2.10 LU factorization of a general M-by-N distributed matrix.

In this section we first briefly describe the sequential block-partitioned versions of the dense LU, QR and Cholesky factorization routines of the LAPACK library. Since we also wish to discuss the parallel factorizations we describe the right-looking versions of the routines. The right-looking variants minimize data communication and distribute the computation across all processes After describing the sequential factorizations the parallel versions will be discussed.

For the implementation of the parallel block partitioned algorithms in ScaLAPACK we assume that a matrix A is distributed over a $P \times Q$ process grid with a block cyclic distribution and a block size of $n_b \times n_b$ matching the block size of the algorithm. Thus each n_b-wide column (or row) panel lies in one column (row) of the process grid. In the LU, QR and Cholesky factorization routines in which the distribution of work be-comes uneven as the computation progresses a larger block size results in greater load imbalance but reduces the frequency of communication between processes. There is therefore a tradeoff between load imbalance and communication startup cost which can be controlled by varying the block size. In addition to the load imbalance that arises as distributed data are eliminated from a computation load imbalance may also arise due to computational "hot spots" where certain processes have more work to do between synchronization points than others. This is the case for example in the LU factorization algorithm where partial pivoting is performed over rows in a single column of the process grid while the other processes are idle. Similarly the evaluation of each block row of the U matrix requires the solution of a lower triangular system across processes in a single row of the process grid The effect of this type of load imbalance can be minimized through the choice of P and Q.

The LU factorization applies a sequence of Gaussian eliminations to form $A=PLU$, where A and L are $M \times N$ matrices, and U is an NxN matrix. L is unit lower triangular (lower triangular with 1's on the main diagonal), *U* is upper triangular, and *P* is a permutation matrix, which is stored in a *min(M;N)* vector.

Constructing the Block LU Factorization

At the *k*-th step of the computation *(k=1,2,…),* it is assumed that the $m \times n$ submatrix of $A^{(k)}$ $(m=M-(k-1)n_b, n=N-(k-1)n_b)$ is to be partioned as follows

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = P \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix} = P \begin{pmatrix} L_{11}U_{11} & L_{11}U_{12} \\ L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{pmatrix}$$

where the block A_{11} is $n_b \times (n-n_b)$, A_{12} is $(m-n_b) \times n_b$, and A_{22} is $(m-n_b) \times (n-n_b)$, L_{11} is a unit lower triangular matrix, and U_{11} is an upper triangular matrix.

At first a sequence of Gaussian eliminations is performed on the first $m \times n_b$ panel of $A^{(k)}$ (i. e. A_{11} and A_{21}). Once this is completed the matrices L_{11}, L_{21} and U_{11} are known and we can rearrange the block equations

$$\begin{aligned} U_{12} &\leftarrow (L_{11})^{-1} A_{12} \\ \tilde{A}_{22} &\leftarrow A_{22} - L_{21}U_{12} = L_{22}U_{22} \end{aligned}$$

The LU factorization can be done by recursively applying the steps outlined above to the $(m-n_b) \times (n-n_b)$ matrix $\tilde{A}_{22}$.

The computation of the above steps in the LAPACK routine **DGETRF** involves the following operations

1. **DGETF2:** Apply the LU factorization on an $m \times n_b$ column panel of *A* (i e A_{11} and A_{2l})
   - **[** Repeat n_b times ($i = 1 \cdots n_b$)]

     −**IDAMAX** find the (absolute) maximum element of the *i*-th column and its location

     −**DSWAP** interchange the *i*-th row with the row which holds the maximum

     − **DSCAL** scale the *i*-th column of the matrix

     − **DGER** update the trailing submatrix
2. **DLASWP** Apply row interchanges to the left and the right of the panel
3. **DTRSM** Compute the $n_b \times (n-n_b)$, row panel of

U, $U_{12} \leftarrow (L_{11})^{-1} A_{12}$

4. **DGEMM** Update the rest of the matrix A_{22}, $\tilde{A}_{22} \leftarrow A_{22} - L_{21}U_{12} = L_{22}U_{22}$

The corresponding parallel implementation of the ScaLAPACK routine **PDGETRF** proceeds as follows

