PRACE PATC Course: Advanced Topics in HPC
High-Level I/O Library: HDF5

Sandra Mendez, PhD - HPC Group, LRZ

sandra.mendez@lrz.de

Outline

High-Level I/O Library

- An API which helps to express scientific simulation data in a more natural way
  - Multi-dimensional data, labels and tags, noncontiguous data, typed data
- Offers
  - Simplicity for visualization and analysis
  - Portable formats - can run on one machine and take output to another
  - Longevity - output will last and be accessible with library tools and no need to remember version number of code

High-Level I/O Library (2)

- Scientific applications work with structured data and desire more self- describing file formats
- netCDF and HDF5 are two popular "higher level" I/O libraries
  - Abstract away details of file layout
  - Provide standard, portable file formats
  - Include metadata describing contents
- Parallel version should be built on top of MPI-IO and can use MPI-IO optimizations.

HDF5 - Hierarchical Data Format 5

Hierarchical Data Format 5

HDF5 is designed at three levels:

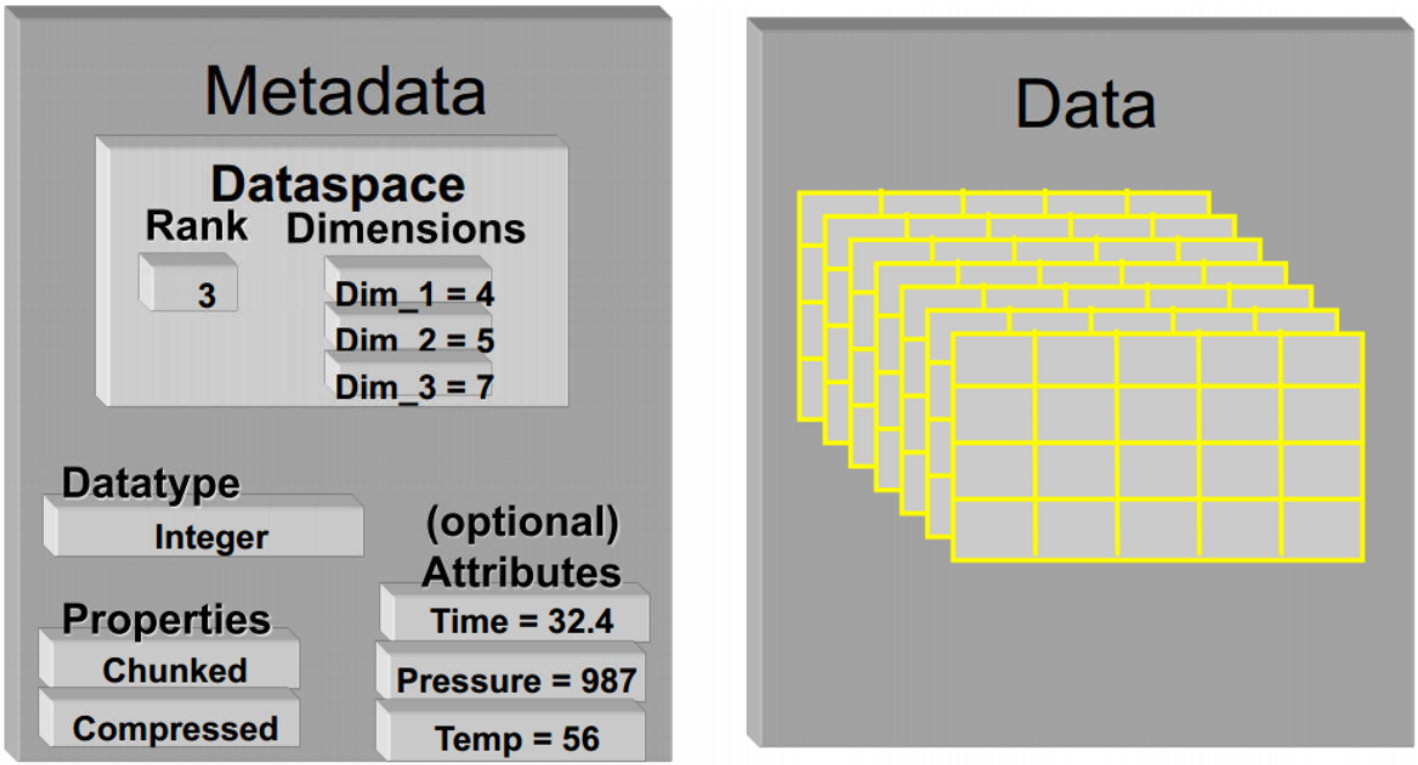
- A data model
  - consists of abstract classes, such as files, datasets, groups, datatypes and dataspaces.
  - Developers use them to construct a model of their higher-level concepts.
- A Software library
  - designed to provide applications with an object-oriented programming interface.
  - a powerful, flexible and high performance interface.
- A file format: provides portable, backward and forward compatible, and extensible instantiation of the HDF5 data model.

