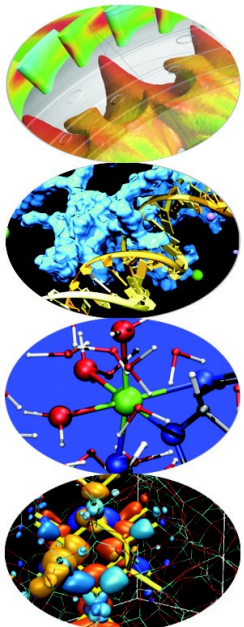
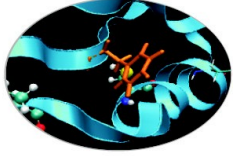


# Scalable Linear Algebra

*Nicola Spallanzani – [n.spallanzani@cineca.it](mailto: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

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

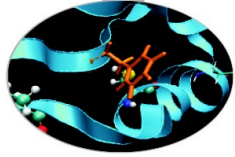
Linear Systems

$$A_{ij} x_j = b_i$$

Eigenvalue Equations

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

# Linear Algebra is Hierarchical



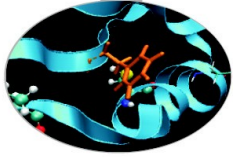
Linear systems, Eigenvalue equations

3  $M \times M$  products

2  $M \times V$  products

1  $V \times V$  products

# Algorithms and Libraries



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

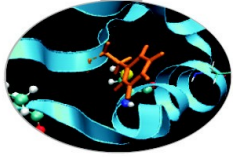
<http://www.nr.com>

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:

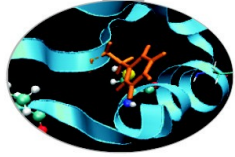
<http://www.netlib.org>

# Standard Linear Algebra Libraries

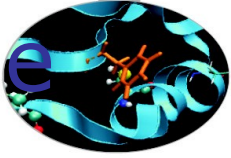


- blas
  - lapack
  - pblas
  - scalapak
  - arpack
  - parpack
  - PETSc
- Serial Linear Algebra Packages
- essl (IBM AIX)
  - mkl (Intel)
  - acml (AMD)
  - magma (ICL – Univ. Tennessee)
- Parallel Linear Algebra Packages (dense matrices)
- plasma (ICL – Univ. Tennessee)
- Eigenvalues Problems (sparse matrices)
- Sparse Linear Systems

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



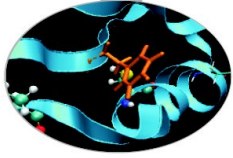
- Level 1 : Vector - Vector operations
- Level 2 : Vector - Matrix operations
- Level 3 : Matrix - Matrix operations



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

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

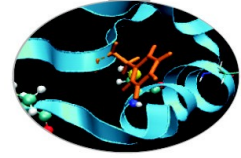
# Levels of Routines



- ***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

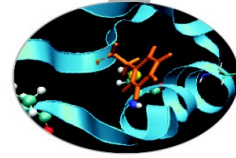
C = COMPLEX

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'                      1.0d0                      max(1,M)                      0.0d0

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

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

Increment for elements

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

```
DSWAP( N, X, INCX, Y, IY, JY, DESCY, INCY )
```

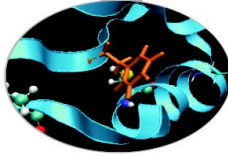
**scalar product:  $p = X' \cdot Y$  (level 1)**

```
p = DDOT( N, X, INCX, Y, INCY )
```

function

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 *mkl.h* header file with the CBLAS interface. The header file specifies enumerated values and prototypes of all the functions.

