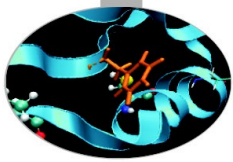
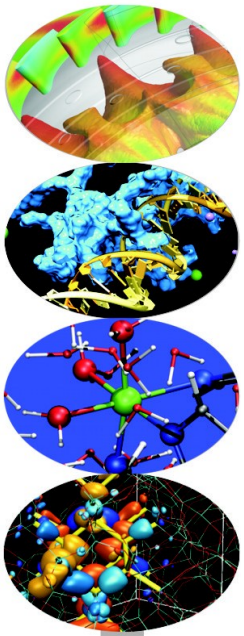
CINECA SCAI
SuperComputing Applications and Innovation



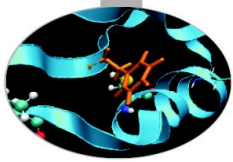


HPC Numerical Libraries

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

CINECA

Algorithms and Libraries



Many numerical algorithms are well known and largely available. See for instance:

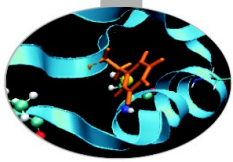
http://www.nr.com

So why should I use libraries?

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

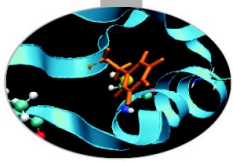
http://www.netlib.org

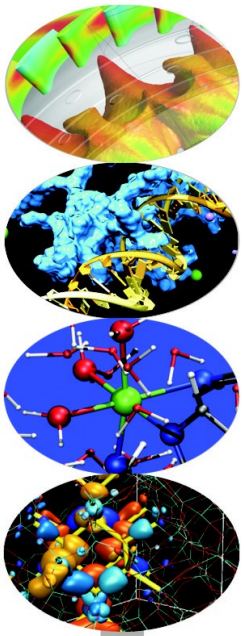
Numerical Libraries



- Scalable Linear Algebra
- Fast Fourier Transform
- Adaptive Mesh Refinement (AMR)
- Input/Output

CINECA SCAI
SuperComputing Applications and Innovation



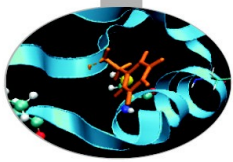


Scalable Linear Algebra

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

CINECA

Standard Linear Algebra Libraries



BLACS

ACML

PETSc

LAPACK

PLASMA

MKL

PSBLAS

ATLAS

MAGMA

BLAS

… but not only

SLEPc

TRILINOS

ESSL

ARPACK

SCALAPACK

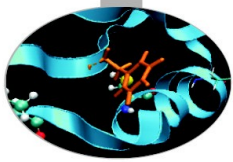
SCOTCH

CUBLAS

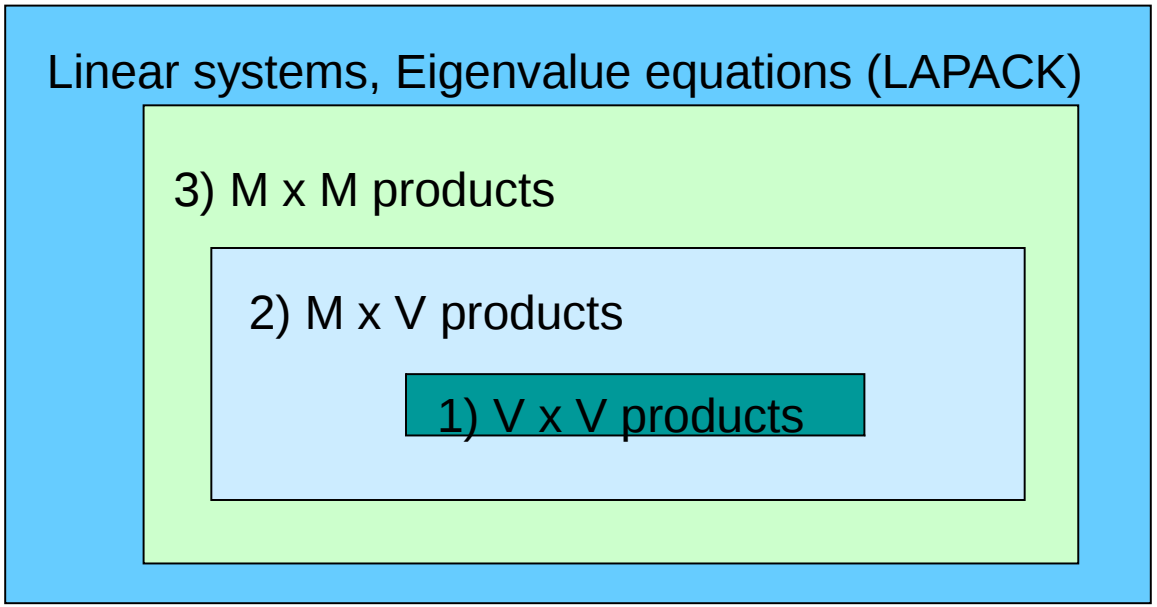
which library should I use?

