Simulation and Visualization of Tornadic Supercells on Blue Waters

PRAC: "Understanding Tornadoes and Their Parent Supercells Through Ultra-High Resolution Simulation/Analysis"

Leigh Orf[1] Robert Wilhelmson[2,3] Roberto Sisneros[3]
Brian Jewett[2] George Bryan[4] Mark Straka[3] Paul Woodward[5]

[1]Central Michigan University
[2]University of Illinois
[3]NCSA
[4]National Center for Atmospheric Research
[5]University of Minnesota

NEIS-P2 Symposium, NCSA Auditorium, May 21, 2013

Outline

Tornadoes

- What occurs within supercell thunderstorms that leads to tornado formation (tornadogenesis)?
- What balance of forces exists during a long-lived tornado in a supercell?
- Can we predict with any skill when such tornadoes will occur vs. when we get a weak tornado or no tornado at all?

The scientific approach

- Utilize an idealized cloud model designed specifically for massively parallel architectures (CM1)
- Initialize simulated storms in environments known to be conducive to creating supercells with strong, long-track tornadoes
- Ultimate scientific goal: Catch tornadogenesis in the act, followed by a strong, long-track tornado, for different storms

The scientific challenge

- Only $\sim$20-25% of supercells produce tornadoes
- Only $\sim$5-10% of supercell tornadoes are strong, long-track tornadoes
- We do not clearly understand why some storms produce tornadoes and others do not, although certain environmental characteristics have been correlated with supercells that produce strong tornadoes
- Much like with the real atmosphere, the details of the storm that will unfold in the model is very difficult to predict
- We therefore cannot be certain that a given simulation will produce the storm we wish to analyze

The computational challenge

- A very small time step is required to maintain computational stability
- Each time step involves many computation and communication cycles, some global communication[1]
- High temporal frequency of model state needs to be saved to properly capture features of interest (winds are moving fast during tornado genesis and maintenance)
- Hence, a tremendous amount of I/O is possible when saving the full model state at high temporal frequency (O(1PB))

[1] Adaptive time stepping only

CM1

- Developed by George Bryan (National Center for Atmospheric Research)
  - Lightweight
  - Fortran 95
  - Dynamic memory allocation
  - Run-time configurability
  - MPI (non-blocking) + OpenMP loop parallelization (`OMP_NUM_THREADS=2`)
  - Several numerics and physics options
  - Somewhat modular, making it relatively straightforward to modify

VisIt

- Developed at Lawrence Livermore, supported on Blue Waters
  - Robust plugin development environment
  - Designed for massively parallel architectures
  - Has its own programming language for derived quantities
  - Has a Python interface
  - Client-server model lets you display data on your workstation while rendering on Blue Waters
  - Easy to create scripts for movie creation
  - etc. . .

Blue-Waters specific code creation/modification

- Primary changes thus far have involved I/O and visualization
  - ✓ **CM1:** Addition of HDF5 output option. Combination of serial (for “3D” files) and parallel (for “2D” files) HDF5 routines.
    - **3D:** So-called 3D files each contain multiple time levels, each time level containing multiple 3D floating-point arrays of the model state with the arrays spanning the computational memory space of each node
    - **2D:** So-called 2D files contain single time levels, with multiple variables (many of which contain statistical and derived fields), each a 2D array spanning the full horizontal model domain at a single vertical level (“weather map” view)
  - ✓ **VisIt:** Creation of VisIt plugin to work with 3D files natively

HDF5 as a CM1 output option: Serial (for 3D files)

