netCDF and HDF5 file formats on CÉCI clusters

Quentin GLAUDE: quentin.glaude@uliege.be
28th November 2024, CYCL09b - Louvain-La-Neuve

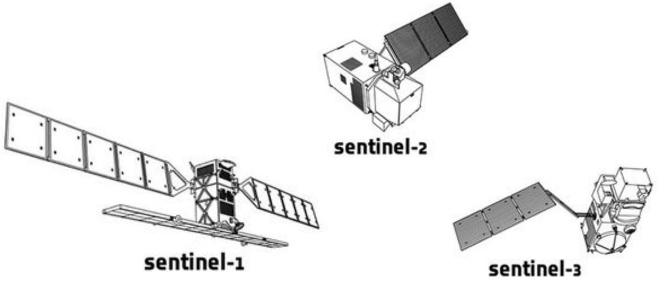
About myself:

- PhD in Sciences: Laboratory of Glaciolology (ULB) and Liège Space Center (ULiège)
- Postdoc in ULiège: Laboratory of Climatology (ULiège)
- Research stay in Columbia University of NYC
- Research Logistician at ULiège (CÉCI)

COLUMBIA UNIVERSITY

Skills:

- Remote sensing, modelling, signal and data processing, SAR interferometry



Objectives:

- Understanding basics of netCDF and HDF5 data formats
  → their importance in Geosciences and in data dissemination

- How to use these formats in your code and on CÉCI clusters
  → Hands-on session (Fortran90, C / C++, Python)

Difficulty:

- Entry level

PART I

-

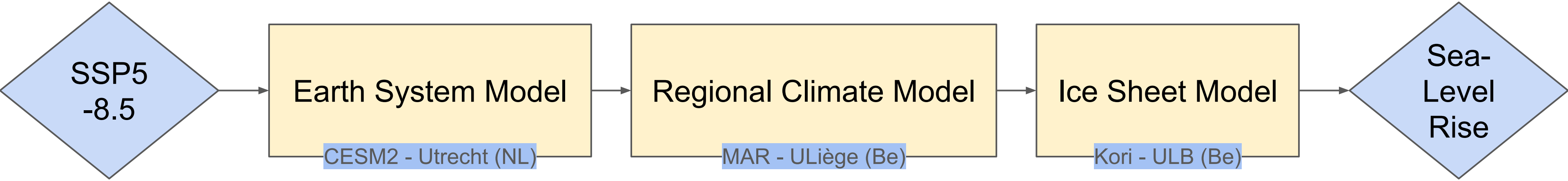
Introduction to netCDF and HDF5

Importance of Data Formats in Sciences

1. Need for structured, efficient data storage
   - Increase in the amount of data: need scalable and efficient data for I/O processes
   - Cross disciplinary research needs data usable on-the-go, from distincts fields that are not familiar to specific data format

2. Data formats impact:
   - Research quality: organized, efficient, and accessible.
   - Reproducibility: Allows experiments to be replicated and validated.
   - **Cross-disciplinary collaboration*:** Facilitates data exchange across domains (e.g., climate science, bioinformatics).

Practical example:



** Also a reason why shifting to new data format (ex: Zarr) takes some time.*

Data Formats in Scientific Research

1. Structured vs Unstructured Formats:
   - Unstructured: Minimal schema, plain-text representation (e.g., JSON, YAML, CSV).
   - Structured: Organized, predictable schema (e.g., netCDF, HDF5, Zarr, GeoPackage).

2. Importance of Structured Formats:
   - Data over Data: Organizes multi-dimensional datasets and interlinked variables (e.g., temporal, spatial, and variable dimensions).
   - Data Cubes: Support for higher-dimensional arrays for rich scientific analysis.
   - Efficiency: Optimized for storage and faster read/write operations.
   - Self-Described: Metadata embedded for portability and clarity.

	Structured	Unstructured
Data Size	Efficient	Bloated
Metadata support	Yes	Minimal
Multi-Dimensional	Yes	No
Read/Write Speed	Fast	Slow

Historical Context and Development

1. Origins
   - **netCDF** : Developed by UCAR (University Corporation for Atmospheric Research). Initially designed for geosciences and climate modelling (1988)
   - **HDF5** : Created by NCSA (National Center for Supercomputing Applications). Serve as a general-purpose format for storing complex scientific data (1987).

2. Evolution of Formats
   - **netCDF** : Transition from netCDF-3 in 1997 (Widely adopted, foundational version for many years) to netCDF-4 in 2008 (HDF5-based, supporting compression and parallel I/O).
   - **HDF5** : Continuously adapted for performance in HPC and integration into new tools

3. Emerging Formats
   - **Zarr** : Modern cloud-optimized, chunked, compressed format, designed for distributed and cloud-based workflows.

Real world example:

1. netCDF vs .csv file size
    - **netCDF** : ~11MB
    - **CSV** : 151 files of 101 x 181 values (avg 12B per FP32 float) + metadata stored separately (5%)
      → ~ 35 MB

2. Other differences
    - Slower read and write processes
    - Dedicated data readers
    - Slower files synchronisation (metadata overhead + latency)

```
qglaude@nic5-login1 ~/formation_netcdf_hdf5 $ ncdump
-h MAR_ME_15km.nc
netcdf MAR_ME_15km {
dimensions:
        TIME = UNLIMITED ; // (151 currently)
        bnds = 2 ;
        X10_110 = 101 ;
        Y20_200 = 181 ;
        SECTOR1_1 = 1 ;
variables:
        float TIME(TIME) ;
                TIME:standard_name = "time" ;
                TIME:long_name = "time" ;
                TIME:bounds = "TIME_bnds" ;
                TIME:units = "days since 1947-09-01
00:00:00" ;
                TIME:calendar = "standard" ;
                TIME:axis = "T" ;
        double TIME_bnds(TIME, bnds) ;
        float X10_110(X10_110) ;
                X10_110:long_name = "x" ;
                X10_110:units = "km" ;
                X10_110:axis = "X" ;
        float Y20_200(Y20_200) ;
                Y20_200:long_name = "y" ;
                Y20_200:units = "km" ;
                Y20_200:axis = "Y" ;
        float SECTOR1_1(SECTOR1_1) ;
                SECTOR1_1:standard_name = "depth" ;
                SECTOR1_1:long_name = "sector" ;
                SECTOR1_1:units = "level" ;
                SECTOR1_1:positive = "down" ;
                SECTOR1_1:axis = "Z" ;
                SECTOR1_1:point_spacing = "even" ;
        float ME(TIME, SECTOR1_1, Y20_200, X10_110) ;
                ME:long_name = "Meltwater production"
;
                ME:units = "mmWE/day" ;
                ME:_FillValue = -1.e+34f ;
                ME:missing_value = -1.e+34f ;
                ME:cell_methods = "TIME: mean" ;
                ME:history = "From
ICE.q01.1950.01.01-05" ;
```

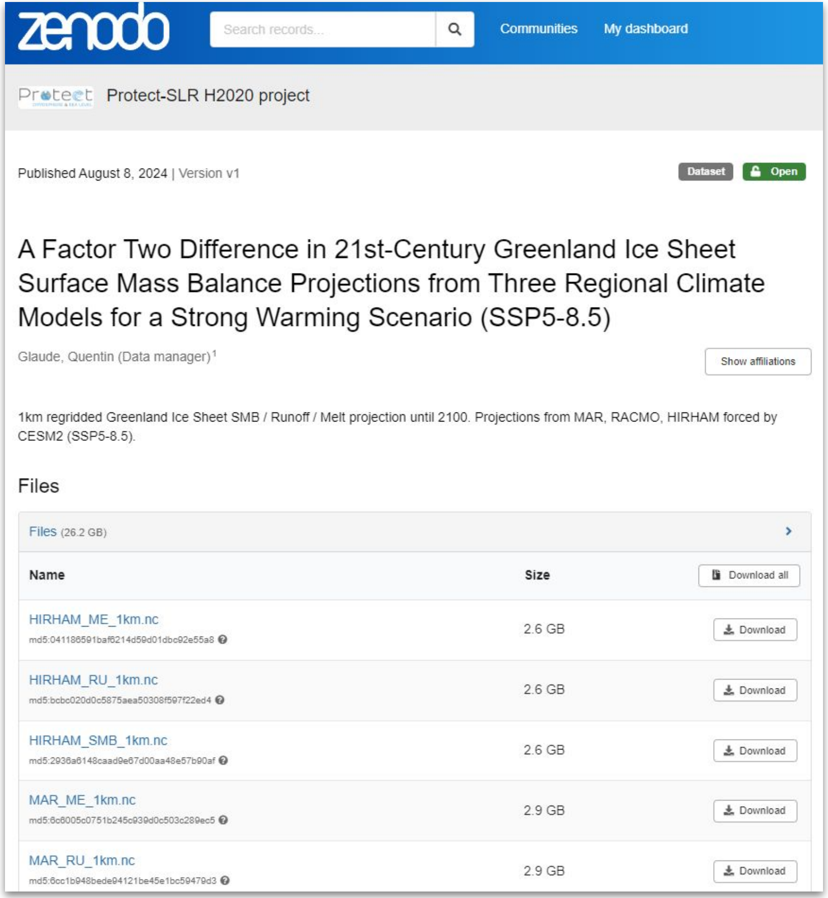
netCDF

Introduction to netCDF

1. Introduction to netCDF
    - A **self-describing data format** for managing multi-dimensional scientific data
    - Designed for geospatial and atmospheric datasets but widely used across domains

2. Why is it important?
    - Facilitates **portability** and **scalability** in high-performance computing environments
    - Provides tools for **efficient I/O operations** in large-scale simulations and real-world data analysis

3. Common Use Cases:
    - Climate modeling / Oceanography
    - Earth system science and remote sensing



netCDF Structure:

How is data stored in netCDF?

