2.11 ScaLAPACk routines for system of linear equations

Syntaxes of the routine for solve a general band system of linear equations.

```
PvDBSV(N,BWL,BWU,NRHS,A,JA,DESCA,B,IB,DESCB,WORK,LWORK,INFO)
pddbsv_(integer *n,integer *bwl,integer *bwu,integer *nrhs,
            doublereal *a,integer *ja,integer *desca,
            doublereal *b,integer *ib,integer *descb,
            doublereal *work,integer *lwork,integer *info);
```

Purpose: The routine solves the system of linear equations

A(1:N,JA:JA+N-1)*X=B(IB:IB+N-1,1:NRHS),

where *A(1:N, JA:JA+N-1)* is an *N*-by-*N* real/complex banded diagonally dominant-like distributed matrix with bandwidth *BWL, BWU*. Gaussian elimination without pivoting is used to factor a reordering of the matrix *X* into LU.

Input Parameters

N (global) INTEGER.

The order of the distributed submatrix A, ($n \geq 0$).

BWL (global) INTEGER.

Number of subdiagonals. $0 \leq BWL \leq N-1$.

BWU (global) INTEGER.

Number of subdiagonals. $0 \leq BWU \leq N-1$.

NRHS (global) INTEGER.

The number of right-hand sides; the number of columns of the distributed submatrix *B, (NRHS ≥ 0)*.

A (local) REAL for PSDBSV, DOUBLE PRECISION for PDDBSV, COMPLEX for PCDBSV, DOUBLE COMPLEX for PZDBSV.

Pointer into the local memory to an array with first dimension *LLD_A ≥ (BWL+BWU+1)* (stored in DESCA). On entry, this array contains the local pieces of the distributed matrix.

JA (global) INTEGER.

The index in the global array A that points to the start of the matrix to be operated on (which may be either all of *A* or a submatrix of *A*).

DESCA (global and local) INTEGER array of dimension DLEN.

The array descriptor for the distributed matrix *A* and contains information of mapping of *A* to memory. If 1d type (DTYPE_A=501 or 502), DLEN ≥ 7, if 2d type (DTYPE_A=1), DLEN ≥ 9.

B (local)REAL for PSDBSV, DOUBLE PRECISON for PDDBSV, COMPLEX for PCDBSV, DOUBLE COMPLEX for PZDBSV.

Pointer into the local memory to an array of local lead dimension LLD_B≥NB. On entry, this array contains the local pieces of the right hand sides *B(IB:IB+N-1,1:NRHS)*.

IB (global) INTEGER.

The row index in the global array b that points to the first row of the matrix to be operated on (which may be either all of b or a submatrix of *B*).

DESCB (global and local) INTEGER array of dimension DLEN.

The array descriptor for the distributed matrix *B* and contains information of mapping of *B* to memory. If 1d type (DTYPE_B =502), DLEN ≥ 7, if 2d type (DTYPE_B =1), DLEN ≥ 9.

WORK (local) REAL for PSDBSV, DOUBLE PRECISON for PDDBSV, COMPLEX for PCDBSV,DOUBLE COMPLEX for PZDBSV.

Temporary workspace. This space may be overwritten in between calls to routines. work must be the size given in LWORK.

LWORK (local or global) INTEGER.

Size of user-input workspace work. If LWORK is too small, the minimal acceptable size will be returned in WORK(1) and an error code is returned. LWORK ≥ NB(BWL+BWU)+
+6MAX(BWL,BWU)*MAX(BWL,BWU)+MAX((MAX(BWL,BWU)NRHS),
MAX(BWL,BWU)*MAX(BWL,BWU))

Output Parameters

A

On exit, this array contains information containing details of the factorization. Note that permutations are performed on the matrix, so that the factors returned are different from those returned by LAPACK.

B

On exit, this contains the local piece of the solutions distributed matrix *X*.

WORK

On exit, WORK(1) contains the minimal LWORK.

INFO (local) INTEGER.

