



# ULTRASCALE CLIMATE DATA VISUALIZATION WITH UV-CDAT

## TECH-X

SIMULATIONS EMPOWERING  
YOUR INNOVATIONS



Dean Williams, Charles Doutriaux, Jerry Potter, Thomas Maxwells, **Alex Pletzer**, and the UV-CDAT team



NYU·poly



 **ParaView**

**ESMF**

 **Kitware**



**SCI**  
www.sci.utah.edu



**SAIC**



Workshop on Frontiers in Computational Physics,  
NCAR, Boulder CO, 17 December 2012

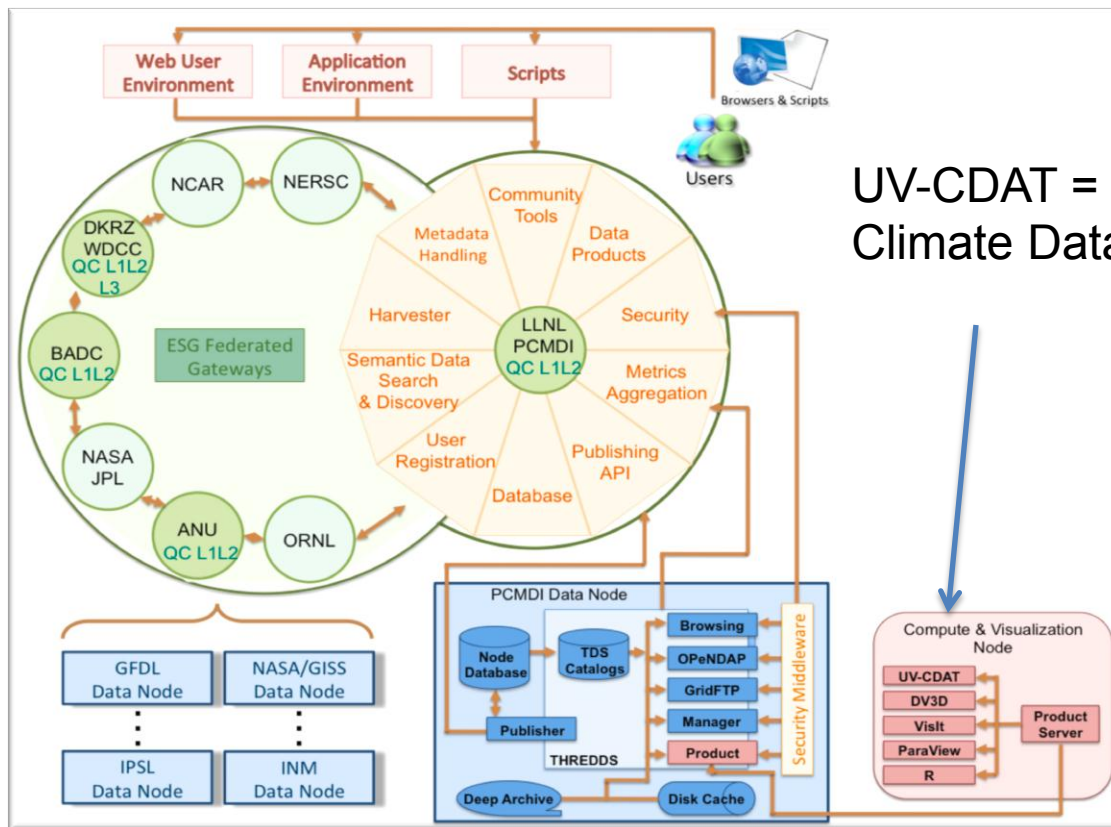


SIMULATIONS EMPOWERING YOUR INNOVATIONS



# UV-CDAT is an open-source environment for analyzing, visualizing, and diagnosing climate data

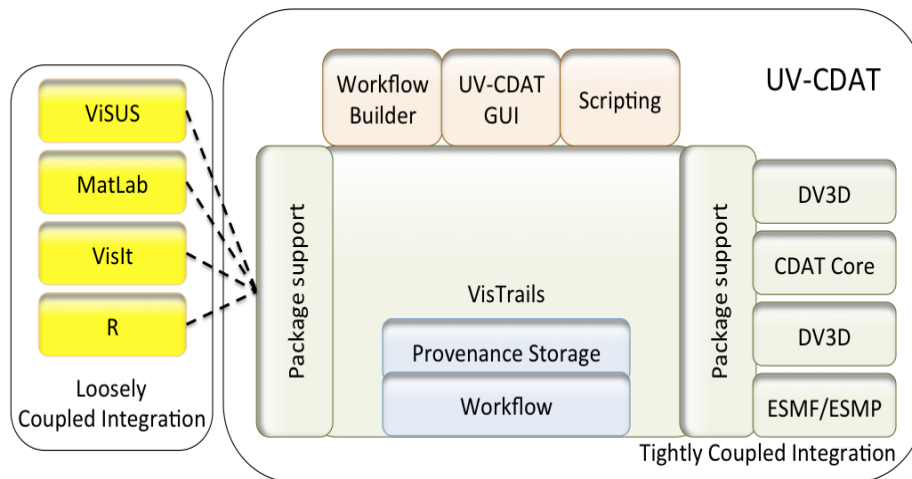
- Used by the Earth System Grid Federation to serve data
- **UV-CDAT is the successor of CDAT (Climate Data Analysis Tools)**