1. **PDGETF2:** The current column of processes performs the LU factorization on an $m \times n_b$ column panel of A (i e A_{11} and A_{2l})
   - [Repeat n_b times ($i = 1 \cdots n_b$)]
     - **PDAMAX** find the (absolute) maximum value of the i-th column and its location (pivot information will be stored on the column of processes)
     - **PDLASWP** interchange the i-th row with the row which holds the maximum
     - **PDSCAL** scale the i-th column of the matrix
     - **PDGER** broadcast the i-th row columnwise $((n_b - i)$ elements) in the current column of processes and update the trailing submatrix
   - Every process in the current process column broadcasts the same pivot information row wise to all columns of processes
2. **PDLASWP:** All processes apply row interchanges to the left and the right of the current panel
3. **PDTRSM:** L_{11} is broadcast along the current row of processes which converts the row panel A_{12} to U_{12}
4. **PDGEMM:** The column panel L_{21} is broadcast rowwise *(построчно)* across all columns of processes. The row panel U_{12} is broadcast columnwise down all rows of processes Then all processes update their local portions of the matrix A_{22}

Syntaxes of the routine for use to computes the LU factorization of a general M-by-N distributed matrix

```
PvGETRF(M,N,A,IA,JA,DESCA,IPIV,INFO)
void pdgetrf_(int *m, int *n, double* a, int *ia, int *ja, int *desc_a, int
             *ipiv, int *info);
```

Purpose: This routine forms the LU factorization of a general M-by-N distributed matrix sub(A)=A(IA:IA+M-1,JA:JA+N-1) as A=P*L*U where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if M >N) and U is upper triangular (upper trapezoidal if M<N). L and U are stored in sub(A). The routine uses partial pivoting, with row interchanges.

Input Parameters

M (global) INTEGER.

The number of rows in the distributed submatrix sub(A); M≥0.

N (global) INTEGER.

The number of columns in the distributed submatrix sub(A); N≥0.

A (local) REAL for PSGETRF, DOUBLE PRECISION for PDGETRF, COMPLEX for PCGETRF, DOUBLE COMPLEX for PZGETRF.

Pointer into the local memory to an array of local dimension (LLD_A,LOCC(JA+N-1)). Contains the local pieces of the distributed matrix sub(A) to be factored.

IA, JA (global) INTEGER.

The row and column indices in the global array A indicating the first row and the first column of the submatrix A(IA:IA+N-1,JA:JA+N-1), respectively.

DESCA (global and local) INTEGER array, dimension (dlen_).

The array descriptor for the distributed matrix A.

Output Parameters

A

Overwritten by local pieces of the factors *L* and *U* from the factorization *A* = *P***L***U*. The unit diagonal elements of *L* are not stored.

IPIV (local) INTEGER array.

The dimension of IPIV is (LOC_R(M_A)+MB_A). This array contains the pivoting information: local row I was interchanged with global row IPIV(I). This array is tied to the distributed matrix A.

INFO (global) INTEGER.

If INFO=0, the execution is successful.

INFO < 0: if the i-th argument is an array and the j-th entry had an illegal value, then INFO = -(I*100+J); if the i-th argument is a scalar and had an illegal value, then INFO = -I.

If INFO = I, u_{ii} is 0. The factorization has been completed, but the factor U is exactly singular. Division by zero will occur if you use the factor U for solving a system of linear equations.

