

2.10 PBLAS routines for matrix-matrix operations (Level 3)

```
PvGEMM(TRANSA,TRANSB,M,N,K,ALPHA,A,IA,JA,DESCA,B,IB,JB,DESCB,
  BETA,C,IC,JC,DESCC)
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
);
```

Purpose: PvGEMM performs one of the matrix-matrix operations

$$sub(C) := alpha \times op(sub(A)) \times op(sub(B)) + beta \times sub(C),$$

where $sub(C)$ denotes $C(IC:IC+M-1,JC:JC+N-1)$,

$$op(sub(A)) = \left. \begin{array}{l} A(IA : IA + M - 1, JA : JA + K - 1) \quad \text{if } TRANSA = 'N' \\ A(IA : IA + K - 1, JA : JA + M - 1)' \quad \text{if } TRANSA = 'T' \\ A(IA : IA + K - 1, JA : JA + M - 1)' \quad \text{if } TRANSA = '' \end{array} \right\}$$

$$op(sub(B)) = \left. \begin{array}{l} B(IB : IB + K - 1, JB : JB + N - 1) \quad \text{if } TRANSB = 'N' \\ B(IB : IB + N - 1, JB : JB + K - 1)' \quad \text{if } TRANSB = 'T' \\ B(IB : IB + N - 1, JB : JB + K - 1)' \quad \text{if } TRANSB = 'C' \end{array} \right\}$$

$alpha$ and $beta$ are scalars, and $sub(A)$, $sub(B)$ and $sub(C)$ are distributed matrices, with $op(sub(A))$ an M -by- K distributed matrix, $op(sub(B))$ a K -by- N distributed matrix and $sub(C)$ an M -by- N distributed matrix.

Arguments

TRANSA (global input) CHARACTER

The form of $op(A)$ to be used in the matrix multiplication as follows:

TRANSA = 'N', $op(A) = A$,

TRANSA = 'T', $op(A) = A^T$,

TRANSA = 'C', $op(A) = A^T$.

TRANSB (global input) CHARACTER

The form of $op(B)$ to be used in the matrix multiplication as follows:

TRANSB = 'N', $op(B) = B$,

TRANSB = 'T', $op(B) = B^T$,

TRANSB = 'C', $op(B) = B^T$.

M (global input) INTEGER

The number of rows of the distributed matrices $op(sub(A))$ and $sub(C)$. $M \geq 0$.

N (global input) INTEGER

The number of columns of the distributed matrices $op(sub(B))$ and $sub(C)$. $N \geq 0$.

K (global input) INTEGER

The number of columns of the distributed matrix $op(sub(A))$ and the number of rows of the distributed matrix $op(B)$. $K \geq 0$.

ALPHA (global input) REAL/COMPLEX

On entry, ALPHA specifies the scalar alpha.

A (local input) array of dimension (LLD_A, KLa)

where KLa is $LOCq(JA+K-1)$ when $TRANSA = 'N'$, and is $LOCq(JA+M-1)$ otherwise.

Before entry, this array must contain the local pieces of the distributed matrix $sub(A)$.

IA (global input) INTEGER

The global row index of the submatrix of the distributed matrix A to operate on.

JA (global input) INTEGER

The global column index of the submatrix of the distributed matrix A to operate on.

DESCA (global and local input) INTEGER array of dimension 8

The array descriptor of the distributed matrix A.

B (local input) array of dimension (LLD_B, KLb)

where KLb is LOCq(JB+N-1) when TRANSB = 'N', and is LOCq(JB+K-1) otherwise. Before entry this array must contain the local pieces of the distributed matrix sub(B).

IB (global input) INTEGER

The global row index of the submatrix of the distributed matrix B to operate on.

JB (global input) INTEGER

The global column index of the submatrix of the distributed matrix B to operate on.

DESCB (global and local input) INTEGER array of dimension 8

The array descriptor of the distributed matrix B.

BETA (global input) REAL/COMPLEX

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then sub(Y) need not be set on input.

C (local input/local output) array of dimension (LLD_C, LOCq(JC+N-1))

Before entry, this array must contain the local pieces of the distributed matrix sub(C). On exit, the distributed matrix sub(C) is overwritten by the M-by-N distributed matrix $\alpha \cdot \text{op}(\text{sub}(A)) \cdot \text{op}(\text{sub}(B)) + \beta \cdot \text{sub}(C)$.

IC (global input) INTEGER

The global row index of the submatrix of the distributed matrix C to operate on.

JC (global input) INTEGER

The global column index of the submatrix of the distributed matrix C to operate on.

DESCC (global and local input) INTEGER array of dimension 8

The array descriptor of the distributed matrix C.

