483                         alpha = 1.0;
484                     }
485 
486                     for ( mi = 0; mi < mt; mi++ )
487                     {
488                         tile_size_m = ddss_tile_size( M, mi );
489 
490                         #pragma oss task in( TILE_A[ki][ki] ) \
491                             inout(  TILE_B[mi][ki] ) \
492                             shared( TILE_A, TILE_B ) \
493                             firstprivate( ki, mi ) \
494                             label( dtrsm )
495                         cblas_dtrsm( CblasRowMajor,
496                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
497                                      ( CBLAS_TRANSPOSE ) TRANS_A,
498                                      ( CBLAS_DIAG ) DIAG,
499                                      tile_size_m,
500                                      tile_size_k,
501                                      alpha, TILE_A[ki][ki], tile_size_k,
502                                             TILE_B[mi][ki], tile_size_k );
503                     }
504 
505                     for ( ni = ki + 1; ni < nt; ni++ )
506                     {
507                         tile_size_n = ddss_tile_size( N, ni );
508                         for ( mmi = 0; mmi < mt; mmi++ )
509                         {
510                             tile_size_mm = ddss_tile_size( M, mmi );
511 
512                             #pragma oss task in( TILE_A[ki][ni] ) \
513                                 in( TILE_B[mmi][ki] ) \
514                                 inout(  TILE_B[mmi][ni] ) \
515                                 shared( TILE_A, TILE_B ) \
516                                 firstprivate( ki, ni, mmi ) \
517                                 label( dgemm )
518                             cblas_dgemm( CblasRowMajor,
519                                          CblasNoTrans, CblasNoTrans,
520                                          tile_size_mm,
521                                          tile_size_n,
522                                          tile_size_k,
523                                          -1.0,  TILE_B[mmi][ki], tile_size_k,
524                                                 TILE_A[ki][ni],  tile_size_n,
525                                          alpha, TILE_B[mmi][ni], tile_size_n );
526                         }
527                     }
528                 }
529             }
530             // --SIDE = Right & UPLO = Upper & TRANS_A = Trans--
531             else
532             {
533                 for ( ki = nt - 1; ki >= 0; ki-- )
534                 {
535                     tile_size_k = ddss_tile_size( N, ki );
536                     if ( ki == nt - 1 )
537                     {
538                         alpha = ALPHA;
539                     }
540                     else
541                     {
542                         alpha = 1.0;
543                     }
544 
545                     for ( mi = 0; mi < mt; mi++ )
546                     {
547                         tile_size_m = ddss_tile_size( M, mi );
548 
549                         #pragma oss task in( TILE_A[ki][ki] ) \
550                             inout( TILE_B[mi][ki] ) \
551                             shared( TILE_A, TILE_B ) \
552                             firstprivate( ki, mi ) \
553                             label( dtrsm )
554                         cblas_dtrsm( CblasRowMajor,
555                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
556                                      ( CBLAS_TRANSPOSE ) TRANS_A,
557                                      ( CBLAS_DIAG ) DIAG,
558                                      tile_size_m,
559                                      tile_size_k,
560                                      alpha, TILE_A[ki][ki], tile_size_k,
561                                             TILE_B[mi][ki], tile_size_k );
562                     }
```

```
563
564                     for ( ni = ki - 1; ni >= 0; ni-- )
565                     {
566                         tile_size_n = ddss_tile_size( N, ni );
567                         for ( mmi = 0; mmi < mt; mmi++ )
568                         {
569                             tile_size_mm = ddss_tile_size( M, mmi );
570
571                             #pragma oss task in( TILE_A[ni][ki] ) \
572                                 in( TILE_B[mmi][ki] ) \
573                                 inout( TILE_B[mmi][ni] ) \
574                                 shared( TILE_A, TILE_B ) \
575                                 firstprivate( ki, ni, mmi ) \
576                                 label( dgemm )
577                             cblas_dgemm( CblasRowMajor,
578                                         CblasNoTrans, CblasTrans,
579                                         tile_size_mm,
580                                         tile_size_n,
581                                         tile_size_k,
582                                         -1.0,  TILE_B[mmi][ki], tile_size_k,
583                                                TILE_A[ni][ki],  tile_size_k,
584                                         alpha, TILE_B[mmi][ni], tile_size_n );
585                         }
586                     }
587                 }
588             }
589         }
590
591         // --SIDE = Right & UPLO = Lower & TRANS_A = NoTrans--
592         else
593         {
594             if ( TRANS_A == NoTrans )
595             {
596                 for ( ki = nt - 1; ki >= 0; ki-- )
597                 {
598                     tile_size_k = ddss_tile_size( N, ki );
599                     if ( ki == nt - 1 )
600                     {
601                         alpha = ALPHA;
602                     }
603                     else
604                     {
605                         alpha = 1.0;
606                     }
607
608                     for ( mi = 0; mi < mt; mi++ )
609                     {
610                         tile_size_m = ddss_tile_size( M, mi );
611
612                         #pragma oss task in( TILE_A[ki][ki] ) \
613                             inout( TILE_B[mi][ki] ) \
614                             shared( TILE_A, TILE_B ) \
615                             firstprivate( ki, mi ) \
616                             label( dtrsm )
617                         cblas_dtrsm( CblasRowMajor,
618                                     ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
619                                     ( CBLAS_TRANSPOSE ) TRANS_A,
620                                     ( CBLAS_DIAG ) DIAG,
621                                     tile_size_m,
622                                     tile_size_k,
623                                     alpha, TILE_A[ki][ki], tile_size_k,
624                                            TILE_B[mi][ki], tile_size_k );
625                     }
626
627                     for ( ni = ki - 1; ni >= 0; ni-- )
628                     {
629                         tile_size_n = ddss_tile_size( N, ni );
630                         for ( mmi = 0; mmi < mt; mmi++ )
631                         {
632                             tile_size_mm = ddss_tile_size( M, mmi );
633
634                             #pragma oss task in( TILE_A[ki][ni] ) \
635                                 in( TILE_B[mmi][ki] ) \
636                                 inout( TILE_B[mmi][ni] ) \
637                                 shared( TILE_A, TILE_B ) \
638                                 firstprivate( ki, ni, mmi ) \
639                                 label( dgemm )
640                             cblas_dgemm( CblasRowMajor,
641                                         CblasNoTrans, CblasNoTrans,
642                                         tile_size_mm,
643                                         tile_size_n,
644                                         tile_size_k,
645                                         -1.0,  TILE_B[mmi][ki], tile_size_k,
646                                                TILE_A[ki][ni],  tile_size_n,
647                                         alpha, TILE_B[mmi][ni], tile_size_n );
648                         }
649                     }
```

```
650                 }
651             }
652
653             // --SIDE = Right & UPLO = Lower & TRANS_A = Trans--
654             else
655             {
656                 for ( ki = 0; ki < nt; ki++ )
657                 {
658                     tile_size_k = ddss_tile_size( N, ki );
659                     if ( ki == 0 )
660                     {
661                         alpha = ALPHA;
662                     }
663                     else
664                     {
665                         alpha = 1.0;
666                     }
667
668                     for ( mi = 0; mi < mt; mi++ )
669                     {
670                         tile_size_m = ddss_tile_size( M, mi );
671
672                         #pragma oss task in( TILE_A[ki][ki] ) \
673                             inout( TILE_B[mi][ki] ) \
674                             shared( TILE_A,TILE_B ) \
675                             firstprivate( ki, mi ) \
676                             label( dtrsm )
677                         cblas_dtrsm( CblasRowMajor,
678                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
679                                      ( CBLAS_TRANSPOSE ) TRANS_A,
680                                      ( CBLAS_DIAG ) DIAG,
681                                      tile_size_m,
682                                      tile_size_k,
683                                      alpha, TILE_A[ki][ki], tile_size_k,
684                                             TILE_B[mi][ki], tile_size_k );
685                     }
686
687                     for ( ni = ki + 1; ni < nt; ni++ )
688                     {
689                         tile_size_n = ddss_tile_size( N, ni );
690                         for ( mmi = 0; mmi < mt; mmi++ )
691                         {
692                             tile_size_mm = ddss_tile_size( M, mmi );
693
694                             #pragma oss task in( TILE_A[ni][ki] ) \
695                                 in( TILE_B[mmi][ki] ) \
696                                 inout(  TILE_B[mmi][ni] ) \
697                                 shared( TILE_A, TILE_B ) \
698                                 firstprivate( ki, ni, mmi ) \
699                                 label( dgemm )
700                             cblas_dgemm( CblasRowMajor,
701                                          CblasNoTrans, CblasTrans,
702                                          tile_size_mm,
703                                          tile_size_n,
704                                          tile_size_k,
705                                          -1.0,  TILE_B[mmi][ki], tile_size_k,
706                                                 TILE_A[ni][ki],  tile_size_k,
707                                          alpha, TILE_B[mmi][ni], tile_size_n );
708                         }
709                     }
710                 }
711             }
712         }
713     }
714
715     /*****************************************
716      --From tiled data layout to flat data layout--
717      *****************************************/
718
719     ddss_dtiled2flat( M, N, B, LDB, Bm, Bn, TILE_B );
720
721     // --Tile matrices free--
722     free( TILE_A );
723     free( TILE_B );
724
725     return Success;
726
727 }
```

2.2 include/lass_macros.h File Reference

Macros definition.

This graph shows which files directly or indirectly include this file:

Macros

- #define **TILE_SIZE** 512
- #define **NUM_CORES** 48
- #define **MAX**(a, b) (((a) > (b)) ? (a) : (b))
- #define **MIN**(a, b) (((a) < (b)) ? (a) : (b))
- #define **FMULS_GEMM**(m_, n_, k_) ((m_) ∗ (n_) ∗ (k_))
- #define **FADDS_GEMM**(m_, n_, k_) ((m_) ∗ (n_) ∗ (k_))
- #define **FLOPS_DGEMM**(m_, n_, k_)
- #define **FMULS_SYMM**(side_, m_, n_) (((side_) == Left) ? FMULS_GEMM((m_), (m_), (n_)) : FMULS_↩ GEMM((m_), (n_), (n_)))
- #define **FADDS_SYMM**(side_, m_, n_) (((side_) == Left) ? FADDS_GEMM((m_), (m_), (n_)) : FADDS_↩ GEMM((m_), (n_), (n_)))
- #define **FLOPS_DSYMM**(side_, m_, n_) (FMULS_SYMM(side_, (double)(m_), (double)(n_)) + FADDS_S↩ YMM(side_, (double)(m_), (double)(n_)))
- #define **FMULS_TRSM_2**(m_, n_) (0.5 ∗ (n_) ∗ (m_) ∗ ((m_) + 1))
- #define **FADDS_TRSM_2**(m_, n_) (0.5 ∗ (n_) ∗ (m_) ∗ ((m_) - 1))
- #define **FMULS_TRSM**(side_, m_, n_) (((side_) == Left) ? FMULS_TRSM_2((m_), (n_)) : FMULS_TRS↩ M_2((n_), (m_)))
- #define **FADDS_TRSM**(side_, m_, n_) (((side_) == Left) ? FADDS_TRSM_2((m_), (n_)) : FADDS_TRS↩ M_2((n_), (m_)))
- #define **FLOPS_DTRSM**(side_, m_, n_) (FMULS_TRSM(side_, (double)(m_), (double)(n_)) + FADDS_T↩ RSM(side_, (double)(m_), (double)(n_)))
- #define **FMULS_TRMM_2**(m_, n_) (0.5 ∗ (n_) ∗ (m_) ∗ ((m_) + 1))
- #define **FADDS_TRMM_2**(m_, n_) (0.5 ∗ (n_) ∗ (m_) ∗ ((m_) - 1))
- #define **FMULS_TRMM**(side_, m_, n_) (((side_) == Left) ? FMULS_TRMM_2((m_), (n_)) : FMULS_TR↩ MM_2((n_), (m_)))
- #define **FADDS_TRMM**(side_, m_, n_) (((side_) == Left) ? FADDS_TRMM_2((m_), (n_)) : FADDS_TR↩ MM_2((n_), (m_)))
- #define **FLOPS_DTRMM**(side_, m_, n_) (FMULS_TRMM(side_, (double)(m_), (double)(n_)) + FADDS_T↩ RMM(side_, (double)(m_), (double)(n_)))
- #define **FMULS_SYRK**(k_, n_) (0.5 ∗ (k_) ∗ (n_) ∗ ((n_) + 1))
- #define **FADDS_SYRK**(k_, n_) (0.5 ∗ (k_) ∗ (n_) ∗ ((n_)+1))
- #define **FLOPS_DSYRK**(k_, n_) (FMULS_SYRK((double)(k_), (double)(n_)) + FADDS_SYRK((double)(k↩ _), (double)(n_)))
- #define **FMULS_SYR2K**(k_, n_) ((k_) ∗ (n_) ∗ (n_))
- #define **FADDS_SYR2K**(k_, n_) ((k_) ∗ (n_) ∗ (n_) + (n_))
- #define **FLOPS_DSYR2K**(k_, n_) (FMULS_SYR2K((double)(k_), (double)(n_)) + FADDS_SYR2K((double)(k↩ _), (double)(n_)))
- #define **FMULS_POTRF**(n_)
- #define **FADDS_POTRF**(n_) ((1./6.) ∗ (n_) ∗ (n_) ∗ (n_) - (1./6.) ∗ (n_))
- #define **FLOPS_DPOTRF**(n_) (FMULS_POTRF((double)(n_)) + FADDS_POTRF((double)(n_)))
- #define **FMULS_GETRF**(m_, n_) (((m_) >= (n_)) ? 0.5 ∗ (m_) ∗ (n_) ∗ (n_) - 1./6. ∗ (n_) ∗ (n_) ∗ (n_) + 0.5 ∗ (m_) ∗ (n_) - 0.5 ∗ (n_) ∗ (n_) + 2./3. ∗ (n_) : 0.5 ∗ (n_) ∗ (m_) ∗ (m_) - 1./6. ∗ (m_) ∗ (m_) ∗ (m_) + 0.5 ∗ (n_) ∗ (m_) - 0.5 ∗ (m_) ∗ (m_) + 2./3. ∗ (m_))
- #define **FADDS_GETRF**(m_, n_) (((m_) >= (n_)) ? 0.5 ∗ (m_) ∗ (n_) ∗ (n_) - 1./6. ∗ (n_) ∗ (n_) ∗ (n_) - 0.5 ∗ (m_) ∗ (n_) + 1./6. ∗ (n_) : 0.5 ∗ (n_) ∗ (m_) ∗ (m_) - 1./6. ∗ (m_) ∗ (m_) ∗ (m_) - 0.5 ∗ (n_) ∗ (m_) + 1./6. ∗ (m_))

- #define **FLOPS_DGETRF**(m_, n_) (FMULS_GETRF((double)(m_), (double)(n_)) + FADDS_GET↩ RF((double)(m_), (double)(n_)))
- #define **DSSS_MACROS_H**
- #define **FLOPS_DGTSV**(n_) (8 ∗ (n_))
- #define **FLOPS_DSPMV**(nnz_) (2 ∗ (nnz_))

2.2.1 Detailed Description

Macros definition.

LASs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es

Date

2017-01-02

2.2.2 Macro Definition Documentation

2.2.2.1 #define FLOPS_DGEMM(*m_*, *n_*, *k_*)

Value:

```
( FMULS_GEMM((double)(m_), (double)(n_), \
    (double)(k_)) + FADDS_GEMM((double)(m_), (double)(n_), (double)(k_)) )
```

Definition at line 40 of file lass_macros.h.

2.2.2.2 #define FMULS_POTRF(*n_*)

Value:

```
( (1./6.) * (n_) * (n_) * (n_) + (0.5) * (n_) * (n_) \
    + (1./3.) * n_ )
```

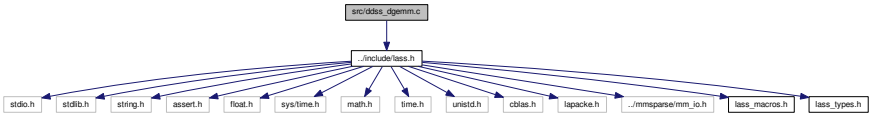
Definition at line 93 of file lass_macros.h.

2.3 src/ddss_dgemm.c File Reference

LASs-DDSs ddss_dgemm routine.

```
#include "../include/lass.h"
```

Include dependency graph for ddss_dgemm.c:



Functions

- int ddss_dgemm (enum DDSS_TRANS TRANS_A, enum DDSS_TRANS TRANS_B, int M, int N, int K, double ALPHA, double ∗A, int LDA, double ∗B, int LDB, double BETA, double ∗C, int LDC)

2.3.1 Detailed Description

LASs-DDSs ddss_dgemm routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es

Date

2017-01-02

2.3.2 Function Documentation

2.3.2.1 int ddss_dgemm (enum DDSS_TRANS *TRANS_A,* enum DDSS_TRANS *TRANS_B,* int *M,* int *N,* int *K,* double *ALPHA,* double ∗ *A,* int *LDA,* double ∗ *B,* int *LDB,* double *BETA,* double ∗ *C,* int *LDC*)

Performs the matrix-matrix operation:

```
C = ALPHA * op( A ) * op( B ) + BETA * C
```

where op(X) is one of:

```
op( X ) = X      or
op( X ) = X**T
```

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

in	*TRANS_A*	enum DDSS_TRANS. TRANS_A specifies the form of op(A) to be used in the matrix multiplication as follows: • NoTrans: op(A) = A. • Trans: op(A) = A**T.
in	*TRANS_B*	enum DDSS_TRANS. TRANS_B specifies the form of op(B) to be used in the matrix multiplication as follows: • NoTrans: op(B) = B. • Trans: op(B) = B**T.
in	*M*	int. M specifies the number of rows of the matrix A and the number of rows of the matrix C. M must be greater than zero.
in	*N*	int. N specifies the number of columns of the matrix B and the number of columns of the matrix C. N must be greater than zero.
in	*K*	int. K specifies the number of columns of the matrix A and the number of rows of the matrix B. K must be greater than zero.
in	*ALPHA*	double.
in	*A*	double *. A is a pointer to a matrix of dimension Ma (rows) by Ka (columns), where Ma is M and Ka is K when TRANS_A = NoTrans, and Ma is K and Ka is M otherwise.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). When TRANS_A = NoTrans then LDA must be at least max(1, K), otherwise LDA must be at least max(1, M).
in	*B*	double *. B is a pointer to a matrix of dimension Kb (rows) by Nb (columns), where Kb is K and Nb is N when TRANS_B = NoTrans, and Kb is N and Nb is K otherwise.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). When TRANS_B = NoTrans then LDB must be at least max(1, N), otherwise LDB must be at least max(1, K).
in	*BETA*	double.
in,out	*C*	double *. C is a pointer to a matrix of dimension M by N. On exit, C is overwritten by the M by N matrix (ALPHA*op(A)*op(B) + BETA*C).
in	*LDC*	int. LDC specifies the number of columns of C (row-major order). LDC must be at least max(1, N).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

kdgemm

Definition at line 128 of file ddss_dgemm.c.

References kdgemm().

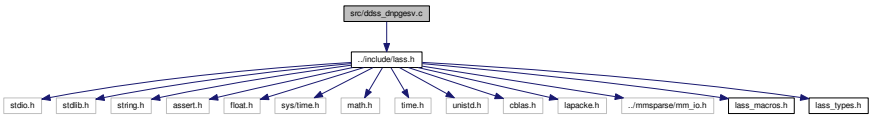
```
133 {
134
135     // Local variables
136     int An, Bn;
```

```
137
138     // Argument checking
139     if ( ( TRANS_A != NoTrans ) && ( TRANS_A != Trans ) )
140     {
141         fprintf( stderr, "Illegal value of TRANS_A, in ddss_dgemm code\n" );
142         return NoSuccess;
143     }
144
145     if ( ( TRANS_B != NoTrans ) && ( TRANS_B != Trans ) )
146     {
147         fprintf( stderr, "Illegal value of TRANS_B, in ddss_dgemm code\n" );
148         return NoSuccess;
149     }
150
151     if ( M < 0 )
152     {
153         fprintf( stderr, "Illegal value of M, in ddss_dgemm code\n" );
154         return NoSuccess;
155     }
156
157     if ( N < 0 )
158     {
159         fprintf( stderr, "Illegal value of N, in ddss_dgemm code\n" );
160         return NoSuccess;
161     }
162
163     if ( K < 0 )
164     {
165         fprintf( stderr, "Illegal value of K, in ddss_dgemm code\n" );
166         return NoSuccess;
167     }
168
169     if ( TRANS_A == NoTrans )
170     {
171         An = K;
172     }
173     else
174     {
175         An = M;
176     }
177
178     if ( LDA < MAX( 1, An ) )
179     {
180         fprintf( stderr, "Illegal value of LDA, in ddss_dgemm code\n" );
181         return NoSuccess;
182     }
183
184     if ( TRANS_B == NoTrans )
185     {
186         Bn = N;
187     }
188     else
189     {
190         Bn = K;
191     }
192
193     if ( LDB < MAX( 1, Bn ) )
194     {
195         fprintf( stderr, "Illegal value of LDB, in ddss_dgemm code\n" );
196         return NoSuccess;
197     }
198
199     if ( LDC < MAX( 1, N ) )
200     {
201         fprintf( stderr, "Illegal value of LDC, in ddss_dgemm code\n" );
202         return NoSuccess;
203     }
204
205     // Quick return
206     if ( M == 0 || N == 0 || ( ( ALPHA == 0.0 || K == 0 ) && BETA == 1.0 ) )
207     {
208         return Success;
209     }
210
211     return kdgemm( TRANS_A, TRANS_B, M, N, K,
212                    (const double) ALPHA, A, LDA,
213                                          B, LDB,
214                    (const double)  BETA, C, LDC );
215
216 }
```

2.4 src/ddss_dnpgesv.c File Reference

LASs-DDSs ddss_dnpgesv routine.

```
#include "../include/lass.h"
```
Include dependency graph for ddss_dnpgesv.c:



Functions

- int ddss_dnpgesv (int N, int NRHS, double ∗A, int LDA, double ∗B, int LDB)

2.4.1 Detailed Description

LASs-DDSs ddss_dnpgesv routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es

Date

2018-07-26

2.4.2 Function Documentation

2.4.2.1 int ddss_dnpgesv (int *N*, int *NRHS*, double ∗ *A*, int *LDA*, double ∗ *B*, int *LDB*)

Solves a system of linear equations A X = B, where A is a N-by-N general matrix and X and B are N-by-NRHS matrices. The matrix A is factorized using the LU descomposition without pivoting. The matrix A is descomposed as:

```
A = L * U
```

where L is a lower triangular matrix with unit diagonal elements and U is an upper triangular matrix.

Parameters

in	*N*	int. N specifies the order of the square matrix A. N >= 0.
in	*NRHS*	int. NRHS specifies the number of right-hand-sides (number of columns of B). NRHS >= 0.
in,out	*A*	double ∗. A is a pointer to a regular matrix of dimension N-by-LDA. On exit, if return value is Success, the matrix A is overwriten by the factors L and U. The unit diagonal elements of L are not stored.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

kdnpgesv

Definition at line 84 of file ddss_dnpgesv.c.

References kdnpgesv().

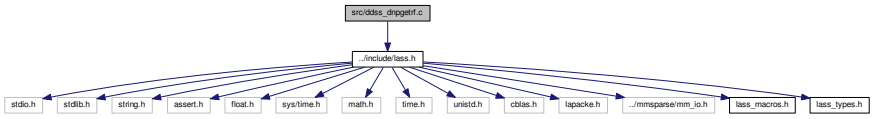
```
87 {
88
89     // Argument checking
90     if ( N < 0 )
91     {
92         fprintf( stderr, "Illegal value of N, in ddss_dnpgesv code\n" );
93         return NoSuccess;
94     }
95
96     if ( NRHS < 0 )
97     {
98         fprintf( stderr, "Illegal value of NRHS, in ddss_dnpgesv code\n" );
99         return NoSuccess;
100     }
101
102     if ( LDA < MAX( 1, N ) )
103     {
104         fprintf( stderr, "Illegal value of LDA, in ddss_dnpgesv code\n" );
105         return NoSuccess;
106     }
107
108     if ( LDB < MAX( 1, NRHS ) )
109     {
110         fprintf( stderr, "Illegal value of LDB, in ddss_dnpgesv code\n" );
111         return NoSuccess;
112     }
113
114     // Quick return
115     if ( MAX( N, 0 ) == 0 || MAX( NRHS, 0 ) == 0 )
116     {
117         return Success;
118     }
119
120     return kdnpgesv( N, NRHS, A, LDA, B, LDB );
121
122 }
```

2.5 src/ddss_dnpgetrf.c File Reference

LASs-DDSs ddss_dnpgetrf routine.

```
#include "../include/lass.h"
```

Include dependency graph for ddss_dnpgetrf.c:



Functions

- int ddss_dnpgetrf (int M, int N, double *A, int LDA)

2.5.1 Detailed Description

LASs-DDSs ddss_dnpgetrf routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es
Boro Sofranac boro.sofranac@bsc.es

Date

2018-04-06

2.5.2 Function Documentation

2.5.2.1 int ddss_dnpgetrf (int *M*, int *N*, double * *A*, int *LDA*)

Performs the LU factorization without pivoting of a general M-by-N matrix A:

```
A = L * U
```

where L is a lower triangular (lower trapezoidal if M > N) matrix with unit diagonal elements and U is an upper triangular (upper trapezoidal if M < N) matrix.

Parameters

in	*M*	int. M specifies the number of rows of the matrix A. M >= 0.
in	*N*	int. N specifies the number of columns of the matrix A. N >= 0.
in,out	*A*	double *. A is a pointer to a regular matrix of dimension M-by-N. On exit, if return value is Success, the matrix A is overwriten by the factors L and U. The unit diagonal elements of L are not stored.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

knpdgetrf

Definition at line 70 of file ddss_dnpgetrf.c.

References kdnpgetrf().

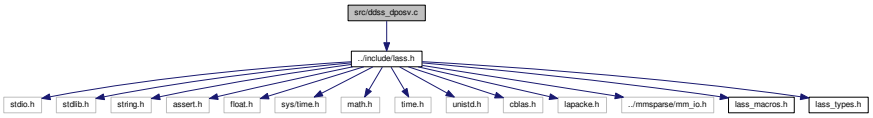
```
71 {
72
73     // Argument checking
74     if ( M < 0 )
75     {
76         fprintf( stderr, "Illegal value of M, in ddss_dnpgetrf code\n" );
77         return NoSuccess;
78     }
79
80     if ( N < 0 )
81     {
82         fprintf( stderr, "Illegal value of N, in ddss_dnpgetrf code\n" );
83         return NoSuccess;
84     }
85
86     if ( LDA < MAX( 1, N ) )
87     {
88         fprintf( stderr, "Illegal value of LDA, in ddss_dnpgetrf code\n" );
89         return NoSuccess;
90     }
91
92     // Qick return
93     if ( MIN( M, N ) == 0 )
94     {
95         return Success;
96     }
97
98     return kdnpgetrf( M, N, A, LDA );
99
100 }
```

2.6 src/ddss_dposv.c File Reference

LASs-DDSs ddss_dpotrf routine.

```
#include "../include/lass.h"
```

Include dependency graph for ddss_dposv.c:



Functions

- int ddss_dposv (enum DDSS_UPLO UPLO, int N, int NRHS, double *A, int LDA, double *B, int LDB)

2.6.1 Detailed Description

LASs-DDSs ddss_dpotrf routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es

Date

2018-05-04

2.6.2 Function Documentation

2.6.2.1 int ddss_dposv (enum DDSS_UPLO *UPLO*, int *N*, int *NRHS*, double * *A*, int *LDA*, double * *B*, int *LDB*)

Solves a system of linear equations A X = B, where A is a m-by-m simmetric positive definite matrix and X and B are m-by-nrhs matrices. The matrix A is descomposed as:

```
A = L \times L^T
or
A = U^T \times U
```

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

Parameters

in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form of A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.
in	*N*	int. N specifies the order of the square matrix A. N >= 0.
in	*NRHS*	int. NRHS specifies the number of right-hand-sides (number of columns of B). NRHS >= 0.
in,out	*A*	double *. A is a pointer to a positive definite matrix of dimension N by LDA. On exit, if return value is Success, the matrix A is overwriten by the factor U or L.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).
in,out	*B*	double *. B is a pointer to a matrix of dimension N by NRHS, which stores the right-hand-sides of the systems of linear equations. (row-major order). On exit, if return value is Success, the matrix B is overwriten by the solution matrix X.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, NRHS).

Return values

Success	sucessful exit

Return values

NoSuccess	unsucessful exit

See also

kdposv

Definition at line 92 of file ddss_dposv.c.

References kdposv().

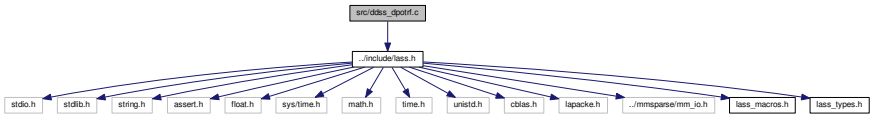
```
96 {
97
98     // Argument checking
99     if ( ( UPLO != Upper ) && ( UPLO != Lower ) )
100     {
101         fprintf( stderr, "Illegal value of UPLO, in ddss_dposv code\n" );
102         return NoSuccess;
103     }
104
105     if ( N < 0 )
106     {
107         fprintf( stderr, "Illegal value of N, in ddss_dposv code\n" );
108         return NoSuccess;
109     }
110
111     if ( NRHS < 0 )
112     {
113         fprintf( stderr, "Illegal value of NRHS, in ddss_dposv code\n" );
114         return NoSuccess;
115     }
116
117     if ( LDA < MAX( 1, N ) )
118     {
119         fprintf( stderr, "Illegal value of LDA, in ddss_dposv code\n" );
120         return NoSuccess;
121     }
122
123     if ( LDB < MAX( 1, NRHS ) )
124     {
125         fprintf( stderr, "Illegal value of LDB, in ddss_dposv code\n" );
126         return NoSuccess;
127     }
128
129     // Quick return
130     if ( MAX( N, 0 ) == 0 || MAX( NRHS, 0 ) == 0 )
131     {
132         return Success;
133     }
134
135     return kdposv( UPLO, N, NRHS, A, LDA, B, LDB );
136
137 }
```

2.7 src/ddss_dpotrf.c File Reference

LASs-DDSs ddss_dpotrf routine.

```
#include "../include/lass.h"
```

Include dependency graph for ddss_dpotrf.c:



Functions

- int ddss_dpotrf (enum DDSS_UPLO UPLO, int N, double *A, int LDA)

2.7.1 Detailed Description

LASs-DDSs ddss_dpotrf routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es

Date

2017-15-08

2.7.2 Function Documentation

2.7.2.1 int ddss_dpotrf (enum DDSS_UPLO *UPLO*, int *N*, double * *A*, int *LDA*)

Performs the Cholesky factorization of a symmetric positive definite matrix A:

```
A = L \times L^T
or
A = U^T \times U
```

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

Parameters

in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form of A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.
in	*N*	int. N specifies the order of the square matrix A. N >= 0.
in,out	*A*	double *. A is a pointer to a positive definite matrix of dimension N by LDA. On exit, if return value is Success, the matrix A is overwriten by the factor U or L.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

kdpotrf

Definition at line 74 of file ddss_dpotrf.c.

References kdpotrf().

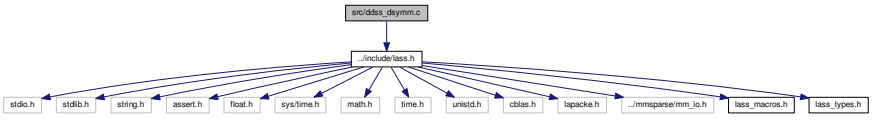
```
75 {
76
77     // Argument checking
78     if ( ( UPLO != Upper ) && ( UPLO != Lower ) )
79     {
80         fprintf( stderr, "Illegal value of UPLO, in ddss_dportf code\n" );
81         return NoSuccess;
82     }
83
84     if ( N < 0 )
85     {
86         fprintf( stderr, "Illegal value of N, in ddss_dportf code\n" );
87         return NoSuccess;
88     }
89
90     if ( LDA < MAX( 1, N ) )
91     {
92         fprintf( stderr, "Illegal value of LDA, in ddss_dportf code\n" );
93         return NoSuccess;
94     }
95
96     // Quick return
97     if ( MAX( N, 0 ) == 0 )
98     {
99         return Success;
100     }
101
102     return kdpotrf( UPLO, N, A, LDA );
103
104 }
```

2.8 src/ddss_dsymm.c File Reference

LASs-DDSs ddss_dsymm routine.

