A simple file system for massively parallel cloud models

Leigh Orf

Cooperative Institute for Meteorological Satellite Studies/SSEC
University of Wisconsin - Madison

2020 Virtual HDF5 Users Group

15 October 2020

Outline

- Cloud modeling and HPC
- CM1 model
- LOFS structure
- A brief word on HDF5_CORE
- ZFP compression
- LOFS API/hdf2nc conversion utility
- Some research results - why I bothered with all of this!
- Final thoughts

Cloud modeling and HPC

- Weather forecasting, whether on global or thunderstorm scale, is computationally expensive to do right
- Typical atmospheric models use explicit finite difference techniques to solve a system of partial differential equations to determine the next 3D model state in time
- Data must be saved periodically in order to analyze and process
- As models have gotten more sophisticated and run at higher resolution, I/O has become a serious bottleneck
- My own research on tornadoes produces a tremendous amount of model output

CM1 model

- Fortran95, 32 bit float computational arrays, MPI, OpenMP
- Currently, the most widely used cloud model for doing basic research involving highly idealized simulations of clouds
- Contains “fast” I/O methods that result in many millions files of a fragmented domain in a single directory, or “slow” I/O methods with more friendly output
- In order to conduct and analyze large simulations of tornado producing thunderstorms on NSF’s Blue Waters, I needed I/O that was very fast (low latency, high throughput), compressed significantly, and easy to analyze/read back. This resulted in LOFS.

LOFS directory/file structure

```
history    (top level directory)
└── 3D     (3D floating point data)
    ├── tornado.03000.000000 (50 s of data in 1 s chunks)
    ├── tornado.03050.000000 (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.07200.000000.0009000.cm1hdf5
            ├── tornado.07200.000000.0009001.cm1hdf5
            └── ...
```

LOFS files

- One file per shared-memory node is written
- I created a new MPI communicator for CM1 that maps ranks to the hardware such that each node contains a continuous volume of data (not scrambled up)
- During each write cycle, one rank per node collects data (using `MPI_Gather`) and does the compressing and buffering to memory to grow the HDF5 file
- This operation is blocking (there is no core dedicated to I/O only)

Example hardware configuration with 4 nodes, 16 cores per node

ranks before reordering

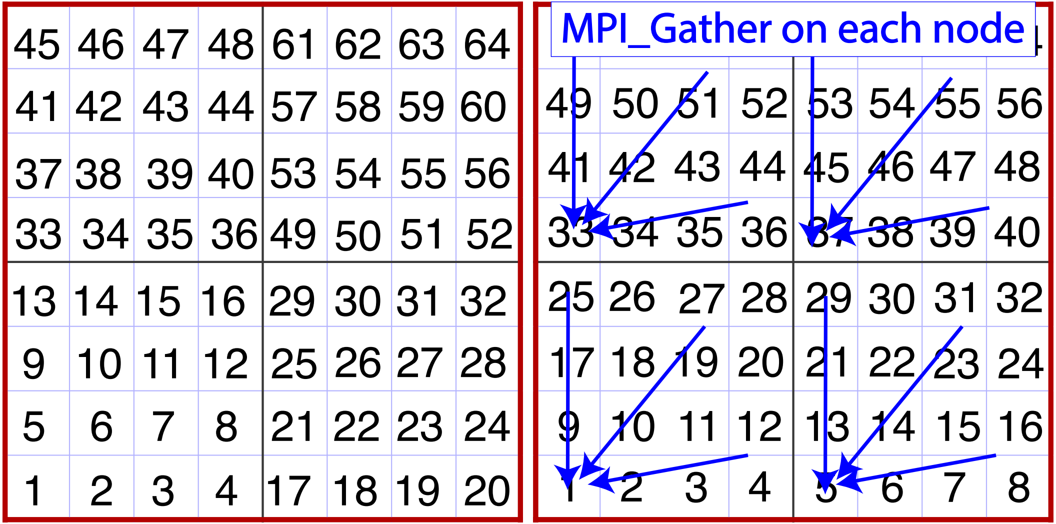
45	46	47	48	61	62	63	64
41	42	43	44	57	58	59	60
37	38	39	40	53	54	55	56
33	34	35	36	49	50	51	52
13	14	15	16	29	30	31	32
9	10	11	12	25	26	27	28
5	6	7	8	21	22	23	24
1	2	3	4	17	18	19	20

ranks after reordering

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

Example hardware configuration with 4 nodes, 16 cores per node

ranks before reordering | **ranks after reordering**



Memory-buffered I/O

- Early on I settled on HDF5, but it took years before I figured out how to exploit some its useful features
- I tried pHDF5 in various configurations, had serious inexplicable performance issues
- Once I discovered the core (HDF5_CORE) driver I devised a driver around the fact that buffering data to memory is a lot faster than writing to disk
- This works because the CM1 model when run at large scale only takes up a few % of available memory on a shared memory node

