xGCM: Staggered grids, topologies, and grid ufuncs in python

Thomas Nicholas*,
Julius Busecke,
Ryan Abernathey

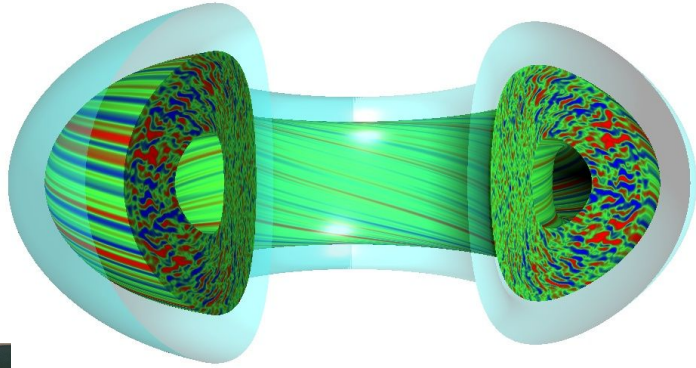
Lamont-Doherty Earth Observatory
COLUMBIA UNIVERSITY | EARTH INSTITUTE

*thomas.nicholas@columbia.edu

*github.com/TomNicholas

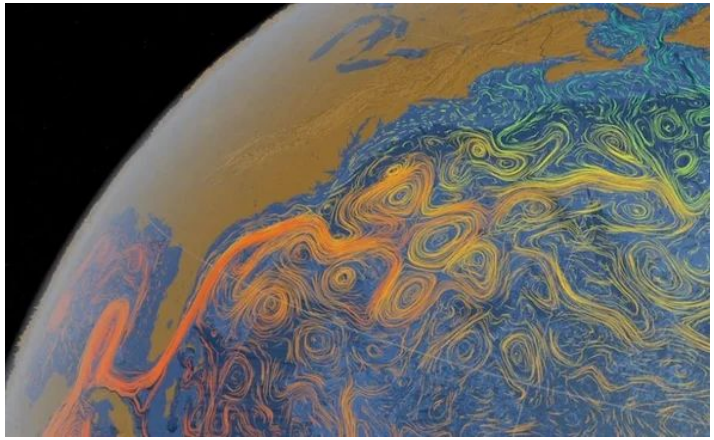
Who am I?

- Oceanographer (ex-plasma physicist)



- Xarray core dev & Pangeo user

- Both these fields have variety of big turbulence models producing gridded data



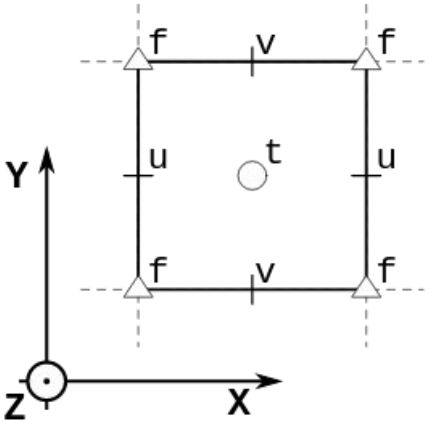
PANGEO

Talk overview

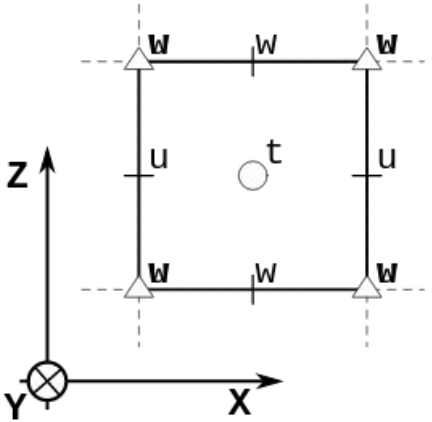
- 3 problems with model grids
  - How xGCM concepts help
- Grid ufuncs for arbitrary operations
- Will it scale?

