An update on model development

Ross Heikes, C.S. Konor and D. Randall

Dept. of Atmospheric Science
Colorado State University

CMMAP summer meeting August 6, 2013

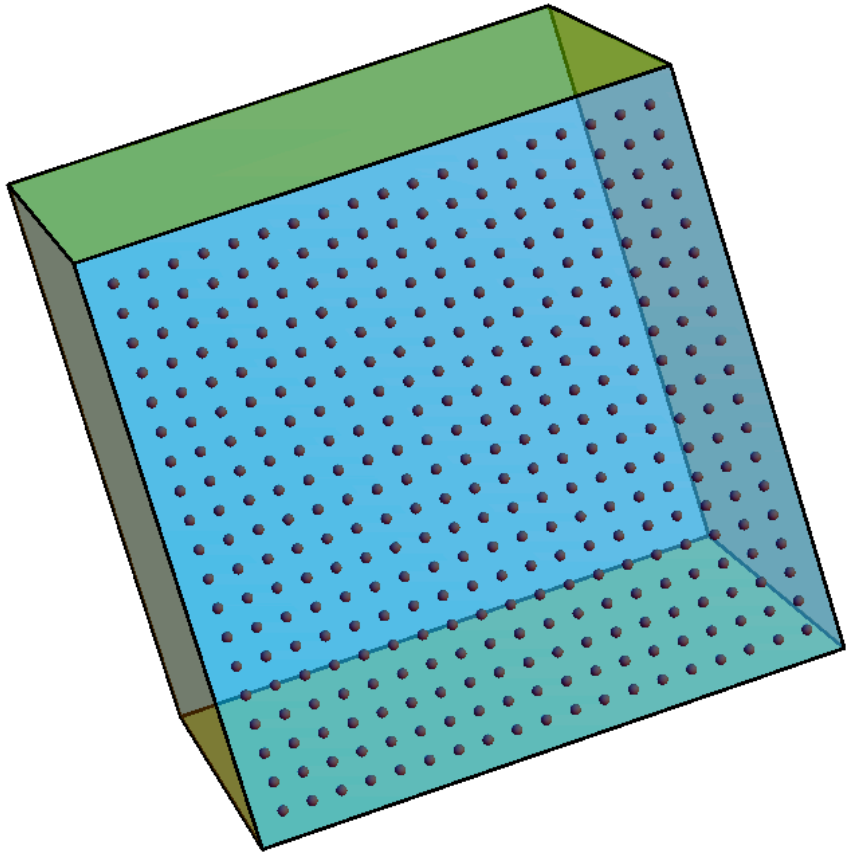
Outline

- Several unrelated little parts:

1. Grid generation on the cubed sphere grid
2. A new way to calculate wind in the icosahedral grid model
3. Experiences on the NSF Blue Waters computer
    a. Parallel scaling of the MPI portion of the model
    b. Experiences (so far) with the accelerators
4. Conjugate gradient method to solve 3D elliptic equation on the icosahedral grid
5. Comparison of the Tweaked Grid to Spring Grid and Centroidal Voronoi Tessellations (CVT) Grid

The cubed sphere grid

- A Cubed sphere grid is formed by projecting a cube onto a sphere.
- A logically quadrilateral grid remain logically quadrilateral.

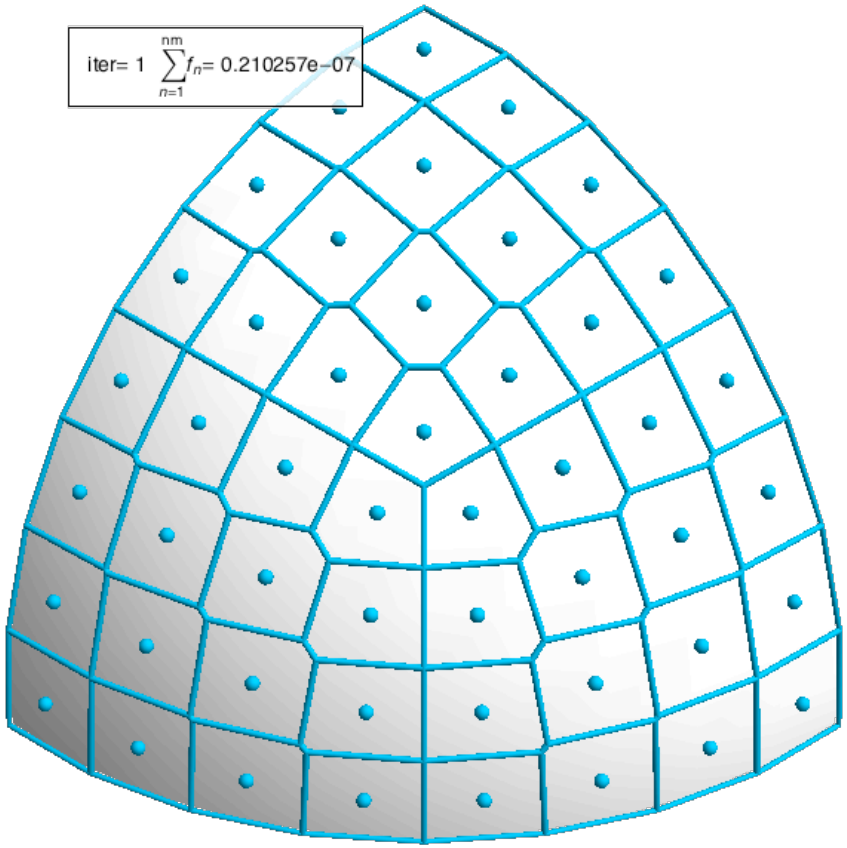


The cubed sphere grid with Voronoi cells

- A Voronoi grid on the sphere is defined like this:

  Given an arbitrary set of points on the sphere, the *Voronoi cell* associated with a particular grid point, say p_0, consists of the points on the sphere closer to p_0 than to any other grid point.

- Nice properties include orthogonality and enhanced isotropy.
- A Voronoi grid for the icosahedral grid is easy.
- A Voronoi grid generated on the cubed sphere does not consist of cells with only 4 wall.



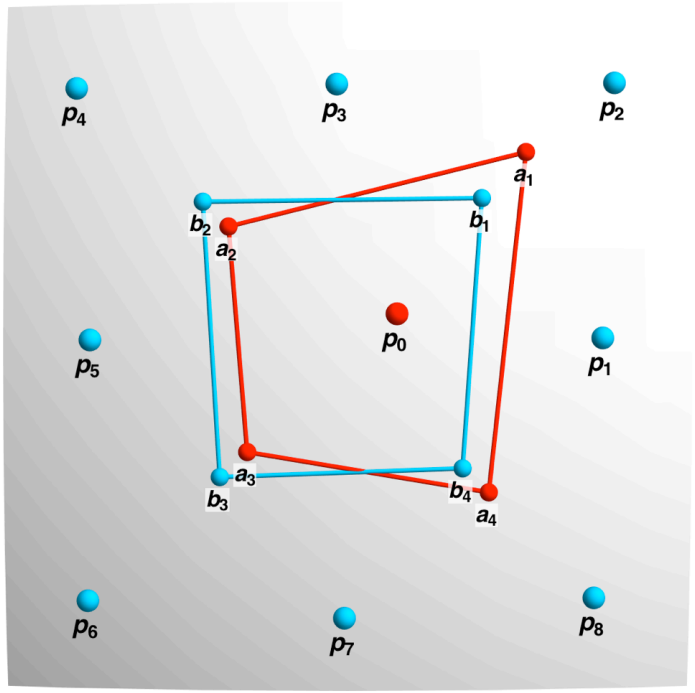
An algorithm to generate a voronoi grid on the cubed sphere

- Consider 9 grid points $p_0, p_1, \ldots, p_9$ on the unit sphere.
- We can consider p_0 to be *home base*.
- Define a function $v(p_{n_1}, p_{n_2}, p_{n_3})$ that given 3 points returns the point on the sphere equidistant from the 3 points. This is the Voronoi corner.
- Define a cell (red lines and corners) using p_0:

$$a_1 = v(p_0, p_1, p_3),\ a_2 = v(p_0, p_3, p_5),\ a_3 = v(p_0, p_5, p_7) \text{ and } a_4 = v(p_0, p_7, p_1)$$

- Define a cell (blue lines and corners) using surrounding grid points:

$$b_1 = v(p_1, p_2, p_3),\ b_2 = v(p_3, p_4, p_5),\ b_3 = v(p_5, p_6, p_7) \text{ and } b_4 = v(p_7, p_8, p_1)$$



We want to move p_0 so that the red cell becomes more coincident with the blue cell.

- Define a cost function

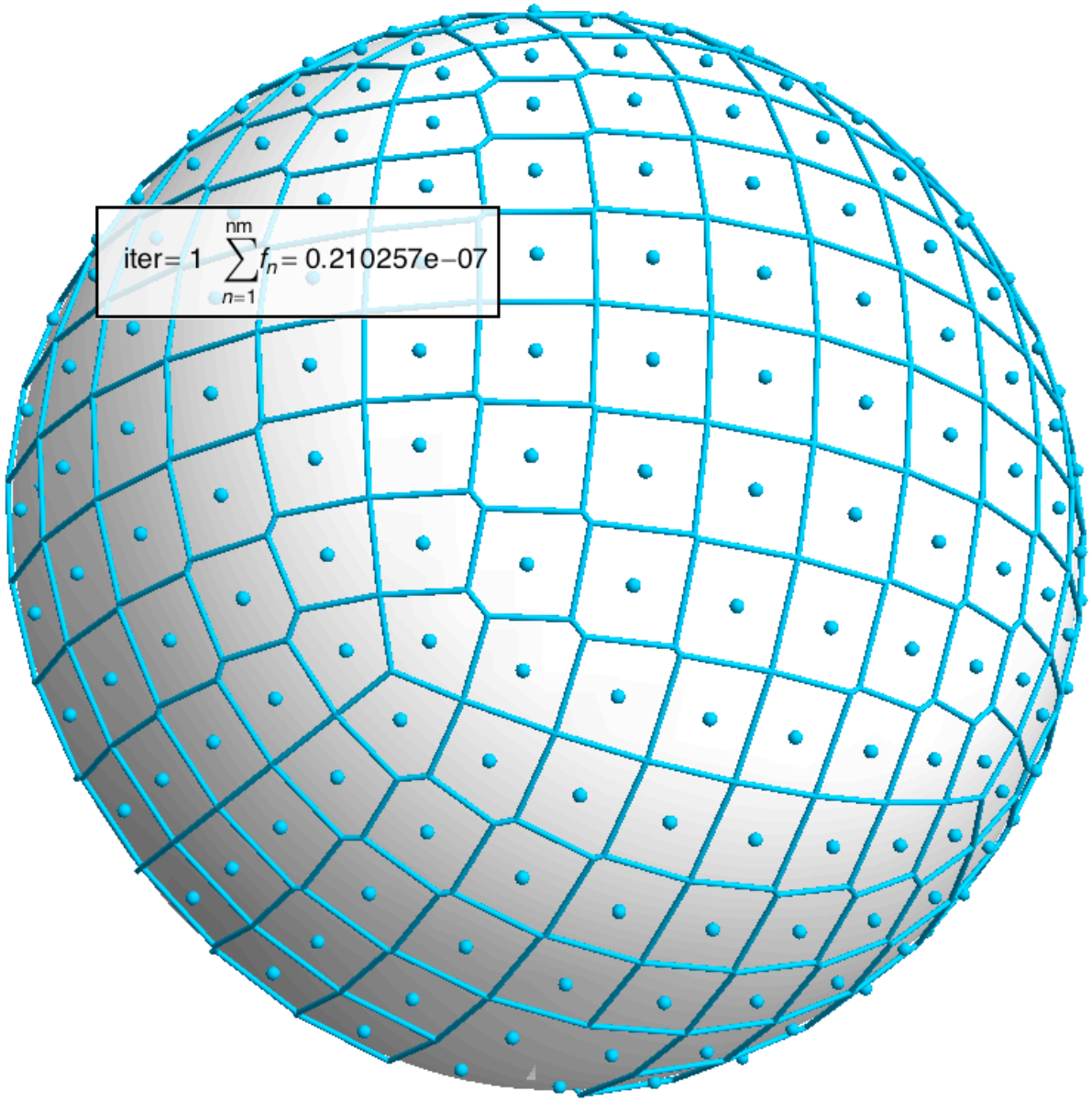
$$f(p_0) = \sum_n \delta(a_n, b_n)^4$$

where $\delta(a_n, b_n)$ is the distance from a_n to b_n.

- We minimize the cost for each cell and loop over all the cells.

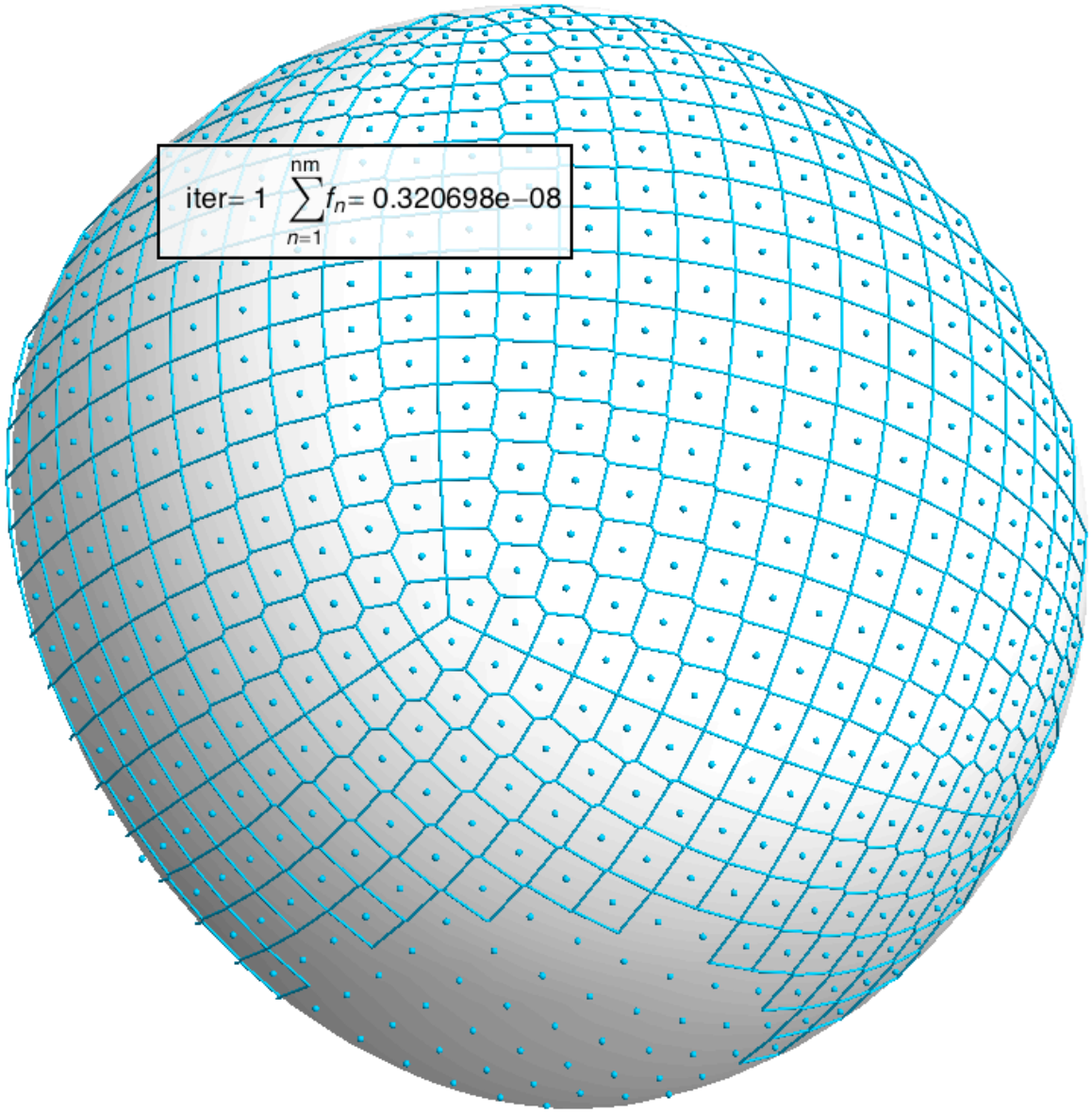
A possible algorithm to generate a voronoi grid on the cubed sphere

- Level 3. 384 cells.
- 24 iterations of the algorithm



A possible algorithm to generate a voronoi grid on the cubed sphere

- Level 4. 1536 cells.
- 24 iterations of the algorithm

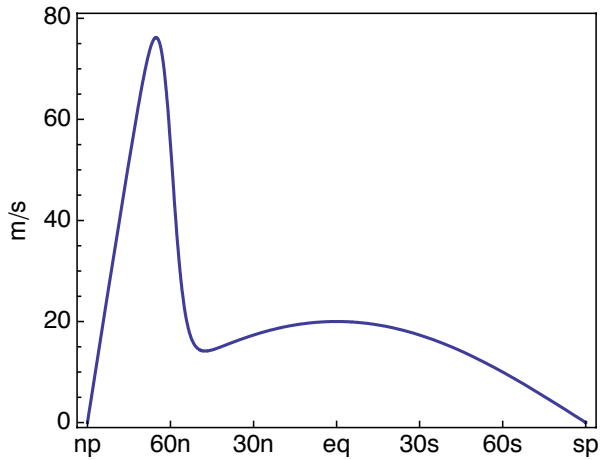


Barotropic vorticity experiment. Initial condition.