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

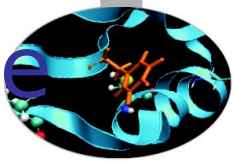
(Parallel) Basic Linear Algebra Subprograms (BLAS and PBLAS)



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



(Scalable) Linear Algebra PACKage (LAPACK and ScaLAPACK)

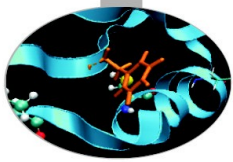


- **Matrix Decomposition**
- **Linear Equation Systems**
- **Eigenvalue Equations**
- **Linear Least Square Equations**

Designed for:

- **dense, banded, triangular matrices**
- **real and complex matrices**

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' — 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 — 0.0d0

vector swap: X ⇔ Y (level 1)

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

scalar product: p = X'·Y (level 1)

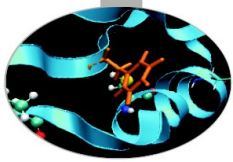
```
p = DDOT( N, X, INCX, Y, INCY )
```

Function — 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)

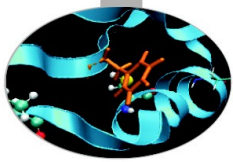
```
cblas_dgemm(ORDER, TRANSA, TRANSB, M, N, L, ALPHA, A, LDA, B, LDB, BETA, C, LDC)
```

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 )
```

- JOBZ: 'V' compute eigenvalues and eigenvectors; 'N' compute eigenvalues only
- UPLO: 'U' use upper triangular part of A; 'L' use lower triangular part of A
- W: Eigenvalues
- WORK: Workspace
- LWORK: if **lwork = -1**, compute workspace dimension. Return it in **work(1)**

```
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 );
```

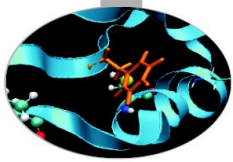
Index of Driver and Computational Routines:

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);
```

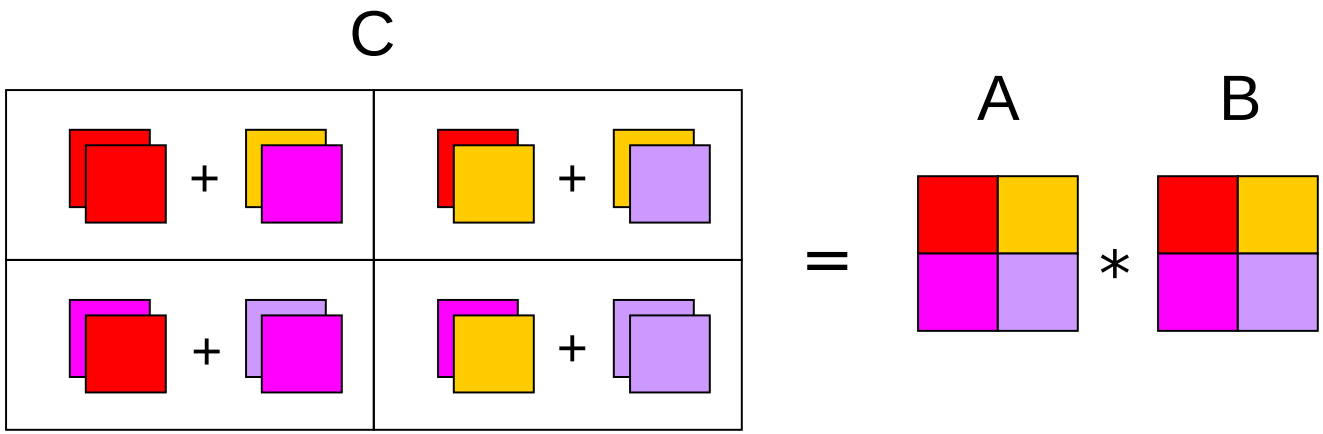
```
        LAPACK_ROW_MAJOR
        LAPACK_COL_MAJOR
```

```
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);
```

Block Operations

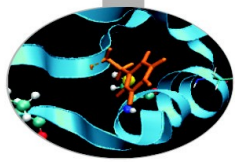
A block representation of a matrix operation constitutes the basic parallelization strategy for dense matrices.