Example 3.3 Utilizarea functiei pdgemm_

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "mpi.h"
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sstream>
using namespace std;
#define AA(i,j) AA[(i)*m+(j)]
#define BB(i,j) BB[(i)*k+(j)]
#define CC(i,j) CC[(i)*m+(j)]
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 indx12g_(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 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 * DESC );
} // extern "C"
int main(int argc, char **argv)
{
int i, j;
int iam,nprocs,nprow,npcol,myrow,mycol;
int descA[9],descB[9],descC[9];
int m,n,k,mb,nb,rsrc,csrc,ictxt,llda,lldb,lldc,info;
int lm,ln,lk;
int iloc,jloc;
int ZERO=0,ONE=1;
double alpha, beta;
m=5; n=5; k=5;
mb=2; nb=2;
double *AA = (double*) malloc(m*k*sizeof(double));// matricea "globala" pentru
inmultirea AA*BB=CC
double *BB = (double*) malloc(k*n*sizeof(double));//matricea "globala" pentru inmultirea
AA*BB=CC
double *CC = (double*) malloc(m*n*sizeof(double));//matricea "globala" pentru inmultirea
AA*BB=CC
// initializarea matricelor globale
for(i=0;i<m;i++)
for(j=0;j<k;j++)
AA[i*m+j]=(i+j);
for(i=0;i<k;i++)
for(j=0;j<n;j++)
BB[i*k+j]=j+i;
nprow=2; npcol=2; // Astfel, programul se executa pe 4 procese
Cblacs_pinfo(&iam,&nprocs);
Cblacs_get(-1, 0, &ictxt);
Cblacs_gridinit( &ictxt,"Row",nprow,npcol);
Cblacs_gridinfo(ictxt,&nprow,&npcol,&myrow,&mycol);
rsrc=0; csrc=0;
if (iam==0)
{
printf("===== RESULT OF THE PROGRAM %s \n",argv[0]);
cout << "Global matrix AA:\n";
for (i = 0; i < m; ++i) {
for (j = 0; j < n; ++j) {
cout << setw(3) << *(AA + m*i + j) << " ";
}
}
cout << "\n";
}
}

```

```

    }
    cout << endl;
    cout << "Global matrix BB:\n";
    for (i = 0; i < k; ++i) {
        for (j = 0; j < n; ++j) {
            cout << setw(3) << *(BB + k*i + j) << " ";
        }
        cout << "\n";
    }
    cout << endl;
}
Cblacs_barrier(ictxt, "All");
int mA = numroc_( &m, &mb, &myrow, &rsrc, &nproc );
int kA = numroc_( &k, &nb, &mycol, &rsrc, &npcol );
int kB = numroc_( &k, &mb, &myrow, &rsrc, &nproc );
int nB = numroc_( &n, &nb, &mycol, &rsrc, &npcol );
int mC = numroc_( &m, &mb, &myrow, &rsrc, &nproc );
int nC = numroc_( &n, &nb, &mycol, &rsrc, &nproc );
descinit_(descA, &m, &k, &mb, &nb, &rsrc, &csrc, &ictxt, &mA, &info);
descinit_(descB, &k, &n, &mb, &nb, &rsrc, &csrc, &ictxt, &kB, &info);
descinit_(descC, &m, &n, &mb, &nb, &rsrc, &csrc, &ictxt, &mC, &info);
double *A = (double*) malloc(mA*kA*sizeof(double)); //matricea locala
double *B = (double*) malloc(kB*nB*sizeof(double)); //matricea locala
double *C = (double*) malloc(mC*nC*sizeof(double)); //matricea locala
// se completeaza cu valori matricele locale
for(iloc=0;iloc<mA;iloc++)
    for(jloc=0;jloc<kA;jloc++){
        int fortidl = iloc + 1;
        int fortjdl = jloc + 1;
        i = indx12g_(&fortidl, &mb, &myrow, &ZERO, &nproc)-1;
        j = indx12g_(&fortjdl, &nb, &mycol, &ZERO, &npcol)-1;
        A[jloc*mA+iloc]=AA(i,j);
    }
for(iloc=0;iloc<kB;iloc++)
    for(jloc=0;jloc<nB;jloc++){
        int fortidl = iloc + 1;
        int fortjdl = jloc + 1;
        i = indx12g_(&fortidl, &mb, &myrow, &ZERO, &nproc)-1;
        j = indx12g_(&fortjdl, &nb, &mycol, &ZERO, &npcol)-1;
        B[jloc*kB+iloc]=BB(i,j);
    }
alpha = 1.0; beta = 0.0;
pdgemm_("No Transpose", "No
Transpose",&m,&n,&k,&alpha,A,&ONE,&ONE,descA,B,&ONE,&ONE,descB,&beta,C,&ONE
,&ONE,descC);
Cblacs_barrier(ictxt, "All");
// Print out the matrix product C
for (int id = 0; id < nprocs; ++id)
{
    Cblacs_barrier(ictxt, "All");
    if (id == iam) {
        cout << "C_loc on node " << iam << endl;

```

```

for (i = 0; i < mC; ++i)
{
    for (j = 0; j < nC; ++j)
        cout << setw(3) << *(C+mC*j+i) << " ";
    cout << endl;
}
Cblacs_barrier(ictxt, "All");
}
}
// Se construiesc matricea CC
for(iloc=0;iloc<mC;iloc++)
    for(jloc=0;jloc<nC;jloc++){
        int fortidl = iloc + 1;
        int fortjdl = jloc + 1;
        i = indx12g_(&fortidl, &mb, &myrow, &ZERO, &nproW)-1;
        j = indx12g_(&fortjdl, &nb, &mycol, &ZERO, &npcol)-1;
        CC(i,j)=C[jloc*mC+iloc];
    }
Cblacs_barrier(ictxt, "All");
/** Tipar rezultate finale
for (int id = 0; id < nprocs; ++id)
{
    Cblacs_barrier(ictxt, "All");
    if (id == iam) {
        cout << "Global matrix CC on node " << iam << endl;
        for (i = 0; i < m; ++i) {
            for (j = 0; j < n; ++j)
                cout << setw(3) << *(CC + m*i + j) << " ";
            cout << "\n";
        }
        cout << endl;
    }
    Cblacs_barrier(ictxt, "All");
}
/* Free memory */
free(A);
free(B);
free(C);
/* Release process grid */
Cblacs_gridexit(ictxt);
} /* main */

```

Rezultatele programului

