Numerical Libraries

Bhupender Thakur

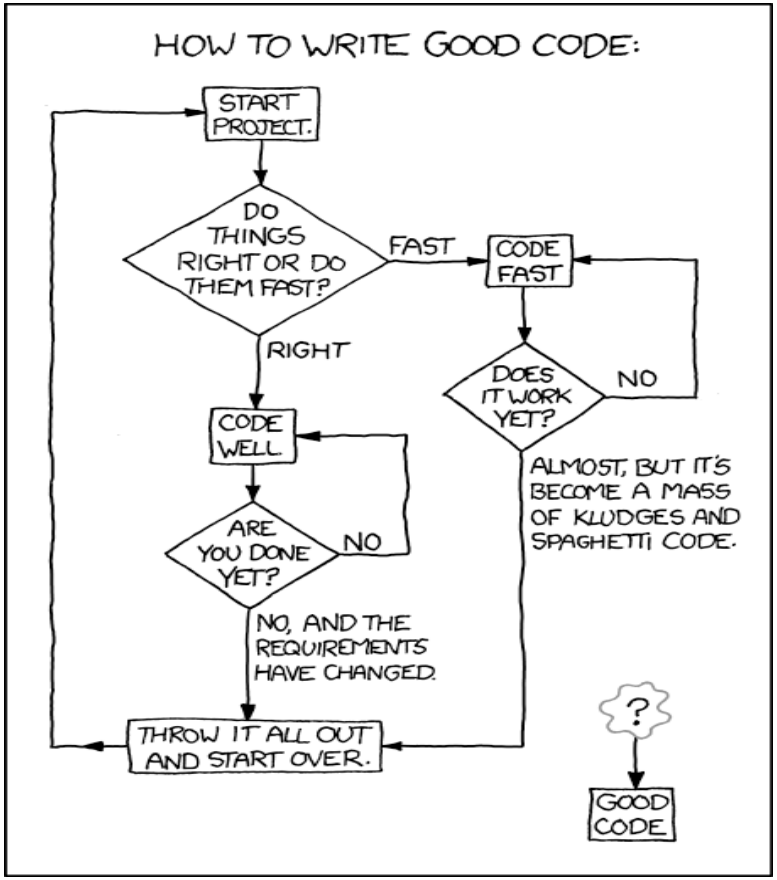
The need for libraries

Why should you try them ?

It is very easy to write bad code !

Loops	: branching, dependencies
I/O	: resource conflicts,
Memory	: long fetch-time
Portability	: code and data portable?
Readability	: Can you still understand it?

Source : http://xkcd.com/844/



The need for libraries

Why should you try them ?

Hopefully these will be some of the advantages

- Computing optimizations
- Easier to debug
- I/O and communication optimizations
- Portability
- Easy to read

Compilers
Serial/parallel across LONI/HPC

- Serial
  - Gcc
  - Intel
  - PGI
- Parallel
  - Mvapich2
  - Openmpi
  - CUDA/ PGI-accel

Know your algorithm

- **CPU bound**: A system in which there is insufficient CPU power to keep the number of runnable processes on the run queue low. This results in poor interactive response by applications.
- **Memory bound**: A system which is short of physical memory, and in which pages of physical memory, but not their contents, must be shared by different processes. This is achieved by paging out, and swapping in cases of extreme shortage of physical memory.
- **I/O bound**: A system prohibited by slow data transfer from I/O device.
- **Communication bound:** Programs where CPU waits for communication to end before computations can be performed. This is the domain of parallel programming.

Libraries

What are they useful for?

- Numerics
- Visualization
- I/O
- Profiling, debugging

Where to find them?

- http://en.wikipedia.org/wiki/List_of_numerical_libraries
- Netlib
- DOE ACTS collection
- ORNL, CERN, NIST, JPL
- Vendor libraries: Intel, PGI, NVIDIA, CRAY

Broswse the list of libraries on any LONI/HPC cluster: softenv

Libraries to be covered

- **BLAS/LAPACK/ARPACK**:
  Linear algebra libraries
- **MPI-IO(ROMIO) /HDF5/NetCDF**:
  I/O libraries
- **PETSc**: Portable, Extensible Toolkit for Scientific Computationl
  Solution of partial differential equations
- **PAPI/TAU:**
  Profiling

Libraries
General features

Know the library location (\root\some\where\libsome.a)

- Lib_DIR=\root\some\where

Look for {lib}, {include}, {bin} subdirectories

- Static : $(Lib_DIR)\lib\libsome.a
- Dynamic: $(Lib_DIR)\lib\libsomeso
- Include files: $(Lib_DIR)\include

Link your executable when compiling

- $(F90) program.f90 –I $(Lib_DIR)\include -L$(Lib_DIR)\lib -lsome

Libraries

Libraries on LONI/HPC

The library location is generally(/usr/local/packages/name/version/build)

- Lib_DIR=/usr/local/packages
  /usr/local/packages/hdf5/1.8.9/Intel-13.0.0-openmpi-1.6.2

Look for {lib}, {include}, {bin} subdirectories

- Static : $(Lib_DIR)/lib/libhdf5.a
- Dynamic: $(Lib_DIR)/lib/libhdf5.so
- Include files: $(Lib_DIR)/include

Link your executable when compiling

- $(F90) program.f90 –I $(Lib_DIR)/include -L$(Lib_DIR)/lib -lsome

Libraries

Libraries on LONI/HPC

Dynamic linker and loader

- DT_RPATH: $(Lib_DIR)/lib/libhdf5.so
- LD_LIBRARY_PATH
- DT_RUNPATH
- /etc/ld.so.cache , /etc/ld.so.conf.d/*
- /lib and /usr/lib

