ALCS Advanced Data Training Course
5/11/2019

NCI AUSTRALIA

Introduction to data formats

Jingbo Wang, Rui Yang, Nigel Rees

NCRIS National Research Infrastructure for Australia
An Australian Government Initiative

Australian Government Bureau of Meteorology

Australian Government Geoscience Australia

Australian Government Australian Research Council

CSIRO

Australian National University

nci.org.au
@NCInews

NCI data collections data formats

netCDF (climate, weather, earth observation)

- classic
- netCDF-3 64-bit offset
- netCDF-4 classic
- netCDF-4

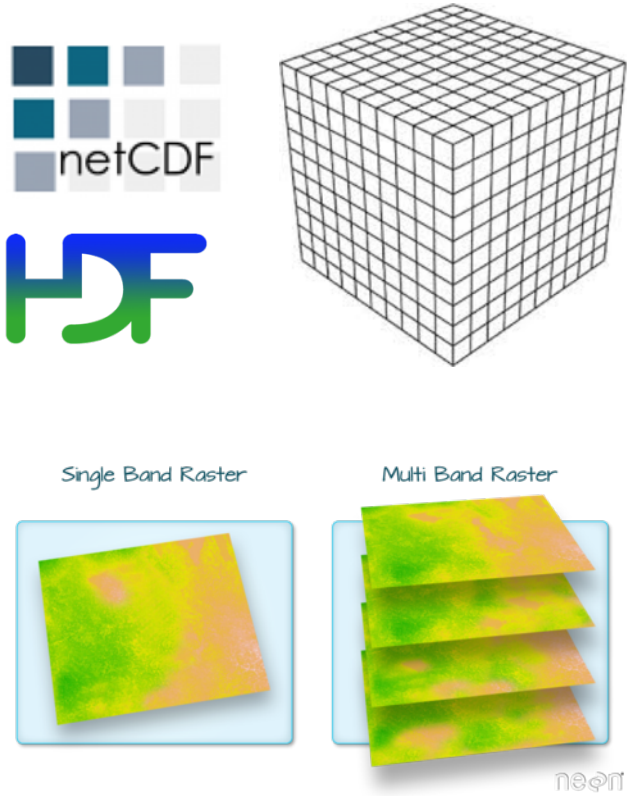
Non-netCDF

- FITS images files (astronomy)
- FASTQ SAM/BAM format (bioinformatics)
- Baglt-0.96 (with PKZIP), TIFF, CSV, PNG, FPF (biosciences)
- ASCII XYZ, CARIS HIPS & SIPS 7 ver1-2 (bathymetry and elevation)
- LAS, ESRI Grid, ASCII grid (elevation)
- avi, mpg, DVD, jpg (Marine Video and Imagery)
- Tiff and JPEG (Aerial Survey)
- ASCII (geological model, natural hazards)
- Segy, ASCII, ErMapper ERS (geophysics)

Types of data formats

Range of formats:

- NetCDF/HDF5
- GeoTIFF
- GRIB
- ...
- (Proprietary formats)



What is netCDF?

- NetCDF is a data storage format commonly used within the geoscience community. The acronym stands for Network Common Data Format and it refers to a "set of software libraries and self-describing, machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data"[1].The software is maintained by developers at Unidata, which is a subsidiary of UCAR.

- One of the main advantages of using NetCDF is that it has a built-in hierarchical structure that facilitates better organization and documentation of data. It is well suited to handle large numerical datasets as it allows users to access portions of a dataset without loading its entirety into memory.

Data Collections with netCDF formats

Data Collections	NCI project codes
CMIP6 replicated	oi10
CMIP5 Australian published / replicated	rr3 / al33
CMIP3 entire data collection	cb20
CORDEX	rr3, al33
Input4MIPs include JRA55-do (forcing for ocean/sea-ice models)	qv56
ERA-Interim (6hrly data)	ub4
ACCESS models (BoM)	rr3, rr4, ua4, ja4, gg6, gg8, fx1, fx3, ub3
LANDSAT, Copernicus, WOfS	rs0, fj7, fk4
MODIS, VIIRS, AVHRR, OFAM, BARN, Soil Moisture, eMAST	u39, gb6, fj4, rr9
BoM Seasonal Climate POAMA2/3	rr8, ub7
BoM Observations	rr5
BoM Ocean-Marine	rr6
BoM Key Water Assets	fj8
ARCCSS/CLEX collections, including ERA-Interim (6hrly)	ua8, ub4
OGCM (ES), YOTC, CABLE	rq5, rq7, wd9

Why do we care about data formats?