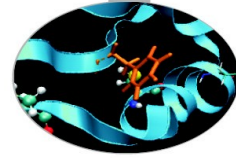
**matrix multiplication:  $C = A * B$  (level 3)**

```
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)**

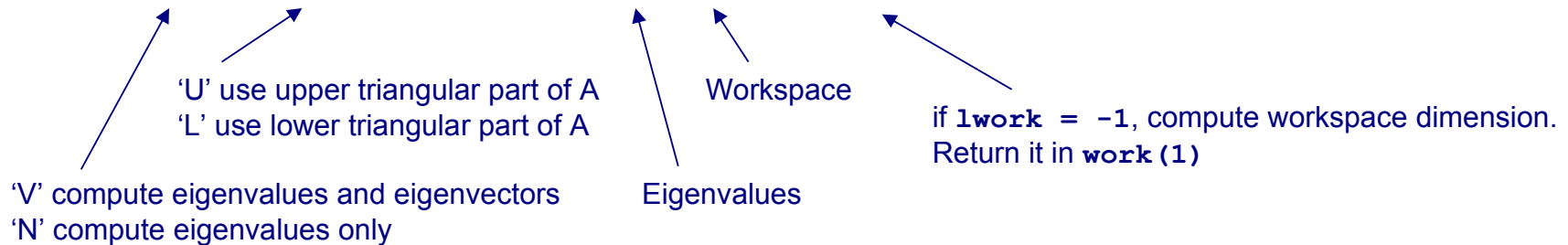
```
cblas_dgemv(ORDER, TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCX)
```



# LAPACK subroutines

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

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



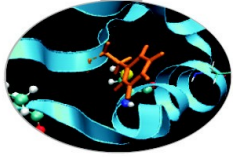
## Index of Driver and Computational Routines:

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

## Index of Auxiliary Routines:

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

# How To Compile



*# load these modules on PLX*

```
module load intel/co-2011.6.233--binary
```

```
module load lapack/3.3.1--intel--co-2011.6.233--binary
```

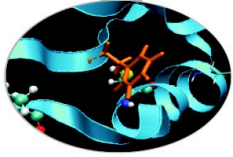
```
module load blas/2007--intel--co-2011.6.233--binary
```

*FORTRAN:*

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

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

# How To Compile



C:

*# remember to include this header file*

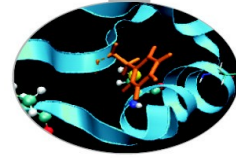
```
#include "mkl.h"
```

*# prefix for CBLAS*

```
cblas_
```

```
LALIB="-I${MKL_HOME}/include -L${MKL_HOME}/lib/intel64  
-lmkl_intel_lp64 -lmkl_sequential -lmkl_core -L${LAPACK_LIB}  
-llapack -L${BLAS_LIB} -lblas"
```

