2.6 Using PDGEADD for Data Paralization

Let present a way to achieve parallelization on data level (i.e. data sharing and distribution for processes) using function PDGEADD.

```
void pdgeadd_(F_CHAR_T 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)
```

Purpose

PDGEADD adds a matrix to another sub(C) := beta*sub(C) + alpha*op(sub(A)), where sub(C) denotes C(IC:IC+M-1,JC:JC+N-1), and op(X) is one of op(X) = X, or op(X) = XT. Thus, op(sub(A)) denotes

1) A(IA:IA+M-1,JA:JA+N-1)   if TRANS = 'N',
2) A(IA:IA+N-1,JA:JA+M-1)' if TRANS = 'T',
3) A(IA:IA+N-1,JA:JA+M-1)' if TRANS = 'C'.

Alpha and beta are scalars, sub(C) and op(sub(A)) are m by n submatrices.

In the other world: to distribute your matrix over process grid (block cyclic 2d distribution) we can do this by means of pdgeadd_ PBLAS routine. This routine computes sum of two matrices C, A: C:=alpha*A+beta*C. Matrices can have different distribution, in particular matrix A can be owned by only one process, thus, setting alpha=1, beta=0 you can simply copy your non-distributed matrix A into distributed matrix C.

Arguments

`TRANS   (global input) CHARACTER*1`

On entry, TRANS specifies the form of op(sub(A)) to be used in the matrix addition as follows:
TRANS = 'N' or 'n'   op(sub(A)) = sub(A),
TRANS = 'T' or 't'   op(sub(A)) = sub(A)',
TRANS = 'C' or 'c'   op(sub(A)) = sub(A)'.

`M       (global input) INTEGER`

On entry, M specifies the number of rows of the submatrix sub(C) and the number of columns of the submatrix sub(A). M must be at least zero.

`N       (global input) INTEGER`

On entry, N specifies the number of columns of the submatrix sub(C) and the number of rows of the submatrix sub(A). N must be at least zero.

`ALPHA   (global input) DOUBLE PRECISION`

On entry, ALPHA specifies the scalar alpha. When ALPHA is supplied as zero then the local entries of the array A corresponding to the entries of the submatrix sub(A) need not be set on input.

`A       (local input) DOUBLE PRECISION array`

On entry, A is an array of dimension (LLD_A, Ka), where Ka is at least Lc(1, JA+M-1). Before entry, this array contains the local entries of the matrix A.

`IA      (global input) INTEGER`

On entry, IA specifies A's global row index, which points to the beginning of the submatrix sub(A).

`JA      (global input) INTEGER`

On entry, JA specifies A's global column index, which points 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_. This s the array *descriptor for the matrix A.*

BETA (global input) DOUBLE PRECISION

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then the local entries of the array C corresponding to the entries of the submatrix sub(C) neednot be set on input.

C (local input/local output) DOUBLE PRECISION array

On entry, C is an array of dimension (LLD_C, Kc), where Kc is at least Lc(1, JC+N-1). Before entry, this array contains the local entries of the matrix C. On exit, the entries of this array corresponding to the local entries of the submatrix sub(C) are overwritten by the local entries of the m by n updated submatrix.

IC (global input) INTEGER

On entry, IC specifies C's global row index, which points to the beginning of the submatrix sub(C).

JC (global input) INTEGER

On entry, JC specifies C's global column index, which points to the beginning of the submatrix sub(C).

DESCC (global and local input) INTEGER array

On entry, DESCC is an integer array of dimension DLEN_. This is the array descriptor for the matrix C.

The following example should demonstrate how the pdgeadd routine is used.