File Organization and Data Model

- HDF5 files are organized in a hierarchical structure, with two primary structures: groups and datasets.
  - HDF5 group: a grouping structure containing instances of zero or more groups or datasets, together with supporting metadata.
  - HDF5 dataset: a multidimensional array of data elements, together with supporting metadata.
- The primary classes in the HDF5 data model are:
  - File
  - Dataset
  - Group
  - Link
  - Attribute

File Organization and Data Model

- HDF5 datasets organize and contain data elements.
- HDF5 datatype describes individual data elements.
- HDF5 dataspace describes the logical layout of the data elements.



Source: http://press3.mcs.anl.gov/computingschool/files/2014/01/QKHDF5-Intro-v2.pdf

The General HDF5 API

- Similarities to NetCDF:
  - a container for storing a variety of scientific data
  - HDF5 group: a grouping structure containing HDF5 objects
  - HDF5 dataset: a multidimensional array of data elements
- Web site: www.hdfgroup.org/HDF5 (see especially tutorial)

HDF5 interface conventions	
H5	general purpose library functions
H5A	annotations: attribute access and manipulation routines
H5D	dataset access and manipulation routines
H5E	error handling routines
H5F	file access routines
H5G	group creation and operation routines
H5I	identifier routines
H5L	link routines
H5O	object routines
H5P	object property list manipulation routines
H5R	reference routines
H5S	dataspace definition and access routines
H5T	datatype creation and manipulation routines
H5Z	compression routine(s)

Creation of a HDF5 file

```
PROGRAM FILEEXAMPLE
  USE HDF5
  IMPLICIT NONE
  CHARACTER(LEN=8), PARAMETER :: &
      filename = "filef.h5"
  INTEGER(HID_T) :: file_id
  INTEGER     ::   error
! Initialize HDF5
  CALL h5open_f(error)
! Create a new file using default
! properties.
  CALL h5fcreate_f(filename, &
   H5F_ACC_TRUNC_F, file_id, error)
! Terminate access to the file.
  CALL h5fclose_f(file_id, error)
! Terminate HDF5.
  CALL h5close_f(error)
END PROGRAM FILEEXAMPLE
```

- Creation modes:
  - H5F_ACC_TRUNC if the file already exists, current contents will be deleted → rewrite the file with new data.
  - H5F_ACC_EXCL the open will fail if the file already exists; ignored of if the file does not already exist.

  all the above → reads and writes are possible
  - H5F_ACC_RDONLY read only.
  - H5F_ACC_RDWR read and write.
- Looking at created (binary) file:

  `h5dump filef.h5`

  dumps a DDL- "data description language" form of the file.

Creating a dataset

```
 PROGRAM DSETEXAMPLE
 USE HDF5
 IMPLICIT NONE
 CHARACTER(LEN=8), PARAMETER :: filename = "dsetf.h5"
 CHARACTER(LEN=4), PARAMETER :: dsetname = "dset"
 INTEGER(HID_T) :: file_id
 INTEGER(HID_T) :: dset_id
 INTEGER(HID_T) :: dspace_id
 INTEGER(HSIZE_T), DIMENSION(2) :: dims = (/6,4/)
 INTEGER     ::   rank = 2
 INTEGER     ::   error
 CALL h5open_f(error)
 CALL h5fcreate_f(filename, H5F_ACC_TRUNC_F, file_id, error)
 ! Create the dataspace:
   CALL h5screate_simple_f(rank, dims, &
                           dspace_id, error)
! Create the dataset with default properties:
   CALL h5dcreate_f(file_id, dsetname, &
H5T_NATIVE_INTEGER, dspace_id, dset_id, error)
! End access and release resources:
  CALL h5dclose_f(dset_id, error)
! Terminate access to the data space:
  CALL h5sclose_f(dspace_id, error)
  CALL h5fclose_f(file_id, error)
  CALL h5close_f(error)
END PROGRAM DSETEXAMPLE
```

- Datatype of a dataset: use pre-defined set or user-defined types
- Can then read from and write to a dataset using:

```
call h5dread_f(dset_id, &
mem_type_id, buf, dims, error)
call h5dwrite_f(dset_id, &
mem_type_id, buf, dims, error)
```

  - where mem_type_id, buf and dims must be consistent with the values defined for the dataset
  - query routines available