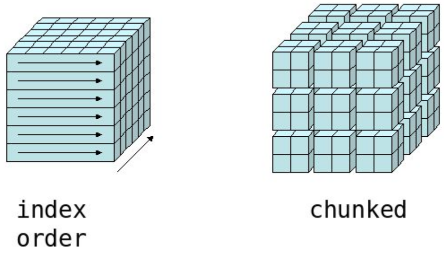
- **Dimensions**:
  - Define the data space
- **Variables**:
  Contain the actual data, linked to dimensions
- **Attributes**:
  Metadata providing context (e.g., units).

```
MAR_ME_15km {
dimensions:
        TIME = UNLIMITED ; // (151 currently)
        bnds = 2 ;
        X10_110 = 101 ;
        Y20_200 = 181 ;
        SECTOR1_1 = 1 ;
variables:
        float TIME(TIME) ;
                TIME:standard_name = "time" ;
                TIME:long_name = "time" ;
                TIME:bounds = "TIME_bnds" ;
                TIME:units = "days since 1947-09-01
00:00:00" ;
                TIME:calendar = "standard" ;
                TIME:axis = "T" ;
        double TIME_bnds(TIME, bnds) ;
        float X10_110(X10_110) ;
                X10_110:long_name = "x" ;
                X10_110:units = "km" ;
                X10_110:axis = "X" ;
        float Y20_200(Y20_200) ;
                Y20_200:long_name = "y" ;
                Y20_200:units = "km" ;
                Y20_200:axis = "Y" ;
        float SECTOR1_1(SECTOR1_1) ;
                SECTOR1_1:standard_name = "depth" ;
                SECTOR1_1:long_name = "sector" ;
                SECTOR1_1:units = "level" ;
                SECTOR1_1:positive = "down" ;
                SECTOR1_1:axis = "Z" ;
                SECTOR1_1:point_spacing = "even" ;
        float ME(TIME, SECTOR1_1, Y20_200, X10_110) ;
                ME:long_name = "Meltwater production" ;
                ME:units = "mmWE/day" ;
                ME:_FillValue = -1.e+34f ;
                ME:missing_value = -1.e+34f ;
                ME:cell_methods = "TIME: mean" ;
                ME:history = "From ICE.q01.1950.01.01-05";

global attributes:
                :CDI = "Climate Data Interface version
2.0.5 (https://mpimet.mpg.de/cdi)" ;
                :Conventions = "CF-1.6" ;
                :institute = "University of Liège
(Belgium)" ;
```

Advanced features (1) : Compression and Chunking

1. Compression
   - Reduces file size using efficient algorithms (**zlib** in netCDF-4).
   - The idea behind *DEFLATE algorithm* is
     - LZ77 compression replaces repeated patterns in the data with shorter references
     - Huffman coding assigns shorter binary codes to more common patterns, making the data even smaller.
   - Trade-off: Smaller files but slightly slower read/write operations. Different compression levels are possible using the `nccopy` tool

2. Chunking
   - Data stored in small, **fixed-size blocks** (chunks) instead of a single continuous stream.
   - Efficient Indexing Mechanism:
     - Chunking uses a **hash table for indexing**, enabling fast lookup of chunks
     - Hash function maps the multi-dimensional indices (e.g., time, latitude, longitude) to the correct chunk
   - Chunking allows you to load only the specific parts of the data you need
     - Reduces memory usage
     - Faster I/O operations and reduced disk usage
     - Faster data access, thanks to efficient data indexing



Advanced features (2) : Efficient Data Access Methods

1. Accessing Subsets of Data
   - Use indexing to load specific dimensions or slices, minimizing memory usage.

```
Import xarray as xr

# Open file in read mode
with xr.open_dataset("example.nc") as ds:
    # Access variable and read specific time step
    melt = ds.["ME"].isel(time=0)  # First time slice
    print(melt)
```

2. Uses chunk-based hash table indexing for efficient retrieval
   - Data at specific coordinates is mapped to the correct chunk using a hash function

```
hash(TIME, LON, LAT) → Chunk ID
```

Advanced features (3) : Parallel File Handling

1. What is Parallel I/O?
   - Simultaneous access by multiple processes to different parts of the same file.
   - Enabled in netCDF-4 **through HDF5's MPI-IO** support.
     - Divide the dataset into chunks.
     - Assign each process to read/write specific chunks concurrently.
     - The complexity of MPI-IO is mostly abstracted
     - Supported in Fortran, C, C++, Python

2. Requirements
   - netCDF-4 compiled with parallel HDF5 support.
   - MPI library installed.

https://docs.unidata.ucar.edu/netcdf-c/current/parallel_io.html

https://docs.unidata.ucar.edu/nug/current/getting_and_building_netcdf.html#build_parallel

Advanced features (3) : Parallel File Handling - Python Example

1. MPI_example.py

```
from mpi4py import MPI
from netCDF4 import Dataset

# Initialize MPI
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

print(f"Process {rank} of {size} is writing to the file.")

# Create a parallel NetCDF file
ncfile = Dataset("MPI_write_example.nc", "w", parallel=True, comm=comm, info=MPI.Info())

ncfile.close()
comm.Barrier()  # Ensure all processes sync
```

2. Running (4 processes)

```
mpirun --mca opal_common_ucx_opal_mem_hooks 1 -np 4 python MPI_example.py
```

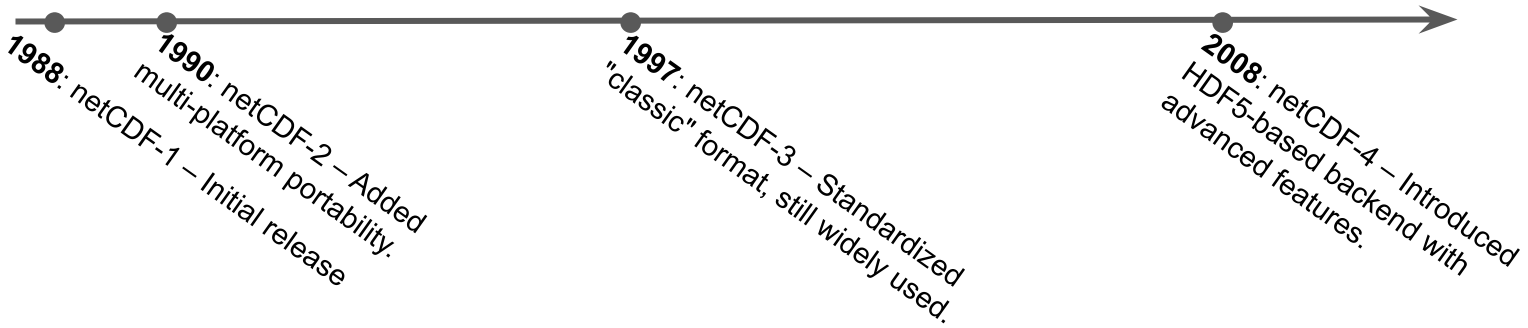
3. Output

```
Process 0 of 4 is writing to the file.
Process 1 of 4 is writing to the file.
Process 2 of 4 is writing to the file.
Process 3 of 4 is writing to the file.
```

Versions of netCDF

2008 : launch of netCDF4 :

- **Built on top of HDF5**
- Fully backward-compatible with netCDF-3
- Compression and Chunking: Improved storage efficiency and performance
- Parallel I/O: Optimized for HPC environments
- Unlimited Dimensions: Easier handling of dynamically growing datasets
- Improved Data Types: Support for complex numbers, unsigned integers, and strings
- Use `ncdump -k example.nc` (see next slides) to get the type



netCDF Reading and Writing in Python

→ module load netcdf4-python

1. Simplest form of netCDF reading:

```
from netCDF4 import Dataset

file = Dataset("example.nc", "r")
temperature = file.variables["temperature"][:]
print(temperature)

file.close()
```

2. Simplest form of netCDF writing:

```
from netCDF4 import Dataset
import numpy as np

file = Dataset("new_file.nc", "w", format="NETCDF4")

file.createDimension("time", None)  # Unlimited dimension
file.createDimension("lat", 10)
file.createDimension("lon", 10)

temp = file.createVariable("temperature", "f4", ("time", "lat", "lon"))
temp[0, :, :] = np.random.random((10, 10))

file.close()
```

Tools and Libraries for netCDF Data

1. Tools already installed on CECI clusters
   - **netCDF**: Data format and library for managing multi-dimensional scientific data.
   - **NCO**: Tools for manipulating and analyzing netCDF files.
   - **CDO**: Tools for climate and atmospheric data operations.
   - **ncview**: Visualization tool for netCDF data (need X11)

2. Loading modules in NIC5 (release **2021b** in Easybuild)

```
module load netCDF
module load NCO
module load CDO
module load ncview
```