Libraries

Libraries on LONI/HPC

Setting your path(/usr/local/packages/name/version/build**)**

```
export PATH=$PATH:$Lib_DIR/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$Lib_DIR/lib
export LIBRARY_PATH=$LD_LIBRARY_PATH
```

Look for

- Dynamic: $(Lib_DIR)/lib/libhdf5.so
- Include files: $(Lib_DIR)/include

Link your executable

MPI library

Basic Library for non-smp based communication

- Main implementation include mpich2, mvapich2, openmpi
  - Compiler wrapper call regular C for Fortran compiler
  - Runtime interface takes care of communication between processes
- Defined by MPI standard. MPI 3.0 is the most recent addition
- API is based on communicator groups and ranks
  - Total mpi processes
  - Rank of each procss
  - Interface to different programming languages

MPI library

A Look at the OpenMPI library

```
MPI_Top=/usr/local/packages/openmpi/1.6.3/Intel-13.0.0

$ls $MPI_Top
bin  etc  include  lib  share
```

```
$ls $MPI_Top/bin
mpicc
mpif90
mpicxx
mpirun
mpiexec
```

```
$ls $MPI_Top/lib
libmpi.so.1
libmpi.so.1.0.6
libmpi.so
libmpi_f90.so
libmpi_f77.so
libmpi_cxx.so
```

```
$ ls $MPI_Top/include
mpi.h
mpif.h
```

MPI library
Example using mpirun

```
$mpirun -n 4 sh -c ' echo $$'
            9319
            9320
            9321
            9322
```

```
$mpirun -hostfile hosts bash -lc 'h=$(hostname -s);p=$$;echo $h.$p'
      mike029.42952
      mike015.118153
```

MPI library

A simple example

```
$ mpif90 –show
ifort -I/usr/local/packages/openmpi/1.6.2/Intel-13.0.0/include \
      -I/usr/local/packages/openmpi/1.6.2/Intel-13.0.0/lib \
      -L/usr/local/packages/openmpi/1.6.2/Intel-13.0.0/lib \
      -lmpi_f90 -lmpi_f77 -lmpi -ldl -lm -Wl,--export-dynamic -lrt -lnsl -libverbs -libumad
-lpthread –lutil
```

```
$ nm libmpi.so|grep MPI_Comm_rank
0000000000078de0 W MPI_Comm_rank

$ nm libmpi.so|grep MPI_Get_process
000000000007c300 W MPI_Get_processor_name
```

MPI library

A simple example

```
Program hello
  use mpi
  character*10 :: name
  integer :: irank, nproc, err
  call MPI_Init(err)
  call MPI_Comm_Size(MPI_COMM_WORLD, nproc, err)
  call MPI_Comm_Rank(MPI_COMM_WORLD, irank, err)
  call MPI_Get_processor_name(name, nlen, ierr)
  write(*,*)" I am",irank,"of",nproc,"on ", name
  call MPI_Finalize(err)
end
```

MPI library
A simple example

```
Program hello
  use mpi
  character*10 :: name
  integer :: irank, nproc, err
  call MPI_Init(err)
  call MPI_Comm_Size(MPI_COMM_WORLD, nproc, err)
  call MPI_Comm_Rank(MPI_COMM_WORLD, irank, err)
  call MPI_Get_processor_name(name, nlen, ierr)
  write(*,*)" I am",irank,"of",nproc,"on ", name
  call MPI_Finalize(err)
end
```

```
$ mpirun -hostfile hosts ./a.out
 I am      0 of      2 on mike015
 I am      1 of      2 on mike029
```

BLAS

Basic Linear Algebra Subprograms

- Available from: www.netlib.org/blas/
- Three levels of BLAS:
  - Level 1: Vector operations, scalar products and norms
  - Level 2: Matrix-vector operations: y=A*x + c
  - Level 3: Matrix-matrix operations: A=B*C + D*y
- Usually ships with LAPACK
- Highly optimized binaries available from vendors

BLAS

Basic Linear Algebra Subprograms

Level 1 contains *vector operations* of the form

Interchange vectors X and Y

Scale a vector by a constant

Copy vector X to Y

Return a constant times a vector plus a vector

Return the dot product of two vectors

…

_SWAP	$x \leftrightarrow y$	S, D, C, Z
_SCAL	$x \leftarrow \alpha x$	S, D, C, Z, CS, ZD
_COPY	$y \leftarrow x$	S, D, C, Z
_AXPY	$y \leftarrow \alpha x + y$	S, D, C, Z
_DOT	$dot \leftarrow x^T y$	S, D, DS
_DOTU	$dot \leftarrow x^T y$	C, Z
_DOTC	$dot \leftarrow x^H y$	C, Z
__DOT	$dot \leftarrow \alpha + x^T y$	SDS
_NRM2	$nrm2 \leftarrow \|\|x\|\|_2$	S, D, SC, DZ