- Amount of Tbyte scale data available worldwide
- Modern workflows use supercomputer and the number of files is an I/O bottleneck
- New approaches should not be limited by data formats that were developed for other applications
- Standards - should follow to allow adaptively different group of users and collaborations among different disciplines.
- Reproducibility - provenance information storage within the data

We need a consistent approach across all different types of data that is easy to use for anyone who wants to do data fusion.

Traditional domain specific data formats	Self described; web service enabled format

- High performance
- Hierarchical structure
- Rich metadata and self-described CF convention using Global attribute + variables
- Data fusion combine different data type
- Web services enabled

Standards enhanced (e.g., ISO19115, CF, GCMD)

Formats make performance difference

The performance of these workflows would be increased if the data was stored by combining all time series into one file and taking advantage of parallel processing capabilities.

e.g. a workflow regularly touch a couple of millions of traces

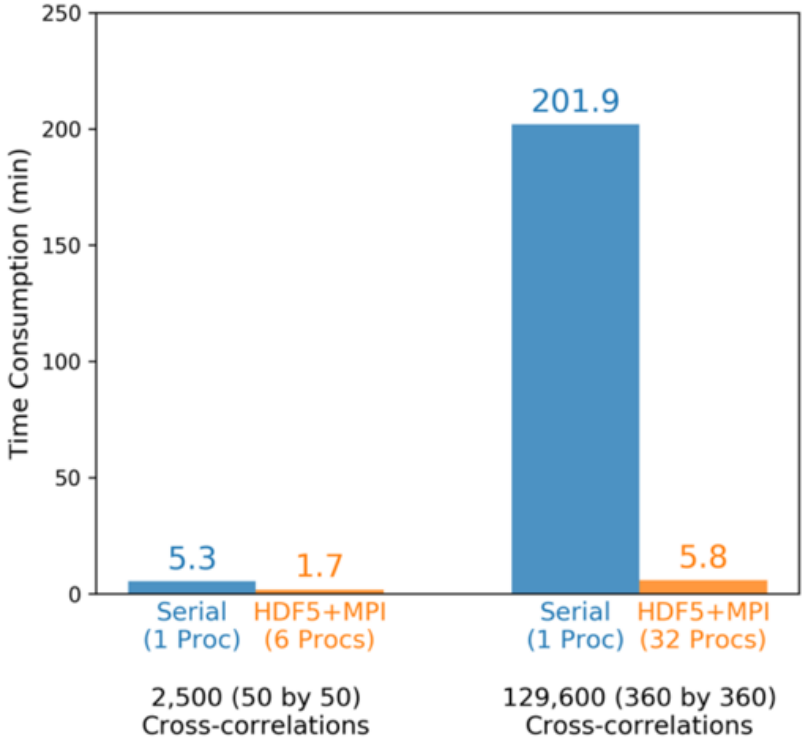
SAC and MiniSEED	vs.	ASDF
hours	vs.	Minutes

SAC and miniseed can both store uncompressed data as can HDF5 so reading speed of those is (for large enough traces) at the speed of the hardware.

A single ASDF file can deal with the order of 1 million traces.

(Per. Comm. With Lion Krischer)

Time series cross-correlation benchmarks



Community addresses data formats issues and challenges

Joint Community solutions

- Harmonised reference datasets using community standards (CF, ISO19115)
- HDF5 based hirechical, self-described, self-compressed, I/O in parallel

NetCDF/HDF in depth...

NetCDF and HDF5

Network Common Data Form (NetCDF) & Hierarchical Data Format (HDF5)