Grid problem #1: Staggered grids

- Fluid variables live on "Arakawa Grids"
- Variables' positions are offset
- Finite-volume calculations must account for this to get correct results



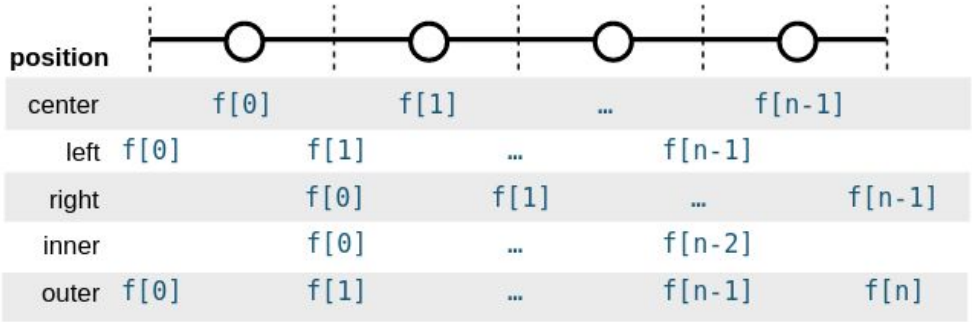
C-grid — horizontal view



C-grid — vertical view

Staggered grids in xGCM

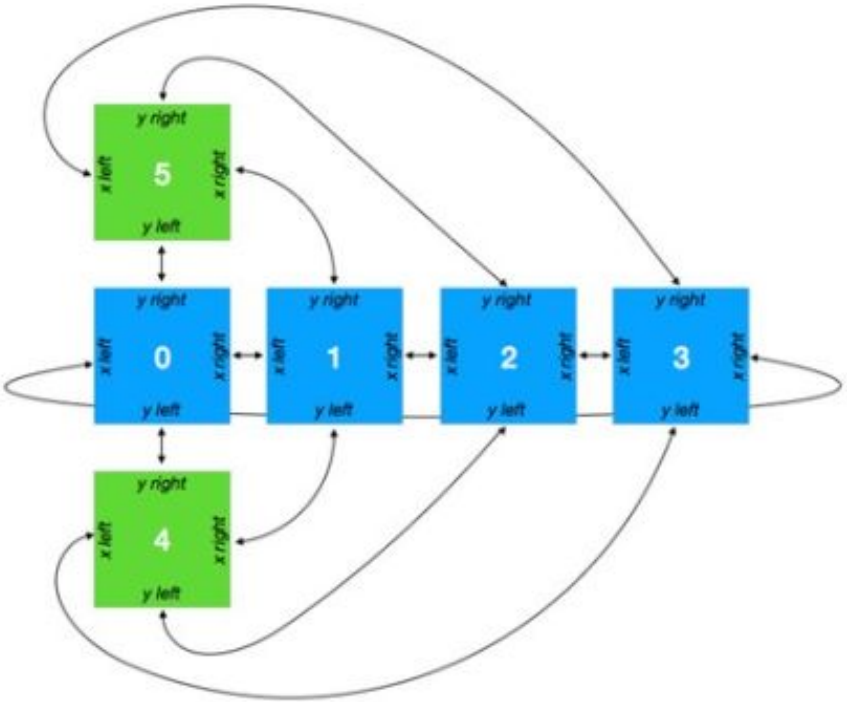
- xGCM handles staggered variables
- Extends xarray's data model with **`Axes`** and **`Grid`** objects
- Variables may live on different positions along **`xgcm.Axes`**
- **`Axes`** stored in **`Grid`** object




```
In [5]: from xgcm import Grid

In [6]: grid = Grid(ds, coords={"X": {"center": "x_c", "left": "x_g"}})

In [7]: grid
Out[7]:
<xgcm.Grid>
X Axis (periodic, boundary=None):
  * center   x_c --> left
  * left     x_g --> center
```

