2.6 BLACS Broadcast Communication Routines

A broadcast sends data possessed by one process to all processes within a scope. Broadcast, much like point to point communication has two complementary operations. The process that owns the data to be broadcast issues a broadcast/send. All processes within the same scope must then issue the complementary broadcast/receive. The BLACS define that both broadcast/send and broadcast/receive are globally-blocking. This has several important implications. The first is that scoped operations (broadcasts or combines) must be strictly ordered, i.e., all processes within a scope must agree on the order of calls to separate scoped operations. This constraint falls in line with that already in place for the computation of message IDs, and is present in point to point communication as well. A less obvious result is that scoped operations with SCOPE='ALL' must be ordered with respect to any other scoped operation. This means that if there are two broadcasts to be done, one along a column, and one involving the entire process grid, all processes within the process column issuing the column broadcast must agree on which broadcast will be performed first.

The topology parameter determines how the messages involved in a distributed operation are sent. There are two main classes of topologies within the BLACS:

- Pipelining topologies (ring-based),
- Non-pipelining topologies (tree-based).

The BLACS have a special default topology to provide for minimizing the single operation time, and this is selected by using the default topology TOP=' '.

A pipeline for increasing direction can be obtained by setting TOP='INCREASING RING' or TOP='I'; a pipeline for codes owing across the processors in the opposite way can be obtained by setting TOP='DECREASING RING' or TOP ='D'.

The syntaxes of the Broadcast/Send routines are the following:

```
vGEBS2D(ICONTXT,SCOPE,TOP,M,N,A,LDA)
void Cdgebs2d(int,char*,char*,int,int,double*,int);
void Cigebs2d(int,char*,char*,int,int,int*,int);

vTRBS2D(ICONTXT,SCOPE,TOP,UPLO,DIAG,M,N,A,LDA)
void Cdtrbs2d(int,char*,char*,char*,char*,int,int,double*,int lda)
```

ICONTXT (input) INTEGER
The BLACS context handle.
SCOPE (input) CHARACTER*1
Scope of processes to participate in operation.
TOP (input) CHARACTER*1
Network topology to be emulated during communication.
UPLO (input) CHARACTER*1
Indicates whether the matrix is upper (UPLO='U') or lower (UPLO='L') trapezoidal.
DIAG (input) CHARACTER*1
Indicates whether the diagonal of the matrix is unit diagonal (DIAG='U'), and thus need not be communicated, or otherwise (DIAG='N').
M (input) INTEGER
The number of matrix rows to be broadcast.
N (input) INTEGER
The number of matrix columns to be broadcast.
A (input) TYPE array (LDA,N)
A pointer to the beginning of the (sub)array to be broadcast.
LDA (input) INTEGER
The leading dimension of the matrix A, i.e., the distance between two successive elements in a matrix row.

The syntaxes of the Broadcast/Receive routines are the following:

```
vGEBR2D(ICONTXT,SCOPE,TOP,M,N,A,LDA,RSRC,CSRC)
void Cdgebr2d(int,char*,char*,int,int,double*,int,int,int);

vTRBR2D(ICONTXT,SCOPE,TOP,UPLO,DIAG,M,N,A,LDA,RSRC,CSRC)
Cdtrbr2d(int,char*,char*,char*,char*,int,int,double*,int,int,int)
```

ICONTXT (input) INTEGER
The BLACS context handle.

SCOPE (input) CHARACTER*1
Scope of processes to participate in operation.
TOP (input) CHARACTER*1
Network topology to be emulated during communication.
UPLO (input) CHARACTER*1
Indicates whether the matrix is upper (UPLO='U') or lower (UPLO='L') trapezoidal.
DIAG (input) CHARACTER*1
Indicates whether the diagonal of the matrix is unit diagonal (DIAG='U'), and thus need not be communicated, or otherwise (DIAG='N').
M (input) INTEGER
The number of matrix rows to be broadcast.
N (input) INTEGER
The number of matrix columns to be broadcast.
A (output) TYPE array (LDA,N)
A pointer to the beginning of the (sub)array to be received/broadcast.
LDA (input) INTEGER
The leading dimension of the matrix A, i.e., the distance between two successive elements in a matrix row.
RSRC (input) INTEGER
Process row coordinate of the source of the broadcast.
CSRC (input) INTEGER
Process column coordinate of the source of the broadcast.

