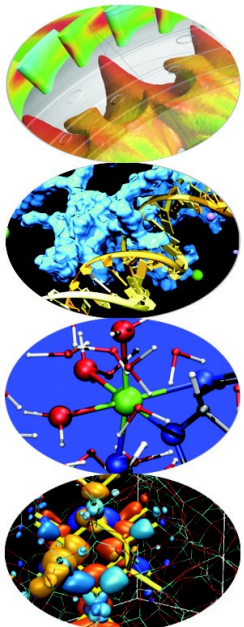
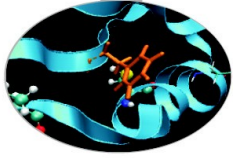


# Scalable Linear Algebra

**Nicola Spallanzani** - [n.spallanzani@cineca.it](mailto:n.spallanzani@ Cineca.it)  
SuperComputing Applications and Innovation Department



# Basic Linear Algebra Algorithms



Linear algebra constitutes the core of most technical-scientific applications

Scalar products

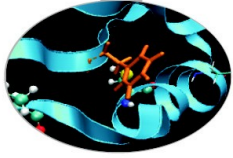
$$s = \sum_i a_i \cdot b_i$$

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:

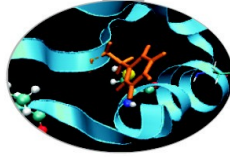
<http://www.nr.com>

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

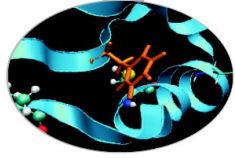


BLACS  
ACML  
PETSc  
LAPACK  
PLASMA  
MKL  
PSBLAS  
ATLAS  
MAGMA  
**BLAS**  
... but not only  
TRILINOS  
SLEPc  
ARPACK  
ESSL  
SCALAPACK  
CUBLAS  
SCOTCH

**which library should I use?**

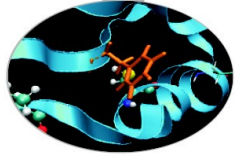
<http://www.netlib.org/utk/people/JackDongarra/la-sw.html>

# Automatically Tuned Linear Algebra Software



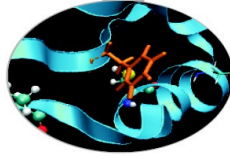
- ATLAS is both a research project and a software package.
- ATLAS's purpose is to provide portably optimal linear algebra software.
- The current version provides a complete BLAS API (for both C and Fortran77), and a very small subset of the LAPACK API.

# (Parallel) ARnoldi PACKage



- ARPACK is a collection of Fortran77 subroutines designed to solve large scale eigenvalue problems.
- It is most appropriate for large sparse or structured matrices. (Where structured means that a matrix-vector product requires order  $n$  rather than the usual order  $n^2$  floating point operations.)
- ARPACK is dependent upon a number of subroutines from LAPACK and the BLAS.
- Main feature: reverse communication interface.
- A parallel version of the ARPACK library is available. The message passing layers currently supported are BLACS and MPI .

# TRILINOS

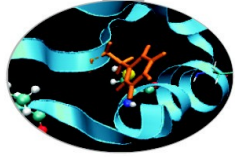


The Trilinos Project is an effort to develop algorithms and enabling technologies within an object-oriented software framework for the solution of large-scale, complex multi-physics engineering and scientific problems. A unique design feature of Trilinos is its focus on packages.

It is organized in Capability Areas:

- User Experience
- Parallel Programming Environments
- Framework & Tools
- Software Engineering Technologies and Integration
- I/O Support
- Meshes, Geometry, & Load Balancing
- Discretizations
- Scalable Linear Algebra
- Linear & Eigen Solvers
- Embedded Nonlinear Analysis Tools

# Linear Algebra is Hierarchical



Linear systems, Eigenvalue equations

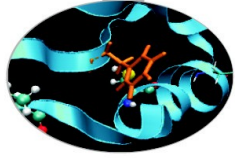
3  $M \times M$  products

2  $M \times V$  products

1  $V \times V$  products

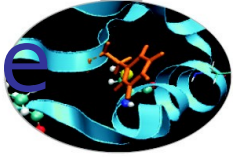


# (Parallel) Basic Linear Algebra Subprograms (BLAS and PBLAS)

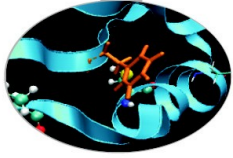


- **Level 1 : Vector - Vector operations**
- **Level 2 : Vector - Matrix operations**
- **Level 3 : Matrix - Matrix operations**

# **(Scalable) Linear Algebra PACKage** (LAPACK and ScaLAPACK)



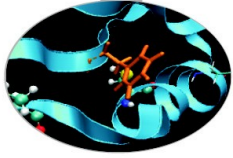
- **Matrix Decomposition**
- **Linear Equation Systems**
- **Eigenvalue Equations**
- **Linear Least Square Equations**
- **for dense, banded, triangular matrices**
- **for real and complex matrices**



# 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*

# BLAS/LAPACK subroutines



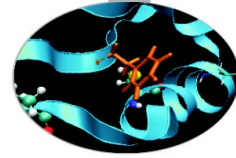
Routines name scheme: **XYZZZ**

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!



