CISM

NetCDF – Network Common Data Form

Jean-Paul Minet

Institut de calcul intensif et de stockage de masse (CISM)

Université Catholique de Louvain (UCL)

Belgium

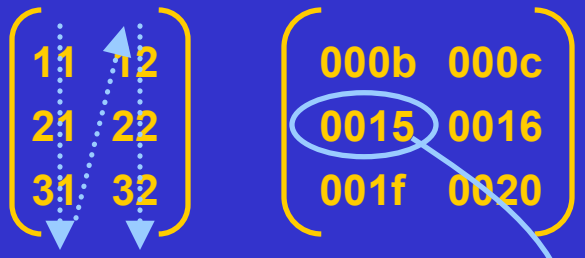
Contents

- Data exchange: what's the problem?
- One possible answer: NetCDF
- NetCDF concepts
- NetCDF data
- function libraries
- integration with ABINIT

Data exchange: the problems…

FORTRAN

```
INTEGER(KIND=2), DIMENSION(3,2)::A
```



FORTRAN: by columns

```
open(8, ´array.txt´)
write(8, *), A
```

```
20 20 20 31 31 20 20 20 32 31.....
```

```
open(9, ´array.bin´, FORM=´UNFORMATTED´)
write(9) A
```

```
0c 00 00 00 0b 00 15 00
1f 00 0c 00 16 00 20 00
0c 00 00 00
```

Sequential access by default!!

for Intel and Alpha CPU's: little endian

for MIPS, PPC,… CPU's: big endian

```
00 00 00 0C 00 0b 00 15
00 1f 00 0c 00 16 00 20
00 00 00 0c
```

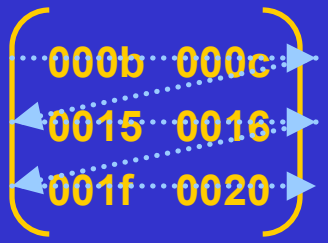
```
open(10, ´array.bin´, FORM=´UNFORMATTED´,
ACCESS=´DIRECT´,)
write(10) A
```

```
0b 00 15 00 1f 00 0c 00
16 00 20 00
```

Data exchange: the problems…

C

```
short a[3][2];
fwrite(a, sizeof(a), 1, fp);
```



C by row

```
0b 00 0c 00 15 00 16 00
1f 00 20 00
```

```
char x,y,z;
int i;
x=10; y=31; z=255;
i = 1023;
....
fwrite(x, sizeof(x), 1, fp);
fwrite(i, sizeof(i), 1, fp);
fwrite(y, sizeof(y), 1, fp);
fwrite(z, sizeof(z), 1, fp);
```

```
0a ff 03 00 00 1f ff
```

memory alignment

```
struct{char x; int i; char y,z;} mystruct
...
fwrite(mystruct, sizeof(mystruct), 1, fp)
```

```
0a f0 be 1c ff 03 00 00
1f ff 04 5d
```

One answer: NetCDF

- set of high-level functions to store/retrieve arrays providing for:
  - portability of data
  - self-describing data
  - efficient access to small subsets of large datasets
  - multiple language support
- developed and maintained by UNIDATA community (UCAR – Boulder, Colorado)
- NetCDF 3.5.1 since Feb2004

NetCDF concepts

- a netCDF dataset (file) contains dimensions, variables and attributes (each with name and ID).
- CDL: common data form language > to describe a dataset
- two utilities:
  - ncgen: generates a netCDF dataset from a CDL file
  - ncdump: displays a netCDF dataset in CDL format

NetCDF concepts

```
netcdf example_1 {

dimensions:
        pos_x=5, time=unlimited;

variables:
        float   temp(time, pos_x);
                  temp:long_name        = "temperature";
                  temp:units            = "celsius";
        float   pos_x(pos_x);
                  pos_x:long_name       = "distance from center";
                  pos_x:units           = "mm";
        int     time(time);
                  time:units            = "seconds";

                  :source = "Test simulation n°1";

data:
        time    = 3, 7;
        pos_x   = -2.5, -1, 0, 1.5, 4
        temp    = 25.34, 28.19, 30.06, 27.74, 25.43,
                  28.12, 30.00, 32.84, 31.67, 29.44;

}
```

name & length

only one in a dataset!!

name, type, shape, attributes

coordinate variables

global attribute

Order of dimensions

time = 3

NetCDF data

- external data types (CDL names):
  - char 8-bit character (text)
  - byte 8-bit signed or unsigned integer
  - short 16-bit signed integer
  - int 32-bit signed integer
  - float or real 32-bit IEEE floating-point
  - double 64-bit IEEE floating-point
- when netCDF data is written into local variable (and inversely), conversion may be required > there might be errors or loss of precision!!

