

Post-processing and visualizing MPAS-Atmosphere output

Michael G. Duda
NSF NCAR/MMM



NCAR
OPERATED BY UCAR

Post-processing overview

Now that you've run MPAS-Atmosphere, how can you take a graphical look at the output?

```
diag.2010-10-23_00.00.00.nc    history.2010-10-23_00.00.00.nc
diag.2010-10-23_03.00.00.nc    history.2010-10-23_06.00.00.nc
diag.2010-10-23_06.00.00.nc    history.2010-10-23_12.00.00.nc
diag.2010-10-23_09.00.00.nc    history.2010-10-23_18.00.00.nc
diag.2010-10-23_12.00.00.nc    history.2010-10-24_00.00.00.nc
diag.2010-10-23_15.00.00.nc
diag.2010-10-23_18.00.00.nc    restart.2010-10-24_00.00.00.nc
diag.2010-10-23_21.00.00.nc
diag.2010-10-24_00.00.00.nc
```

Above: Typical output files from an MPAS-Atmosphere simulation

1. Interpolate to a regular lat-lon grid
2. Visualize output directly with Python

What's in these output files, anyway?

By default, the “diag” files contain:

RH, T, height, winds @ 200, 250, 500, 700, 850, 925 hPa

CAPE, CIN, LCL, LFC, updraft helicity

U10, V10, T2, Q2

Simulated radar reflectivity

PMSL

Surface, 1km AGL, 6km AGL winds

(various other 2-d fields)

In the "Computing new diagnostics" lecture, we'll say more about the framework for adding new diagnostics to MPAS-A.

What's in these output files, anyway?

By default, the "history" files contain:

q_v , q_c , q_r , ...

theta

zonal, meridional wind

vertical velocity

full pressure

dry density

accumulated rain (cumulus and microphysics)

soil moisture, soil temperature

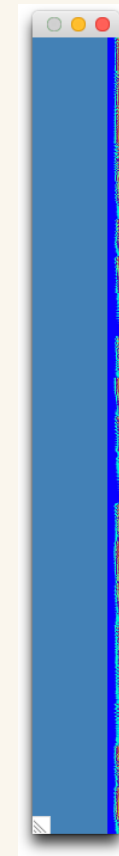
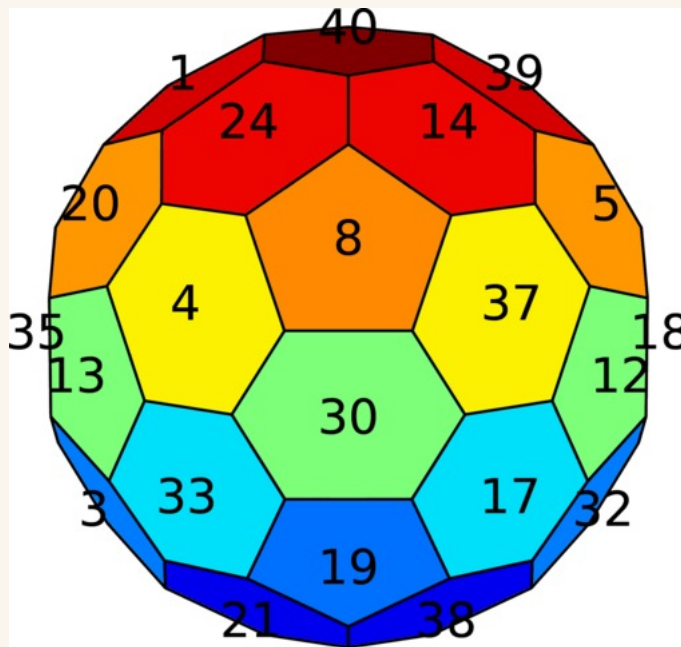
(various other fields)

Full mesh information (horizontal and vertical)

In the "Running MPAS, part 2" talk, we discussed how to modify the set of fields written to model output files using *streams*

Interpolating output to a regular lat-lon grid

MPAS stores 2-d horizontal fields in 1-d arrays; 3-d fields are 2-d arrays with the vertical dimension innermost (contiguous), e.g., `qv(nVertLevels, nCells)`.