Example 2.6.1. *This example illustrates the modalities of using the BLACS functions for carrying out the operations of parallelization on data level.*

Attention! If the grid of processes is not a square, then the distributions do not correspond to 2D-ciclic algorithm.

```
#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 A_distr(i,j) A_distr[(i)*n+(j)]
#define B(i,j) B[(i)*n+(j]
#define B_distr(i,j) B_distr[(i)*n+(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);
```

```
} // extern "C"
int main(int argc, char **argv) {
// Useful constants
int i_one = 1,  i_zero = 0;
double zero=0.0E+0, one=1.0E+0;
int descA[9],descA_distr[9],descB[9],descB_distr[9];
int iam,nprocs,nprow,npcol,myrow,mycol;
int m,n,mb,nb,mp,nq,nqrhs, nrhs;
int i, j,mypnum;
int lld,lld_distr;
int ictxt,info,lwork;
m=9; n=9;
/*
Where m-number of columns in matrix A, n-number of rows in matrix A
*/
mb=2; nb=2;
nrhs=1;
nprow=2; npcol=2; // Astfel, programul se executa pe 4 procese
double *A, *A_distr,*B, *B_distr, *work,*tau;
// Part with invoking of ScaLAPACK routines. Initialize process grid, first
Cblacs_pinfo(&iam,&nprocs);
Cblacs_get( -1, 0, &ictxt );
Cblacs_gridinit(&ictxt, "R", nprow, npcol );
Cblacs_gridinfo(ictxt, &nprow, &npcol, &myrow, &mycol );
// Matricea A se initializeaza numai pentru procesul cu rankul 0
if ( iam==0 ){
 A = (double*) malloc(m*n*sizeof(double));
 B = (double*) malloc(m*nrhs*sizeof(double));
//input matrix A abd B
for(i=0;i<m;i++ )
     for(j=0;j<n;j++)
             A[i*n+j]=(10*i+j);
for(i=0;i<m;i++ )
for (j=0;j<nrhs;j++)
     B[i*nrhs+j]=i+j;
}
else{
A = NULL;
B = NULL;
//other processes don't contain parts of A
}
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(3) << *(A + n*i + j) << " ";
                   }
                   cout << "\n";
               }
               cout << endl;
cout << "Global vector B:\n";
               for (i = 0; i < m; ++i) {
              for (j = 0; j < nrhs; ++j)
                   cout << setw(3) << *(B + i*nrhs+j ) << "";
                   cout << "\n";
               }
               cout << endl;
}
Cblacs_barrier(ictxt, "All");
// Compute dimensions of local part of distributed matrix A_distr and B_distr
mp = numroc_( &m, &mb, &myrow, &i_zero, &nprow );
nq = numroc_( &n, &nb, &mycol, &i_zero, &npcol );
nqrhs = numroc_( &nrhs, &mb, &mycol, &i_zero, &npcol );
A_distr =(double*) malloc( mp*nq*sizeof(double));
```

```
B_distr = (double*) malloc( mp*nqrhs*sizeof(double)) ;
// Initialize discriptors (local matrix A is considered as distributed with blocking
parameters
// m, n, i.e. there is only one block - whole matrix A - which is located on process
(0,0) )
lld = MAX( numroc_( &m, &n, &myrow, &i_zero, &nprow ), 1 );
descinit_(descA, &m, &n, &m, &n, &i_zero, &i_zero, &ictxt, &lld, &info );
descinit_(descB, &n, &nrhs, &n, &nrhs, &i_zero, &i_zero, &ictxt, &lld, &info );
lld_distr = MAX( mp, 1 );//lld_distr = MAX( nq, 1 );
descinit_(descA_distr, &m, &n, &mb, &nb, &i_zero, &i_zero, &ictxt, &lld_distr, &info );
descinit_(descB_distr, &m, &nrhs, &mb, &nb, &i_zero, &i_zero, &ictxt, &lld_distr, &info
);
// Call pdgeadd_ to distribute matrix (i.e. copy A into A_distr and B into B_dist)
pdgeadd_( "N", &m, &n, &one, A, &i_one, &i_one, descA, &zero, A_distr, &i_one, &i_one,
descA_distr );
pdgeadd_( "N", &m, &nrhs, &one, B, &i_one, &i_one, descB, &zero, B_distr, &i_one,
&i_one, descB_distr);
/*
Here m-number of rows in A_distr, n- number of columns in A_distr
*/
// print A_distr and B_distr
for (int id = 0; id < nprocs; ++id)
{
Cblacs_barrier(ictxt, "All");
if (id == iam) {
printf("Local A(%d*%d) on node %d (%d,%d) \n", mp,nq,iam,myrow,mycol);
//cout << "A_distr on node " << iam << endl;
for (i = 0; i < mp; i++)
 //for (j = 0; j < nq; j++)
 {
  for (j = 0; j < nq; j++)
  //for (i = 0; i < mp; i++)
  cout << setw(3) << *(A_distr+nq*i+j) << " ";//cout << setw(3) << *(A_distr+mp*j+i) <<
" ";
  cout << endl;
 }
if (nqrhs > 0) {
//if (mycol==0) {
printf("Local B(%d*%d) on node %d (%d,%d) \n", mp,nqrhs,iam,myrow,mycol);
//cout << "B_distr on node " << iam << endl;
for (i = 0; i < mp; ++i)
 {
for (j = 0; j < nqrhs; ++j)
cout << setw(3) << *(B_distr+nqrhs*i+j) << " ";
cout << endl;
 }
cout << endl;
             }
Cblacs_barrier(ictxt, "All");
}
}
if( iam==0 ){
for(i=0;i<m;i++ )
     for(j=0;j<n;j++)
             A[i*m+j]=0;
cout << "Global matrix A (pana la restabilire):\n";
               for (i = 0; i < m; ++i) {
                   for (j = 0; j < n; ++j) {
                       cout << setw(3) << *(A + n*i + j) << " ";
                   }
                   cout << "\n";
               }

               cout << endl;
for(i=0;i<m;i++ )
     for(j=0;j<nrhs;j++)
             B[i*nrhs+j]=-10;
cout << "Global matrix B (pana la restabilire):\n";
               for (i = 0; i < m; ++i) {
                   for (j = 0; j < nrhs; ++j) {
                       cout << setw(3) << *(B + nrhs*i + j) << " ";
                   }
                   cout << "\n";
               }
               cout << endl;
}
// Copy result into local matrix (adica "restabilirea matrice A si B)
pdgeadd_( "N", &m, &n, &one, A_distr, &i_one, &i_one, descA_distr, &zero, A, &i_one,
&i_one, descA );
pdgeadd_( "N", &m, &nrhs, &one, B_distr, &i_one, &i_one, descB_distr, &zero, B, &i_one,
&i_one, descB);
// Tipar A si B
if (iam==0)
  {
   cout << "Global matrix A (dupa restabilire):\n";
               for (i = 0; i < m; ++i) {
                   for (j = 0; j < n; ++j) {
                       cout << setw(3) << *(A + n*i + j) << " ";
                   }
                   cout << "\n";
               }
               cout << endl;
cout << "Global vector B (dupa restabilire):\n";
               for (i = 0; i < m; ++i) {
             for (j = 0; j < nrhs; ++j)
                   cout << setw(3) << *(B + i*nrhs+j ) << " ";
                   cout << "\n";
               }
               cout << endl;
}
free( A_distr );free( B_distr );
if( myrow==0 && mycol==0 ){
free( A );
free( B );
}
// End of ScaLAPACK part. Exit process grid.
Cblacs_gridexit(ictxt );
Cblacs_exit( 0);
}
```

