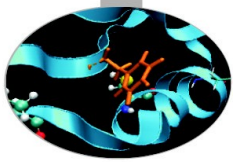
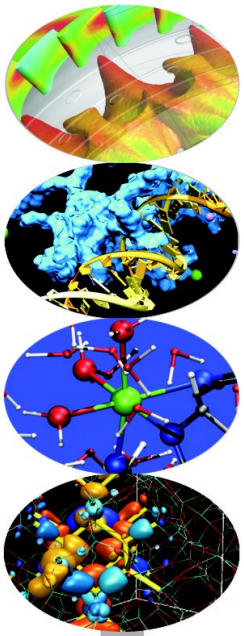
CINECA SCAI
SuperComputing Applications and Innovation



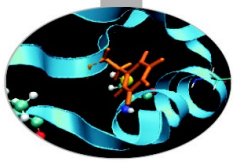


Scalable Linear Algebra

Nicola Spallanzani – n.spallanzani@cineca.it
SuperComputing Applications and Innovation Department

CINECA

Basic Linear Algebra Algorithms



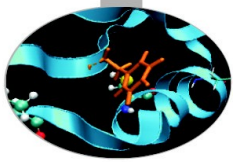
Linear algebra constitutes the core of most technical-scientific applications

Scalar products $$s = \sum_i a_i \cdot b_i$$

Linear Systems $$A_{ij} x_j = b_i$$

Eigenvalue Equations $$A_{ij} x_j = \alpha x_i$$

Algorithms and Libraries



Basic Linear Algebra algorithms are well known and largely available. See for instance:

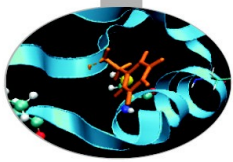
http://www.nr.com

Why should I use libraries?

- They are available on many platforms
- … and they are usually optimized by vendors
- In the case vendor libraries are not installed:

