Article

A Violently Tornadic Supercell Thunderstorm Simulation Spanning a Quarter-Trillion Grid Volumes: Computational Challenges, I/O Framework, and Visualizations of Tornadogenesis

Leigh Orf [1]

[1] The Cooperative Institute for Meteorological Satellite Studies (CIMSS), University of Wisconsin – Madison; leigh.orf@ssec.wisc.edu

Abstract: Tornadoes remain an active subject of observational and numerical research due to the damage and fatalities they cause worldwide as well as poor understanding of their behavior, such as what processes result in their genesis and what determines their longevity and morphology. A numerical model executed on a supercomputer run at high resolution can serve as a powerful tool for exploring the rapidly evolving tornado-scale features within a simulated storm, but saving large amounts data for meaningful analysis can result in unacceptably slow model performance, an unwieldy amount of saved data, and saved data spread across millions of files. In this paper, a system for efficiently saving and managing hundreds of terabytes of compressed model output is described in order to support a supercomputer simulation of a tornadic supercell thunderstorm. The challenges of managing a simulation spanning over a quarter trillion grid volumes across the Blue Waters supercomputer are also described. The simulated supercell produces a long-track EF5 tornado, and the near-tornado environment is described during tornadogenesis, where single upward-growing vortex undergoes several vortex mergers before transitioning into a multiple vortex tornado.

Keywords: Tornadogenesis; numerical simulation; volume rendering; visualization; supercell thunderstorm; Hierarchical Data Format; ZFP compression; VAPOR3

1. Introduction

Supercell thunderstorms remain an active target of both observational and numerical study, as they are the source of the majority of observed tornadoes, and are always associated with the most damaging, those rated EF4 and EF5 on the Enhanced Fujita scale. Since the 1970s, when computing technology had advanced enough, numerical models have been used to explore the three-dimensional structure and morphology of supercell thunderstorms. The earliest simulations (e.g., [1–3]) were coarsely resolved and spanned a domain too small to hold the full storm, which limited their utility; however these same simulations captured important physical and morphological characteristics of supercells that have been found to be consistent with both theory and observations. Since this time, additional 3D cloud model simulations, e.g., [4–13], containing tornadoes or tornado-like vortices have been conducted at increasingly higher resolutions over time; however it is not clear that data is being saved at high enough spatial and temporal resolution to capture the relevant processes involving tornado genesis and maintenance in many of these simulations.

Despite advances in computing technology and numerical model sophistication, adequately resolved simulations of tornado producing supercells are lacking. Observational and numerical studies of tornado producing supercells have shown an abundance of subtornadic vortices along cold pool boundaries in supercells [14–17]. The smallest of these vortices exhibit diameters of tens of meters, and hence could not be present in supercell simulations run with O(100) meter resolution found in many recent studies, nor would even the most sophisticated subgrid turbulence closure be able to account for these unresolved features. Further, many numerical studies of supercells use a stretched mesh focusing the most vertical resolution at the ground but decreasing with height. This configuration can result in improperly dissipating turbulent kinetic energy, piling up energy at high wave numbers [18]. These choices of problem size and the use of a stretched mesh are chosen pragmatically to make the simulation less computationally intensive. While most, but not all, modern thunderstorm models are able to take advantage of distributed memory architectures such as those found on modern supercomputing hardware, scaling a simulation to run efficiently on hundreds of thousands to millions of processing cores is a nontrivial exercise. Without special attention to the underlying communication infrastructure, numerical models will scale poorly to large core counts to make such high resolution simulations unfeasible. In the United States, federally funded supercomputing systems such as Blue Waters [19,20] require a full NSF proposal process that includes evidence of reasonable scaling to large core counts. A model that scales poorly will likely not be granted access to run at large core counts, and even if it was, the amount of useful science produced would be minimal.

Further limiting scientists' ability to run large simulations on distributed memory supercomputers is the input/output (I/O) bottleneck. Typically, simulations of supercells are executed whereby floating-point data is saved periodically (typically on the order of once per model minute or more) for *post hoc* analysis, which can take many months to years to complete due to the complexity of the simulation and the traditional methods for doing analysis (such as the use of Lagrangian tracers). A recent trend in large-scale modeling is the use of *in situ* visualization and analysis [21] which enables investigators to visualize model output (and even manipulate the simulation) on the same supercomputing resources while the model's state is held in core memory. The utility of *in situ* visualization and analysis for tornado research is questionable, however, considering that a scientifically interesting simulation may take months to years to analyze, and this analysis will be done from files saved to disk that are repeatedly accessed with analysis and visualization software, much of which may be written by investigators long after the simulation was executed. This typical analysis workflow does not benefit from the ability to visualize and/or analyze a simulation while it is in core memory of a supercomputer.

Thus, a significant limiting factor in conducting research on tornadic supercells is the ability to save significant amounts of data frequently in order to capture, at high resolution, the rapidly evolving flow associated with tornado behavior. Abundant visual evidence and radar observations indicate that the process of tornadogenesis often occurs rapidly, over periods of dozens of seconds, where surface flow quickly goes from nontornadic to producing visible debris from tornado strength damage. These observations further support the notion that in order to properly model tornado behavior in supercells, a small time step (associated with a small grid spacing) is required.

The well-known issue with increasing resolution in a three-dimensional model is that halving the grid spacing (doubling the resolution) requires 2^3 (8) times more computer memory just to hold the arrays that define the model state. Further, in order to maintain computational stability, a halving of the time step is required, resulting in (at least) 16 times more calculations than at twice the grid spacing. In reality, due to having larger communication buffers for exchanging halo data between nodes and the increased latency and jitter inherent in larger simulations of this kind, performance is even worse than back-of-the-envelope calculations imply. If one considers a supercell simulation with isotropic 100 m isotropic grid spacing, the same simulation run with 10 m grid spacing would require more than 1,000 times more memory and, owing to a halving of the time step to assure computational stability, more than 10,000 times more calculations. Such a simulation requires both supercomputing resources as well as a model that is able to efficiently utilize these resources, including the ability to save large amounts of data frequently in order to have scientific utility.

In this paper, the author describes the use of a modified version of the CM1 model [22,23] in executing an isotropic, ten meter grid spacing, quarter-trillion grid volume simulation on the

Blue Waters supercomputer. Modifications to CM1 were done exclusively on its I/O code with no modification to the model physics. The new I/O driver, dubbed LOFS, uses the HDF5 file format [24] with the core driver (which buffers files to memory) and ZFP lossy floating point compression [25] on all 3D arrays. LOFS has enabled the author to save over 270 TB of total data in the simulation described below that includes a 42 minute segment saved in 0.2 second intervals over a 20.8 km $\times$ 20.8 km $\times$ 5.0 km subdomain, whereby I/O took up only 37% of total wallclock utilization. Post-processing code will also be described, with the main focus being a utility that converts LOFS data to individual NetCDF files. NetCDF [26] is an open-source self-describing data format developed by Unidata that is commonly used in the atmospheric sciences. Visualization of the simulation data reveals a fascinating evolution of the storm's low level vorticity field that includes formation and merging of dozens of vortices, some of which come together to form a powerful multiple-vortex tornado.

The paper is organized as follows: In the first section, the author's use of the CM1 model is briefly described. The second section describes LOFS and its objectives; how the author implemented LOFS in CM1; its file and directory layout; the internal structure of individual HDF5 files that comprise LOFS; and how data can be read back from LOFS and converted to NetCDF files. The performance of ZFP floating point compression on saved 3D data is also described. In the third section, a primarily descriptive discussion of tornadogenesis is presented using output from 2D and 3D graphical software as support, revealing the benefit of saving high resolution data at a very high temporal frequency in order to capture the rapid, fine-scale processes of tornadogenesis. The paper concludes with a brief summary and discussion of future work to be conducted.

2. The CM1 model

The CM1 model [22] was written to run efficiently on distributed memory supercomputers, using non-blocking MPI communication that overlaps communication with calculation, resulting in a computational kernel that scales efficiently to hundreds of thousands of MPI ranks. As of this this writing, CM1 had been cited in over 220 peer-reviewed scientific studies published in 31 different scientific journals [23]. CM1 offers several solver options and physics parameterizations, and the author finds CM1 especially suited for highly idealized, high resolution thunderstorm modeling due to its parallel performance at large scale, high-order accurate numerics and sophisticated microphysics parameterization options. The model mechanics and governing equations are well documented and its author makes the source code freely available.

CM1 is a hybrid-parallel model, offering both OpenMP parallelization (via the parallelization of many triply-nested for loops found throughout the code) and Message Passing Interface (MPI) for exchanging halo data between MPI ranks during model integration as well as executing occasional global communication operations. On the Blue Waters supercomputer, which contains 32 integer cores and 16 floating point co-processors per node, the best performance was found running 2 OpenMP threads per MPI rank, with 16 MPI ranks per shared memory node, hence using all 32 cores per node. It is worth noting that ideally, one would prefer a single MPI rank to span a compute node with intranode parallelization being handled entirely by OpenMP across all cores such that halo data was not exchanged on a shared memory node; however with CM1's OpenMP implementation, this results in poor performance. It is also worth noting that the performant behavior of the author's configuration on Blue Waters reflects the efforts of the MPI authors to optimize the code over the years, including its performance on shared memory hardware (e.g., [27]).

CM1 by default contains output options that include unformatted binary output and NetCDF, with one uniquely named file being saved per MPI rank per save cycle. On distributed-memory multicore computers, the option of saving one NetCDF file per node per save cycle is also available, reducing the total number of files being saved by a factor of the number of ranks per node. However, the author was unable to achieve desired research goals with all available options. This led to experimentation with the HDF5 file format and ultimately resulted in the development of a file system called LOFS, described below.

3. LOFS: Lack Of a File System

The LOFS abbreviation came about following the author's realization that what was being achieved in his work was the creation of a file system, which can be loosely defined as a method for naming and storing files for storage and retrieval. LOFS is not a file system in the traditional sense such as what is found on modern operating systems such as Linux (e.g., ext4, Btrfs, XFS, Lustre), macOS (e.g., HPFS+, APFS) or Windows (e.g., FAT, VFAT, NTFS). Rather, LOFS is a "file-based file system" comprised of directories and files that lives on top of the actual file system of the operating system being used (on Blue Waters, the underlying file system is Lustre). The LOFS abbreviation was initially chosen from the author's initials, but officially LOFS stands for Lack Of a File System, acknowledging that it's only a file system in the most basic sense, and should be discerned from the sophisticated code that underlies traditional file systems underpinning modern operating systems.

3.1. Historical context and the path to LOFS

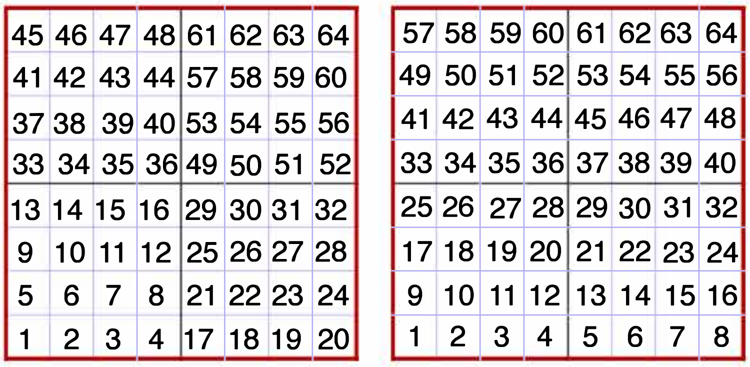
Following the acquisition of early access to the Blue Waters supercomputer in 2010, the author quickly found that I/O was be the major bottleneck with regards to wallclock utilization for large simulations on the machine with CM1. This marked the beginning of a process of trial and error with different I/O configurations with the HDF5 file format, so chosen due to its flexibility, extensibility, parallelization capability and hierarchical storage model that enable saved data to be organized much like that found on the familiar Linux operating system. At the same time, the author was collaborating with Matthieu Dorier, the software architect of Damaris, a new approach to I/O involving the use of dedicated I/O cores [28,29]. Damaris was undergoing rapid development, and was considered as an option; however due to the uncertainty of the software's future as well as the author's reticence to rely on an external package that itself relied on other external libraries written in different languages, this approach was not pursued. Ultimately, an "all Fortran" solution was pursued that would not require additional external software beyond the HDF5 and ZFP libraries or the use of multiple compilers to build the model.