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



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

Process Grid



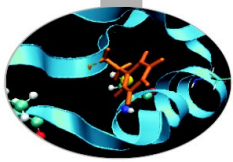
N processes are organized into a logical 2D mesh with p rows and q columns, such that p x q = N

p

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

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

Distribution of matrix elements



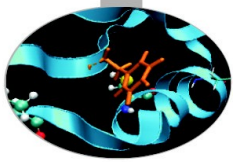
a_{11}	a_{12}	a_{13}	a_{14}	a_{15}	a_{16}	a_{17}	a_{18}	a_{19}
a_{21}	a_{22}	a_{23}	a_{24}	a_{25}	a_{26}	a_{27}	a_{28}	a_{29}
a_{31}	a_{32}	a_{33}	a_{34}	a_{35}	a_{36}	a_{37}	a_{38}	a_{39}
a_{41}	a_{42}	a_{43}	a_{44}	a_{45}	a_{46}	a_{47}	a_{48}	a_{49}
a_{51}	a_{52}	a_{53}	a_{54}	a_{55}	a_{56}	a_{57}	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}

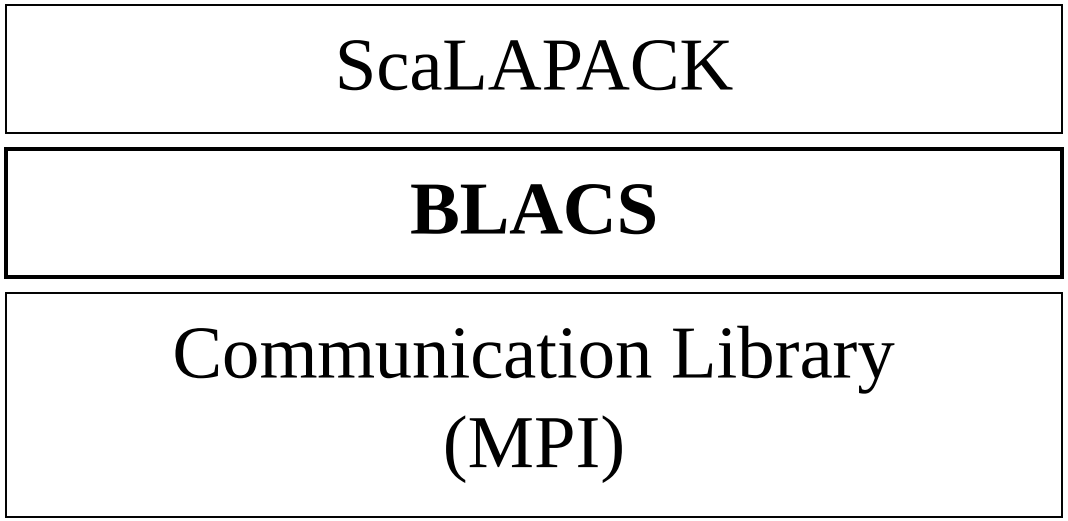
Local View (CPUs)

BLACS

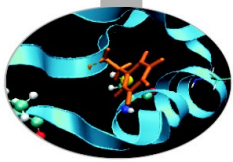


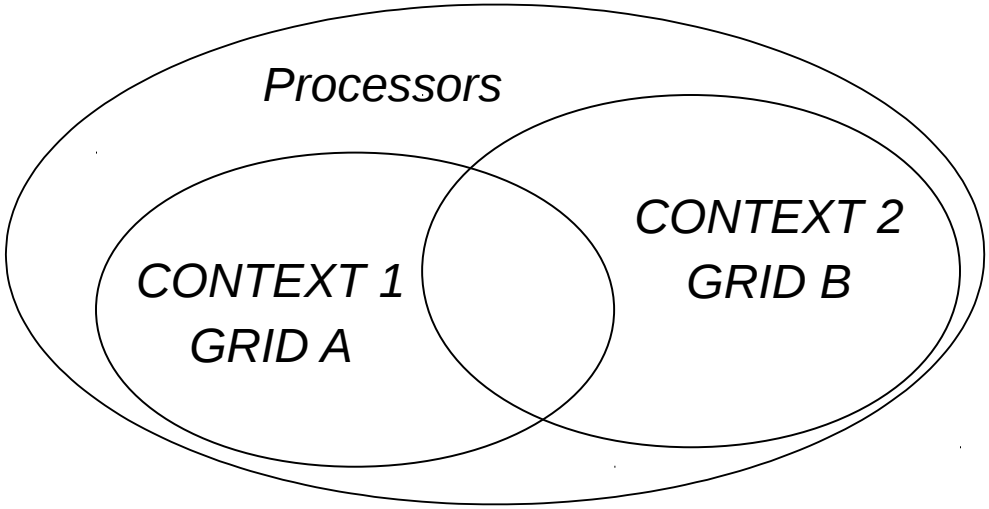
(**B**asic **L**inear **A**lgebra **C**ommunication **S**ubprograms)

