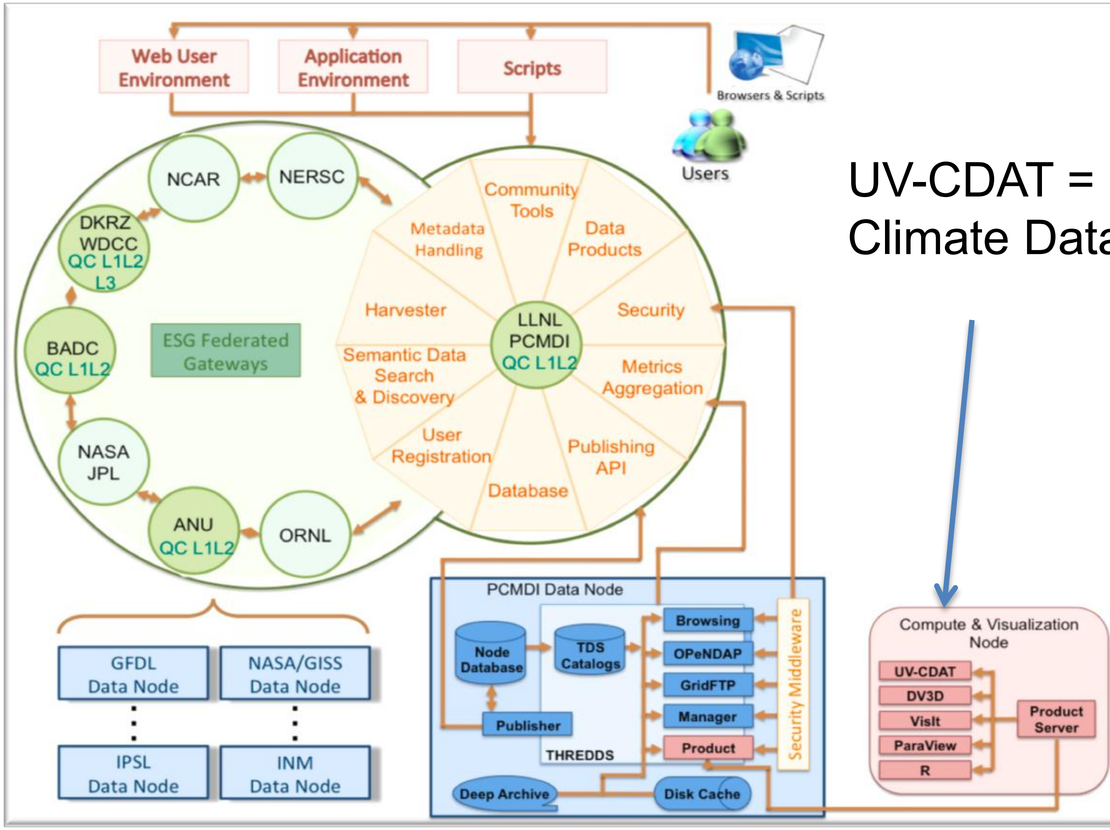
ULTRASCALE CLIMATE DATA VISUALIZATION WITH UV-CDAT

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

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

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

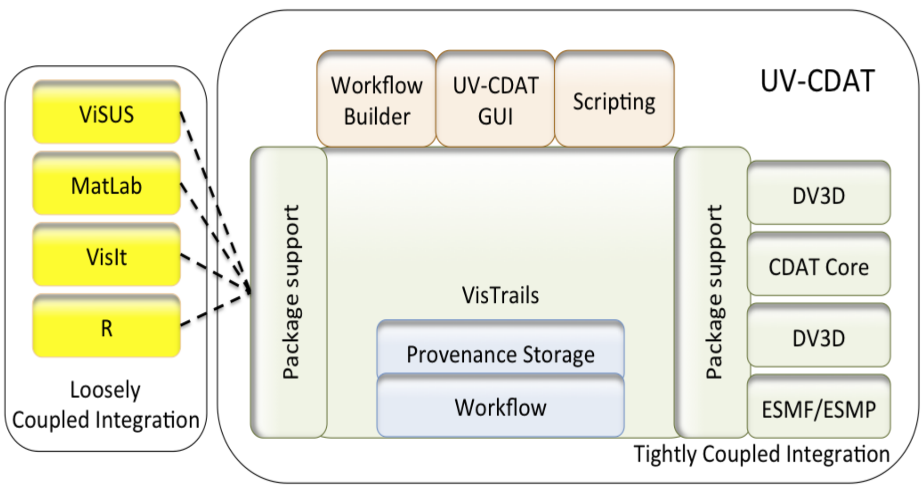
- Used by the Earth System Grid Federation to serve data
- **UV-CDAT is the successor of CDAT (Climate Data Analysis Tools)**