Program results.

```
[MI_gr_TPS1@hpc]$ ./mpiCC_ScL -o Example2.6.1.exe Example2.6.1.cpp
[MI_gr_TPS1@hpc]$ /opt/openmpi/bin/mpirun -n 4 -host compute-0-0,compute-0-1
Example2.6.1.exe
Global matrix A:
  0   1   2   3   4   5   6   7   8
 10  11  12  13  14  15  16  17  18
 20  21  22  23  24  25  26  27  28
 30  31  32  33  34  35  36  37  38
 40  41  42  43  44  45  46  47  48
 50  51  52  53  54  55  56  57  58
 60  61  62  63  64  65  66  67  68
 70  71  72  73  74  75  76  77  78
 80  81  82  83  84  85  86  87  88

Global vector B:
  0
  1
  2
  3
  4
  5
  6
  7
  8

Local A(5*5) on node 0 (0,0)
  0   1   4   5   8
 10  11  14  15  18
 40  41  44  45  48
 50  51  54  55  58
 80  81  84  85  88
Local B(5*1) on node 0 (0,0)
  0
  1
  4
  5
  8

Local A(5*4) on node 1 (0,1)
 20  21  24  25
 28  30  31  34
 35  38  60  61
 64  65  68  70
 71  74  75  78
Local A(4*5) on node 2 (1,0)
  2   3   6   7  12
 13  16  17  42  43
 46  47  52  53  56
 57  82  83  86  87
Local B(4*1) on node 2 (1,0)
  2
  3
  6
  7

Local A(4*4) on node 3 (1,1)
 22  23  26  27
 32  33  36  37
 62  63  66  67
 72  73  76  77

Global matrix A (before restoring):
  0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0

Global matrix B (before restoring):
-10
-10
-10
-10
-10
-10
-10
-10
-10

Global matrix A (after restoring):
  0   1   2   3   4   5   6   7   8
 10  11  12  13  14  15  16  17  18
 20  21  22  23  24  25  26  27  28
 30  31  32  33  34  35  36  37  38
 40  41  42  43  44  45  46  47  48
 50  51  52  53  54  55  56  57  58
 60  61  62  63  64  65  66  67  68
 70  71  72  73  74  75  76  77  78
 80  81  82  83  84  85  86  87  88

Global vector B (after restoring):
  0
  1
  2
  3
  4
  5
  6
  7
  8
```