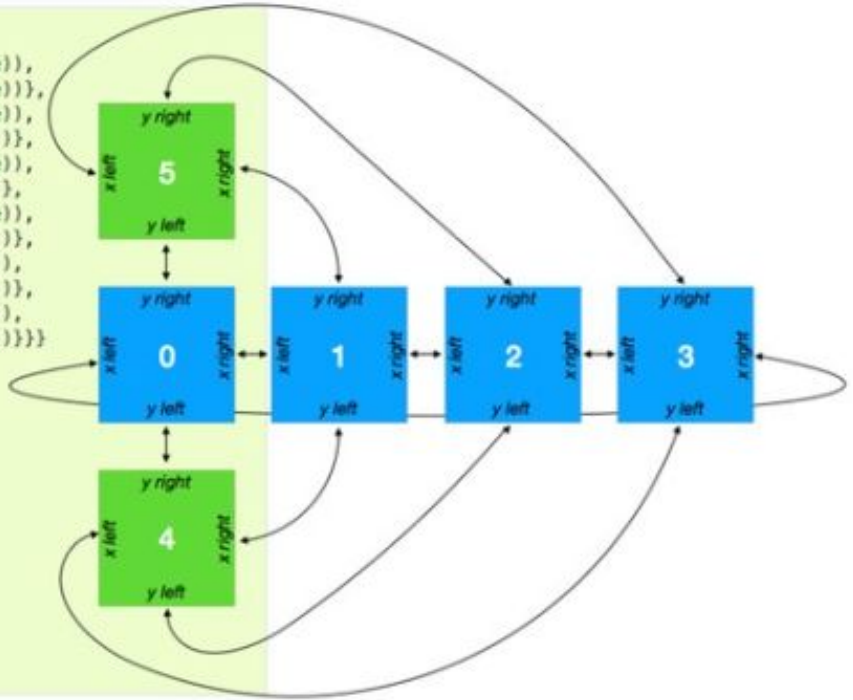
github.com/xgcm/xgcm

Grid problem #2: Topologies



Topologies in xGCM

```
In [15]: face_connections = {'face':
   ....:                     {0: {'X': ((3, 'X', False), (1, 'X', False)),
   ....:                          'Y': ((4, 'Y', False), (5, 'Y', False))},
   ....:                      1: {'X': ((0, 'X', False), (2, 'X', False)),
   ....:                          'Y': ((4, 'X', False), (5, 'X', True))},
   ....:                      2: {'X': ((1, 'X', False), (3, 'X', False)),
   ....:                          'Y': ((4, 'Y', True), (5, 'Y', True))},
   ....:                      3: {'X': ((2, 'X', False), (0, 'X', False)),
   ....:                          'Y': ((4, 'X', True), (5, 'X', False))},
   ....:                      4: {'X': ((3, 'Y', True), (1, 'Y', False)),
   ....:                          'Y': ((2, 'Y', True), (0, 'Y', False))},
   ....:                      5: {'X': ((3, 'Y', False), (1, 'Y', True)),
   ....:                          'Y': ((0, 'Y', False), (2, 'Y', True))}}}
   ....:

In [16]: grid = xgcm.Grid(ds, face_connections=face_connections)

In [17]: grid
Out[17]:
<xgcm.Grid>
Y Axis (periodic):
  * center   y --> left
  * left     yl --> center
X Axis (periodic):
  * center   x --> left
  * left     xl --> center
```



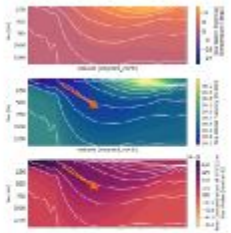
Grid problem #3: Vertical coordinates

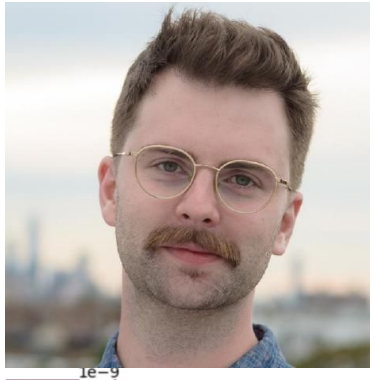
```
so_sigma = grid.transform(
  ds.so,
  'Z',
  target_values,
```