```

[Hancu_B_S@hpc Pentru_Masterat]$ ./mpiCC_ScL -o Example3.3.exe Example3.3.cpp

[Hancu_B_S@hpc Pentru_Masterat]$ /opt/openmpi/bin/mpirun -n 4 -host compute-0-0,compute-0-1 Example3.3.exe
===== RESULT OF THE PROGRAM Example3.3.exe
[Hancu_B_S@hpc Pentru_Masterat]$ ./mpiCC_ScL -o pmm.exe pmm.cpp

```

```
[Hancu_B_S@hpc Pentru_Masterat]$ /opt/openmpi/bin/mpirun -n 4 -host compute-0-0,compute-0-1 pmm.exe
```

```
===== REZULT OF THE PROGRAM pmm.exe
```

```
Global matrix AA:
```

```
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
```

```
Global matrix BB:
```

```
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
```

```
C_loc on node 0
```

```
30 40 70
40 55 100
70 100 190
```

```
C_loc on node 1
```

```
50 60
70 85
130 160
```

```
C_loc on node 2
```

```
50 70 130
60 85 160
```

```
C_loc on node 3
```

```
90 110
110 135
```

```
Global matrix CC on node 0
```

```
30 40 0 0 70
40 55 0 0 100
0 0 0 0 0
0 0 0 0 0
70 100 0 0 190
```

```
Global matrix CC on node 1
```

```
0 0 50 60 0
0 0 70 85 0
0 0 0 0 0
0 0 0 0 0
0 0 130 160 0
```

```
Global matrix CC on node 2
```

```
0 0 0 0 0
0 0 0 0 0
50 70 0 0 130
60 85 0 0 160
0 0 0 0 0
```

```
Global matrix CC on node 3
```

```
0 0 0 0 0
0 0 0 0 0
0 0 90 110 0
```

```
0 0 110 135 0
0 0 0 0 0
```

Pentru verificarea rezultatelor vom folosi algoritmul sequential bazat pe functia dgemm. Textul programului se prezinta mai jos:

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

extern "C" {
void dgemm_(const char &transa, const char &transb,
            const int &l, const int &n, const int &m, const double &alpha,
            const double *a, const int &lda, const double *b, const int &ldb,
            const double &beta, double *c, const int &ldc);
}

int main() {
int i,j,m=5,k=5,n=5;
//Matrizen werden als eindimensionale Arrays gespeichert
double *AA = new double[m*k];
double *BB = new double[k*n];
double *CC = new double[m*n];
//Matrizen fuelln
for(i=0;i<m;i++)
    for(j=0;j<k;j++)
        AA[i*k+j]=i+j; //(10*i+j);
for(i=0;i<k;i++)
    for(j=0;j<n;j++)
        BB[i*n+j]=i+j;//AA[i*5+j]+5;

for( i=0 ; i<m*n ; i++)
    CC[i] = 0.0;
cout << "Global matrix AA:\n";
    for( i = 0; i < m; ++i) {
        for( j = 0; j < k; ++j) {
            cout << setw(3) << *(AA + k*i + j) << " ";
        }
        cout << "\n";
    }
    cout << endl;
cout << "Global matrix BB:\n";
    for( i = 0; i < k; ++i) {
        for( j = 0; j < n; ++j) {
            cout << setw(3) << *(BB + n*i + j) << " ";
        }
    }
}
```

```

        cout << "\n";
    }
    cout << endl;
    double alpha = 1.0;
    double beta = 0.0;
    //dgemm berechnet C=alpha*A*B+beta*C
    dgemm_('N','N',m,n,k,alpha,AA,m,BB,k,beta,CC,m);

    cout << "Global matrix CC:\n " << endl;
    for(i=0 ; i<m ; i++) {
        for(j=0 ; j<n ; j++)
    cout << setw(3) << *(CC + n*i + j) << " ";
        cout << endl;
    }
    cout << endl;

    delete[] AA;
    delete[] BB;
    delete[] CC;

    return 0;
}

```

Rezultatele sunt:

```

[Hancu_B_S@hpc Pentru_Masterat]$ /opt/openmpi/bin/mpirun -n 1 -host compute-0-0,compute-0-4 dgemm1.exe
Global matrix AA:
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8

Global matrix BB:
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8

Global matrix CC:

30 40 50 60 70
40 55 70 85 100
50 70 90 110 130
60 85 110 135 160
70 100 130 160 190

```

Explicatii suplimentare pentru Exemplu 3.3 (vezi fisierul *Pentru Example3.3.docx*)

In programul pmm.cpp paralelizarea la nivel de date este “nerationala” sub aspect de utilizare a memoriei operative, deoarece pentru toate procesele se rezerveaza memorie operative pentru matricele globale! Folosind functia pdgeadd se poate elabora un program in care deja memoria operative se utilizeaza rational. Textul programului este prezentat mai jos.

Example 3.3.1

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "mpi.h"
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sstream>
using namespace std;
#define AA(i,j) AA[(i)*n+(j)]
#define BB(i,j) BB[(i)*k+(j)]
#define CC(i,j) CC[(i)*n+(j)]
#define A(i,j) A[(i)*n+(j)]
#define B(i,j) B[(i)*n+(j)]
#define C(i,j) C[(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 *isrproc, 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 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 * DESC );
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)
{
int i, j;

```

