Scaling UXarray: Bridging the Gap for High-Performance Unstructured Grid Analysis and Documentation Enhancements

Rachel Tam[1,2], Philip Chmielowiec[1], Orhan Eroglu[1]

[1] NSF National Center for Atmospheric Research, [2] University of Illinois, Urbana-Champaign

Motivation

The climate and global weather modeling communities have been transitioning to dynamical cores employing more flexible unstructured grids upon which more complex governing numerical equation.

The NSF EarthCube-funded effort, Project Raijin, has spearheaded to create the UXarray Python Package. It is a community-owned, sustainable, and scalable tool for analyzing and visualizing unstructured grids.



Fig. 1 Structured (left) and unstructured grids (center and right) side-to-side comparison, where unstructured grids can comprise of uniform or varying shapes.

This poster highlights development efforts in UXarray to **develop a scalable workflow**, leveraging tools in the Scientific Python Ecosystem. An **in-depth user guide on using Dask in UXarray and visualization examples are also created** within the package to enhance user experience and demonstrate its full capabilities.

Overview and Highlighted Features

UXarray provides Xarray-styled functionality for better read-in and usage of datasets that follows standard unstructured grid conventions, such as UGRID, MPAS, SCRIP, ESMF, and Exodus.

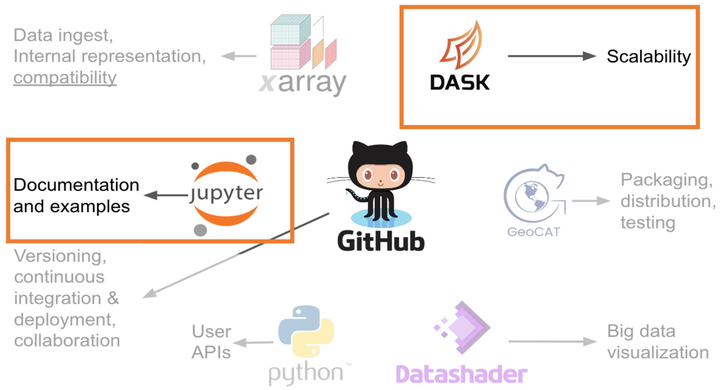


Fig. 2. Package Overview and Core Technologies. This work focuses on implementing Dask for parallelizing UXarray workflows and using Jupyter notebook for documentation and examples.

All functionality operators directly on unstructured grids, including but not limited to:

- Statistical Operators
- Calculus Operators
- Topological Aggregations
- High-resolution Visualization (3.75km)
- Remapping
- Subsetting

Direct computations on unstructured-grid data **reduces overhead and uncertainties from the regridding procedures and avoids duplicating storage consumption.**

Optimized Functionalities

Weighted-Mean

Data variables are typically mapped to geometries that are either nodes, edges, or faces of an unstructured grid. One of the most common operations is taking average over global data.

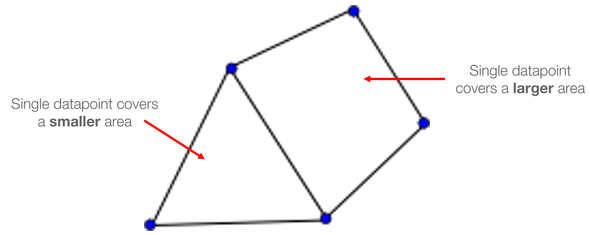


Fig. 3 Simplified grid schematic. Data is stored on the center of each face. When `weighted_mean()` is called, the area of the face will be taken into account.

Data at each geometry may not be equally weighted, as permitted by the highly flexible configuration in unstructured grids. UXarray takes the geometries as weights and would scale the datapoints by face, edge or node and return the global mean.

Preliminary Benchmarking Results

Benchmarking is conducted to evaluate the function's performance for grids at 7.5, 15, and 30km resolution from simulations for DYAMOND, the global storm resolving models intercomparison project (Stevens et al., 2019). Chunking is done automatically for the time and vertical levels, while the geometry dimension remains unchunked for the calculations.