Example 2.10.1*. The using of function* pdgetrf

```
#include <stdio.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "mpi.h"
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sstream>
using namespace std;
#define A(i,j) A[(i)*n+(j)]
#define L(i,j) L[(i)*n+(j)]
#define U(i,j) U[(i)*n+(j)]
#define A_distr(i,j) A_distr[(i)*k+(j)]
#define L_distr(i,j) L_distr[(i)*k+(j)]
#define U_distr(i,j) U_distr[(i)*k+(j)]
static int MAX( int a, int b ){
  if (a>b) return(a); else return(b);
}
extern "C"
{
void Cblacs_pinfo( int* mypnum, int* nprocs);
void Cblacs_get( int context, int request, int* value);
int  Cblacs_gridinit( int* context, char * order, int np_row, int np_col);
void Cblacs_gridinfo( int context, int*  np_row, int* np_col, int*  my_row, int*  my_col);
void Cblacs_gridexit( int context);
void Cblacs_barrier(int, const char*);
void Cblacs_exit( int error_code);
void Cblacs_pcoord(int, int, int*, int*);
int  numroc_(int *n, int *nb, int *iproc, int *isrcproc, int *nprocs);
int indxl2g_(int*, int*, int*, int*, int*);
```

```
void descinit_(int *desc, int *m, int *n, int *mb, int *nb, int *irsrc, int *icsrc, int *ictxt,
                    int *lld, int *info);
void pdgeadd_(char *TRANS,int *M, int *N,double * ALPHA,double *A,int *IA,int *JA,int
                      *DESCA,double *BETA,double *C, int *IC,int *JC,int *DESCC);
void pdgetrf_(int *m, int *n, double* a, int *ia, int *ja, int *desc_a, int *ipiv, int *info);
void pdgemm_( char *TRANSA,char *TRANSB,int *M,int *N,int *K,double *ALPHA,
                      double *A,int *IA,int *JA,int *DESCA,  double * B, int * IB, int * JB, int *
                      DESCB,double * BETA,double * C, int * IC, int * JC, int * DESCC );
                  } // extern "C"
int main(int argc, char **argv) {
int i_one = 1,  i_zero = 0;
double zero=0.0E+0, one=1.0E+0;
int descA[9],descL[9],descU[9],descA_distr[9],descL_distr[9],descU_distr[9];
int iam,nprocs,nprow,npcol,myrow,mycol;
int m,n,mb,nb,mp,nq;
int i, j,mypnum;
int iloc,jloc;
int lld,lld_distr;
int ictxt,info,lwork;
int *ippiv;
double alpha, beta;
m=5; n=5;
mb=2; nb=2;
nprow=2; npcol=2;
double *A, *L,*U, *A_distr,*L_distr,*U_distr,*work,*tau;
Cblacs_pinfo(&iam,&nprocs);
Cblacs_get( -1, 0, &ictxt );
Cblacs_gridinit(&ictxt, "R", nprow, npcol );
Cblacs_gridinfo(ictxt, &nprow, &npcol, &myrow, &mycol );
if ( iam==0 ){
A =  (double*)malloc(m*n*sizeof(double));
L =  (double*)malloc(m*n*sizeof(double));
U = (double*)malloc(m*n*sizeof(double));
for(i=0;i<m;i++ )
  for(j=0;j<n;j++)
                  A[i*n+j]=(10*i+j);//(i+j);
}else{
A = NULL;
L = NULL;
U = NULL;
}
if (iam==0)
  {
  printf("============ REZULT OF THE PROGRAM %s \n",argv[0]);
  cout << "Global matrix A:\n";
          for (i = 0; i < m; ++i) {
           for (j = 0; j < n; ++j) {
            cout << setw(5) << *(A + n*i + j) << " ";
           }
           cout << "\n";
          }
          cout << endl;
```

```
}
mp = numroc_( &m, &mb, &myrow, &i_zero, &nprow );
nq = numroc_( &n, &nb, &mycol, &i_zero, &npcol );
A_distr =(double*) malloc( mp*nq*sizeof(double));
L_distr =(double*) malloc( mp*nq*sizeof(double));
U_distr =(double*) malloc( mp*nq*sizeof(double));
ippiv = (int*) malloc( (mp+mb)*sizeof(int));
lld = MAX( numroc_( &n, &n, &myrow, &i_zero, &nprow ), 1 );
descinit_(descA, &m, &n, &m, &n, &i_zero, &i_zero, &ictxt, &lld, &info );
descinit_(descL, &m, &n, &m, &n, &i_zero, &i_zero, &ictxt, &lld, &info );
descinit_(descU, &m, &n, &m, &n, &i_zero, &i_zero, &ictxt, &lld, &info );
lld_distr = MAX( mp, 1 );
descinit_( descA_distr, &m, &n, &mb, &nb, &i_zero, &i_zero, &ictxt, &lld_distr, &info );
descinit_( descL_distr, &m, &n, &mb, &nb, &i_zero, &i_zero, &ictxt, &lld_distr, &info );
descinit_( descU_distr, &m, &n, &mb, &nb, &i_zero, &i_zero, &ictxt, &lld_distr, &info );
pdgeadd_( "N", &m, &n, &one, A, &i_one, &i_one, descA, &zero, A_distr, &i_one,
            &i_one, descA_distr );
pdgetrf_(&m, &n, A_distr, &i_one, &i_one, descA_distr, ippiv, &info);
pdgeadd_( "N", &m, &n, &one, A_distr, &i_one, &i_one, descA_distr, &zero, A, &i_one,
            &i_one, descA );
if (iam==0)
{
   cout << "Global matrix A (after use functionin pdgetrf_): \n";
            for (j = 0; j < n; ++j){
            //for (i = 0; i < m; ++i) {
            for (i = 0; i < m; ++i){
                  // for (j = 0; j < n; ++j){
             cout << setw(5) << *(A + n*i + j) << " ";
            }
            cout << "\n";
           }
           cout << endl;
   for (j = 0; j < n; ++j){
   //for (i = 0; i < m; ++i){
          for (i = 0; i < m; ++i){
         //for (j = 0; j < n; ++j){
          if (i==j) L[i*n+j]=1;
     if (i<j) L[i*n+j]=A[i*n+j];
     if (i>j) L[i*n+j]=0;
     if (i>=j) U[i*n+j]=A[i*n+j]; else U[i*n+j]=0;
     }
 }
cout << "Global matrix L :\n";
 for (j = 0; j < n; ++j)
//   for (i = 0; i < m; ++i)
   {
          for (i = 0; i < m; ++i){
         // for (j = 0; j < n; ++j){
                         cout << setw(5) << *(L + n*i + j) << " ";
            }
            cout << "\n";
           }
          cout << endl;
cout << "Global matrix U :\n";
for (j = 0; j < n; ++j){
         for (i = 0; i < m; ++i){
                cout << setw(5) << *(U + n*i + j) << " ";
         }
           cout << "\n";
          }
          cout << endl;
}

if( myrow==0 && mycol==0 ){
free( A );free( L );free( U );
}
Cblacs_gridexit(ictxt );
Cblacs_exit( 0);
}
```