http://www.netlib.org

Standard Linear Algebra Libraries



BLACS

ACML

PETSc

LAPACK

PLASMA

MKL

PSBLAS

ATLAS

MAGMA

BLAS

… but not only

SLEPc

TRILINOS

ESSL

ARPACK

SCALAPACK

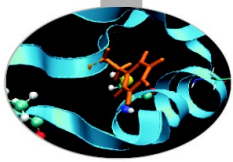
SCOTCH

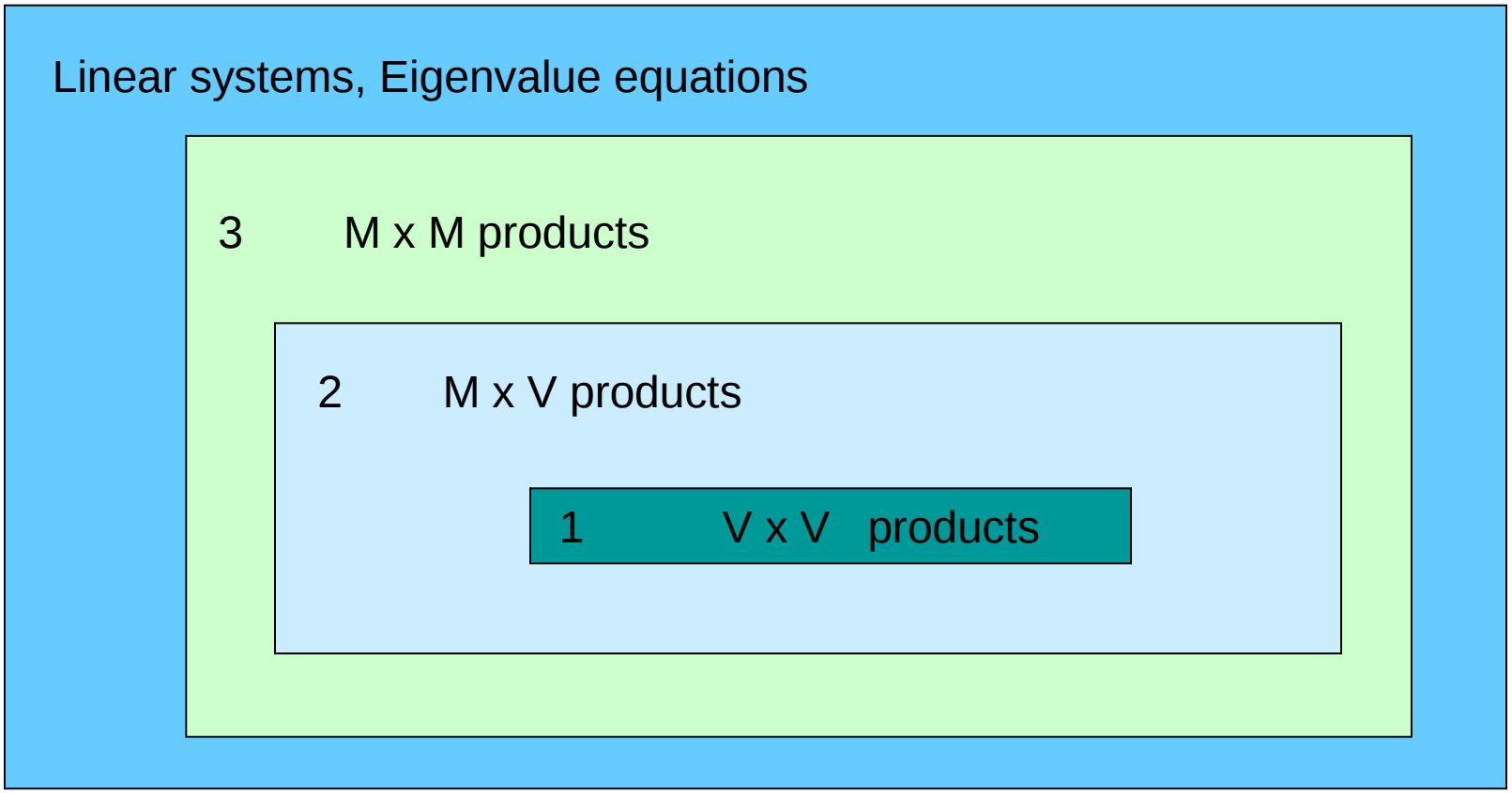
CUBLAS

which library should I use?

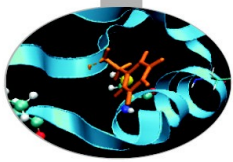
http://www.netlib.org/utk/people/JackDongarra/la-sw.html

Linear Algebra is Hierarchical



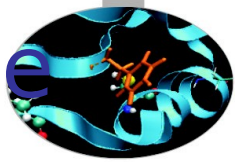


(Parallel) Basic Linear Algebra Subprograms (BLAS and PBLAS)



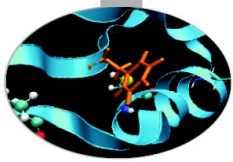
- **Level 1 : Vector - Vector operations**
- **Level 2 : Vector - Matrix operations**
- **Level 3 : Matrix - Matrix operations**

(Scalable) Linear Algebra PACKage (LAPACK and ScaLAPACK)



- **Matrix Decomposition**
- **Linear Equation Systems**
- **Eigenvalue Equations**
- **Linear Least Square Equations**
- **for dense, banded, triangular matrices**
- **for real and complex matrices**

Levels of Routines



- ***Driver*** *routines*
  to solve a complete problem

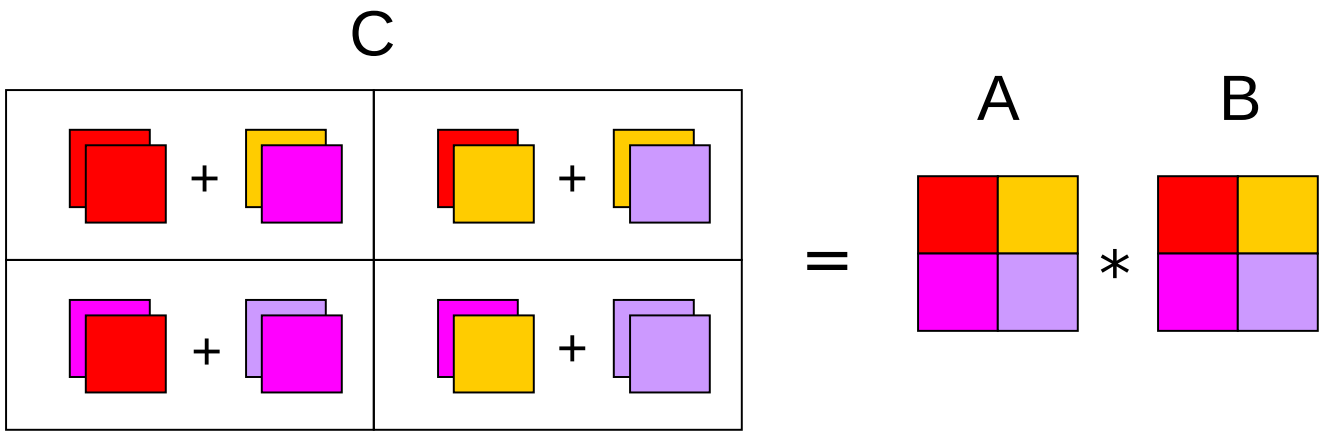
- ***Computational*** *routines*
  to perform a distinct computational task

- ***Auxiliary*** *routines*
  to perform subtasks of block-partitioned algorithms or low-level computations

Block Operations

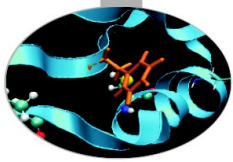
A block representation of a matrix operation constitutes the basic parallelization strategy for dense matrices.

