Structured File Formats: NetCDF and HDF

Adam Carter
EPCC, The University of Edinburgh

This lecture material

For full acknowledgements and more details about re-use please see the final slide of this presentation.

Outline

- An **introduction** to NetCDF and HDF5 as **examples** of structured file formats suitable for HPC applications
- Overview of both formats
- HDF5 (in more detail)
- NetCDF (in less detail)

I/O

- I/O essential for all applications/codes
  - Some data must be read in or produced
  - Instructions and Data
- Basic hierarchy
  - CPU – Cache – Memory – Devices (including I/O)
- Often “forgotten” for HPC systems
  - Linpack not I/O bound
  - Not based on CPU clock speed or memory size
- Often “forgotten” in program
  - Start and end so un-important
  - Just assumed overhead

I/O

- Small parallel programs (i.e. under 1000 processors)
  - Cope with I/O overhead
- Large parallel programs (i.e. tens of thousand processors)
  - Can completely dominate performance
  - Exacerbate by poor functionality/performance of I/O systems
- Any opportunity for program optimisation important
  - Improve performance without changing program

Parallel I/O

- …is covered elsewhere in this course
- …is not easy to do well, but:

- You don’t have to choose performance over usability!
- Using I/O libraries like NetCDF and HDF5 can get you advantages of structured data and you will also benefit from work that others have put in to optimise I/O performance and scaling.

NetCDF

- **Net**work **C**ommon **D**ata **F**ormat
  - Data model
  - File format
  - Application programming interface (API)
  - Library implementing the API

- NetCDF
  - Created in the US by unidata for earth science and geoscience data, supported by the NSF

- NetCDF
  - Software library and self-describing data format
  - Portable, machine independent data
  - Can use HDF5 or NetCDF format (HDF5 gives larger files and unlimited array dimensions) in NetCDF 4.0 (latest version)

What is HDF5?

Hierarchical Data Format (version 5)

From www.hdfgroup.org:

HDF5 is a unique technology suite that makes possible the management of extremely large and complex data collections.

What is HDF5?

- A versatile **data model** that can represent very complex data objects and a wide variety of metadata.
- A completely **portable file format** with no limit on the number or size of data objects in the collection.
- A **software library that** runs on a range of computational platforms, from laptops to massively parallel systems, and implements a high-level API with C, C++, Fortran 90, and Java interfaces.
- A rich set of integrated **performance** features that allow for access time and storage space optimizations.
- Tools and applications for managing, manipulating, viewing, and analyzing the data in the collection.

Source: www.hdf5group.org (emphasis mine)

Data Model

- In very basic terms, HDF is like a directory and file *hierarchy* in a file
- The data model is based on *groups* and *datasets*
    - can think of groups like directories/folders, datasets like files
    - both can have (user-defined) *attributes*

Portable File Format

- HDF5 files are binary but portable
  - HDF5 model take care of types, endianness etc.

Why HDF5?

- Structure
- Portability

HDF5 files are Self-Describing

Tool Support

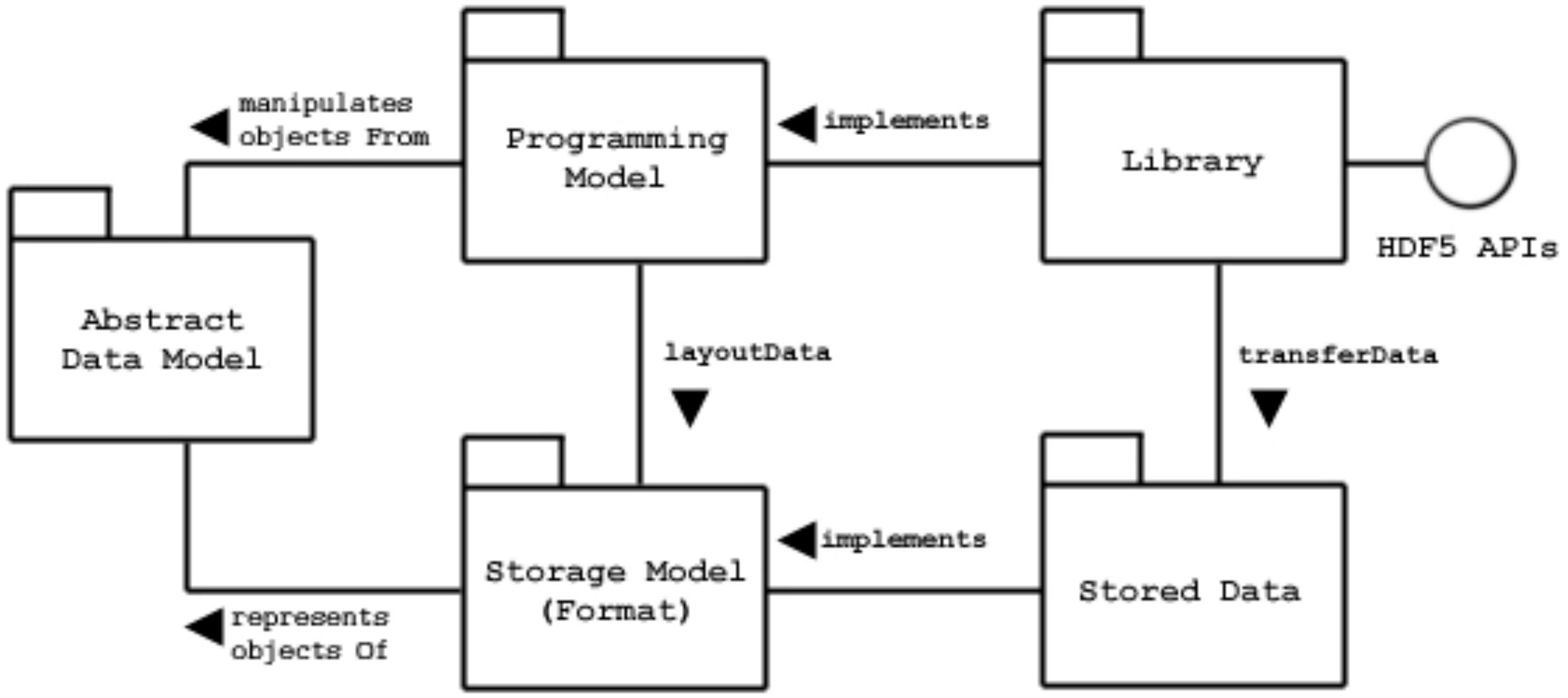
- Performance

Pre-Optimised

Parallel-Ready

- Free & Open Source!

