2021 Summer School on Effective HPC for Climate and Weather

Input/Output and Middleware

Julian Kunkel

24 August 2021

esiwace
CENTRE OF EXCELLENCE IN SIMULATION OF WEATHER AND CLIMATE IN EUROPE

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 – Talk

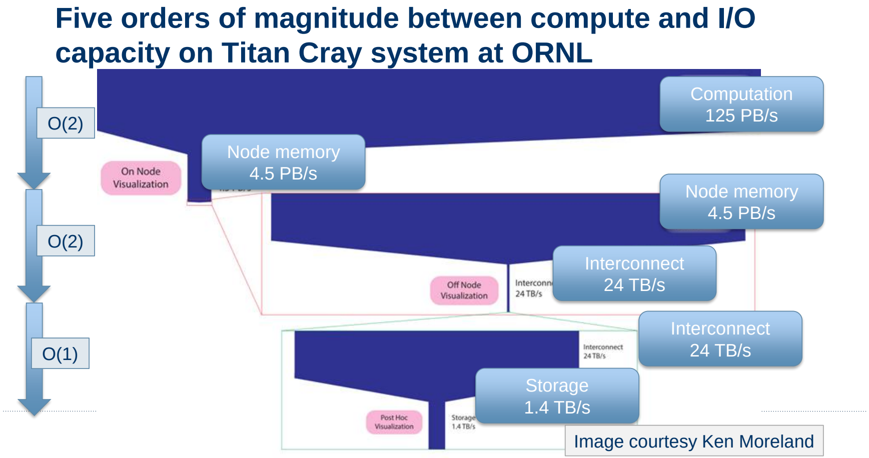
- Discuss challenges for data-driven research (Section Introduction)
- Describe the role of middleware and file formats (Section I/O Solutions)
- Identify typical I/O performance issues and their causes (Section I/O Solutions)
- Apply performance models to assess and optimise the application I/O performance (Section I/O Performance)
- Design a data model for NetCDF/CF (Section NetCDF)
- Implement an application that utilises parallel I/O to store and analyse data (Section Parallel I/O)
- Describe ongoing research activities in high-performance storage (Section Research Activities)

Learning Objectives – Lab

- Execute programs in C that read and write NetCDF files in a metadata-aware manner
- Analyse, manipulate and visualise NetCDF data
- Implement an application that utilises parallel I/O to store and analyse data

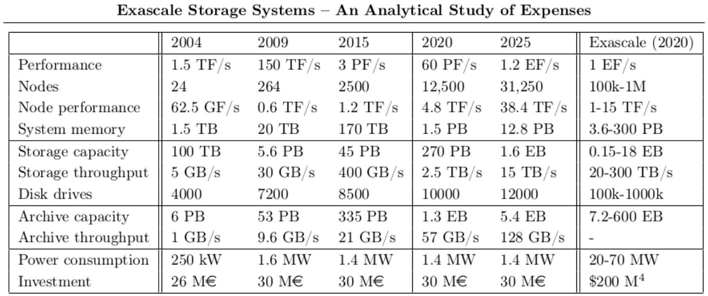
Outline

I/O Bottleneck – Titan



I/O Bottleneck – Mistral

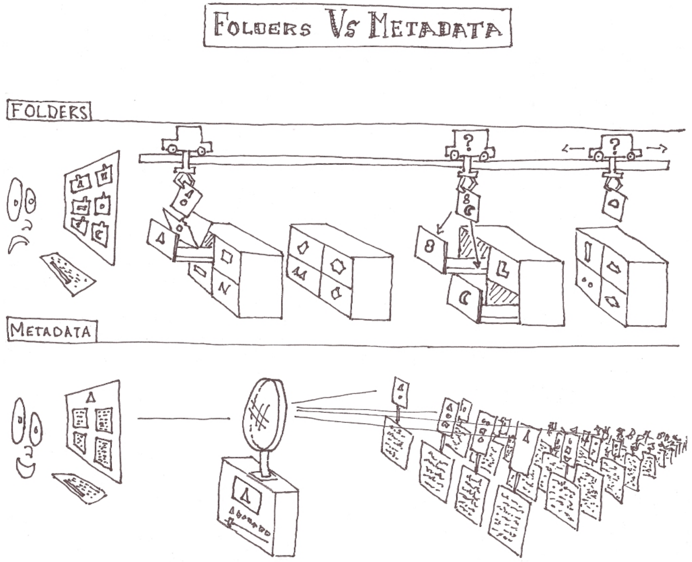
- Predict processor performance growth by 20x each generation (~5 years).
- Storage throughput/capacity improves by just 6x.
- Costs and performance come together.


Exascale Storage Systems – An Analytical Study of Expenses

	2004	2009	2015	2020	2025	Exascale (2020)
Performance	1.5 TF/s	150 TF/s	3 PF/s	60 PF/s	1.2 EF/s	1 EF/s
Nodes	24	264	2500	12,500	31,250	100k-1M
Node performance	62.5 GF/s	0.6 TF/s	1.2 TF/s	4.8 TF/s	38.4 TF/s	1-15 TF/s
System memory	1.5 TB	20 TB	170 TB	1.5 PB	12.8 PB	3.6-300 PB
Storage capacity	100 TB	5.6 PB	45 PB	270 PB	1.6 EB	0.15-18 EB
Storage throughput	5 GB/s	30 GB/s	400 GB/s	2.5 TB/s	15 TB/s	20-300 TB/s
Disk drives	4000	7200	8500	10000	12000	100k-1000k
Archive capacity	6 PB	53 PB	335 PB	1.3 EB	5.4 EB	7.2-600 EB
Archive throughput	1 GB/s	9.6 GB/s	21 GB/s	57 GB/s	128 GB/s	-
Power consumption	250 kW	1.6 MW	1.4 MW	1.4 MW	1.4 MW	20-70 MW
Investment	26 M€	30 M€	30 M€	30 M€	30 M€	\$200 M[4]

Real Values – 2018

	Mistral
Characteristics	Value
Performance	3.1 PF/s
Nodes	2882
Node performance	1.0 TF/s
System memory	200 TB
Storage capacity	52 PB
Storage throughput	700 GB/s
Disk drives	10600
Archive capacity	500 PB
Archive throughput	18 GB/s
Compute costs	15.75 M EUR
Network costs	5.25 M EUR
Storage costs	7.5 M EUR
Archive costs	5 M EUR
Building costs	5 M EUR
Investment	38.5 M EUR
Compute power	1100 kW
Network power	50 kW
Storage power	250 kW
Archive power	25 kW
Power consumption	1.20 MW


Folders Vs Metadata
FOLDERS
METADATA

Data-driven Research

- **Data-driven Research** is the science of letting data tell us what we are looking for.
  - **Database Management** is the science of efficiently storing and retrieving data.
  - **Data Mining** is the science of discovering hidden correlations in data.
- In HPC, the concerns of **storage** and **computing** are traditionally separated and optimised independently from each other and the needs of the end-to-end user.
- Workflows composed of data, computing, and communication-intensive tasks should drive interfaces and hardware configurations to best support the programming models.
- Data-driven workflows may benefit from the explicit and simultaneous use of a locally heterogeneous set of computing and storage technologies.

Outline

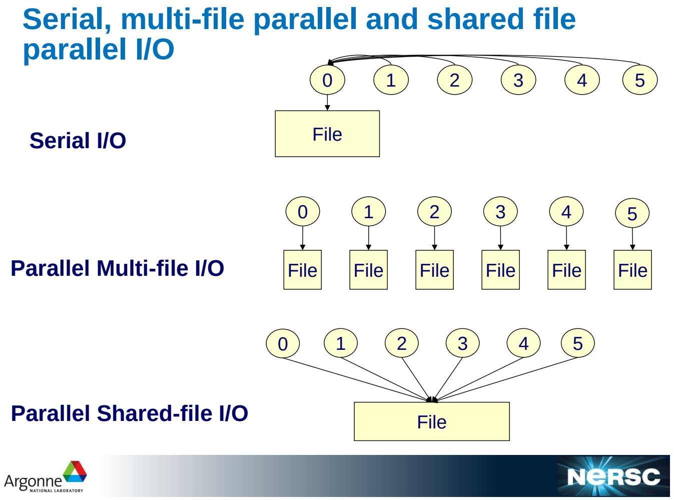
Input/Output

- Input/Output (I/O) is simply data migration.
  - Memory ⇔ Disk
- I/O is a very expensive operation!
- How is I/O performed?
  - I/O Pattern
    - Number of processes and files.
    - Characteristics of file access.
- Where is I/O performed?
  - Characteristics of the computational system.
  - Characteristics of the file system.

