ScalaPACK

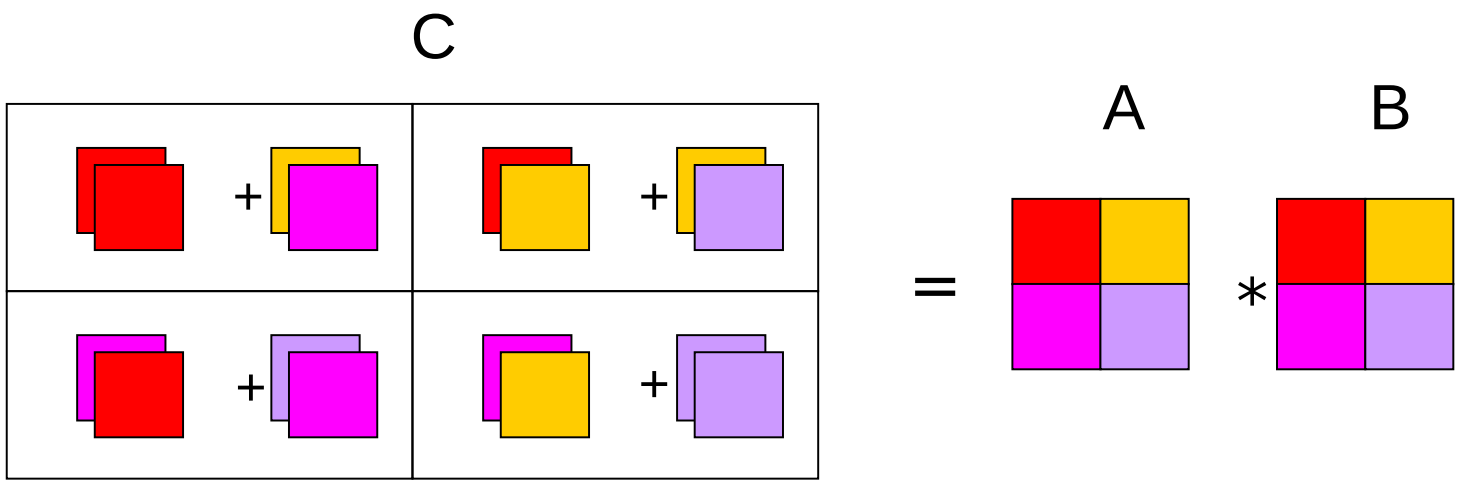
Hands-on

Nicola Spallanzani – n.spallanzani@cineca.it
CINECA - SCAI Department

Block Operations

A block representation of a matrix operation constitutes the basic parallelization strategy for dense matrices.

For instance, a matrix-matrix product can be split in a sequence of smaller operations of the same type acting on subblocks of the original matrix



$$c_{ij} = \sum_{k=1}^{N} a_{ik} \cdot b_{kj}$$

Process Grid

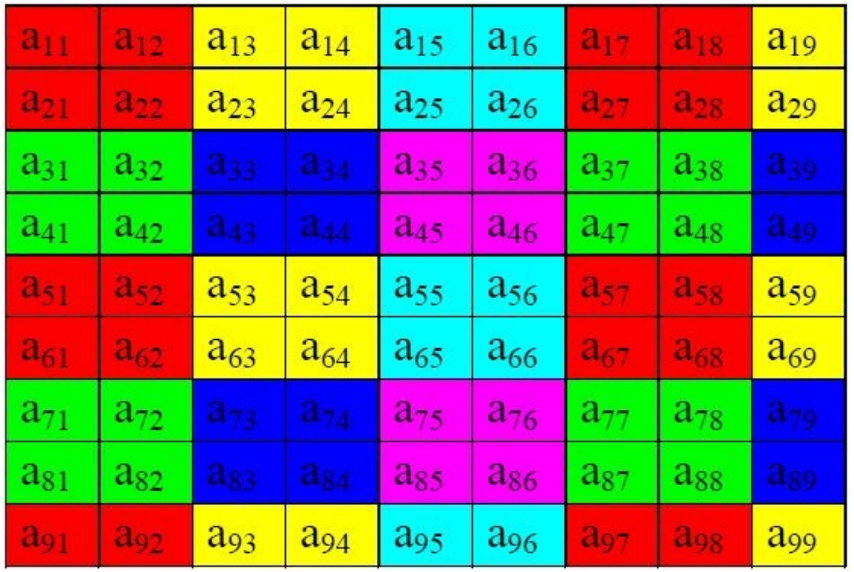
N processes are organized into a logical 2D mesh with p rows and q columns, such that p x q = N