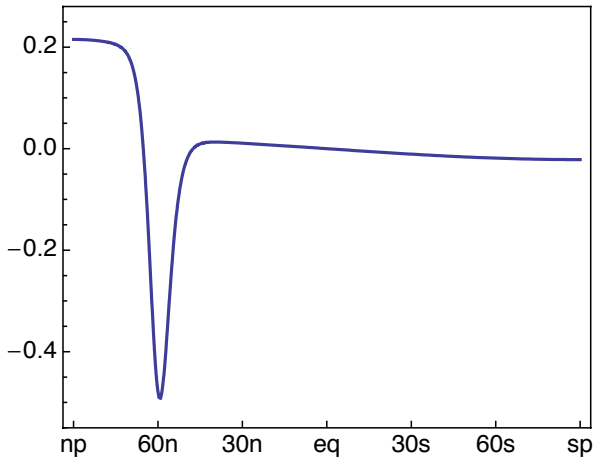
- Predict the absolute vorticity

$$\frac{\partial \eta}{\partial t} - \nabla \bullet \left(\eta \mathbf{v}_{\psi} \right) = 0 \qquad \text{where} \qquad \nabla^2 \psi = \eta - f$$

- Construct a zonal velocity that rapidly transitions between two profiles of solid body rotation.



zonal velocity

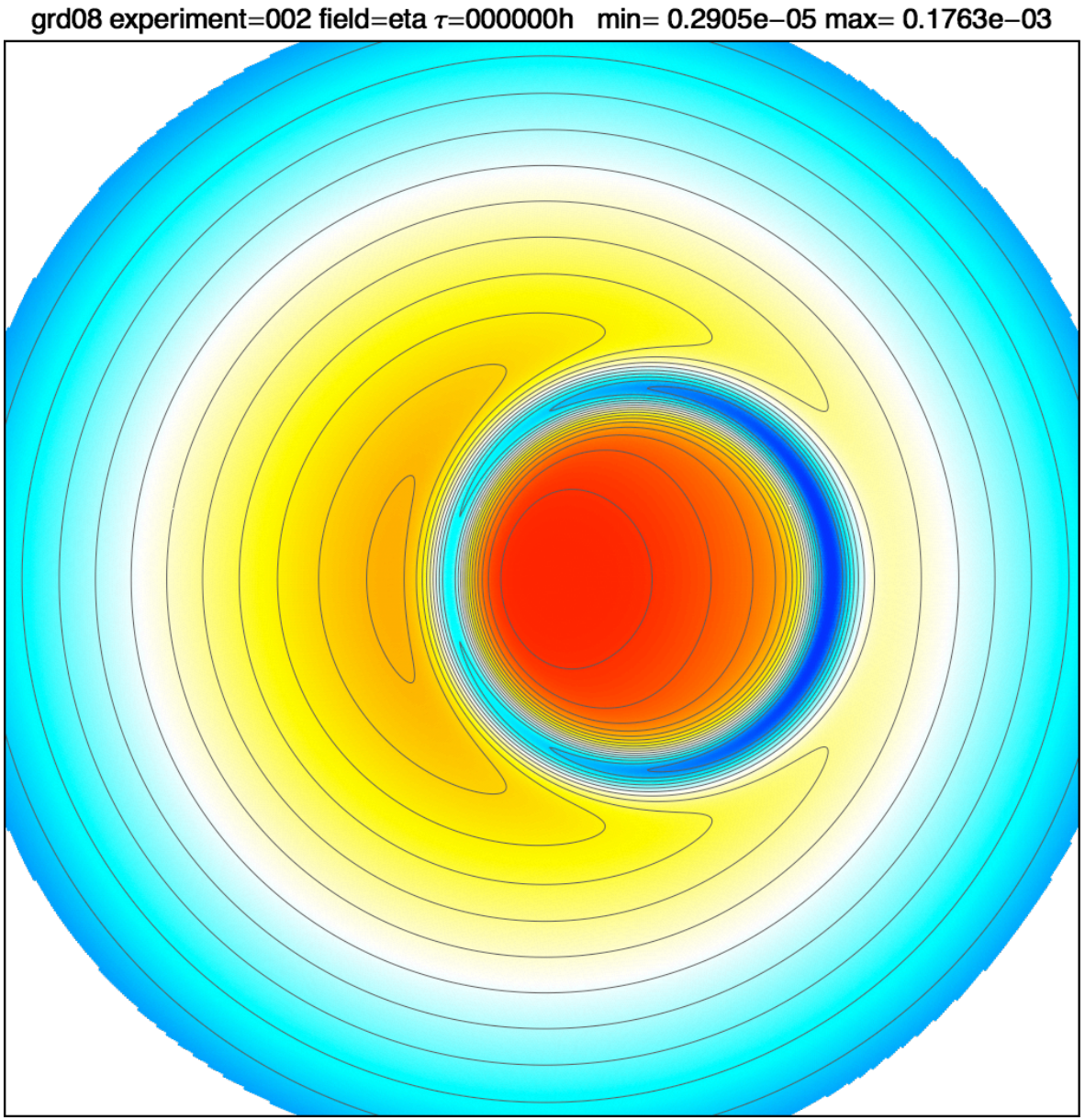


normalized relative vorticity

- Rotate the field so that it is no longer rotationally symmetric with respect to the poles.

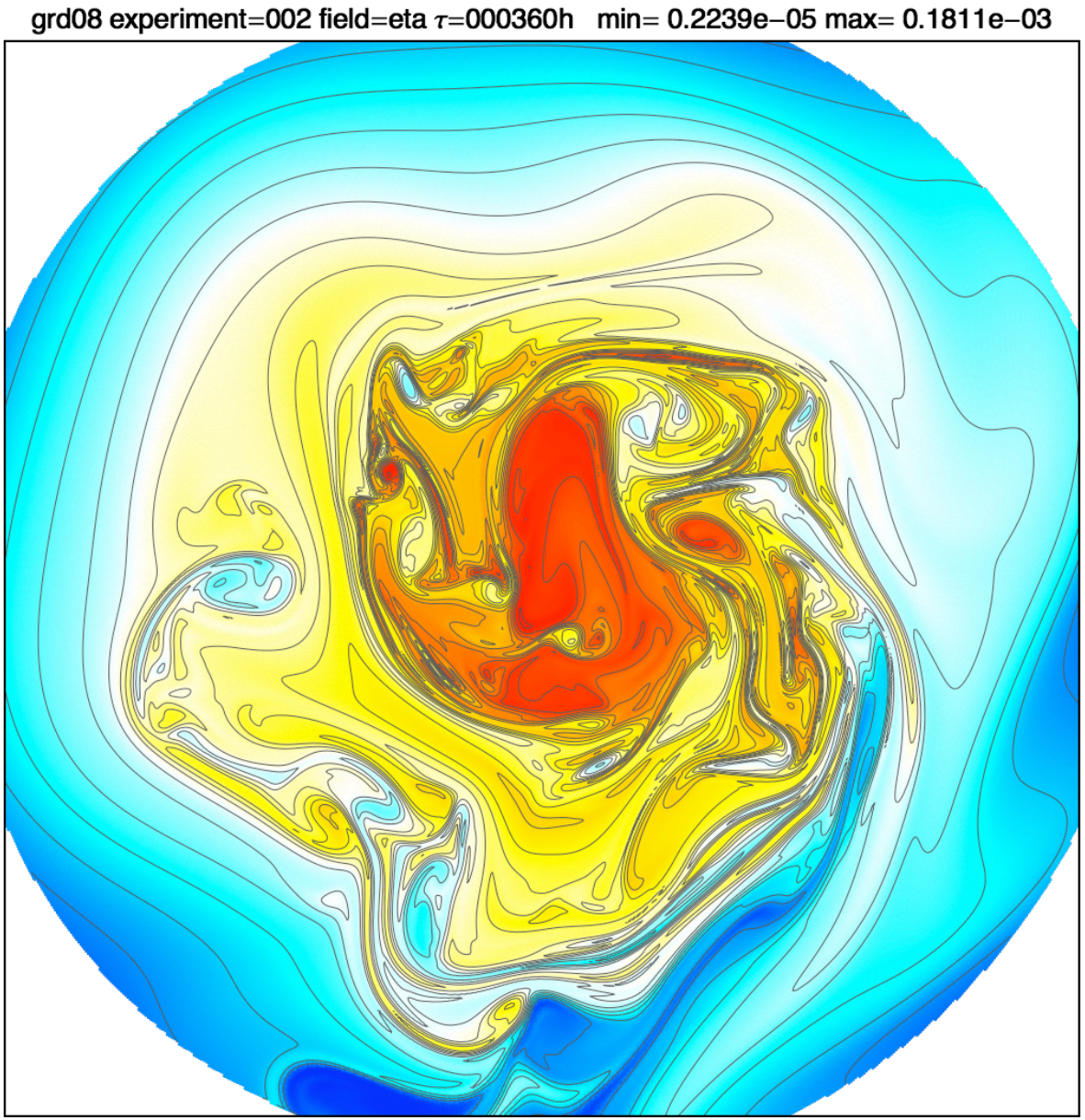
Barotropic vorticity experiment. Initial condition.

- the simulation with 655842 cells (32 km) for 27 days
- View point is the north pole
- Absolute vorticity



Barotropic vorticity experiment. Day 15.

- the simulation with 655842 cells (32 km) for 27 days
- View point is the north pole
- Absolute vorticity



Wind at cell edges as a function of stream function

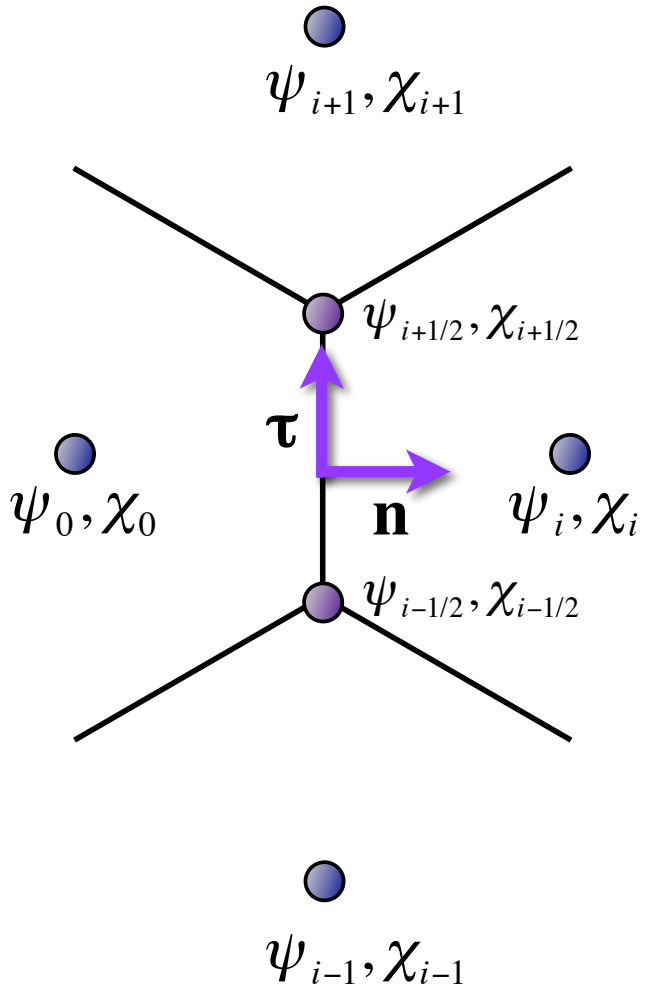
- Helmholtz decomposition

$$\mathbf{v} = \mathbf{k} \times \nabla\psi + \nabla\chi = \mathbf{v}_\psi + \mathbf{v}_\chi$$

- Here we ignore the divergent part
- Interpolate $\psi_{i+1/2}$ to cell corners from surrounding cell centers
- Vector wind (rotational) at edges as a function of stream function:

$$\left(\mathbf{v}_\psi\right)_i = \frac{\psi_{i+1/2} - \psi_{i-1/2}}{l}\mathbf{n} + \frac{\psi_i - \psi_0}{L}\boldsymbol{\tau}$$

where l is the length of a cell wall and L is the distance between cell centers.