Program's rezults

```
[Hancu_B_S@hpc ScaLAPACK_Exemple_Curs_Online]$ ./mpiCC_ScL -o
Example2.10.1.exe Example2.10.1.cpp
[Hancu_B_S@hpc ScaLAPACK_Exemple_Curs_Online]$ /opt/openmpi/bin/mpirun -n 4 -
host compute-0-0  Example2.10.1.exe
============ REZULT OF THE PROGRAM Example2.10.1.exe
Global matrix A:
    0     1     2     3     4
   10    11    12    13    14
   20    21    22    23    24
   30    31    32    33    34
   40    41    42    43    44

Global matrix A (after use functioni pdgetrf_):
    4    14    24    34    44
    0    10    20    30    40
  0.5   0.5     0     0     0
 0.75  0.25     0     0     0
 0.25  0.75     0     0     0

Global matrix L :
    1     0     0     0     0
    0     1     0     0     0
  0.5   0.5     1     0     0
 0.75  0.25     0     1     0
 0.25  0.75     0     0     1

Global matrix U :
    4    14    24    34    44
    0    10    20    30    40
    0     0     0     0     0
    0     0     0     0     0
    0     0     0     0     0
```

```
[Hancu_B_S@hpc ScaLAPACK_Exemple_Curs_Online]$
```