BLAS

Basic Linear Algebra Subprograms

Level 2 contains matrix *vector operations*

General Matrix vector multiply

Solve system of equations

Perform rank 1 operation

…

Name	Operation	Prefixes
_GEMV	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S,D,C,Z
_GBMV	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S,D,C,Z
_HEMV	$y \leftarrow \alpha Ax + \beta y$	C,Z
_HBMV	$y \leftarrow \alpha Ax + \beta y$	C,Z
_HPMV	$y \leftarrow \alpha Ax + \beta y$	C,Z
_SYMV	$y \leftarrow \alpha Ax + \beta y$	S,D
_SBMV	$y \leftarrow \alpha Ax + \beta y$	S,D
_SPMV	$y \leftarrow \alpha Ax + \beta y$	S,D
_TRMV	$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S,D,C,Z
_TBMV	$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S,D,C,Z
_TPMV	$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S,D,C,Z
_TRSV	$x \leftarrow A^{-1} x, x \leftarrow A^{-T} x, x \leftarrow A^{-H} x$	S,D,C,Z

BLAS

Basic Linear Algebra Subprograms

Level 3 contains *matrix-matrix operations*

General Matrix matrix multiply

Rank-k update

…

Name	Operation	Prefixes
_GEMM	$C \leftarrow \alpha op(A) op(B) + \beta C, op(X) = X, X^T, X^H, C - m \times n$	S,D,C,Z
_SYMM	$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C, C - m \times n, A = A^T$	S,D,C,Z
_HEMM	$C \leftarrow \alpha AB + \beta C, C \leftarrow \alpha BA + \beta C, C - m \times n, A = A^H$	C,Z
_SYRK	$C \leftarrow \alpha AA^T + \beta C, C \leftarrow \alpha A^T A + \beta C, C - n \times n$	S,D,C,Z
_HERK	$C \leftarrow \alpha AA^H + \beta C, C \leftarrow \alpha A^H A + \beta C, C - n \times n$	C,Z
_SYR2K	$C \leftarrow \alpha AB^T + \alpha BA^T + \beta C, C \leftarrow \alpha A^T B + \alpha B^T A + \beta C, C - n \times n$	S,D,C,Z
_HER2K	$C \leftarrow \alpha AB^H + \bar{\alpha} BA^H + \beta C, C \leftarrow \alpha A^H B + \bar{\alpha} B^H A + \beta C, C - n \times n$	C,Z
_TRMM	$B \leftarrow \alpha op(A)B, B \leftarrow \alpha Bop(A), op(A) = A, A^T, A^H, B - m \times n$	S,D,C,Z
_TRSM	$B \leftarrow \alpha op(A^{-1})B, B \leftarrow \alpha Bop(A^{-1}), op(A) = A, A^T, A^H, B - m \times n$	S,D,C,Z

LAPACK
Linear Algebra PACKage

- Routines for
  - Solving systems of linear equations and linear least squares,
  - Eigenvalue problems,
  - Singular value decomposition.
- Important implementations:
  - Netlib.org/lapack, Intel MKL, ATLAS, LACAML, CLAPACK, SciPy, PLASMA, MAGMA,…

LAPACK

Naming convention: **X-YY-ZZZ**

X

- S: Single
- D: Double
- C: Complex

YY

- DI: Diagonal
- SY: Symmetric
- GE: General

ZZZ

- LQF:
  LQ factorization
- EGR:
  Few eigenvalues
- TRD:
  Tridiagonal reduction

Example

Calculate the eigenvalue and eigenvectors of A(n,n):

Call DSYEVD('V', 'U' ,N, A, N, w, work, lwork, iwork, liwork, info)

'V'	Compute eigenvalue and eigenvectors;
'U'	Upper triangle of mat is stored;
A	N*N matrix on (input) and eigenvectors on (output);
w	Array of (output) eigenvalue;
work	Real (workspace/output) of dimension lwork;
iwork	Integer (workspace/output) of dimension liwork;
info	(Output) flag for successful run.

Example (cont.)

```
program ex_dsyev
  use global
…
  allocate( w (n), &
            work (1000), &
             iwork(1000) )
…
Call DSYEV('V', 'U', n, mat, n, w, &
 work, 1000, iwork,1000,info)
…
```

Link to lapack and blas:

On LONI add following to the soft file

```
+lapack-3.2-intel-11.1

DIR=/usr/local/packages/lapack/3.2/
intel-11.1
LIBS= -L$(DIR)/lib -llapack  -lblas

ifort global.f90 ex_dsyev.f90 $(LIBS)
```

Intel MKL

Fast LAPACK by Intel (http://software.intel.com/sites/products/mkl/)
LDIR = /usr/local/compilers/Intel/mkl-10.2.2.025/lib/em64t

Dynamic linking

```
LIBS=-shared-intel -Wl, -rpath,$
        (LDIR)  -L$(LDIR)
        -lmkl_intel_lp64
        -lmkl_intel_thread
        -lmkl_core -lguide –lpthread
```

Static linking

```
LIBS=-shared-intel -Wl,--start group
    $(LDIR)/libmkl_intel_lp64.a
    $(LDIR)/libmkl_intel_thread.a
    $(LDIR)/em64t/libmkl_core.a
    -Wl,--end-group  -lguide
    -lpthread
```

ARPACK

Large sparse eigenvalue problem

- LAPACK requires as input A(N, N)
- Consider N=1,000,000

  Storage = $(10^{12} * 4)/(1024)^3$ = 3725 GB

- Fortunately,
  - Matrices are usually sparse
  - Iterative algorithms need not store full matrix
  - +arpack-96-intel-11.1 on LONI

Sparse matrices

Sparse matrix formats store non-zero elements and arrays referencing them

- *Coordinate format*

  idx(k), jdx(k), r(k)

- *Matvec operation (v=Au)*

  v(idx(k)) = v(idx(k))
  + u(jdx(k))*r(k)

10	30	0
20	40	60
0	50	70

idx= [1 2 1 2 3 2 3]
jdx= [1 1 2 2 2 3 3]
r = [10 20 30 40 50 60 70]

ARPACK

Large sparse eigenvalue problem

```
progran test_lanczos

use modarpack
use sparpack

…

    which_eig          =  "LM"
    n_eigvalues        =  2
    tolerance          = .01d0
    get_vectors        = .true.
    i_guesspivot       = 0
…
```

Code is a modified example from arpack
examples/SYM/dsdrv1.f

What spectrum of eigenvalues is requested? Largest magnitude

Number of eigenvalues

Accuracy of eigenvalues

Need eigenvectors as well?

