CDO - advanced data operations

Uwe Schulzweida  
Luis Kornblueh  
Ralf Müller  
Karl-Hermann Wieners  
Oliver Heidmann  

MPI Met  

6. Dezember 2016
Overview

Max-Planck-Institut für Meteorologie

Overview

- no, thank you - I guess you know it already
- So I will concentrate on central and new features:
  - interesting options
  - possibly unknown operators
  - new operators
  - scripting with Python/Ruby

Let's have some fun!

no, thank you - I guess you know it already

So I will concentrate on central **and** new features:

- interesting options
- possibly unknown operators
- new operators
- scripting with Python/Ruby
no, thank you - I guess you know it already

So I will concentrate on central and new features:
- interesting options
- possibly unknown operators
- new operators
- scripting with Python/Ruby

Let’s have some fun!
Main Feature: One Rule to Combine them all!

Shared Memory Parallelisation

- Smallest IO unit is a **record**: one horizontal field - like a GRIB record
- Output stream of right operator is input stream of left operator

```bash
  cdo -output -selname,temp2 <ifile>
```
Main Feature: One Rule to Combine them all!

Shared Memory Parallelisation

- Smallest IO unit is a record: one horizontal field - like a GRIB record
- Output stream of right operator is input stream of left operator

```bash
  cdo -output -selname,temp2 <ifile>
```

What’s the benefit?

- Huge files can be processed as long as a single record fits into memory
- No need for temporary files - all is done in parallel!
- Other parallelisation techniques can be use on top or below: File splitting, OpenMP, multiprocessing
### Run multiple OpenMP threads

- `P <threads>`

OpenMP is mostly used in horizontal interpolation, ensemble analysis, filtering and eofs:

<table>
<thead>
<tr>
<th>Command</th>
<th>Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>cdo -P 8 remapcon</td>
<td>(conservative)</td>
</tr>
<tr>
<td>cdo -P 16 genlaf</td>
<td>(largst ares fraction)</td>
</tr>
<tr>
<td>cdo -P 2 coffee</td>
<td>(not yet released)</td>
</tr>
</tbody>
</table>
Highlights: Useful options - Part I

Run multiple OpenMP threads

-P <threads>
OpenMP is mostly used in horizontal interpolation, ensemble analysis, filtering and eofs:

<table>
<thead>
<tr>
<th>Command</th>
<th>Options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cdo -P 8 remapcon</td>
<td>conservative</td>
<td></td>
</tr>
<tr>
<td>cdo -P 16 genlaf</td>
<td>largest area fraction</td>
<td></td>
</tr>
<tr>
<td>cdo -P 2 coffee</td>
<td>not yet released</td>
<td></td>
</tr>
</tbody>
</table>

Set netcdf header size

--hdr_pad <numberOfBytes>
If the memory dedicated to data definitions is large enough, meta information can be changed without rewriting the data. [netcdf only]
Run multiple OpenMP threads

-\texttt{P <threads>}

OpenMP is mostly used in horizontal interpolation, ensemble analysis, filtering and eofs:

\begin{verbatim}
cdo -P 8 remapcon (conservative)  
cdo -P 16 genlaf (largst ares fraction)  
cdo -P 2 coffee (not yet released)
\end{verbatim}

Set netcdf header size

--hdr_pad <number0fBytes>

If the memory dedicated to data definitions is large enough, meta information can be changed \textit{without} rewriting the data. \textit{[netcdf only]}

Set output precision

-\texttt{b <number0fBits>}

Possible values are I8/I16/I32/F32/F64 for nc/nc2/nc4/nc4c  
P1 - P24 for grb/grb2
colors

-C|--color
Get colourful output with this option.
**colors**

- `C|--color`
  Get colourful output with this option.

**keep calm**

- `s|--silent`
  Suppress all warnings
**colors**

- C|--color
Get colourful output with this option.

**keep calm**

- s|--silent
Suppress all warnings

**get rid of dimensions**

--reduce_dim
Remove dimensions with length 1: time, lon, lat, lev
How to convert meta data of variables in a single step

`setpartabn` and `setpartabp` allow meta data transformations based on a fortran namelist syntax:

```fortran
& parameter
  name   = topo
  out_name = topography
  standard_name = surface_height
  units = "cm"
/
```

Other transformation keys are: `long_name`, `missing_value`, `type`, `valid_min`, `factor`, `delete`, `convert`, ...