To verified the rezults we use the secvential varionat, i.e. the function dgetrf_.

```
#include <string>
#include <iostream>
#include <stdio.h>
#include <math.h>
#include <iomanip>
#include <fstream>
#include <sstream>
using namespace std;

extern "C" {
 void  dgetrf_(int *m, int *n, double *a, int *lda, int *ipiv, int *info);
}
int main() {
int i,j,m=5,n=5;
int info;
int LDA;
  double *A = new double[m*n];
  double *L = new double[m*n];
  double *U = new double[m*n];
  int *ipiv=new int[m];
  LDA=m;
for(i=0;i<m;i++ )
  for(j=0;j<n;j++)
                 A[i*n+j]=(10*i+j); //i+j;
  cout << "Global matrix AA:\n";
           for (i = 0; i < m; ++i) {
            for (j = 0; j < n; ++j) {
             cout << setw(5) << *(A + n*i + j) << " ";
            }
            cout << "\n";
           }
           cout << endl;
 dgetrf_(&m, &n, A, &LDA, ipiv, &info);
  for (j = 0; j < n; ++j)
   {
         //for (j = 0; j < n; ++j)
         for (i = 0; i < m; ++i)
            {
     if (i==j) L[i*n+j]=1;
     if (i<j) L[i*n+j]=A[i*n+j];
     if (i>j) L[i*n+j]=0;
     if (i>=j) U[i*n+j]=A[i*n+j]; else U[i*n+j]=0;
     }
 }
cout << "Global matrix L :\n";
//for (i = 0; i < m; ++i)
   for (j = 0; j < n; ++j)
   {
        //for (j = 0; j < n; ++j)
        for (i = 0; i < m; ++i) {
                        cout << setw(5) << *(L + n*i + j) << " ";
          }
          cout << "\n";
         }
         cout << endl;
cout << "Global matrix U :\n";
//for (i = 0; i < m; ++i)
  for (j = 0; j < n; ++j)
  {
        //for (j = 0; j < n; ++j)
        for (i = 0; i < m; ++i) {
                        cout << setw(5) << *(U + n*i + j) << " ";
          }
          cout << "\n";
         }
         cout << endl;

 cout << "LU decomposed matrix:\n " << endl;
  //for(i=0 ; i<m ; i++)
  for(j=0 ; j<n ; j++){
  for(i=0 ; i<m ; i++)
//for(j=0 ; j<n ; j++)
cout << setw(5) << *(A + n*i + j) << " ";
 cout << endl;
  }
  cout << endl;
   delete[] A;
   return 0;
}
```

Rezultatele programului:

```
[[Hancu_B_S@hpc Pentru_Masterat]$ ./mpiCC_ScL -o dgetrf.exe dgetrf.cpp
[Hancu_B_S@hpc Pentru_Masterat]$  /opt/openmpi/bin/mpirun -n 1  -host compute-0-
0,compute-0-4  dgetrf.exe
Global matrix AA:
    0  1  2  3  4
   10 11 12 13 14
   20 21 22 23 24
   30 31 32 33 34
   40 41 42 43 44

Global matrix L :
 1  0  0  0  0
 0  1  0  0  0
  0.5   0.5  1  0  0
 0.75  0.25  0  1  0
 0.25  0.75  0  0  1

Global matrix U :
 4 14 24 34 44
 0 10 20 30 40
 0  0  0  0  0
 0  0  0  0  0
 0  0  0  0  0

LU decomposed matrix:

 4 14 24 34 44
 0 10 20 30 40
  0.5   0.5  0  0  0
 0.75  0.25  0  0  0
 0.25  0.75  0  0  0
```

The auxiliar functions to LU factorization

SUBROUTINE <u>PDLASWP</u>(DIREC, ROWCOL, N, A, IA, JA, DESCA, K1, K2,IPIV)

```
  .. Scalar Arguments ..
  CHARACTER          DIREC, ROWCOL
  INTEGER            IA, JA, K1, K2, N
  .. Array Arguments ..
  INTEGER            DESCA( * ), IPIV( * )
  DOUBLE PRECISION   A( * )
```

Purpose:
========
PDLASWP performs a series of row or column interchanges on
the distributed matrix sub(A) = A(IA:IA+M-1,JA:JA+N-1). One
interchange is initiated for each of rows or columns K1 trough K2 of
sub(A). This routine assumes that the pivoting information has
already been broadcast along the process row or column.
Also note that this routine will only work for K1-K2 being in the
same MB (or NB) block. If you want to pivot a full matrix, use
PDLAPIV.
Arguments
=========
```
DIREC   (global input) CHARACTER
        Specifies in which order the permutation is applied:
        = 'F' (Forward)
        = 'B' (Backward)
ROWCOL  (global input) CHARACTER
        Specifies if the rows or columns are permuted:
        = 'R' (Rows)
        = 'C' (Columns)
N       (global input) INTEGER
        If ROWCOL = 'R', the length of the rows of the distributed
        matrix A(*,JA:JA+N-1) to be permuted;
        If ROWCOL = 'C', the length of the columns of the ted
        matrix A(IA:IA+N-1,*) to be permuted.
A       (local input/local output) DOUBLE PRECISION pointer into the
        local memory to an array of dimension (LLD_A, * ).
```

On entry, this array contains the local pieces of the
buted matrix to which the row/columns interchanges will be
applied. On exit the permuted distributed matrix.