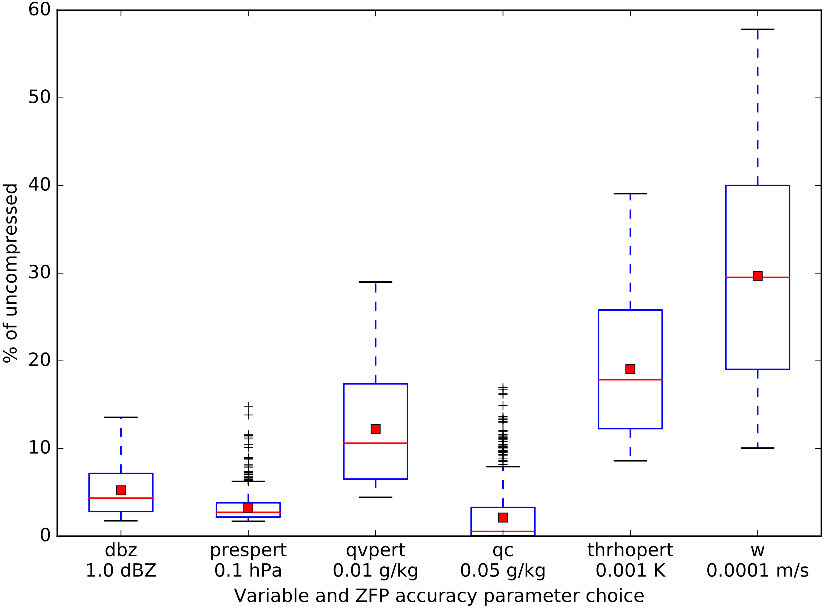
A brief word on HDF5_CORE

- Seriously undersold! From the HDF5 website:
  H5FD_CORE: *This driver performs I/O directly to memory and can be used to create small temporary files that never exist on permanent storage.*
- My files are friggin' huge, and for me it's all about the permanent storage!
- Using the core driver allowed me to save 50-100 time steps in a single file, all buffered to memory, and later flushed to disk, one file per node.

ZFP compression

- ZFP can be installed as a HDF5 plugin
- ZFP is a **lossy** compression algorithm (we should all be using these more!)
- Accuracy for any field can be specified upfront. Example: I only need temperature data that is accurate within 0.1 °C: `h5pset_zfp_accuracy(chunk_id,0.1);`
- Compression ratios are strongly a function of the magnitude of the accuracy parameter relative to the magnitude of the underlying data