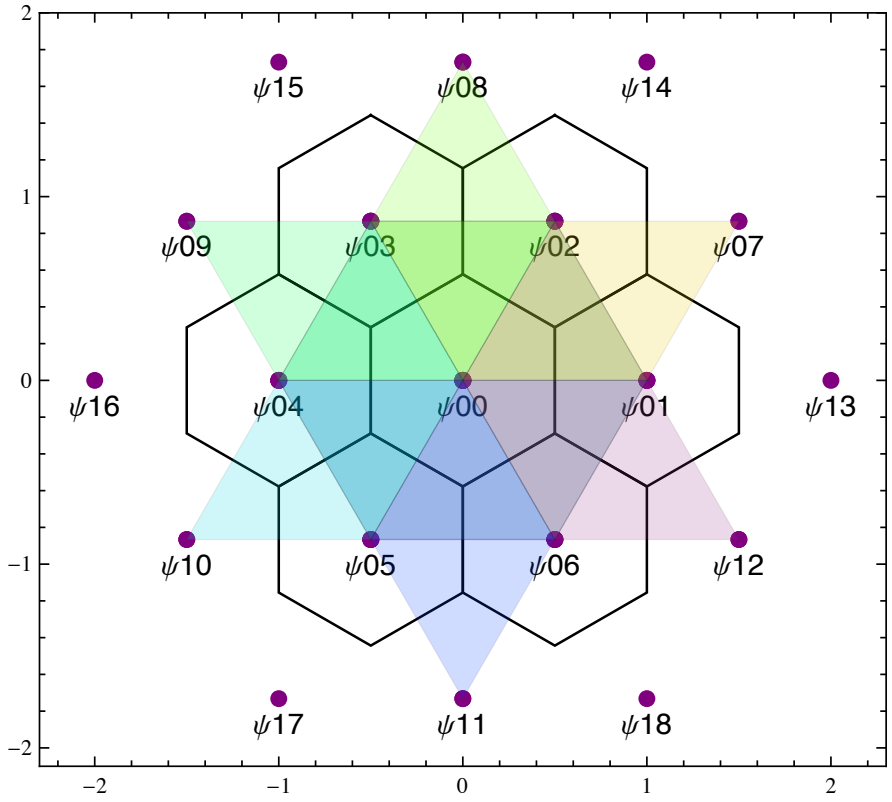


Construct interpolated values at corners

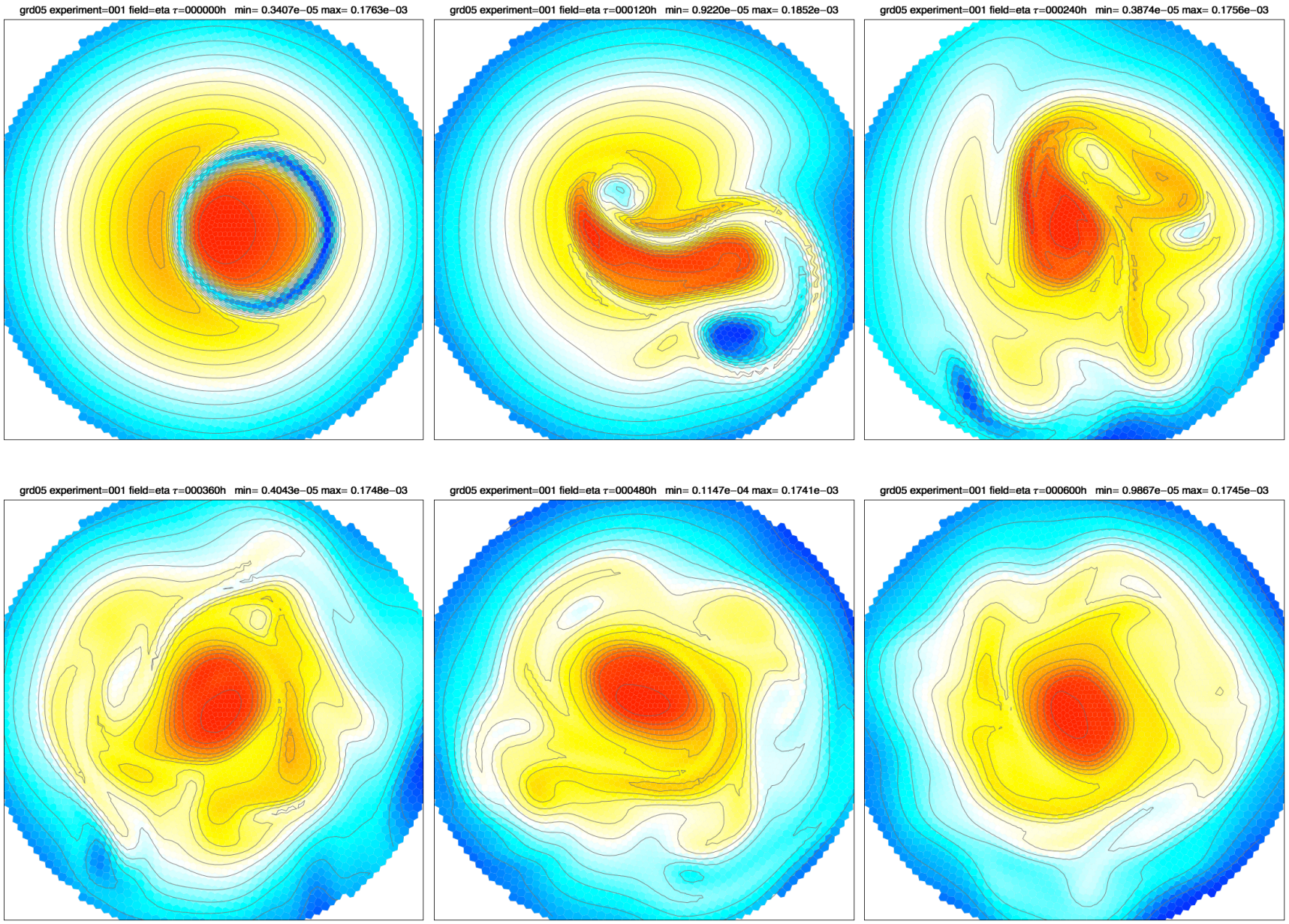
- For We can construct the values at corners in several ways. For example,
- A very **simple** interpolation:

$$\psi_{3/2} = \frac{1}{3}\left(\psi_{00} + \psi_{01} + \psi_{02}\right)$$

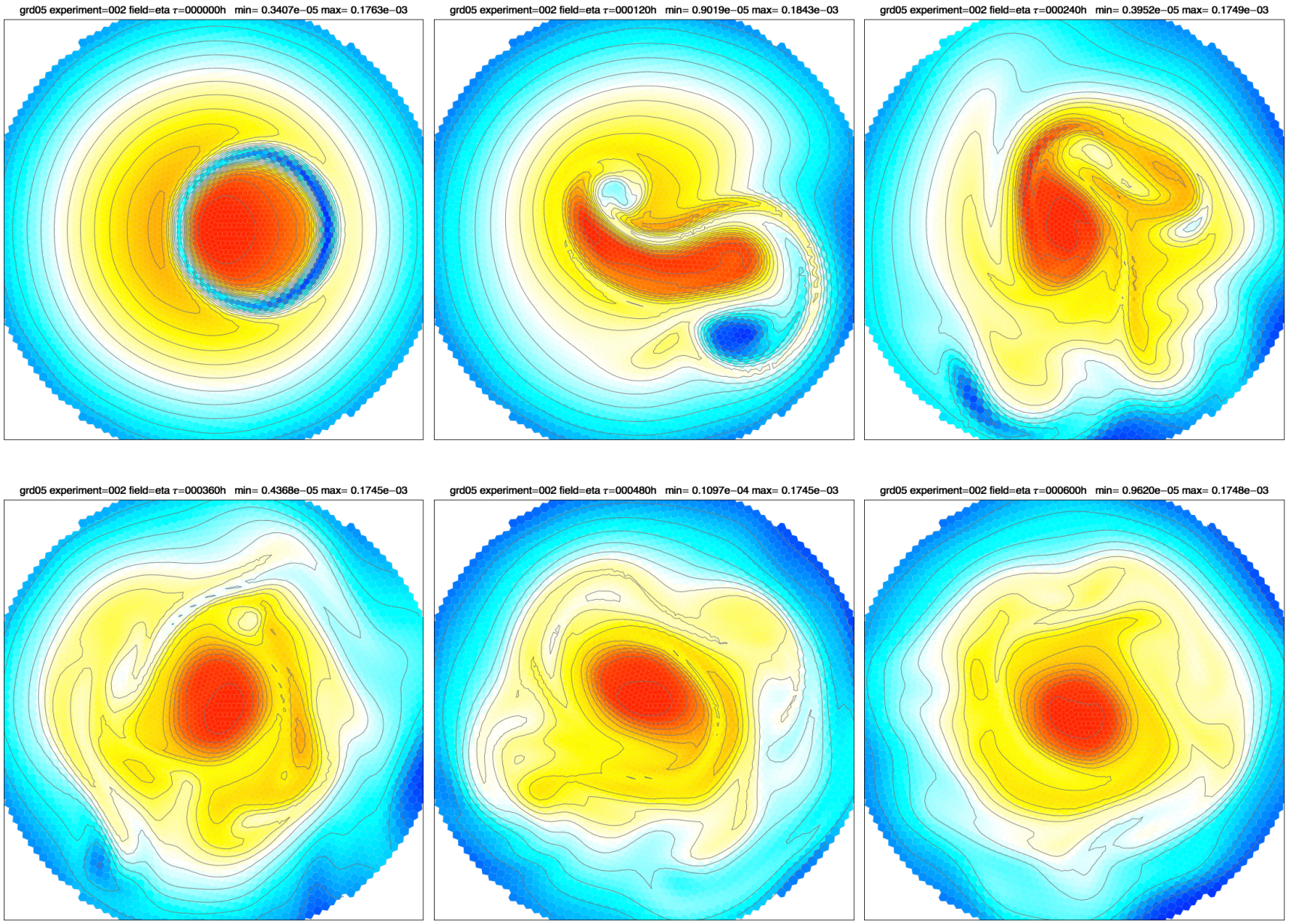
- Or we can construct a **quadratic** interpolation using data from six cell centers.
  For example, for corner $\{\psi_{00}, \psi_{01}, \psi_{02}\}$ we can use $\{\psi_{00}, \psi_{01}, \psi_{02}, \psi_{06}, \psi_{07}, \psi_{03}\}$.



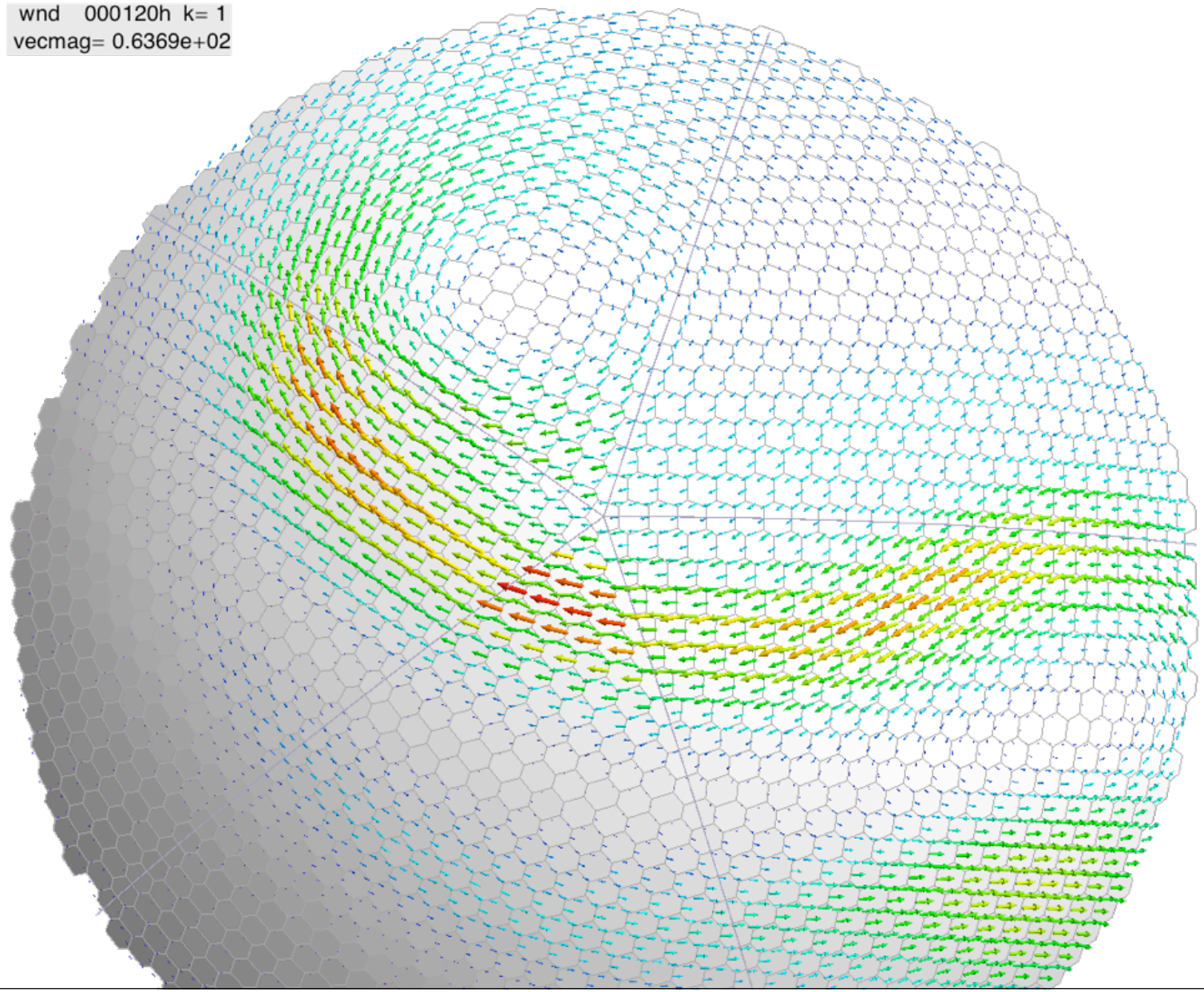
Simple interpolation. Absolute vorticity.



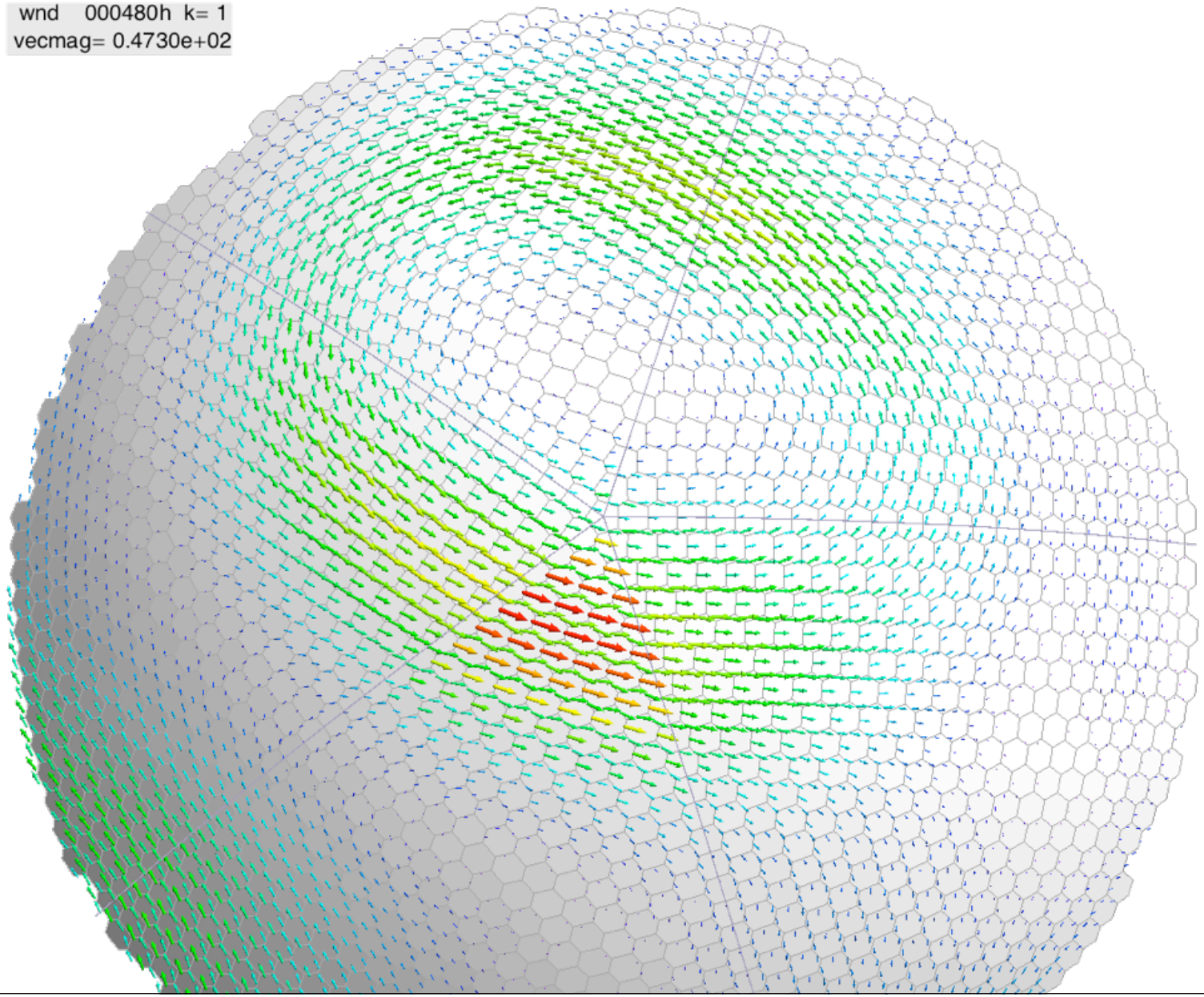
Quadratic interpolation. Absolute vorticity.