p

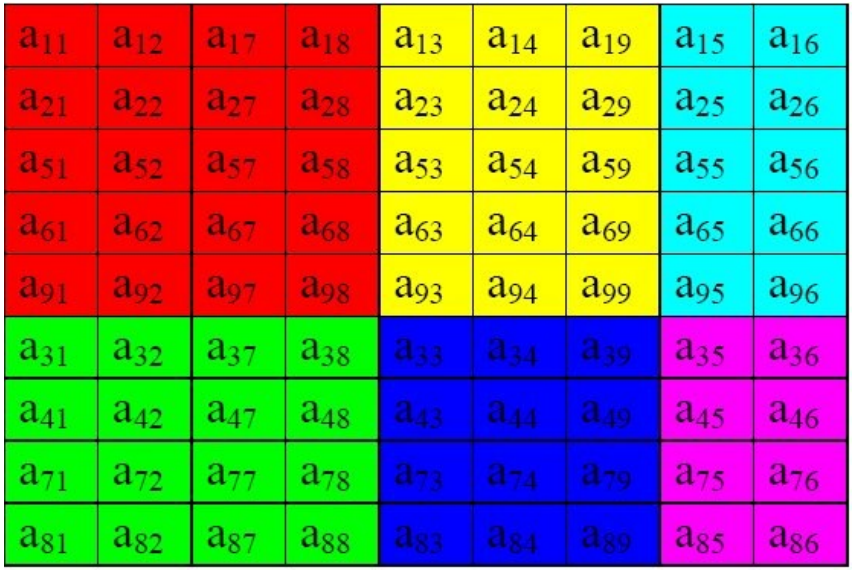
q	0	1	2
0	rank = 0	rank = 1	rank = 2
1	rank = 3	rank = 4	rank = 5

A process is referenced by its coordinates within the grid rather than a single number

Distribution of matrix elements



Logical View (Matrix)



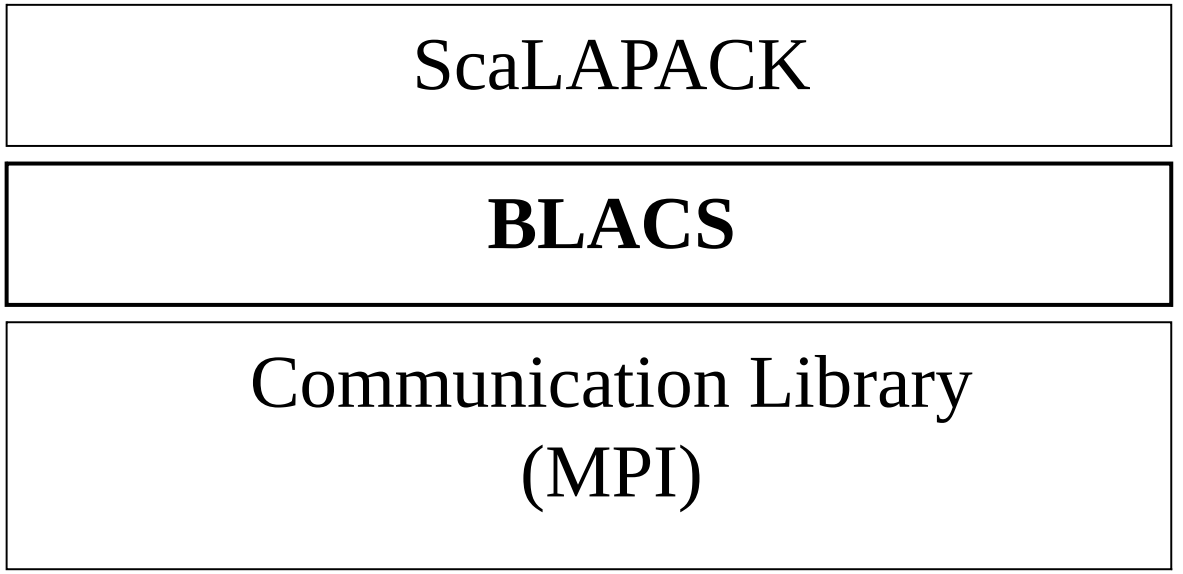
Local View (CPUs)

http://acts.nersc.gov/scalapack/hands-on/datadist.html
http://acts.nersc.gov/scalapack/hands-on/addendum.html