```
icc -o program.x program.c ${LALIB}
```



# 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

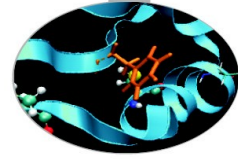
Print all matrices and vectors generated.

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  $i=j$

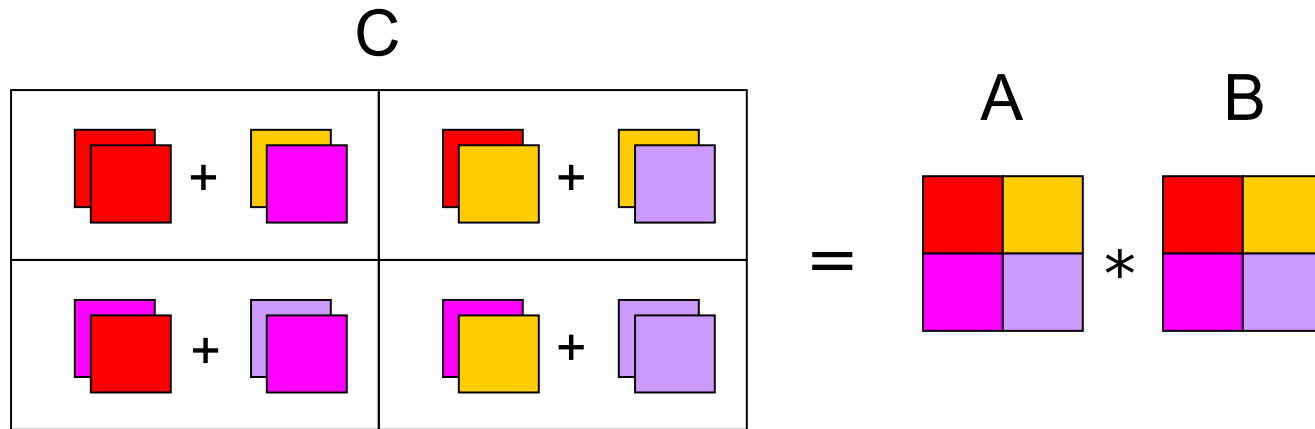
$A(i,j) = i+j/2$  if  $i \neq j$



# Block Operations

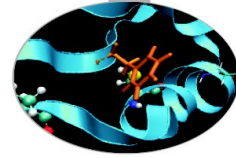
A block representation of a matrix operation constitutes the basic parallelization strategy for dense matrices.

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



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





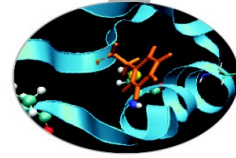
## Example: Partitioning into 2x2 Blocks

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

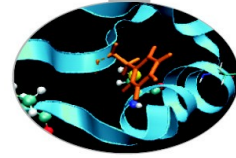


## Process Grid

$N$  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

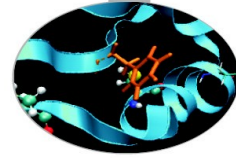
B <sub>11</sub>	B <sub>12</sub>	B <sub>13</sub>	B <sub>14</sub>	B <sub>15</sub>
B <sub>21</sub>	B <sub>22</sub>	B <sub>23</sub>	B <sub>24</sub>	B <sub>25</sub>
B <sub>31</sub>	B <sub>32</sub>	B <sub>33</sub>	B <sub>34</sub>	B <sub>35</sub>
B <sub>41</sub>	B <sub>42</sub>	B <sub>43</sub>	B <sub>44</sub>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



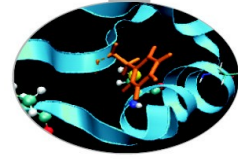
# Distribution of matrix elements

	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)

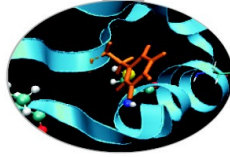
a <sub>11</sub>	a <sub>12</sub>	a <sub>17</sub>	a <sub>18</sub>	a <sub>13</sub>	a <sub>14</sub>	a <sub>19</sub>	a <sub>15</sub>	a <sub>16</sub>