UV-CDAT = Ultrascale Visualization  
Climate Data Analysis Tools

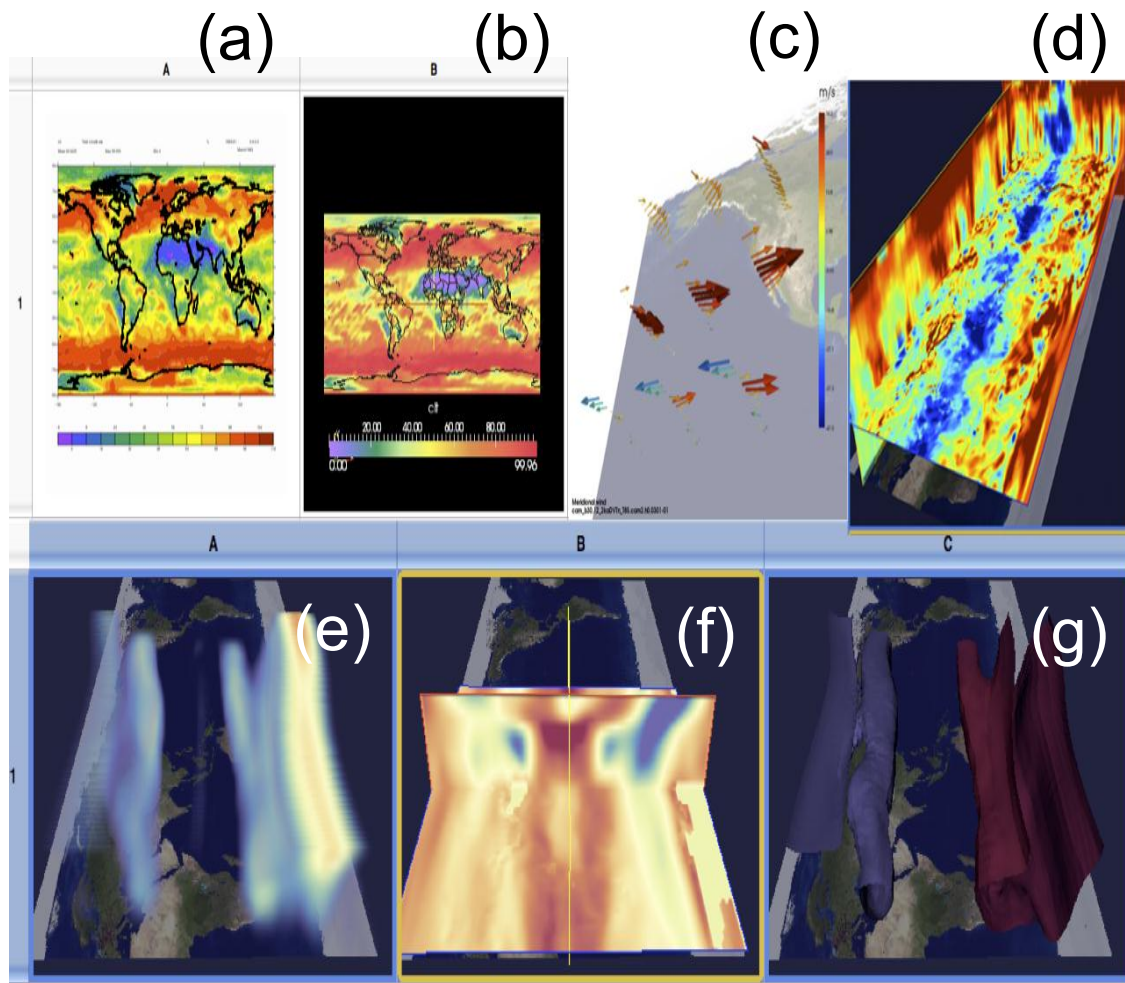
# UV-CDAT's architecture and build system are designed to scale up

- **40+ open-source packages** integrated within a single application
- Highly configurable cross-platform build system (CMake)