ARPACK

Large sparse eigenvalue problem

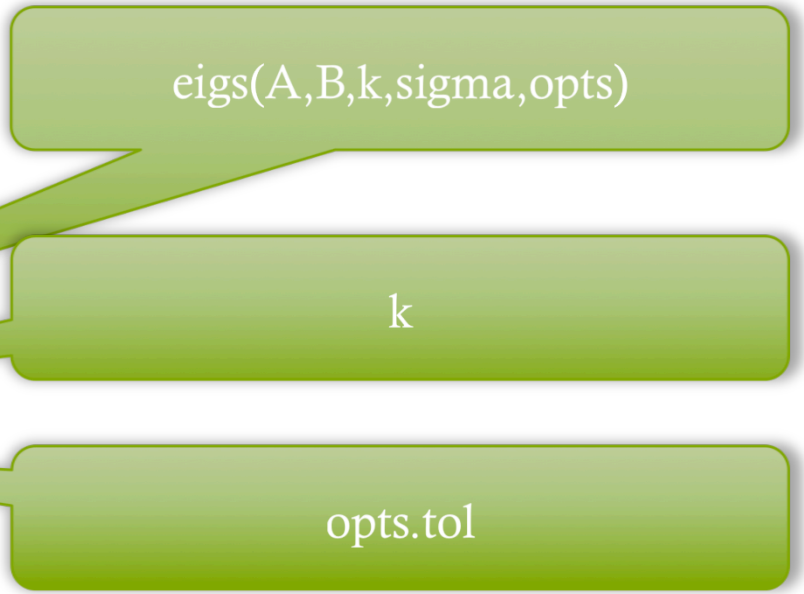
```
progran test_lanczos

use modarpack
use sparpack

…

    which_eig          =  "LM"
    n_eigvalues        =  2
    tolerance          = .01d0
    get_vectors        = .true.
    i_guesspivot       = 0
…
```

Code is a modified example from arpack
examples/SYM/dsdrv1.f



ARPACK

Large sparse eigenvalue problem

- **Matlab incorporates ARPACK as eigs**

```
eigs(A,B,k,sigma,opts)
d = eigs(A)
[V,D] = eigs(A)
[V,D,flag] = eigs(A)
eigs(A,B)
eigs(A,k)
eigs(A,B,k)
eigs(A,k,sigma)
eigs(A,B,k,sigma)
eigs(A,K,sigma,opts)
eigs(A,B,k,sigma,opts)
eigs(Afun,n,...)
```

opts.tol : Tolerance
opts.maxit : Max iterations
opts.v0 : Starting vector

References

[1] Lehoucq, R.B. and D.C. Sorensen, "Deflation Techniques for an Implicitly Re-Started Arnoldi Iteration," *SIAM J. Matrix Analysis and Applications*, Vol. 17, 1996, pp. 789–821.

ARPACK

Large sparse eigenvalue problem

Reverse interface for user matvec

```
call dsaupd ( ido, bmat, n, which, nev, tol, resid, &
              ncv, v, ldv, iparam, ipntr, workd, workl, &
              lworkl, info )

if (ido .eq. -1 .or. ido .eq. 1) then

   %-----------------------------------%
   | Perform matrix vector multiplication |
   |              y <--- OP*x          |
   | The user should supply his/her own |
   | matrix vector multiplication routine |
   | here that takes workd(ipntr(1)) as |
   | the input, and return the result to |
   | workd(ipntr(2)).                  |
   %-----------------------------------%

   call av (nx, workd(ipntr(1)), workd(ipntr(2)))
```

User supplied matrix-vector multiplication

LSU

ARPACK

Large sparse eigenvalue problem

```
#!/bin/bash

arpack_dir=/usr/local/packages/arpack/96/intel-11.1/lib
lapack_dir=/usr/local/packages/lapack/3.2/intel-11.1/lib

ifort -o cpu.exe \
      \
        sparpack.f90 modarpack.f90 test_lanczos.f90 \
      \
       -L${arpack_dir} -larpack_LINUX \
      \
       -L${lapack_dir} -llapack -lblas
```

ARPACK

Large sparse eigenvalue problem

```
progran test_lanczos

use modarpack
use sparpack

…

   which_eig          =  "LM"   ! Which ?
   n_eigvalues        =  2      ! Eigenvalues
   tolerance          = .01d0   ! Tolerance
   get_vectors        = .true.  ! Eigenvectors
   i_guesspivot       = 0       ! Guess
…
```

```
[bthakur@eric2]$ ./cpu.exe

 E-vals  -6.38745949An HDF5 file is a
container for storing a variety of scientific
data and is composed of two primary types of
objects: groups and datasets.107957
 E-vals   6.07252812124572

 _SDRV1
 ======

 Size of the matrix is                51537
 Ritz values requested is             2
 Arnoldi vecs generated(NCV)          3
 Portion of the spectrum:             LM
 Number of converged values           2
 Implicit Arnoldi iterations          135
 The number of OP*x is                137
 The convergence criterion            0.01
```