Example ZFP performance for 10-m tornado simulation



HDF5 internal structure (some h5ls -r output)

```
/basestate/u0            Dataset {266}
/basestate/v0            Dataset {266}
/grid                    Group
/grid/MCMnx              Dataset {1}
/grid/MCMny              Dataset {1}
/grid/cores_per_MCM      Dataset {1}
/grid/corex              Dataset {1}
/grid/corey              Dataset {1}
/grid/myMCMid            Dataset {1}
/grid/myi                Dataset {1}
/grid/myj                Dataset {1}
/grid/ni                 Dataset {1}
/grid/nj                 Dataset {1}
/grid/nodex              Dataset {1}
/grid/nodey              Dataset {1}
/grid/nx                 Dataset {1}
/grid/ny                 Dataset {1}
/grid/nz                 Dataset {1}
/grid/x0                 Dataset {1}
/grid/x1                 Dataset {1}
/grid/y0                 Dataset {1}
/grid/y1                 Dataset {1}
/mesh                    Group
/mesh/dx                 Dataset {1}
/mesh/dy                 Dataset {1}
/mesh/dz                 Dataset {1}
/mesh/umove              Dataset {1}
/mesh/vmove              Dataset {1}
/mesh/xf                 Dataset {50}
/mesh/xffull             Dataset {1601}
/mesh/xh                 Dataset {50}
/mesh/xhfull             Dataset {1600}
/mesh/yf                 Dataset {399}
/mesh/yffull             Dataset {1597}
/mesh/yh                 Dataset {399}
```

```
/00029/2D/swath/zeta_min_sfc Dataset {399, 50}
/00029/2D/swath/zeta_min_sfc_move Dataset {399, 50}
/00029/3D                Group
/00029/3D/dbz            Dataset {266, 399, 50}
/00029/3D/khh            Dataset {266, 399, 50}
/00029/3D/khv            Dataset {266, 399, 50}
/00029/3D/kmh            Dataset {266, 399, 50}
/00029/3D/kmv            Dataset {266, 399, 50}
/00029/3D/ncg            Dataset {266, 399, 50}
/00029/3D/nci            Dataset {266, 399, 50}
/00029/3D/ncr            Dataset {266, 399, 50}
/00029/3D/ncs            Dataset {266, 399, 50}
/00029/3D/prespert       Dataset {266, 399, 50}
/00029/3D/qc             Dataset {266, 399, 50}
/00029/3D/qg             Dataset {266, 399, 50}
/00029/3D/qi             Dataset {266, 399, 50}
/00029/3D/qr             Dataset {266, 399, 50}
/00029/3D/qs             Dataset {266, 399, 50}
/00029/3D/qvpert         Dataset {266, 399, 50}
/00029/3D/rhopert        Dataset {266, 399, 50}
/00029/3D/thpert         Dataset {266, 399, 50}
/00029/3D/thrhopert      Dataset {266, 399, 50}
/00029/3D/tke_sg         Dataset {266, 399, 50}
/00029/3D/u              Dataset {266, 399, 50}
/00029/3D/v              Dataset {266, 399, 50}
/00029/3D/w              Dataset {266, 399, 50}
/00030                   Group
/00030/2D                Group
/00030/2D/static         Group
/00030/2D/static/snapshot_dbz_0500 Dataset {399, 50}
/00030/2D/static/snapshot_prespert_0500 Dataset {399, 50}
/00030/2D/static/snapshot_prespert_sfc Dataset {399, 50}
/00030/2D/static/snapshot_qc_1000 Dataset {399, 50}
/00030/2D/static/snapshot_qc_2000 Dataset {399, 50}
/00030/2D/static/snapshot_thrho_sfc Dataset {399, 50}
```