# BLAS subroutines

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

**DGEMM( TRANSA, TRANSB, M, N, L, ALPHA, A, LDA, B, LDB, BETA, C, LDC )**  
 'N' or 'T'  $\rightarrow$   $\max(1, M)$

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

**DGEMV( TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY )**  
 1.0d0  $\rightarrow$  0.0d0

**vector swap:  $X \Leftrightarrow Y$  (level 1)**

**DSWAP( N, X, INCX, Y, INCY )**

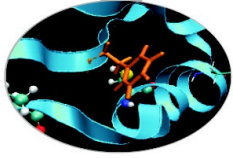
**scalar product:  $p = X' \cdot Y$  (level 1)**

**$p = \text{DDOT}( N, X, \text{INCX}, Y, \text{INCY} )$**   
 Function  $\leftarrow$  Increment for elements

Quick Reference Guide to the BLAS

<http://www.netlib.org/lapack/lug/node145.html>

# CBLAS subroutines



Instead of calling BLAS routines from a C-language program, you can use the CBLAS interface.

CBLAS is a C-style interface to the BLAS routines. You can call CBLAS routines using regular C-style calls. Use the ***cblas.h*** header file with the CBLAS interface. The header file specifies enumerated values and prototypes of all the functions.

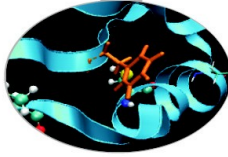
**matrix multiplication:  $C = A * B$  (level 3)**

`cblas_dgemm(ORDER, TRANSA, TRANSB, M, N, L, ALPHA, A, LDA, B, LDB, BETA, C, LDC)`

`CblasRowMajor`      `CblasNoTrans`

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

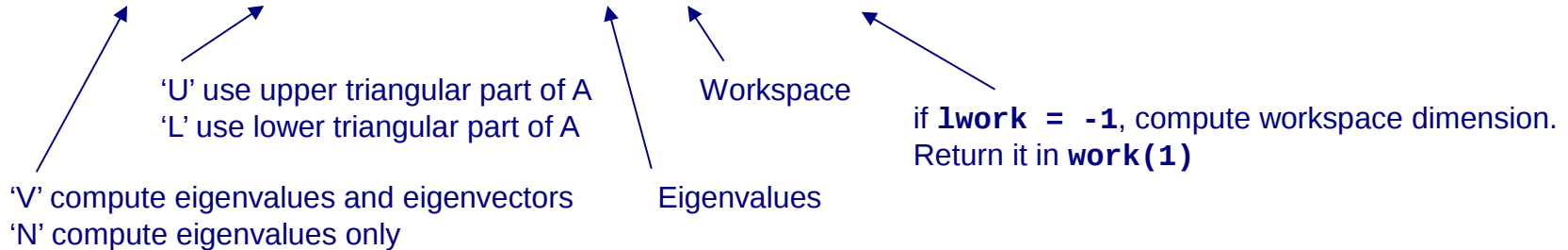
`cblas_dgemv(ORDER, TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCX)`



# LAPACK subroutines

## Eigenvalues and, optionally, eigenvectors of a real symmetric matrix:

**DSYEV( JOBZ, UPLO, N, A, LDA, W, WORK, LWORK, INFO )**



```
void dsyev_( const char* jobz, const char* uplo, const MKL_INT* n,  
            double* a, const MKL_INT* lda, double* w, double* work,  
            const MKL_INT* lwork, MKL_INT* info );
```

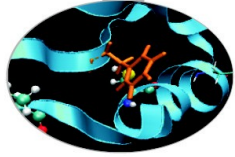
## Index of Driver and Computational Routines:

<http://www.netlib.org/lapack/lug/node142.html>

## Index of Auxiliary Routines:

<http://www.netlib.org/lapack/lug/node144.html>

# LAPACKE subroutines



Two-level C interface to LAPACK, consisting of a high-level interface and a middle-level interface.

Use the **lapacke.h** header file with the LAPACKE interface. The header file specifies enumerated values and prototypes of all the functions.

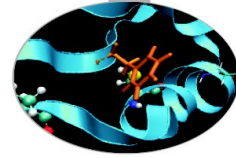
system of linear equations:

```
lapack_int LAPACKE_dgesv (int matrix_layout, lapack_int n, lapack_int nrhs,  
                          double *a, lapack_int lda, lapack_int *ipiv,  
                          double *b, lapack_int ldb);
```

LAPACK\_ROW\_MAJOR  
LAPACK\_COL\_MAJOR

```
lapack_int LAPACKE_dgesv_work (int matrix_layout, lapack_int n, lapack_int nrhs,  
                               double *a, lapack_int lda, lapack_int *ipiv,  
                               double *b, lapack_int ldb);
```





# How To Compile

*# load these modules on Galileo*

```
module load gnu/4.9.2
```

```
module load blas/3.5.0--gnu--4.9.2
```

```
module load lapack/3.5.0--gnu--4.9.2
```

*#FORTRAN:*

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

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

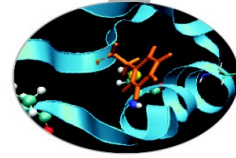
*#C: remember to include the header files "cblas.h" and/or "lapacke.h"*

```
LALIB="-I${BLAS_INC} -I${LAPACK_INC} \
```

```
    -L${LAPACK_LIB} -llapacke -llapack \
```

```
    -L${BLAS_LIB} -lcblas -lblas -lgfortran -lm"
```