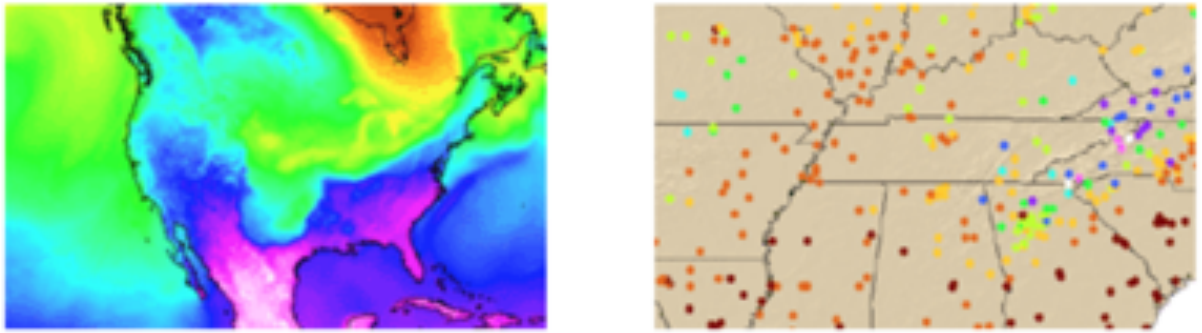
- Support a wide variety of data types as well as data structures:
  - Scientific data arrays
  - Tables
  - Raster/image data
  - String data
  - Etc...
- Used across large (and growing) spectrum of subject disciplines

Big pro: Common formats enable transdisciplinary interoperability. NetCDF4 uses HDF5 as data storage layer.

NetCDF is a machine-independent, array-oriented, multi-dimensional, self-describing, and portable data format used by various scientific communities. It has a filename extension of *.nc* or *.cdf* (though it is believed that there are subtle differences between the two).

netCDF basics

NetCDF (network Common Data Form) is a file format designed to support the creation, access, and sharing of scientific data. It is used extensively in the atmospheric and oceanographic communities to store variables, such as temperature, pressure, wind speed, and wave height.



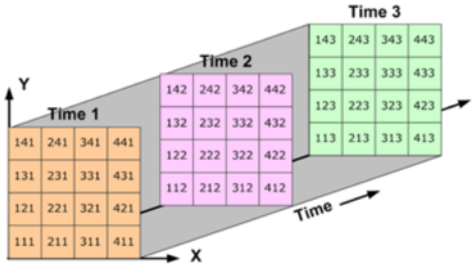
Examples of netCDF data: Left–Temperature; Right–Pressure at specific locations.

(source: https://pro.arcgis.com/en/pro-app/help/data/multidimensional/fundamentals-of-netcdf-data-storage.htm)

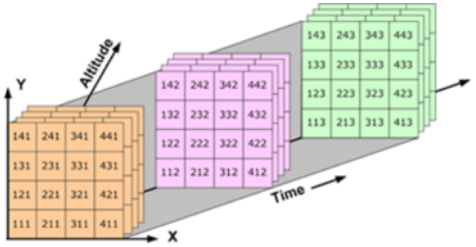
Fundamentals of netCDF data storage

The data in a netCDF file is stored in the form of arrays. For example, temperature varying over time at a location is stored as a one-dimensional array. Temperature over an area for a given time is stored as a two-dimensional array.

Three-dimensional (3D) data, like temperature over an area varying with time, or four-dimensional (4D) data, like temperature over an area varying with time and altitude, is stored as a series of two-dimensional arrays.