More libraries

- **Parallel libraries beyond LAPACK/ARPACK are available**
  - **SLEPSc** Scalable Library for Eigenvalue Problem Computations
  - **Hypre** Solves large, sparse linear systems of equations on massivelly parallel computers
  - **Blopex** parallel preconditioned eigenvalue solvers
  - **Plasma** Parallel Linear Algebra Software for Multicore Architectures
  - **Scalapack** Scalable LAPACK

Parallel I/O
HDF5/NetCDF/PNetCDF

HDF / HDF4/ HDF5 (Hierarchical Data Format)

- Originally developed by NCSA, now supported by non-profit HDF5 group
- Current release version 5-1.8.10 / Nov 13, 2012

NetCDF (Network Common Data Form)

- Originally based on Common Data Format developed by NASA
- Current release version 4.2.1.1 / Aug 3, 2012
- NetCDF API version 4.0 allows the use of the HDF5 data format.

Parallel-netCDF

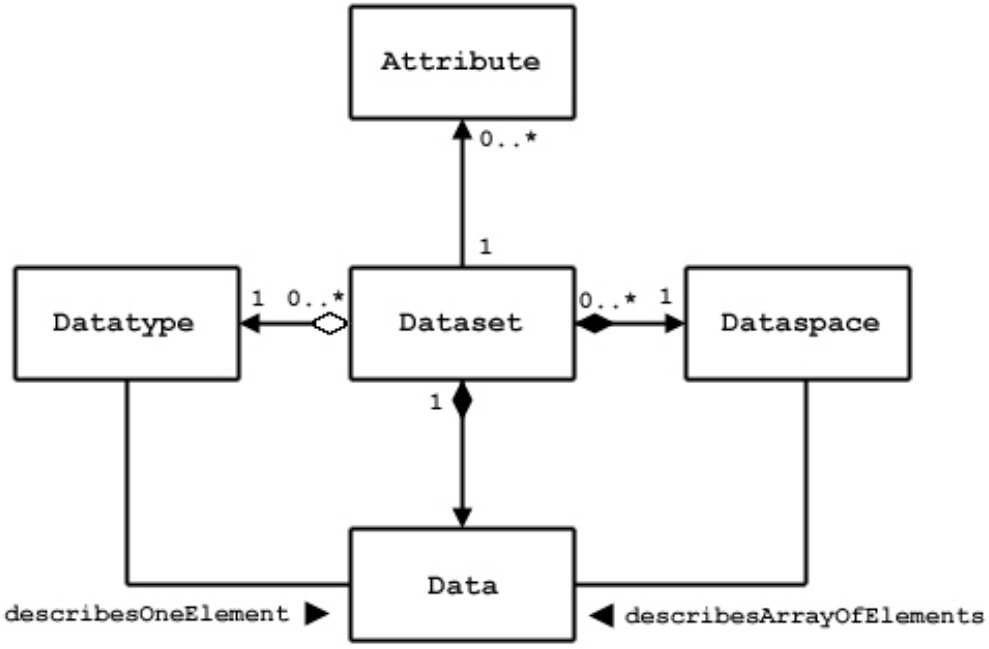
- An extension of netCDF for parallel computing developed by Argonne.
- Uses MPI-IO and communications and high-level netCDF data structures.
- Uses C/Fortran APIs, different from, but similar to those of netdcf.

HDF5: Organization

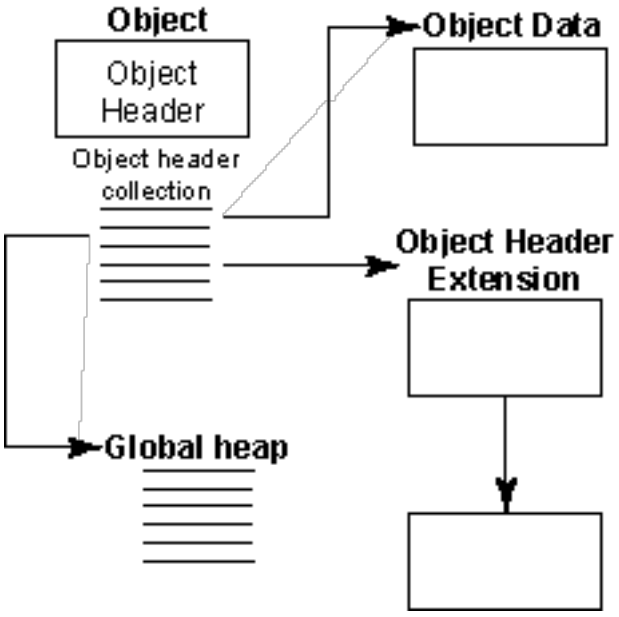
An HDF5 file is a container for storing a variety of scientific data and is composed of two primary types of objects: groups and datasets.