BLACS

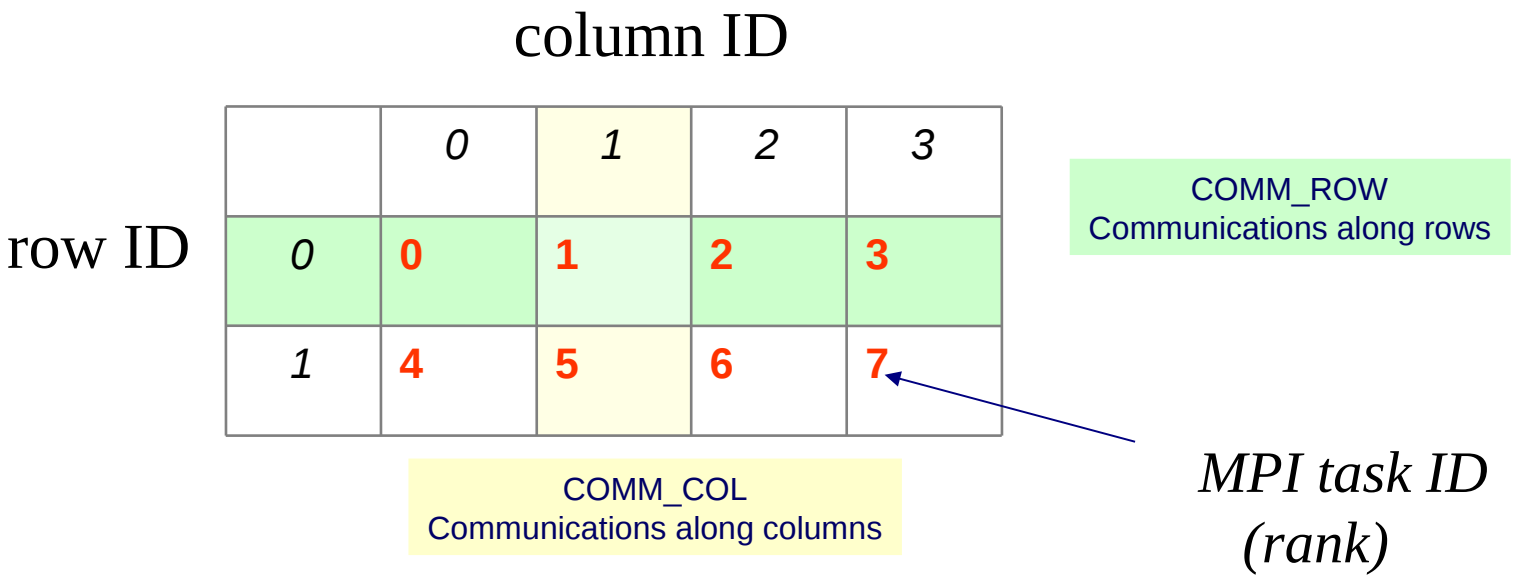
(**B**asic **L**inear **A**lgebra **C**ommunication **S**ubprograms)

The BLACS project is an ongoing investigation whose purpose is to create a linear algebra oriented message passing interface that may be implemented efficiently and uniformly across a large range of distributed memory platforms



BLACS Process Grid

- Processes are distributed on a 2D mesh using row-order or column-order
- (ORDER='R' or 'C'). Each process is assigned a row/column ID as well as a scalar ID



BLACS_GRIDINIT(CONTEXT, ORDER, NPROW, NPCOL)

Initialize a 2D grid of NPROW x NPCOL processes with an order specified

- by ORDER in a given CONTEXT

Context ⟷ *MPI Communicator*

BLACS: Subroutines

BLACS_PINFO(MYPNUM, NPROCS)

Query the system for process ID MYPNUM (output) and number of processes NPROCS (output).