```
#include "../include/lass.h"
```

Include dependency graph for ddss_dsymm.c:



Functions

- int ddss_dsymm (enum DDSS_SIDE SIDE, enum DDSS_UPLO UPLO, int M, int N, double ALPHA, double *A, int LDA, double *B, int LDB, double BETA, double *C, int LDC)

2.8.1 Detailed Description

LASs-DDSs ddss_dsymm routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es

Date

2017-07-11

2.8.2 Function Documentation

2.8.2.1 int ddss_dsymm (enum DDSS_SIDE *SIDE*, enum DDSS_UPLO *UPLO*, int *M*, int *N*, double *ALPHA*, double * *A*, int *LDA*, double * *B*, int *LDB*, double *BETA*, double * *C*, int *LDC*)

Performs one of the matrix-matrix operations:

```
C = ALPHA * A * B + BETA * C
```

or

```
C = ALPHA * B * A + BETA * C
```

where op(X) is one of:

```
op( X ) = X      or
op( X ) = X**T
```

ALPHA and BETA are scalars, A is a symmetric matrix, and B and C are M by N matrices.

Parameters

in	*SIDE*	enum DDSS_SIDE. UPLO specifies the position of the symmetric A matrix in the operation: • Left: C = ALPHA * A * B + BETA * C • Right: C = ALPHA * B * A + BETA * C
in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form of A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.
in	*M*	int. M specifies the number of rows of the matrix C. M must be equal or greater than zero.
in	*N*	int. N specifies the number of columns of the matrix C. N must be equal or greater than zero.
in	*ALPHA*	double.

Parameters

in	*A*	double *. A is a pointer to a matrix of dimension Ma (rows) by Na (columns), where Ma is M and Na is M when SIDE = Left, and Ma is N and Na is N when SIDE = Right
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, Na).
in	*B*	double *. B is a pointer to a matrix of dimension M by N.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, N).
in	*BETA*	double.
in,out	*C*	double *. C is a pointer to a matrix of dimension M by N. On exit, C is overwritten by the M by N matrix.
in	*LDC*	int. LDC specifies the number of columns of C (row-major order). LDC must be at least max(1, N).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

kdsymm

Definition at line 120 of file ddss_dsymm.c.

References kdsymm().

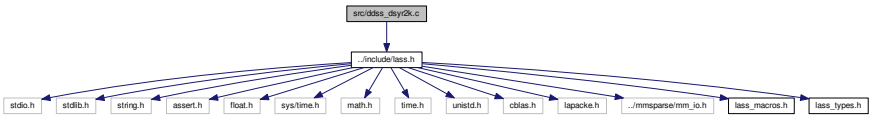
```
125 {
126
127     // Local variables
128     int nA;
129
130     // Argument checking
131     if ( ( SIDE != Left ) && ( SIDE != Right ) )
132     {
133         fprintf( stderr, "Illegal value of SIDE, in ddss_dsymm code\n" );
134         return NoSuccess;
135     }
136
137     if ( ( UPLO != Upper ) && ( UPLO != Lower ) )
138     {
139         fprintf( stderr, "Illegal value of UPLO, in ddss_dsymm code\n" );
140         return NoSuccess;
141     }
142
143     if ( M < 0 )
144     {
145         fprintf( stderr, "Illegal value of M, in ddss_dsymm code\n" );
146         return NoSuccess;
147     }
148
149     if ( N < 0 )
150     {
151         fprintf( stderr, "Illegal value of N, in ddss_dsymm code\n" );
152         return NoSuccess;
153     }
154
155     // Quick return
156     if ( M == 0 || N == 0 || ( ALPHA == 0.0 && BETA == 1.0 ) )
157     {
158         return Success;
159     }
160
161     if ( SIDE == Left )
162     {
```

```
163         nA = M;
164     }
165     else
166     {
167         nA = N;
168     }
169
170     if ( LDA < MAX( 1, nA ) )
171     {
172         fprintf( stderr, "Illegal value of LDA, in ddss_dsymm code\n" );
173         return NoSuccess;
174     }
175
176     if ( LDB < MAX( 1, N ) )
177     {
178         fprintf( stderr, "Illegal value of LDB, in ddss_dsymm code\n" );
179         return NoSuccess;
180     }
181
182     if ( LDC < MAX( 1, N ) )
183     {
184         fprintf( stderr, "Illegal value of LDC, in ddss_dsymm code\n" );
185         return NoSuccess;
186     }
187
188     return kdsymm( SIDE, UPLO, M, N,
189                    ( const double ) ALPHA, A, LDA,
190                                            B, LDB,
191                    ( const double )  BETA, C, LDC );
192
193 }
```

2.9 src/ddss_dsyr2k.c File Reference

LASs-DDSs ddss_dsyr2k routine.

`#include "../include/lass.h"`
Include dependency graph for ddss_dsyr2k.c:



Functions

- int ddss_dsyr2k (enum DDSS_UPLO UPLO, enum DDSS_TRANS TRANS, int N, int K, const double ALPHA, double *A, int LDA, double *B, int LDB, const double BETA, double *C, int LDC)

2.9.1 Detailed Description

LASs-DDSs ddss_dsyr2k routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es
Boro Sofranac boro.sofranac@bsc.es

Date

2018-02-21

2.9.2 Function Documentation

2.9.2.1 int ddss_dsyr2k (enum DDSS_UPLO *UPLO,* enum DDSS_TRANS *TRANS,* int *N,* int *K,* const double *ALPHA,* double * *A,* int *LDA,* double * *B,* int *LDB,* const double *BETA,* double * *C,* int *LDC*)

Performs one of the symmetric rank 2k operations:

```
C = ALPHA * A * B**T + ALPHA * B * A**T + BETA * C or
C = ALPHA * A**T * B + ALPHA * B**T * A + BETA * C
```

ALPHA and BETA are scalars, C is an N by N symmetric matrix and A and B are N by K matrices in the first case and K by N matrices in the second case.

Parameters

in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form in which C is stored: • Lower: Lower triangular part of C is stored. The upper traingular part is not referenced. • Upper: Upper triangular part of C is stored. The lower triangular part is not referenced.
in	*TRANS*	enum DDSS_TRANS. TRANS specifies the operation to be performed as follows: • NoTrans: C = ALPHA * A * B**T + ALPHA * B * A**T + BETA * C • Trans: C = ALPHA * A**T * B + ALPHA * B**T * A + BETA * C
in	*N*	int. N specifies the order of matrix C. N must be at least zero.
in	*K*	int. With TRANS = NoTrans, K specifies the number of columns of the matrices A and B, and with TRANS = Trans, K specifies the number of rows of the matrices A and B. K must be at least zero.
in	*ALPHA*	double. ALPHA specifies the scalar alpha.
in	*A*	double *. A is a pointer to a matrix of dimension Na (rows) by Ka (columns), where Na is N and Ka is K when TRANS = NoTrans, and Na is K and Ka is N otherwise.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). When TRANS = NoTrans then LDA must be at least max(1, K), otherwise LDA must be at least max(1, N).
in	*B*	double *. B is a pointer to a matrix of dimension Nb (rows) by Kb (columns), where Nb is N and Kb is K when TRANS = NoTrans, and Nb is K and Kb is N otherwise.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). When TRANS = NoTrans then LDB must be at least max(1, K), otherwise LDB must be at least max(1, N).
in	*BETA*	double. BETA specifies the scalar beta.
in,out	*C*	double *. C is a pointer to a matrix of dimension N by N. When UPLO = Uppper the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of C is overwritten by the upper triangular part of the updated solution matrix C. When UPLO = Lower the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of C is overwritten by the lower triangular part of the updated solution matrix C.
in	*LDC*	int. LDC specifies the number of columns of C (row-major order). LDC must be at least max(1, N).

See also

kdsyr2k

Definition at line 114 of file ddss_dsyr2k.c.

References kdsyr2k().

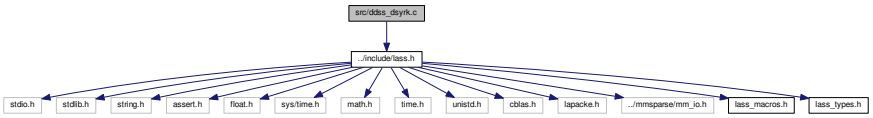
```
119 {
120     // Local variables
121     int nA, nB;
122
123     // Argument checking
124     if ( ( UPLO != Upper ) && ( UPLO != Lower ) )
125     {
126         fprintf( stderr, "Illegal value of UPLO, in ddss_dsyr2k code\n" );
127         return NoSuccess;
128     }
129
130     if ( ( TRANS != NoTrans ) && ( TRANS != Trans ) )
131     {
132         fprintf( stderr, "Illegal value of TRANS, in ddss_dsyr2k code\n" );
133         return NoSuccess;
134     }
135
136     if ( N < 0 )
137     {
138         fprintf( stderr, "Illegal value of N, in ddss_dsyr2k code\n" );
139         return NoSuccess;
140     }
141
142     if ( K < 0 )
143     {
144         fprintf( stderr, "Illegal value of K, in ddss_dsyr2k code\n" );
145         return NoSuccess;
146     }
147
148     if ( TRANS == NoTrans )
149     {
150         nA = nB = K;
151     }
152     else
153     {
154         nA = nB = N;
155     }
156
157     if ( LDA < MAX( 1, nA ) )
158     {
159         fprintf( stderr, "Illegal value of LDA, in ddss_dsyr2k code\n" );
160         return NoSuccess;
161     }
162
163     if ( LDB < MAX( 1, nB ) )
164     {
165         fprintf( stderr, "Illegal value of LDB, in ddss_dsyr2k code\n" );
166         return NoSuccess;
167     }
168
169     if ( LDC < MAX( 1, N ) )
170     {
171         fprintf( stderr, "Illegal value of LDC, in ddss_dsyr2k code\n" );
172         return NoSuccess;
173     }
174
175     // Quick return
176      if ( N == 0 || ( ( ALPHA == 0.0 || K == 0 ) && BETA == 1.0 ) )
177      {
178          return Success;
179      }
180
181     return kdsyr2k( UPLO, TRANS,
182                     N, K,
183                     ALPHA, A, LDA,
184                            B, LDB,
185                     BETA,  C, LDC );
186
187 }
```

2.10 src/ddss_dsyrk.c File Reference

LASs-DDSs ddss_dsyrk routine.

```
#include "../include/lass.h"
```

Include dependency graph for ddss_dsyrk.c:



Functions

- int ddss_dsyrk (enum DDSS_UPLO UPLO, enum DDSS_TRANS TRANS_A, int N, int K, const double AL↩ PHA, double ∗A, int LDA, const double BETA, double ∗C, int LDC)

2.10.1 Detailed Description

LASs-DDSs ddss_dsyrk routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es
Boro Sofranac boro.sofranac@bsc.es

Date

2018-02-13

2.10.2 Function Documentation

2.10.2.1 int ddss_dsyrk (enum DDSS_UPLO *UPLO,* enum DDSS_TRANS *TRANS_A,* int *N,* int *K,* const double *ALPHA,* double ∗ *A,* int *LDA,* const double *BETA,* double ∗ *C,* int *LDC*)

Performs one of the symmetric rank k operations:

```
C = ALPHA * A * op( A ) + BETA * C  or
C = ALPHA * op( A ) * A + BETA * C
```

where op(X) is:

```
op( X ) = X**T
```

ALPHA and BETA are scalars, C is an N by N symmetric matrix and A is an N by K matrix in the first case and a K by N matrix in the second case.

Parameters

in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form in which C is stored: • Lower: Lower triangular part of C is stored. The upper traingular part is not referenced. • Upper: Upper triangular part of C is stored. The lower triangular part is not referenced.
in	*TRANS_A*	enum DDSS_TRANS. TRANS_A specifies the operation to be performed as follows: • NoTrans: C = ALPHA ∗ A ∗ A**T + BETA ∗ C • Trans: C = ALPHA ∗ A**T ∗ A + BETA ∗ C
in	*N*	int. N specifies the order of matrix C. N must be at least zero.
in	*K*	int. With TRANS_A = NoTrans, K specifies the number of columns of the matrix A, and with TRANS_A = Trans, K specifies the number of rows of the matrix A. K must be at least zero.
in	*ALPHA*	double. ALPHA specifies the scalar alpha.
in	*A*	double ∗. A is a pointer to a matrix of dimension Na (rows) by Ka (columns), where Na is N and Ka is K when TRANS_A = NoTrans, and Na is K and Ka is N otherwise.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). When TRANS_A = NoTrans then LDA must be at least max(1, K), otherwise LDA must be at least max(1, N).
in	*BETA*	double. BETA specifies the scalar beta.
in,out	*C*	double ∗. C is a pointer to a matrix of dimension N by N. When UPLO = Uppper the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of C is overwritten by the upper triangular part of the updated solution matrix C. When UPLO = Lower the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of C is overwritten by the lower triangular part of the updated solution matrix C.
in	*LDC*	int. LDC specifies the number of columns of C (row-major order). LDC must be at least max(1, N).

See also

kdsyrk

Definition at line 106 of file ddss_dsyrk.c.

References kdsyrk().

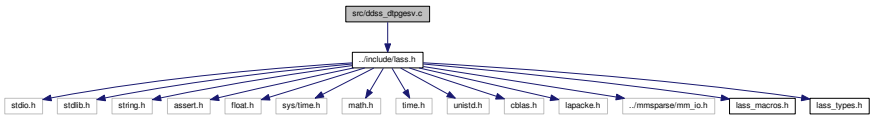
```
110 {
111
112     // Local variables
113     int nA;
114
115     // Argument checking
116     if ( ( UPLO != Upper ) && ( UPLO != Lower ) )
117     {
118         fprintf( stderr, "Illegal value of UPLO, in ddss_dsyrk code\n" );
119         return NoSuccess;
120     }
121
122     if ( ( TRANS_A != NoTrans ) && ( TRANS_A != Trans ) )
123     {
124         fprintf( stderr, "Illegal value of TRANS_A, in ddss_dsyrk code\n" );
125         return NoSuccess;
126     }
127
128     if ( N < 0 )
```

```
129     {
130         fprintf( stderr, "Illegal value of N, in ddss_dsyrk code\n" );
131         return NoSuccess;
132     }
133 
134     if ( K < 0 )
135     {
136         fprintf( stderr, "Illegal value of K, in ddss_dsyrk code\n" );
137         return NoSuccess;
138     }
139 
140     if ( TRANS_A == NoTrans )
141     {
142         nA = K;
143     }
144     else
145     {
146         nA = N;
147     }
148 
149     if ( LDA < MAX( 1, nA ) )
150     {
151         fprintf( stderr, "Illegal value of LDA, in ddss_dsyrk code\n" );
152         return NoSuccess;
153     }
154 
155     if ( LDC < MAX( 1, N ) )
156     {
157         fprintf( stderr, "Illegal value of LDC, in ddss_dsyrk code\n" );
158         return NoSuccess;
159     }
160 
161     // Quick return
162      if ( N == 0 || ( ( ALPHA == 0.0 || K == 0 ) && BETA == 1.0 ) )
163      {
164          return Success;
165      }
166 
167     return kdsyrk( UPLO, TRANS_A,
168                    N, K,
169                    ALPHA, A, LDA,
170                    BETA,  C, LDC );
171 
172 }
```

2.11 src/ddss_dtpgesv.c File Reference

LASs-DDSs ddss_dtpgesv routine.

`#include "../include/lass.h"`
Include dependency graph for ddss_dtpgesv.c:



Functions

- int ddss_dtpgesv (int N, int NRHS, double ∗A, int LDA, int ∗IPIV, double ∗B, int LDB)

2.11.1 Detailed Description

LASs-DDSs ddss_dtpgesv routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es

Date

2018-09-20

2.11.2 Function Documentation

2.11.2.1 int ddss_dtpgesv (int *N*, int *NRHS*, double * *A*, int *LDA*, int * *IPIV*, double * *B*, int *LDB*)

Solves a system of linear equations A X = B, where A is a N-by-N general matrix and X and B are N-by-NRHS matrices. The matrix A is factorized using the LU descomposition with tiled-pivoting. The matrix A is descomposed as:

```
A = P * L * U
```

where P is a permutation matrix, L is a lower triangular matrix with unit diagonal elements and U is an upper triangular matrix.

Parameters

in	*N*	int. N specifies the order of the square matrix A. N >= 0.

NRHS int. NRHS specifies the number of right-hand-sides (number of columns of B). NRHS >= 0.

Parameters

in,out	*A*	double *. A is a pointer to a regular matrix of dimension N-by-LDA. On exit, if return value is Success, the matrix A is overwriten by the factors L and U. The unit diagonal elements of L are not stored.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).
out	*IPIV*	int *. ipiv is a pointer to an array of dimesion at least max(1, min (M, N)). ipiv(i) = j, 1 <= i <= min(M, N) implies that rows i and j have been interchanged.
in,out	*B*	double *. B is a pointer to a matrix of dimension N by NRHS, which stores the right-hand-sides of the systems of linear equations. (row-major order). On exit, if return value is Success, the matrix B is overwriten by the solution matrix X.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, NRHS).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

kdtpgesv

Definition at line 89 of file ddss_dtpgesv.c.

References kdtpgesv().

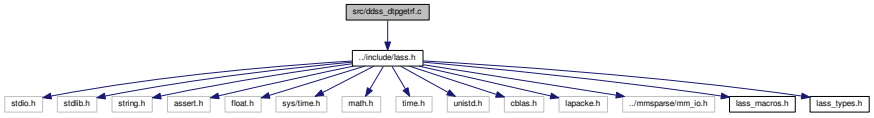
```
93 {
94
95     // Argument checking
96     if ( N < 0 )
97     {
98         fprintf( stderr, "Illegal value of N, in ddss_dtpgesv code\n" );
99         return NoSuccess;
100     }
101
102     if ( NRHS < 0 )
103     {
104         fprintf( stderr, "Illegal value of NRHS, in ddss_dtpgesv code\n" );
105         return NoSuccess;
106     }
107
108     if ( LDA < MAX( 1, N ) )
109     {
110         fprintf( stderr, "Illegal value of LDA, in ddss_dtpgesv code\n" );
111         return NoSuccess;
112     }
113
114     if ( LDB < MAX( 1, NRHS ) )
115     {
116         fprintf( stderr, "Illegal value of LDB, in ddss_dtpgesv code\n" );
117         return NoSuccess;
118     }
119
120     // Quick return
121     if ( MAX( N, 0 ) == 0 || MAX( NRHS, 0 ) == 0 )
122     {
123         return Success;
124     }
125
126     return kdtpgesv( N, NRHS, A, LDA, IPIV, B, LDB );
127
128 }
```

2.12 src/ddss_dtpgetrf.c File Reference

LASs-DDSs ddss_dtpgetrf routine.

```
#include "../include/lass.h"
```

Include dependency graph for ddss_dtpgetrf.c:



Functions

- int ddss_dtpgetrf (int M, int N, double *A, int LDA, int *IPIV)

2.12.1 Detailed Description

LASs-DDSs ddss_dtpgetrf routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es
Boro Sofranac boro.sofranac@bsc.es

Date

2018-04-08 2018-05-08

2.12.2 Function Documentation

2.12.2.1 int ddss_dtpgetrf (int *M*, int *N*, double * *A*, int *LDA*, int * *IPIV*)

Performs the LU factorization with tiled pivoting (row interchanges) of a general M-by-N matrix A:

```
A = P * L * U
```

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

Parameters

in	*M*	int. M specifies the number of rows of the matrix A. M >= 0.
in	*N*	int. N specifies the number of columns of the matrix A. N >= 0.
in,out	*A*	double *. A is a pointer to a regular matrix of dimension M-by-N. On exit, if return value is Success, the matrix A is overwriten by the factors L and U. The unit diagonal elements of L are not stored.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).
out	*IPIV*	int *. ipiv is a pointer to an array of dimesion at least max(1, min (M, N)). ipiv(i) = j, 1 <= i <= min(M, N) implies that rows i and j have been interchanged.

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

kdtpgetrf

Definition at line 77 of file ddss_dtpgetrf.c.

References kdtpgetrf().

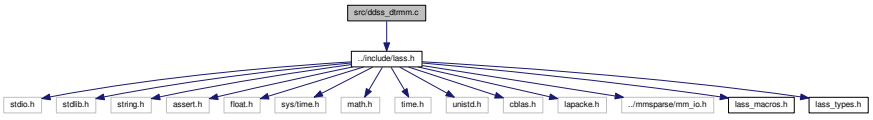
```
78 {
79
80     // Argument checking
81     if ( M < 0 )
82     {
83         fprintf( stderr, "Illegal value of M, in ddss_dtpgetrf code\n" );
84         return NoSuccess;
85     }
86
87     if ( N < 0 )
88     {
89         fprintf( stderr, "Illegal value of N, in ddss_dtpgetrf code\n" );
90         return NoSuccess;
91     }
92
93     if ( LDA < MAX( 1, N ) )
94     {
95         fprintf( stderr, "Illegal value of LDA, in ddss_dtpgetrf code\n" );
96         return NoSuccess;
97     }
98
99     // Qick return
100     if ( MIN( M, N ) == 0 )
101     {
102         return Success;
103     }
104
105     return kdtpgetrf( M, N, A, LDA, IPIV );
106
107 }
```

2.13 src/ddss_dtrmm.c File Reference

LASs-DDSs ddss_dtrmm routine.

```
#include "../include/lass.h"
```

Include dependency graph for ddss_dtrmm.c:



Functions

- int ddss_dtrmm (enum DDSS_SIDE SIDE, enum DDSS_UPLO UPLO, enum DDSS_TRANS TRANS_A, enum DDSS_DIAG DIAG, int M, int N, const double ALPHA, double *A, int LDA, double *B, int LDB)

2.13.1 Detailed Description

LASs-DDSs ddss_dtrmm routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es
Boro Sofranac boro.sofranac@bsc.es

Date

2018-03-19

2.13.2 Function Documentation

2.13.2.1 int ddss_dtrmm (enum DDSS_SIDE *SIDE*, enum DDSS_UPLO *UPLO*, enum DDSS_TRANS *TRANS_A*, enum DDSS_DIAG *DIAG*, int *M*, int *N*, const double *ALPHA*, double * *A*, int *LDA*, double * *B*, int *LDB*)

Performs one of the matrix-matrix operations:

```
B = ALPHA * op( A ) * B, or B = ALPHA * B * op( A )
```

where op(A) is one of:

```
op( A ) = A      or
op( A ) = A**T
```

ALPHA is a scalar, B is a M by N matrix and A is a unit, or non-unit, upper or lower triangular matrix.

Parameters

in	*SIDE*	enum DDSS_SIDE. SIDE specifies the position of the triangular A matrix in the operations: • Left: B = ALPHA * op(A) * B. • Right: B = ALPHA * B * op(A).
in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form in which A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.
in	*TRANS_A*	enum DDSS_TRANS. TRANS_A specifies the form of op(A) to be used: • NoTrans: op(A) = A. • Trans: op(A) = A**T.

Parameters

in	*DIAG*	enum DDSS_DIAG. DIAG specifies whether or not A is unit triangular as follows: • Unit: A is assumed to be unit triangular. • NonUnit: A is not assumed to be unit triangular.
in	*M*	int. M specifies the number of rows of B. M must be at least zero.
in	*N*	int. N specifies the number of columns of B. N must be at least zero.
in	*ALPHA*	double. ALPHA specifies the scalar alpha.
in	*A*	double *. A is a pointer to a matrix of dimension K by K, where K is M when SIDE = Left and is N otherwise. When UPLO = Uppper the strictly lower triangular part of A is not referenced and when UPLO = Lower the strictly upper triangular part of A is not referenced. Note that when DIAG = Unit, the diagonal elements of A are not referenced either, but are assumed to be unity.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). When SIDE = Left then LDA must be at least max(1, M), otherwise LDA must be at least max(1, N).
in,out	*B*	double *. B is a pointer to a matrix of dimension M by N. On exit the matrix B is overwritten by the transformed matrix.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, N).

See also

kdtrmm

Definition at line 113 of file ddss_dtrmm.c.

References kdtrmm().

```
118 {
119
120     // Local variables
121     int nA;
122
123     // Argument checking
124     if ( ( SIDE != Left ) && ( SIDE != Right ) )
125     {
126         fprintf( stderr, "Illegal value of SIDE, in ddss_dtrmm code\n" );
127         return NoSuccess;
128     }
129
130     if ( ( UPLO != Upper ) && ( UPLO != Lower ) )
131     {
132         fprintf( stderr, "Illegal value of UPLO, in ddss_dtrmm code\n" );
133         return NoSuccess;
134     }
135
136     if ( ( TRANS_A != NoTrans ) && ( TRANS_A != Trans ) )
137     {
138         fprintf( stderr, "Illegal value of TRANS_A, in ddss_dtrmm code\n" );
139         return NoSuccess;
140     }
141
142     if ( ( DIAG != Unit ) && ( DIAG != NonUnit ) )
143     {
144         fprintf( stderr, "Illegal value of DIAG, in ddss_dtrmm code\n" );
145         return NoSuccess;
146     }
147
148     if ( M < 0 )
149     {
150         fprintf( stderr, "Illegal value of M, in ddss_dtrmm code\n" );
151         return NoSuccess;
152     }
153
154     if ( N < 0 )
```

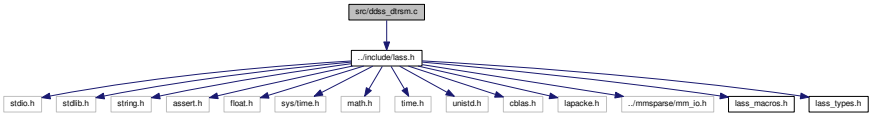
```
155     {
156         fprintf( stderr, "Illegal value of N, in ddss_dtrmm code\n" );
157         return NoSuccess;
158     }
159 
160     if ( SIDE == Left )
161     {
162         nA = M;
163     }
164     else
165     {
166         nA = N;
167     }
168 
169     if ( LDA < MAX( 1, nA ) )
170     {
171         fprintf( stderr, "Illegal value of LDA, in ddss_dtrmm code\n" );
172         return NoSuccess;
173     }
174 
175     if ( LDB < MAX( 1, N ) )
176     {
177         fprintf( stderr, "Illegal value of LDB, in ddss_dtrmm code\n" );
178         return NoSuccess;
179     }
180 
181     // Quick return
182     if ( M == 0 || N == 0 )
183     {
184         return Success;
185     }
186 
187     return kdtrmm( SIDE, UPLO,
188                    TRANS_A, DIAG,
189                    M, N,
190                    ALPHA, A, LDA,
191                           B, LDB );
192 
193 }
```

2.14 src/ddss_dtrsm.c File Reference

LASs-DDSs ddss_dtrsm routine.

```
#include "../include/lass.h"
```

Include dependency graph for ddss_dtrsm.c:



Functions

- int ddss_dtrsm (enum DDSS_SIDE SIDE, enum DDSS_UPLO UPLO, enum DDSS_TRANS TRANS_A, enum DDSS_DIAG DIAG, int M, int N, const double ALPHA, double *A, int LDA, double *B, int LDB)

2.14.1 Detailed Description

LASs-DDSs ddss_dtrsm routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es
Boro Sofranac boro.sofranac@bsc.es

Date

2017-12-14

2.14.2 Function Documentation

2.14.2.1 int ddss_dtrsm (enum DDSS_SIDE *SIDE*, enum DDSS_UPLO *UPLO*, enum DDSS_TRANS *TRANS_A*, enum DDSS_DIAG *DIAG*, int *M*, int *N*, const double *ALPHA*, double * *A*, int *LDA*, double * *B*, int *LDB*)

Solves one of the matrix equations:

```
op( A ) * X = ALPHA * B, or X * op( A ) = ALPHA * B
```

where op(A) is one of:

```
op( A ) = A      or
op( A ) = A**T
```

ALPHA is a scalar, X and B are M by N matrices, A is a unit, or non-unit, upper or lower triangular matrix. The matrix X is overwritten on B

Parameters

in	*SIDE*	enum DDSS_SIDE. SIDE specifies the position of the triangular A matrix in the operations: • Left: op(A) * X = ALPHA * B. • Right: X * op(A) = ALPHA * B.
in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form of A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.
in	*TRANS_A*	enum DDSS_TRANS. TRANS_A specifies the form of op(A) to be used: • NoTrans: op(A) = A. • Trans: op(A) = A**T.
in	*DIAG*	enum DDSS_DIAG. DIAG specifies whether or not A is unit triangular as follows: • Unit: A is assumed to be unit triangular. • NonUnit: A is not assumed to be unit triangular.
in	*M*	int. M specifies the number of rows of B. M must be at least zero.
in	*N*	int. N specifies the number of columns of B. N must be at least zero.
in	*ALPHA*	double. ALPHA specifies the scalar alpha.

Parameters

in	*A*	double *. A is a pointer to a matrix of dimension K by K, where K is M when SIDE = Left and is N otherwise. When UPLO = Uppper the strictly lower triangular part of A is not referenced and when UPLO = Lower the strictly upper triangular part of A is not referenced. Note that when DIAG = Unit, the diagonal elements of A are not referenced either, but are assumed to be unity.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). When SIDE = Left then LDA must be at least max(1, M), otherwise LDA must be at least max(1, N).
in,out	*B*	double *. B is a pointer to a matrix of dimension M by N. On exit the matrix B is overwritten by the solution matrix X.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, N).

See also

kdtrsm

Definition at line 113 of file ddss_dtrsm.c.

References kdtrsm().

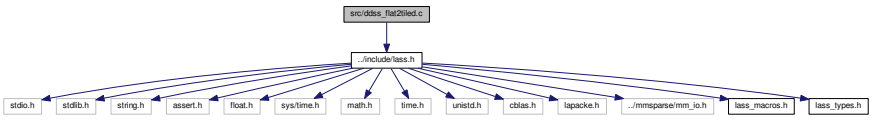
```
118 {
119
120     // Local variables
121     int nA;
122
123     // Argument checking
124     if ( ( SIDE != Left ) && ( SIDE != Right ) )
125     {
126         fprintf( stderr, "Illegal value of SIDE, in ddss_dtrsm code\n" );
127         return NoSuccess;
128     }
129
130     if ( ( UPLO != Upper ) && ( UPLO != Lower ) )
131     {
132         fprintf( stderr, "Illegal value of UPLO, in ddss_dtrsm code\n" );
133         return NoSuccess;
134     }
135
136     if ( ( TRANS_A != NoTrans ) && ( TRANS_A != Trans ) )
137     {
138         fprintf( stderr, "Illegal value of TRANS_A, in ddss_dtrsm code\n" );
139         return NoSuccess;
140     }
141
142     if ( ( DIAG != Unit ) && ( DIAG != NonUnit ) )
143     {
144         fprintf( stderr, "Illegal value of DIAG, in ddss_dtrsm code\n" );
145         return NoSuccess;
146     }
147
148     if ( M < 0 )
149     {
150         fprintf( stderr, "Illegal value of M, in ddss_dtrsm code\n" );
151         return NoSuccess;
152     }
153
154     if ( N < 0 )
155     {
156         fprintf( stderr, "Illegal value of N, in ddss_dtrsm code\n" );
157         return NoSuccess;
158     }
159
160     if ( SIDE == Left )
161     {
162         nA = M;
163     }
164     else
165     {
166         nA = N;
167     }
```

```
168
169     if ( LDA < MAX( 1, nA ) )
170     {
171         fprintf( stderr, "Illegal value of LDA, in ddss_dtrsm code\n" );
172         return NoSuccess;
173     }
174
175     if ( LDB < MAX( 1, N ) )
176     {
177         fprintf( stderr, "Illegal value of LDB, in ddss_dtrsm code\n" );
178         return NoSuccess;
179     }
180
181     // Quick return
182     if ( M == 0 || N == 0 )
183     {
184         return Success;
185     }
186
187     return kdtrsm( SIDE, UPLO,
188                    TRANS_A, DIAG,
189                    M, N,
190                    ALPHA, A, LDA,
191                           B, LDB );
192
193 }
```

2.15 src/ddss_flat2tiled.c File Reference

LASs-DDSs ddss_flat2tiled routines.

`#include "../include/lass.h"`
Include dependency graph for ddss_flat2tiled.c:



Functions

- void ddss_dflat2tiled (int M, int N, double ∗A, int LDA, int MT, int NT, double(∗TILE_A)[NT][TILE_SIZE ∗TI↩ LE_SIZE])
- void ddss_dsymflat2tiled (int M, int N, double ∗A, int LDA, int MT, int NT, double(∗TILE_A)[NT][TILE_SIZE ∗TILE_SIZE], enum DDSS_UPLO UPLO)
- void ddss_dgather_tile (int M, int N, double ∗A, int LDA, double ∗TILE_A, int MID, int NID)

2.15.1 Detailed Description

LASs-DDSs ddss_flat2tiled routines.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es

Date

2017-05-11

2.15.2 Function Documentation

2.15.2.1 void ddss_dflat2tiled (int *M*, int *N*, double * *A*, int *LDA*, int *MT*, int *NT*, double(*) *TILE_A[NT][TILE_SIZE*TILE_SIZE]*)

ddss_dflat2tiled: Performs the change of the data layout from flat layout to tiled layout according to row-major order.

Parameters

in	*M*	int. M specifies the number of rows of the flat matrix.
in	*N*	int. N specifies the number of columns of the flat matrix.
in	*A*	double *. A is a pointer to the flat matrix.
in	*LDA*	int. LDA specifies the number of columns (row-major order) of matrix A.
in	*MT*	int. MT specifies the number of rows of the matrix TILE_A.
in	*NT*	int. NT specifies the number of columns of the matrix TILE_A.
in,out	*TILE_A*	double *. TILE_A is a pointer to the tile matrix.

See also

ddss_dgather_tile
ddss_tile_size

Definition at line 68 of file ddss_flat2tiled.c.

References ddss_dgather_tile().

Referenced by kdgemm(), kdnpgesv(), kdnpgetrf(), kdposv(), kdsymm(), kdsyr2k(), kdsyrk(), kdtpgesv(), kdtpgetrf(), kdtrmm(), and kdtrsm().

```
70 {
71
72     // Local variables
73     int m, n;
74
75     for ( m = 0; m < MT; m++ )
76     {
77         for ( n = 0; n < NT; n++ )
78         {
79             #pragma oss task inout(TILE_A[m][n]) \
80             label( dflat2tiled )
81             {
82                 ddss_dgather_tile( M, N, &A[m * TILE_SIZE * N + n * TILE_SIZE],
83                     LDA, TILE_A[m][n], m, n );
84             }
85         }
86     }
87
88 }
```

2.15.2.2 void ddss_dgather_tile (int *M*, int *N*, double * *A*, int *LDA*, double * *TILE_A*, int *MID*, int *NID*)

ddss_dgather_tile: Performs the copy of a tile from the flat matrix A to the tile matrix TILE_A for the MT, NT tile.

Parameters

in	*M*	int. M specifies the number of rows of the flat matrix.

Parameters

in	*N*	int. N specifies the number of columns of the flat matrix.
in	*A*	double *. A is a pointer to the flat matrix.
in	*LDA*	int. LDA specifies the number of columns (row-major order) of matrix A.
in,out	*TILE_A*	double *. TILE_A is a pointer to the tile matrix.
in	*MID*	int. MID specifies the row id of the tile.
in	*NID*	int. NID specifies the column id of the tile.

See also

ddss_tile_size
ddss_dflat2tiled

Definition at line 238 of file ddss_flat2tiled.c.

References ddss_tile_size().

Referenced by ddss_dflat2tiled(), and ddss_dsymflat2tiled().