HDF5 Data Model and File Structure



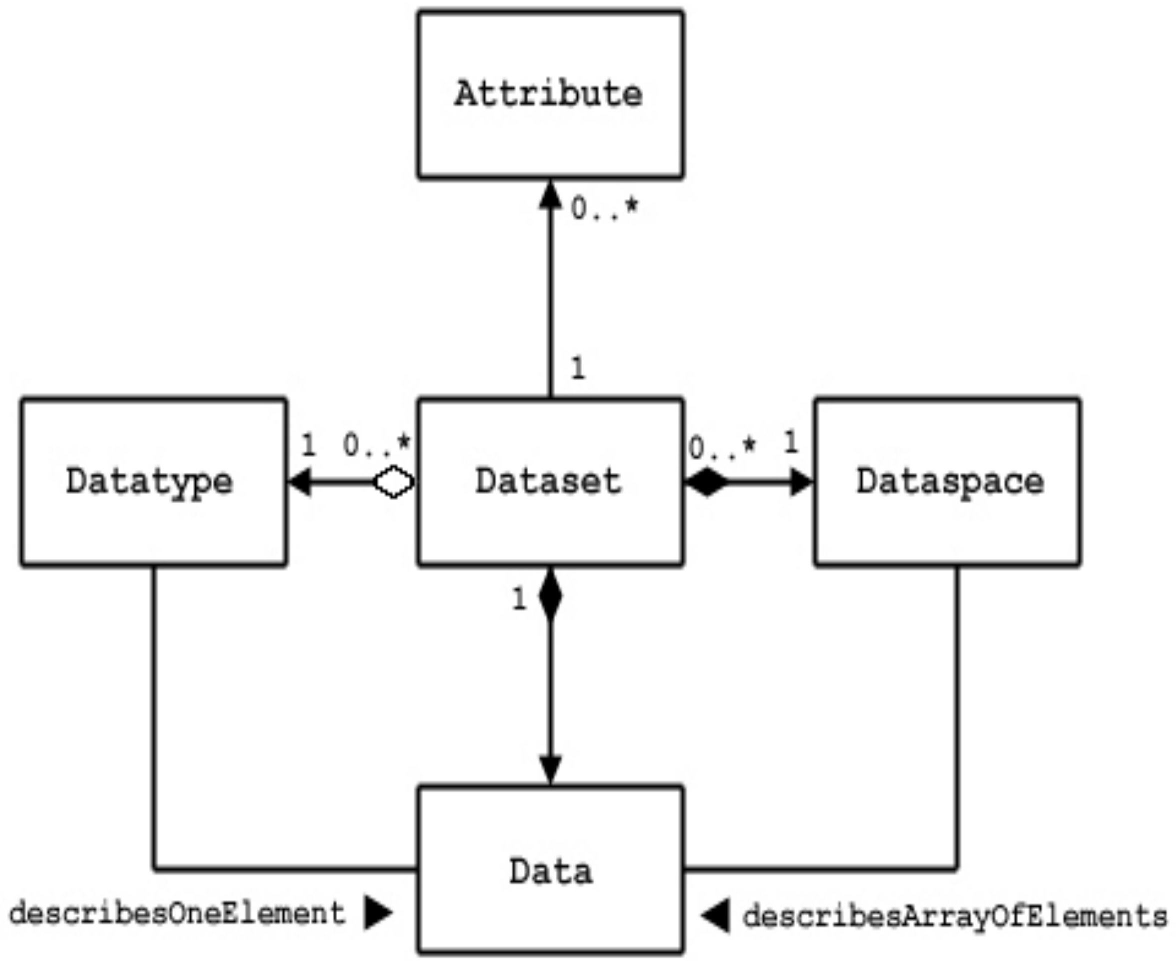
Source: http://www.hdfgroup.org/HDF5/doc/UG/Images/Dmodel_fig1.JPG

HDF5 Groups

- *HDF5 group:* "a grouping structure containing instances of zero or more groups or datasets, together with supporting metadata."
- A group has two parts:
  - A *group header* containing name & attributes
  - A group *symbol table* listing the group's contents
- Like UNIX directories, you can identify an object with a path name:
  - **/** the root group
  - **/foo** a member of the root group, called foo
  - **/foo/zoo** a member of the foo group, called zoo

HDF5 Datasets

- A dataset has two parts:
  - A *header*
  - A *data array*
- Header provides information on:
  - Name
    - The dataset name. A sequence of alphanumeric characters.
  - Datatypes
    - Atomic, Compound, NATIVE, Named
  - Dataspace
    - Describes the dimensionality (including *unlimited* option)
  - Storage Layout
    - Contiguous, compact, chunked



Source: https://www.hdfgroup.org/HDF5/doc1.6/UG/UG_frame.html

HDF5 Attributes

- *Attributes* are small named datasets that are attached to primary datasets, groups, or named datatypes
- Name, value pairs
  - Value can have multiple entries of the same datatype
- There's a separate API for attribute read/write
- Excessively large attribute sets will impact performance

The HDF5 API

- **H5F**: **F**ile-level access routines
  - e.g. H5Fopen
- **H5G**: **G**roup functions, for creating and operating on groups of objects
  - e.g. H5Gset
- **H5T:** Data**T**ype functions, for creating and operating on simple and compound datatypes to be used as the elements in data arrays
- **H5S:** Data**S**pace functions, which create and manipulate the dataspace in which the elements of a data array are stored

- **H5D: D**ataset functions, which manipulate the data within datasets and determine how the data is to be stored in the file.
- **H5P**: **P**roperty list functions, for manipulating object creation and access properties.
- **H5A**: **A**ttribute access and manipulating routines.
- **H5Z**: **C**ompression registration routine.
- **H5E**: **E**rror handling routines.
- **H5R**: **R**eference routines.
- **H5I**: **I**dentifier routine.

