2.4 Using Distributed Matrices

After a global vector or matrix has been block-cyclically distributed over a process grid, the user may choose to perform an operation on a portion of the global matrix. This subset of the global matrix is referred to as a ''submatrix'' and is referenced through the use of six arguments in the calling sequence: the number of rows of the submatrix M, the number of columns of the submatrix N, the local array A containing the global array, the row index IA, the column index JA and the array descriptor of the global array DESCA. This argument convention allows for a global view of the matrix operands and the global addressing of distributed matrices as illustrated in Figure 3. This scheme allows the complete specification of the submatrix A(IA:IA+M-1,JA:JA+N-1) on which to be operated. The description of a global dense subarray consists of (M,N,A,IA,JA,DESCA) that consist of:

- the number of rows and columns M and N of the global subarray,
- a pointer to the local array containing the entire global array (A, for example),
- the row and column indices, (IA, JA), in the global array,
- the array descriptor, DESCA, for the global array.

The names of the row and column indices for the global array have the form I<array_name> and J<array_name> ,respectively.

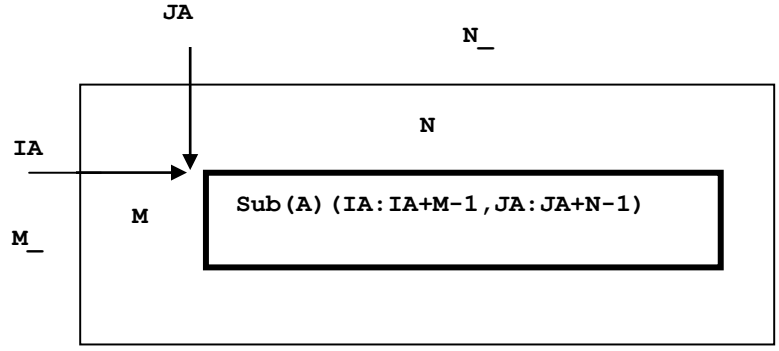

Figure 3

So for using the subroutines of LAPACK/BLAS and ScaLAPACK/PBLAS will used the following subarray syntax.

a) A long-standing design feature of single-processor LAPACK/BLAS routines has been the ability of the user to only use a section of an array - a subarray- in the call to a calculation subroutine.
   - The entire array must still be declared and initialized before the library call.
   - The syntax for specifying a subarray has always followed the same pattern.

b) CALL LAPACK_routine (…, M, N, A(I,J), …).
   - This trio of arguments tells the routine do not use the entire array A, but use a subarrray starting at element A(I,J) and extending for M rows and N columns.

c) The developers of the parallel ScaLAPACK/PBLAS libraries decided to also allow users the ability to use subarrays as the operands for parallel calculations. But the syntax is different.
   - All the preparatory steps must still being carried out before the call to the

ScaLAPACK routines.
- Syntax for ScaLAPACK routines now involves the following arguments.

d) CALL ScaLPACK_routine (…, M, N, LA, IA, JA, DESCA, …).
- These arguments indicate to use a subarray of the global array A starting at element A(IA,JA) and extending for M rows and N columns.
- Notice that the actual array argument - LA - is still the local array holding blocks of the global array A and that the array descriptor vector is (as always) needed.

The next ScaLAPACK tools routine, named INFOG2L, computes the starting local indexes LRINDX, LCINDX corresponding to the distributed submatrix starting globally at the entry pointed by GRINDX, GCINDX. This routine returns the coordinates in the grid of the process owning the matrix entry of global indexes GRINDX, GCINDX, namely RSRC and CSRC.

The syntaxes of the routine INFOG2L is presented below.

```
INFOG2L(GRINDX,GCINDX,DESC,NPROW,NPCOL,MYROW,MYCOL,LRINDX,
        LCINDX,RSRC,CSRC)
int infog2l_(int*,int*,int*,int*,int*,int*,int*,int*,int*,int*,int*);
```

GRINDX (global input) INTEGER
The global row starting index of the submatrix.

GCINDX (global input) INTEGER
The global column starting index of the submatrix.