```
240 {
241
242     //Local variables
243     int i, j;
244     int tile_size_m, tile_size_n;
245
246     tile_size_m = ddss_tile_size( M, MID );
247     tile_size_n = ddss_tile_size( N, NID );
248
249     for ( i = 0; i < tile_size_m; i++ )
250     {
251         for ( j = 0; j < tile_size_n; j++ )
252         {
253             TILE_A[i * tile_size_n + j] = A[i * LDA + j];
254         }
255     }
256
257 }
```

2.15.2.3 void ddss_dsymflat2tiled (int *M*, int *N*, double * *A*, int *LDA*, int *MT*, int *NT*, double(*) *TILE_A[NT][TILE_SIZE *TILE_SIZE]*, enum DDSS_UPLO *UPLO*)

ddss_dsymflat2tiled: Performs the change of the data layout from flat layout to tiled layout for symmetric matrices according to row-major order.

Parameters

in	*M*	int. M specifies the number of rows of the flat matrix.
in	*N*	int. N specifies the number of columns of the flat matrix.
in	*A*	double *. A is a pointer to the flat matrix.
in	*LDA*	int. LDA specifies the number of columns (row-major order) of matrix A.
in	*MT*	int. MT specifies the number of rows of the matrix TILE_A.
in	*NT*	int. NT specifies the number of columns of the matrix TILE_A.
in,out	*TILE_A*	double *. TILE_A is a pointer to the tile matrix.
in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form of A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.

See also

ddss_dgather_tile
ddss_tile_size

Definition at line 147 of file ddss_flat2tiled.c.

References ddss_dgather_tile().

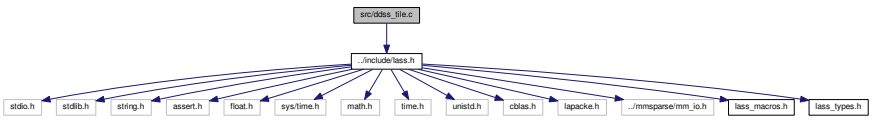
Referenced by kdposv(), kdpotrf(), kdsymm(), kdsyr2k(), kdsyrk(), kdtrmm(), and kdtrsm().

```
149 {
150
151     // Local variables
152     int m, n;
153
154     if ( UPLO == Upper )
155     {
156         for ( m = 0; m < MT; m++ )
157         {
158             for ( n = NT-1; n >= m; n-- )
159             {
160                 #pragma oss task inout(TILE_A[m][n]) \
161                 label( dsymmflat2tiled )
162                 {
163                     ddss_dgather_tile( M, N,
164                         &A[m * TILE_SIZE * N + n * TILE_SIZE],
165                         LDA, TILE_A[m][n], m, n );
166                 }
167             }
168         }
169     }
170     else if ( UPLO == Lower )
171     {
172         for ( m = 0; m < MT; m++ )
173         {
174             for ( n = 0; n <= m; n++ )
175             {
176                 #pragma oss task inout(TILE_A[m][n]) \
177                 label( dsymmflat2tiled )
178                 {
179                     ddss_dgather_tile( M, N,
180                         &A[m * TILE_SIZE * N + n * TILE_SIZE],
181                         LDA, TILE_A[m][n], m, n );
182                 }
183             }
184         }
185     }
186
187 }
```

2.16 src/ddss_tile.c File Reference

LASs-DDSs ddss_tile routines.

`#include "../include/lass.h"`
Include dependency graph for ddss_tile.c:



Functions

- int ddss_tile_size (int M, int MT)

2.16.1 Detailed Description

LASs-DDSs ddss_tile routines.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es

Date

2017-05-10

2.16.2 Function Documentation

2.16.2.1 int ddss_tile_size (int *M*, int *MT*)

tile_size: Computes the size of the tile passed as parameter.

Parameters

in	*M*	int. M specifies the size (rows of columns) of the matrix.
in	*MT*	int. MT specifies the id of the tile.

Return values

int	size of the tile passed as parameter.

See also

ddss_gather_tile

Definition at line 52 of file ddss_tile.c.

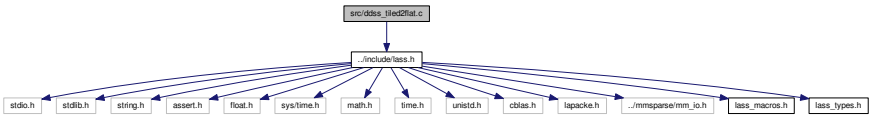
Referenced by ddss_dgather_tile(), ddss_dscatter_tile(), kdgemm(), kdnpgesv(), kdnpgetrf(), kdposv(), kdpotrf(), kdsymm(), kdsyr2k(), kdsyrk(), kdtpgesv(), kdtpgetrf(), kdtrmm(), and kdtrsm().

```
53 {
54
55     if ( M - ( MT * TILE_SIZE ) > TILE_SIZE )
56         return TILE_SIZE;
57     else
58         return M - ( MT * TILE_SIZE );
59
60 }
```

2.17 src/ddss_tiled2flat.c File Reference

LASs-DDSs ddss_tiled2flat routines.

```
#include "../include/lass.h"
```
Include dependency graph for ddss_tiled2flat.c:



Functions

- void ddss_dtiled2flat (int M, int N, double ∗A, int LDA, int MT, int NT, double(∗TILE_A)[NT][TILE_SIZE ∗TI↩ LE_SIZE])
- void ddss_dtiled2flat_nb (int M, int N, double ∗A, int LDA, int MT, int NT, double(∗TILE_A)[NT][TILE_SIZE ∗TILE_SIZE])
- void ddss_dsymtiled2flat (int M, int N, double ∗A, int LDA, int MT, int NT, double(∗TILE_A)[NT][TILE_SIZE ∗TILE_SIZE], enum DDSS_UPLO UPLO)
- void ddss_dsymtiled2flat_nb (int M, int N, double ∗A, int LDA, int MT, int NT, double(∗TILE_A)[NT][TILE_SIZE ∗TILE_SIZE], enum DDSS_UPLO UPLO)
- void ddss_dscatter_tile (int M, int N, double ∗A, int LDA, double ∗TILE_A, int MID, int NID)

2.17.1 Detailed Description

LASs-DDSs ddss_tiled2flat routines.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es

Date

2017-05-10

2.17.2 Function Documentation

2.17.2.1 void ddss_dscatter_tile (int *M,* int *N,* double ∗ *A,* int *LDA,* double ∗ *TILE_A,* int *MID,* int *NID*)

ddss_dscatter_tile: Performs the copy of a tile from the tile matrix TILE_A to the flat matrix A for the MT, NT tile.

Parameters

in	*M*	int. M specifies the number of rows of the flat matrix.
in	*N*	int. N specifies the number of columns of the flat matrix.
in,out	*A*	double *. A is a pointer to the flat matrix.
in	*LDA*	int. LDA specifies the number of columns (row-major order) of matrix A.
in	*TILE_A*	double *. TILE_A is a pointer to the tile matrix.
in	*MID*	int. MID specifies the row id of the tile.
in	*NID*	int. NID specifies the column id of the tile.

See also

ddss_tile_size
ddss_dtiled2flat

Definition at line 414 of file ddss_tiled2flat.c.

References ddss_tile_size().

Referenced by ddss_dsymtiled2flat(), ddss_dsymtiled2flat_nb(), ddss_dtiled2flat(), and ddss_dtiled2flat_nb().

```
416 {
417
418     //Local variables
419     int i, j;
420     int tile_size_m, tile_size_n;
421
422     tile_size_m = ddss_tile_size( M, MID );
423     tile_size_n = ddss_tile_size( N, NID );
424
425     for ( i = 0; i < tile_size_m; i++ )
426     {
427         for ( j = 0; j < tile_size_n; j++ )
428         {
429             A[i * LDA + j] = TILE_A[i * tile_size_n + j];
430         }
431     }
432
433 }
```

2.17.2.2 void ddss_dsymtiled2flat (int *M*, int *N*, double * *A*, int *LDA*, int *MT*, int *NT*, double(*) *TILE_A[NT][TILE_SIZE *TILE_SIZE]*, enum DDSS_UPLO *UPLO*)

ddss_dsymtiled2flat: Performs the change of the data layout from tile layout to flat layout for symmetric matrices according to row-major order.

Parameters

in	*M*	int. M specifies the number of rows of the flat matrix.
in	*N*	int. N specifies the number of columns of the flat matrix.
in,out	*A*	double *. A is a pointer to the flat matrix.
in	*LDA*	int. LDA specifies the number of columns (row-major order) of matrix A.
in	*MT*	int. MT specifies the number of rows of the matrix TILE_A.
in	*NT*	int. NT specifies the number of columns of the matrix TILE_A.
in	*TILE_A*	double *. TILE_A is a pointer to the tile matrix.
in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form of A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.

See also

ddss_dscatter_tile
ddss_tile_size

Definition at line 223 of file ddss_tiled2flat.c.

References ddss_dscatter_tile().

Referenced by kdpotrf(), kdsyr2k(), and kdsyrk().

```
225 {
226
227     // Local variables
228     int m, n;
229
230     if ( UPLO == Upper )
231     {
232         for ( m = 0; m < MT; m++ )
233         {
234             for ( n = NT-1; n >= m; n-- )
235             {
236                 #pragma oss task inout(TILE_A[m][n]) \
237                 label( dsymmtiled2flat )
238                 {
239                     ddss_dscatter_tile( M, N,
240                         &A[m * TILE_SIZE * N + n * TILE_SIZE], LDA,
241                         TILE_A[m][n], m, n );
242                 }
243             }
244         }
245     }
246     else if ( UPLO == Lower )
247     {
248         for ( m = 0; m < MT; m++ )
249         {
250             for ( n = 0; n <= m; n++ )
251             {
252                 #pragma oss task inout(TILE_A[m][n]) \
253                 label( dsymmtiled2flat )
254                 {
255                     ddss_dscatter_tile( M, N,
256                         &A[m * TILE_SIZE * N + n * TILE_SIZE], LDA,
257                         TILE_A[m][n], m, n );
258                 }
259             }
260         }
261     }
262
263     #pragma oss taskwait
264 }
```

2.17.2.3 void ddss_dsymtiled2flat_nb (int *M*, int *N*, double * *A*, int *LDA*, int *MT*, int *NT*, double(*) *TILE_A[NT][TILE_SIZE *TILE_SIZE]*, enum DDSS_UPLO *UPLO*)

ddss_dsymtiled2flat_nb: Performs the change of the data layout from tile layout to flat layout for symmetric matrices according to row-major order in a non-blocking execution mode.

Parameters

in	*M*	int. M specifies the number of rows of the flat matrix.
in	*N*	int. N specifies the number of columns of the flat matrix.
in,out	*A*	double *. A is a pointer to the flat matrix.
in	*LDA*	int. LDA specifies the number of columns (row-major order) of matrix A.
in	*MT*	int. MT specifies the number of rows of the matrix TILE_A.
in	*NT*	int. NT specifies the number of columns of the matrix TILE_A.
in	*TILE_A*	double *. TILE_A is a pointer to the tile matrix.

Parameters

in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form of A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.

See also

ddss_dscatter_tile
ddss_tile_size

Definition at line 324 of file ddss_tiled2flat.c.

References ddss_dscatter_tile().

Referenced by kdposv().

```
326 {
327
328     // Local variables
329     int m, n;
330
331     if ( UPLO == Upper )
332     {
333         for ( m = 0; m < MT; m++ )
334         {
335             for ( n = NT-1; n >= m; n-- )
336             {
337                 #pragma oss task inout(TILE_A[m][n]) \
338                 label( dsymmtiled2flat_nb )
339                 {
340                     ddss_dscatter_tile( M, N,
341                         &A[m * TILE_SIZE * N + n * TILE_SIZE], LDA,
342                         TILE_A[m][n], m, n );
343                 }
344             }
345         }
346     }
347     else if ( UPLO == Lower )
348     {
349         for ( m = 0; m < MT; m++ )
350         {
351             for ( n = 0; n <= m; n++ )
352             {
353                 #pragma oss task inout(TILE_A[m][n]) \
354                 label( dsymmtiled2flat_nb )
355                 {
356                     ddss_dscatter_tile( M, N,
357                         &A[m * TILE_SIZE * N + n * TILE_SIZE], LDA,
358                         TILE_A[m][n], m, n );
359                 }
360             }
361         }
362     }
363
364 }
```

2.17.2.4 void ddss_dtiled2flat (int *M*, int *N*, double * *A*, int *LDA*, int *MT*, int *NT*, double(*) *TILE_A[NT][TILE_SIZE*TILE_SIZE]*)

ddss_dtiled2flat: Performs the change of the data layout from tile layout to flat layout according to row-major order.

Parameters

in	*M*	int. M specifies the number of rows of the flat matrix.
in	*N*	int. N specifies the number of columns of the flat matrix.
in,out	*A*	double *. A is a pointer to the flat matrix.
in	*LDA*	int. LDA specifies the number of columns (row-major order) of matrix A.
in	*MT*	int. MT specifies the number of rows of the matrix TILE_A.
in	*NT*	int. NT specifies the number of columns of the matrix TILE_A.
in	*TILE_A*	double *. TILE_A is a pointer to the tile matrix.

See also

ddss_dscatter_tile
ddss_tile_size

Definition at line 68 of file ddss_tiled2flat.c.

References ddss_dscatter_tile().

Referenced by kdgemm(), kdnpgesv(), kdnpgetrf(), kdposv(), kdsymm(), kdtpgesv(), kdtpgetrf(), kdtrmm(), and kdtrsm().

```
70 {
71
72     // Local variables
73     int m, n;
74
75     for ( m = 0; m < MT; m++ )
76     {
77         for ( n = 0; n < NT; n++ )
78         {
79             #pragma oss task inout(TILE_A[m][n]) \
80             label( dtiled2flat)
81             {
82                 ddss_dscatter_tile( M, N,
83                     &A[m * TILE_SIZE * N + n * TILE_SIZE], LDA,
84                     TILE_A[m][n], m, n );
85             }
86         }
87     }
88
89     #pragma oss taskwait
90
91 }
```

2.17.2.5 void ddss_dtiled2flat_nb (int *M*, int *N*, double * *A*, int *LDA*, int *MT*, int *NT*, double(*) *TILE_A[NT][TILE_SIZE *TILE_SIZE]*)

ddss_dtiled2flat_nb: Performs the change of the data layout from tile layout to flat layout according to row-major order in a non-blocking execution mode.

Parameters

in	*M*	int. M specifies the number of rows of the flat matrix.
in	*N*	int. N specifies the number of columns of the flat matrix.
in,out	*A*	double *. A is a pointer to the flat matrix.
in	*LDA*	int. LDA specifies the number of columns (row-major order) of matrix A.
in	*MT*	int. MT specifies the number of rows of the matrix TILE_A.
in	*NT*	int. NT specifies the number of columns of the matrix TILE_A.
in	*TILE_A*	double *. TILE_A is a pointer to the tile matrix.

See also

ddss_dscatter_tile
ddss_tile_size

Definition at line 142 of file ddss_tiled2flat.c.

References ddss_dscatter_tile().

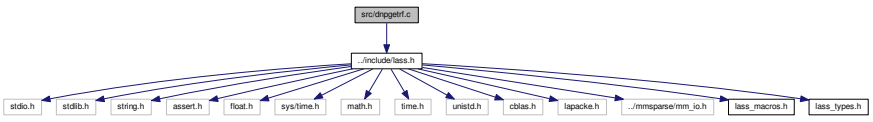
Referenced by kdnpgesv().

```
144 {
145
146     // Local variables
147     int m, n;
148
149     for ( m = 0; m < MT; m++ )
150     {
151         for ( n = 0; n < NT; n++ )
152         {
153             #pragma oss task inout(TILE_A[m][n]) \
154             label( dtiled2flat)
155             {
156                 ddss_dscatter_tile( M, N,
157                     &A[m * TILE_SIZE * N + n * TILE_SIZE], LDA,
158                     TILE_A[m][n], m, n );
159             }
160         }
161     }
162
163 }
```

2.18 src/dnpgetrf.c File Reference

LASs-DDSs dnpgetrf routine.

`#include "../include/lass.h"`
Include dependency graph for dnpgetrf.c:



Functions

- enum LASS_RETURN dnpgetrf (int M, int N, double *A, int LDA)

2.18.1 Detailed Description

LASs-DDSs dnpgetrf routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es
Boro Sofranac boro.sofranac@bsc.es

Date

2017-21-11

2.18.2 Function Documentation

2.18.2.1 enum LASS_RETURN dnpgetrf (int *M*, int *N*, double * *A*, int *LDA*)

Performs the LU factorization without pivoting of a general M-by-N matrix A:

```
A = L * U
```

where L is a lower triangular (lower trapezoidal if M > N) matrix with unit diagonal elements and U is an upper triangular (upper trapezoidal if M < N) matrix.

Parameters

in	*M*	int. M specifies the number of rows of the matrix A. M >= 0.
in	*N*	int. N specifies the number of columns of the matrix A. N >= 0.
in,out	*A*	double *. A is a pointer to a regular matrix of dimension M-by-N. On exit, if return value is Success, the matrix A is overwriten by the factors L and U. The unit diagonal elements of L are not stored.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

kdnpgetrf

Definition at line 70 of file dnpgetrf.c.

Referenced by kdnpgesv(), and kdnpgetrf().

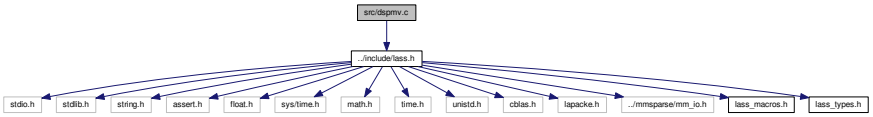
```
71 {
72
73     // Local variable
74     double alpha;
75     double sfmin;
76     int i, j, k;
77     int info = 0;
78
79     // Argument checking
80     if ( M < 0 )
81     {
82         fprintf( stderr, "Illegal value of M, in dnpgetrf code\n" );
83         return NoSuccess;
84     }
85
86     if ( N < 0 )
87     {
88         fprintf( stderr, "Illegal value of N, in dnpgetrf code\n" );
89         return NoSuccess;
90     }
91
92     if ( LDA < MAX( 1, N ) )
93     {
94         fprintf( stderr, "Illegal value of LDA, in dnpgetrf code\n" );
95         return NoSuccess;
96     }
97
```

```
98      // Qick return
99      if ( MIN( M, N ) == 0 )
100      {
101          return Success;
102      }
103
104      /****************
105      --DNPGETRF tile--
106      ****************/
107
108      // Minimum value in double precision
109      sfmin = LAPACKE_dlamch_work( 'S' );
110      k = MIN( M, N );
111
112      for ( i = 0; i < k; i++ )
113      {
114          alpha = A[i * LDA + i];
115          if ( alpha != ( double )0.0 )
116          {
117              // Compute elements from J+1 to M of the J-th column
118              if ( i < M )
119              {
120                  if ( fabs( alpha ) > fabs( sfmin ) )
121                  {
122                      alpha = 1.0 / alpha;
123                      cblas_dscal( M - i - 1, alpha,
124                          &( A[( i + 1 ) * LDA + i] ), LDA );
125                  }
126                  else
127                  {
128                      for( j= i + 1; j < M; j++ )
129                      {
130                          A[j * LDA + i] = A[j * LDA + i] / alpha;
131                      }
132                  }
133              }
134          }
135          else if ( info == 0 )
136          {
137              info = i;
138          }
139          if ( i < k )
140          {
141              cblas_dger( CblasRowMajor,
142                          M - i - 1, N - i - 1,
143                          -1.0,
144                          &A[( i + 1 ) * LDA + i], LDA,
145                          &A[ i * LDA + ( i + 1 )], 1,
146                          &A[( i + 1 ) * LDA + ( i + 1 ) ], LDA );
147          }
148      }
149
150      return Success;
151
152  }
```

2.19 src/dspmv.c File Reference

LASs-DSS dspmv routine.

`#include "../include/lass.h"`
Include dependency graph for dspmv.c:



Functions

- void dspmvseq (int M, int N, double ALPHA, const double *VAL_A, const int *ROW_PTR_A, const int *C↩ OL_IND_A, const double *X, double BETA, double *Y)

2.19.1 Detailed Description

LASs-DSS dspmv routine.

LASs-DSSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Sandra Catalán sandra.catalan@bsc.es
Pedro Valero-Lara pedro.valero@bsc.es

Date

2018-09-07

2.19.2 Function Documentation

2.19.2.1 void dspmvseq (int *M*, int *N*, double *ALPHA*, const double * *VAL_A*, const int * *ROW_PTR_A*, const int * *COL_IND_A*, const double * *X*, double *BETA*, double * *Y*)

Performs the sparse matrix-vector operation:

```
Y = ALPHA * A * X + BETA * Y
```

where ALPHA and BETA are scalars, X and Y are vectors, and A is a sparse matrix stored in thhe next three vectors VAL_A, COL_IND_A and ROW_PTR_A according to CSR format.

Parameters

in	*M*	int. M specifies the number of rows of A.
in	*N*	int. N specifies the number of columns of A.
in	*ALPHA*	double.
in	*VAL_A*	double *. VAL_A is a pointer to a vector which specifies the non-zero values of the original matrix A.
in	*COL_IND_A*	unsigned int *. COL_IND_A is a pointer to a vector which specifies the column of each value of A in the original matrix.
in	*ROW_PTR_A*	unsigned int *. ROW_PTR_A is a pointer to a vector which specifies the number of values that A contains in each row of the original matrix.
in	*X*	double *. X is a pointer to a vector of dimension N.
in	*BETA*	double .
in,out	*Y*	double *. Y is a pointer to a vector of dimension M. On exit, Y is overwritten by the sparse matrix-vector result.

Return values

Success	sucessful exit
NoSuccess	unsucessful exit

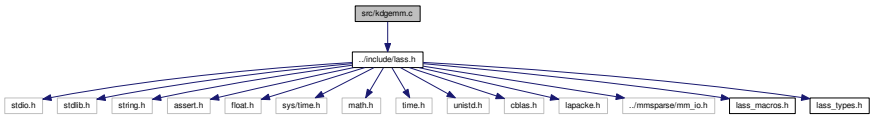
Definition at line 84 of file dspmv.c.

```
90 {
91
92     // Local variables
93     double svalue = 0.0, value = 0.0;
94     int max = 0, col = 0;
95     int i, x;
96
97     for ( x = 0; x < M; x++ )
98     {
99         svalue = 0.0;
100         max = ROW_PTR_A[x + 1] - ROW_PTR_A[x];
101
102         for( i = 0; i < max; i++ )
103         {
104             value = VAL_A[ROW_PTR_A[x] + i];
105             col = COL_IND_A[ROW_PTR_A[x] + i];
106             svalue += value * X[col] * ALPHA;
107
108
109         }
110
111         Y[x] = svalue + Y[x] * BETA;
112
113     }
114
115 }
```

2.20 src/kdgemm.c File Reference

LASs-DDSs kdgemm routine.

`#include "../include/lass.h"`
Include dependency graph for kdgemm.c:



Functions

- enum LASS_RETURN kdgemm (enum DDSS_TRANS TRANS_A, enum DDSS_TRANS TRANS_B, int M, int N, int K, const double ALPHA, double ∗A, int LDA, double ∗B, int LDB, const double BETA, double ∗C, int LDC)

2.20.1 Detailed Description

LASs-DDSs kdgemm routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es

Date

2017-01-05

2.20.2 Function Documentation

2.20.2.1 enum LASS_RETURN kdgemm (enum DDSS_TRANS *TRANS_A*, enum DDSS_TRANS *TRANS_B*, int *M*, int *N*, int *K*, const double *ALPHA*, double * *A*, int *LDA*, double * *B*, int *LDB*, const double *BETA*, double * *C*, int *LDC*)

Performs the matrix-matrix operation:

```
C = ALPHA * op( A ) * op( B ) + BETA * C
```

where op(X) is one of:

```
op( X ) = X      or
op( X ) = X**T
```

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

in	*TRANS_A*	enum DDSS_TRANS. TRANS_A specifies the form of op(A) to be used in the matrix multiplication as follows: • NoTrans: op(A) = A. • Trans: op(A) = A**T.
in	*TRANS_B*	enum DDSS_TRANS. TRANS_B specifies the form of op(B) to be used in the matrix multiplication as follows: • NoTrans: op(B) = B. • Trans: op(B) = B**T.
in	*M*	int. M specifies the number of rows of the matrix A and of the matrix C. M must be greater than zero.
in	*N*	int. N specifies the number of columns of the matrix B and the number of columns of the matrix C. N must be greater than zero.
in	*K*	int. K specifies the number of columns of the matrix A and the number of rows of the matrix B. K must be greater than zero.
in	*ALPHA*	double.
in	*A*	double *. A is a pointer to a matrix of dimension Ma (rows) by Ka (columns), where Ma is M and Ka is K when TRANS_A = NoTrans, and Ma is K and Ka is M otherwise.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). When TRANS_A = NoTrans then LDA must be at least max(1, K), otherwise LDA must be at least max(1, M).
in	*B*	double *. B is a pointer to a matrix of dimension Kb (rows) by Nb (columns), where Kb is K and Nb is N when TRANS_B = NoTrans, and Kb is N and Nb is K otherwise.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). When TRANS_B = NoTrans then LDB must be at least max(1, N), otherwise LDB must be at least max(1, K).
in	*BETA*	double.
in,out	*C*	double *. C is a pointer to a matrix of dimension LDC by N. On exit, C is overwritten by the M by N matrix (ALPHA*op(A)*op(B) + BETA*C).
in	*LDC*	int. LDC specifies the number of columns of C (row-major order). LDC must be at least max(1, M)

Return values

Success	sucessful exit
NoSuccess	unsucessful exit

See also

ddss_dgemm
ddss_tile
ddss_flat2tiled
ddss_tiled2flat

Definition at line 131 of file kdgemm.c.

References ddss_dflat2tiled(), ddss_dtiled2flat(), and ddss_tile_size().

Referenced by ddss_dgemm().

```
136 {
137
138     // Local variables
139     int mt, kt, nt;
140     int mi, ki, ni;
141     int Am, An;
142     int Bm, Bn;
143     int tile_size_m;
144     int tile_size_n;
145     int tile_size_k;
146     int k_check;
147     double betat;
148
149     // Number of tiles
150     if ( M % TILE_SIZE == 0 )
151     {
152         mt = M / TILE_SIZE;
153     }
154     else
155     {
156         mt = ( M / TILE_SIZE ) + 1;
157     }
158
159     if ( K % TILE_SIZE == 0 )
160     {
161         kt = K / TILE_SIZE;
162     }
163     else
164     {
165         kt = ( K / TILE_SIZE ) + 1;
166     }
167
168     if ( N % TILE_SIZE == 0 )
169     {
170         nt = N / TILE_SIZE;
171     }
172     else
173     {
174         nt = ( N / TILE_SIZE ) + 1;
175     }
176
177     /***************************
178     --Tile matrices declaration--
179     ***************************/
180
181     if ( TRANS_A == NoTrans )
182     {
183         Am = mt;
184         An = kt;
185     }
186     else
187     {
188         Am = kt;
189         An = mt;
190     }
191
192     if ( TRANS_B == NoTrans )
```

```
193     {
194         Bm = kt;
195         Bn = nt;
196     }
197     else
198     {
199         Bm = nt;
200         Bn = kt;
201     }
202 
203     /***************************
204     --Tile matrices allocation--
205     ***************************/
206 
207     double (*TILE_A)[An][TILE_SIZE * TILE_SIZE] = malloc ( Am * An *
208         TILE_SIZE * TILE_SIZE * sizeof( double ) );
209 
210     if ( TILE_A == NULL)
211     {
212         fprintf( stderr, "Failure in kdgemm for matrix TILE_A\n" );
213         return NoSuccess;
214     }
215 
216     double (*TILE_B)[Bn][TILE_SIZE * TILE_SIZE] = malloc ( Bm * Bn *
217         TILE_SIZE * TILE_SIZE * sizeof( double ) );
218 
219     if ( TILE_B == NULL)
220     {
221         fprintf( stderr, "Failure in kdgemm for matrix TILE_B\n" );
222         return NoSuccess;
223     }
224 
225     double (*TILE_C)[nt][TILE_SIZE * TILE_SIZE] = malloc ( mt * nt *
226         TILE_SIZE * TILE_SIZE * sizeof( double ) );
227 
228     if ( TILE_C == NULL)
229     {
230         fprintf( stderr, "Failure in kdgemm for matrix TILE_C\n" );
231         return NoSuccess;
232     }
233 
234     /*****************************************
235     --From flat data layout to tiled data layout--
236     *****************************************/
237 
238     // From flat matrix A to tile matrix TILE_A
239     if ( TRANS_A == NoTrans )
240     {
241         ddss_dflat2tiled( M, K, A, LDA, mt, kt, TILE_A );
242     }
243     else
244     {
245         ddss_dflat2tiled( K, M, A, LDA, kt, mt, TILE_A );
246     }
247 
248     // From flat matrix B to tile matrix TILE_B
249     if ( TRANS_B == NoTrans )
250     {
251         ddss_dflat2tiled( K, N, B, LDB, kt, nt, TILE_B );
252     }
253     else
254     {
255         ddss_dflat2tiled( N, K, B, LDB, nt, kt, TILE_B );
256     }
257 
258     // From flat matrix C to tile matrix TILE_C
259     ddss_dflat2tiled( M, N, C, LDC, mt, nt, TILE_C );
260 
261     /*************
262     --DGEMM tile--
263     *************/
264 
265     for ( mi = 0; mi < mt; mi++ )
266     {
267         tile_size_m = ddss_tile_size( M, mi );
268         for ( ni = 0; ni < nt; ni++ )
269         {
270             tile_size_n = ddss_tile_size( N, ni );
271             if ( TRANS_A == NoTrans )
272             {
273                 k_check = K;
274             }
275             else
276             {
277                 k_check = M;
278             }
279             // --Scale on C ( C = BETA * C )--
```

```
280             if ( ( ALPHA == 0.0 ) || ( k_check == 0 ) )
281             {
282                 #pragma oss task inout( TILE_C[mi][ni] ) \
283                     shared( TILE_A, TILE_B, TILE_C ) \
284                     firstprivate( mi, ni ) \
285                     no_copy_deps
286                 cblas_dgemm( CblasRowMajor,
287                              ( CBLAS_TRANSPOSE ) TRANS_A,
288                              ( CBLAS_TRANSPOSE ) TRANS_B,
289                              tile_size_m,
290                              tile_size_n,
291                                        0,
292                              ALPHA,   TILE_A[0][0], 1,
293                                       TILE_B[0][0], 1,
294                               BETA, TILE_C[mi][ni], tile_size_n );
295             }
296             else if ( TRANS_A == NoTrans )
297             {
298                 // --TRANS_A = NoTrans & TRANS_B = NoTrans--
299                 if ( TRANS_B == NoTrans )
300                 {
301                     for ( ki = 0; ki < kt; ki++ )
302                     {
303                         tile_size_k = ddss_tile_size( K, ki );
304                         if (ki == 0)
305                         {
306                             betat = BETA;
307                         }
308                         else
309                         {
310                             betat = 1.0;
311                         }
312
313                         #pragma oss task in( TILE_A[mi][ki] ) \
314                             in( TILE_B[ki][ni] ) \
315                             inout( TILE_C[mi][ni] ) \
316                             shared( TILE_A, TILE_B, TILE_C ) \
317                             firstprivate( mi, ni, ki, betat ) \
318                             no_copy_deps
319                         cblas_dgemm( CblasRowMajor,
320                                      ( CBLAS_TRANSPOSE ) TRANS_A,
321                                      ( CBLAS_TRANSPOSE ) TRANS_B,
322                                      tile_size_m,
323                                      tile_size_n,
324                                      tile_size_k,
325                                      ALPHA, TILE_A[mi][ki], tile_size_k,
326                                             TILE_B[ki][ni], tile_size_n,
327                                      betat, TILE_C[mi][ni], tile_size_n );
328                     }
329                 }
330                 // --TRANS_A = NoTrans & TRANS_B = Trans--
331                 else
332                 {
333                     for ( ki = 0; ki < kt; ki++ )
334                     {
335                         tile_size_k = ddss_tile_size( K, ki );
336                         if (ki == 0)
337                         {
338                             betat = BETA;
339                         }
340                         else
341                         {
342                             betat = 1.0;
343                         }
344
345                         #pragma oss task in( TILE_A[mi][ki] ) \
346                             in( TILE_B[ni][ki] ) \
347                             inout( TILE_C[mi][ni] ) \
348                             shared( TILE_A, TILE_B, TILE_C ) \
349                             firstprivate( mi, ni, ki, betat )
350                         cblas_dgemm( CblasRowMajor,
351                                      ( CBLAS_TRANSPOSE ) TRANS_A,
352                                      ( CBLAS_TRANSPOSE ) TRANS_B,
353                                      tile_size_m,
354                                      tile_size_n,
355                                      tile_size_k,
356                                      ALPHA,  TILE_A[mi][ki], tile_size_k,
357                                              TILE_B[ni][ki], tile_size_k,
358                                      betat,  TILE_C[mi][ni], tile_size_n );
359                     }
360                 }
361             }
362             else
363             {
364                 // --TRANS_A = Trans & TRANS_B = NoTrans--
365                 if ( TRANS_B == NoTrans )
366                 {
```

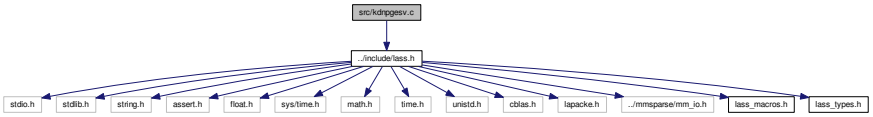
```
367                     for ( ki = 0; ki < kt; ki++ )
368                     {
369                         tile_size_k = ddss_tile_size( K, ki );
370                         if (ki == 0)
371                         {
372                             betat = BETA;
373                         }
374                         else
375                         {
376                             betat = 1.0;
377                         }
378 
379                         #pragma oss task in( TILE_A[ki][mi] ) \
380                             in( TILE_B[ki][ni] ) \
381                             inout( TILE_C[mi][ni] ) \
382                             shared( TILE_A, TILE_B, TILE_C ) \
383                             firstprivate( mi, ni, ki, betat )
384                         cblas_dgemm( CblasRowMajor,
385                                     ( CBLAS_TRANSPOSE ) TRANS_A,
386                                     ( CBLAS_TRANSPOSE ) TRANS_B,
387                                     tile_size_m,
388                                     tile_size_n,
389                                     tile_size_k,
390                                     ALPHA,  TILE_A[ki][mi], tile_size_m,
391                                             TILE_B[ki][ni], tile_size_n,
392                                     betat,  TILE_C[mi][ni], tile_size_n );
393                     }
394                 }
395                 // --TRANS_A = Trans & TRANS_B = Trans--
396                 else
397                 {
398                     for ( ki = 0; ki < kt; ki++ )
399                     {
400                         tile_size_k = ddss_tile_size( K, ki );
401                         if (ki == 0)
402                         {
403                             betat = BETA;
404                         }
405                         else
406                         {
407                             betat = 1.0;
408                         }
409 
410                         #pragma oss task in( TILE_A[ki][mi] ) \
411                             in( TILE_B[ni][ki] ) \
412                             inout( TILE_C[mi][ni] ) \
413                             shared( TILE_A, TILE_B, TILE_C ) \
414                             firstprivate( mi, ni, ki, betat )
415                         cblas_dgemm( CblasRowMajor,
416                                     ( CBLAS_TRANSPOSE ) TRANS_A,
417                                     ( CBLAS_TRANSPOSE ) TRANS_B,
418                                     tile_size_m,
419                                     tile_size_n,
420                                     tile_size_k,
421                                     ALPHA,  TILE_A[ki][mi], tile_size_m,
422                                             TILE_B[ni][ki], tile_size_k,
423                                     betat,  TILE_C[mi][ni], tile_size_n );
424                     }
425                 }
426             }
427         }
428     }
429 
430     /*********************************************
431     // --From tiled data layout to flat data layout--
432     *********************************************/
433 
434     // From tile matrix TILE_C to flat matrix C
435     ddss_dtiled2flat( M, N, C, LDC, mt, nt, TILE_C );
436 
437     // --Tile matrices free--
438     free( TILE_A );
439     free( TILE_B );
440     free( TILE_C );
441 
442     return Success;
443 
444 }
```

2.21 src/kdnpgesv.c File Reference

LASs-DDSs kdnpgesv routine.