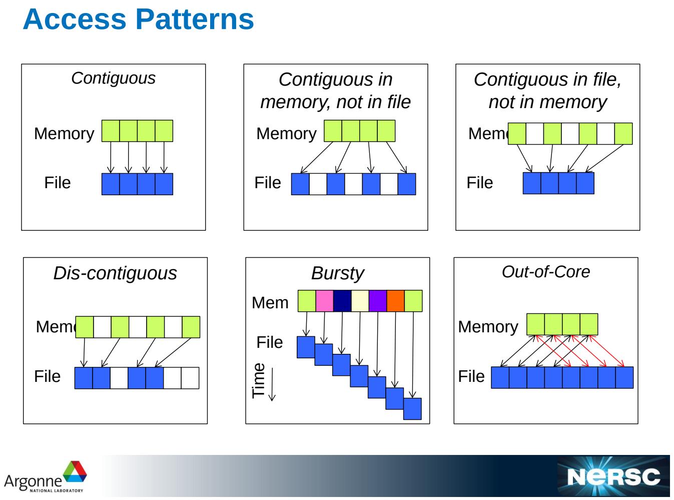
I/O Performance

- There is no "One Size Fits All" solution to the I/O problem.
- Bottlenecks in performance can occur in many locations.
  - Application and/or file system.
- Many I/O patterns work well for some range of parameters.
- Going to extremes with an I/O pattern will typically lead to problems.
- **Golden Rule:** Increase performance by decreasing the number of I/O operations and increasing the size of each operation.

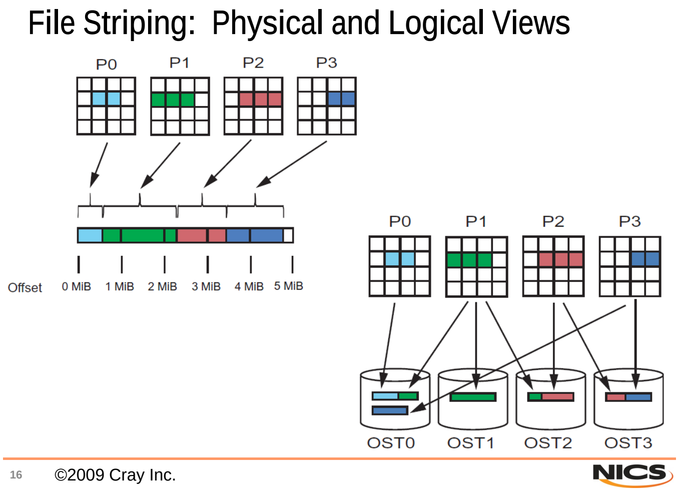
I/O Types


Serial, multi-file parallel and shared file parallel I/O
Serial I/O
0 1 2 3 4 5
File
Parallel Multi-file I/O
0 1 2 3 4 5
File File File File File File
Parallel Shared-file I/O
0 1 2 3 4 5
File
Argonne NATIONAL LABORATORY
NERSC

I/O Access Patterns



File Striping



File Striping: Physical and Logical Views

16 ©2009 Cray Inc.

P0
P1
P2
P3
Offset
0 MiB
1 MiB
2 MiB
3 MiB
4 MiB
5 MiB
OST0
OST1
OST2
OST3
NICS

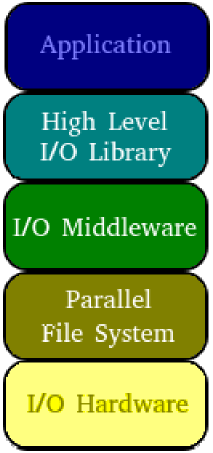
I/O Stack

Application

- Weather forecasts.
- Climate simulations, impacts, predictions and projections.
- Decisions on emission reductions.
- Strategies for housing, cities, farming, coastal defenses and other parts of society.

High Level I/O Library

- Match storage abstraction to domain.
- Provide self-describing, structured files.
- Map to middleware interface.
- Implement further optimisations.



I/O Middleware

- Match the programming model (e.g. MPI).
- Facilitate concurrent access by groups of processes.
- Expose a generic interface.
- Efficiently map middleware operations into operations in the Parallel File System.

Parallel File System

- Manage storage hardware.
- In the I/O software stack, focus on concurrent, independent access.
- Publish an interface that middleware can use effectively.

I/O Problems

- Not enough I/O capacity on current HPC systems, and the trend is getting worse.
- If there is not enough I/O, you cannot write data and then you can not analyse it.
  - Lost science!
- Missing opportunity of doing better science (analysis) when have access to full spatiotemporal resolution data.
- Energy consumption: it costs a lot of power to write data to disk.

Outline

I/O Solutions

- As we are moving towards exascale, the gap between computing power and I/O bandwidth will widen and researchers are looking for solutions to tackle this problem.
- There are essentially three lines of research:
  - at hardware level,
  - at middleware level,
  - and at application level.

Hardware Level

- Non-volatile memory (NVM)
  - Non-volatile memory (NVM) is a type of computer memory that can retrieve stored information even after having been power cycled.
  - The capabilities of NVM (i.e., capacity, bandwidth, energy consumption) are somewhere in-between main memory and persistent storage, thus it is often used as a "caching" solution between these two layers.
  - Examples of non-volatile memory include read-only memory (ROM), non-volatile random-access memory (NVRAM), magnetic (tape data storage and hard disk), and optical (disc and 5D optical data storage).

Hardware Level

- Burst buffer (BB)
  - Burst buffer (BB) is a fast and intermediate storage layer positioned between the front-end computing processes and the backend storage systems.
  - HPC applications often show bursty I/O behavior (i.e., all processes read/write at the same time) and burst buffers help to absorb these workloads.
  - Burst buffer is built from arrays of high-performance storage devices, such as NVRAM and SSD.

Middleware Level

- Solutions in I/O middleware.
  - E.g., file systems, I/O interfaces.
- **Damaris:** Software framework that overlaps computation and I/O operations by dedicating a single core to I/O tasks.
- **ADIOS:** I/O abstraction framework for HPC applications that enables switching between different I/O transport methods with little modification to application code and enabling integration of new I/O solutions.
- **DeltaFS:** File systems that improves the scalability of file systems by letting compute nodes manage metadata instead of a centralised server.

Ongoing Activity: Earth-System Data Middleware

- ESDM provides a transitional approach towards a vision for I/O addressing
  - Scalable data management practice
  - The inhomogeneous storage stack
  - Suboptimal performance and performance portability
  - Data conversion/merging
- Part of the ESiWACE Project[1]

[1]Centre of Excellence in Simulation of Weather and Climate in Europe.

Application Level

- In-situ analysis
  - In biology and biomedical engineering, in situ means to examine the phenomenon exactly in place where it occurs (i.e., without moving it to some special medium).
  - Rather than applications writing their raw output to storage to later be read again for post-processing (e.g., visualisation, filtering, statistics), in-situ processing removes this overhead by performing the analysis directly on the same machines as where the applications run.
  - ParaView, Dax, and Damaris/Viz are tools for large-scale in-situ visualisation.