a <sub>21</sub>	a <sub>22</sub>	a <sub>27</sub>	a <sub>28</sub>	a <sub>23</sub>	a <sub>24</sub>	a <sub>29</sub>	a <sub>25</sub>	a <sub>26</sub>
a <sub>51</sub>	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



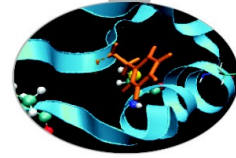
## (**B**asic **L**inear **A**lgebra **C**ommunication **S**ubprograms)

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

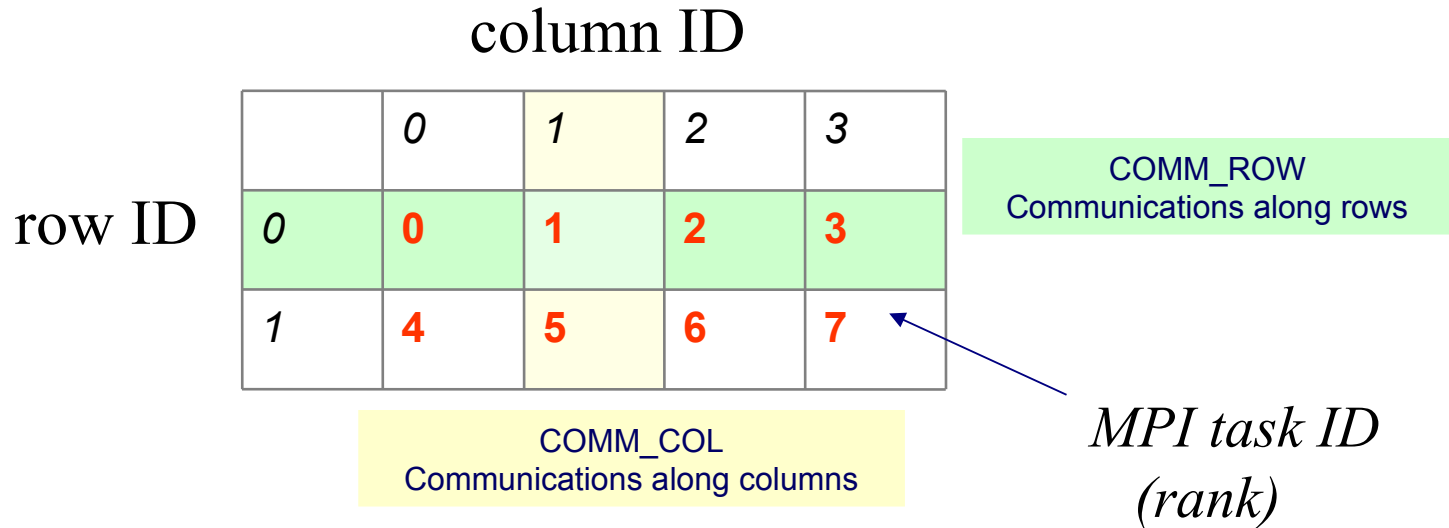
**BLACS**

Communication Library  
(MPI)



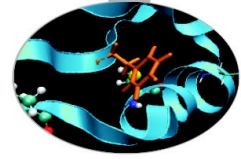
# BLACS Process Grid

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

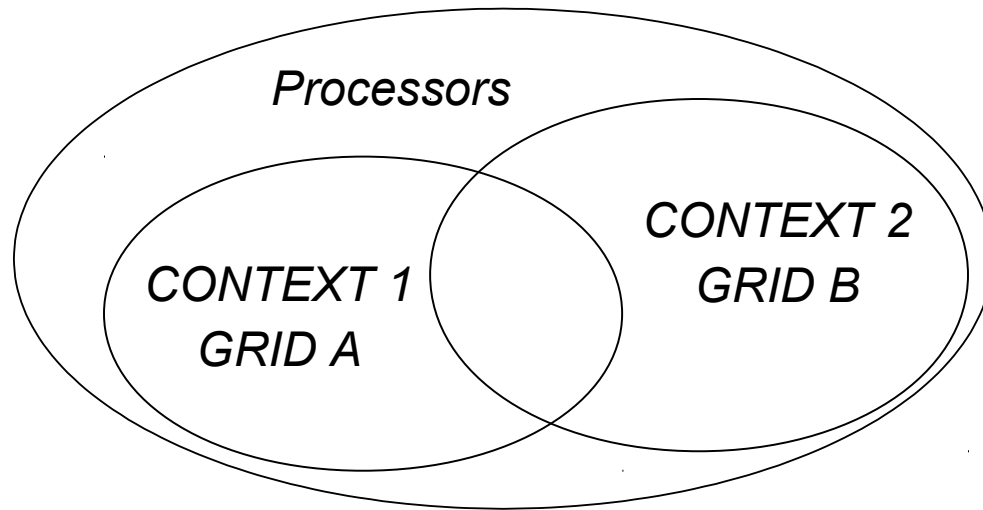


**BLACS\_GRIDINIT( CONTEXT, ORDER, NPROW, NPCOL )**

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



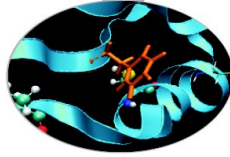
# CONTEXT



*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 )**

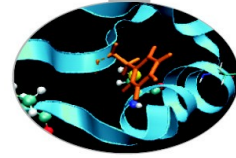
Query **CONTEXT** for the dimension of the grid of processes (**NPROW**, **NPCOL**) and for row-ID and col-ID (**MYROW**, **MYCOL**)

**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

**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 )**

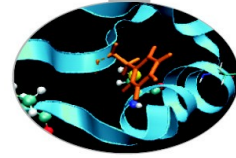
Receive matrix A(M,N) sent from process (RSRC,CSRC) with a broadcast operation

## Global reduction

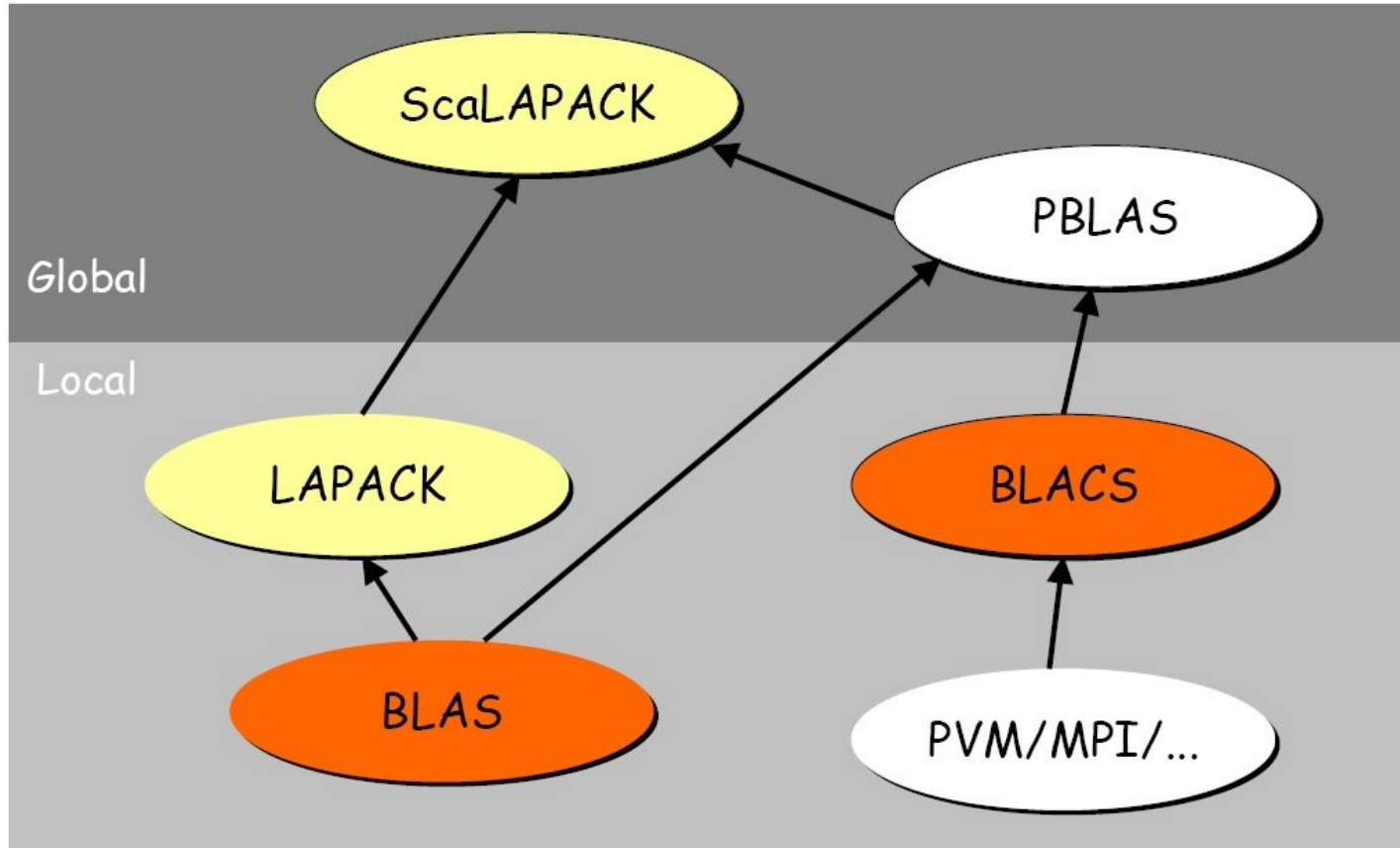
**DGSUM2D ( ICONTXT , SCOPE , TOP , M , N , A , LDA , RDST , CDST )**

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

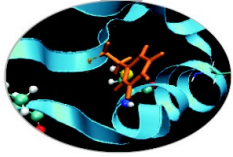
<http://www.netlib.org/blacs/BLACS/QRef.html>



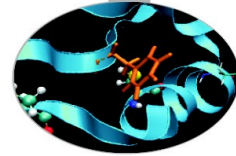
# Dependencies



# 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 )
```