If INFO=0, the execution is successful. < 0: if the i-th argument is an array and the j-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. INFO > 0: if INFO = k < NPROCS, the submatrix stored on processor info and factored locally was not positive definite, and the factorization was not completed. If INFO = k > NPROCS, the submatrix stored on processor info-NPROCS representing interactions with other processors was not positive definite, and the factorization was not completed.

Syntaxes of the routine for solve a general tridiagonal system of linear equations.

```
PvDTSV(N,NRHS,DL,D,DU,JA,DESCA,B,IB,DESCB,WORK,LWORK,INFO)
```

Purpose: The routine solves a system of linear equations

*A(1:N, JA:JA+N-1)*X = B(IB:IB+N-1, 1:NRHS),*

where *A(1:N, JA:JA+N-1)* is an *N*-by-*N* complex tridiagonal diagonally dominant-like distributed matrix. Gaussian elimination without pivoting is used to factor a reordering of the matrix into LU.

Input Parameters

N (global) INTEGER.

The order of the distributed submatrix *A (N ≥ 0)*.

NRHS INTEGER.

The number of right hand sides; the number of columns of the distributed matrix B (NRHS ≥ 0).

DL (local) REAL for PSDTSV, DOUBLE PRECISION for PDDTSV, COMPLEX for PCDTSV, DOUBLE COMPLEX for PZDTSV.

Pointer to local part of global vector storing the lower diagonal of the matrix. Globally, DL(1) is not referenced, and DL must be aligned with D. Must be of size > DESCA(NB_).

D (local) REAL for PSDTSV, DOUBLE PRECISION for PDDTSV, COMPLEX for PCDTSV, DOUBLE COMPLEX for PZDTSV.

Pointer to local part of global vector storing the main diagonal of the matrix.

DU (local) REAL for PSDTSV, DOUBLE PRECISION for PDDTSV, COMPLEX for PCDTSV, DOUBLE COMPLEX for PZDTSV.

Pointer to local part of global vector storing the upper diagonal of the matrix. Globally, DU(N) is not referenced, and DU must be aligned with D.

JA (global) INTEGER.

The index in the global array a that points to the start of the matrix to be operated on (which may be either all of *A* or a submatrix of *A*).

DESCA (global and local) INTEGER array of dimension DLEN.

The array descriptor for the distributed matrix *A*. Contains information of mapping of *A* to memory.

B (local) REAL for PSDTSV, DOUBLE PRECISON for PDDTSV, COMPLEX for PCDTSV, DOUBLE COMPLEX for PZDTSV.

Pointer into the local memory to an array of local lead dimension LLD_B > NB. On entry, this array contains the local pieces of the right hand sides *B(IB:IB+N-1,1:NRHS)*.

IB (global) INTEGER.

The row index in the global array b that points to the first row of the matrix to be operated on (which may be either all of *B* or a submatrix of *B*).

DESCB (global and local) INTEGER array of dimension DLEN.

The array descriptor for the distributed matrix *B*. Contains information of mapping of B to memory.

WORK (local) REAL for PSDTSV, DOUBLE PRECISON for PDDTSV, COMPLEX for PCDTSV, DOUBLE COMPLEX for PZDTSV

Temporary workspace. This space may be overwritten in between calls to routines. WORK must be the size given in LWORK.

LWORK (local or global) INTEGER.

Size of user-input workspace work. If lwork is too small, the minimal acceptable size will be returned in WORK(1) and an error code is returned. LWORK>(12*NPCOL+3*NB)+MAX((10+2*MIN(100, NRHS))*NPCOL+4*NRHS, 8*NPCOL)

Output Parameters

DL

On exit, this array contains information containing the * factors of the matrix.

D

On exit, this array contains information containing the * factors of the matrix. Must be of size > DESCA(NB_).

DU

On exit, this array contains information containing the * factors of the matrix. Must be of size > DESCA(NB_).

B

On exit, this contains the local piece of the solutions distributed matrix *X*.

WORK

On exit, WORK(1) contains the minimal LWORK.

INFO (local) INTEGER.