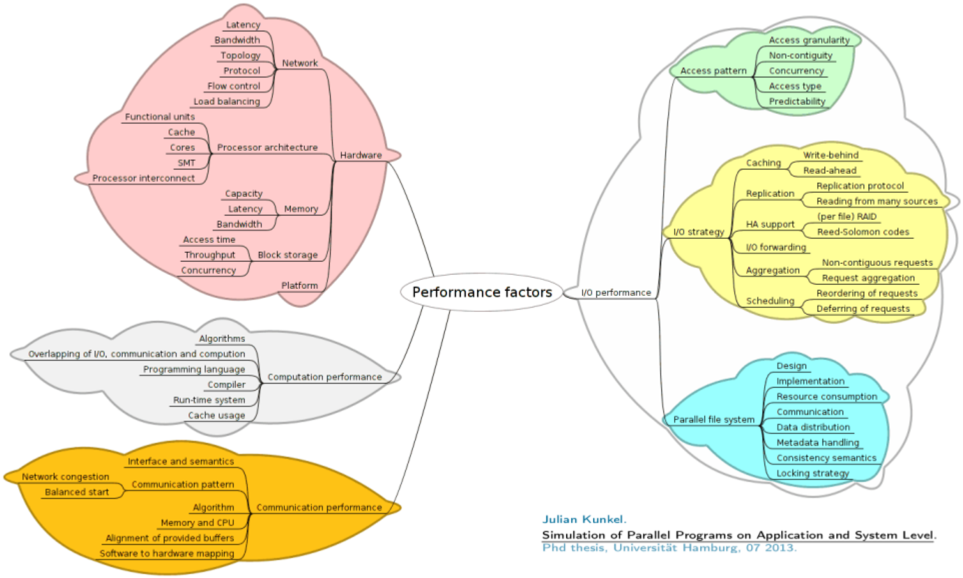
Discussion

- No “One Size Fits All” solution to the storage problem and programmers must take I/O into careful consideration when developing applications.
- Mismatch between the massive computational performance of processors and relatively limited I/O bandwidth of storage systems.
- Three methods to alleviate this problem: new hardware technology, new I/O middleware, and application-specific solutions.
  - Hardware technology shows promising solutions, but different systems might employ different solutions, reducing the portability and increasing the complexity.
  - Middleware can alleviate some of this complexity with solutions such as ADIOS and ESDM.
  - In-situ analysis is an example of how application-specific solutions can be used to improve I/O throughput and thus application performance.

Outline

I/O Performance

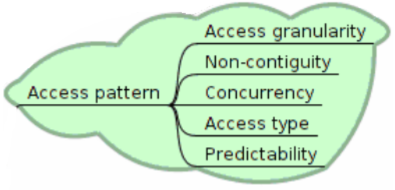
- There are several aspects involved in delivering high I/O performance to parallel applications, from hardware characteristics to methods that manipulate workloads to improve achievable performance.
- Running the same application with different I/O configurations gives the possibility to tune the I/O system according to the application access pattern.
- Another way to predict application performance in HPC systems with different I/O configurations is by using modeling and simulation techniques.



Julian Kunkel.
Simulation of Parallel Programs on Application and System Level.
Phd thesis, Universität Hamburg, 07 2013.

I/O Performance Factor – Access Patterns

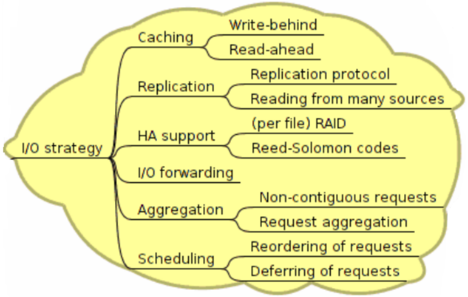
- The access pattern describes how spatial access is performed over time.



- With an access pattern, the I/O of a single client process can be described, but also the actual observable patterns on the I/O servers, or on a single block device.
- The pattern on the I/O servers is caused by all clients and defines the performance of the I/O subsystems.

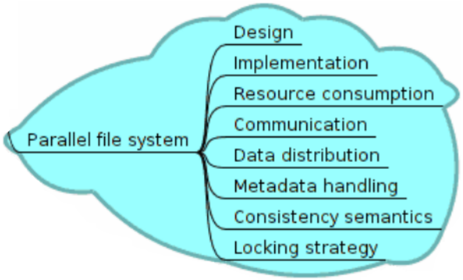
I/O Performance Factor – I/O Strategy

- In general, the mechanisms introduced here are orthogonal to the hardware and the architecture of the parallel file system.
- On the client-side, for instance, requests could already be tuned to improve the access pattern which will be observed on the servers.
- Similar to optimisations in communication, these strategies could be applied on any layer involved in I/O.



I/O Performance Factor – Parallel File System

- Performance of a parallel file system highly depends on its design as it provides the frame for the deployed optimisation strategies.
- Several aspects like consistency semantics also apply to higher level interfaces like domain specific I/O libraries.

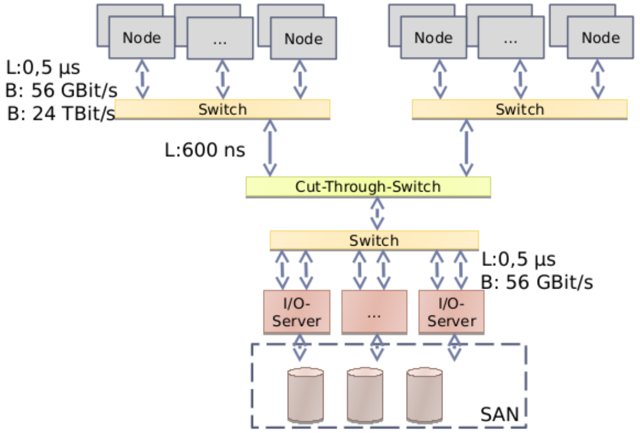


I/O Performance Analysis

- Problem
  - Assessing observed time for I/O is difficult.
  - What best-case performance can we expect?
- Support for performance analysis
  - Models and simulation
    - Trivial models – Using throughput and latency
  - Tools to capture and analyse system statistics and I/O activities
    - Darshan – Tool to identify I/O patterns and assess the performance
    - Grafana – Online monitoring
  - Benchmarks – On various levels, e.g., Metadata (md-workbench, IOR)

HPC Cluster Characteristics

- High-end components
- Extra fast interconnect, global/shared storage with dedicated servers
- Network provides high (near-full) bisection bandwidth.



I/O Performance Analysis – A Simple Model

- Communication between different machines is limited by the network performance.
- **Network throughput** is the amount of data moved successfully from one place to another in a given time period. The maximum throughput depends on the number of storage servers, client nodes and their network connectivity.
- Users typically know:
  - Output/input file size.
  - Number of nodes a job run on.
  - Runtime of the I/O during a job (this can be obtained with simple means).
- Now compute the observed throughput (`tp_obs`) in MiB/s per node. If `tp_obs` is much smaller than the network throughput, then there might be an I/O problem.
- This is a very basic model that every user should understand and apply.

I/O Performance Analysis – Numeric Example

- Consider the following scenario:
  - File size: 10 GiB
  - Number of nodes: 10
  - Time to transfer the data: 10 seconds
- Calculating the throughput in this example, one will find:
  - $\frac{10\text{ GiB}}{10\text{ nodes} \times 10\text{ seconds}} = 0.1\text{ GiB/s per node} = 100\text{ MiB/s per node}.$
- Considering that a Gigabit Ethernet network should be capable of delivering a theoretical maximum transfer of about 125 MiB/s, the estimate throughput is close to the optimal value.
- However, if you are using an Infiniband capable of delivering 6 GiB/s, then 0.1 GiB/s is a problem.

I/O Performance Analysis – Latency

- **Network latency:** Time between the sending of a message and its arrival at the receiver side.
- **Network bandwidth:** Number of bits which can be transferred in a specific time.
- Because protocols like TCP have some overhead and control algorithms, the throughput is smaller than the bandwidth.
- Latency and bandwidth depend on the used network technology and topology.
- Say, for instance, you also know the number of I/O calls as well. Then, you can compute the latency per call.
  - This information can actually be measured using Darshan, for example.
- Now compute the calls per second per node and relate it to the network latency.

Improving I/O Performance

- Software and hardware tries to hide I/O penalty.
- Caching of data:
  - Allow application to continue while I/O completes in the background (write-behind).
  - Allow to aggregate multiple (small) operations into larger operations.
  - Read data from disk before it is needed (read-ahead).
  - **Require memory! Hiding vs. increased problem size!**
- Programming:
  - Overlap I/O (or communication) with computation.
  - I/O and communication comes almost for free.
  - Optimise file format and access pattern (more complicated).

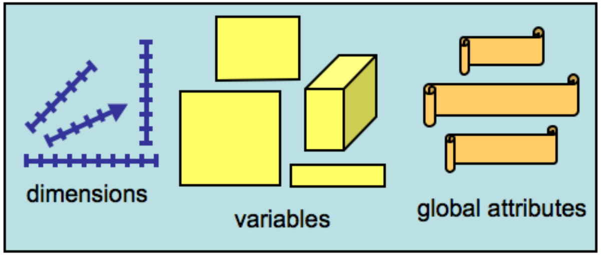
Outline

NetCDF

- In a simple view, NetCDF is:
  - A data model.
  - A file format.
  - A set of APIs and libraries for various programming languages.
- Together, the data model, file format, and APIs support the creation, access, and sharing of scientific data.
- NetCDF allows the user to describe multidimensional data and include metadata which further characterises the data.
- NetCDF APIs are available for most programming languages used in geosciences.