Using HDF5 on ARCHER / RDF-DAC

- In your program you'll need:

```
#include <hdf5.h>
```

- On ARCHER you should

```
module load cray-hdf5-parallel
```

- On RDF-DAC you don't need to load anything to use HDF5
- To use HDF5 from Python on either ARCHER or RDF-DAC, you should

```
module load anaconda
```

Example: Create and Close a File

A type defined in HDF5. An HDF ID, used to keep track of objects like files

```
hid_t       file;                          /* identifier */
/*
* Create a new file using H5ACC_TRUNC access,
* default file creation properties, and default file
* access properties.
* Then close the file.
*/
file = H5Fcreate(FILE, H5ACC_TRUNC, H5P_DEFAULT, H5P_DEFAULT);
status = H5Fclose(file);
```

Allows an existing file (if present) to be overwritten

Example: Creating a dataset so that data can be written to it

```
hid_t    dataset, datatype, dataspace;   /* declare identifiers */
/* Create dataspace: Describe the size of the array and
 * create the data space for fixed size dataset.
 */
dimsf[0] = NX;
dimsf[1] = NY;
dataspace = H5Screate_simple(RANK, dimsf, NULL);
/* Define datatype for the data in the file.
 * We will store little endian integer numbers.
 */
datatype = H5Tcopy(H5T_NATIVE_INT);
status = H5Tset_order(datatype, H5T_ORDER_LE);
/* Create a new dataset within the file using defined
 * dataspace and datatype and default dataset creation
 * properties.
*/
dataset = H5Dcreate(file, DATASETNAME, datatype, dataspace, H5P_DEFAULT);
```

Example: Write data to a file

```
/*
* Write the data to the dataset using default transfer
* properties.
*/
status = H5Dwrite(dataset, H5T_NATIVE_INT, H5S_ALL,
        H5S_ALL, H5P_DEFAULT, data);
```

Here's the data itself, stored as a standard array of ints in C.

Example: Read data from a file

```
/*
* Write the data to the dataset using default transfer
* properties.
*/
status = H5Dread(dataset, H5T_NATIVE_INT, H5S_ALL,
      H5S_ALL, H5P_DEFAULT, data);
```

- Exactly analogous to write!

Hyperslabs

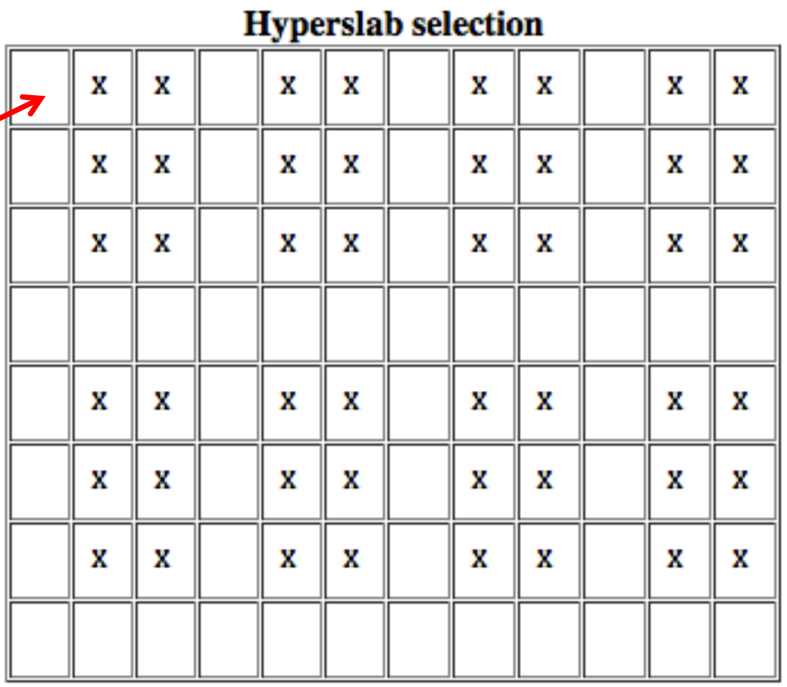
- Hyperslabs are portions of datasets

start = (0,1)

stride = (4,3)