IA (global input) INTEGER
The row index in the global array A indicating the first
row of sub(A).

JA (global input) INTEGER
The column index in the global array A indicating the
first column of sub(A).

DESCA (global and local input) INTEGER array of dimension DLEN_.
The array descriptor for the distributed matrix A.

K1 (global input) INTEGER
The first element of IPIV for which a row or column inter-
change will be done.

K2 (global input) INTEGER
The last element of IPIV for which a row or column inter-
change will be done.

IPIV (local input) INTEGER array, dimension LOCr(M_A)+MB_A for
row pivoting and LOCc(N_A)+NB_A for column pivoting. This
array is tied to the matrix A, IPIV(K) = L implies rows
(or columns) K and L are to be interchanged.

void pdscal_(int * N, double * ALPHA,double * X, int * IX, int * JX, int * DESCX, int * INCX)

```
.. Scalar Arguments ..
  int           * INCX, * IX, * JX, * N;
  double          * ALPHA;
  .. Array Arguments ..
  int           * DESCX;
  double          * X;
endif
```

Purpose
=======
PDSCAL multiplies an n element subvector sub(X) by the
alpha,
where
   sub(X) denotes X(IX,JX:JX+N-1) if INCX = M_X,
                    X(IX:IX+N-1,JX) if INCX = 1 and INCX <> M_X.

Arguments
=========

N (global input) INTEGER
On entry, N specifies the length of the subvector sub(X
N must be at least zero.

ALPHA (global input) DOUBLE PRECISION
On entry, ALPHA specifies the scalar alpha. When ALPHA
supplied as zero then the local entries of the array X
responding to the entries of the subvector sub(X) need
be set on input.

X (local input/local output) DOUBLE PRECISION array
On entry, X is an array of dimension (LLD_X, Kx), where

is at least MAX(1, Lr(1, IX)) when INCX = M_X
MAX(1, Lr(1, IX+N-1)) otherwise, and, Kx is at
Lc(1, JX+N-1) when INCX = M_X and Lc(1, JX) se.
Before entry, this array contains the local entries of
matrix X. On exit, sub(X) is overwritten with the
subvector.

IX (global input) INTEGER
On entry, IX specifies X's global row index, which points
the beginning of the submatrix sub(X).

JX (global input) INTEGER
On entry, JX specifies X's global column index, which
to the beginning of the submatrix sub(X).

DESCX (global and local input) INTEGER array
On entry, DESCX is an integer array of dimension DLEN_.
is the array descriptor for the matrix X.

INCX (global input) INTEGER
On entry, INCX specifies the global increment for
elements of X. Only two values of INCX are supported
this version, namely 1 and M_X. INCX must not be zero.

void pdtrsm_(F_CHAR_T SIDE, F_CHAR_T UPLO, F_CHAR_T TRANS, F_CHAR_T DIAG,int M, int N, double ALPHA,double A, int IA, int JA, int DESCA,double B, int IB, int JB, int DESCB)

```
.. Scalar Arguments ..
 F_CHAR_T       DIAG, SIDE, TRANS, UPLO;
 int            IA,  IB,  JA,  JB,  M,  N;
 double           ALPHA;
.. Array Arguments ..
 int            DESCA,  DESCB;
 double           A,  B;
```

Purpose
=======

PDTRSM solves one of the matrix equations op(sub(A))X = alphasub(B), or Xop(sub(A)) = alphasub(B),
where
sub(A) denotes A(IA:IA+M-1,JA:JA+M-1) if SIDE = 'L',
A(IA:IA+N-1,JA:JA+N-1) if SIDE = 'R', and,
sub(B) denotes B(IB:IB+M-1,JB:JB+N-1).
Alpha is a scalar, X and sub(B) are m by n submatrices, sub(A)
a unit, or non-unit, upper or lower triangular submatrix and op(Y
is one of
op(Y) = Y or op(Y) = Y'.
The submatrix X is overwritten on sub(B).