```

int iam,nprocs,nprow,npcol,myrow,mycol;
// descriptori pentru matrici locale
int descA[9],descB[9],descC[9];
// descriptori pentru matrici globale
int descAA[9],descBB[9],descCC[9];
int m,n,k,mb,nb,rsrc,csrc,ictxt,llda,lldb,lldc,info;
int lm,ln,lk;
int iloc,jloc;
int ZERO=0,ONE=1;
double zero=0.0E+0, one=1.0E+0;
int i_one = 1, i_zero = 0;
int lld_AA, lld_BB,lld_CC;
double alpha, beta;
double *AA, *BB,*CC,*A,*B,*C, *work,*tau;
m=5; n=5; k=5;
mb=2; nb=2;
nprow=2; npcol=2; // Astfel, programul se executa pe 4 procese
Cblacs_pinfo(&iam,&nprocs);
Cblacs_get(-1, 0, &ictxt);
Cblacs_gridinit( &ictxt,"Row",nprow,npcol);
Cblacs_gridinfo(ictxt,&nprow,&npcol,&myrow,&mycol);
rsrc=0; csrc=0;
if ( iam==0 )
{
AA = (double*) malloc(m*k*sizeof(double));// matricea "globala" (de dim. m*k) pentru
inmultirea AA*BB=CC
BB = (double*) malloc(k*n*sizeof(double));//matricea "globala" (de dim. k*n) pentru
inmultirea AA*BB=CC
CC = (double*) malloc(m*n*sizeof(double));//matricea "globala" (de dim. m*n) pentru
inmultirea AA*BB=CC
// initializarea matricelor globale
for(i=0;i<m;i++)
for(j=0;j<k;j++)
AA[i*k+j]=(10*i+j)/(i+j);
for(i=0;i<k;i++)
for(j=0;j<n;j++)
BB[i*n+j]=AA[i*5+j]+5; //(i+j);
}
else{
AA = NULL;
BB = NULL;
//other processes don't contain parts of A
}
if (iam==0)
{
printf("===== RESULT OF THE PROGRAM %s \n",argv[0]);
cout << "Global matrix AA:\n";
for (i = 0; i < m; ++i) {
for (j = 0; j < k; ++j) {
cout << setw(3) << *(AA + k*i + j) << " ";
}
cout << "\n";
}
}

```

```

    }
    cout << endl;
    cout << "Global matrix BB:\n";
    for (i = 0; i < k; ++i) {
        for (j = 0; j < n; ++j) {
            cout << setw(3) << *(BB + n*i + j) << " ";
        }
        cout << "\n";
    }
    cout << endl;
}

```

```

Cblacs_barrier(ictxt, "All");
int mA = numroc_( &m, &mb, &myrow, &rsrc, &nrow );
int kA = numroc_( &k, &nb, &mycol, &rsrc, &npcol );
int kB = numroc_( &k, &mb, &myrow, &rsrc, &nrow );
int nB = numroc_( &n, &nb, &mycol, &rsrc, &npcol );
int mC = numroc_( &m, &mb, &myrow, &rsrc, &nrow );
int nC = numroc_( &n, &nb, &mycol, &rsrc, &nrow );
lld_AA = MAX( numroc_( &m, &k, &myrow, &rsrc, &nrow ), 1 );
lld_BB = MAX( numroc_( &k, &n, &myrow, &rsrc, &nrow ), 1 );
lld_CC = MAX( numroc_( &m, &n, &myrow, &rsrc, &nrow ), 1 );
// Initialize descriptors (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) )
descinit_(descAA, &m, &k, &m, &k, &rsrc, &csrc, &ictxt, &lld_AA, &info );
descinit_(descBB, &k, &n, &k, &n, &rsrc, &csrc, &ictxt, &lld_BB, &info );
descinit_(descCC, &m, &n, &m, &n, &rsrc, &csrc, &ictxt, &lld_CC, &info );
descinit_(descA, &m, &k, &mb, &nb, &rsrc, &csrc, &ictxt, &mA, &info);
descinit_(descB, &k, &n, &mb, &nb, &rsrc, &csrc, &ictxt, &kB, &info);
descinit_(descC, &m, &n, &mb, &nb, &rsrc, &csrc, &ictxt, &mC, &info);
A = (double*) malloc(mA*kA*sizeof(double)); //matricea locala
B = (double*) malloc(kB*nB*sizeof(double)); //matricea locala
C = (double*) malloc(mC*nC*sizeof(double)); //matricea locala
// Call pdgeadd_ to distribute matrix (i.e. copy AA into A and BB into B )
pdgeadd_( "N", &m, &k, &one, AA, &i_one, &i_one, descAA, &zero, A, &i_one, &i_one,
descA);
pdgeadd_( "N", &k, &n, &one, BB, &i_one, &i_one, descBB, &zero, B, &i_one, &i_one,
descB);
// Tipar matricele locale A si B
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",kA, mA,iam,myrow,mycol);
        //cout << "A on node " << iam << endl;
        //for (i = 0; i < mA; ++i)
        for (j = 0; j < kA; ++j)
        {
            //for (j = 0; j < kA; ++j)
            for (i = 0; i < mA; ++i)
                cout << setw(3) << *(A+mA*j+i) << " ";
            cout << endl;
        }
    }
}

```

```

    }
    printf("Local B(%d*%d) on node %d (%d,%d) \n", nB,kB,iam,myrow,mycol);
    //cout << "B on node " << iam << endl;
    //for (i = 0; i < kB; ++i)
    for (j = 0; j < nB; ++j)
    {
        //for (j = 0; j < nB; ++j)
        for (i = 0; i < kB; ++i)
            cout << setw(3) << *(B+kB*j+i) << " ";
        cout << endl;
    }
    cout << endl;
}
Cblacs_barrier(ictxt, "All");
}
alpha = 1.0; beta = 0.0;
pdgemm_("No Transpose",&m,&n,&k,&alpha,A,&ONE,&ONE,descA,B,&ONE,&ONE,descB,&beta,C,&ONE
,&ONE,descC);
Cblacs_barrier(ictxt, "All");
// Print out the matrix product C
for (int id = 0; id < nprocs; ++id)
{
    Cblacs_barrier(ictxt, "All");
    if (id == iam) {
        printf("Local C(%d*%d) on node %d (%d,%d) \n", mC,nC,iam,myrow,mycol);
        //cout << "C_loc on node " << iam << endl;
        for (i = 0; i < mC; ++i)
        {
            for (j = 0; j < nC; ++j)
                cout << setw(3) << *(C+mC*j+i) << " ";
            cout << endl;
        }
        Cblacs_barrier(ictxt, "All");
    }
}
// Se construiesc matricea CC

pdgeadd_("N", &m, &n, &one, C, &i_one, &i_one, descC, &zero, CC, &i_one, &i_one,
descCC);