Published in **pangeo** · Pinned

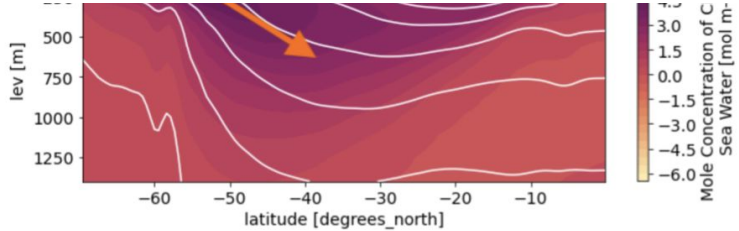
Vertical coordinate transformation with Pangeo — Have some pancakes and let Xgcm do the work

The ocean, a vast, mysterious, and beautiful place, but did it ever make you think of pancakes? Well if you are an oceanographer, it makes a l...

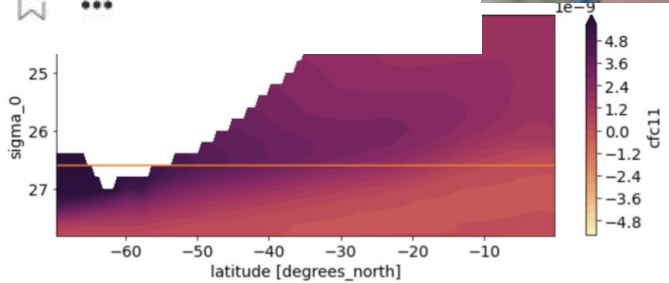




Xgcm 9 min read



Latitude-Depth sections along 130W in the Southern Ocean. From top to bottom: Potential temperature, Salinity, CFC11 concentrations

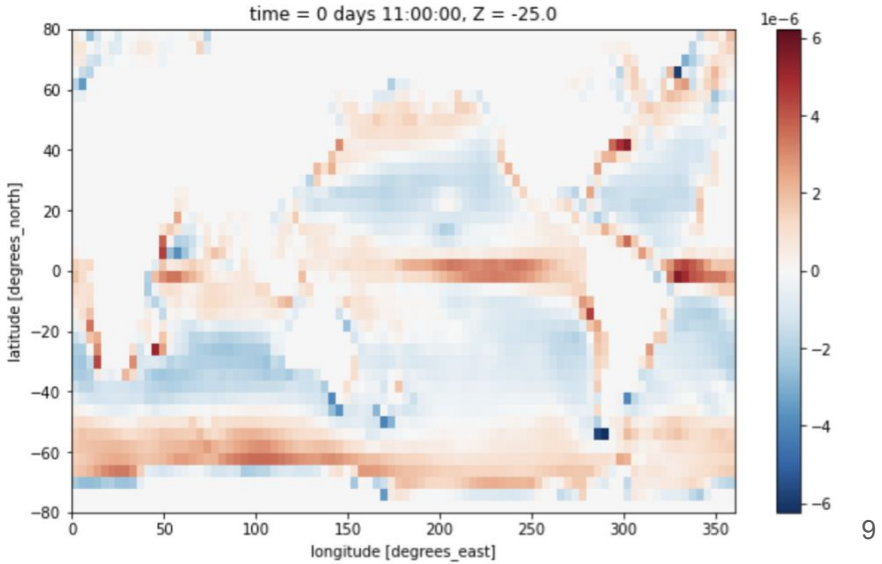


Latitude-Density sections along 130W in the Southern Ocean. Top: salinity; Bottom: CFC11 concentrations

Features: calculus operations

- Basic calculus operations .diff, .interp etc.
- Uses correct numerical scheme for grid position!
- Consumes and produces xarray objects

```
u_transport = ds.UVEL * ds.dyG * ds.hFacW * ds.drF
v_transport = ds.VVEL * ds.dxG * ds.hFacS * ds.drF

div_uv = (
    grid.diff(u_transport, 'X') + grid.diff(v_transport, 'Y')
    ) / ds.rA
```



New idea: “grid ufuncs”

