The use of ZFP lossy compression in tornado-resolving thunderstorm simulations

Leigh Orf, Cooperative Institute for Meteorological Satellite Studies, UW-Madison

AGU Fall Meeting, New Orleans, 11 Dec 2017. Invited poster IN11B-0039

Overview

Numerical models run on supercomputers can create tremendous amounts of data. In our work, we are simulating tornado-producing supercell thunderstorms at ultra-high resolution on the Blue Waters supercomputer. In a typical simulation, we save data every 1 model second for 90 minutes of cloud time in order to capture rapidly varying winds in the vicinity of the tornado. This results in thousands of saved times, with each saved time containing a dozen or more 3D floating point model variables. **A single simulation can result in several petabytes of floating point output if data is written uncompressed across the full model domain**.

The main motivation for this work is to save 3D floating point data at extremely high temporal resolution such that I/O does not dominate model run time. We achieve this by:

1. Creating a file system (LOFS) for massively parallel MPI simulations in which files are spread across many directories, each file containing a continuous portion of model domain
2. Buffering data to memory during most writes in order to reduce latency associated with doing I/O to Lustre frequently
3. Only saving data in a subset of the full model domain (e.g., centered on the storm's updraft where the tornado forms)
4. Compressing all 3D floating point arrays with ZFP lossy floating point compression

LOFS

LOFS is a file system containing HDF5 files spread across many directories. LOFS has been ported to the CM1 cloud model but could be applied to any 3D fluid code that utilizes MPI with a 2D domain decomposition on a shared file system such as Lustre.

LOFS features:

1. Files are written on a per-node basis, with only intranode communication required to assemble a 3D chunk of model domain
2. Each HDF5 file can contain as many times as can be buffered to memory without memory exhaustion
3. Files and directories all require a strict naming convention such that metadata can be extracted by the name of the directory/file
4. Redundant metadata stored within each HDF5 contains enough information to reconstruct the full filesystem structure
5. Only serial HDF5 is used for LOFS; this allows us to use compression on a per-file basis
6. An API for reading data which takes as input i,j,k range (with respect to the full model domain) requested of any model variable

LOFS directory structure

In the below tree listing, each directory contains up to 1000 HDF files, with each HDF5 file containing 50 times per file and a dozen 3D variables per time. Files are concurrently written across several directories at each write cycle.

```
history       (top level directory)
 ├─2D          (horiz. slices saved with pHDF5)
 └─3D          (3D floating point data)
    ├─tornado.03000  (50 s of data in 1 s chunks)
    ├─tornado.03050  (50 s of data in 1 s chunks)
    ├─...
    ├─tornado.07150  (50 s of data in 1 s chunks)
    └─tornado.07200  (50 s of data in 1 s chunks)
       ├─0000000  (contains up to 1000 files)
       ├─0001000  (contains up to 1000 files)
       ├─...
       ├─0008000  (contains up to 1000 files)
       └─0009000  (contains up to 1000 files)
          ├─tornado.07200.0009000.cm1hdf5
          ├─tornado.07200.0009001.cm1hdf5
          ├─tornado.07200.0009002.cm1hdf5
          ├─tornado.07200.0009003.cm1hdf5
          ├─tornado.07200.0009004.cm1hdf5
          ├─...
          └─tornado.07200.0009999.cm1hdf5
```

Rank reordering: 2x2 example

CM1 utilizes a 2D domain decomposition, where each MPI rank spans a horizontal patch that contains the full vertical extent of the domain. Each file contains 3D variables that span a continuous 3D volume of the model domain. By reordering ranks, one file per node can be saved that traverses the node only, with only intranode communication required to grow the files in memory.

45	46	47	48	61	62	63	64
41	42	43	44	57	58	59	60
37	38	39	40	53	54	55	56
33	34	35	36	49	50	51	52
13	14	15	16	29	30	31	32
9	10	11	12	25	26	27	28
5	6	7	8	21	22	23	24
1	2	3	4	17	18	19	20

Before: Ranks are ordered in "SMP" mode where each subsequent node is populated with ranks

57	58	59	60	61	62	63	64
49	50	51	52	53	54	55	56
41	42	43	44	45	46	47	48
33	34	35	36	37	38	39	40
25	26	27	28	29	30	31	32
17	18	19	20	21	22	23	24
9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8

After: Nodes/ranks are mapped to the physical model domain. One rank on each node collects, assembles, compresses, and buffers to memory.

ZFP lossy compression

ZFP is an open source C/C++ library for compressed floating-point arrays that support very high throughput read and write random access. A zfp filter has been written for the HDF5 data format and has been adopted in this work. **In this work, all compression is done utilizing "accuracy mode" where the amount of accuracy is specified per variable.** This is very intuitive, allowing compression ratios to vary from file to file but retaining the specified accuracy in all files.

HDF5 internal structure

All 3D floating point data has been compressed with ZFP in accuracy mode, with each variable given its own accuracy parameter.

```
/
├─00000  (time index - 50 per file)
├─00001  These 50 times were buffered
├─...    to memory before written to disk.
└─00049
   ├─2D
   └─3D  (each 3D array ZFP compressed)
      ├─dbz(nk,nj,ni)
      ├─pressure(nk,nj,ni)
      ├─temperature(nk,nj,ni)
      ├─u_wind(nk,nj,ni)
      ├─v_wind(nk,nj,ni)
      ├─w_wind(nk,nj,ni)
      └─...
```

Activate ZFP in CM1/HDF5

```
module module_orfio
use hdf5
use h5zzfp_props_f
...
ierr = H5Z_zfp_initialize()
...
call h5pcreate_f(H5P_DATASET_CREATE_F,chunk_id,ierr)
call h5pset_chunk_f(chunk_id,rank,chunkdims,ierr)
ierr = h5pset_zfp_accuracy(chunk_id,accuracy)
call h5dcreate_f(f_id,trim(varname),H5T_NATIVE_REAL,
     dspace_id, dset_id,ierr,dcpl_id=chunk_id)
call h5dwrite_f(...)
...
```

LOFS middleware / tools

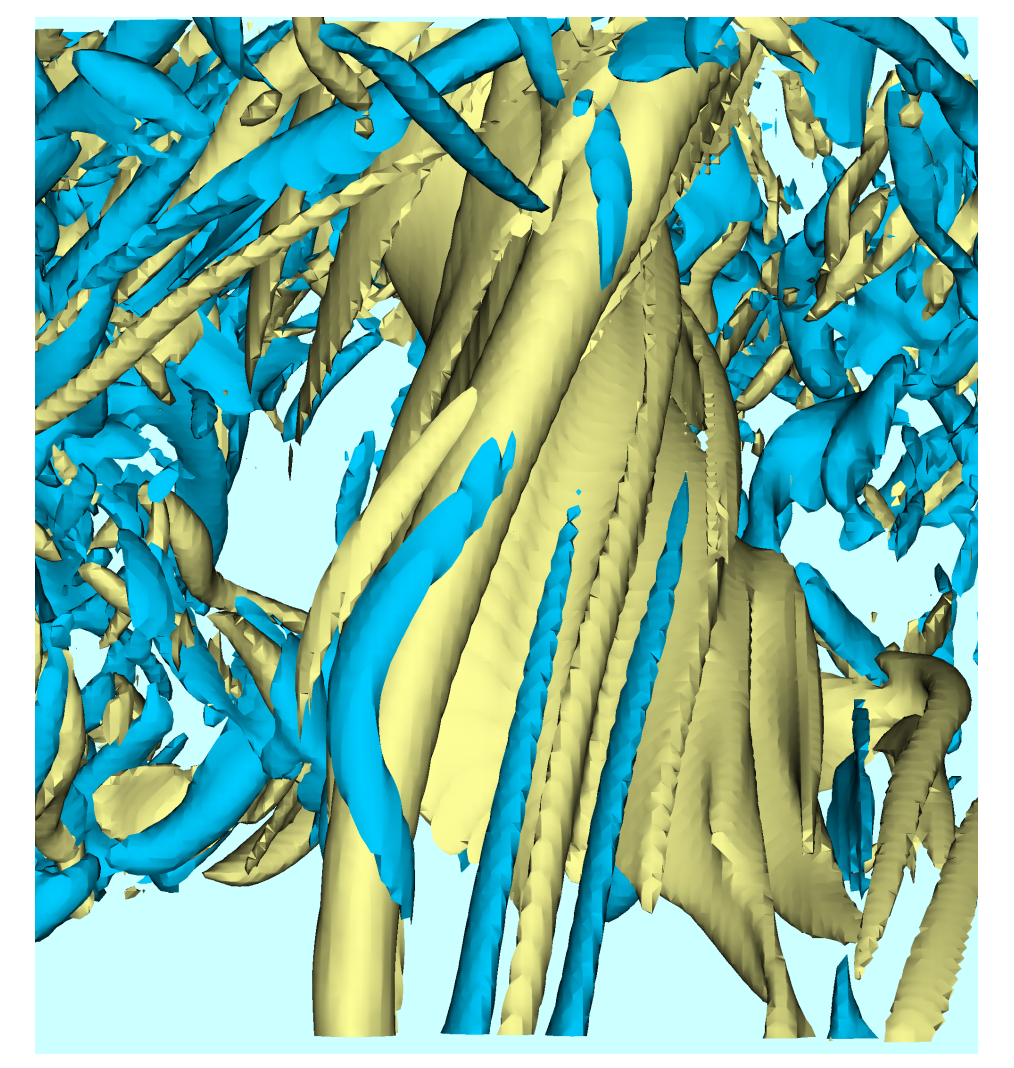
hdf2nc: Convert LOFS data to a single netCDF file

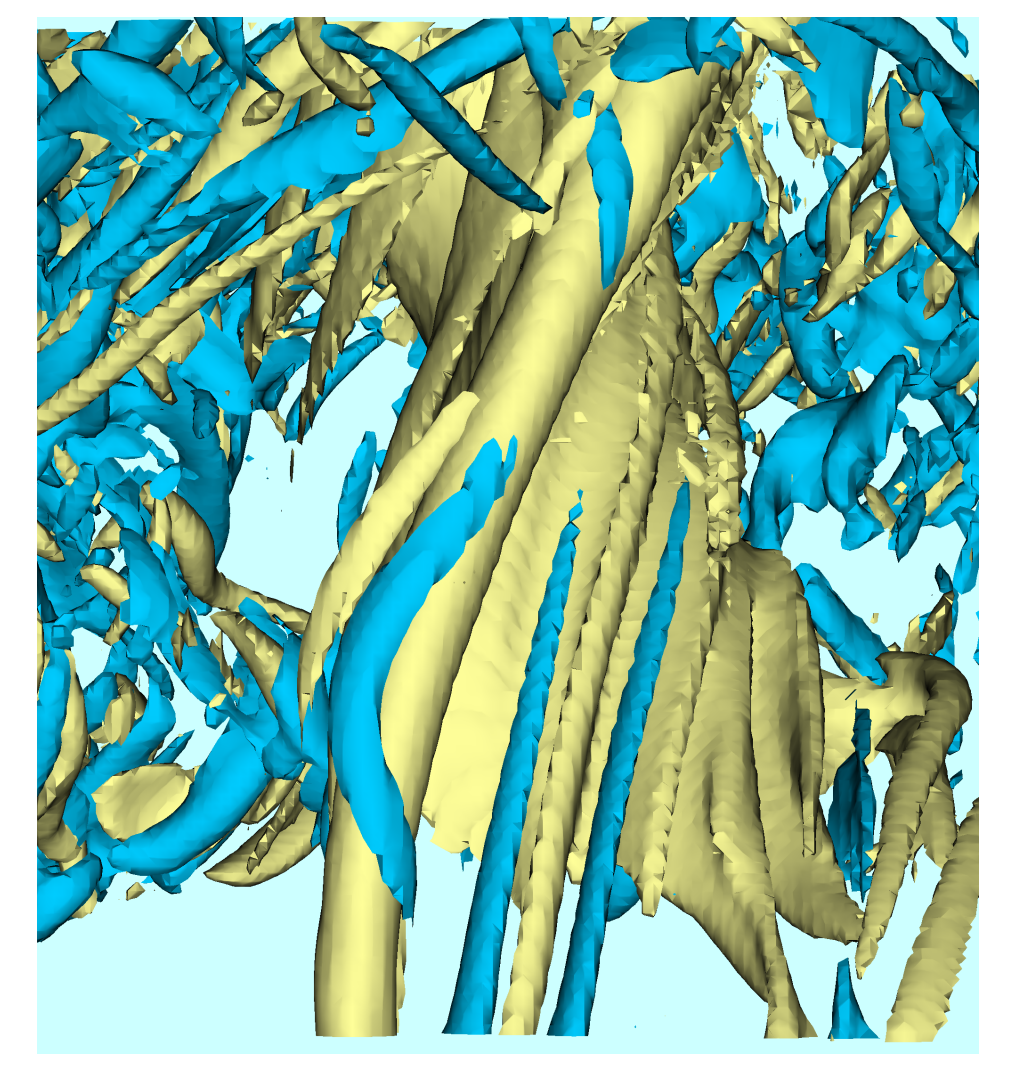
```
Usage:  hdf2nc -histpath=[histpath] -ncbase=[ncbase]
-x0=[X0] -y0=[Y0] -x1=[X1] -y1=[Y1] -z0=[Z0] -z1=[Z1]
-time=[time] [varname1 ...  varnameN]
```

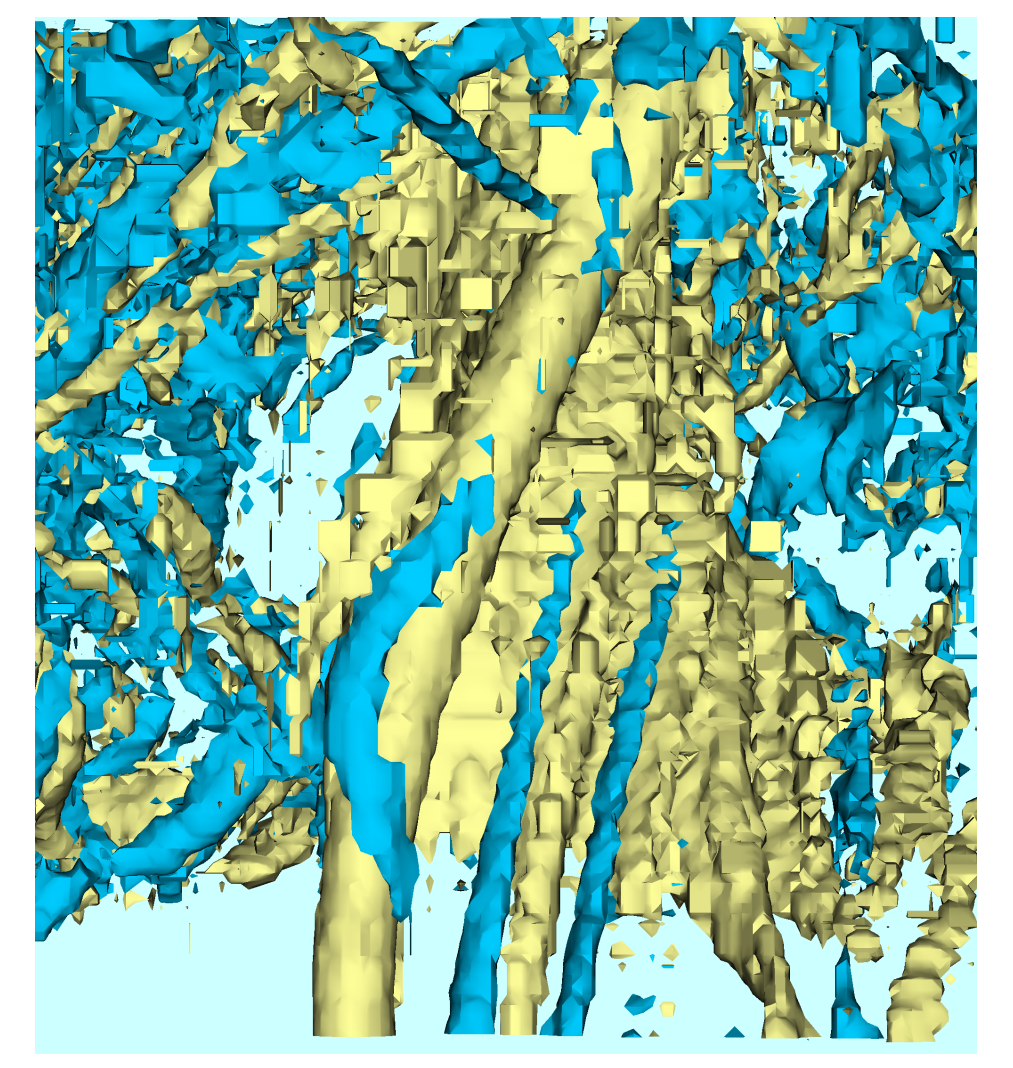
makevisit: Create a file to be read by VisIt using LOFS plugin code, which reads the LOFS file system natively (serially, in parallel)

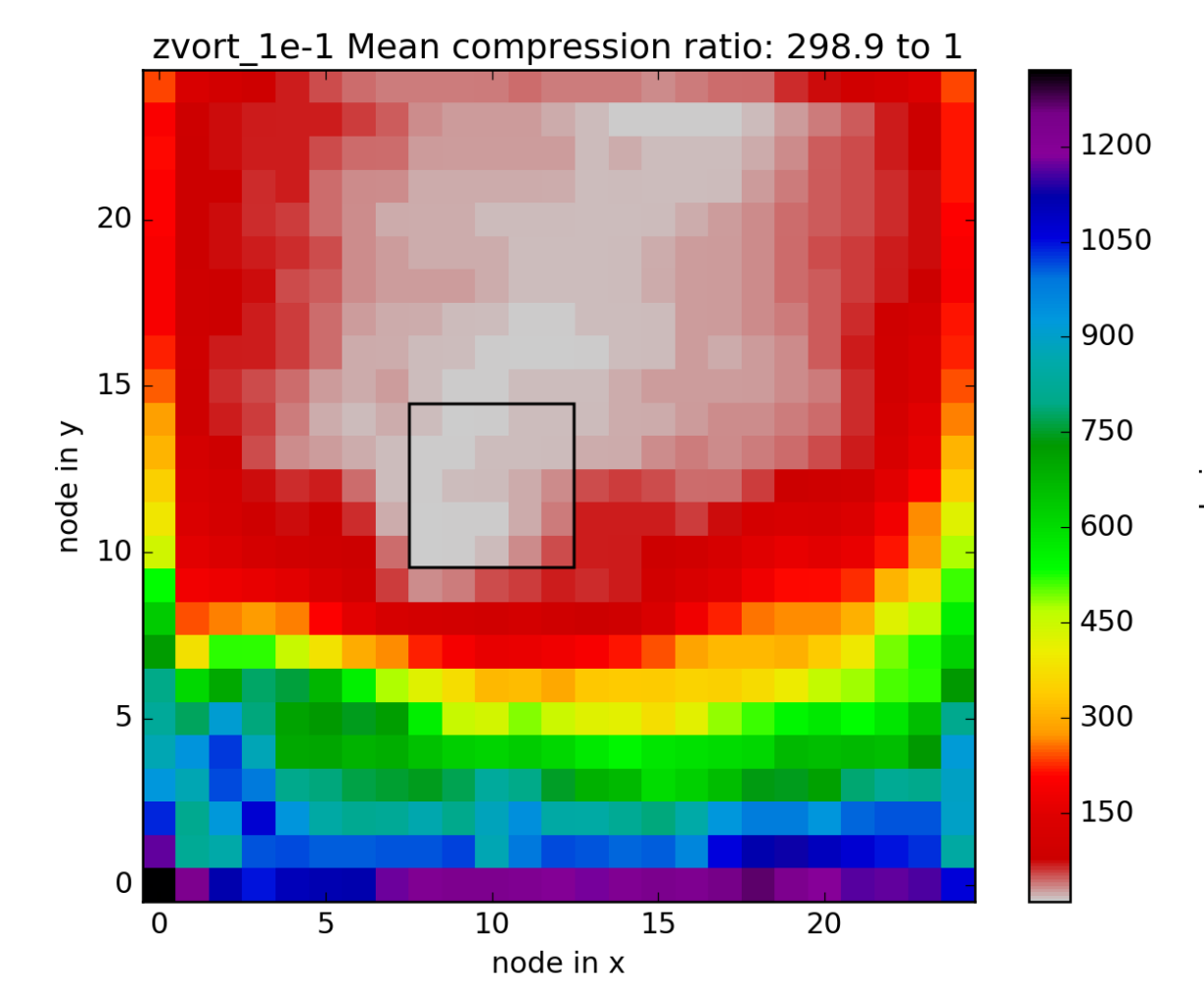
```
Usage:  makevisit -histpath=[path] -base=[base]
-x0=[x0] -y0=[y0] -x1=[x1] -y1=[y1] -z0=[z0] -z1=[z1]
```

Compressed vertical vorticity

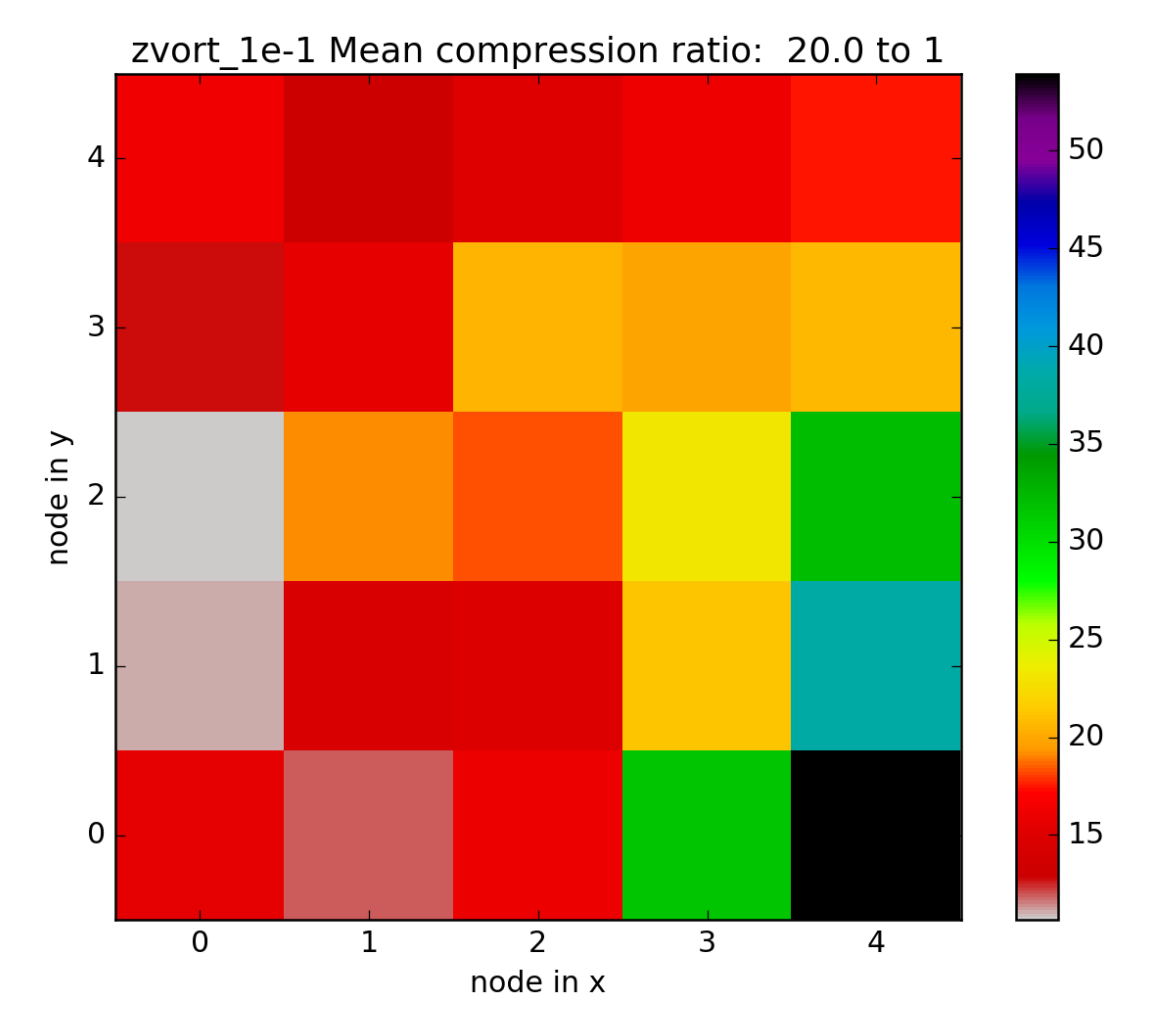

Vertical vorticity isosurfaces of $+/-0.1\,\mathrm{s}^{-1}$ (uncompressed)


Accuracy $= 0.1\,\mathrm{s}^{-1}$


Accuracy $= 1.0\,\mathrm{s}^{-1}$

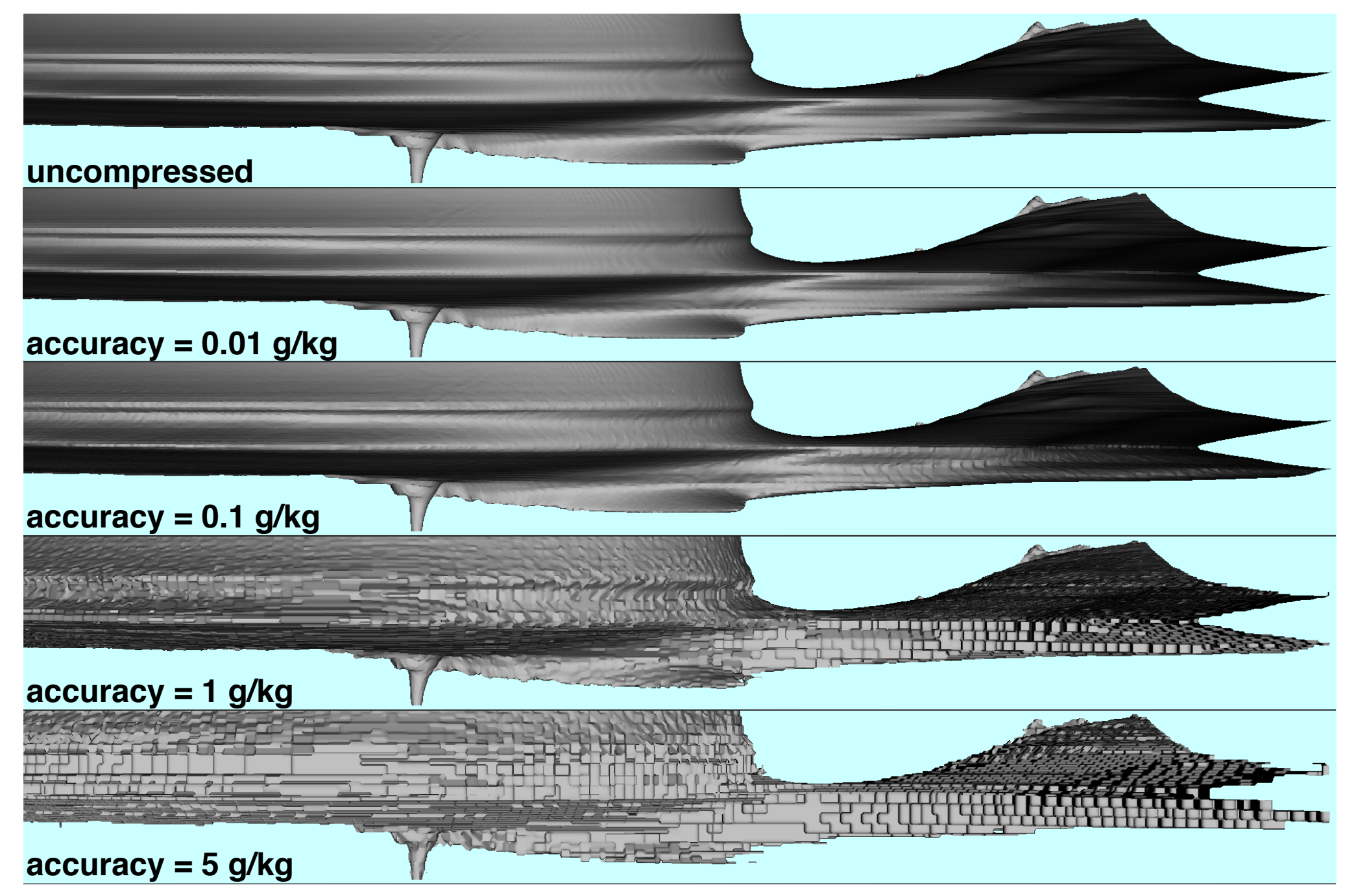


Full domain compression ratios of vertical vorticity for each file (accuracy $= 0.1\,\mathrm{s}^{-1}$)

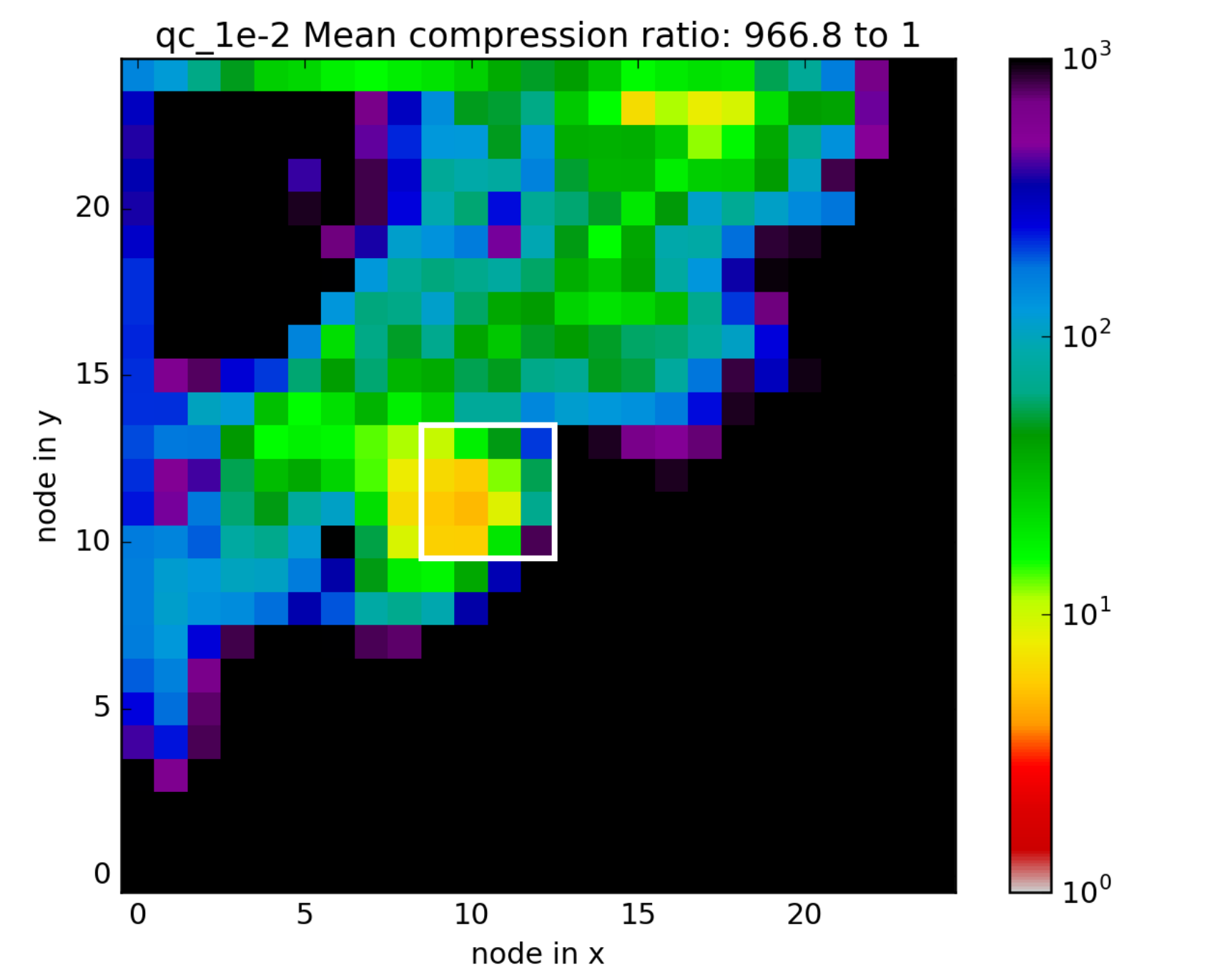


Inset region in image to left

Compressed cloud mixing ratios

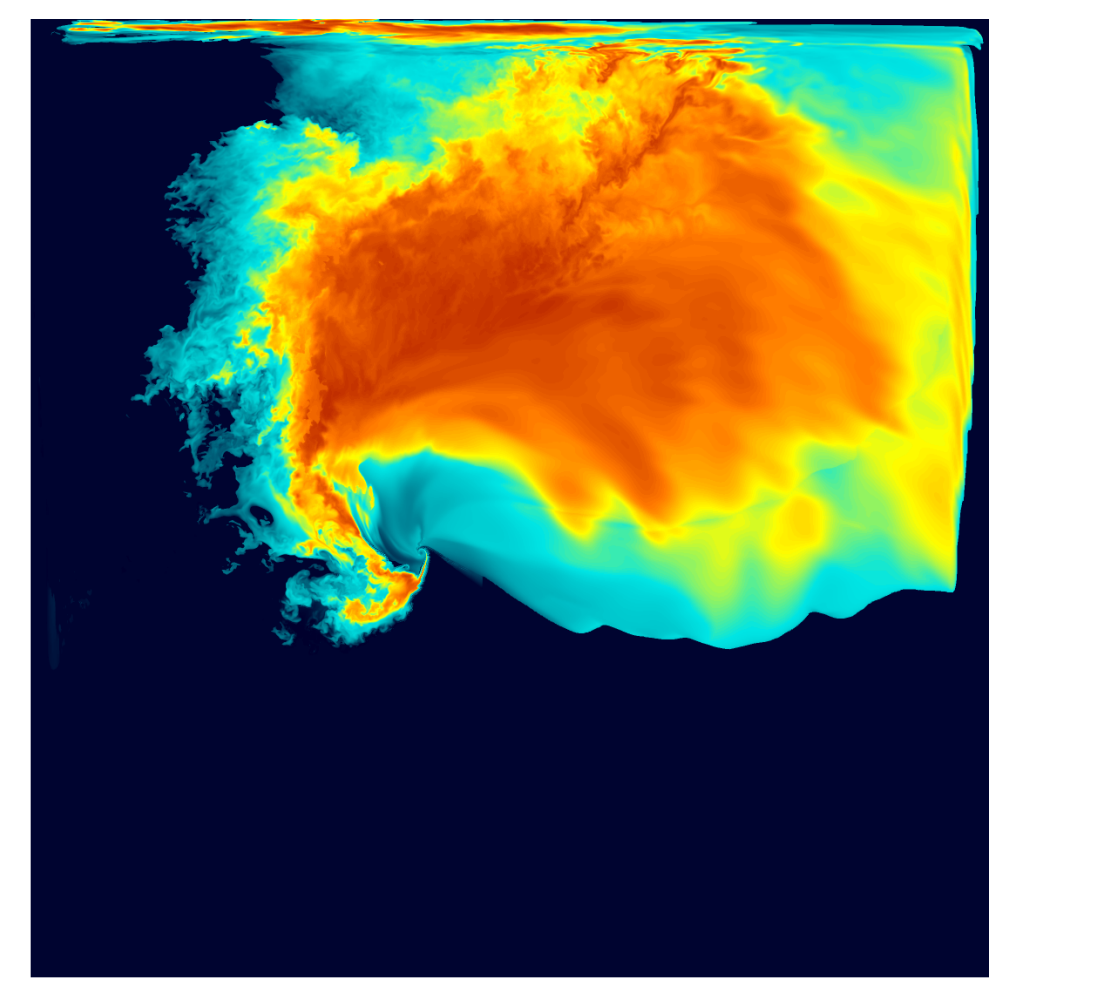


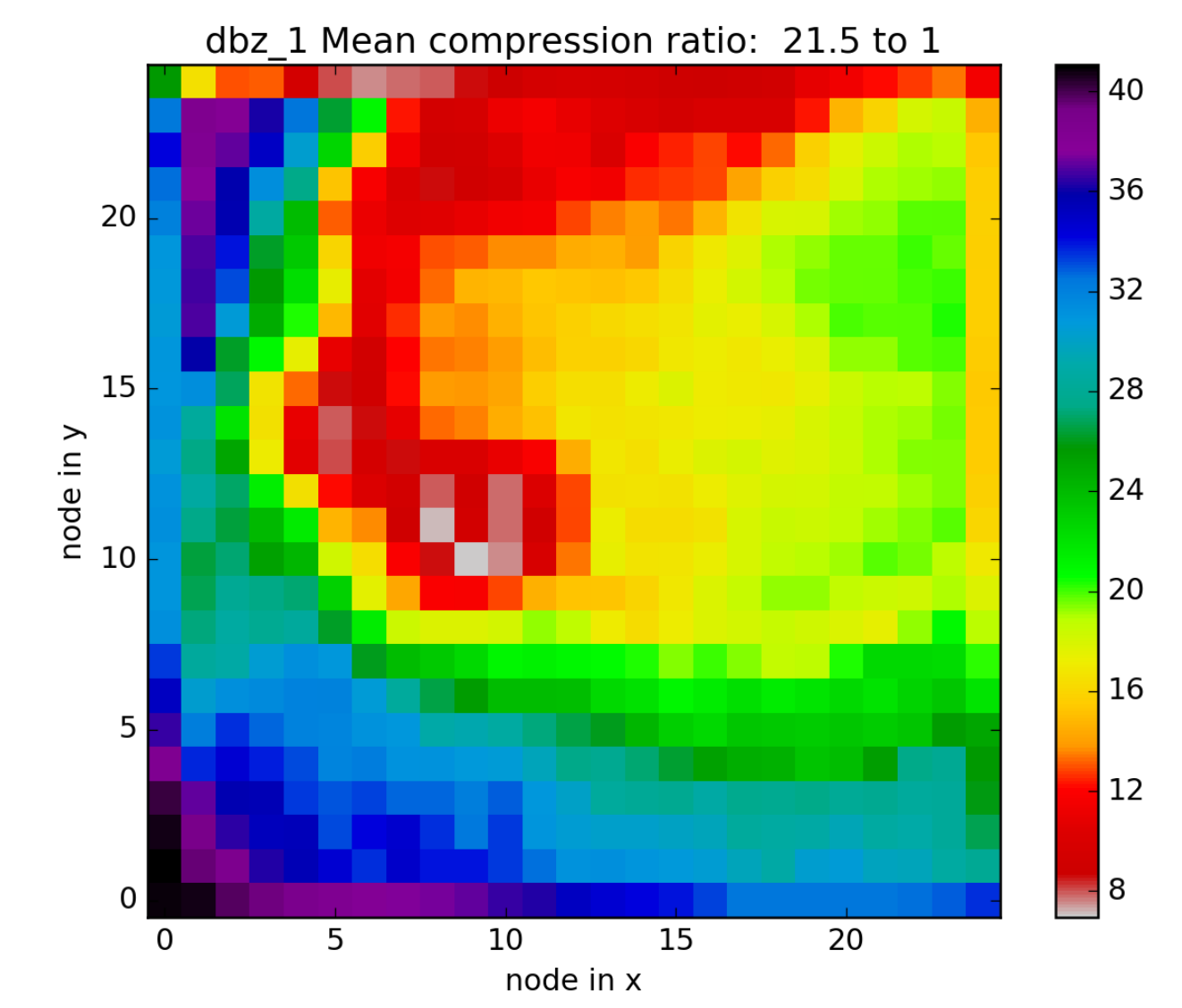
0.2 g/kg Isosurface of cloud water mixing ratio for different accuracy choices. Domain-wide values range from 0 to 15 g/kg.



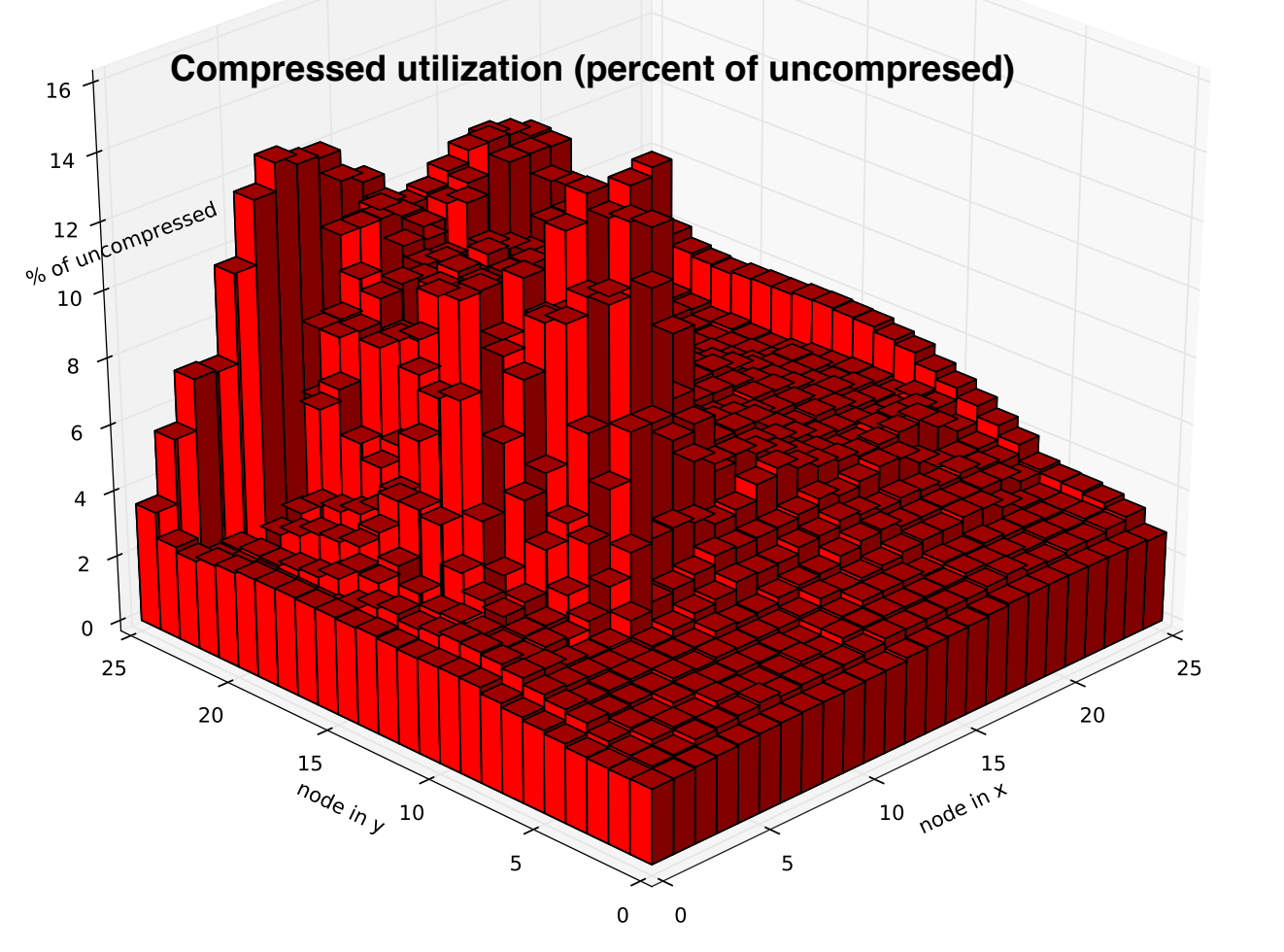
Domain-wide compression ratios per file for accuracy = 0.01 g/kg. Box roughly indicates region visible in image to the left.

Compressed simulated radar reflectivity


Uncompressed surface radar reflectivity across entire domain (ranging from -40 to 60 dBZ)



Domain-wide compression ratios per file of reflectivity (accuracy = 1 dBZ)



Domain-wide compressed utilization as a percentage of uncompressed utilization for each file (accuracy = 1 dBZ)

Results

- ZFP compression is successfully applied to floating point data in HDF5 files created in massively parallel thunderstorm simulations
- Variables in each HDF5 file are continuous chunks of the model subdomain, and by applying the same accuracy parameter to a given variable in each file, compression ratios vary across files, with the "least interesting" portions of the domain compressing the best
- For visualization / plotting, you can get away with an accuracy parameter that results in domain-wide compression ratios ranging from 20:1 to 1000:1, depending on the variable
- Very small accuracy parameters result in compression ratios significantly better than lossless (with only a slight loss in accuracy)
- A practical result of the application of ZFP to floating point data is the ability to save on the order of 20 to 100 times more data for a given simulation as compared to lossless compression

Acknowledgments

This work was supported by NSF grant OAC-1614973 and the Space Science and Engineering Center at UW-Madison. This research is part of the Blue Waters sustained-petascale computing project, which is supported by the National Science Foundation (award number ACI 1238993) and the state of Illinois. Blue Waters is a joint effort of the University of Illinois at Urbana-Champaign and its National Center for Supercomputing Applications.

References

Lindstrom, P., 2014: Fixed-Rate Compressed Floating-Point Arrays. IEEE Trans. Vis. Comput. Graph., 20, 2674–2683.

Orf, L., R. Wilhelmson, B. Lee, C. Finley, and A. Houston, 2017: Evolution of a Long-Track Violent Tornado within a Simulated Supercell. Bull. Am. Meteorol. Soc., 98, 45–68.

Visit http://orf.media for more