# Summer School on Effective HPC for Climate and Weather

## Input/Output and Middleware

Luciana Pedro, Julian Kunkel

Department of Computer Science, University of Reading

18 June 2020

# Outline



*Disclaimer: This material reflects only the author's view and the EU-Commission is not responsible for any use that may be made of the information it contains*

# Learning Objectives

- Describe the role of middleware and file formats (Section Middleware)
- Discuss challenges for data-driven research (Section Data)
- Identify typical I/O performance issues and their causes (Section Performance)
- Apply performance models to assess and optimize the application I/O performance (Section Model)
- Design a data model for NetCDF/CF (Section NetCDF)
- Execute programs in C and Python that read and write NetCDF files in a metadata-aware manner (Section Progs)
- Analyze, manipulate and visualise NetCDF data (Section NetCDF2)
- Implement an application that utilizes parallel I/O to store and analyze data (Section Parallel I/O)
- Describe ongoing research activities in high-performance storage (Section Storage)

# Building NetCDF from Scratch

- The usual way of building netCDF requires the HDF5, zlib, and curl libraries.
- Files for the libraries can be found in:

```
ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf-4
```

# Installing curl

- apt-get install libcurl4-openssl-dev

# Installing zlib

- wget `ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf-4/zlib-1.2.8.tar.gz`
  - Newest version to later use ncview
  - wget `https://sourceforge.net/projects/libpng/files/zlib/1.2.9/zlib-1.2.9.tar.gz`
- tar -xvzf zlib-1.2.8.tar.gz
- cd zlib-1.2.8
- mkdir /home/username/local/
- ./configure –prefix=/home/username/local/
- make check install

# Installing HDF5

- wget ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf-4/hdf5-1.8.13.tar.gz
- tar -xvzf hdf5-1.8.13.tar.gz
- cd hdf5-1.8.13
- ./configure –with-zlib=/home/username/local/ –prefix=/home/username/local/
- make
- make check
- make install
  - make check install
  - If not done separately, it might not work!

# Installing NetCDF

- Check the latest version at `https://www.unidata.ucar.edu/downloads/netcdf/`
- wget `ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf-c-4.7.4.tar.gz`
- tar -xvzf netcdf-c-4.7.4.tar.gz
- cd netcdf-c-4.7.4
- CPPFLAGS=-I/home/username/local/include LDFLAGS=-L/home/username/local/lib ./configure –prefix=/home/username/local
- make check install

# Finishing the set up

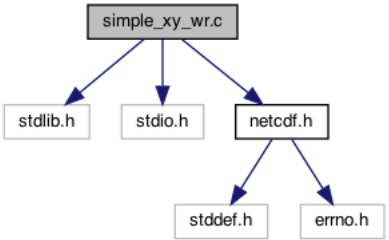
- Link the NetCDF library
  - export LD_LIBRARY_PATH=/home/username/local/lib/
  - sudo ldconfig
- Create a new directory (for instance, /home/username/example) and create the file from the given source using an editor of your choice.

# File Reference: simple_xy_wr.c

This is an example program demonstrating a simple 2D write. It is intended to illustrate the use of the netCDF C API.

- https://www.unidata.ucar.edu/software/netcdf/docs/simple__xy__wr_8c.html
- https://www.unidata.ucar.edu/software/netcdf/docs/simple__xy__wr_8c_source.html

Dependency graph for simple_xy_wr:



# File simple_xy_wr.c: Header and Constants Declaration

```
#include <stdlib.h>
#include <stdio.h>
#include <netcdf.h>

/* This is the name of the data file we will create. */
#define FILE_NAME "simple_xy.nc"

/* We are writing 2D data, a 6 x 12 grid. */
#define NDIMS 2
#define NX 6
#define NY 12

/* Handle errors by printing an error message and exiting with a
* non-zero status. */
#define ERRCODE 2
#define ERR(e) {printf("Error: %s\n", nc_strerror(e)); exit(ERRCODE);}

int
main()
{
  ...
  ...
  ...
}
```

# File simple_xy_wr.c: Variables Declaration

```
...
...
...

int
main()
{
  /* When we create netCDF variables and dimensions, we get back an
   * ID for each one. */
  int ncid, x_dimid, y_dimid, varid;
  int dimids[NDIMS];

  /* This is the data array we will write. It will be filled with a
   * progression of numbers for this example. */
  int data_out[NX][NY];

  /* Loop indexes, and error handling. */
  int x, y, retval;

  ...
  ...
  ...
}
```

# File simple_xy_wr.c: Creating (loading!) data

```
...

int
main()
{
  ...

  /* Create some pretend data. If this wasn't an example program, we
   * would have some real data to write, for example, model
   * output. */
  for (x = 0; x < NX; x++)
     for (y = 0; y < NY; y++)
    data_out[x][y] = x * NY + y;

  ...
}
```

# File simple_xy_wr.c: Creating the NetCDF file

```
...

int
main()
{
  ...

  /* Always check the return code of every netCDF function call. In
   * this example program, any retval which is not equal to NC_NOERR
   * (0) will cause the program to print an error message and exit
   * with a non-zero return code. */

  /* Create the file. The NC_CLOBBER parameter tells netCDF to
   * overwrite this file, if it already exists.*/
  if ((retval = nc_create(FILE_NAME, NC_CLOBBER, &ncid)))
     ERR(retval);

  ...
}
```