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



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

Example: Partitioning into 2x2 Blocks



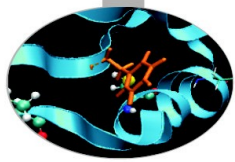
a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}	a_{18}	a_{19}
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	a_{26}	a_{27}	a_{28}	a_{29}
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}	a_{36}	a_{37}	a_{38}	a_{39}
a_{41}	a_{42}	a_{43}	a_{44}	a_{45}	a_{46}	a_{47}	a_{48}	a_{49}
a_{51}	a_{52}	a_{53}	a_{54}	a_{55}	a_{56}	a_{57}	a_{58}	a_{59}
a_{61}	a_{62}	a_{63}	a_{64}	a_{65}	a_{66}	a_{67}	a_{68}	a_{69}
a_{71}	a_{72}	a_{73}	a_{74}	a_{75}	a_{76}	a_{77}	a_{78}	a_{79}
a_{81}	a_{82}	a_{83}	a_{84}	a_{85}	a_{86}	a_{87}	a_{88}	a_{89}
a_{91}	a_{92}	a_{93}	a_{94}	a_{95}	a_{96}	a_{97}	a_{98}	a_{99}

B_{11}	B_{12}	B_{13}	B_{14}	B_{15}
B_{21}	B_{22}	B_{23}	B_{24}	B_{25}
B_{31}	B_{32}	B_{33}	B_{34}	B_{35}
B_{41}	B_{42}	B_{43}	B_{44}	B_{45}
B_{51}	B_{52}	B_{53}	B_{54}	B_{55}

Block Representation

Next Step: distribute blocks among processors

Process Grid



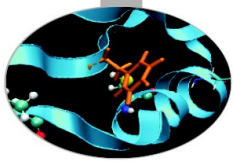
N processes are organized into a logical 2D mesh with p rows and q columns, such that p x q = N

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

Cyclic Distribution of Blocks



B_{11}	B_{12}	B_{13}	B_{14}	B_{15}
B_{21}	B_{22}	B_{23}	B_{24}	B_{25}
B_{31}	B_{32}	B_{33}	B_{34}	B_{35}
B_{41}	B_{42}	B_{43}	B_{44}	B_{45}
B_{51}	B_{52}	B_{53}	B_{54}	B_{55}

p

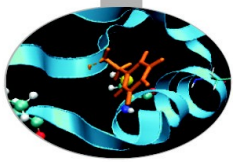
q

p \ q	0		1		2
0	B_{11}	B_{14}	B_{12}	B_{15}	B_{13}
	B_{31}	B_{34}	B_{32}	B_{35}	B_{33}
	B_{51}	B_{54}	B_{52}	B_{55}	B_{53}
1	B_{21}	B_{24}	B_{22}	B_{25}	B_{23}
	B_{41}	B_{44}	B_{42}	B_{45}	B_{43}

$$B_{h,k} \rightarrow (p,q) \qquad p = MOD(N_p + h - 1, N_p)$$
$$q = MOD(N_q + k - 1, N_q)$$

Blocks are distributed on processors in a cyclic manner on each index

Distribution of matrix elements



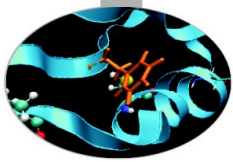
	0		1		2
0	B_{11}	B_{14}	B_{12}	B_{15}	B_{13}
	B_{31}	B_{34}	B_{32}	B_{35}	B_{33}
	B_{51}	B_{54}	B_{52}	B_{55}	B_{53}
1	B_{21}	B_{24}	B_{22}	B_{25}	B_{23}
	B_{41}	B_{44}	B_{42}	B_{45}	B_{43}

The indexes of a single element can be traced back to the processor

	0				1			2	
0	a_{11}	a_{12}	a_{17}	a_{18}	a_{13}	a_{14}	a_{19}	a_{15}	a_{16}
	a_{21}	a_{22}	a_{27}	a_{28}	a_{23}	a_{24}	a_{29}	a_{25}	a_{26}
	a_{51}	a_{52}	a_{57}	a_{58}	a_{53}	a_{54}	a_{59}	a_{55}	a_{56}
	a_{61}	a_{62}	a_{67}	a_{68}	a_{63}	a_{64}	a_{69}	a_{65}	a_{66}
	a_{91}	a_{92}	a_{97}	a_{98}	a_{93}	a_{94}	a_{99}	a_{95}	a_{96}
1	a_{31}	a_{32}	a_{37}	a_{38}	a_{33}	a_{34}	a_{39}	a_{35}	a_{36}
	a_{41}	a_{42}	a_{47}	a_{48}	a_{43}	a_{44}	a_{49}	a_{45}	a_{46}
	a_{71}	a_{72}	a_{77}	a_{78}	a_{73}	a_{74}	a_{79}	a_{75}	a_{76}
	a_{81}	a_{82}	a_{87}	a_{88}	a_{83}	a_{84}	a_{89}	a_{85}	a_{86}

myid=0	myid=1	myid=2	myid=3	myid=4	myid=5
p=0 q=0	p=0 q=1	p=0 q=2	p=1 q=0	p=1 q=1	p=1 q=2

