Introduction to NetCDF

Adrian Jackson
adrianj@epcc.ed.ac.uk

This lecture material

Some of this material (including images) has been borrowed from the unidata 2010 NetCDF tutorial:

http://www.unidata.ucar.edu/software/netcdf/workshops/2010/

archer

epcc

THE UNIVERSITY OF EDINBURGH

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

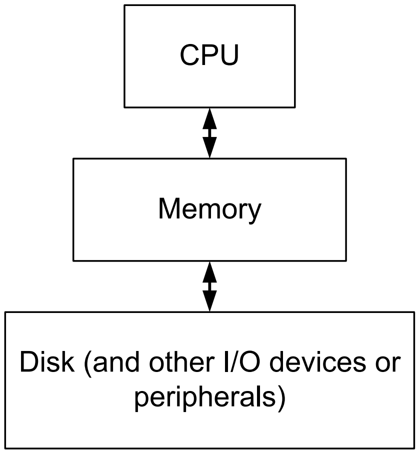
- Small parallel programs (i.e. under 1000 processors)
  - Cope with I/O overhead
- Large parallel programs (i.e. tens of thousand processors)
  - Can completely dominate performance
  - Exacerbate by poor functionality/performance of I/O systems
- Any opportunity for program optimisation important
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

Challenges of I/O

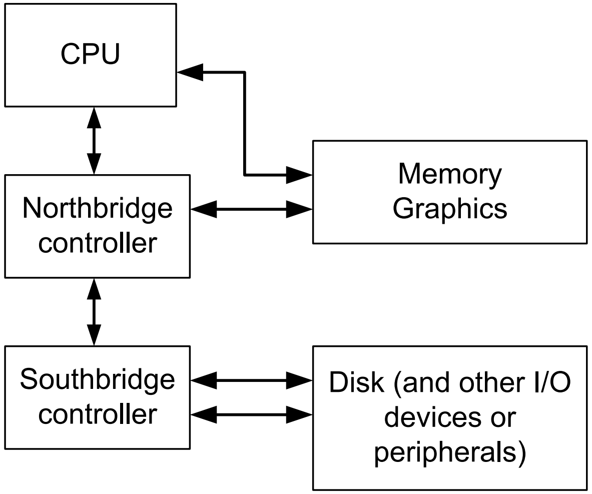
- Moves beyond process-memory model
  - data in memory has to physically appear on an external device
- Files are very restrictive
  - Don't often map well to common program data structures (i.e. flat file/ array)
  - Often no description of data in file
- I/O libraries or options system specific
  - Hardware different on different systems
- Lots of different formats
  - text, binary, big/little endian, Fortran unformatted, ...
  - Different performance and usability characteristics
- Disk systems are very complicated
  - RAID disks, caching on disk, in memory, I/O nodes, network, etc…

Challenges of I/O

- Standard computer hardware
  - Possibly multiple disks
  - PATA, SATA, SCSI (SAS)
- Optimisations
  - RAID (striping and replication)
  - Fast disks (SSD or server)
- HPC/Server/SAN hardware
  - Many disks
  - SCSI (SAS), Fibre channel
- Optimisations
  - Striped
  - Multiple adapters and network interfaces
- Network filesystems
  - Provide access to data from many machines and for many users



Abstract Hardware Hierarchy



Actual Hardware Hierarchy

Performance

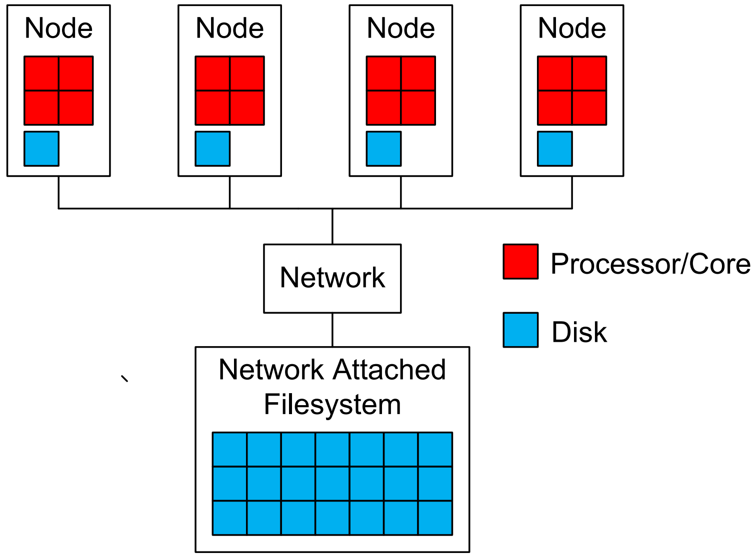
Interface	Throughput Bandwith (MB/s)
PATA (IDE)	133
SATA	600
Serial Attached SCSI (SAS)	600
Fibre Channel	2,000