/***/ Tipar rezultate finale

if (iam==0)
{
    cout << "Global matrix CC =AA*BB:\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) << *(CC + n*i + j) << " ";
        }
    }
}

```

```

        cout << "\n";
    }
    cout << endl;
}

/* Free memory */
free(A);
free(B);
free(C);
if( myrow==0 && mycol==0 ){
free( AA );
free( BB);
free( CC);
}

/* Release process grid */
Cblacs_gridexit(ictxt);
} /* main */

```

Rezultatele acestui program sunt:

```

[Hancu_B_S@hpc Pentru_Masterat]$ ./mpiCC_ScL -o Example3.3.1.exeExample3.3.1.cpp
[Hancu_B_S@hpc Pentru_Masterat]$ /opt/openmpi/bin/mpirun -n 4 -host compute-0-
0,compute-0-4 Example3.3.1.exe
===== RESULT OF THE PROGRAM Use_pdgemm_HB.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 BB:
 5  6  7  8  9
15 16 17 18 19
25 26 27 28 29
35 36 37 38 39
45 46 47 48 49

Local A(3*3) on node 0 (0,0)
 0  1  4
10 11 14
40 41 44
Local B(3*3) on node 0 (0,0)
 5  6  9
15 16 19
45 46 49

Local A(2*3) on node 1 (0,1)

```

```
20 21 24
30 31 34
Local B(2*3) on node 1 (0,1)
25 26 29
35 36 39
```

```
Local A(3*2) on node 2 (1,0)
 2 3
12 13
42 43
Local B(3*2) on node 2 (1,0)
 7 8
17 18
47 48
```

```
Local A(2*2) on node 3 (1,1)
22 23
32 33
Local B(2*2) on node 3 (1,1)
27 28
37 38
```

```
Local C(3*3) on node 0 (0,0)
800 1800 4800
835 1885 5035
940 2140 5740
Local C(3*2) on node 1 (0,1)
2800 3800
2935 3985
3340 4540
Local C(2*3) on node 2 (1,0)
870 1970 5270
905 2055 5505
Local C(2*2) on node 3 (1,1)
3070 4170
3205 4355
Global matrix CC =AA*BB:
 800 1800 2800 3800 4800
 835 1885 2935 3985 5035
 870 1970 3070 4170 5270
 905 2055 3205 4355 5505
 940 2140 3340 4540 5740
```

Corectitudinea rezultatelor este confirmata de programul sequential:

```
[Hancu_B_S@hpc Pentru_Masterat]$ ./mpiCC_ScL -o dgemm1.exe dgemm1.cpp
[Hancu_B_S@hpc Pentru_Masterat]$ /opt/openmpi/bin/mpirun -n 1 -host compute-0-0,compute-0-4 dgemm1.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 BB:

5 6 7 8 9
15 16 17 18 19
25 26 27 28 29
35 36 37 38 39
45 46 47 48 49

Global matrix CC:

800 1800 2800 3800 4800
835 1885 2935 3985 5035
870 1970 3070 4170 5270
905 2055 3205 4355 5505
940 2140 3340 4540 5740

```
PvSYMM(SIDE,UPLO,M,N,ALPHA,A,IA,JA,DESCA,B,IB,JB,DESCB,BETA,
C,IC,JC,DESCC)
void pdsymm_(F_CHAR_T SIDE,F_CHAR_T UPLO,int *M,int *N,          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 )
```

Purpose: PvSYMM performs one of the distributed matrix-matrix operations

$sub(C) := \alpha \text{'sub}(A) \text{'sub}(B) + \beta \text{'sub}(C)$, or
 $sub(C) := \alpha \text{'sub}(B) \text{'sub}(A) + \beta \text{'sub}(C)$,
where $sub(C)$ denotes $C(IC:IC+M-1,JC:JC+N-1)$,

$$sub(A) \text{ denotes } \begin{cases} A(IA : IA + M - 1, JA : JA + M - 1) & \text{if SIDE = 'L',} \\ A(IA : IA + N - 1, JA : JA + N - 1) & \text{if SIDE = 'R'} \end{cases}$$

$sub(B)$ denotes $B(IB:IB+M-1,JB:JB+N-1)$.

α and β are scalars, $sub(A)$ is a symmetric distributed matrix and $sub(B)$ and $sub(C)$ are M -by- N distributed matrices.

Arguments

SIDE (global input) CHARACTER

On entry, SIDE specifies whether the symmetric distributed matrix $sub(A)$ appears on the left or right in the operation as follows:

SIDE = 'L' $sub(C) := \alpha * sub(A) * sub(B) + \beta * sub(C)$,

SIDE = 'R' $sub(C) := \alpha * sub(B) * sub(A) + \beta * sub(C)$,

UPLO (global input) CHARACTER

On entry, UPLO specifies whether the upper or lower triangular part of the symmetric distributed matrix $sub(A)$ is to be referenced.

M (global input) INTEGER

The number of rows to be operated on i.e., the number of rows of the distributed submatrix $sub(C)$. $M \geq 0$.