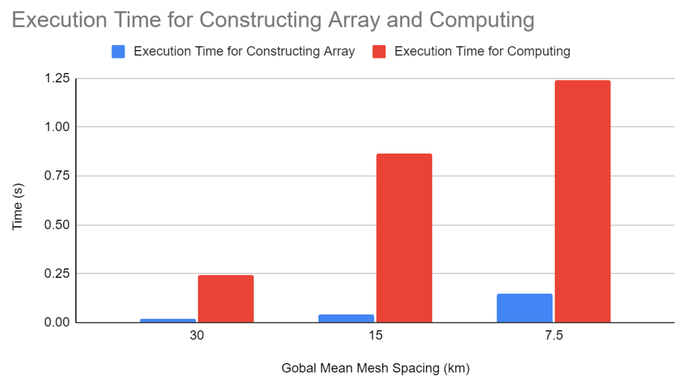


Fig. 4. Execution time for constructing and conducting weighted-mean calculations on Dask arrays of data at 30, 15, and 7.5km resolution.

Main Takeaways

1. The 'heavy-lifting' of the computation mainly lies in calculating the weighted mean, as compared to constructing the array for calculation.
2. The array calculations are delayed to minimze computational time through Dask, as seen by the rapid construction time.
3. The computation scales linearly.

Additional benchmarking of the weighted mean function with varying configuration of the local cluster is also recorded as supplementary data.

Documentation Enhancements

As the package is built for supporting kilometer-scale datasets with terabites of size, Dask is implemented to parallelize operations. A user guide page is dedicated to demonstrate how to use UXarray with Dask for scalable workflows.

```
%%time
# Regular Load
uxds_e3sm_basic_load = ux.open_mfdataset(grid_file, data_files, parallel=False)
```

```
CPU times: user 18.6 s, sys: 244 ms, total: 18.9 s
Wall time: 18.9 s
```

```
%%time
# Parallel Load
uxds_e3sm_parallel_load = ux.open_mfdataset(grid_file, data_files, parallel=True)
```

```
CPU times: user 11.5 s, sys: 1.5 s, total: 13 s
Wall time: 12.7 s
```

Fig. 5. Performance comparison between regular and parallel loading with Dask for E3SM-v2 output at 1 degree resolution (72 timesteps, 21600 faces, 72 vertical levels and 471 variables). Local cluster is set up with 128 workers and 2 threads per worker.

UXarray's analysis and visualization functionalities are also showcased using E3SM-v2 model output. The new documentation presents unstructured grid arithmetic operations, and tests UXarray's performance on large datasets and different unstructured grid convention (relative to MPAS) in the package development process.

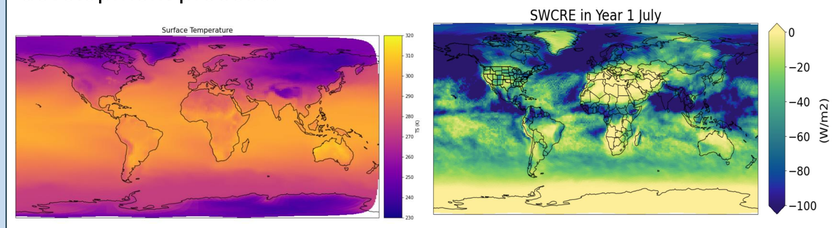


Fig. 6. E3SMv2 simulation with Present Day Control Forcings at 1 Degree Resolution. Left panel shows Year 1 January Surface Temperature. Right panel shows shortwave cloud radiative effect (SWCRE) by doing arithmetic operations with climate output variables on native unstructured grids like net shortwave radiation (FSNT) and outgoing longwave radiation (FLUT).

Future Work

With majority of the package written in Numpy, other existing functionalities like topological aggregations will be translated to Dask for better scalability and memory performances.

Additional benchmarking on the NSF-NCAR supercomputer, Derecho, will be conducted to evaluate the performances for all functionalities in Uxarray with Airspeed Velocity (ASV).

User Guide

GitHub

Acknowledgements

This work part of Project Raijin, funded by the National Science Foundation under Grant No. 2126458. A great thank you to my mentors Philip and Orhan for their guidance, support, patience and understanding on the project and beyond. Thank you NCAR, CISL and SIParCS, particularly Virginia, Jessica and Eva, for all resources and facilitation throughout the SIParCS internship that made this project all possible.