High Performance or Parallel I/O

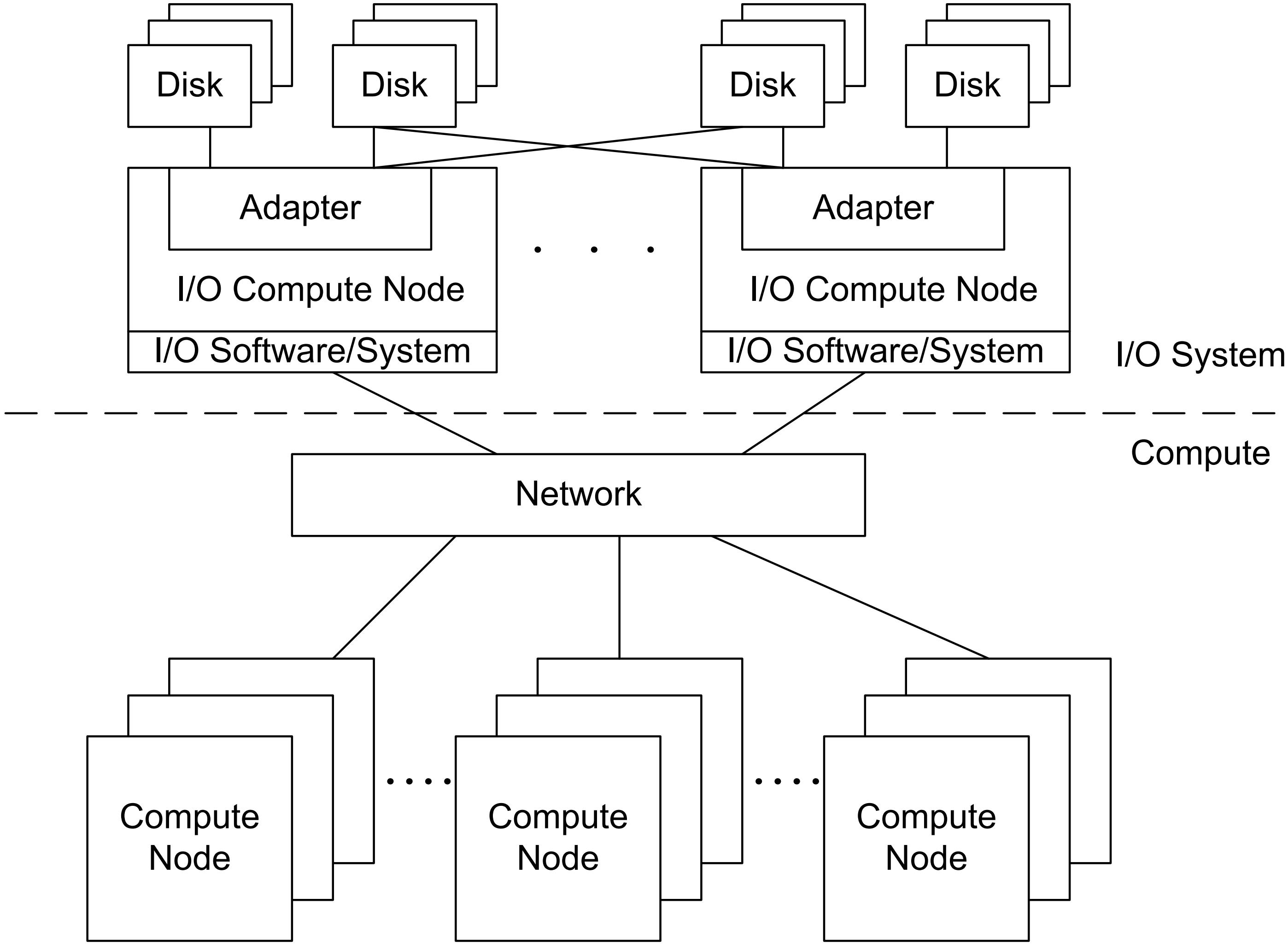
- Lots of different methods for providing high performance I/O
- Hard to support multiple processes writing to same file
  - Basic O/S does not support
  - Data cached in units of disk blocks (eg 4K) and is *not coherent*
  - Not even sufficient to have processes writing to distinct parts of file
- Even reading can be difficult
  - 1024 processes opening a file can overload the filesystem Limit on file handles etc….
- Data is distributed across different processes
  - Dependent on number of processors used, etc…
- Parallel file systems may allow multiple access
  - but complicated and difficult for the user to manage

HPC/Parallel Systems

- Basic cluster
  - Individual nodes
  - Network attached filesystem
  - Local scratch disks
- Multiple I/O systems
  - Home and work
  - Optimised for production or for user access
- Many options for optimisations
  - Filesystem servers, caching, etc…



Hierarchy



Disk
Disk
Disk
Disk
Adapter
I/O Compute Node
I/O Software/System
Adapter
I/O Compute Node
I/O Software/System
I/O System
Compute
Network
Compute Node
Compute Node
Compute Node

I/O Strategies

- Basic one file for a program
    - Works fine for serial
    - Most codes use this initially
    - Works for shared memory parallelism
- Distributed memory
    - Data now not in single memory space
- Master I/O
    - Use communication to get and send all data from one process
    - High overhead
    - Use single file
    - Memory issues, no access to I/O resources at scale

I/O Strategies cont.