Example for DDL output

```
HDF5 "dsetf.h5" {
 GROUP "/" {
  DATASET "dset" {
    DATATYPE { H5T_STD_I32BE }
    DATASPACE { SIMPLE ( 4, 6 ) / ( 4, 6 ) }
    DATA {
     1, 2, 3, 4, 5, 6,
     7, 8, 9, 10, 11, 12,
     13, 14, 15, 16, 17, 18,
     19, 20, 21, 22, 23, 24
      }
    }
  }
}
```

native integer: 32 bit / big endian

simple array structure (note ordering)

DATASPACE{SIMPLE<current_dims>/<max_dims>

Attributes

- Attributes are small datasets
  - contained inside "data" datasets
  - usually used for providing information about the nature and/or the intended usage of the object they are attached to
  - attribute creation:

```
call h5acreate_f(dset_id, attr_nam, type_id, space_id, &
                 attr_id, hdferr)

call h5aclose_f(attr_id, hdferr)
```

  - reading and writing attributes:

```
call h5awrite_f(attr_id, mem_type_id, buf, dims, hdferr)

call h5aread_f(attr_id, mem_type_id, buf, dims, hdferr)
```

Groups

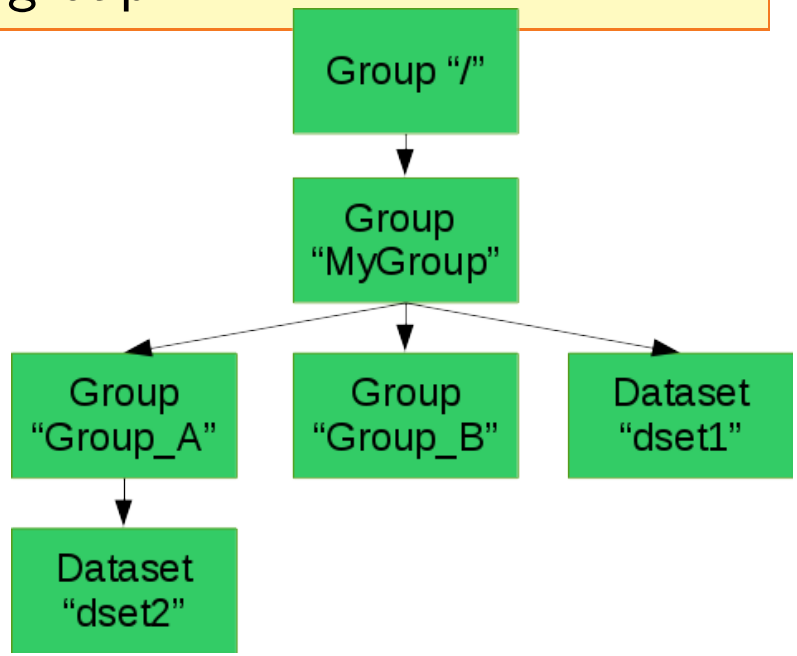
- Create as subgroups of the default root group:

```
call h5gcreate_f(loc_id, name, &
group_id, error)

call h5gclose_f(group_id, error)
```

loc_id absolute or relative

name may be root group (file_id) or other existing group

- Create datasets inside groups:
  - use group id instead of file id as an argument for h5dcreate_f()



Extendible datasets

- May want to change the size of the dataset
  - grow or shrink any of the defined dimensions
  - need to enable chunking via the properties interface → no reorganization of storage required
- Call sequence after creation of file
  - see example code
  - dims → initial dimension
  - dims1 → fixes chunks
  - both: integer arrays of size 2

```
! Create rank 2 data space with
! unlimited dimensions.
maxdims = (/H5S_UNLIMITED_F, &
            H5S_UNLIMITED_F/)
CALL h5screate_simple_f(2,&
     dims, dataspace, error, &
     maxdims)
! Modify dataset creation
! properties (here chunking)
CALL h5pcreate_f( &
   H5P_DATASET_CREATE_F, &
   crp_list, error)
CALL h5pset_chunk_f(crp_list, &
   2, dims1, error)
! continued on next slide
```

Extension procedure

Writing of data

- need to keep size of written data consistent with presently configured size
- chunking size – tune for performance