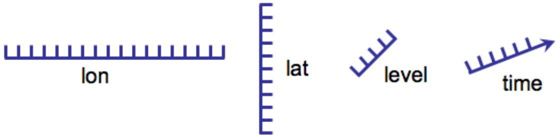
The Classic NetCDF Model

- NetCDF files are containers for Dimensions, Variables, and Global Attributes.
- A NetCDF file (dataset) has a path name and possibly some dimensions, variables, global (file-level) attributes, and data values associated with the variables.



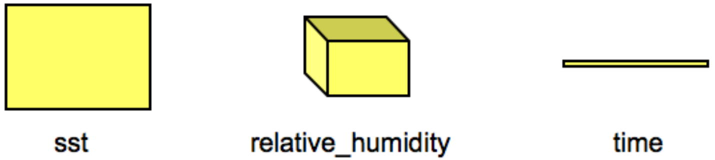
The Classic NetCDF Model – Dimensions

- Dimensions are used to specify variable shapes, grids, and coordinate systems.
- A dimension has a name and a length. Dimensions are used to define the shape of one or more variables in a NetCDF file.
- 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.



The Classic NetCDF Model – Variables

- Variables hold data values. In the classic NetCDF data model, a variable can hold a multidimensional array of values of the same type.
- A variable has a name, type, shape, attributes, and values.
- In the classic data model, the type of a variable is the external type of its data as represented on disk, one of: `char` (text character), `byte` (8 bits), `short` (16 bits), `int` (32 bits), `float` (32 bits), `double` (64 bits).

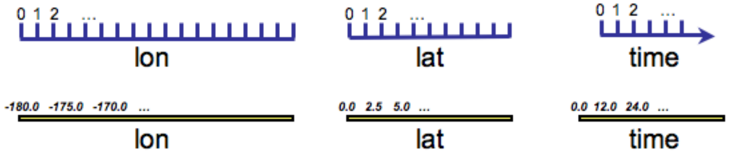


The Classic NetCDF Model – Variables

- The shape of a variable is specified with a list of dimensions.
- A variable may have attributes to specify properties such as units.
- The values of variables are the data in a NetCDF file.
- A record variable is a variable that uses a record (or unlimited) dimension. It can grow efficiently by having data added along the record dimension.
- Things you can do with a NetCDF variable include getting information about it, putting data values into it, and getting data values out of it.

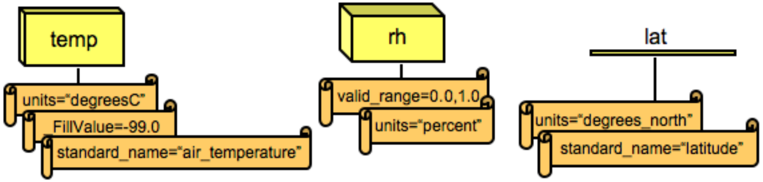
The Classic NetCDF Model – Coordinate Variables

- A one-dimensional variable with the same name as a dimension is a **coordinate variable**.
- The coordinate variable is associated with a dimension of one or more data variables and typically defines a physical coordinate corresponding to that dimension.
- Many programs that read NetCDF files recognise and use any coordinate values they find.



The Classic NetCDF Model – Attributes

- Attributes hold metadata (data about data). An attribute contains information about properties of a variable or dataset.
- An attribute has a name, type, and values. Attributes are used to specify such properties as units, standard names (that identify types of quantity), special values, maximum and minimum valid values, scaling factors, offsets, and measurement parameters.

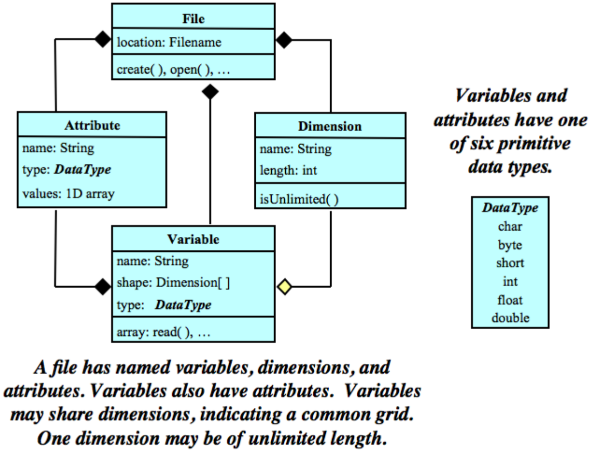


The Classic NetCDF Model – Attributes

- Things you can do with a NetCDF attribute include inquiring about its type or length, defining its value, and getting its value.
- Like variables, the type of an attribute may be one of char, byte, short, int, float, or double for the classic model.
- An attribute may have multiple values, but attributes cannot be multidimensional.
- Global attributes apply to a whole file. Variable attributes apply to a specific variable.
- NetCDF conventions are defined primarily in terms of attributes. Thus the names of attributes are standardised rather than the names of variables.
- Attributes cannot have attributes.
- Attributes are scalars or 1-D arrays.

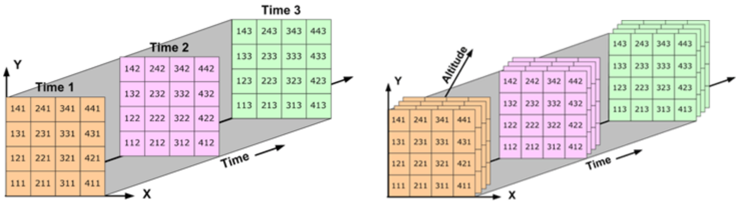
The Classic NetCDF Model – UML Diagram

- The classic NetCDF data model uses dimensions, variables, and attributes, to capture the meaning of array-oriented scientific data.


File
location: Filename
create( ), open( ), …
Attribute
name: String
type: DataType
values: 1D array
Dimension
name: String
length: int
isUnlimited( )
Variable
name: String
shape: Dimension[ ]
type: DataType
array: read( ), …
Variables and attributes have one of six primitive data types.
DataType
char
byte
short
int
float
double
A file has named variables, dimensions, and attributes. Variables also have attributes. Variables may share dimensions, indicating a common grid. One dimension may be of unlimited length.

The Classic NetCDF Model – Data

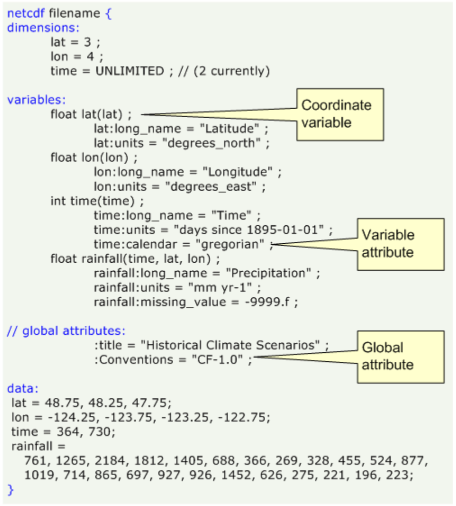
- The data in a NetCDF file is stored in the form of arrays. For example:
  - Temperature varying over time at a location is stored as a **one-dimensional array**.
  - Temperature over an area for a given time is stored as a **two-dimensional array**.
  - Three-dimensional (3D) data, like temperature 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**.