2.6.1 BLACS Combines Routines

In a combine operation, each participating process contributes data which is combined with other processes' data to produce a result. This result can be left on a particular process (called the destination process), or on all participating processes. If the result is left on only one process, we refer to the operation as a *leave-on-one combine*, and if the result is given to all participating processes we reference it as a *leave-on-all combine*. The BLACS package three kinds of combines are supported. They are element-wise summation, element-wise absolute value maximization, and element-wise absolute value minimization of general rectangular arrays. Element-wise indicates that each element of the input array will be combined with the corresponding element from all other processes' arrays to produce the result.

The syntaxes of the Combines routines are the following:

```
vGSUM2D(ICONTXT,SCOPE,TOP,M,N,A,LDA,RDEST,CDEST)
void Cdgsum2d(int,char*,char*,int,int,double*,int,int,int);

vGAMX2D(ICONTXT,SCOPE,TOP,M,N,A,LDA,RA,CA,RCFLAG,RDEST,CDEST)
void Cdgamx2d(int,char*,char*,int,int,double*,int,int*,int*, int,int,int);

vGAMN2D(ICONTXT,SCOPE,TOP,M,N,A,LDA,RA,CA,RCFLAG,RDEST,CDEST)
void Cdgamn2d(int,char*,char*,int,int,double*,int,int*,int*, int,int,int);
```

ICONTXT (input) INTEGER
The BLACS context handle.
SCOPE (input) CHARACTER*1
Scope of processes to participate in operation.
TOP (input) CHARACTER*1
Network topology to be emulated during communication.
M (input) INTEGER
The number of matrix rows to be combined.
N (input) INTEGER
The number of matrix columns to be combined.
A (input/output) TYPE array (LDA,N)
A pointer to the beginning of the (sub)array to be combined.
LDA (input) INTEGER

The leading dimension of the matrix A, i.e., the distance between two successive elements in a matrix row.
RA (output) INTEGER array (RCFLAG,N)

If RCFLAG=-1, this array will not be referenced, and need not exist. Otherwise it is an integer array (of size at least RCFLAGxN) indicating the row index of the process that provided the maximum/minimum. If the calling process is not selected to receive the result, this array will contain intermediate (useless) results.

CA (output) INTEGER array (RCFLAG,N)

If RCFLAG=-1, this array will not be referenced, and need not exist. Otherwise it is an integer array (of size at least RCFLAGxN) indicating the column index of the process that provided the maximum/minimum. If the calling process is not selected to receive the result, this array will contain intermediate (useless) results.

RCFLAG (input) INTEGER

If RCFLAG=-1, then the arrays RA and CA are not referenced and need not exist. Otherwise, RCFLAG indicates the leading dimension of these arrays, and so must be ≥M.

RDEST (input) INTEGER

The process row coordinate of the process who should receive the result. If RDEST or CDEST= -1, all processes within the indicated scope receive the answer.
CDEST (input) INTEGER

The process column coordinate of the process who should receive the result. If RDEST or CDEST= -1, all processes within the indicated scope receive the answer.