- Individual files
  - Each process writes own file (either on shared filesystem or local scratch space)
  - Use as much of I/O system as possible
  - file contents dependent on number of CPUs and decomposition
  - pre / post-processing steps needed to change number of processes
  - Filesystem breaks down for large numbers of processors
  - File handles or number of files a problem
- Look to better solution
  - I/O libraries

MPI-I/O

- Aim to provide distributed access to single file
  - File shared
  - Control by programmer
  - Look like a serial program has written the data
- Part of MPI-2 standard
  - Not always available in MPI implementations
  - http://www.mpi-forum.org/docs/docs.html
  - Can use ROMIO (MPI-IO built on MPI-1 calls)
  - Performance dependent on implementation
- Built on MPI collective operations
  - Data structure defined by programmer

MPI-I/O cont.

- Array based I/O
  - Each process creates description of subset it holds (derived datatype)
  - No checking of correctness
- Library handles read and write to files
  - Don’t ever have all in memory
  - Everything done with MPI calls
  - Scale as well as MPI communications
  - Best performance for big reads/writes
- Info object for passing system specific information
  - Lots of optimisations, tweaking, etc…

HDF5

- **H**ierarchical **D**ata **F**ormat
  - Model for managing and storing data
  - Binary data format, library, and tools
  - HDF5 library implements model and provides functionality to transform data between stored forms
  - Extensible and portable
  - Data preservation
- Based on two types of objects
  - Datasets: Multidimensional arrays
  - Groups: Containers for holding datasets (or other groups)
- Hierachical storage
  - Filesystem like access possible
  - **/path/to/resource**

Optimisation and Parallel HDF5

- Optimisation options
  - File level, Data transfer level, Memory management, File space management, Chunking, Compact storage
  - H5Pset_buffer: set size of internal data transfer buffer

- Parallel version
  - Uses MPI and MPI-I/O
  - Same functionality as HDF5

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

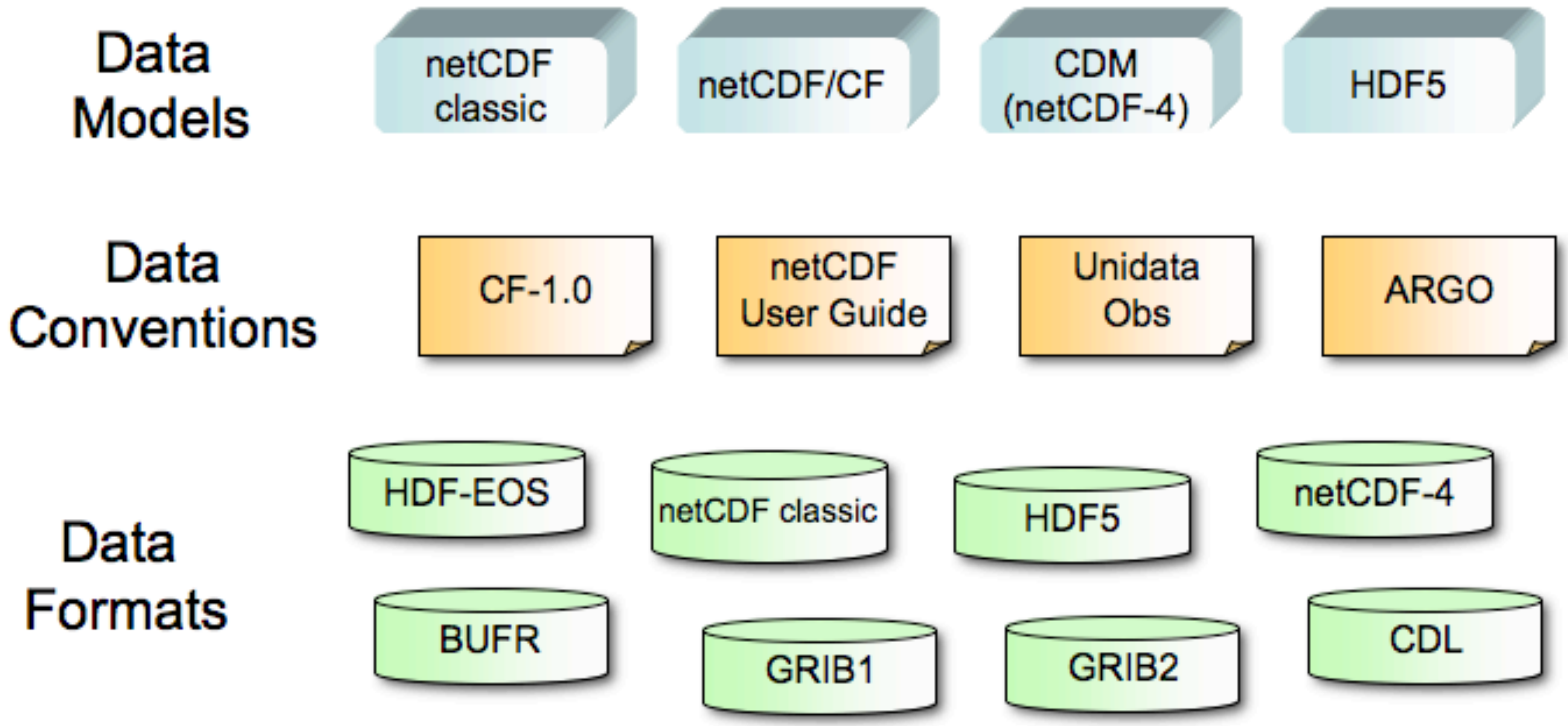
- NetCDF
  - Software library and self-describing data format
  - Portable, machine independent data
  - Can use HDF5 or NetCDF format (HDF5 gives larger files and unlimited array dimensions) in NetCDF 4.0 (latest version)