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



# How To Compile

```
module load intel/pe-xe-2016--binary  
module load mkl/11.3.0--binary
```

```
LALIB="-I${MKL_INC} -L${MKL_LIB} -lmkl_intel_lp64 \  
-lmkl_sequential -lmkl_core -lpthread -lm -ldl"
```

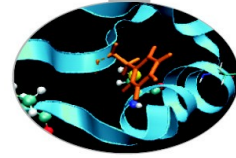
*#C: remember to include the header file "mkl.h"*

```
icc -o program.x program.c ${LALIB}  
icc -mkl -o program.x program.c -lpthread -lm -ldl
```

*#FORTRAN:*

```
ifort -o program.x program.f90 ${LALIB}  
ifort -mkl -o program.x program.f90 -lpthread -lm -ldl
```

<http://software.intel.com/en-us/articles/intel-mkl-link-line-advisor>



# Exercises:

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

Level 1: DCOPY, DSCAL, DNRM2, DDOT

Level 2: DGEMV, DGER

Level 3: DGEMM

Print all matrices and vectors generated.

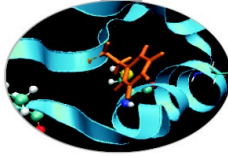
2) Write a program that uses the LAPACK routine DGESV. Print all matrices and vectors generated.

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

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

$A(i,j) = i+j/2$  if  $i \neq j$

# MAGMA



**Matrix Algebra for GPU and Multicore Architecture**

<http://icl.cs.utk.edu/magma/>

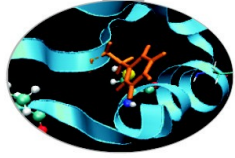
The MAGMA project aims to develop a dense linear algebra library similar to LAPACK but for heterogeneous/hybrid architectures, starting with current "Multicore+GPU" systems.

## **Methodology: CPU and GPU overlapping**

MAGMA uses HYBRIDIZATION methodology based on

- Representing linear algebra algorithms as collections of TASKS and DATA DEPENDENCIES among them
- Properly SCHEDULING tasks' execution over multicore and GPU hardware components

# MAGMA



## CPU versus GPU interfaces

Why two different interfaces?

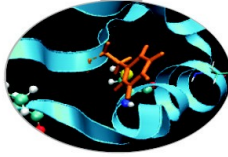
If data is already on the GPU

- pointer to GPU memory
- (some) additional memory allocation on CPU side

If data is already on the CPU

- no changes on the prototype
- internal overlap communication/computation (it uses pinned)
- (some) additional memory allocation on GPU side

# MAGMA



## How to compile/link

module load profile/advanced autoloader magma/2.0.1--intel--pe-xe-2016--binary

### C/C++:

```
#include "magma.h"           #include "cublas.h"
magma_init();               magma_finalize();
```

### FORTRAN:

```
USE magma
call magma_init()           call magma_finalize()
```

### COMPILE:

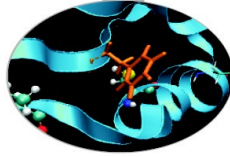
```
-I$MAGMA_INC -I$CUDA_INC -I$MKL_INC -fPIC -DHAVE_CUBLAS
```

### LINKING:

```
-L$MAGMA_LIB -lmagma -L$CUDA_LIB -lcublas -lcudart -mkl
```

***put MAGMA before CUDA and multi-threading library (like MKL)***

# MAGMA



## How to use in the code

DGETRF: Computes an LU factorization of a general matrix A, using partial pivoting with row interchanges.

PROTOTYPE: `DGETRF( M, N, A, LDA, IPIV, INFO )`

### *CPU interface:*

**FORTRAN:**     `call magma_dgetrf( M, N, A, lda, ipiv, info )`

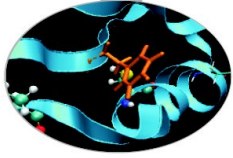
**C:**             `magma_dgetrf( &M, &N, A, &lda, ipiv, &info );`

### *GPU interface:*

`call cublas_set_matrix( M, N, size_of_elt, A, lda, d_A, ldda )`

`call magma_dgetrf_gpu( M, N, d_A, ldda, ipiv, info )`

# MKL on Intel Xeon Phi (MIC)



Intel released a version for Xeon Phi of the MKL mathematical libraries

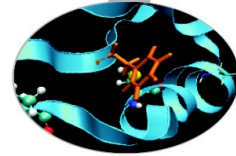
MKL have three different usage models:

- Automatic offload (AO)
- Compiler assisted offload (CAO)
- Native execution

<https://wiki.u-gov.it/confluence/display/SCAIUS/Guide+for+Intel+Xeon+Phi+%28MIC%29+Usage>



# MKL on Intel Xeon Phi (MIC)



Not all the MKL functions are enabled to AO. With this model offload is automatic and transparent. The library decides **when** to offload and **how much** to offload (workdivision).

Users can control parameters through environment variables or API. You can enable automatic offload with:

```
MKL_MIC_ENABLE=1
```

or

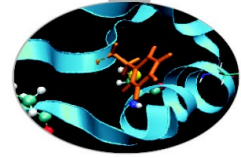
```
mkl_mic_enable()
```

MKL functions can be offloaded as other “ordinary” functions using the LEO (language extensions for offloading) pragmas.