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



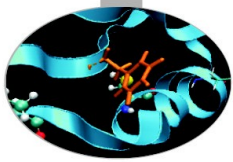
CONTEXT

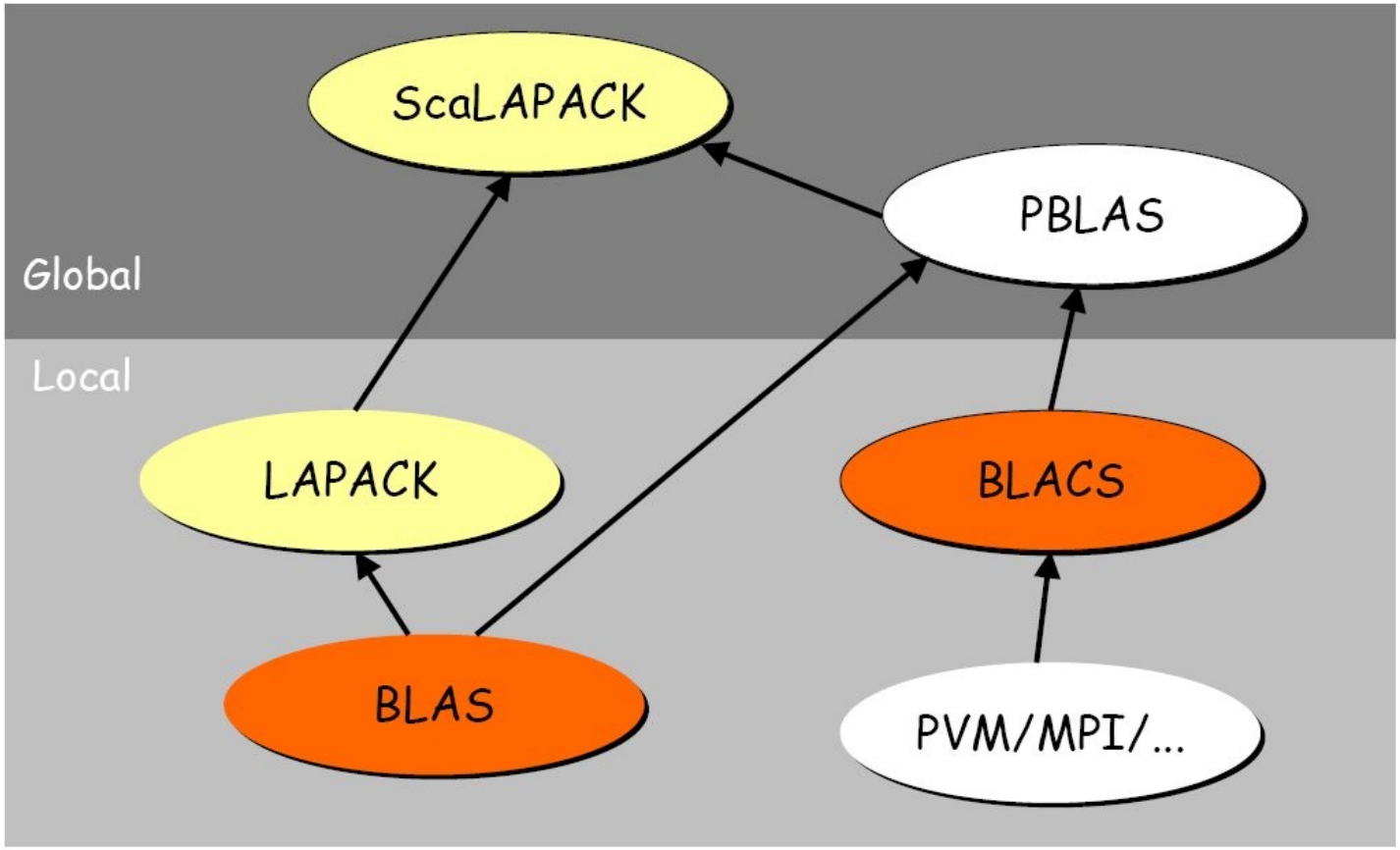




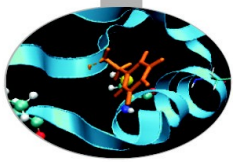
Context ⟷ *MPI Communicators*

Dependencies



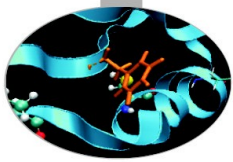


ScaLAPACK and PBLAS: template



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*

PBLAS/ScaLAPACK subroutines



Routines name scheme: PXYYZZZ

Parallel

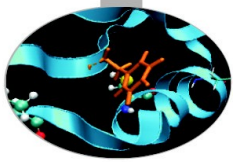
X data type → S = REAL
D = DOUBLE PRECISION
C = COMPLEX
Z = DOUBLE COMPLEX

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

ZZZ algorithm used to perform computation

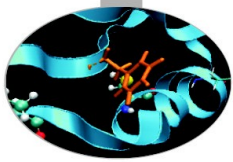
Some auxiliary functions don't make use of this naming scheme!

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.