Reference: `https://pro.arcgis.com/en/pro-app/help/data/multidimensional/fundamentals-of-netcdf-data-storage.htm`

Common Data form Language (CDL)

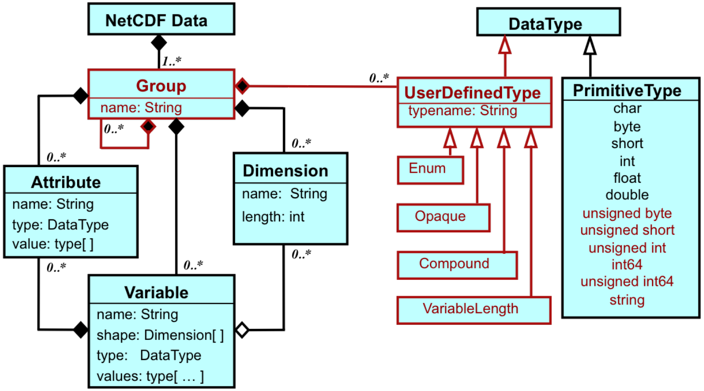
- The notation used to describe a NetCDF object is called CDL (network Common Data form Language), which provides a convenient way of describing NetCDF datasets.
- The NetCDF system includes utilities for producing human-oriented CDL text files from binary NetCDF datasets and vice-versa.
- A NetCDF file contains dimensions, variables, and attributes.
- These components are used together to capture the meaning of data and relations among data fields in an array-oriented dataset.



```
netcdf filename {
dimensions:
        lat = 3 ;
        lon = 4 ;
        time = UNLIMITED ; // (2 currently)

variables:
        float lat(lat) ;
                lat:long_name = "Latitude" ;
                lat:units = "degrees_north" ;
        float lon(lon) ;
                lon:long_name = "Longitude" ;
                lon:units = "degrees_east" ;
        int time(time) ;
                time:long_name = "Time" ;
                time:units = "days since 1895-01-01" ;
                time:calendar = "gregorian" ;
        float rainfall(time, lat, lon) ;
                rainfall:long_name = "Precipitation" ;
                rainfall:units = "mm yr-1" ;
                rainfall:missing_value = -9999.f ;

// global attributes:
                :title = "Historical Climate Scenarios" ;
                :Conventions = "CF-1.0" ;

data:
 lat = 48.75, 48.25, 47.75;
 lon = -124.25, -123.75, -123.25, -122.75;
 time = 364, 730;
 rainfall =
   761, 1265, 2184, 1812, 1405, 688, 366, 269, 328, 455, 524, 877,
   1019, 714, 865, 697, 927, 926, 1452, 626, 275, 221, 196, 223;
}
```

NetCDF Data Models

- Classic: Simplest model – Dimensions, variables, attributes
- Enhanced: More powerful model – Adds groups, types, nesting



The NetCDF-4 Enhanced Data Model

- The NetCDF-4 Enhanced Data Model, which is known as the “Common Data Model”, is part of an effort of Unidata to find a common engineering language for the development of scientific data solutions.
- The model contains the variables, dimensions, and attributes of the classic data model, but adds:
  - **Groups** – A way of hierarchically organising data, similar to directories in a Unix file system.
    - A file has a top-level unnamed group.
    - Each group may contain one or more named subgroups, user-defined types, variables, dimensions, and attributes.
  - **User-defined types** – The user can now define compound types (like C structures), enumeration types, variable length arrays, and opaque types.
  - One or more dimensions may be of **unlimited** length.

The NetCDF-4 Enhanced Data Model – CDL Files

Groups

```
netcdf regions {
group: UK {
  dimensions:
    time = UNLIMITED ; // (2 currently)
  variables:
    float average_temperature(time) ;
  data:
   average_temperature = 13.4167, 63.4167 ;
  group: Reading {
    dimensions:
      stations = 5 ;
    variables:
      float temperature(time, stations) ;
    data:
     temperature =
       11, 12, 13, 14, 15,
       61, 62, 63, 64, 65 ;
    } // group Reading
  } // group UK
}
```

Vlen

```
netcdf vlens_example {
types:
  compound obs_t {
    float pressure ;
    float temperature ;
    float salinity ;
  }; // obs_t
  obs_t(*) some_obs_t ;
  compound profile_t {
    float latitude ;
    float longitude ;
    int time ;
    some_obs_t obs ;
  }; // profile_t
  profile_t(*) some_profiles_t ;
  compound track_t {
    string id ;
    string description ;
    some_profiles_t profiles ;
  }; // track_t
dimensions:
  tracks = 42 ;
variables:
  track_t cruise(tracks) ;
data:
}
```

Compound

```
netcdf nested_compound_example {
types:
  compound wind_vector_t {
    float eastward ;
    float northward ;
    }
  compound ob_t {
      int station_id ;
      double time ;
      float temperature ;
      float pressure ;
      wind_vector_t wind ;
  }
dimensions:
    stations = unlimited ;
variables:
    ob_t obs(stations) ;
data:
    obs = {42, 0.0, 20.5, 950.0, {2.5, 3.5}};
}
```

The NetCDF-4 Enhanced Data Model – CDL Files

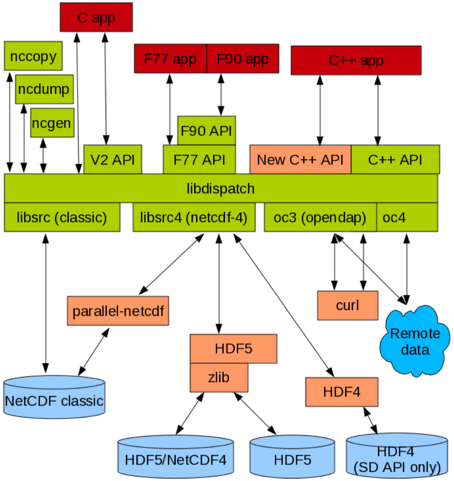
Enum

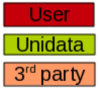
```
netcdf enums_example {
types:
  byte enum cloud_t {Clear = 0, Cumulonimbus = 1,
      Stratus = 2, Stratocumulus = 3, Cumulus = 4,
      Altostratus = 5, Nimbostratus = 6, Altocumulus = 7,
      Cirrostratus = 8, Cirrocumulus = 9, Cirrus = 10,
      Missing = 127} ;
dimensions:
  time = UNLIMITED ; // (5 currently)
variables:
  cloud_t primary_cloud(time) ;
    cloud_t primary_cloud:_FillValue = Missing ;
data:
 primary_cloud = Clear, Stratus, Clear, Cumulonimbus, _ ;
}
```

Opaque

```
netcdf opaque {
types:
  opaque(11) raw_obs_t ;
dimensions:
  time = 5 ;
variables:
  raw_obs_t raw_obs(time) ;
    raw_obs_t raw_obs:_FillValue = 0XCAFEBABECAFEBABECAFEBA ;
data:
 raw_obs = 0X0102030405060708090A0B, 0XAABBCCDDEEFFEEDDCCBBAA,
    0XFFFFFFFFFFFFFFFFFFFFFF, _, 0XCF0DEFACED0CAFE0FACADE ;
}
```

NetCDF Library Architecture





Experience-based "Best Practices" for Writing NetCDF Files

- Conventions
  - Developers should be familiar with and use existing NetCDF conventions.
- Coordinate systems
  - Spatial and temporal location of data are supported by use of coordinate systems.
- Variable grouping
  - How you group data into variables can determine whether common analysis and visualisation software can effectively use the data.
- Variable attributes
  - Conventional variable attributes supply necessary metadata.

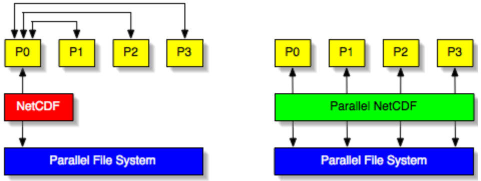
Experience-based "Best Practices" for Writing NetCDF Files

- Strings and character variables
  - Use character data properly for representing text strings.
- Calendar date and time
  - Represent calendar dates and times with standards and conventions.
- Packed data values
  - Conventions for packing numeric data to save space have some subtleties.
- Missing data values
  - To indicate that data values are missing, invalid, or not written, special values are conventionally used.

Outline

Parallel I/O

- Parallel I/O allows each processor in a multi-processor system to read and write data from multiple processes to a common file independently.



- Data-intensive scientific applications use parallel I/O software to access files.
- In HPC, increasing demands in the I/O system can cause bottlenecks. Parallel I/O plays a fundamental role to balance the fast increase in computational power and the progress of processor architectures.

Parallel I/O in NetCDF-4

- NetCDF-4 provides parallel file access to both classic and NetCDF-4/HDF5 files.
- The parallel I/O to classic files is achieved through PNetCDF while the parallel I/O to NetCDF-4 files is through the HDF5 library.
- NetCDF-4 exposes the parallel I/O features of HDF5.
  - HDF5 provides easy-to-use parallel I/O feature.
- Parallel NetCDF uses MPI I/O to perform parallel I/O. It is a complete rewrite of the core C library using MPI I/O instead of POSIX.

Using Parallel I/O in NetCDF-4

- Functions `nc_create_par` and `nc_open_par` are used to create/open NetCDF files.

```
EXTERNL int
nc_create_par(const char *path, int cmode, MPI_Comm comm,
              MPI_Info info, int *ncidp);
```

```
EXTERNL int
nc_open_par(const char *path, int mode, MPI_Comm comm,
            MPI_Info info, int *ncidp);
```

- The files they open are normal NetCDF-4/HDF5 files, but these functions also take MPI parameters.
- The parallel access associated with these functions is not a characteristic of the data file, but the way it was opened. The data file is the same, but using the parallel `open/create` function allows parallel I/O to take place.