All MKL functions can take advantage of the CAO model.

It's a more flexible option in terms of data management (you can use data persistence or mechanisms to hide the latency...)

# MKL on MIC: CAO



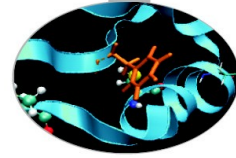
## C/C++

```
#pragma offload target (mic) \  
in (transa, transb, N, alpha, beta) \  
in (A:length(matrix_elements)) in (B:length(matrix_elements)) \  
inout (C:length(matrix_elements))  
{  
sgemm(&transa, &transb, &N, &N, &N, &alpha, A, &N, B, &N, &beta, C, &N);  
}
```

## Fortran

```
!dir$ attributes offload : mic : sgemm  
!dir$ offload target(mic) &  
!dir$ in (transa, transb, m, n, k, alpha, beta, lda, ldb, ldc), &  
!dir$ in (a:length(ncola*lda)), in (b:length(ncolb*ldb)) &  
!dir$ inout (c:length(n*ldc))  
CALL sgemm (transa, transb,m,n,k,alpha,a,lda,b,ldb,beta,c,ldc)
```

# MKL on MIC: Native mode



MKL libraries are also available when using the native mode. Tips:

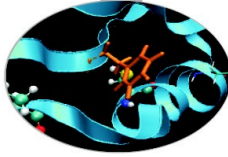
- Use all the 240 threads: `MIC_OMP_NUM_THREADS=240`
- Set the thread affinity: `MIC_KMP_AFFINITY = ...`

*Compilation:*

```
module load intel/pe-xe-2016--binary mkl/11.3.0--binary
source $INTEL_HOME/bin/compilervars.sh intel64
source $MKL_HOME/bin/mklvars.sh mic
```

```
icc -o program.x program.c -openmp -mmic
-I${MKLRROOT}/include -L${MKLRROOT}/lib/mic
-lmkl_intel_lp64 -lmkl_core -lmkl_intel_thread
-lpthread -lm
```

# MKL on MIC: Native mode



MIC-native programs need to be executed inside the MIC card itself:

login to a MIC node with a PBS interactive session requesting at least 1 mic (nmics=1);

use the "get\_dev\_list" script (available by loading the "superc" module) in order to get the name of the specific MIC card assigned to you;

connect through ssh into the MIC card (in the example node137-mic0)

```
qsub -I -A <account_name> -l select=1:ncpus=1:nmics=1 -q <queue_name>
```

```
qsub: waiting for job 1208049.io01 to start      (wait for long time)
```

```
qsub: job 1208049.io01 ready
```

```
...
```

```
module load superc
```

```
get_dev_list
```

```
cat ${PBS_JOBID}_dev_hostfile
```

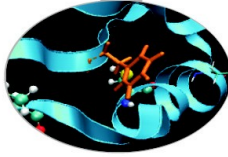
```
node137-mic0
```

```
...
```

```
ssh node137-mic0
```

```
$
```

# MKL on MIC: Native mode



For executing your native-MIC program, you need to set the `LD_LIBRARY_PATH` environment variable manually, by adding the path of the intel libraries specific for MIC execution:

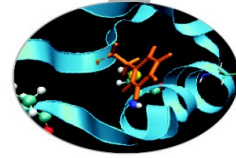
```
cd /path/to/the/working/directory/  
export MIC_OMP_NUM_THREADS=60
```

```
export LD_LIBRARY_PATH=/cineca/prod/compilers/intel/pe-xe-  
2016/binary/lib/mic:${LD_LIBRARY_PATH}
```

```
export LD_LIBRARY_PATH=/cineca/prod/compilers/intel/pe-xe-  
2016/binary/mkl/lib/mic:${LD_LIBRARY_PATH}
```

```
export LD_LIBRARY_PATH=/cineca/prod/compilers/intel/pe-xe-  
2016/binary/tbb/lib/mic:${LD_LIBRARY_PATH}
```

