Parallel I/O for 3-D Global FDTD Earth–Ionosphere Waveguide Models at Resolutions on the Order of ∼1 km and Higher Using HDF5

Santosh Pokhrel, *Student Member, IEEE*, Miguel Rodriguez, Alireza Samimi, Gerd Heber, and Jamesina J. Simpson, *Senior Member, IEEE*

***Abstract*— 3-D finite-difference time-domain (FDTD) models of the global Earth–ionosphere waveguide have reached grid resolutions on the order of 1 × 1 × 1 km and higher. At these high resolutions, efficient parallel input–output (I/O) schemes are required to keep the simulation run times reasonable. However, generating optimal parallel I/O schemes is challenging due to the spherical-coordinate global mesh that includes merging of cells in the east–west direction as either pole is approached. Hierarchical Data Format 5 (HDF5) is applied and optimized for a global latitude–longitude FDTD model. The performance of the parallel HDF5 approach is compared with a message passing interface-based approach and a brute-force approach. The methodology presented here may be simplified to Cartesian-based grids in a straightforward manner.**

***Index Terms*— Earth–ionosphere waveguide, finite-difference time-domain (FDTD), Hierarchical Data Format 5 (HDF5), high-resolution models, message passing interface (MPI).**

I. Introduction

THE availability of petascale supercomputers in the recent years has presented the opportunity to run finite-difference time-domain (FDTD) [1] models of electromagnetic waves propagating in the global Earth–ionosphere waveguide [2]–[7] at resolutions on the order of ∼1 km and higher [4]. At these resolutions, propagation of electromagnetic waves at frequencies on the order of 10 kHz may be simulated on a global scale while taking into account magnetized ionospheric physics.

The global FDTD models of the Earth–ionosphere waveguide are efficiently parallelized on supercomputers by using message passing interface (MPI) for communicating between processors [4]. However, in order to run the models at such high resolutions, input data must be distributed to all of the processing cores, such as the earth's topography (several gigabytes at 1 km resolution), the composition of the ionosphere, and the geomagnetic field. Furthermore, the following output data are generally of interest: sampled electric and magnetic fields over time at specific locations of interest, spatial snapshots of the grid at specific time steps, and the values of all of the electromagnetic field components within the entire 3-D computational domain when the simulation must be restarted (such as when the wallclock time limit of the supercomputer is reached before the simulation is completed). However, the global FDTD model in [4] is implemented in spherical coordinates and includes merging of cells in the east–west direction as either pole is approached. This complicates how the input–output (I/O) is performed compared to that for a regular Cartesian mesh.

There are generally three ways to read and write the field components of a 3-D grid distributed across a supercomputer (e.g., a 2-D slice of electric field components).

1) *Use MPI_BROADCAST (for Input) and MPI_REDUCE (for Output):* In this case, a data file can be uploaded to a single processor, and this processor can broadcast the data from the file to all the remaining processors using MPI_BROADCAST [8]. Each receiving processor can extract the section of the data corresponding to the FDTD subgrid assigned to that processor. For output, the data from all of the corresponding processors are sent (reduced) to a single processor using MPI_REDUCE [8] along with some arithmetic operations, such as MPI_SUM. The processor receiving the data writes all of the data to an output file. Unfortunately, for very

Manuscript received May 22, 2017; revised February 6, 2018; accepted April 16, 2018. Date of publication May 10, 2018; date of current version July 3, 2018. This work was supported in part by the National Science Foundation through the Blue Waters sustained-petascale computing project under Award OCI-0725070 and Award ACI-1238993, in part by the state of Illinois, in part by the National Science foundation through the PRAC allocation support under Award CSA OCI-0725070, and in part by NSF under Award 1662318. Blue Waters is a joint effort of the University of Illinois at Urbana–Champaign and its National Center for Supercomputing Applications. *(Corresponding author: Santosh Pokhrel.)*

S. Pokhrel is with the Electrical Engineering Department, University of Utah, Salt Lake City, UT 84112 USA (e-mail: santosh_everest@yahoo.com).

M. Rodriguez is with the University of Utah, Salt Lake City, UT 84112 USA (e-mail: xyzmiguel@hotmail.com).

A. Samimi is with Nanometrics Inc., Milpitas, CA 95035 USA (e-mail: arsamimi450@gmail.com).

G. Heber is with The HDF Group, University of Illinois Research Park, University of Illinois at Urbana–Champaign, Champaign, IL 61820 USA (e-mail: gheber@hdfgroup.org).

J. J. Simpson is with the Electrical and Computer Engineering Department, University of Utah, Salt Lake City, UT 84112 USA (e-mail: jamesina.simpson@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TAP.2018.2835163

large grids, MPI_BROADCAST and MPI_REDUCE are infeasible due to the memory limitations of one processor or one node of a supercomputer (typically 4 GB).

2) *Have Each Processor Read–Write Its Own Data in Turn (a Brute-Force Approach):* In this case, all of the processors take turns sending the data in their subgrid to a single processor which then writes all of the data to a file. This procedure works for arbitrarily large grids, but for codes running on a large number of processors as in the focus of this paper, this method is too slow even for a 2-D slice of data (on the order of hours for each file).
3) *Employ Parallel I/O:* Parallel I/O, as the name implies, involves multiple processors writing data from their FDTD subgrid simultaneously to one or more files in a parallel file system such as Lustre or the general parallel file system. The advantages of parallel I/O are that it can be completed quickly even for large grids. Furthermore, special compression techniques can be utilized to generate smaller data files than regular output. However, it is not easily implemented in the global FDTD mesh.