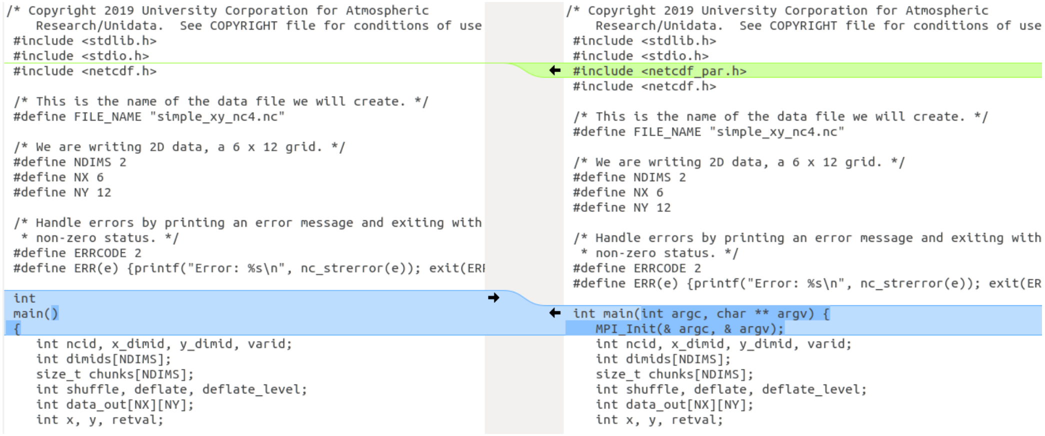
Collective and Independent Operations with Parallel I/O

- In MPI programs, I/O may be collective or independent.
  - Collective: It must be done by all processes at the same time.
  - Independent: It can be done by any process at any time.
- All NetCDF metadata writing operations are collective. All creation of dimensions, variables, attributes, types, and groups are done by all processes at the same time.
- Reading and writing data (ex. calls to `nc_put_var_int` and `nc_get_var_int`) may be independent (the default) or collective. To write into a variable collectively, call the `nc_var_par_access` function.

```
EXTERNL int
nc_var_par_access(int ncid, int varid, int par_access);
```

```
/* Use these with nc_var_par_access(). */
#define NC_INDEPENDENT 0
#define NC_COLLECTIVE 1
```

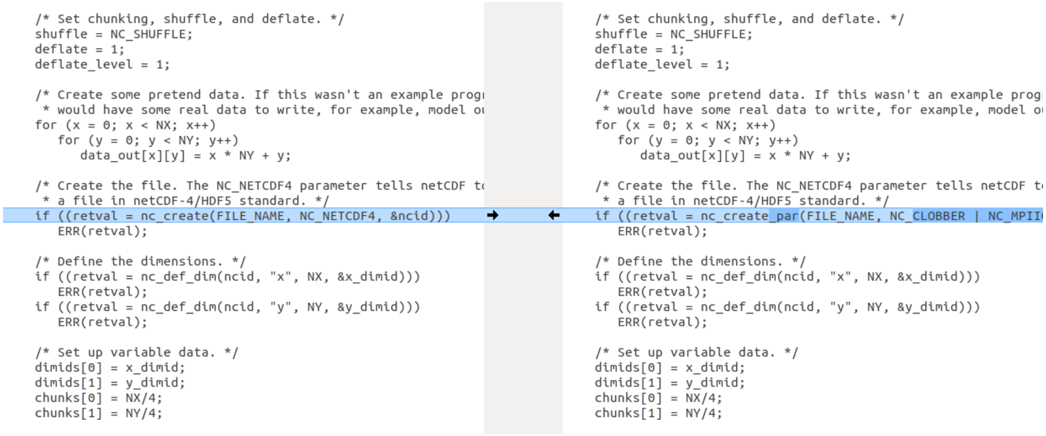
Converting NetCDF-4 to Parallel NetCDF-4



```
/* Copyright 2019 University Corporation for Atmospheric
   Research/Unidata.  See COPYRIGHT file for conditions of use
 #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_nc4.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
  * non-zero status. */
 #define ERRCODE 2
 #define ERR(e) {printf("Error: %s\n", nc_strerror(e)); exit(ER

 int
 main()
 {
    int ncid, x_dimid, y_dimid, varid;
    int dimids[NDIMS];
    size_t chunks[NDIMS];
    int shuffle, deflate, deflate_level;
    int data_out[NX][NY];
    int x, y, retval;
```

```
/* Copyright 2019 University Corporation for Atmospheric
   Research/Unidata.  See COPYRIGHT file for conditions of use
 #include <stdlib.h>
 #include <stdio.h>
 #include <netcdf_par.h>
 #include <netcdf.h>

 /* This is the name of the data file we will create. */
 #define FILE_NAME "simple_xy_nc4.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
  * non-zero status. */
 #define ERRCODE 2
 #define ERR(e) {printf("Error: %s\n", nc_strerror(e)); exit(ER

 int main(int argc, char ** argv) {
    MPI_Init(& argc, & argv);
    int ncid, x_dimid, y_dimid, varid;
    int dimids[NDIMS];
    size_t chunks[NDIMS];
    int shuffle, deflate, deflate_level;
    int data_out[NX][NY];
    int x, y, retval;
```

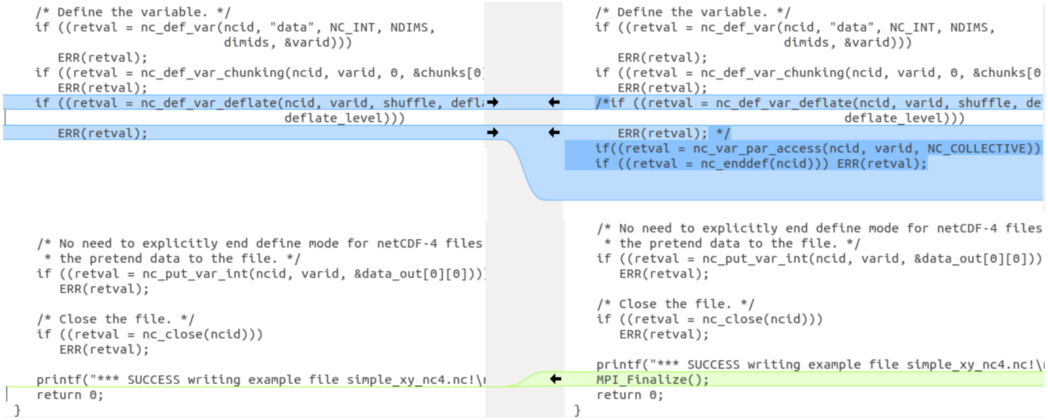
Converting NetCDF-4 to Parallel NetCDF-4



```
/* Set chunking, shuffle, and deflate. */
shuffle = NC_SHUFFLE;
deflate = 1;
deflate_level = 1;

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

/* Create the file. The NC_NETCDF4 parameter tells netCDF tı
 * a file in netCDF-4/HDF5 standard. */
if ((retval = nc_create(FILE_NAME, NC_NETCDF4, &ncid)))
   ERR(retval);

/* Define the dimensions. */
if ((retval = nc_def_dim(ncid, "x", NX, &x_dimid)))
   ERR(retval);
if ((retval = nc_def_dim(ncid, "y", NY, &y_dimid)))
   ERR(retval);

/* Set up variable data. */
dimids[0] = x_dimid;
dimids[1] = y_dimid;
chunks[0] = NX/4;
chunks[1] = NY/4;
```

```
/* Set chunking, shuffle, and deflate. */
shuffle = NC_SHUFFLE;
deflate = 1;
deflate_level = 1;

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

/* Create the file. The NC_NETCDF4 parameter tells netCDF tı
 * a file in netCDF-4/HDF5 standard. */
if ((retval = nc_create_par(FILE_NAME, NC_CLOBBER | NC_MPII
   ERR(retval);

/* Define the dimensions. */
if ((retval = nc_def_dim(ncid, "x", NX, &x_dimid)))
   ERR(retval);
if ((retval = nc_def_dim(ncid, "y", NY, &y_dimid)))
   ERR(retval);

/* Set up variable data. */
dimids[0] = x_dimid;
dimids[1] = y_dimid;
chunks[0] = NX/4;
chunks[1] = NY/4;
```