If INFO=0, the execution is successful. < 0: if the i-th argument is an array and the j-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. INFO > 0: if INFO = k < NPROCS, the submatrix stored on processor info and factored locally was not positive definite, and the factorization was not completed. If INFO = k > NPROCS, the submatrix stored on processor info-NPROCS representing interactions with other processors was not positive definite, and the factorization was not completed.

Syntaxes of the routine for compute the solution to the system of linear equations with a square distributed matrix and multiple right-hand sides.

Steps to solve a system using an LU decomposition:

1. Set up the equation $Ax = b$.
2. Find an LU decomposition for A. This will yield the equation $(LU)x = b$.
3. Let $y = Ux$. Then solve the equation $Ly = b$ for y.
4. Take the values for y and solve the equation $y = Ux$ for x. This will give the solution to the system $Ax=b$.

```
PvGESV(N,NRHS,A,IA,JA,DESCA,IPIV,B,IB,JB,DESCB,INFO)
int psgesv_(int *n,int *nrhs,float *a,int *ia,int *ja,int
           *desca,int *ipiv,float *b,int *ib,int *jb,int
            *descb,int *info);
```

Purpose: The routine PvGESV computes the solution to a real or complex system of linear equations *sub(A)*X = sub(B)*, where *sub(A) = A(IA:IA+N-1, JA:JA+N-1)* is an *N*-by-*N* distributed matrix and *X* and *sub(B) = B(IB:IB+N-1, JB:JB+NRHS-1)* are *N*-by-*NRHS* distributed matrices. The LU decomposition with partial pivoting and row interchanges is used to factor *sub(A)* as *sub(A) = P*L*U*, where *P* is a permutation matrix, *L* is unit lower triangular, and *U* is upper triangular. *L* and *U* are stored in *sub(A)*. The factored form of *sub(A)* is then used to solve the system of equations *sub(A)*X = sub(B)*. So, these routine use PDGETRF function for compute the LU factorization of *sub(A)*) and PDGETRS for solve the system *sub(A) * X = sub(B)*, overwriting *sub(B)* with *X*.

Input Parameters

N (global) INTEGER.

The number of rows and columns to be operated on, that is, the order of the distributed submatrix *sub(A)* (*N* ≥ *0*).

NRHS (global) INTEGER.

The number of right hand sides, that is, the number of columns of the distributed submatrices B and X (NRHS ≥ 0).

A, B (local) REAL for PSGESV, DOUBLE PRECISION for PDGESV, COMPLEX for PCGESV, DOUBLE COMPLEX for PZGESV.

Pointers into the local memory to arrays of local dimension *A(LLD_A,LOC$_c$(JA+N-1))* and *B(LLD_B,LOC$_c$(JB+NRHS-1))*, respectively. On entry, the array a contains the local pieces of the *N*-by-*N* distributed matrix *sub(A)* to be factored. On entry, the array *B* contains the right hand side distributed matrix *sub(B)*.

IA, JA (global) INTEGER.

The row and column indices in the global array *A* indicating the first row and the first column of *sub(A)*, respectively.

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

The array descriptor for the distributed matrix *A*.

IB, JB (global) INTEGER.

The row and column indices in the global array *B* indicating the first row and the first column of *sub(B)*, respectively.

DESCB (global and local) INTEGER array, dimension (DLEN_).

The array descriptor for the distributed matrix *B*.

Output Parameters

A

Overwritten by the factors *L* and *U* from the factorization *sub(A) = P*L*U;* the unit diagonal elements of *L* are not stored .

B

Overwritten by the solution distributed matrix *X*.

IPIV (local) INTEGER array.

The dimension of IPIV is $(LOC_r(M\_A)+MB\_A)$. This array contains the pivoting information. The (local) row *i* of the matrix was interchanged with the (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*.

INFO > 0:

If INFO = *K, U(IA+K-1,JA+K-1)* is exactly zero. The factorization has been completed, but the factor *U* is exactly singular, so the solution could not be computed.