UV-CDAT = Ultrascale Visualization Climate Data Analysis Tools

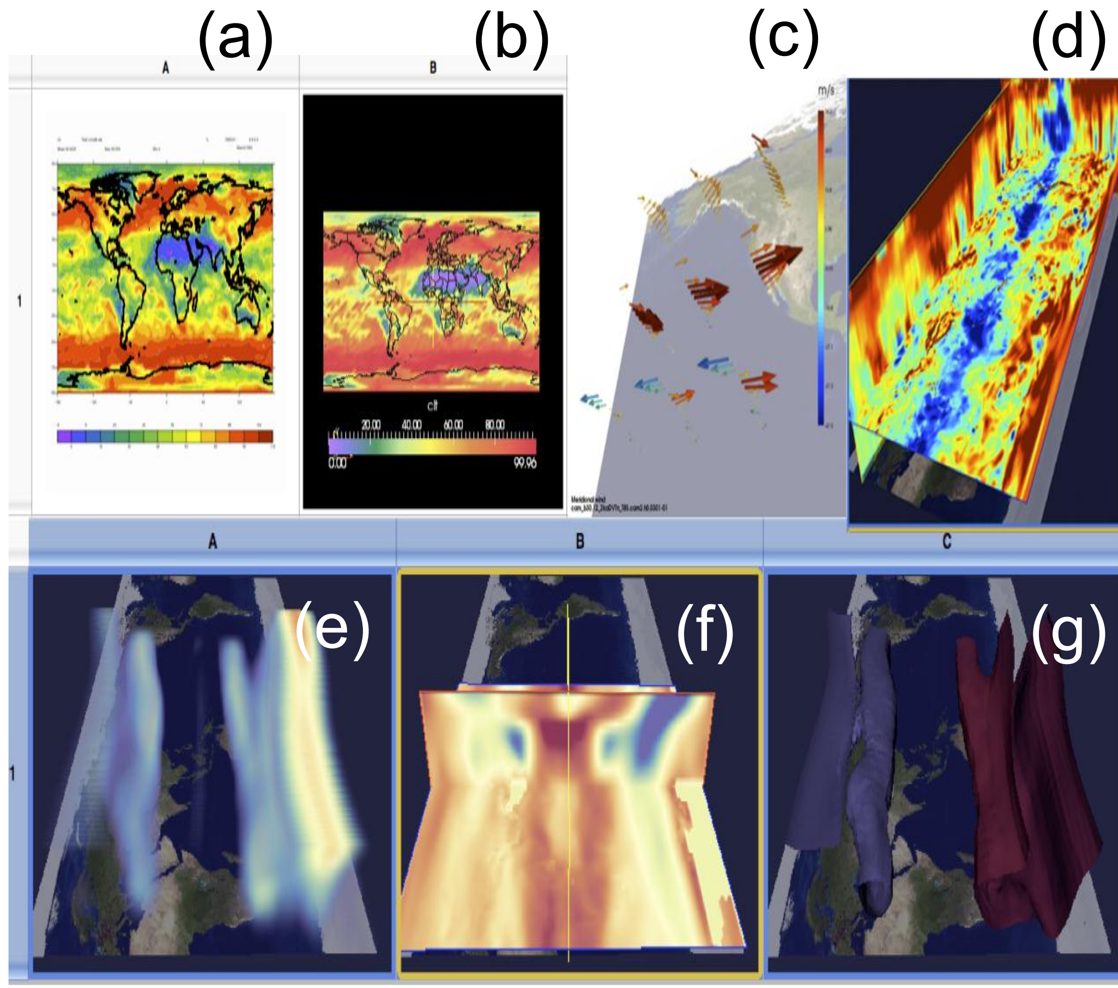
UV-CDAT's architecture and build system are designed to scale up