Distribution of matrix elements



a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}	a_{18}	a_{19}
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	a_{26}	a_{27}	a_{28}	a_{29}
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}	a_{36}	a_{37}	a_{38}	a_{39}
a_{41}	a_{42}	a_{43}	a_{44}	a_{45}	a_{46}	a_{47}	a_{48}	a_{49}
a_{51}	a_{52}	a_{53}	a_{54}	a_{55}	a_{56}	a_{57}	a_{58}	a_{59}
a_{61}	a_{62}	a_{63}	a_{64}	a_{65}	a_{66}	a_{67}	a_{68}	a_{69}
a_{71}	a_{72}	a_{73}	a_{74}	a_{75}	a_{76}	a_{77}	a_{78}	a_{79}
a_{81}	a_{82}	a_{83}	a_{84}	a_{85}	a_{86}	a_{87}	a_{88}	a_{89}
a_{91}	a_{92}	a_{93}	a_{94}	a_{95}	a_{96}	a_{97}	a_{98}	a_{99}

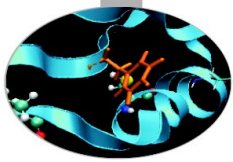
Logical View (Matrix)

a_{11}	a_{12}	a_{17}	a_{18}	a_{13}	a_{14}	a_{19}	a_{15}	a_{16}
a_{21}	a_{22}	a_{27}	a_{28}	a_{23}	a_{24}	a_{29}	a_{25}	a_{26}
a_{51}	a_{52}	a_{57}	a_{58}	a_{53}	a_{54}	a_{59}	a_{55}	a_{56}
a_{61}	a_{62}	a_{67}	a_{68}	a_{63}	a_{64}	a_{69}	a_{65}	a_{66}
a_{91}	a_{92}	a_{97}	a_{98}	a_{93}	a_{94}	a_{99}	a_{95}	a_{96}
a_{31}	a_{32}	a_{37}	a_{38}	a_{33}	a_{34}	a_{39}	a_{35}	a_{36}
a_{41}	a_{42}	a_{47}	a_{48}	a_{43}	a_{44}	a_{49}	a_{45}	a_{46}
a_{71}	a_{72}	a_{77}	a_{78}	a_{73}	a_{74}	a_{79}	a_{75}	a_{76}
a_{81}	a_{82}	a_{87}	a_{88}	a_{83}	a_{84}	a_{89}	a_{85}	a_{86}

Local View (CPUs)

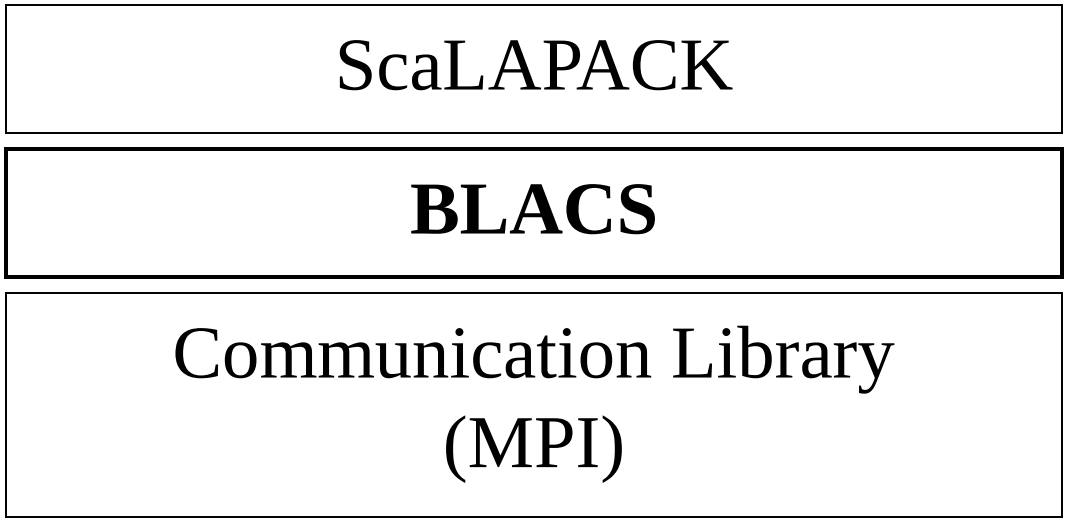
http://acts.nersc.gov/scalapack/hands-on/datadist.html
http://acts.nersc.gov/scalapack/hands-on/addendum.html