Converting NetCDF-4 to Parallel NetCDF-4

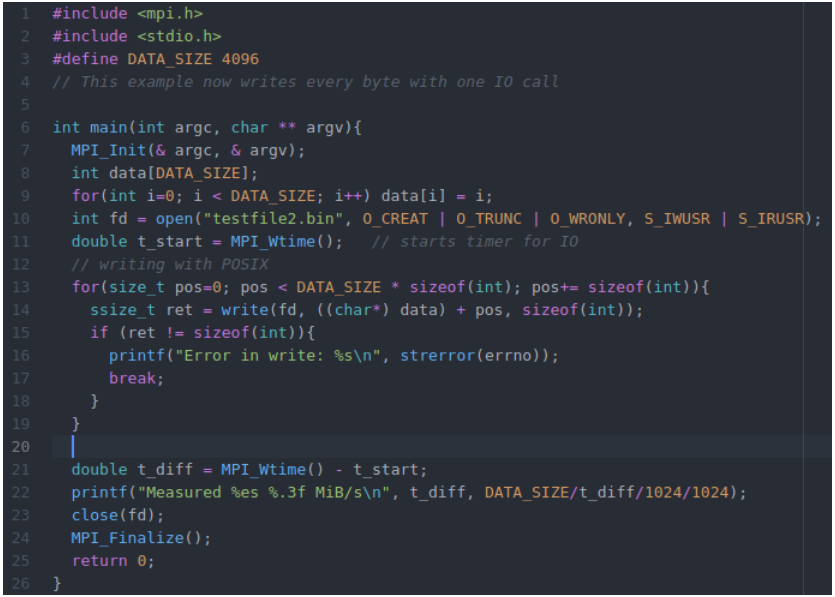


```
/* Define the variable. */
if ((retval = nc_def_var(ncid, "data", NC_INT, NDIMS,
                         dimids, &varid)))
   ERR(retval);
if ((retval = nc_def_var_chunking(ncid, varid, 0, &chunks[0]
   ERR(retval);
if ((retval = nc_def_var_deflate(ncid, varid, shuffle, defl
                                 deflate_level)))
   ERR(retval);


/* No need to explicitly end define mode for netCDF-4 files
 * the pretend data to the file. */
if ((retval = nc_put_var_int(ncid, varid, &data_out[0][0]))
   ERR(retval);

/* Close the file. */
if ((retval = nc_close(ncid)))
   ERR(retval);

printf("*** SUCCESS writing example file simple_xy_nc4.nc!\r
return 0;
}
```

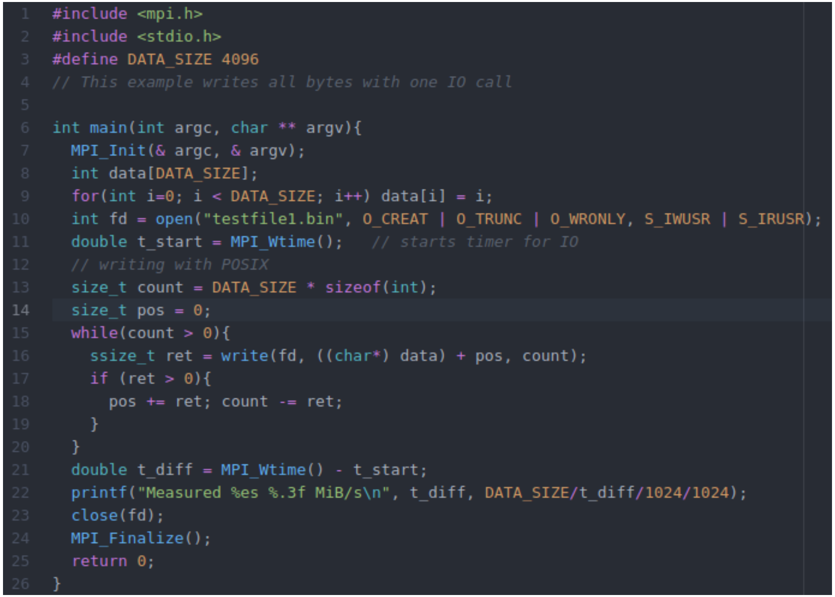
```
/* Define the variable. */
if ((retval = nc_def_var(ncid, "data", NC_INT, NDIMS,
                         dimids, &varid)))
   ERR(retval);
if ((retval = nc_def_var_chunking(ncid, varid, 0, &chunks[0
   ERR(retval);
/*if ((retval = nc_def_var_deflate(ncid, varid, shuffle, de
                                 deflate_level)))
   ERR(retval); */
if((retval = nc_var_par_access(ncid, varid, NC_COLLECTIVE))
if ((retval = nc_enddef(ncid))) ERR(retval);

/* No need to explicitly end define mode for netCDF-4 files
 * the pretend data to the file. */
if ((retval = nc_put_var_int(ncid, varid, &data_out[0][0]))
   ERR(retval);

/* Close the file. */
if ((retval = nc_close(ncid)))
   ERR(retval);

printf("*** SUCCESS writing example file simple_xy_nc4.nc!\r
MPI_Finalize();
return 0;
}
```

Examples of Performance – POSIX × MPI-I/O

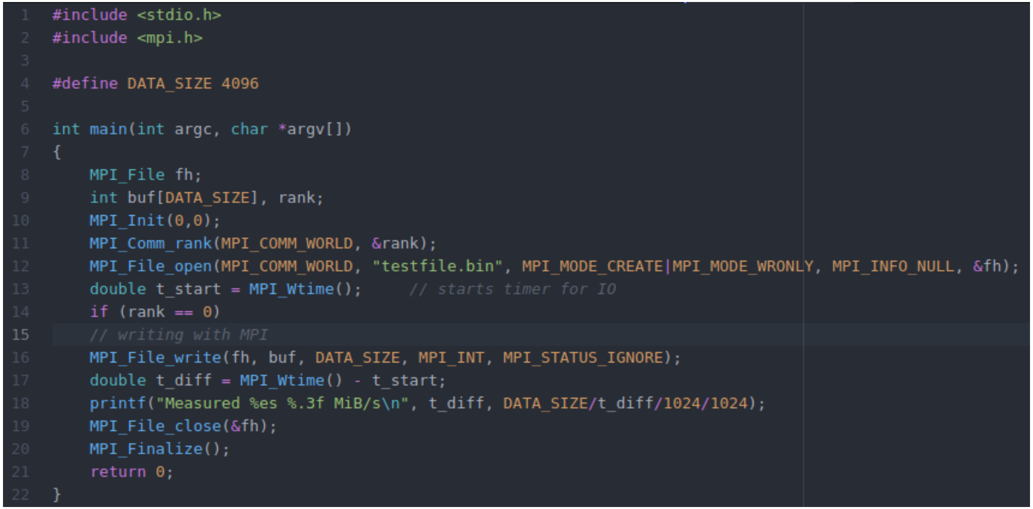
- Two programs using POSIX to write a 16k file.
  - The first program writes **every byte** with one I/O call.
  - The second program writes **all bytes** with one I/O call.
- One program using MPI-I/O to write a 16k file.
  - The program writes **all bytes** with one I/O call.
- Codes are available at:
  - `https://github.com/ESiWACE/io-training/tree/master/POSIX`
  - Also available in the Virtual Machine.



```c
#include <mpi.h>
#include <stdio.h>
#define DATA_SIZE 4096
// This example now writes every byte with one IO call

int main(int argc, char ** argv){
  MPI_Init(& argc, & argv);
  int data[DATA_SIZE];
  for(int i=0; i < DATA_SIZE; i++) data[i] = i;
  int fd = open("testfile2.bin", O_CREAT | O_TRUNC | O_WRONLY, S_IWUSR | S_IRUSR);
  double t_start = MPI_Wtime();   // starts timer for IO
  // writing with POSIX
  for(size_t pos=0; pos < DATA_SIZE * sizeof(int); pos+= sizeof(int)){
    ssize_t ret = write(fd, ((char*) data) + pos, sizeof(int));
    if (ret != sizeof(int)){
      printf("Error in write: %s\n", strerror(errno));
      break;
    }
  }

  double t_diff = MPI_Wtime() - t_start;
  printf("Measured %es %.3f MiB/s\n", t_diff, DATA_SIZE/t_diff/1024/1024);
  close(fd);
  MPI_Finalize();
  return 0;
}
```



```c
#include <mpi.h>
#include <stdio.h>
#define DATA_SIZE 4096
// This example writes all bytes with one IO call

int main(int argc, char ** argv){
  MPI_Init(& argc, & argv);
  int data[DATA_SIZE];
  for(int i=0; i < DATA_SIZE; i++) data[i] = i;
  int fd = open("testfile1.bin", O_CREAT | O_TRUNC | O_WRONLY, S_IWUSR | S_IRUSR);
  double t_start = MPI_Wtime();   // starts timer for IO
  // writing with POSIX
  size_t count = DATA_SIZE * sizeof(int);
  size_t pos = 0;
  while(count > 0){
    ssize_t ret = write(fd, ((char*) data) + pos, count);
    if (ret > 0){
      pos += ret; count -= ret;
    }
  }
  double t_diff = MPI_Wtime() - t_start;
  printf("Measured %es %.3f MiB/s\n", t_diff, DATA_SIZE/t_diff/1024/1024);
  close(fd);
  MPI_Finalize();
  return 0;
}
```



```c
#include <stdio.h>
#include <mpi.h>

#define DATA_SIZE 4096

int main(int argc, char *argv[])
{
    MPI_File fh;
    int buf[DATA_SIZE], rank;
    MPI_Init(0,0);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_File_open(MPI_COMM_WORLD, "testfile.bin", MPI_MODE_CREATE|MPI_MODE_WRONLY, MPI_INFO_NULL, &fh);
    double t_start = MPI_Wtime();      // starts timer for IO
    if (rank == 0)
    // writing with MPI
    MPI_File_write(fh, buf, DATA_SIZE, MPI_INT, MPI_STATUS_IGNORE);
    double t_diff = MPI_Wtime() - t_start;
    printf("Measured %es %.3f MiB/s\n", t_diff, DATA_SIZE/t_diff/1024/1024);
    MPI_File_close(&fh);
    MPI_Finalize();
    return 0;
}
```