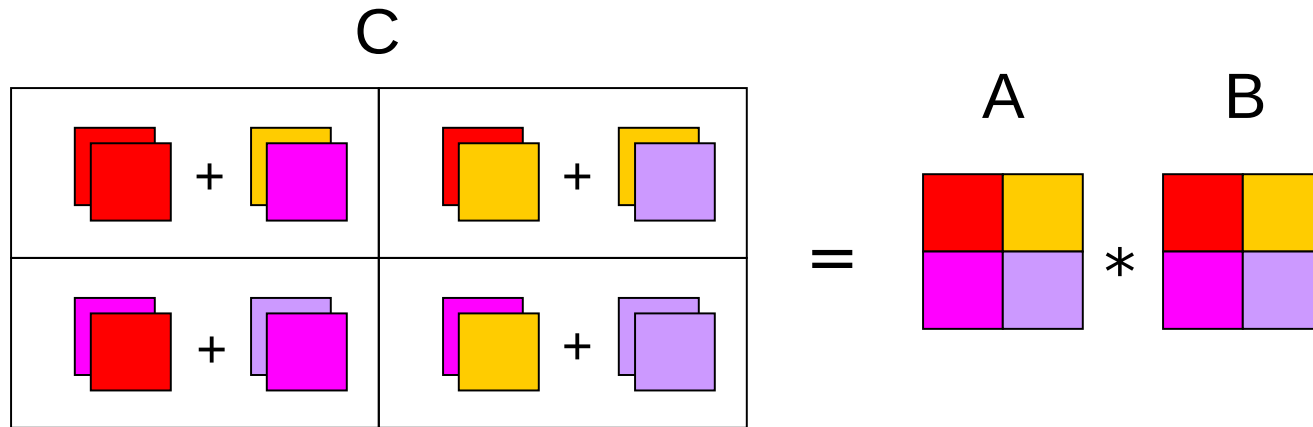
```
./program.x > program.out
```



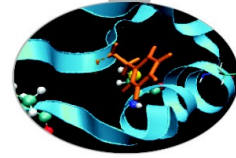
# Block Operations

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



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



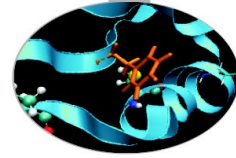
## Example: Partitioning into 2x2 Blocks

a11	a12	a13	a14	a15	a16	a17	a18	a19
a21	a22	a23	a24	a25	a26	a27	a28	a29
a31	a32	a33	a34	a35	a36	a37	a38	a39
a41	a42	a43	a44	a45	a46	a47	a48	a49
a51	a52	a53	a54	a55	a56	a57	a58	a59
a61	a62	a63	a64	a65	a66	a67	a68	a69
a71	a72	a73	a74	a75	a76	a77	a78	a79
a81	a82	a83	a84	a85	a86	a87	a88	a89
a91	a92	a93	a94	a95	a96	a97	a98	a99

B <sub>11</sub>	B <sub>12</sub>	B <sub>13</sub>	B <sub>14</sub>	B <sub>15</sub>
B <sub>21</sub>	B <sub>22</sub>	B <sub>23</sub>	B <sub>24</sub>	B <sub>25</sub>
B <sub>31</sub>	B <sub>32</sub>	B <sub>33</sub>	B <sub>34</sub>	B <sub>35</sub>
B <sub>41</sub>	B <sub>42</sub>	B <sub>43</sub>	B <sub>44</sub>	B <sub>45</sub>
B <sub>51</sub>	B <sub>52</sub>	B <sub>53</sub>	B <sub>54</sub>	B <sub>55</sub>

## Block Representation

Next Step: distribute blocks among processors



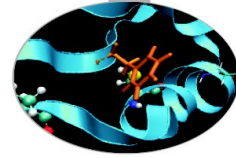
## Process Grid

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

		$p$		
		0	1	2
$q$	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

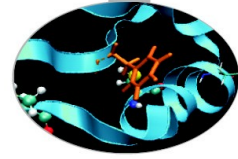
B <sub>11</sub>	B <sub>12</sub>	B <sub>13</sub>	B <sub>14</sub>	B <sub>15</sub>
B <sub>21</sub>	B <sub>22</sub>	B <sub>23</sub>	B <sub>24</sub>	B <sub>25</sub>
B <sub>31</sub>	B <sub>32</sub>	B <sub>33</sub>	B <sub>34</sub>	B <sub>35</sub>
B <sub>41</sub>	B <sub>42</sub>	B <sub>43</sub>	B <sub>44</sub>	B <sub>45</sub>
B <sub>51</sub>	B <sub>52</sub>	B <sub>53</sub>	B <sub>54</sub>	B <sub>55</sub>

p

		q				
		0		1		2
0		B <sub>11</sub>	B <sub>14</sub>	B <sub>12</sub>	B <sub>15</sub>	B <sub>13</sub>
		B <sub>31</sub>	B <sub>34</sub>	B <sub>32</sub>	B <sub>35</sub>	B <sub>33</sub>
		B <sub>51</sub>	B <sub>54</sub>	B <sub>52</sub>	B <sub>55</sub>	B <sub>53</sub>
1		B <sub>21</sub>	B <sub>24</sub>	B <sub>22</sub>	B <sub>25</sub>	B <sub>23</sub>
		B <sub>41</sub>	B <sub>44</sub>	B <sub>42</sub>	B <sub>45</sub>	B <sub>43</sub>

$$B_{h,k} \rightarrow (p, q) \quad \begin{aligned} p &= \text{MOD}(N_p + h - 1, N_p) \\ q &= \text{MOD}(N_q + k - 1, N_q) \end{aligned}$$

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



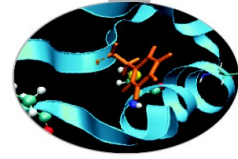
# Distribution of matrix elements

	0		1		2
0	B <sub>11</sub>	B <sub>14</sub>	B <sub>12</sub>	B <sub>15</sub>	B <sub>13</sub>
	B <sub>31</sub>	B <sub>34</sub>	B <sub>32</sub>	B <sub>35</sub>	B <sub>33</sub>
	B <sub>51</sub>	B <sub>54</sub>	B <sub>52</sub>	B <sub>55</sub>	B <sub>53</sub>
1	B <sub>21</sub>	B <sub>24</sub>	B <sub>22</sub>	B <sub>25</sub>	B <sub>23</sub>
	B <sub>41</sub>	B <sub>44</sub>	B <sub>42</sub>	B <sub>45</sub>	B <sub>43</sub>

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

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

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

Logical View (Matrix)