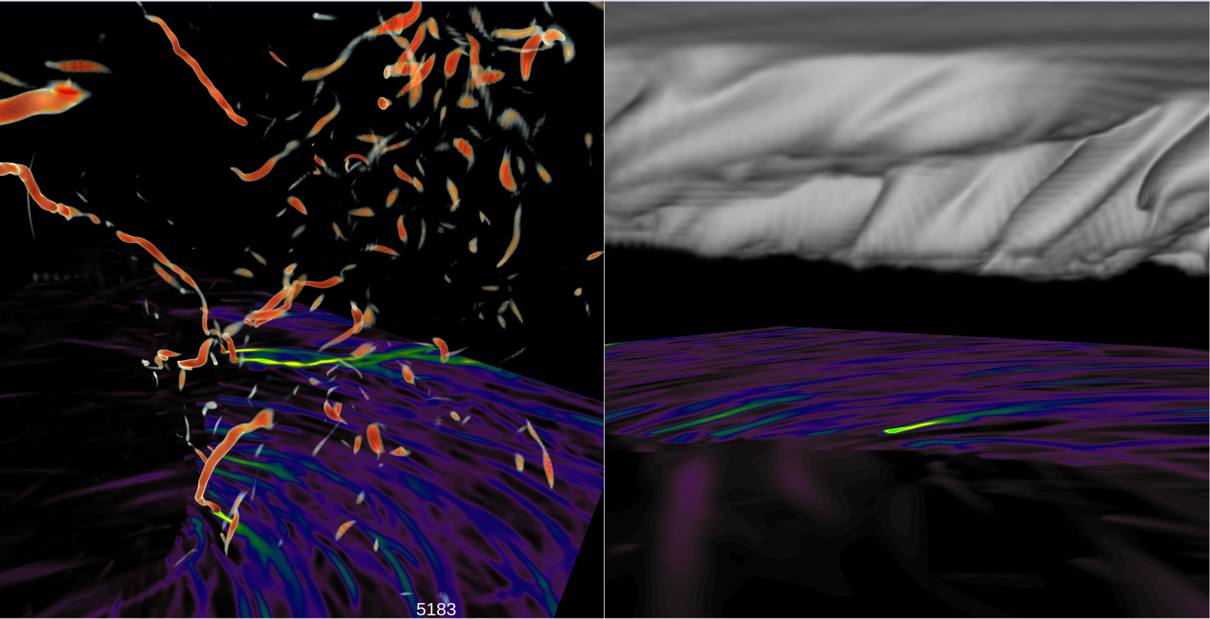
LOFS API/hdf2nc conversion utility

- Main routine “hdf2nc” (written in C) converts any subdomain of LOFS data to a single CF-compliant NetCDF file
- NetCDF is read by most major analysis/viz tools and highly used in weather/climate (developed at Unidata)
- Completely refactored code earlier this year, making is much easier to access data through a simple API
- LOFT, offline GPU trajectory analysis code (C++ and CUDA C++) developed by Kelton Halbert accesses the LOFS API directly

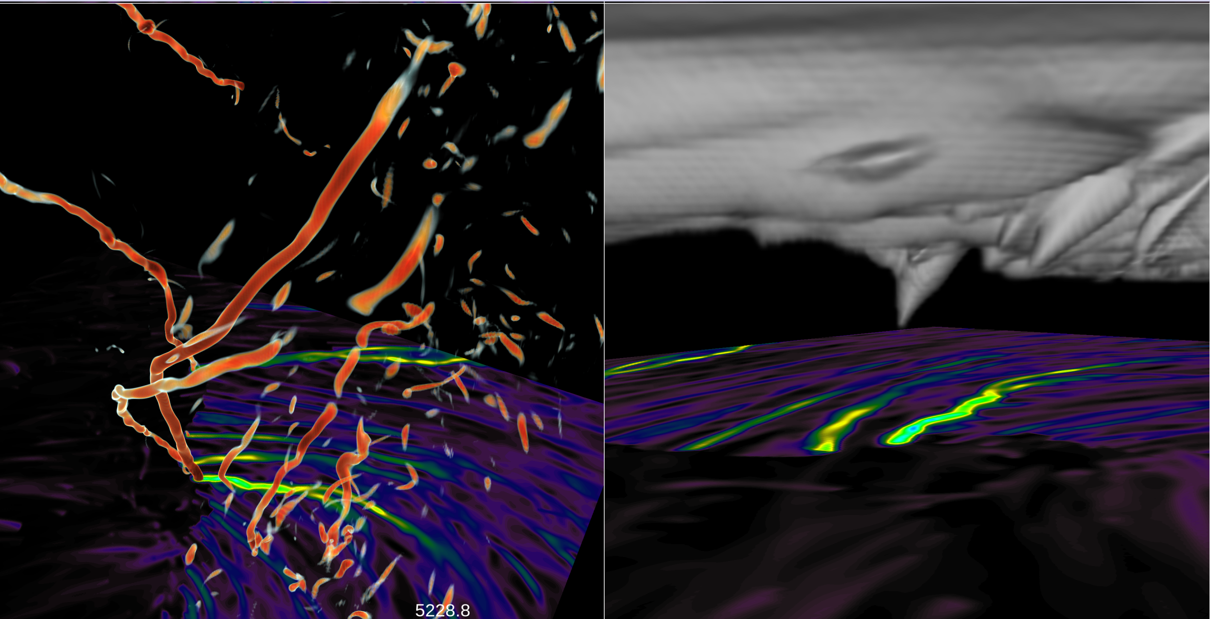
hdf2nc pseudocode

```
#include "../include/lofs-read.h"
#include "../include/lofs-dirstruct.h"
#include "../include/lofs-hdf2nc.h"
#include "../include/lofs-limits.h"
int main(int argc, char *argv[])
{
    init_structs(&cmd,&dm,&gd,&nc,&rh);
    parse_cmdline_hdf2nc(argc,argv,&cmd,&dm,&gd);
    get_num_time_dirs(&dm,cmd); get_sorted_time_dirs(&dm,cmd); get_num_node_dirs(&dm,cmd);
    get_sorted_node_dirs(&dm,cmd); get_saved_base(dm.timedir[0],dm.saved_base);
    get_all_available_times(&dm,&gd,cmd);
    hdf_file_id = H5Fopen (dm.firstfilename, H5F_ACC_RDONLY,H5P_DEFAULT)
    get_hdf_metadata(dm,&hm,&cmd,argv,&hdf_file_id);
    set_span(&gd,hm,cmd); allocate_1d_arrays(hm, gd, &msh, &snd);
    set_1d_arrays(hm,gd,&msh,&snd,&hdf_file_id);
    status = nc_create (nc.ncfilename, NC_CLOBBER|cmd.filetype, &nc.ncid);
    set_netcdf_attributes(&nc,gd,&cmd,&b,&hm,&hdf_file_id); status = nc_enddef (nc.ncid);
    nc_write_1d_data(nc,gd,msh,snd,cmd); set_readahead(&rh,nc,cmd);
    malloc_3D_arrays(&b,gd,rh,cmd); H5Z_zfp_initialize();
    if (cmd.do_swaths) do_the_swaths(hm,nc,dm,gd,cmd);
    do_readahead(&b,gd,rh,dm,hm,cmd); do_requested_variables(&b,nc,gd,msh,rh,dm,hm,cmd);
    status = nc_close(nc.ncid); H5Z_zfp_finalize(); free_3D_arrays(&b,gd,rh,cmd);
    if(cmd.gzip) compress_with_nccopy(nc,cmd);
}
```