DESC (input) INTEGER array of dimension DLEN_.
The array descriptor for the underlying distributed matrix.

NPROW (global input) INTEGER
The total number of process rows over which the distributed matrix is distributed.

NPCOL (global input) INTEGER
The total number of process columns over which the distributed matrix is distributed.

MYROW (local input) INTEGER
The row coordinate of the process calling this routine.

MYCOL (local input) INTEGER
The column coordinate of the process calling this routine.

LRINDX (local output) INTEGER
The local rows starting index of the submatrix.

LCINDX (local output) INTEGER
The local columns starting index of the submatrix.

RSRC (global output) INTEGER
The row coordinate of the process that possesses the first row and column of the submatrix.

CSRC (global output) INTEGER
The column coordinate of the process that possesses the first row and column of the submatrix.

You must not confusing INFOG2L with the subroutine pair INXG2L/INDXG2P. The purpose of INFOG2L is: consider a distributed matrix A(1:M,1:N), and consider the submatrix A(GRINDX:M,GCINDX:N) where GRINDX,GCINDX are global indices. Each process will own a (possibly empty) different "slice" of this subarray; and on each process this "slice" will start at specific indices (LRINDX,LCINDX), which are exactly

the output of INFOG2L. RSRC, and CSRC will return the same values to all processes, identifying the process holding the very first entry A(GRINDX,GCINDX) of the submatrix.

Example 2.4.1. *This example illustrates the arrangements for using the utility infog2l_.*

```
#include <string.h>
#include <stdlib.h>
#include <cmath>
#include "mpi.h"
#include <iomanip>
#include <fstream>
#include <iostream>
#include <sstream>
#include <cstdlib>
extern "C" {
// Cblacs declarations
      void Cblacs_pinfo(int*,int*);
      void Cblacs_get(int,int,int*);
      void Cblacs_gridinit(int*,const char*,int,int);
     void Cblacs_gridinfo(int,int*,int*,int*,int*);
      void Cblacs_gridexit(int);
      void Cblacs_exit(int);
     int numroc_(int*, int*, int*, int*, int*);
      void Cblacs_barrier(int,const char*);
      void descinit_(int*,int*,int*,int*,int*,int*,int*,int*,int*,int*);
      int infog2l_(int*,int*,int*,int*,int*,int*,int*,int*,int*,int*,int*);
            }
// function returning the max between two numbers
  int max(int num1, int num2)
  {
    // local variable declaration
    int result;
    if (num1 > num2)
            result = num1;
          else
            result = num2;
    return result;
  }
int main(int argc, char **argv)
{
int myrank_mpi, nprocs_mpi;
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank_mpi);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs_mpi);
int descA[9];
int ictxt, myrow, mycol;
int info, itemp;
int ZERO = 0, ONE = 1;
int grindx,gcindx;
int nprow = 2, npcol = 3;
Cblacs_pinfo(&myrank_mpi, &nprocs_mpi);
Cblacs_get(-1, 0, &ictxt);
Cblacs_gridinit(&ictxt, "Row", nprow, npcol);
Cblacs_gridinfo(ictxt, &nprow, &npcol, &myrow, &mycol);
int m = 9, n = 9;// m-rows, n-columns
int mb = 2, nb = 2;  //mb rows of the bloc, nb column of the bloc
int rsrc = 0, csrc = 0; //Processor grid row and column which has the first block of A
int lrindx,lcindx;
// Determine local dimensions : np-by-nq.
int np = numroc_(&m, &mb, &myrow, &ZERO, &nprow);
int nq = numroc_(&n, &nb, &mycol, &ZERO, &npcol);
int lda = max(1,nq);//max(1,np);
```

```
     info=0;
     descinit_(descA, &m, &n, &mb, &nb, &ZERO,&ZERO,&ictxt, &lda, &info);
      if (myrank_mpi==0)
     {
     printf("======== \n");
      if (info == 0){
         printf("Description init sucesss!\n");
           }
           if(info < 0){
        printf("Error Info = %d, if INFO = -i, the i-th argument had an illegal
             value\n",info);
           }
     }
     Cblacs_barrier(ictxt, "All");
     if ((myrow==1)&(mycol==2))
     {
     printf("For process (%d,%d) descA[0]=%d, descA[1]=%d, descA[2]=%d, descA[3]=%d,
      descA[4]=%d, descA[5]=%d,descA[6]=%d,descA[7]=%d, descA[8]=%d. \n", myrow, mycol,
      descA[0], descA[1], descA[2], descA[3], descA[4],descA[5],descA[6],descA[7],
      descA[8]);
     grindx=6; gcindx=6;
    infog2l_(&grindx,&gcindx,descA,&nprow,&npcol,&myrow,&mycol,&lrindx,&lcindx,&rsrc,&csrc);
     printf("The glabal element is (%d,%d) and process (%d,%d) will own it. \n",grindx,
             gcindx, rsrc,csrc);
     printf("Process (%d,%d) will own a slice of this subarray that start at  local indice
       (%d,%d).\n",myrow, mycol,lrindx,lcindx);
     }
     Cblacs_gridexit(0);
     MPI_Finalize();
     return 0;
     }
```