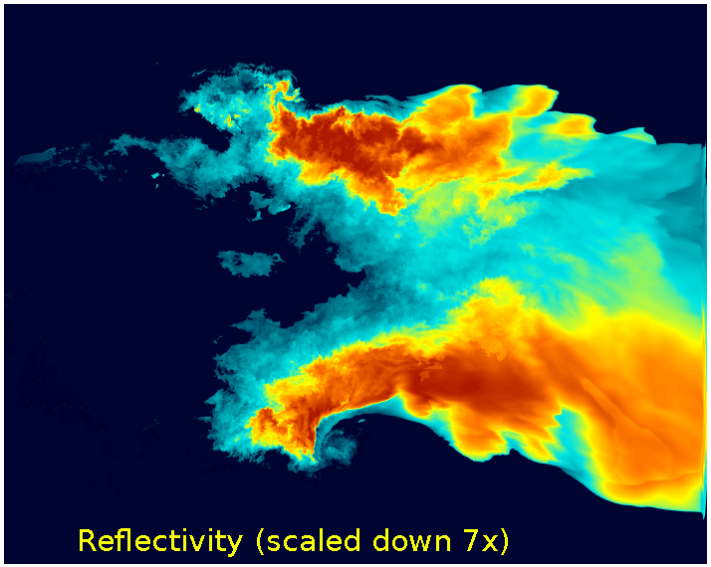
- Assign one I/O core per node
- Intranode communication done to collect data to I/O core (fast, does not hit HSN)
  - This requires rank reordering to ensure virtual topology is mapped to the hardware topology
- HDF5 “core” driver is utilized. One HDF5 file per node is buffered to memory
- File contains a group for each time level saved, with 3D floating point arrays saved in each group
- Buffered writes are nearly instantaneous (some overhead for compression)
- **gzip** compression on floating point data reduces I/O footprint

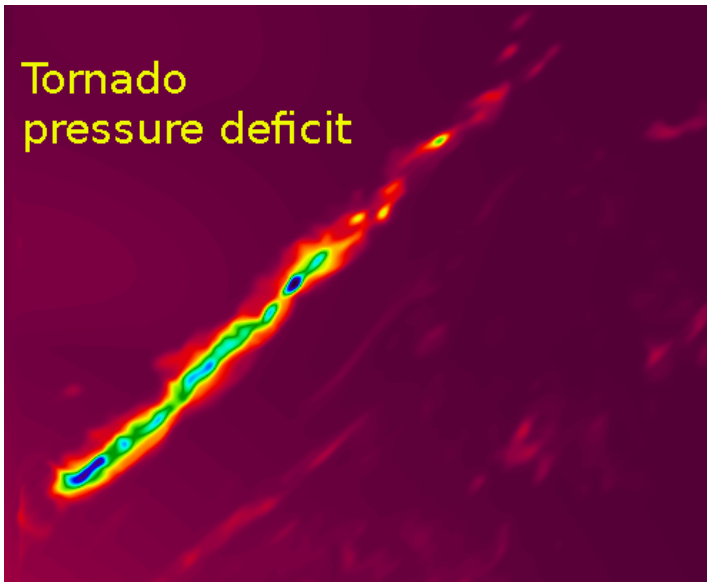
HDF5 as a CM1 output option: Serial (for 3D files)

- CM1 model itself only utilizes $\sim$3% of available memory, we can write 50-100 time levels to memory before flushing to disk
- Hence, 1 file per node is written, and each file contains around 10 3D variables at 50-100 time levels
  - This approach reduces the number of times data is written to disk, and ensures large files (which reduces latency associated with writing many small files)
- Memory utilization is less predictable than we had hoped with the core driver, OOM killer has bit us more than once

HDF5 as a CM1 output option: Parallel (for 2D files)

- All-ranks-to-one-file written infrequently in 2D files. No buffering or compression options for Parallel HDF5.
- pHDF5 transparently handles the communication, setting the number of writers, etc., when writing a single file spread out amongst all MPI ranks
- 2D files provide a snapshot of the full model domain at a given level, good for monitoring progress - can be viewed immediately with tools like **ncview**
- Performance is not very good for writing this way, need to look into this (have tried changing number of OSTs, no discernible change)





Creation of a VisIt plugin for 3D HDF5 data

- Plugin was created after HDF5 3D file format strategy was worked out
- API / low-level file management code written in C “hides” the complexity of the multiple file / multiple time levels per file structure
- In order to reduce disk operations, a one-time utility is run to create a small HDF5 file that contains data about the CM1 3D file contents - this is what VisIt is pointed to.
  - Can specify only subdomain, hiding full domain from VisIt (drastically reducing I/O time)
- Due to I/O abstraction, plugin is not restricted to the size and number of VisIt domains - plugin can therefore set domain dimensions that work best for given number of ranks, hardware, etc.
- API used for all 3D access (not just VisIt)