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

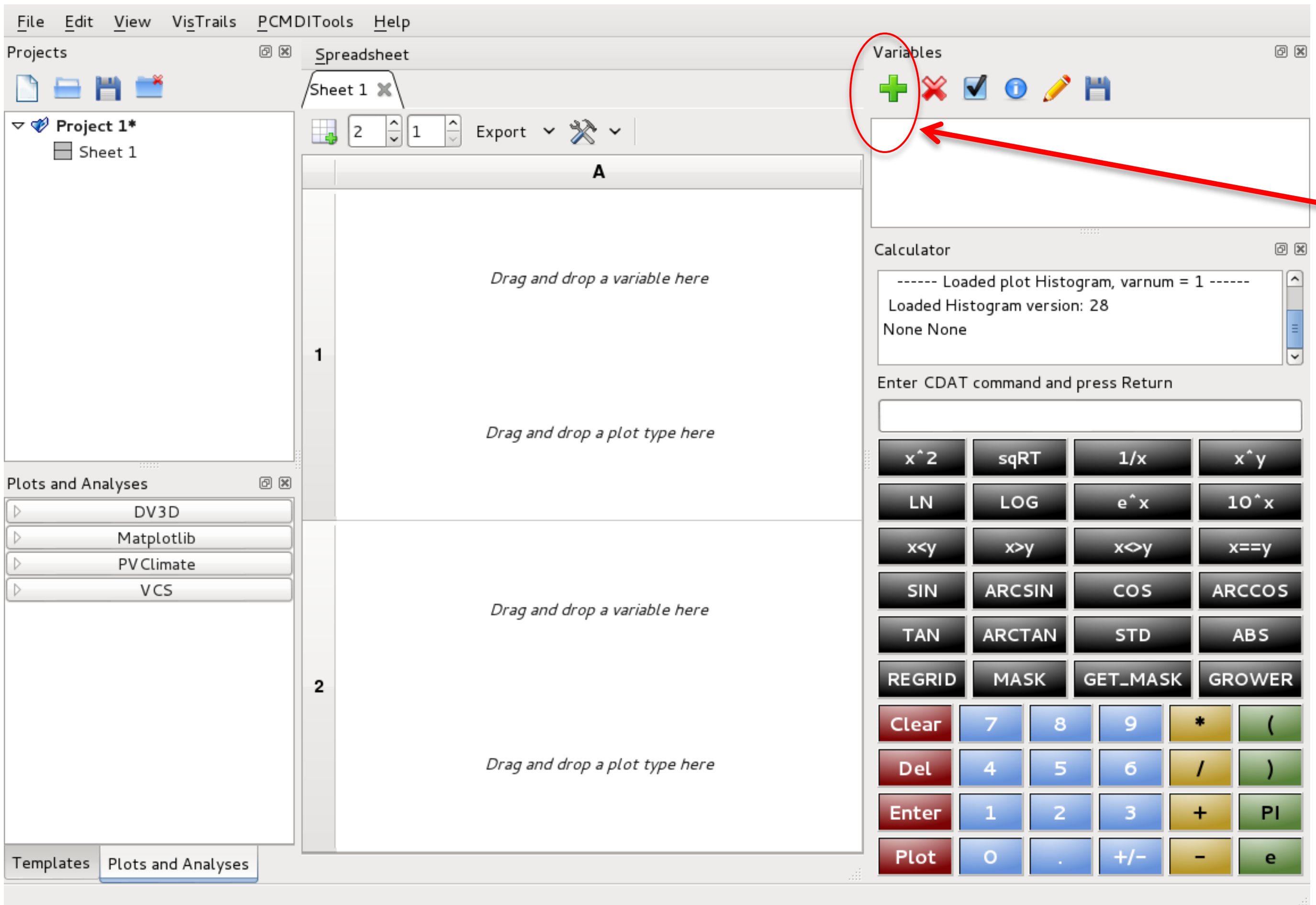


Multiple visualization tools can be invoked from within UV-CDAT

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

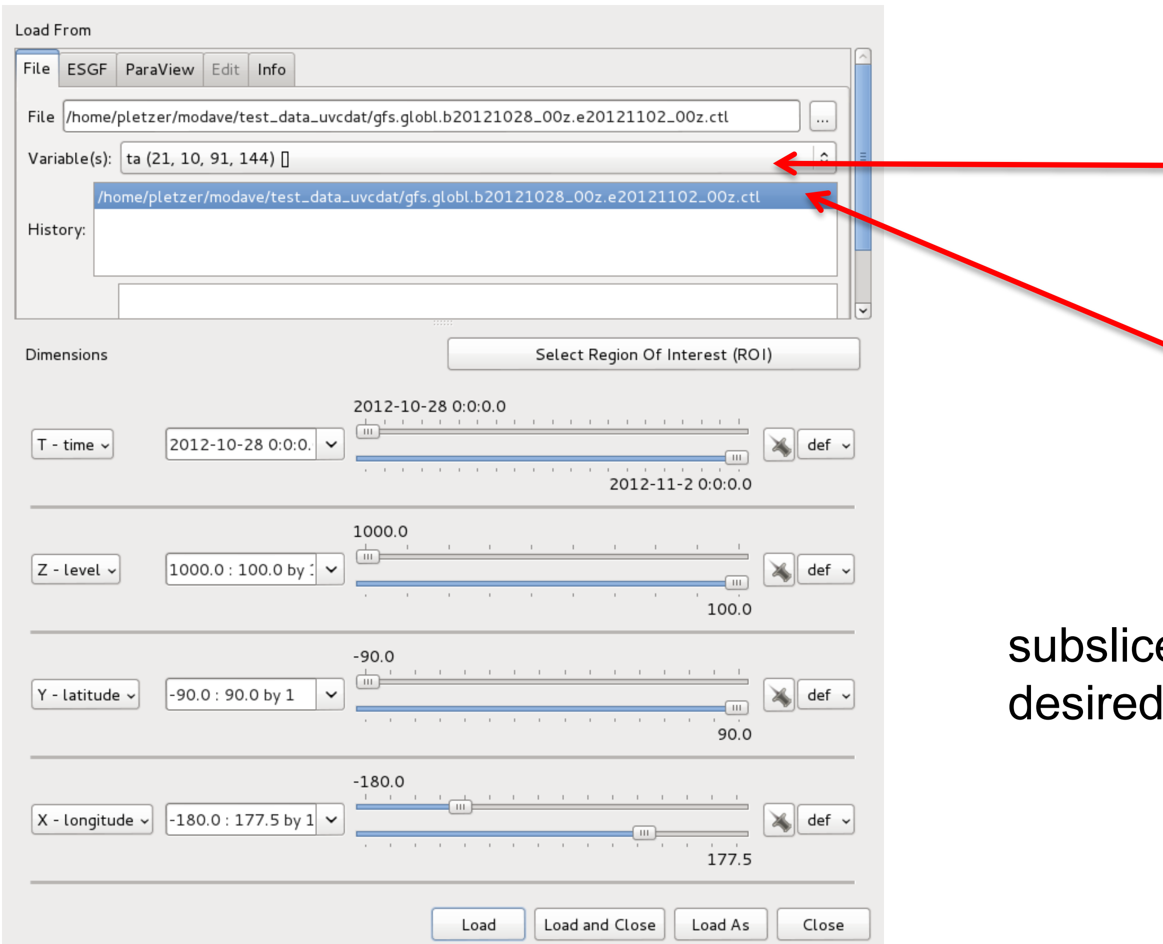