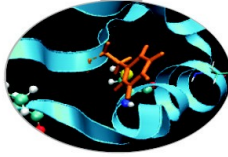
.....

.....

```
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`

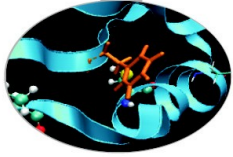
`DESCA ( 2 ) = ICTXT`

`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

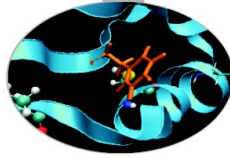
`MB, NB` (global input) blocking factors used to distribute matrix A

`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>

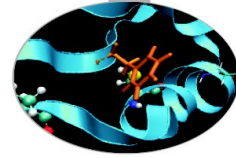
Computation of the local matrix size for a  $M \times N$  matrix distributed over processes in blocks of dimension  $MB \times NB$

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

Computation of global indexes

```
iloc = INDYG2L( i, MB, ROWID, 0, NPROW )  
jloc = INDYG2L( 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 = INDYG2P( i, MB, ROWID, 0, NPROW )  
jpcol = INDYG2P( j, NB, COLID, 0, NPCOL )
```

Define/read a local element, knowing global indexes

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

local array

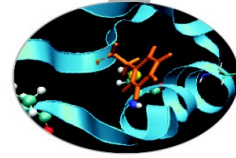
input value

```
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

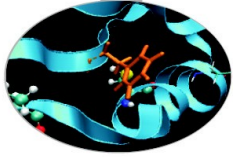
→ S = REAL  
D = DOUBLE PRECISION  
C = COMPLEX  
Z = DOUBLE COMPLEX

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

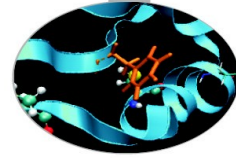
ZZZ algorithm used to perform computation