```
#include "../include/lass.h"
```

Include dependency graph for kdnpgesv.c:



Functions

- enum LASS_RETURN kdnpgesv (int N, int NRHS, double ∗A, int LDA, double ∗B, int LDB)

2.21.1 Detailed Description

LASs-DDSs kdnpgesv routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es

Date

2018-07-26

2.21.2 Function Documentation

2.21.2.1 enum LASS_RETURN kdnpgesv (int *N*, int *NRHS*, double ∗ *A*, int *LDA*, double ∗ *B*, int *LDB*)

Solves a system of linear equations A X = B, where A is a N-by-N general matrix and X and B are N-by-NRHS matrices. The matrix A is factorized using the LU descomposition without pivoting. The matrix A is descomposed as:

```
A = L * U
```

where L is a lower triangular matrix with unit diagonal elements and U is an upper triangular matrix.

Parameters

in	*N*	int. N specifies the order of the square matrix A. N >= 0.

NRHS int. NRHS specifies the number of right-hand-sides (number of columns of B). NRHS >= 0.

Parameters

in,out	*A*	double ∗. A is a pointer to a regular matrix of dimension N-by-LDA. On exit, if return value is Success, the matrix A is overwriten by the factors L and U. The unit diagonal elements of L are not stored.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).
in,out	*B*	double ∗. B is a pointer to a matrix of dimension N by NRHS, which stores the right-hand-sides of the systems of linear equations. (row-major order). On exit, if return value is Success, the matrix B is overwriten by the solution matrix X.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, NRHS).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

dnpgesv
ddss_tile
ddss_flat2tiled
ddss_tiled2flat

Definition at line 86 of file kdnpgesv.c.

References ddss_dflat2tiled(), ddss_dtiled2flat(), ddss_dtiled2flat_nb(), ddss_tile_size(), and dnpgetrf().

Referenced by ddss_dnpgesv().

```
89 {
90
91     // Local variables
92     int nt, nrhst;
93     int mi, mmi, ki, ni, nni;
94     int tile_size_m;
95     int tile_size_mm;
96     int tile_size_n;
97     int tile_size_nn;
98     int tile_size_k;
99
100     // Number of tiles
101     if ( N % TILE_SIZE == 0 )
102     {
103         nt = N / TILE_SIZE;
104     }
105     else
106     {
107         nt = ( N / TILE_SIZE ) + 1;
108     }
109
110     if ( NRHS % TILE_SIZE == 0 )
111     {
112         nrhst = NRHS / TILE_SIZE;
113     }
114     else
115     {
116         nrhst = ( NRHS / TILE_SIZE ) + 1;
117     }
118
119     /*************************
120     --Tile matrices allocation--
121     *************************/
122
123     double (*TILE_A)[nt][TILE_SIZE * TILE_SIZE] = malloc ( nt * nt *
```

```
124         TILE_SIZE * TILE_SIZE * sizeof( double ) );
125
126     if ( TILE_A == NULL )
127     {
128         fprintf( stderr, "Failure in kdnpgesv for matrix TILE_A\n" );
129         return NoSuccess;
130     }
131
132     double ( *TILE_B )[nrhst][TILE_SIZE * TILE_SIZE] = malloc(
133             nt * nrhst * TILE_SIZE * TILE_SIZE * sizeof( double ) );
134
135     if ( TILE_B == NULL )
136     {
137         fprintf( stderr, "Failure in kdnpgesv for matrix TILE_B\n" );
138         return NoSuccess;
139     }
140
141     /*********************************************
142     // --From flat data layout to tiled data layout--
143     *********************************************/
144
145     // From flat matrix A to tile matrix TILE_A
146     ddss_dflat2tiled( N, N, A, LDA, nt, nt, TILE_A );
147
148     // From flat matrix B to tile matrix TILE_B
149     ddss_dflat2tiled( N, NRHS, B, LDB, nt, nrhst, TILE_B );
150
151     /****************
152     --DNPGESV tile--
153     ****************/
154
155     // LU descomposition without pivoting
156     for ( ki = 0; ki < nt; ki++ )
157     {
158         tile_size_k = ddss_tile_size( N, ki );
159
160         #if defined(LASs_WITH_MKL)
161
162         #pragma oss task inout( TILE_A[ki][ki] ) \
163                 shared( TILE_A ) \
164                 firstprivate( ki ) \
165                 label( mkl_dgetrfnpi )
166         LAPACKE_mkl_dgetrfnpi( CblasRowMajor,
167                                tile_size_k, tile_size_k,
168                                tile_size_k,
169                                TILE_A[ki][ki], tile_size_k );
170
171         #else
172
173         #pragma oss task inout( TILE_A[ki][ki] ) \
174                 shared( TILE_A ) \
175                 firstprivate( ki ) \
176                 label( dnpgetrf )
177         dnpgetrf( tile_size_k, tile_size_k,
178                   TILE_A[ki][ki],
179                   tile_size_k );
180
181         #endif
182
183         for ( mi = ki + 1; mi < nt; mi++ )
184         {
185             tile_size_m = ddss_tile_size( N, mi );
186
187             #pragma oss task in( TILE_A[ki][ki] ) \
188                     inout(TILE_A[mi][ki]) \
189                     shared( TILE_A ) \
190                     firstprivate( mi, ki ) \
191                     label( dnpgetrf_dtrsm_below )
192             cblas_dtrsm( CblasRowMajor,
193                          ( CBLAS_SIDE ) Right, ( CBLAS_UPLO ) Upper,
194                          ( CBLAS_TRANSPOSE ) NoTrans, ( CBLAS_DIAG ) NonUnit,
195                          tile_size_m, tile_size_k,
196                          1.0, TILE_A[ki][ki], tile_size_k,
197                               TILE_A[mi][ki], tile_size_k );
198         }
199         for ( ni = ki + 1; ni < nt; ni++ )
200         {
201             tile_size_n = ddss_tile_size( N, ni );
202
203             #pragma oss task in( TILE_A[ki][ki] ) \
204                     inout(TILE_A[ki][ni]) \
205                     shared( TILE_A ) \
206                     firstprivate( ni, ki ) \
207                     label( dnpgetrf_dtrsm_right )
208             cblas_dtrsm( CblasRowMajor,
209                          ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Lower,
210                          ( CBLAS_TRANSPOSE ) NoTrans, ( CBLAS_DIAG ) Unit,
```

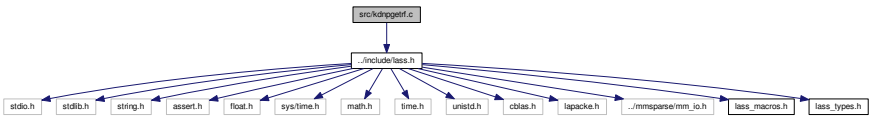
```
211                         tile_size_k, tile_size_n,
212                         1.0, TILE_A[ki][ki], tile_size_k,
213                              TILE_A[ki][ni], tile_size_n );
214
215             for ( mmi = ki + 1; mmi < nt; mmi++ )
216             {
217                 tile_size_mm = ddss_tile_size( N, mmi );
218
219                 #pragma oss task in( TILE_A[mmi][ki] ) \
220                         in( TILE_A[ki][ni] ) \
221                         inout( TILE_A[mmi][ni] ) \
222                         shared( TILE_A ) \
223                         firstprivate( ni, mmi, ki ) \
224                         label( dnpgetrf_dgemm )
225                 cblas_dgemm( CblasRowMajor,
226                              ( CBLAS_TRANSPOSE ) NoTrans,
227                              ( CBLAS_TRANSPOSE ) NoTrans,
228                              tile_size_mm,
229                              tile_size_n,
230                              TILE_SIZE,
231                              -1.0, TILE_A[mmi][ki], tile_size_k,
232                                    TILE_A[ki][ni],  tile_size_n,
233                               1.0, TILE_A[mmi][ni], tile_size_n );
234             }
235         }
236     }
237
238     /*****************************************
239     --From tiled data layout to flat data layout--
240     *****************************************/
241     ddss_dtiled2flat_nb( N, N, A, LDA, nt, nt, TILE_A );
242
243     // Triandular solve
244     // --SIDE = Left & UPLO = Lower & TRANS_A = NoTrans & Unit--
245     for ( ki = 0; ki < nt; ki++ )
246     {
247         tile_size_k = ddss_tile_size( N, ki );
248
249         for ( ni = 0; ni < nrhst; ni++ )
250         {
251             tile_size_n = ddss_tile_size( NRHS, ni );
252
253             #pragma oss task in( TILE_A[ki][ki] ) \
254                 inout(  TILE_B[ki][ni] ) \
255                 shared( TILE_A, TILE_B ) \
256                 firstprivate( ki, ni ) \
257                 label( dtrsm_dtrsm1 )
258             cblas_dtrsm( CblasRowMajor,
259                          ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Lower,
260                          ( CBLAS_TRANSPOSE ) NoTrans,
261                          ( CBLAS_DIAG ) Unit,
262                          tile_size_k,
263                          tile_size_n,
264                          1.0, TILE_A[ki][ki], tile_size_k,
265                               TILE_B[ki][ni], tile_size_n );
266         }
267         for ( nni = 0; nni < nrhst; nni++ )
268         {
269             tile_size_nn = ddss_tile_size( NRHS, nni );
270
271             for ( mi = ki + 1; mi < nt; mi++ )
272             {
273
274                 #pragma oss task in( TILE_A[mi][ki] ) \
275                     in( TILE_B[ki][nni] ) \
276                     inout( TILE_B[mi][nni] ) \
277                     shared( TILE_A, TILE_B ) \
278                     firstprivate( ki, mi, nni ) \
279                     label( dtrsm_dgemm1 )
280                 cblas_dgemm( CblasRowMajor,
281                              CblasNoTrans, CblasNoTrans,
282                              tile_size_k,
283                              tile_size_nn,
284                              tile_size_k,
285                              -1.0,  TILE_A[mi][ki],  tile_size_k,
286                                    TILE_B[ki][nni], tile_size_nn,
287                               1.0, TILE_B[mi][nni], tile_size_nn );
288             }
289         }
290     }
291
292     // Triandular solve
293     // --SIDE = Left & UPLO = Upper & TRANS_A = NoTrans & NonUnit--
294     for ( ki = nt - 1; ki >= 0; ki-- )
295     {
296         tile_size_k = ddss_tile_size( N, ki );
297
```

```
298         for ( ni = 0; ni < nrhst; ni++ )
299         {
300             tile_size_n = ddss_tile_size( NRHS, ni );
301 
302             #pragma oss task in( TILE_A[ki][ki] ) \
303                 inout( TILE_B[ki][ni] ) \
304                 shared( TILE_A, TILE_B ) \
305                 firstprivate( ki, ni ) \
306                 label( dtrsmi_dtrsm2 )
307             cblas_dtrsm( CblasRowMajor,
308                          ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Upper,
309                          ( CBLAS_TRANSPOSE ) NoTrans,
310                          ( CBLAS_DIAG ) NonUnit,
311                          tile_size_k,
312                          tile_size_n,
313                          1.0, TILE_A[ki][ki], tile_size_k,
314                               TILE_B[ki][ni], tile_size_n );
315         }
316 
317         for ( mi = ki - 1; mi >= 0; mi-- )
318         {
319             tile_size_m = ddss_tile_size( N, mi );
320 
321             for ( nni = 0; nni < nrhst; nni++ )
322             {
323                 tile_size_nn = ddss_tile_size( NRHS, nni );
324 
325                 #pragma oss task in( TILE_A[mi][ki] ) \
326                     in( TILE_B[ki][nni] ) \
327                     inout( TILE_B[mi][nni] ) \
328                     shared( TILE_A, TILE_B ) \
329                     firstprivate( ki, mi, nni ) \
330                     label( dtrsm_dgemm2 )
331                 cblas_dgemm( CblasRowMajor,
332                              CblasNoTrans, CblasNoTrans,
333                              tile_size_m,
334                              tile_size_nn,
335                              tile_size_k,
336                              -1.0, TILE_A[mi][ki],  tile_size_k,
337                                    TILE_B[ki][nni], tile_size_nn,
338                               1.0, TILE_B[mi][nni], tile_size_nn );
339             }
340         }
341     }
342 
343     /*****************************************
344     --From tiled data layout to flat data layout--
345     *****************************************/
346     ddss_dtiled2flat( N, NRHS, B, LDB, nt, nrhst, TILE_B );
347 
348     // --Tile A and B matrices free--
349     free( TILE_A );
350     free( TILE_B );
351 
352     return Success;
353 
354 }
```

2.22 src/kdnpgetrf.c File Reference

sLASs-DDSs kdnpgetrf routine.

`#include "../include/lass.h"`
Include dependency graph for kdnpgetrf.c:



Functions

- enum LASS_RETURN kdnpgetrf (int M, int N, double *A, int LDA)

2.22.1 Detailed Description

sLASs-DDSs kdnpgetrf routine.

sLASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es

Date

2018-07-24

2.22.2 Function Documentation

2.22.2.1 enum LASS_RETURN kdnpgetrf (int *M*, int *N*, double * *A*, int *LDA*)

Performs the LU factorization without pivoting of a general M-by-N matrix A:

```
A = L * U
```

where L is a lower triangular (lower trapezoidal if M > N) matrix with unit diagonal elements and U is an upper triangular (upper trapezoidal if M < N) matrix.

Parameters

in	*M*	int. M specifies the number of rows of the matrix A. M >= 0.
in	*N*	int. N specifies the number of columns of the matrix A. N >= 0.
in,out	*A*	double *. A is a pointer to a regular matrix of dimension M-by-N. On exit, if return value is Success, the matrix A is overwriten by the factors L and U. The unit diagonal elements of L are not stored.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

dnpgetrf
tune_dnpgetrf
smart_dnpgetrf
ddss_tile
ddss_flat2tiled
ddss_tiled2flat

Definition at line 74 of file kdnpgetrf.c.

References ddss_dflat2tiled(), ddss_dtiled2flat(), ddss_tile_size(), and dnpgetrf().

Referenced by ddss_dnpgetrf().

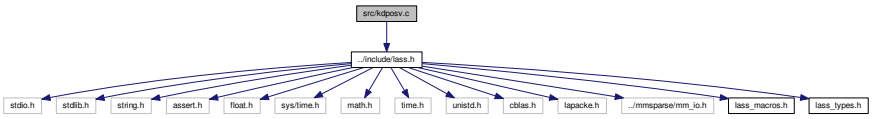
```
75 {
76
77     // Local variables
78     int mt, nt;
79     int mi, mmi, ki, ni, nni, li;
80     int tile_size_m;
81     int tile_size_mm;
82     int tile_size_n;
83     int tile_size_nn;
84     int tile_size_k;
85     int tile_size_km;
86     int tile_size_kn;
87
88     // Number of tiles
89     if ( M % TILE_SIZE == 0 )
90     {
91         mt = M / TILE_SIZE;
92     }
93     else
94     {
95         mt = ( M / TILE_SIZE ) + 1;
96     }
97
98     if ( N % TILE_SIZE == 0 )
99     {
100         nt = N / TILE_SIZE;
101     }
102     else
103     {
104         nt = ( N / TILE_SIZE ) + 1;
105     }
106
107     /***************************
108     --Tile A matrix allocation--
109     ***************************/
110
111     double (*TILE_A)[nt][TILE_SIZE * TILE_SIZE] = malloc ( mt * nt *
112         TILE_SIZE * TILE_SIZE * sizeof( double ) );
113
114     if ( TILE_A == NULL )
115     {
116         fprintf( stderr, "Failure in kdnpgetrf for matrix TILE_A\n" );
117         return NoSuccess;
118     }
119
120     /*******************************************
121     --From flat data layout to tiled data layout--
122     *******************************************/
123
124     ddss_dflat2tiled( M, N, A, LDA, mt, nt, TILE_A );
125
126     /****************
127     --DNPGETRF tile--
128     ****************/
129
130     for ( ki = 0; ki < MIN( mt, nt ); ki++ )
131     {
132         tile_size_km = ddss_tile_size( M, ki );
133         tile_size_kn = ddss_tile_size( N, ki );
134
135         #if defined(LASs_WITH_MKL)
136
137         tile_size_k = MIN( tile_size_km, tile_size_kn );
138
139         #pragma oss task inout( TILE_A[ki][ki] ) \
140                 shared( TILE_A ) \
141                 firstprivate( ki ) \
142                 priority( nt ) \
143                 label( mkl_dgetrfnpi )
144         LAPACKE_mkl_dgetrfnpi( CblasRowMajor,
145                                tile_size_km, tile_size_kn,
146                                tile_size_k,
147                                TILE_A[ki][ki], tile_size_kn );
148
149         #else
150
151         #pragma oss task inout( TILE_A[ki][ki] ) \
152                 shared( TILE_A ) \
```

```
153                 firstprivate( ki ) \
154                 priority( nt ) \
155                 label( dnpgetrf )
156         dnpgetrf( tile_size_km, tile_size_kn,
157                   TILE_A[ki][ki],
158                   tile_size_kn );
159
160         #endif
161
162         for ( mi = ki + 1; mi < mt; mi++ )
163         {
164             tile_size_m = ddss_tile_size( M, mi );
165
166             #pragma oss task in( TILE_A[ki][ki] ) \
167                     inout(TILE_A[mi][ki]) \
168                     shared( TILE_A ) \
169                     firstprivate( mi, ki ) \
170                     priority( nt ) \
171                     label( dtrsm_below )
172             cblas_dtrsm( CblasRowMajor,
173                          ( CBLAS_SIDE ) Right, ( CBLAS_UPLO ) Upper,
174                          ( CBLAS_TRANSPOSE ) NoTrans, ( CBLAS_DIAG ) NonUnit,
175                          tile_size_m, tile_size_kn,
176                          1.0, TILE_A[ki][ki], tile_size_kn,
177                               TILE_A[mi][ki], tile_size_kn );
178         }
179         for ( ni = ki + 1; ni < nt; ni++ )
180         {
181             tile_size_n = ddss_tile_size( N, ni );
182
183             #pragma oss task in( TILE_A[ki][ki] ) \
184                     inout(TILE_A[ki][ni]) \
185                     shared( TILE_A ) \
186                     firstprivate( ni, ki ) \
187                     priority( nt ) \
188                     label( dtrsm_right )
189             cblas_dtrsm( CblasRowMajor,
190                          ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Lower,
191                          ( CBLAS_TRANSPOSE ) NoTrans, ( CBLAS_DIAG ) Unit,
192                          tile_size_km, tile_size_n,
193                          1.0, TILE_A[ki][ki], tile_size_kn,
194                               TILE_A[ki][ni], tile_size_n );
195         }
196
197         for ( nni = ki + 1; nni < nt; nni++ )
198         {
199             tile_size_nn = ddss_tile_size( N, nni );
200
201             for ( mmi = ki + 1; mmi < mt; mmi++ )
202             {
203                 tile_size_mm = ddss_tile_size( M, mmi );
204
205                 #pragma oss task in( TILE_A[mmi][ki] ) \
206                     in( TILE_A[ki][nni] ) \
207                     inout( TILE_A[mmi][nni] ) \
208                     shared( TILE_A ) \
209                     firstprivate( nni, mmi, ki ) \
210                     priority( nt - nni ) \
211                     label( dgemm )
212                 cblas_dgemm( CblasRowMajor,
213                              ( CBLAS_TRANSPOSE ) NoTrans,
214                              ( CBLAS_TRANSPOSE ) NoTrans,
215                              tile_size_mm,
216                              tile_size_nn,
217                              TILE_SIZE,
218                              -1.0, TILE_A[mmi][ki],  tile_size_kn,
219                                    TILE_A[ki][nni],  tile_size_nn,
220                               1.0, TILE_A[mmi][nni], tile_size_nn );
221             }
222         }
223     }
224
225     // --From tile data layout to flat data layout--
226     ddss_dtiled2flat( M, N, A, LDA, mt, nt, TILE_A );
227
228     // --Tile A matrix free--
229     free( TILE_A );
230
231     return Success;
232
233 }
```

2.23 src/kdposv.c File Reference

LASs-DDSs kdposv routine.

```
#include "../include/lass.h"
```
Include dependency graph for kdposv.c:



Functions

- enum LASS_RETURN kdposv (enum DDSS_UPLO UPLO, int N, int NRHS, double ∗A, int LDA, double ∗B, int LDB)

2.23.1 Detailed Description

LASs-DDSs kdposv routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es

Date

2018-04-05

2.23.2 Function Documentation

2.23.2.1 enum LASS_RETURN kdposv (enum DDSS_UPLO *UPLO*, int *N*, int *NRHS*, double ∗ *A*, int *LDA*, double ∗ *B*, int *LDB*)

Solves a system of linear equations A X = B, where A is a M-by-M simmetric positive definite matrix and X and B are M-by-NRHS matrices. The matrix A is descomposed as:

```
A = L \times L^T
or
A = U^T \times U
```

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

Parameters

in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form of A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.
in	*N*	int. N specifies the order of the square matrix A. N >= 0.
in	*NRHS*	int. NRHS specifies the number of right-hand-sides (number of columns of B). NRHS >= 0.
in,out	*A*	double *. A is a pointer to a positive definite matrix of dimension N by LDA. On exit, if return value is Success, the matrix A is overwriten by the factor U or L.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).
in,out	*B*	double *. B is a pointer to a matrix of dimension N by NRHS, which stores the right-hand-sides of the systems of linear equations. (row-major order). On exit, if return value is Success, the matrix B is overwriten by the solution matrix X.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, NRHS).

Return values

Success	sucessful exit
NoSuccess	unsucessful exit

See also

ddss_dpotrf
ddss_tile
ddss_symflat2tiled
ddss_symtiled2flat

Definition at line 95 of file kdposv.c.

References ddss_dflat2tiled(), ddss_dsymflat2tiled(), ddss_dsymtiled2flat_nb(), ddss_dtiled2flat(), and ddss_tile_size().

Referenced by ddss_dposv().

```
99 {
100
101     // Local variables
102     int nt, nrhst;
103     int mi, mmi, ki, ni, nni;
104     int Am, An;
105     int Bm, Bn;
106     int tile_size_m;
107     int tile_size_mm;
108     int tile_size_n;
109     int tile_size_nn;
110     int tile_size_k;
111
112     // Number of tiles
113     if ( N % TILE_SIZE == 0 )
114     {
115         nt = N / TILE_SIZE;
116     }
117     else
118     {
119         nt = ( N / TILE_SIZE ) + 1;
120     }
```

```
121     if ( NRHS % TILE_SIZE == 0 )
122     {
123         nrhst = NRHS / TILE_SIZE;
124     }
125     else
126     {
127         nrhst = ( NRHS / TILE_SIZE ) + 1;
128     }
129
130
131     /****************************
132     --Tile A matrix declaration--
133     ****************************/
134
135     Am = nt;
136     An = nt;
137
138     Bm = nt;
139     Bn = nrhst;
140
141     /***************************
142     --Tile matrices allocation--
143     ***************************/
144
145     double (*TILE_A)[An][TILE_SIZE * TILE_SIZE] = malloc ( Am * An *
146         TILE_SIZE * TILE_SIZE * sizeof( double ) );
147
148     if ( TILE_A == NULL)
149     {
150         fprintf( stderr, "Failure in kdposv for matrix TILE_A\n" );
151         return NoSuccess;
152     }
153
154     double ( *TILE_B )[Bn][TILE_SIZE * TILE_SIZE] = malloc(
155              Bm * Bn * TILE_SIZE * TILE_SIZE * sizeof( double ) );
156
157     if ( TILE_B == NULL )
158     {
159         fprintf( stderr, "Failure in kdposv for matrix TILE_B\n" );
160         return NoSuccess;
161     }
162
163     /*********************************************
164     // --From flat data layout to tiled data layout--
165     *********************************************/
166
167     // From flat matrix A to tile matrix TILE_A
168     ddss_dsymflat2tiled( N, N, A, LDA, nt, nt, TILE_A, UPLO );
169
170     // From flat matrix B to tile matrix TILE_B
171     ddss_dflat2tiled( N, NRHS, B, LDB, nt, nrhst, TILE_B );
172
173     /**************
174     --DPOSV tile--
175     **************/
176
177     if ( UPLO == Lower )
178     {
179         // Cholesky descomposition
180         // --UPLO = Lower--
181         for ( ki = 0; ki < nt; ki++ )
182         {
183             tile_size_k = ddss_tile_size( N, ki );
184
185             #pragma oss task inout( TILE_A[ki][ki] ) \
186                 firstprivate( ki ) \
187                 label( dpotrf )
188             LAPACKE_dpotrf_work( LAPACK_ROW_MAJOR,
189                                  'L',
190                                  tile_size_k,
191                                  TILE_A[ki][ki], tile_size_k );
192
193             for ( mi = ki + 1; mi < nt; mi++ )
194             {
195                 tile_size_m = ddss_tile_size( N, mi );
196
197                 #pragma oss task in( TILE_A[ki][ki] ) \
198                     inout(TILE_A[mi][ki]) \
199                     firstprivate( TILE_A ) \
200                     firstprivate( mi, ki ) \
201                     label( dpotrf_dtrsm )
202                 cblas_dtrsm( CblasRowMajor,
203                              ( CBLAS_SIDE ) Right, ( CBLAS_UPLO ) Lower,
204                              ( CBLAS_TRANSPOSE ) Trans, ( CBLAS_DIAG ) NonUnit,
205                              tile_size_m, TILE_SIZE,
206                              1.0, TILE_A[ki][ki], TILE_SIZE,
207                                   TILE_A[mi][ki], TILE_SIZE );
```

```
208             }
209             for ( mmi = ki + 1; mmi < nt; mmi++ )
210             {
211                 tile_size_mm = ddss_tile_size( N, mmi );
212 
213                 #pragma oss task in( TILE_A[mmi][ki] ) \
214                     inout( TILE_A[mmi][mmi] ) \
215                     firstprivate( mmi, ki ) \
216                     label( dpotrf_dsyrk )
217                 cblas_dsyrk( CblasRowMajor,
218                              ( CBLAS_UPLO ) Lower, ( CBLAS_TRANSPOSE ) NoTrans,
219                              tile_size_mm, TILE_SIZE,
220                              -1.0,  TILE_A[mmi][ki], TILE_SIZE,
221                               1.0, TILE_A[mmi][mmi], tile_size_mm );
222 
223                 for ( ni = ki + 1; ni < mmi; ni++ )
224                 {
225 
226                     #pragma oss task in( TILE_A[mmi][ki] ) \
227                         in( TILE_A[ni][ki] ) \
228                         inout( TILE_A[mmi][ni] ) \
229                         firstprivate( ni, mmi, ki ) \
230                         label( dpotrf_dgemm )
231                     cblas_dgemm( CblasRowMajor,
232                                  ( CBLAS_TRANSPOSE ) NoTrans,
233                                  ( CBLAS_TRANSPOSE ) Trans,
234                                  TILE_SIZE,
235                                  TILE_SIZE,
236                                  TILE_SIZE,
237                                  -1.0, TILE_A[mmi][ki], TILE_SIZE,
238                                        TILE_A[ni][ki], TILE_SIZE,
239                                   1.0, TILE_A[mmi][ni], TILE_SIZE );
240                 }
241             }
242         }
243 
244         /*******************************************
245         --From tiled data layout to flat data layout--
246         *******************************************/
247         ddss_dsymtiled2flat_nb( N, N, A, LDA, nt, nt, TILE_A, UPLO );
248 
249         // Triangular solve
250         // --SIDE = Left & UPLO = Lower & TRANS_A = NoTrans--
251         for ( ki = 0; ki < nt; ki++ )
252         {
253             tile_size_k = ddss_tile_size( N, ki );
254 
255             for ( ni = 0; ni < nrhst; ni++ )
256             {
257                 tile_size_n = ddss_tile_size( NRHS, ni );
258 
259                 #pragma oss task in( TILE_A[ki][ki] ) \
260                     inout(  TILE_B[ki][ni] ) \
261                     shared( TILE_A, TILE_B ) \
262                     firstprivate( ki, ni ) \
263                     label( dtrsm_dtrsm1 )
264                 cblas_dtrsm( CblasRowMajor,
265                              ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Lower,
266                              ( CBLAS_TRANSPOSE ) NoTrans,
267                              ( CBLAS_DIAG ) NonUnit,
268                              tile_size_k,
269                              tile_size_n,
270                              1.0, TILE_A[ki][ki], tile_size_k,
271                                   TILE_B[ki][ni], tile_size_n );
272             }
273             for ( nni = 0; nni < nrhst; nni++ )
274             {
275                 tile_size_nn = ddss_tile_size( NRHS, nni );
276 
277                 for ( mi = ki + 1; mi < nt; mi++ )
278                 {
279 
280                     #pragma oss task in( TILE_A[mi][ki] ) \
281                         in( TILE_B[ki][nni] ) \
282                         inout( TILE_B[mi][nni] ) \
283                         shared( TILE_A, TILE_B ) \
284                         firstprivate( ki, mi, nni ) \
285                         label( dtrsm_dgemm1 )
286                     cblas_dgemm( CblasRowMajor,
287                                  CblasNoTrans, CblasNoTrans,
288                                  tile_size_k,
289                                  tile_size_nn,
290                                  tile_size_k,
291                                  -1.0,  TILE_A[mi][ki],  tile_size_k,
292                                         TILE_B[ki][nni], tile_size_nn,
293                                   1.0, TILE_B[mi][nni], tile_size_nn );
294                 }
```

```
295             }
296         }
297 
298         // Triangular solve
299         // --SIDE = Left & UPLO = Lower & TRANS_A = Trans--
300         for ( ki = 0; ki < nt; ki++ )
301         {
302             tile_size_k = ddss_tile_size( N, ki );
303 
304             for ( ni = 0; ni < nrhst; ni++ )
305             {
306                 tile_size_n = ddss_tile_size( NRHS, ni );
307 
308                 #pragma oss task in( TILE_A[ki][ki] ) \
309                     inout( TILE_B[ki][ni] ) \
310                     shared( TILE_A, TILE_B ) \
311                     firstprivate( ki, ni ) \
312                     label( dtrsm_dtrsm2 )
313                 cblas_dtrsm( CblasRowMajor,
314                              ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Lower,
315                              ( CBLAS_TRANSPOSE ) Trans,
316                              ( CBLAS_DIAG ) NonUnit,
317                              tile_size_k,
318                              tile_size_n,
319                              1.0,   TILE_A[ki][ki], tile_size_k,
320                                     TILE_B[ki][ni], tile_size_n );
321             }
322 
323             for ( nni = 0; nni < nrhst; nni++ )
324             {
325                 tile_size_nn = ddss_tile_size( NRHS, nni );
326 
327                 for ( mi = ki - 1; mi >= 0; mi-- )
328                 {
329                     tile_size_m = ddss_tile_size( N, mi );
330 
331                     #pragma oss task in( TILE_A[ki][mi] ) \
332                         in( TILE_B[ki][nni] ) \
333                         inout(  TILE_B[mi][nni] ) \
334                         shared( TILE_A, TILE_B ) \
335                         firstprivate( ki, mi, nni ) \
336                         label( dtrsm_dgemm2 )
337                     cblas_dgemm( CblasRowMajor,
338                                  CblasTrans, CblasNoTrans,
339                                  tile_size_m,
340                                  tile_size_nn,
341                                  tile_size_k,
342                                  -1.0,  TILE_A[ki][mi],  tile_size_m,
343                                         TILE_B[ki][nni], tile_size_nn,
344                                   1.0, TILE_B[mi][nni], tile_size_nn );
345                 }
346             }
347         }
348     }
349     else
350     {
351         // Cholesky descomposition
352         // --UPLO = Upper--
353         for ( ki = 0; ki < nt; ki++ )
354         {
355             tile_size_k = ddss_tile_size( N, ki );
356 
357             #pragma oss task inout( TILE_A[ki][ki] ) \
358                 shared( TILE_A ) \
359                 firstprivate( ki ) \
360                 label( dpotrf )
361             LAPACKE_dpotrf_work( LAPACK_ROW_MAJOR,
362                                  'U',
363                                  tile_size_k, TILE_A[ki][ki],
364                                  tile_size_k );
365 
366             for ( mi = ki + 1; mi < nt; mi++ )
367             {
368                 tile_size_m = ddss_tile_size( N, mi );
369 
370                 #pragma oss task in( TILE_A[ki][ki] ) \
371                     inout(TILE_A[ki][mi]) \
372                     shared( TILE_A ) \
373                     firstprivate( mi, ki ) \
374                     label( dpotrf_dtrsm )
375                 cblas_dtrsm( CblasRowMajor,
376                              ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Upper,
377                              ( CBLAS_TRANSPOSE ) Trans, ( CBLAS_DIAG ) NonUnit,
378                              TILE_SIZE, tile_size_m,
379                              1.0, TILE_A[ki][ki], TILE_SIZE,
380                                   TILE_A[ki][mi], tile_size_m );
381             }
```

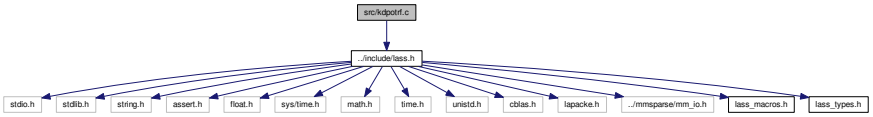
```
382             for ( mmi = ki + 1; mmi < nt; mmi++ )
383             {
384                 tile_size_mm = ddss_tile_size( N, mmi );
385 
386                 #pragma oss task in( TILE_A[ki][mmi] ) \
387                     inout( TILE_A[mmi][mmi] ) \
388                     shared( TILE_A ) \
389                     firstprivate( mmi, ki ) \
390                     label( dpotrf_dsyrk )
391                 cblas_dsyrk( CblasRowMajor,
392                              ( CBLAS_UPLO ) Upper, ( CBLAS_TRANSPOSE ) Trans,
393                              tile_size_mm, TILE_SIZE,
394                              -1.0,  TILE_A[ki][mmi], tile_size_mm,
395                               1.0, TILE_A[mmi][mmi], tile_size_mm );
396 
397                 for ( ni = ki + 1; ni < mmi; ni++ )
398                 {
399                     #pragma oss task in( TILE_A[ki][ni] ) \
400                         in( TILE_A[ki][mmi] ) \
401                         inout( TILE_A[ni][mmi] ) \
402                         shared( TILE_A ) \
403                         firstprivate( ni, mmi, ki ) \
404                         label( dpotrf_dgemm )
405                     cblas_dgemm( CblasRowMajor,
406                                  ( CBLAS_TRANSPOSE ) Trans,
407                                  ( CBLAS_TRANSPOSE ) NoTrans,
408                                  TILE_SIZE,
409                                  TILE_SIZE,
410                                  TILE_SIZE,
411                                  -1.0, TILE_A[ki][ni],  TILE_SIZE,
412                                        TILE_A[ki][mmi], TILE_SIZE,
413                                   1.0, TILE_A[ni][mmi], TILE_SIZE );
414                 }
415             }
416         }
417         /*****************************************
418         --From tiled data layout to flat data layout--
419          *****************************************/
420         ddss_dsymtiled2flat_nb( N, N, A, LDA, nt, nt, TILE_A, UPLO );
421         // Triangular solve
422         // --SIDE = Left & UPLO = Upper & TRANS_A = Trans--
423         for ( ki = 0; ki < nt; ki++ )
424         {
425             tile_size_k = ddss_tile_size( N, ki );
426 
427             for ( ni = 0; ni < nrhst; ni++ )
428             {
429                 tile_size_n = ddss_tile_size( NRHS, ni );
430 
431                 #pragma oss task in( TILE_A[ki][ki] ) \
432                     inout( TILE_B[ki][ni] ) \
433                     shared( TILE_A, TILE_B ) \
434                     firstprivate( ki, ni ) \
435                     label( dtrsm_dtrsm1 )
436                 cblas_dtrsm( CblasRowMajor,
437                              ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Lower,
438                              ( CBLAS_TRANSPOSE ) Trans,
439                              ( CBLAS_DIAG ) NonUnit,
440                              tile_size_k,
441                              tile_size_n,
442                              1.0, TILE_A[ki][ki], tile_size_k,
443                                   TILE_B[ki][ni], tile_size_n );
444             }
445 
446             for ( mi = ki + 1; mi < nt; mi++ )
447             {
448                 tile_size_m = ddss_tile_size( N, mi );
449 
450                 for ( nni = 0; nni < nrhst; nni++ )
451                 {
452                     tile_size_nn = ddss_tile_size( NRHS, nni );
453 
454                     #pragma oss task in( TILE_A[ki][mi]) \
455                         in( TILE_B[ki][nni] ) \
456                         inout(  TILE_B[mi][nni] ) \
457                         shared( TILE_A, TILE_B ) \
458                         firstprivate( ki, mi, nni ) \
459                         label( dtrsm_dgemm1 )
460                     cblas_dgemm( CblasRowMajor,
461                                  CblasTrans, CblasNoTrans,
462                                  tile_size_m,
463                                  tile_size_nn,
464                                  tile_size_k,
465                                  -1.0,  TILE_A[ki][mi],  tile_size_m,
466                                         TILE_B[ki][nni], tile_size_nn,
467                                   1.0, TILE_B[mi][nni], tile_size_nn );
468                 }
```