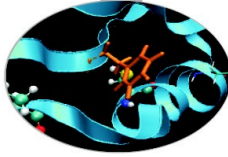
a <sub>11</sub>	a <sub>12</sub>	a <sub>17</sub>	a <sub>18</sub>	a <sub>13</sub>	a <sub>14</sub>	a <sub>19</sub>	a <sub>15</sub>	a <sub>16</sub>
a <sub>21</sub>	a <sub>22</sub>	a <sub>27</sub>	a <sub>28</sub>	a <sub>23</sub>	a <sub>24</sub>	a <sub>29</sub>	a <sub>25</sub>	a <sub>26</sub>
a <sub>51</sub>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

Local View (CPUs)

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

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

# BLACS



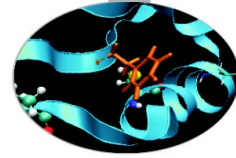
## (**B**asic **L**inear **A**lgebra **C**ommunication **S**ubprograms)

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

ScaLAPACK

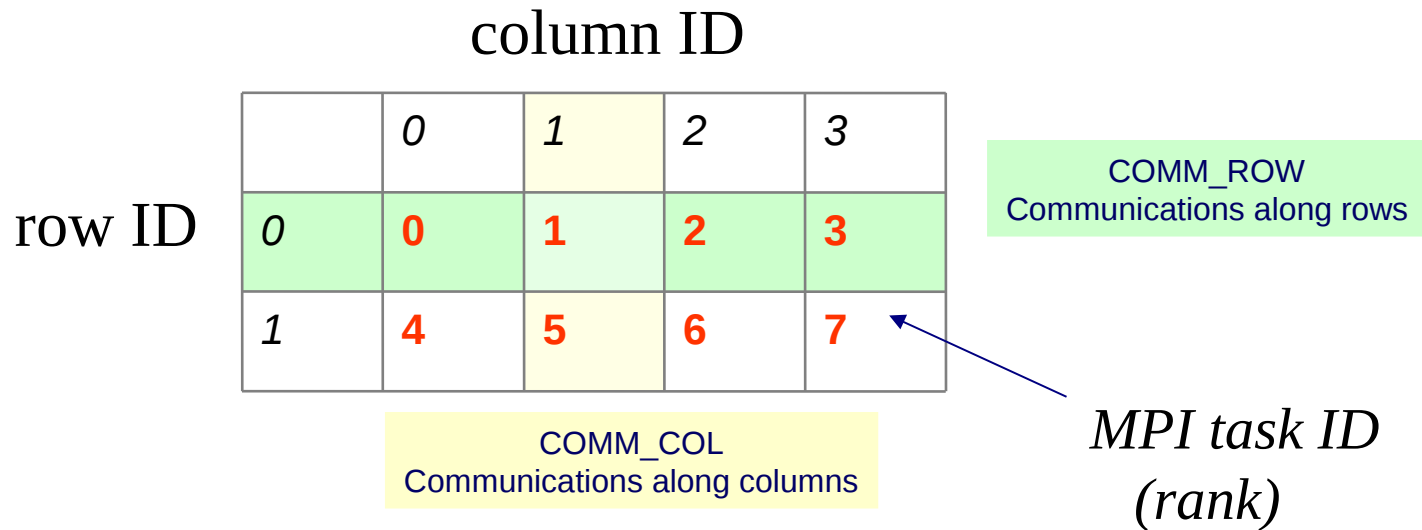
**BLACS**

Communication Library  
(MPI)



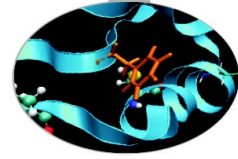
# BLACS Process Grid

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

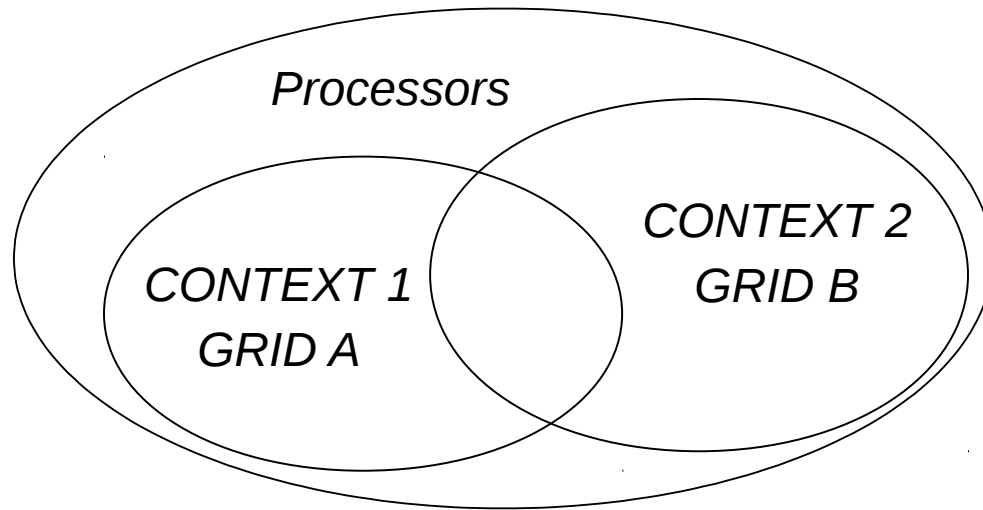


**BLACS\_GRIDINIT( CONTEXT, ORDER, NPROW, NPCOL )**

Initialize a 2D grid of **NPROW** x **NPCOL** processes with an order specified by **ORDER** in a given **CONTEXT**