Example 2.2 *(Acest exemplu ilustrezamodalitatile de utilizare a functiilor BLACS pentru calcularea in paralel a produsului scala a doi vectori)*

```
/*
This program  does a bone-headed parallel double precision dot product of two vectors.
 Arguments are input on process {0,0}, and output everywhere else.
*/
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sstream>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <cmath>
extern "C" {
// Cblacs and Blas declarations
   void Cblacs_get(int, int, int*);
   void Cblacs_gridinit(int*, const char*, int, int);
   void Cblacs_gridinfo(int,int*,int*,int*,int*);
   void Cdgebs2d(int,char*,char*,int,int,double*,int);
   void Cigebs2d(int,char*,char*,int,int,int*,int);
   void Cigebr2d(int,char*,char*,int,int,int*,int,int,int);
   void Cdgebr2d(int,char*,char*,int,int,double*,int,int,int);
   void Cdgsum2d(int,char*,char*,int,int,double*,int,int,int);
   double ddot_(int*,double*,int*,double*,int*);
   void Cblacs_gridexit(int);
   void Cblacs_exit(int);
      }
int main(int argc, char **argv)
{
int CONTXT, N=100,one = 1;
double *X= NULL, *Y = NULL;
char scope='A',top='I';
// Local Scalars ..
int IAM, NPROCS, NPROW, NPCOL, MYPROW, MYPCOL, I, LN;
double LDDOT;
int procrows = 3, proccols = 1;
// Executable Statements ..
// Find out what grid has been set up, and pretend it's 1-D
Cblacs_get(0, 0, &CONTXT);
Cblacs_gridinit(&CONTXT, "Row-major", procrows, proccols);
```

```
    Cblacs_gridinfo(CONTXT,&NPROW,&NPCOL,&MYPROW,&MYPCOL);
        IAM = MYPROW*NPCOL + MYPCOL;
        NPROCS = NPROW * NPCOL;
    X  = new double[N];
    Y = new double[N];
       // Do bone-headed thing, and just send entire X and Y to * everyone
    if ((MYPROW==0)&(MYPCOL==0))
    {
    for(int i=0;i<N;i++)

       {
       X[i]=rand()/1000000000.0;
          Y[i]=rand()/1000000000.0;
       }
    Cigebs2d(CONTXT,&scope,&top,one,one,&N, one);// Se distribue valoarea lul N
    Cdgebs2d(CONTXT, &scope ,&top,N,1,X, N);// Se distribue vectorul X
    Cdgebs2d(CONTXT, &scope ,&top,N,1,Y, N);// Se distribue vectorul Y
    }
    else
    {
    Cigebr2d(CONTXT,&scope,&top,1,1,&N,1,0,0);
    Cdgebr2d(CONTXT,&scope,&top,N,1,X,N,0,0);
    Cdgebr2d(CONTXT,&scope,&top,N,1,Y,N,0,0);
    }
    // Find out the number of local rows to multiply (LN), and where in vectors to start (I)
        LN = N/NPROCS;
        I = IAM*LN;
    // Last process does any extra rows
        if (IAM == NPROCS-1) LN = LN + N%NPROCS;
    // Figure dot product of my piece of X and Y
    printf("IAM=%d, LN=%d I=%d for (%d,%d) proces \n",IAM,LN,I,MYPROW,MYPCOL);
        LDDOT = ddot_(&LN,&X[I],&one, &Y[I], &one);
    /*=== Fiecare proces determina produsul scaler al X[I] si Y[I] de dimensiunea (LN-1)*one+1 adica produsul
scalar al vectorilor X[I:LN-1)*one+1] si Y[I:LN-1)*one+1]
    ==== */
    printf("Local dot product for (%d,%d) proces is %f \n",MYPROW,MYPCOL,LDDOT);
    // Add local dot products to get global dot product;
    // give all procs the answer
        Cdgsum2d(CONTXT,&scope, &top, 1,1, &LDDOT, 1, -1, 0);
    if ((MYPROW==0)&(MYPCOL==0))
       printf("Global dot product for (%d,%d) proces is %f \n",MYPROW,MYPCOL,LDDOT);
    Cblacs_gridexit(CONTXT);
    Cblacs_exit(0);
    }
```

Rezultatele programului

```
    [UAS_M31@hpc ScaLAPCK_for_C]$ ./mpiCC_ScL Example2.2.cpp -o Example2.2.exe
    [UAS_M31@hpc ScaLAPCK_for_C]$ /opt/openmpi/bin/mpirun -n 3 -host compute-0-10 Example2.2.exe
    IAM=0, LN=33 I=0 for (0,0) proces
    IAM=1, LN=33 I=33 for (1,0) proces
    Local dot product for (1,0) proces is 46.684624
    IAM=2, LN=34 I=66 for (2,0) proces
    Local dot product for (2,0) proces is 38.270497
    Local dot product for (0,0) proces is 40.759342
    Global dot product for (0,0) proces is 125.714463
    [Hancu_B_S@hpc Pentru_Masterat]$
```

The Blas function vDDOT computes a vector-vector dot product and have the sistaxis:

```
    res = vdot(n, x, incx, y, incy)
```

```
res= vdot_(int*,double*,int*,double*,int*);
```

Input Parameters

n
INTEGER. Specifies the number of elements in vectors *x* and *y*.
x
REAL for sdot, DOUBLE PRECISION for ddot
Array, DIMENSION at least (1+(*n*-1)*abs(*incx*)).
incx
INTEGER. Specifies the increment for the elements of *x*.
y
REAL for sdot DOUBLE PRECISION for ddot
Array, DIMENSION at least (1+(*n*-1)*abs(*incy*)).
incy
INTEGER. Specifies the increment for the elements of *y*.

Output Parameters
res
REAL for sdot, DOUBLE PRECISION for *ddot*
Contains the result of the dot product of *x* and *y*, if *n* is positive. Otherwise, *res* contains 0