```
469             }
470         }
471         // Triangular solve
472         // --SIDE = Left & UPLO = Upper & TRANS_A = NoTrans--
473         for ( ki = 0; ki < nt; ki++ )
474         {
475             tile_size_k = ddss_tile_size( N, ki );
476 
477             for ( ni = 0; ni < nrhst; ni++ )
478             {
479                 tile_size_n = ddss_tile_size( NRHS, ni );
480 
481                 #pragma oss task in( TILE_A[ki][ki] ) \
482                     inout( TILE_B[ki][ni] ) \
483                     shared( TILE_A, TILE_B ) \
484                     firstprivate( ki, ni ) \
485                     label( dtrsm_dtrsm2 )
486                 cblas_dtrsm( CblasRowMajor,
487                              ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Lower,
488                              ( CBLAS_TRANSPOSE ) NoTrans,
489                              ( CBLAS_DIAG ) NonUnit,
490                              tile_size_k,
491                              tile_size_n,
492                              1.0, TILE_A[ki][ki], tile_size_k,
493                                   TILE_B[ki][ni], tile_size_n );
494             }
495 
496             for ( nni = 0; nni < nrhst; nni++ )
497             {
498                 tile_size_nn = ddss_tile_size( NRHS, nni );
499 
500                 for ( mi = ki - 1; mi >= 0; mi-- )
501                 {
502                     tile_size_m = ddss_tile_size( N, mi );
503 
504                     #pragma oss task in( TILE_A[mi][ki] ) \
505                         in( TILE_B[ki][nni] ) \
506                         inout( TILE_B[mi][nni] ) \
507                         shared( TILE_A, TILE_B ) \
508                         firstprivate( ki, mi, nni ) \
509                         label( dtrsm_dgemm2 )
510                     cblas_dgemm( CblasRowMajor,
511                                  CblasNoTrans, CblasNoTrans,
512                                  tile_size_m,
513                                  tile_size_nn,
514                                  tile_size_k,
515                                  -1.0,  TILE_A[mi][ki],  tile_size_k,
516                                         TILE_B[ki][nni], tile_size_nn,
517                                   1.0, TILE_B[mi][nni], tile_size_nn );
518                 }
519             }
520         }
521     }
522 
523     /*****************************************
524     --From tiled data layout to flat data layout--
525     *****************************************/
526     ddss_dtiled2flat( N, NRHS, B, LDB, nt, nrhst, TILE_B );
527 
528     // --Tile A and B matrices free--
529     free( TILE_A );
530     free( TILE_B );
531 
532     return Success;
533 
534 }
```

2.24 src/kdpotrf.c File Reference

LASs-DDSs kdpotrf routine.

```
#include "../include/lass.h"
```
Include dependency graph for kdpotrf.c:



Functions

- enum LASS_RETURN kdpotrf (enum DDSS_UPLO UPLO, int N, double ∗A, int LDA)

2.24.1 Detailed Description

LASs-DDSs kdpotrf routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es

Date

2017-24-10

2.24.2 Function Documentation

2.24.2.1 enum LASS_RETURN kdpotrf (enum DDSS_UPLO *UPLO*, int *N*, double ∗ *A*, int *LDA*)

Performs the Cholesky factorization of a symmetric positive definite matrix A:

```
A = L \times L^T
or
A = U^T \times U
```

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

Parameters

in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form of A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.
in	*N*	int. N specifies the order of the square matrix A. N >= 0.
in,out	*A*	double ∗. A is a pointer to a positive definite matrix of dimension N by LDA. On exit, if return value is Success, the matrix A is overwriten by the factor U or L.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).

Return values

Success	sucessful exit
NoSuccess	unsucessful exit

See also

ddss_dpotrf
ddss_tile
ddss_symflat2tiled
ddss_symtiled2flat

Definition at line 78 of file kdpotrf.c.

References ddss_dsymflat2tiled(), ddss_dsymtiled2flat(), and ddss_tile_size().

Referenced by ddss_dpotrf().

```
79 {
80
81     // Local variables
82     int nt;
83     int mi, mmi, ki, ni;
84     int Am;
85     int An;
86     int tile_size_m;
87     int tile_size_mm;
88     int tile_size_n;
89     int tile_size_k;
90     int k_check;
91     double betat;
92
93     // Number of tiles
94     if ( N % TILE_SIZE == 0 )
95     {
96         nt = N / TILE_SIZE;
97     }
98     else
99     {
100         nt = ( N / TILE_SIZE ) + 1;
101     }
102
103     /****************************
104     --Tile A matrix declaration--
105     ****************************/
106
107     Am = nt;
108     An = nt;
109
110     /***************************
111     --Tile A matrix allocation--
112     ***************************/
113
114     double (*TILE_A)[An][TILE_SIZE * TILE_SIZE] = malloc ( Am * An *
115         TILE_SIZE * TILE_SIZE * sizeof( double ) );
116
117     if ( TILE_A == NULL)
118     {
119         fprintf( stderr, "Failure in kdpotrf for matrix TILE_A\n" );
120         return NoSuccess;
121     }
122
123     /***********************************************
124     // --From flat data layout to tiled data layout--
125     ***********************************************/
126
127     ddss_dsymflat2tiled( N, N, A, LDA, nt, nt, TILE_A, UPLO );
128
129     /**************
130     --DPOTRF tile--
131     **************/
132
133     if ( UPLO == Lower )
134     {
135         // --UPLO = Lower--
```

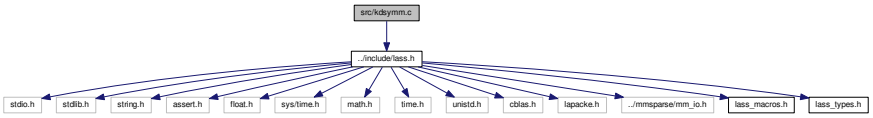
```
136         for ( ki = 0; ki < nt; ki++ )
137         {
138             tile_size_k = ddss_tile_size( N, ki );
139 
140             #pragma oss task inout( TILE_A[ki][ki] ) \
141                 firstprivate( ki ) \
142                 label( dpotrf )
143             LAPACKE_dpotrf_work( LAPACK_ROW_MAJOR,
144                                  'L',
145                                  tile_size_k, TILE_A[ki][ki], tile_size_k );
146 
147             for ( mi = ki + 1; mi < nt; mi++ )
148             {
149                 tile_size_m = ddss_tile_size( N, mi );
150 
151                 #pragma oss task in( TILE_A[ki][ki] ) \
152                     inout(TILE_A[mi][ki]) \
153                     firstprivate( mi, ki ) \
154                     label( dtrsm )
155                 cblas_dtrsm( CblasRowMajor,
156                              ( CBLAS_SIDE ) Right, ( CBLAS_UPLO ) Lower,
157                              ( CBLAS_TRANSPOSE ) Trans, ( CBLAS_DIAG ) NonUnit,
158                              tile_size_m, TILE_SIZE,
159                              1.0, TILE_A[ki][ki], TILE_SIZE,
160                                   TILE_A[mi][ki], TILE_SIZE );
161             }
162             for ( mmi = ki + 1; mmi < nt; mmi++ )
163             {
164                 tile_size_mm = ddss_tile_size( N, mmi );
165 
166                 #pragma oss task in( TILE_A[mmi][ki] ) \
167                     inout( TILE_A[mmi][mmi] ) \
168                     firstprivate( mmi, ki ) \
169                     label( dsyrk )
170                 cblas_dsyrk( CblasRowMajor,
171                              ( CBLAS_UPLO ) Lower, ( CBLAS_TRANSPOSE ) NoTrans,
172                              tile_size_mm, TILE_SIZE,
173                              -1.0,  TILE_A[mmi][ki], TILE_SIZE,
174                               1.0, TILE_A[mmi][mmi], tile_size_mm );
175 
176                 for ( ni = ki + 1; ni < mmi; ni++ )
177                 {
178 
179                     #pragma oss task in( TILE_A[mmi][ki] ) \
180                         in( TILE_A[ni][ki] ) \
181                         inout( TILE_A[mmi][ni] ) \
182                         firstprivate( ni, mmi, ki ) \
183                         label( dgemm )
184                         cblas_dgemm( CblasRowMajor,
185                                      ( CBLAS_TRANSPOSE ) NoTrans,
186                                      ( CBLAS_TRANSPOSE ) Trans,
187                                      TILE_SIZE,
188                                      TILE_SIZE,
189                                      TILE_SIZE,
190                                      -1.0, TILE_A[mmi][ki], TILE_SIZE,
191                                            TILE_A[ni][ki], TILE_SIZE,
192                                       1.0, TILE_A[mmi][ni], TILE_SIZE );
193                 }
194             }
195         }
196     }
197     else
198     {
199         // --UPLO = Upper--
200         for ( ki = 0; ki < nt; ki++ )
201         {
202             tile_size_k = ddss_tile_size( N, ki );
203 
204             #pragma oss task inout( TILE_A[ki][ki] ) \
205                 shared( TILE_A ) \
206                 firstprivate( ki ) \
207                 label( dpotrf )
208             LAPACKE_dpotrf_work( LAPACK_ROW_MAJOR,
209                                  'U',
210                                  tile_size_k, TILE_A[ki][ki],
211                                  tile_size_k );
212 
213             for ( mi = ki + 1; mi < nt; mi++ )
214             {
215                 tile_size_m = ddss_tile_size( N, mi );
216 
217                 #pragma oss task in( TILE_A[ki][ki] ) \
218                     inout(TILE_A[ki][mi]) \
219                     shared( TILE_A ) \
220                     firstprivate( mi, ki ) \
221                     label( dtrsm )
222                 cblas_dtrsm( CblasRowMajor,
```

```
223                             ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Upper,
224                             ( CBLAS_TRANSPOSE ) Trans, ( CBLAS_DIAG ) NonUnit,
225                             TILE_SIZE, tile_size_m,
226                             1.0, TILE_A[ki][ki], TILE_SIZE,
227                                  TILE_A[ki][mi], tile_size_m );
228             }
229             for ( mmi = ki + 1; mmi < nt; mmi++ )
230             {
231                 tile_size_mm = ddss_tile_size( N, mmi );
232 
233                 #pragma oss task in( TILE_A[ki][mmi] ) \
234                     inout( TILE_A[mmi][mmi] ) \
235                     shared( TILE_A ) \
236                     firstprivate( mmi, ki ) \
237                     label( dsyrk )
238                 cblas_dsyrk( CblasRowMajor,
239                             ( CBLAS_UPLO ) Upper, ( CBLAS_TRANSPOSE ) Trans,
240                             tile_size_mm, TILE_SIZE,
241                             -1.0,  TILE_A[ki][mmi], tile_size_mm,
242                              1.0, TILE_A[mmi][mmi], tile_size_mm );
243 
244                 for ( ni = ki + 1; ni < mmi; ni++ )
245                 {
246                     #pragma oss task in( TILE_A[ki][ni] ) \
247                         in( TILE_A[ki][mmi] ) \
248                         inout( TILE_A[ni][mmi] ) \
249                         shared( TILE_A ) \
250                         firstprivate( ni, mmi, ki ) \
251                         label( dgemm )
252                         cblas_dgemm( CblasRowMajor,
253                                     ( CBLAS_TRANSPOSE ) Trans,
254                                     ( CBLAS_TRANSPOSE ) NoTrans,
255                                     TILE_SIZE,
256                                     TILE_SIZE,
257                                     TILE_SIZE,
258                                     -1.0, TILE_A[ki][ni],  TILE_SIZE,
259                                           TILE_A[ki][mmi], TILE_SIZE,
260                                      1.0, TILE_A[ni][mmi], TILE_SIZE );
261                 }
262             }
263         }
264     }
265 
266     // --From tile data layout to flat data layout--
267     ddss_dsymtiled2flat( N, N, A, LDA, nt, nt, TILE_A, UPLO );
268 
269     // --Tile A matrix free--
270     free( TILE_A );
271 
272     return Success;
273 
274 }
```

2.25 src/kdsymm.c File Reference

LASs-DDSs kdsymm routine.

`#include "../include/lass.h"`
Include dependency graph for kdsymm.c:



Functions

- enum LASS_RETURN kdsymm (enum DDSS_SIDE SIDE, enum DDSS_UPLO UPLO, int M, int N, const double ALPHA, double ∗A, int LDA, double ∗B, int LDB, const double BETA, double ∗C, int LDC)

2.25.1 Detailed Description

LASs-DDSs kdsymm routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es

Date

2017-08-11

2.25.2 Function Documentation

2.25.2.1 enum LASS_RETURN kdsymm (enum DDSS_SIDE *SIDE*, enum DDSS_UPLO *UPLO*, int *M*, int *N*, const double *ALPHA*, double * *A*, int *LDA*, double * *B*, int *LDB*, const double *BETA*, double * *C*, int *LDC*)

Performs one of the matrix-matrix operations:

```
C = ALPHA * A * B + BETA * C
```

or

```
C = ALPHA * B * A + BETA * C
```

where op(X) is one of:

```
op( X ) = X      or
op( X ) = X**T
```

ALPHA and BETA are scalars, A is a symmetric matrix, and B and C are M by N matrices.

Parameters

in	*SIDE*	enum DDSS_SIDE. UPLO specifies the position of the symmetric A matrix in the operation: • Left: C = ALPHA * A * B + BETA * C • Right: C = ALPHA * B * A + BETA * C
in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form of A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.
in	*M*	int. M specifies the number of rows of the matrix C. M must be equal or greater than zero.
in	*N*	int. N specifies the number of columns of the matrix C. N must be equal or greater than zero.
in	*ALPHA*	double.

Parameters

in	*A*	double ∗. A is a pointer to a matrix of dimension Ma (rows) by Na (columns), where Ma is M and Na is M when SIDE = Left, and Ma is N and Na is N when SIDE = Right
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, Na).
in	*B*	double ∗. B is a pointer to a matrix of dimension M by N.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, N).
in	*BETA*	double.
in,out	*C*	double ∗. C is a pointer to a matrix of dimension M by N. On exit, C is overwritten by the M by N matrix.
in	*LDC*	int. LDC specifies the number of columns of C (row-major order). LDC must be at least max(1, N).

Return values

Success	sucessful exit
NoSuccess	unsucessful exit

See also

ddss_dgemm
ddss_tile
ddss_flat2tiled
ddss_symflat2tiled
ddss_tiled2flat
ddss_symtiled2flat

Definition at line 125 of file kdsymm.c.

References ddss_dflat2tiled(), ddss_dsymflat2tiled(), ddss_dtiled2flat(), and ddss_tile_size().

Referenced by ddss_dsymm().

```
130 {
131
132     // Local variables
133     int mt, kt, nt;
134     int mi, ki, ni;
135     int Am, An;
136     int Amt, Ant;
137     int tile_size_m;
138     int tile_size_n;
139     int tile_size_k;
140     int k_check;
141     double betat;
142
143     // Number of tiles
144     if ( M % TILE_SIZE == 0 )
145     {
146         mt = M / TILE_SIZE;
147     }
148     else
149     {
150         mt = ( M / TILE_SIZE ) + 1;
151     }
152
153     if ( N % TILE_SIZE == 0 )
154     {
155         nt = N / TILE_SIZE;
156     }
157     else
```

```
158     {
159         nt = ( N / TILE_SIZE ) + 1;
160     }
161
162     if ( SIDE == Left )
163     {
164         if ( M % TILE_SIZE == 0 )
165         {
166             kt = M / TILE_SIZE;
167         }
168         else
169         {
170             kt = ( M / TILE_SIZE ) + 1;
171         }
172     }
173     else
174     {
175         if ( N % TILE_SIZE == 0 )
176         {
177             kt = N / TILE_SIZE;
178         }
179         else
180         {
181             kt = ( N / TILE_SIZE ) + 1;
182         }
183     }
184
185     /*****************************
186     --Tile matrices declaration--
187     *****************************/
188
189     if ( SIDE == Left )
190     {
191         Am = M;
192         An = M;
193         Amt = mt;
194         Ant = mt;
195     }
196     else
197     {
198         Am = N;
199         An = N;
200         Amt = nt;
201         Ant = nt;
202     }
203
204     /****************************
205     --Tile matrices allocation--
206     ****************************/
207
208     double (*TILE_A)[Ant][TILE_SIZE * TILE_SIZE] = malloc ( Amt * Ant *
209         TILE_SIZE * TILE_SIZE * sizeof( double ) );
210
211     if ( TILE_A == NULL)
212     {
213         fprintf( stderr, "Failure in kdsymm for matrix TILE_A\n" );
214         return NoSuccess;
215     }
216
217     double (*TILE_B)[nt][TILE_SIZE * TILE_SIZE] = malloc ( mt * nt *
218         TILE_SIZE * TILE_SIZE * sizeof( double ) );
219
220     if ( TILE_B == NULL)
221     {
222         fprintf( stderr, "Failure in kdsymm for matrix TILE_B\n" );
223         return NoSuccess;
224     }
225
226     double (*TILE_C)[nt][TILE_SIZE * TILE_SIZE] = malloc ( mt * nt *
227         TILE_SIZE * TILE_SIZE * sizeof( double ) );
228
229     if ( TILE_C == NULL)
230     {
231         fprintf( stderr, "Failure in kdsymm for matrix TILE_C\n" );
232         return NoSuccess;
233     }
234
235     /*********************************************
236     // --From flat data layout to tiled data layout--
237     *********************************************/
238
239     // From flat and symmetric matrix A to tile matrix TILE_A
240     ddss_dsymflat2tiled( Am, An, A, LDA, Amt, Ant, TILE_A, UPLO );
241
242     // From flat matrix B to tile matrix TILE_B
243     ddss_dflat2tiled( M, N, B, LDB, mt, nt, TILE_B );
244
```

```
245     // From flat matrix C to tile matrix TILE_C
246     ddss_dflat2tiled( M, N, C, LDC, mt, nt, TILE_C );
247
248     /*************
249     --DSYMM tile--
250     *************/
251
252     for ( mi = 0; mi < mt; mi++ )
253     {
254         tile_size_m  = ddss_tile_size( M, mi );
255         for ( ni = 0; ni < nt; ni++ )
256         {
257             tile_size_n  = ddss_tile_size( N, ni );
258             if ( SIDE == Left )
259             {
260                 // --SIDE = Left & UPLO = Lower--
261                 if ( UPLO == Lower)
262                 {
263                     for ( ki = 0; ki < kt; ki++ )
264                     {
265                         if (ki == 0)
266                         {
267                             betat = BETA;
268                         }
269                         else
270                         {
271                             betat = 1.0;
272                         }
273                         tile_size_k  = ddss_tile_size( M, ki );
274
275                         if ( ki < mi )
276                         {
277                             #pragma oss task in( TILE_A[mi][ki] ) \
278                                 in( TILE_B[ki][ni] ) \
279                                 inout( TILE_C[mi][ni] ) \
280                                 shared( TILE_A, TILE_B, TILE_C ) \
281                                 firstprivate( mi, ni, ki, betat )
282                             cblas_dgemm( CblasRowMajor,
283                                         ( CBLAS_TRANSPOSE ) NoTrans,
284                                         ( CBLAS_TRANSPOSE ) NoTrans,
285                                         tile_size_m,
286                                         tile_size_n,
287                                         tile_size_k,
288                                         ALPHA, TILE_A[mi][ki], tile_size_k,
289                                                TILE_B[ki][ni], tile_size_n,
290                                         betat, TILE_C[mi][ni], tile_size_n );
291                         }
292                         else
293                         {
294                             if ( ki == mi )
295                             {
296                                 #pragma oss task in( TILE_A[ki][ki] ) \
297                                     in( TILE_B[ki][ni] ) \
298                                     inout( TILE_C[mi][ni] ) \
299                                     shared( TILE_A, TILE_B, TILE_C ) \
300                                     firstprivate( mi, ni, ki, betat )
301                                 cblas_dsymm( CblasRowMajor,
302                                             ( CBLAS_SIDE ) SIDE,
303                                             ( CBLAS_UPLO ) UPLO,
304                                             tile_size_m, tile_size_n,
305                                             ALPHA, TILE_A[ki][ki], tile_size_k,
306                                                    TILE_B[ki][ni], tile_size_n,
307                                             betat, TILE_C[mi][ni],
308                                             tile_size_n );
309                             }
310                             else
311                             {
312                                 #pragma oss task in( TILE_A[ki][mi] ) \
313                                     in( TILE_B[ki][ni] ) \
314                                     inout( TILE_C[mi][ni] ) \
315                                     shared( TILE_A, TILE_B, TILE_C ) \
316                                     firstprivate( mi, ni, ki, betat )
317                                 cblas_dgemm( CblasRowMajor,
318                                             ( CBLAS_TRANSPOSE ) Trans,
319                                             ( CBLAS_TRANSPOSE ) NoTrans,
320                                             tile_size_m,
321                                             tile_size_n,
322                                             tile_size_k,
323                                             ALPHA, TILE_A[ki][mi], tile_size_m,
324                                                    TILE_B[ki][ni], tile_size_n,
325                                             betat, TILE_C[mi][ni],
326                                             tile_size_n );
327                             }
328                         }
329                     }
330                 }
331                 // --SIDE = Left & UPLO = Upper--
```

```
332                 else
333                 {
334                     for ( ki = 0; ki < kt; ki++ )
335                     {
336                         if (ki == 0)
337                         {
338                             betat = BETA;
339                         }
340                         else
341                         {
342                             betat = 1.0;
343                         }
344                         tile_size_k  = ddss_tile_size( M, ki );
345 
346                         if ( ki < mi )
347                         {
348                             #pragma oss task in( TILE_A[ki][mi] ) \
349                                 in( TILE_B[ki][ni] ) \
350                                 inout( TILE_C[mi][ni] ) \
351                                 shared( TILE_A, TILE_B, TILE_C ) \
352                                 firstprivate( mi, ni, ki, betat )
353                             cblas_dgemm( CblasRowMajor,
354                                         ( CBLAS_TRANSPOSE ) Trans,
355                                         ( CBLAS_TRANSPOSE ) NoTrans,
356                                         tile_size_m,
357                                         tile_size_n,
358                                         tile_size_k,
359                                         ALPHA, TILE_A[ki][mi], tile_size_m,
360                                                TILE_B[ki][ni], tile_size_n,
361                                         betat, TILE_C[mi][ni], tile_size_n );
362                         }
363                         else
364                         {
365                             if ( ki == mi )
366                             {
367                                 #pragma oss task in( TILE_A[ki][ki] ) \
368                                     in( TILE_B[ki][ni] ) \
369                                     inout( TILE_C[mi][ni] ) \
370                                     shared( TILE_A, TILE_B, TILE_C ) \
371                                     firstprivate( mi, ni, ki, betat )
372                                 cblas_dsymm( CblasRowMajor,
373                                             ( CBLAS_SIDE ) SIDE,
374                                             ( CBLAS_UPLO ) UPLO,
375                                             tile_size_m, tile_size_n,
376                                             ALPHA, TILE_A[ki][ki], tile_size_k,
377                                                    TILE_B[ki][ni], tile_size_n,
378                                             betat, TILE_C[mi][ni],
379                                             tile_size_n );
380                             }
381                             else
382                             {
383                                 #pragma oss task in( TILE_A[mi][ki] ) \
384                                     in( TILE_B[ki][ni] ) \
385                                     inout( TILE_C[mi][ni] ) \
386                                     shared( TILE_A, TILE_B, TILE_C ) \
387                                     firstprivate( mi, ni, ki, betat )
388                                 cblas_dgemm( CblasRowMajor,
389                                             ( CBLAS_TRANSPOSE ) NoTrans,
390                                             ( CBLAS_TRANSPOSE ) NoTrans,
391                                             tile_size_m,
392                                             tile_size_n,
393                                             tile_size_k,
394                                             ALPHA, TILE_A[mi][ki], tile_size_k,
395                                                    TILE_B[ki][ni], tile_size_n,
396                                             betat, TILE_C[mi][ni],
397                                             tile_size_n );
398                             }
399                         }
400                     }
401                 }
402             }
403             else
404             {
405                 // --SIDE = Right & UPLO = Lower--
406                 if ( UPLO == Lower)
407                 {
408                     for ( ki = 0; ki < kt; ki++ )
409                     {
410                         if (ki == 0)
411                         {
412                             betat = BETA;
413                         }
414                         else
415                         {
416                             betat = 1.0;
417                         }
418                         tile_size_k  = ddss_tile_size( N, ki );
```

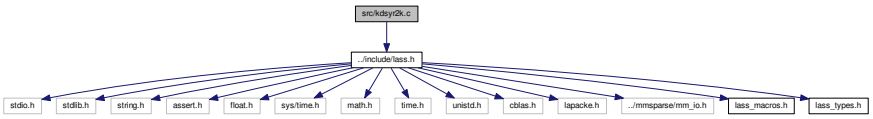
```
419
420                         if ( ki < ni )
421                         {
422                             #pragma oss task in( TILE_B[mi][ki] ) \
423                                 in( TILE_A[ni][ki] ) \
424                                 inout( TILE_C[mi][ni] ) \
425                                 shared( TILE_A, TILE_B, TILE_C ) \
426                                 firstprivate( mi, ni, ki, betat )
427                             cblas_dgemm( CblasRowMajor,
428                                         ( CBLAS_TRANSPOSE ) NoTrans,
429                                         ( CBLAS_TRANSPOSE ) Trans,
430                                         tile_size_m,
431                                         tile_size_n,
432                                         tile_size_k,
433                                         ALPHA, TILE_B[mi][ki], tile_size_k,
434                                                TILE_A[ni][ki], tile_size_k,
435                                         betat, TILE_C[mi][ni], tile_size_n );
436                         }
437                         else
438                         {
439                             if ( ki == ni )
440                             {
441                                 #pragma oss task in( TILE_A[ki][ki] ) \
442                                     in( TILE_B[mi][ki] ) \
443                                     inout( TILE_C[mi][ni] ) \
444                                     shared( TILE_A, TILE_B, TILE_C ) \
445                                     firstprivate( mi, ni, ki, betat )
446                                 cblas_dsymm( CblasRowMajor,
447                                             ( CBLAS_SIDE ) SIDE,
448                                             ( CBLAS_UPLO ) UPLO,
449                                             tile_size_m, tile_size_n,
450                                             ALPHA, TILE_A[ki][ki], tile_size_k,
451                                                    TILE_B[mi][ki], tile_size_k,
452                                             betat, TILE_C[mi][ni],
453                                             tile_size_n );
454                             }
455                             else
456                             {
457                                 #pragma oss task in( TILE_B[mi][ki] ) \
458                                     in( TILE_A[ki][ni] ) \
459                                     inout( TILE_C[mi][ni] ) \
460                                     shared( TILE_A, TILE_B, TILE_C ) \
461                                     firstprivate( mi, ni, ki, betat )
462                                 cblas_dgemm( CblasRowMajor,
463                                             ( CBLAS_TRANSPOSE ) NoTrans,
464                                             ( CBLAS_TRANSPOSE ) NoTrans,
465                                             tile_size_m,
466                                             tile_size_n,
467                                             tile_size_k,
468                                             ALPHA, TILE_B[mi][ki], tile_size_k,
469                                                    TILE_A[ki][ni], tile_size_n,
470                                             betat, TILE_C[mi][ni],
471                                             tile_size_n );
472                             }
473                         }
474                     }
475                 }
476                 // --SIDE = Right & UPLO = Upper--
477                 else
478                 {
479                     for ( ki = 0; ki < kt; ki++ )
480                     {
481                         if (ki == 0)
482                         {
483                             betat = BETA;
484                         }
485                         else
486                         {
487                             betat = 1.0;
488                         }
489                         tile_size_k  = ddss_tile_size( N, ki );
490
491                         if ( ki < ni )
492                         {
493                             #pragma oss task in( TILE_B[mi][ki] ) \
494                                 in( TILE_A[ki][ni] ) \
495                                 inout( TILE_C[mi][ni] ) \
496                                 shared( TILE_A, TILE_B, TILE_C ) \
497                                 firstprivate( mi, ni, ki, betat )
498                             cblas_dgemm( CblasRowMajor,
499                                         ( CBLAS_TRANSPOSE ) NoTrans,
500                                         ( CBLAS_TRANSPOSE ) NoTrans,
501                                         tile_size_m,
502                                         tile_size_n,
503                                         tile_size_k,
504                                         ALPHA, TILE_B[mi][ki], tile_size_k,
505                                                TILE_A[ki][ni], tile_size_n,
```

```
506                                         betat, TILE_C[mi][ni], tile_size_n );
507                         }
508                         else
509                         {
510                             if ( ki == ni )
511                             {
512                                 #pragma oss task in( TILE_A[ki][ki] ) \
513                                     in( TILE_B[ki][ni] ) \
514                                     inout( TILE_C[mi][ni] ) \
515                                     shared( TILE_A, TILE_B, TILE_C ) \
516                                     firstprivate( mi, ni, ki, betat )
517                                 cblas_dsymm( CblasRowMajor,
518                                             ( CBLAS_SIDE ) SIDE,
519                                             ( CBLAS_UPLO ) UPLO,
520                                             tile_size_m, tile_size_n,
521                                             ALPHA, TILE_A[ki][ki], tile_size_k,
522                                                    TILE_B[mi][ki], tile_size_k,
523                                             betat, TILE_C[mi][ni],
524                                             tile_size_n );
525                             }
526                             else
527                             {
528                                 #pragma oss task in( TILE_B[mi][ki] ) \
529                                     in( TILE_A[ni][ki] ) \
530                                     inout( TILE_C[mi][ni] ) \
531                                     shared( TILE_A, TILE_B, TILE_C ) \
532                                     firstprivate( mi, ni, ki, betat )
533                                 cblas_dgemm( CblasRowMajor,
534                                             ( CBLAS_TRANSPOSE ) NoTrans,
535                                             ( CBLAS_TRANSPOSE ) Trans,
536                                             tile_size_m,
537                                             tile_size_n,
538                                             tile_size_k,
539                                             ALPHA, TILE_B[mi][ki], tile_size_k,
540                                                    TILE_A[ni][ki], tile_size_k,
541                                             betat, TILE_C[mi][ni],
542                                             tile_size_n );
543                             }
544                         }
545                     }
546                 }
547             }
548         }
549     }
550
551     // From tile matrix TILE_C to flat matrix C
552     ddss_dtiled2flat( M, N, C, LDC, mt, nt, TILE_C );
553
554     // --Tile matrices free--
555     free( TILE_A );
556     free( TILE_B );
557     free( TILE_C );
558
559     return Success;
560
561 }
```

2.26 src/kdsyr2k.c File Reference

LASs-DDSs kdsyr2k routine.

`#include "../include/lass.h"`
Include dependency graph for kdsyr2k.c:



Functions

- enum LASS_RETURN kdsyr2k (enum DDSS_UPLO UPLO, enum DDSS_TRANS TRANS, int N, int K, const double ALPHA, double *A, int LDA, double *B, int LDB, const double BETA, double *C, int LDC)

2.26.1 Detailed Description

LASs-DDSs kdsyr2k routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es
Boro Sofranac boro.sofranac@bsc.es

Date

2018-02-21

2.26.2 Function Documentation

2.26.2.1 enum LASS_RETURN kdsyr2k (enum DDSS_UPLO *UPLO*, enum DDSS_TRANS *TRANS*, int *N*, int *K*, const double *ALPHA*, double * *A*, int *LDA*, double * *B*, int *LDB*, const double *BETA*, double * *C*, int *LDC*)

Performs one of the symmetric rank 2k operations:

```
C = ALPHA * A * B**T + ALPHA * B * A**T + BETA * C or
C = ALPHA * A**T * B + ALPHA * B**T * A + BETA * C
```

ALPHA and BETA are scalars, C is an N by N symmetric matrix and A and B are N by K matrices in the first case and K by N matrices in the second case.

Parameters

in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form in which C is stored: • Lower: Lower triangular part of C is stored. The upper traingular part is not referenced. • Upper: Upper triangular part of C is stored. The lower triangular part is not referenced.
in	*TRANS*	enum DDSS_TRANS. TRANS specifies the operation to be performed as follows: • NoTrans: C = ALPHA * A * B**T + ALPHA * B * A**T + BETA * C • Trans: C = ALPHA * A**T * B + ALPHA * B**T * A + BETA * C
in	*N*	int. N specifies the order of matrix C. N must be at least zero.
in	*K*	int. With TRANS = NoTrans, K specifies the number of columns of the matrices A and B, and with TRANS = Trans, K specifies the number of rows of the matrices A and B. K must be at least zero.
in	*ALPHA*	double. ALPHA specifies the scalar alpha.
in	*A*	double *. A is a pointer to a matrix of dimension Na (rows) by Ka (columns), where Na is N and Ka is K when TRANS = NoTrans, and Na is K and Ka is N otherwise.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). When TRANS = NoTrans then LDA must be at least max(1, K), otherwise LDA must be at least max(1, N).

Parameters

in	*B*	double *. B is a pointer to a matrix of dimension Nb (rows) by Kb (columns), where Nb is N and Kb is K when TRANS = NoTrans, and Nb is K and Kb is N otherwise.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). When TRANS = NoTrans then LDB must be at least max(1, K), otherwise LDB must be at least max(1, N).
in	*BETA*	double. BETA specifies the scalar beta.
in,out	*C*	double *. C is a pointer to a matrix of dimension N by N. When UPLO = Uppper the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of C is overwritten by the upper triangular part of the updated solution matrix C. When UPLO = Lower the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of C is overwritten by the lower triangular part of the updated solution matrix C.
in	*LDC*	int. LDC specifies the number of columns of C (row-major order). LDC must be at least max(1, N).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

ddss_tile
ddss_flat2tiled
ddss_dsymflat2tiled
ddss_dsymtiled2flat

Definition at line 124 of file kdsyr2k.c.

References ddss_dflat2tiled(), ddss_dsymflat2tiled(), ddss_dsymtiled2flat(), and ddss_tile_size().

Referenced by ddss_dsyr2k().