- Wrap numpy ufuncs to be grid-aware
- Positions specified through “signature”
  - “(X:left)->(X:center)”
- Signature is property of computational function
  - i.e. language-agnostic idea

```python
from xgcm import as_grid_ufunc


@as_grid_ufunc(signature="(ax1:center)->(ax1:left)")
def diff_center_to_left(a):
    return a - np.roll(a, -1, axis=-1)
```

`interp_left_to_center(a)`	`"(X:left)->(X:center)"`	Forward interpolation
`diff_center_to_center(a)`	`"(X:center)->(X:center)"`	Second order central difference
`mean_depth(w)`	`"(depth:center)->()"`	Reduction

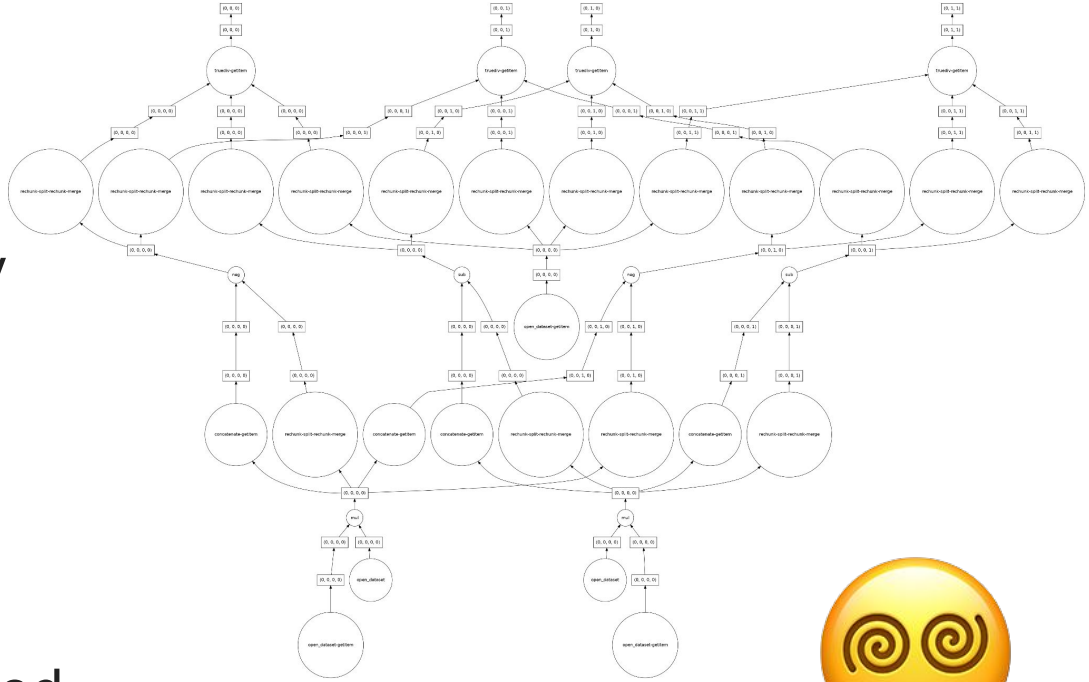
`@as_grid_ufunc` decorator

- Allows custom ufuncs
  - User-specific algorithms (e.g. from climate model)
  - Can auto-dispatch to correct ufunc for data
- Can specify grid positions via annotated type hints
- Could chain with other decorators, e.g. **`numba.jit`**

```python
from typing import Annotated
from xgcm import as_grid_ufunc


@as_grid_ufunc(
    boundary_width={"X": (0, 1), "Y": (0, 1)},
)
def divergence(
    u: Annotated[np.ndarray, "(X:left,Y:center)"],
    v: Annotated[np.ndarray, "(X:center,Y:left)"],
) -> Annotated[np.ndarray, "(X:center,Y:center)"]:

    du_dx = u[..., 1:, :] - u[..., :-1, :]
    dv_dy = v[..., 1:]    - v[..., :-1]

    return du_dx + dv_dy
```