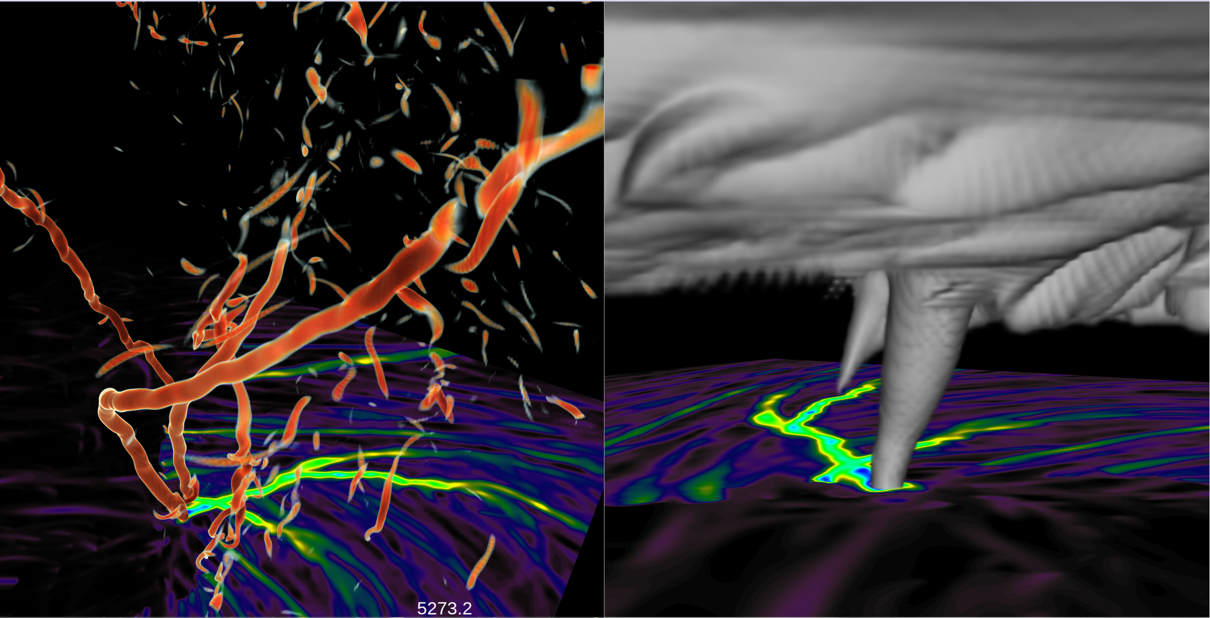
VAPOR3: Vorticity(L) and cloud(R) during tornadogenesis