```
129 {
130
131     // Local variables
132     int nt, kt;
133     int mi, ni, ki;
134     int Am, An;
135     int Bm, Bn;
136     int Cm, Cn;
137     int tile_size_m;
138     int tile_size_n;
139     int tile_size_k;
140     int ka;
141     int kb;
142     double beta;
143
144     // Number of tiles
145     if ( N % TILE_SIZE == 0 )
146     {
147         nt = N / TILE_SIZE;
148     }
149     else
150     {
151         nt = ( N / TILE_SIZE ) + 1;
152     }
153
154     if ( K % TILE_SIZE == 0 )
155     {
156         kt = K / TILE_SIZE;
```

```
157     }
158     else
159     {
160         kt = ( K / TILE_SIZE ) + 1;
161     }
162
163     /****************************
164      --Tile matrices declaration--
165      ****************************/
166
167     if ( TRANS == NoTrans )
168     {
169         Am = Bm = nt;
170         An = Bn = kt;
171         ka = kb = N;
172     }
173     else
174     {
175         Am = Bm = kt;
176         An = Bn = nt;
177         ka = kb = K;
178     }
179
180     Cm = Cn = nt;
181
182     /****************************
183      --Tile matrices allocation--
184      ****************************/
185
186     double ( *TILE_A )[An][TILE_SIZE * TILE_SIZE] = malloc(
187             Am * An * TILE_SIZE * TILE_SIZE * sizeof( double ) );
188
189     if ( TILE_A == NULL )
190     {
191         fprintf( stderr, "Failure in ddss_dtile_alloc for matrix TILE_A\n" );
192         return NoSuccess;
193     }
194
195     double ( *TILE_B )[Bn][TILE_SIZE * TILE_SIZE] = malloc(
196             Bm * Bn * TILE_SIZE * TILE_SIZE * sizeof( double ) );
197
198     if ( TILE_B == NULL )
199     {
200         fprintf( stderr, "Failure in ddss_dtile_alloc for matrix TILE_B\n" );
201         return NoSuccess;
202     }
203
204     double ( *TILE_C )[Cn][TILE_SIZE * TILE_SIZE] = malloc(
205             Cm * Cn * TILE_SIZE * TILE_SIZE * sizeof( double ) );
206
207     if ( TILE_C == NULL )
208     {
209         fprintf( stderr, "Failure in ddss_dtile_alloc for matrix TILE_C\n" );
210         return NoSuccess;
211     }
212
213     /*****************************************
214      --From flat data layout to tiled data layout--
215      *****************************************/
216
217     // From flat matrix A to tile matrix TILE_A
218     ddss_dflat2tiled( ka, LDA, A, LDA, Am, An, TILE_A );
219
220     // From flat matrix B to tile matrix TILE_B
221     ddss_dflat2tiled( kb, LDB, B, LDB, Bm, Bn, TILE_B );
222
223     // From flat matrix C to tile matrix TILE_C
224     ddss_dsymflat2tiled( N, N, C, LDC, Cm, Cn, TILE_C, UPLO );
225
226     /*************
227      --DSYRK tile--
228      *************/
229
230     // --TRANS = NoTrans & UPLO = Upper--
231     if ( TRANS == NoTrans )
232     {
233         if ( UPLO == Upper )
234         {
235             for ( mi = 0; mi < nt; mi++ )
236             {
237                 tile_size_m = ddss_tile_size( N, mi );
238                 for ( ni = 0; ni < kt; ni++ )
239                 {
240                     tile_size_n = ddss_tile_size( K, ni );
241
242                     if ( ni == 0 )
243                     {
```

```
244                     beta = BETA;
245                 }
246                 else
247                 {
248                     beta = 1.0;
249                 }
250 
251                 #pragma oss task in( TILE_A[mi][ni] ) \
252                     in( TILE_B[mi][ni] ) \
253                     inout( TILE_C[mi][mi] ) \
254                     shared( TILE_A, TILE_B, TILE_C ) \
255                     firstprivate( mi,ni ) \
256                     label( dsyr2k )
257                 cblas_dsyr2k( CblasRowMajor,
258                             ( CBLAS_UPLO ) UPLO, ( CBLAS_TRANSPOSE ) TRANS,
259                             tile_size_m,
260                             tile_size_n,
261                             ALPHA, TILE_A[mi][ni], tile_size_n,
262                                    TILE_B[mi][ni], tile_size_n,
263                             beta,  TILE_C[mi][mi], tile_size_m );
264 
265                 for ( ki = mi + 1; ki < nt; ki++ )
266                 {
267                     tile_size_k = ddss_tile_size( N, ki );
268 
269                     #pragma oss task in( TILE_A[mi][ni] ) \
270                         in( TILE_A[ki][ni] ) \
271                         in( TILE_B[mi][ni] ) \
272                         in( TILE_B[ki][ni] ) \
273                         inout( TILE_C[mi][ki] ) \
274                         shared( TILE_A, TILE_B, TILE_C ) \
275                         firstprivate( mi,ni,ki ) \
276                         label( dgemm )
277                     {
278                         cblas_dgemm( CblasRowMajor,
279                                     CblasNoTrans, CblasTrans,
280                                     tile_size_m,
281                                     tile_size_k,
282                                     tile_size_n,
283                                     ALPHA, TILE_A[mi][ni], tile_size_n,
284                                            TILE_B[ki][ni], tile_size_n,
285                                     beta,  TILE_C[mi][ki], tile_size_k );
286 
287                         cblas_dgemm( CblasRowMajor,
288                                     CblasNoTrans, CblasTrans,
289                                     tile_size_m,
290                                     tile_size_k,
291                                     tile_size_n,
292                                     ALPHA, TILE_B[mi][ni], tile_size_n,
293                                            TILE_A[ki][ni], tile_size_n,
294                                     1.0,   TILE_C[mi][ki], tile_size_k );
295                     }
296                 }
297 
298             }
299         }
300     }
301     // --TRANS = NoTrans & UPLO = Lower--
302     else
303     {
304         for ( mi = 0; mi < nt; mi++ )
305         {
306             tile_size_m = ddss_tile_size( N, mi );
307             for ( ni = 0; ni < kt; ni++ )
308             {
309                 tile_size_n = ddss_tile_size( K, ni );
310 
311                 if ( ni == 0 )
312                 {
313                     beta = BETA;
314                 }
315                 else
316                 {
317                     beta = 1.0;
318                 }
319 
320                 #pragma oss task in( TILE_A[mi][ni] ) \
321                     in( TILE_B[mi][ni] ) \
322                     inout( TILE_C[mi][mi] ) \
323                     shared( TILE_A, TILE_B, TILE_C ) \
324                     firstprivate( mi,ni ) \
325                     label( dsyr2k )
326                 cblas_dsyr2k( CblasRowMajor,
327                             ( CBLAS_UPLO ) UPLO, ( CBLAS_TRANSPOSE ) TRANS,
328                             tile_size_m,
329                             tile_size_n,
330                             ALPHA, TILE_A[mi][ni], tile_size_n,
```

```
331                                     TILE_B[mi][ni], tile_size_n,
332                                beta,  TILE_C[mi][mi], tile_size_m );
333
334                     for ( ki = mi + 1; ki < nt; ki++ )
335                     {
336                         tile_size_k = ddss_tile_size( N, ki );
337
338                         #pragma oss task in( TILE_A[ki][ni] ) \
339                             in( TILE_A[mi][ni] ) \
340                             in( TILE_B[ki][ni] ) \
341                             in( TILE_B[mi][ni] ) \
342                             inout( TILE_C[ki][mi] ) \
343                             shared( TILE_A, TILE_B, TILE_C ) \
344                             firstprivate( mi,ni,ki ) \
345                             label( dgemm )
346                         {
347                             cblas_dgemm( CblasRowMajor,
348                                          CblasNoTrans, CblasTrans,
349                                          tile_size_k,
350                                          tile_size_m,
351                                          tile_size_n,
352                                          ALPHA, TILE_A[ki][ni], tile_size_n,
353                                                 TILE_B[mi][ni], tile_size_n,
354                                          beta,  TILE_C[ki][mi], tile_size_m );
355
356                             cblas_dgemm( CblasRowMajor,
357                                          CblasNoTrans, CblasTrans,
358                                          tile_size_k,
359                                          tile_size_m,
360                                          tile_size_n,
361                                          ALPHA, TILE_B[ki][ni], tile_size_n,
362                                                 TILE_A[mi][ni], tile_size_n,
363                                          1.0,   TILE_C[ki][mi], tile_size_m );
364                         }
365                     }
366
367                 }
368             }
369         }
370     }
371     // --TRANS = Trans & UPLO = Upper--
372     else
373     {
374         if ( UPLO == Upper )
375         {
376             for ( mi = 0; mi < kt; mi++ )
377             {
378                 tile_size_m = ddss_tile_size( K, mi );
379                 for ( ni = 0; ni < nt; ni++ )
380                 {
381                     tile_size_n = ddss_tile_size( N, ni );
382
383                     if ( mi == 0 )
384                     {
385                         beta = BETA;
386                     }
387                     else
388                     {
389                         beta = 1.0;
390                     }
391
392                     #pragma oss task in( TILE_A[mi][ni] ) \
393                         in( TILE_B[mi][ni] ) \
394                         inout( TILE_C[ni][ni] ) \
395                         shared( TILE_A, TILE_B, TILE_C ) \
396                         firstprivate( mi,ni ) \
397                         label( dsyr2k )
398                     cblas_dsyr2k( CblasRowMajor,
399                                   ( CBLAS_UPLO ) UPLO, ( CBLAS_TRANSPOSE ) TRANS,
400                                   tile_size_n,
401                                   tile_size_m,
402                                   ALPHA, TILE_A[mi][ni], tile_size_n,
403                                          TILE_B[mi][ni], tile_size_n,
404                                   beta,  TILE_C[ni][ni], tile_size_n );
405
406                     for ( ki = ni + 1; ki < nt; ki++ )
407                     {
408                         tile_size_k = ddss_tile_size( N, ki );
409
410                         #pragma oss task in( TILE_A[mi][ni] ) \
411                             in( TILE_A[mi][ki] ) \
412                             in( TILE_B[mi][ni] ) \
413                             in( TILE_B[mi][ki] ) \
414                             inout( TILE_C[ni][ki] ) \
415                             shared( TILE_A, TILE_B, TILE_C ) \
416                             firstprivate( mi,ni,ki ) \
417                             label( dgemm )
```

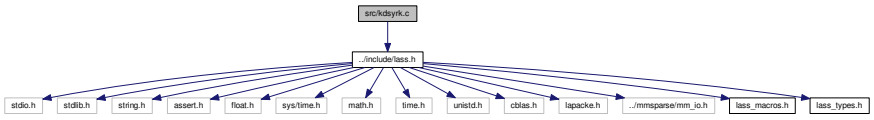
```
418                         {
419                             cblas_dgemm( CblasRowMajor,
420                                         CblasTrans, CblasNoTrans,
421                                         tile_size_n,
422                                         tile_size_k,
423                                         tile_size_m,
424                                         ALPHA, TILE_A[mi][ni], tile_size_n,
425                                                TILE_B[mi][ki], tile_size_k,
426                                         beta,  TILE_C[ni][ki], tile_size_k );
427 
428                             cblas_dgemm( CblasRowMajor,
429                                         CblasTrans, CblasNoTrans,
430                                         tile_size_n,
431                                         tile_size_k,
432                                         tile_size_m,
433                                         ALPHA, TILE_B[mi][ni], tile_size_n,
434                                                TILE_A[mi][ki], tile_size_k,
435                                         1.0,   TILE_C[ni][ki], tile_size_k );
436                         }
437                     }
438                 }
439             }
440         }
441         // --TRANS = Trans & UPLO = Lower--
442         else
443         {
444             for ( mi = 0; mi < kt; mi++ )
445             {
446                 tile_size_m = ddss_tile_size( K, mi );
447                 for ( ni = 0; ni < nt; ni++ )
448                 {
449                     tile_size_n = ddss_tile_size( N, ni );
450 
451                     if ( mi == 0 )
452                     {
453                         beta = BETA;
454                     }
455                     else
456                     {
457                         beta = 1.0;
458                     }
459 
460                     #pragma oss task in( TILE_A[mi][ni] ) \
461                         in( TILE_B[mi][ni] ) \
462                         inout( TILE_C[ni][ni] ) \
463                         shared( TILE_A, TILE_B, TILE_C ) \
464                         firstprivate( mi,ni ) \
465                         label( dsyr2k )
466                     cblas_dsyr2k( CblasRowMajor,
467                                  ( CBLAS_UPLO ) UPLO, ( CBLAS_TRANSPOSE ) TRANS,
468                                  tile_size_n,
469                                  tile_size_m,
470                                  ALPHA, TILE_A[mi][ni], tile_size_n,
471                                         TILE_B[mi][ni], tile_size_n,
472                                  beta,  TILE_C[ni][ni], tile_size_n );
473 
474                     for ( ki = ni + 1; ki < nt; ki++ )
475                     {
476                         tile_size_k = ddss_tile_size( N, ki );
477 
478                         #pragma oss task in( TILE_A[mi][ki] ) \
479                             in( TILE_A[mi][ni] ) \
480                             in( TILE_B[mi][ki] ) \
481                             in( TILE_B[mi][ni] ) \
482                             inout( TILE_C[ki][ni] ) \
483                             shared( TILE_A, TILE_B, TILE_C ) \
484                             firstprivate( mi,ni,ki ) \
485                             label( dgemm )
486                         {
487                             cblas_dgemm( CblasRowMajor,
488                                         CblasTrans, CblasNoTrans,
489                                         tile_size_k,
490                                         tile_size_n,
491                                         tile_size_m,
492                                         ALPHA, TILE_A[mi][ki], tile_size_k,
493                                                TILE_B[mi][ni], tile_size_n,
494                                         beta,  TILE_C[ki][ni], tile_size_n );
495 
496                             cblas_dgemm( CblasRowMajor,
497                                         CblasTrans, CblasNoTrans,
498                                         tile_size_k,
499                                         tile_size_n,
500                                         tile_size_m,
501                                         ALPHA, TILE_B[mi][ki], tile_size_k,
502                                                TILE_A[mi][ni], tile_size_n,
503                                         1.0,   TILE_C[ki][ni], tile_size_n );
504                         }
```

```
505                     }
506                 }
507             }
508         }
509     }
510
511     /*****************************************
512      --From tiled data layout to flat data layout--
513      *****************************************/
514
515     ddss_dsymtiled2flat( N, N, C, LDC, Cm, Cn, TILE_C, UPLO );
516
517     // --Tile matrices free--
518     free( TILE_A );
519     free( TILE_B );
520     free( TILE_C );
521
522     return Success;
523
524 }
```

2.27 src/kdsyrk.c File Reference

LASs-DDSs kdsyrk routine.

`#include "../include/lass.h"`
Include dependency graph for kdsyrk.c:



Functions

- enum LASS_RETURN kdsyrk (enum DDSS_UPLO UPLO, enum DDSS_TRANS TRANS_A, int N, int K, const double ALPHA, double ∗A, int LDA, const double BETA, double ∗C, int LDC)

2.27.1 Detailed Description

LASs-DDSs kdsyrk routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es
Boro Sofranac boro.sofranac@bsc.es

Date

2018-02-13

2.27.2 Function Documentation

2.27.2.1 enum LASS_RETURN kdsyrk (enum DDSS_UPLO *UPLO*, enum DDSS_TRANS *TRANS_A*, int *N*, int *K*, const double *ALPHA*, double * *A*, int *LDA*, const double *BETA*, double * *C*, int *LDC*)

Performs one of the symmetric rank k operations:

```
C = ALPHA * A * op( A ) + BETA * C  or
C = ALPHA * op( A ) * A + BETA * C
```

where op(X) is:

```
op( X ) = X**T
```

ALPHA and BETA are scalars, C is an N by N symmetric matrix and A is an N by K matrix in the first case and a K by N matrix in the second case.

Parameters

in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form in which C is stored: • Lower: Lower triangular part of C is stored. The upper traingular part is not referenced. • Upper: Upper triangular part of C is stored. The lower triangular part is not referenced.
in	*TRANS_A*	enum DDSS_TRANS. TRANS_A specifies the operation to be performed as follows: • NoTrans: C = ALPHA * A * A**T + BETA * C • Trans: C = ALPHA * A**T * A + BETA * C
in	*N*	int. N specifies the order of matrix C. N must be at least zero.
in	*K*	int. With TRANS_A = NoTrans, K specifies the number of columns of the matrix A, and with TRANS_A = Trans, K specifies the number of rows of the matrix A. K must be at least zero.
in	*ALPHA*	double. ALPHA specifies the scalar alpha.
in	*A*	double *. A is a pointer to a matrix of dimension Na (rows) by Ka (columns), where Na is N and Ka is K when TRANS_A = NoTrans, and Na is K and Ka is N otherwise.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). When TRANS_A = NoTrans then LDA must be at least max(1, K), otherwise LDA must be at least max(1, N).
in	*BETA*	double. BETA specifies the scalar beta.
in,out	*C*	double *. C is a pointer to a matrix of dimension N by N. When UPLO = Uppper the strictly lower triangular part of C is not referenced. On exit, the upper triangular part of C is overwritten by the upper triangular part of the updated solution matrix C. When UPLO = Lower the strictly upper triangular part of C is not referenced. On exit, the lower triangular part of C is overwritten by the lower triangular part of the updated solution matrix C.
in	*LDC*	int. LDC specifies the number of columns of C (row-major order). LDC must be at least max(1, N).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

ddss_tile
ddss_flat2tiled
ddss_dsymflat2tiled
ddss_dsymtiled2flat

Definition at line 117 of file kdsyrk.c.

References ddss_dflat2tiled(), ddss_dsymflat2tiled(), ddss_dsymtiled2flat(), and ddss_tile_size().

Referenced by ddss_dsyrk().

```
121 {
122
123     // Local variables
124     int nt, kt;
125     int mi, ni, ki;
126     int Am, An;
127     int Cm, Cn;
128     int tile_size_m;
129     int tile_size_n;
130     int tile_size_k;
131     int ka;
132     double beta;
133
134     // Number of tiles
135     if ( N % TILE_SIZE == 0 )
136     {
137         nt = N / TILE_SIZE;
138     }
139     else
140     {
141         nt = ( N / TILE_SIZE ) + 1;
142     }
143
144     if ( K % TILE_SIZE == 0 )
145     {
146         kt = K / TILE_SIZE;
147     }
148     else
149     {
150         kt = ( K / TILE_SIZE ) + 1;
151     }
152
153     /****************************
154      --Tile matrices declaration--
155      ****************************/
156
157     if ( TRANS_A == NoTrans )
158     {
159         Am = nt;
160         An = kt;
161         ka = N;
162     }
163     else
164     {
165         Am = kt;
166         An = nt;
167         ka = K;
168     }
169
170     Cm = Cn = nt;
171
172     /****************************
173      --Tile matrices allocation--
174      ****************************/
175
176     double ( *TILE_A )[An][TILE_SIZE * TILE_SIZE] = malloc(
177             Am * An * TILE_SIZE * TILE_SIZE * sizeof( double ) );
```

```
178
179     if ( TILE_A == NULL )
180     {
181         fprintf( stderr, "Failure in ddss_dtile_alloc for matrix TILE_A\n" );
182         return NoSuccess;
183     }
184
185     double ( *TILE_C )[Cn][TILE_SIZE * TILE_SIZE] = malloc(
186             Cm * Cn * TILE_SIZE * TILE_SIZE * sizeof( double ) );
187
188     if ( TILE_C == NULL )
189     {
190         fprintf( stderr, "Failure in ddss_dtile_alloc for matrix TILE_C\n" );
191         return NoSuccess;
192     }
193
194     /*******************************************
195      --From flat data layout to tiled data layout--
196      *******************************************/
197
198     // From flat matrix A to tile matrix TILE_A
199     ddss_dflat2tiled( ka, LDA, A, LDA, Am, An, TILE_A );
200
201     // From flat matrix C to tile matrix TILE_C
202     ddss_dsymflat2tiled( N, N, C, LDC, Cm, Cn, TILE_C, UPLO );
203
204     /*************
205      --DSYRK tile--
206      *************/
207
208     // --TRANS_A = NoTrans & UPLO = Upper--
209     if ( TRANS_A == NoTrans )
210     {
211         if ( UPLO == Upper )
212         {
213             for ( mi = 0; mi < nt; mi++ )
214             {
215                 tile_size_m = ddss_tile_size( N, mi );
216                 for ( ni = 0; ni < kt; ni++ )
217                 {
218                     tile_size_n = ddss_tile_size( K, ni );
219
220                     if ( ni == 0 )
221                     {
222                         beta = BETA;
223                     }
224                     else
225                     {
226                         beta = 1.0;
227                     }
228
229                     #pragma oss task in( TILE_A[mi][ni] ) \
230                         inout( TILE_C[mi][mi] ) \
231                         shared( TILE_A, TILE_C ) \
232                         firstprivate( mi, ni ) \
233                         label( dsyrk )
234                     cblas_dsyrk( CblasRowMajor,
235                                  ( CBLAS_UPLO ) UPLO,
236                                  ( CBLAS_TRANSPOSE ) TRANS_A,
237                                  tile_size_m,
238                                  tile_size_n,
239                                  ALPHA, TILE_A[mi][ni], tile_size_n,
240                                  beta,  TILE_C[mi][mi], tile_size_m );
241
242                     for ( ki = mi + 1; ki < nt; ki++ )
243                     {
244                         tile_size_k = ddss_tile_size( N, ki );
245
246                         #pragma oss task in( TILE_A[mi][ni] ) \
247                             in( TILE_A[ki][ni] ) \
248                             inout( TILE_C[mi][ki] ) \
249                             shared( TILE_A, TILE_C ) \
250                             firstprivate( mi, ni, ki ) \
251                             label( dgemm )
252                         cblas_dgemm( CblasRowMajor,
253                                      CblasNoTrans, CblasTrans,
254                                      tile_size_m,
255                                      tile_size_k,
256                                      tile_size_n,
257                                      ALPHA, TILE_A[mi][ni], tile_size_n,
258                                             TILE_A[ki][ni], tile_size_n,
259                                      beta,  TILE_C[mi][ki], tile_size_k );
260                     }
261
262                 }
263             }
264         }
```

```
265         // --TRANS_A = NoTrans & UPLO = Lower--
266         else
267         {
268             for ( mi = 0; mi < nt; mi++ )
269             {
270                 tile_size_m = ddss_tile_size( N, mi );
271                 for ( ni = 0; ni < kt; ni++ )
272                 {
273                     tile_size_n = ddss_tile_size( K, ni );
274 
275                     if ( ni == 0 )
276                     {
277                         beta = BETA;
278                     }
279                     else
280                     {
281                         beta = 1.0;
282                     }
283 
284                     #pragma oss task in( TILE_A[mi][ni] ) \
285                         inout( TILE_C[mi][mi] ) \
286                         shared( TILE_A, TILE_C ) \
287                         firstprivate( mi, ni ) \
288                         label( dsyrk )
289                     cblas_dsyrk( CblasRowMajor,
290                                  ( CBLAS_UPLO ) UPLO,
291                                  ( CBLAS_TRANSPOSE ) TRANS_A,
292                                  tile_size_m,
293                                  tile_size_n,
294                                  ALPHA, TILE_A[mi][ni], tile_size_n,
295                                  beta,  TILE_C[mi][mi], tile_size_m );
296 
297                     for ( ki = mi + 1; ki < nt; ki++ )
298                     {
299                         tile_size_k = ddss_tile_size( N, ki );
300 
301                         #pragma oss task in( TILE_A[ki][ni] ) \
302                             in( TILE_A[mi][ni] ) \
303                             inout( TILE_C[ki][mi] ) \
304                             shared( TILE_A, TILE_C ) \
305                             firstprivate( mi, ni, ki ) \
306                             label( dgemm )
307                         cblas_dgemm( CblasRowMajor,
308                                      CblasNoTrans, CblasTrans,
309                                      tile_size_k,
310                                      tile_size_m,
311                                      tile_size_n,
312                                      ALPHA, TILE_A[ki][ni], tile_size_n,
313                                             TILE_A[mi][ni], tile_size_n,
314                                      beta,  TILE_C[ki][mi], tile_size_m );
315                     }
316 
317                 }
318             }
319         }
320     }
321     // --TRANS_A = Trans & UPLO = Upper--
322     else
323     {
324         if ( UPLO == Upper )
325         {
326             for ( mi = 0; mi < kt; mi++ )
327             {
328                 tile_size_m = ddss_tile_size( K, mi );
329                 for ( ni = 0; ni < nt; ni++ )
330                 {
331                     tile_size_n = ddss_tile_size( N, ni );
332 
333                     if ( mi == 0 )
334                     {
335                         beta = BETA;
336                     }
337                     else
338                     {
339                         beta = 1.0;
340                     }
341 
342                     #pragma oss task in( TILE_A[mi][ni] ) \
343                         inout( TILE_C[ni][ni] ) \
344                         shared( TILE_A, TILE_C ) \
345                         firstprivate( mi, ni ) \
346                         label( dsyrk )
347                     cblas_dsyrk( CblasRowMajor,
348                                  ( CBLAS_UPLO ) UPLO,
349                                  ( CBLAS_TRANSPOSE ) TRANS_A,
350                                  tile_size_n,
351                                  tile_size_m,
```

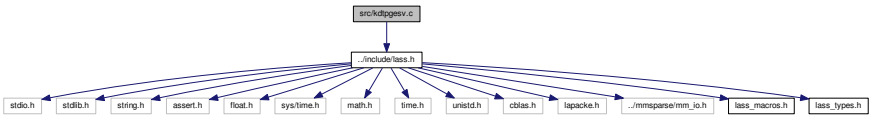
```
352                                 ALPHA, TILE_A[mi][ni], tile_size_n,
353                                 beta,  TILE_C[ni][ni], tile_size_n );
354
355                     for ( ki = ni + 1; ki < nt; ki++ )
356                     {
357                         tile_size_k = ddss_tile_size( N, ki );
358
359                         #pragma oss task in( TILE_A[mi][ni] ) \
360                             in( TILE_A[mi][ki] ) \
361                             inout( TILE_C[ni][ki] ) \
362                             shared( TILE_A, TILE_C ) \
363                             firstprivate( mi, ni, ki ) \
364                             label( dgemm )
365                         cblas_dgemm( CblasRowMajor,
366                                      CblasTrans, CblasNoTrans,
367                                      tile_size_n,
368                                      tile_size_k,
369                                      tile_size_m,
370                                      ALPHA, TILE_A[mi][ni], tile_size_n,
371                                             TILE_A[mi][ki], tile_size_k,
372                                      beta,  TILE_C[ni][ki], tile_size_k );
373                     }
374                 }
375             }
376         }
377         // --TRANS_A = Trans & UPLO = Lower--
378         else
379         {
380             for ( mi = 0; mi < kt; mi++ )
381             {
382                 tile_size_m = ddss_tile_size( K, mi );
383                 for ( ni = 0; ni < nt; ni++ )
384                 {
385                     tile_size_n = ddss_tile_size( N, ni );
386
387                     if ( mi == 0 )
388                     {
389                         beta = BETA;
390                     }
391                     else
392                     {
393                         beta = 1.0;
394                     }
395
396                     #pragma oss task in( TILE_A[mi][ni] ) \
397                         inout( TILE_C[ni][ni] ) \
398                         shared( TILE_A, TILE_C ) \
399                         firstprivate( mi, ni ) \
400                         label( dsyrk )
401                     cblas_dsyrk( CblasRowMajor,
402                                  ( CBLAS_UPLO ) UPLO,
403                                  ( CBLAS_TRANSPOSE ) TRANS_A,
404                                  tile_size_n,
405                                  tile_size_m,
406                                  ALPHA, TILE_A[mi][ni], tile_size_n,
407                                  beta,  TILE_C[ni][ni], tile_size_n );
408
409                     for ( ki = ni + 1; ki < nt; ki++ )
410                     {
411                         tile_size_k = ddss_tile_size( N, ki );
412
413                         #pragma oss task in( TILE_A[mi][ki] ) \
414                             in( TILE_A[mi][ni] ) \
415                             inout( TILE_C[ki][ni] ) \
416                             shared( TILE_A, TILE_C ) \
417                             firstprivate( mi, ni, ki ) \
418                             label( dgemm )
419                         cblas_dgemm( CblasRowMajor,
420                                      CblasTrans, CblasNoTrans,
421                                      tile_size_k,
422                                      tile_size_n,
423                                      tile_size_m,
424                                      ALPHA, TILE_A[mi][ki], tile_size_k,
425                                             TILE_A[mi][ni], tile_size_n,
426                                      beta,  TILE_C[ki][ni], tile_size_n );
427                     }
428                 }
429             }
430         }
431     }
432
433     /*****************************************
434      --From tiled data layout to flat data layout--
435      *****************************************/
436
437     ddss_dsymtiled2flat( N, N, C, LDC, Cm, Cn, TILE_C, UPLO );
438
```

```
439     // --Tile matrices free--
440     free( TILE_A );
441     free( TILE_C );
442
443     return Success;
444
445 }
```

2.28 src/kdtpgesv.c File Reference

LASs-DDSs kdtpgesv routine.

`#include "../include/lass.h"`
Include dependency graph for kdtpgesv.c:



Functions

- enum LASS_RETURN kdtpgesv (int N, int NRHS, double ∗A, int LDA, int ∗IPIV, double ∗B, int LDB)

2.28.1 Detailed Description

LASs-DDSs kdtpgesv routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es

Date

2018-09-20

2.28.2 Function Documentation

2.28.2.1 enum LASS_RETURN kdtpgesv (int *N*, int *NRHS*, double ∗ *A*, int *LDA*, int ∗ *IPIV*, double ∗ *B*, int *LDB*)

Solves a system of linear equations A X = B, where A is a N-by-N general matrix and X and B are N-by-NRHS matrices. The matrix A is factorized using the LU descomposition with tiled-pivoting. The matrix A is descomposed as:

```
A = P * L * U
```

where P is a permutation matrix, L is a lower triangular matrix with unit diagonal elements and U is an upper triangular matrix.

Parameters

in	*N*	int. N specifies the order of the square matrix A. N >= 0.

NRHS int. NRHS specifies the number of right-hand-sides (number of columns of B). NRHS >= 0.

Parameters

in,out	*A*	double *. A is a pointer to a regular matrix of dimension N-by-LDA. On exit, if return value is Success, the matrix A is overwriten by the factors L and U. The unit diagonal elements of L are not stored.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).
out	*IPIV*	int *. ipiv is a pointer to an array of dimesion at least max(1, min (M, N)). ipiv(i) = j, 1 <= i <= min(M, N) implies that rows i and j have been interchanged.
in,out	*B*	double *. B is a pointer to a matrix of dimension N by NRHS, which stores the right-hand-sides of the systems of linear equations. (row-major order). On exit, if return value is Success, the matrix B is overwriten by the solution matrix X.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, NRHS).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

ddss_tile
ddss_flat2tiled
ddss_tiled2flat

Definition at line 91 of file kdtpgesv.c.

References ddss_dflat2tiled(), ddss_dtiled2flat(), and ddss_tile_size().

Referenced by ddss_dtpgesv().