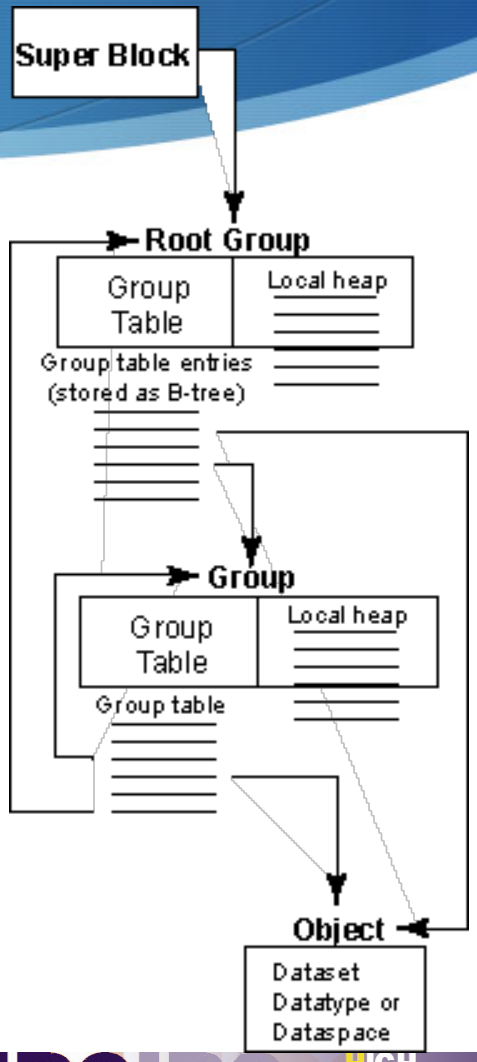
- **HDF5 group:** a grouping structure containing zero or more HDF5 objects, together with supporting metadata
- **HDF5 dataset**: a multidimensional array of data elements, together with supporting metadata. To create a dataset, the application program must specify the location at which to create the dataset, the dataset name, the datatype and dataspace of the data array, and the property lists.

HDF5

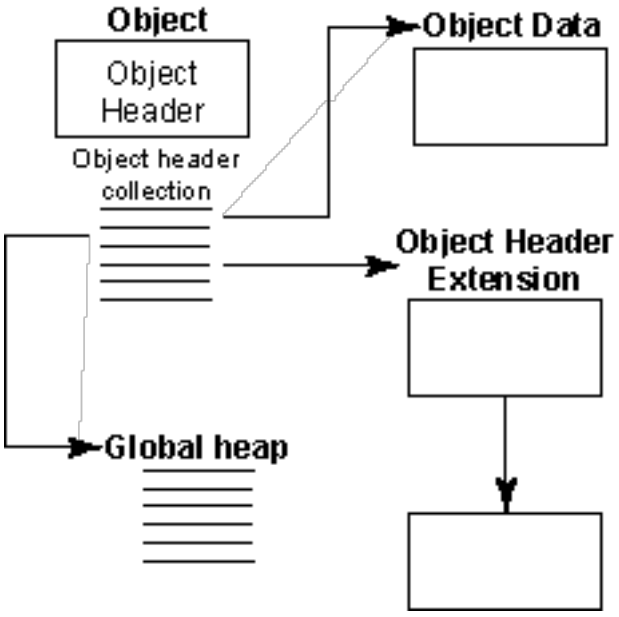

Attribute
0..*
1
Datatype
1
0..*
Dataset
0..*
1
Dataspace
1
Data
describesOneElement
describesArrayOfElements

HDF5


Object
Object Header
Object header collection
Object Data
Object Header Extension
Global heap


Super Block
Root Group
Group Table
Local heap
Group table entries (stored as B-tree)
Group
Group Table
Local heap
Group table
Object
Dataset
Datatype or
Dataspace

HDF5



HDF5 datatypes
- atomic
  - integer
  - floating point
  - date and time
  - character string
  - bitfield
  - opaque
- compound

HDF5 dataspaces
- simple
- complex

HDF5

```
    #include "hdf5.h"
    #define FILE "file.h5"

    int main()
{
    hid_t       file_id;   /* file identifier */
    herr_t      status;

/* Create a new file. */
    file_id = H5Fcreate(FILE, H5F_ACC_TRUNC,
                H5P_DEFAULT, H5P_DEFAULT);

/* Terminate access to the file. */
    status = H5Fclose(file_id);
}
```

```
PROGRAM FILEEXAMPLE

    use hdf5
    implicit none
    character(len=8), parameter :: fl= "filef.h5"
    integer(hid_t) :: file_id
    integer     ::   error
    !
    call h5open_f (error)
    !
    call h5fcreate_f(fl, H5F_ACC_TRUNC_F, file_id, error)
    !
    ! Terminate access to the file.
    !
    call h5fclose_f(file_id, error)
    !
    call h5close_f(error)
END PROGRAM FILEEXAMPLE
```

HDF5

```
PROGRAM DSETEXAMPLE
   USE HDF5 ! This module contains all necessary modules
   IMPLICIT NONE
   CHARACTER(LEN=8), PARAMETER :: filename =
"dsetf.h5" ! File name
   CHARACTER(LEN=4), PARAMETER :: dsetname = "dset"


   INTEGER(HID_T) :: file_id                  ! File identifier
   INTEGER(HID_T) :: dset_id                  ! Dataset identifier
   INTEGER(HID_T) :: dspace_id                ! Dataspace identifier
   INTEGER(HSIZE_T), DIMENSION(2) :: dims = (/4,6/)
   INTEGER     ::   rank = 2                  ! Dataset rank
   INTEGER     ::   error                     ! Error flag
   !
   ! Initialize FORTRAN interface.
   !
   CALL h5open_f(error)

   ! Create a new file using default properties.

   CALL h5fcreate_f(filename, H5F_ACC_TRUNC_F, file_id, error)
   CALL h5screate_simple_f(rank, dims, dspace_id, error)
   CALL h5dcreate_f(file_id, dsetname, H5T_NATIVE_INTEGER,
   dspace_id, &dset_id, error)

   CALL h5dclose_f(dset_id, error)!
   CALL h5sclose_f(dspace_id, error)     !

   CALL h5fclose_f(file_id, error)

   ! Close FORTRAN interface.

   CALL h5close_f(error)
   END PROGRAM DSETEXAMPLE
```