The focus of this paper is on the third option mentioned earlier, parallel I/O, and specifically Hierarchical Data Format 5 (HDF5) [9]–[11] applied to the global FDTD model of [4]. HDF5 is chosen because of its wide availability and because of its effectiveness in storing and retrieving large quantities of scientific data. To the best of our knowledge, the implementation details and testing of HDF5 for FDTD models have not yet appeared in the literature. Here, HDF5 is optimized for FDTD, and furthermore, the unique implementation of HDF5 for the case of the nonstandard, spherical-coordinate global FDTD Earth–ionosphere models of [4] is provided for a high-resolution grid. In fact, the resolution is so high even a 2-D slice of data from the fully 3-D grid requires more memory than a single processor of a supercomputer typically has access to. The performance of HDF5 is compared with a MPI-based approach [above-mentioned 1)] as well as a brute-force approach [above-mentioned 2)]. The methodology of the HDF5 implementation discussed here may be extended to arbitrary FDTD grids in a straightforward manner.

Using HDF5, it is feasible to perform I/O operations on global FDTD models of the Earth–ionosphere waveguide at very high resolutions [4]. Initially, the application of global FDTD models was limited to the ultralow-frequency ($<$3 Hz) and extremely low frequency (3 Hz–3 kHz) ranges, since they were run at $40 \times 40 \times 5$ km resolution. However, the recent advances to these models have extended their applicability into the very low-frequency (VLF: 3–30 kHz) range, which requires grid resolutions on the order of $\sim$1 km and higher. These advances have included the development of more efficient magnetized ionospheric plasma FDTD algorithms [12]–[14] and more efficient grid parallelization schemes [4]. For example, global FDTD models are being used to develop a robust geolocation system based on propagating VLF waves to be used as a backup to satellite-based GPS. In this case, accurate phase information is needed for VLF signals propagating in the Earth–ionosphere waveguide; hence, the resulting required high grid resolution. Also, global models operating at VLF are helpful because in some scenarios, a stronger VLF signal may be detected over a long propagation path rather than the short propagation path between a transmitter and receiver. This phenomenon is better to study with a global model rather with a localized or "moving window" cylindrically symmetric FDTD model [15]. Other applications at VLF include communications and propagation from lightning and sprites.

Using HDF, it is also possible to perform I/O on global FDTD models extending farther up into the ionosphere. To date, global FDTD models have primarily been used to simulate up to an altitude of 110 km or so. However, especially for space weather applications, the global FDTD models would ideally extend all the way through the entire ionosphere and to the magnetosphere. Extending to such altitudes requires a large number of grid cells in 3-D, making the use of parallel I/O crucial.

A variety of other electromagnetic solvers [16]–[21] have also been parallelized and developed for high performance computing (HPC) platforms to date. For example, Garcia-Doñoro *et al.* [16] developed an electromagnetic code based on "higher order finite element method" [22] that is parallelized using MPI and open multiprocessing commands. Similarly, parallel "method of moment (MoM)" [23] codes using higher order basis functions and a parallel solver called target identification software have been developed [17]. The complexity of MoM analysis was reduced in [18] using domain decomposition. As another example, Richter *et al.* [19] used NVIDIA's CUDA to solve finite difference schemes. This demonstrated the ultrafast parallel capabilities of graphical processing units for an HPC environment. Likewise, CUDA has been successfully implemented in finite-element boundary-integral method models [20] and for multilevel fast multipole algorithm models for scattering problems [21]. In this paper, parallel I/O HDF is demonstrated for a global FDTD model. However, HDF could be adapted to other numerical approaches as well.

II. Distribution of Grid Cells Onto Processors

In the global FDTD model of [2] and [4], as either pole is approached, every two grid cells in the east–west direction are iteratively merged together in order to keep all the grid cells roughly the same size. For this reason, the grid cells in the Polar Regions are the most challenging to distribute between processors and to read–write to files.

Note that although the global FDTD models are fully 3-D, only 2-D horizontal (latitude–longitude) examples are provided in this paper. The methodology for a 2-D slice of data may be extended to the 3-D case in a straightforward manner, since there are no cell mergings in the radial (vertical) direction.

Fig. 1 is a zoomed-in view of the distribution of cells and processors around the North Pole. As discussed in [4], the interfaces between processors are strategically placed at latitudes just north (in the northern hemisphere) and South (in the southern hemisphere) of latitudes at which cell mergings are performed. This reduces the number of ghost cells [2], [4] in the model, which take up memory but do not hold any data

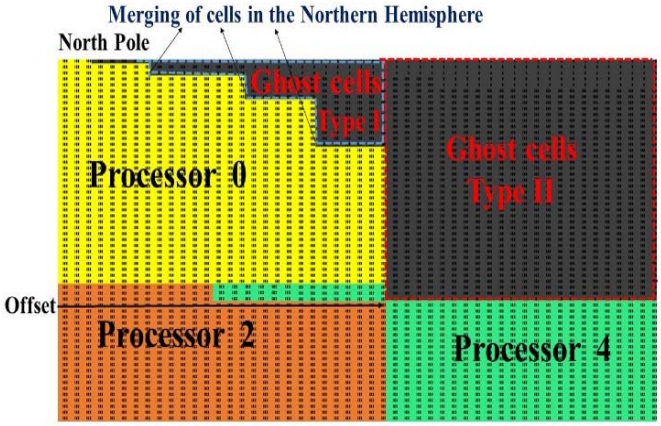

Fig. 1. Illustration of the grid cells immediately surrounding the North Pole, including both types of ghost cells corresponding to the data on processor 0. Yellow cells are on processor 0. Orange cells are on processor 2. Green cells are on processor 4. (Odd numbered processors correspond to the southern hemisphere).

used in the FDTD calculations. For detail explanation of ghost cells, see Appendix B.