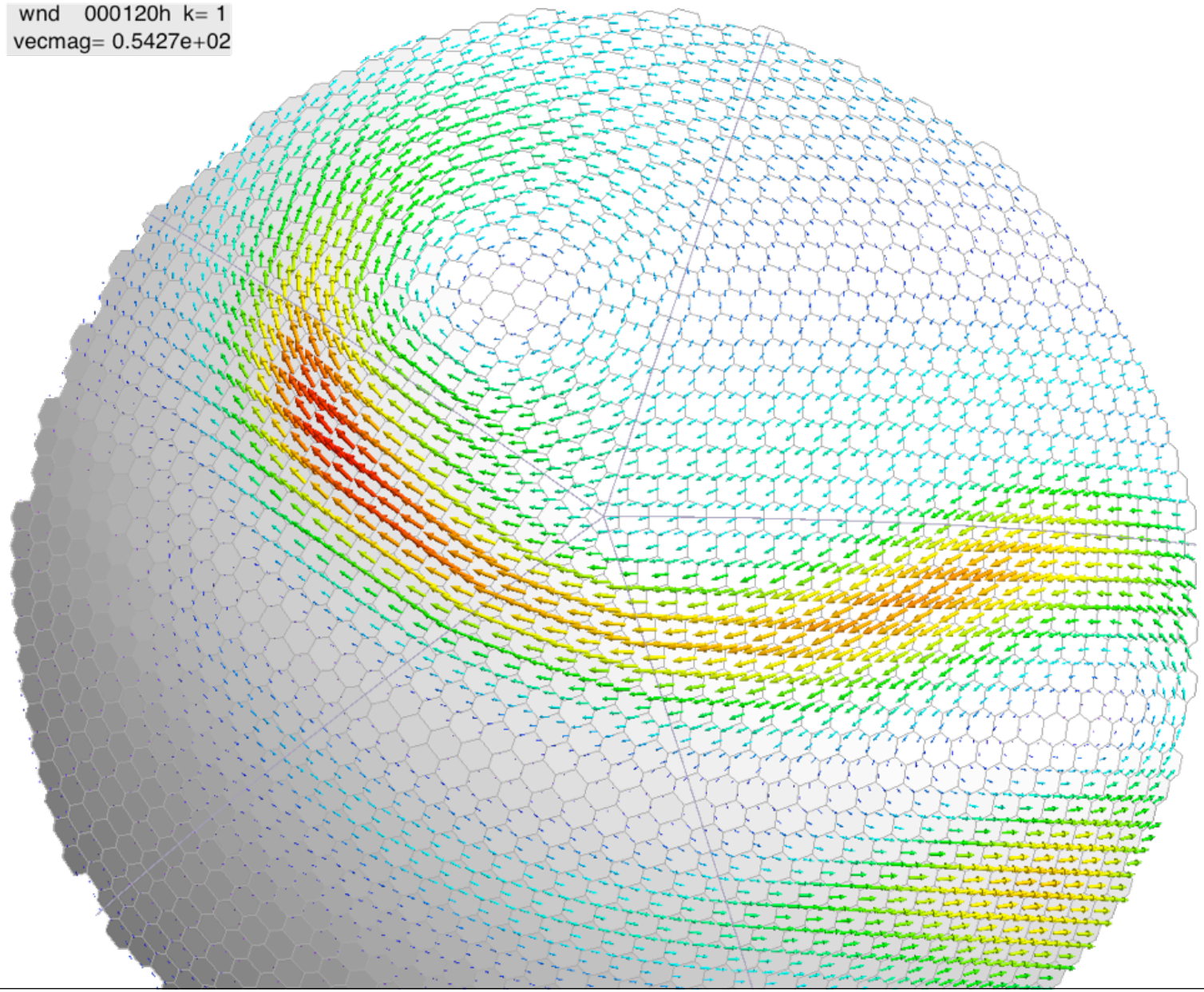
Simple interpolation. Wind vectors (normal and tangent component) at cell walls.



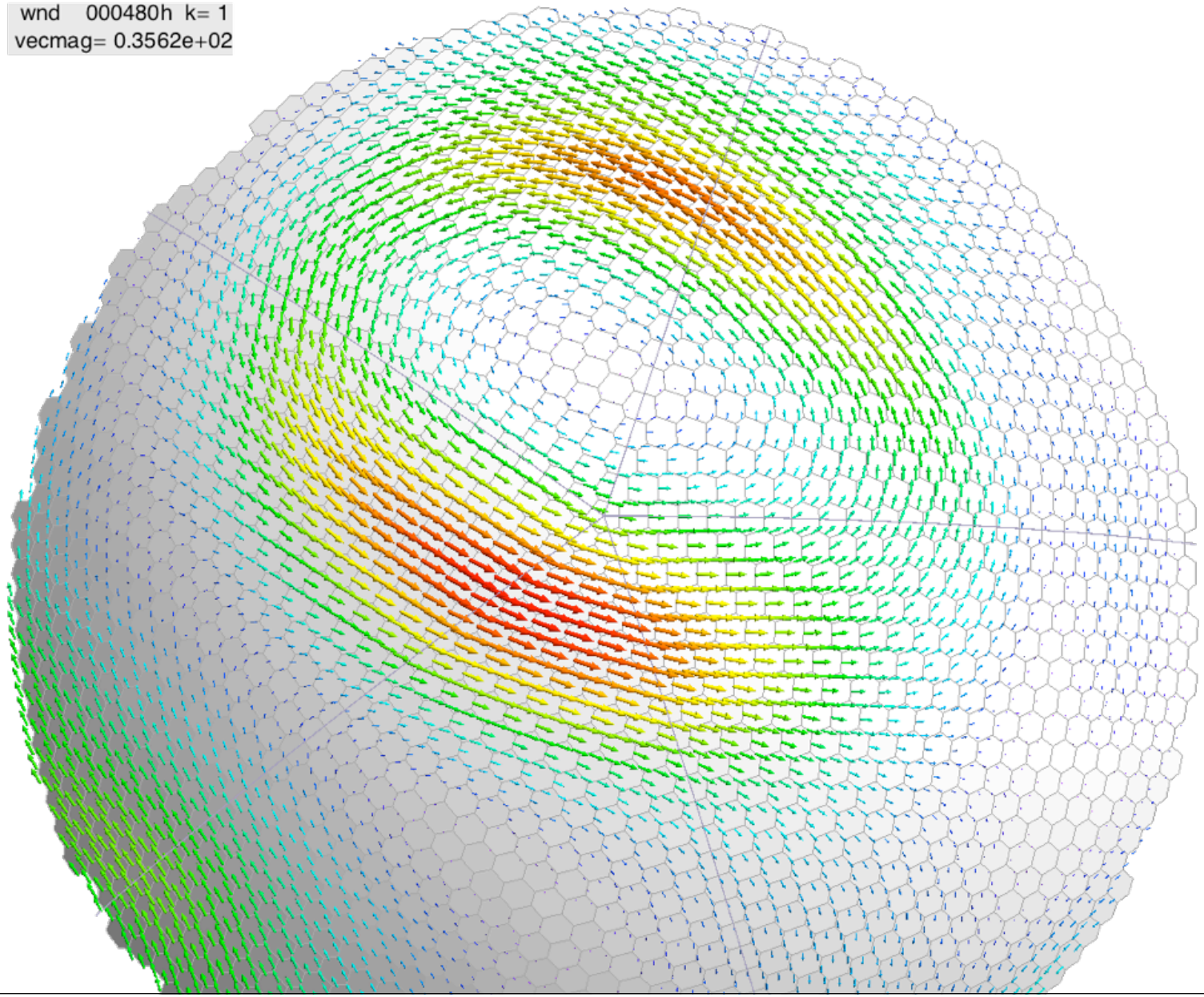
Simple interpolation. Wind vectors (normal and tangent component) at cell walls.



Quadratic interpolation. Wind vectors (normal and tangent component) at cell walls.



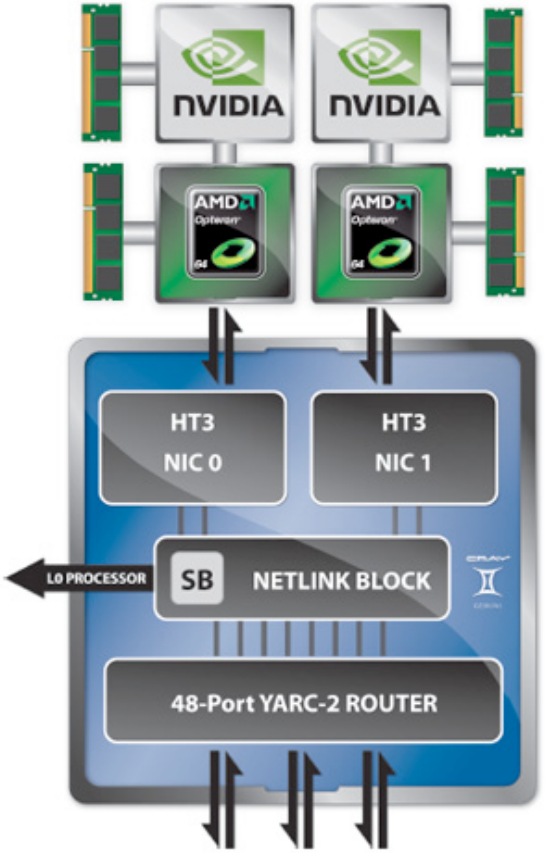
Quadratic interpolation. Wind vectors (normal and tangent component) at cell walls.



NSF Blue Waters supercomputer at the NCSA

- The Blue Waters system is a Cray XE/XK **hybrid** machine composed of AMD 6276 "Interlagos" processors (nominal clock speed of at least 2.3 GHz) and NVIDIA GK110 "Kepler" accelerators all connected by the Cray Gemini torus interconnect.
  1) 237 Cray XE6 cabinets. 362,240 cores
  2) 32 Cray XK7 cabinets. 24,576 cores. 3072 GPUs.

- The Kepler GK110 implementation includes 15 Streaming Multiprocessor (SMX) units and six 64-bit memory controllers. Each of the SMX units feature 192 single-precision CUDA cores.



Counting the cells.

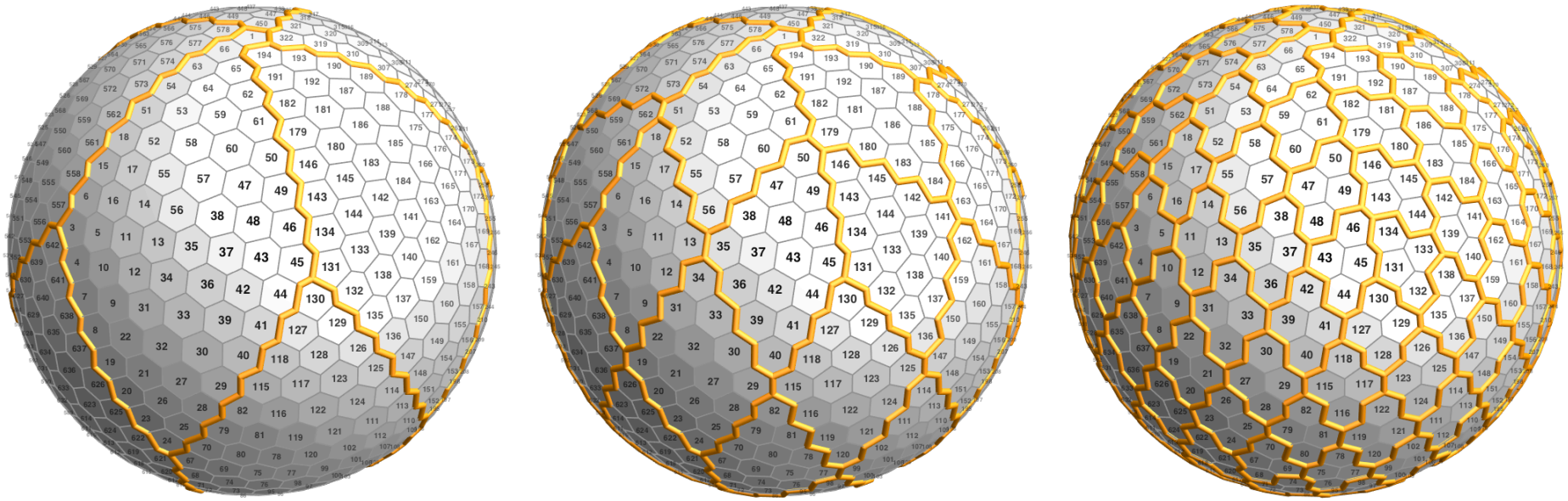
- Our target resolutions are:

resolution (r)	number of cells	global grid point spacing (km)
9	2,621,442	14.99
10	10,485,762	7.495
11	41,943,042	3.747
12	167,772,162	1.874

- The vertical resolution depends on the horizontal resolution. The vertical resolution is typically 32 to 256 layers.

Icosahedral grid. Parallel domain decomposition. Distribution to MPI tasks.

- Pieces of the grid are assigned to MPI tasks. Parallel domain decomposition.
- MPI non-blocking sends/receives are used to update ghost regions (halo regions) with data from neighboring processes.

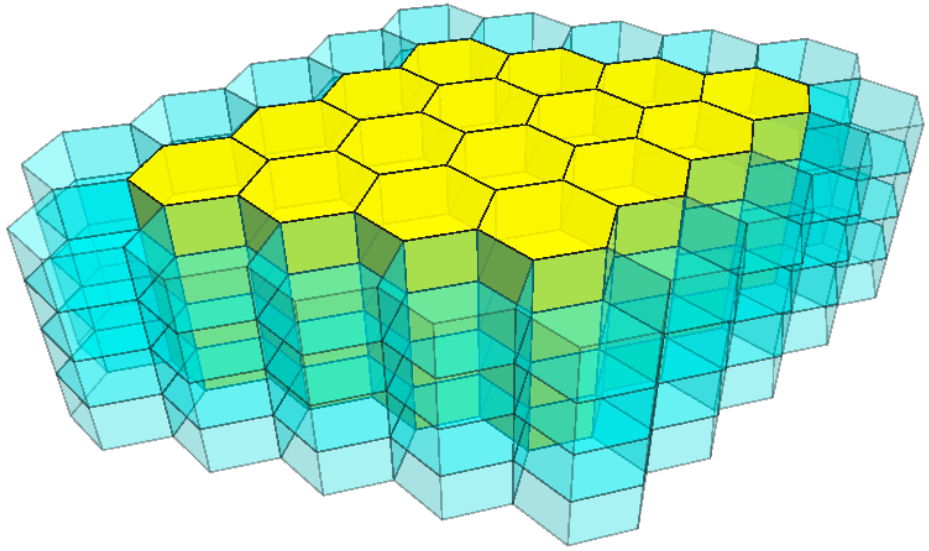


Define parallel efficiency

- Each grid block requires information from neighboring subdomains to fill ghost cells.
- We can define **parallel efficiency** to be:

$$\text{parallel efficiency} \approx \frac{\text{number of local cells}}{\text{number of ghost cells}}$$

- **Larger parallel efficiency is better**. More useful work is done per ghost cells.