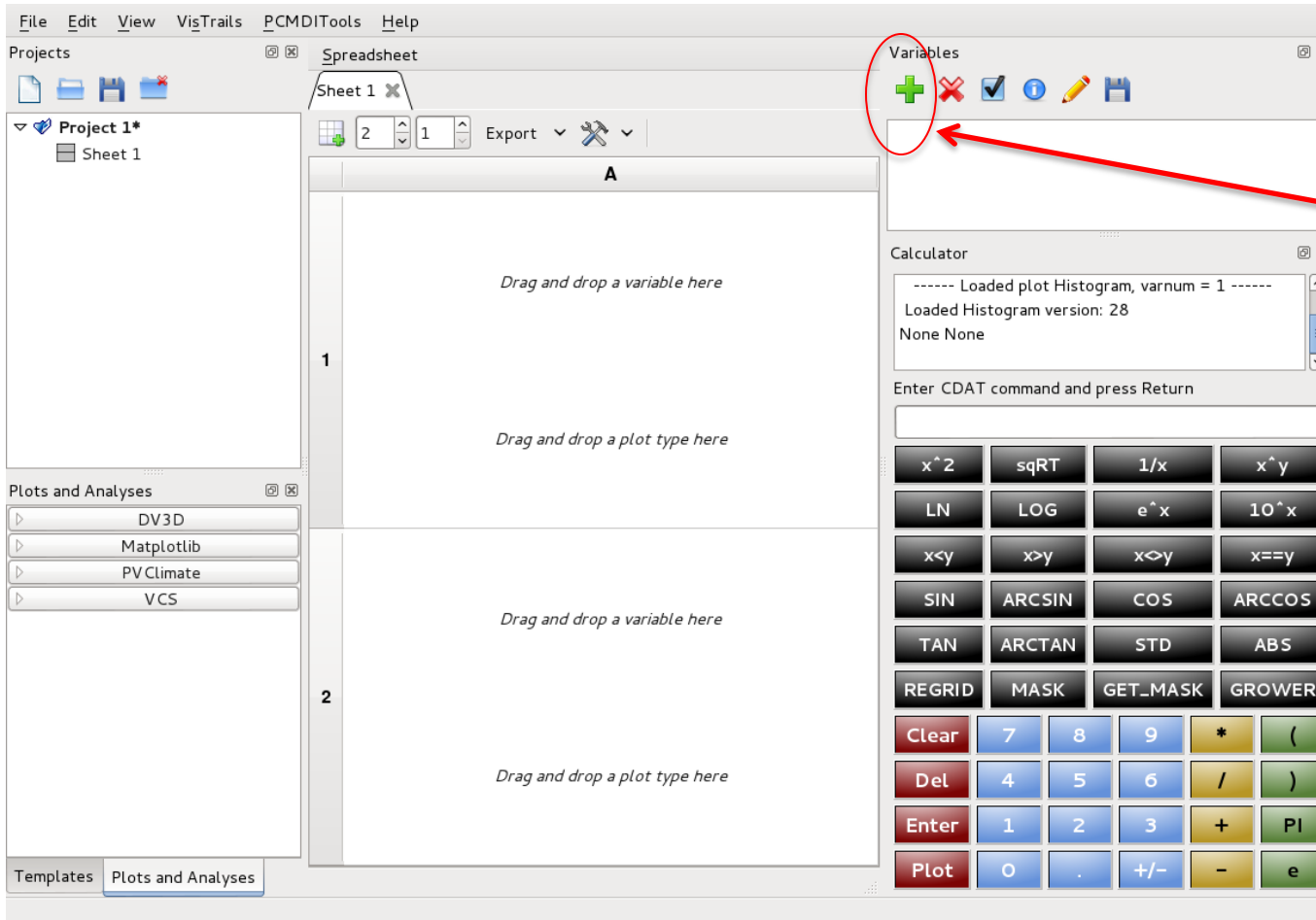


# Multiple visualization tools can be invoked from within UV-CDAT

- Accurate visualization is essential for data exploration
  - ◆ VCS (a)
  - ◆ Paraview
  - ◆ VisIt
  - ◆ DV3D:
    - c: vector
    - d: slice
    - e: volume
    - g: iso-surface