API for 3D HDF5 data

Call from avtcm1visitFileFormat.C:

```
read_hdf_mult_md((float *)rv->GetVoidPointer(0),
   topdir, timedir, nodedir, ntimedirs, dn,
   dirtimes, alltimes, ntottimes,
   alltimes[timestate], (char *)varname,
   x_start, y_start, x_stop, y_stop,
   z_start, z_stop, nx, ny, nz,
   nodex, nodey);
```

For details, see whitepaper
"An I/O strategy for CM1 on Blue Waters"

Improving CM1 performance using Paul Woodward's techniques

- Paul Woodward's group has achieved sustained performance of 1.5 Pflop s^{-1} on Blue Waters with his PPM code
- This is due in part to the numerical technique employed and the way blocks of memory (briquettes) are manually mapped to cache, increasing computational intensity (a measure of the number of flops performed divided by the number of off-chip words either read or written)
- A code translator is being developed which takes "standard" PPM Fortran as input and outputs "enhanced" code that runs faster
- Work is underway to apply these techniques to CM1

CM1 performance

- Entire 2 hours of cloud simulation at 20 m horizontal resolution can run in fewer than 12 hours of wallclock time on 204,800 cores (6400 nodes) for 10 s 3D I/O[2]
- Wallclock time is most sensitive to I/O frequency, less sensitive to amount of microphysical calculations (not anticipated)
- Communication takes up about 20% of wallclock time

[2] There were issues with /scratch for this run; calculations indicate we could have saved every 5 s with healthy /scratch and still fit in 12 hour wallclock window

CM1 timings

3s I/O, 5s stats, GS/LFO microphy 8:1 wall:cloud	
mp_total	22.80%
write	19.63%
stat	18.53%
microphy[3]	13.90%
sound	7.28%
advs	5.87%
tmix	3.52%

10s I/O, 10s stats, Morrison microphy 5:1 wall:cloud	
write[4]	20.87%
mp_total	17.67%
stat	16.96%
advs	11.62%
sound	9.50%
tmix	6.62%
microphy	4.53%

60s I/O, 30s stats, Morrison microphy 4:1 wall:cloud	
mp_total	23.60%
advs	14.73%
sound	11.98%
stat	11.54%
write	8.91%
tmix	8.39%
microphy	5.23%