Early approaches to tackling the I/O problem involved the use of parallel HDF5 (pHDF5), a natural choice for a massively parallel model writing to a parallel file system. It was quickly determined that this would result in files that were very large and unwieldy, and performance for writing single files was not acceptable. Next, a similar approach was tried where new MPI communicators that evenly divided the full model domain into identically sized blocks wrote their own files, each with pHDF5. This provided an additional advantage that allowed the saving of subdomains of the full model domain in order to reduce the total I/O load. While this offered performance advantages, a major disadvantage of the use of pHDF5 was that it did not allow for the use of any compression plugins (at the time of this writing, some compression choices were available, but still considered experimental).

Following the discovery that compression was not possible with pHDF5, only serial HDF5 options were considered. A breakthrough came following the discovery of the core HDF5 driver [30] which allows HDF5 files to be entirely constructed in local memory and later flushed to disk. Because frequent I/O was a cornerstone of the required approach, writing to memory (as opposed to disk) frequently was an attractive option. While supercomputing manufacturers, acknowledging the I/O bottleneck found in many HPC applications, have begun to address this issue with approaches such as using burst buffers [31], memory operations remain faster than the act of writing files to disk. Writing files to memory, in addition to being faster due to higher available bandwdith, also involves less latency than doing frequent I/O to disk, which is often exacerbated by the fact that the underlying parallel file system is being concurrently utilized by other jobs running on the machine. Further, Blue Waters XE nodes each contain 64 GB of memory, and, for large simulations, the memory footprint of CM1 was only about 3% of that, allowing much headroom for the construction of large files on each node. Of course, at some point data must be written to disk, but it was found that supercomputer such as

Blue Waters actually perform quite well when a lot (but not too many) large files (each on the order of several to dozens of GB each) are written concurrently - and infrequently - to multiple directories.

A final challenge in the implementation of LOFS was finding a way to map the MPI ranks to the hardware in such a way that only intranode communication to a single core on each node was required to assemble a continuous node-sized chunk of the domain (so that viewing any individual written file would be like looking at a tiny patch or column of the full physical domain). On multicore supercomputers, by default, MPI ranks are assigned in what Cray refers to as "SMP-style placement" [32] where node 1 contains the first N ranks numbered consecutively, node 2 contains the next N consecutive ranks, etc., which, for CM1's 2D node decomposition, means each node contains a distorted piece of the actual physical model domain. Rank reordering was first achieved by using external tools provided by Cray that involved the construction a file containing the reordered rank numbers that was read at runtime. This approach, however, was found to be unwieldy and in order to make rank reordering not dependent upon the type of supercomputer being used, an all-MPI approach was devised in which a new global communicator was assembled whose rank numbers were assigned to match the 2D decomposition (see Fig 1).



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

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

Figure 1. Example of rank reordering for a 4 node (2x2 2D node decomposition) simulation with 16 cores (4x4 2D core decomposition) per node. Numbers indicate MPI rank, starting at 1, for each communicator. Initially (left image, communicator MPI_COMM_WORLD) ranks are ordered in "SMP" mode where each subsequent node is populated with ranks in ascending order. Following rank reordering (right image, new MPI_COMM_CM1 communicator), nodes/ranks are mapped to the physical horizontal model domain (each rank contains the full vertical extent of the physical model domain). The lowest numbered rank on each node collects, assembles, compresses, buffers to memory, and writes to disk.

3.2. *Objectives of the LOFS approach*

LOFS is a form of what has been called "poor man's parallel I/O" [33] or, using more modern terminology, a system of Multiple Independent Files (MIF) [34]. LOFS uses serial HDF5, but from within CM1, files are written in parallel (concurrently) to disk. Supercomputers such as Blue Waters use a parallel file system that is writable by all MPI ranks, and experience has shown that so long as not too many files are written concurrently to a single directory, but spread out amongst many different directories as is the case with LOFS, excellent I/O bandwidth is achieved. However, LOFS was designed such that "stitching together" files into one single file as an intermediate step is not required to read from a continuous region of stored data. One component of LOFS is a conversion program (`lofs2nc` described below) that writes single CF compliant NetCDF [35] files which can be read by a variety of analysis and visualization software. In the case of the tornadic supercell simulation described below, the area surrounding the region of interest for studying tornado genesis and maintenance spans roughtly 5 km $\times$ 5 km $\times$ 5 km, or 500 $\times$ 500 $\times$ 500 gridpoints, which does not create too unwieldy a file size when several variables of that dimension are written to a NetCDF file.

The objectives of LOFS are as follows:

1. Reduce the number of files written to disk that would occur if each MPI rank wrote one file per save, as is traditionally done, to a reasonable number
2. Minimize the number of times actual I/O is done to the underlying file system
3. Write big (but not too big) files
4. Offer users the ability to use lossy floating point compression to reduce the amount of written data
5. Make it easy to save only a subdomain of the full model domain
6. Make it easy to read in data for analysis and visualization after it is written

These objectives are achieved as follows:

1. LOFS files each contain multiple time levels, as opposed to a single time per file, as well as having only one file per node (as opposed to one file per MPI rank per time) be written to disk. If, for example, 50 times are buffered to memory per file and each multicore node contains 16 MPI ranks, the number of files saved is reduced by a factor of 800 under a typical one time per file and one file per MPI rank paradigm.
2. Similarly, by buffering many time levels to memory as the file is assembled, many "saves" occur without writing to disk.
3. This occurs naturally as a result of the prior two items. Files will never exceed the total memory available on a given node. For the 10-m simulation described below, a typical file size in an active area of the domain was only 2.5 GB when buffering 50 ZFP compressed times to memory and saving 15 3D arrays each of the 50 times
4. This is achieved through the HDF5 plugin system that allows for external compression algorithms to be used.
5. Since hundreds to thousands of shared memory nodes are participating in the simulation, and each node independently writes LOFS data, namelist options are available that save over user-specified ranges in both the horizontal and vertical, resulting in only activating nodes that write "data of interest" over any continuous 3D subdomain. In the simulation described herein, the saved subdomain spanned 2,080 by 2,080 by 500 grid volumes (20.8 km by 20.8 km by 5 km) in x, y and z respectively, centered on the low level mesocyclone of the supercell.
6. This is achieved through the use of reading and conversion code described in section 3.6.

3.3. CM1 modifications

LOFS is written in Fortran95 and uses Fortran modules, and was written to minimize the amount of changes made to the default CM1 code for easier porting to new versions of CM1 and/or other distributed memory MPI models. LOFS has been used with CM1 release 16 (the CM1 version that produced the simulation described below) as well as CM1 release 18. The process for adapting LOFS to CM1 can be summarized as follows:

1. Remove references to existing writeout code from CM1's makefile (due to its scope, LOFS is not a "user selectable" option like NetCDF)
2. Add appropriate references to LOFS source code files and compiler options for LOFS to the CM1 makefile
3. Using the `sed` command, replace all instances of the communicator MPI_COMM_WORLD with MPI_COMM_CM1 in CM1 source code
4. Insert rank reordering code, which initializes the MPI_COMM_CM1 communicator (this is done where CM1 initializes MPI).
5. Replace existing writeout subroutine with a new writeout subroutine that calls several other LOFS subroutines that exist in their own source code files
6. Include the appropriate LOFS modules in existing CM1 source code that requires it
7. In select parts of the code, insert subroutines that do tasks such as allocating LOFS-only variables, broadcasting LOFS-only namelist values, etc.

While LOFS has only been used with CM1, the process outlined above could is designed to be similar to any similar distributed memory model that uses a 2D domain decomposition.

3.4. Writing data

In order to provide an overview of how LOFS data is written within CM1, pseudocode is first presented. In the pseudocode below, the domain decomposition has already been done such that each node has been mapped to the 2D domain decomposition shown in Fig 1, with each MPI rank containing the full vertical extent of the domain.

```
do while model_time .lt. total_integration_time
  integrate_one_time_step !Run the solver to model_time+dt
  if mod(current_model_time,savefreq) .eq. 0 !Time to do I/O
    if i am an io core:
      if first visit to this routine:
        if iamroot: make directories
        initialize_core_driver
        initialize_compression_driver
        open_hdf5_file !this creates an empty file on disk
        create_hdf5_groups
      else if this is a new write to disk cycle:
        close_groups
        close_file !This flushes the file to disk
        if iamroot: make_directories
        initialize_core_driver
        initialize_compression_driver
        open_hdf5_file !this creates an empty file on disk
        create_hdf5_groups
      else:
        close_current_time_group
        create_new_time_group
      end if
      write_metadata
      for var in all 3D selected variables:
        MPI_Gather 3D data to io core from other cores on shared memory module
        Reassemble 3D data on I/O core to continuous subdomain chunk
        write 3D data to time group
      end for
    end if !i am an io core
  end if !mod(current_model_time,savefreq) .eq. 0
end do
```

LOFS operates by first initializing the directory structure by calling the `execute_command_line` subroutine to call the Linux command `mkdir -p` to create directories that make up LOFS directory structure. Next, the HDF5 core driver and ZFP compression driver are initialized, and one core on each node that is participating in writing data opens its unique HDF5 file, which creates an empty HDF5 file on disk and in memory (data to the disk file is only written when the file in memory is closed). When it is time to write data, a single core on each participating node collects data (using `MPI_Gather`) from the remaining cores for each requested variable to be written and assembles it into a continuous block of model data that is then written to memory. Following the first write, subsequent writes build the HDF5 files in memory with each new time resulting in a new base-level group (see Table 2). When the number of times buffered to memory reaches a user-selectable value (in this case, 50), all groups and files are closed, which flushes the files to disk. Then, new uniquely named directories are created to house new files, and the process repeats until the model finishes.