BLACS

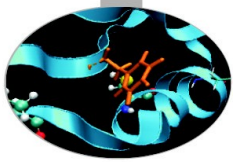


(**B**asic **L**inear **A**lgebra **C**ommunication **S**ubprograms)

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

column ID

row ID

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

COMM_ROW
Communications along rows

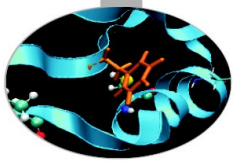
COMM_COL
Communications along columns

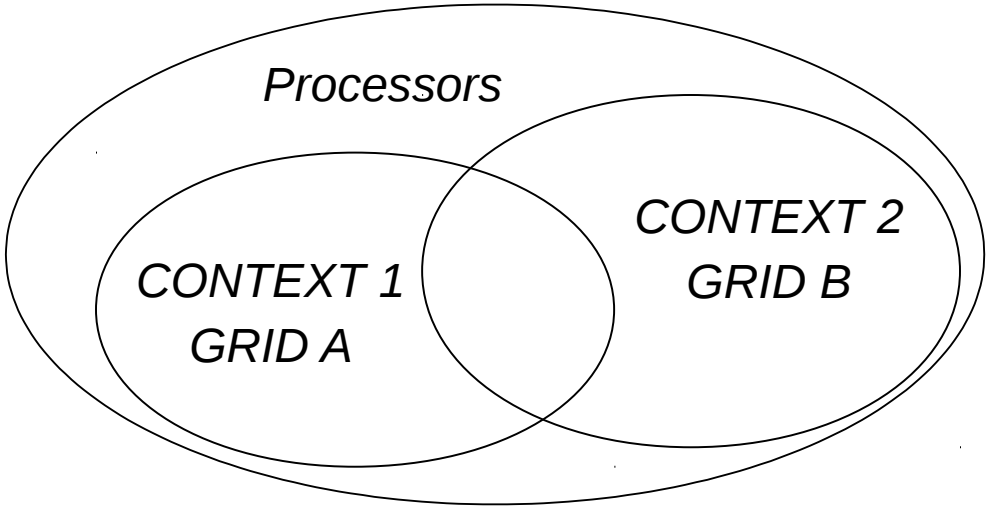
MPI task ID (rank)

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

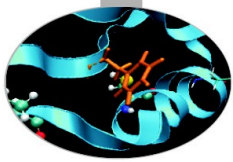
CONTEXT





Context ⟷ *MPI Communicators*

BLACS: Subroutines



BLACS_PINFO(MYPNUM, NPROCS)

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

BLACS_GET(ICONTEXT, 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)

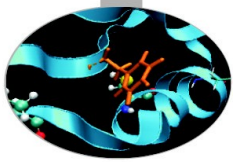
Query CONTEXT for the dimension of the grid of processes (NPROW, NPCOL) and for row-ID and col-ID (MYROW, MYCOL)

BLACS_GRIDEXIT(CONTEXT)

Release the 2D mesh associated with CONTEXT

BLACS_EXIT(CONTINUE) Exit from BLACS environment

BLACS: Subroutines



Point to Point Communication

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

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

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

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

Broadcast

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

Execute a Broadcast of matrix A(M,N)

DGEBR2D(ICONTEX,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(ICONTXT,SCOPE,TOP,M,N,A,LDA,RDST,CDST)

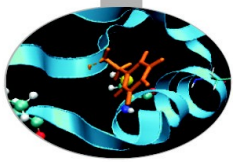
Execute a parallel element-wise sum of matrix A(M,N) and store the result in process (RDST,CDST) buffer