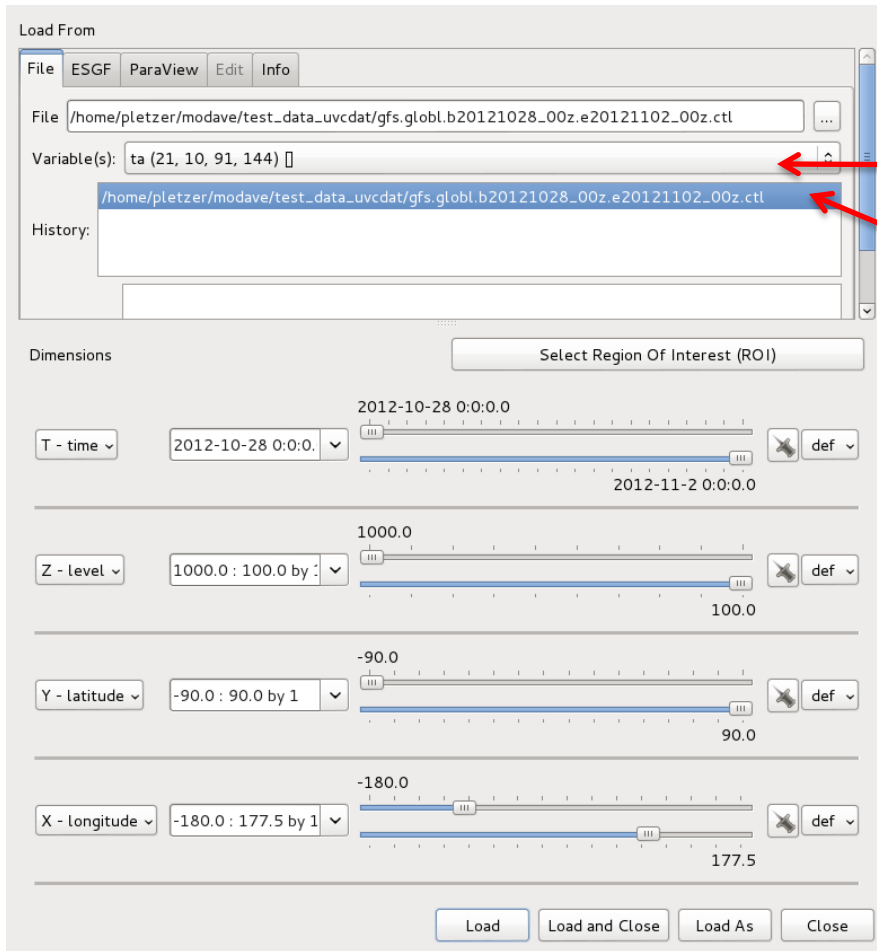


# UV-CDAT's start up graphical user interface



Open file  
and add  
variable

# Selecting the file and variable

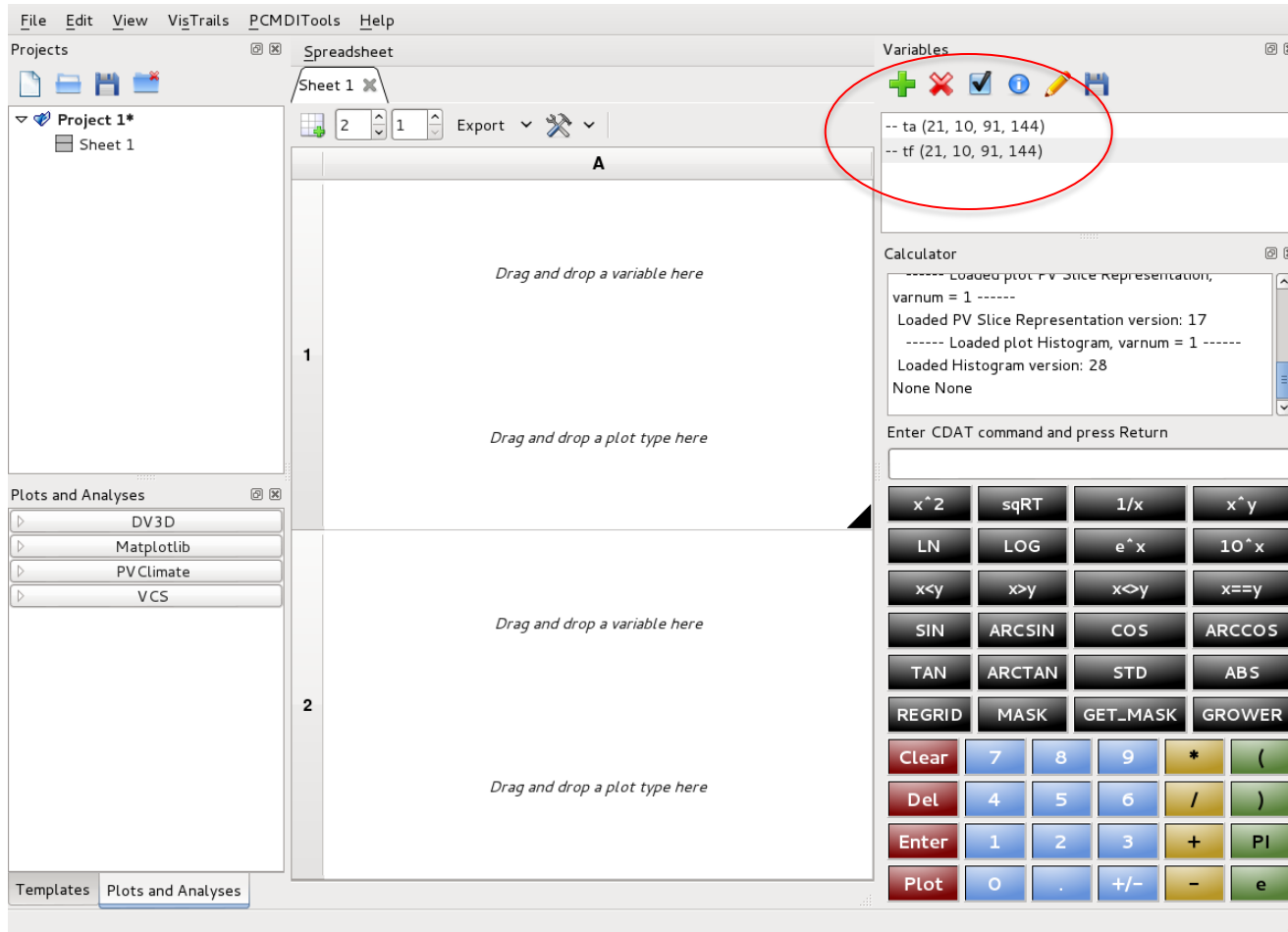


Variable name

File name

subslice the variable if desired

# After loading the variables ta and tf



The screenshot shows a software interface with several panels:

- Projects:** Project 1\* (Sheet 1)
- Spreadsheet:** Sheet 1, column A. It contains two rows with the text "Drag and drop a variable here" and "Drag and drop a plot type here".
- Variables:** A panel on the right containing a red circle around two entries:
 

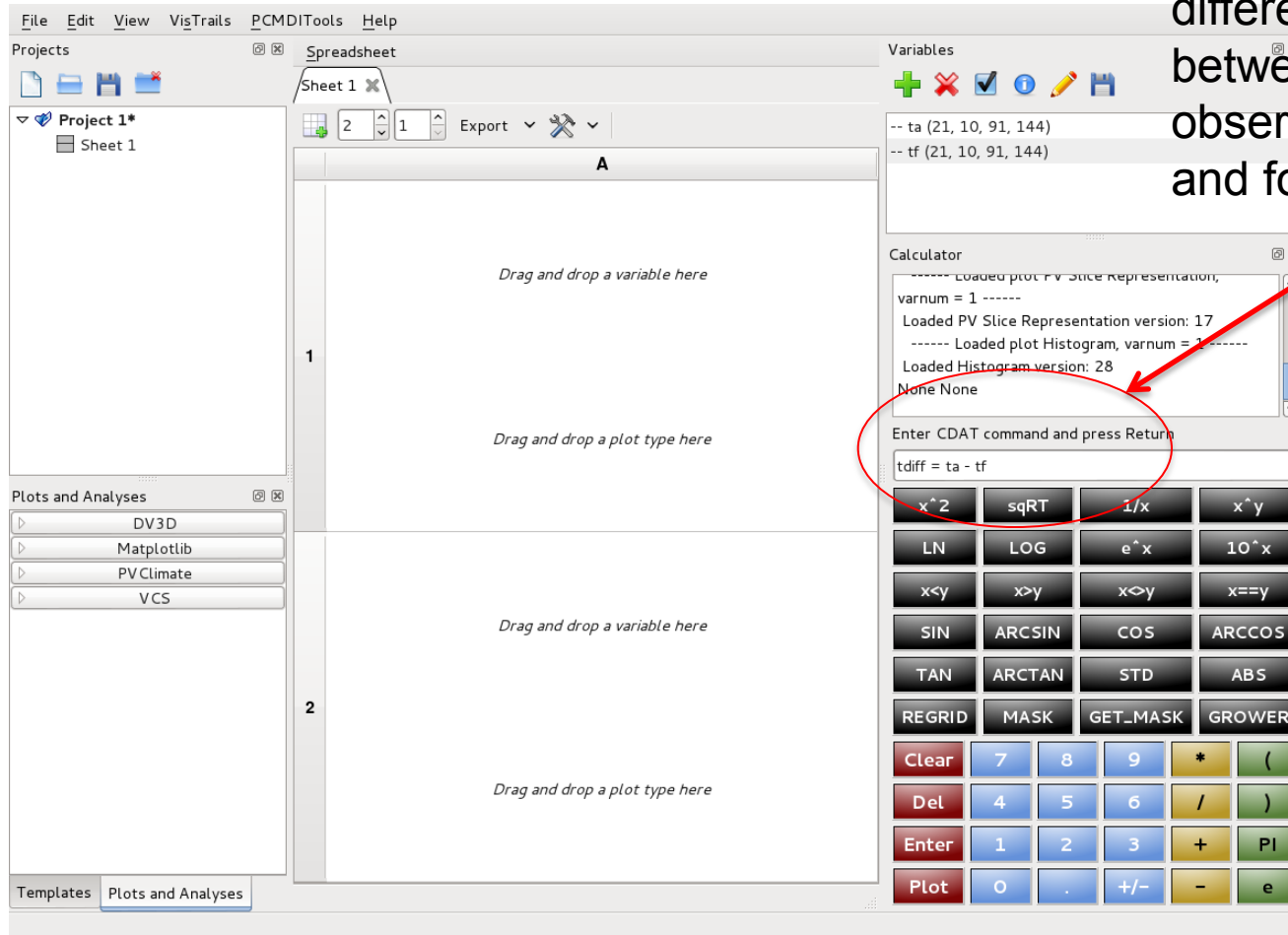
```
-- ta (21, 10, 91, 144)
-- tf (21, 10, 91, 144)
```
- Calculator:** A panel on the right containing a text area with the following text:
 

```
----- Loaded plot PV Slice Representation,
varnum = 1 -----
Loaded PV Slice Representation version: 17
----- Loaded plot Histogram, varnum = 1 -----
Loaded Histogram version: 28
None None
```

 Below the text area is a text input field and a calculator keypad with buttons for mathematical operations and functions.
- Plots and Analyses:** A panel on the left containing a list of plot types: DV3D, Matplotlib, PVClimate, and VCS.