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

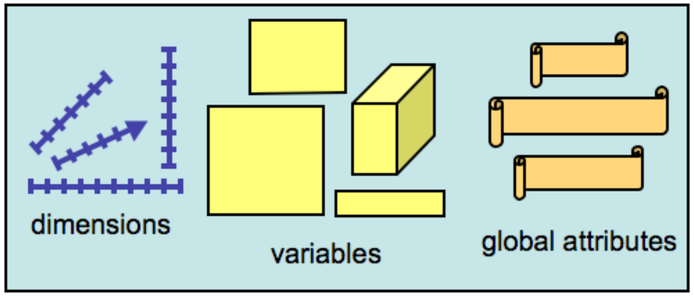
NetCDF

- 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
  - Parallel NetCDF

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

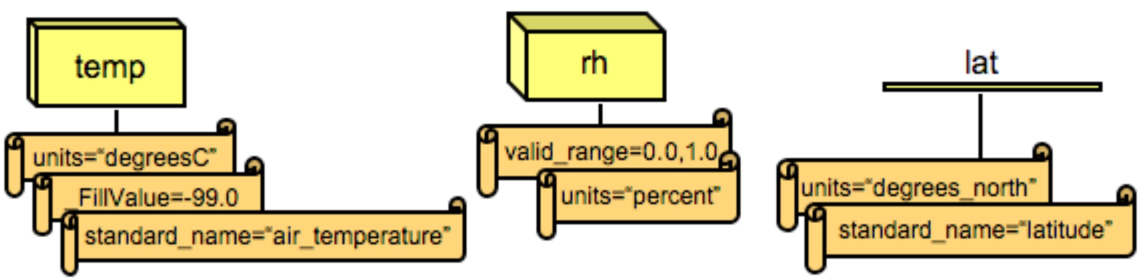
- Variables define the things that hold data:
  - Has a name, type, shape, can have attributes, and values.
  - Type:
    - Classic NetCDF type is the *external type* of its data as represented on disk, i.e.
      - char
      - byte (8 bits)
      - short (16 bits)
      - int (32 bits)
      - float (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
- C interface is used as the core of all but the Java interface

```
#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.nc2.Dimension;
import ucar.ma2.*;
import ucar.nc2.NetcdfFileWriter;
import ucar.nc2.Variable;
    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
  - Uses 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

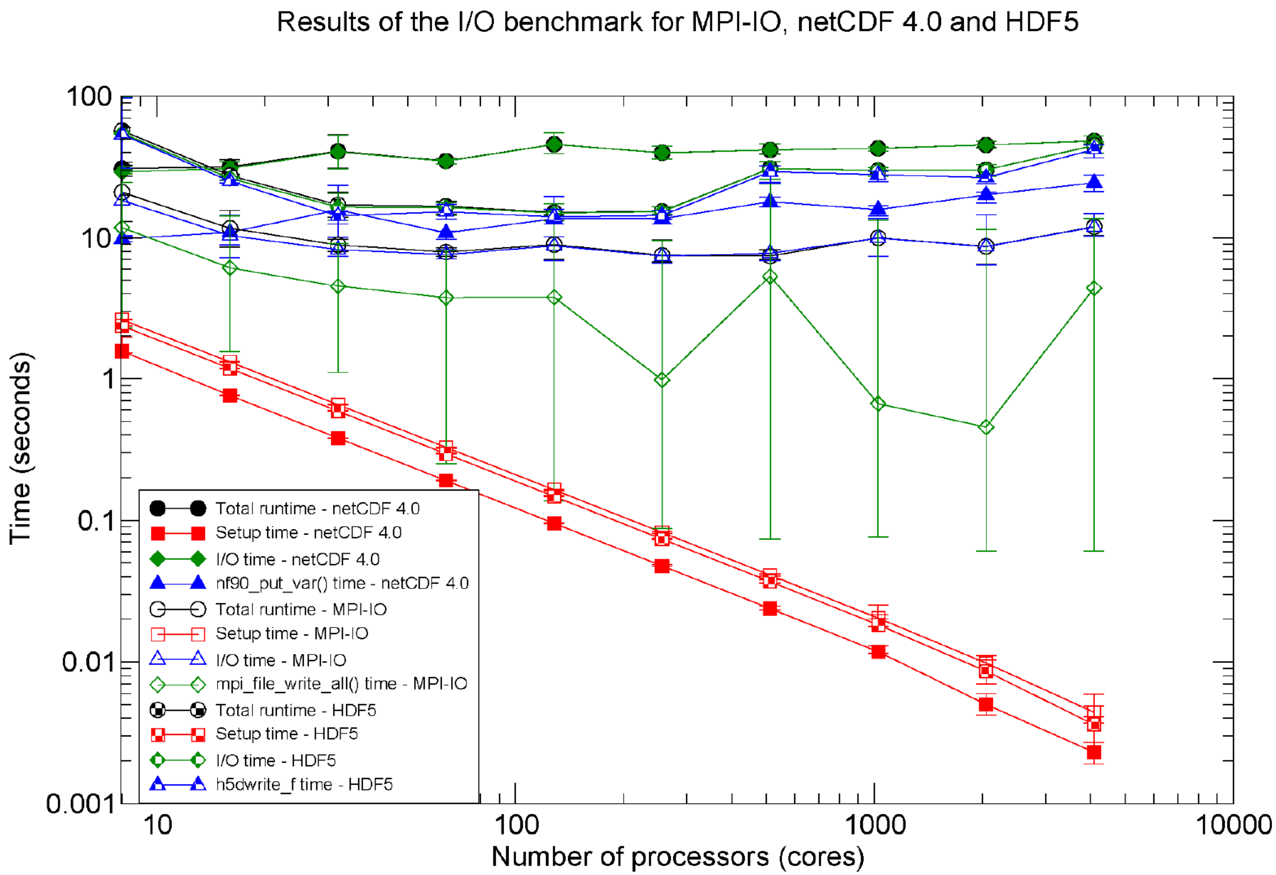
- Parallel-NetCDF
  - Parallel I/O library to support parallel I/O in NetCDF (CDF-1 and CDF-2)
  - Also supports extended CDF-2 (CDF-5)