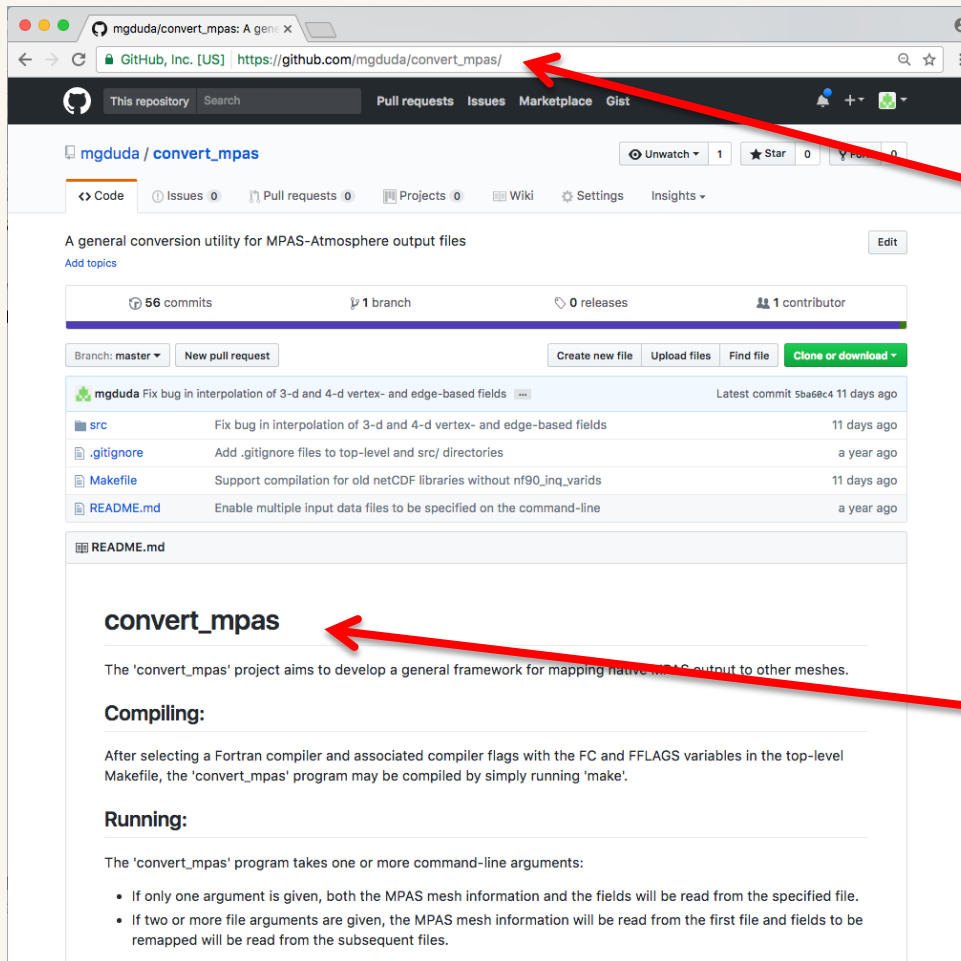


Left: Can you find Hurricane Matthew in the MPAS 'qv' field seen in ncview?

Using 'ncview' directly on MPAS netCDF files doesn't work well...

Interpolating output to a regular lat-lon grid

The `convert_mpas` tool can quickly interpolate MPAS files to a specified lat-lon grid