UV-CDAT's start up graphical user interface



Open file and add variable

Selecting the file and variable

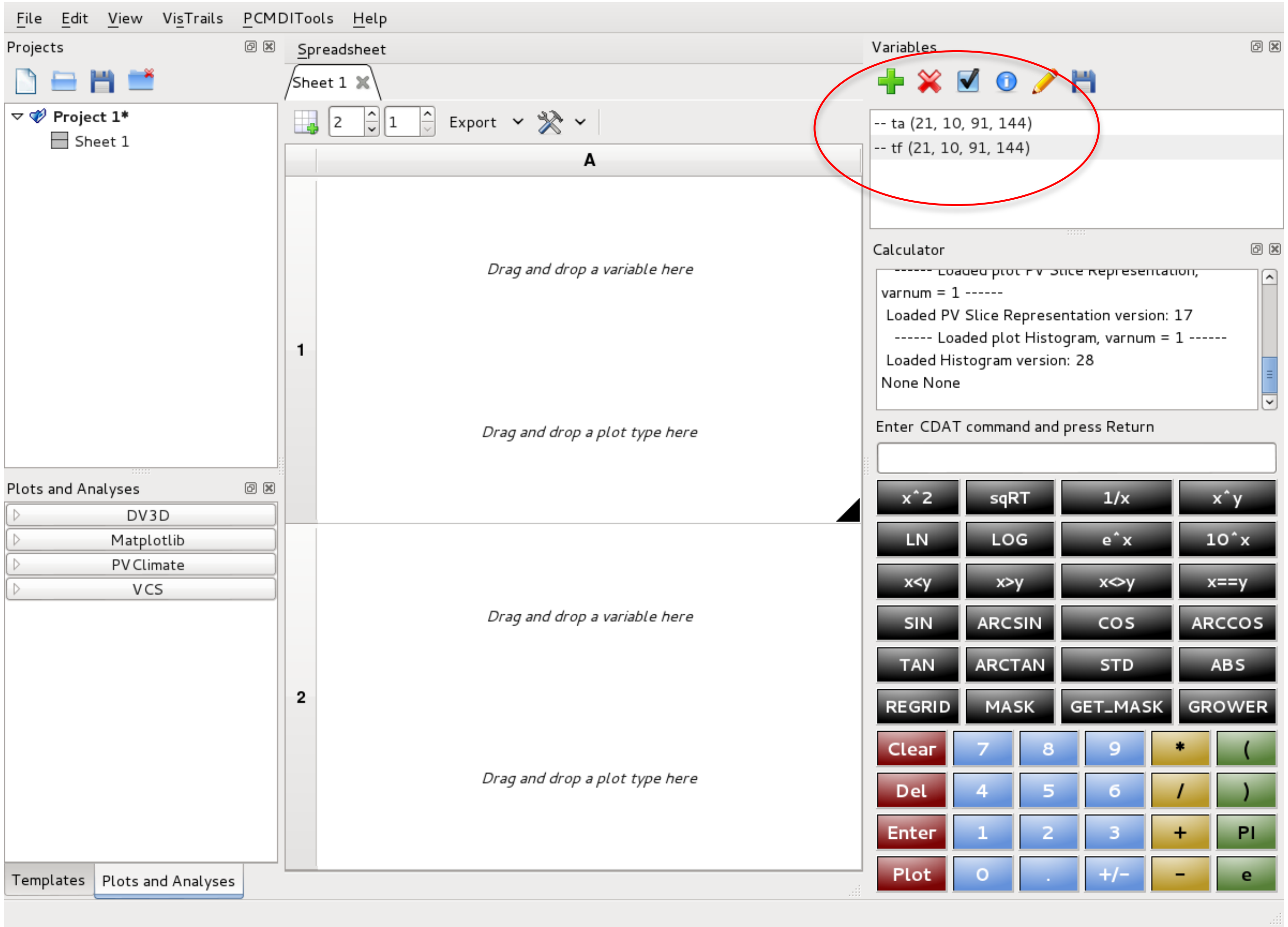


Variable name

File name

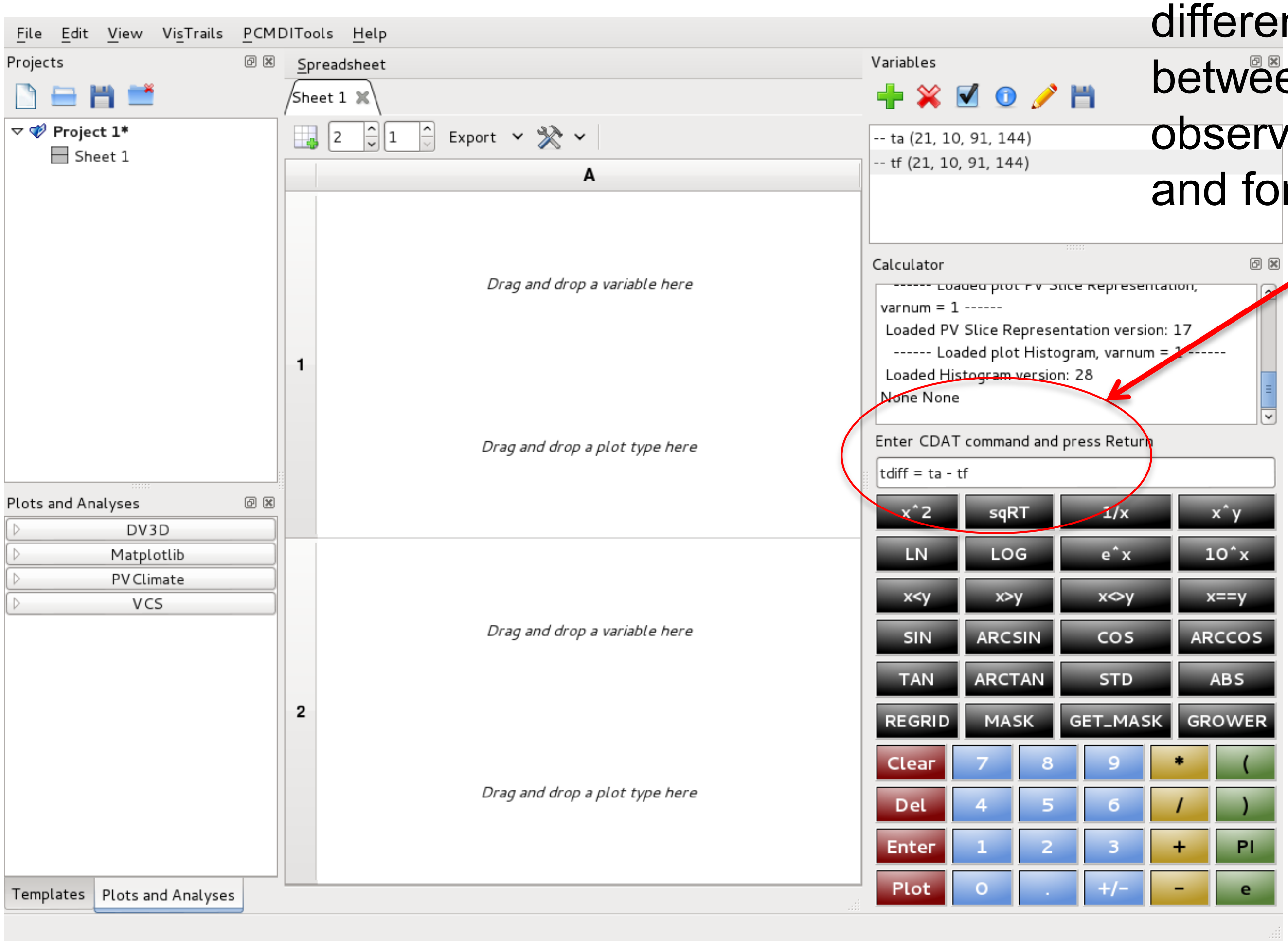
subslice the variable if desired

After loading the variables ta and tf