```
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();
```

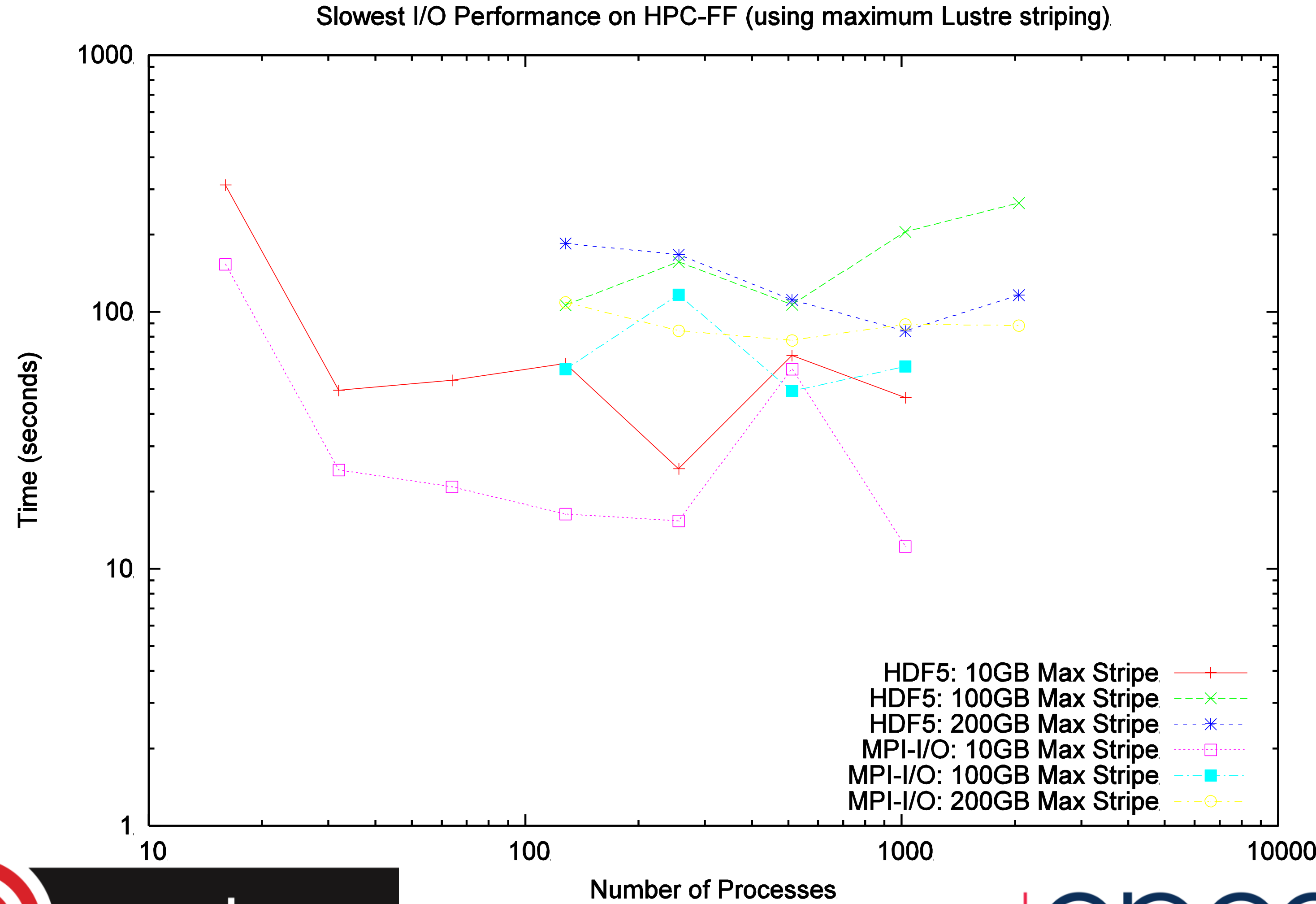
NetCDF on ARCHER

- We have three versions of NetCDF on ARCHER all available through modules:
  - NetCDF version 4
    - module: Cray-netcdf: versions: **4.3.2** 4.3.0, 4.3.1, 4.3.2
  - NetCDF version 4 built with HDF5 parallel functionality
    - module:cray-netcdf-hdf5parallel: versions: **4.3.2** 4.3.0, 4.3.1, 4.3.2
  - Parallel NetCDF
    - module: cray-parallel-netcdf: versions: **1.5.0** 1.3.1.1, 1.4.0, 1.4.1, 1.5.0

Performance



Performance – HDF5 vs MPI-I/O



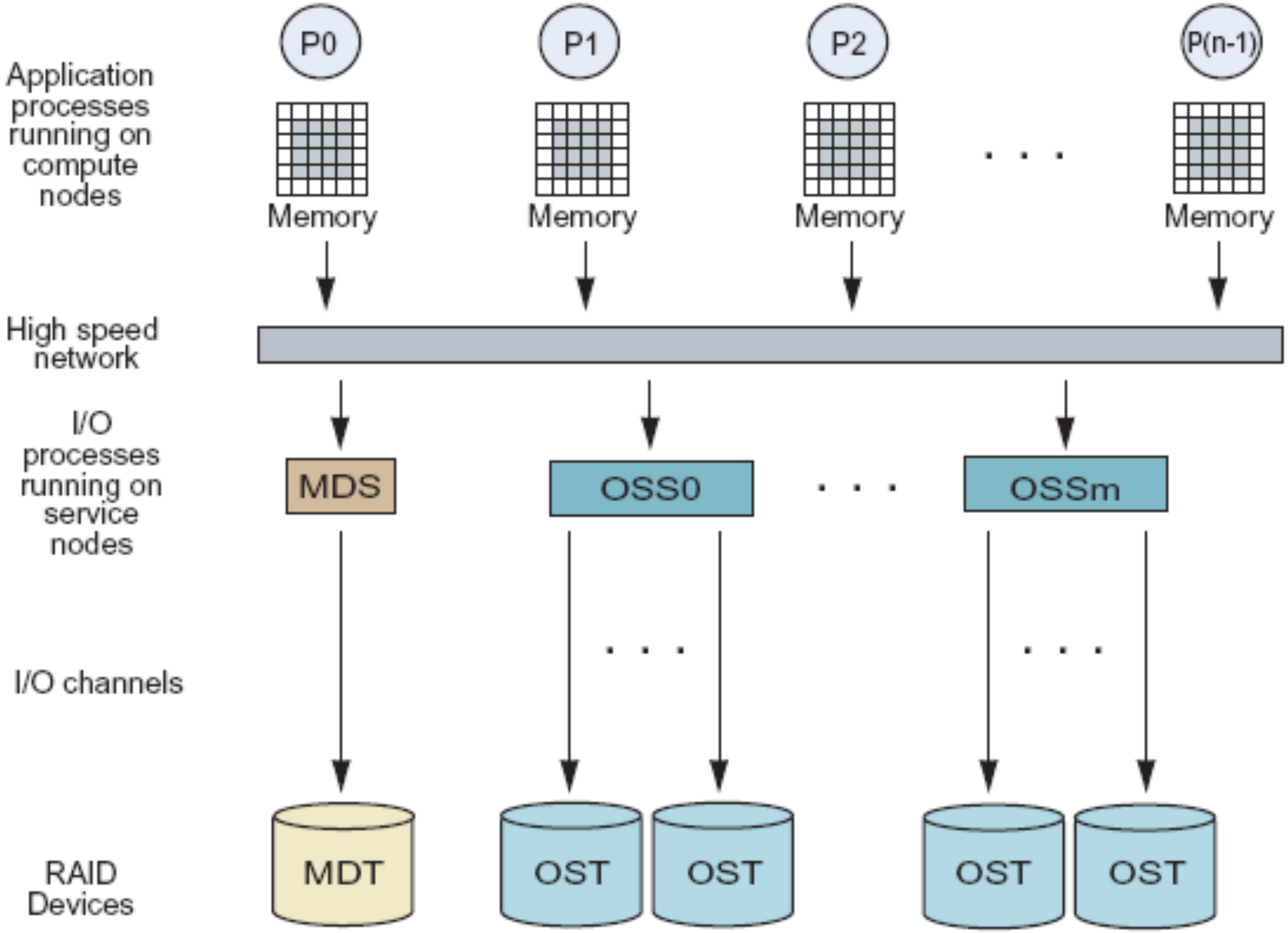
What to use?

- 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

Lustre

- Three functional units
  - Object Storage Servers (OSS)
    - Store data on one or more Object Storage Targets (OST)
    - The **OST** handles interaction between client data request and underlying physical storage
    - An **OSS** typically serves 2-8 targets, each target a local disk system. The capacity of the Lustre file system is the sum of the capacities provided by the targets