The yellow, green, and orange grid cells of Fig. 1 represent the cells in the global mesh. The black cells in Fig. 1 represent ghost cells, which should not be read or written to files because they are not used in the FDTD computations. There are two types of ghost cells. In the case of Fig. 1, Type-I ghost cells are stored on processor 0 along with the FDTD grid cell data (yellow cells) because the grid cells on processor 0 (as well as all of the other processors) are stored in a rectangular matrix. Type-II ghost cells in Fig. 1 are not stored on any of the processors, but exist when a 2-D slice of global data is written to a file as a 2-D rectangular matrix, such as when using MPI_REDUCE.

To avoid reading–writing ghost cells with HDF5, field components are selected row by row and assembled into coarser "chunks," since the number of cells in the east–west direction changes often in the Polar Regions as shown in Fig. 1.

Writing data from each grid cell used in the FDTD computations on a row by row basis is not immediately straightforward because the grid cells are not all left-aligned within each processor's memory. The data in some rows are offset horizontally (i-index, or east–west direction). An example of this is seen in the bottom section of green grid cells of processor 4 in Fig. 1, which are not left-aligned with the green cells immediately above. The possible memory offset of the data within each processor and at each latitude must be accounted for when defining the HDF5 selections.

The reason for the horizontal offset is that grid cells are merged in the east–west direction at specific latitudes as seen in Fig. 1. Neighboring grid cell data must be shared between processors in the north–south direction in order to update grid cells at the boundary. Even if the processors are changed at the same latitude as a merging of cells, we have to share data from north to south (in the northern hemisphere) above the merging cell region in order to update all of the data below the merging of cells. This sharing of data is also true for Cartesian FDTD grids, which would not include merging of cells.

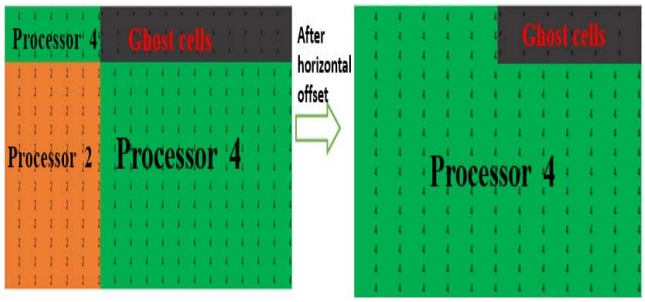

Fig. 2. Small rectangular chunk of grid cells (located on processor 4) extracted from the global distribution of grid cells as shown in Fig. 1 (left). Position of grid cells in memory on processor 4 after applying a horizontal offset in the first few rows of grid cells on processor 4 (right).

By using a horizontal offset, the memory (and 3-D arrays) on processor 4 is optimized (shown in Fig. 2).

III. I/O Using MPI Commands

An example output procedure performed using MPI commands is provided for the global model in order to give an overview of the process in a more simplified manner before describing the HDF5 approach. For this example, a 2-D slice of the radial electric field components, E_r, within the global 3-D grid are written to a file at a specific time step and for a specific k-index. Including "ghost" cells, the global E_r matrix extends in the east–west direction (i-index) from 1 to *imax*, in the north–south direction (j-index) from 1 to *jmax*, and in the radial/vertical direction (k-index) from 1 to *kmax* in the east–west. This global E_r matrix is subdivided onto each processor according to its processor number. The subgrids extend from *istart* to *iend*, *jstart* to *jend*, and *kstart* to *kend* (these parameters are separately defined on each processor). Note that *istart* and *iend* are a function of j (latitude) because the number of grid cells in the east–west directions change as either pole is approached from the equator.

To collect E_r data spread out among many processors, local and global versions of the E_r arrays are allocated and initialized to zero

```
1 allocate (Er_global (1:imax, 1:jmax))
2 allocate (Er_local (1:imax, 1:jmax))
3 Er_local(:,:) = 0.0
4 Er_global(:,:) = 0.0
```

Next, the *Er_local* array within each processor is populated with the desired 2-D slice of E_r values

```
5   Do j=jstart: jend
6       Do i=istart(j), iend(j)
7         Er_local (i, j) = Er (i, j, k)
8       Enddo
9   Enddo
```

Using the MPI command MPI_REDUCE [24], the *Er_local* values on all of the different processors are sent to the *Er_global* matrix on processor 1 (processor 1 is arbitrarily chosen)

```
10 CALL MPI_REDUCE (Er_local, Er_global, imax * jmax,
MPI_DOUBLE_PRECISION, MPI_SUM, 1, MPI_COMM_
WORLD, ierr)
```

MPI_DOUBLE_PRECISION, MPI_SUM, and MPI_COMM_WORLD are the standard MPI variables as defined in [24].

As a last step, processor 1 can write out the entire 2-D slice of data to a single file (for $i = 1$ to *imax* and $j = 1$ to *jmax*, if including ghost cells).

Input files are read using a similar procedure (in reverse), but using MPI_BROADCAST to distribute the data to all of the processors in place of MPI_REDUCE.