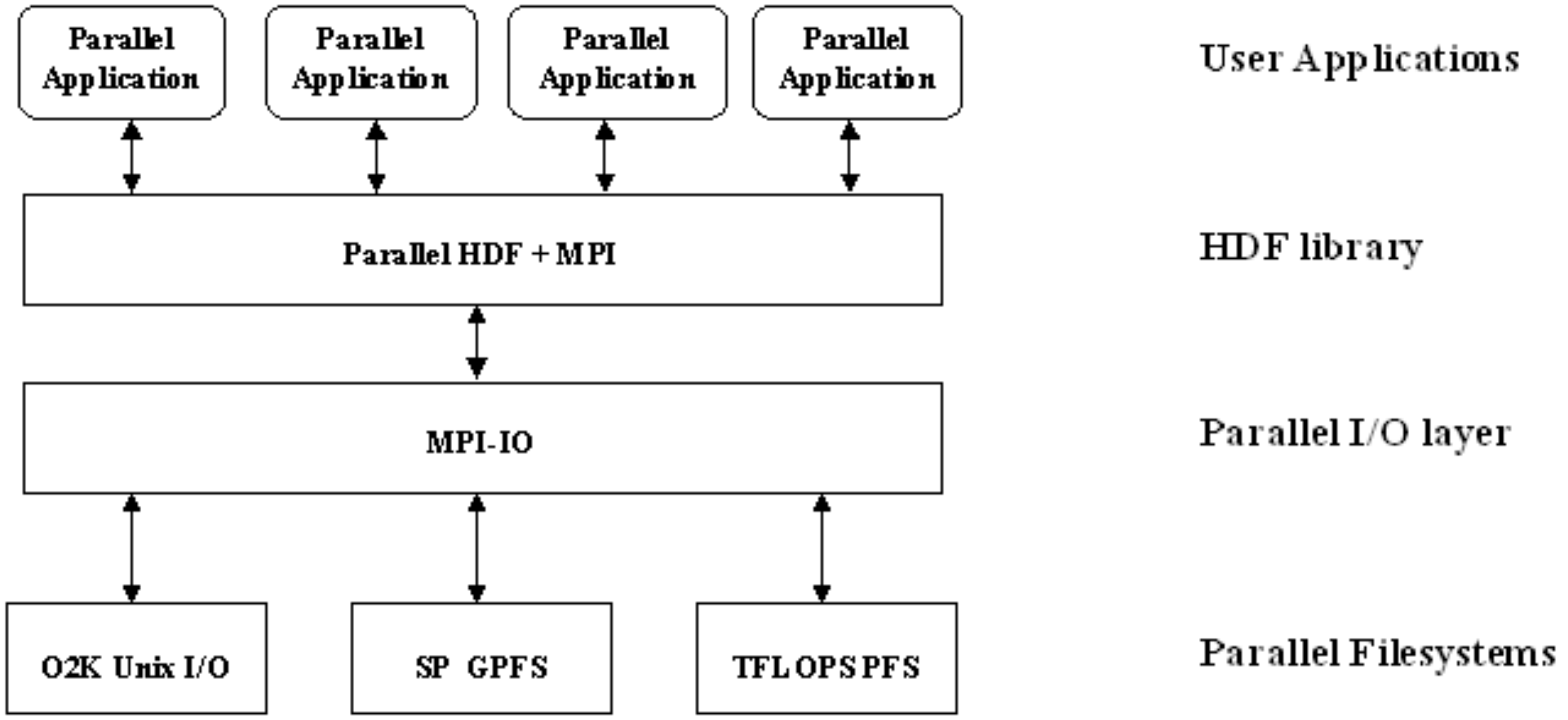
count = (2,4)

block = (3,2)



Parallel HDF5

- Designed to work with MPI and MPI-IO
- Parallel HDF5 files are compatible with serial HDF5 files and sharable between different serial and parallel platforms
- Parallel HDF5 had to be designed to have 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 had to be portable to different platforms.



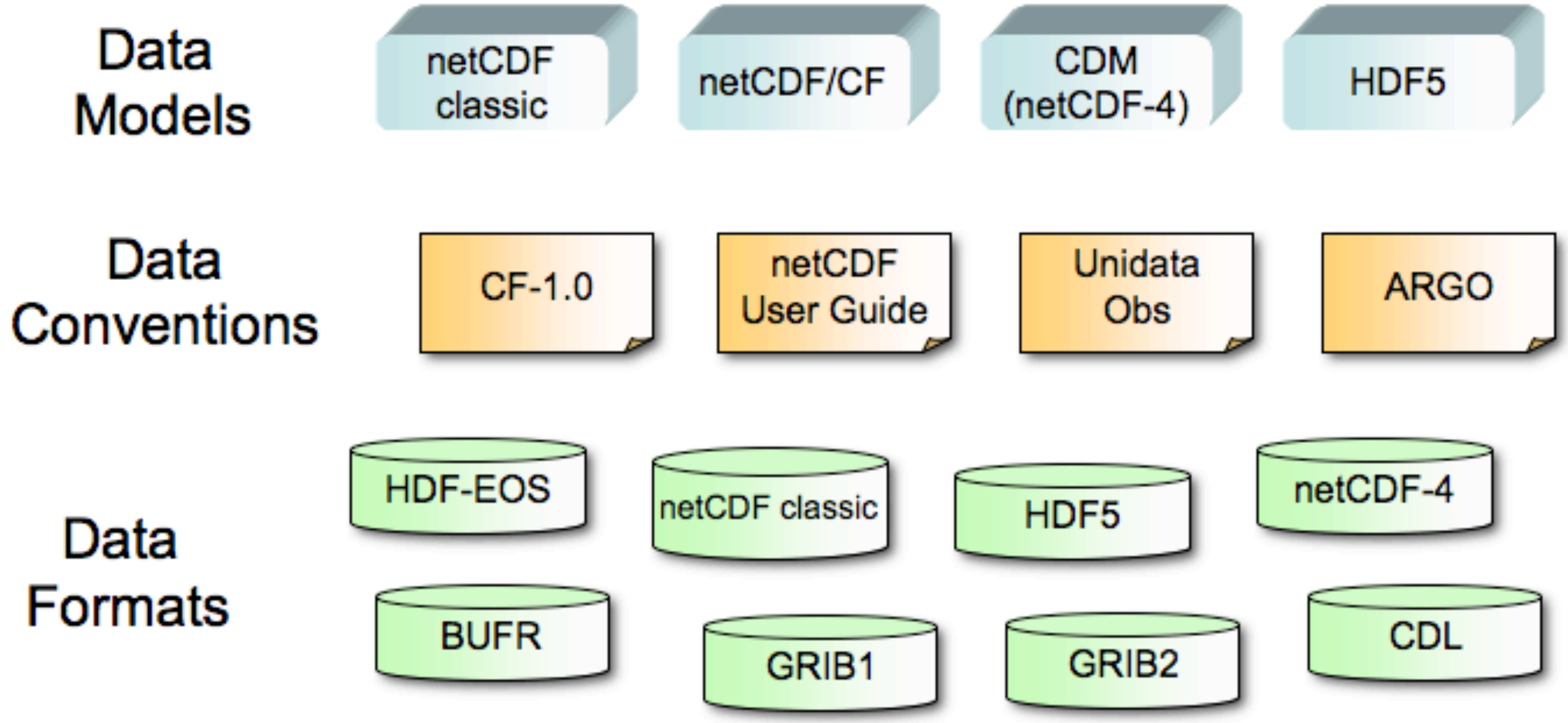
Source: http://www.hdfgroup.org/HDF5/Tutor/poverview.html