Reading data

- not shown here
- query calls for required properties are available in H5S and H5D

```
! Create a dataset with 3X3 dimensions
! using cparms creation properties:
   CALL h5dcreate_f(file_id, dsetname,  &
         H5T_NATIVE_INTEGER, dataspace, &
         dset_id, error, crp_list)
! Extend the dataset:
   size = (/ 3, 3 /) ! assured size
   CALL h5dextend_f(dset_id, size, error)
! Extend to 10 x 3:
   size = (/ 10, 3 /)
   CALL h5dextend_f(dset_id, size, error)
! Write data of size 10 x 3 to dataset:
   data_dims = (/ 10, 3 /)
   CALL h5dwrite_f(dset_id, &
         H5T_NATIVE_INTEGER, data_in, &
         data_dims, error)
! Close the dataspace, property list,
! the dataset and the file (not shown)
```

Handling subsets of datasets

Perform a selection on a dataspace

- calls to select hyperslabs or element sets

```
call h5sselect_hyperslab_f(space_id, operator, start, count, &
                           hdferr [, stride, block])
```

- arguments:
  - operator: **H5S_SELECT_SET_F** (set new selection) or **H5S_SELECT_OR_F** (add to existing selection)
  - start: offset, count: number of blocks (integer(**HSIZE_T**) arrays)
  - stride, block: optional integer(**HSIZE_T**) arrays

```
call h5sselect_elements_f(space_id, operator, rank, &
                          num_elements, coord, hdferr)
```

- arguments:
  - integer(HSIZE_T) :: coord(rank, num_elements) – coordinates of selected elements

Illustration

- An 8 x 10 array dataset
  - after a 3 x 4 subset has been overwritten
- Note that the absolute index of arguments for hyperslab creation is zero-based:

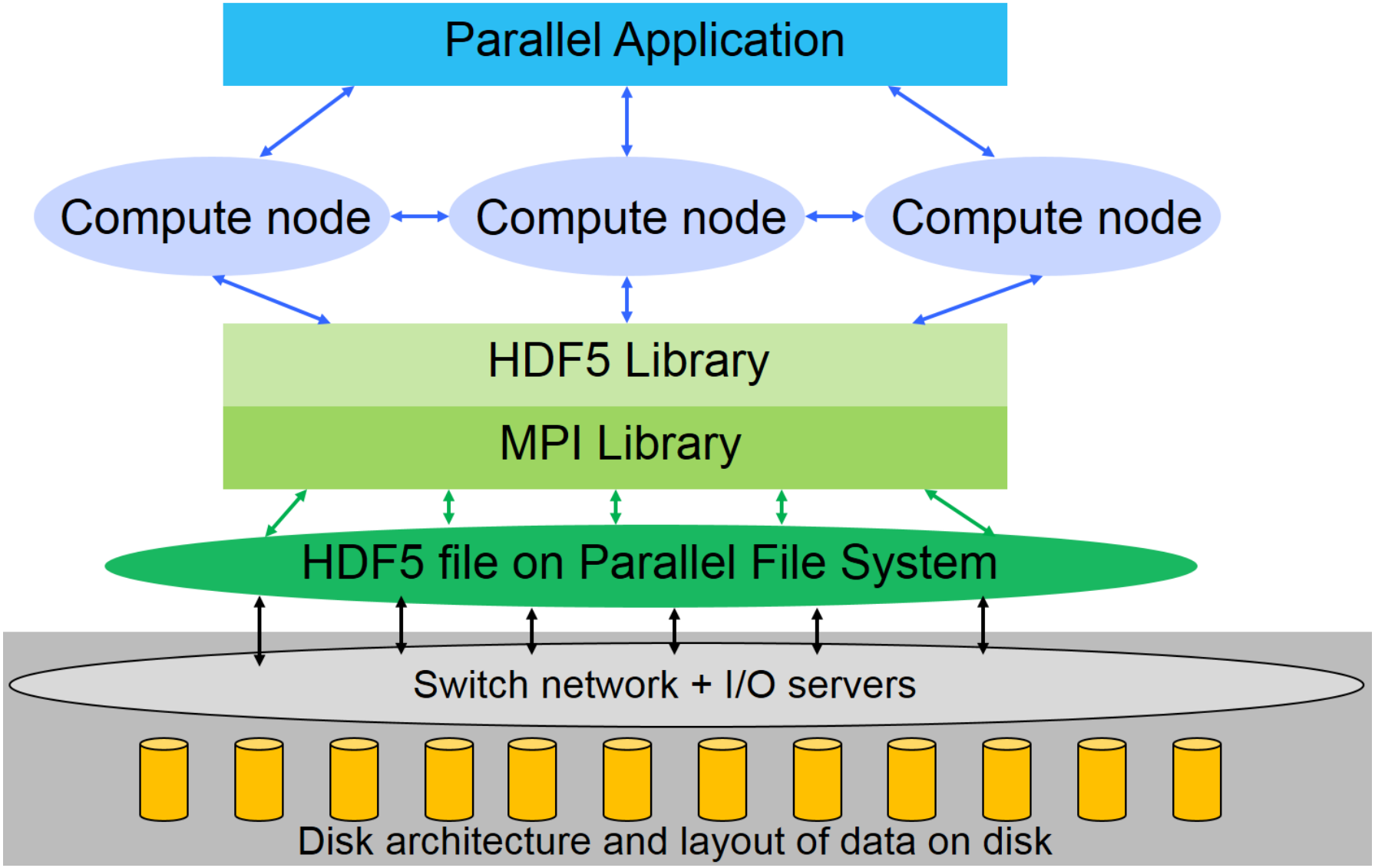
```
start = (/ 1, 2 /)
count = (/ 3, 4 /)
stride = (/ 1, 1 /)
```