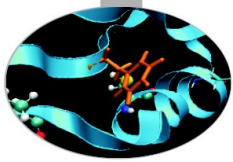
scalar product: p = X'·Y (level 1)

```
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

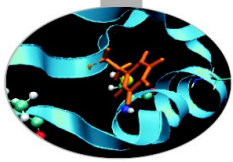
if **lwork = -1**, compute workspace dimension.
Return it in **work(1)**

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

How To Compile (GNU)



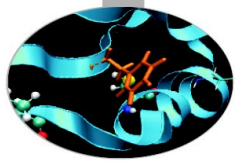
```
# load these modules on Galileo
module load autoload profile/advanced
module load scalapack/2.0.2--openmpi--1.8.4--gnu--4.9.2

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

FORTRAN:

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

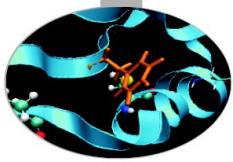
How To Compile (GNU)



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)

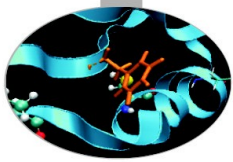


C:

```
// 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)

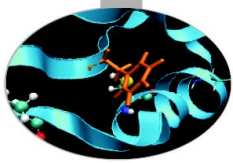
```
mpicc -o program.x program.c -mkl -lmkl_scalapack_lp64 \
      -lmkl_blacs_intelmpi_lp64 -lpthread -lm
```

FORTRAN:

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

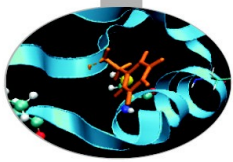
https://software.intel.com/en-us/articles/intel-mkl-link-line-advisor

Automatically **T**uned **L**inear **A**lgebra **S**oftware



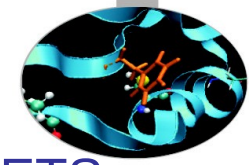
- ATLAS is both a research project and a software package.
- ATLAS's purpose is to provide portably optimal linear algebra software (recommended as a way to automatically generate an optimized BLAS library).
- The current version provides a complete BLAS API (for both C and Fortran77), and a very small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

(**P**arallel) **AR**noldi **PACK**age



- 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 availible. The message passing layers currently supported are BLACS and MPI .

PETSc



The Portable, Extensible Toolkit for Scientific Computation (PETSc, pronounced PET-see; the S is silent), is a suite of data structures and routines developed by Argonne National Laboratory for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It employs the Message Passing Interface (MPI) standard for all message-passing communication. PETSc includes a large suite of parallel linear and nonlinear equation solvers that are easily used in application codes written in C, C++, Fortran and now Python.

Particularly well suited for Sparse Linear Algebra.

Language: C, C++, Fortran, Python

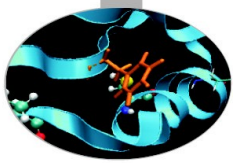
Availability: open source

Developers: various

Distributors: Mathematics and Computer Science Division, Argonne National Laboratory

Ref.: Argonne National Laboratory

TRILINOS

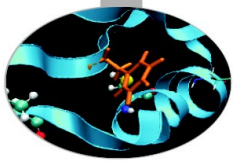


The Trilinos Project is an effort to develop algorithms and enabling technologies within an object-oriented software framework (written in C++) for the solution of large-scale, complex multi-physics engineering and scientific problems. A unique design feature of Trilinos is its focus on packages.

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

MAGMA



Matrix **A**lgebra for **G**PU and **M**ulticore **A**rchitecture

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

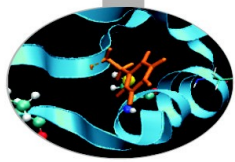
The MAGMA project aims to develop a dense linear algebra library similar to LAPACK but for heterogeneous/hybrid architectures, starting with current "Multicore+GPU" systems.

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?

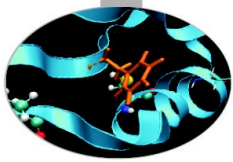
If data is already on the GPU

- 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 autoload 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:

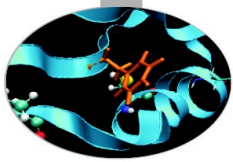
-I$MAGMA_INC -I$CUDA_INC -I$MKL_INC -fPIC -DHAVE_CUBLAS

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)**

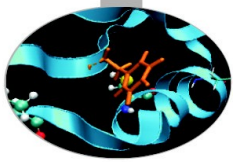
CPU interface:

```
FORTRAN:    call magmaf_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)



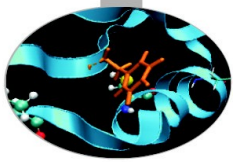
Intel released a version for Xeon Phi of the MKL mathematical libraries