NetCDF

- The netCDF niche is array-oriented scientific data.
  - Uses portable files as unit of self-describing data (unlike databases)
  - Emphasizes efficient direct access to data within files (unlike XML)
  - Provides a multidimensional array abstraction for scientific applications (unlike databases and XML)
  - Avoids dependencies on external tables and registries (unlike GRIB and BUFR)
  - Emphasizes simplicity over power (unlike HDF5)
  - Has built-in client support for network access to structured data from servers
  - Has a large enough community of users to foster development of:
    - support in many third-party applications
    - third-party APIs for other programming and scripting languages
    - community conventions, such as Climate and Forecast (CF) metadata conventions

NetCDF

- NetCDF has changed over time, so it includes the following:
  - Two data models
    - classic model (netCDF-1, netCDF-2, netCDF-3)
    - enhanced model (netCDF-4)
  - Two formats with variants
    - classic format and 64-bit offset variant for large files
    - netCDF-4 (HDF5-based) format and classic model variant
  - Two independent flavors of APIs
    - C-based interfaces (C, C++, Fortran-77, Fortran-90, Perl, Python, Ruby, Matlab, ...)
    - Java interface
- However, newer versions support:
  - all previous netCDF data models
  - all previous netCDF formats and their variants
  - all previous APIs
  - Files written through one language API are readable through other language APIs.


Data Models
netCDF classic
netCDF/CF
CDM (netCDF-4)
HDF5
Data Conventions
CF-1.0
netCDF User Guide
Unidata Obs
ARGO
Data Formats
HDF-EOS
netCDF classic
HDF5
netCDF-4
BUFR
GRIB1
GRIB2
CDL

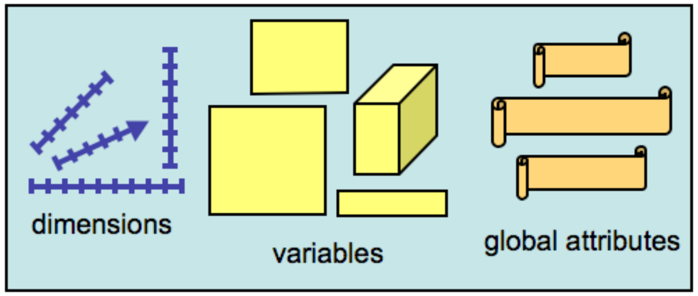
NetCDF

- Common data model
  - Variables: N-dimensional arrays of char, byte, short, int, float, double
  - Dimensions: Name and length
  - Attributes: Annotations and other metadata
  - Groups: Hierarchical, similar to directories
  - User-defined types

- Parallel functionality
  - Parallel HDF5
    - NetCDF4 version
  - Parallel NetCDF (PNetCDF)
    - Separate library, can write NetCDF 3 (and below) flies in parallel
    - Later versions of NetCDF 4 also include some PNetCDF functions

NetCDF file

- NetCDF files are containers for Dimensions, Variables, and Global Attributes
- File (dataset) contains the following:
  - path name
  - dimensions*
  - variables*
  - global (file-level) attribute*
  - data values associated with the variables.*
  - (*optional)
- enhanced data model can contain multiple groups
  - group -> dataset
  - groups can be nested



NetCDF file

```
netcdf pres_temp_4D {
  dimensions:
                  level = 2 ;
                  latitude = 6 ;
                  longitude = 12 ;
                  time = UNLIMITED ;
  variables:
                  float latitude(latitude);
                           latitude:units = "degrees_north" ;
                  float longitude(longitude) ;
                           longitude:units = "degrees_east" ;
                  float pressure(time, level, latitude, longitude) ;
                           pressure:units = "hPa" ;
                  float temperature(time, level, latitude, longitude) ;
                                               temperature:units = "celsius" ;
  data:
                  latitude = 25, 30, 35, 40, 45, 50 ;
                  longitude = -125, -120, … ;
                  pressure = 900, 901, 902, … ;
                  temperature = 9, 10, 11, …;
}
```

NetCDF dimensions

- Specify variable shapes, common grids, and co-ordinate systems
  - Has a name and length
  - can be used by multiple variables
  - can associated with *coordinate variables* to identify coordinate axes.
- classic netCDF
  - at most one dimension can have the *unlimited* length (record dimension)
- enhanced netCDF
  - multiple dimensions can have the unlimited length.

Variables

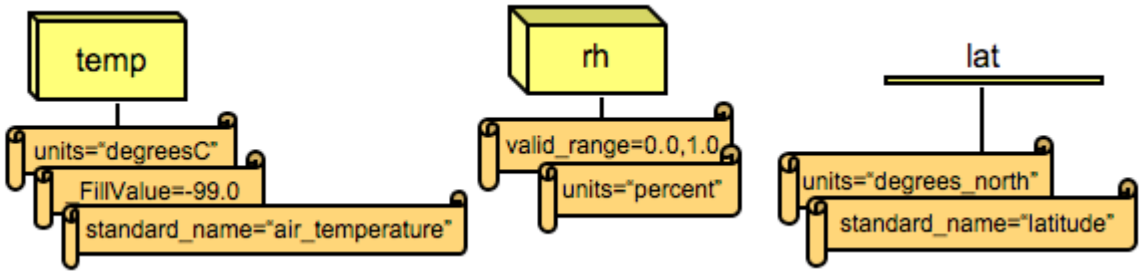
- Variables define the things that hold data:
  - Has a name, type, shape, can have attributes, and values.
  - Type:
    - Classic NetCDF type is the *external type* of its data as represented on disk, i.e.
      - char
      - byte (8 bits)
      - short (16 bits)
      - int (32 bits)
      - float (32 bits)
      - double (64 bits)
    - Enhanced NetCDF
      - Adds unsigned type; ubyte, ushort, uint, uint64
      - Adds int64 (64 bits), string (variable-length string of characters)
      - User defined types
  - Shape:
    - list of dimensions.
    - no dimensions: a *scalar variable* with only one value
    - 1 dimension: a 1-D (vector) variable
    - 2 dimensions: a 2-D (matrix or grid) variable
  - Attribute:
    - specify properties, i.e. units