IV. I/O Using HDF5

When writing a 2-D slice of electromagnetic field data from an FDTD grid (stored in memory) to a file, an HDF5 command (H5S_SELECT_OR) is used to join "hyperslabs" of the data in memory and map them to their corresponding locations in the filespace. For the case of the global FDTD model, the "union" of the data in memory and its corresponding position in the filespace are determined for each row of data separately for two reasons: 1) the number of cells in the east–west direction changes with latitude as either pole is approached and 2) if there is one or more merging of cells in the east–west direction on a processor, the data to be written may be offset horizontally (the starting i index may not be 1). Once all of the data of interest in memory is assigned to a corresponding location in the file space, all of the processors write their data simultaneously to a file. An analogous process can be followed to read data from input files, or to read–write 3-D data.

In Fortran, the first data entry of a 3-D field array on each processor has the address of (1, 1, 1), and in C/C++ the first address is (0, 0, 0). In this example, a starting address of (1, 1, 1) is used. In other words, the E_r data starting at (*istart*, *jstart*, *kstart*) on a given processor is actually located at the memory address starting at (1, 1, 1), even when *istart*, *jstart*, and *kstart* are not 1. If there are ghost cells in that area of the E_r matrix, those would need to be accounted for (the ghost cells would need to be skipped before reaching the data in the FDTD cells).

The variable name "offset" is used to indicate how many rows or columns must be skipped to locate the data in memory (*offset_mem*) or in the filespace (*offset_file*). The first index refers the number of grid cells that should be skipped in the east–west (i) direction, and the second index refers to the number of cells that should be skipped in the north–south (j) direction. The following would be repeated for each j-index of subgrid values stored on each processor:

```
11    offset_mem (1) =istart(j)-istart
12    offset_mem (2) = j-jstart
13    offset_file (1) = istart(j)-1
14    offset_file (2) = j − 1
```

Also, within the j-index loop, the variable name "count" is used to indicate the length of the hyperslab. Specifically, *count_mem* (1) and *count_mem* (2) give the length of the hyperslab in memory in the east–west and north–south directions, respectively. In this example, the data are written out row by row due to the merging of cells (*count_mem* (2) always has a value of 1). See for example the row of data in the yellow box of Fig. 3. Note however, as discussed later, combining multiple rows of the same length into "chunks" speeds up the writing, because a larger amount of data is written from a single write operation.

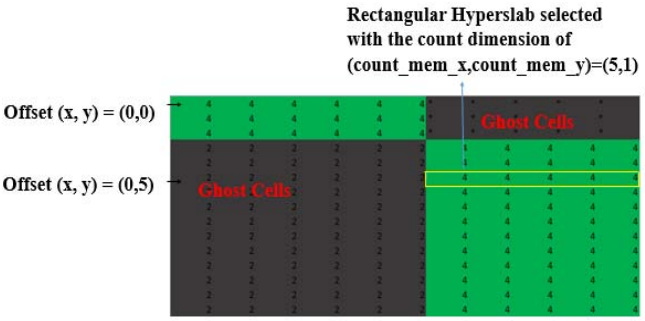

Fig. 3. Illustration of how the alignment of the data changes with latitude (j index) in the processor's memory. This case is for processor 4 near the North Pole, as seen in Fig. 1.

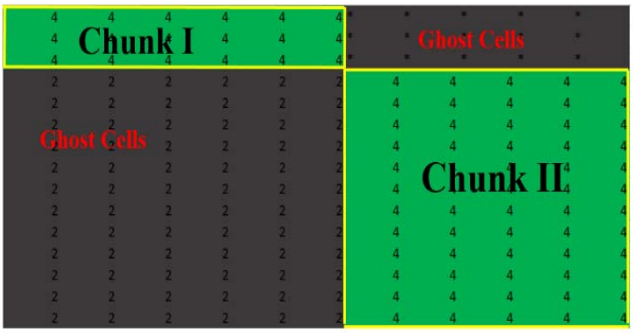

Fig. 4. Selection of 2-D hyperslabs as chunks rather than row by row as shown in Fig. 3.

```
15    count_mem (1) = iend(j) - istart(j) +1
16    count_mem (2) = 1
```

The length of the hyperslab in memory must be the same length as its corresponding location in the file (where it will be written). Thus,

```
17    count_file (1) = count_mem (1)
18    count_file (2) = count_mem (2)
```

The following two lines create the union between the hyperslabs in memory and their corresponding location in the file (see Appendix A and [8]):

```
19    CALL h5sselect_hyperslab_f (filespace,
      H5S_SELECT_OR_F, offset_file, count_file, hdferr)
20    CALL h5sselect_hyperslab_f (memspace,
      H5S_SELECT_OR_F, offset_mem, count_mem, hdferr)
```

Lines 11 to 20 are repeated at each latitude (each row or j-index) of the grid. Once all of the unions between data in memory and the filespace are made, all of the processors write their data simultaneously to the file (see Appendix A and [8])

```
21    CALL h5dwrite_f (dset_id,
      H5T_NATIVE_DOUBLE, Er, dimsf, hdferr,
      file_space_id=filespace, mem_space_id= memspace,
      xfer_prp=plist_id)
```

As mentioned earlier, instead of creating hyperslabs row by row, the data can be selected as 2-D chunks. Fig. 4 illustrates two example chunks of data for processor 4. Chunks can be formed between the latitudes of mergings of cells and/or between the interfaces between processors. In this case, instead of *count_mem* (2) and *count_file* (2) always equaling 1

TABLE I
PERFORMANCE ANALYSIS WHILE READING THE FILE

Parameter	Using MPI_BROADCAST	Using HDF
Size of file	3827 KB (.dat file)	832 KB (compressed .h5 file)
Time to read	0.2 sec	0.4 sec

each time as in Lines 16 and 18, they would be set to a number larger than one determined by how far apart the mergings of cells are and/or interfaces between processors. Additional tests of the chunk by chunk method demonstrated that it is 25% faster than the row by row method.