The following table contains a list of all functions BLACS:

BLACS Routine List

	C Name	**Fortran Name**	**Date type** v	**Description**
Initialization	Cblacs_pinfo	BLACS_PINFO		Get initial system information that is required before BLACS is set up
	Cblacs_setup	BLACS_SETUP		Functionally equivalent to blas_pinfo
	Cblacs_get	BLACS_GET		Returns values BLACS is using for internal defaults
	Cblacs_set	BLACS_SET		Sets BLACS internal defaults
	Cblacs_gridinit	BLACS_GRIDINIT		Assigns processors to BLACS process grid
	Cblacs_gridmap	BLACS_GRIDMAP		Assigns processors to BLACS process grid in arbitrary manner
Destruction	Cblacs_freebuff	BLACS_FREEBUFF		Releases BLACS buffer
	Cblacs_gridexit	BLACS_GRIDEXIT		Frees a BLACS context
	Cblacs_abort	BLACS_ABORT		Aborts all BLACS processes
	Cblacs_exit	BLACS_EXIT		Frees all BLACS contexts and allocated memory
Sending	Cvgesd2d	vGESD2D	S D C Z I	General send 2-d
	Cvgebs2d	vGEBS2D	S D C Z I	General broadcast send 2-d
	Cvtrsd2d	vTRSD2D	S D C Z I	Trapezoidal send 2-d
	Cvtrbs2d	vTRBS2D	S D C Z I	Trapezoidal broadcast send 2-d
Receiving	Cvgerv2d	vGERV2D	S D C Z I	General receive
	Cvgebr2d	vGEBR2D	S D C Z I	General broadcast receive
	Cvtrrv2d	vTRRV2D	S D C Z I	Trapezoidal receive
	Cvtrbr2d	vTRBR2D	S D C Z I	Trapezoidal broadcast receive

Combine	Cvgamx2d	vGAMX2D	S D C Z I	General element-wise absolute value maximum
	Cvgamn2d	vGAMN2D	S D C Z I	General element-wise absolute value minimum
	Cvgsum2d	vGSUM2D	S D C Z I	General element-wise summation
Information and Miscellaneous	Cblacs_gridinfo	BLACS_GRIDINFO		Returns information on BLACS grid
	Cblacs_pnum	BLACS_PNUM		Returns system process number
	Cblacs_pcoord	BLACS_PCOORD		Returns row and col in BLACS process grid
	Cblacs_barrier	BLACS_BARRIER		Holds up execution of all processes till all processes call this routine
Non-Standard	Csetpvmtids	SETPVMTIDS		PVM routine, not used
	Cdcputime00	DCPUTIME00		Returns CPU seconds since arbitrary starting point
	Cdwalltime00	DWALLTIME00		Returns wall clock seconds since arbitrary starting point
	Cksendid	SENDID		Returns BLACS message ID
	Cdrecvid	DRECVID		Returns BLACS message ID for receive
	Ckbsid	KBSID		Returns BLACS message ID for source
	Ckbrid	KBRID		Returns BLACS message ID for destination in broadcast