    - The **OSS** operate in parallel, independent of one another
  - Metadata Target (MDT)
    - One per filesystem, storing all metadata: filenames, directories, permissions, file layout
    - Stored on Metadata Server (MDS)
  - Clients
    - Supports standard POSIX access

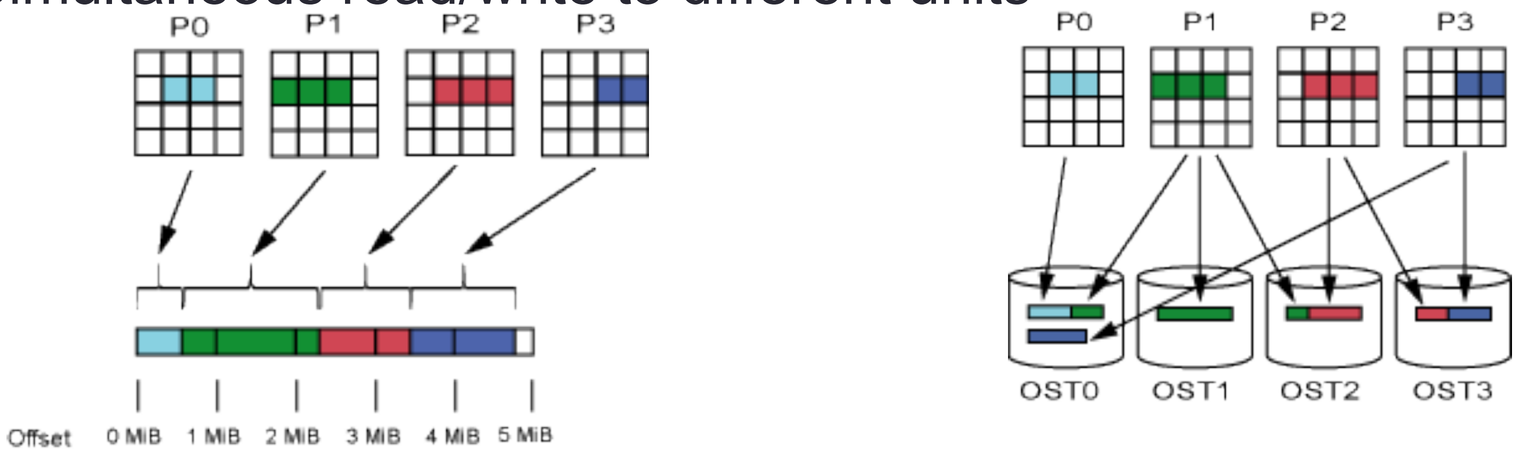
Lustre cont.



Application processes running on compute nodes
P0
P1
P2
P(n-1)
Memory
Memory
Memory
Memory
High speed network
I/O processes running on service nodes
MDS
OSS0
OSSm
I/O channels
RAID Devices
MDT
OST
OST
OST
OST

Lustre cont…

- Supports different networks
  - Infiniband, Ethernet, Myrinet, Quadrics
- Striping
  - Data striped across OSTs (round robin)
  - File split into units
  - Simultaneous read/write to different units



archer

epcc

THE UNIVERSITY OF EDINBURGH

Lustre commands

- Striping cont.
  - Improves bandwidths, overall performance available, and maximum file size