HDF5

- **HDF5 Demos**
  - Create group:
  - Create file:
  - Create datasets
  - Parallel examples
    http://www.hdfgroup.org/HDF5/Tutor/parallel.html

HDF5

- **Examples of what you can do with the Parallel HDF5 collective API:**
  - File Operation: Create, open and close a file
  - Object Creation: Create, open, and close a dataset
  - Object Structure: Extend a dataset (increase dimension sizes)
  - Dataset Operations: Write to or read from a dataset (Array data transfer can be collective or independent.)

- **Once a file is opened by the processes of a communicator**
  - All parts of the file are accessible by all processes.
  - All objects in the file are accessible by all processes.
  - Multiple processes write to the same dataset.
  - Each process writes to a individual dataset.

NetCDF

- **What is NetCDF?**
  - NetCDF is a set of data formats, programming interfaces, and software libraries that help read and write scientific data files.
- **The Classic NetCDF Data Model**
  - The classic netCDF data model consists of variables, dimensions, and attributes. This way of thinking about data was introduced with the very first netCDF release, and is still the core of all netCDF files.

NetCDF

- **The Classic NetCDF Data Model**
  - ***Variables***: N-dimensional arrays of data. Variables in netCDF files can be one of six types (char, byte, short, int, float, double)
  - ***Dimensions*** describe the axes of the data arrays. A dimension has a name and a length. An unlimited dimension has a length that can be expanded at any time. NetCDF files can contain at most one unlimited dimension.
  - ***Attributes*** annotate variables or files with small notes or supplementary metadata. Attributes are always scalar values or 1D arrays, which can be associated with either a variable or the file.

NetCDF
Example

```
program simple_xy_wr
  use netcdf
  implicit none
! Name of the data file
  character (len = *), parameter ::
FILE_NAME = "simple_xy.nc”
! We are writing 2D data, a 6 x 12 grid.

integer, parameter :: NDIMS = 2
integer, parameter :: NX = 6, NY = 12
integer :: ncid, varid, dimids(NDIMS)
integer :: x_dimid, y_dimid

! This is the data array
 integer :: data_out(NY, NX)
```

NetCDF

```
! Create the netCDF file.
  call check( nf90_create(FILE_NAME,
                NF90_CLOBBER, ncid) )

! Define dimensions. It hands ID for each.
  call check( nf90_def_dim(ncid, "x", NX,
                x_dimid) )

  call check( nf90_def_dim(ncid, "y", NY,
                y_dimid) )
  dimids =  (/ y_dimid, x_dimid /)

! Define the variable ttype: NF90_INT
  call check( nf90_def_var(ncid, "data",
                NF90_INT, dimids, varid) )

  call check( nf90_put_var(ncid, varid,
  data_out) )

  call check( nf90_close(ncid) )

  end
```

NetCDF

Utilities

- nc-config
- nf-config
- ncdump

PETSc

- Aimed at parallel non-trivial PDE solvers
- Portable to any parallel system supporting MPI
- Offers robust scaling
- Supports many languages: C, Fortran, Python

PETSc
Components

Vec:
Provides the vector operations required for setting up and solving large-scale linear and nonlinear problems. Includes easy-to-use parallel scatter and gather operations, as well as special-purpose code for handling ghost points for regular data structures.

Mat:
A large suite of data structures and code for the manipulation of parallel sparse matrices. Includes four different parallel matrix data structures, each appropriate for a different class of problems.

PC:
A collection of sequential and parallel preconditioners, including (sequential) ILU(k), LU, and (both sequential and parallel) block Jacobi, overlapping additive Schwarz methods and structured MG using DMMG.