1	1	1	1	1	2	2	2	2	2
1	1	5	5	5	5	2	2	2	2
1	1	5	5	5	5	2	2	2	2
1	1	5	5	5	5	2	2	2	2
1	1	1	1	1	2	2	2	2	2
1	1	1	1	1	2	2	2	2	2
1	1	1	1	1	2	2	2	2	2
1	1	1	1	1	2	2	2	2	2

Parallel HDF5 Requirements

- Parallel HDF5 should allow multiple processes to perform I/O to an HDF5 file at the same time
  - A single file image to all processes, rather than having one file per process.
  - Having one file per process can cause expensive post processing, and the files are not usable by different processes.
- A standard parallel I/O interface that must be portable to different platforms.
- Support Message Passing Interface (MPI) programming
- Parallel HDF5 files had to be compatible with serial HDF5 files and sharable between different serial and parallel platforms.

Parallel HDF5 implementation



Programming restrictions

- PHDF5 opens a parallel file with an MPI communicator
- Returns a file ID
- Future access to the file via the file ID
- All processes must participate in collective PHDF5 APIs
- Different files can be opened via different communicators

Collective HDF5 calls

- All HDF5 APIs that modify structural metadata are collective.
  - File operations

    H5Fcreate, H5Fopen, H5Fclose, etc
  - Object creation

    H5Dcreate, H5Dclose, etc
  - Object structure modification (e.g., dataset extent modification)

    H5Dset_extent, etc

  http://www.hdfgroup.org/HDF5/doc/RM/CollectiveCalls.html
- Array data transfer can be collective or independent
  - Dataset operations: H5Dwrite, H5Dread
- Collectiveness is indicated by function parameters, not by function names as in MPI

What does PHDF5 support ?

- After a file is opened by all 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 may write to the same data array (i.e. collective I/O).
  - Each process may write to individual data array (i.e. independent I/O).
- API languages
  - C and F90, 2003 language interfaces
  - Most platforms with MPI-IO supported
- Programming model: HDF5 uses access property list to control the file access mechanism.
  General model to access HDF5 file in parallel:
  1. Set up MPI-IO file access property list
  2. Open File
  3. Access Data
  4. Close File

Creating a File

- The programming model for creating and accessing a file is as follows:
  - Set up an access template object to control the file access mechanism.
  - Open the file.
  - Close the file.

```
MPI_Comm comm  = MPI_COMM_WORLD;
MPI_Info info  = MPI_INFO_NULL;

/* Initialize MPI */
MPI_Init(&argc, &argv);
MPI_Comm_size(comm, &mpi_size);
MPI_Comm_rank(comm, &mpi_rank);

/* Set up file access property list
with parallel I/O access */
plist_id = H5Pcreate(H5P_FILE_ACCESS);
H5Pset_fapl_mpio(plist_id, comm, info);
/* Create a new file collectively.*/
file_id = H5Fcreate(H5FILE_NAME, &
H5F_ACC_TRUNC, H5P_DEFAULT, plist_id);
/*Close property list and file
(not shown)*/
```

```
comm = MPI_COMM_WORLD
info = MPI_INFO_NULL

CALL MPI_INIT(mpierror)
CALL MPI_COMM_SIZE(comm, mpi_size, &
                   mpierror)
CALL MPI_COMM_RANK(comm, mpi_rank, &
                   mpierror)
! Initialize FORTRAN interface
CALL h5open_f(error)
! Setup file access property list
! with parallel I/O access.
CALL h5pcreate_f(H5P_FILE_ACCESS_F, &
                 plist_id, error)
CALL h5pset_fapl_mpio_f(plist_id, &
                   comm, info, error)
! Create the file collectively.
CALL h5fcreate_f(filename,H5F_ACC_TRUNC_F,&
file_id, error, access_prp = plist_id)
!Close property list and file (not shown)
```