Attributes

- Metadata about variables or datasets
- Attribute has:
  - Name
  - Type (same as variable types)
  - Values
- Can have scalar or 1-D values
- Cannot be nested

When to use attributes

- intended for metadata
- for single values, strings, or small 1-D arrays
- atomic access, must be written or read all at once
- values typically don't change after creation
- length specified when created
- attributes are read when file is opened



Co-ordinate variables

- Variable with same name as a dimension
  - By convention these specify physical co-ordinate (i.e. lat, lon, level, time, etc…) associated with that dimension
  - Not special in NetCDF, but often interpreted by programs that use NetCDF as special.
  - Allows indexing through position on dimension and matching to co-ordinates

CDL (Common Data Language)

- Human readable notation for NetCDF datasets and data
  - Obtain from NetCDF file using the `ncdump` program

```
netcdf example {   // example of CDL notation
 dimensions:
         x = 3 ;
         y = 8 ;
 variables:
         float rh(x, y) ;
                 rh:units = "percent" ;
                 rh:long_name = "relative humidity" ;
 // global attributes
         :title = "simple example, lacks some conventions" ;
 data:
  rh =
   2, 3, 5, 7, 11, 13, 17, 19,
   23, 29, 31, 37, 41, 43, 47, 53,
   59, 61, 67, 71, 73, 79, 83, 89 ;
 }
```

NetCDF utilities

- `ncdump`
  - Produce CDL version of NetCDF file
  - Dump everything, or subset, or just metadata, show indices in C or FORTRAN order, etc…
- `ncgen`
  - Generate NetCDF file from CDL version
  - Generate C, FORTRAN, or Java program which would produce the NetCDF file
- `ncdump` and `ncgen` let you edit NetCDF files manually, or create the program structure that will read/write a NetCDF file in the format you desire automatically
- `nccopy`
  - Copy NetCDF file to new file
  - Can compress and change file format (i.e. classic to enhanced)
- `nc-config`
  - Generate flags necessary to link a program with NetCDF, i.e.:

```
cc `nc-config --cflags` myapp.c -o myapp `nc-config --libs`
f95 `nc-config --fflags` yrapp.f -o yrapp `nc-config --flibs`
```

NetCDF programming interfaces

- NetCDF APIs
  - C, FORTRAN 77, FORTRAN 90, C++, Perl, Java, Python, Ruby, NCL, Matlab, Objective C, Ada, R
  - Some of these are third party

- C interface is used as the core of all but the Java interface

C API example

```
#include <netcdf.h>
…
  int ncid, x_dimid, y_dimid, varid;
  int dimids[NDIMS];
  int data_out[NX][NY];
  …
  if ((retval = nc_create(FILE_NAME, NC_CLOBBER, &ncid))){
      printf("Error: %s\n", nc_strerror(retval));
      exit(1);
  }
  nc_def_dim(ncid, "x", NX, &x_dimid);
  nc_def_dim(ncid, "y", NY, &y_dimid);
  dimids[0] = x_dimid;
  dimids[1] = y_dimid;
  nc_def_var(ncid, "data", NC_INT, NDIMS,dimids, &varid);
  nc_enddef(ncid));
  nc_put_var_int(ncid, varid, &data_out[0][0]);
  nc_close(ncid)
```

F90 API example

```
use netcdf
  integer :: ncid, varid, dimids(NDIMS)
  integer :: x_dimid, y_dimid
  call check( nf90_create(FILE_NAME, NF90_CLOBBER, ncid) )
  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 /)
  call check( nf90_def_var(ncid, "data", NF90_INT, dimids, varid) )
  call check( nf90_enddef(ncid) )
  call check( nf90_put_var(ncid, varid, data_out) )
  call check( nf90_close(ncid) )

contains
  subroutine check(status)
    integer, intent ( in) :: status

    if(status /= nf90_noerr) then
      print *, trim(nf90_strerror(status))
      stop "Stopped"
    end if
  end subroutine check
```

Java API example

```
import ucar.ma2.*;
import ucar.nc2.*;
    NetcdfFileWriter dataFile = null;
    try {
      dataFile = NetcdfFileWriter.createNew(NetcdfFileWriter.Version.netcdf3, 
filename);
      Dimension xDim = dataFile.addDimension(null, "x", NX);
      Dimension yDim = dataFile.addDimension(null, "y", NY);
      List<Dimension> dims = new ArrayList<>();
      dims.add(xDim);
      dims.add(yDim);
      Variable dataVariable = dataFile.addVariable(null, "data", DataType.INT, 
dims);
      dataFile.create();
      dataFile.write(dataVariable, dataOut);
    } catch (IOException e) {
      e.printStackTrace();
    } catch (InvalidRangeException e) {
      e.printStackTrace();
    } finally {
      if (null != dataFile)
        try {
          dataFile.close();
        } catch (IOException ioe) {
          ioe.printStackTrace();
        }
    }
```

Python API example

```
#from netCDF4_classic import Dataset
#from numpy import arange, dtype
nx = 6; ny = 12
ncfile = Dataset('simple_xy.nc','w')
data_out = arange(nx*ny) # 1d array
data_out.shape = (nx,ny) # reshape to 2d array.
ncfile.createDimension('x',nx)
ncfile.createDimension('y',ny)
data =
ncfile.createVariable('data',dtype('int32').char,('x','
y'))
data[:] = data_out
ncfile.close()
print '*** SUCCESS writing example file simple_xy.nc!'
```

C++ API example

```
#include <netcdf>
using namespace netCDF;
using namespace netCDF::exceptions;
  try
    {
      NcFile dataFile("simple_xy.nc", NcFile::replace);
      NcDim xDim = dataFile.addDim("x", NX);
      NcDim yDim = dataFile.addDim("y", NY);
      vector<NcDim> dims;
      dims.push_back(xDim);
      dims.push_back(yDim);
      NcVar data = dataFile.addVar("data", ncInt, dims);
      data.putVar(dataOut);
      return 0;
    }
  catch(NcException& e)
    {e.what();
      return NC_ERR;
    }
}
```