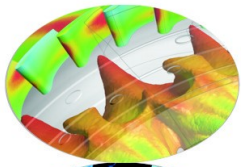
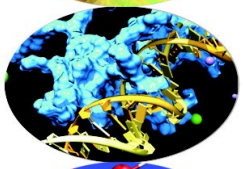
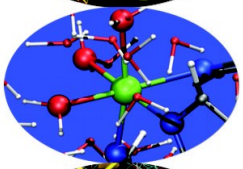
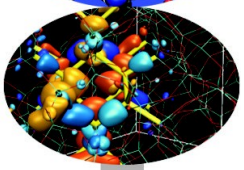
MKL have three different usage models:

- 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

CINECA SCAI
SuperComputing Applications and Innovation



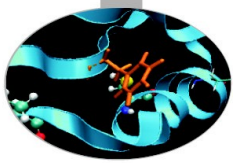





Fast Fourier Transform

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

CINECA

Fast Fourier Transform

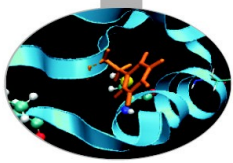


A fast Fourier transform (FFT) algorithm computes the discrete Fourier transform (DFT) of a sequence, or its inverse. Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain and vice versa.

An FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors.

As a result, it manages to reduce the complexity of computing the DFT from O(n^2), which arises if one simply applies the definition of DFT, to O(n log n), where n is the data size.

Fast Fourier Transform



FFTPACK

FFTPACK is a package of Fortran subroutines for the fast Fourier transform. It includes complex, real, sine, cosine, and quarter-wave transforms. It is included in the general-purpose mathematical library SLATEC.

Language: FORTRAN

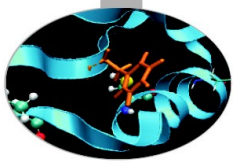
Availability: public domain

Developers: Paul N. Swarztrauber, National Center for Atmospheric Research, Boulder, CO

Distributors: NETLIB

Ref.: The University of Tennessee at Knoxville and Bell Laboratories

Fast Fourier Transform



FFTW

(Fastest Fourier Transform in the West)

FFTW is a C subroutine library for computing the discrete Fourier transform (DFT) in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as of even/odd data, i.e. the discrete cosine/sine transforms or DCT/DST).

The currently most used free library to compute DFTs.

It uses both MPI and OpenMP.

Language: C

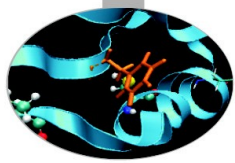
Availability: public domain

Developers: Matteo Frigo and Steven G. Johnson.

Distributors: FFTW

Ref.: MIT

Fast Fourier Transform

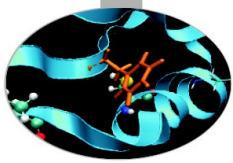


FFTW

(Fastest Fourier Transform in the West)

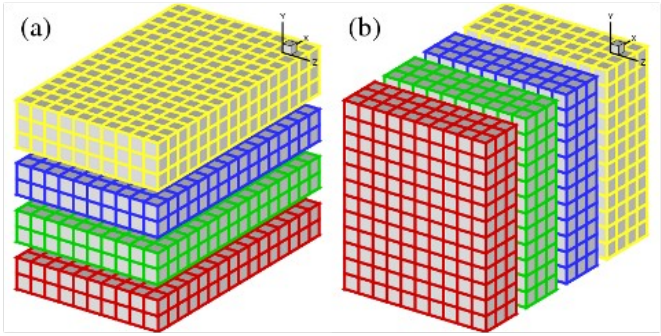
- Written in C (FORTRAN 77/90/2003 wrappers are included)
- FFTW adapt itself to your system and your problem: It check both total memory, size of your array, cache size register, CPU type, …
- … and it chose the best algorithm for you!
- 2 setps is required:
- Plan creation
- Execution
- It supports FFTs for any different array size (< total memory)

Fast Fourier Transform

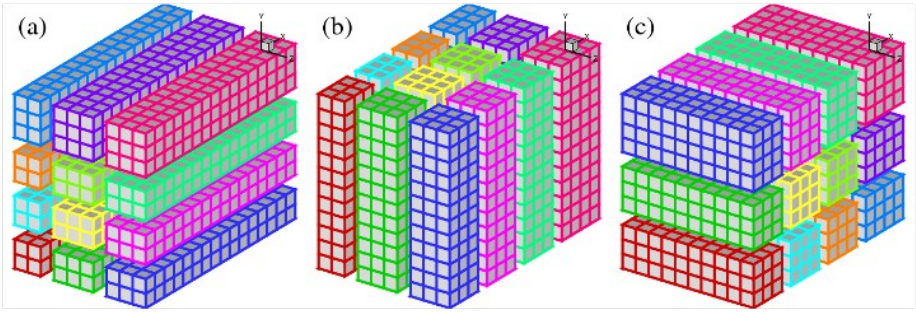


2Decomp&FFT

The 2DECOMP&FFT library is a software framework in Fortran to build large-scale parallel applications. It is designed for applications using three-dimensional structured mesh and spatially implicit numerical algorithms. At the foundation it implements a general-purpose 2D pencil decomposition for data distribution on distributed-memory platforms. On top it provides a highly scalable and efficient interface to perform three-dimensional distributed FFTs. The library is optimised for supercomputers and scales well to hundreds of thousands of cores. It relies on MPI but provides a user-friendly programming interface that hides communication details from application developers.