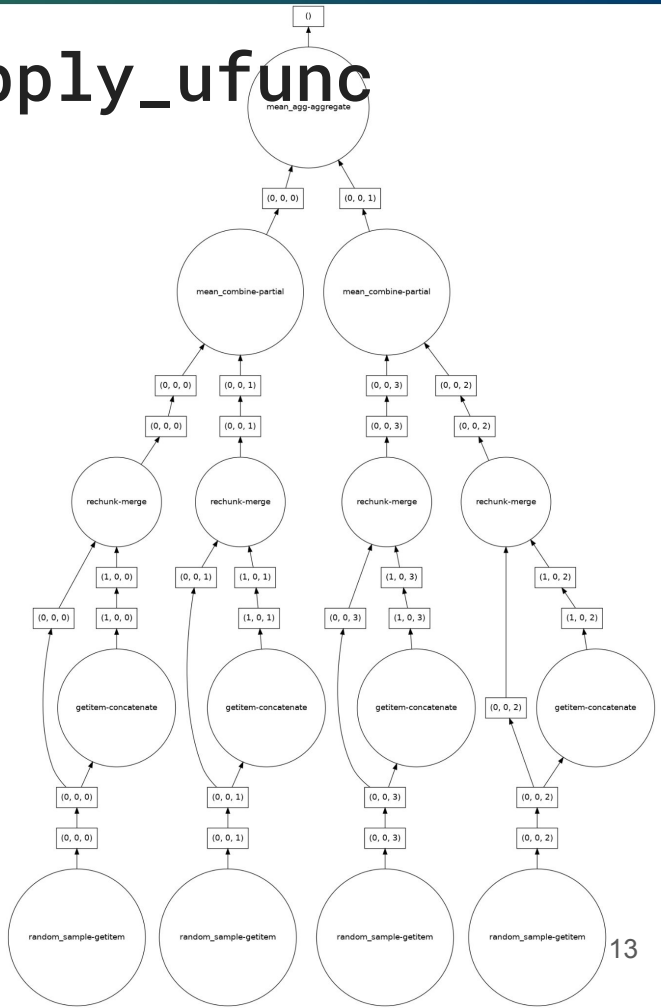
Will it scale? Originally no…

- xGCM's finite-volume functions, e.g. diff, interp
- Require padding to apply boundary conditions
- Previously used custom reduction code
- Chaining diff, interp etc. led to explosion of dask tasks



Dask-optimised xGCM via `xarray.apply_ufunc`

- Apply all grid ufuncs through `xarray.apply_ufunc`
  - Common code path for all functions
- Only pad once
  - Avoids task explosion
- Creates minimal dask graph
- Ex. reduction: Almost blockwise (+ a rechunk-merge operation after padding)



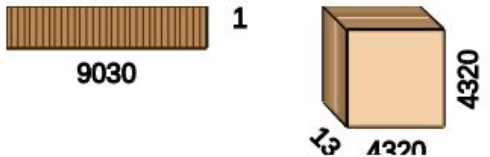
Now does it scale?

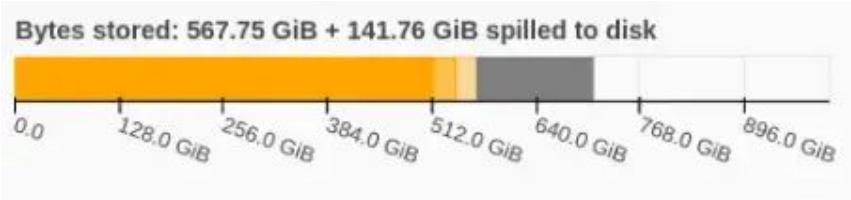
- Tried it on huge global ocean simulation dataset
- Single variable is 8.76 TB
  - 117390 Zarr chunks
- But saw dreaded orange bars!
- Nice task graph, but dask using way too much RAM!