(some redundancy among modules' functionalities)

Tools and Libraries for netCDF Data

netCDF examples

```
ncdump -h file.nc
      Show only the header (metadata).

ncdump -v varname file.nc # careful with this one
      Dumps specific variable content.

nccopy -k 4 file.nc file_nc4.nc
      Convert to NetCDF-4 format.

nccopy -d9 file.nc compressed_file.nc
      Compress a file (level 1-9). High compression
means high decompression time (I/O).

nccopy -c time/10,lat/360,lon/720 file.nc
chunked_file.nc
      Create a chunked dataset.
```

```
qglaude@nic5-login1 ~/formation_netcdf_hdf5 $ ncdump
-h MAR_ME_15km.nc
netcdf MAR_ME_15km {
dimensions:
        TIME = UNLIMITED ; // (151 currently)
        bnds = 2 ;
        X10_110 = 101 ;
        Y20_200 = 181 ;
        SECTOR1_1 = 1 ;
variables:
        float TIME(TIME) ;
                TIME:standard_name = "time" ;
                TIME:long_name = "time" ;
                TIME:bounds = "TIME_bnds" ;
                TIME:units = "days since 1947-09-01
00:00:00" ;
                TIME:calendar = "standard" ;
                TIME:axis = "T" ;
        double TIME_bnds(TIME, bnds) ;
        float X10_110(X10_110) ;
                X10_110:long_name = "x" ;
                X10_110:units = "km" ;
                X10_110:axis = "X" ;
        float Y20_200(Y20_200) ;
                Y20_200:long_name = "y" ;
                Y20_200:units = "km" ;
                Y20_200:axis = "Y" ;
        float SECTOR1_1(SECTOR1_1) ;
                SECTOR1_1:standard_name = "depth" ;
                SECTOR1_1:long_name = "sector" ;
                SECTOR1_1:units = "level" ;
                SECTOR1_1:positive = "down" ;
                SECTOR1_1:axis = "Z" ;
                SECTOR1_1:point_spacing = "even" ;
        float ME(TIME, SECTOR1_1, Y20_200, X10_110) ;
                ME:long_name = "Meltwater production"
;

                ME:units = "mmWE/day" ;
                ME:_FillValue = -1.e+34f ;
                ME:missing_value = -1.e+34f ;
                ME:cell_methods = "TIME: mean" ;
                ME:history = "From
ICE.q01.1950.01.01-05" ;
```

Tools and Libraries for netCDF Data

CDO examples

```
cdo sinfo file.nc
      Displays detailed file information.

cdo info file.nc
      Outputs metadata and variable stats.

cdo selvar,varname file.nc output.n
      Select specific variable(s).

cdo sellonlatbox,lon1,lon2,lat1,lat2 file.nc output.nc
      Subset data to a region.

cdo timmean file.nc output.nc
      Calculate the temporal mean.

cdo mergetime file1.nc file2.nc output.nc
      Merge time-sliced files.

cdo remapbil,gridfile file.nc output.nc
      Bilinear interpolation.

cdo genbil,gridfile file.nc weights.nc
      Generate interpolation weights.
```

```
qglaude@nic5-login1 ~/formation_netcdf_hdf5 $
cdo timmean MAR_ME_15km.nc average_melt.nc

cdo    timmean: Processed 2760431 values from 1
variable over 151 timesteps [0.03s 37MB].
```

```
dimensions:
        TIME = UNLIMITED ; // (1 currently)
        bnds = 2 ;
        X10_110 = 101 ;
        Y20_200 = 181 ;
        SECTOR1_1 = 1 ;
```

No man page → `cdo -h`

https://code.mpimet.mpg.de/projects/cdo/embedded/index.html

Tools and Libraries for netCDF Data

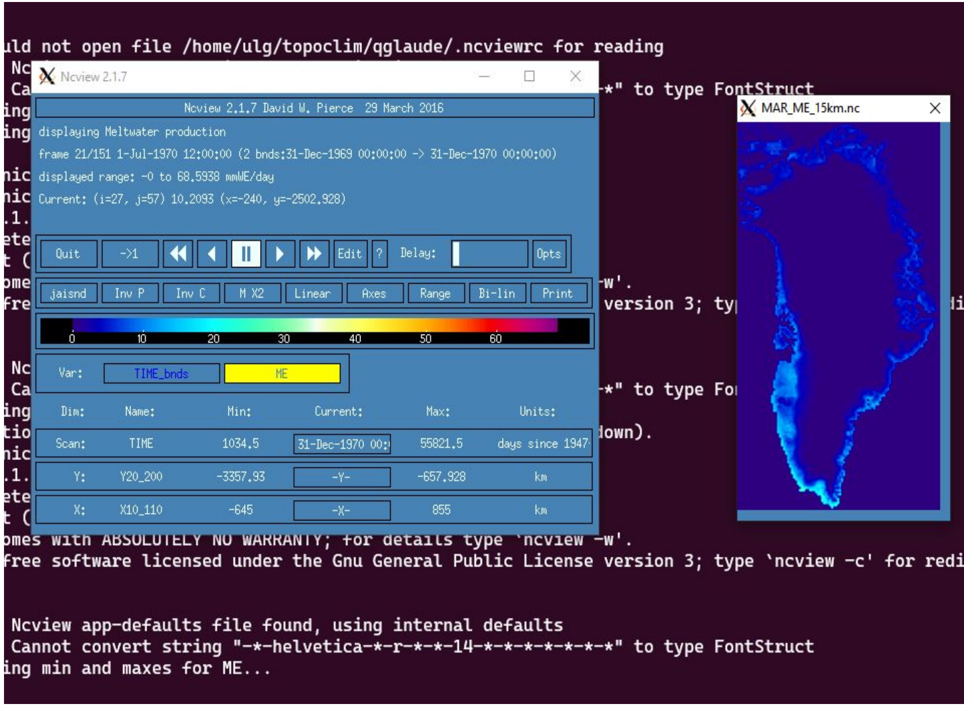
NCO examples - https://nco.sourceforge.net/nco.html (lots of examples !)

```
ncap2 -s 'new_var=var*10' input.nc output.nc
      Adds a new variable new_var which is 10 * var.

ncks -v varname file.nc output.nc
      Extract specific variable(s).

ncatted -a units,temperature,o,c,"K" input.nc
      Changes the units attribute of the temperature variable to K.

ncks -d lat,30.,60. -d lon,-10.,40. input.nc output.nc
      Extracts data within latitude 30–60 and longitude -10–40.

ncbo --add -v var1,var2 file1.nc file2.nc output.nc
      Add variables in file2.nc from file1.nc and stores the result.

ncra input1.nc input2.nc output.nc
      Averages variables across files.

ncrename -v old_varname,new_varname input.nc
      Renames a variable from old_name to new_name.
```

Tools and Libraries for netCDF Data

NCVIEW (Visualization for netCDF Data)

- Easy navigation through time-series data.
- Displays spatial data as color-filled plots.



Requirements - X11 Forwarding*

- Use MobaXTerm (recommended)

or

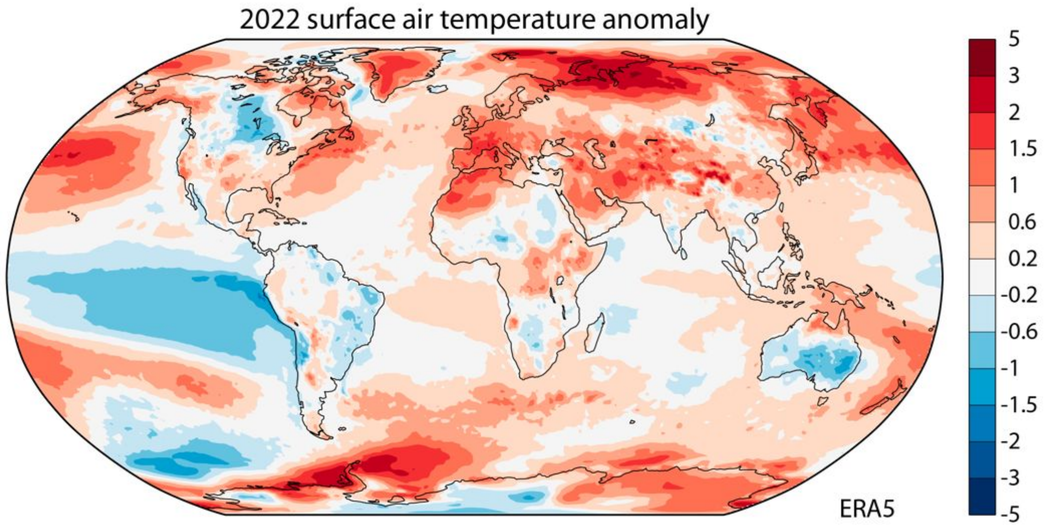
- Use WSL with X11 server (ex: XMing)
- Configure your shell for X11
  `export DISPLAY=127.0.0.1:0`
- Connect to the server using `-X` argument

*https://support.ceci-hpc.be/doc/_contents/QuickStart/ConnectingToTheClusters/WSL.html

Use Case - Climate Science with ERA5 Reanalysis

1. What is ERA5?
    - ERA5 is the fifth-generation atmospheric reanalysis produced by ECMWF (European Centre for Medium-Range Weather Forecasts).
    - Provides hourly estimates of atmospheric, land-surface, and oceanic parameters.
    - Widely used in climate modeling, weather forecasting, and research as data forcing
    - Python API to download data (`pip install cdsapi`)

2. Why netCDF
    - ERA5 data is distributed in netCDF format, making it portable, scalable, and easy to integrate into HPC workflows.
    - Supports efficient chunking, compression, and variable metadata for multi-dimensional data.