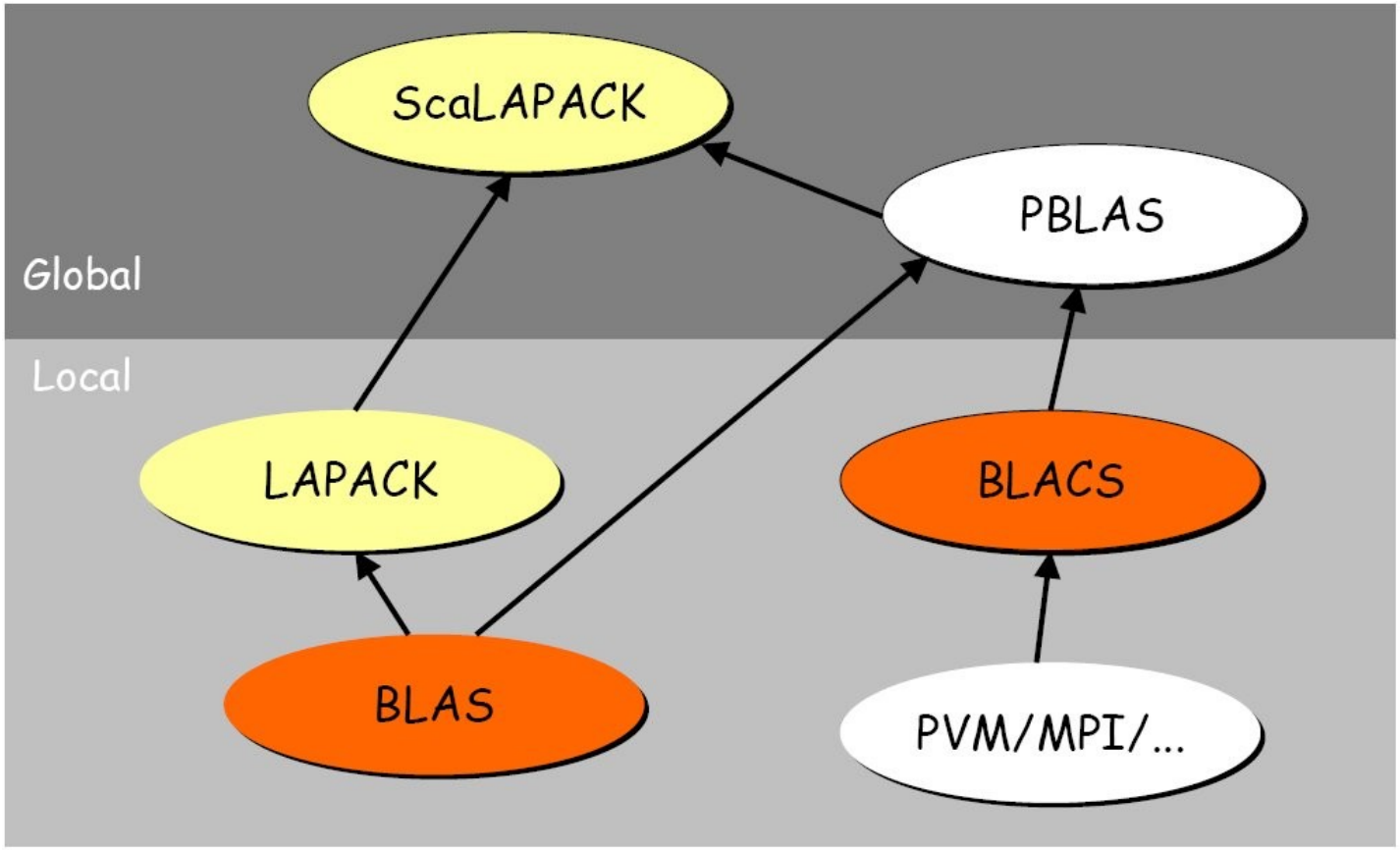
http://www.netlib.org/blacs/BLACS/QRef.html

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

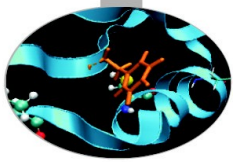
http://www.netlib.org/blacs/cblacsqref.ps

Dependencies



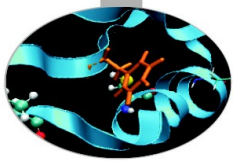


ScaLAPACK and PBLAS: template



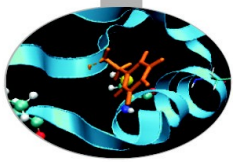
1. *Initialize BLACS*
2. *Initialize BLACS grids*
3. *Distribubute matrix among grid processes (cyclic block distribution)*
4. *Calls to ScaLAPACK/PBLAS routines*
5. *Harvest results*
6. *Release BLACS grids*
7. *Close BLACS environment*

Example:



```
!     Initialize the BLACS

CALL BLACS_PINFO( IAM, NPROCS )

!     Set the dimension of the 2D processors grid

CALL GRIDSETUP( NPROCS, NPROW, NPCOL ) ! User defined

write (*,100) IAM, NPROCS, NPROW, NPCOL
100   format(' MYPE ',I3,', NPE ',I3,', NPE ROW ',I3,', NPE COL ',I3)

!     Initialize a single BLACS context

CALL BLACS_GET( -1, 0, CONTEXT )
CALL BLACS_GRIDINIT( CONTEXT, 'R', NPROW, NPCOL )
CALL BLACS_GRIDINFO( CONTEXT, NPROW, NPCOL, MYROW, MYCOL )
……
……
CALL BLACS_GRIDEXIT( CONTEXT )
CALL BLACS_EXIT( 0 )
```

Descriptor

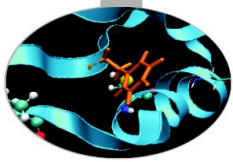


The Descriptor is an integer array that stores the information required to establish the mapping between each global array entry and its corresponding process and memory location.