BLACS_GET(CONTEXT, WHAT, VAL)

Query to BLACS environment based on WHAT (input) and ICONTEXT (input)
If WHAT=0, ICONTEX is ignored and the routine returns in VAL (output) a value indicating the default system context

BLACS_GRIDINIT(CONTEXT, ORDER, NPROW, NPCOL)

Initialize a 2D mesh of processes

BLACS_GRIDINFO(CONTEXT, NPROW, NPCOL, MYROW, MYCOL)

Query CONTEXT for the dimension of the grid of processes (NPROW, NPCOL) and for row-ID and col-ID (MYROW, MYCOL)

BLACS_GRIDEXIT(CONTEXT)

Release the 2D mesh associated with CONTEXT

BLACS_EXIT(CONTINUE)

Exit from BLACS environment

BLACS: Subroutines

Point to Point Communication

DGESD2D(CONTEXT,M,N,A,LDA,RDEST,CDEST)

Send matrix A(M,N) to process (RDEST,CDEST)

DGERV2D(CONTEXT,M,N,A,LDA,RSOUR,CSOUR)

Receive matrix A(M,N) from process (RSOUR,CSOUR)

Broadcast

DGEBS2D(CONTEXT,SCOPE,TOP,M,N,A,LDA)

Execute a Broadcast of matrix A(M,N)

DGEBR2D(CONTEXT,SCOPE,TOP,M,N,A,LDA,RSRC,CSRC)

Receive matrix A(M,N) sent from process (RSRC,CSRC) with a broadcast operation

Global reduction

DGSUM2D(CONTEXT,SCOPE,TOP,M,N,A,LDA,RDST,CDST)

Execute a parallel element-wise sum of matrix A(M,N) and store the result in process (RDST,CDST) buffer

http://www.netlib.org/blacs/BLACS/QRef.html

http://www.netlib.org/blacs/f77blacsqref.ps

http://www.netlib.org/blacs/cblacsqref.ps

Template: MPI + BLACS

FORTRAN:

```
CALL MPI_INIT(IERR)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, NPROCS, IERR)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, IERR)

CALL MPI_DIMS_CREATE( NPROCS, 2, dims, IERR)
NPROW = dims(1)  ! cartesian direction 0
NPCOL = dims(2)  ! cartesian direction 1

! Get a default BLACS context
CALL BLACS_GET( -1, 0, CONTEXT )

! Initialize the BLACS context
CALL BLACS_GRIDINIT(CONTEXT, 'R', NPROW, NPCOL)
CALL BLACS_GRIDINFO(CONTEXT, NPROW, NPCOL, &
                    ROWID, COLID)
```

```
! Close BLACS environment
CALL BLACS_GRIDEXIT(CONTEXT)
CALL BLACS_EXIT(0)
! CALL MPI_FINALIZE(IERR)
```

C:

```
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

MPI_Dims_create(nprocs, 2, dims);
nprow = dims[0];  // cartesian direction 0
npcol = dims[1];  // cartesian direction 1

// Get a default BLACS context
Cblacs_get( -1, 0, &context );

// Initialize the BLACS context // char order = 'R';
Cblacs_gridinit(&context, &order, nprow, npcol);
Cblacs_gridinfo(context, &nprow, &npcol,
                &myrow, &mycol);
```

```
// Close BLACS environment
Cblacs_gridexit( context );
Cblacs_exit( 0 );
// MPI_Finalize();
```

Descriptor Initialization

DESCINIT(DESCA, M, N, MB, NB, RSRC, CSRC, ICTXT, LLDA, INFO)

DESCA(9) (global output) matrix A ScaLAPACK Descriptor

M, N (global input) global dimensions of matrix A

MB, NB (global input) blocking factors used to distribute matrix A

RSRC, CSRC (global input) process coordinates over which the first element of A is distributed

ICTXT (global input) BLACS context handle, indicating the global context of the operation on matrix

LLDA (local input) leading dimension of the local array (depends on process!)

Template: DESCRIPTOR

FORTRAN:

```
CALL MPI_INIT(IERR)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, NPROCS, IERR)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, IERR)

CALL MPI_DIMS_CREATE( NPROCS, 2, dims, IERR)
NPROW = dims(1)  ! cartesian direction 0
NPCOL = dims(2)  ! cartesian direction 1

! Get a default BLACS context
CALL BLACS_GET( -1, 0, CONTEXT )

! Initialize the BLACS context
CALL BLACS_GRIDINIT(CONTEXT, 'R', NPROW, NPCOL)
CALL BLACS_GRIDINFO(CONTEXT, NPROW, NPCOL, &
                    ROWID, COLID)

! Descriptor
CALL DESCINIT(descA, M, N, MB, NB, 0, 0, CONTEXT, &
              Mloc, info) ! Mloc -> LLDA

! Close BLACS environment
CALL BLACS_GRIDEXIT(CONTEXT)
CALL BLACS_EXIT(0)
! CALL MPI_FINALIZE(IERR)
```