Copernicus Climate Change Service
Climate Indicators | 2022
PROGRAMME OF THE EUROPEAN UNION
Copernicus Europe's eyes on Earth
IMPLEMENTED BY ECMWF

Recap Slide : Key Advantages of netCDF

1. **Self-Describing Format**: Built-in metadata, encoded dimensions/variables/attributes in a structured format.
2. **Efficient Handling of Large Multi-Dimensional Data**: Optimized for fast and efficient input/output (I/O).
3. **Interoperability**: Supported by a wide range of programming languages, Software, and research teams.
4. **Scalability**: Handles datasets that grow dynamically.
5. **Compression and Chunking**: Reduces storage costs with efficient data access.
6. **Reproducibility and Collaboration**: Facilitates cross-disciplinary collaboration due to wide adoption
7. **Backward and Forward Compatibility**: And include new advanced features like HDF5 integration

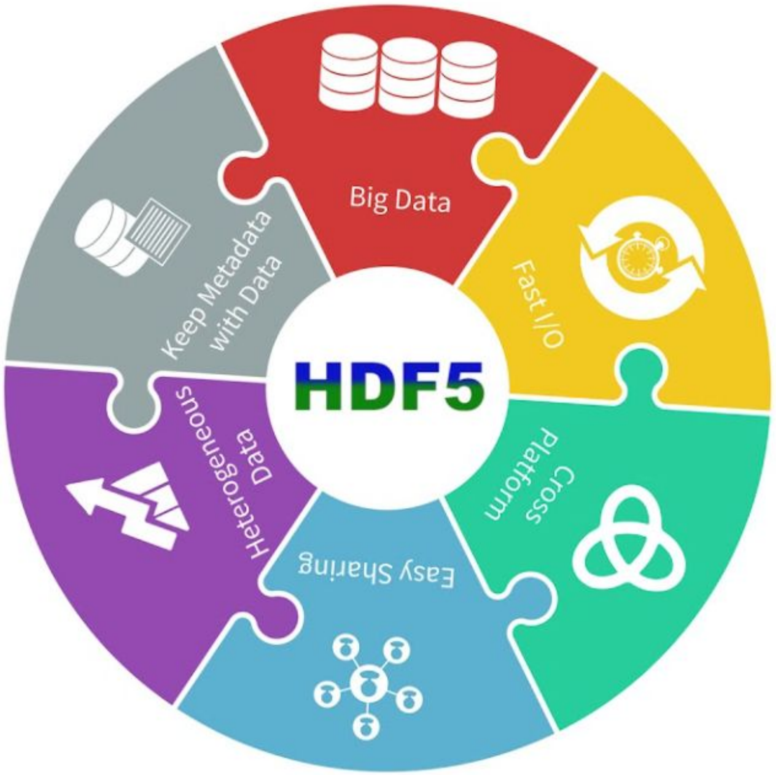
HDF5

Introduction to HDF5

1. HDF5 is the backbone of modern netCDF (version 4), with features such as
   - Compression
   - Chunking
   - Parallel I/O

2. Introduction to HDF5
   - Hierarchical Data Format
   - Versatile, portable file format designed to store and organize large, complex data efficiently
   - General-purpose

3. Common Use Cases:
   - Bioinformatics: sequencing results, 3D structures
   - Medical Imaging: CT scans, MRI
   - Particle Physics
   - Remote Sensing (NASA products)



Comparison with netCDF

1. Similarities:

    - Both are self-describing formats with embedded metadata for variables and dimensions.
    - Widely supported across operating systems and programming languages.

2. Key difference - Data model flexibility

    - HDF5 supports diverse data types, nested structures, and non-uniform data, making it ideal for general-purpose applications
      → HDF5 structure is **Hierarchical** (groups, datasets, attributes).

    - netCDF is specialized for structured, grid-based scientific data (e.g., climate, geoscience).
      → netCDF structure is **Flat** (dimensions, variables, attributes).

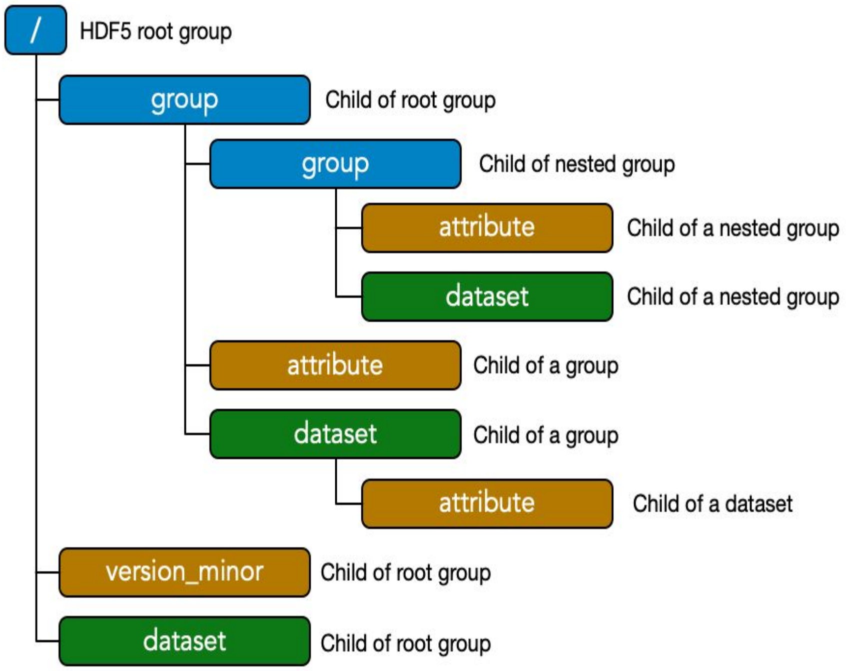
HDF5 Structure:

HDF5 organizes data into a tree-like structure

- **Groups**:
  - Containers that can hold datasets and other groups (like a folder)
- **Datasets**:
  Containers that hold the actual data (e.g., arrays, tables, images)
- **Attributes**:
  Metadata attached to groups or datasets.

Every HDF5 file starts with a single **root group** (/), serving as the top-level directory.

Groups can contain nested groups, datasets, and attributes, symbolic links, allowing for complex relationships between data elements



HDF5 Reading and Writing in Python

`→ module load SciPy-bundle`

1. Simplest form of netCDF reading:

```
import h5py

file = h5py.File("example.h5", "r")
dataset = file["/group/dataset_name"]
print(dataset [:])

file.close()
```

2. Simplest form of netCDF writing:

```
import h5py
import numpy as np

file = h5py.File("example.h5", "w")

group = file.create_group("group_name")

group.create_dataset("dataset_name", data=np.random.random((10, 10)))

group["dataset_name"].attrs["description"] = "Random data example"

file.close()
```

Tools and Libraries for HDF5 Data

1. Tools already installed on CECI clusters (or upon request)
    - **HDF5** : collection of command-line utilities for working with HDF5 files.
2. Loading modules in NIC5 (release **2021b** in Easybuild)

```
module load HDF5
```

3. Create a simple HDF5 file by yourself (on lemaitre4)

```
sbatch --profile=all YourSlurmFile.bash
```

4. Example

```
h5dump file.h5
      Dumps the entire file.

h5dump -g /group_name file.h5
      Dumps only a specific group.

h5dump -d /dataset_name file.h5
      Dumps the content of a dataset.
```

```
qglaude@nic5-login1 ~/formation_netcdf_hdf5 $ h5dump batch_lm4-w019.h5
HDF5 "batch_lm4-w019.h5" {
GROUP "/" {
   GROUP "lm4-w019" {
      ATTRIBUTE "CPUs per Task" {
         DATATYPE  H5T_STD_I32LE
         DATASPACE  SIMPLE { ( 1 ) / ( 1 ) }
         DATA {
         (0): 8
         }
      }
      GROUP "Tasks" {
         DATASET "0" {
            DATATYPE  H5T_COMPOUND {
               H5T_STD_U64LE "ElapsedTime";
               H5T_STD_U64LE "EpochTime";
               H5T_STD_U64LE "CPUFrequency";
               H5T_IEEE_F64LE "CPUTime";
               H5T_IEEE_F64LE "CPUUtilization";
               H5T_STD_U64LE "GPUMemMB";
               H5T_IEEE_F64LE "GPUUtilization";
               H5T_STD_U64LE "RSS";
               H5T_STD_U64LE "VMSize";
               H5T_STD_U64LE "Pages";
               H5T_IEEE_F64LE "ReadMB";
               H5T_IEEE_F64LE "WriteMB";
            }

[...]

            (96): {
                  2880,
                  1730393568,
                  21,
                  238.5,
                  795,
                  0,
                  0,
                  6306280,
                  7236076,
                  265,
                  16.4153,
                  2.92087
               },
            }
         }
      }
   }
}
```