Arguments
=========

SIDE (global input) CHARACTER1
On entry, SIDE specifies whether op(sub(A)) appears
the left or right of X as follows:
SIDE = 'L' or 'l' op(sub(A))X = alphasub(B),
SIDE = 'R' or 'r' Xop(sub(A)) = alphasub(B).

UPLO (global input) CHARACTER1
On entry, UPLO specifies whether the submatrix sub(A)

```
        an upper or lower triangular submatrix as follows:
           UPLO = 'U' or 'u'   sub( A ) is an upper triangular
                               submatrix,
           UPLO = 'L' or 'l'   sub( A ) is a  lower triangular
                               submatrix.
TRANSA  (global input) CHARACTER1
        On entry,  TRANSA  specifies the form of op( sub( A ) ) to
        used in the matrix multiplication as follows:
           TRANSA = 'N' or 'n'   op( sub( A ) ) = sub( A ),
           TRANSA = 'T' or 't'   op( sub( A ) ) = sub( A )',
           TRANSA = 'C' or 'c'   op( sub( A ) ) = sub( A )'.
DIAG    (global input) CHARACTER1
        On entry,  DIAG  specifies  whether or not  sub( A )  is
        triangular as follows:
           DIAG = 'U' or 'u'  sub( A )  is  assumed to be unit
                              gular,
           DIAG = 'N' or 'n'  sub( A ) is not assumed to be unit
                              angular.
M       (global input) INTEGER
        On entry,  M  specifies the number of rows of  the  rix
        sub( B ). M  must be at least zero.
N       (global input) INTEGER
        On entry, N  specifies the number of columns of the rix
        sub( B ). N  must be at least zero.
ALPHA   (global input) DOUBLE PRECISION
        On entry, ALPHA specifies the scalar alpha.   When  ALPHA
        supplied  as  zero  then  the  local entries of  the array
        corresponding to the entries of the submatrix  sub( B )
        not be set on input.
A       (local input) DOUBLE PRECISION array
        On entry, A is an array of dimension (LLD_A, Ka), where Ka
        at least Lc( 1, JA+M-1 ) when  SIDE = 'L' or 'l'   and  is
        least Lc( 1, JA+N-1 ) otherwise.  Before  entry,  this
        contains the local entries of the matrix A.
        Before entry with  UPLO = 'U' or 'u', this array contains
        local entries corresponding to  the entries of the upper
        angular submatrix  sub( A ), and the local entries pon-
        ding to the entries of the strictly lower triangular part
        the submatrix  sub( A )  are not referenced.
        Before entry with  UPLO = 'L' or 'l', this array contains
        local entries corresponding to  the entries of the lower
        angular submatrix  sub( A ), and the local entries pon-
        ding to the entries of the strictly upper triangular part
        the submatrix  sub( A )  are not referenced.
        Note  that  when DIAG = 'U' or 'u', the local entries -
        ponding to the  diagonal elements  of the submatrix  sub( A
        are not referenced either, but are assumed to be unity.
IA      (global input) INTEGER
        On entry, IA  specifies A's global row index, which points
        the beginning of the submatrix sub( A ).
JA      (global input) INTEGER
        On entry, JA  specifies A's global column index, which
        to the beginning of the submatrix sub( A ).
DESCA   (global and local input) INTEGER array
        On entry, DESCA  is an integer array of dimension DLEN_.
        is the array descriptor for the matrix A.
B        (local input/local output) DOUBLE PRECISION array
        On entry, B is an array of dimension (LLD_B, Kb), where Kb
        at least Lc( 1, JB+N-1 ).  Before  entry, this array ns
        the local entries of the matrix  B.
        On exit, the local entries of this array corresponding to
        to  the entries of the submatrix sub( B ) are  overwritten
        the local entries of the m by n  solution submatrix.
IB       (global input) INTEGER
        On entry, IB  specifies B's global row index, which points
        the beginning of the submatrix sub( B ).
JB       (global input) INTEGER
        On entry, JB  specifies B's global column index, which
        to the beginning of the submatrix sub( B ).
DESCB   (global and local input) INTEGER array
        On entry, DESCB  is an integer array of dimension DLEN_.
        is the array descriptor for the matrix B.
```