Reading input files using HDF5 is performed in an analogous manner as for writing except for a few notable differences. Instead of creating a new data set (as is done when writing), we open an existing data set using h5dopen_f command (since the data set already exists in the file that will be read). Then, the h5dwrite_f command of Line 21 is replaced with the h5dread_f command. The process of selecting hyperslabs is similar for both reading and writing. When 3-D hyperslabs are to be read or write in 3-D problems, one more dimension is added to offset_mem, offset_file, count_mem, and count_file, making all of these parameters 3-D.

V. PERFORMANCE COMPARISON

The HDF5 approach of Section IV is first compared to the MPI approach of Section III by having one processor read in a 2-D slice of topography data. In the case of HDF5, each processor reads its portion of the file in parallel. In the case of the MPI commands, one processor reads the entire file and broadcasts the data to each processor. In order to conduct this test involving MPI_BROADCAST, the 2-D slice of topography data must fit onto one processor. Thus, a relatively low global grid resolution of 40×40 km in the north–south and east–west directions is used. At this resolution, the maximum number of cells in the east–west direction (which occurs along the equator) is 1024, and there are 512 cells in the north–south direction. The code is run on 38 processors.

Table I summarizes the results for this comparison. In Table I, HDF is seen to take twice as long, but still well under 1 s. On the other hand, the input HDF file is seen to be 80% smaller than the corresponding data file used with MPI_BROADCAST (a .dat file). HDF files (.h5) are generally smaller than .txt and .dat files because of their ability to store data in binary format [9]. Note that a .dat file can be directly converted into an .h5 file using the *h5import* command. For instance, typing the following at the command line converts a 1024×512-sized topo.dat file into a topo.h5 file with a double precision (64 bits) data type.

$\gg$ *h5import topo.dat –dims 1024,512 –type TEXTFP –size 64 –o topo.h5*

In order to better demonstrate the performance of the HDF5 methodology applied to a global FDTD model, the remaining tests are performed at a resolution of 1.2 km $\times$ 1.2 km. At this resolution, the maximum number of cells in the east–west direction (which occurs along the equator) is 32 768, and there are 16 384 cells in the north–south direction. The code is run on 52 672 processors on the Blue Waters supercomputer. Fig. 5 illustrates an example (zoomed-in) snapshot of the data written out for this test.

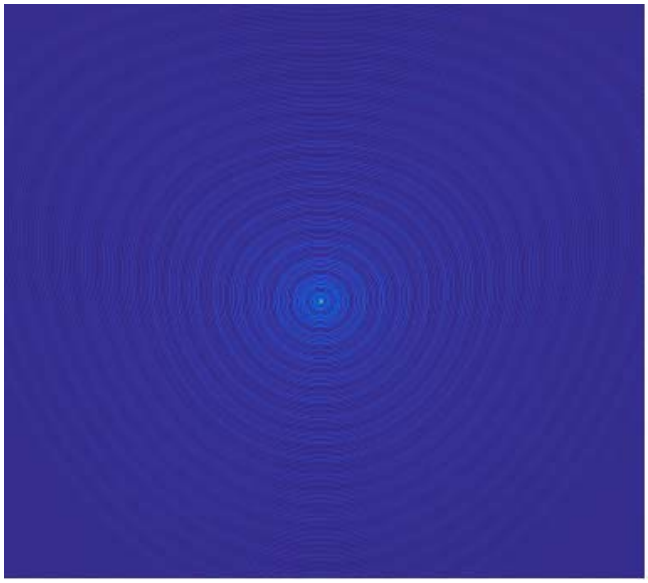

Fig. 5. Zoomed-in snapshot of an electromagnetic wave propagating outward from a vertical "lightning" strike occurring at the surface of the Earth using HDF5 at a global resolution of 1.2 km.

TABLE II
PERFORMANCE ANALYSIS WHILE WRITING A RECTANGULAR SLICE OF ($32\,768 \times 16\,384 = 5.3 \times 10^8$) REAL NUMBERS TO A FILE

Parameter	Using MPI_REDUCE	Each processor writing sequentially	Using HDF5
Size of file	N/A (Insufficient memory on any one processor)	~ 12 GB (.dat file)	~4 GB (.h5 file)
Time to write	N/ A	~ 4-5 hours	~100 sec

Table II compares the speed and file size when HDF5 is used versus the brute-force approach of having each processor write its data in turn to a single file (since MPI commands are not feasible at such high resolutions). It must be noted that the HDF5 result in Table II is far from what the I/O subsystem is capable. Writing 4 GB in about 100 s translates into an aggregate throughput of 40 MB/s. (The Blue Waters I/O system is capable of writing at rates close to 1 TB/s.)

The reason for the poor performance is that, on an average, each MPI process writes only about 80 KB of data. To drastically increase the throughput, we would have to first gather the data in a smaller number of write processes (e.g., 512), which in turn would write larger amounts of data

TABLE III
PERFORMANCE ANALYSIS WHILE WRITING 3-D DATA OF SIZE 32 768 × 16 384 × 20 INTO A FILE

Parameter	Each processor writing into the separate file sequentially	All processors writing into the same file sequentially using HDF
Size of output file	7.4 TB	620 GB

per write operation. The additional code required to implement this kind of aggregation is not very difficult to write, but increases code complexity and is additional burden on the application programmer, and ideally, would be handled by HDF5 and MPI. Alternatively, an (albeit smaller) increase in performance can be expected if all field components are written in one write operation. Support for this feature, the so-called multidata set I/O, is in preparation and will be part of a future HDF5 release.