Yellow cells *belong to the local process*

Blue cells *are ghost cells filled from neighboring process*

Parallel domain decomposition and parallel efficiency

- We would like each MPI task to have a 32×32 cell block or a 64×64 cell block:
  - Smaller block. The parallel efficiency is bad.
  - Bigger block. Too much work per task.
- For a given resolution increasing the number of tasks reduces parallel efficiency.

block size (parallel efficiency)		number of MPI tasks			
		640	**2560**	**10240**	**40960**
resolution (grid spacing)	**9** (14.99 km)	64×64 (15.7)	32×32 (7.76)	16×16 (3.76)	
	10 (7.495 km)	128×128 (31.7)	64×64 (15.7)	32×32 (7.76)	16×16 (3.76)
	11 (3.747 km)	256×256 (63.7)	128×128 (31.7)	64×64 (15.7)	32×32 (7.76)
	12 (1.874 km)		256×256 (63.7)	128×128 (31.7)	64×64 (15.7)

2D multigrid

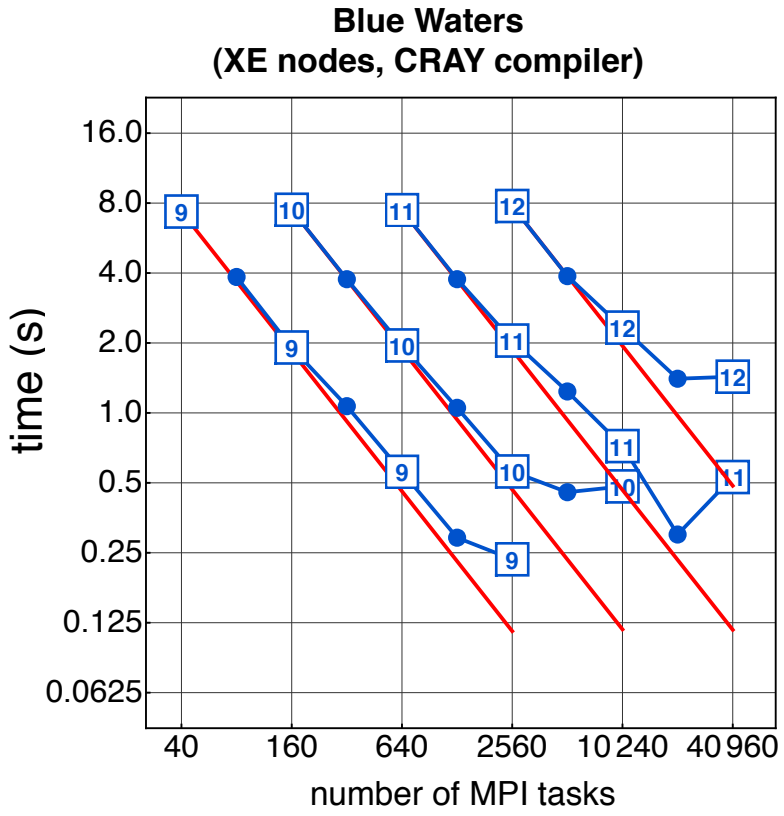
- The mathematical formulation of our prognostic equations requires solving Poisson's equation every time step in each model layer.
- The recursive structure of the grid facilitates the use of multigrid methods.
- This is most communication intensive portion of the model and challenging to parallelize. The lessons learned can be apply to other parts of the model.
- There are two main parts to the multigrid algorithm:

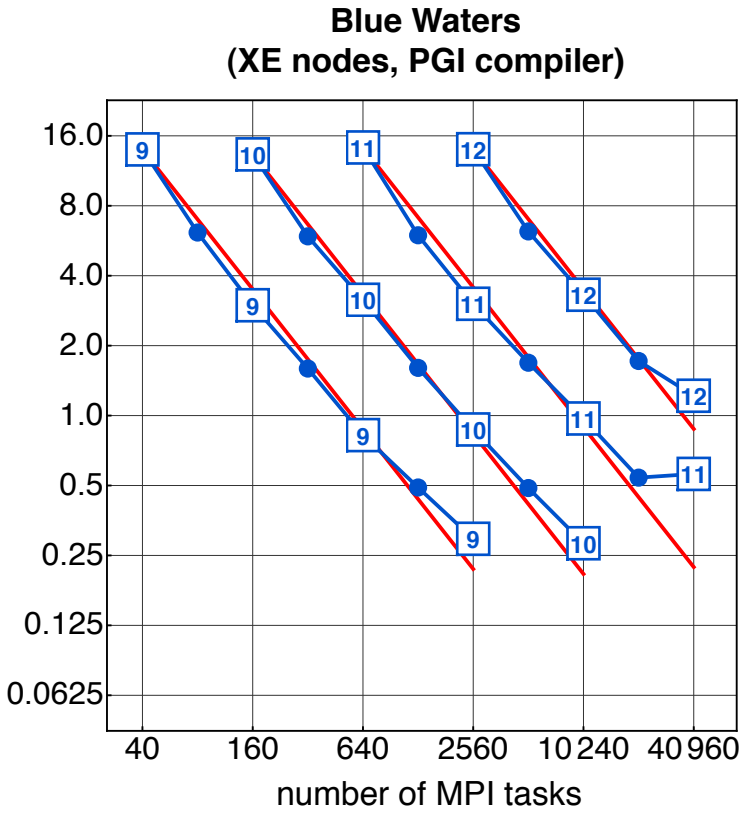
  (1) **Relaxation sweep**. Requires global communication. Similar to a standard Jacobi iteration. Most expensive.

  (2) **Information transfer** between grid resolutions. Less expensive.

Parallel scaling with MPI on Blue Waters. XE nodes.

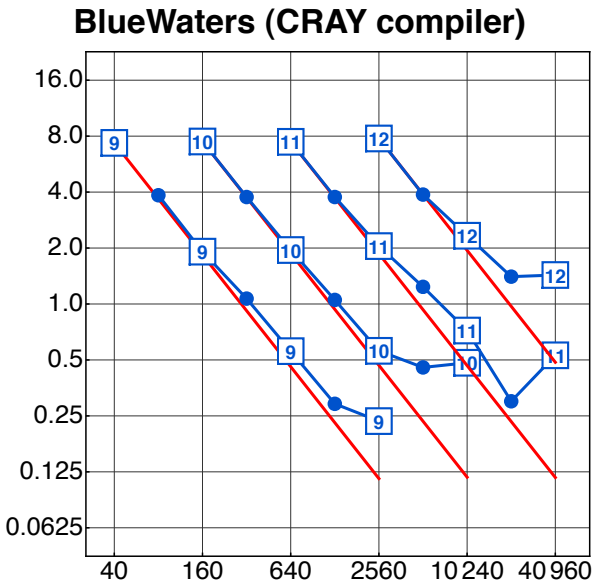
- Plot show the time to do 10 multigrid v-cycles
- X-axis is number of MPI tasks. Y-axis is time. Both are log scale.
- Each blue line indicates a particular grid resolution. Grids 09, 10, 11 and 12.
- The red line is the idealized speed-up.
- For each resolution the red line and the blue line should be coincident.

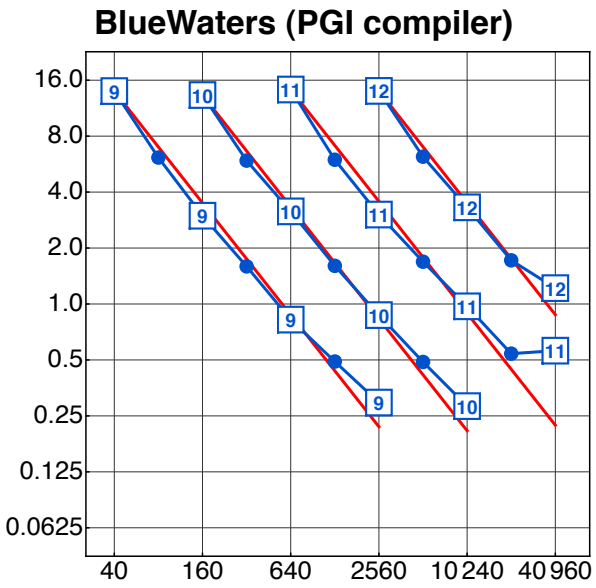


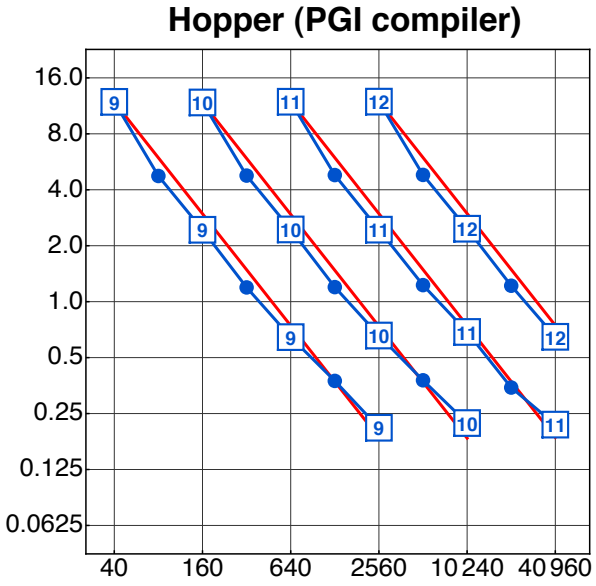


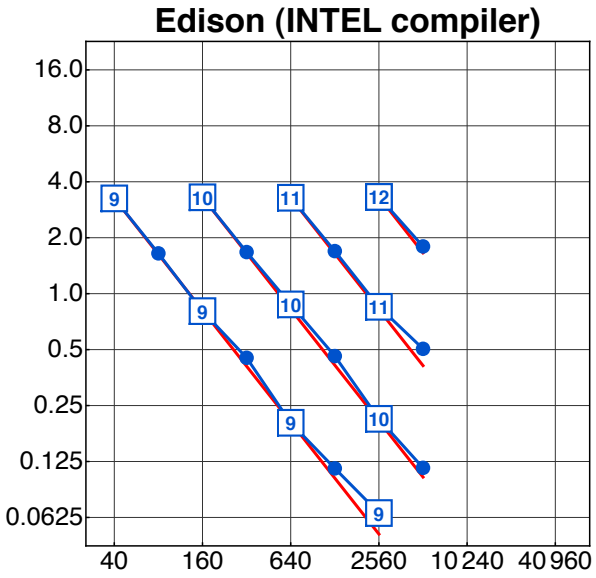
Parallel scaling with MPI. Comparisons.

- All the same code with no heroic optimization.
- We can see:
  - CRAY 2X faster than PGI on BW.
  - PGI scales better than CRAY on BW.
  - BW (PGI) and Hopper (PGI) have similar time
  - Hopper scales well.
  - Edison scales well (but with a relatively low number of cores).
  - Edison is pretty fast.









Multigrid on the accelerators.

- We are very interested in modifying the code to use the **accelerators**.
- We focus on the **relaxation sweep** portion of the multigrid algorithm. Experiments show this is the most expensive part of the code.
- The lessons learned can be apply to other parts of the model since the form of the code mimics other finite-difference operators in the model.
- Schematically the pure MPI code looks like this:

```
SUBROUTINE mltgrd2D_rlx (lvl,itermax,im0,jm0,km0,nsdm0,area,wght,beta,alph)

DO iter = 1,itermax ! number of sweeps
    MPI communication
    Relaxation Sweep
ENDDO ! iter

END SUBROUTINE mltgrd2D_rlx
```

Multigrid on the accelerators. The ideal best case with no MPI communication.

- Initially we can suppose no MPI communication was necessary. (Note that this gives the wrong answer.) Add a few **OpenACC directives**.
- What speed-up can we expect running code on host vs. accelerator?

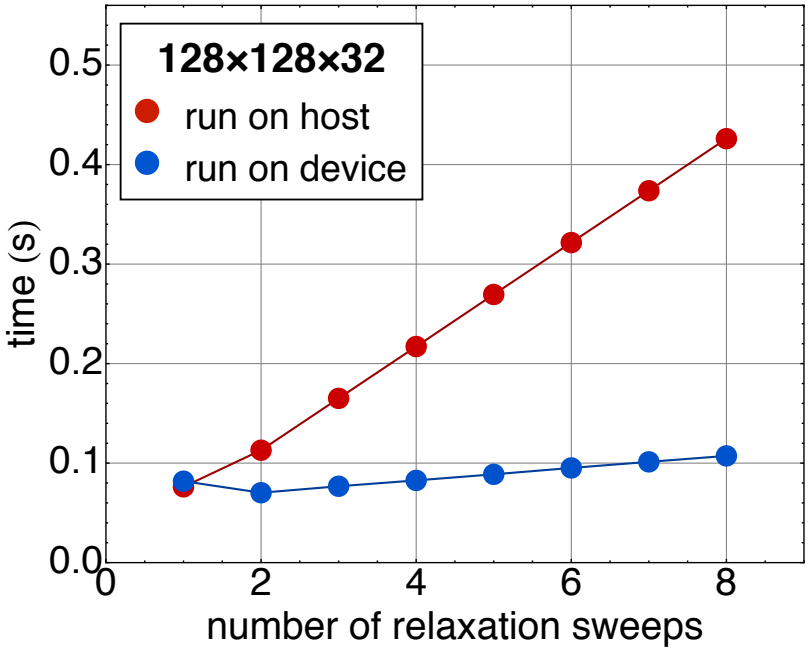
```
   SUBROUTINE mltgrd2D_rlx (lvl,itermax,im0,jm0,km0,nsdm0,area,wght,beta,alph)
!$acc data copyin (om1,om2,area,wght,beta) create (tmpry,work) copy (alph)
   DO iter = 1,itermax ! number of sweeps
!$acc kernels
```

Relaxation Sweep

```
!$acc end kernels
   ENDDO ! iter
!$acc end data
   END SUBROUTINE mltgrd2D_rlx
```

Multigrid on the accelerators. The ideal case with no MPI communication.

- Loading (unloading) the appropriate modules on the xk nodes, we can **toggle** to run on host or accelerator.
- We can see the latency associated with transfer of data from the host to the accelerator through the PCI express.
- But, when data is on the accelerator, it is very fast. Blue line very flat.



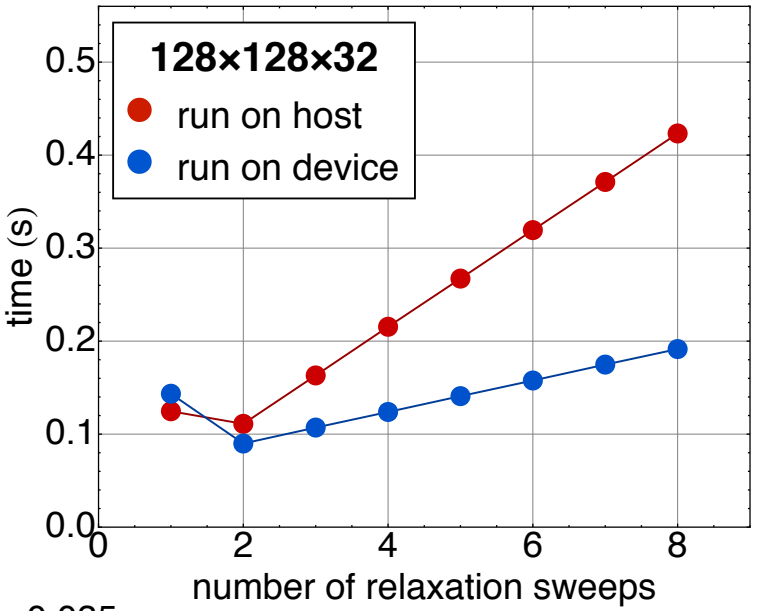
Multigrid on the accelerators. With MPI communication.