For simulations in which data is saved at a very small time interval, the best performance is achieved when a large number of model times are buffered to memory. This is accomplished by creating a new base level group that is a zero-padded integer corresponding to the time index (starting with 0) being written. Groups are a useful way to organize data in HDF5 files, and they are also used to organize metadata, mesh, and the base state sounding data. Table 1 contains a listing of the scalar values (which include integer metadata unique to each file) and 1D arrays (corresponding to model sounding data) contained within each HDF5 file. Because metadata is so small compared to

variable	description
/basestate/pi0	Exner function
/basestate/pres0	pressure
/basestate/qv0	water vapor mixing ratio
/basestate/rh0	relative humidity
/basestate/th0	potential temperature
/basestate/u0	u component of wind
/basestate/v0	v component of wind
/grid/corex	number of cores in E/W direction on node
/grid/corey	number of cores in N/S direction on node
/grid/myi	E/W index of node
/grid/myj	N/S index of node
/grid/ni	number of grid points on a node in E/W direction
/grid/nj	number of grid points on a node in N/S direction
/grid/nkwrite_val	Actual number of vertical points saved
/grid/nodex	number of nodes in E/W direction
/grid/nodey	number of nodes in N/S direction
/grid/nx	number of grid points in E/W direction for full domain
/grid/ny	number of grid points in N/S direction for full domain
/grid/nz	number of grid points in the vertical for full domain
/grid/x0	first E/W grid index in file
/grid/x1	last E/W grid index in file
/grid/y0	first N/S grid index in file
/grid/y1	last N/S grid index in file
/mesh/dx	grid spacing in x
/mesh/dy	grid spacing in y
/mesh/dz	grid spacing in z
/mesh/umove	E/W box translation component
/mesh/vmove	N/S box translation component
/mesh/xf	Cartesian staggered mesh locations in E/W direction in file
/mesh/xffull	Cartesian staggered mesh locations in E/W direction in full domain
/mesh/xh	Cartesian scalar mesh locations in E/W direction in file
/mesh/xhfull	Cartesian scalar mesh locations in E/W direction in full domain
/mesh/yf	Cartesian staggered mesh locations in N/S direction in file
/mesh/yffull	Cartesian staggered mesh locations in N/S direction in file
/mesh/yh	Cartesian scalar mesh locations in N/S direction in file
/mesh/yhfull	Cartesian scalar mesh locations in N/S direction in full domain
/mesh/zf	Cartesian staggered mesh locations in vertical
/mesh/zh	Cartesian scalar mesh locations in vertical
/times	Model times in file

Table 1. A listing and description of scalar values and 1D arrays within a single LOFS HDF5 file for the 10-m run. The `basestate` group contains 1D floating-point arrays of model sounding variables mapped to the model mesh. Variables in the `grid` group refer to integer quantities that indicate the dimension of the model domain, 2D decomposition information, and indices describing where data is within the full domain. The `mesh` group contains floating-piont arrays containing the local and global Cartesian coordinates in each dimension, as well as the domain translation values chosen to keep the simulated storm centered within the model domain (i.e., the storm motion vector components). The `times` array contains a listing of the double-precision model time corresponding to each of the saves within the file.

the floating point data that contains the model state, the author chose to write identical redundant metadata to all HDF5 files. Such redundant metadata includes the number of nodes and cores per node used in the simulation, Cartesian mesh coordinates and dimensions of the full model domain, the model base state (from the CM1 `input_sounding` file) mapped to the vertical mesh, etc. Routines for interrogating and reading in LOFS data exploit this redundancy as "any file will do" to retrieve enough metadata to describe and assemble any subdomain of the simulation. The `/times` dataset contains the floating point model time that corresponds to each saved time level within the file. Metadata that varies between files includes the grid indices and Cartesian location with respect to the full model domain that are contained in the file.

array name	description of contents	dimensions
/00000/2D/static/snapshot_dbz_0500	reflectivity 500m AGL	(80, 80)
/00000/2D/static/snapshot_prespert_0500	perturbation pressure 500m AGL	(80, 80)
/00000/2D/swath/hwin_max_sfc	max sfc horiz wind swath	(80, 80)
/00000/2D/swath/hwin_max_sfc_move	max sfc translated horiz wind swath	(80, 80)
/00000/2D/swath/prespert_min_1000	min 1km AGL pressure perturbation swath	(80, 80)
/00000/3D/dbz	reflectivity	(500, 80, 80)
/00000/3D/khh	subgrid eddy viscosity	(500, 80, 80)
/00000/3D/kmh	subgrid eddy diffusivity	(500, 80, 80)
/00000/3D/ncg	number concentration of graupel	(500, 80, 80)
/00000/3D/nci	number concentration of cloud ice	(500, 80, 80)
/00000/3D/ncr	number concentration of rain	(500, 80, 80)
/00000/3D/ncs	number concentration of snow	(500, 80, 80)
/00000/3D/prespert	pressure perturbation	(500, 80, 80)
/00000/3D/qc	cloud liquid mixing ratio	(500, 80, 80)
/00000/3D/qg	graupel mixing ratio	(500, 80, 80)
/00000/3D/qr	rain mixing ratio	(500, 80, 80)
/00000/3D/qvpert	perterubation water vapor mixing ratio	(500, 80, 80)
/00000/3D/rhopert	perturbation density	(500, 80, 80)
/00000/3D/thpert	perturbation potential temperature	(500, 80, 80)
/00000/3D/thrhopert	perturbation density potential temperature	(500, 80, 80)
/00000/3D/tke_sg	subgrid TKE	(500, 80, 80)
/00000/3D/u	u component of wind on staggered mesh	(500, 80, 80)
/00000/3D/v	v component of wind on staggered mesh	(500, 80, 80)
/00000/3D/w	w component of wind on staggered mesh	(500, 80, 80)

Table 2. A listing and description of 2D and 3D arrays stored within a single HDF5 file for the 10-m run for the first of fifty stored times. All 3D arrays are shown, but only 5 of 48 2D arrays are shown for brevity. The top level group is a zero-padded index corresponding to the time level stored in the file, starting at 0 (and ending at 49 for this run). The 2D subgroup contains two subgroups of its own: `static` contains 2D horizontal snapshots of model prognostics and diagnostics while `swath` contains horizontal slices of statistical properties. The levels above ground of each 2D slice are chosen by the user. The 3D subgroup contains all of the chosen three-dimensional floating point model prognostics and diagnostics. Each 2D and 3D floating point array is a continuous chunk of the full physical model domain that can be viewed independently, but is typically used as a building block for the assembling of larger subdomains for conversion to file formats such as NetCDF, or read in directly to an array for analysis or visualization.

Table 2 contains a listing of 2D and 3D floating point data found in each HDF5 file. These include two-dimensional horizontal patches of user-selected static variables at various model heights, CM1 "swath" variables that provide a statistical look at both mesh relative and ground relative quantities over time, and 3D data describing the model's state. The top level group is a zero-padded index describing what time level (starting at 0, and ending at 49 in this example) is being saved in the file, with the 2D and 3D subgroups indicating the rank of the arrays that follow. Actual array dimensions are also found in the listing; for 2D arrays, they are in (ny, nx) order and (nz, ny, nx) order for 3D arrays.

symbolic notation	Example	description
`history`	`history`	Prefix always used for history data
`String`	`ElReno10m-a`	User supplied descriptive string
`3D`	`3D`	Indicates the directory contains three-dimensional data
`TTTTT`	`05000`	zero padded integer time in seconds
`ttttttt`	`2000000`	fractional part of time in seconds
`nnnnnnn`	`0009000`	node directory containing files on nodes 9000-9999
`NNNNNNN`	`0009151`	node number of file

Table 3. Naming convention for one member of a collection of HDF5 files that comprise LOFS. The full file with path for this example is `history.ElReno10m-a/3D/ElReno10m-a.05000.2000000/0009000/ElReno10m-a.05000.2000000_0009151.cm1hdf5`. The simulation described herein contains a subdomain of the full model domain, and is comprised of 676 HDF5 files out of a total possible 19,600

3.5. Directory layout and file naming convention

A set of Fortran95 subroutines constructs the total path and filename for each HDF5 file based upon strict rules, with variations in the file and directories that are a function of the node number and time of the first saved time within the HDF5 file. The naming convention is demonstrated by first looking at the full path to a single file, corresponding to a file written by node number 9151 starting at time=5000.2000000 seconds:

```
history.ElReno10m-a/3D/ElReno10m-a.05000.2000000/0009000/ElReno10m-a.05000.2000000_0009151.cm1hdf5
```

The naming convention is as follows:

```
history.String/3D/String.TTTTT.tttttttt/nnnnnnn/String.TTTTT.ttttttt_NNNNNN.cm1hdf5
```

Table 3 describes the components of the entire path and file name. The hierarchy within LOFS data is based upon model time, with new directories whose name contain the floating point model time being created for each time data is flushed to disk. In order to avoid performance issues related to having too many files in a single directory, data is spread in 1000 file chunks in subdirectories whose names are simply a sequence of zero-padded factors of 1000.

In the simulation described below, a maximum of 1000 HDF5 files will be found in any node subdirectory. Spreading the many files that make up a simulation amongst many directories has been found to produce much better performance than placing them all in one directory. In the example above, node directory 0009000 would contain, for a simulation in which the full domain was saved, a thousand files with node numbers 9000 through 9999.

It should be emphasized that on the read side, metadata is extracted from the file and directory names themselves (in addition to retrieving metadata from files) in order to reconstruct the full domain space such that the proper files can be accessed when users request data. Further, because of the high temporal frequency of data saved in this simulation (in this case, every 0.2 s), the time string in seconds embedded within the HDF5 files is actually a floating point representation of the time that is then converted to floating point data in the read-side code. Because each HDF5 file contains 50 time levels in this simulation, the HDF5 time embedded within the file name descriptor refers to the first of 50 times contained within the file, with all saved times comprising the `/times` dataset (a 1D double precision array) stored in each file.

By enforcing strict file and directory naming conventions, read-side applications programmed with knowledge of these conventions (or using the existing LOFS read-side routines) can traverse the directory structure and extract metadata from directory names, file names, and finally the files themselves. Then, using HDF5 calls, a single floating point buffer is created for each requested variable from multiple files and is stored into an array for analysis, visualization, conversion, etc.

argument	description
`--time=5400.0`	Select $t = 5400$
`--offset`	Indices are with respect to what was saved, not the full domain origin (0,0)
`--x0=800`	west boundary index
`--y0=750`	south boundary index
`--x1=1300`	east boundary index
`--y1=1250`	north boundary index
`--z1=300`	top boundary index
`--histpath=3D`	Top level 3D LOFS directory
`--base=torscale-10m`	base name given to netcdf files
`--swaths`	Retrieve and save all swaths
`--nthreads=4`	Number of OpenMP threads to use for calculations
`thrhopert prespert...`	The variables requested

Table 4. Description of command line arguments to `lof2nc` such as those used to create NetCDF files that are shown in all storm imagery in this paper.

3.6. Reading data from LOFS

For reading LOFS data, routines written in C have been created that provide a way to access any saved variable at any time spanning any arbitrary subdomain found within the LOFS data. To demonstrate, an example of the `lofs2nc` utility is described below, creating a NetCDF file containing all of the 2D found within a subset the LOFS data as well as selected 3D data.

Consider a situation where a user wishes to make a NetCDF file that spans the region immediately surrounding the tornado in the simulation for visualization. This is achieved with `lofs2nc`, a front end to a series of underlying routines that traverse the LOFS structure to extract metadata from directory names and file names, reads redundant metadata from one of the HDF5 files, and then loops across files, incrementally filling a buffer with user-requested data. An example command follows:

```
lofs2nc --time=5400.0 --offset --x0=800 --y0=750 --x1=1300 --y1=1250 --z1=300 \
--histpath=3D --base=torscale-10m --swaths --nthreads=4 \
thrhopert prespert uinterp vinterp winterp xvort yvort zvort qc qr qg ncg ncr dbz
```

Table 4 breaks down the different options to the above `lofs2nc` command. The user has requested a 501 (in x) by 501 (in y) by 301 (in z) volume of data whose indices are relative to the origin of the file corresponding to the lowest saved node number. The user has requested a list of variables, some of which are read directly from the LOFS data (`thrhopert`, `prespert`, `qc`, `qr`, `qg`, `ncg`, `ncr`, `dbz`) and some which require calculation based upon other saved data (`uinterp`, `vinterp`, `winterp`, `xvort`, `yvort`, `zvort`). Due to the large amount of data produced by simulations run at this scale, the author has chosen a save strategy that focuses on saving the minimum number of 3D arrays needed to calculate any possible other diagnostic/derived fields and calculating those fields on the fly (with calculations parallelized using OpenMP on multicore machines). For instance, the LOFS data saved for this simulation saved the native u, v, and w CM1 variables that lie on the staggered Arakawa C grid [36] such that the velocity components interpolated to the scalar mesh, as well as the vorticity components interpolated to the scalar mesh, are calculated within the `lofs2nc` code. This specific strategy ensures that the highest amount of accuracy is preserved for all calculations involving velocity variables in methods consistent with those done within the CM1 model.