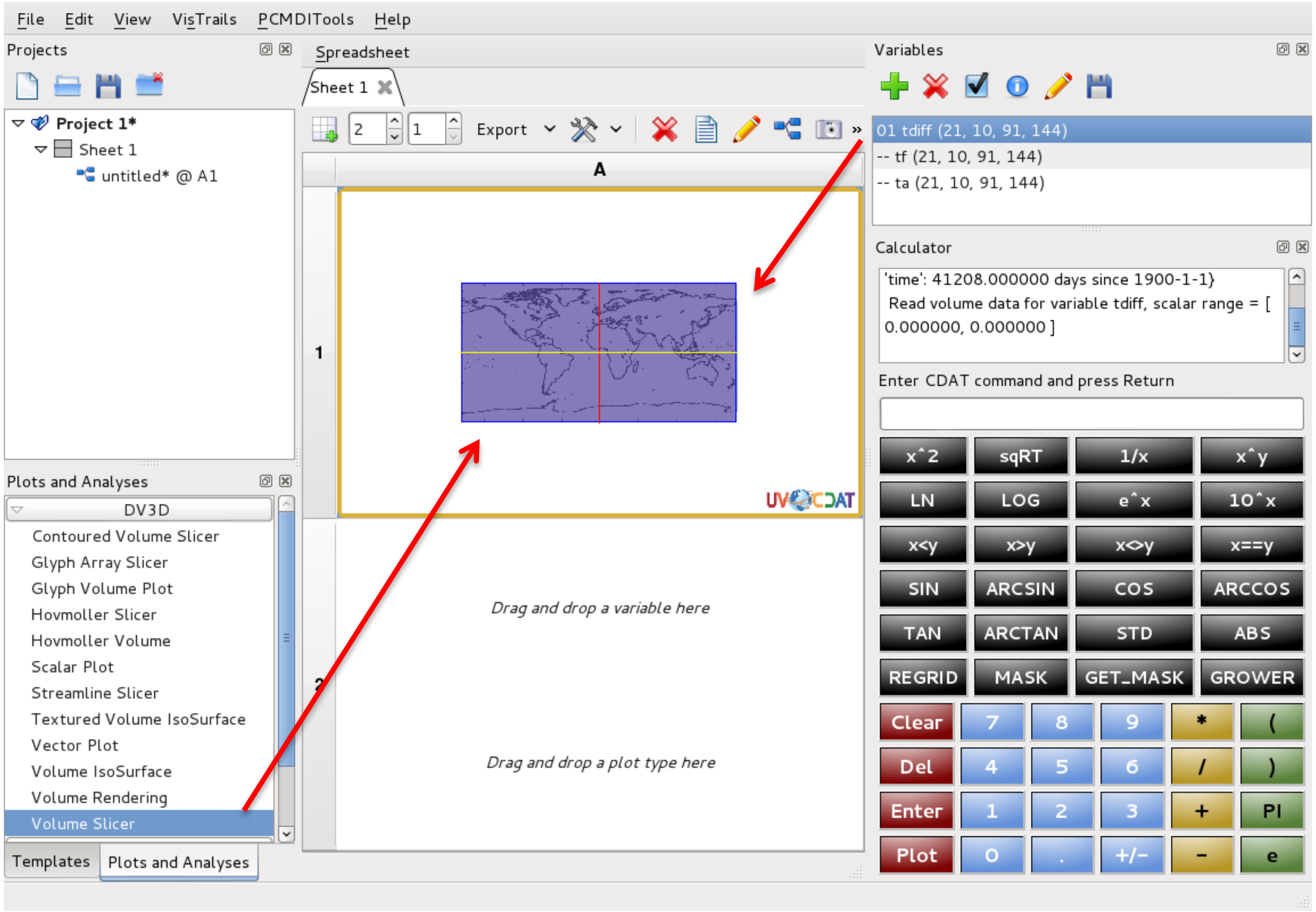


Defining a new variable on the fly

Taking the difference between observation and forecast

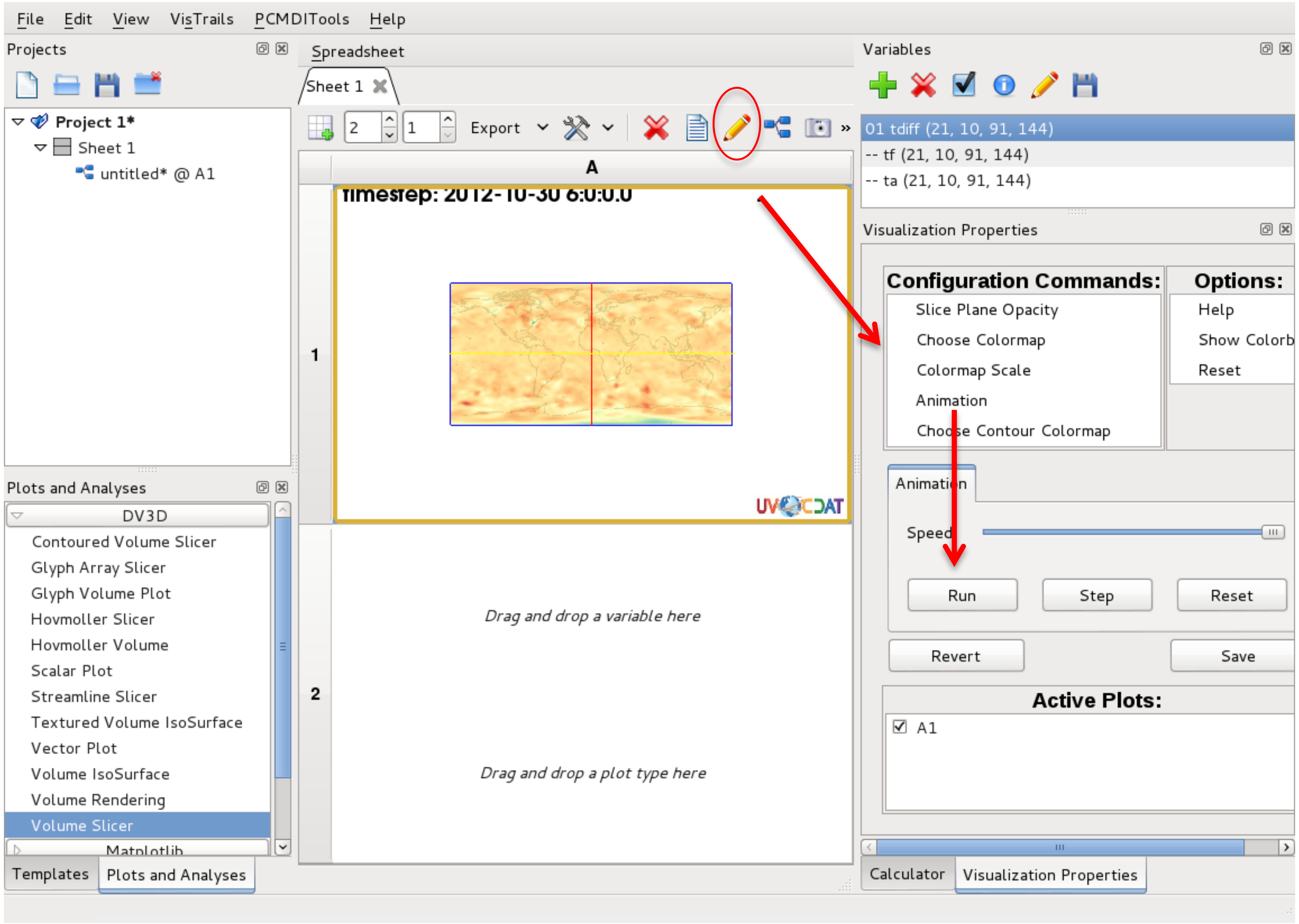


Dragging the variable and the plot method to the scene generates the plot

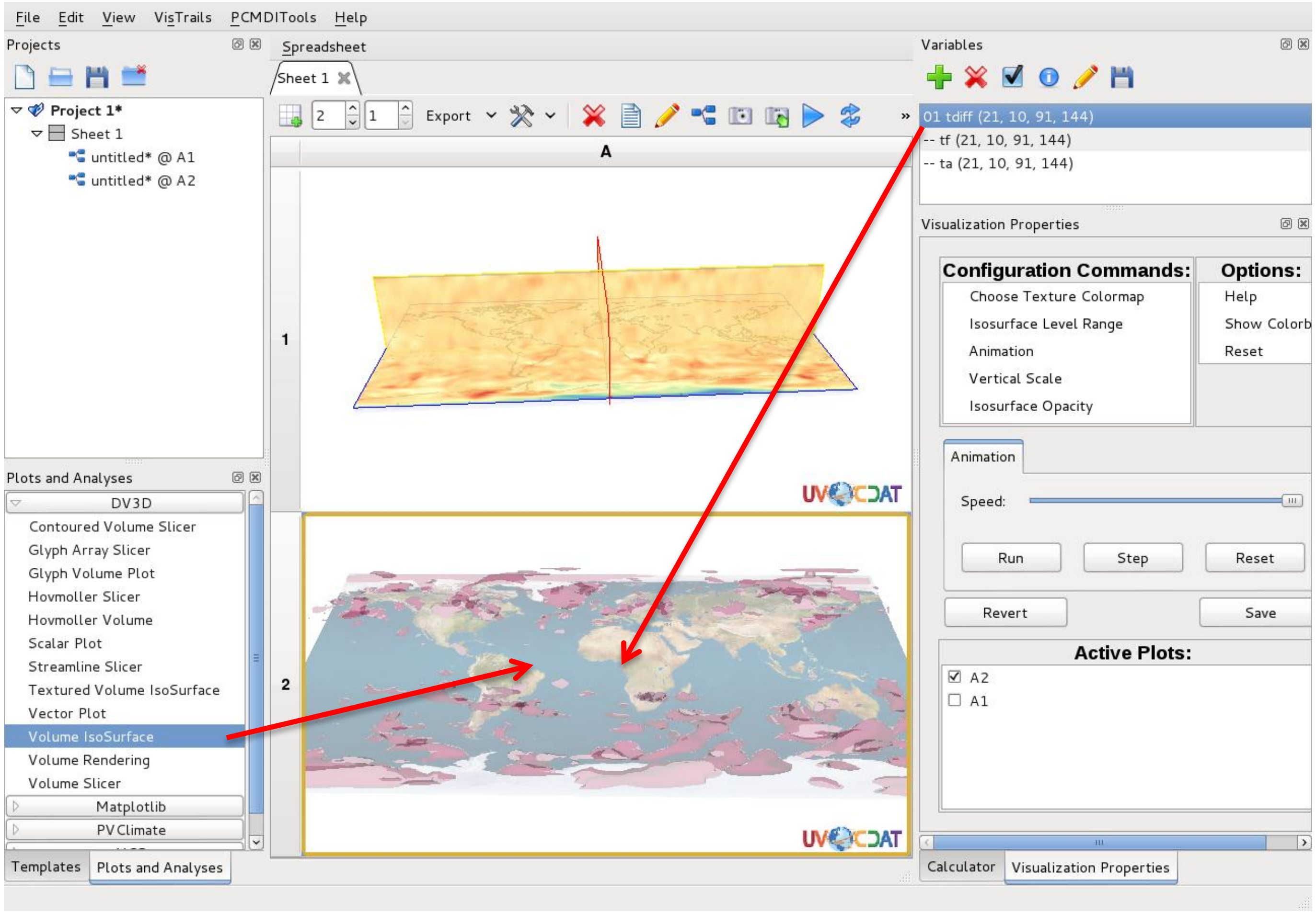