Notes and Coding Rules

1. In your C program, argument info must be passed by reference.
2. If n > 0 and nrhs=0, only the factorization is computed.
3. The matrices and vector must have no common elements; otherwise, results are unpredictable.
4. The way these subroutines handle singularity differs from ScaLAPACK. These subroutines use the info argument to provide information about the singularity of *A*, like ScaLAPACK, but also provide an error message.
5. The NUMROC utility subroutine can be used to determine the values of LOCp(M_) and LOCq(N_) used in the argument descriptions above.
6. On both input and output, matrices *A* and *B* conform to ScaLAPACK format.
7. The following values must be equal: CTXT_A=CTXT_B.
8. The global general matrix *A* must be distributed using a square block-cyclic distribution; that is, MB_A = NB_A.
9. The following block sizes must be equal: MB_A=MB_B.
10. The global general matrix *A* must be aligned on a block row boundary; that is, ia-1 must be a multiple of MB_A.
11. The block row offset of *A* must be equal to the block column offset of *A*; that is, mod(ia-1, MB_A) = mod(ja-1, NB_A).
12. The block row offset of *A* must be equal to the block row offset of *B*; that is, mod(ia-1, MB_A) = mod(ib-1, MB_B).
13. In the process grid, the process row containing the first row of the submatrix *A* must also contain the first row of the submatrix *B*; that is, iarow = ibrow, where:
    - iarow = mod((((ia-1)/MB_A)+RSRC_A), p)
    - ibrow = mod((((ib-1)/MB_B)+RSRC_B), p)
14. There is no array descriptor for *ipvt*. It is a column-distributed vector with block size MB_A, local arrays of dimension LOCp(ia+m-1) by 1, and global index ia. A copy of this vector exists on each column of the process grid, and the process row over which the first column of *ipvt* is distributed is RSRC_A.

Example 2.11. The using of function psgesv

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "mpi.h"
#include <sys/time.h>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sstream>
using namespace std;
#define AA(i,j) AA[(i)*n+(j)]
#define BB(i) BB[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 exit( int error code);
void   Cblacs barrier(int, const char*);
int    numroc ( int *n, int *nb, int *iproc, int *isrcproc, int *nprocs);
void   descinit ( int *desc, int *m, int *n, int *mb, int *nb, int *irsrc,
int *icsrc,
            int *ictxt, int *lld, int *info);
double pdlamch ( int *ictxt , char *cmach);
double pdlange ( char *norm, int *m, int *n, double *A, int *ia, int *ja, int
*desca, double *work);
void pdlacpy ( char *uplo, int *m, int *n, double *a, int *ia, int *ja, int
*desca,
            double *b, int *ib, int *jb, int *descb);
void pdgesv ( int *n, int *nrhs, double *A, int *ia, int *ja, int *desca,
int* ipiv,
            double *B, int *ib, int *jb, int *descb, 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 );
int  indxg2p ( int *indxglob, int *nb, int *iproc, int *isrcproc, int
*nprocs);
int    indxl2g (int*, int*, int*, int*, int*);
}
int main(int argc, char **argv) {
   int iam, nprocs;
   int myrank mpi, nprocs mpi;
   int ictxt, nprow, npcol, myrow, mycol;
   int np, nq, n, nb, nqrhs, nrhs;
   int i, j, k, info, itemp;
   int descA[9], descB[9];
   double *A, *Acpy, *B, *X, *R, eps, *work;
```

```
   double *AA,*BB;
   int iloc,jloc;
   double AnormF, XnormF, RnormF, BnormF, residF;
   int *ippiv;
   int izero=0,ione=1;
   double mone=(-1.0e0),pone=(1.0e0);
   double MPIt1, MPIt2, MPIelapsed;
   MPI Init( &argc, &argv);
   MPI Comm rank(MPI COMM WORLD, &myrank mpi);
   MPI Comm size(MPI COMM WORLD, &nprocs mpi);
   nprow = 2; npcol = 2;
   n = 8; nrhs = 1;  nb = 2;
   if (nb>n)
      nb = n;
   if (nprow*npcol>nprocs mpi){
      if (myrank mpi==0)
         printf(" **** ERROR : we do not have enough processes available to
make a p-by-q process grid ***\n");
         printf(" **** Bye-bye
***\n");
      MPI Finalize(); exit(1);
   }
   Cblacs pinfo( &iam, &nprocs ) ;
   Cblacs get( -1, 0, &ictxt );
   Cblacs gridinit( &ictxt, "Row", nprow, npcol );
   Cblacs gridinfo( ictxt, &nprow, &npcol, &myrow, &mycol );
AA = new double[n*n];// matricea "globala" pentru AA*X=B
BB = new double[n*nrhs];//vectorul "global" pentru AA*X=B ( in el se
pasteraza solutia)
// initializarea marticelor globale
for(i=0;i<n;i++ )
     for(j=0;j<n;j++)