C:

```
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

MPI_Dims_create(nprocs, 2, dims);
nprow = dims[0];  // cartesian direction 0
npcol = dims[1];  // cartesian direction 1

// Get a default BLACS context
Cblacs_get( -1, 0, &context );

// Initialize the BLACS context // char order = 'R';
Cblacs_gridinit(&context, &order, nprow, npcol);
Cblacs_gridinfo(context, &nprow, &npcol,
                &myrow, &mycol);

// Descriptor
descinit_( descA, &m, &n, &mb, &nb, &zero, &zero,
           &context, &mloc, &info ); // mloc -> LLDA

// Close BLACS environment
Cblacs_gridexit( context );
Cblacs_exit( 0 );
// MPI_Finalize();
```

ScaLAPACK tools

http://www.netlib.org/scalapack/tools

Computation of the local matrix size for a M x N matrix distributed over processes in blocks of dimension MB x NB

```
Mloc = NUMROC( M, MB, ROWID, 0, NPROW )
Nloc = NUMROC( N, NB, COLID, 0, NPCOL )
allocate( Aloc( Mloc, Nloc ) )
```

Computation of local and global indexes

```
iloc = INDXG2L( i, MB, ROWID, 0, NPROW )
jloc = INDXG2L( j, NB, COLID, 0, NPCOL )

i = INDXL2G( iloc, MB, ROWID, 0, NPROW )
j = INDXL2G( jloc, NB, COLID, 0, NPCOL )
```

ScaLAPACK tools

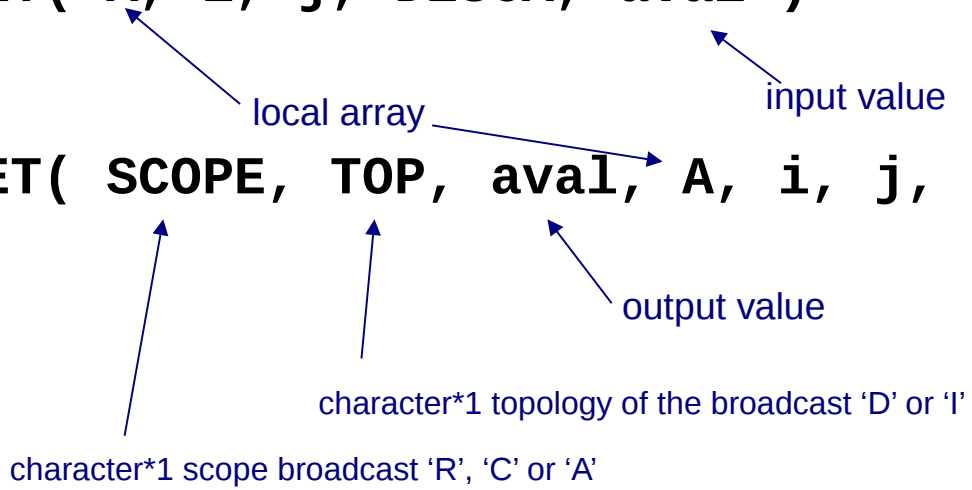
Compute the process to which a certain global element **(i,j)** belongs

```
iprow = INDXG2P( i, MB, ROWID, 0, NPROW )
jpcol = INDXG2P( j, NB, COLID, 0, NPCOL )
```

Define/read a local element, knowing global indexes

```
CALL PDELSET( A, i, j, DESCA, aval )

CALL PDELGET( SCOPE, TOP, aval, A, i, j, DESCA )
```



Template: TOOLS

FORTRAN:

```
CALL MPI_INIT(IERR)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, NPROC, IERR)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, IERR)

CALL MPI_DIMS_CREATE( NPROC, 2, dims, IERR)
NPROW = dims(1)  ! cartesian direction 0
NPCOL = dims(2)  ! cartesian direction 1

! Get a default BLACS context
CALL BLACS_GET( -1, 0, CONTEXT )

! Initialize the BLACS context
CALL BLACS_GRIDINIT(CONTEXT, 'R', NPROW, NPCOL)
CALL BLACS_GRIDINFO(CONTEXT, NPROW, NPCOL, &
                    ROWID, COLID)

! Computation of local matrix size
Mloc = NUMROC( M, MB, myrow, 0, NPROW )
Nloc = NUMROC( N, NB, mycol, 0, NPCOL )
ALLOCATE( A( Mloc, Nloc ) )

! Descriptor
CALL DESCINIT(descA, M, N, MB, NB, 0, 0, CONTEXT, &
              Mloc, info) ! Mloc -> LLDA
```

```
DEALLOCATE( A )

! Close BLACS environment
CALL BLACS_GRIDEXIT(CONTEXT)
CALL BLACS_EXIT(0)
! CALL MPI_FINALIZE(IERR)
```