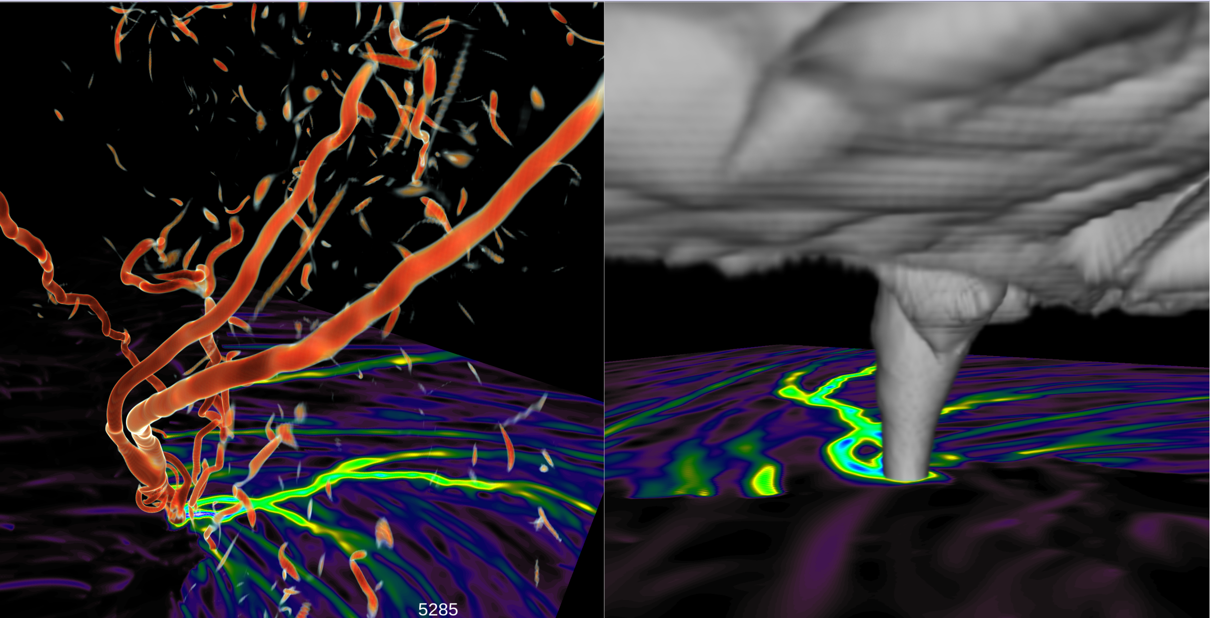
Vorticity(L) and cloud(R) during tornadogenesis



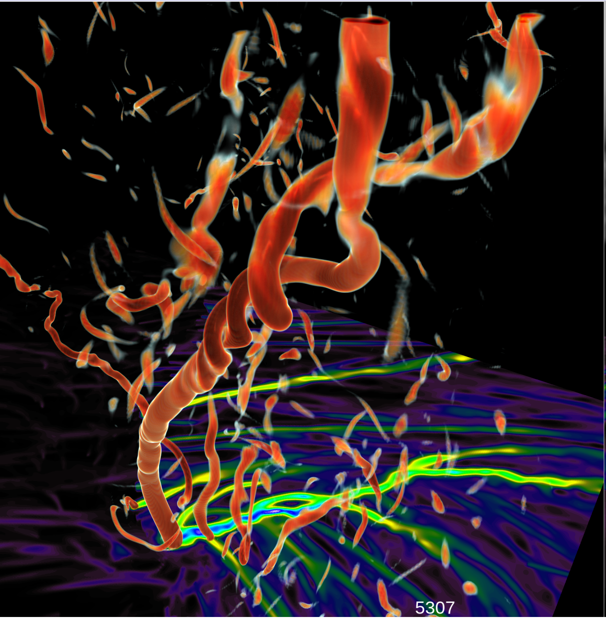
Vorticity(L) and cloud(R) during tornadogenesis

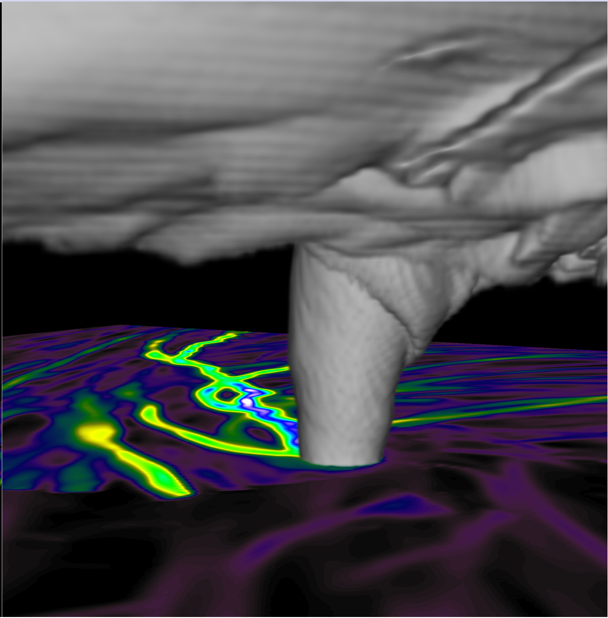


Vorticity(L) and cloud(R) during tornadogenesis

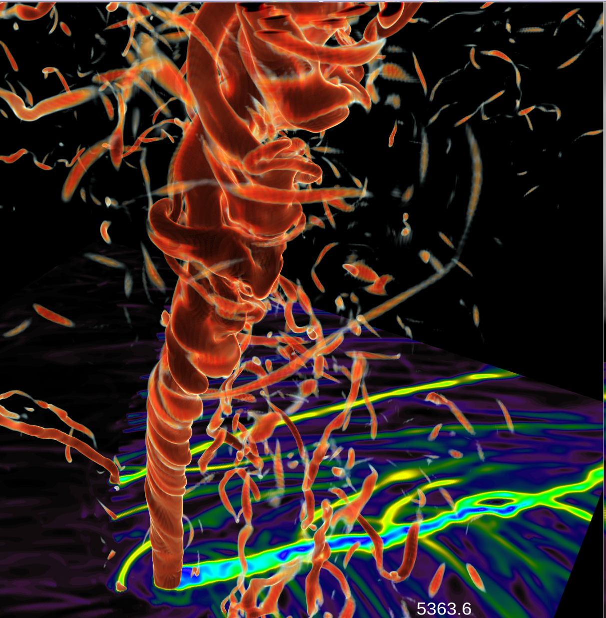


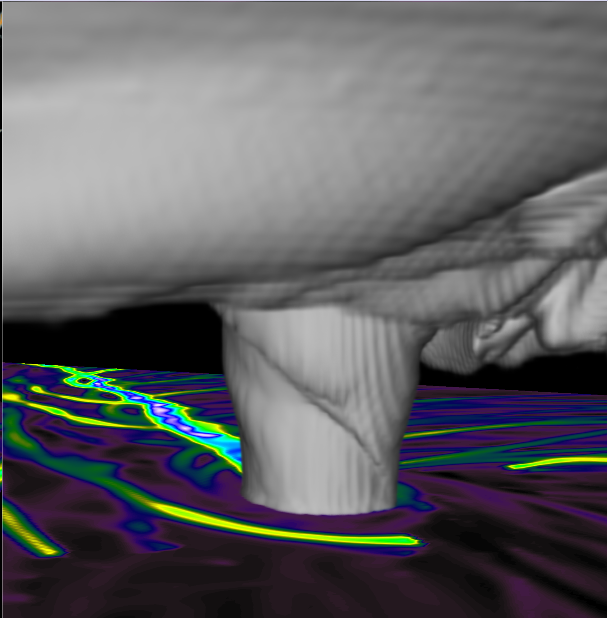
Vorticity(L) and cloud(R) during tornado maintenance





Vorticity(L) and cloud(R) during tornado maintenance





Final thoughts

- It’s all about performance and getting that data to disk as efficiently as possible, in a well organized fashion
- HDF5 functionality has enabled science that wouldn’t have been possible otherwise
- Lossy floating point compression (ZFP in our case) is key for performance and lets us save more data than would otherwise be possible
- Core I/O driver in a “serial I/O in parallel” configuration for large MPI runs performs extremely well
- LOFS file system comprised of HDF5 files could be used in any code that uses a similar 2D domain decomposition strategy

Acknowledgments/Links

- Simulations conducted on the NSF-sponsored Blue Waters supercomputer
- Leigh Orf is supported by NSF grants AGS-1832327, OAC-1663954
- Thanks to Gerd Heber for assistance (2016)

- Source code available at `github.com/leighorf`
- Some LOFS documentation: `lofs.io`
- Links to talks, movies, publications, etc: `orf.media`

Reference: Orf, L., 2019: A Violently Tornadic Supercell Thunderstorm Simulation Spanning a Quarter-Trillion Grid Volumes: Computational Challenges, I/O Framework, and Visualizations of Tornadogenesis. *Atmosphere* **(10)** 578.
open access; DOI: 10.3390/atmos10100578

Questions?