# Defining a new variable on the fly

Taking the difference between observation and forecast

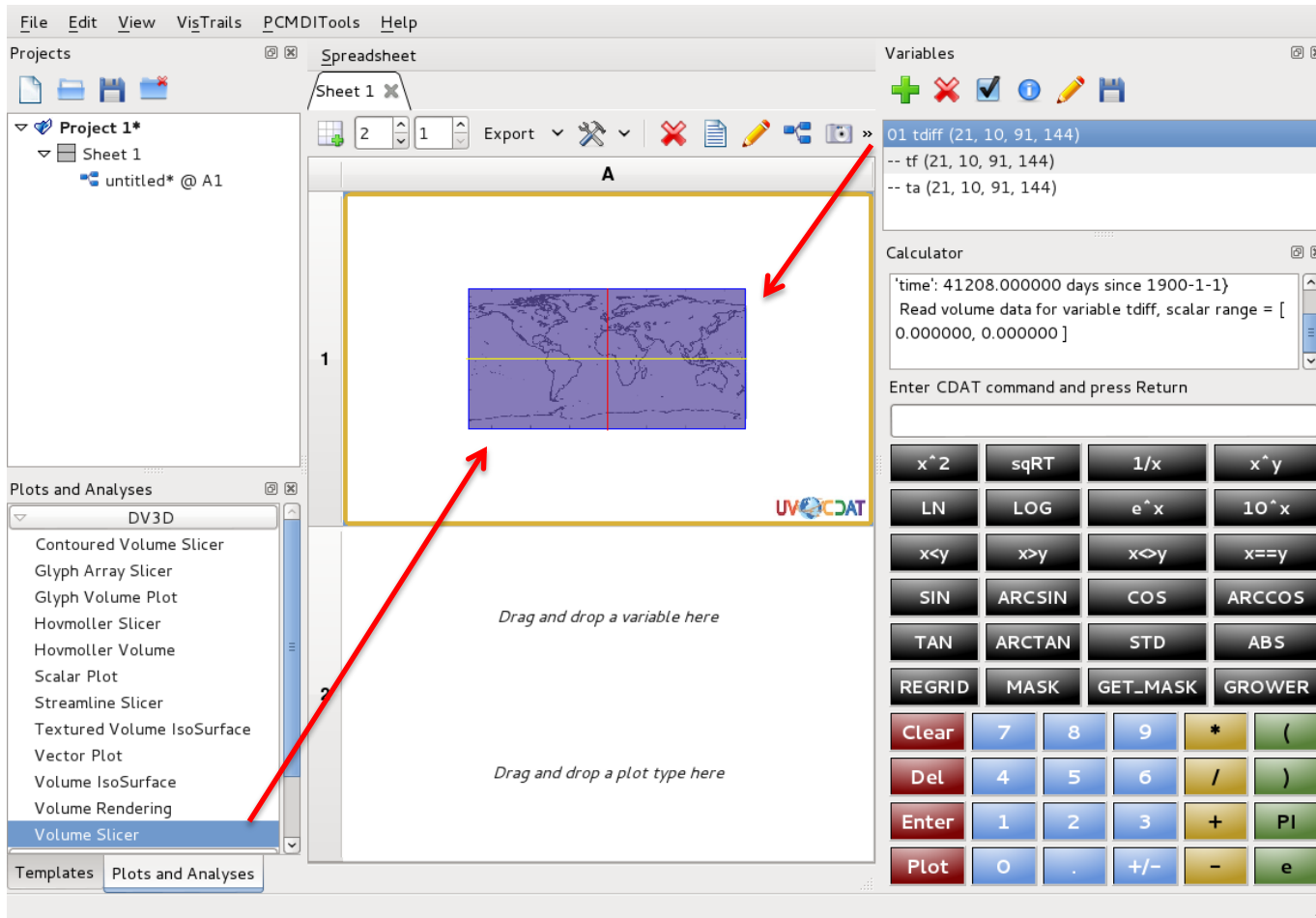


The screenshot displays the software interface with several panels:

- Projects:** Shows 'Project 1\*' with 'Sheet 1' selected.
- Spreadsheet:** The main workspace with a grid and a column header 'A'. It contains two rows with instructions: 'Drag and drop a variable here' and 'Drag and drop a plot type here'.
- Variables:** Lists variables: '-- ta (21, 10, 91, 144)' and '-- tf (21, 10, 91, 144)'. A red circle highlights the input field in the calculator below.
- Calculator:** Shows the command 'tdiff = ta - tf' entered. A red arrow points from the text 'Taking the difference between observation and forecast' to the calculator's input field.
- Plots and Analyses:** Lists analysis tools: DV3D, Matplotlib, PVClimate, and VCS.
- Templates:** Shows 'Plots and Analyses' selected.



# Dragging the variable and the plot method to the scene generates the plot



The screenshot displays the VisTrails software interface. On the left, the 'Plots and Analyses' panel lists various visualization methods, with 'Volume Slicer' selected. In the center, the 'Scene' area contains a world map plot. A red arrow indicates the 'Volume Slicer' being dragged to the scene. Another red arrow shows the 'tdiff' variable being dragged from the 'Variables' panel to the plot. The 'Variables' panel shows the variable 'tdiff' with its value and range. The 'Calculator' panel displays the variable's value and range.