Source code can be obtained from

https://github.com/mgduda/convert_mpas/

The README.md file summarizes the key details of compiling and running

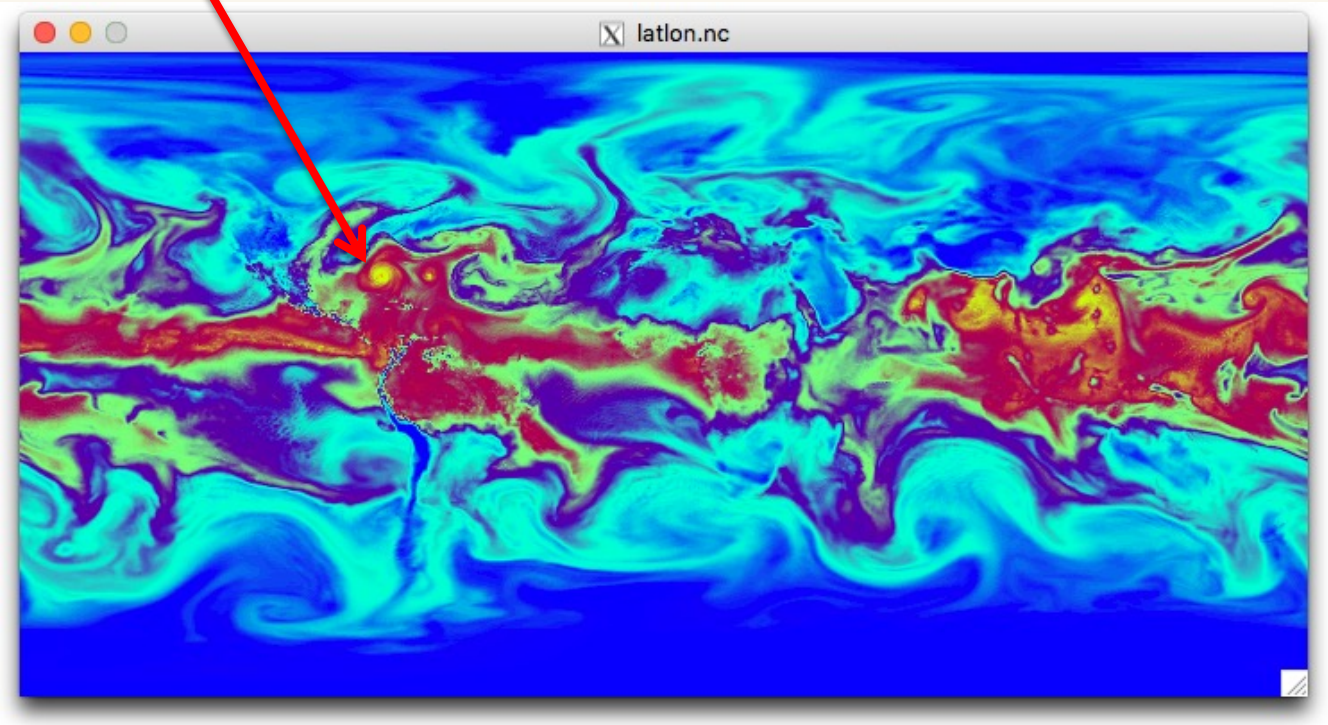
The *convert_mpas* utility

Basic usage of `convert_mpas`:

- If just one argument is given, it specifies an MPAS file that has mesh information as well as fields to be interpolated
 - E.g.: `convert_mpas x1.40962.init.nc`
- If more than one argument is given:
 - First argument is used *only to obtain mesh information*
 - All remaining arguments contain fields to be interpolated
 - E.g.: `convert_mpas x1.40962.grid.nc diag*nc`
 - E.g.: `convert_mpas history.2017-06-16_00.nc history*nc`
- Output file is always called `latlon.nc`
 - Probably best to remove this file before re-running `convert_mpas`
- Default output grid is 0.5-degree lat-lon grid