High-performance NetCDF

- Enhanced NetCDF (version 4 and beyond)
  - Built on HDF5 for parallel/high performance I/O
  - Files need to be stored in HDF5 format

```
#include "netcdf.h"
#include "hdf5.h"
       MPI_Comm comm = MPI_COMM_WORLD;
       MPI_Info info = MPI_INFO_NULL;
       int ncid, v1id, dimids[NDIMS];
       size_t start[NDIMS], count[NDIMS];
       res = nc_create_par(FILE, NC_NETCDF4|NC_MPIIO, comm, info, &ncid)l
       res =nc_def_dim(ncid, "d1", DIMSIZE, dimids);
       res = nc_def_dim(ncid, "d2", DIMSIZE, &dimids[1];
       res = nc_def_var(ncid, "v1", NC_INT, NDIMS, dimids, &v1id);
       res = nc_enddef(ncid);
       start[0] = mpi_rank * DIMSIZE/mpi_size;
       start[1] = 0;
       count[0] = DIMSIZE/mpi_size;
       count[1] = DIMSIZE;
       res =nc_var_par_access(ncid, v1id, NC_INDEPENDENT);
       res = nc_put_vara_int(ncid, v1id, start, count, &data[mpi_rank*QTR_DATA]);
       res = nc_close(ncid);
       MPI_Finalize();
```

Parallel NetCDF

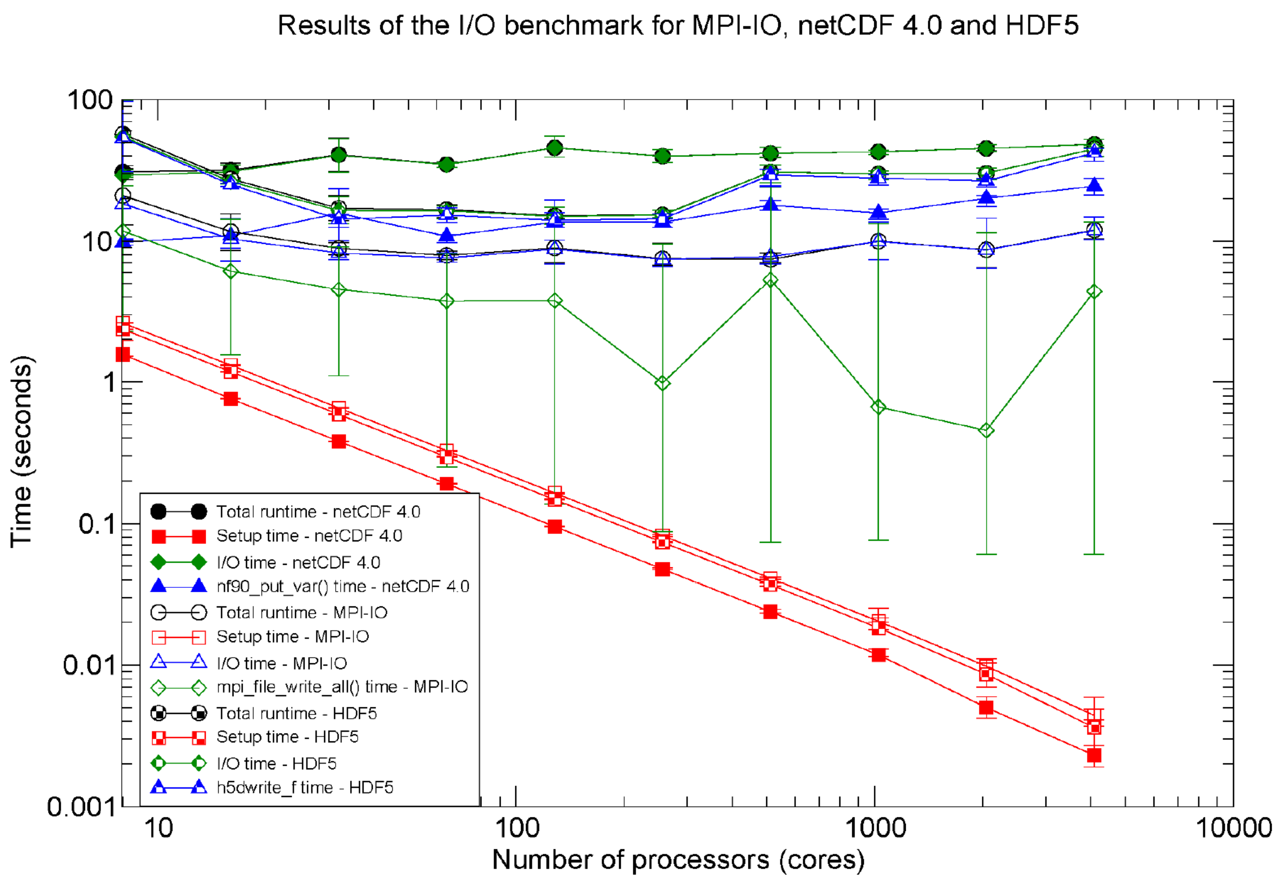
- PNetCDF
  - Separate library, add parallel I/O library to support parallel I/O in NetCDF (CDF-1 and CDF-2)
  - Also supports extended CDF-2 (CDF-5)
  - Only functionality that will support parallel I/O for NetCDF format files
  - Some PnetCDF functionality also integrated into later versions of NetCDF 4

```
ret = ncmpi_create(MPI_COMM_WORLD, argv[1],
                   NC_CLOBBER|NC_64BIT_OFFSET,MPI_INFO_NULL,&ncfile);
ret = ncmpi_def_dim(ncfile, "d1", nprocs, &dimid);
ret = ncmpi_def_var(ncfile, "v1", NC_INT, ndims, &dimid, &varid1);
ret = ncmpi_def_var(ncfile, "v2", NC_INT, ndims, &dimid, &varid2);
ret = ncmpi_put_att_text(ncfile, NC_GLOBAL, "string", 13, buf);
ret = ncmpi_enddef(ncfile);
ret = ncmpi_put_vara_int_all(ncfile, varid1, &start, &count, &data);
ret = ncmpi_put_vara_int_all(ncfile, varid2, &start, &count, &data);
ret = ncmpi_close(ncfile);
MPI_Finalize();
```