Each matrix MUST be associated with a Descriptor. Anyhow it's responsibility of the programmer to distribute the matrix coherently with the Descriptor.

```
DESCA( 1 ) = 1          DESCA( 2 ) = ICTXT
DESCA( 3 ) = M          DESCA( 4 ) = N
DESCA( 5 ) = MB         DESCA( 6 ) = NB
DESCA( 7 ) = RSRC       DESCA( 8 ) = CSRC
DESCA( 9 ) = LDA
```

Descriptor Initialization



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

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

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

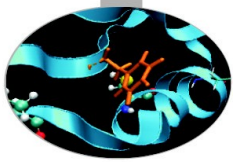
MB, NB (global input) blocking factors used to distribute matrix A

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

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

ScaLAPACK tools



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

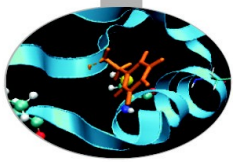
Computation of the local matrix size for a M x N matrix distributed over processes in blocks of dimension MB x NB

```
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 )
```

local array

input value

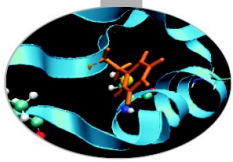
```
CALL PDELGET( SCOPE, TOP, aval, A, i, j, DESCA )
```

output value

character*1 topology of the broadcast ‘D’ or ‘I’

character*1 scope broadcast ‘R’, ‘C’ or ‘A’

PBLAS/ScaLAPACK subroutines



Routines name scheme: PXYYZZZ

Parallel

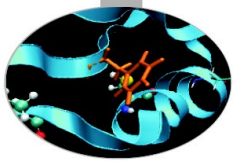
X data type → S = REAL
D = DOUBLE PRECISION
C = COMPLEX
Z = DOUBLE COMPLEX

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

ZZZ algorithm used to perform computation

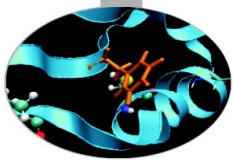
Some auxiliary functions don't make use of this naming scheme!

Calls to ScaLAPACK routines



- It's responsibility of the programmer to correctly distribute a global matrix before calling ScaLAPACK routines
- ScaLAPACK routines are written using a message passing paradigm, therefore each subroutine access directly ONLY local data
- Each process of a given CONTEXT must call the same ScaLAPACK routine...
- … providing in input its local portion of the global matrix
- Operations on matrices distributed on processes belonging to different contexts are not allowed

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)
```

X(1:N,JX:JX)

Y(1:M,JY:JY)

row / column swap: X ⇔ Y (level 1)

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

X(IX,JX:JX+N-1) if INCX = M_X, X(IX:IX+N-1,JX) if INCX = 1 and INCX <> M_X,

Y(IY,JY:JY+N-1) if INCY = M_Y, Y(IY:IY+N-1,JY) if INCY = 1 and INCY <> M_Y.

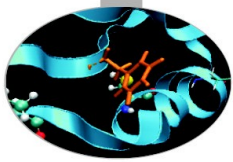
scalar product: p = X'·Y (level 1)

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

X(IX,JX:JX+N-1) if INCX = M_X, X(IX:IX+N-1,JX) if INCX = 1 and INCX <> M_X,

Y(IY,JY:JY+N-1) if INCY = M_Y, Y(IY:IY+N-1,JY) if INCY = 1 and INCY <> M_Y.

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 )
```

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

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

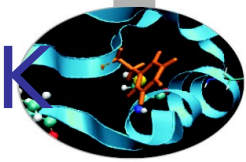
if **lwork = -1**, compute workspace dimension.
Return it in **work(1)**

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

BLAS/LAPACK vs. PBLAS/ScaLAPACK



- **“P” prefix for parallel routines!**
- **The “Leading dimension” turns into a “Descriptor”**
- **Global indexes are additional parameters of the subroutine**

BLAS routine:

```
DGEMM('N', 'N', M, N, L, 1.0, A(1,1), LDA, B(1,1), LDB, 0.0, C(1,1),LDC)
```

PBLAS routine:

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

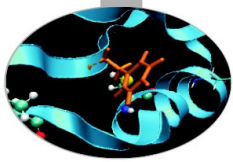
LAPACK routine:

```
DGESV(N, NRHS, A(I,J), LDA, IPIV, B(I,1), LDB, INFO)
```

SCALAPACK routine:

```
PDGESV(N, NRHS, A, I, J, DESCA, IPIV, B, I, 1, DESCB, INFO)
```

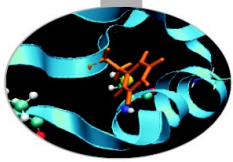
ScaLAPACK Users' Guide



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

At the end of the “Contents” you can find the
“Quick Reference Guides”
for ScaLAPACK, PBLAS and BLACS routines

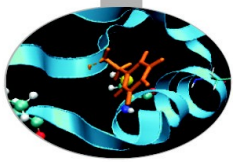
BLACS/ScaLAPACK + MPI



It is quite tricky to write a program using BLACS as a communication library, therefore:

→ MPI and BLACS must be used consistently!