N (global input) INTEGER

The number of columns to be operated on i.e the number of columns of the distributed submatrix sub(C). $N \geq 0$.

ALPHA (global input) REAL/COMPLEX

On entry, ALPHA specifies the scalar alpha.

A (local input) array of dimension (LLD_A, LOC_q(JA+NA-1))

Before entry this array contains the local pieces of the symmetric distributed matrix sub(A), such that when UPLO = 'U', the NA-by-NA upper triangular part of the distributed matrix sub(A) must contain the upper triangular part of the symmetric distributed matrix and the strictly lower triangular part of sub(A) is not referenced, and when UPLO = 'L', the NA-by-NA lower triangular part of the distributed matrix sub(A) must contain the lower triangular part of the symmetric distributed matrix and the strictly lower triangular part of sub(A) is not referenced.

IA (global input) INTEGER

The global row index of the submatrix of the distributed matrix A to operate on.

JA (global input) INTEGER

The global column index of the submatrix of the distributed matrix A to operate on.

DESCA (global and local input) INTEGER array of dimension 8

The array descriptor of the distributed matrix A.

B (local input) array of dimension (LLD_B, LOC_q(JB+N-1))

Before entry, this array contains the local pieces of the distributed matrix sub(B).

IB (global input) INTEGER

The global row index of the submatrix of the distributed matrix B to operate on.

JB (global input) INTEGER

The global column index of the submatrix of the distributed matrix B to operate on.

DESCB (global and local input) INTEGER array of dimension 8

The array descriptor of the distributed matrix B.

BETA (global input) REAL/COMPLEX

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then sub(C) need not be set on input.

C (local input/local output) array of dimension (LLD_C, LOC_q(JC+N-1))

Before entry, this array must contain the local pieces of the distributed matrix sub(C). On exit, the distributed matrix sub(C) is overwritten by the M-by-N updated distributed matrix.

IC (global input) INTEGER

The global row index of the submatrix of the distributed matrix C to operate on.

JC (global input) INTEGER

The global column index of the submatrix of the distributed matrix C to operate on.

DESCC (global and local input) INTEGER array of dimension 8

The array descriptor of the distributed matrix C.

```
PvTRAN(M,N,ALPHA,A,IA,JA,DESCA,BETA,C,IC,JC,DESCC)
```

```
void pdtran_(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: PvTRAN transposes a distributed matrix $sub(C) = beta \cdot sub(C) + alpha \cdot op(sub(A))$ where $sub(C)$ denotes $C(IC:IC+M-1, JC:JC+N-1)$, $sub(A)$ denotes $A(IA:IA+N-1, JA:JA+M-1)$, $op(A)$ denotes A^T . Beta is a scalar, $sub(C)$ is an M-by-N distributed matrix, $sub(A)$ is an N-by-M distributed matrix.

Arguments

M (global input) INTEGER

The number of rows to be operated on i.e., the number of rows of the distributed submatrix $sub(C)$. $M \geq 0$.

N (global input) INTEGER

The number of columns to be operated on i.e the number of columns of the distributed submatrix $sub(C)$. $N \geq 0$.

ALPHA (global input) REAL

On entry, ALPHA specifies the scalar alpha.

A (local input) REAL array of dimension (LLD_A, LOCq(JA+M-1))

This array contains the local pieces of the distributed matrix $sub(A)$.

IA (global input) INTEGER

The global row index of the submatrix of the distributed matrix A to operate on.

JA (global input) INTEGER

The global column index of the submatrix of the distributed matrix A to operate on.

DESCA (global and local input) INTEGER array of dimension 8

The array descriptor of the distributed matrix A.

BETA (global input) REAL

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then $sub(C)$ need not be set on input.

C (local input/local output) array of dimension (LLD_C, LOCq(JC+N-1))

This array contains the local pieces of the distributed matrix $sub(C)$. On exit, the distributed matrix $sub(C)$ is over- written by the updated matrix.

IC (global input) INTEGER

The global row index of the submatrix of the distributed matrix C to operate on.

JC (global input) INTEGER

The global column index of the submatrix of the distributed matrix C to operate on.

DESCC (global and local input) INTEGER array of dimension 8

The array descriptor of the distributed matrix C.