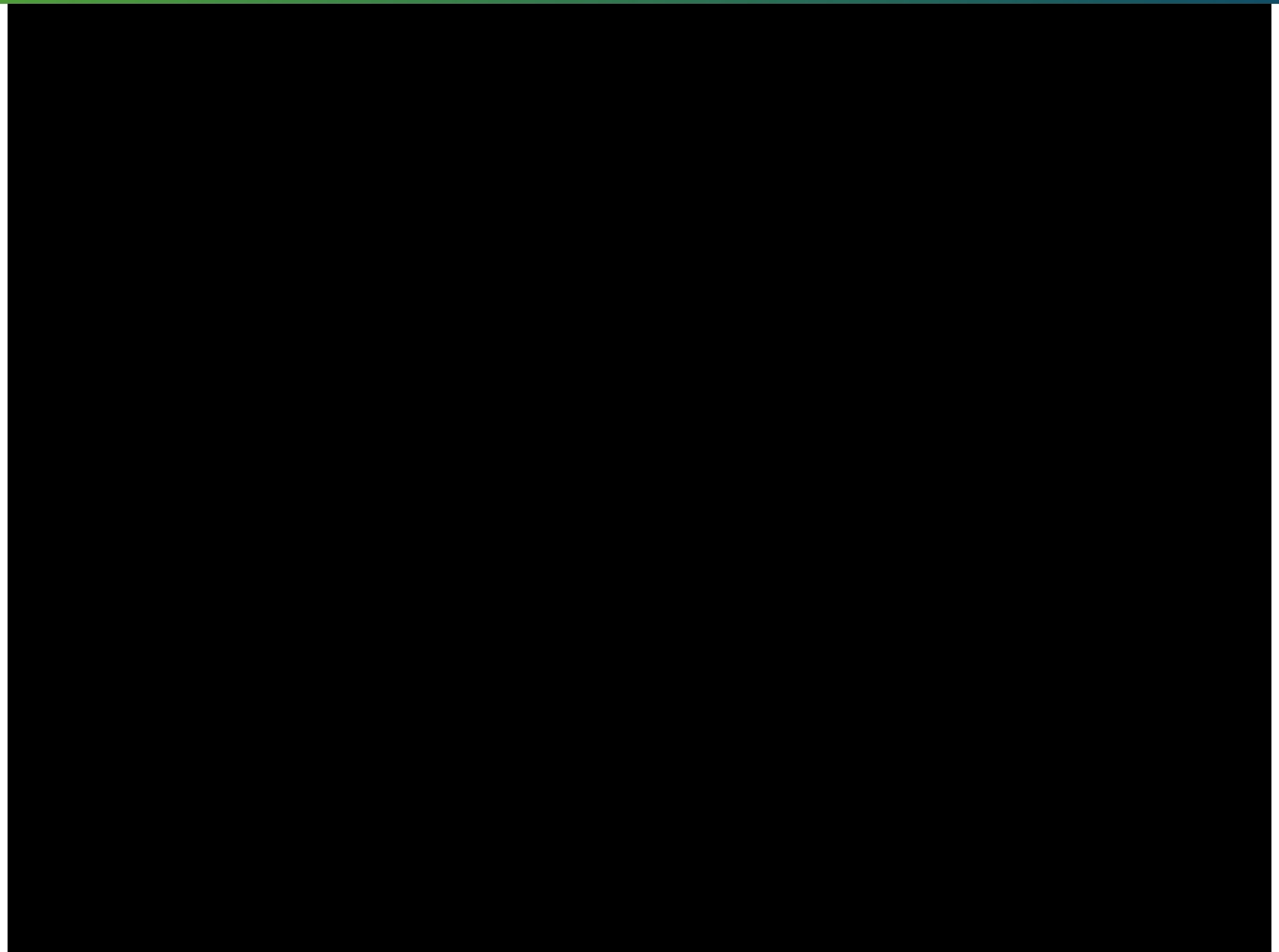
▾ Data variables:

U	(time, face, j, i_g)	float32	dask.array<chunksize=(1, 1, 4320, 4320)