Tools and Libraries for HDF5 Data

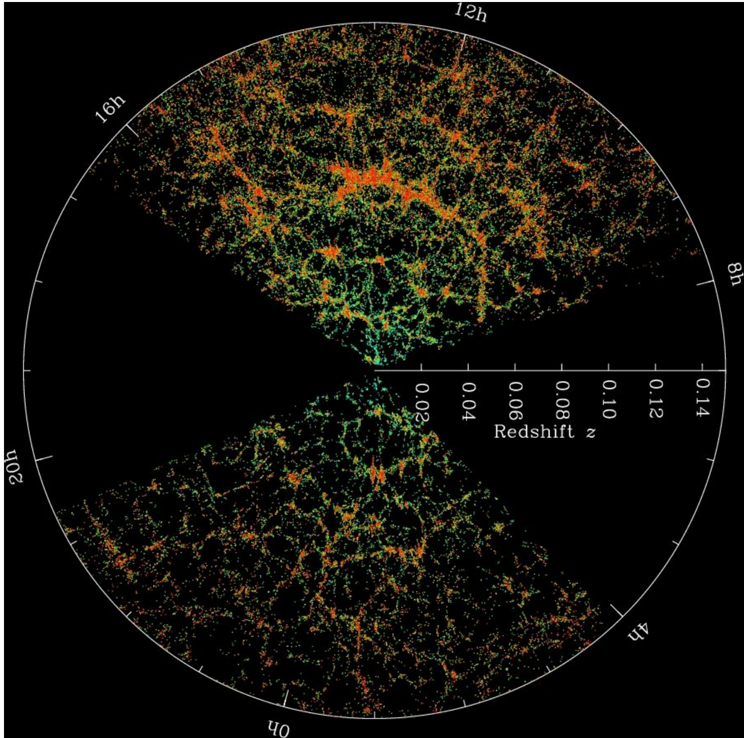
Other HDF5 examples - Don’t hesitate to explore the man pages

```
man h5dump
      Display the General Commands Manual of “h5dump”
```

```
h5ls -r file.h5
      Recursively lists all objects.

h5stat file.h5
      Summarizes statistics about an HDF5 file.

h5copy -i source.h5 -o dest.h5 -s /group1/dataset -d /group2/dataset
      Copies a dataset from one file to another.

h5repack -f GZIP=6 source.h5 dest.h5
      Compresses the file using GZIP with level 6 compression.

h5repack -c chunk[10x10] source.h5 dest.h5
      Rechunks datasets to use 10x10 chunks.

h5repart -f family source%05d.h5 single_file.h5
      Combines a family of files into one.

h5diff file1.h5 file2.h5
      Compares all objects in the files.
```

Use Case - SDSS Galaxy Datasets (Astronomy)

1. What is a “Sky Survey”?
   - Astronomy projects like SDSS (Sloan Digital Sky Survey) and LSST (Vera C. Rubin Observatory) generate massive datasets to map the universe.
   - Data includes:
     - High-resolution images.
     - Spectroscopic observations (> 4M obs).
     - Time-series data for transient events (e.g., supernovae, asteroid tracking).



2. Why HDF5?
   - Handles multi-modal data: Images (2D/3D arrays), Spectra (1D arrays), Object catalogs (tables with hundreds of columns).
   - Hierarchical organization (datasets are grouped by Observations, each having spectra, images, etc) in a Tree format
   - Parallel I/O, efficient Storage, etc

Interoperability Between netCDF and HDF5

1. Conversion Limits
    - **netCDF to HDF5** : Always possible as netCDF-4 files are natively HDF5-based
    - **HDF5 to netCDF** : Fails if
        - Hierarchical structures (e.g., nested groups) that cannot be flattened
        - Non-standard data types unsupported by netCDF (mixed data types)
        - Linked datasets or objects netCDF can’t represent (a dataset can exist in multiple groups)

2. Conversion Tools
    - Mostly rely on Python libraries like netCDF4 and h5py.

```python
from netCDF4 import Dataset
import h5py


nc = Dataset("file.nc", "r")
h5 = h5py.File("file.h5", "w")

for var_name in nc.variables:
    h5.create_dataset(var_name, data=nc.variables[var_name][:])

h5.close()
nc.close()
```

Other Formats Beyond netCDF and HDF5

1. Zarr
   - Structured format for storing multi-dimensional arrays.
   - Native support for cloud-based object storage (e.g., AWS S3).
   - Increasingly popular and well suited for distributed and cloud-native workflows

2. GeoTIFF
   - Self-described, embedded metadata about projections and spatial references.
   - Stores 2D raster datasets
   - Extremely used in remote sensing
   - GIS oriented

3. GeoPackage
   - Self-described
   - Can store raster (e.g., GeoTIFF) and vector data in a single compact file.
   - SQLite for database-like storage
   - GIS oriented

PART II

-

Hands-on

Fortran 90

C

C++

Python
(netCDF4 or xarray)

Fortran 90

C

C++

Python

(netCDF4 or xarray)

Extensive documentation with examples:
https://docs.unidata.ucar.edu/netcdf-fortran/current/f90_datasets.html

Search bar

Exist for other languages

NetCDF-Fortran 4.6.1

Main Page | Related Pages | Search

2 Datasets

2.1 Datasets Introduction

This chapter presents the interfaces of the netCDF functions that deal with a netCDF dataset or the whole netCDF library.

A netCDF dataset that has not yet been opened can only be referred to by its dataset name. Once a netCDF dataset is opened, it is referred to by a netCDF ID, which is a small nonnegative integer returned when you create or open the dataset. A netCDF ID is much like a file descriptor in C or a logical unit number in FORTRAN. In any single program, the netCDF IDs of distinct open netCDF datasets are distinct. A single netCDF dataset may be opened multiple times and will then have multiple distinct netCDF IDs; however at most one of the open instances of a single netCDF dataset should permit writing. When an open netCDF dataset is closed, the ID is no longer associated with a netCDF dataset.

Functions that deal with the netCDF library include:

- Get version of library.
- Get error message corresponding to a returned error code.

The operations supported on a netCDF dataset as a single object are:

- Create, given dataset name and whether to overwrite or not.
- Open for access, given dataset name and read or write intent.
- Put into define mode, to add dimensions, variables, or attributes.
- Take out of define mode, checking consistency of additions.
- Close, writing to disk if required.
- Inquire about the number of dimensions, number of variables, number of global attributes, and ID of the unlimited dimension, if any.
- Synchronize to disk to make sure it is current.
- Set and unset nofill mode for optimized sequential writes.
- After a summary of conventions used in describing the netCDF interfaces, the rest of this chapter presents a detailed description of the interfaces for

Table of Contents

Explanations and examples

0. Loading dependencies (release 2021b) :

```
Fortran :

    module load netCDF-Fortran/4.5.3-gompi-2021b

C       :

    module load netCDF/4.8.1-gompi-2021b

C++     :

    module load netCDF-C++4/4.3.1-gompi-2021b

Python  :

    module load xarray netcdf4-python
```

0. Compiling :

```
Fortran : gfortran your_program.f90 -o your_program
-I${EBROOTNETCDFMINFORTRAN}/include -L${EBROOTNETCDFMINFORTRAN}/lib
-lnetcdff

C       : gcc your_program.c -o your_program
-I${EBROOTNETCDF}/include -L${EBROOTNETCDF}/lib -lnetcdf

C++     : g++ your_program.cpp -o your_program
-I${EBROOTNETCDFMINCPLUSPLUS4}/include
-L${EBROOTNETCDFMINCPLUSPLUS4}/lib -lnetcdf_c++4 -lnetcdf
```

0. Running:

```
Fortran / C / C++ : ./your_program

Python            : python your_program.py
```

Goal : cover basics of netcdf files manipulation

Fortran / C / C++ :

- Fortran code is a given
- Try to adapt to C and C++ using the documentation

Python :

- The following exercices contains missing elements, and look-alike erroneous content
- Try to correct them using the documentation or your knowledge

Goal : cover basics of netcdf files manipulation

1. Open and close a NetCDF file
2. Inquire about NetCDF File Structure
3. Reading Data from a NetCDF File
4. Handling Multidimensional Data
5. Error Handling in NetCDF
6. Modifying the Content of a Variable
7. Creating a New Variable in a NetCDF File
8. Creating a New NetCDF File
9. Adding Descriptive Elements to a NetCDF File

Goal : cover basics of netcdf files manipulation

1. **>> Open and close a NetCDF file**
2. Inquire about NetCDF File Structure
3. Reading Data from a NetCDF File
4. Handling Multidimensional Data
5. Error Handling in NetCDF
6. Modifying the Content of a Variable
7. Creating a New Variable in a NetCDF File
8. Creating a New NetCDF File
9. Adding Descriptive Elements to a NetCDF File

1. Open and close a NetCDF file. (Fortran)

```
program open_netcdf

   use netcdf
   implicit none

   integer :: ncid, retval

   ! Open the NetCDF file
   retval = nf90_open('../input/MAR_ME_15km.nc', nf90_nowrite, ncid)

   if (retval /= nf90_noerr) then
      print *, 'Error: Unable to open the NetCDF file!'
      stop
   end if

   print *, 'NetCDF file opened successfully.'

   ! Close the file
   retval = nf90_close(ncid)
   if (retval /= nf90_noerr) then
      print *, 'Error: Unable to close  the NetCDF file!'
      stop
   end if

   print *, 'NetCDF file closed successfully.'

end program open_netcdf
```