The *convert_mpas* utility

Now we can see Hurricane Matthew in our MPAS output

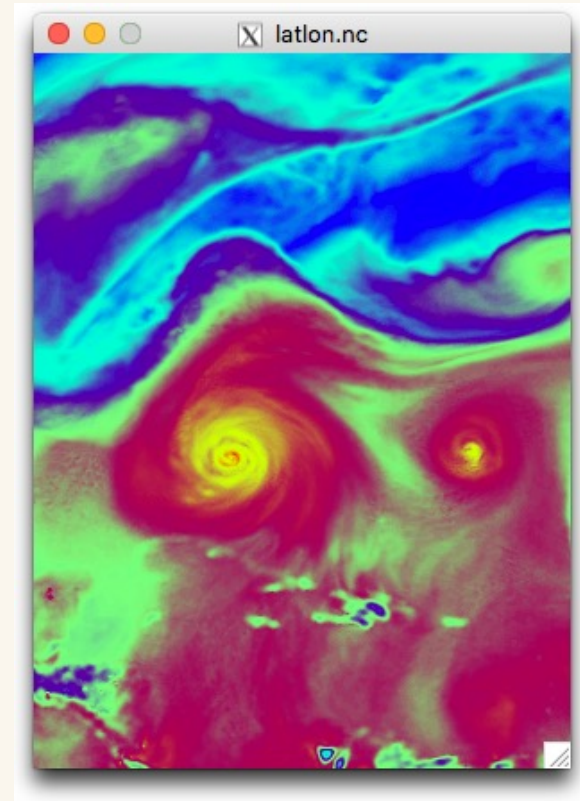


How can we interpolate to just the region of interest and at higher resolution?

The *convert_mpas* utility

A text file named `target_domain` in your working directory may be used to specify parameters of the lat-lon grid:

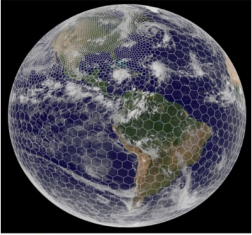
```
startlat=10.0  
endlat=50.0  
startlon=-90.0  
endlon=-60.0  
nlat=400  
nlon=300
```



A text file named `include_fields` in your working directory may also be used to list the fields that should be interpolated

To plot fields directly from the native MPAS mesh, try Python or NCL

[Home](#)
[News & Events](#)
[Contact](#)
[Support Forum](#)
[Site Map](#)




MPAS Atmosphere

Plotting MPAS-A Global Simulations +

- Documentation +
- Access Code +
- GPU-enabled MPAS-A +
- Visualization
- News & Events
- Publications

Search this project

[Home](#) / [Plotting MPAS-A Global Simulations](#)
[View source](#)
[GitHub](#)

Plotting MPAS-A Global Simulations

Plotting with Python

Python is NSF NCAR's primary language for scripting and visualization. See the [MPAS Plotting](#) GitHub repository that hosts examples and tutorials on using Python to create MPAS-A visualizations.

There, you can find examples of visualization scripts and references to Python visualization modules, including the 'mpas-patches/mpas_patches.py' script, which can be useful for plotting individual MPAS grid cells (instead of interpolating to a rectangular latitude and longitude grid).

NCL Scripts

Note

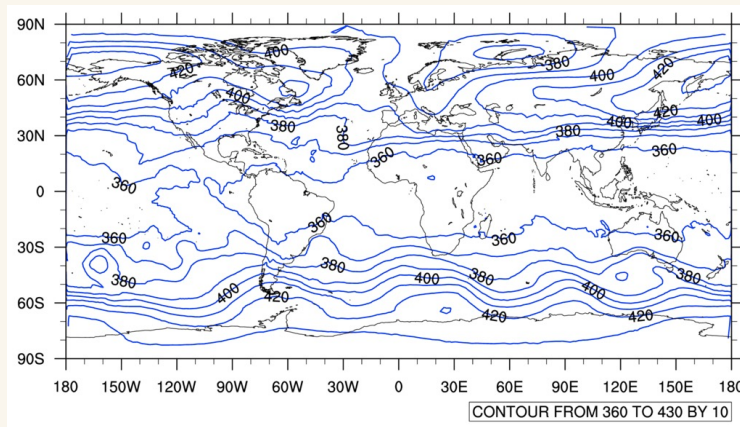
NSF NCAR has [discontinued further development of NCL](#)

The [NCAR Command Language](#) (NCL) is an interpreted language netCDF input and output files. Several example scripts are provided.