The process by which the LOFS read-side code retrieves metadata involves an entire traversal of the saved directory structure as well as reading metadata from at least one of the HDF5 files in each time directory. These combined operations can take dozens of seconds to complete when hundreds to thousands of time levels are saved, but because the results of these operations always return identical information, the caching of this information is done the first time the command is run, and cache files are read for subsequent executions. This makes metadata acquisition essentially instantaneous (only requiring the formatted reading of several small text files), leaving the code to spend its time to read and write and, if requested, calculate derived variables.

Pseudocode for `lofs2nc` is found below. Note that the first four routines will first check for cache files and read from these if they exist; otherwise, the LOFS directory structure is traversed and, using routines of the `dirent` C library, file names within directories are read, sorted, and a single HDF5 file is selected for internal metadata acquisition. Cache files are written if they do not exist, such that this process only occurs once, and does not need to be done again until the LOFS file structure is modified (for example, by adding more time levels as a simulation progresses through time).

```
get_num_time_dirs !get number of top level directories in 3D directory
get_sorted_time_dirs !get the directory names and sort using quicksort
get_sorted_node_dirs !get a sorted list of the zero-padded node directories
get_all_available_times !get a list of every time level saved in all the data
get_available_3D_variable_names !get a list of all 3D variable names
get_LOFS_metadata_from_a_HDF5_file !get mesh, grid, and basestate information
initialize_netcdf !Set dimensions, attributes, and define requested variables
if getswaths: get_all_swaths; write_all_swaths !All 2D fields read and written
get_data_to_buffer !Buffer variables for diagnostic calculations, if requested
for each requested variable:
        if exists_in_LOFS and not_already_buffered: read variable into buffer
        if is_diagnostic: calculate diagnostic into buffer
        write_requested_quantity_to_netcdf_file
```

Following metadata acquisition, the list of requested variables is checked against what is available, and, if diagnostic quantities (such as vorticity components) are requested, temporary arrays are allocated and a flag is set to indicate which LOFS variable is needed for the diagnostic quantity for buffering to memory. As an example, if the vertical component of vorticity (ζ) was requested (called `zvort` in CM1/LOFS), u and v would be buffered to their own arrays such that `zvort` could be calculated. If u and v were also requested for writing to NetCDF files, these buffered values would be written without re-reading LOFS data. Hence, the code only allocates memory for what is needed and re-uses buffered variables when needed. The main loop over variable names contains code for calculating diagnostic quantities, and new quantities can be added at the end user's leisure.

All LOFS HDF operations are serial, and this is also true with `lofs2nc`. However, because modern supercomputers use parallel file systems, excellent performance is found executing serial I/O operations in parallel by writing scripts that execute many instances of `lofs2nc` concurrently. This workflow is nearly always used since a main motivation for LOFS is doing analysis at very high temporal resolution. Running several instances of any code is easily achievable using shell scripting where jobs are forked into the background. A tremendous amount of data can be read and written quickly in this manner, exploiting the inherent parallelization of the operating system running on a multicore machine and reading (writing) from (to) a parallel file system such as Lustre or GPFS.

3.7. The use of ZFP compression

The primary motivation of the development of LOFS was to enable the saving of very high resolution data at extremely high temporal resolution. This combination will, without care, result in unacceptably poor model I/O performance due to insufficient I/O bandwidth, as well producing as an overwhelmingly large amount of data. LOFS offers, through the use of input parameters in the `namelist.input` file that CM1 reads upon execution, the ability to save only a subset of the model domain, which is practical when one is interested in studying the region of the storm directly involved in tornado processes near the ground. In the simulation described below, a volume spanning 2,080 by 2,080 by 500 grid points (corresponding to 20.8 km by 20.8 km by 5 km in x, y and z respectively) was saved in LOFS files from $t = 5000.2$ s through $t = 7490$ s, spanning 0.86% of the volume of the full model domain (see Fig. 3 to see the horizontal extent of saved data). However, this reduction still results in an unwieldy amount of uncompressed 32 bit floating point data when saved in 0.2 s intervals over the life cycle of the tornado. In order to reduce the amount of data saved to disk to a reasonable value, ZFP [25] floating point compression was used.

ZFP uses a *lossy* algorithm that results in three-dimensional floating-point data that contains less than the original 32 bits of precision after being uncompressed. However, the resulting data exhibits compression performance that (usually) far exceeds that of lossless compression algorithms. Further, an attractive aspect of ZFP is that it offers a dynamic compression option whereby the absolute maximum amount of accuracy required for each value in each 3D array may be specified, and that this accuracy parameter is specified in the same units as the saved data. For instance, in the supercell simulation, simulated reflectivity (shown for example in Fig. 3) is saved in order to create plots that can be compared to real radar observations of supercells. This variable is used for plotting purposes only, and will never be differentiated or used in *post hoc* analysis in equations where high amounts of accuracy are needed. A value of 1 dBZ was chosen as an accuracy parameter for the `dbz` variable, which means every value in the uncompressed 3D `dbz` arrays will exceed 1 dBZ of accuracy. ZFP's algorithm is conservative, and in reality, the accuracy of the data will typically far exceed this specified value throughout the 3D arrays.

Figure 2 provides a box and whiskers plot of compression performance for several 3D variables at a snapshot in time when the tornado is mature and exhibiting a multiple vortex structure. These statistics were chosen from all 676 files spanning the saved subdomain. Data within this subdomain contains regions of both weak and sharp gradients, and the widely varying compression performance in this example reflects this. For the aforementioned `dbz` variable, which was saved with 1.0 dBZ accuracy, the space taken by the 3D array was 5.2% of its uncompressed size, providing an average compression ratio for this variable of approximately 19:1. Had a larger accuracy parameter been chosen, compression performance would have increased accordingly.

Because future *post hoc* analysis will involve Lagrangian trajectory analysis and the effects of lossy compression on trajectory performance have yet to be determined by the author, a very small value ($0.1\,\mathrm{mm\,s^{-1}}$) was chosen for each of the three components of velocity, each of which exhibited very similar compression performance. This resulted in an average reduction to 30% of uncompressed, or roughly a 3:1 average compression ratio, with values ranging from 10% to 58% of uncompressed across all saved files. This wide variation in compression performance across files is indicative of the dynamic nature of ZFP compression, where regions of large gradients result in less compression than regions of weaker gradients. The `prespert` variable (perturbation pressure in hPa) shows several outlier compression reductions that correspond to regions of abnormally large gradients that are found along the length of the tornado (which is tilted, spanning several files) and other weaker vortices. The cloud mixing ratio variable `qc` exhibits compression performance in some files where nearly no data is saved, corresponding to cloud free areas of the simulation, but also many outliers where large gradients are found within the cloud including along the periphery of the condensation funnel of the tornado.

So long as accuracy values are chosen carefully for each variable, one may be assured that post processing and visualization will not result in artifacts or a substantial loss of accuracy that would result in faulty *post hoc* analysis. For work of this nature, the use of lossy compression is unfortunate but necessary and can be thought of as a trade-off between spatial accuracy and temporal accuracy; without lossy compression, in order to reduce the data load, data would need to be saved with significantly coarser temporal resolution, removing the ability to do the kind of visualization and analysis needed to achieve desired research goals.

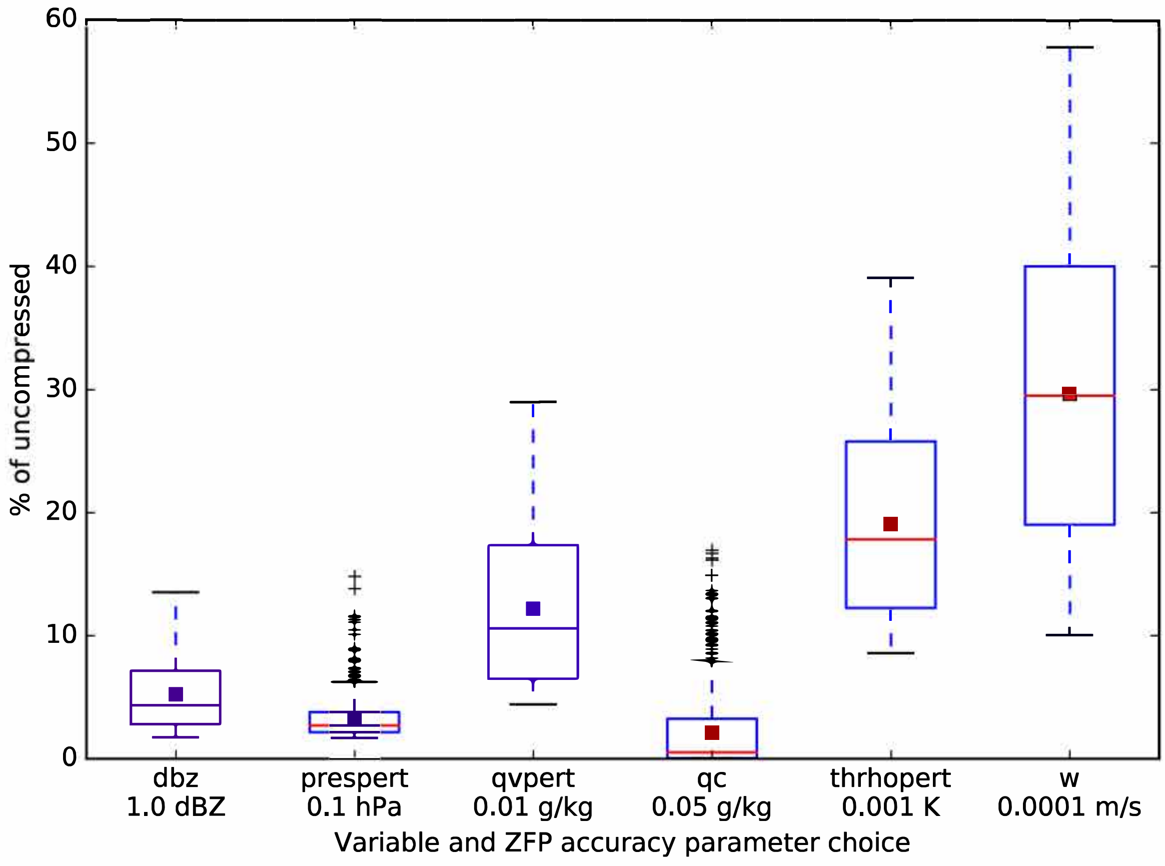


Figure 2. A Tukey [37] box and whiskers plot of compression performance statistics (in terms of percentage of uncompressed size) of six of the saved variables within the 676 files spanning the full saved subdomain, which is roughly centered on the tornado. The green line represents the median value, the red square the mean value, while the box contains the interquartile range (IQR) with the bottom and top whiskers within 1.5 IQR of the lower and upper quartile, respectively. Outliers are plotted with the '+' symbol. The accuracy parameter chosen at runtime is noted below each of the 3D variables names. Larger (smaller) values of the accuracy parameter would have resulted in smaller (larger) file sizes. Variables displayed are simulated reflectivity (`dbz`, in dBZ), perturbation pressure (`prespert`, in hPa), perturbation water vapor mixing ratio (`qvpert`, in g/kg), liquid cloud water mixing ratio (`qc`, in g/kg), perturbation density potential temperature (`thrhopert`, in K) and vertical wind speed (`w`, in m/s). The mean compression performance for the shown variables is 11.9% of uncompressed, or a compression ratio of 8.4:1

4. The 10 m resolution tornadic supercell simulation

The simulation was run on an isotropic mesh with a grid spacing of 10 m, in a box spanning 112 km by 112 km by 20 km (using 11, 200 × 11, 200 × 2000, or 250,880,000,000, grid points). A model time step of 0.04 s was used in order to maintain stability, and a short time step of 0.01 s was used for the acoustic time step. The chosen time step is smaller than what was necessary for computational stability due to aliasing; however, a larger time step resulted in sporadic failure of CM1's saturation adjustment scheme, which is iterative, to converge, causing the model to abort. The remaining model parameters were identical to those of [17] with the exception that the TKE closure option of Deardorff et al [38] was used in place of that of Smagorinsky [39].

4.1. Execution of the simulation on Blue Waters

The supercell simulation was executed on the Blue Waters supercomputer in 22 segments between April 19 and August 2, 2019. Each segment of the simulation beyond the first was run from checkpoint files that were saved in HDF5 format with all 3D state variables compressed with lossless gzip compression. The simulation took approximately ten million node hours (320 million core hours) in total to run to a model time of 7490 s, a few minutes after the cyclonic tornado fully dissipated. This corresponds to just over 21 days of execution using 87% of all available XE nodes on Blue Waters.

Each segment of the simulation ran on 19,600 nodes (672,200 cores) when executing, with a maximum requested run time of 48 hours per segment, the largest allowed on the machine. However, when running over such a large portion of the machine, node failures were common, occurring on average four times per 48 hour block. These node failures were out of the author's control and not due to model instability or any other failure of the CM1 model. When a node failure occurred, a message such as the following would appear in the CM1 standard output file:

```
[NID 19080] Apid 78797135 killed.  Received node event ec_node_failed for nid 19075
```

Consultation with Blue Waters support staff indicated that these node failures occurred on the Opteron processors with a return value of `COMPUTE_UNIT_DATA` which indicates an uncorrectable error occurring on one of the node's CPUs. Recognizing the regularity of these types of unavoidable errors, the author chose to save checkpoint files every 10 model seconds, and, via a loop within the PBS script file, restart the model from the most recent checkpoint file automatically such that the entire requested reservation time could be used. This required the allocation of a handful of extra nodes for each job submission as a reserve such that there were enough healthy nodes available for each execution, as the failed nodes would be removed from the pool. With these issues in mind, it is estimated that about a quarter of the utilized node hours on the machine were "wasted" due to these failures.

Because it was not known whether the simulation would produce a long track EF5 tornado similar to [17], full domain data was saved only every 10 s until the beginning of tornado maintenance was observed. Then, using one of the saved checkpoint files, the model was restarted from a time approximately 10 minutes prior to tornadogenesis and subsequent data was saved over a smaller subdomain with a save interval of 0.2 s until tornado decay occurred. The 10 s full-domain data, saved between $t = 1800.0$ s and $t = 4990$ s comprised 83 TB of LOFS data, while the subdomain data saved every 0.2 s from $t = 5000.2$ s and $t = 7490$ s weighed in at 187 TB.

4.2. A first look at tornadogenesis

Here a description of the process of tornadogenesis is provided, focusing primarily on the growth of near-ground vorticity. The model times of the figures following in this section were chosen after creating a video animation of the 0.2 s data and stepping forwards and backwards through the video until key moments in the simulation were identified. It should be emphasized that only a topical analysis of tornadogenesis is provided in this section, and that a more exhaustive quantitative analysis is beyond the scope of this paper. The supplemental video file SupV1 of the vorticity magnitude field and surface maximum vertical vorticity swaths from $t = 5000$ s through $t = 5390$ s is found at http://orf.media/Atmosphere2019. This video sequence begins approximately three minutes prior to tornadogenesis and continues through about three minutes of tornado maintenance, with frames displayed every 0.2 s at 30 frames per second.

Figures 4–6 provide an overview of cyclonic vortex paths, superimposed upon density potential temperature perturbation (θ'_ρ), to provide a context for the more information-dense three-dimensional images in subsequent figures (these images originated from the output of the `ncview` [41] utility, read from NetCDF files created with `lofs2nc`). The bulk of surface vortices identified by swaths of maximum vertical vorticity (ζ) shown in Figs. 4–9 originate on the cold (west) side of what [42] call the Left Flank Convergence Boundary (LFCB) (see their Fig. 8). This boundary is clearly visible in both the surface horizontal wind and θ'_ρ fields, with its location along the sharp θ'_ρ gradient noted in Figs 4–6. At $t = 5000.2$ s, approximately three minutes prior to tornadogensis, vortex path directions

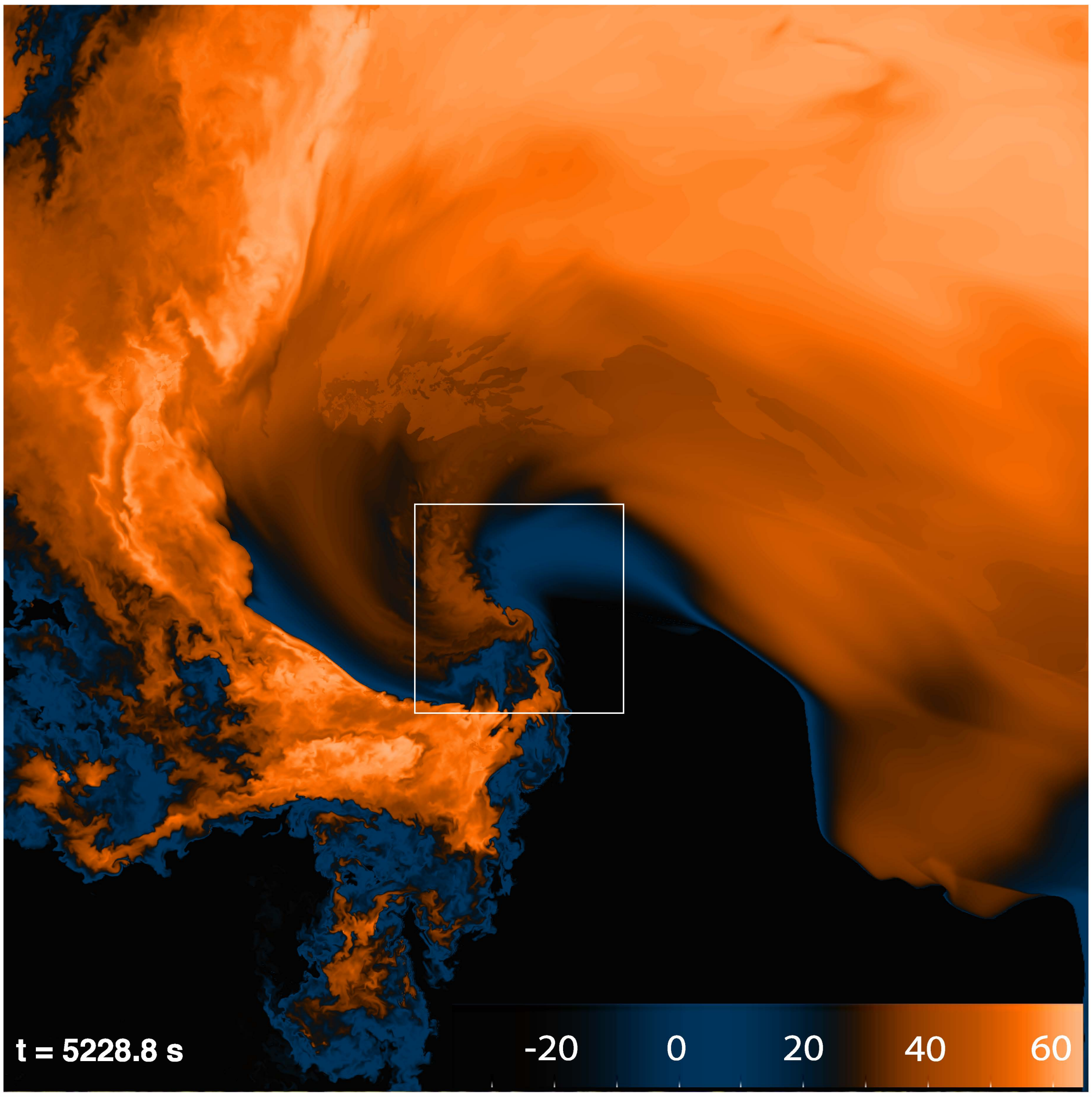


Figure 3. Simulated radar reflectivity in dBZ, 500 m AGL at $t = 5228.8$ s, created with `ncview`. The displayed horizontal domain of 20.8 km × 20.8 km represents what was saved across 676 files in 0.2 s intervals. White box inset of 4 km × 4 km covers shown horizontal extent in Figs 4–6. The hook echo in the reflectivity field in the center of the inset box corresponds to vortex **A**.

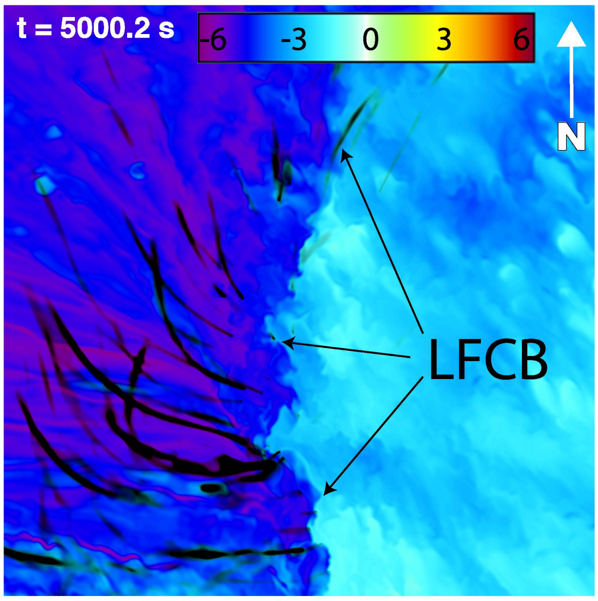


Figure 4. Density potential temperature perturbation (θ'_ρ) in K at $t = 5000.2\,\mathrm{s}$. Horizontal domain covers $4\,\mathrm{km} \times 4\,\mathrm{km}$. Black swaths are the ground-relative paths of cyclonic vortices at the model's lowest vertical level (motion is generally southward to eastward). The sharp θ'_ρ boundary representing the Left Front Convergence Boundary (LFCB) is noted. The color map indicates values of θ'_ρ in K.

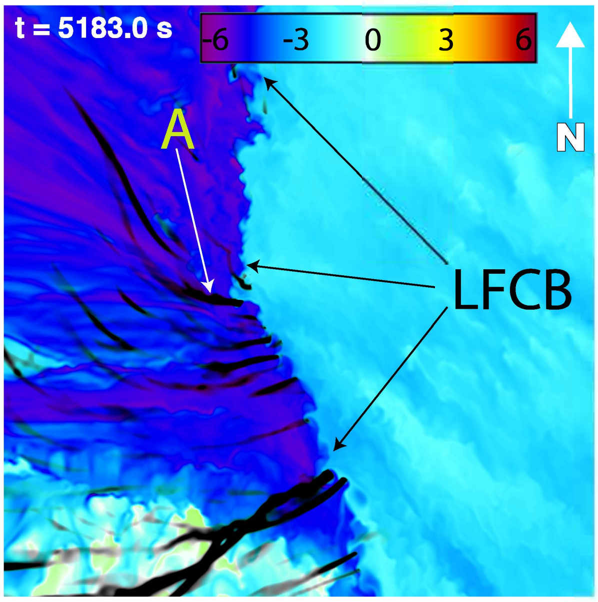


Figure 5. As in Fig. 4 but at $t = 5183.0\,\mathrm{s}$. Vortex **A** (shown in subsequent figures) is noted.