HDF5 performs best for 3-D problems. Typically, most of the supercomputers allow the simulation to run for 24–48 h at one go. Therefore, all of the electromagnetic field components (electric fields and magnetic fields on all three axes) within the entire 3-D computational domain must be written into the file for longer simulations wherein the simulation must be restarted. A test case is run for writing all the six components (three electric fields and three magnetic fields) of size 32 768 × 16 384 × 20 cells into a file. Table III compares the performance based on the size of the file. The output file generated after using HDF is almost twelve times smaller than the file generated without using HDF. This reduction in size of output data improves the portability of the files.

VI. COMPRESSION

Depending on the application, the global FDTD models may need to be run at double precision. In this case, the resulting input–output files would be especially large when run at high resolutions. These files would be difficult to create and move between different computers. As a result, it is highly useful to compress all of the files using compression filters (SZIP, GZIP, deflate etc.) provided by HDF5 and depending on the data type. Our tests showed that the topography file and output electromagnetic field components involving floating data types are sufficiently compressed using an SZIP filter and the nearest neighbor (NN) coding method (H5_SZIP_NN_OM_F) (see Appendix A) with 32 pixels/block. The following is the call within a FORTRAN code:

22 CALL h5pset_szip_f (plist_id, H5_SZIP_NN_OM_F, 32, hdferr)

The compression can also be done using h5repack [25] and selecting an appropriate filter type on the supercomputer (Blue Waters). For example, the text in the following can be written at the command to compress a file1 into a new file2 using a compression level of 6. A compression level of six is chosen because it is a default compression level for many applications. Gzip compression levels range from 0 (no compression) to 9 (maximal compression).

$\gg$ *h5repack –v –f GZIP = 6 file1 file2*

TABLE IV
PERFORMANCE ANALYSIS OF COMPRESSION OF A TOPOGRAPHY FILE AT A RESOLUTION OF 1.2 km

Type of Compression Filters	Compression Ratio	Time taken for compression
GZIP Level 6	3.59	6 min 25 sec
SZIP (32 pixels/block)	3.9	5 min 02 sec
SHUF + GZIP Level 6	4.91	3 min 51 sec
ZFP (Lossy compression)	2.86	3 min

TABLE V
PERFORMANCE ANALYSIS OF COMPRESSION FOR ELECTRIC FIELD COMPONENTS OF MATRIX SIZE 32 768 × 16 384

Type of Compression Filter	Compression Ratio	Time taken for compression
GZIP Level 6	1.385	3 min 40 sec
SZIP (32 pixels/block)	1.38	2 min 08 sec
SHUF + GZIP Level 6	1.71	2 min 59 sec

Tables IV and V list the compression ratios and compression time for some example compression filters applied to the topography file versus an example electric field output file at 1.2 km resolution. The electric field data of Table V is obtained after running the global model for a sufficiently long time so as to populate the whole matrix with nonzero numbers. In both of the example tests of Tables IV and V, the SHUF + GZIP filter yields the best performance (the best compression ratio) for floating point data if the compression ratio is the benchmark. This combined filter performs better than the GZIP filter alone and in a shorter amount of time. We took the compression ratio as the benchmark because when the data is compressed, it takes the processor less time to write the data to a file.

VII. CONCLUSION

A parallel I/O scheme based on HDF5 was developed and applied to the latitude–longitude global FDTD models of the Earth–ionosphere waveguide of [4]. The process of creating hyperslabs for data offset in memory and in the filespace due to cell mergings in the Polar Regions was described. The advantage of selecting data as coarser "chunks" rather than row by row was also covered. The HDF5 methodology covered here for the more complex case of a spherical-coordinate FDTD model may be adapted more readily to regular Cartesian meshes.

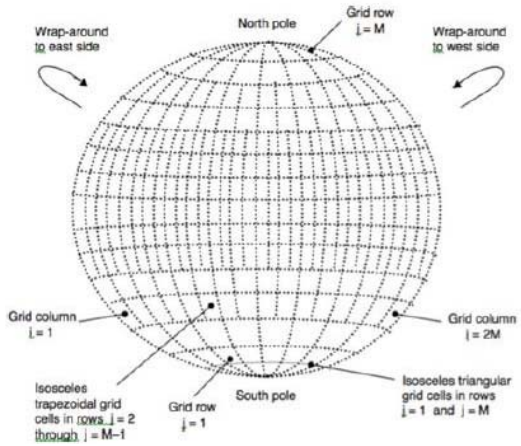

Fig. 6. Layout of the 3-D FDTD latitude–longitude grid as seen from a constant radial coordinate. Adapted from [26].

HDF5 was found to work particularly well when FDTD grids are too large to make MPI_BROADCAST and MPI_REDUCE commands feasible. Furthermore, compression techniques offered by HDF5 may be used to reduce the file size significantly, making it easier to store the file and move it between computers. Note that HDF5 may be installed and used on any supercomputing resources.

Using parallel I/O approaches such as HDF5, global FDTD models of the Earth–ionosphere waveguide may be run at very high resolutions As supercomputers continue to advance in the future, the global FDTD models may be applied to even more applications than previously possible and frequency ranges as the resolution of the models is further increased or the upper boundary of the model is raised in altitude.