C:

```
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

MPI_Dims_create(nprocs, 2, dims);
nprow = dims[0];  // cartesian direction 0
npcol = dims[1];  // cartesian direction 1

// Get a default BLACS context
Cblacs_get( -1, 0, &context );

// Initialize the BLACS context // char order = 'R';
Cblacs_gridinit( &context, &order, nprow, npcol);
Cblacs_gridinfo( context, &nprow, &npcol,
                 &myrow, &mycol);

// Computation of local matrix size
mloc = numroc_( &m, &mb, &myrow, &zero, &nprow );
nloc = numroc_( &n, &nb, &mycol, &zero, &npcol );
A = malloc(mloc*nloc*sizeof(double));

// Descriptor
descinit_( descA, &m, &n, &mb, &nb, &zero, &zero,
           &context, &mloc, &info ); // mloc -> LLDA
```

```
free( A );

// Close BLACS environment
Cblacs_gridexit( context );
Cblacs_exit( 0 );
// MPI_Finalize();
```

PBLAS/ScaLAPACK subroutines

Routines name scheme: PXYYZZZ

Parallel

X data type → S = REAL
D = DOUBLE PRECISION
C = COMPLEX
Z = DOUBLE COMPLEX

YY matrix type (GE = general, SY = symmetric, HE = hermitian)

ZZZ algorithm used to perform computation

Some auxiliary functions don't make use of this naming scheme!

PBLAS subroutines

matrix multiplication: C = A * B (level 3)

```
PDGEMM('N', 'N', M, N, L, 1.0d0, A, 1, 1, DESCA, B, 1, 1, DESCB, 0.0d0, C, 1, 1, DESCC)
```

matrix transposition: C = A' (level 3)

```
PDTRAN( M, N, 1.0d0, A, 1, 1, DESCA, 0.0d0, C, 1, 1, DESCC )
```

matrix times vector: Y = A * X (level 2)

```
PDGEMV('N', M, N, 1.0d0, A, 1, 1, DESCA, X, 1, JX, DESCX, 1, 0.0d0, Y, 1, JY, DESCY, 1)
```

row / column swap: X ⇔ Y (level 1)

```
PDSWAP( N, X, IX, JX, DESCX, INCX, Y, IY, JY, DESCY, INCY )
```

scalar product: p = X'·Y (level 1)

```
PDDOT( N, p, X, IX, JX, DESCX, INCX, Y, IY, JY, DESCY, INCY )
```

ScaLAPACK subroutines

Eigenvalues and, optionally, eigenvectors: A Z = w Z

```
PDSYEV( 'V', 'U', N, A, 1, 1, DESCA, W, Z, 1, 1, DESCZ, WORK, LWORK, INFO )
```

'V' compute eigenvalues and eigenvectors
'N' compute eigenvalues only

'U' use upper triangular part of A
'L' use lower triangular part of A

if **lwork = -1**, compute workspace dimension. Return it in **work(1)**

Print matrix

```
PDLAPRNT( M, N, A, 1, 1, DESCA, IR, IC, CMATNM, NOUT, WORK)
```

M	global first dimension of A	**IR, IC**	coordinates of the printing process
N	global second dimension of A	**CMATNM**	character*(*) title of the matrix
A	local part of matrix A	**NOUT**	output fortran units (0 stderr, 6 stdout)
DESCA	descriptor of A	**WORK**	workspace

