Software for climate sciences

Peter Kuma[1,*]

[1]Stockholm University, Stockholm, Sweden

9 February 2022

[*]peter.kuma@misu.su.se, peterkuma.net/science

Introduction

Problems:

- Software is a neglected part of climate sciences.
- Software in science enables progress to happen faster.
- Little credit given to software – only to papers.
- Most papers do not include software used in the analysis.
- Open source software has revolutionised the technological sector, the same could happen in sciences.
- Very few climate models are open source.

Solution:

More open source software for science.

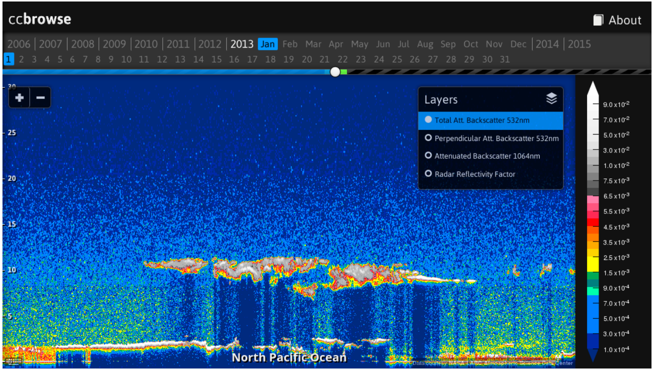
In this presentation:

1. **ccbrowse**: Visualisation of CALIPSO and CloudSat data.
2. **ds-format**: Data storage access and a data format.

ccbrowse

- Aim: visualise data from CALIPSO and CloudSat in a similar way as Google Maps.
- Initial support for CALIPSO Level 1B Profile and CloudSat 2B-GEOPROF products.
- Client/Server paradigm. Server: Python with SQLite backend. Client: JavaScript with a Leaflet 'slippy' map.

browse.ccplot.org



ccbrowse (cont.)

- **Steps:**

1. Source data: HDF4 (CALIPSO) and HDF-EOS (CloudSat) product files.
2. Break down profile data into tiles by time and height in a number of zoom levels.
3. Tiles rendered on the server in a given colormap on the fly. Caching of tiles for speed.
4. Tiles displayed with Leaflet in the browser.
5. Geolocation using the Natural Earth dataset.

- **Problem:** Large amount of data. CALIPSO Level 1B Profile is about 5 TB per year, with about 16 years of data available (up to 80 TB). However, it might be possible to use the CALIPSO OPeNDAP server as a live backend.

ds-format

github.com/peterkuma/ds-format

- Aim: Simply interface to NetCDF in Python and the command-line.
- Python library and a command-line program for reading and writing scientific data.
- NetCDF **pros:** self-describing, space efficient, cross platform, widely used; **cons:** can be very slow, massive code base (100k+ lines of code), features which no one ever uses, hard to compile on some platforms.
- NetCDF, the good parts: variables, dimensions, attributes.
- Goals:

1. Clear separation between data and metadata.
2. Everything is a JSON-like structure. (composed of Python dictionaries, lists and scalars) – *what you see is what you get*. Classes are slow and opaque.
3. Copy on write (CoW) paradigm.
4. Common structure which can be stored in NetCDF, HDF, CSV (if simple enough), ...

ds-format (cont.)

- Example: Two variables `time` and and `temperature`, one dimension `time`.

```
import numpy as np
import ds_format as ds
d = {
    'time': np.array([1, 2, 3]), # Variable "time" (numpy array)
    'temperature': np.array([16., 18., 21.]), # Variable "temperature" (numpy array)
    '.': {
        '.': { 'title': 'Temperature data' },
        'time': { # Metadata of variable "time"
            '.dims': ['time'], # Single dimension named "time"
        },
        'temperature': { # Metadata of variable "temperature"
            '.dims': ['time'], # Single dimension named "time"
            'units': 'degree_celsius', # Arbitray attributes
        },
    }
}
ds.write('dataset.nc', d) # Save the dataset as NetCDF
```

- Functions for: merging datasets, reading whole directories, subsetting with selectors.

ds native format (experimental)

- Can we improve over NetCDF?
- Aims: speed, simplicity of implementation.
- The ds native format: JSON header followed by binary data of variables in sequential order.

```
<json-header>\n
<data-var-1><data-var-2>...
```

Example ds file:


```
{"time": {".dims": ["time"], ".size": [3], ".dsize": 64, ".offset": 0, ".len": 24, ".missing": false, ".type": "int",
".endian": "l"}, "temperature": {".dims": ["time"], "units": "degree_celsius", ".size": [3], ".dsize": 64,
".offset": 24, ".len": 24, ".missing": false, ".type": "float", ".endian": "l"}, ".": {"title": "Temperature data"}}
^A^@^@^@^@^@^@^@^B^@^@^@^@^@^@^@^C^@^@^@^@^@^@^@^@^@^@^@^@0@^@^@^@^@^@2@^@^@^@^@^@^@5@
```


- Most common types supported: 32/64-bit signed/unsigned integer, 32/64-bit floating-point, boolean, byte string, Unicode string.
- Arrays with missing values supported by implementing a missing value bitmask.
- Efficient bit packing of boolean values, and byte packing of strings.
- Compatible with most commonly used features of NetCDF.

ds native format performance

- Very small implementation: 200 lines of Python code (compared to 100k+ line of code of NetCDF+HDF).
- Description of the format fits on one page.
- Up to ten times faster than NetCDF for reading and writing small files.

Performance tests:

- tiny: one int64 variable of size 1 (`{'x': 1}`).
- small: one int64 variable of size 1000 (`{'x': np.arange(1000)}`).
- large: one float64 variable of size $100 \times 1000 \times 1000$ (`{'x': np.ones(100, 1000, 1000)}`).

	time nc (s)	time ds (s)	speed factor	size nc (MB)	size ds (MB)	size factor
write tiny 100k	56	11	5	394	394	1
write small 100k	82	12	7	1566	785	2
write large 10	11	11	1	7633	7633	1
read tiny 100k	60	6	10			
read small 100k	70	8	9			
read large 10	3.3	2.5	1.3			

Outlook

ccbrowse:

- Fixing bugs.
- Making the entire catalogue of CloudSat and CALIPSO available.
- Support for mobile and touch screens.

ds-format:

- Fixing bugs.
- Stabilisation of the ds native format.
- More dataset processing functions.

Other software

- Automatic Lidar and Ceilometer Framework (ALCF): processing of lidar data and comparison with models.
- rstool: processing of data from radiosondes (iMet and Windsond).
- mpl2nc: converting binary Mini Micropulse Lidar (MiniMPL) data to NetCDF.
- mrr2c: converting Micro Rain Radar 2 (MRR-2).
- cl2nc: converting Vaisala CL31, CL51 lidar messages to NetCDF.
- aquarius-time: scientific time library.
- ccplot: visualisation of data from CloudSat and CALIPSO satellites.

More on **github.com/peterkuma**