NetCDF data access

- direct access:
  - a small subset can be accessed without reading the previous elements
  - reading of a variable independent of other variables (and dataset changes)
- data access:
  - all elements
  - individual elements (index vector)
  - array section (index vector and count vector)
  - subsampled array section (index vector, count vector, stride vector)

Contents

- Data exchange: what's the problem?
- One possible answer: NetCDF
- NetCDF concepts
- NetCDF data
- NetCDF library
- integration with ABINIT

NetCDF library

- function prefix: nf_ (FORTAN), nf90_ (FORTRAN90), nc_ (C) (interfaces exist for C++, perl, python)
- all functions return an error code (=0 if OK)
- f90 interface much simpler (use of optional arguments and overloaded functions): less than 30 functions instead of more than 130!!!
- a dataset: data mode or define mode

NetCDF library

- sets of functions related to:
  - datasets (create, open, inquire, close, enddef, redef, sync, abort)
  - dimensions (def_dim, inq_dimid, inq_dim, rename_dim)
  - variables (def_var, inq_var, inq_varid, put_var, get_var, rename_var)
  - attributes (put_att, inq_att, get_att, copy_att, del_att, rename_att)
- variables and dimensions are referenced by ID's, allocated sequentially upon creation

NetCDF library: nf90_open

```
function nf90_open(path, mode, ncid, chunksize)
  character (len = *), intent(in )     :: path
  integer,             intent(in )     :: mode
  integer,             intent( out)    :: ncid
  integer, optional,   intent(inout)   :: chunksize
  integer                              :: nf90_open
```

Example

```
use netcdf

implicit none

integer :: ncid, status

…

status = nf90_open(path = "foo.nc", cmode = nf90_nowrite, ncid = ncid)

if (status /= nf90_noerr) call handle_err(status)
```

NetCDF library: nf90_Inquire

```
function nf90_Inquire(ncid, nDimensions, nVariables, nAttributes, &
       unlimitedDimId)
 integer,              intent( in)     :: ncid
 integer, optional,    intent(out)     :: nDimensions, nVariables, &
      nAttributes, unlimitedDimId
 integer                               :: nf90_Inquire
```

Example

```
…
integer :: ncid, status, nDims, nVars, nGlobalAtts, unlimDimID
…
status = nf90_open("foo.nc", nf90_nowrite, ncid)
…
status = nf90_Inquire(ncid, nDims, nVars, nGlobalAtts, unLimdimID)

        ou

status = Nf90_Inquire(ncid, nDimensions = nDims, &
        unlimitedDimID = unlimdimid)
```

NetCDF library

- sets of functions related to:
  - datasets (create, open, inquire, close, enddef, redef, sync, abort)
  - dimensions (def_dim, inq_dimid, inq_dim, rename_dim)
  - variables (def_var, inq_var, inq_varid, put_var, get_var, rename_var)
  - attributes (put_att, inq_att, get_att, copy_att, del_att, rename_att)
- variables and dimensions are referenced by ID's, allocated sequentially upon creation

NetCDF library: nf90_def_dim

```
function nf90_def_dim(ncid, name, len, dimid)
  integer,              intent( in)     :: ncid
  character (len = *), intent( in)     :: name
  integer,              intent( in)     :: len
  integer,              intent(out)     :: dimid
  integer                               :: nf90_def_dim
```

Example

```
…
integer :: ncid, status, LatDimID, RecordDimID
…
status = nf90_create("foo.nc", nf90_noclobber, ncid)
…
status = nf90_def_dim(ncid, "Lat", 18, LatDimID)
status = nf90_def_dim(ncid, "Record", nf90_unlimited, RecordDimID)
```

NetCDF library: nf90_inq_dimid

```
function nf90_inq_dimid(ncid, name, dimid)
  integer,             intent( in)     :: ncid
  character (len = *), intent( in)     :: name
  integer,             intent(out)     :: dimid
  integer                              :: nf90_inq_dimid
```

Example

```
use netcdf
implicit none
integer :: ncid, status, LatDimID
…
status = nf90_open("foo.nc", nf90_nowrite, ncid)
…
status = nf90_inq_dimid(ncid, "Lat", LatDimID)
…
```

NetCDF library: nf90_Inquire_Dimension

```
function nf90_Inquire_Dimension(ncid, dimid, name, len)
  integer,                          intent( in) :: ncid, dimid
  character (len = *), optional, intent(out) :: name
  integer, optional,                intent(out) :: len
  integer                                  :: nf90_Inquire_Dimension
```