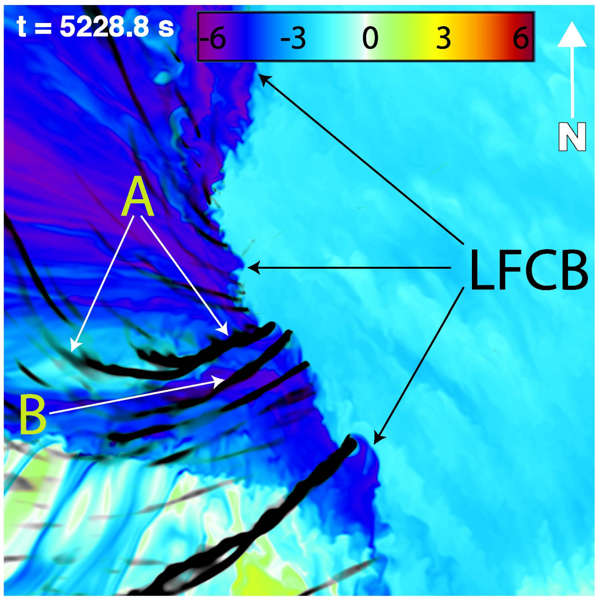


Figure 6. As in Fig. 5 but at $t = 5228.8$ s, corresponding to the same time in Fig. 3.

range from southward to eastward, with many paths abruptly turning from southward to eastward (see Fig. 4) along their respective paths. Three minutes later, at $t = 5183.0$ s, the incipient vortex that "becomes" the tornado (tagged to as vortex **A** in subsequent figures) has turned from southeastward to eastward and has begun to produce a 20 m swath of instantaneous ground-relative winds just exceeding 39 m s^{-1}, the EF1 threshold. The direction of vortex paths to the south of vortex **A** seen in Fig. 5 exhibit a slight to moderate northward component, indicating a convergence of vortices occurring in its direct vicinity. At this time, vortex **A** is located along an inflection point in the LFCB, which is oriented directly north/south to its north, and towards the SSE to its south. The reorientation of the LFCB is seen to persist in Fig. 6, 46 s later, with the LFCB showing an eastward bulge consistent with an increase in eastward momentum to the south of vortex **A**, which is now exhibiting a 40 m swath of EF2 strength winds (averaging 55 m s^{-1}). A second vortex, vortex **B** noted in Fig. 6, follows a convergent path with vortex **A** and is discussed below.

Figures 7–9 provide six snapshots in time, the first two of which overlap with Figs. 5 and 6. These images were created with the NCAR's VAPOR3 software [40], which can read NetCDF files containing 2D and 3D floating point data natively. The left panel of these images contains three-dimensional volume rendered vorticity magnitude with a visible threshold value of approximately 1.25 s^{-1}, along with surface maximum ζ swaths that can be matched with those in Figs 5 and 6. SupV1 corresponds to the left panels of these figures, while the right panels present a different viewing angle, and include, in addition to surface maximum ζ swaths, the liquid cloud water mixing ratio (qc) field to show the behavior of the condensation funnels associated with strong rotation. In Fig. 7a, vortex **A** can be seen extending upwards from the ground but is not associated with a visible condensation funnel at this time. In Fig. 7b, 45 s later, vortex **A** has strengthened at the surface and extended upwards to a height of nearly 3 km. A second vortex, vortex **B**, is also visible, having formed in a similar "bottom-up" manner as vortex **A**, and is in the process of being assimilated into the circulation of vortex **A**. Both vortices are associated with visible condensation funnels also seen in Fig. 7a. Figure 8a shows the same fields 44 s later, where vortex **A** is exhibiting a 50 m wide swath of EF3 strength surface winds (averaging 70 m s^{-1}) with another vortex, vortex **C**, having just swept behind the path of vortex **A** and extending upwards to a height of approximately 1 km. Vortex **A** is now "the tornado" as is evidenced by its co-location with a columnar condensation funnel that extends from the parent supercell cloud base to the ground. Over the next 12 s, vortex **C** has grown upward to a height of exceeding 2 km and has begun a process of merging and wrapping into vortex **A** (see Fig. 8b). This process is shown most

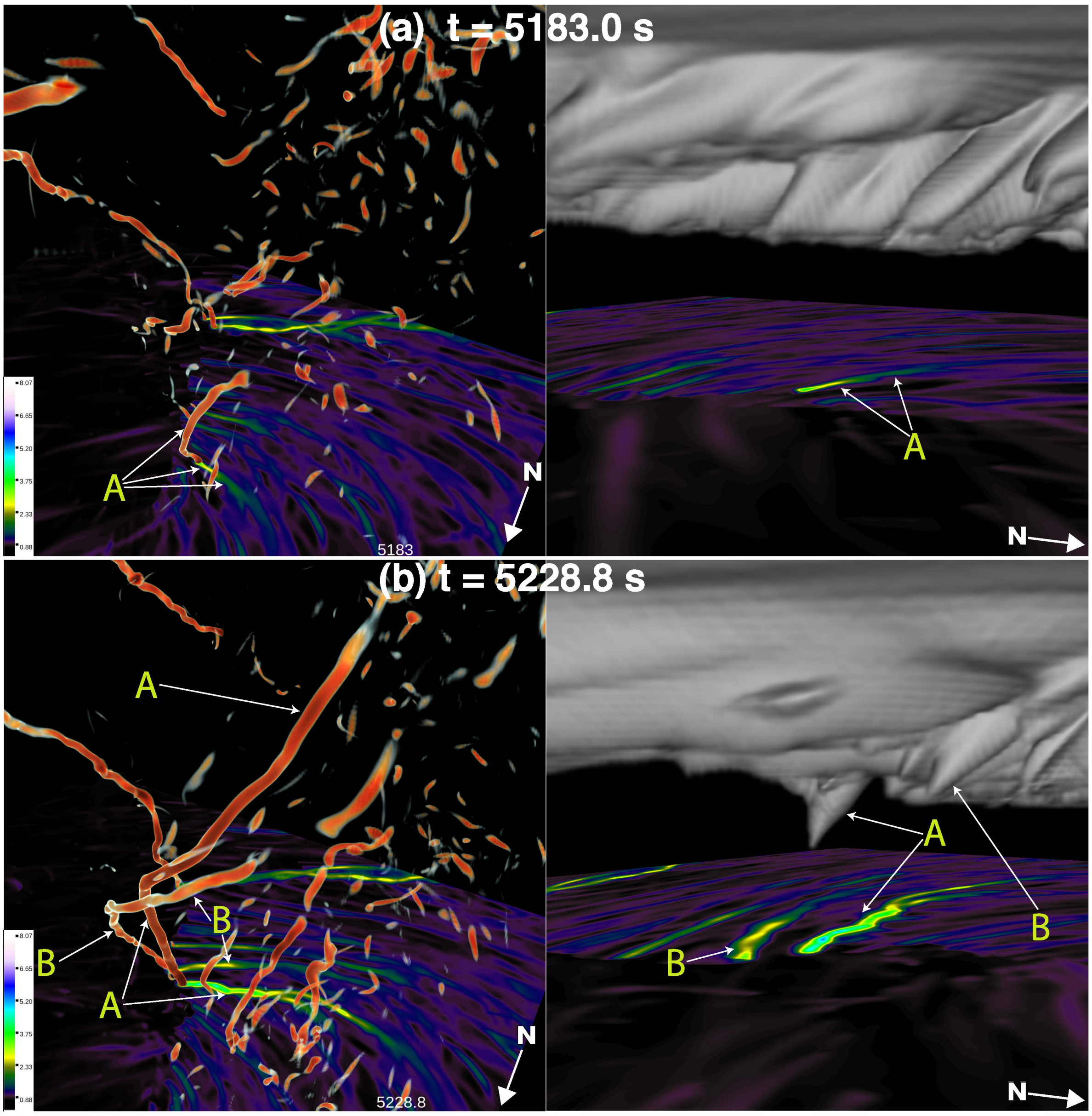


Figure 7. Left column: Three-dimensional vorticity magnitude (volume rendered field, visible threshold at about 1.2 s^{-1}) and surface maximum ζ swaths, representing the paths of cyclonic vortices at the model's lowest vertical level. The domain of the 3D volume spans 4 km $\times$ 4 km $\times$ 3 km in x, y, z. The color map references surface ζ values in s^{-1}. Right column: Cloud water mixing ratio (volume rendered field) and surface maximum ζ swaths. Vortex **A** and vortex **B** and their paths are annotated at (a) $t = 5183.0$ s and (b) $t = 5228.0$ s, which correspond to the times in Figs. 5 and 6. Images were created with VAPOR3 [40].

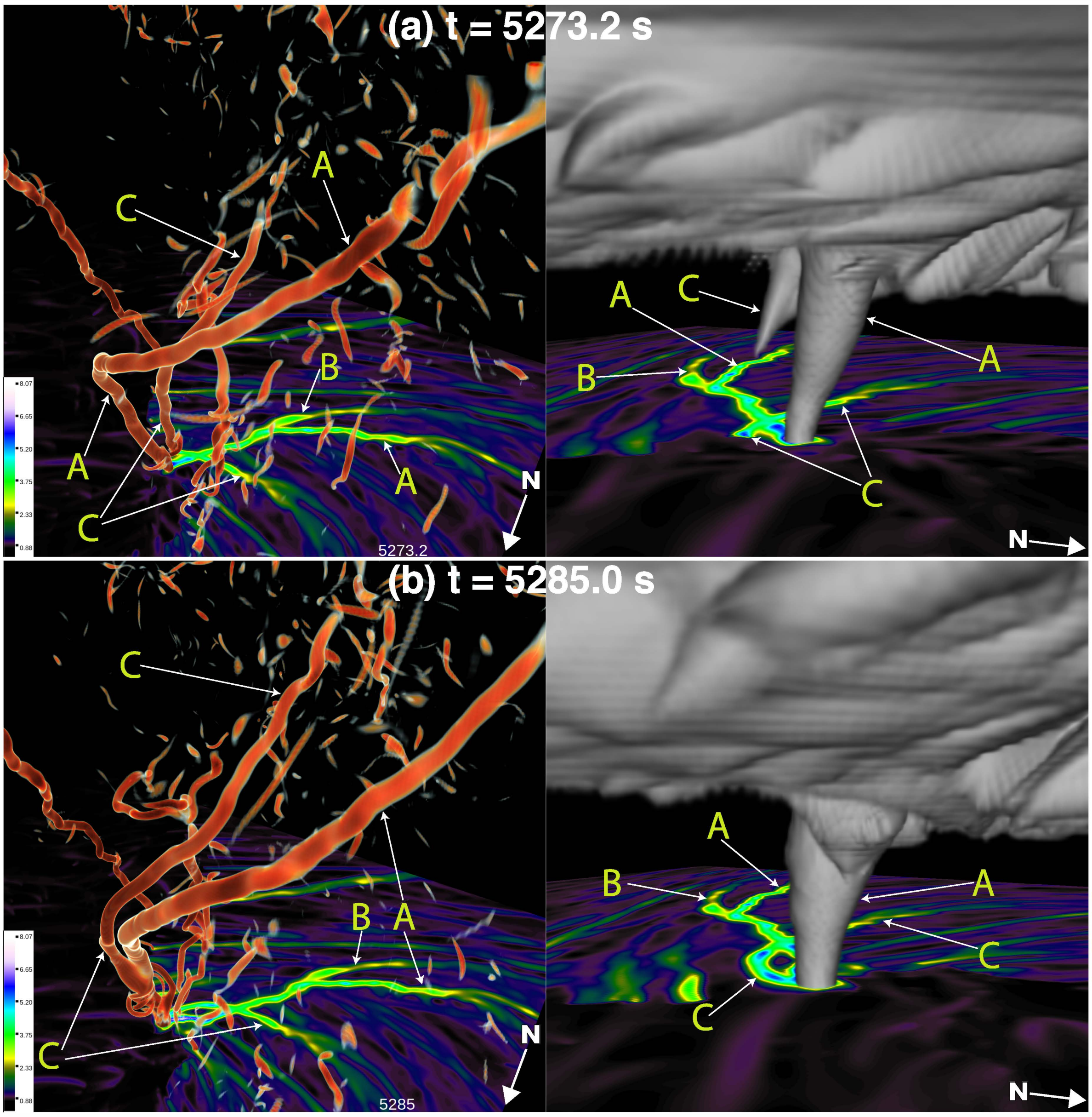


Figure 8. As in Fig. 7, but at (a) $t = 5273.2$ s and (b) $t = 5285.0$ s. Vortex **C**, which has begun to wrap around vortex **A**, is also noted.