# File simple_xy_wr.c: Defining the dimensions

```
...

int
main()
{
  ...

  /* Define the dimensions. NetCDF will hand back an ID for each. */
  if ((retval = nc_def_dim(ncid, "x", NX, &x_dimid)))
     ERR(retval);
  if ((retval = nc_def_dim(ncid, "y", NY, &y_dimid)))
     ERR(retval);

  /* The dimids array is used to pass the IDs of the dimensions of
   * the variable. */
  dimids[0] = x_dimid;
  dimids[1] = y_dimid;

  ...
}
```

# File simple_xy_wr.c: Defining the variable

```
...

int
main()
{
  ...

  /* Define the variable. The type of the variable in this case is
   * NC_INT (4-byte integer). */
  if ((retval = nc_def_var(ncid, "data", NC_INT, NDIMS,
                dimids, &varid)))
     ERR(retval);

  /* End define mode. This tells netCDF we are done defining
   * metadata. */
  if ((retval = nc_enddef(ncid)))
     ERR(retval);

  ...
}
```

# File simple_xy_wr.c: Writing data to the file

```
...

int
main()
{
  ...

  /* Write the pretend data to the file. Although netCDF supports
   * reading and writing subsets of data, in this case we write all
   * the data in one operation. */
  if ((retval = nc_put_var_int(ncid, varid, &data_out[0][0])))
     ERR(retval);

  /* Close the file. This frees up any internal netCDF resources
   * associated with the file, and flushes any buffers. */
  if ((retval = nc_close(ncid)))
     ERR(retval);

  ...
}
```

# File simple_xy_wr.c: Getting SUCCESS!

```
...

int
main()
{
  ...

  printf("*** SUCCESS writing example file simple_xy.nc!\n");
  return 0;
}
```

# Compiling and running the file simple_xy_wr.c

Create (copy!) and compile the file simple_xy_wr.c

- `gcc -I/home/username/local/include simple_xy_wr.c -o simple_xy_wr -L/home/username/local/lib -lnetcdf`
  - What does that mean?!
- Run the file simple_xy_wr
  - `./simple_xy_wr`
  - `*** SUCCESS writing example file simple_xy.nc!`
- Check that the file `cmp test.nc simple_xy.nc` is in your directory
  - `ls`

# Using ncdump

Inspect the output file `simple_xy.nc` using `ncdump`

- `ncdump simple_xy.nc`
- Only works like that in my laptop:
- `/home/lucy/netcdf/netcdf-c-4.7.4/ncdump/ncdump simple_xy.nc`
- (It should be just ncdump file, but esdm is on my way and I don't know how to make another link)

# NetCDF CDL Format

```
netcdf simple_xy {
dimensions:
x = 6 ;
y = 12 ;
variables:
int data(x, y) ;
data:

 data =
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,
  12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
  24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35,
  36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47,
  48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
  60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71 ;
}
```

# Using ncgen

Create a NetCDF file using ncgen and the CDL output

- /home/lucy/netcdf/netcdf-c-4.7.4/ncdump/ncdump simple_xy.nc > test.cdl
- more test.cdl
- /home/lucy/netcdf/netcdf-c-4.7.4/ncgen/ncgen -b test.cdl
- ls
- cmp test.nc simple_xy.nc

# Creating the C File

Create a C file using ncgen and the CDL output

- `/home/lucy/netcdf/netcdf-c-4.7.4/ncgen/ncgen -lc test.cdl > test.c`
- more test.c

  - Start all over again!!!

# Starting all over again!

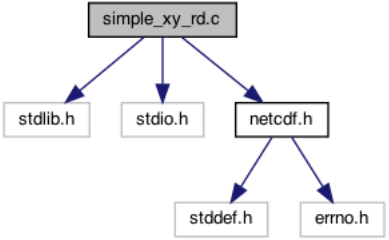
- gcc -I/home/lucy/local/include test.c -o test -L/home/lucy/local/lib -lnetcdf
- mv test.nc test2.nc
- ./test
- ls
- cmp test.nc test2.nc

# File Reference: simple_xy_rd.c

This is a simple example which reads a small dummy array, which was written by simple_xy_wr.c. It is intended to illustrate the use of the netCDF C API.

- https://www.unidata.ucar.edu/software/netcdf/docs/simple__xy__rd_8c.html
- https://www.unidata.ucar.edu/software/netcdf/docs/simple__xy__rd_8c_source.html

Dependency graph for simple_xy_wr:



# Reading the file simple_xy.nc

Create (copy!), compile and run the file simple_xy_rd.c

- Compile the file simple_xy_rd.c
  - ```
    gcc -I/home/lucy/local/include simple_xy_rd.c -o simple_xy_rd -L/home/lucy/local/
    ```
- Run the file simple_xy_rd
  - ./simple_xy_rd
  - *** SUCCESS reading example file simple_xy.nc!
    ```

# ncview

- Installation is fine.
- What does it do?

The ESiWACE1/2 projects have received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No **675191** and No **823988**


*Disclaimer: This material reflects only the author's view and the EU-Commission is not responsible for any use that may be made of the information it contains*