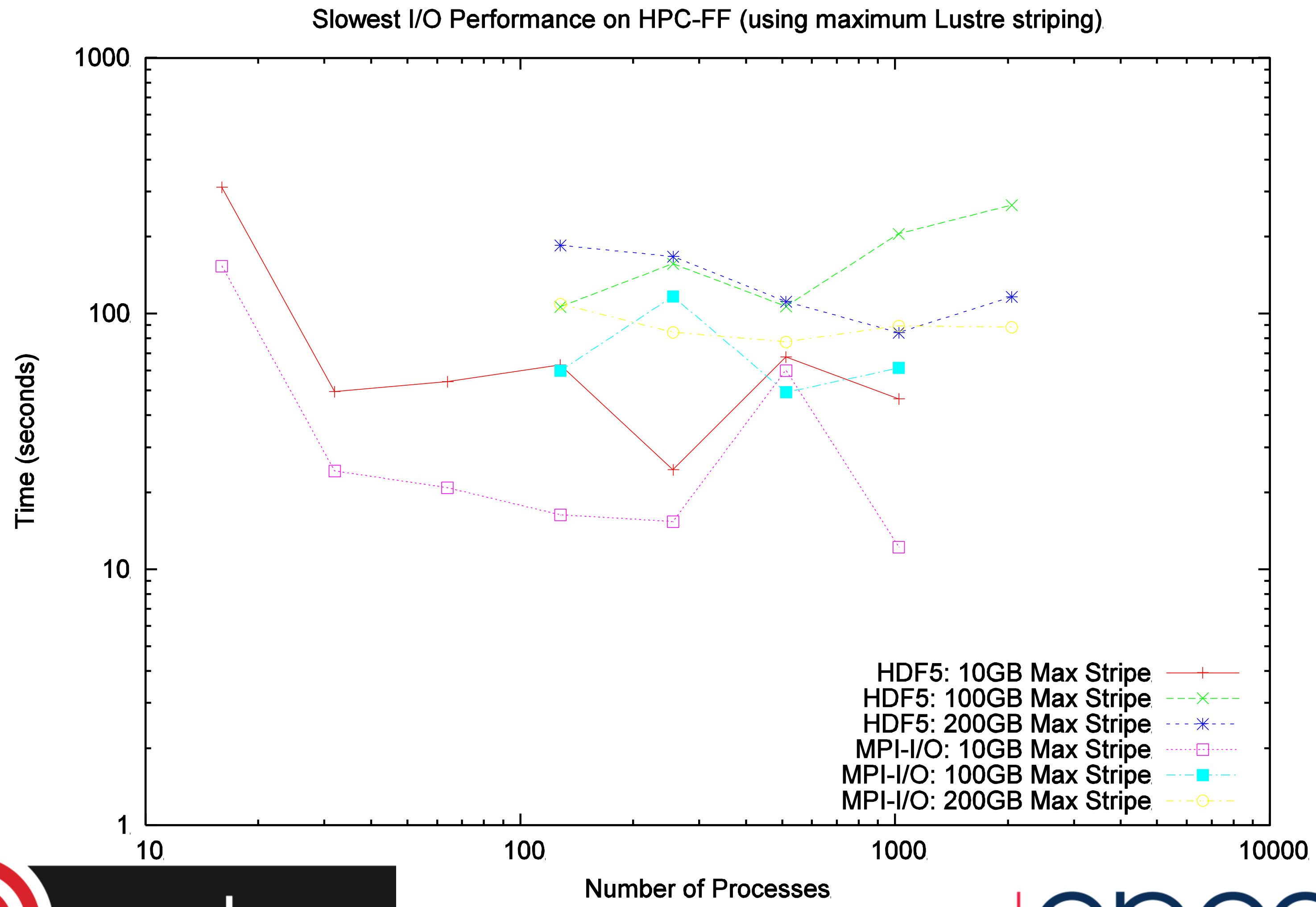
NetCDF on ARCHER

- We have multiple versions of NetCDF on ARCHER all available through modules:
  - NetCDF version 4 (Serial NetCDF)
    - module: cray-netcdf: versions: **4.3.3.1**, 4.3.2
  - NetCDF version 4 built with HDF5 parallel functionality
    - module:cray-netcdf-hdf5parallel: versions: **4.3.3.1** 4.3.2, 4.3.1, 4.3.2
  - Parallel NetCDF (Separate parallel library that can write NetCDF files in parallel)
    - module: cray-parallel-netcdf: versions: **1.6.1**, 1.5.0
  - NetCDF v4 for use with Python
    - module: pc-netcdf4-python: **1.1.0**, 1.1.5-python3

Performance



Performance – HDF5 vs MPI-I/O



What to use?

- This all assumes you are interested in parallel computing
- If raw performance is biggest issue for you
  - MPI-I/O
- If metadata/storage format is biggest issue for you
  - HDF5
- If you want to integrate with earth science tools
  - NetCDF

Further Reading

- *Introduction to HDF5*
  - www.hdfgroup.org/HDF5/doc/H5.intro.html
- *HDF5 User Guide*
  - www.hdfgroup.org/HDF5/doc/UG/index.html
- *HDF5 Reference Manual*
  - www.hdfgroup.org/HDF5/doc/RM/RM_H5Front.html
- *Introduction to Scientific I/O*
  - http://www.nersc.gov/users/training/online-tutorials/introduction-to-scientific-i-o/

Acknowledgements & Re Use

HDF5 Tutorial

- C version:
  - http://www2.epcc.ed.ac.uk/~amrey/ARCHER_Data_Management/
- Python version:
  - http://www2.epcc.ed.ac.uk/~amrey/FDM_2015/Python/
  - Don’t forget to `module load anaconda`