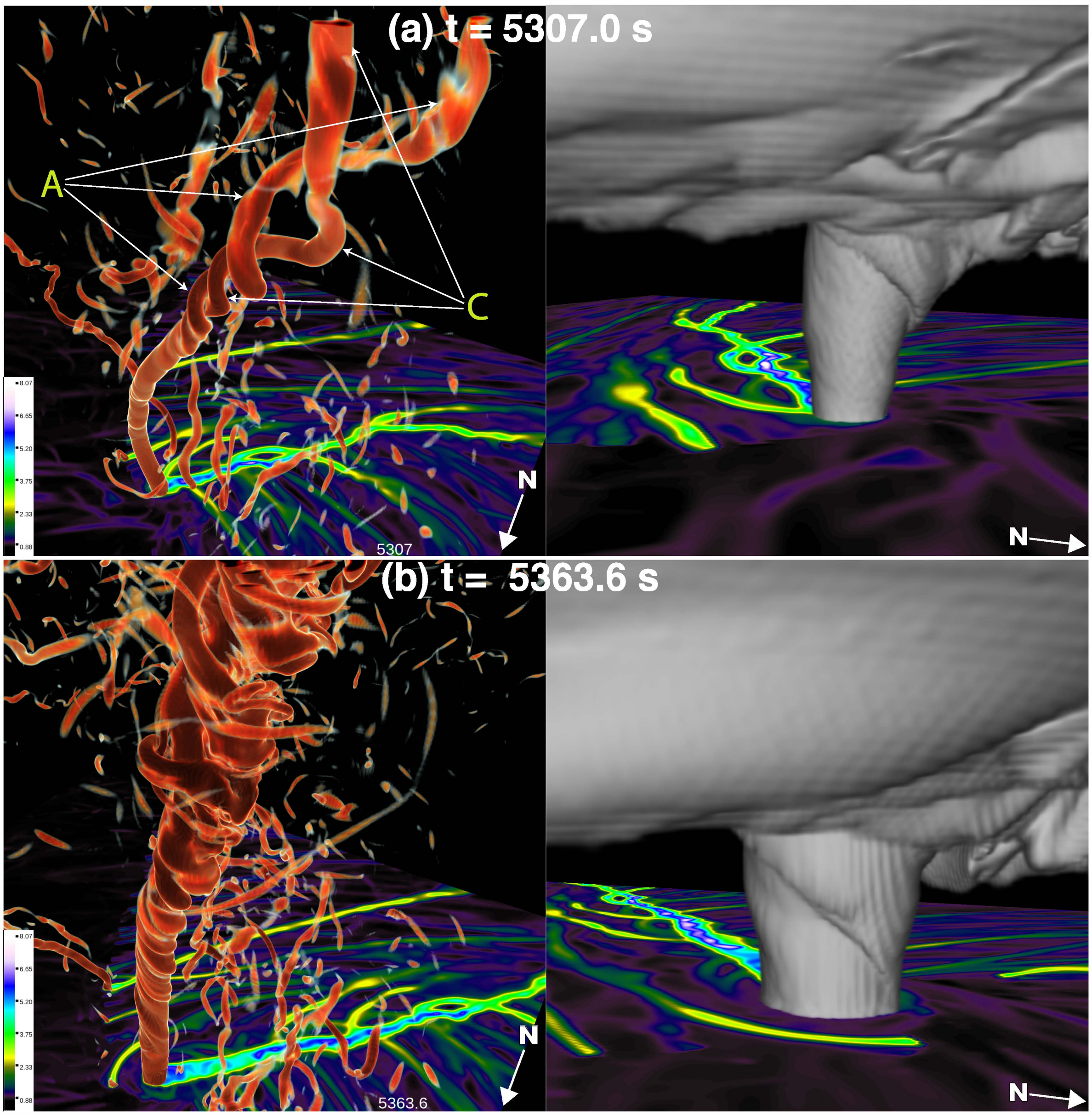


Figure 9. As in Fig. 8, but at (a) $t = 5307.0\,\mathrm{s}$ and (b) $t = 5363.6\,\mathrm{s}$. The wrapping of vortex **C** into vortex **A** has reached a height of about 2.5 km by $t = 5307\,\mathrm{s}$, and at $t = 5363.6\,\mathrm{s}$, the tornado, exhibiting ground-relative instantaneous winds exceeding $140\,\mathrm{m\,s^{-1}}$, is in the process of transitioning into a multiple-vortex structure and has become nearly vertically erect below 3 km, moving in a linear fashion towards the northeast.

clearly in SupV1 where the two vortices wrap around one another in a "bottom-up" manner. While mergers of helical vortices in three dimensions have been observed and modeled [43,44], the author is unaware that this phenomenon has ever been seen in an observed supercell or in simulations of a supercell. By $t = 5307$ s, when the wrapping process has extended to a height of approximately 2.5 km, the tornado is exhibiting EF5 surface winds exceeding 105 m s^{-1} (see Fig. 9a). A minute later, the tornado's diameter has widened, as is evident from the size of the visible condensation funnel and maximum ζ swath field, and has also assumed a more vertically erect position, and is moving in a linear fashion towards the northeast. Maximum instantaneous surface winds exceeding 140 m s^{-1} are found at this time, shortly before the tornado obtains a multiple vortex structure.

5. Discussion and Future Work

In this paper, LOFS, a simple file system designed for saving large amounts of massively parallel cloud model data efficiently, was described, along with a use example of a the `lofs2nc` utility for converting LOFS data to the widely-read NetCDF format. NetCDF files created from `lofs2nc` by the author were then read and displayed with `ncview` [41] (Figs. 3–6) and VAPOR3 [40] where 3D images were created (Figs. 7–9) and saved to disk to be statically viewed as well as animated (SupV1). The author's experience with running the simulation on 19,600 Blue Waters nodes (672,200 cores), spanning 87% of the supercomputer's XE nodes, was described, which included a process for recovering from node failures automatically by restarting the model from the most recent checkpoint files. The performance of dynamic ZFP lossy floating point compression was described for six 3D fields saved by CM1, with each variable having maximum accuracy specified at runtime. The use of ZFP and the choice of saving a small subdomain, focused on the low level mesocyclone, allowed data to be saved every 0.2 s, and this I/O approach took up a reasonable 37% of the model's execution time. A total of 270 TB of data was saved, with 83 TB of full domain data saved in 10 s intervals from $t = 1800$ s through $t = 4990$ s, and the remaining 187 TB of data saved from $t = 5000.2$ s and $t = 7490$ s over a 20.8 km by 20.8 km by 5 km subdomain every 0.2 s.

Visualizations of the vorticity and cloud field reveal a process of tornadogenesis characterized by the convergence, merging, and upward growth of several near-ground vortices, one of which wraps around the nascent tornado in a bottom-up fashion. The tornado forms along the Left Front Convergence Boundary [42] which is prominently displayed as a sharp buoyancy gradient in the cold pool. A few minutes prior to tornadogenesis, vortices on the cold side of this boundary take a sharp turn towards their left (moving southward to eastward) and then begin to travel towards the north east. Genesis occurs shortly thereafter, with the tornado forming along an inflection point in the LFCB, which has surged forward to the south of the developing tornado. While only a topical discussion of tornadogenesis, focusing on vorticity, has been presented, the value of using 3D volume rendering software such as VAPOR3 [40] has been clearly demonstrated, with animations of vorticity at full 0.2 s frame spacing telling a compelling story. Only one aspect of the tornadogenesis process has been presented here, and only over a short span of time; the tornado lasts 42 minutes and transitions to a wide, multiple vortex tornado that eventually becomes occluded before dissipating.

Much future work needs to be completed in order to quantify the complex morphology of the simulation. Such work will include:

- The use of temporal averaging to "smooth out" the details of the simulation in order to focus on underlying, steady forcing prior to and during tornadogenesis
- Examining the momentum and pressure characteristics of the updraft prior to and during tornado formation.
- Conducting Lagrangian parcel analysis to explore the forces acting on parcels in the vicinity of the tornado
- Exploring the sensitivity of the simulation to a turbulence kinetic energy closure by conducting a second run using the Smagorinsky [39] closure scheme, which was used in a 30 m resolution simulation of the same storm [17].

Work is already underway to develop Lagrangian parcel tracking code, using LOFS data as input, optimized for graphical processing units (GPUs) that will enable the tracking of millions of parcels throughout chosen segments of the simulation. Code has already been created to create the temporally averaged fields, the utility of which has been shown [45] with the 30 m resolution simulation of [17].

LOFS read-side and write-side source code is available as supplementary material to this paper. The write-side code is released in isolation from the CM1 model; it is the author's intent to explore the possibility of releasing an LOFS-enabled branch of the latest version of the CM1 model on a repository such as Github such that other researchers can take advantage of LOFS. The LOFS read-side code is currently being refactored, and will be released on a public source code repository in the near future.

Funding: This work was supported by the Cooperative Institute for Meteorological Satellite Studies at the University of Wisconsin and the National Science Foundation via grants OAC-1614973, OAC-1663954, OAC-1550405, and AGS-1832327.

Acknowledgments: Work on LOFS can be traced back to the late 2000s when I was faculty in the Department of Earth and Atmospheric Sciences at Central Michigan University, and I would like to acknowledge the support from the College of Science and Technology (now Science and Engineering). I am very grateful for support from the Mesoscale and Microscale Meteorology group at NCAR, which hosted my academic sabbatical for eight months in 2010-2011 when much of the trial and error that led to early versions of what became LOFS was conducted. Since the early 2000's, Robert Wilhelmson provided me with encouragement and support for the high resolution supercell modeling work, and was the PI on the first Blue Waters NSF grant that provided me with early access to the machine where much LOFS development was done. Finally, I am very grateful for George Bryan's continuing work on the CM1 model and his willingness to make the source code available with abundant documentation.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Wilhelmson, R. The Life Cycle of a Thunderstorm in Three Dimensions. *J. Atmos. Sci.* **1974**, *31*, 1629–1651, [http://dx.doi.org/10.1175/1520-0469(1974)031<1629:TLCOAT>2.0.CO;2]. doi:10.1175/1520-0469(1974)031<1629:TLCOAT>2.0.CO;2.
2. Schlesinger, R.E. A Three-Dimensional Numerical Model of an Isolated Deep Convective Cloud: Preliminary Results. *J. Atmos. Sci.* **1975**, *32*, 934–957, [http://dx.doi.org/10.1175/1520-0469(1975)032<0934:ATDNMO>2.0.CO;2]. doi:10.1175/1520-0469(1975)032<0934:ATDNMO>2.0.CO;2.
3. Klemp, J.B.; Wilhelmson, R.B. The Simulation of Three-Dimensional Convective Storm Dynamics. *J. Atmos. Sci.* **1978**, *35*, 1070–1096. doi:10.1175/1520-0469(1978)035<1070:TSOTDC>2.0.CO;2.
4. Wicker, L.J.; Wilhelmson, R.B. Simulation and Analysis of Tornado Development and Decay within a Three-Dimensional Supercell Thunderstorm. *J. Atmos. Sci.* **1995**, *52*, 2675–2703. doi:10.1175/1520-0469(1995)052<2675:SAAOTD>2.0.CO;2.
5. Grasso, L.D.; Cotton, W.R. Numerical Simulation of a Tornado Vortex. *J. Atmos. Sci.* **1995**, *52*, 1192–1203. doi:10.1175/1520-0469(1995)052<1192:NSOATV>2.0.CO;2.
6. Finley, C.A.; Cotton, W.R.; Pielke, R.A. Numerical Simulation of Tornadogenesis in a High-Precipitation Supercell. Part I: Storm Evolution and Transition into a Bow Echo. *J. Atmos. Sci.* **2001**, *58*, 1597–1629, [http://dx.doi.org/10.1175/1520-0469(2001)058<1597:NSOTIA>2.0.CO;2]. doi:10.1175/1520-0469(2001)058<1597:NSOTIA>2.0.CO;2.
7. Lee, B.D.; Wilhelmson, R.B. The Numerical Simulation of Nonsupercell Tornadogenesis. Part II: Evolution of a Family of Tornadoes along a Weak Outflow Boundary. *J. Atmos. Sci.* **1997**, *54*, 2387–2415. doi:10.1175/1520-0469(1997)054<2387:TNSONT>2.0.CO;2.
8. Xue, M. Tornadogenesis within a Simulated Supercell Storm. 22nd Severe Local Storms Conference. Hyannis, MA, Amer. Meteor. Soc., 9.6 [Available online at https://ams.confex.com/ams/11aram22sls/techprogram/paper_81574.htm], 2004.
9. Noda, A.T.; Niino, H. Genesis and Structure of a Major Tornado in a Numerically-Simulated Supercell Storm: Importance of Vertical Vorticity in a Gust Front. *SOLAIAT* **2005**, *1*, 5–8. doi:10.2151/sola.2005-002.
10. Noda, A.T.; Niino, H. A Numerical Investigation of a Supercell Tornado: Genesis and Vorticity Budget. *Journal of the Meteorological Society of Japan. Ser. II* **2010**, *88*, 135–159. doi:10.2151/jmsj.2010-203.