```
95 {
96
97     // Local variables
98     int i;
99     int nt, nrhst;
100     int ni, mi, nni, ki, nrhsi;
101     int tile_size_n;
102     int tile_size_m;
103     int tile_size_nn;
104     int tile_size_nrhs;
105     int tile_size_k;
106     int local_ipiv_size;
107     int current_position;
108     enum LASS_RETURN ret;
109     int *ipiv_local;
110
111     // Number of tiles
112     if ( N % TILE_SIZE == 0 )
113     {
114         nt = N / TILE_SIZE;
115     }
116     else
```

```
117     {
118         nt = ( N / TILE_SIZE ) + 1;
119     }
120
121     if ( NRHS % TILE_SIZE == 0 )
122     {
123         nrhst = NRHS / TILE_SIZE;
124     }
125     else
126     {
127         nrhst = ( NRHS / TILE_SIZE ) + 1;
128     }
129
130     /***************************
131     --Tile matrices allocation--
132     ***************************/
133
134     double ( *TILE_A )[nt][TILE_SIZE * TILE_SIZE] = malloc ( nt * nt *
135         TILE_SIZE * TILE_SIZE * sizeof( double ) );
136
137     if ( TILE_A == NULL )
138     {
139         fprintf( stderr, "Failure in kdtpgesv for matrix TILE_A\n" );
140         return NoSuccess;
141     }
142
143     double ( *TILE_B )[nrhst][TILE_SIZE * TILE_SIZE] = malloc(
144              nt * nrhst * TILE_SIZE * TILE_SIZE * sizeof( double ) );
145
146     if ( TILE_B == NULL )
147     {
148         fprintf( stderr, "Failure in kdtpgesv for matrix TILE_B\n" );
149         return NoSuccess;
150     }
151
152     ipiv_local = ( int* ) malloc( N * sizeof( int ) );
153
154     if ( ipiv_local == NULL )
155     {
156         fprintf( stderr, "Failure in kdtpgesv for ipiv_local\n" );
157         return NoSuccess;
158     }
159
160     /***********************************************
161     // --From flat data layout to tiled data layout--
162     ***********************************************/
163
164     // From flat matrix A to tile matrix TILE_A
165     ddss_dflat2tiled( N, N, A, LDA, nt, nt, TILE_A );
166
167     // From flat matrix B to tile matrix TILE_B
168     ddss_dflat2tiled( N, NRHS, B, LDB, nt, nrhst, TILE_B );
169
170     /****************
171     --DGETRF tile--
172     ****************/
173
174     current_position = 0;
175     i = 0;
176     for ( ki = 0; ki < nt; ki++ )
177     {
178         tile_size_k = ddss_tile_size( N, ki );
179
180         #pragma oss task inout( TILE_A[ki][ki] ) \
181             inout( ipiv_local ) \
182             inout( current_position ) \
183             out( local_ipiv_size ) \
184             shared( TILE_A ) \
185             firstprivate( ki, i ) \
186             label( dgetrf )
187         {
188             LAPACKE_dgetrf( CblasRowMajor,
189                             tile_size_k, tile_size_k,
190                             TILE_A[ki][ki], tile_size_k,
191                             ipiv_local + current_position );
192
193             // Update the global ipiv array
194             local_ipiv_size = tile_size_k;
195             for ( i = 0; i < local_ipiv_size; i++ )
196             {
197                 ( IPIV + current_position )[i] =
198                         ( ipiv_local + current_position )[i] + current_position;
199             }
200             current_position += local_ipiv_size;
201         }
202         for ( mi = ki + 1; mi < nt; mi++ )
203         {
```

```
204             tile_size_m = ddss_tile_size( N, mi );
205
206             #pragma oss task in( TILE_A[ki][ki] ) \
207                 inout(TILE_A[mi][ki]) \
208                 shared( TILE_A ) \
209                 firstprivate( mi, ki ) \
210                 label( dtrsm_below )
211             cblas_dtrsm( CblasRowMajor,
212                          ( CBLAS_SIDE ) Right, ( CBLAS_UPLO ) Upper,
213                          ( CBLAS_TRANSPOSE ) NoTrans, ( CBLAS_DIAG ) NonUnit,
214                          tile_size_m, tile_size_k,
215                          1.0, TILE_A[ki][ki], tile_size_k,
216                               TILE_A[mi][ki], tile_size_k );
217         }
218         for ( ni = 0; ni < ki; ni++ )
219         {
220             tile_size_n = ddss_tile_size( N, ni );
221             // Swap the tiles to the left of A[ki][ki]
222             #pragma oss task in ( ipiv_local ) \
223                 in( current_position ) \
224                 in( local_ipiv_size ) \
225                 inout( TILE_A[ki][ni] ) \
226                 shared( TILE_A ) \
227                 firstprivate( ki, ni ) \
228                 label( dlaswp_left )
229             LAPACKE_dlaswp( CblasRowMajor,
230                             tile_size_n, TILE_A[ki][ni], tile_size_n,
231                             1, local_ipiv_size,
232                             ipiv_local + current_position - local_ipiv_size,
233                             1 );
234
235         }
236         for ( nrhsi = 0; nrhsi < nrhst; nrhsi++ )
237         {
238             tile_size_nrhs = ddss_tile_size( NRHS, nrhsi );
239             // Swap on the B[ki][nrhsi] tile
240             #pragma oss task in ( ipiv_local ) \
241                 in( current_position ) \
242                 in( local_ipiv_size ) \
243                 inout( TILE_B[ki][nrhsi] ) \
244                 shared( TILE_B ) \
245                 firstprivate( ki, nrhsi ) \
246                 label( dlaswp_on_B )
247             LAPACKE_dlaswp( CblasRowMajor,
248                             tile_size_nrhs, TILE_B[ki][nrhsi], tile_size_nrhs,
249                             1, local_ipiv_size,
250                             ipiv_local + current_position - local_ipiv_size,
251                             1 );
252
253         }
254         for ( ni = ki + 1; ni < nt; ni++ )
255         {
256             tile_size_n = ddss_tile_size( N, ni );
257             // Swap the tiles to the right of A[ki][ki]
258             #pragma oss task in ( ipiv_local ) \
259                 in( current_position ) \
260                 in( local_ipiv_size ) \
261                 inout( TILE_A[ki][ni] ) \
262                 shared( TILE_A ) \
263                 firstprivate( ki, ni ) \
264                 label( dlaswp_right )
265             LAPACKE_dlaswp( CblasRowMajor,
266                             tile_size_n, TILE_A[ki][ni], tile_size_n,
267                             1, local_ipiv_size,
268                             ipiv_local + current_position - local_ipiv_size,
269                             1 );
270
271             #pragma oss task in( TILE_A[ki][ki] ) \
272                 inout(TILE_A[ki][ni]) \
273                 shared( TILE_A ) \
274                 firstprivate( ni, ki ) \
275                 label( dtrsm_right )
276             cblas_dtrsm( CblasRowMajor,
277                          ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Lower,
278                          ( CBLAS_TRANSPOSE ) NoTrans, ( CBLAS_DIAG ) Unit,
279                          tile_size_k, tile_size_n,
280                          1.0, TILE_A[ki][ki], tile_size_k,
281                               TILE_A[ki][ni], tile_size_n );
282
283             for ( nni = ki + 1; nni < nt; nni++ )
284             {
285                 tile_size_nn = ddss_tile_size( N, nni );
286
287                 #pragma oss task in( TILE_A[nni][ki] ) \
288                     in( TILE_A[ki][ni] ) \
289                     inout( TILE_A[nni][ni] ) \
290                     shared( TILE_A ) \
```

```
291                     firstprivate( ni, nni, ki ) \
292                     label( dgemm )
293                 cblas_dgemm( CblasRowMajor,
294                              ( CBLAS_TRANSPOSE ) NoTrans,
295                              ( CBLAS_TRANSPOSE ) NoTrans,
296                              tile_size_nn,
297                              tile_size_n,
298                              TILE_SIZE,
299                              -1.0, TILE_A[nni][ki], tile_size_k,
300                                    TILE_A[ki][ni],  tile_size_n,
301                               1.0, TILE_A[nni][ni], tile_size_n );
302             }
303         }
304     }
305
306     // --From tile data layout to flat data layout--
307     ddss_dtiled2flat( N, N, A, LDA, nt, nt, TILE_A );
308
309     // Triandular solve
310     // --SIDE = Left & UPLO = Lower & TRANS_A = NoTrans & Unit--
311     for ( ki = 0; ki < nt; ki++ )
312     {
313         tile_size_k = ddss_tile_size( N, ki );
314
315         for ( ni = 0; ni < nrhst; ni++ )
316         {
317             tile_size_n = ddss_tile_size( NRHS, ni );
318
319             #pragma oss task in( TILE_A[ki][ki] ) \
320                 inout(  TILE_B[ki][ni] ) \
321                 shared( TILE_A, TILE_B ) \
322                 firstprivate( ki, ni ) \
323                 label( dtrsm_dtrsm1 )
324             cblas_dtrsm( CblasRowMajor,
325                          ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Lower,
326                          ( CBLAS_TRANSPOSE ) NoTrans,
327                          ( CBLAS_DIAG ) Unit,
328                          tile_size_k,
329                          tile_size_n,
330                          1.0, TILE_A[ki][ki], tile_size_k,
331                               TILE_B[ki][ni], tile_size_n );
332         }
333         for ( nni = 0; nni < nrhst; nni++ )
334         {
335             tile_size_nn = ddss_tile_size( NRHS, nni );
336
337             for ( mi = ki + 1; mi < nt; mi++ )
338             {
339
340                 #pragma oss task in( TILE_A[mi][ki] ) \
341                     in( TILE_B[ki][nni] ) \
342                     inout( TILE_B[mi][nni] ) \
343                     shared( TILE_A, TILE_B ) \
344                     firstprivate( ki, mi, nni ) \
345                     label( dtrsm_dgemm1 )
346                 cblas_dgemm( CblasRowMajor,
347                              CblasNoTrans, CblasNoTrans,
348                              tile_size_k,
349                              tile_size_nn,
350                              tile_size_k,
351                              -1.0,  TILE_A[mi][ki],  tile_size_k,
352                                     TILE_B[ki][nni], tile_size_nn,
353                               1.0, TILE_B[mi][nni], tile_size_nn );
354             }
355         }
356     }
357
358     // Triandular solve
359     // --SIDE = Left & UPLO = Upper & TRANS_A = NoTrans & NonUnit--
360     for ( ki = nt - 1; ki >= 0; ki-- )
361     {
362         tile_size_k = ddss_tile_size( N, ki );
363
364         for ( ni = 0; ni < nrhst; ni++ )
365         {
366             tile_size_n = ddss_tile_size( NRHS, ni );
367
368             #pragma oss task in( TILE_A[ki][ki] ) \
369                 inout( TILE_B[ki][ni] ) \
370                 shared( TILE_A, TILE_B ) \
371                 firstprivate( ki, ni ) \
372                 label( dtrsmi_dtrsm2 )
373             cblas_dtrsm( CblasRowMajor,
374                          ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Upper,
375                          ( CBLAS_TRANSPOSE ) NoTrans,
376                          ( CBLAS_DIAG ) NonUnit,
377                          tile_size_k,
```

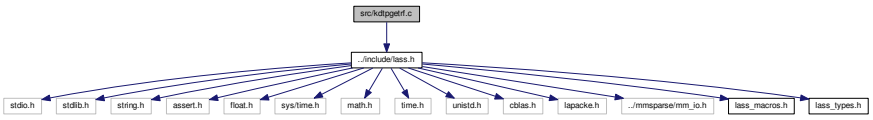
```
378                         tile_size_n,
379                         1.0, TILE_A[ki][ki], tile_size_k,
380                              TILE_B[ki][ni], tile_size_n );
381         }
382
383         for ( mi = ki - 1; mi >= 0; mi-- )
384         {
385             tile_size_m = ddss_tile_size( N, mi );
386
387             for ( nni = 0; nni < nrhst; nni++ )
388             {
389                 tile_size_nn = ddss_tile_size( NRHS, nni );
390
391                 #pragma oss task in( TILE_A[mi][ki] ) \
392                     in( TILE_B[ki][nni] ) \
393                     inout( TILE_B[mi][nni] ) \
394                     shared( TILE_A, TILE_B ) \
395                     firstprivate( ki, mi, nni ) \
396                     label( dtrsm_dgemm2 )
397                 cblas_dgemm( CblasRowMajor,
398                              CblasNoTrans, CblasNoTrans,
399                              tile_size_m,
400                              tile_size_nn,
401                              tile_size_k,
402                              -1.0, TILE_A[mi][ki],  tile_size_k,
403                                    TILE_B[ki][nni], tile_size_nn,
404                               1.0, TILE_B[mi][nni], tile_size_nn );
405             }
406         }
407     }
408
409     /*****************************************
410     --From tiled data layout to flat data layout--
411     *****************************************/
412     ddss_dtiled2flat( N, NRHS, B, LDB, nt, nrhst, TILE_B );
413
414     // --Free--
415     free( TILE_A );
416     free( TILE_B );
417     free( ipiv_local );
418
419     return Success;
420
421 }
```

2.29 src/kdtpgetrf.c File Reference

LASs-DDSs kdtpgetrf routine.

```
#include "../include/lass.h"
```

Include dependency graph for kdtpgetrf.c:



Functions

- enum LASS_RETURN kdtpgetrf (int M, int N, double *A, int LDA, int *IPIV)

2.29.1 Detailed Description

LASs-DDSs kdtpgetrf routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es
Boro Sofranac boro.sofranac@bsc.es

Date

2018-04-08 2018-05-08

2.29.2 Function Documentation

2.29.2.1 enum LASS_RETURN kdtpgetrf (int *M*, int *N*, double * *A*, int *LDA*, int * *IPIV*)

Performs the LU factorization with tiled pivoting (row interchanges) of a general M-by-N matrix A:

```
A = P * L * U
```

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

Parameters

in	*M*	int. M specifies the number of rows of the matrix A. M >= 0.
in	*N*	int. N specifies the number of columns of the matrix A. N >= 0.
in,out	*A*	double *. A is a pointer to a regular matrix of dimension M-by-N. On exit, if return value is Success, the matrix A is overwriten by the factors L and U. The unit diagonal elements of L are not stored.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). LDA must be at least max(1, N).
out	*IPIV*	int *. ipiv is a pointer to an array of dimesion at least max(1, min (M, N)). ipiv(i) = j, 1 <= i <= min(M, N) implies that rows i and j have been interchanged.

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

ddss_tile
ddss_flat2tiled
ddss_tiled2flat

Definition at line 80 of file kdtpgetrf.c.

References ddss_dflat2tiled(), ddss_dtiled2flat(), and ddss_tile_size().

Referenced by ddss_dtpgetrf().

```
81 {
82
83     // Local variables
84     int i;
85     int mt, nt;
86     int mi, mmi, ki, ni;
87     int tile_size_m;
88     int tile_size_mm;
89     int tile_size_n;
90     int tile_size_km;
91     int tile_size_kn;
92     int local_ipiv_size;
93     int current_position;
94     enum LASS_RETURN ret;
95     int *ipiv_local;
96
97     // Number of tiles
98     if ( M % TILE_SIZE == 0 )
99     {
100         mt = M / TILE_SIZE;
101     }
102     else
103     {
104         mt = ( M / TILE_SIZE ) + 1;
105     }
106
107     if ( N % TILE_SIZE == 0 )
108     {
109         nt = N / TILE_SIZE;
110     }
111     else
112     {
113         nt = ( N / TILE_SIZE ) + 1;
114     }
115
116     /***************************
117     --Tile A matrix allocation--
118     ***************************/
119
120     double ( *TILE_A )[nt][TILE_SIZE * TILE_SIZE] = malloc ( mt * nt *
121         TILE_SIZE * TILE_SIZE * sizeof( double ) );
122
123     if ( TILE_A == NULL )
124     {
125         fprintf( stderr, "Failure in kdnpgetrf for matrix TILE_A\n" );
126         return NoSuccess;
127     }
128
129     ipiv_local = ( int* ) malloc( MIN( M, N ) * sizeof( int ) );
130
131     if ( ipiv_local == NULL )
132     {
133         fprintf( stderr, "Failure in kdnpgetrf for ipiv_local\n" );
134         return NoSuccess;
135     }
136
137     /***********************************************
138     // --From flat data layout to tiled data layout--
139     ***********************************************/
140
141     ddss_dflat2tiled( M, N, A, LDA, mt, nt, TILE_A );
142
143     /****************
144     --DGETRF tile--
145     ****************/
146
147     current_position = 0;
148     i = 0;
149     for ( ki = 0; ki < MIN( mt, nt ); ki++ )
150     {
151         tile_size_km = ddss_tile_size( M, ki );
152         tile_size_kn = ddss_tile_size( N, ki );
153
154         #pragma oss task inout( TILE_A[ki][ki] ) \
155             inout( ipiv_local ) \
156             inout( current_position ) \
157             out( local_ipiv_size ) \
158             shared( TILE_A ) \
```

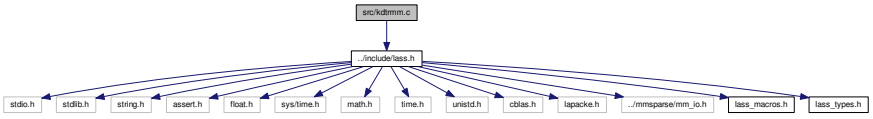
```
159             firstprivate( ki, i ) \
160             label( dgetrf )
161         {
162             LAPACKE_dgetrf( CblasRowMajor,
163                             tile_size_km, tile_size_kn,
164                             TILE_A[ki][ki], tile_size_kn,
165                             ipiv_local + current_position );
166
167             // Update the global ipiv array
168             local_ipiv_size = MIN( tile_size_km, tile_size_kn );
169             for ( i = 0; i < local_ipiv_size; i++ )
170             {
171                 ( IPIV + current_position )[i] =
172                         ( ipiv_local + current_position )[i] + current_position;
173             }
174             current_position += local_ipiv_size;
175         }
176         for ( mi = ki + 1; mi < mt; mi++ )
177         {
178             tile_size_m = ddss_tile_size( M, mi );
179
180             #pragma oss task in( TILE_A[ki][ki] ) \
181                 inout( TILE_A[mi][ki] ) \
182                 shared( TILE_A ) \
183                 firstprivate( mi, ki ) \
184                 label( dtrsm_below )
185             cblas_dtrsm( CblasRowMajor,
186                          ( CBLAS_SIDE ) Right, ( CBLAS_UPLO ) Upper,
187                          ( CBLAS_TRANSPOSE ) NoTrans, ( CBLAS_DIAG ) NonUnit,
188                          tile_size_m, tile_size_kn,
189                          1.0, TILE_A[ki][ki], tile_size_kn,
190                               TILE_A[mi][ki], tile_size_kn );
191         }
192         for ( ni = 0; ni < ki; ni++ )
193         {
194             tile_size_n = ddss_tile_size( N, ni );
195             // Swap the tiles to the left of A[ki][ki]
196             #pragma oss task in( ipiv_local ) \
197                 in( current_position ) \
198                 in( local_ipiv_size ) \
199                 inout( TILE_A[ki][ni] ) \
200                 shared( TILE_A ) \
201                 firstprivate( ki, ni ) \
202                 label( dlaswp_left )
203             LAPACKE_dlaswp( CblasRowMajor,
204                             tile_size_n, TILE_A[ki][ni], tile_size_n,
205                             1, local_ipiv_size,
206                             ipiv_local + current_position - local_ipiv_size,
207                             1 );
208
209         }
210         for ( ni = ki + 1; ni < nt; ni++ )
211         {
212             tile_size_n = ddss_tile_size( N, ni );
213             // Swap the tiles to the right of A[ki][ki]
214             #pragma oss task in( ipiv_local ) \
215                 in( current_position ) \
216                 in( local_ipiv_size ) \
217                 inout( TILE_A[ki][ni] ) \
218                 shared( TILE_A ) \
219                 firstprivate( ki, ni ) \
220                 label( dlaswp_right )
221             LAPACKE_dlaswp( CblasRowMajor,
222                             tile_size_n, TILE_A[ki][ni], tile_size_n,
223                             1, local_ipiv_size,
224                             ipiv_local + current_position - local_ipiv_size,
225                             1 );
226
227             #pragma oss task in( TILE_A[ki][ki] ) \
228                 inout( TILE_A[ki][ni] ) \
229                 shared( TILE_A ) \
230                 firstprivate( ni, ki ) \
231                 label( dtrsm_right )
232             cblas_dtrsm( CblasRowMajor,
233                          ( CBLAS_SIDE ) Left, ( CBLAS_UPLO ) Lower,
234                          ( CBLAS_TRANSPOSE ) NoTrans, ( CBLAS_DIAG ) Unit,
235                          tile_size_km, tile_size_n,
236                          1.0, TILE_A[ki][ki], tile_size_kn,
237                               TILE_A[ki][ni], tile_size_n );
238
239             for ( mmi = ki + 1; mmi < mt; mmi++ )
240             {
241                 tile_size_mm = ddss_tile_size( M, mmi );
242
243                 #pragma oss task in( TILE_A[mmi][ki] ) \
244                     in( TILE_A[ki][ni] ) \
245                     inout( TILE_A[mmi][ni] ) \
```

```
246                     shared( TILE_A ) \
247                     firstprivate( ni, mmi, ki ) \
248                     label( dgemm )
249                 cblas_dgemm( CblasRowMajor,
250                              ( CBLAS_TRANSPOSE ) NoTrans,
251                              ( CBLAS_TRANSPOSE ) NoTrans,
252                              tile_size_mm,
253                              tile_size_n,
254                              TILE_SIZE,
255                              -1.0, TILE_A[mmi][ki], tile_size_kn,
256                                    TILE_A[ki][ni],  tile_size_n,
257                               1.0, TILE_A[mmi][ni], tile_size_n );
258             }
259         }
260     }
261
262     // --From tile data layout to flat data layout--
263     ddss_dtiled2flat( M, N, A, LDA, mt, nt, TILE_A );
264
265     // --Free--
266     free( TILE_A );
267     free( ipiv_local );
268
269     return Success;
270
271 }
```

2.30 src/kdtrmm.c File Reference

LASs-DDSs kdtrmm routine.

`#include "../include/lass.h"`
Include dependency graph for kdtrmm.c:



Functions

- enum LASS_RETURN kdtrmm (enum DDSS_SIDE SIDE, enum DDSS_UPLO UPLO, enum DDSS_TRANS TRANS_A, enum DDSS_DIAG DIAG, int M, int N, const double ALPHA, double ∗A, int LDA, double ∗B, int LDB)

2.30.1 Detailed Description

LASs-DDSs kdtrmm routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es
Boro Sofranac boro.sofranac@bsc.es

Date

2018-03-19

2.30.2 Function Documentation

2.30.2.1 enum LASS_RETURN kdtrmm (enum DDSS_SIDE *SIDE*, enum DDSS_UPLO *UPLO*, enum DDSS_TRANS *TRANS_A*, enum DDSS_DIAG *DIAG*, int *M*, int *N*, const double *ALPHA*, double * *A*, int *LDA*, double * *B*, int *LDB*)

Performs one of the matrix-matrix operations:

```
B = ALPHA * op( A ) * B, or B = ALPHA * B * op( A )
```

where op(A) is one of:

```
op( A ) = A      or
op( A ) = A**T
```

ALPHA is a scalar, B is a M by N matrix and A is a unit, or non-unit, upper or lower triangular matrix.

Parameters

in	*SIDE*	enum DDSS_SIDE. SIDE specifies the position of the triangular A matrix in the operations: • Left: B = ALPHA * op(A) * B. • Right: B = ALPHA * B * op(A).
in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form in which A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.
in	*TRANS_A*	enum DDSS_TRANS. TRANS_A specifies the form of op(A) to be used: • NoTrans: op(A) = A. • Trans: op(A) = A**T.
in	*DIAG*	enum DDSS_DIAG. DIAG specifies whether or not A is unit triangular as follows: • Unit: A is assumed to be unit triangular. • NonUnit: A is not assumed to be unit triangular.
in	*M*	int. M specifies the number of rows of B. M must be at least zero.
in	*N*	int. N specifies the number of columns of B. N must be at least zero.
in	*ALPHA*	double. ALPHA specifies the scalar alpha.
in	*A*	double *. A is a pointer to a matrix of dimension K by K, where K is M when SIDE = Left and is N otherwise. When UPLO = Uppper the strictly lower triangular part of A is not referenced and when UPLO = Lower the strictly upper triangular part of A is not referenced. Note that when DIAG = Unit, the diagonal elements of A are not referenced either, but are assumed to be unity.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). When SIDE = Left then LDA must be at least max(1, M), otherwise LDA must be at least max(1, N).
in,out	*B*	double *. B is a pointer to a matrix of dimension M by N. On exit the matrix B is overwritten by the transformed matrix.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, N).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

ddss_tile
ddss_flat2tiled
ddss_dsymflat2tiled
ddss_tiled2flat

Definition at line 123 of file kdtrmm.c.

References ddss_dflat2tiled(), ddss_dsymflat2tiled(), ddss_dtiled2flat(), and ddss_tile_size().

Referenced by ddss_dtrmm().

```
128 {
129
130     // Local variables
131     int k;
132     int mt, nt;
133     int ki, ni, mi, nni, mmi;
134     int Am, An;
135     int Bm, Bn;
136     int tile_size_m;
137     int tile_size_n;
138     int tile_size_k;
139
140     // Number of tiles
141     if ( M % TILE_SIZE == 0 )
142     {
143         mt = M / TILE_SIZE;
144     }
145     else
146     {
147         mt = ( M / TILE_SIZE ) + 1;
148     }
149
150     if ( N % TILE_SIZE == 0 )
151     {
152         nt = N / TILE_SIZE;
153     }
154     else
155     {
156         nt = ( N / TILE_SIZE ) + 1;
157     }
158
159     /***************************
160      --Tile matrices declaration--
161      ***************************/
162
163     if ( SIDE == Left )
164     {
165         Am = An = mt;
166         k = M;
167     }
168     else
169     {
170         Am = An = nt;
171         k = N;
172     }
173
174     Bm = mt;
175     Bn = nt;
176
177     /***************************
178      --Tile matrices allocation--
179      ***************************/
180
181     double ( *TILE_A )[An][TILE_SIZE * TILE_SIZE] = malloc(
182               Am * An * TILE_SIZE * TILE_SIZE * sizeof( double ) );
183
184     if ( TILE_A == NULL )
```

```
185     {
186         fprintf( stderr, "Failure in ddss_dtile_alloc for matrix TILE_A\n" );
187         return NoSuccess;
188     }
189 
190     double ( *TILE_B )[Bn][TILE_SIZE * TILE_SIZE] = malloc(
191             Bm * Bn * TILE_SIZE * TILE_SIZE * sizeof( double ) );
192 
193     if ( TILE_B == NULL )
194     {
195         fprintf( stderr, "Failure in ddss_dtile_alloc for matrix TILE_B\n" );
196         return NoSuccess;
197     }
198 
199     /*******************************************
200      --From flat data layout to tiled data layout--
201      *******************************************/
202 
203     // From flat matrix A to tile matrix TILE_A
204     ddss_dsymflat2tiled( k, k, A, LDA, Am, An, TILE_A, UPLO );
205 
206     // From flat matrix B to tile matrix TILE_B
207     ddss_dflat2tiled( M, N, B, LDB, Bm, Bn, TILE_B );
208 
209     /*************
210      --DTRMM tile--
211      *************/
212 
213     if ( SIDE == Left )
214     {
215         if ( UPLO == Upper )
216         {
217             // --SIDE = Left & UPLO = Upper & TRANS_A = NoTrans--
218             if ( TRANS_A == NoTrans )
219             {
220                 for ( ki = 0; ki < mt; ki++ )
221                 {
222                     tile_size_k = ddss_tile_size( M, ki );
223 
224                     for ( ni = 0; ni < nt; ni++ )
225                     {
226                         tile_size_n = ddss_tile_size( N, ni );
227 
228                         #pragma oss task in( TILE_A[ki][ki] ) \
229                                 inout( TILE_B[ki][ni] ) \
230                                 shared( TILE_A, TILE_B ) \
231                                 firstprivate( ki, ni ) \
232                                 label( dtrmm )
233                         cblas_dtrmm( CblasRowMajor,
234                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
235                                      ( CBLAS_TRANSPOSE ) TRANS_A,
236                                      ( CBLAS_DIAG ) DIAG,
237                                      tile_size_k,
238                                      tile_size_n,
239                                      ALPHA, TILE_A[ki][ki], tile_size_k,
240                                             TILE_B[ki][ni], tile_size_n );
241 
242                         for ( mi = ki + 1; mi < mt; mi++ )
243                         {
244                             tile_size_m = ddss_tile_size( M, mi );
245 
246                             #pragma oss task in( TILE_A[ki][mi] ) \
247                                     in( TILE_B[mi][ni] ) \
248                                     inout( TILE_B[ki][ni] ) \
249                                     shared( TILE_A, TILE_B ) \
250                                     firstprivate( ki, mi, ni ) \
251                                     label( dgemm )
252                             cblas_dgemm( CblasRowMajor,
253                                          CblasNoTrans, CblasNoTrans,
254                                          tile_size_k,
255                                          tile_size_n,
256                                          tile_size_m,
257                                          ALPHA, TILE_A[ki][mi],  tile_size_m,
258                                                 TILE_B[mi][ni],  tile_size_n,
259                                          1.0,   TILE_B[ki][ni],  tile_size_n );
260                         }
261                     }
262                 }
263             }
264             // --SIDE = Left & UPLO = Upper & TRANS_A = Trans--
265             else
266             {
267                 for ( ki = mt-1; ki > -1; ki-- )
268                 {
269                     tile_size_k = ddss_tile_size( M, ki );
270 
271                     for ( ni = 0; ni < nt; ni++ )
```

```
272                     {
273                         tile_size_n = ddss_tile_size( N, ni );
274
275                         #pragma oss task in( TILE_A[ki][ki] ) \
276                                 inout( TILE_B[ki][ni] ) \
277                                 shared( TILE_A, TILE_B ) \
278                                 firstprivate( ki, ni ) \
279                                 label( dtrmm )
280                         cblas_dtrmm( CblasRowMajor,
281                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
282                                      ( CBLAS_TRANSPOSE ) TRANS_A,
283                                      ( CBLAS_DIAG ) DIAG,
284                                      tile_size_k,
285                                      tile_size_n,
286                                      ALPHA, TILE_A[ki][ki], tile_size_k,
287                                             TILE_B[ki][ni], tile_size_n );
288
289                         for ( mi = 0; mi < ki; mi++ )
290                         {
291                             tile_size_m = ddss_tile_size( M, mi );
292
293                             #pragma oss task in( TILE_A[mi][ki] ) \
294                                     in( TILE_B[mi][ni] ) \
295                                     inout( TILE_B[ki][ni] ) \
296                                     shared( TILE_A, TILE_B ) \
297                                     firstprivate( ki, mi, ni ) \
298                                     label( dgemm )
299                             cblas_dgemm( CblasRowMajor,
300                                          CblasTrans, CblasNoTrans,
301                                          tile_size_k,
302                                          tile_size_n,
303                                          tile_size_m,
304                                          ALPHA, TILE_A[mi][ki],  tile_size_k,
305                                                 TILE_B[mi][ni],  tile_size_n,
306                                          1.0,   TILE_B[ki][ni],  tile_size_n );
307                         }
308                     }
309                 }
310
311             }
312         }
313         // --SIDE = Left & UPLO = Lower & TRANS_A = NoTrans--
314         else
315         {
316             if ( TRANS_A == NoTrans )
317             {
318                 for ( ki = mt-1; ki > -1; ki-- )
319                 {
320                     tile_size_k = ddss_tile_size( M, ki );
321
322                     for ( ni = 0; ni < nt; ni++ )
323                     {
324                         tile_size_n = ddss_tile_size( N, ni );
325
326                         #pragma oss task in( TILE_A[ki][ki] ) \
327                                 inout( TILE_B[ki][ni] ) \
328                                 shared( TILE_A, TILE_B ) \
329                                 firstprivate( ki, ni ) \
330                                 label( dtrmm )
331                         cblas_dtrmm( CblasRowMajor,
332                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
333                                      ( CBLAS_TRANSPOSE ) TRANS_A,
334                                      ( CBLAS_DIAG ) DIAG,
335                                      tile_size_k,
336                                      tile_size_n,
337                                      ALPHA, TILE_A[ki][ki], tile_size_k,
338                                             TILE_B[ki][ni], tile_size_n );
339
340                         for ( mi = 0; mi < ki; mi++ )
341                         {
342                             tile_size_m = ddss_tile_size( M, mi );
343
344                             #pragma oss task in( TILE_A[ki][mi] ) \
345                                     in( TILE_B[mi][ni] ) \
346                                     inout( TILE_B[ki][ni] ) \
347                                     shared( TILE_A, TILE_B ) \
348                                     firstprivate( ki, mi, ni ) \
349                                     label( dgemm )
350                             cblas_dgemm( CblasRowMajor,
351                                          CblasNoTrans, CblasNoTrans,
352                                          tile_size_k,
353                                          tile_size_n,
354                                          tile_size_m,
355                                          ALPHA, TILE_A[ki][mi],  tile_size_m,
356                                                 TILE_B[mi][ni],  tile_size_n,
357                                          1.0,   TILE_B[ki][ni],  tile_size_n );
358                         }
```