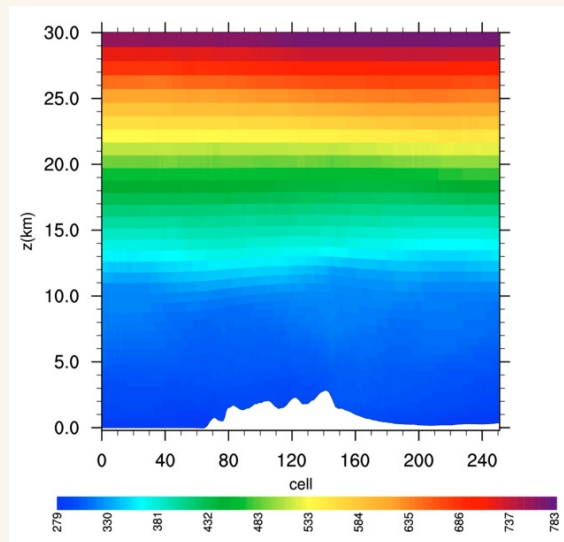
In the practical exercises, we will use only Python scripts!

(NCL use for MPAS-A is being phased out)

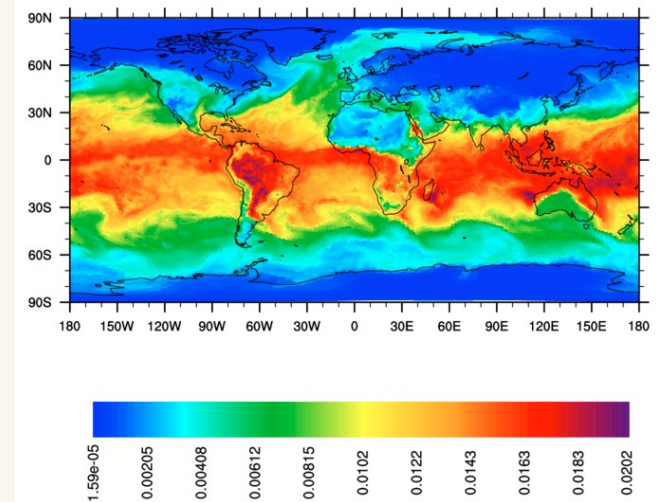
Example NCL scripts from the MPAS-Atmosphere downloads page