Practice: complete with ScaLAPACK routines

FORTRAN:

```
CALL MPI_INIT(IERR)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, NPROC, IERR)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, myrank, IERR)

CALL MPI_DIMS_CREATE( NPROC, 2, dims, IERR)
NPROW = dims(1)  ! cartesian direction 0
NPCOL = dims(2)  ! cartesian direction 1

! Get a default BLACS context
CALL BLACS_GET( -1, 0, CONTEXT )

! Initialize the BLACS context
CALL BLACS_GRIDINIT(CONTEXT, 'R', NPROW, NPCOL)
CALL BLACS_GRIDINFO(CONTEXT, NPROW, NPCOL, &
                    ROWID, COLID)

! Computation of local matrix size
Mloc = NUMROC( M, MB, myrow, 0, NPROW )
Nloc = NUMROC( N, NB, mycol, 0, NPCOL )
ALLOCATE( A( Mloc, Nloc ) )

! Descriptor
CALL DESCINIT(descA, M, N, MB, NB, 0, 0, CONTEXT, &
              Mloc, info) ! Mloc -> LLDA

!
! Some operations on matrix A
!

DEALLOCATE( A )

! Close BLACS environment
CALL BLACS_GRIDEXIT(CONTEXT)
CALL BLACS_EXIT(0)
! CALL MPI_FINALIZE(IERR)
```

C:

```
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

MPI_Dims_create(nprocs, 2, dims);
nprow = dims[0];  // cartesian direction 0
npcol = dims[1];  // cartesian direction 1

// Get a default BLACS context
Cblacs_get( -1, 0, &context );

// Initialize the BLACS context // char order = 'R';
Cblacs_gridinit( &context, &order, nprow, npcol);
Cblacs_gridinfo( context, &nprow, &npcol,
                 &myrow, &mycol);

// Computation of local matrix size
mloc = numroc_( &m, &mb, &myrow, &zero, &nprow );
nloc = numroc_( &n, &nb, &mycol, &zero, &npcol );
A = malloc(mloc*nloc*sizeof(double));

// Descriptor
descinit_( descA, &m, &n, &mb, &nb, &zero, &zero,
           &context, &mloc, &info ); // mloc -> LLDA

//
// Some operations on matrix A
//

free( A );

// Close BLACS environment
Cblacs_gridexit( context );
Cblacs_exit( 0 );
// MPI_Finalize();
```


http://hpc.cineca.it/content/scalapack-exercise

How To Compile (GNU)

```
# load these modules on Eurora
module load autoload profile/advanced
module load scalapack/2.0.2--openmpi--1.6.5--gnu--4.6.3

LALIB="-L${SCALAPACK_LIB} -lscalapack \
       -L${LAPACK_LIB} -llapack -L${BLAS_LIB} -lblas"
```

FORTRAN:

```
mpif90 -o program.x program.f90 ${LALIB}
```

How To Compile (GNU)

C:

```
// CBLACS PROTOTYPES
extern void Cblacs_pinfo( int* mypnum, int* nprocs );
extern void Cblacs_get( int context, int request, int* value );
extern int  Cblacs_gridinit( int* context, char* order, int np_row,
                             int np_col );
extern void Cblacs_gridinfo( int context, int* np_row, int* np_col,
                             int* my_row, int* my_col );
extern void Cblacs_gridexit( int context );
extern void Cblacs_exit( int error_code );
extern void Cblacs_barrier( int context, char* scope );
```

How To Compile (GNU)

C:

```
// BLACS/SCALAPACK PROTOTYPES
int numroc_( int* n, int* nb, int* iproc, int* isrcproc, int* nprocs );
void descinit_( int * desca, int * m, int * n, int * mb, int * nb,
     int * irsrc, int * icsrc, int * context,int * llda, int * info );
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 pdelset_( double * A, int * i, int * j, int * desca, double * alpha );
void pdlaprnt_( int * m, int * n, double * A, int * ia, int * ja,
     int * desca, int * irprnt, int * icprn, char * cmatnm, int * nout,
     double * work );
```

```
mpicc -o program.x program.c ${LALIB} -lgfortran
```

How To Compile (INTEL, MKL)

 # load these modules on Eurora

```
module load autoload intelmpi/4.1.1–binary
module load mkl/11.0.1--binary
```

C:

(remember to include mkl.h, mkl_scalapack.h, mkl_blacs.h)

```
mpicc -o program.x program.c -mkl -lmkl_scalapack_lp64 \
      -lmkl_blacs_intelmpi_lp64 -lpthread -lm
```

FORTRAN:

```
mpif90 -o program.x program.f90 -mkl -lmkl_scalapack_lp64 \
       -lmkl_blacs_intelmpi_lp64 -lpthread -lm
```

https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor