Open-Source Tools for the Community Satellite Processing Package

R.Garcia, K.Strabala, E.Schiffer, G.Cureton

ABSTRACT

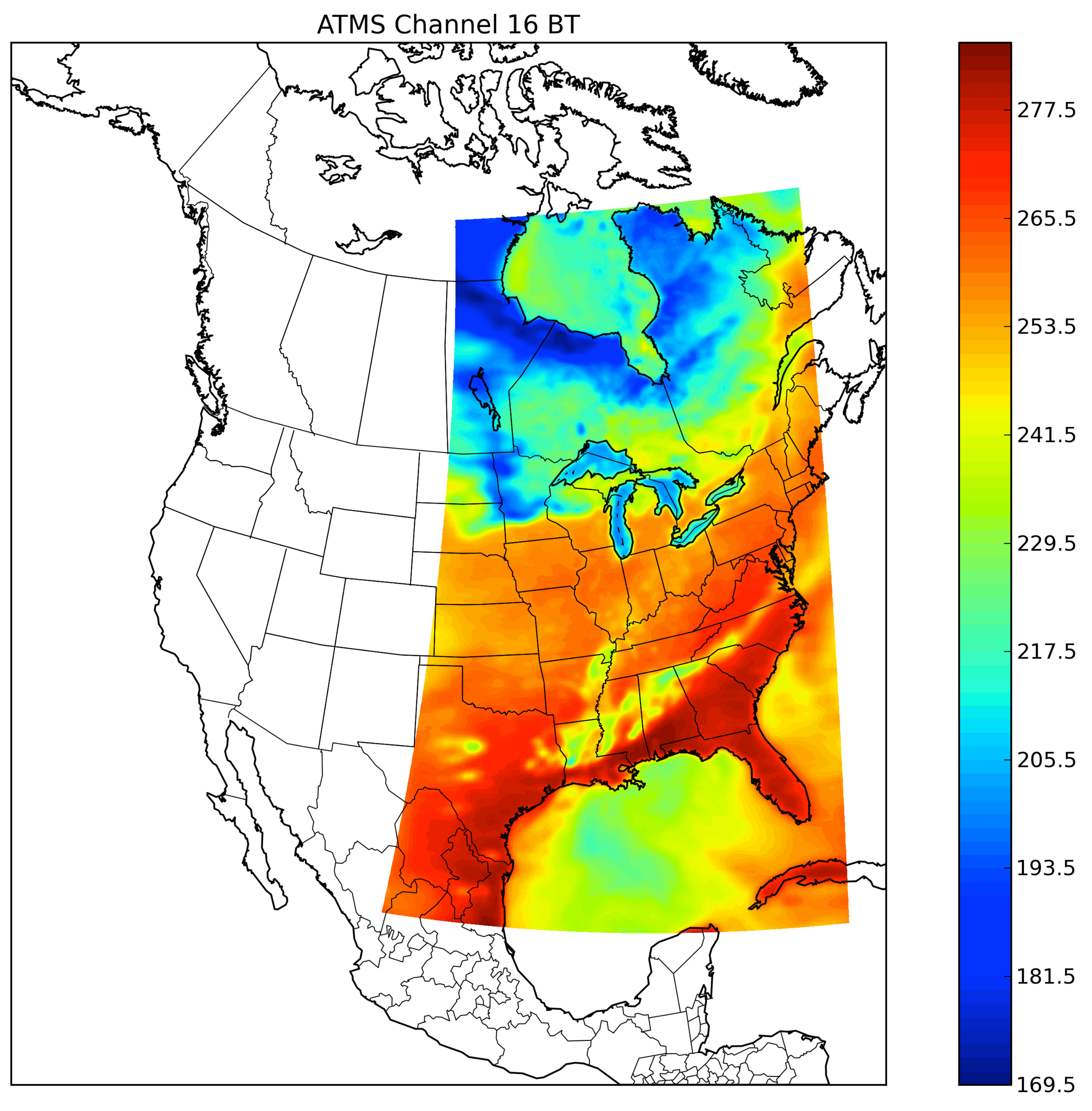
The Community Satellite Processing Package (CSPP) provides worldwide users an easy-to-install, easy-to-use data processing capability for NPP satellite observations. In this technical presentation we illustrate open-source tools, techniques and work-flows for bringing NPP product data to end-users quickly and easily. This includes generation of quick-look imagery and verification reports, forming interactive visualizations, and creating bridges to third-party services capable of manipulating direct-broadcast (DB) meteorological data. Also included is discussion of techniques and software used in the construction of the CSPP.

CSPP SOFTWARE

The CSPP bundles precompiled open-source software in C++ and Fortran with script automation. This simplifies deployment, processing and manipulation of NPP data for end-users. In order to manage the complexity of the NPP processing as implemented in the Raytheon Algorithm Development Library (ADL) hosted operational algorithms, Python has been included in the CSPP tool set. The included Python script environment ("ShellB3") has many of the functions available in the commercial IDL and MATLAB packages built-in. Also included are modules immediately useful for NPP data access, on top of which SSEC provides automation utilities, wrappers, and diagnostic tools.

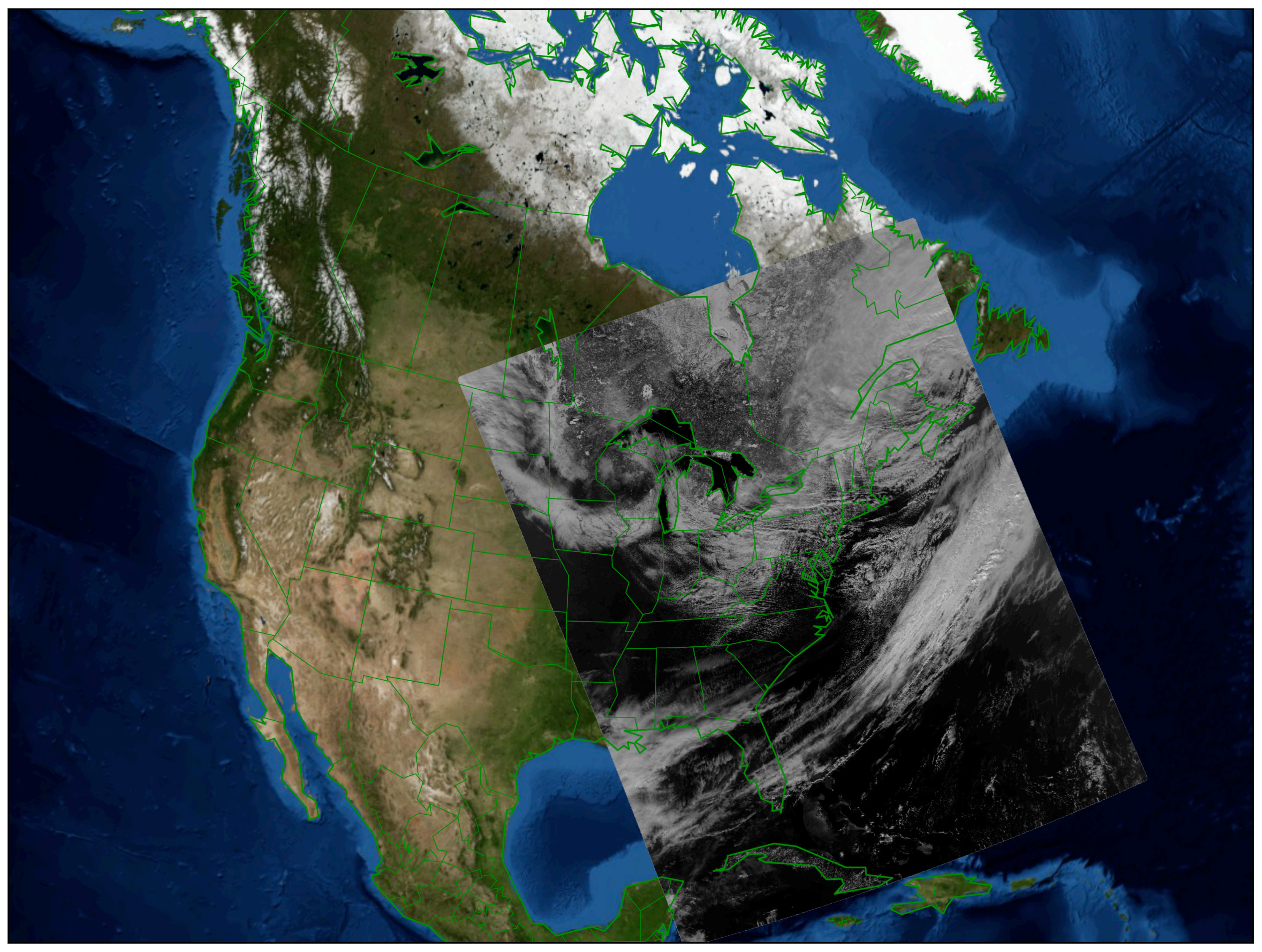
Python Module	CSPP Usage
h5py + pytables	Access NPP HDF5 files
xml.etree.ElementTree	Parse ADL guidebook XML files
numpy + scipy	Large matrix operations
pygrib	Read NWP for EDR ancillary creation
ctypes	Read/write "BLOB" ADL file formats
pyresample	Resample swaths to gridded images
matplotlib + basemap	Create raster/vector images
netcdf4py + pycdf	Access/create NetCDF3/4 for AWIPS

ATMS SDR GRANULE QUICKLOOKS



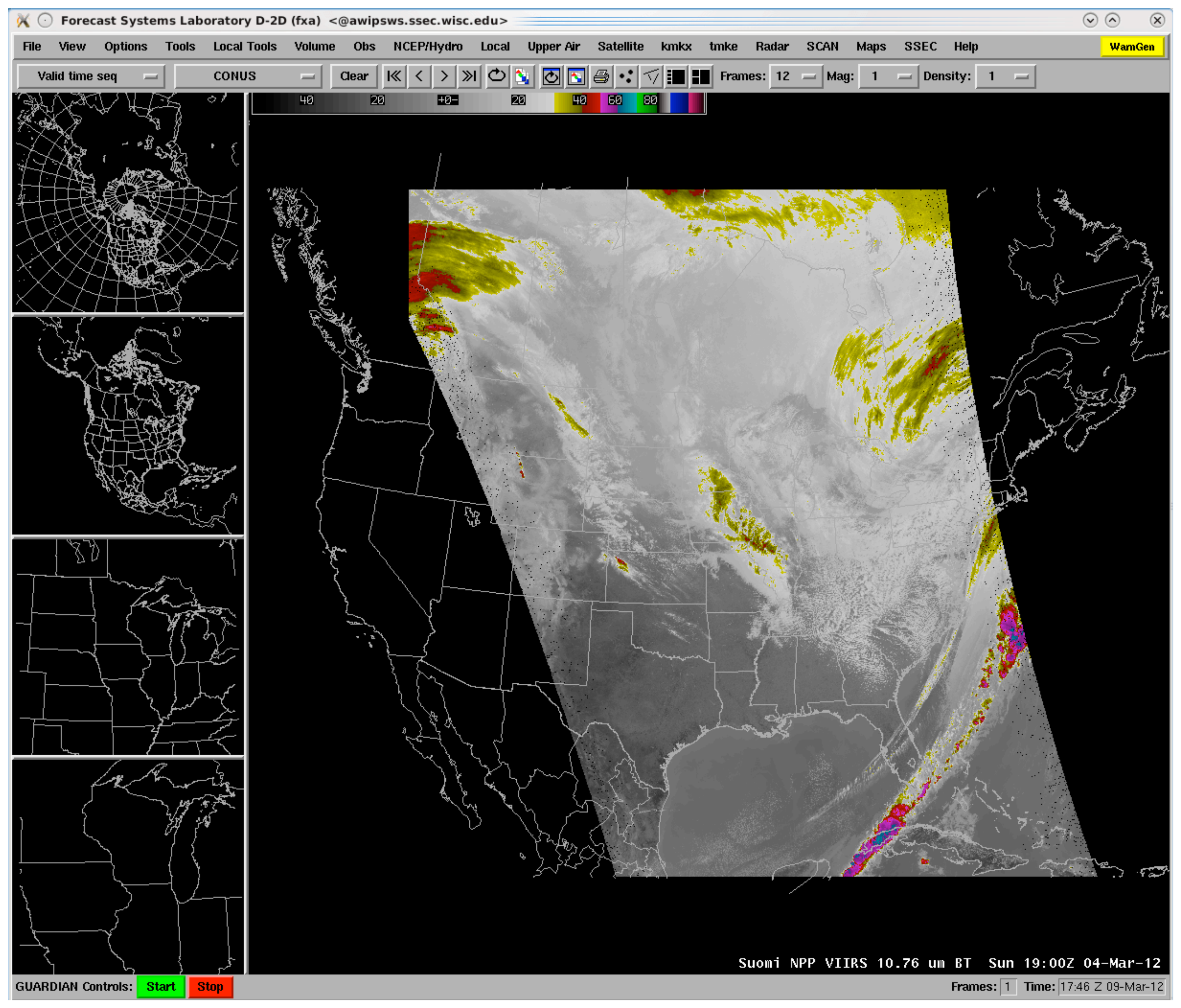
```
# load libraries
from pylab import *
import h5py, numpy, glob
from mpl_toolkits.basemap import Basemap

# open a directory with a pass of CSPP SDR files in time order
sdrs = [h5py.File(filename) for filename in sorted(glob.glob('data/SATMS*'))]
# read all unscaled BTs, and their scaling slope and intercept
bts = [f['All_Data']['ATMS-SDR_All']['BrightnessTemperature'][:] for f in sdrs]
btscale = [f['All_Data']['ATMS-SDR_All']['BrightnessTemperatureFactors'][:] for f in sdrs]
# scale them and concatenate into a contiguous array
atms_bt = numpy.concatenate([(x*m+b) for (x,(m,b)) in zip(bts,btscale)])

# load latitude and longitude arrays
geos = [h5py.File(filename) for filename in sorted(glob.glob('data/GATMO*'))]
lat = numpy.concatenate([f['All_Data']['ATMS-SDR-GEO_All']['Latitude'][:] for f in geos])
lon = numpy.concatenate([f['All_Data']['ATMS-SDR-GEO_All']['Longitude'][:] for f in geos])

# build a map projection
m = Basemap(projection='stere',lon_0=-100.0,lat_0=90.,lat_ts=40.0,\
            llcrnrlat=12.0,urcrnrlat=58.0,\
            llcrnrlon=-120.0,urcrnrlon=-35.0,\
            rsphere=6371200.,resolution='l',area_thresh=10000)
m.drawcoastlines(); m.drawcountries(); m.drawstates()

# project a contour map with 100 levels for Channel 16
x,y = m(lon, lat)
m.contourf(x, y, atms_bt[:,:,15].squeeze(), 100)
colorbar()
title('ATMS BT Chnl 16')
show()
```