Arbitrary attributes are supported with upcoming release `cdo-1.8.0`
cmorlite operator - upcoming cdo-1.8.0

Similar to setpartab in, but with JSON input format based on CMOR3 or CMOR2 tables.

```bash
$ cdo partab -selname,tauu iconAtm.nc
&parameter
  name=tauu
  param=17.2.0
  standard_name=u_stress
  long_name="u-momentum flux at the surface (time mean)"
  units="N m^{-2}"
/

$ cdo -partab -cmorlite,CMIP6_Amon.json,convert -selname,
  tauu iconAtm.nc tauu_cmorized.nc
&parameter
  name=tauu
  param=17.2.0
  standard_name=surface_downward_eastward_stress
  long_name="Surface Downward Eastward Wind Stress"
  units="Pa"
/
```
Highlights: built-in topography with *topo* operator

```
cdo -topo topo.grb
```

```
cdo -setrtomiss,0,10000
    -topo topo_ocean.grb
```

```
cdo -setrtimiss,-20000,0
    -topo topo_land.grb
```

See also *temp*, *const*, *random* or *stdatm*.
More than + and -

cdo -f nc \
  -expr ,'P = 1013.25 * exp(-1.602769777072154*log((exp(topo /10000.0)*213.15+75.0)/288.15))' \ 
  -topo surface_pressure.nc
**More then + and -**

```bash
cdo -f nc \
 -expr,'P = 1013.25 * \exp(-1.602769777072154*\log((\exp(topo/10000.0)*213.15+75.0)/288.15))' \
 -topo surface-pressure.nc
```

**Mask valued expressions**

```bash
cdo -f nc \
 -expr,'topo = ((topo >= 0.0)) ? topo : (topo/0.0)' \
 -topo orog.nc
```

==, !=, <, <=, >, >=, &&, ||, ?: (ternary operator)
**expr vs. aexpr**

*aexpr* performs a copy on all input fields to the output stream and appends the computation results to it. *expr* writes computed fields only.

**And what if formulars are getting lengthy?**

*exprf* and *aexprf* accept textfile names as arguments from where the formulars will be read in. See [here](#) or [here](#) for more.
Highlights: Split the grid with \textit{distgrid} - \textit{collgrid}

Break your regular grid into $n \times m$ parts

cdo -distgrid,2,3 -topo topo\_splitted

Put your pieces together with

cdo -collgrid topo\_splitted*grb collectedtopo.grb
Possible plot types

- **contour**

- **shaded**
Possible plot types

- coloured cells: *grfill*

- more: line plots, vectors, animations, output formats: png, svg, ps, pdf, ... *more examples*
Fill missing values

How to overwrite missing data with something reasonable

Model initial data for ocean salinity is on low resolution, usually 1deg.

For higher resolution runs, a simple interpolation could lead to wrong values in the Baltic Sea. Nearest-neighbor interpolation does the trick.
Smooth’em all

Even more control over how to fill missing values

Interpolation based on neighborhood or distance for arbitrary grids

Check for more options with: cdo -h smooth
cdo.{rb,py}

- is a *smart* caller of a CDO binary (with all the pros and cons)
- doesn't need to be re-installed for a new CDO version
- directly bridges your data to the scientific package in Ruby/Python
cdo.\{rb,py\}

- is a *smart* caller of a CDO binary (with all the pros and cons)
- doesn’t need to be re-installed for a new CDO version
- directly bridges your data to the scientific package in Ruby/Python
- ! isn’t a shared library, which keeps everything in memory
- ! doesn’t allow write access to files via the numpy or masked arrays

homepage: https://code.zmaw.de/projects/cdo/wiki/Cdo\{rbpy\} or directly join development at https://github.com/Try2Code/cdo-bindings
cdo.{rb,py}

- is a *smart* caller of a CDO binary (with all the pros and cons)
- doesn’t need to be re-installed for a new CDO version
- directly bridges your data to the scientific package in Ruby/Python
- ! isn’t a shared library, which keeps everything in memory
- ! doesn’t allow write access to files via the numpy or masked arrays

homepage:

https://code.zmaw.de/projects/cdo/wiki/Cdo{rbpy}

or directly join development at

https://github.com/Try2Code/cdo-bindings
from cdo import *
cdo = Cdo()