1. Open and close a NetCDF file. (C)

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

int main() {
    int ncid, retval;

    // Open the NetCDF file
    retval = nc_open("../input/MAR_ME_15km.nc", NC_NOWRITE, &ncid);
    if (retval != NC_NOERR) {
        printf("Error: Unable to open the NetCDF file!\n");
        return -1;
    }

    printf("NetCDF file opened successfully.\n");

    // Close the file
    retval = nc_close(ncid);
    if (retval != NC_NOERR) {
        printf("Error: Unable to close the NetCDF file!\n");
        return -1;
    }

    printf("NetCDF file closed successfully.\n");
    return 0;
}
```

1. Open and close a NetCDF file. (C++)

```
#include <netcdf>
#include <iostream>

using namespace netCDF;
using namespace std;

int main() {
    try {
        // Open the NetCDF file
        NcFile dataFile("../input/MAR_ME_15km.nc", NcFile::read);
        cout << "NetCDF file opened successfully." << endl;

        // File automatically closes when NcFile object goes out of scope
    } catch (const exceptions::NcException& e) {
        cerr << "Error: " << e.what() << endl;
        return -1;
    }

    cout << "NetCDF file closed successfully." << endl;
    return 0;
}
```

1. Open and close a NetCDF file. (Python)

```
def open_netcdf(file_path):
    try:
        dataset = Dataset(file_path, mode="r")
        print("NetCDF file opened successfully.")
        dataset.close()
        print("NetCDF file closed successfully.")
    except Exception as e:
        print(f"Error: {e}")

if __name__ == "__main__":
    open_netcdf("../input/MAR_ME_15km.nc")
```

```
def open_netcdf(file_path):
    try:
        dataset = xr.open_dataset(file_path)
        print("NetCDF file opened successfully.")
        dataset.close()
        print("NetCDF file closed successfully.")
    except Exception as e:
        print(f"Error: {e}")

if __name__ == "__main__":
    open_netcdf("../input/MAR_ME_15km.nc")
```

Goal : cover basics of netcdf files manipulation

1. Open and close a NetCDF file
2. **>> Inquire about NetCDF File Structure**
3. Reading Data from a NetCDF File
4. Handling Multidimensional Data
5. Error Handling in NetCDF
6. Modifying the Content of a Variable
7. Creating a New Variable in a NetCDF File
8. Creating a New NetCDF File
9. Adding Descriptive Elements to a NetCDF File

2. Inquire about NetCDF File Structure (Fortran)

```
program inquire_netcdf
    use netcdf
    implicit none

    integer :: ncid, ndims, nvars, natts, unlimdimid, retval

    ! Open the NetCDF file
    retval = nf90_open('../input/MAR_ME_15km.nc', nf90_nowrite, ncid)

    ! Inquire about file structure
    retval = nf90_inquire(ncid, ndims, nvars, natts, unlimdimid)

    if (retval /= nf90_noerr) then
        print *, 'Error: Unable to inquire about the NetCDF file!'
        stop
    end if

    ! Print the file structure
    print *, 'Number of dimensions:', ndims
    print *, 'Number of variables:', nvars
    print *, 'Number of attributes:', natts
    print *, 'Unlimited dimension ID:', unlimdimid

    ! Close the NetCDF file
    retval = nf90_close(ncid)

end program inquire_netcdf
```

NB: from now on, I won't include file opening and closing verification for conciseness (best advice is to keep them for debug)

2. Inquire about NetCDF File Structure (Python - netCDF4)

```
import netCDF4 as nc

def nc_struct():
    try:
        # Open the NetCDF file
        dataset = nc.Dataset('../input/MAR_ME_15km.nc', 'r')

        # Inquire about file structure
        ndims = len(dataset.dimensions)
        nvars = len(dataset.variables)
        natts = len(dataset.ncattrs())
        unlimdim = dataset.dimensions.get(None, None)

        # Print the file structure
        print("Number of dimensions:", ndims)
        print("Number of variables:", nvars)
        print("Number of attributes:", natts)
        print("Unlimited dimension ID:", unlimdim if unlimdim else "None")

        # Close the file
        dataset.close()
    except Exception as e:
        print(f"Error: {e}")

if __name__ == "__main__":
    nc_struct()
```

NB: from now on, I won't include file opening and closing verification for conciseness (best advice is to keep them for debug)

2. Inquire about NetCDF File Structure (Python - xarray)

```python
import xarray as xr

def nc_struct():
    try:
        # Open the NetCDF file
        dataset = xr.open_dataset('../input/MAR_ME_15km.nc')

        # Inquire about file structure
        ndims = len(dataset.dims)
        nvars = len(dataset.data_vars)
        natts = len(dataset.attrs)
        unlimdim = [dim for dim, size in dataset.dims.items() if size == float('inf')]

        # Print the file structure
        print("Number of dimensions:", ndims)
        print("Number of variables:", nvars)
        print("Number of attributes:", natts)
        print("Unlimited dimension:", unlimdim if unlimdim else "None")

        # Close the file
        dataset.close()
    except Exception as e:
        print(f"Error: {e}")

if __name__ == "__main__":
    nc_struct()
```

NB: from now on, I won't include file opening and closing verification for conciseness (best advice is to keep them for debug)

2. netCDF4 and xarray propositions are giving 2 different outputs : an idea why ?

- netCDF4 :

```
Number of dimensions: 5
Number of variables: 6
Number of attributes: 8
Unlimited dimension: None
```

- Xarray :

```
Number of dimensions: 5
Number of variables: 2
Number of attributes: 8
Unlimited dimension: None
```

Goal : cover basics of netcdf files manipulation

1. Open and close a NetCDF file
2. Inquire about NetCDF File Structure
3. **>> Reading Data from a NetCDF File**
4. Handling Multidimensional Data
5. Error Handling in NetCDF
6. Modifying the Content of a Variable
7. Creating a New Variable in a NetCDF File
8. Creating a New NetCDF File
9. Adding Descriptive Elements to a NetCDF File

3. Reading Data from a NetCDF File (Fortran90)

```
program read_time_variable

   use netcdf
   implicit none

   integer :: ncid, varid, dimid, retval
   integer, dimension(:), allocatable :: dimids
   integer :: ndims, time_len
   real, dimension(:), allocatable :: time_data

   ! Open the NetCDF file
   retval = nf90_open('../input/MAR_ME_15km.nc',
nf90_nowrite, ncid)

   ! Get the varid of 'TIME'
   retval = nf90_inq_varid(ncid, 'TIME', varid)

   ! Get the number of dimensions and their IDs for 'TIME'
   retval = nf90_inquire_variable(ncid, varid, ndims = ndims)

   allocate(dimids(ndims))

   retval = nf90_inquire_variable(ncid, varid, dimids =
dimids)

[...]
```

```
[...]

   ! Retrieve the length of the 'TIME' dimension
   dimid = dimids(1)  ! 'TIME' is associated with the first
dimension
   retval = nf90_inquire_dimension(ncid, dimid, len =
time_len)

   ! Allocate memory for TIME data
   allocate(time_data(time_len))

   ! Read the 'TIME' variable
   retval = nf90_get_var(ncid, varid, time_data)

   ! Display the TIME data
   print *, 'TIME data:'
   print *, time_data

   ! Close the NetCDF file
   retval = nf90_close(ncid)

end program read_time_variable
```

3. Reading Data from a NetCDF File (Python)

```
import netCDF4 as nc

def read_data():
    # Open the NetCDF file
    dataset = nc.Dataset('../input/MAR_ME_15km.nc', 'r')

    # Read the 'TIME' variable
    time_data = dataset.variables['TIME_DATA'][:]

    # Display the TIME data
    print("TIME data:")
    print(time_data)

if __name__ == "__main__":
    read_data()
```

```
import xarray as xr

def read_data():
    # Open the NetCDF file
    dataset = xr.open_dataset('MAR_ME_15km.nc')

    # Read the 'TIME' variable
    time_data = dataset['TIME_DATA'].values

    # Display the TIME data
    print("TIME data:")
    print(time_data)

if __name__ == "__main__":
    read_data()
```

Goal : cover basics of netcdf files manipulation

1. Open and close a NetCDF file
2. Inquire about NetCDF File Structure
3. Reading Data from a NetCDF File
4. **>> Handling Multidimensional Data**
5. Error Handling in NetCDF
6. Modifying the Content of a Variable
7. Creating a New Variable in a NetCDF File
8. Creating a New NetCDF File
9. Adding Descriptive Elements to a NetCDF File

4. Handling Multidimensional Data (Fortran90)

```
program display_ME_variable

   use netcdf
   implicit none

   integer :: ncid, varid_me, retval
   integer, dimension(4) :: dimids
   integer :: time_len, sector_len, y_len, x_len
   real, dimension(:,:,:,:), allocatable :: me_data

   ! Open the NetCDF file in read-only mode
   retval = nf90_open('../input/MAR_ME_15km.nc', nf90_nowrite, ncid)

   ! Get the variable ID of 'ME'
   retval = nf90_inq_varid(ncid, 'ME', varid_me)

   ! Get the dimensions of 'ME'
   retval = nf90_inquire_variable(ncid, varid_me, dimids = dimids)

   ! Retrieve dimension lengths
   retval = nf90_inquire_dimension(ncid, dimids(1), len = time_len)
   retval = nf90_inquire_dimension(ncid, dimids(2), len = sector_len)
   retval = nf90_inquire_dimension(ncid, dimids(3), len = y_len)
   retval = nf90_inquire_dimension(ncid, dimids(4), len = x_len)

   ! Allocate memory for ME data
   allocate(me_data(time_len, sector_len, y_len, x_len))

[...]
```

```
[...]