SIMULATIONS EMPOWERING YOUR INNOVATIONS

Selecting the pencil to control the visualization



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



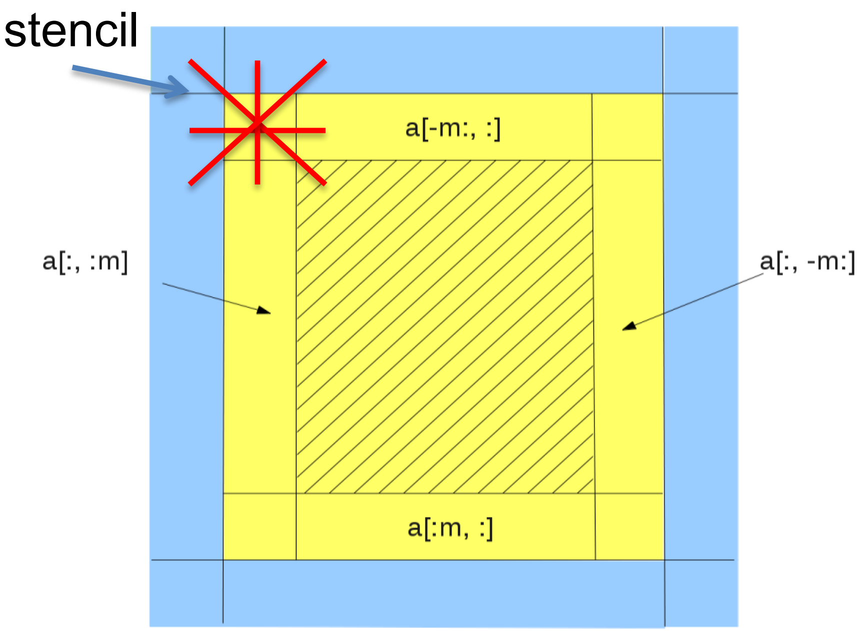
UV-CDAT can leverage parallelism at different levels