# concatenate list of files, relative time axis
cdo.cat(input = ' '.join(ofiles),
          output = ofile,
          options = '-r')

# vertical interpolation
input = 'Temperatures_L199.grb',
output = 'TempOnTargetLevels.grb')

# zonal mean after interpolation in nc4 classic format
input = '-remapbil,r1400x720 ' + myData,
output = zonmeanFile,
options = '-P 8 -f nc4c')
return numpy and masked arrays

```python
cdo.div(input='salinity.nc landSeaMask.nc',
        returnArray='S')
cdo.copy(input='--div salinity.grb landSeaMask.grb',
        returnMaArray='S', options='--f nc')
```

See this interactive session. Please share yours!
Usage: Advanced

Return numpy and masked arrays

```python
cdo.div(input='salinity.nc landSeaMask.nc',
        returnArray='S')
cdo.copy(input='-div salinity.grb landSeaMask.grb',
        returnMaArray='S', options='-f nc')
```

See this interactive session. Please share yours!

Get cdf handles: access to all variables

```python
cdf = cdo.fldmin(input=ifile, returnCdf=true)
tData = cdf.variables['T'][:]
```
return numpy and masked arrays

```python
cdo.div(input='salinity.nc landSeaMask.nc',
        returnArray='S')
cdo.copy(input='--div salinity.grb landSeaMask.grb',
        returnMaArray='S', options='--f nc')
```

See this interactive session. Please share yours!

get cdf handles: access to all variables

```python
cdf = cdo.fldmin(input=ifile, returnCdf=True)
tData = cdf.variables['T'][:]
```

conditional output: no execution if output file is present

```python
cdo.forceOutput = False # or
cdo.operator(....., force=False)
```
Beyond the shell

```python
def grepYear(ifiles, year):
    yearFiles = []
    for ifile in ifiles:
        if (year in cdo.showyear(input = ifile).split()):
            yearFiles.append(ifile)
    cdo.cat(input = ' '.join(yearFiles),
            output = yearFile)

pool = multiprocessing.Pool(8)
yearFiles = []
for year, files in filesOfYears.iteritems():
    yearFile = pool.apply_async(grepYear, [files, str(year)])
    yearFiles.append([year, yearFile, yearMeanFile])

pool.close()
pool.join()
```
Our Plans

- Have changed to C++ for using more advanced high-level data structure and algorithms for future developments.
- provide operators as library for use in many more apps (direct Python or Ruby usage, maybe even even models)
- extend for user defined operators by a plugin system
- additional optimization (OpenMP, OpenACC, OpenCL) and parallelization techniques based on distributed memory (HPX, MPI)
Our Plans

- Have changed to C++ for using more advanced high-level data structure and algorithms for future developments.
- provide operators as library for use in many more apps (direct Python or Ruby usage, maybe even even models)
- extend for user defined operators by a plugin system
- additional optimization (OpenMP, OpenACC, OpenCL) and parallelization techniques based on distributed memory (HPX, MPI)

What feature do YOU need most?
Problem

How to keep the chaining of operators working, when their number of input streams is arbitrary? - Polish notation only works for operators with fixed arity

Might not be a problem for operators like info or copy, but concatenation (cat) and merging (merge/mergetime) would create large temporary data
Problem

How to keep the chaining of operators working, when their number of input streams is arbitrary? - Polish notation only works for operators with fixed arity

Might not be a problem for operators like info or copy, but concatenation (cat) and merging (merge/mergetime) would create large temporary data

... let CDO do the wildcard evaluation

Given single quoted wildcard as input stream, CDO evaluates it into a fixed length list

```
cdo -timmean -cat 'exp004_201?_global.nc*' exp004.nc
```
Problem

How to select collections of data without explicitly given names or parameters
Play the wildcard ... with variables

Problem
How to select collections of data without explicitly given names or parameters

... use select
CDO’s select operator accepts wildcards for the 'name' and 'param' key

```bash
  cdo -select,'name=s*' $ifile $ofile
  cdo -select,'param=1.?..0' $ifile $ofile
```
Another hidden gem: *map*

Visualization, a terminal-based approach:

```
cdo -map -invertlat -topo,r360x90
```