[3] Unclear as to why GS/LFO is so much more expensive than Morrison

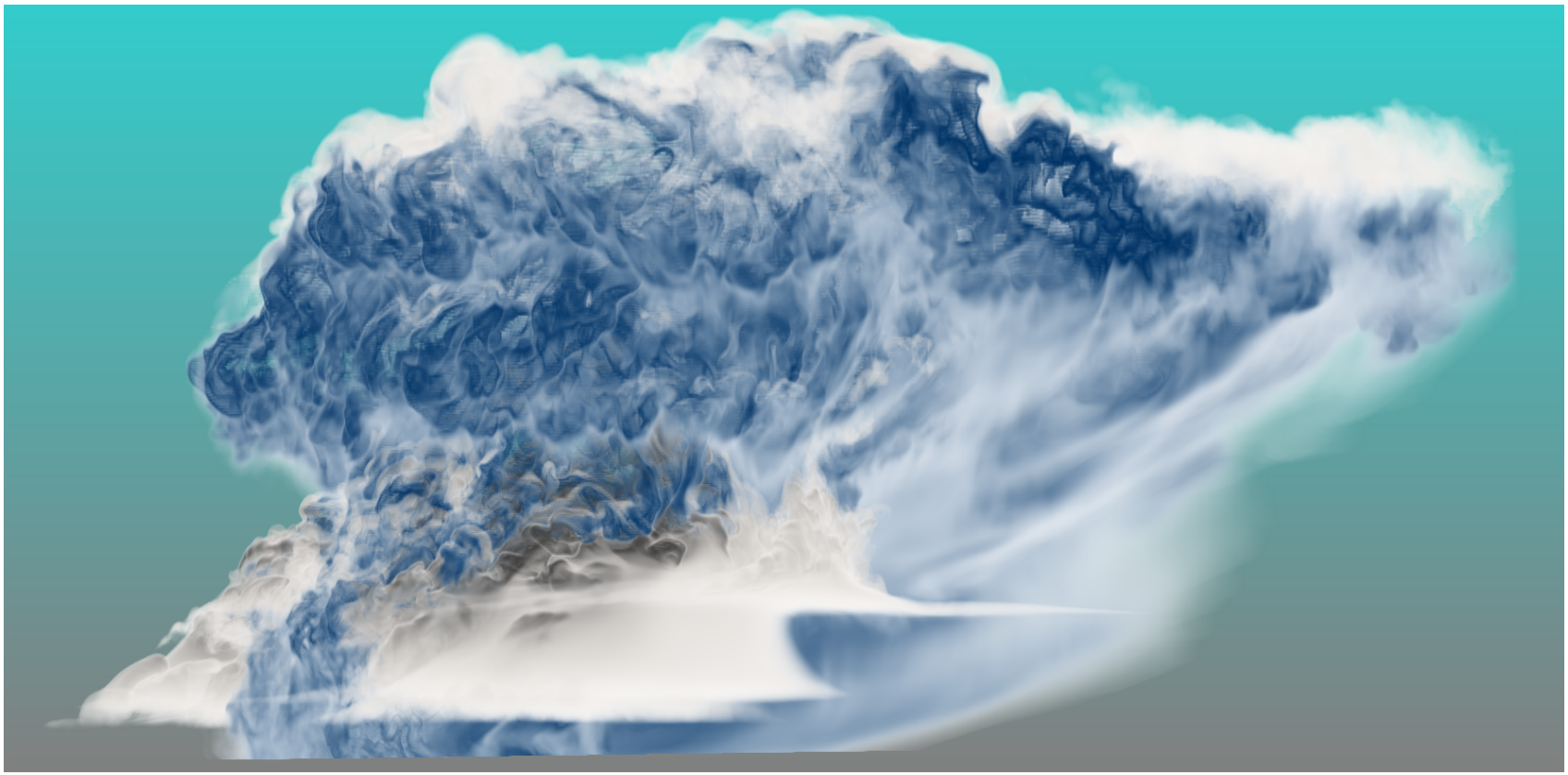
[4] Scratch issues, write probably more like 12–15% for healthy scratch

Take-home message for CM1 timings

- For 20 meter grid spacing and 3 second I/O we can fit 2 hours of cloud into two 12-hour runs using Morrison microphysics
- For doing science, we do not need to save entire simulation with high temporal frequency over full domain (For making full-storm production movies/video we do)
- Adaptive time stepping costs us early (with collective stats being calculated at each time step) but pays off later when tornado forms, ratcheting down the time step, keeping model stable
  - Could go with static time step early, run adaptive from restart file
  - Could calculate stability parameter every n^{th} time step instead of every time step