Three-dimensional data: Data over an area varying with time



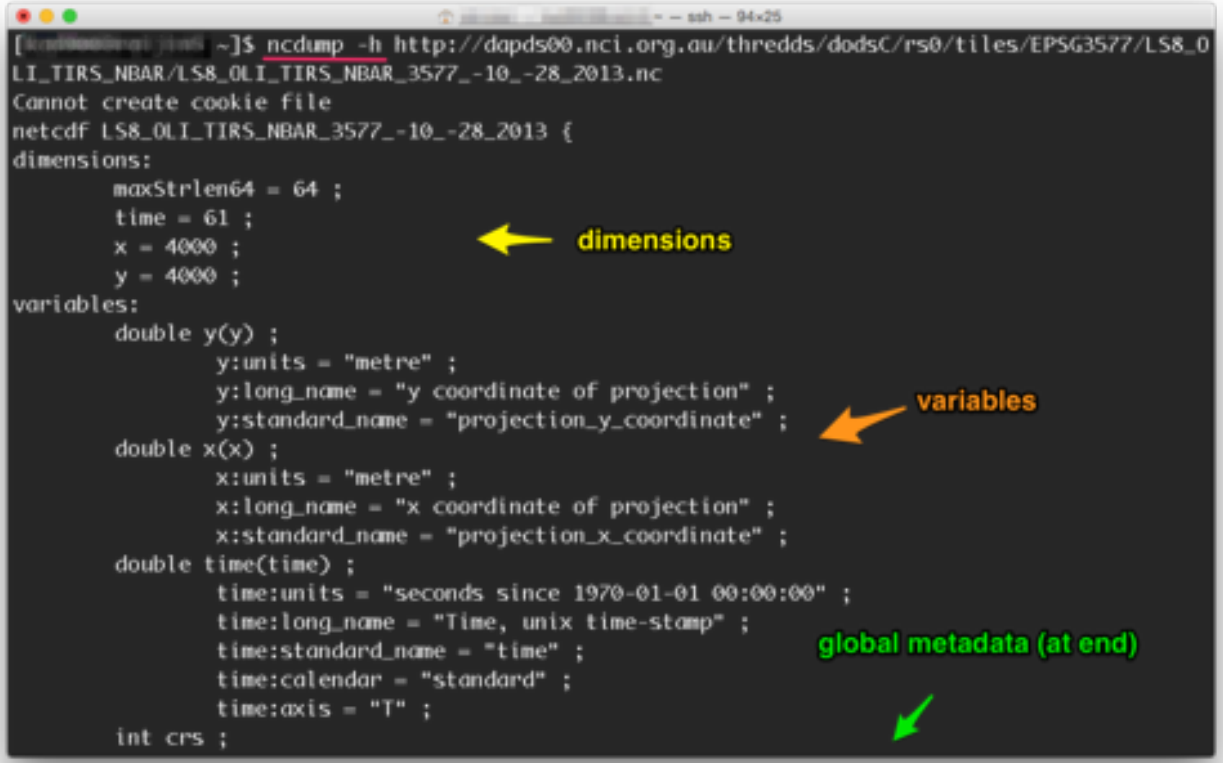
Four-dimensional data: Data over an area varying with time and altitude

(source: https://pro.arcgis.com/en/pro-app/help/data/multidimensional/fundamentals-of-netcdf-data-storage.htm)

A quick look inside...

In particular, what does “self-describing” look like?

- File metadata information
- Dimensions
- Variables
- Variable-level metadata
- Special attributes
  - Compression
  - Chunking
  - Endianness



Dimensions

```
dimensions:
        maxStrlen64 = 64 ;
        time = 61 ;
        x = 4000 ;
        y = 4000 ;
```

Dimensions

A netCDF dimension has both a name and a size. A dimension size is an arbitrary positive integer. Only one dimension in a netCDF file can have the size UNLIMITED. Such a dimension is the unlimited dimension or record dimension. A variable with an unlimited dimension can grow to any length along that dimension.

A dimension can be used to represent a real physical dimension, for example, time, latitude, longitude, or height. A dimension can also be used to index other quantities, for example, station or model run number. It is possible to use the same dimension more than once in specifying a variable shape.

Variables

Variables

A variable represents an array of values of the same type. Variables are used to store the bulk of the data in a netCDF file. A variable has a name, data type, and shape described by its list of dimensions specified when the variable is created. The number of dimensions is the rank (also known as dimensionality). A scalar variable has a rank of 0, a vector has a rank of 1, and a matrix has a rank of 2. A variable can also have associated attributes that can be added, deleted, or changed after the variable is created.

```
variables:
        double y(y) ;
                y:units = "metre" ;
                y:long_name = "y coordinate of projection" ;
                y:standard_name = "projection_y_coordinate" ;
        double x(x) ;
                x:units = "metre" ;
                x:long_name = "x coordinate of projection" ;
                x:standard_name = "projection_x_coordinate" ;
        double time(time) ;
                time:units = "seconds since 1970-01-01 00:00:00"
                time:long_name = "Time, unix time-stamp" ;
                time:standard_name = "time" ;
                time:calendar = "standard" ;
                time:axis = "T" ;
        int crs ;
```

Coordinate variables
A one-dimensional variable with the same name as a dimension is a coordinate variable. It is associated with a dimension of one or more data variables and typically defines a physical coordinate corresponding to that dimension.
Coordinate variables have no special meaning to the netCDF library. However, the software using this library should handle coordinate variables in a specialized way.

Attributes and Conventions

Attributes

NetCDF attributes are used to store ancillary data or metadata. Most attributes provide information about a specific variable. These attributes are identified by the name of the variable together with the name of the attribute.