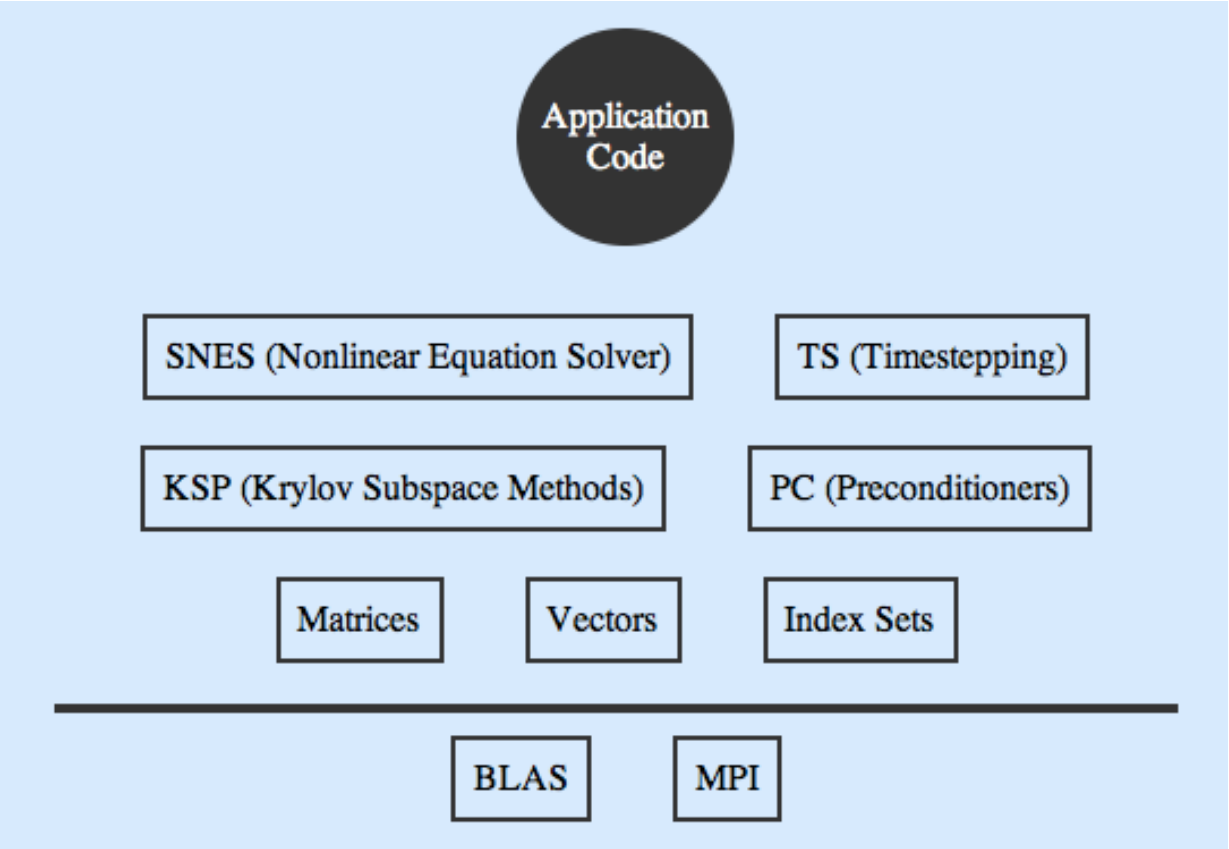
PETSc
Components

KSP:
Parallel implementations of many popular Krylov subspace iterative methods, including GMRES, CG, CGS, Bi-CG-Stab, two variants of TFQMR, CR, and LSQR, immediately usable with any preconditioners and any matrix data structures, including matrix-free methods.

SNES:
Data-structure-neutral implementations of Newton-like methods for nonlinear systems. Includes both line search and trust region techniques with a single interface. Users can set custom monitoring routines, convergence criteria, etc.

TS:
Code for the time evolution of solutions of PDEs. In addition, provides pseudo-transient continuation techniques for computing steady-state solutions.

PETSc
Components



LSU

PETSc

- **Initialize PETSc**

```
#include "finclude/petsc.h90”
#include "finclude/petscvec.h90”

#include "finclude/petsc.h"
#include "finclude/petscvec.h"

Call PetscInitialize()
Call MPI_Comm_rank(PETSC_COMM_WORLD,rank,ierr)
```

PETSc

PETSc objects and procedures

VecCreateSeq()	VecScale(), VecNorm()
VecCreateMPI()	VecAXPY(), VecDot()

Example

```
Vec                          v
PetscScalar pointer ::       array(:,:)
PetscInt                     n, i
PetscErrorCode               ierr
call VecGetArrayF90(v, array, ierr)
call VecGetLocalSize(v, n, ierr)
do i=1,n
  array(i) = array(i) + rank
end do
```

PETSc

PETSc objects and procedures

MatCreate(MPI_Comm, Mat *)

MatSetSizes(Mat, PetscInt m, PetscInt n, M, N)

MatSetType(Mat, MatType typeName)

MatSetFromOptions(Mat)

Single user interface but multiple underlying implementations

PETSc

Higher abstractions

The PETSc DA class is a topology and discretization interface.

The PETSc Mesh class is a topology interface.

The PETSc DM class is a hierarchy interface.

The PetscSection class is a helper class for data layout.

Profiling and debugging

PETSc has integrated profiling, logging events.

Higher level of error detection

PETSc
Example

```
program main

#include "finclude/petsc.h"
#include "finclude/petscvec.h"

!  Variables:
!     x, y, w - vectors
!     z       - array of vectors

      Vec             x,y,w,z(5)
      PetscReal       norm,v,v1,v2
      PetscInt        n,ithree
      PetscTruth      flg
      PetscErrorCode ierr
      PetscMPIInt  rank
      PetscScalar  one,two,three

call PetscInitialize &
(PETSC_NULL_CHARACTER,ierr)
…
call PetscOptionsGetInt &
(PETSC_NULL_CHARACTER,'n',n,flg,ierr)
…
call MPI_Comm_rank &
(PETSC_COMM_WORLD,rank,ierr)
…
call VecCreate &
(PETSC_COMM_WORLD,x,ierr)
…
call VecDot(x,x,dot,ierr)
call VecNorm(x,NORM_2,norm,ierr)
call VecDestroy(x,ierr)
…
call PetscFinalize(ierr)

end
```

PETSc
Example

 # Makefile

```
PETSC_DIR = /usr/local/packages/petsc/3.0.0.p3/intel-11.1-mpich-1.2.7p1

include ${PETSC_DIR}/conf/base

ex1f: ex1f.o  chkopts
          -${FLINKER} -o ex1f ex1f.o ${PETSC_VEC_LIB}
           ${RM} -f ex1f.o
```

Summary

- We reviewed some basic libraries, which form the core of many computational algorithms.
- Hopefully, through this tutorial, you have learnt how to use libraries to write more efficient programs.