VS



Language: FORTRAN

Availability: public domain

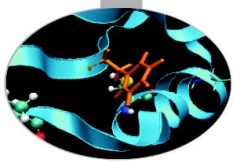
Developers: Ning LI.

Distributors: 2decomp.org

Ref.: HECToR Distributed Computational Science and Engineering (CSE) Service operated by NAG Ltd.

http://www.2decomp.org/

Fast Fourier Transform



P3DFFT

3DFFT is a scalable software library implementing three-dimensional spectral transforms. It has been used in a variety of codes from many areas of computational science. It has been tested and used on many high-end computational system. It uses two-dimensional domain decomposition in order to overcome a scaling bottleneck of one-dimensional decomposition. This allows the programs with this library to scale well on a large number of cores, consistent with bisection bandwidth scaling of interconnect of the underlying hardware system.

Language: C

Availability: public domain

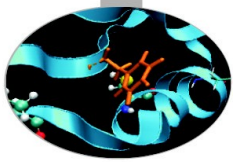
Developers: Dmitry Pekurovsky.

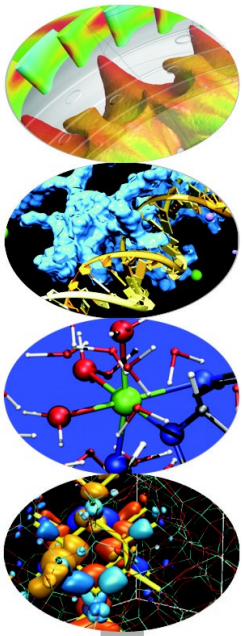
Distributors: 2decomp.org

Ref.: San Diego Supercomputer Center (SDSC) at UC San Diego.

https://code.google.com/p/p3dfft/

CINECA SCAI
SuperComputing Applications and Innovation



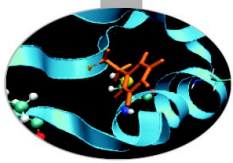


Adaptive Mesh Refinement (AMR)

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

CINECA

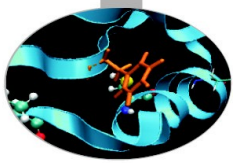
AMR libraries



Many problems in numerical analysis, do not require a uniform precision in the numerical grids used for computational simulation.

Adaptive mesh refinement provides a dynamic programming environment for adapting the precision of the numerical computation based on the requirements of a computation problem in specific areas.

AMR libraries



METIS

METIS is a software package for graph partitioning that implements various multilevel algorithms. It was written in C, and it can generate, manipulate and partition, graphs grids and matrices. Particularly well suited for partitioning regular grids on a parallel distributed memory system.

Language: C

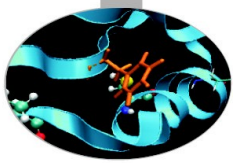
Availability: free for academy and research

Developers: George Karypis Lab

Distributors: George Karypis Lab

Ref.: Department of Computer Science & Engineering, University of Minnesota

AMR libraries



ParMETIS

ParMETIS is an MPI-based parallel library that implements a variety of algorithms for partitioning unstructured graphs, meshes, and for computing fill-reducing orderings of sparse matrices. ParMETIS extends the functionality provided by METIS and includes routines that are especially suited for parallel AMR computations and large scale numerical simulations

Language: C

Availability: free for academy and research

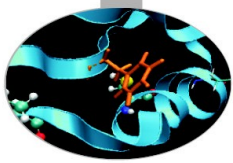
Developers: George Karypis Lab

Distributors: George Karypis Lab

Ref.: Department of Computer Science & Engineering, University of Minnesota

http://glaros.dtc.umn.edu/gkhome/views/metis

AMR libraries



PARAMESH

PARAMESH is a package of Fortran 90 subroutines designed to provide an application developer with an easy route to extend an existing serial code which uses a logically Cartesian structured mesh into a parallel code with adaptive mesh refinement(AMR).

Language: FORTRAN

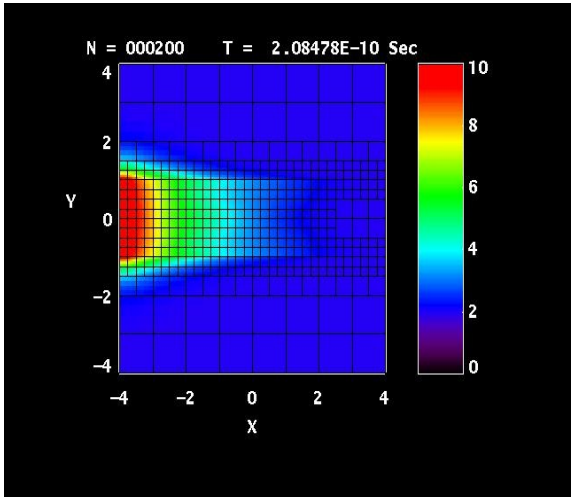
Availability: free for academy and research

Developers: Peter MacNeice, Kevin M. Olson et al.