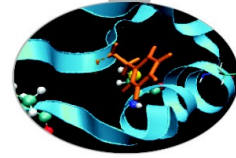
# CONTEXT



*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**, **ICONTEXT** 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**)

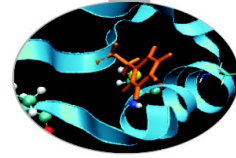
## **BLACS\_GRIDEXIT( CONTEXT )**

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

**DGEB2D ( 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

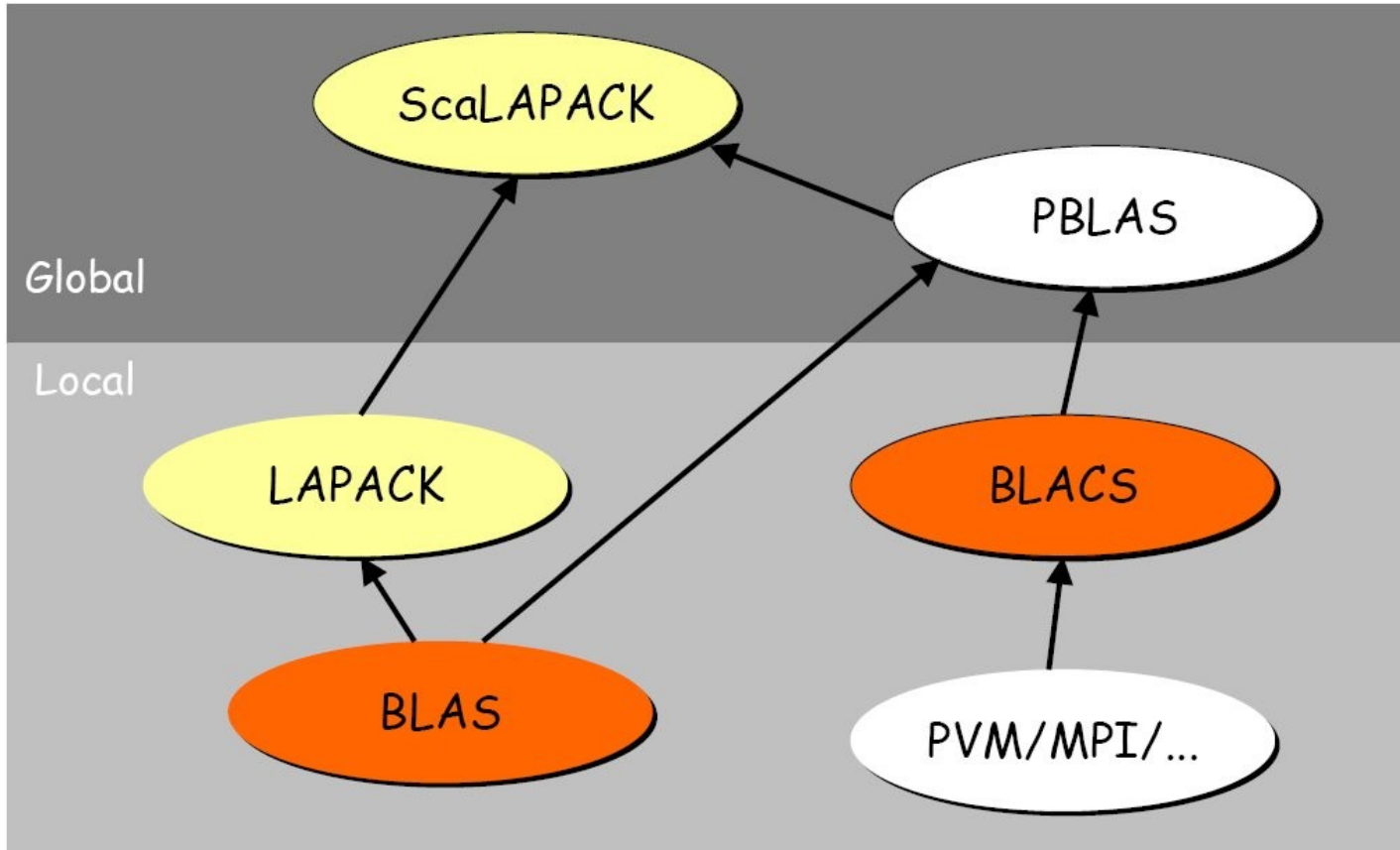
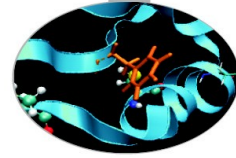
<http://www.netlib.org/blacs/BLACS/QRef.html>

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

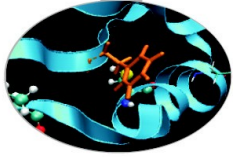
<http://www.netlib.org/blacs/cblacsqref.ps>