Example

```
integer :: ncid, status, LatDimID, RecordDimID
integer :: nLats, nRecords
character(len = nf90_max_name) :: RecordDimName
…
status = nf90_open("foo.nc", nf90_nowrite, ncid)
status = nf90_Inquire(ncid, unlimitedDimId = RecordDimID)
…
status = nf90_inq_dimid(ncid, "Lat", LatDimID)
status = nf90_Inquire_Dimension(ncid, LatDimID, len = nLats)
status = nf90_Inquire_Dimension(ncid, RecordDimID, &
       name = RecordDimName, len = Records)
```

NetCDF library

- sets of functions related to:
  - datasets (create, open, inquire, close, enddef, redef, sync, abort)
  - dimensions (def_dim, inq_dimid, inq_dim, rename_dim)
  - variables (def_var, inq_var, inq_varid, put_var, get_var, rename_var)
  - attributes (put_att, inq_att, get_att, copy_att, del_att, rename_att)
- variables and dimensions are referenced by ID's, allocated sequentially upon creation

NetCDF library: nf90_def_var

```
function nf90_def_var(ncid, name, xtype, dimids, varid)
  integer,                          intent( in)     :: ncid
  character (len = *),              intent( in)     :: name
  integer,                          intent( in)     :: xtype
  integer, dimension(:),            intent( in)     :: dimids
  integer                                           :: nf90_def_var
```

dimids is optional: if omitted, variable is scalar; if dimids is scalar, variable will be vector; if dimids is vector, variable will be array. If unlimited dimension exists, it must be last.

Example

```
integer :: LonDimId, LatDimId, TimeDimId
integer :: RhVarId
…
status = nf90_def_dim(ncid, "lat", 5, LatDimId)
status = nf90_def_dim(ncid, "lon", 10, LonDimId)
status = nf90_def_dim(ncid, "time", nf90_unlimited, TimeDimId)
…
status = nf90_def_var(ncid, "rh", nf90_double, &
        (/ LonDimId, LatDimID, TimeDimID /), RhVarId)
```

NetCDF library: nf90_put_var

```
function nf90_put_var(ncid, varid, values, start, count, stride, map)
  integer,                                  intent( in)    :: ncid, varid
  any valid type, scalar or array of any rank, &
                                            intent( in)    :: values
  integer, dimension(:), optional,          intent( in)    :: start, count,&
                                                              stride, map

  integer :: nf90_put_var
```

Example

```
integer :: ncId, rhVarId, status
integer, parameter :: numLons = 10, numLats = 5, numTimes = 3
real, dimension(numLons, numLats) :: rhValues
…
status = nf90_open("foo.nc", nf90_Write, ncid)
…
status = nf90_inq_varid(ncid, "rh", rhVarId)
rhValues(:, :) = 0.5
status = nf90_put_var(ncid, rhVarId,rhvalues, &
  start = (/ 1, 1, numTimes /), &
  count = (/ numLats, numLons, 1 /))
```

NetCDF library

- creating and writing a dataset:
  - CREATE — `create dataset, enter define mode`
    - DEF_DIM... — `define dimensions (name and length)`
    - DEF_VAR... — `define variables (name, type, dims)`
    - PUT_ATT... — `assign attributes values`
  - ENDDEF — `end of define mode`
    - PUT_VAR... — `write values in variables`
  - CLOSE — `close dataset`

NetCDF library

- adding new dimensions, variables, attributes:
  - OPEN — `open dataset`
  - REDEF — `enter define mode`
    - DEF_DIM... — `define dimensions (name and length)`
    - DEF_VAR... — `define variables (name, type, dims)`
    - PUT_ATT... — `assign attributes values`
  - ENDDEF — `end of define mode`
    - PUT_VAR... — `write values in variables`
  - CLOSE — `close dataset`

NetCDF library

- writing in an existing dataset:
  - OPEN — `open dataset in data mode`
    - INQ_VARID... — `get variable IDs`
    - PUT_VAR... — `write new values in variables`
    - PUT_ATTR... — `write new values in attributes`
  - CLOSE — `close dataset`

NetCDF library

- reading a dataset with known names:
  - OPEN — `open dataset (data mode)`
    - INQ_DIMID... — `get dimension IDs`
    - INQ_VARID... — `get variable IDs`
    - GET_ATT... — `get attribute values`
    - GET_VAR... — `get variable values`
  - CLOSE — `close dataset`

netcdf 3.5.1 and ABINIT

- ABINIT sources: ~ABINIT/Lib_netcdf/netcdf-3.5.1.tar
- make netcdf: untar package & triggers netcdf…
  - configure
  - make
  - make install
  - make test

  (with import of specific environment variables from makefile_macros: NC_F90, NC_F90FLAGS,….)
- make abinetcdf: compiles small test program within ABINIT
- ./abinetcdf: runs OK <> code written/compiled within ABINIT can call netcdf functions