**Variables**

```

O1 tdiff (21, 10, 91, 144)
-- tf (21, 10, 91, 144)
-- ta (21, 10, 91, 144)

```

**Calculator**

```

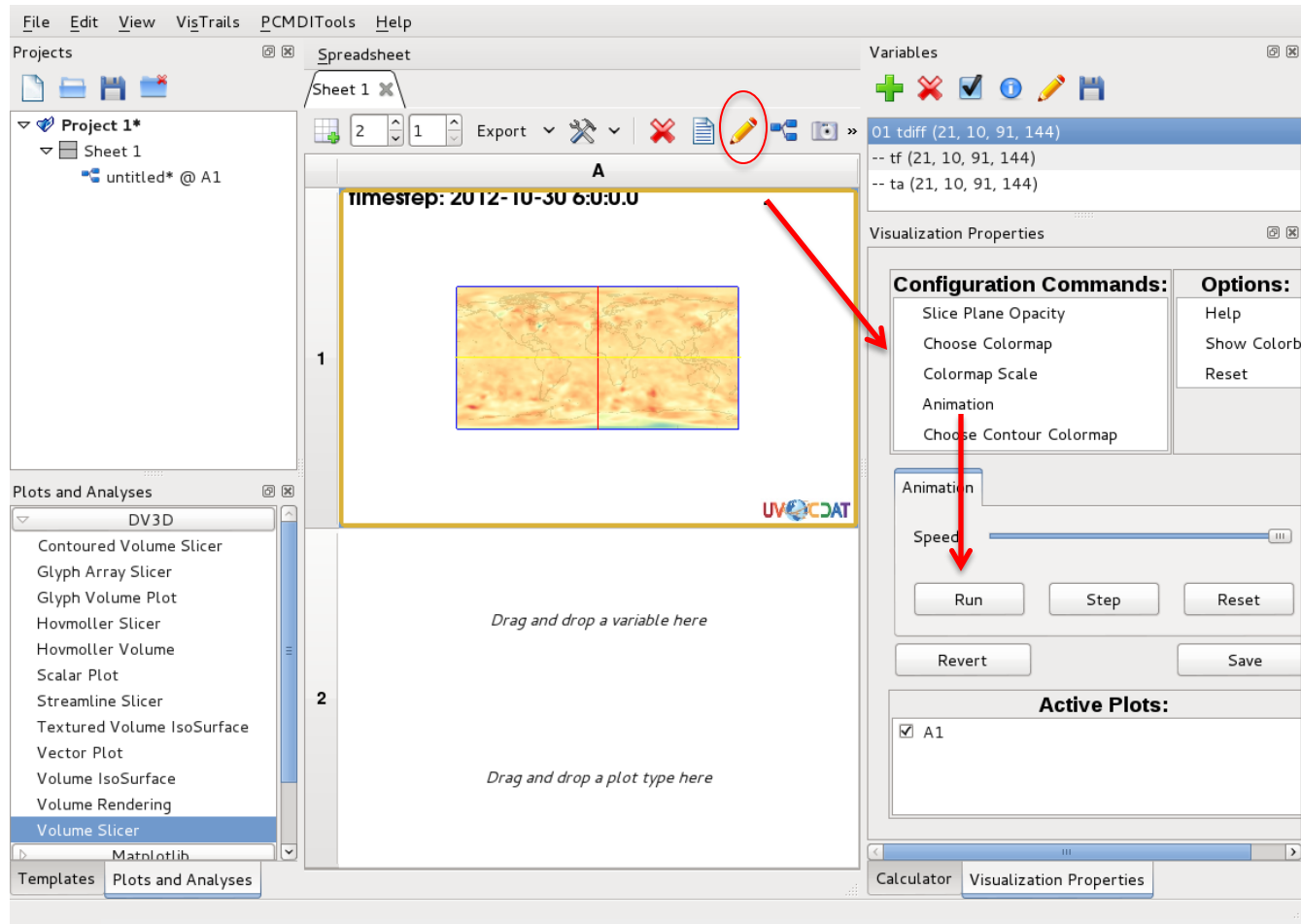
'time': 41208.000000 days since 1900-1-1}
Read volume data for variable tdiff, scalar range = [
0.000000, 0.000000 ]

```

Enter CDAT command and press Return

Calculator buttons:  $x^2$ , sqRT,  $1/x$ ,  $x^y$ , LN, LOG,  $e^x$ ,  $10^x$ ,  $x < y$ ,  $x > y$ ,  $x < > y$ ,  $x == y$ , SIN, ARCSIN, COS, ARCCOS, TAN, ARCTAN, STD, ABS, REGRID, MASK, GET\_MASK, GROWER, Clear, 7, 8, 9, \*, (, Del, 4, 5, 6, /, ), Enter, 1, 2, 3, +, PI, Plot, 0, ., +/-, -, e

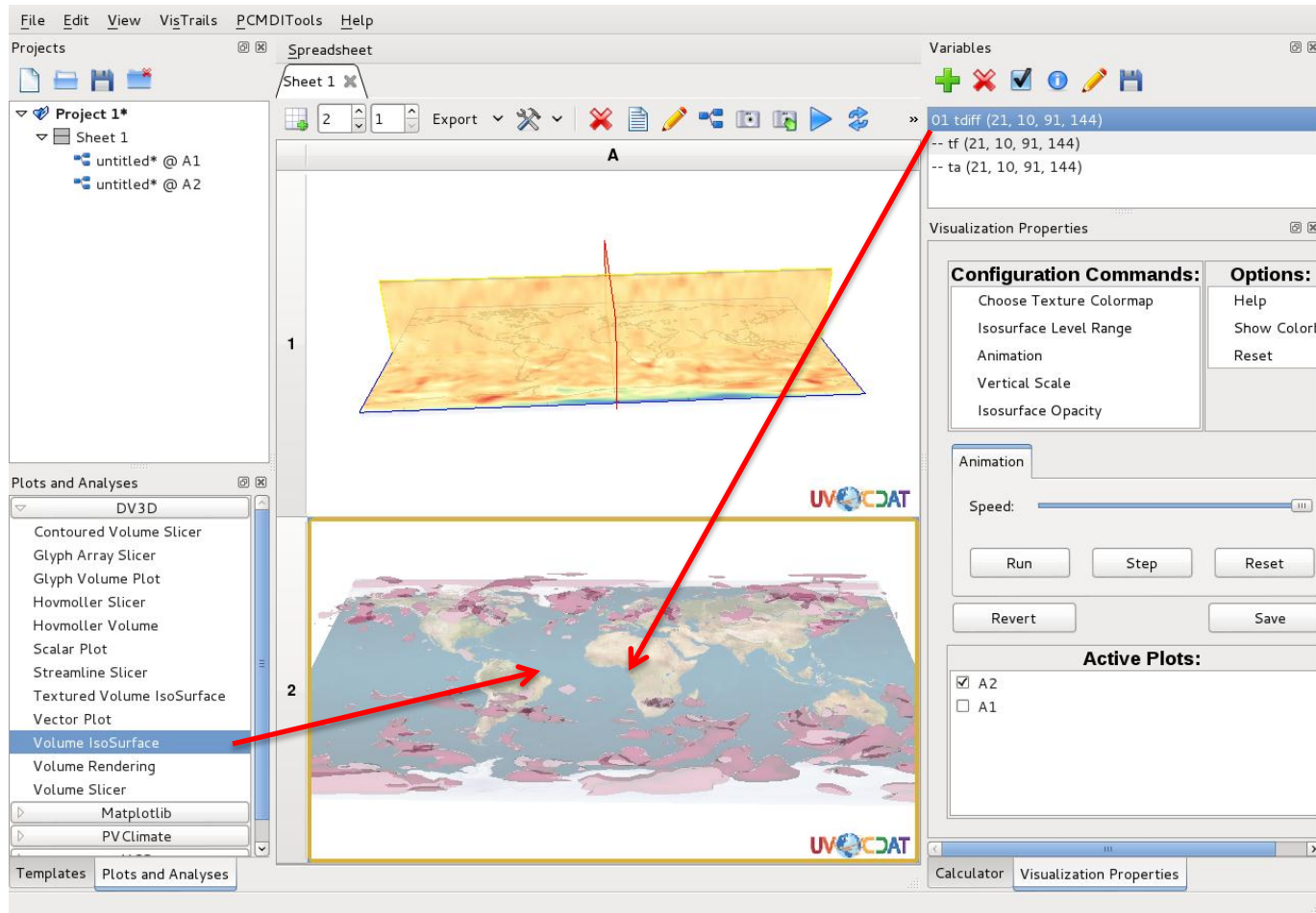
# Selecting the pencil to control the visualization