```
359                     }
360                 }
361
362             }
363             // --SIDE = Left & UPLO = Lower & TRANS_A = Trans--
364             else
365             {
366                 for ( ki = 0; ki < mt; ki++ )
367                 {
368                     tile_size_k = ddss_tile_size( M, ki );
369
370                     for ( ni = 0; ni < nt; ni++ )
371                     {
372                         tile_size_n = ddss_tile_size( N, ni );
373
374                         #pragma oss task in( TILE_A[ki][ki] ) \
375                                 inout( TILE_B[ki][ni] ) \
376                                 shared( TILE_A, TILE_B ) \
377                                 firstprivate( ki, ni ) \
378                                 label( dtrmm )
379                         cblas_dtrmm( CblasRowMajor,
380                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
381                                      ( CBLAS_TRANSPOSE ) TRANS_A,
382                                      ( CBLAS_DIAG ) DIAG,
383                                      tile_size_k,
384                                      tile_size_n,
385                                      ALPHA, TILE_A[ki][ki], tile_size_k,
386                                             TILE_B[ki][ni], tile_size_n );
387
388                         for ( mi = ki + 1; mi < mt; mi++ )
389                         {
390                             tile_size_m = ddss_tile_size( M, mi );
391
392                             #pragma oss task in( TILE_A[mi][ki] ) \
393                                     in( TILE_B[mi][ni] ) \
394                                     inout( TILE_B[ki][ni] ) \
395                                     shared( TILE_A, TILE_B ) \
396                                     firstprivate( ki, mi, ni ) \
397                                     label( dgemm )
398                             cblas_dgemm( CblasRowMajor,
399                                          CblasTrans, CblasNoTrans,
400                                          tile_size_k,
401                                          tile_size_n,
402                                          tile_size_m,
403                                          ALPHA, TILE_A[mi][ki],  tile_size_k,
404                                                 TILE_B[mi][ni],  tile_size_n,
405                                          1.0,   TILE_B[ki][ni],  tile_size_n );
406                         }
407                     }
408                 }
409             }
410         }
411     }
412     // --SIDE = Right & UPLO = Upper & TRANS_A = NoTrans--
413     else
414     {
415         if ( UPLO == Upper )
416         {
417             if ( TRANS_A == NoTrans )
418             {
419                 for ( ki = nt-1; ki > -1; ki-- )
420                 {
421                     tile_size_k = ddss_tile_size( N, ki );
422
423                     for ( ni = 0; ni < mt; ni++ )
424                     {
425                         tile_size_n = ddss_tile_size( M, ni );
426
427                         #pragma oss task in( TILE_A[ki][ki] ) \
428                                 inout( TILE_B[ni][ki] ) \
429                                 shared( TILE_A, TILE_B ) \
430                                 firstprivate( ki, ni ) \
431                                 label( dtrmm )
432                         cblas_dtrmm( CblasRowMajor,
433                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
434                                      ( CBLAS_TRANSPOSE ) TRANS_A,
435                                      ( CBLAS_DIAG ) DIAG,
436                                      tile_size_n,
437                                      tile_size_k,
438                                      ALPHA, TILE_A[ki][ki], tile_size_k,
439                                             TILE_B[ni][ki], tile_size_k );
440
441                         for ( mi = 0; mi < ki; mi++ )
442                         {
443                             tile_size_m = ddss_tile_size( N, mi );
444
445                             #pragma oss task in( TILE_B[ni][mi] ) \
```

```
446                                     in( TILE_A[mi][ki] ) \
447                                     inout( TILE_B[ni][ki] ) \
448                                     shared( TILE_A, TILE_B ) \
449                                     firstprivate( ki, mi, ni ) \
450                                     label( dgemm )
451                                 cblas_dgemm( CblasRowMajor,
452                                              CblasNoTrans, CblasNoTrans,
453                                              tile_size_n,
454                                              tile_size_k,
455                                              tile_size_m,
456                                              ALPHA, TILE_B[ni][mi],  tile_size_m,
457                                                     TILE_A[mi][ki],  tile_size_k,
458                                              1.0,   TILE_B[ni][ki],  tile_size_k );
459                             }
460                         }
461                     }
462                 }
463                 // --SIDE = Right & UPLO = Upper & TRANS_A = Trans--
464                 else
465                 {
466                     for ( ki = 0; ki < nt; ki++ )
467                     {
468                         tile_size_k = ddss_tile_size( N, ki );
469 
470                         for ( ni = 0; ni < mt; ni++ )
471                         {
472                             tile_size_n = ddss_tile_size( M, ni );
473 
474                             #pragma oss task in( TILE_A[ki][ki] ) \
475                                     inout( TILE_B[ni][ki] ) \
476                                     shared( TILE_A, TILE_B ) \
477                                     firstprivate( ki, ni ) \
478                                     label( dtrmm )
479                             cblas_dtrmm( CblasRowMajor,
480                                          ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
481                                          ( CBLAS_TRANSPOSE ) TRANS_A,
482                                          ( CBLAS_DIAG ) DIAG,
483                                          tile_size_n,
484                                          tile_size_k,
485                                          ALPHA, TILE_A[ki][ki], tile_size_k,
486                                                 TILE_B[ni][ki], tile_size_k );
487 
488                             for ( mi = ki+1; mi < nt; mi++ )
489                             {
490                                 tile_size_m = ddss_tile_size( N, mi );
491 
492                                 #pragma oss task in( TILE_B[ni][mi] ) \
493                                         in( TILE_A[ki][mi] ) \
494                                         inout( TILE_B[ni][ki] ) \
495                                         shared( TILE_A, TILE_B ) \
496                                         firstprivate( ki, mi, ni ) \
497                                         label( dgemm )
498                                 cblas_dgemm( CblasRowMajor,
499                                              CblasNoTrans, CblasTrans,
500                                              tile_size_n,
501                                              tile_size_k,
502                                              tile_size_m,
503                                              ALPHA, TILE_B[ni][mi],  tile_size_m,
504                                                     TILE_A[ki][mi],  tile_size_m,
505                                              1.0,   TILE_B[ni][ki],  tile_size_k );
506                             }
507                         }
508                     }
509                 }
510             }
511             // --SIDE = Right & UPLO = Lower & TRANS_A = NoTrans--
512             else
513             {
514                 if ( TRANS_A == NoTrans )
515                 {
516                     for ( ki = 0; ki < nt; ki++ )
517                     {
518                         tile_size_k = ddss_tile_size( N, ki );
519 
520                         for ( ni = 0; ni < mt; ni++ )
521                         {
522                             tile_size_n = ddss_tile_size( M, ni );
523 
524                             #pragma oss task in( TILE_A[ki][ki] ) \
525                                     inout( TILE_B[ni][ki] ) \
526                                     shared( TILE_A, TILE_B ) \
527                                     firstprivate( ki, ni ) \
528                                     label( dtrmm )
529                             cblas_dtrmm( CblasRowMajor,
530                                          ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
531                                          ( CBLAS_TRANSPOSE ) TRANS_A,
532                                          ( CBLAS_DIAG ) DIAG,
```

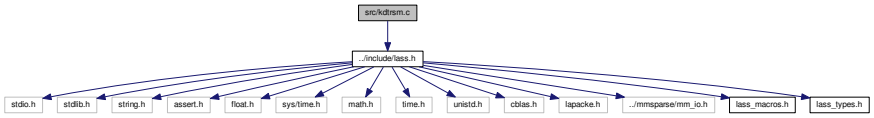
```
533                                     tile_size_n,
534                                     tile_size_k,
535                                     ALPHA, TILE_A[ki][ki], tile_size_k,
536                                            TILE_B[ni][ki], tile_size_k );
537
538                         for ( mi = ki+1; mi < nt; mi++ )
539                         {
540                             tile_size_m = ddss_tile_size( N, mi );
541
542                             #pragma oss task in( TILE_B[ni][mi] ) \
543                                     in( TILE_A[mi][ki] ) \
544                                     inout( TILE_B[ni][ki] ) \
545                                     shared( TILE_A, TILE_B ) \
546                                     firstprivate( ki, mi, ni ) \
547                                     label( dgemm )
548                             cblas_dgemm( CblasRowMajor,
549                                          CblasNoTrans, CblasNoTrans,
550                                          tile_size_n,
551                                          tile_size_k,
552                                          tile_size_m,
553                                          ALPHA, TILE_B[ni][mi],  tile_size_m,
554                                                 TILE_A[mi][ki],  tile_size_k,
555                                          1.0,   TILE_B[ni][ki],  tile_size_k );
556                         }
557                     }
558                 }
559             }
560             // --SIDE = Right & UPLO = Lower & TRANS_A = Trans--
561             else
562             {
563                 for ( ki = nt-1; ki > -1; ki-- )
564                 {
565                     tile_size_k = ddss_tile_size( N, ki );
566
567                     for ( ni = 0; ni < mt; ni++ )
568                     {
569                         tile_size_n = ddss_tile_size( M, ni );
570
571                         #pragma oss task in( TILE_A[ki][ki] ) \
572                                 inout( TILE_B[ni][ki] ) \
573                                 shared( TILE_A, TILE_B ) \
574                                 firstprivate( ki, ni ) \
575                                 label( dtrmm )
576                         cblas_dtrmm( CblasRowMajor,
577                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
578                                      ( CBLAS_TRANSPOSE ) TRANS_A,
579                                      ( CBLAS_DIAG ) DIAG,
580                                      tile_size_n,
581                                      tile_size_k,
582                                      ALPHA, TILE_A[ki][ki], tile_size_k,
583                                             TILE_B[ni][ki], tile_size_k );
584
585                         for ( mi = 0; mi < ki; mi++ )
586                         {
587                             tile_size_m = ddss_tile_size( N, mi );
588
589                             #pragma oss task in( TILE_B[ni][mi] ) \
590                                     in( TILE_A[ki][mi] ) \
591                                     inout( TILE_B[ni][ki] ) \
592                                     shared( TILE_A, TILE_B ) \
593                                     firstprivate( ki, mi, ni ) \
594                                     label( dgemm )
595                             cblas_dgemm( CblasRowMajor,
596                                          CblasNoTrans, CblasTrans,
597                                          tile_size_n,
598                                          tile_size_k,
599                                          tile_size_m,
600                                          ALPHA, TILE_B[ni][mi],  tile_size_m,
601                                                 TILE_A[ki][mi],  tile_size_m,
602                                          1.0,   TILE_B[ni][ki],  tile_size_k );
603                         }
604                     }
605                 }
606             }
607         }
608     }
609
610     /*****************************************
611      --From tiled data layout to flat data layout--
612      *****************************************/
613
614     ddss_dtiled2flat( M, N, B, LDB, Bm, Bn, TILE_B );
615
616     // --Tile matrices free--
617     free( TILE_A );
618     free( TILE_B );
619
```

```
620     return Success;
621
622 }
```

2.31 src/kdtrsm.c File Reference

LASs-DDSs kdtrsm routine.

`#include "../include/lass.h"`
Include dependency graph for kdtrsm.c:



Functions

- enum LASS_RETURN kdtrsm (enum DDSS_SIDE SIDE, enum DDSS_UPLO UPLO, enum DDSS_TRANS TRANS_A, enum DDSS_DIAG DIAG, int M, int N, const double ALPHA, double ∗A, int LDA, double ∗B, int LDB)

2.31.1 Detailed Description

LASs-DDSs kdtrsm routine.

LASs-DDSs is a software package provided by: Barcelona Supercomputing Center - Centro Nacional de Supercomputacion

Author

Pedro Valero-Lara pedro.valero@bsc.es
Boro Sofranac boro.sofranac@bsc.es

Date

2017-12-14

2.31.2 Function Documentation

2.31.2.1 enum LASS_RETURN kdtrsm (enum DDSS_SIDE *SIDE*, enum DDSS_UPLO *UPLO*, enum DDSS_TRANS *TRANS_A*, enum DDSS_DIAG *DIAG*, int *M*, int *N*, const double *ALPHA*, double ∗ *A*, int *LDA*, double ∗ *B*, int *LDB*)

Solves one of the matrix equations:

```
op( A ) * X = ALPHA * B, or X * op( A ) = ALPHA * B
```

where op(A) is one of:

```
op( A ) = A      or
op( A ) = A**T
```

ALPHA is a scalar, X and B are M by N matrices, A is a unit, or non-unit, upper or lower triangular matrix. The matrix X is overwritten on B

Parameters

in	*SIDE*	enum DDSS_SIDE. SIDE specifies the position of the triangular A matrix in the operations: • Left: op(A) ∗ X = ALPHA ∗ B. • Right: X ∗ op(A) = ALPHA ∗ B.
in	*UPLO*	enum DDSS_UPLO. UPLO specifies the form of A is stored: • Lower: Lower triangle of A is stored. The upper traingular part is not referenced. • Upper: Upper triangle of A is stored. The lower triangular part is not referenced.
in	*TRANS↩ _A*	enum DDSS_TRANS. TRANS_A specifies the form of op(A) to be used: • NoTrans: op(A) = A. • Trans: op(A) = A∗∗T.
in	*DIAG*	enum DDSS_DIAG. DIAG specifies whether or not A is unit triangular as follows: • Unit: A is assumed to be unit triangular. • NonUnit: A is not assumed to be unit triangular.
in	*M*	int. M specifies the number of rows of B. M must be at least zero.
in	*N*	int. N specifies the number of columns of B. N must be at least zero.
in	*ALPHA*	double. ALPHA specifies the scalar alpha.
in	*A*	double ∗. A is a pointer to a matrix of dimension K by K, where K is M when SIDE = Left and is N otherwise. When UPLO = Uppper the strictly lower triangular part of A is not referenced and when UPLO = Lower the strictly upper triangular part of A is not referenced. Note that when DIAG = Unit, the diagonal elements of A are not referenced either, but are assumed to be unity.
in	*LDA*	int. LDA specifies the number of columns of A (row-major order). When SIDE = Left then LDA must be at least max(1, M), otherwise LDA must be at least max(1, N).
in,out	*B*	double ∗. B is a pointer to a matrix of dimension M by N. On exit the matrix B is overwritten by the solution matrix X.
in	*LDB*	int. LDB specifies the number of columns of B (row-major order). LDB must be at least max(1, N).

Return values

Success	successful exit
NoSuccess	unsuccessful exit

See also

ddss_tile
ddss_flat2tiled
ddss_dsymflat2tiled
ddss_tiled2flat

Definition at line 123 of file kdtrsm.c.

References ddss_dflat2tiled(), ddss_dsymflat2tiled(), ddss_dtiled2flat(), and ddss_tile_size().

Referenced by ddss_dtrsm().

```
128 {
129
130     // Local variables
131     int k;
132     int mt, nt;
133     int ki, ni, mi, nni, mmi;
134     int Am, An;
135     int Bm, Bn;
136     int tile_size_m;
137     int tile_size_mm;
138     int tile_size_n;
139     int tile_size_nn;
140     int tile_size_k;
141     double alpha;
142
143     // Number of tiles
144     if ( M % TILE_SIZE == 0 )
145     {
146         mt = M / TILE_SIZE;
147     }
148     else
149     {
150         mt = ( M / TILE_SIZE ) + 1;
151     }
152
153     if ( N % TILE_SIZE == 0 )
154     {
155         nt = N / TILE_SIZE;
156     }
157     else
158     {
159         nt = ( N / TILE_SIZE ) + 1;
160     }
161
162     /*****************************
163      --Tile matrices declaration--
164      *****************************/
165
166     if ( SIDE == Left )
167     {
168         Am = An = mt;
169         k = M;
170     }
171     else
172     {
173         Am = An = nt;
174         k = N;
175     }
176
177     Bm = mt;
178     Bn = nt;
179
180     /***************************
181      --Tile matrices allocation--
182      ***************************/
183
184     double ( *TILE_A )[An][TILE_SIZE * TILE_SIZE] = malloc(
185               Am * An * TILE_SIZE * TILE_SIZE * sizeof( double ) );
186
187     if ( TILE_A == NULL )
188     {
189         fprintf( stderr, "Failure in ddss_dtile_alloc for matrix TILE_A\n" );
190         return NoSuccess;
191     }
192
193     double ( *TILE_B )[Bn][TILE_SIZE * TILE_SIZE] = malloc(
194               Bm * Bn * TILE_SIZE * TILE_SIZE * sizeof( double ) );
195
196     if ( TILE_B == NULL )
197     {
198         fprintf( stderr, "Failure in ddss_dtile_alloc for matrix TILE_B\n" );
199         return NoSuccess;
200     }
201
202     /*****************************************
203      --From flat data layout to tiled data layout--
204      *****************************************/
205
206     // From flat matrix A to tile matrix TILE_A
207     ddss_dsymflat2tiled( k, k, A, LDA, Am, An, TILE_A, UPLO );
208
209     // From flat matrix B to tile matrix TILE_B
210     ddss_dflat2tiled( M, N, B, LDB, Bm, Bn, TILE_B );
211
212     /*************
213      --DTRSM tile--
214      *************/
```

```
215
216     if ( SIDE == Left )
217     {
218         if ( UPLO == Upper )
219         {
220             // --SIDE = Left & UPLO = Upper & TRANS_A = NoTrans--
221             if ( TRANS_A == NoTrans )
222             {
223                 for ( ki = mt - 1; ki >= 0; ki-- )
224                 {
225                     tile_size_k = ddss_tile_size( M, ki );
226                     if ( ki == mt - 1 )
227                     {
228                         alpha = ALPHA;
229                     }
230                     else
231                     {
232                         alpha = 1.0;
233                     }
234
235                     for ( ni = 0; ni < nt; ni++ )
236                     {
237                         tile_size_n = ddss_tile_size( N, ni );
238
239                         #pragma oss task in( TILE_A[ki][ki] ) \
240                             inout( TILE_B[ki][ni] ) \
241                             shared( TILE_A, TILE_B ) \
242                             firstprivate( ki, ni ) \
243                             label( dtrsm )
244                         cblas_dtrsm( CblasRowMajor,
245                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
246                                      ( CBLAS_TRANSPOSE ) TRANS_A,
247                                      ( CBLAS_DIAG ) DIAG,
248                                      tile_size_k,
249                                      tile_size_n,
250                                      alpha, TILE_A[ki][ki], tile_size_k,
251                                             TILE_B[ki][ni], tile_size_n );
252                     }
253
254                     for ( mi = ki - 1; mi >= 0; mi-- )
255                     {
256                         tile_size_m = ddss_tile_size( M, mi );
257                         for ( nni = 0; nni < nt; nni++ )
258                         {
259                             tile_size_nn = ddss_tile_size( N, nni );
260
261                             #pragma oss task in( TILE_A[mi][ki] ) \
262                                 in( TILE_B[ki][nni] ) \
263                                 inout( TILE_B[mi][nni] ) \
264                                 shared( TILE_A, TILE_B ) \
265                                 firstprivate( ki, mi, nni ) \
266                                 label( dgemm )
267                             cblas_dgemm( CblasRowMajor,
268                                          CblasNoTrans, CblasNoTrans,
269                                          tile_size_m,
270                                          tile_size_nn,
271                                          tile_size_k,
272                                          -1.0,  TILE_A[mi][ki],  tile_size_k,
273                                                 TILE_B[ki][nni], tile_size_nn,
274                                          alpha, TILE_B[mi][nni], tile_size_nn );
275                         }
276                     }
277                 }
278             }
279
280             // --SIDE = Left & UPLO = Upper & TRANS_A = Trans--
281             else
282             {
283                 for ( ki = 0; ki < mt; ki++ )
284                 {
285                     tile_size_k = ddss_tile_size( M, ki );
286                     if ( ki == 0 )
287                     {
288                         alpha = ALPHA;
289                     }
290                     else
291                     {
292                         alpha = 1.0;
293                     }
294
295                     for ( ni = 0; ni < nt; ni++ )
296                     {
297                         tile_size_n = ddss_tile_size( N, ni );
298
299                         #pragma oss task in( TILE_A[ki][ki] ) \
300                             inout( TILE_B[ki][ni] ) \
301                             shared( TILE_A, TILE_B ) \
```

```
302                             firstprivate( ki, ni ) \
303                             label( dtrsm )
304                         cblas_dtrsm( CblasRowMajor,
305                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
306                                      ( CBLAS_TRANSPOSE ) TRANS_A,
307                                      ( CBLAS_DIAG ) DIAG,
308                                      tile_size_k,
309                                      tile_size_n,
310                                      alpha, TILE_A[ki][ki], tile_size_k,
311                                             TILE_B[ki][ni], tile_size_n );
312                     }
313
314                     for ( mi = ki + 1; mi < mt; mi++ )
315                     {
316                         tile_size_m = ddss_tile_size( M, mi );
317                         for ( nni = 0; nni < nt; nni++ )
318                         {
319                             tile_size_nn = ddss_tile_size( N, nni );
320
321                             #pragma oss task in( TILE_A[ki][mi]) \
322                                 in( TILE_B[ki][nni] ) \
323                                 inout(  TILE_B[mi][nni] ) \
324                                 shared( TILE_A, TILE_B ) \
325                                 firstprivate( ki, mi, nni ) \
326                                 label( dgemm )
327                             cblas_dgemm( CblasRowMajor,
328                                          CblasTrans, CblasNoTrans,
329                                          tile_size_m,
330                                          tile_size_nn,
331                                          tile_size_k,
332                                          -1.0,  TILE_A[ki][mi],  tile_size_m,
333                                                 TILE_B[ki][nni], tile_size_nn,
334                                          alpha, TILE_B[mi][nni], tile_size_nn );
335                         }
336                     }
337                 }
338             }
339         }
340
341         // --SIDE = Left & UPLO = Lower & TRANS_A = NoTrans--
342         else
343         {
344             if ( TRANS_A == NoTrans )
345             {
346                 for ( ki = 0; ki < mt; ki++ )
347                 {
348                     tile_size_k = ddss_tile_size( M, ki );
349                     if ( ki == 0 )
350                     {
351                         alpha = ALPHA;
352                     }
353                     else
354                     {
355                         alpha = 1.0;
356                     }
357
358                     for ( ni = 0; ni < nt; ni++ )
359                     {
360                         tile_size_n = ddss_tile_size( N, ni );
361
362                         #pragma oss task in( TILE_A[ki][ki] ) \
363                             inout(  TILE_B[ki][ni] ) \
364                             shared( TILE_A, TILE_B ) \
365                             firstprivate( ki, ni ) \
366                             label( dtrsm )
367                         cblas_dtrsm( CblasRowMajor,
368                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
369                                      ( CBLAS_TRANSPOSE ) TRANS_A,
370                                      ( CBLAS_DIAG ) DIAG,
371                                      tile_size_k,
372                                      tile_size_n,
373                                      alpha, TILE_A[ki][ki], tile_size_k,
374                                             TILE_B[ki][ni], tile_size_n );
375
376                     }
377                     for ( nni = 0; nni < nt; nni++ )
378                     {
379                         tile_size_nn = ddss_tile_size( N, nni );
380
381                         for ( mi = ki + 1; mi < mt; mi++ )
382                         {
383
384                             tile_size_m = ddss_tile_size( M, mi );
385
386                             #pragma oss task in( TILE_A[mi][ki] ) \
387                                 in( TILE_B[ki][nni] ) \
388                                 inout( TILE_B[mi][nni] ) \
```

```
389                                 shared( TILE_A, TILE_B ) \
390                                 firstprivate( ki, mi, nni ) \
391                                 label( dgemm )
392                             cblas_dgemm( CblasRowMajor,
393                                         CblasNoTrans, CblasNoTrans,
394                                         tile_size_m,
395                                         tile_size_nn,
396                                         tile_size_k,
397                                         -1.0,   TILE_A[mi][ki],  tile_size_k,
398                                                TILE_B[ki][nni], tile_size_nn,
399                                         alpha, TILE_B[mi][nni], tile_size_nn );
400                         }
401                     }
402                 }
403             }
404 
405             // --SIDE = Left & UPLO = Lower & TRANS_A = Trans--
406             else
407             {
408                 for ( ki = mt - 1; ki >= 0; ki-- )
409                 {
410                     tile_size_k = ddss_tile_size( M, ki );
411                     if ( ki == mt - 1 )
412                     {
413                         alpha = ALPHA;
414                     }
415                     else
416                     {
417                         alpha = 1.0;
418                     }
419 
420                     for ( ni = 0; ni < nt; ni++ )
421                     {
422                         tile_size_n = ddss_tile_size( N, ni );
423 
424                         #pragma oss task in( TILE_A[ki][ki] ) \
425                             inout( TILE_B[ki][ni] ) \
426                             shared( TILE_A, TILE_B ) \
427                             firstprivate( ki, ni ) \
428                             label( dtrsm )
429                         cblas_dtrsm( CblasRowMajor,
430                                     ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
431                                     ( CBLAS_TRANSPOSE ) TRANS_A,
432                                     ( CBLAS_DIAG ) DIAG,
433                                     tile_size_k,
434                                     tile_size_n,
435                                     alpha, TILE_A[ki][ki], tile_size_k,
436                                            TILE_B[ki][ni], tile_size_n );
437                     }
438 
439                     for ( mi = ki - 1; mi >= 0; mi-- )
440                     {
441                         tile_size_m = ddss_tile_size( M, mi );
442                         for ( nni = 0; nni < nt; nni++ )
443                         {
444                             tile_size_nn = ddss_tile_size( N, nni );
445 
446                             #pragma oss task in( TILE_A[ki][mi] ) \
447                                 in( TILE_B[ki][nni] ) \
448                                 inout(  TILE_B[mi][nni] ) \
449                                 shared( TILE_A, TILE_B ) \
450                                 firstprivate( ki, mi, nni ) \
451                                 label( dgemm )
452                             cblas_dgemm( CblasRowMajor,
453                                         CblasTrans, CblasNoTrans,
454                                         tile_size_m,
455                                         tile_size_nn,
456                                         tile_size_k,
457                                         -1.0,  TILE_A[ki][mi],  tile_size_m,
458                                                TILE_B[ki][nni], tile_size_nn,
459                                         alpha, TILE_B[mi][nni], tile_size_nn );
460                         }
461                     }
462                 }
463             }
464         }
465     }
466 
467     // --SIDE = Right & UPLO = Upper & TRANS_A = NoTrans--
468     else
469     {
470         if ( UPLO == Upper )
471         {
472             if ( TRANS_A == NoTrans )
473             {
474                 for ( ki = 0; ki < nt; ki++ )
475                 {
```

```
476                     tile_size_k = ddss_tile_size( N, ki );
477                     if ( ki == 0 )
478                     {
479                         alpha = ALPHA;
480                     }
481                     else
482                     {
483                         alpha = 1.0;
484                     }
485 
486                     for ( mi = 0; mi < mt; mi++ )
487                     {
488                         tile_size_m = ddss_tile_size( M, mi );
489 
490                         #pragma oss task in( TILE_A[ki][ki] ) \
491                             inout(  TILE_B[mi][ki] ) \
492                             shared( TILE_A, TILE_B ) \
493                             firstprivate( ki, mi ) \
494                             label( dtrsm )
495                         cblas_dtrsm( CblasRowMajor,
496                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
497                                      ( CBLAS_TRANSPOSE ) TRANS_A,
498                                      ( CBLAS_DIAG ) DIAG,
499                                      tile_size_m,
500                                      tile_size_k,
501                                      alpha, TILE_A[ki][ki], tile_size_k,
502                                             TILE_B[mi][ki], tile_size_k );
503                     }
504 
505                     for ( ni = ki + 1; ni < nt; ni++ )
506                     {
507                         tile_size_n = ddss_tile_size( N, ni );
508                         for ( mmi = 0; mmi < mt; mmi++ )
509                         {
510                             tile_size_mm = ddss_tile_size( M, mmi );
511 
512                             #pragma oss task in( TILE_A[ki][ni] ) \
513                                 in( TILE_B[mmi][ki] ) \
514                                 inout(  TILE_B[mmi][ni] ) \
515                                 shared( TILE_A, TILE_B ) \
516                                 firstprivate( ki, ni, mmi ) \
517                                 label( dgemm )
518                             cblas_dgemm( CblasRowMajor,
519                                          CblasNoTrans, CblasNoTrans,
520                                          tile_size_mm,
521                                          tile_size_n,
522                                          tile_size_k,
523                                          -1.0,  TILE_B[mmi][ki], tile_size_k,
524                                                 TILE_A[ki][ni],  tile_size_n,
525                                          alpha, TILE_B[mmi][ni], tile_size_n );
526                         }
527                     }
528                 }
529             }
530             // --SIDE = Right & UPLO = Upper & TRANS_A = Trans--
531             else
532             {
533                 for ( ki = nt - 1; ki >= 0; ki-- )
534                 {
535                     tile_size_k = ddss_tile_size( N, ki );
536                     if ( ki == nt - 1 )
537                     {
538                         alpha = ALPHA;
539                     }
540                     else
541                     {
542                         alpha = 1.0;
543                     }
544 
545                     for ( mi = 0; mi < mt; mi++ )
546                     {
547                         tile_size_m = ddss_tile_size( M, mi );
548 
549                         #pragma oss task in( TILE_A[ki][ki] ) \
550                             inout( TILE_B[mi][ki] ) \
551                             shared( TILE_A, TILE_B ) \
552                             firstprivate( ki, mi ) \
553                             label( dtrsm )
554                         cblas_dtrsm( CblasRowMajor,
555                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
556                                      ( CBLAS_TRANSPOSE ) TRANS_A,
557                                      ( CBLAS_DIAG ) DIAG,
558                                      tile_size_m,
559                                      tile_size_k,
560                                      alpha, TILE_A[ki][ki], tile_size_k,
561                                             TILE_B[mi][ki], tile_size_k );
562                     }
```

```
563
564                     for ( ni = ki - 1; ni >= 0; ni-- )
565                     {
566                         tile_size_n = ddss_tile_size( N, ni );
567                         for ( mmi = 0; mmi < mt; mmi++ )
568                         {
569                             tile_size_mm = ddss_tile_size( M, mmi );
570
571                             #pragma oss task in( TILE_A[ni][ki] ) \
572                                 in( TILE_B[mmi][ki] ) \
573                                 inout( TILE_B[mmi][ni] ) \
574                                 shared( TILE_A, TILE_B ) \
575                                 firstprivate( ki, ni, mmi ) \
576                                 label( dgemm )
577                             cblas_dgemm( CblasRowMajor,
578                                         CblasNoTrans, CblasTrans,
579                                         tile_size_mm,
580                                         tile_size_n,
581                                         tile_size_k,
582                                         -1.0,  TILE_B[mmi][ki], tile_size_k,
583                                                TILE_A[ni][ki],  tile_size_k,
584                                         alpha, TILE_B[mmi][ni], tile_size_n );
585                         }
586                     }
587                 }
588             }
589         }
590
591         // --SIDE = Right & UPLO = Lower & TRANS_A = NoTrans--
592         else
593         {
594             if ( TRANS_A == NoTrans )
595             {
596                 for ( ki = nt - 1; ki >= 0; ki-- )
597                 {
598                     tile_size_k = ddss_tile_size( N, ki );
599                     if ( ki == nt - 1 )
600                     {
601                         alpha = ALPHA;
602                     }
603                     else
604                     {
605                         alpha = 1.0;
606                     }
607
608                     for ( mi = 0; mi < mt; mi++ )
609                     {
610                         tile_size_m = ddss_tile_size( M, mi );
611
612                         #pragma oss task in( TILE_A[ki][ki] ) \
613                             inout( TILE_B[mi][ki] ) \
614                             shared( TILE_A, TILE_B ) \
615                             firstprivate( ki, mi ) \
616                             label( dtrsm )
617                         cblas_dtrsm( CblasRowMajor,
618                                     ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
619                                     ( CBLAS_TRANSPOSE ) TRANS_A,
620                                     ( CBLAS_DIAG ) DIAG,
621                                     tile_size_m,
622                                     tile_size_k,
623                                     alpha, TILE_A[ki][ki], tile_size_k,
624                                            TILE_B[mi][ki], tile_size_k );
625                     }
626
627                     for ( ni = ki - 1; ni >= 0; ni-- )
628                     {
629                         tile_size_n = ddss_tile_size( N, ni );
630                         for ( mmi = 0; mmi < mt; mmi++ )
631                         {
632                             tile_size_mm = ddss_tile_size( M, mmi );
633
634                             #pragma oss task in( TILE_A[ki][ni] ) \
635                                 in( TILE_B[mmi][ki] ) \
636                                 inout( TILE_B[mmi][ni] ) \
637                                 shared( TILE_A, TILE_B ) \
638                                 firstprivate( ki, ni, mmi ) \
639                                 label( dgemm )
640                             cblas_dgemm( CblasRowMajor,
641                                         CblasNoTrans, CblasNoTrans,
642                                         tile_size_mm,
643                                         tile_size_n,
644                                         tile_size_k,
645                                         -1.0,  TILE_B[mmi][ki], tile_size_k,
646                                                TILE_A[ki][ni],  tile_size_n,
647                                         alpha, TILE_B[mmi][ni], tile_size_n );
648                         }
649                     }
```

```
650                 }
651             }
652
653             // --SIDE = Right & UPLO = Lower & TRANS_A = Trans--
654             else
655             {
656                 for ( ki = 0; ki < nt; ki++ )
657                 {
658                     tile_size_k = ddss_tile_size( N, ki );
659                     if ( ki == 0 )
660                     {
661                         alpha = ALPHA;
662                     }
663                     else
664                     {
665                         alpha = 1.0;
666                     }
667
668                     for ( mi = 0; mi < mt; mi++ )
669                     {
670                         tile_size_m = ddss_tile_size( M, mi );
671
672                         #pragma oss task in( TILE_A[ki][ki] ) \
673                             inout( TILE_B[mi][ki] ) \
674                             shared( TILE_A,TILE_B ) \
675                             firstprivate( ki, mi ) \
676                             label( dtrsm )
677                         cblas_dtrsm( CblasRowMajor,
678                                      ( CBLAS_SIDE ) SIDE, ( CBLAS_UPLO ) UPLO,
679                                      ( CBLAS_TRANSPOSE ) TRANS_A,
680                                      ( CBLAS_DIAG ) DIAG,
681                                      tile_size_m,
682                                      tile_size_k,
683                                      alpha, TILE_A[ki][ki], tile_size_k,
684                                             TILE_B[mi][ki], tile_size_k );
685                     }
686
687                     for ( ni = ki + 1; ni < nt; ni++ )
688                     {
689                         tile_size_n = ddss_tile_size( N, ni );
690                         for ( mmi = 0; mmi < mt; mmi++ )
691                         {
692                             tile_size_mm = ddss_tile_size( M, mmi );
693
694                             #pragma oss task in( TILE_A[ni][ki] ) \
695                                 in( TILE_B[mmi][ki] ) \
696                                 inout(  TILE_B[mmi][ni] ) \
697                                 shared( TILE_A, TILE_B ) \
698                                 firstprivate( ki, ni, mmi ) \
699                                 label( dgemm )
700                             cblas_dgemm( CblasRowMajor,
701                                          CblasNoTrans, CblasTrans,
702                                          tile_size_mm,
703                                          tile_size_n,
704                                          tile_size_k,
705                                          -1.0,  TILE_B[mmi][ki], tile_size_k,
706                                                 TILE_A[ni][ki],  tile_size_k,
707                                          alpha, TILE_B[mmi][ni], tile_size_n );
708                         }
709                     }
710                 }
711             }
712         }
713     }
714
715     /*****************************************
716      --From tiled data layout to flat data layout--
717      *****************************************/
718
719     ddss_dtiled2flat( M, N, B, LDB, Bm, Bn, TILE_B );
720
721     // --Tile matrices free--
722     free( TILE_A );
723     free( TILE_B );
724
725     return Success;
726
727 }
```

Index