  - Incurs communication overhead and contention potentials including serialisation if multiple processes accessing same units
- lfs command for more information and configuration

`adrianj@nid16958:~>lfs df -h`

(query number of OSTs)

`adrianj@nid16958:~> lfs getstripe dirname`

(query `stripe count, stripe size`)

`adrianj@nid16958:~> lfs setstripe dirname 0 -1 -1`

(set large file `stripe size, start index, stripe count`)

`adrianj@nid16958:~> lfs setstripe dirname 0 -1 1`

(set lots of files `stripe size, start index, stripe count`)

Lustre on ARCHER

- See white paper on I/O performance on ARCHER:

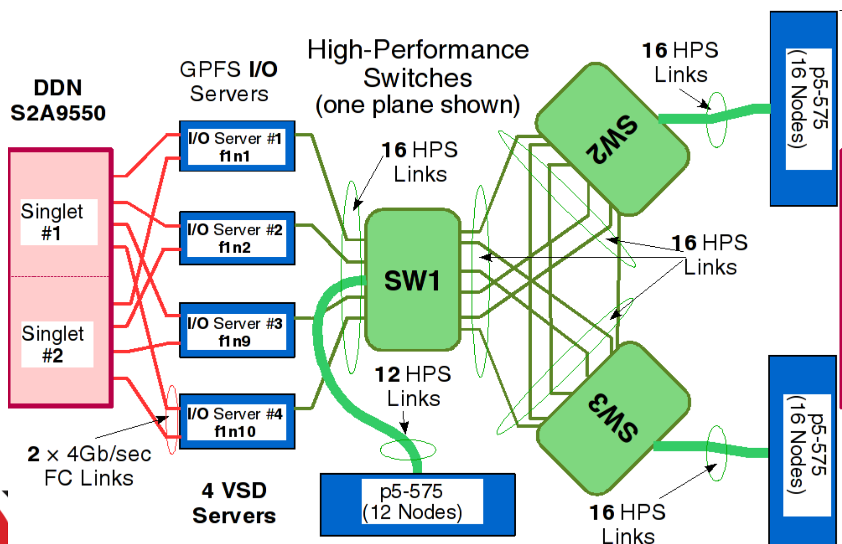
- http://www.archer.ac.uk/documentation/white-papers/parallelIO/ARCHER_wp_parallelIO.pdf

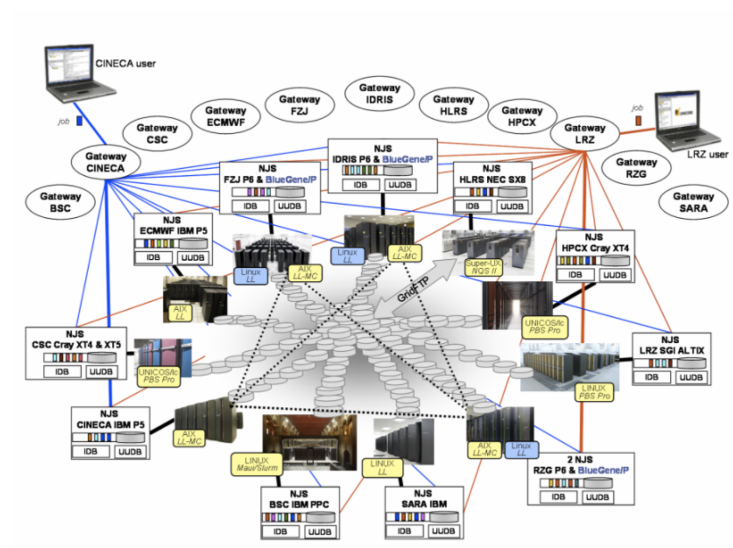
GPFS

- IBM **G**eneral **P**urpose **F**ile**s**ystem
  - Files broken into blocks, striped over disks
  - Distributed metadata (including dir tree)
  - Extended directory indexes
  - Failure aware (partition based)
  - Fully POSIX compliant
- Storage pools and policies
  - Groups disks
  - Tiered on performance, reliability, locality
  - Policies move and manage data
  - Active management of data and location
- High performance

GPFS cont…

- Configuration
  - Shared disks (i.e. SAN attached to cluster)
  - Network Shared disks (NSD) using NSD servers
  - NSD across clusters (higher performance NFS)





AFS

- Andrews Filesystem
  - Large/wide scale NFS
  - Distributed, transparent
  - Designed for scalability
- Server caching
  - File cached local, read and writes done locally
  - Servers maintain list of open files (callback coherence)
  - Local and shared files
- File locking
  - Doesn’t support large databases or updating shared files
- Kerberos authentication
  - Access control list on directories for users and groups

POSIX I/O

- Standard interface to files
  - Unix/Linux approach
  - Based on systems with single filesystem
  - open, close, write, read, etc…
- Does not support parallel or HPC I/O well
  - Many NFS don't fully implement it for performance reasons
- Some work on extending for HPC