The result of the execution of the program

```
[MI_gr_TPS1@hpc]$./mpiCC_ScL Example2.4.1.cpp -o Example2.4.1.exe
[MI_gr_TPS1@hpc]$/opt/openmpi/bin/mpirun -n 6 -host compute-0-10,compute-0-12 Example2.4.1.exe
========
Description init sucesss!
For process (1,2) descA[0]=1, descA[1]=0, descA[2]=9, descA[3]=9, descA[4]=2,  descA[5]=2,
descA[6]=0,descA[7]=0, descA[8]=4
The global element is (6,6) and process (0,2) will own it.
Process (1,2) will own a slice of this subarray that start at  local indice(3,2).
```

Attention! The function infog21_ can be used to initialize (to assign the values) the local submatrices on the base of indices of the global matrix. If in the above program it will be added the following fragment.

```
.
.
.
/ == the use of the function infog21_to assign the values of local submatrices on the base of
indices of the global matrix
double *A_loc = NULL;
A_loc = new double[np*nq];// space reservation for local matrices
for (int r = 0; r < m; ++r)
{
        for (int c = 0; c <n; ++c)
{
if (r==c)
{
infog2l_(&r,&c,descA,&nprow,&npcol,&myrow,&mycol,&lrindx,&lcindx,&rsrc,&csrc);
if ((myrow==rsrc)&(mycol==csrc))
{
A_loc[lrindx,lcindx]=100;
printf("On node (%d,%d) A_loc[%d,%d](global[%d,%d])=%f
```

```
\n",myrow,mycol,lrindx,lcindx,r,c,A_loc[lrindx,lcindx]);
//Cblacs_barrier(ictxt, "All");
}
}
}
}
.
.
.
```

then the local matrices there are assigned the values that correspond to the elements on the main diagonal of the global matrix.

```
[MI_gr_TPS1@hpc]$/opt/openmpi/bin/mpirun -n 6 -host compute-0-0,compute-0-1 Example2.4.2.exe
========
Description init success!
Process (0,0) will own a slice of this subarray that start at  local indice (1,1).
Process (0,2) will own a slice of this subarray that start at  local indice (1,1).
Process (1,0) will own a slice of this subarray that start at  local indice (1,1).
Process (0,1) will own a slice of this subarray that start at  local indice (1,1).
Process (1,2) will own a slice of this subarray that start at  local indice (1,1).
Process (1,1) will own a slice of this subarray that start at  local indice (1,1).
On node (0,0) A_loc[1,1](global[1,1])=100.000000
On node (0,0) A_loc[2,2](global[2,2])=100.000000
On node (1,1) A_loc[1,1](global[3,3])=100.000000
On node (1,1) A_loc[2,2](global[4,4])=100.000000
On node (0,2) A_loc[3,1](global[5,5])=100.000000
On node (0,2) A_loc[4,2](global[6,6])=100.000000
On node (1,0) A_loc[3,3](global[7,7])=100.000000
On node (1,0) A_loc[4,4](global[8,8])=100.000000
On node (0,1) A_loc[5,3](global[9,9])=100.000000
```