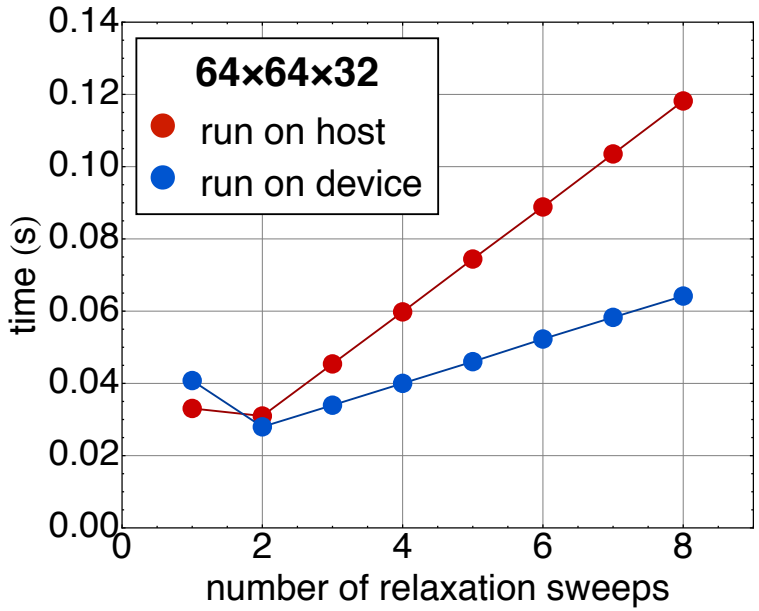
- Now we include the MPI and use the **`!$acc update`** directive:

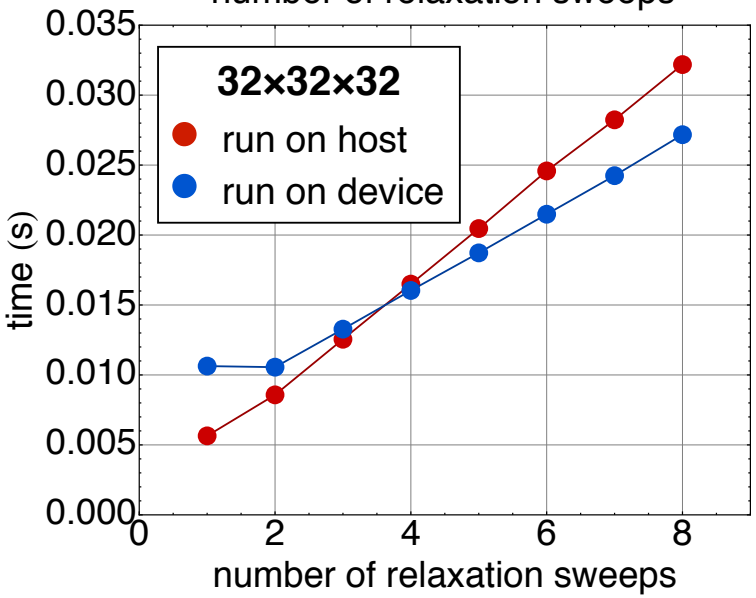
```
      SUBROUTINE mltgrd2D_rlx (lvl,itermax,im0,jm0,km0,nsdm0,area,wght,beta,alph)

!$acc data copyin (om1,om2,area,wght,beta) create (tmpry,work) copy (alph)

   DO iter = 1,itermax ! number of sweeps

      MPI communication

!$acc update device (alph(1:im0-1, 1     ,:,:))
!$acc update device (alph(  im0  ,1:jm0-1,:,:))
!$acc update device (alph(2:im0  ,  jm0  ,:,:))
!$acc update device (alph(   1   ,2:jm0  ,:,:))

!$acc kernels

      Relaxation Sweep

!$acc end kernels

!$acc update host (alph(2:im0-2, 2     ,:,:))
!$acc update host (alph(  im0-1,2:jm0-2,:,:))
!$acc update host (alph(3:im0-1,  jm0-1,:,:))
!$acc update host (alph(   2   ,3:jm0-1,:,:))

   ENDDO ! iter

!$acc end data

   END SUBROUTINE mltgrd2D_rlx
```

Multigrid on the accelerators. With MPI communication.

- The speed-up depends on the block size. Less speed-up on smaller blocks.
- This will become an issue on coarser grid resolution within the multigrid v-cycle.
- Coarser grids may run exclusively on the host





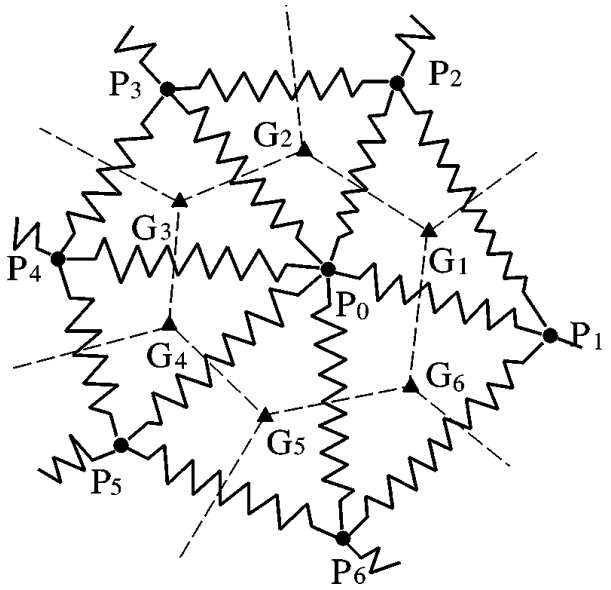


Alternative ways to generate an icosahedral grid.

SPRING GRID

Tomita, H., M. Tsugawa, M. Satoh, and K. Goto, 2001: Shallow water model on a modified icosahedral geodesic grid by using spring dynamics. J. Comput. Phys., 174, 579–613.

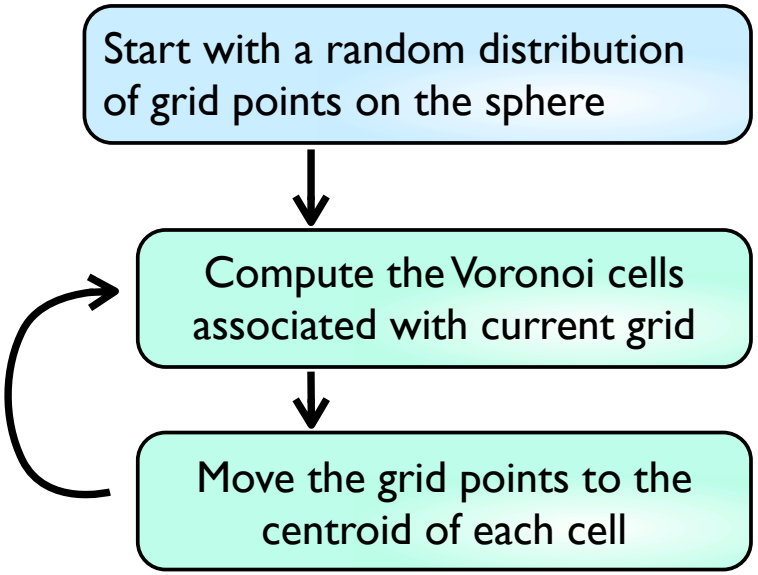
- The basic idea is to imagine that the grid points are connected with springs and move with damped motion to a position of equilibria.



CENTROIDAL VORONOI TESSELLATION (CVT)

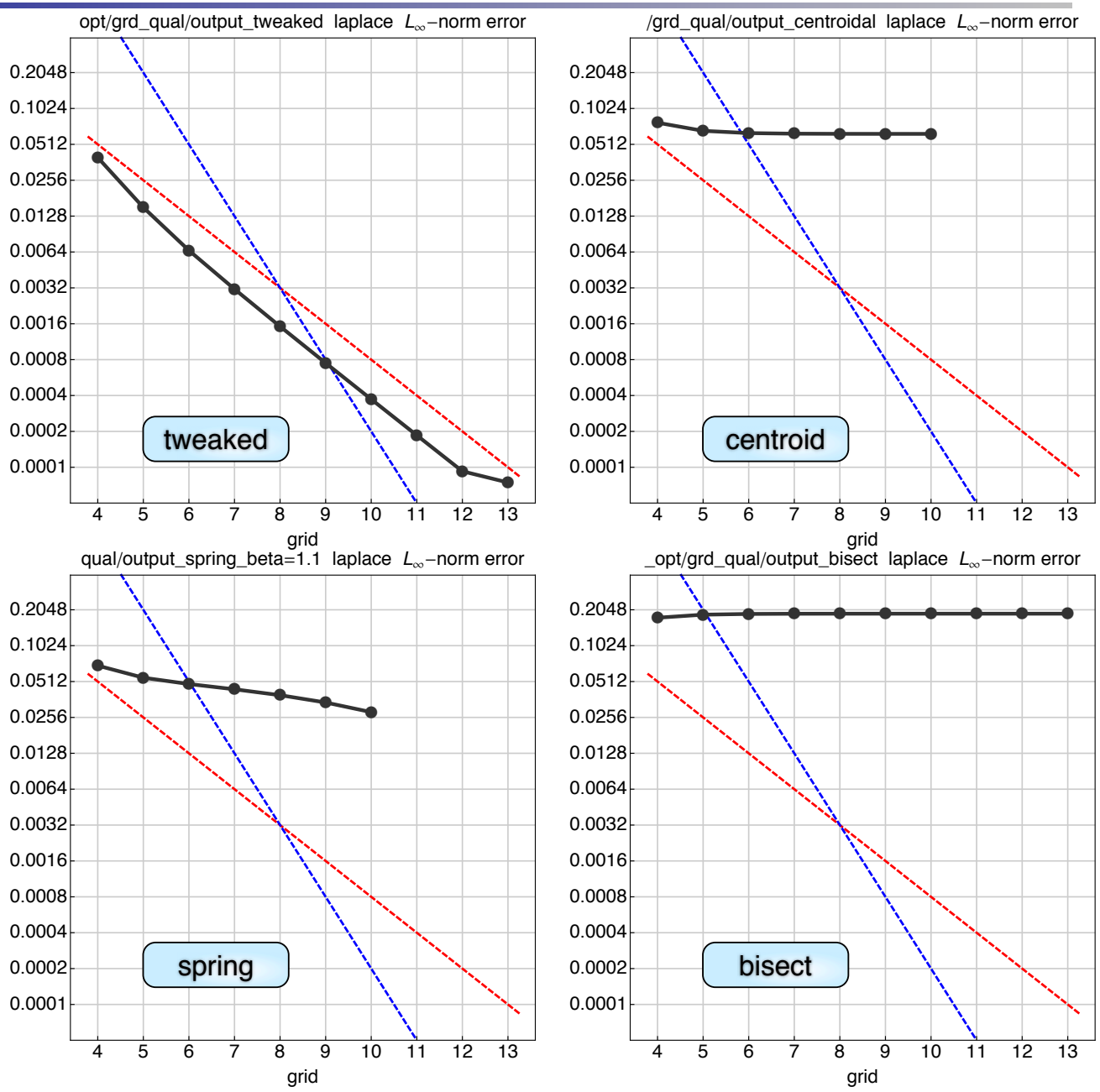
Du, Q., V. Faber, and M. Gunzburger, 1999: Centroidal Voronoi tessellations: Application and algorithms. SIAM Rev., 41, 637–676.

- The basic idea is to position the grid points so that they are the centroid (center of mass) of each cell.
- Llody's algorithm:



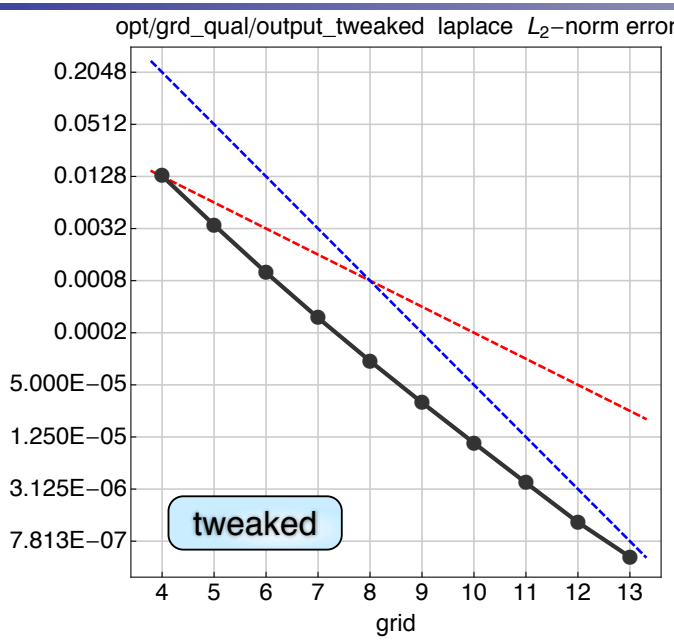
Maximum errors of finite-difference Laplacian

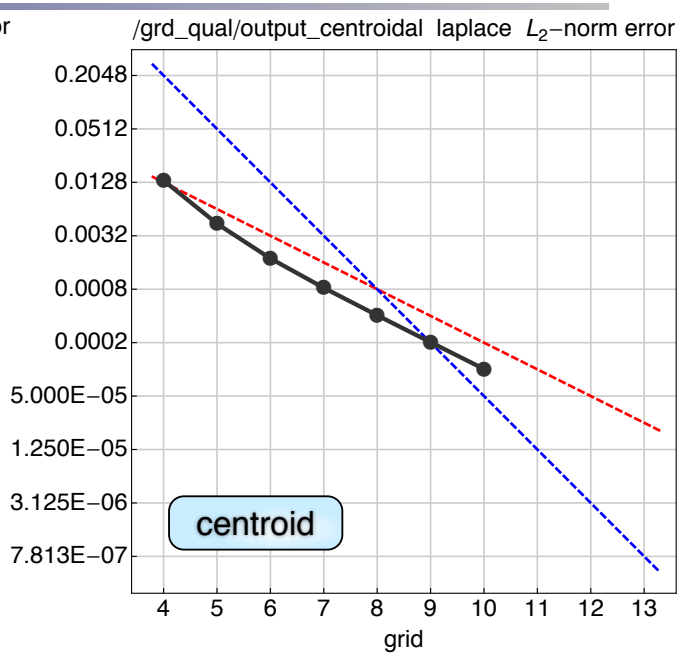
- With a very smooth analytic test function show the convergence properties of the various grids
- The **inf-norm error** measures the worst case of the grid
- The red line shows idealized 1st-order convergence
- The blue line shows idealized 2st-order convergence