if(i == j){
          AA[i*n+i]=i+1;
         }
         else {
            AA[i*n+j]=0.1*(i+1)+0.001*(j+1);
         }
for(i=0;i<n;i++ )
     BB[i]=1;
if (iam==0)
  {
  printf("============ REZULT OF THE PROGRAM %s \n",argv[0]);
  cout << "Global matrix AA:\n";
              for (i = 0; i < n; ++i) {
                  for (j = 0; j < n; ++j) {
                      cout << setw(3) << *(AA + n*i + j) << "   ";
                  }
                  cout << "\n";
              }
              cout << endl;
  cout << "Global vector B:\n";
              for (i = 0; i < n; ++i) {
                  cout << setw(3) << *(BB + i ) << "";
                  cout << "\n";
              }
              cout << endl;
```

```
  }
Cblacs barrier(ictxt, "All");

//   if ((myrow>-1)&(mycol>-1)&(myrow<nprow)&(mycol<npcol)) {
      np    = numroc ( &n   , &nb, &myrow, &izero, &nprow );
      nq    = numroc ( &n   , &nb, &mycol, &izero, &npcol );
      nqrhs = numroc ( &nrhs, &nb, &mycol, &izero, &npcol );
A = (double *)calloc(np*nq,sizeof(double)) ;
      if (A==NULL){ printf("error of memory allocation A on proc
%dx%d\n",myrow,mycol); exit(0); }
      Acpy = (double *)calloc(np*nq,sizeof(double)) ;
      if (Acpy==NULL){ printf("error of memory allocation Acpy on proc
%dx%d\n",myrow,mycol); exit(0); }
      B = (double *)calloc(np*nqrhs,sizeof(double)) ;
      if (B==NULL){ printf("error of memory allocation B on proc
%dx%d\n",myrow,mycol); exit(0); }
      X = (double *)calloc(np*nqrhs,sizeof(double)) ;
      if (X==NULL){ printf("error of memory allocation X on proc
%dx%d\n",myrow,mycol); exit(0); }
      R = (double *)calloc(np*nqrhs,sizeof(double)) ;
      if (R==NULL){ printf("error of memory allocation R on proc
%dx%d\n",myrow,mycol); exit(0); }
      ippiv = (int *)calloc(np+nb,sizeof(int)) ;
      if (ippiv==NULL){ printf("error of memory allocation IPIV on proc
%dx%d\n",myrow,mycol); exit(0); }

//=== se complecteaza cu valori matricele locale
for(iloc=0;iloc<np;iloc++)
for(jloc=0;jloc<nq;jloc++){
int fortidl = iloc + 1;
int fortjdl = jloc + 1;
i = indxl2g (&fortidl, &nb, &myrow, &izero, &nprow)-1;
j = indxl2g (&fortjdl, &nb, &mycol, &izero, &npcol)-1;
A[iloc*np+jloc]=AA(i,j);
                          }
for(iloc=0;iloc<np;iloc++)
   {
int fortidl = iloc + 1;
i = indxl2g (&fortidl, &nb, &myrow, &izero, &nprow)-1;