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)
```

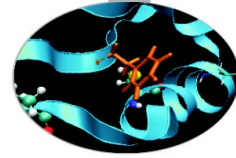


**row / column swap:  $X \leftrightarrow Y$  (level 1)**

```
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 <> 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:  $p = X' \cdot Y$  (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 <> 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

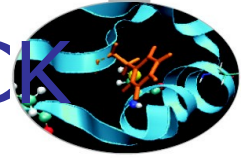
if `lwork = -1`, compute workspace dimension.  
 Return it in `work(1)`

'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)
```

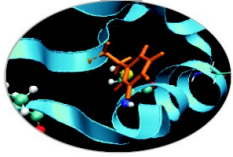
## LAPACK routine:

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

## SCALAPACK routine:

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

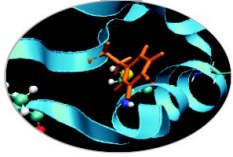
# ScaLAPACK Users' Guide



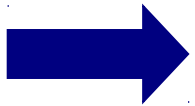
<http://www.netlib.org/scalapack/slug/>

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

# BLACS/ScaLAPACK + MPI



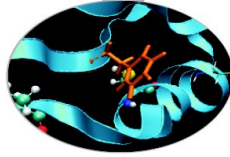
It is quite tricky to write a program using BLACS as a communication library, therefore:



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)
```

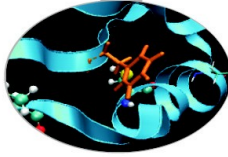
Initialize MPI environment

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

Initialize BLACS process grid  
of size nrow x ncol

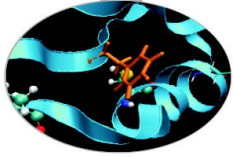
Create a row and a  
column communicator  
using BLACS indexes  
rowid and colid

# 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



*# load these modules on PLX*

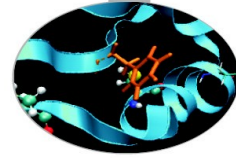
```
module load autoload scalapack/1.8.0--openmpi--1.3.3--  
intel--11.1--binary  
module load mkl/10.2.2--binary
```

*FORTRAN:*

```
LALIB="-L${SCALAPACK_LIB} -lscalapack -L${BLACS_LIB}  
-lblacs -lblacsF77init -lblacs"
```

```
MKL="-I{MKL_INC} -L${MKL_LIB} -lmkl_scalapack_lp64  
-lmkl_blacs_openmpi_lp64 -lmkl_intel_lp64  
-lmkl_sequential -lmkl_core"
```

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



# 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 another file. 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

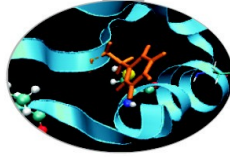
3) Write a program that uses the ScaLAPACK routine PDGESV. Print in files all matrices and vectors generated.

$\mathbf{Ax}=\mathbf{b}$  ;  $\mathbf{b}(i) = 207-i$  ;

$\mathbf{A}(i,j) = 10000$  if  $i=j$

$\mathbf{A}(i,j) = i+j/2$  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

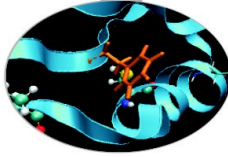
- Properly SCHEDULING tasks' execution over multicore and GPU hardware components

Hybridization means...

- Panels (Level 2 BLAS) are factored on CPU using LAPACK

- Trailing matrix updates (Level 3 BLAS) are done on the GPU using "look-ahead"

# 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

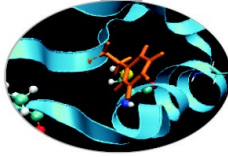
If data is already on the CPU

- no changes on the prototype

- internal overlap communication/computation (it uses pinned)

- (some) additional memory allocation on GPU side

# MAGMA



## How to compile/link

C/C++:

```
#include "magma.h"
```

FORTRAN:

```
USE magma
```

COMPILE:

```
-I$(MAGMADIR)/include -I$(CUDADIR)/include
```

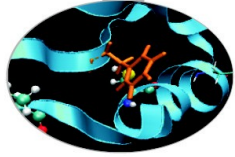
LINKING:

```
-L$(MAGMADIR)/lib -lmagma -lmagmablas
```

```
$(MAGMADIR)/lib/libmagma.a $(MAGMADIR)/lib/libmagma
```

*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:*

```
call 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 )
```



# Thanks for your attention!