Creating a Dataset (1)

The programming model for accessing a dataset with Parallel HDF5 is:

- Create or open a Parallel HDF5 file with a collective call to:

```
H5Dcreate (C) / h5dcreate_f (F90)
H5Dopen (C) / h5dopen_f (F90)
```

- Obtain a copy of the file transfer property list and set it to use collective or independent I/O. Do this by first passing a data transfer property list class type to:

```
H5Pcreate (C) / h5pcreate_f (F90)
```

- Then set the data transfer mode to either use independent I/O access or to use collective I/O, with a call to:

```
H5Pset_dxpl_mpio (C) / h5pset_dxpl_mpio_f (F90)
```

Creating a Dataset(2)

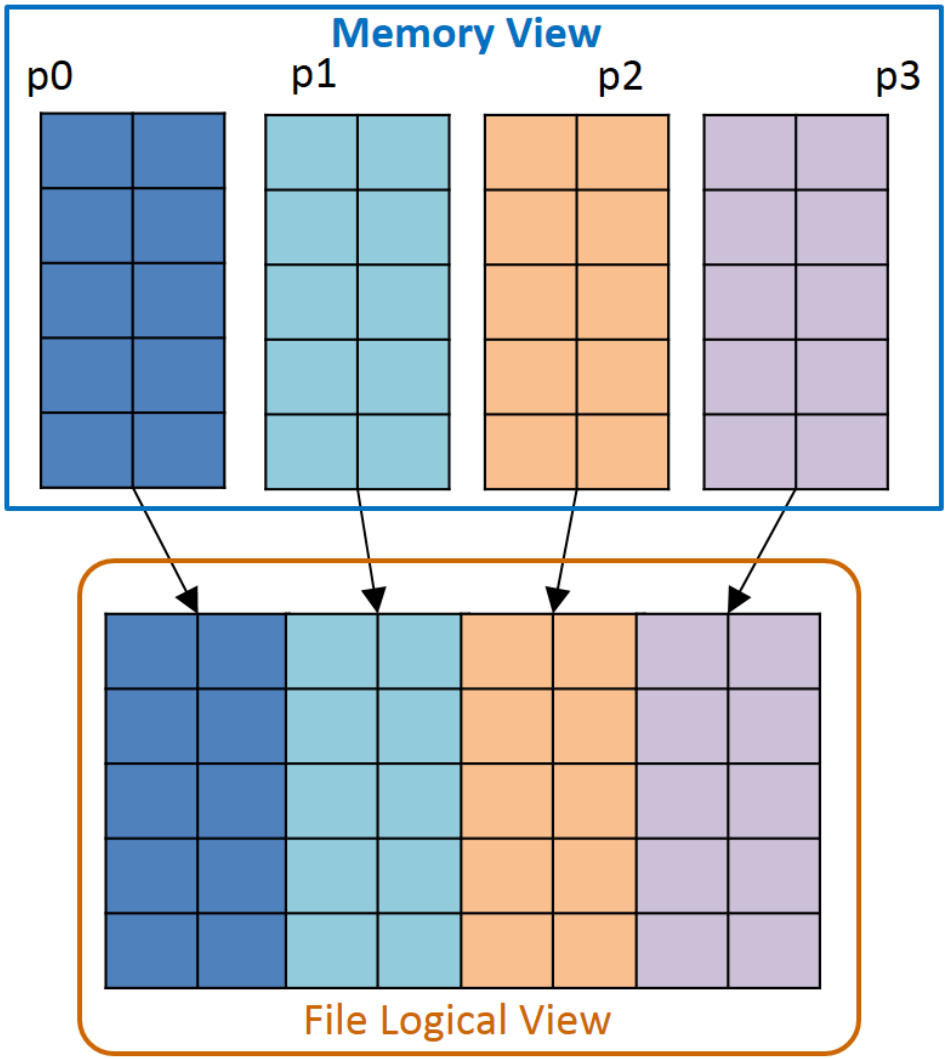
- Access the dataset with the defined transfer property list.
  - Each process may do an independent and arbitrary number of data I/O access calls, using:

```
H5Dwrite (C) / h5dwrite_f (F90)
H5Dread (C) / h5dread_f (F90)
```

```
/* Create the dataset with default
properties and close filespace.*/
dset_id = H5Dcreate(file_id, DATASETNAME,
H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
/* Create property list for
collective dataset write. */

plist_id = H5Pcreate(H5P_DATASET_XFER);

H5Pset_dxpl_mpio(plist_id,
H5FD_MPIO_COLLECTIVE);

/* Write the dataset collectively.*/
status = H5Dwrite(dset_id,
H5T_NATIVE_INT,
memspace, filespace,plist_id, data);
```

```
! Create the dataset with default
! properties.
CALL h5dcreate_f(file_id, dsetname, &
H5T_NATIVE_INTEGER, filespace, &
dset_id, error)
! Create property list for collective
! dataset write
CALL h5pcreate_f(H5P_DATASET_XFER_F, &
plist_id, error)
CALL h5pset_dxpl_mpio_f(plist_id, &
H5FD_MPIO_COLLECTIVE_F, error)
! Write the dataset collectively.
 CALL h5dwrite_f(dset_id, &
 H5T_NATIVE_INTEGER, data, dimsfi, &
 error, file_space_id = filespace, &
 mem_space_id = memspace, &
 xfer_prp = plist_id)
```