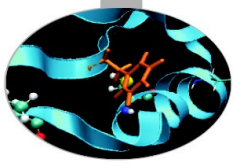
Distributors: Kevin M. Olson

Ref.: NASA e Drexel University

http://sourceforge.net/projects/paramesh/



AMR libraries



CHOMBO

Chombo provides a set of tools for implementing finite difference and finite volume methods for the solution of partial differential equations on block-structured adaptively refined rectangular grids.

Language: C

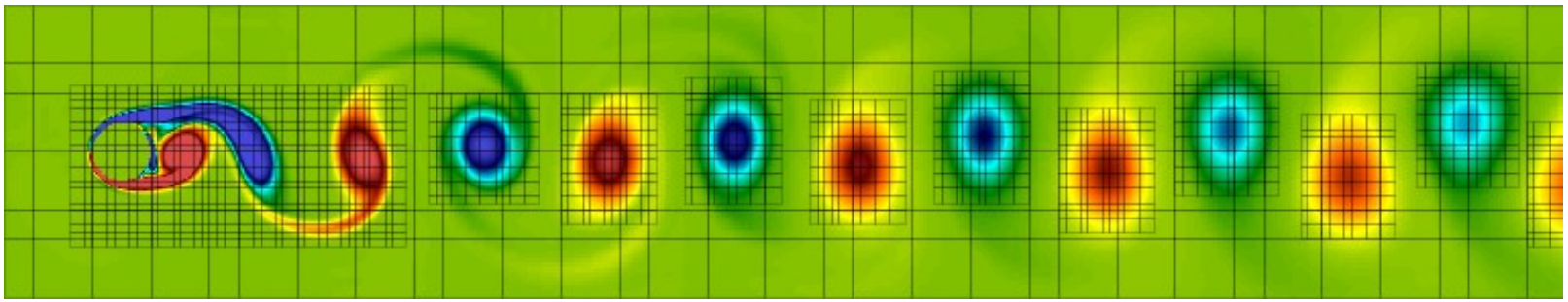
Availability: free for academy and research

Developers: P. Colella, D. T. Graves, J. N. Johnson et al.

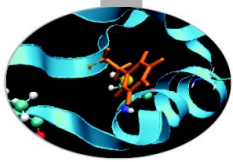
Distributors: D. T. Graves

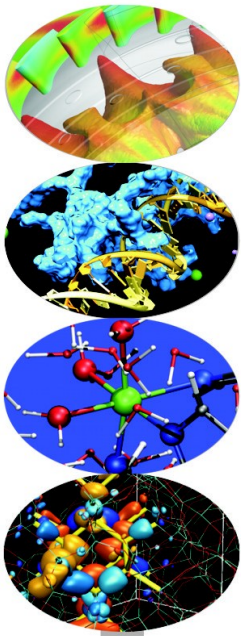
Ref.: Applied Numerical Algorithms Group - Computational Research Division - Lawrence Berkeley National Laboratory- Berkeley, CA, USA

https://commons.lbl.gov/display/chombo/



CINECA SCAI
SuperComputing Applications and Innovation



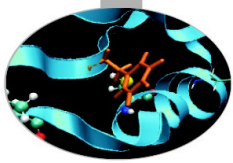


I/O Libraries

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

CINECA

I/O libraries



HDF5

HDF5 is a data model, library, and file format for storing and managing data. It supports an unlimited variety of datatypes, and is designed for flexible and efficient I/O and for high volume and complex data. HDF5 is portable and is extensible, allowing applications to evolve in their use of HDF5. The HDF5 Technology suite includes tools and applications for managing, manipulating, viewing, and analyzing data in the HDF5 format.

Language: Fortran/C/C++/Java

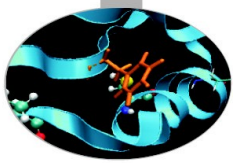
Availability: open source.

Developers: NCSA at the University of Illinois

Distributors: HDF Group

Ref.: http://www.hdfgroup.org

I/O libraries



NetCDF

NetCDF is a set of software libraries and self-describing, machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data.

The NetCDF source-code is hosted at GitHub, and may be found directly at http://github.com/Unidata/netcdf-c.

NetCDF-4 (and pNetCDF) work with MPI version 2 (MPI-2).

Language: Fortran/C/C++/Java

Availability: open source.

Developers: Unidata

Distributors: Unidata

Ref.: http://www.unidata.ucar.edu/netcdf