Appendix A
Meaning of Abbreviations

dismf: This represents the dimension of the data set.

H5T_NATIVE_DOUBLE: Double precision real data type representation.

Dset_id: ID representation for the data set created by using H5Dcreate.

Er: This is the variable (radial electric field component) that is written into the data set.

hdferr: Flag that is used to track the errors during the simulation.

H5_SZIP_NN_OM_F: SZIP compression filter with NN coding method.

plistid: Property list identifier.

Appendix B
Ghost Cells

Ghost cells are grid cells that are present in the mesh and consume memory, but do not contribute to the computations or solutions. They arise in the Earth–ionosphere waveguide model because of merging of cells along the east–west direction as either pole is approached. Fig. 6 is a 2-D layout of the 3-D FDTD latitude–longitude grid as seen from a constant radial coordinate. Fig. 7 (which is shown in one of the cited references) is an example view of the merging of cells as the South Pole is approached. The merging of every two cells in the east–West direction occurs whenever the width of the cells in the east–west direction reduces to less than one half of the width of the cells at the equator. Fig. 7 (right) shows the ghost cells (in the gray region) that have arisen in memory due to the merging of cells.

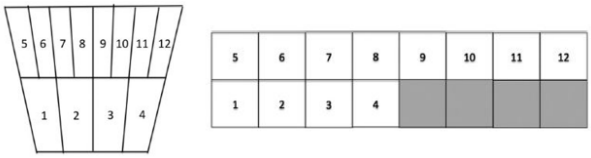

Fig. 7. Example view of twelve spherical grid cells in the Southern hemisphere. Grid cell numbers 5 and 6, 7 and 8, 9 and 10, and 11 and 12 are merged into grid cell numbers 1, 2, 3, and 4, respectively (left). The storage of the grid cell data in memory (right). The grey cells are the ghost cells that consume the memory but do not contribute to the FDTD computations. Adapted from [4].

Acknowledgment

The authors also would like to thank the University of Utah's Center for high Performance Computing for providing supercomputing resources.

References

[1] A. Taflove and S. C. Hagness, *Computational Electrodynamics: The Finite-Difference Time-Domain Method*. Norwell, MA, USA: Artech House, 2005.

[2] J. J. Simpson and A. Taflove, "Three-dimensional FDTD modeling of impulsive ELF propagation about the earth-sphere," *IEEE Trans. Antennas Propag.*, vol. 52, no. 2, pp. 443–451, Feb. 2004.

[3] J. J. Simpson and A. Taflove, "A novel ELF radar for major oil deposits," *IEEE Geosci. Remote Sens. Lett.*, vol. 3, no. 1, pp. 36–39, Jan. 2006, doi: 10.1109/LGRS.2005.856118.

[4] A. Samini and J. J. Simpson, "Parallelization of 3-D global FDTD earth-ionosphere waveguide models at resolutions on the order of ~1 km and higher," *IEEE Antennas Wireless Propag. Lett.*, vol. 15, pp. 1959–1962, 2016.

[5] W. Hu and S. A. Cummer, "An FDTD model for low and high altitude lightning-generated EM fields," *IEEE Trans. Antennas Propag.*, vol. 54, no. 5, pp. 1513–1522, May 2006.

[6] J. J. Simpson and A. Taflove, "ELF radar system proposed for localized D-region ionospheric anomalies," *IEEE Geosci. Remote Sens. Lett.*, vol. 3, no. 4, pp. 500–503, Oct. 2006.

[7] S. A. Cummer, "Modeling electromagnetic propagation in the Earth-ionosphere waveguide," *IEEE Trans. Antennas Propag.*, vol. 48, no. 9, pp. 1420–1429, Sep. 2000.

[8] A. Skjellum, E. Lusk, and W. Gropp, "Early applications in the message-passing interface (MPI)," *Int. J. Supercomput. Appl.*, vol. 9, no. 2, pp. 79–94, 1995.

[9] (2009). *5-A New Generation of HDF, HDF5 User Documentation*. [Online]. Available: https://support.hdfgroup.org/HDF5/doc1.6/

[10] *Why HDF?* Accessed: Feb. 10, 2017. [Online]. Available: www.hdfgroup.org and htttp://www.hdfgroup.org/why-hdf/

[11] D. Pearah. (Oct. 11, 2016). *HDF: The Next 30 Years*. [Online]. Available: htttp://www.hdfgroup.org

[12] Y. Yu and J. J. Simpson, "An E–J collocated 3-D FDTD model of electromagnetic wave propagation in magnetized cold plasma," *IEEE Trans. Antennas Propag.*, vol. 58, no. 2, pp. 469–478, Feb. 2010, doi: 10.1109/TAP.2009.2037706.

[13] A. Samimi and J. J. Simpson, "An efficient 3-D FDTD model of electromagnetic wave propagation in magnetized plasma," *IEEE Trans. Antennas Propag.*, vol. 63, no. 1, pp. 269–279, Jan. 2015.

[14] S. Pokhrel, V. Shankar, and J. J. Simpson, "3-D FDTD modeling of electromagnetic wave propagation in magnetized plasma requiring singular updates to the current density equation," *IEEE Trans. Antennas Propag.*, to be published.

[15] J. Bérenger, "FDTD computation of VLF-LF propagation in the Earth-ionosphere waveguide," *Ann. Telecommun.*, vol. 57, nos. 11–12, pp. 1059–1090, 2002.
[16] D. Garcia-Doñoro, I. Martinez-Fernandez, L. E. Garcia-Castillo, and M. Salazar-Palma, "HOFEM: A higher order finite element method electromagnetic simulator," in *Proc. IEEE Int. Conf. Comput. Electromagn.*, Hong Kong, Feb. 2015, pp. 10–12.
[17] Y. Zhang, T. K. Sarkar, H. Moon, M. Taylor, D. G. Donoro, and M. Salazar-Palma, "Parallel MoM simulation of complex EM problems," in *Proc. IEEE Antennas Propag. Soc. Int. Symp.*, Charleston, SC, USA, Jun. 2009, pp. 1–4.
[18] O. Terzo, P. Ruiu, L. Mossucca, G. Caragnano, M. A. Francavilla, and F. Vipiana, "Low-cost high performance infrastructure for domain-decomposition methods in computational electromagnetics," in *Proc. Int. Conf. P2P, Parallel, Grid, Cloud Internet Comput.*, Barcelona, Spain, 2011, pp. 94–99.
[19] C. Richter, S. Schöps, and M. Clemens, "GPU acceleration of finite difference schemes used in coupled electromagnetic/thermal field simulations," *IEEE Trans. Magn.*, vol. 49, no. 5, pp. 1649–1652, May 2013.
[20] J. Guan, S. Yan, and J.-M. Jin, "A CUDA implementation of the finite element-boundary integral method for electromagnetic scattering simulation," in *Proc. USNC-URSI Radio Sci. Meeting (Joint AP-S Symp.)*, Lake Buena Vista, FL, USA, Jul. 2013, p. 152.
[21] X.-M. Pan and X.-Q. Sheng, "A high performance parallel MLFMA for scattering by extremely large targets," in *Proc. Asia–Pacific Microw. Conf.*, Macau, China, 2008, pp. 1–4, doi: 10.1109/APMC.2008.4958479.
[22] P. Solin, K. Segeth, and I. Dolezel, *Higher-Order Finite Element Methods*. Boca Raton, FL, USA: CRC Press, 2003.
[23] R. F. Harrington, *Field Computation by Moment Methods*. New York, NY, USA: Wiley, 1993.
[24] W. Gropp, E. Lusk, and A. Skjellum, *USING MPI: Portable Parallel Programming With the Message-Passing Interface*. London, U.K.: MIT Press, 1994.
[25] *HDF5 Tools*. Accessed: Feb. 10, 2017. [Online]. Available: https://support.hdfgroup.org/HDF5/doc/RM/Tools.html
[26] J. J. Simpson, "Current and future applications of 3-D global earth-ionosphere models based on the full-vector Maxwell's equations FDTD method," *Surv. Geophys.*, vol. 30, no. 2, pp. 105–130, 2009. [Online]. Available: https://doi.org/10.1007/s10712-009-9063-5

Santosh Pokhrel (S'16) received the bachelor's degree in electronics and communication engineering from Jawaharlal Nehru Technological University, Kakinada, Andhra Pradesh, India, in 2015. He is currently pursuing the Ph.D. degree with the Electrical and Computer Engineering Department, University of Utah, Salt Lake City, UT, USA.

His current research interests include finite-difference time-domain solution of Maxwell's equations and its applications in investigating space weather hazards to electric power grids and numerical modeling of engineering problems and design of analog ICs.

Miguel Rodriguez received the B.S. degree in electrical engineering from California State Polytechnic University, Pomona, CA, USA, in 2003, the M.S.E. degree in electrical engineering from Arizona State University, Tempe, AZ, USA, in 2010, and the Ph.D. degree in electrical engineering from the University of Utah, Salt Lake City, UT, USA.

He joined Dr. Simpson's group, University of Utah, as a Post-Doctoral Researcher. He is currently a Senior Engineer with Raytheon Company, Waltham, MA, USA. His current research interests include optical antennas, implantable antennas, and computational electromagnetics using the finite-difference time-domain method.

Alireza Samimi received the B.S. degree in electrical engineering from Shiraz University, Shiraz, Iran in 2005, the M.S. degree in electrical engineering from the University of Tabriz, Tabriz, Iran, in 2008, and the Ph.D. degree in electrical engineering from Virginia Tech, Blacksburg, VA, USA, in 2013.

He joined Prof. Simpson's Computational Electrodynamics Laboratory, University of Utah, Salt Lake City, UT, USA, as a Post-Doctoral Fellow until 2015. He is currently a Senior Research Scientist with Nanometrics Inc., Milpitas, CA, USA. His current research interests include physics of the upper atmosphere, finite-difference time-domain solution of Maxwell's equations and its applications in wave propagation in the earth ionosphere system.

Gerd Heber received the master's and Ph.D. degrees in mathematics from Ernst-Moritz-Arndt-University, Greifswald, Germany.

He is currently an Applications Architect with The HDF Group, Champaign, IL, USA. His current research interests include applied mathematics, scientific computing, and data modeling.

Jamesina J. Simpson (S'01–M'07–SM'12) received the B.S. and Ph.D. degrees in electrical engineering from Northwestern University, Evanston, IL, USA, in 2003 and 2007, respectively.

She is currently an Associate Professor with the Electrical and Computer Engineering Department, University of Utah, Salt Lake City, UT, USA. Her current research interests include the application of the full-vector Maxwell's equations finite-difference time-domain method to electromagnetic wave propagation spanning 15 orders of magnitude across the electromagnetic spectrum.

Dr. Simpson was a recipient of the 2010 National Science Foundation CAREER award, the 2012 Donald G. Dudley, Jr. Undergraduate Teaching Award of the IEEE AP-S, and the Santimay Basu Medal from URSI in 2017.