Parallel HDF5 Example (1)



- An 5 x 8 array dataset

```
dimsf = (/5,8/)
```

- After each process will be write a subset:

```
5 x (dimsf(2) / mpi_size)
```

- Note that the absolute index of arguments for hyperslab creation is zero-based:

```
offset = (/ 0, mpi_rank * count(2) /)
count = (/ 5, dimsf(2)/ mpi_size /)
stride = (/ 1, 1 /)
```

Parallel HDF5 Example (2)

```
CALL h5pcreate_f(H5P_FILE_ACCESS_F, plist_id, error)
CALL h5pset_fapl_mpio_f(plist_id, comm, info, error)

CALL h5fcreate_f(filename, H5F_ACC_TRUNC_F, &
file_id, error, access_prp = plist_id)
CALL h5pclose_f(plist_id, error)

CALL h5screate_simple_f(rank, dimsf, filespace, error)

CALL h5dcreate_f(file_id, dsetname, &
H5T_NATIVE_INTEGER, filespace, dset_id, error)
CALL h5sclose_f(filespace, error)

count(1) = dimsf(1)
count(2) = dimsf(2)/mpi_size
offset(1) = 0
offset(2) = mpi_rank * count(2)
CALL h5screate_simple_f(rank, count, memspace, error)

CALL h5dget_space_f(dset_id, filespace, error)
CALL h5sselect_hyperslab_f(filespace, &
H5S_SELECT_SET_F,& offset, count, error)
```

Setup file access property list with parallel I/O access

Create the file collectively

Create the data space for the dataset

Create the dataset with default properties

Each process defines dataset in memory and writes it to the hyperslab in the file.

Select hyperslab in the file

Parallel HDF5 Example (3)

```
ALLOCATE ( data(count(1),count(2)))
data = mpi_rank + 10

CALL h5pcreate_f(H5P_DATASET_XFER_F, plist_id, &
error)
CALL h5pset_dxpl_mpio_f(plist_id, &
H5FD_MPIO_COLLECTIVE_F, error)

 CALL h5dwrite_f(dset_id, H5T_NATIVE_INTEGER, data,&
 dimsfi, error, file_space_id = filespace, &
 mem_space_id = memspace, xfer_prp = plist_id)

DEALLOCATE(data)

CALL h5sclose_f(filespace, error)
CALL h5sclose_f(memspace, error)

CALL h5dclose_f(dset_id, error)
CALL h5pclose_f(plist_id, error)

CALL h5fclose_f(file_id, error)
```

Initialize data buffer with trivial data

Create property list for collective dataset write

Write the dataset collectively

Deallocate data buffer

Close dataspaces

Close the dataset and property list

Close the file

Summary

Network Common Data Form

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

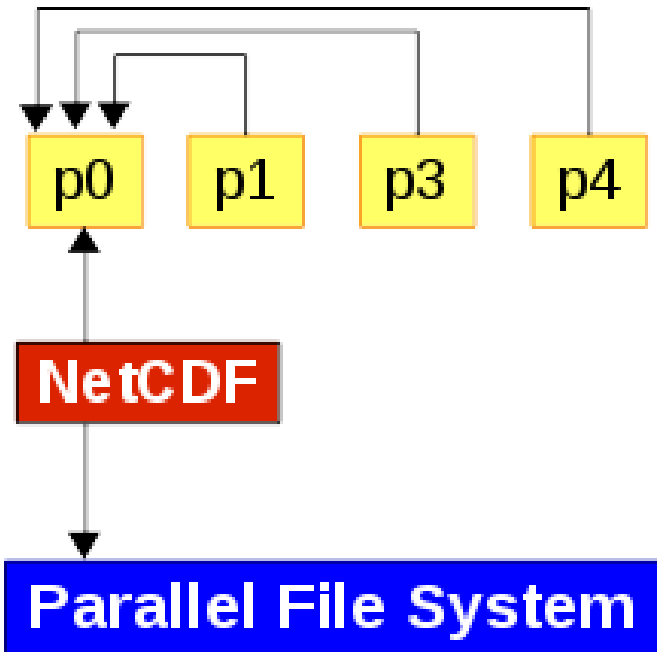
- First released in 1989.
- NetCDF-4.0 (June, 2008) introduces many new features, while maintaining full code and data compatibility.