               B[iloc]=BB(i);
                             }
// === se tiparesc matricele locale
sleep(iam);
printf("For process (%2d%2d) np=%2d,nq %2d and A:\n",myrow,mycol,np,nq);
for (i = 0; i < np; ++i)
 {
  for (j = 0; j < np; ++j)
  cout << setw(3) << *(A+np*i+j) << "   ";
  cout << endl;
 }
printf("For process (%2d%2d) np=%2d,nqrhs %2d and
B:\n",myrow,mycol,np,nqrhs);
for (i = 0; i < np; ++i)
 {
cout << setw(3) << *(B+i) << "";
cout << endl;
 }
```

```
cout << endl;
Cblacs barrier(ictxt, "All");

      itemp = max( 1, np );
      descinit ( descA, &n, &n   , &nb, &nb, &izero, &izero, &ictxt, &itemp,
&info );
      descinit ( descB, &n, &nrhs, &nb, &nb, &izero, &izero, &ictxt, &itemp,
&info );
//Make a copy of A and the rhs for checking purposes
      pdlacpy ( "All", &n, &n   , A, &ione, &ione, descA, Acpy, &ione, &ione,
descA );
      pdlacpy ( "All", &n, &nrhs, B, &ione, &ione, descB, X   , &ione, &ione,
descB );
/*
*********************************************************************
*     Call ScaLAPACK PDGESV routine
*********************************************************************
*/
Cblacs barrier(ictxt, "All");
      if( iam==0 ) {
         printf("                                                  \n");
         printf("**************************************************\n");
         printf("  Example of ScaLAPACK routine call: (PDGESV)  \n");
         printf("**************************************************\n");
         printf("                                                  \n");
         printf("\tn = %d\tnrhs = %d\tprocess grid (%d,%d)\t with blocks:
%dx%d\n",n,nrhs,nprow,npcol,nb,nb);
         printf("                                                  \n");
      }
      MPIt1 = MPI Wtime();
      pdgesv ( &n, &nrhs, A, &ione, &ione, descA, ippiv, X, &ione, &ione,
descB, &info );
/*=== se verifica forma LU a matricei
 sleep(2*iam);
cout << "A LU on node " << iam << endl;
for (i = 0; i < np; ++i)
 {
  for (j = 0; j < np; ++j)
  cout << setw(3) << *(A+np*i+j) << "   ";
  cout << endl;
 }
cout << "B loc on node " << iam << endl;
for (i = 0; i < np; ++i)
 {
cout << setw(3) << *(B+i) << "";
cout << endl;
 }
cout << endl;
Cblacs barrier(ictxt, "All");
*/

if(mycol == 0 ){
sleep(iam);
printf("For process (%2d%2d) solutions matrix X:\n",myrow,mycol);
             for (i = 0; i < np; ++i)
                                {
                        for (j = 0; j < nqrhs; ++j)
                              printf( " %6.2f", *(X + i*nqrhs+j) );
```

```
                printf( "\n" );
                  }
}
Cblacs barrier(ictxt, "All");
/* if(mycol == 0 ){
printf("For process (%2d%2d) the solution is  %8.4f %8.4f %8.4f
%8.4f\n",myrow,mycol,X[0],X[1],X[2],X[3]);
      } */
      MPIt2 = MPI Wtime();
      MPIelapsed=MPIt2-MPIt1;
      if( iam==0 ) {
         printf("\ttime MPI     = %f s\n",MPIelapsed);
      }

      if( iam==0 ) {
         printf("                                                    \n");
         printf("\tINFO code returned by PDGESV = %d              \n",info);
         printf("                                                    \n");
      }

//Compute residual ||A * X  - B|| / ( ||X|| * ||A|| * eps * N )
      pdlacpy ( "All", &n, &nrhs, B, &ione, &ione, descB, R   , &ione, &ione,
descB );
      eps = pdlamch ( &ictxt, "Epsilon" );
      pdgemm ( "N", "N", &n, &nrhs, &n, &pone, Acpy, &ione, &ione, descA, X,
&ione, &ione, descB,
            &mone, R, &ione, &ione, descB);
      AnormF = pdlange ( "F", &n, &n   , A, &ione, &ione, descA, work);
      BnormF = pdlange ( "F", &n, &nrhs, B, &ione, &ione, descB, work);
      XnormF = pdlange ( "F", &n, &nrhs, X, &ione, &ione, descB, work);
      RnormF = pdlange ( "F", &n, &nrhs, R, &ione, &ione, descB, work);
      residF = RnormF / ( AnormF * XnormF * eps * ((double) n));