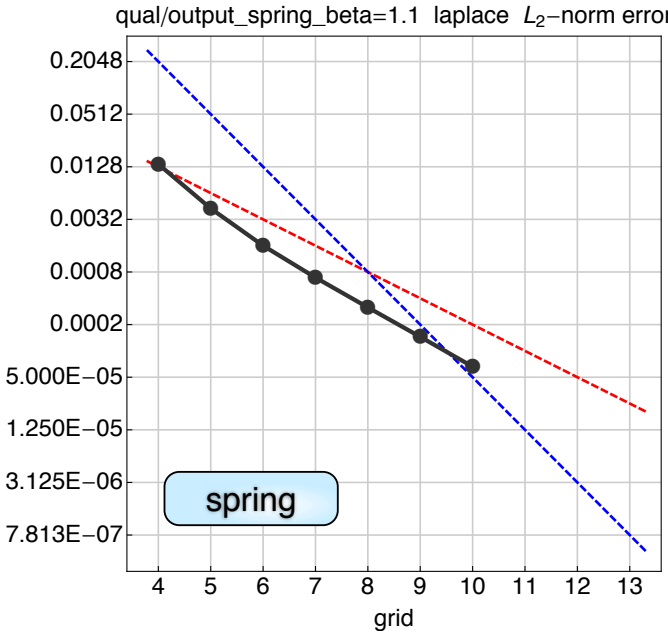


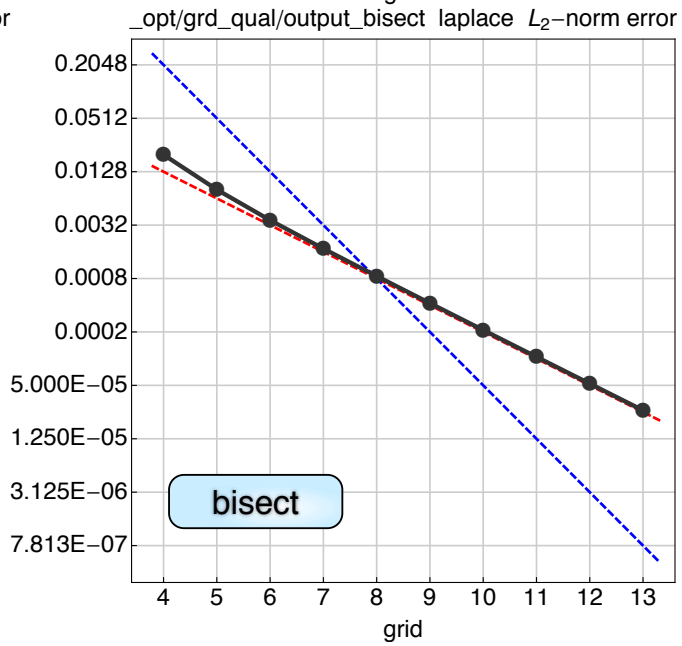
RMS errors of finite-difference Laplacian

- With a very smooth analytic test function show the convergence properties of the various grids
- The **RMS error** measures the overall goodness of the grid
- The red line shows idealized 1st-order convergence
- The blue line shows idealized 2st-order convergence









The Conjugate Gradient Method. A possible strategy to improvement the 3D elliptic solver

- The non-hydrostatic model requires the solution to a 3D poisson equation.
- We currently use standard multigrid methods in the horizontal direction coupled with a line relaxation in the vertical direction.
- We explored the use of algebraic multigrid methods with little improvement.
- The conjugate gradient method has been shown to work well in highly anisotropic problems.

The cells range from very flat to not so flat

Let δx be the average distance between cell centers and δz be the layer thickness.

Define

$$\text{aspect ratio} \equiv \frac{\delta x}{\delta z}$$

The aspect ratio for various configurations

	grd05 δx=239.8(km)	grd06 δx=119.9(km)	grd07 δx=59.96(km)	grd08 δx=29.98(km)	grd09 δx=14.99(km)	grd10 δx=7.495(km)	grd11 δx=3.747(km)	grd12 δx=1.874(km)
km= 32, δz=1000(m)	239.812	119.915	59.9584	29.9793	14.9897	7.49485	3.74742	1.87371
km= 64, δz= 500(m)	479.623	239.829	119.917	59.9587	29.9794	14.9897	7.49485	3.74742
km=128, δz= 250(m)	959.247	479.659	239.834	119.917	59.9588	29.9794	14.9897	7.49485
km=256, δz= 125(m)	1918.49	959.317	479.667	239.835	119.918	59.9588	29.9794	14.9897

There is a considerable range of aspect ratios depending on the simulation:

- ~100 to 200 for large scale simulations. (Jablonowski, HS)
- ~1 to 10 for small scale simulations. (bubble tests)

The Poisson equation with constant coefficients

When the area of the cells is large, the coefficients linking grid points in the horizontal direction become very small.

Continuous form:

$$\nabla_H^2 \alpha + \frac{\partial^2 \alpha}{\partial z^2} = \beta$$

Discrete form:

k = 1

$$\frac{1}{A_i}\sum_{i'}\frac{\alpha_{i+i',k}-\alpha_{i,k}}{L_{i;i+i'}}l_{i;i+i'} + \frac{1}{(\delta z)_k}\frac{\alpha_{i,k+1}-\alpha_{i,k}}{(\delta z)_{k+1/2}} = \beta_{i,k}$$

$$\frac{1}{A_i}\sum_{i'}\frac{l_{i;i+i'}}{L_{i;i+i'}}\alpha_{i+i',k} + \frac{1}{(\delta z)_k(\delta z)_{k+1/2}}\alpha_{i,k+1} - \left(\frac{1}{A_i}\sum_{i'}\frac{l_{i;i+i'}}{L_{i;i+i'}} + \frac{1}{(\delta z)_k(\delta z)_{k+1/2}}\right)\alpha_{i,k} = \beta_{i,k}$$

k = 2,…, km - 1

$$\frac{1}{A_i}\sum_{i'}\frac{\alpha_{i+i',k}-\alpha_{i,k}}{L_{i;i+i'}}l_{i;i+i'} + \frac{1}{(\delta z)_k}\left[\frac{\alpha_{i,k+1}-\alpha_{i,k}}{(\delta z)_{k+1/2}} - \frac{\alpha_{i,k}-\alpha_{i,k-1}}{(\delta z)_{k-1/2}}\right] = \beta_{i,k}$$

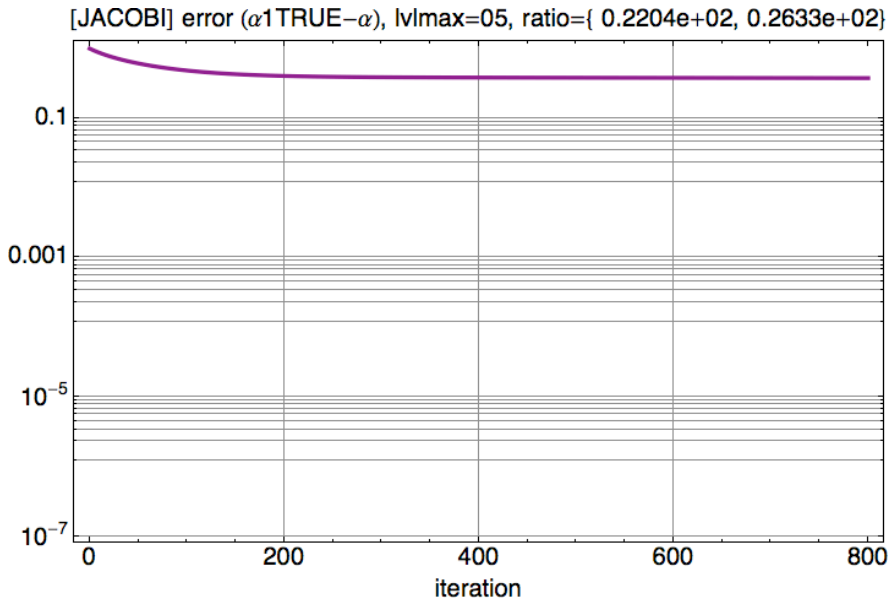
$$\frac{1}{A_i}\sum_{i'}\frac{l_{i;i+i'}}{L_{i;i+i'}}\alpha_{i+i',k} + \frac{1}{(\delta z)_k(\delta z)_{k+1/2}}\alpha_{i,k+1} - \left(\frac{1}{A_i}\sum_{i'}\frac{l_{i;i+i'}}{L_{i;i+i'}} + \frac{1}{(\delta z)_k(\delta z)_{k+1/2}} + \frac{1}{(\delta z)_k(\delta z)_{k-1/2}}\right)\alpha_{i,k} + \frac{1}{(\delta z)_k(\delta z)_{k-1/2}}\alpha_{i,k-1} = \beta_{i,k}$$

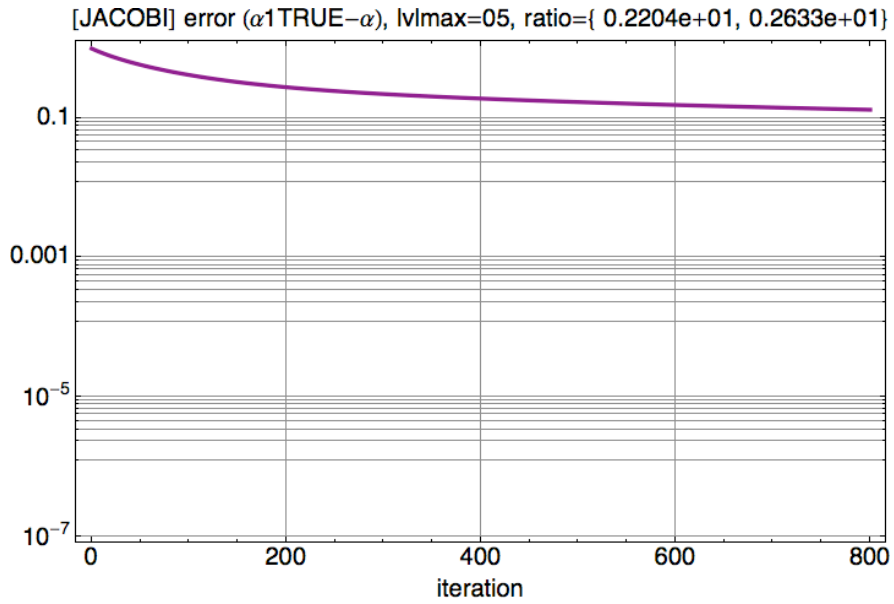
k = km

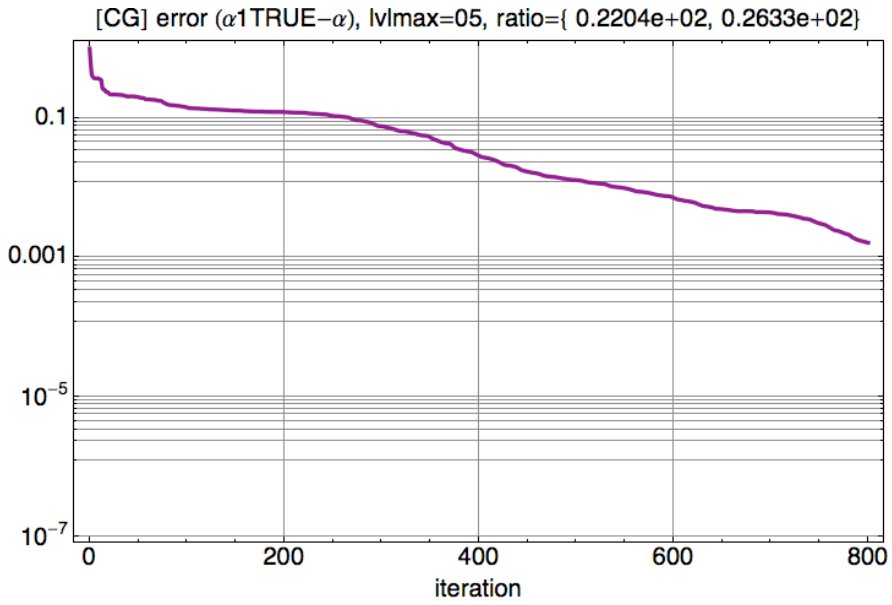
$$\frac{1}{A_i}\sum_{i'}\frac{\alpha_{i+i',k}-\alpha_{i,k}}{L_{i;i+i'}}l_{i;i+i'} - \frac{1}{(\delta z)_k}\frac{\alpha_{i,k}-\alpha_{i,k-1}}{(\delta z)_{k-1/2}} = \beta_{i,k}$$

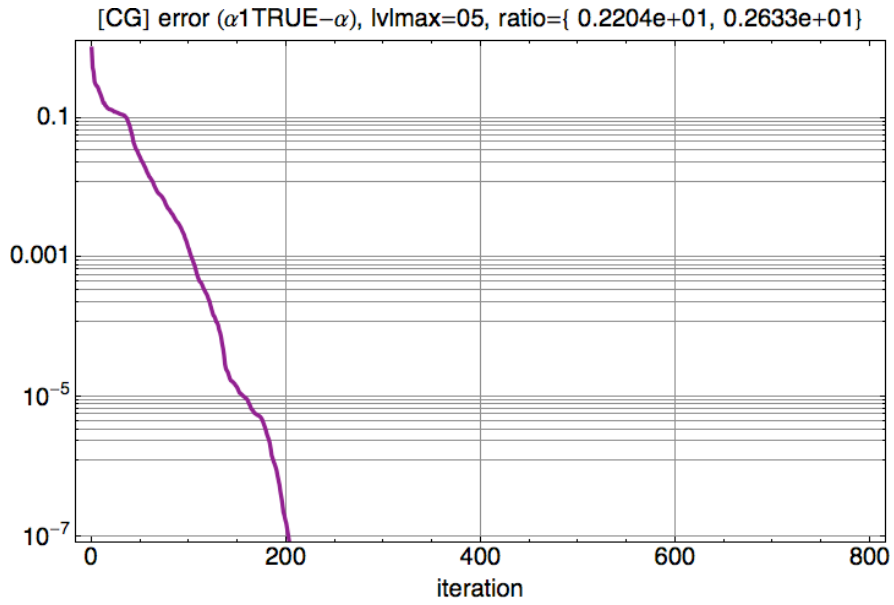
$$\frac{1}{A_i}\sum_{i'}\frac{l_{i;i+i'}}{L_{i;i+i'}}\alpha_{i+i',k} - \left(\frac{1}{A_i}\sum_{i'}\frac{l_{i;i+i'}}{L_{i;i+i'}} + \frac{1}{(\delta z)_k(\delta z)_{k-1/2}}\right)\alpha_{i,k} + \frac{1}{(\delta z)_k(\delta z)_{k-1/2}}\alpha_{i,k-1} = \beta_{i,k}$$

The Conjugate Gradient and Jacobi Method. Convergence as a function of aspect ratio.