Attributes that provide information about the entire netCDF file are global attributes. These attributes are identified by the attribute name together with a blank variable name (in CDL) or a special null variable ID (in C or Fortran).

Conventions

The conventions define metadata that provides a definitive description of the data in each variable and their spatial and temporal properties. A convention helps users of data from different sources to decide which quantities are comparable. The name of the convention is presented as a global attribute in a netCDF file.

Currently, the Climateand Forecast (CF) and Cooperative Ocean/Atmosphere Research Data Service (COARDS) conventions are supported in ArcGIS.

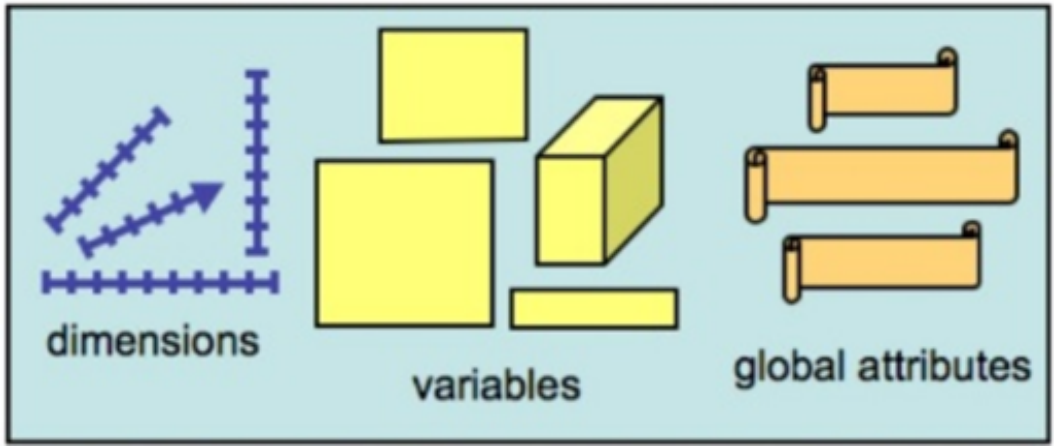
Types of NetCDF files

Classic Data Model:

- ➔ NetCDF3 (or just “classic”)
- ➔ 64-bit offset format

Enhanced Data Model:

- ➔ NetCDF4
- ➔ NetCDF4-classic

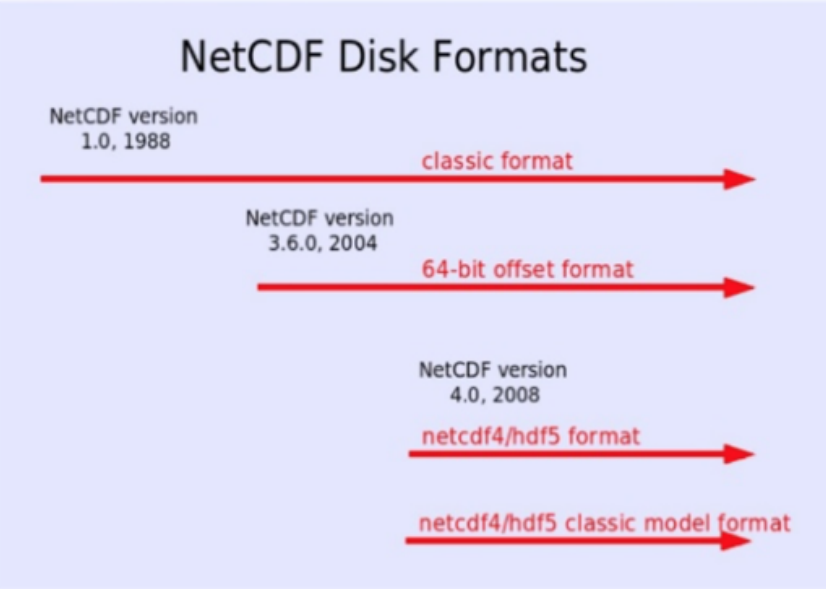


http://www.slideshare.net/HDFEOS/netcdf4-tutorial-ws14

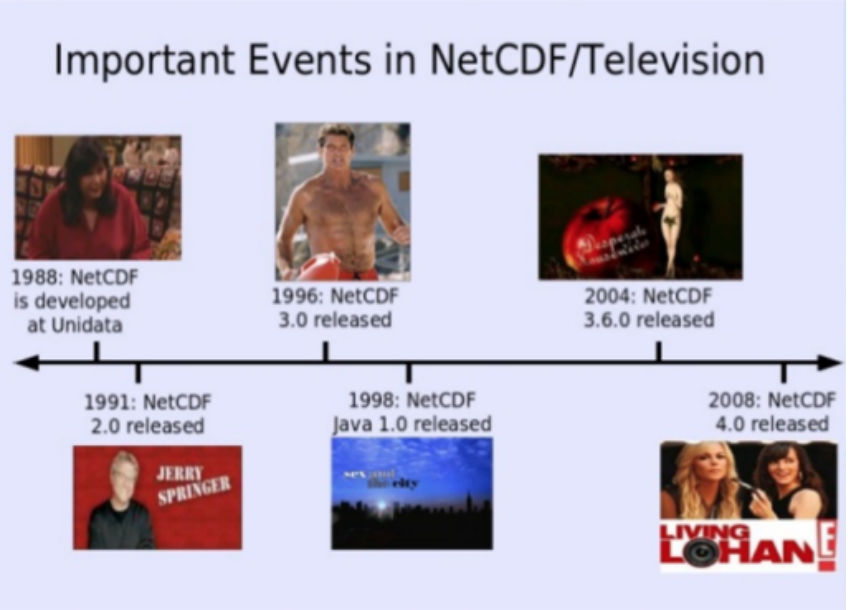
The enhanced 4.0 model adds expandable dimensions, strings and 64-bit integers, unsigned integers, groups and user-defined types.

The 4.0 release also adds some features that need not use the enhanced model, like compressions, chunking, endianness control, checksums, parallel I/O.
The netCFF core library is written in C and Java.

NetCDF history



(https://www.slideshare.net/HDFEOS/netcdf4-tutorial-ws14)



Network Common Data Form (NetCDF) & Hierarchical Data Format (HDF)

(from Unidata: http://www.unidata.ucar.edu/software/netcdf/docs/faq.html#whatisit)

- **Self-Describing.** A netCDF file includes information about the data it contains.
- **Portable.** A netCDF file can be accessed by computers with different ways of storing integers, characters, and floating-point numbers.
- **Scalable.** A small subset of a large dataset may be accessed efficiently.
- **Appendable.** Data may be appended to a properly structured netCDF file without copying the dataset or redefining its structure.
- **Sharable.** One writer and multiple readers may simultaneously access the same netCDF file.
- **Archivable.** Access to all earlier forms of netCDF data will be supported by current and future versions of the software.

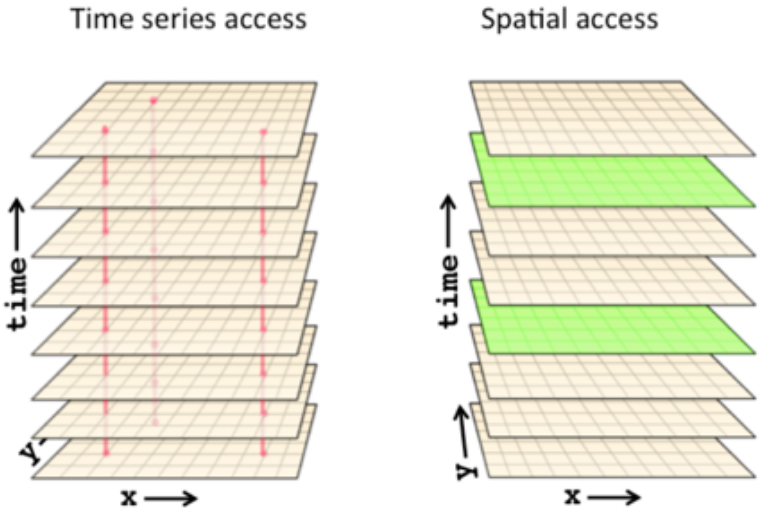
List of tools that read and/or write netCDF

- C, Fortran, Java netCDF programs
- netCDF library: ncdump, ncgen, nccopy, CDL
- ArcGIS suite
- QGIS
- Paraview
- Panoply
- Metview
- Unidata NetCDF4 module for Python
- Matlab (ncread, ncwrite)
- R (ncvar_get)

NetCDF is widely used in University Earth Science communities, for IPCC datasets, by NASA and others https://confluence.ecmwf.int/display/CKB/What+are+NetCDF+files+and+how+can+I+read+themr large data producers.