Examples of Performance – POSIX × MPI-I/O

- **Best values measured in a standard laptop (Intel Core i5).**
  - Similar order of magnitude in the VM.
- Two programs using POSIX to write a 16k file.
  - The first program writes every byte with one I/O call.
    - Measured s at MiB/s.
  - The first program writes all bytes with one I/O call.
    - Measured s at MiB/s.
- One program using MPI-I/O to write a 16k file.
  - The program writes all bytes with one I/O call.
    - Measured s at MiB/s.

Examples of Performance – POSIX × MPI-I/O

- **Best values measured in a standard laptop (Intel Core i5).**
  - Similar order of magnitude in the VM.
- Two programs using POSIX to write a 16k file.
  - The first program writes every byte with one I/O call.
    - Measured 2.760475e-02s at MiB/s.
  - The first program writes all bytes with one I/O call.
    - Measured 3.928800e-05s at MiB/s.
- One program using MPI-I/O to write a 16k file.
  - The program writes all bytes with one I/O call.
    - Measured 1.160006e-07s at MiB/s.

Examples of Performance – POSIX × MPI-I/O

- **Best values measured in a standard laptop (Intel Core i5).**
  - Similar order of magnitude in the VM.
- Two programs using POSIX to write a 16k file.
  - The first program writes every byte with one I/O call.
    - Measured 2.760475e-02s at 0.142 MiB/s.
  - The first program writes all bytes with one I/O call.
    - Measured 3.928800e-05s at MiB/s.
- One program using MPI-I/O to write a 16k file.
  - The program writes all bytes with one I/O call.
    - Measured 1.160006e-07s at MiB/s.

Examples of Performance – POSIX × MPI-I/O

- **Best values measured in a standard laptop (Intel Core i5).**
  - Similar order of magnitude in the VM.
- Two programs using POSIX to write a 16k file.
  - The first program writes every byte with one I/O call.
    - Measured 2.760475e-02s at 0.142 MiB/s.
  - The first program writes all bytes with one I/O call.
    - Measured 3.928800e-05s at 99.426 MiB/s.
- One program using MPI-I/O to write a 16k file.
  - The program writes all bytes with one I/O call.
    - Measured 1.160006e-07s at MiB/s.

Examples of Performance – POSIX × MPI-I/O

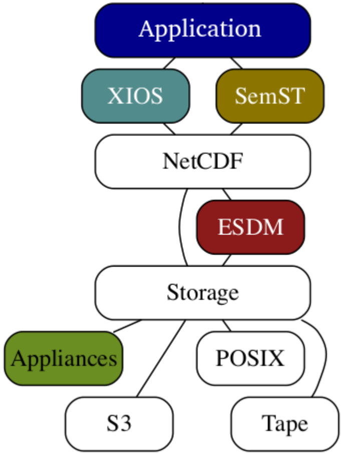
- **Best values measured in a standard laptop (Intel Core i5).**
  - Similar order of magnitude in the VM.
- Two programs using POSIX to write a 16k file.
  - The first program writes every byte with one I/O call.
    - Measured 2.760475e-02s at 0.142 MiB/s.
  - The first program writes all bytes with one I/O call.
    - Measured 3.928800e-05s at 99.426 MiB/s.
- One program using MPI-I/O to write a 16k file.
  - The program writes all bytes with one I/O call.
    - Measured 1.160006e-07s at 33674.397 MiB/s.

Outline

ESiWACE – Work Package 4

Objectives

- Support data reduction in ensembles by providing tools to carry out ensemble statistics "in-flight" and compress ensemble members.
- Hide complexity of multiple-storage tiers (middleware between NetCDF and storage) with industrial prototype backends.
- Deliver portable workflow support for manual migration of semantically important content between disk, tape, and object stores.



Next Generation Interfaces (NGI)

- Towards a new I/O stack considering:
  - User metadata and workflows as first-class citizens
  - Smart hardware and software components
  - Liquid-Computing: Smart-placement of computing
  - Self-aware instead of unconscious
  - Improving over time (self-learning, hardware upgrades)
- Why do we need a new domain-independent API?
  - Other domains have similar issues
  - It is a hard problem approached by countless approaches
  - Harness RD&E effort across domains

Smart Compression

- The main purpose of compression methods is to shrink data and to save storage space.
- Compression methods also possess a huge potential to reduce the gap between computational power and I/O performance.
  - Often, after compression, less data has to be moved.
- Many modern file formats, in particular HDF5 and NetCDF-4, provide native support for compression.
  - Beneficial in climate science, where data amounts are huge and are growing constantly.
- The compression algorithms used in HDF5 and NetCDF-4 are lossless and do not meet the requirements of climate science to full extent.

Bibliography I

Sihem Amer-Yahia, Vincent Leroy, Alexandre Termier, Martin Kirchgessner, and Behrooz Omidvar-Tehrani.
Interactive Data-Driven Research: the place where databases and data mining research meet.
Research Report RR-LIG-049, LIG, 2015.
Les rapports de recherche du LIG - ISSN : 2105-0422.

E. Wes Bethel, Andrew C. Bauer, Brad Whitlock, Matthew Wolf, Burlen Loring, and Silvio Rizzi.
In Situ Analysis and Visualization with SENSEI, Accessed in August 2020.

Lonnie Crosby.
Parallel I/O Techniques and Performance Optimization, 2012 (Accessed in August 2020).

German Climate Computing Center (DKRZ).
ESiWACE – For Future Exascale Weather and Climate Simulations, Accessed in August 2020.

Stijn Heldens, Pieter Hijma, Ben Van Werkhoven, Jason Maassen, Adam S. Z. Belloum, and Rob V. Van Nieuwpoort.
The landscape of exascale research: A data-driven literature analysis.
ACM Comput. Surv., 53(2), March 2020.

Julian Kunkel, Michael Kuhn, and Thomas Ludwig.
Exascale Storage Systems – An Analytical Study of Expenses.
Supercomputing Frontiers and Innovations, pages 116–134, 06 2014.

Julian Kunkel and Luciana Pedro.
Potential of I/O Aware Workflows in Climate and Weather.
Supercomputing Frontiers and Innovations, 7(2), 2020.

Bibliography II

Julian Kunkel.
Simulation of Parallel Programs on Application and System Level.
Phd thesis, Universität Hamburg, 07 2013.

Rob Latham, Rob Ross, Brent Welch, and Katie Antypas.
Parallel I/O in Practice, Accessed in August 2020.

UCAR/Unidata.
Network Common Data Form (NetCDF), Accessed in August 2020.

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**