VisIt performance

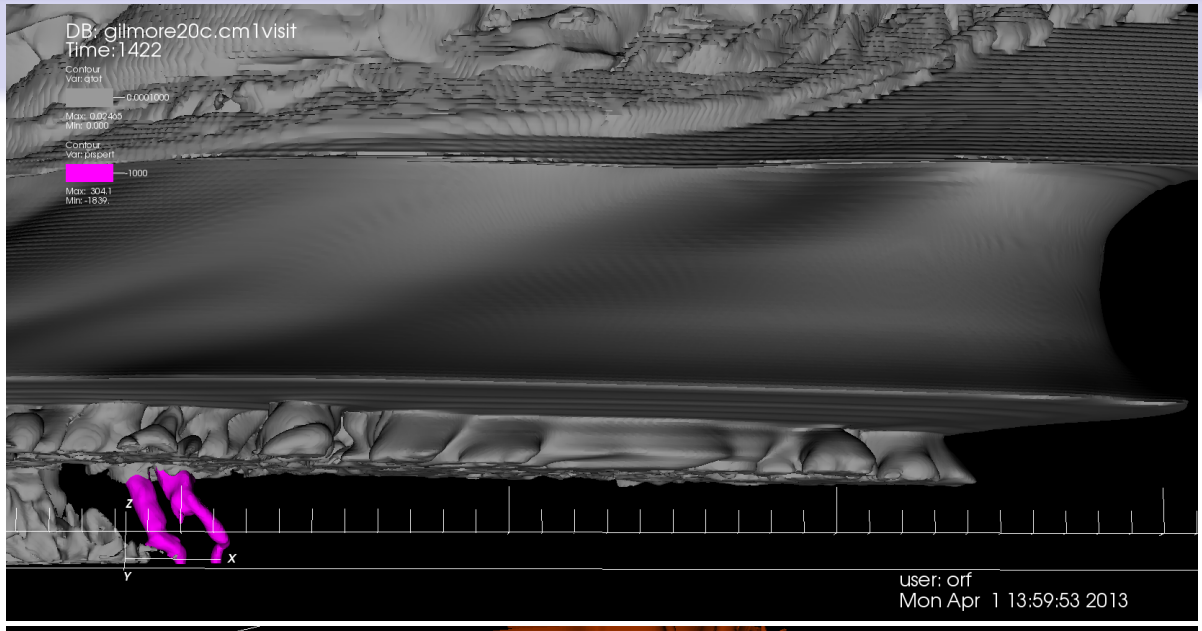
- Plug-in fully operational
- Raycasting produces good results but sometimes crashes (OOM, communicating samples)