```
PvTRSM(SIDE,UPLO,TRANSA,DIAG,M,N,ALPHA,A,IA,JA,DESCA, B,IB,JB,DESCB )
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)
```

Purpose: PvTRSM solves one of the distributed matrix equations

$op(sub(A)) \prime X = alpha \prime sub(B)$, or $X \prime op(sub(A)) = alpha \prime sub(B)$, where

$$sub(A) \text{ denotes } \begin{cases} A(IA : IA + M - 1, JA : JA + M - 1) & \text{if SIDE = 'L',} \\ A(IA : IA + N - 1, JA : JA + N - 1) & \text{if SIDE = 'R'} \end{cases}$$

$sub(B)$ denotes $B(IB:IB+M-1,JB:JB+N-1)$, $alpha$ is a scalar, X and $sub(B)$ are an M -by- N distributed matrix, $sub(A)$ is a unit, or non-unit, upper or lower triangular distributed matrix and $op(A)$ is one of $op(A) = A$ or $op(A) = A^T$. The distributed matrix X is overwritten on $sub(B)$.

Arguments

SIDE (global input) CHARACTER

On entry, SIDE specifies whether $op(A)$ appears on the left or right of X as follows:

SIDE = 'L', $op(sub(A)) * X = alpha * sub(B)$,

SIDE = 'R', $X * op(sub(A)) = alpha * sub(B)$.

UPLO (global input) CHARACTER

On entry, UPLO specifies whether the distributed matrix $sub(A)$ is an upper or lower triangular distributed matrix.

TRANSA (global input) CHARACTER

The form of $op(A)$ to be used in the matrix multiplication as follows:

TRANSA = 'N', $op(A) = A$,

TRANSA = 'T', $op(A) = A^T$,

TRANSA = 'C', $op(A) = A^T$.

DIAG (global input) CHARACTER

On entry, DIAG specifies whether or not $sub(A)$ is unit triangular as follows:

DIAG = 'U', $sub(A)$ is assumed to be unit triangular,

DIAG = 'N', $sub(A)$ is not assumed to be unit triangular.

M (global input) INTEGER

The number of rows to be operated on i.e., the number of rows of the distributed submatrix $sub(B)$. $M \geq 0$.

N (global input) INTEGER

The number of columns to be operated on i.e., the number of columns of the distributed submatrix $sub(B)$. $N \geq 0$.

ALPHA (global input) REAL/COMPLEX

On entry, ALPHA specifies the scalar alpha.

A (local input) array of dimension (LLD_A, LOCq(JA+NA-1))

Before entry with UPLO = 'U', the leading NA-by-NA upper triangular part of the distributed matrix $sub(A)$ must contain the local pieces of the upper triangular distributed matrix and its strictly lower triangular part is not referenced. Before entry with UPLO = 'L', the leading NA-by-NA lower triangular part of the distributed matrix $sub(A)$ must contain the lower triangular distributed matrix and its strictly upper triangular part is not referenced. Note that when DIAG = 'U', the diagonal elements of $sub(A)$ are not referenced either, but are assumed to be unity.

IA (global input) INTEGER

The global row index of the submatrix of the distributed matrix A to operate on.

JA (global input) INTEGER

The global column index of the submatrix of the distributed matrix A to operate on.

DESCA (global and local input) INTEGER array of dimension 8

The array descriptor of the distributed matrix A.

B (local input) array of dimension (LLD_B, LOCq(JB+N-1))

Before entry, this array contains the local pieces of the distributed matrix $sub(B)$. On exit, $sub(B)$ is overwritten by the solution distributed matrix.

IB (global input) INTEGER

The global row index of the submatrix of the distributed matrix B to operate on.

JB (global input) INTEGER

The global column index of the submatrix of the distributed matrix B to operate on.

DESCB (global and local input) INTEGER array of dimension 8

The array descriptor of the distributed matrix B.

PvSYMM(SIDE,UPLO,M,N,ALPHA,A,IA,JA,DESCA,B,IB,JB,DESCB,BETA,C,IC,JC,DESCC)

```
void pdsymm_(F_CHAR_T SIDE, F_CHAR_T UPLO, int *M, int *N, 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)
```

Purpose: Performs a scalar-matrix-matrix product (one matrix operand is symmetric) and adds the result to a scalar-matrix product for distributed matrices. The operation is defined as

$sub(C) := ALPHA * sub(A) * sub(B) + BETA * sub(C)$ or $sub(C) := ALPHA * sub(B) * sub(A) + BETA * sub(C)$,

where ALPHA and BETA are scalars, $sub(A)$ is a symmetric distributed matrix, $sub(A) = A(IA:IA+M-1, JA:JA+M-1)$, if SIDE = 'L', and $sub(A) = A(IA:IA+N-1, JA:JA+N-1)$, if

SIDE='R', sub(B) and sub(C) are M-by-N distributed matrices, sub(B)=B(IB:IB+M-1,JB:JB+N-1),sub(C)=C(IC:IC+M-1, JC:JC+N-1).

Input Parameters

SIDE (GLOBAL) CHARACTER*1.

Specifies whether the symmetric distributed matrix SUB(A) appears on the left or right in the operation:

if *side* = 'L' or 'l', then $SUB(C):=ALPHA*SUB(A)*SUB(B)+BETA*SUB(C)$;

if *SIDE* = 'R' or 'r', then $SUB(C):=ALPHA*SUB(B)*SUB(A)+BETA*SUB(C)$.

UPLO (global) CHARACTER*1.

Specifies whether the upper or lower triangular part of the symmetric distributed matrix SUB(A) is used:

if *UPLO* = 'U' or 'u', then the upper triangular part is used;

if *uplo* = 'L' or 'l', then the lower triangular part is used.

M (global) INTEGER.

Specifies the number of rows of the distribute submatrix SUB(C), $M \geq 0$.

N(global) INTEGER.

Specifies the number of columns of the distribute submatrix SUB(C), $M \geq 0$.

ALPHA (global) REAL for PSSYMM, DOUBLE PRECISION for PDSYMM, COMPLEX for PCSYMM,DOUBLE COMPLEX for PZSYMM.

Specifies the scalar *ALPHA*.

A (local) REAL for PSSYMM, DOUBLE PRECISION for PDSYMM, COMPLEX for PCSYMM, DOUBLE COMPLEX for PZSYMM. Array, DIMENSION (LLD_A, LOC_q(JA+NA-1)).

Before entry this array must contain the local pieces of the symmetric distributed matrix SUB(A), such that when *UPLO* = 'U' or 'u', the NA-by-NA upper triangular part of the distributed matrix SUB(A) must contain the upper triangular part of the symmetric distributed matrix and the strictly lower triangular part of SUB(A) is not referenced, and when *UPLO* = 'L' or 'l', the NA-by-NA lower triangular part of the distributed matrix SUB(A) must contain the lower triangular part of the symmetric distributed matrix and the strictly upper triangular part of SUB(A) is not referenced.

IA, JA (global) INTEGER.

The row and column indices in the distributed matrix A indicating the first row and the first column of the submatrix SUB(A), respectively.

DESCA (global and local) INTEGER array of dimension 8.

The array descriptor of the distributed matrix A.

B (local) REAL for PSSYMM, DOUBLE PRECISION for PDSYMM, COMPLEX for PCSYMM, DOUBLE COMPLEX for PZSYMM.

Array, DIMENSION (LLD_B, LOC_q(JB+N-1)). Before entry this array must contain the local pieces of the distributed matrix SUB(B).

IB, JB (global) INTEGER.

The row and column indices in the distributed matrix B indicating the first row and the first column of the submatrix SUB(B), respectively.

DESCB (global and local) INTEGER array of dimension 8.

The array descriptor of the distributed matrix B.

BETA (global) REAL for PSSYMM, DOUBLE PRECISION for PDSYMM, COMPLEX for PCSYMM, DOUBLE COMPLEX for PZSYMM.

Specifies the scalar *BETA*. When *BETA* is set to zero, then SUB(C) need not be set on input.

C (local) REAL for PSSYMM, DOUBLE PRECISION for PDSYMM, COMPLEX for PCSYMM, DOUBLE COMPLEX for PZSYMM.

Array, DIMENSION (LLD_C, LOC_q(JC+N-1)). Before entry this array must contain the local pieces of the distributed matrix SUB(C).

IC, JC (global) INTEGER.

The row and column indices in the distributed matrix C indicating the first row and the first column of the submatrix $SUB(C)$, respectively.

DESCC (global and local) INTEGER array of dimension 8.

The array descriptor of the distributed matrix C

Output Parameters

C

Overwritten by the m-by-n updated matrix.