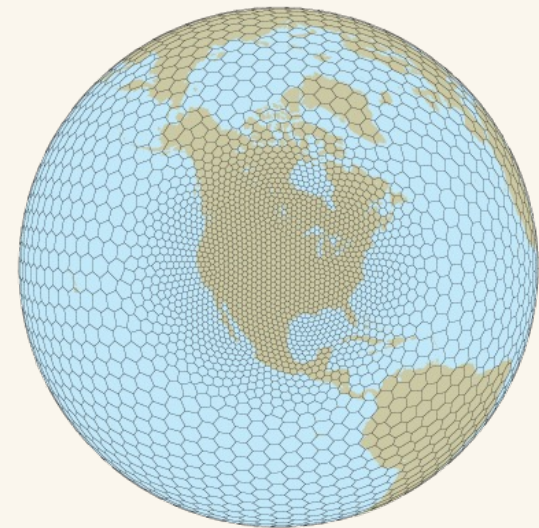
Contours – simple or color-filled



*Vertical cross-sections with specified
endpoints*



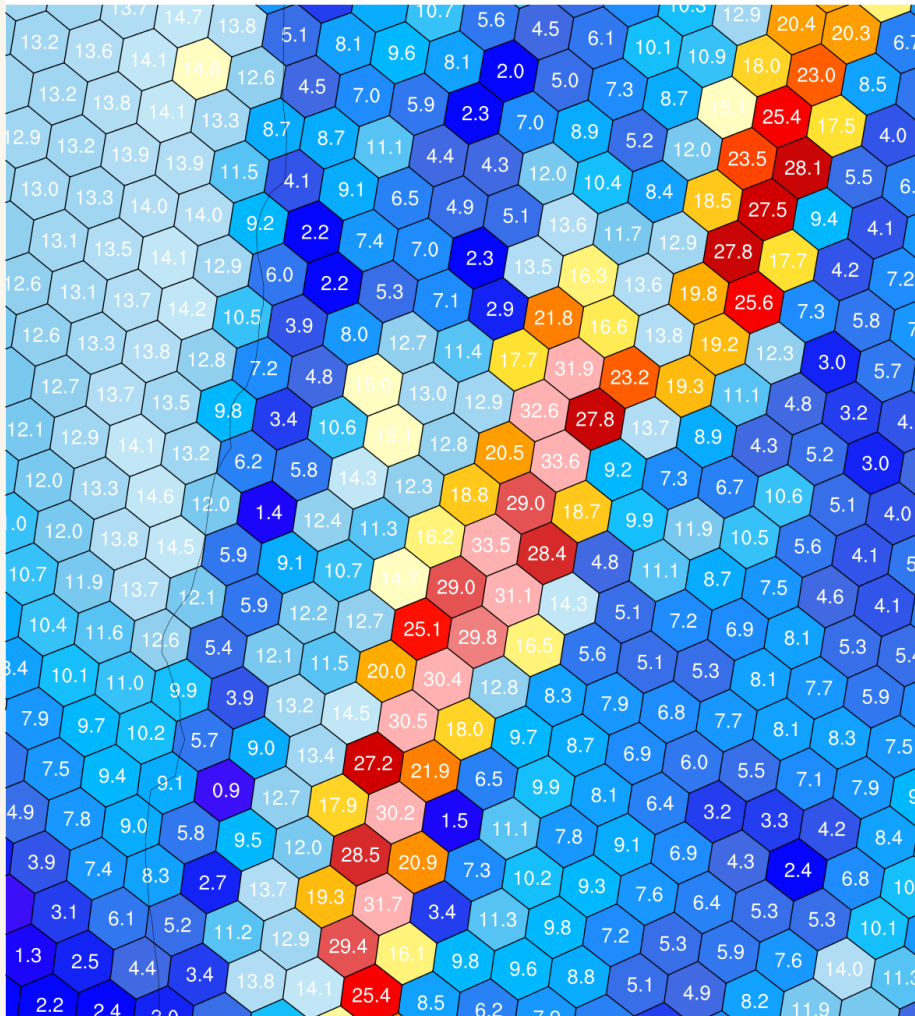
Individual grid cells as a color-filled polygons



Voronoi mesh against a map background

Plotting values on cells is also possible

wind speed @ k=1 [m s⁻¹]



Given *latVertex*, *lonVertex*, *verticesOnCell*, and *nEdgesOnCell*, we can plot each MPAS Voronoi cell as a color-filled polygon

- Overlaying numeric values can be quite helpful in debugging

Making use of the MPAS mesh representation to more efficiently work with MPAS output

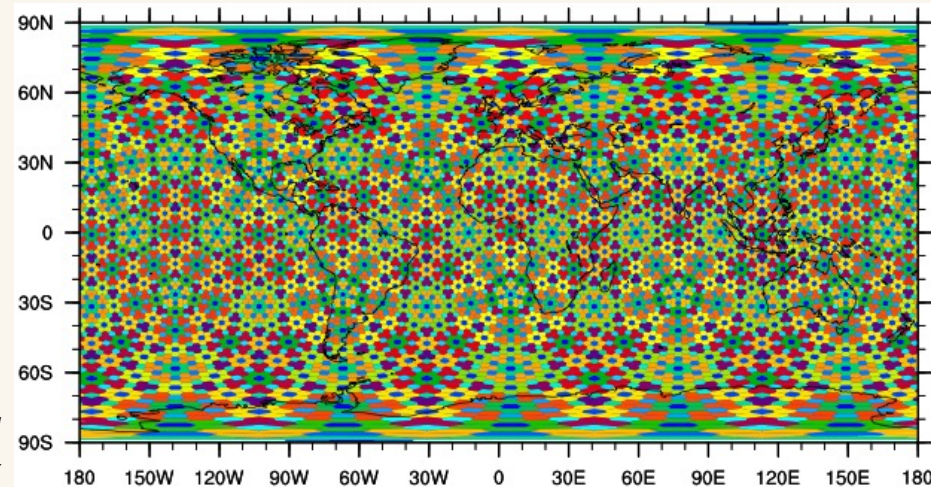
In many limited-area models, finding the nearest grid cell to a given (lat,lon) location is a constant-time operation:

1. Using the map projection equations for the model grid projection, compute the real-valued (x,y) coordinates of the (lat,lon) location
2. Round the real-valued coordinates to the nearest integer

However, in MPAS, *there is no projection*, and the horizontal cells may be indexed in any order.