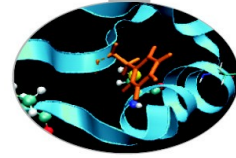
# Dependencies



# 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 )
```

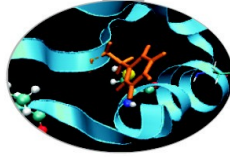
```
.....
```

```
.....
```

```
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( 3 ) = M**

**DESCA( 5 ) = MB**

**DESCA( 7 ) = RSRC**

**DESCA( 9 ) = LDA**

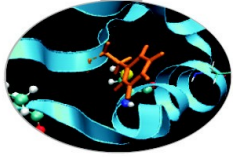
**DESCA( 2 ) = ICTXT**

**DESCA( 4 ) = N**

**DESCA( 6 ) = NB**

**DESCA( 8 ) = CSRC**

# 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

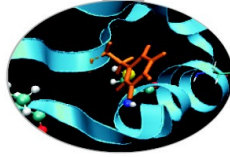
**MB, NB** (global input) blocking factors used to distribute matrix A

**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



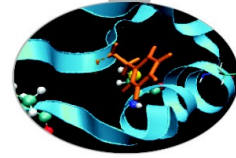
<http://www.netlib.org/scalapack/tools>

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

```
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

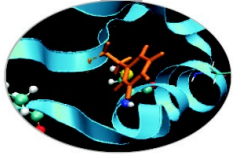
```

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:

PXYZZZ



Parallel

X data type

→

S = REAL

D = DOUBLE PRECISION

C = COMPLEX

Z = DOUBLE COMPLEX

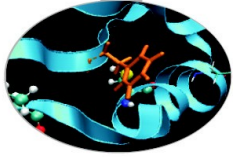
YY matrix type (GE = general, SY = symmetric, HE = hermitian)

ZZZ algorithm used to perform computation

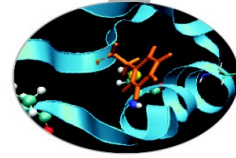
Some auxiliary functions don't make use of this naming scheme!



# 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)
```



**row / column swap:  $X \Leftrightarrow Y$  (level 1)**

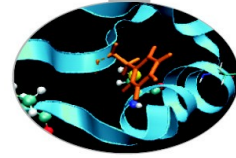
```
PDSWAP( N, X, IX, JX, DESCX, INCX, Y, IY, JY, DESCY, INCY )
```

$X(IX, JX:JX+N-1)$  if  $INCX = M_X$ ,     $X(IX:IX+N-1, JX)$  if  $INCX = 1$  and  $INCX \neq M_X$ ,  
 $Y(IY, JY:JY+N-1)$  if  $INCY = M_Y$ ,     $Y(IY:IY+N-1, JY)$  if  $INCY = 1$  and  $INCY \neq M_Y$ .

**scalar product:  $p = X' \cdot Y$  (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  $INCX \neq M_X$ ,  
 $Y(IY, JY:JY+N-1)$  if  $INCY = M_Y$ ,     $Y(IY:IY+N-1, JY)$  if  $INCY = 1$  and  $INCY \neq M_Y$ .



# 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

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

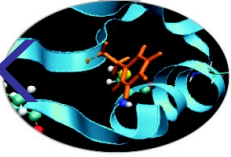
'V' compute eigenvalues and eigenvectors  
 'N' compute eigenvalues only

## Print matrix

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

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

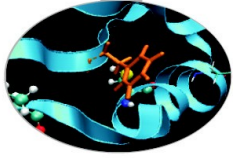
## LAPACK routine:

```
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)
```

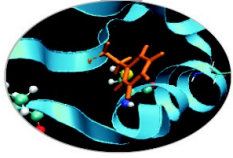
# ScaLAPACK Users' Guide



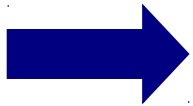
<http://www.netlib.org/scalapack/slug/>

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

# BLACS/ScaLAPACK + MPI

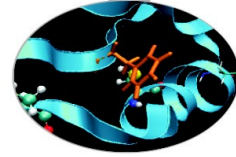


It is quite tricky to write a program using BLACS as a communication library, therefore:



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)
```

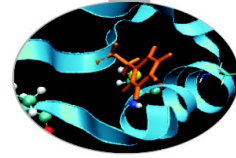
Initialize MPI environment

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

Initialize BLACS process grid  
of size nrow x ncol

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

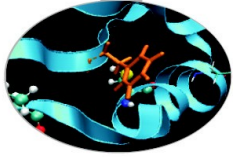
# Matrix redistribution



```
! 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
```

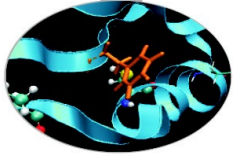
```
module load scalapack/2.0.2--openmpi--1.8.4--gnu--4.9.2
```

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

*FORTRAN:*

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

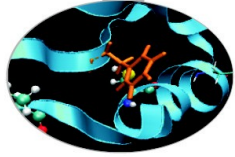
# How To Compile (GNU)



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 );
```

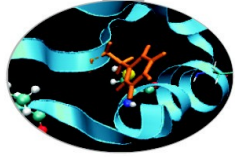
```
void pdgesv_( int * n, int * nrhs, double * A, int * ia, int * ja,  
             int * desca, int * ipiv, double * b, int * ib, int * jb, int * descb,  
             int * info );
```

```
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:

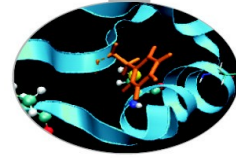
*(remember to include mkl.h, mkl\_scalapack.h, mkl\_blacs.h)*

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

*FORTTRAN:*

```
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  $i \neq j$

# Thanks for your attention!