The screenshot displays the software interface with the following components:

- Toolbar:** A red circle highlights the pencil icon, which is used to edit visualization settings.
- Visualization Area:** A 3D plot titled "timesstep: 2012-10-30 6:0:0.0" showing a cross-section of a simulation. A red arrow points from the pencil icon to the "Choose Contour Colormap" option in the Visualization Properties panel.
- Visualization Properties Panel:**
  - Configuration Commands:**
    - Slice Plane Opacity
    - Choose Colormap
    - Colormap Scale
    - Animation
    - Choose Contour Colormap
  - Options:**
    - Help
    - Show Colorb
    - Reset
  - Animation:**
    - Speed: A slider control.
    - Buttons: Run, Step, Reset, Revert, Save.
  - Active Plots:**
    - A1 (checked)
- Plots and Analyses Panel:**
  - Volume Slicer (selected)
  - Other options: Contoured Volume Slicer, Glyph Array Slicer, Glyph Volume Plot, Hommolter Slicer, Hommolter Volume, Scalar Plot, Streamline Slicer, Textured Volume IsoSurface, Vector Plot, Volume IsoSurface, Volume Rendering.

# Adding volume iso-surface, use mouse to control the camera



The screenshot displays a software interface with the following components:

- File Edit View VisTrails PCMDITools Help** (Menu Bar)
- Projects** (Left Panel): Project 1\* > Sheet 1 > untitled\* @ A1, untitled\* @ A2
- Spreadsheet** (Top Center): Sheet 1, grid view, Export, and various editing tools.
- Variables** (Top Right):
  - 01 tdiff (21, 10, 91, 144)
  - tf (21, 10, 91, 144)
  - ta (21, 10, 91, 144)
- Visualization Properties** (Right Panel):
  - Configuration Commands:** Choose Texture Colormap, Isosurface Level Range, Animation, Vertical Scale, Isosurface Opacity.
  - Options:** Help, Show Colorb, Reset.
  - Animation:** Speed slider, Run, Step, Reset buttons.
  - Active Plots:**  A2,  A1.
- Plots and Analyses** (Bottom Left):
  - DV3D
  - Contoured Volume Slicer
  - Glyph Array Slicer
  - Glyph Volume Plot
  - Hovmoller Slicer
  - Hovmoller Volume
  - Scalar Plot
  - Streamline Slicer
  - Textured Volume IsoSurface
  - Vector Plot
  - Volume IsoSurface** (highlighted)
  - Volume Rendering
  - Volume Slicer
  - Matplotlib
  - PV Climate
- 3D Visualization (Top Center):** A 3D plot showing a volume iso-surface (yellow/orange) over a 2D map (A).
- 2D Visualization (Bottom Center):** A 2D map plot (B) showing a global map with a red arrow pointing to the same location as the 3D plot.



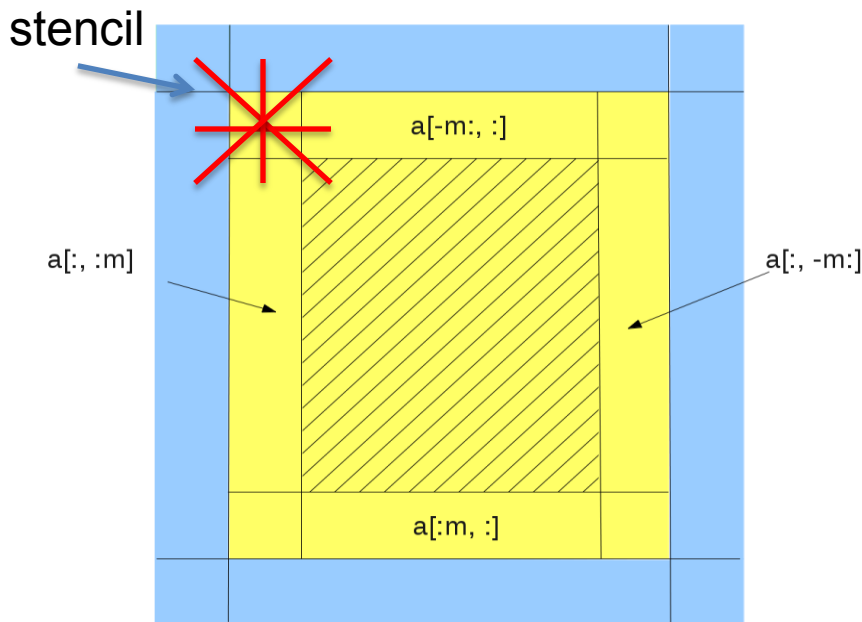
# UV-CDAT can leverage parallelism at different levels

- Intra-component:
  - ◆ ESMF (user executes `mpiexec` with ESMF handling the domain decomposition)
  - ◆ ParaView (in a client server environment)
- MPI job: User can impose his/her own domain decomposition
  - ◆ with the Python `mpi4py` module
  - ◆ N-dimensional distributed arrays (also based on `mpi4py`). Suited for discretization involving communication between neighbors.