   ! Read the 'ME' variable
   retval = nf90_get_var(ncid, varid_me, me_data)

   ! Display a subset of the ME variable (time step 1, sector 1)
   print *, "ME data (time step 1, sector 1):"
   print *, me_data(1, 1, :, :)

   ! Close the NetCDF file
   retval = nf90_close(ncid)

   print *, "ME variable displayed successfully."

end program display_ME_variable
```

4. Handling Multidimensional Data (Python)

```python
import netCDF4 as nc

def read_me_data():
    # Open the NetCDF file
    dataset = nc.Dataset('../input/MAR_ME_15km.nc', 'r')

    # Read the 'ME' variable
    me_data = dataset.variables['ME'][:]

    # Display a subset of ME data
    print("ME data (time step 1, sector 2):")
    print(me_data[0, 1, :, :])

if __name__ == "__main__":
    read_me_data()
```

```python
import xarray as xr

def read_me_data():
    # Open the NetCDF file
    dataset = xr.open_dataset('../input/MAR_ME_15km.nc')

    # Read the 'ME' variable
    me_data = dataset['ME']

    # Display a subset of ME data
    print("ME data (time step 1, sector 2):")
    print(me_data.isel(time=0, sector=1).values)

if __name__ == "__main__":
    read_me_data()
```

Goal : cover basics of netcdf files manipulation

1. Open and close a NetCDF file
2. Inquire about NetCDF File Structure
3. Reading Data from a NetCDF File
4. Handling Multidimensional Data
5. **>> Error Handling in NetCDF**
6. Modifying the Content of a Variable
7. Creating a New Variable in a NetCDF File
8. Creating a New NetCDF File
9. Adding Descriptive Elements to a NetCDF File

5. Error Handling in NetCDF (Fortran90)

```
program error_handling
   use netcdf
   implicit none

   integer :: ncid, varid, retval

   ! Open the NetCDF file
   retval = nf90_open('../input/MAR_ME_15km.nc', nf90_nowrite, ncid)
   call check_error(retval, 'Error during opening file')

   ! Get the variable ID of 'ME'
   retval = nf90_inq_varid(ncid, 'ME', varid)
   call check_error(retval, 'Error accessing the ME variable')

   ! Close the NetCDF file
   retval = nf90_close(ncid)
   call check_error(retval, 'Error closing the file')

   contains
      subroutine check_error(retval, error_message)
         integer, intent(in) :: retval
         character(len=*), intent(in) :: error_message

         if (retval /= nf90_noerr) then
            print *, error_message
            stop
         end if
      end subroutine check_error

end program error_handling
```

5. Error Handling in NetCDF (Python - netCDF4)

```
from netCDF4 import Dataset
import sys

def check_error(success, message):
    if not success:
        print(message)
        sys.exit(1)

def error_handling():
    try:
        # Open the NetCDF file
        file_path = "../input/MAR_ME_15km.nc"
        try:
            dataset = Dataset(file_path, "r")
            print("File opened successfully.")
        catch Exception as e:
            check_error(False, f"Error during opening file: {e}")

        # Access the variable 'ME'
        variable_name = "ME"
        try:
            var = dataset.variables[variable_name]
            print(f"Variable '{variable_name}' accessed successfully.")
        catch KeyError:
            check_error(False, f"Error accessing the variable '{variable_name}'")

        # Close the NetCDF file
        try:
            dataset.close()
            print("File closed successfully.")
        catch Exception as e:
            check_error(False, f"Error during file closing: {e}")

    catch Exception as e:
        print(f"An unexpected error occurred: {e}")
        sys.exit(1)

if __name__ == "__main__":
    error_handling()
```

5. Error Handling in NetCDF (Python - xarray)

```
import xarray as xr
import sys

def check_error(success, message):
    if not success:
        print(message)
        sys.exit(1)

def error_handling():
    try:
        # Open the NetCDF file
        file_path = "../input/MAR_ME_15km.nc"
        try:
            dataset = xr.open_dataset(file_path)
            print("File opened successfully.")
        catch Exception as e:
            check_error(False, f"Error during opening file: {e}")

        # Access the variable 'ME'
        variable_name = "ME"
        try:
            if variable_name in dataset:
                var = dataset[variable_name]
                print(f"Variable '{variable_name}' accessed successfully.")
            else:
                raise KeyError(f"Variable '{variable_name}' not found.")
        catch KeyError as e:
            check_error(False, f"Error accessing the variable '{variable_name}': {e}")

        # Close the NetCDF file
        try:
            dataset.close()
            print("File closed successfully.")
        catch Exception as e:
            check_error(False, f"Error during file closing: {e}")

    catch Exception as e:
        print(f"An unexpected error occurred: {e}")
        sys.exit(1)

if __name__ == "__main__":
    error_handling()
```

Goal : cover basics of netcdf files manipulation

1. Open and close a NetCDF file
2. Inquire about NetCDF File Structure
3. Reading Data from a NetCDF File
4. Handling Multidimensional Data
5. Error Handling in NetCDF
6. **>> Modifying the Content of a Variable**
7. Creating a New Variable in a NetCDF File
8. Creating a New NetCDF File
9. Adding Descriptive Elements to a NetCDF File

6. Modifying the Content of a Variable (Fortran)

```
program modify_ME_variable

   use netcdf
   implicit none

   integer :: ncid, varid, retval
   integer, dimension(4) :: dimids
   integer :: time_len, sector_len, y_len, x_len
   real, dimension(:,:,:,:), allocatable :: me_data

   ! Open the NetCDF file in write mode
   retval = nf90_open('../input/MAR_ME_15km.nc', nf90_write, ncid)

   ! Get the variable ID of 'ME'
   retval = nf90_inq_varid(ncid, 'ME', varid)

   ! Get the dimensions of 'ME'
   retval = nf90_inquire_variable(ncid, varid, dimids = dimids)

   ! Retrieve dimension lengths
   retval = nf90_inquire_dimension(ncid, dimids(1), len = time_len)
   retval = nf90_inquire_dimension(ncid, dimids(2), len = sector_len)
   retval = nf90_inquire_dimension(ncid, dimids(3), len = y_len)
   retval = nf90_inquire_dimension(ncid, dimids(4), len = x_len)

   ! Allocate memory for ME data
   allocate(me_data(time_len, sector_len, y_len, x_len))

   ! Read the 'ME' variable
   retval = nf90_get_var(ncid, varid, me_data)

   ! Convert yearly values to daily values (divide by 365)
   me_data = me_data / 365.0

   ! Write the modified 'ME' variable back to the file
   retval = nf90_put_var(ncid, varid, me_data)

   ! Close the NetCDF file
   retval = nf90_close(ncid)

   print *, "Content of 'ME' variable modified successfully."

end program modify_ME_variable
```

6. Modifying the Content of a Variable (Python)

```
import netCDF4 as nc

def modify_variable():
    # Open the NetCDF file in write mode
    dataset = nc.Dataset('../input/MAR_ME_15km.nc', 'r+')

    # Access the 'ME' variable
    me_data = dataset.variables['ME'][:]

    # Convert yearly values to daily
    me_data /= 365.0

    # Close the file
    dataset.close()

if __name__ == "__main__":
    modify_variable()
```

```
import xarray as xr

def modify_variable():
    # Open the NetCDF file in write mode
    dataset = xr.open_dataset('../input/MAR_ME_15km.nc', mode='a')

    # Access the 'ME' variable
    me_data = dataset['ME']

    # Convert yearly values to daily
    me_data /= 365.0

    # Close the file
    dataset.close()

if __name__ == "__main__":
    modify_variable()
```

Goal : cover basics of netcdf files manipulation

1. Open and close a NetCDF file
2. Inquire about NetCDF File Structure
3. Reading Data from a NetCDF File
4. Handling Multidimensional Data
5. Error Handling in NetCDF
6. Modifying the Content of a Variable
7. **>> Creating a New Variable in a NetCDF File**
8. Creating a New NetCDF File
9. Adding Descriptive Elements to a NetCDF File

7. Creating a New Variable in a NetCDF File (Fortran90)

```
program create_ME_daily_variable

   use netcdf
   implicit none

   integer :: ncid, varid_me, varid_daily, retval
   integer, dimension(4) :: dimids
   integer :: time_len, sector_len, y_len, x_len
   real, dimension(:,:,:,:), allocatable :: me_data, daily_data

   ! Open the NetCDF file in write mode
   retval = nf90_open('../input/MAR_ME_15km.nc', nf90_write, ncid)

   ! Get the variable ID of 'ME'
   retval = nf90_inq_varid(ncid, 'ME', varid_me)

   ! Get the dimensions of 'ME'
   retval = nf90_inquire_variable(ncid, varid_me, dimids = dimids)

   ! Retrieve dimension lengths
   retval = nf90_inquire_dimension(ncid, dimids(1), len = time_len)
   retval = nf90_inquire_dimension(ncid, dimids(2), len = sector_len)
   retval = nf90_inquire_dimension(ncid, dimids(3), len = y_len)
   retval = nf90_inquire_dimension(ncid, dimids(4), len = x_len)

   ! Allocate memory for ME data and the new daily variable
   allocate(me_data(time_len, sector_len, y_len, x_len))
   allocate(daily_data(time_len, sector_len, y_len, x_len))

[...]
```

```
[...]