- We could just compute the distance from (lat,lon) to every cell center in the mesh and choose the nearest cell, or we could do something more efficient...

Right: Cells in the x1.10242 mesh colored according to their global index



Making use of the MPAS mesh representation to more efficiently work with MPAS output

One solution would be to use search trees – perhaps a *kd*-tree – to store the cells in a mesh

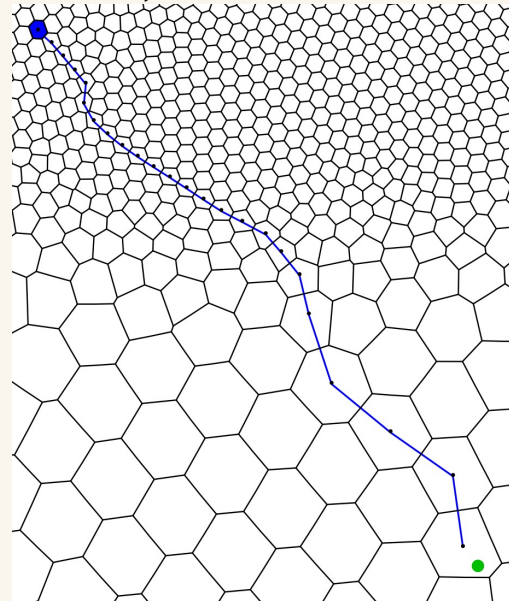
- $O(n \log n)$ setup cost; each search takes $O(\log n)$ time, for a mesh with n cells

Alternatively, we can make use of the grid connectivity arrays `nEdgesOnCell` and `cellsOnCell` to navigate a path of monotonically decreasing distance to the (lat,lon) location

- No setup cost, $O(n^{1/2})$ cost per search (depending on mesh geometry...)
- For successive searches of “nearby” locations, almost constant cost!

```

C_nearest = any starting cell
C_target = NULL
do while (C_nearest ≠ C_target)
    C_target = C_nearest
    d = distance from C_target to (lat,lon)
    for i = 1 to nEdgesOnCell(C_target)
        k = cellsOnCell(i, C_target)
        d' = distance from k to (lat,lon)
        if ( d' < d )
            d = d'; C_nearest = k
    
```



Left: Path taken from starting cell (blue) to target location (green circle).

Making use of the MPAS mesh representation to more efficiently work with MPAS output

Problem: Scan all cells within a specified radius of a given (lat,lon) location

Option 1: We could check all cells in the mesh (very inefficient!)

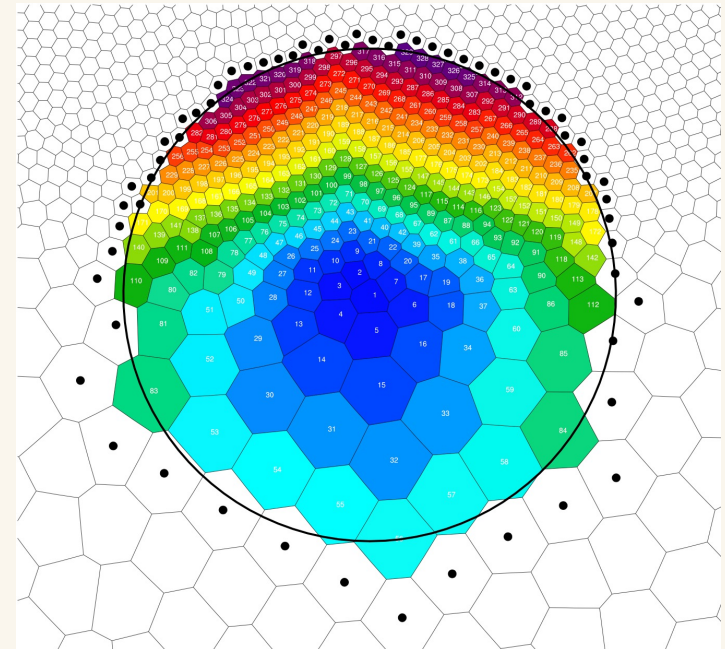
Option 2: We could make use of the connectivity arrays (efficient!)

```

C = origin of the search
mark C as visited
insert C into the queue
do while (queue not empty)
    C = next cell from the queue

    C is within search radius, so process C

    for i = 1 to nEdgesOnCell(C)
        k = cellsOnCell(i,C)
        if ( k not already visited )
            mark k as visited
            if (k within search radius)
                insert k into the queue
    
```