# UV-CDAT's distributed array works in any number of dimensions

- Based on MPI-2 remote memory access

- Example of 4 data windows shared to other procs.
- Python syntax allows slices to be expressed for any data size (m is the halo width)



Yellow represents data owned by a processor

Shaded area represents data private to the proc.

Any proc. can access data in the yellow, non-shaded blocks

No need for yellow proc. to hold data in the blue Regions (no exterior ghosts)



# Distributed array API is minimalistic and syntax follows numpy

Distarray uses `mpi4py` (by Lisandro Dalcin) as communication layer and the class derives from `numpy`

```
from mpi4py import MPI
import numpy
import distarray

# MPI stuff
comm = MPI.COMM_WORLD
rank = comm.Get_rank(); nprocs = comm.Get_size()

# create distarray
da = distarray.daZeros( (2, 3), numpy.float32 )

... # populate da

# expose sub-domain to other procs
north = ( slice(-1, None, None), slice(0, None, None) )
da.expose(slce = north, winID = 'north' )

# access the north slabs of south neighbors (collective)
northData = da.get( (rank - 1)%nprocs, winID='north' )

# clean up
da.free()
```



# Things get easier when using ghosted distarray

- Choose the thickness of the halo and the windows will be constructed for you

```
from mpi4py import MPI
import numpy
import distarray
```

```
# create ghosted distarray
```

```
ga = distarray.ghZeros( (2, 3), numpy.float32, 1 )
```

Number of ghost points



```
... # populate ga
```

```
# access the north slabs of south neighbors
(collective)
```

```
northData = ga.get( (rank - 1)%nprocs, winID=(1,0) )
```

```
# clean up
```

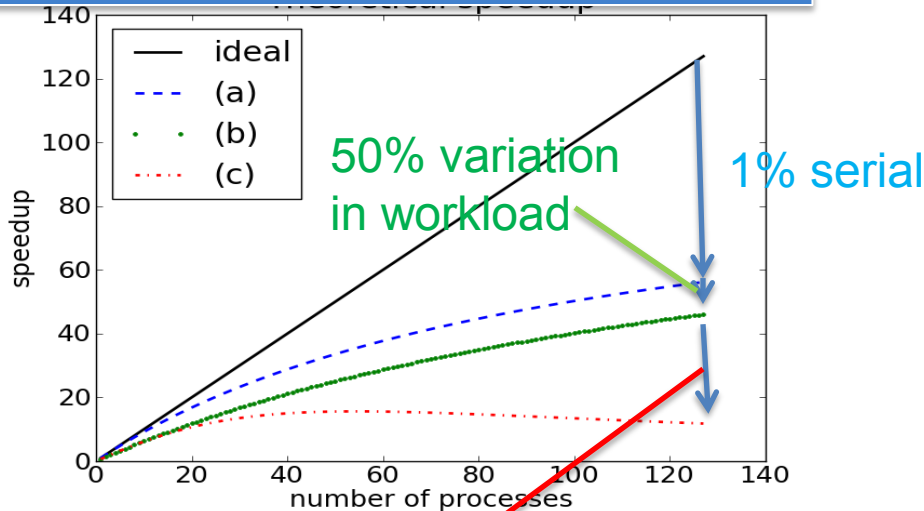
```
da.free()
```



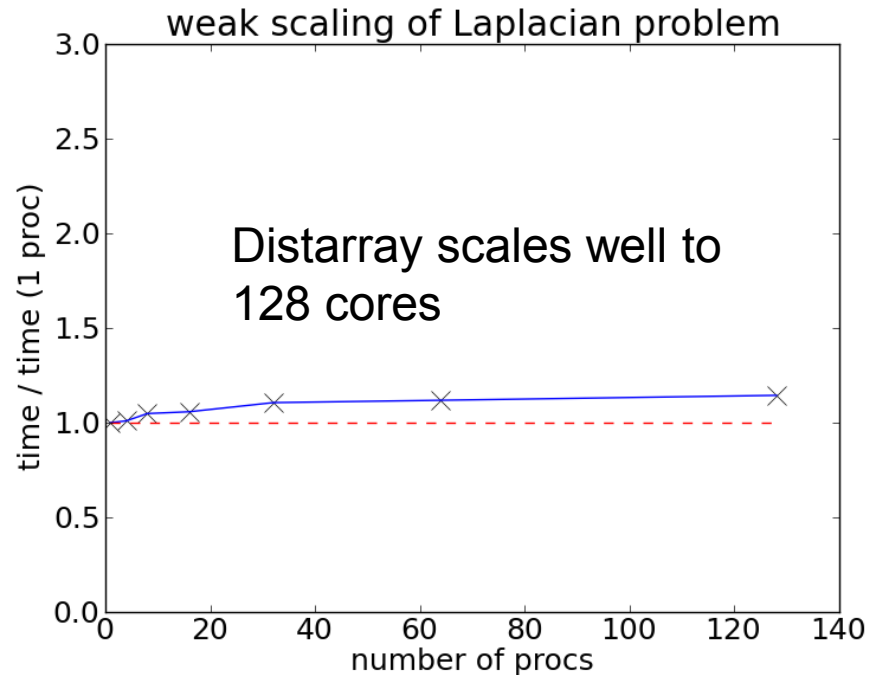
# Amdahl's law, load balancing, and communication limit parallel scaling

- Amdahl's law and imperfect load balance tend to flatten the scaling
- Communication cost tends to introduce negative scaling

Typical speedup of an application



Linearly increasing communication cost



Local resolution  $2000^2$  kept constant



# Summary and future work

- January 8 2013: UV-CDAT 1.2 will be released
- Check out <http://uvcdat.llnl.gov/>
- How to get CDAT users to transition to UV-CDAT and use the latest viz and parallel computing facility
- How to leverage parallelism in the UV-CDAT GUI?
- Other forms of parallelism: threads, task farming, GPUs...
- Postprocessing and visualization require accurate understanding of the data's stagger location (Arakawa C-D, edge/face)

Thanks for your attention!