	Array	Chunk
Bytes	7.97 TiB	71.19 MiB
Shape	(9030, 13, 4320, 4320)	(1, 1, 4320, 4320)
Count	117390 Tasks	117390 Chunks
Type	float32	numpy.ndarray







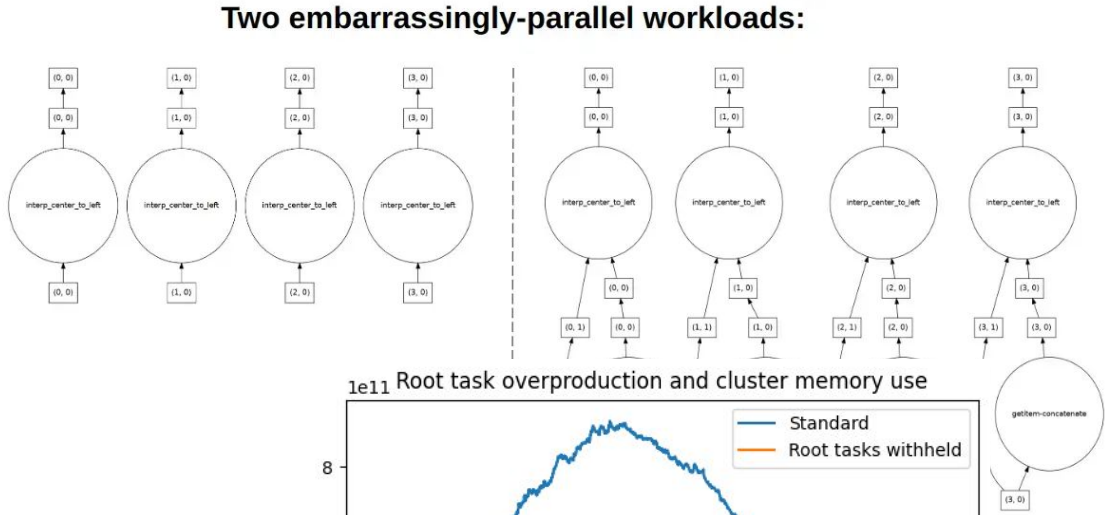
Dask needed fixing!

- Exposed a bug in `dask.distributed`
- Gabe Joseph of Coiled fixed it!
- See pangeo blog post

Dask.distributed and Pangeo: Better performance for everyone thanks to science / software...

Improvements to dask's performance at scale are a success story for pangeo too

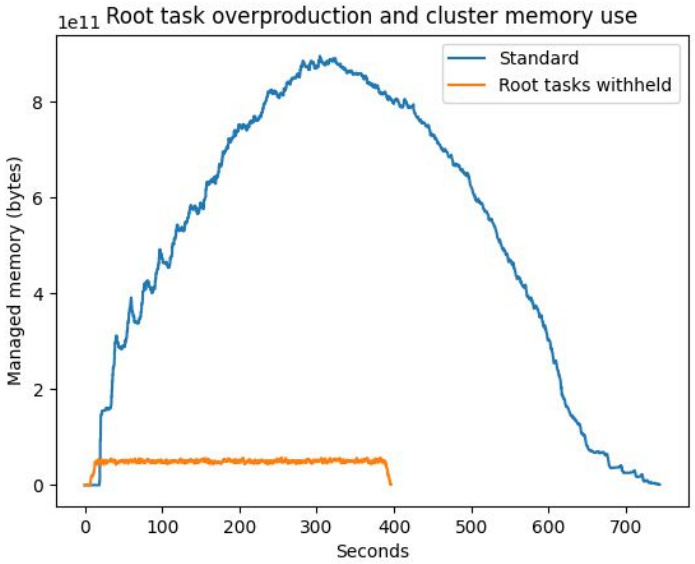
Tom Nicholas
Jan 4 · 10 min read



`xarray.apply_`

Streams smoot

nc



Summary

- GCM grids are a pain
- But xGCM can help!
- Grid ufuncs are a cool idea
- We improved dask along the way!

P.S. I am looking for my next big project 😁

github.com/xgcm/xgcm