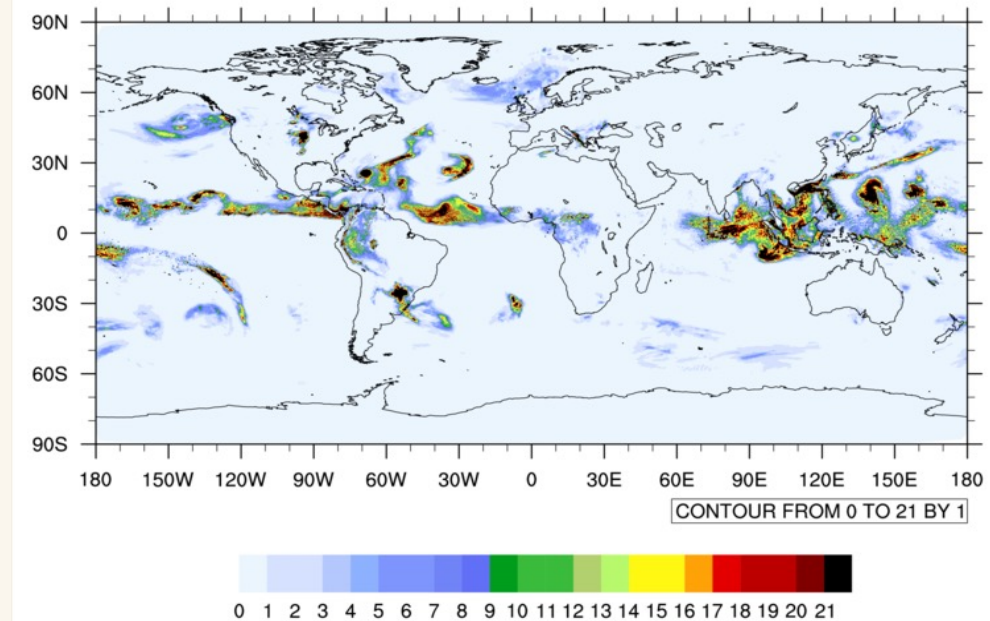
Above: Cells shaded according to the order in which they were visited by a 750-km radius search; dots indicate cells that were considered but found to be at a radius >750 km.

Important considerations for post-processing on variable-resolution meshes

Consider the computation of the daily mean precipitation rate on a variable-resolution MPAS mesh:



Above: An MPAS 60-15 km variable-resolution mesh with refinement over North America



Above: The accumulated total precipitation between 2016-10-14 00 UTC and 2016-10-15 00 UTC on from MPAS with the 'mesoscale_reference' physics suite.

How much can the way in which we compute the daily precipitation rate affect our results?

An obvious conclusion, but one that's important...

Taking a simple average of the precipitation rate in all cells gives **3.43 mm/day**

In an MPAS simulation with a variable-resolution mesh with a refinement factor of **N** (e.g., $N=4$ for a 60-15 km mesh), the cell area ratio between the largest and smallest cells in the mesh is **N^2** !

```
f1 = addfile("diag.2016-10-14_00.00.00.nc", "r")
f2 = addfile("diag.2016-10-15_00.00.00.nc", "r")
fld = (f2->rainc(0,:) + f2->rainnc(0,:)) -
      (f1->rainc(0,:) + f1->rainnc(0,:))
fg = addfile("init.nc", "r")
print(sum(fld * fg->areaCell(:)) / sum(fg->areaCell(:)))
```

Weighting the precipitation rate by cell area gives **2.93 mm/day**