What's chunk and why it is important? (1)

First let's consider a single large 3D variable from the NCEP North American Regional Reanalysis representing air temperature (if you *must* know, it's at the 200 millibars level, at every 3 hours, on a 32.463 km resolution grid, over 33 years from 1979-01-01 through 2012-07-31). In total, it's 9.5 billion values comprising 38 GB of data.



For now, let's access small subsets of the data through a remote server. Two common access patterns are:

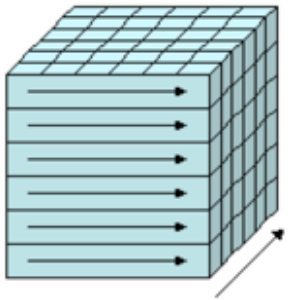
1. Get a 1D time-series of all the data from a specific spatial grid point. (red lines on left)
2. Get a 2D spatial cross section of all the data at a specific time. (green patch on right)

	x/y dimension fast	Time dimension fast
Spatial access	0.013s	200s
Time access	180s	0.012s

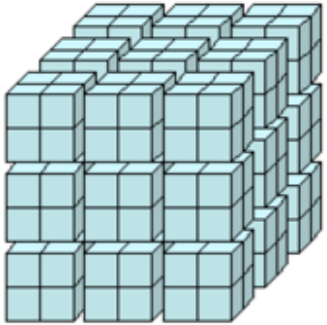
Source: https://www.unidata.ucar.edu/blogs/developer/en/entry/chunking_data_why_it_matters

What's chunk and why it is important? (2)

NetCDF and HDF5 are multidimensional array data containers. One could store multidimensional data in multi-dimensional rectangular chunks to speed up slow accesses at the cost of slowing down fast accesses. Programs that access chunked data can be oblivious to whether or how chunking is used. Chunking is supported in the HDF5 layer of netCDF-4 files



index order



chunked

Tuning parameters affecting performance:

- Order of dimensions
  - last dim is contiguous, rest are stridded.
- Chunk shape
  - Dramatically speeds access along chunked dimensions.
- Compression (per chunk)
  - Reduces size but slows access to individual chunks

Source: https://www.unidata.ucar.edu/blogs/developer/en/entry/chunking_data_why_it_matters

What's chunk and why it is important? (3)

Here's a table of timings for various shapes and sizes of chunks, using conventional local 7200 rpm spinning disk with 4096-byte physical disk blocks, the kind of storage that's still prevalent on desk-top and departmental scale platforms:

Storage layout, chunk shapes	Read time series (sec)	Read spatial slice (sec)	Performance bias (slowest / fastest)
Contiguous favoring time range	0.013	180	14000
Contiguous favoring spatial slice	200	0.012	17000
Default (all axes equal) chunks, 4673 x 12 x 16	1.4	34	24
36 KB chunks, 92 x 9 x 11	2.4	1.7	1.4
8 KB chunks, 46 x 6 x 8	1.4	1.1	1.2

Summary on chunking and more…

Large performance gains are possible with good choices of chunk shapes and sizes.

Chunking also supports efficiently extending multidimensional data along multiple axes (multiple unlimited dimensions) as well as efficient per-chunk compression, so reading a subset of a compressed variable doesn't require uncompressing the whole variable. See tools: nccopy and h5repack for rechunking.

There are no general-purpose chunking defaults that are optimal for all uses. Different patterns of access lead to different chunk shapes and sizes for optimum access. Optimizing for a single specific pattern of access can degrade performance for other access patterns.

It's costly to rewrite big datasets that use conventional contiguous layouts to use chunking instead. For example, even if you can fit the whole variable, uncompressed, in memory, chunking a 38GB variable can take 20 or 30 minutes.

The cost of chunking data means that you either need to get it right when the data is created, or the data must be important enough that the cost of rechunking for many read accesses is justified. In the latter case, you may want to consider acquiring a computing platform with lots of memory and SSD, just for the purpose of rechunking important datasets.

Read more for optimal shapes in chunking data.

Source: https://www.unidata.ucar.edu/blogs/developer/en/entry/chunking_data_why_it_matters

Performance Examples of Accessing Data: Serial IO

https://nci-data-training.readthedocs.io/en/latest/_notebook/general/data_access_serial_io.html