Initialize MPI + BLACS



```
CALL MPI_INIT(IERR)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD,NPROC,IERR)
CALL MPI_COMM_RANK(MPI_COMM_WORLD,MPIME,IERR)
!
comm_world = MPI_COMM_WORLD
!
ndims = 2
dims  = 0
CALL MPI_DIMS_CREATE( NPROC, ndims, dims, IERR)

NPROW = dims(1)  !  cartesian direction 0
NPCOL = dims(2)  !  cartesian direction 1

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

!     Initialize a default BLACS context
CALL BLACS_GRIDINIT(ICONTEXT, 'R', NPROW, NPCOL)
CALL BLACS_GRIDINFO(ICONTEXT, NPROW, NPCOL, ROWID, COLID)

CALL MPI_COMM_SPLIT(comm_world, COLID, ROWID, COMM_COL, IERR)
CALL MPI_COMM_RANK(COMM_COL, coor(1), IERR)
!
CALL MPI_COMM_SPLIT(comm_world, ROWID, COLID, COMM_ROW, IERR)
CALL MPI_COMM_RANK(COMM_ROW, coor(2), IERR)
```

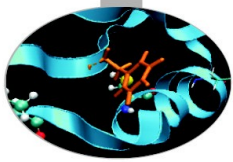
Initialize MPI environment

Compute the dimensions of a 2D mesh compatible with NPROCS processes

Initialize BLACS process grid of size nprow x npcol

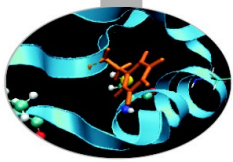
Create a row and a column communicator using BLACS indexes rowid and colid

Matrix redistribution



```
! Distribute matrix A0 (M x N) from root node to all processes in context ictxt.
!
call SL_INIT(ICTXT, NPROW, NPCOL)
call SL_INIT(rootNodeContext, 1, 1) ! create 1 node context
                                    ! for loading matrices
call BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL)
!
! LOAD MATRIX ON ROOT NODE AND CREATE DESC FOR IT
!
if (MYROW == 0 .and. MYCOL == 0) then
   NRU = NUMROC( M, M, MYROW, 0, NPROW )
   call DESCINIT( DESCA0, M, N, M, N, 0, 0, rootNodeContext, max(1, NRU), INFO )
else
   DESCA0(1:9) = 0
   DESCA0(2) = -1
end if
!
!  CREATE DESC FOR DISTRIBUTED MATRIX
!
NRU = NUMROC( M, MB, MYROW, 0, NPROW )
CALL DESCINIT( DESCA, M, N, MB, NB, 0, 0, ICTXT, max(1, NRU), INFO )
!
!  DISTRIBUTE DATA
!
if (debug) write(*,*) "node r=", MYROW, "c=", MYCOL, "M=", M, "N=", N
call PDGEMR2D( M, N, A0, 1, 1, DESCA0, A, 1, 1, DESCA, DESCA( 2 ) )
```

How To Compile (GNU)

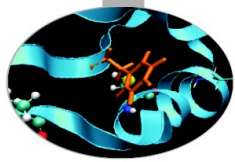


```
# load these modules on Galileo
module load autoload profile/advanced
module load scalapack/2.0.2--openmpi--1.8.4--gnu--4.9.2

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

FORTRAN:

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

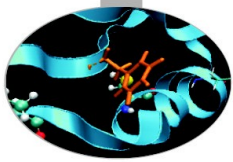
How To Compile (GNU)



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)

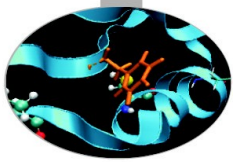


C:

```
// 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 Galileo

```
module load autoload intelmpi/5.1.1–binary
module load mkl/11.3.0--binary
```

C:

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

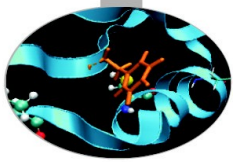
```
mpicc -o program.x program.c -mkl -lmkl_scalapack_lp64 \
      -lmkl_blacs_intelmpi_lp64 -lpthread -lm
```

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

Exercises:



1) Write a program that initializes the BLACS environment, define a matrix and write it to file. Then modifies the program to read the matrix from the previous file and rewrite it to standard output. For I/O use ScaLAPACK routines.

2) Write a program that uses PBLAS routines; at least one routine for each PBLAS level. For example:

Level 1: PDCOPY, PDSCAL, PDNRM2, PDDOT

Level 2: PDGEMV, PDGER

Level 3: PDGEMM

3) Write a program that uses the ScaLAPACK routine PDGESV. Print in files all matrices and vectors generated.

Ax=b ; b(i) = 207-i ;

A(i,j) = 10000 if i=j

A(i,j) = i+j/2 if i≠j

Thanks for your attention!