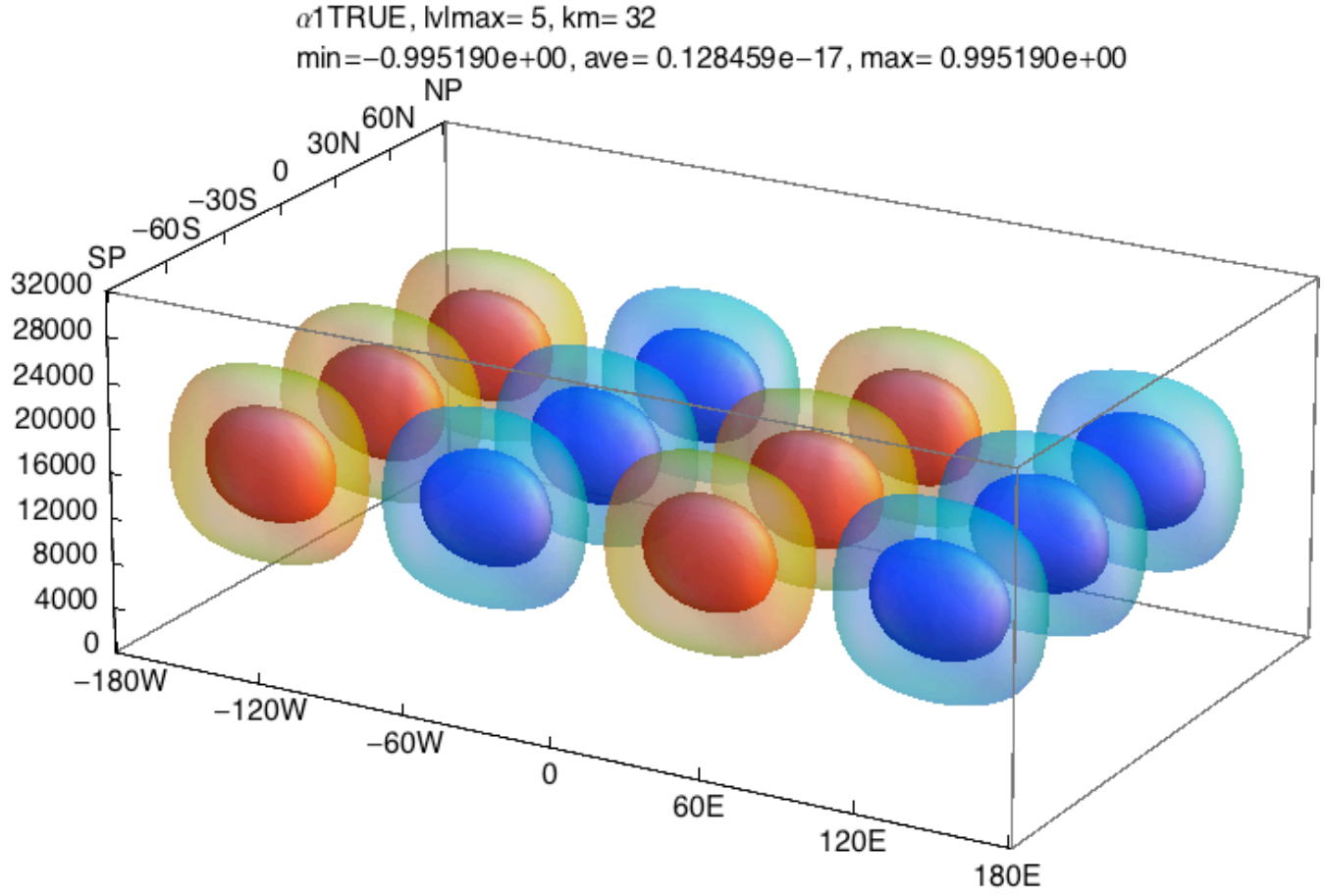






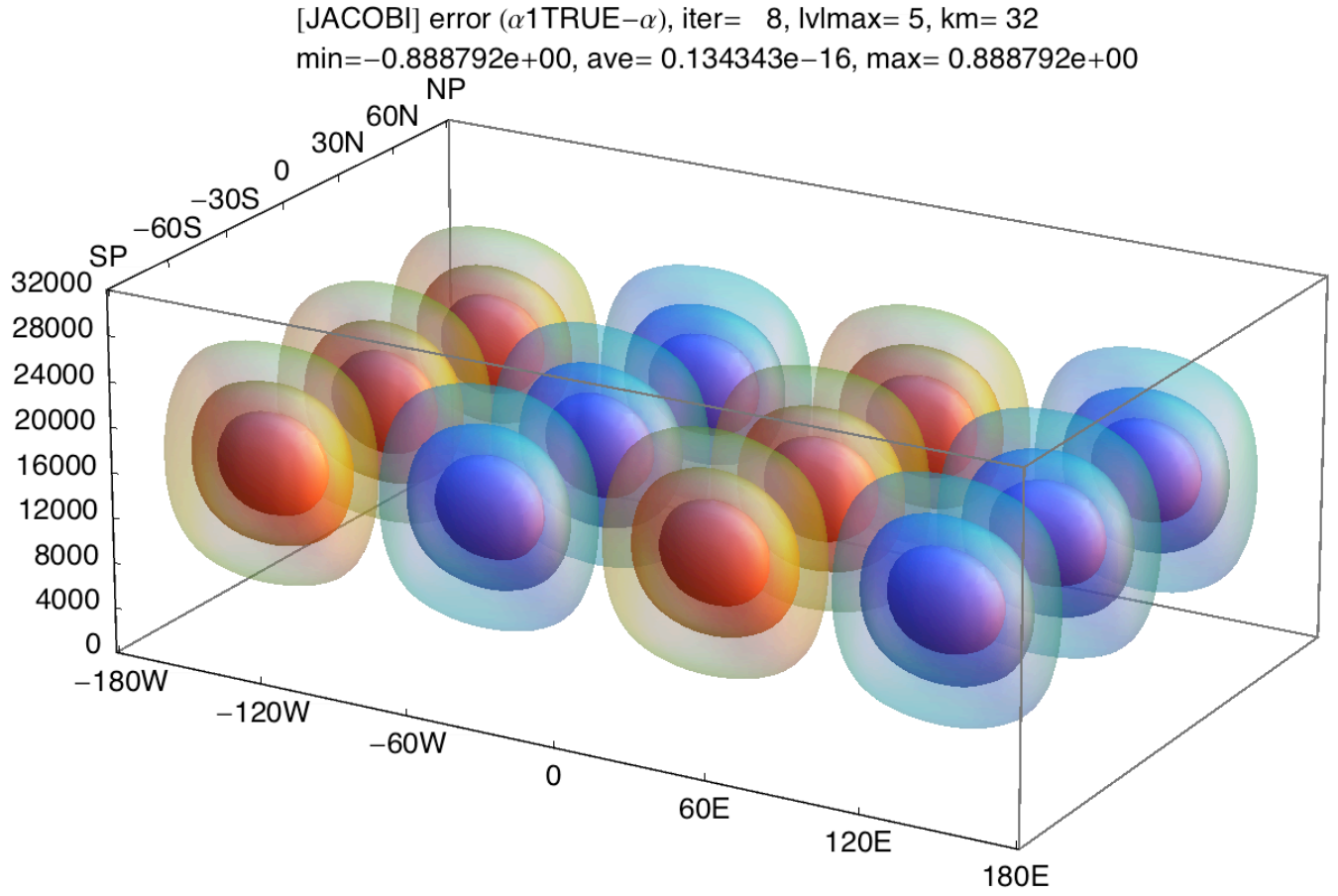
The Conjugate Gradient Method.

- We can gain some insight into the problem with the 3D multigrid by looking at the error in the solution.
- Consider a very smooth analytic test case: (Note the 4 by 3 blob structure)



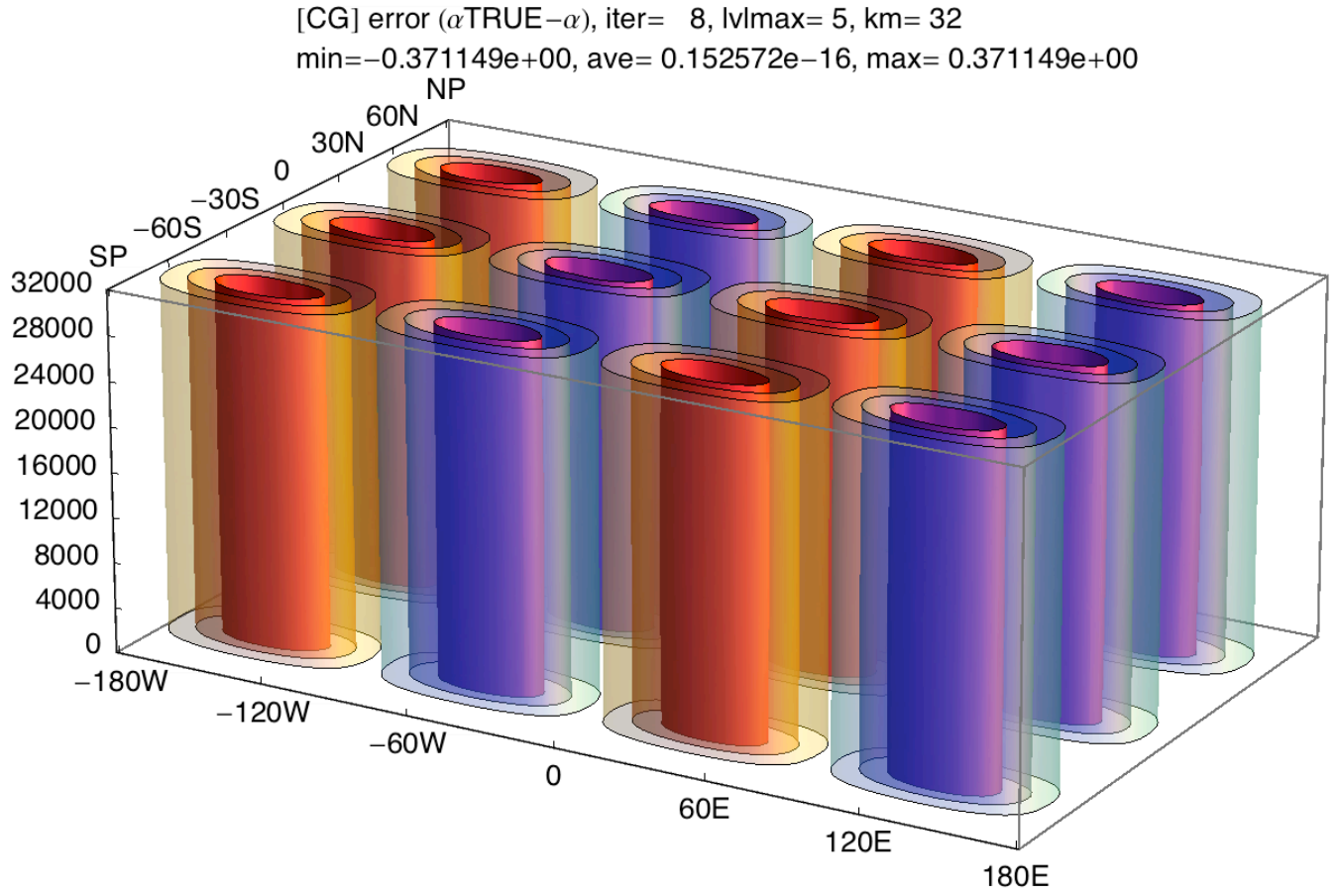
The Conjugate Gradient Method.

- The movie shows the error (TRUE-APPX) as a function of iteration for the **Jacobi iteration**.
- There is an additive constant for each column that the algorithm cannot remove.



The Conjugate Gradient Method.

- The movie shows the error (TRUE-APPX) as a function of iteration for the **CG iteration**.
- Not locked into column additive constant.



Summary

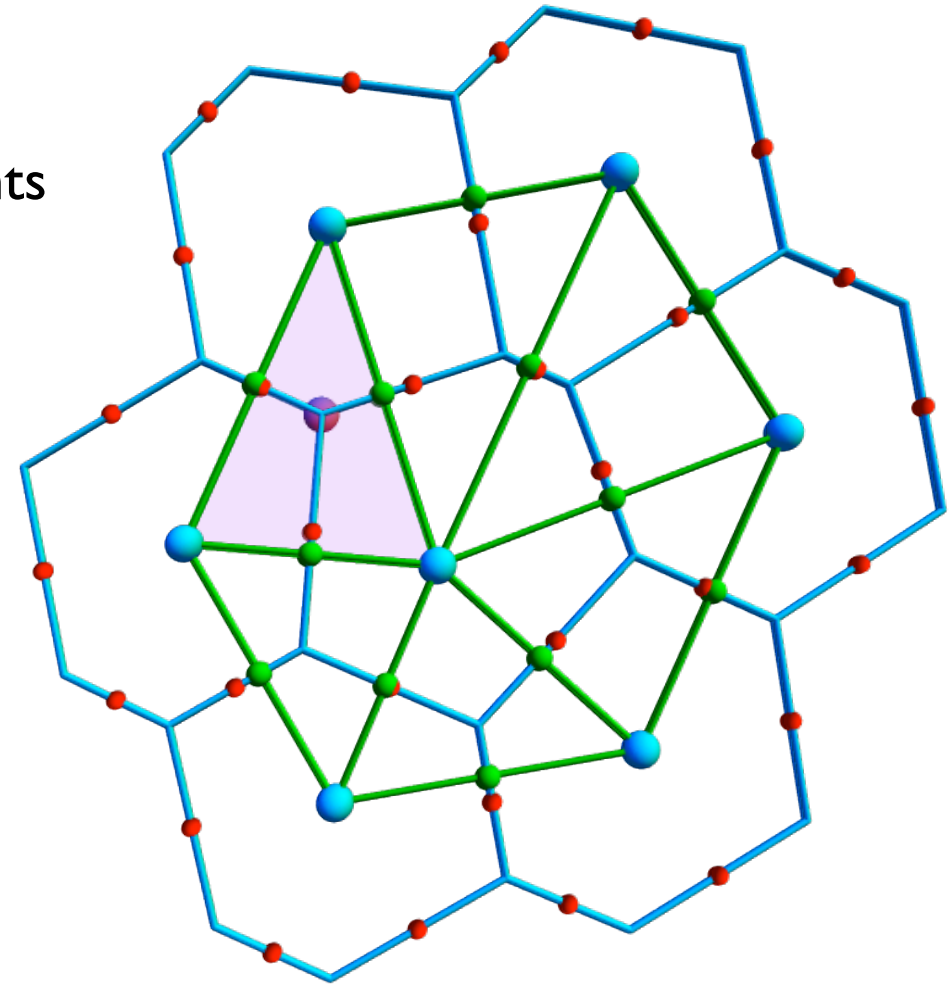
1. We have shown a new algorithm to generate grids for the cubed sphere.
2. The normal component of wind on the icosahedral grid shows some unusual structure which still needs to be explained.
3. Experiences on the NSF Blue Waters computer. Potential to improve the speed of the model.
4. Conjugate gradient method. Preconditioner for the multigrid. Could be used as a smoother within the multigrid.
5. Comparison of the Tweaked Grid to Spring Grid and Centroidal Voronoi Tessellations (CVT) Grid

Grid Optimization Algorithm

- The ***Voronoi Corner*** (purple dot) is defined as the point equidistant from surrounding grid points (blue dots).

- There is a flaw with the Voronoi grid -- a line connecting grid points does not bisect the cell wall.

- The algorithm positions all grid points so that red points are coincident with green points (or at least it does the best it can)



Grid Optimization Algorithm

- The *goodness* of particular configuration of points can be expressed as a cost function:

$$F = \sum_{n=1}^{all\ cells} \sum_{i=1}^{cell\ walls} \left(goodness\ of\ wall \right)_{n,i}^{2}$$

- Solved using quasi-Newton methods:

$$\underset{\mathbf{x}}{\text{minimize}}\, F\left(x_1, x_2, \ldots, x_m \right)$$

- Other grid properties can be optimized using different cost functions

Grid Optimization Algorithm

- Number of independent variables:
- The number of independent variables (and computer memory) is reduced by using symmetries intrinsic to the grid. Great circles can partition the sphere into 120 triangular subdomains.
- Parallelization of the algorithm
- Many weird problems pop up at higher resolutions

grid resolution	number of independent variables
(8) 655,362 (31.27km)	8,192
(9) 2,621,442 (15.64km)	32,768
(10) 10,485,762 (7.819km)	131,072
(11) 41,943,042 (3.909km)	524,288
(12) 167,772,162 (1.955km)	2,097,152