VIIRS SDR GRANULE QUICKLOOKS



In contrast to the simple ATMS contour plot projection of a low-resolution dataset, VIIRS requires swath-to-gridded-image resampling and a bow-tie interpolation. However, end-to-end generation of a 2048x2048 image from a 3200x7000 input swath is still completed in seconds using pyresample.

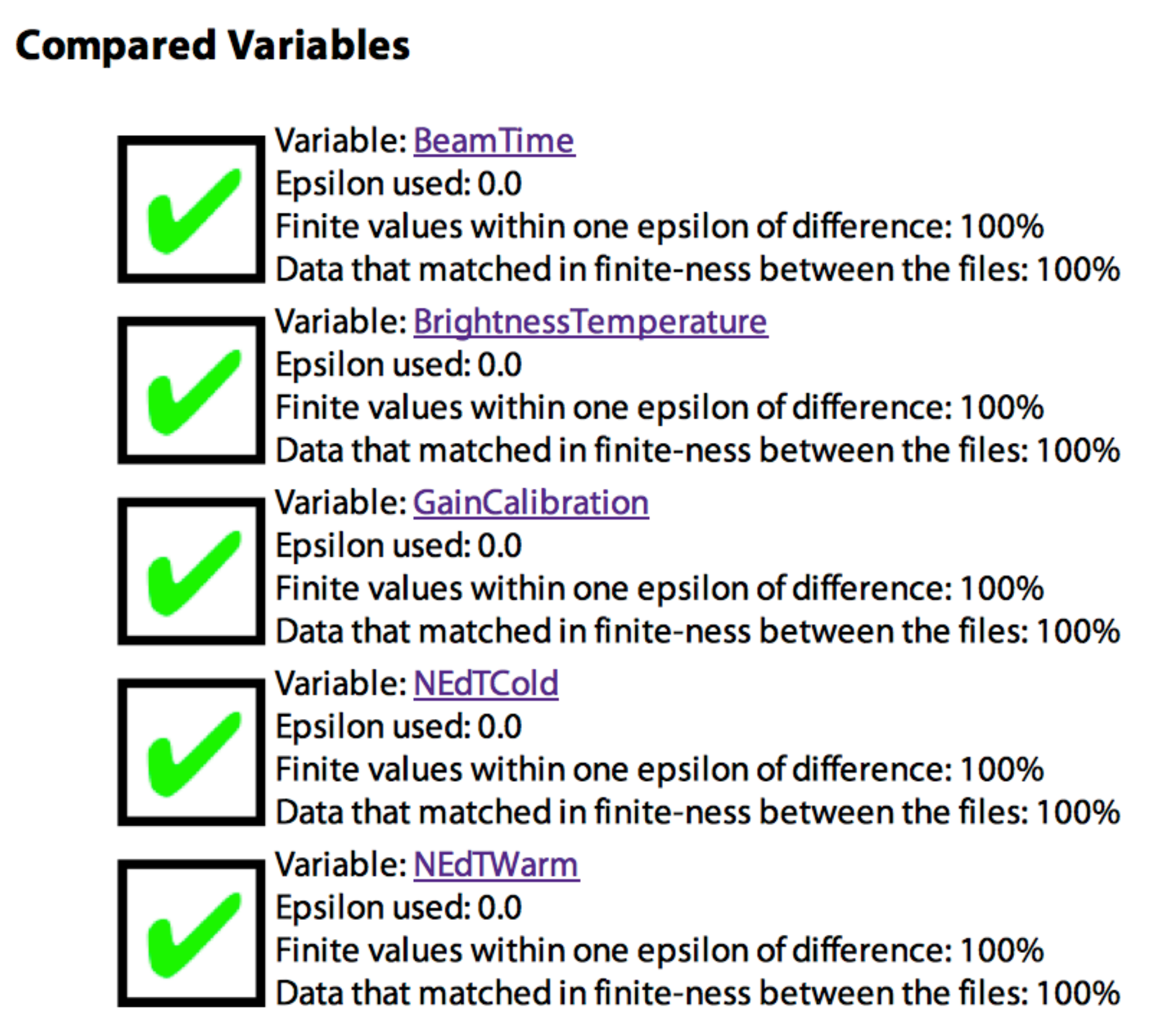
BRIDGING VIIRS to AWIPS



Using the same *ll2cr* and *fornav* mapx-based open reprojection software as is used to export MODIS to AWIPS, scripts readily automate conversion of VIIRS direct broadcast swaths into AWIPS-consumable form.

VERIFYING CSPP-ADL vs IDPS

Test products for CSPP SDR binaries are statistically verified against corresponding IDPS operations granules. This is achieved in part using 'glance', a generalized large dataset comparator and report generator. This tool can operate on HDF4/5 and NetCDF products among other formats, and can be configured to generate a variety of reports and score for a variety of satisfaction criteria.



Compared Variables

✓ Variable: BeamTime
Epsilon used: 0.0
Finite values within one epsilon of difference: 100%
Data that matched in finite-ness between the files: 100%

✓ Variable: BrightnessTemperature
Epsilon used: 0.0
Finite values within one epsilon of difference: 100%
Data that matched in finite-ness between the files: 100%

✓ Variable: GainCalibration
Epsilon used: 0.0
Finite values within one epsilon of difference: 100%
Data that matched in finite-ness between the files: 100%

✓ Variable: NEdTCold
Epsilon used: 0.0
Finite values within one epsilon of difference: 100%
Data that matched in finite-ness between the files: 100%

✓ Variable: NEdTWarm
Epsilon used: 0.0
Finite values within one epsilon of difference: 100%
Data that matched in finite-ness between the files: 100%