Three conceptual components

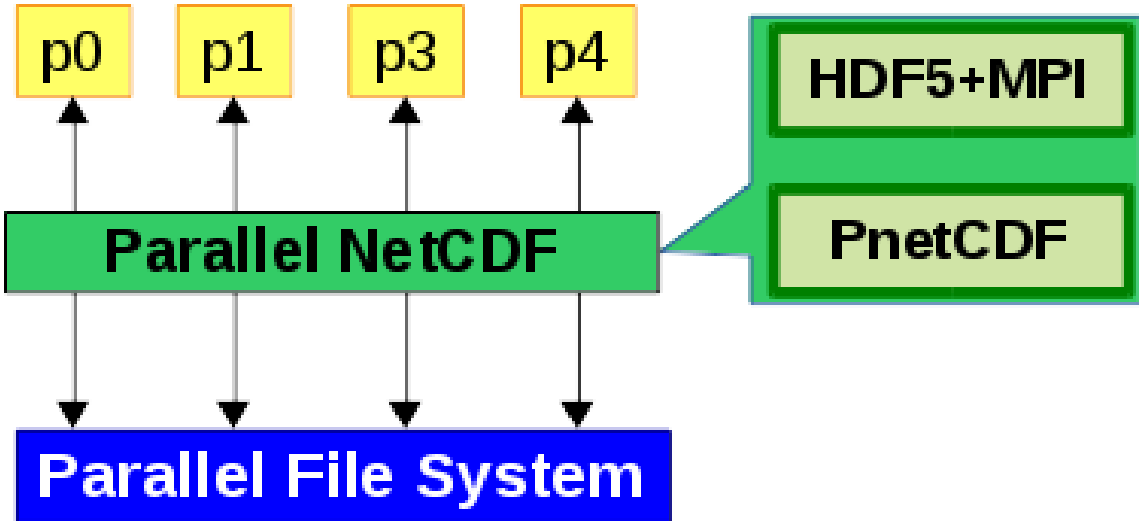
- data model
- file format (self-describing $\rightarrow$ metadata)
- API/libraries (implementations)

Original area of deployment: earth sciences.
Available from http://www.unidata.ucar.edu/software/netcdf/
Unidata $\rightarrow$ data services for earth system sciences

Parallel I/O with NetCDF (1)



Serial I/O



Parallel I/O

Parallel I/O with NetCDF (2)

- NetCDF support is based on MPI-IO+pnetcdf or MPI-IO+HDF5
  - requires to be built into the libraries via a configuration option
  - establishes dependency on MPI implementation
  - PnetCDF library has a more elaborate interface (http://cucis.ece.northwestern.edu/projects/PnetCDF/index.html)
- Initialization:
  - nf90_create() and nf90_open() have two additional optional arguments: an MPI communicator comm, and an MPI_Info object info (may be MPI_INFO_NULL)
- Switching between collective and independent access:

```
ierr =  nf90_var_par_access(ncid, varid, access)
```

  - access may be NF90_INDEPENDENT or NF90_COLLECTIVE
  - applies for writes of that variable while the file is open
  - default: independent access

NetCDF and HDF5 Data Models

- The netCDF classic data model: simple and flat
  - Dimensions
  - Variables
  - Attributes
- The netCDF enhanced data model added
  - More primitive types
  - Multiple unlimited dimensions
  - Hierarchical groups
  - User-defined data types
- The HDF5 data model has even more features
  - Non-hierarchical groups
  - User-defined primitive data types
  - References (pointers to objects and data regions in a file)
  - Attributes attached to user-defined types

Assessment

- HDF5 has
  - a more complex structure
  - is therefore more powerful and flexible

  than NetCDF
- This also may have disadvantages: more complex and possibly error-prone to program to (difficult call sequence)
- Simplification: HDF5 “lite“ high level interface – H5LT makes usage easier by providing a way to aggregate several API calls
- Image processing: H5IM provides a standard storage scheme for data which can be interpreted as images – e.g. 2-dimensional raster data
- Note: from version 1.6 to 1.8, the API has undergone evolution. HDF5-1.10 contains several important new features.