- 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

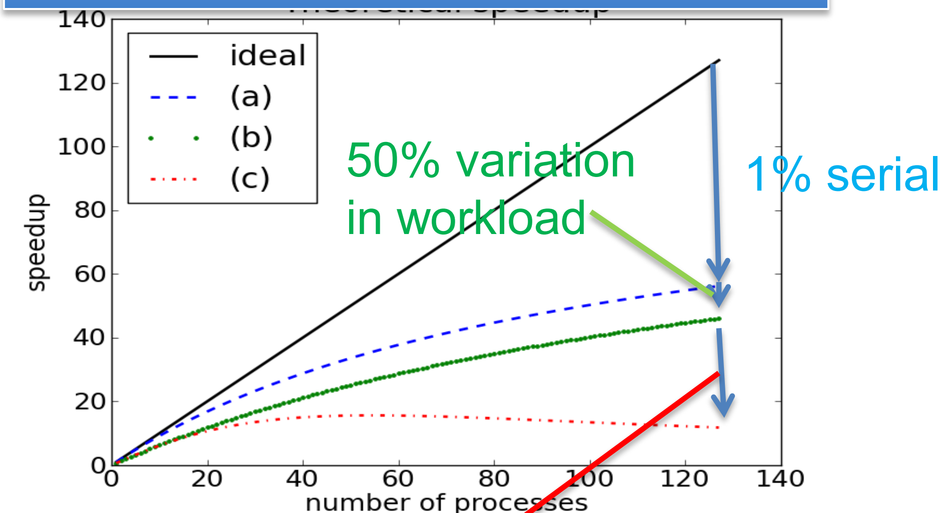
Number of ghost points

```
from mpi4py import MPI
import numpy
import distarray

# create ghosted distarray
ga = distarray.ghZeros( (2, 3), numpy.float32, 1 )

… # populate ga