      if ( iam==0 ){
         printf("
\n");
         printf("\t||A * X  - B|| F / ( ||X|| F * ||A|| F * eps * N ) = %e
\n",residF);
         printf("
\n");
         if (residF<10.0e+0)
            printf("\tThe answer is correct.                          \n");
         else
            printf("\tThe answer is suspicious.                       \n");
      }


   if (iam==0){
      printf("END OF TESTS\n");
      printf("*************************************************\n");
      printf("                                                 \n");
   }
/*      free(A);
      free(AA);
      free(Acpy);
      free(B);
      free(BB);
      free(X);
```

```
      free(ippiv);
      free(R); */
Cblacs gridexit(ictxt);
//Cblacs exit(0);
MPI Finalize();
exit(0);
}
```

Program's result

```
[Hancu B S@hpc ScaLAPACK Exemple Curs Online]$ ./mpiCC ScL -o Example2.11.exe
Example2.11.cpp
[Hancu B S@hpc ScaLAPACK Exemple Curs Online]$ /opt/openmpi/bin/mpirun -n 4 -
host compute-0-0  Example2.11.exe
============ REZULT OF THE PROGRAM Example2.11.exe
Global matrix AA:
  1   0.102   0.103   0.104   0.105   0.106   0.107   0.108
0.201     2   0.203   0.204   0.205   0.206   0.207   0.208
0.301   0.302     3   0.304   0.305   0.306   0.307   0.308
0.401   0.402   0.403     4   0.405   0.406   0.407   0.408
0.501   0.502   0.503   0.504     5   0.506   0.507   0.508
0.601   0.602   0.603   0.604   0.605     6   0.607   0.608
0.701   0.702   0.703   0.704   0.705   0.706     7   0.708
0.801   0.802   0.803   0.804   0.805   0.806   0.807     8

Global vector B:
  1
  1
  1
  1
  1
  1
  1
  1

For process ( 0 0) np= 4,nq  4 and A:
  1   0.102   0.105   0.106
0.201     2   0.205   0.206
0.501   0.502     5   0.506
0.601   0.602   0.605     6
For process ( 0 0) np= 4,nqrhs  1 and B:
  1
  1
  1
  1

For process ( 0 1) np= 4,nq  4 and A:
0.103   0.104   0.107   0.108
0.203   0.204   0.207   0.208
0.503   0.504   0.507   0.508
0.603   0.604   0.607   0.608
For process ( 0 1) np= 4,nqrhs  0 and B:
  1
  1
  1
  1

For process ( 1 0) np= 4,nq  4 and A:
0.301   0.302   0.305   0.306
0.401   0.402   0.405   0.406
```

```
0.701   0.702   0.705   0.706
0.801   0.802   0.805   0.806
For process ( 1 0) np= 4,nqrhs  1 and B:
  1
  1
  1
  1

For process ( 1 1) np= 4,nq  4 and A:
  3   0.304   0.307   0.308
0.403     4   0.407   0.408
0.703   0.704     7   0.708
0.803   0.804   0.807     8
For process ( 1 1) np= 4,nqrhs  0 and B:
  1
  1
  1
  1


*********************************************
  Example of ScaLAPACK routine call: (PDGESV)
*********************************************

      n = 8 nrhs = 1    process grid (2,2)       with blocks: 2x2

For process ( 0 0) solutions matrix X:
   0.71
   0.35
   0.14
   0.12
For process ( 1 0) solutions matrix X:
   0.12
   0.09
   0.05
   0.04
      time MPI     = 2.001759 s

      INFO code returned by PDGESV = 0


      ||A * X  - B|| F / ( ||X|| F * ||A|| F * eps * N ) = 2.427904e-02

      The answer is correct.
END OF TESTS
*********************************************

[Hancu_B_S@hpc ScaLAPACK_Exemple_Curs_Online]$
```