11. Roberts, B.; Xue, M.; Schenkman, A.D.; Dawson, D.T. The Role of Surface Drag in Tornadogenesis within an Idealized Supercell Simulation. *J. Atmos. Sci.* **2016**, *73*, 3371–3395, [http://dx.doi.org/10.1175/JAS-D-15-0332.1]. doi:10.1175/JAS-D-15-0332.1.
12. Mashiko, W.; Niino, H. Super High-Resolution Simulation of the 6 May 2012 Tsukuba Supercell Tornado: Near-Surface Structure and Its Evolution. *SOLAIAT* **2017**, *13*, 135–139. doi:10.2151/sola.2017-025.
13. Yao, D.; Xue, H.; Yin, J.; Sun, J.; Liang, X.; Guo, J. Investigation into the Formation, Structure, and Evolution of an EF4 Tornado in East China Using a High-Resolution Numerical Simulation. *J. Meteorol. Res.* **2018**, *32*, 157–171. doi:10.1007/s13351-018-7083-0.
14. Snyder, J.C.; Bluestein, H.B.; Venkatesh, V.; Frasier, S.J. Observations of Polarimetric Signatures in Supercells by an X-Band Mobile Doppler Radar. *Mon. Weather Rev.* **2013**, *141*, 3–29. doi:10.1175/MWR-D-12-00068.1.
15. Wurman, J.; Kosiba, K. Finescale Radar Observations of Tornado and Mesocyclone Structures. *Weather Forecast.* **2013**, *28*, 1157–1174. doi:10.1175/WAF-D-12-00127.1.
16. Wurman, J.; Kosiba, K.; Robinson, P.; Marshall, T. The Role of Multiple-Vortex Tornado Structure in Causing Storm Researcher Fatalities. *Bull. Am. Meteorol. Soc.* **2014**, *95*, 31–45. doi:10.1175/BAMS-D-13-00221.1.
17. Orf, L.; Wilhelmson, R.; Lee, B.; Finley, C.; Houston, A. Evolution of a Long-Track Violent Tornado within a Simulated Supercell. *Bull. Am. Meteorol. Soc.* **2017**, *98*, 45–68, [http://dx.doi.org/10.1175/BAMS-D-15-00073.1]. doi:10.1175/BAMS-D-15-00073.1.
18. Nishizawa, S.; Yashiro, H.; Sato, Y.; Miyamoto, Y.; Tomita, H. Influence of grid aspect ratio on planetary boundary layer turbulence in large-eddy simulations. *Geoscientific Model Development* **2015**, *8*, 3393–3419. doi:10.5194/gmd-8-3393-2015.
19. Bode, B.; Butler, M.; Dunning, T.; Hoefler, T.; Kramer, W.; Gropp, W.; Hwu, W.M. In *Contemporary High Performance Computing From Petascale Towards Exascale*; Chapman & Hall/CRC Computational Science, Chapman and Hall/CRC, 2013; chapter The Blue Water Super-System for Super-Science, pp. 339–366. doi:10.1201/b14677-16.
20. Kramer, W.; Butler, M.; Bauer, G.; Chadalavada, K.; Mendes, C. In *High-Performance Parallel I/O*; Chapman & Hall/CRC Computational Science, Chapman and Hall/CRC, 2014; chapter National Center for Supercomputing Applications, pp. 17–31. doi:10.1201/b17572-5.
21. Bauer, A.C.; Abbasi, H.; Ahrens, J.; Childs, H.; Geveci, B.; Klasky, S.; Moreland, K.; O'Leary, P.; Vishwanath, V.; Whitlock, B.; Bethel, E.W. In Situ Methods, Infrastructures, and Applications on High Performance Computing Platforms. *Comput. Graph. Forum* **2016**, *35*, 577–597. doi:10.1111/cgf.12930.
22. Bryan, G.H.; Fritsch, J.M. A Benchmark Simulation for Moist Nonhydrostatic Numerical Models. *Mon. Weather Rev.* **2002**, *130*, 2917–2928. doi:10.1175/1520-0493(2002)130<2917:ABSFMN>2.0.CO;2.
23. Bryan, G. CM1 Homepage, 2019. Accessed: 2019-8-31.
24. The HDF Group. Hierarchical Data Format, version 5, 1997-2019. http://www.hdfgroup.org/HDF5/.
25. Lindstrom, P. Fixed-Rate Compressed Floating-Point Arrays. *IEEE Trans. Vis. Comput. Graph.* **2014**, *20*, 2674–2683. doi:10.1109/TVCG.2014.2346458.
26. Unidata. Network Common Data Form (netCDF) version 4 [software]., 2019. Boulder, CO: UCAR/Unidata. http://doi.org/10.5065/D6H70CW6.
27. Gropp, W.; Lusk, E. A high-performance MPI implementation on a shared-memory vector supercomputer. *Parallel Comput.* **1997**, *22*, 1513–1526. doi:10.1016/S0167-8191(96)00062-2.
28. Dorier, M.; Antoniu, G.; Cappello, F.; Snir, M.; Orf, L. Damaris: How to Efficiently Leverage Multicore Parallelism to Achieve Scalable, Jitter-free I/O. Cluster Computing (CLUSTER), 2012 IEEE International Conference on; IEEE Computer Society: Los Alamitos, CA, USA, 2012; Vol. 0, pp. 155–163. doi:10.1109/CLUSTER.2012.26.
29. Dorier, M.; Antoniu, G.; Cappello, F.; Snir, M.; Sisneros, R.; Yildiz, O.; Ibrahim, S.; Peterka, T.; Orf, L. Damaris: Addressing Performance Variability in Data Management for Post-Petascale Simulations. *ACM Trans. Parallel Comput.* **2016**, *3*, 15:1–15:43. doi:10.1145/2987371.
30. The HDF Group - Information, Support, and Software: File Drivers Defined in HDF5. https://support.hdfgroup.org/HDF5/Tutor/filedrvr.html#predef, 2019. Accessed: 2019-8-31.
31. Liu, N.; Cope, J.; Carns, P.; Carothers, C.; Ross, R.; Grider, G.; Crume, A.; Maltzahn, C. On the role of burst buffers in leadership-class storage systems. 2012 IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST), 2012, pp. 1–11. doi:10.1109/MSST.2012.6232369.

32. Johansen, G. Managing Cray XT MPI runtime environment variables to optimize and scale applications. Proceedings of the Cray User Group 2008 Meeting (CUG2008), 2008. https://cug.org/5-publications/proceedings_attendee_lists/2008CD/S08_Proceedings/pages/Authors/01-5Monday/Johansen-Monday3C/Johansen-Monday3C-slides.pdf.
33. Jing Fu.; Ning Liu.; Sahni, O.; Jansen, K.E.; Shephard, M.S.; Carothers, C.D. Scalable parallel I/O alternatives for massively parallel partitioned solver systems. 2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010, pp. 1–8. doi:10.1109/IPDPSW.2010.5470887.
34. Multiple Independent File (MIF, aka N:M) Parallel I/O With HDF5 - The HDF Group. https://www.hdfgroup.org/2017/03/mif-parallel-io-with-hdf5/, 2017. Accessed: 2019-8-31.
35. Eaton, B.; Gregory, J.; Drach, B.; Taylor, K.; Hankin, S.; Caron, J.; Signell, R.; Bentley, P.; Rappa, G.; Höck, H.; others. NetCDF Climate and Forecast (CF) metadata conventions, 2003. Accessed: 2019-09-01. http://cfconventions.org/cf-conventions/v1.6.0/cf-conventions.pdf.
36. Arakawa, A.; Lamb, V.R. A Potential Enstrophy and Energy Conserving Scheme for the Shallow Water Equations. *Mon. Weather Rev.* **1981**, *109*, 18–36. doi:10.1175/1520-0493(1981)109<0018:APEAEC>2.0.CO;2.
37. Tukey, J.W. *Exploratory data analysis*; Addison-Wesley, 1977.
38. Deardorff, J.W. A numerical study of three-dimensional turbulent channel flow at large Reynolds numbers. *J. Fluid Mech.* **1970**, *41*, 453–480. doi:10.1017/S0022112070000691.
39. Smagorinsky, J. General circulation experiments with the primitive equations. *Mon. Weather Rev.* **1963**, *91*, 99–164. doi:10.1175/1520-0493(1963)091<0099:GCEWTP>2.3.CO;2.
40. Li, S.; Jaroszynski, S.; Pearse, S.; Orf, L.; Clyne, J. VAPOR: A Visualization Package Tailored to Analyze Simulation Data in Earth System Science. *Atmosphere* **2019**, *10*, 488. doi:10.3390/atmos10090488.
41. Pierce, D.W. Ncview. http://meteora.ucsd.edu/~pierce/ncview_home_page.html. Accessed: 2019-8-31.
42. Beck, J.; Weiss, C. An Assessment of Low-Level Baroclinity and Vorticity within a Simulated Supercell. *Mon. Weather Rev.* **2013**, *141*, 649–669. doi:10.1175/MWR-D-11-00115.1.
43. Devenport, W.J.; Vogel, C.M.; Zsoldos, J.S. Flow structure produced by the interaction and merger of a pair of co-rotating wing-tip vortices. *J. Fluid Mech.* **1999**, *394*, 357–377. doi:10.1017/S0022112099005777.
44. Delbende, I.; Piton, B.; Rossi, M. Merging of two helical vortices. *Eur. J. Mech. B. Fluids* **2015**, *49*, 363–372. doi:10.1016/j.euromechflu.2014.04.005.
45. Orf, L. The Role of the Streamwise Vorticity Current in Tornado Genesis and Maintenance. 29th Conference on Severe Local Storms. AMS, 2018. https://youtu.be/yMHsip9DAUY?t=241.