Supercell cloud and hydrometeors

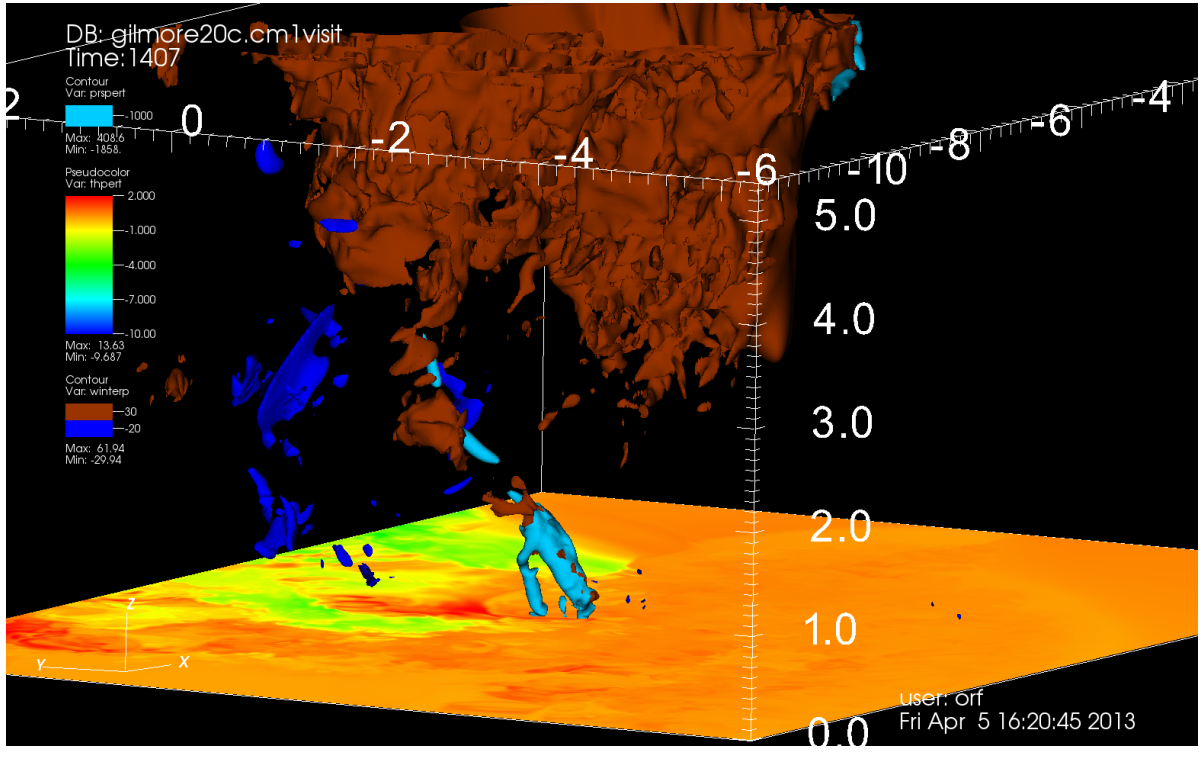
VisIt performance

Cloud with multiple vortex tornado



Updraft and downdraft with multiple vortex tornado, surface temperature

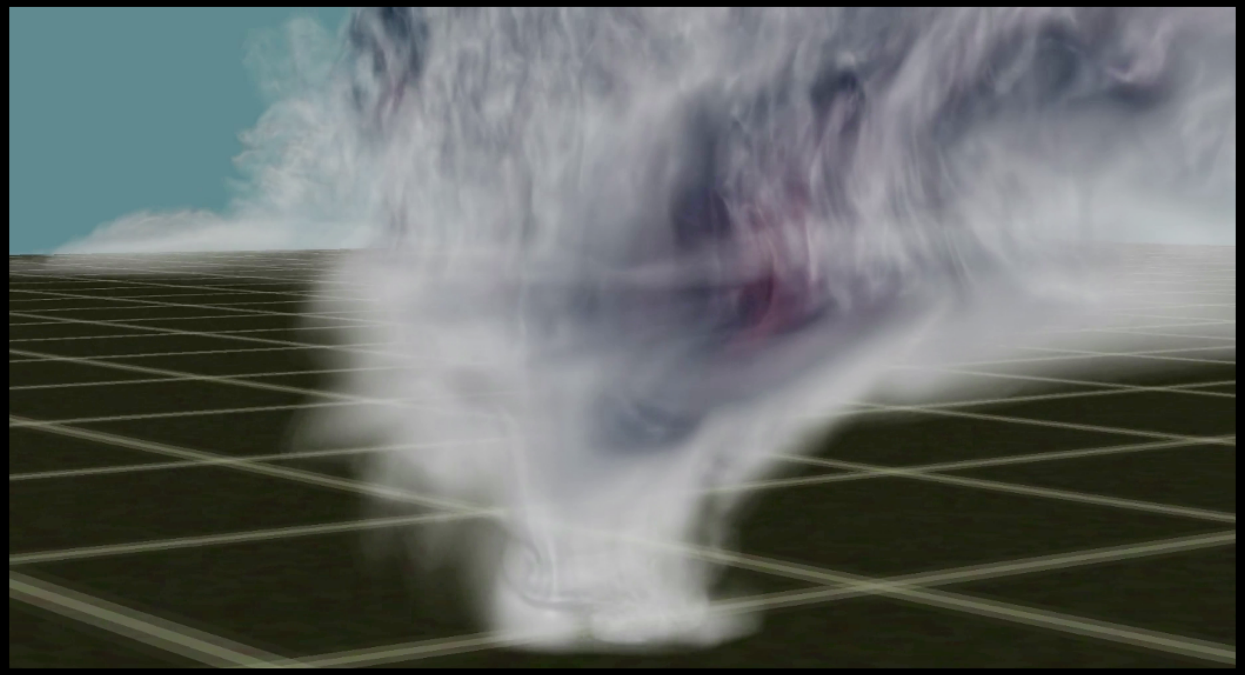
Note: only subdomain rendered



Closing Remarks

- We are able to successfully run supercell simulations that were previously not possible with other (e.g., XSEDE) resources
- Visualization and conversion tools (VisIt, tools to convert model data to other formats) are working and stable
- I/O is a potential bottleneck but we have found solutions that allow our scientific goals to be achieved
- Biggest challenge is very basic: Getting the model to produce the type of simulation we are interested in achieving
  - This issue is fundamental to computational meteorology. We know CM1 can produce a tornado in a supercell simulation, but these high-resolution simulations are uncharted territory

Questions?



Downburst-producing thunderstorm rendered on Blue Waters

(CM1 model run on kraken)

Vis. credit: Rob Sisneros, NCSA