   ! Read the 'ME' variable
   retval = nf90_get_var(ncid, varid_me, me_data)

   ! Compute the daily variable
   daily_data = me_data / 365.0

   ! Switch to define mode to create a new variable
   retval = nf90_redef(ncid)

   ! Define a new variable 'ME_DAILY'
   retval = nf90_def_var(ncid, 'ME_DAILY', nf90_float, dimids, varid_daily)

   ! End define mode
   retval = nf90_enddef(ncid)

   ! Write the daily data to the new variable
   retval = nf90_put_var(ncid, varid_daily, daily_data)

   ! Close the NetCDF file
   retval = nf90_close(ncid)

   print *, "ME_DAILY variable created successfully."

end program create_ME_daily_variable
```

7. Creating a New Variable in a NetCDF File (Python)

```
import netCDF4 as nc

def new_variable():
    # Open the NetCDF file in write mode
    dataset = nc.Dataset('../input/MAR_ME_15km.nc', 'r+')

    # Read the 'ME' variable
    me_data = dataset.variables['ME'][:]

    # Compute the daily variable
    daily_data = me_data / 365.0

    # Define the new variable 'ME_DAILY'
    dataset.createVariable('ME_DAILY', 'f4')
    dataset.variables['ME_DAILY'][:] = daily_data

    # Close the file
    dataset.close()

if __name__ == "__main__":
    new_variable()
```

```
import xarray as xr

def new_variable():
    # Open the NetCDF file in write mode
    dataset = xr.open_dataset('../input/MAR_ME_15km.nc', mode='a')

    # Compute the daily variable
    daily_data = dataset['ME'] / 365.0

    # Add the new variable 'ME_DAILY'
    dataset['ME_DAILY'] = (('time', 'sector', 'y', 'x'), daily_data.values)

    # Save changes
    dataset.to_netcdf("../input/MAR_ME_15km.nc")

    # Close the file
    dataset.close()

if __name__ == "__main__":
    new_variable()
```

Goal : cover basics of netcdf files manipulation

1. Open and close a NetCDF file
2. Inquire about NetCDF File Structure
3. Reading Data from a NetCDF File
4. Handling Multidimensional Data
5. Error Handling in NetCDF
6. Modifying the Content of a Variable
7. Creating a New Variable in a NetCDF File
8. **>> Creating a New NetCDF File**
9. Adding Descriptive Elements to a NetCDF File

8. Creating a New NetCDF File (Fortran90)

```
program create_new_netcdf

   use netcdf
   implicit none

   integer :: ncid, varid, retval
   integer :: dimids(4)  ! Dimension IDs
   integer :: time_len, sector_len, y_len, x_len
   real, dimension(:,:,:,:), allocatable :: daily_data

   ! Define dimensions
   time_len = 10
   sector_len = 2
   y_len = 50
   x_len = 50

   ! Allocate memory for data
   allocate(daily_data(time_len, sector_len, y_len, x_len))
   daily_data = 1.0 / 365.0  ! Fill with example daily data

   ! Create a new NetCDF file
   retval = nf90_create('new_daily_data.nc', nf90_clobber, ncid)

   ! Define dimensions
   retval = nf90_def_dim(ncid, 'TIME', time_len, dimids(1))
   retval = nf90_def_dim(ncid, 'SECTOR', sector_len, dimids(2))
   retval = nf90_def_dim(ncid, 'Y', y_len, dimids(3))
   retval = nf90_def_dim(ncid, 'X', x_len, dimids(4))

[...]
```

```
[...]

   ! Define a new variable
   retval = nf90_def_var(ncid, 'ME_DAILY', nf90_real, dimids, varid)

   ! End define mode
   retval = nf90_enddef(ncid)

   ! Write data to the new variable
   retval = nf90_put_var(ncid, varid, daily_data)

   ! Close the NetCDF file
   retval = nf90_close(ncid)

   print *, 'NetCDF file created successfully.'

end program create_new_netcdf
```

8. Creating a New NetCDF File (Python)

```
import netCDF4 as nc
import numpy as np

def main():
    dataset = nc.Dataset('new_daily_data.nc', 'w', format='NETCDF4')

    time_len, sector_len, y_len, x_len = 10, 2, 50, 50
    dataset.createDimension('TIME', time_len)
    dataset.createDimension('SECTOR', sector_len)
    dataset.createDimension('Y', y_len)
    dataset.createDimension('X', x_len)

    # Define a new variable
    daily_var = dataset.createVariable('ME_DAILY', 'f4', ('X', 'Y',
'SECTOR', 'TIME'))

    daily_data = np.full((time_len, sector_len, y_len, x_len), 1.0 /
365.0, dtype='float32')
    daily_var[:] = daily_data

    dataset.close()

if __name__ == "__main__":
    main()
```

```
import xarray as xr
import numpy as np

def main():
    # Define dimensions
    time_len, sector_len, y_len, x_len = 10, 2, 50, 50
    daily_data = np.full((x_len, y_len, sector_len, time_len), 1.0 / 365.0,
dtype='float32')

    # Create a new NetCDF file
    dataset = xr.Dataset(
        data_vars={"ME_DAILY": (("X", "Y", "SECTOR", "TIME"), daily_data)},
# Reversed dimensions
        coords={
            "TIME": range(time_len),
            "SECTOR": range(sector_len),
            "Y": range(y_len),
            "X": range(x_len)
        }
    )

    # Write to file
    dataset.to_netcdf("new_daily_data.nc")

if __name__ == "__main__":
    main()
```

Goal : cover basics of netcdf files manipulation

1. Open and close a NetCDF file
2. Inquire about NetCDF File Structure
3. Reading Data from a NetCDF File
4. Handling Multidimensional Data
5. Error Handling in NetCDF
6. Modifying the Content of a Variable
7. Creating a New Variable in a NetCDF File
8. Creating a New NetCDF File
9. **>> Adding Descriptive Elements to a NetCDF File**

9. Adding Descriptive Elements to a NetCDF File (Fortran)

```
program add_attribute

   use netcdf
   implicit none

   integer :: ncid, varid, retval

   ! Open the NetCDF file in write mode
   retval = nf90_open('new_daily_data.nc', nf90_write, ncid)

   ! Switch to define mode to add attributes
   retval = nf90_redef(ncid)

   ! Get the variable ID of 'ME_DAILY'
   retval = nf90_inq_varid(ncid, 'ME_DAILY', varid)

   ! Add a global attribute
   retval = nf90_put_att(ncid, nf90_global, 'title', 'Daily melt data')

   ! Add variable-specific attributes
   retval = nf90_put_att(ncid, varid, 'units', 'mm/day')
   retval = nf90_put_att(ncid, varid, 'description', 'Daily surface melt data derived from yearly values')

   ! Leave define mode
   retval = nf90_enddef(ncid)

   ! Close the NetCDF file
   retval = nf90_close(ncid)

   print *, "Attributes added successfully."

end program add_attribute
```

https://docs.unidata.ucar.edu/netcdf-fortran/current/f90-attributes.html

9. Adding Descriptive Elements to a NetCDF File (Python)

```
import netCDF4 as nc

def main():
    # Open the NetCDF file in write mode
    dataset = nc.Dataset('new_daily_data.nc', 'r')

    # Add a global attribute
    dataset.title = "Daily melt data"

    # Add variable-specific attributes
    daily_var = dataset.variables['ME_DAILY']
    daily_var.units = "mm/day"
    daily_var.description = "Daily surface melt data derived from 
yearly values"

    # Close the file
    dataset.close()

if __name__ == "__main__":
    main()
```

```
import xarray as xr

def main():
    # Open the NetCDF file in write mode
    dataset = xr.open_dataset('new_daily_data.nc')

    # Add a global attribute
    dataset.attrs['title'] = "Daily melt data"

    # Add variable-specific attributes
    dataset['ME_DAILY'].attrs['units'] = "mm/day"
    dataset['ME_DAILY'].attrs['description'] = "Daily surface melt data 
derived from yearly values"

    # Save and close the file
    dataset.to_netcdf("new_daily_data.nc")

if __name__ == "__main__":
    main()
```

Well done !

1. ~~Open and close a NetCDF file~~
2. ~~Inquire about NetCDF File Structure~~
3. ~~Reading Data from a NetCDF File~~
4. ~~Handling Multidimensional Data~~
5. ~~Error Handling in NetCDF~~
6. ~~Modifying the Content of a Variable~~
7. ~~Creating a New Variable in a NetCDF File~~
8. ~~Creating a New NetCDF File~~
9. ~~Adding Descriptive Elements to a NetCDF File~~

netCDF and HDF5 file formats on CÉCI clusters

Thank You !

Quentin GLAUDE: quentin.glaude@uliege.be
28th November 2024, CYCL09b - Louvain-La-Neuve