# access the north slabs of south neighbors
(collective)
northData = ga.get( (rank - 1)%nprocs, winID=(1,0) )

# clean up
da.free()
```

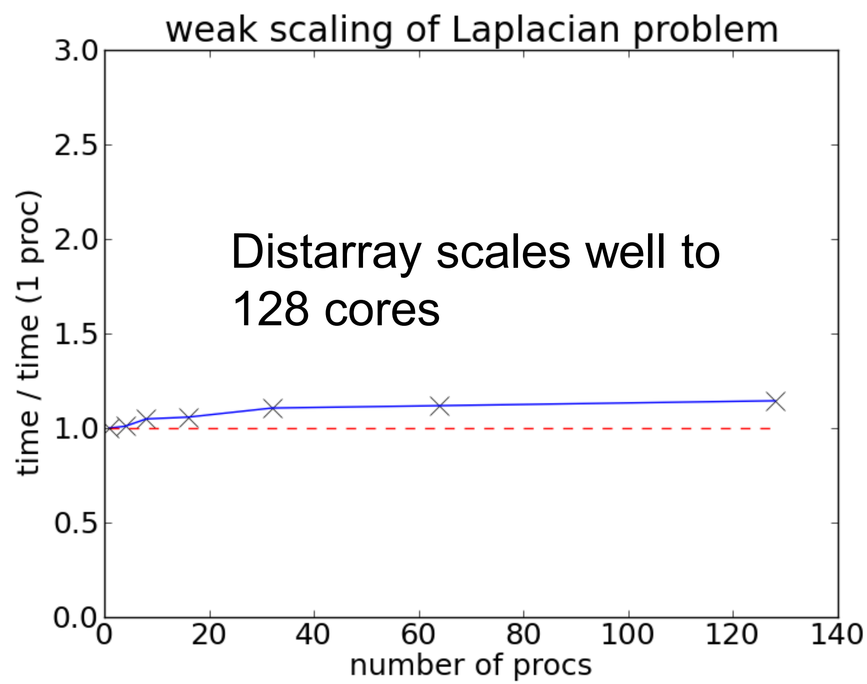
Amdahl's law, load balancing, and communication limit parallel scaling

- 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!