```
PvLACPY(UPLO,M,N,A,IA,JA,DESCA,B,IB,JB,DESCB)
```

```
void pdlacy_(char *uplo,int *m,int *n,double *a,int *ia,int *ja,int *desca,double *b,int *ib,int *jb,int *descb);
```

Purpose: The routine copies all or part of a distributed matrix A to another distributed matrix B . No communication is performed, PvLACPY performs a local copy $sub(A)=sub(B)$, where $sub(A)$ denotes $A(IA:IA+M-1,JA:JA+N-1)$ and $sub(B)$ denotes $B(IB:IB+M-1,JB:JB+N-1)$.

Input Parameters

UPLO (global) CHARACTER.

Specifies the part of the distributed matrix $sub(A)$ to be copied:

= 'U': Upper triangular part; the strictly lower triangular part of $sub(A)$ is not referenced;

= 'L': Lower triangular part; the strictly upper triangular part of $sub(A)$ is not referenced.

Otherwise: all of the matrix $sub(A)$ is copied.

M (global) INTEGER.

The number of rows to be operated on, that is, the number of rows of the distributed submatrix $sub(A)$ ($M \geq 0$).

N (global) INTEGER.

The number of columns to be operated on, that is, the number of columns of the distributed submatrix $sub(A)$ ($N \geq 0$).

A (local) REAL for PSLACPY, DOUBLE PRECISION for PDLACPY, COMPLEX for PCLACPY, COMPLEX*16 for PZLACPY.

Pointer into the local memory to an array of DIMENSION($LLD_A, LOC_C(JA+N-1)$). On entry, this array contains the local pieces of the distributed matrix $sub(A)$.

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 $sub(A)$, 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 A .

Output Parameters

B (local) REAL for PSLACPY, DOUBLE PRECISION for PDLACPY, COMPLEX for PCLACPY, COMPLEX*16 for PZLACPY.

Pointer into the local memory to an array of DIMENSION ($LLD_B, LOCC(JB+N-1)$). This array contains on exit the local pieces of the distributed matrix $sub(B)$ set as follows:

if $UPLO='U'$, $B(IB+I-1,JB+J-1)=A(IA+I-1,JA+J-1), 1 \leq I \leq J, 1 \leq J \leq N$;

if $UPLO='L'$, $B(IB+I-1,JB+J-1)=A(IA+I-1,JA+J-1), J \leq I \leq M, 1 \leq J \leq n$;

otherwise, $B(IB+I-1,JB+J-1)=A(IA+I-1,JA+J-1), 1 \leq I \leq M, 1 \leq J \leq N$.

The PBLAS Level 3 routines perform distributed matrix-matrix operations. [Table "PBLAS Level 3 Routine Groups and Their Data Types"](#) lists the PBLAS Level 3 routine groups and the data types associated with them.

| PBLAS Level 3 Routine Groups and Their Data Types | | |
|---|------------|---|
| Routine Group | Data Types | Description |
| p?geadd | s, d, c, z | Distributed matrix-matrix sum of general matrices |
| p?tradd | s, d, c, z | Distributed matrix-matrix sum of triangular matrices |
| p?gemm | s, d, c, z | Distributed matrix-matrix product of general matrices |
| p?hemm | c, z | Distributed matrix-matrix product, one matrix is Hermitian |
| p?herk | c, z | Rank-k update of a distributed Hermitian matrix |
| p?her2k | c, z | Rank-2k update of a distributed Hermitian matrix |
| p?symm | s, d, c, z | Matrix-matrix product of distributed symmetric matrices |
| p?syrk | s, d, c, z | Rank-k update of a distributed symmetric matrix |
| p?syr2k | s, d, c, z | Rank-2k update of a distributed symmetric matrix |
| p?tran | s, d | Transposition of a real distributed matrix |
| p?tranc | c, z | Transposition of a complex distributed matrix (conjugated) |
| p?tranu | c, z | Transposition of a complex distributed matrix |
| p?trmm | s, d, c, z | Distributed matrix-matrix product, one matrix is triangular |
| p?trsm | s, d, c, z | Solution of a distributed matrix equation, one matrix is triangular |