Adaptation of HDF5 for FD-FDTD Data Manipulation Tasks

Maksims Abaļenkovs, Fumie Costen, Anthony K. Brown

***Abstract*—Building of Vivaldi antenna arrays is a complicated process. Geometrical and material design determine the physical antenna characteristics. The numerical simulation of electromagnetic wave propagation in time domain is often used for the verification of a Vivaldi antenna structure. Modern implementations of numerical schemes such as the Frequency Dependent – Finite Difference Time Domain (FD-FDTD) suffer from inefficient input/output (I/O) procedures. This research proposes a novel approach of efficient FD-FDTD output data production and storage by means of the Hierarchical Data Format 5 (HDF5).**

I. Introduction

Ultra Wide Band (UWB) is becoming an increasingly popular technology. According to the contemporary definition by the Federal Communications Commission (FCC), an UWB is any signal occupying 500 MHz within the 3.1 to 10.3 GHz frequency range. This makes an UWB suitable for application in areas of vehicular radars, imaging and communication systems. No carrier is required for UWB signal propagation. A particularly wide spectrum results in reduced operating power, low sensitivity to multipath propagation effects and relatively high transmission rates of an UWB-based communication system.

The Finite Difference Time Domain (FDTD) is a classical numerical scheme for the simulation of the UWB wave propagation. Being developed by Kane S. Yee in 1966 it is still the most widely used simulation technique [Yee66]. It has survived a number of substantial changes and improvements in more than 40 years of application in different research domains. Equations 1 to 6 describe the general Maxwell's equations, which comprise the basis of the FDTD calculation.

$$\nabla \cdot \boldsymbol{D} = \rho \tag{1}$$

$$\nabla \cdot \boldsymbol{B} = 0 \tag{2}$$

$$-\nabla \times \boldsymbol{E} = \partial_t \boldsymbol{B} \tag{3}$$

$$\nabla \times \boldsymbol{H} = \boldsymbol{J} + \partial_t \boldsymbol{D} \tag{4}$$

$$\boldsymbol{D} = \varepsilon \boldsymbol{E} \tag{5}$$

$$\boldsymbol{B} = \mu \boldsymbol{H} \tag{6}$$

The general idea of the FDTD method lays in the replacement of derivatives ∂_t in Maxwell's equations by finite differences. This substitution allows to obtain numerical result for a theoretical electromagnetic case described by Maxwell's equations. Recalling the Taylor series expansion, which states that centred average a_v and difference δ_v operators serve the second order approximations to the derivatives ∂_t, provides a mathematical mechanism for the FDTD method.

The FDTD simulation domain is divided into a number of unitary space elements called the *Yee unit cells*. The x, y, z constituents of the electric and magnetic fields $\boldsymbol{E}$ and $\boldsymbol{H}$ are measured at certain positions of each Yee cell. Furthermore the entire set of $E^n{}_{(x,y,z)}$ and $H^n{}_{(x,y,z)}$ values of a single Yee cell is divided into two mutually exclusive subsets of electromagnetic field values. Where each subset contains $\boldsymbol{E}$ and $\boldsymbol{H}$ values for the specific unitary cell locations x, y, z. The FDTD calculation proceeds in a "leap-frog" manner in time domain. The electromagnetic field values comprising the first subset are calculated at one time step and $\boldsymbol{E}$ and $\boldsymbol{H}$ values, which belong to another subset, are estimated at the following time step. The FDTD algorithm performs in the marching-on-in-time way for the complete range of the simulation time steps.

Despite of its simplicity and ease of implementation the original FDTD method has a number of drawbacks. The most serious one is the algorithm's inability to reflect the medium properties wherein the UWB signal is propagating. A numerical scheme's variation called the Frequency Dependent – Finite Difference Time Domain (FD-FDTD) method was developed specifically to address this issue. In this case the source medium responds to the frequency of the scattered signal. Similarly to the original approach the medium geometry is specified by the spacial grid.

The main difference of the FD-FDTD is that the medium permittivity ε is defined at each spacial location of the simulation grid. The permittivity ε is set to be dependent on UWB signal's frequency, but permeability μ stays constant during the entire computation process. The medium frequency dependency is achieved by the application of the Debye model:

$$\varepsilon = \varepsilon_0 \varepsilon_r = \varepsilon_0 \left(\varepsilon_s + \frac{\varepsilon_s - \varepsilon_\infty}{1 + \jmath\omega\tau_D} - \jmath \frac{\sigma}{\omega \varepsilon_0} \right) \quad (7)$$

To involve the variable medium permittivity in the computation procedure the displacement vector $D^n{}_{(x,y,z)}$ values have to be calculated along with the original $E^n{}_{(x,y,z)}$ and $H^n{}_{(x,y,z)}$ field parameters. This is done by replacing the constant permittivity value in the Eq. 5 by the variable permittivity parameter obtained from Eq. 7. This puts higher requirements on the computational resources executing the FD-FDTD method, but ensures a more realistic simulation producing more precise results.

The initial FD-FDTD method used in the research group for the simulation of UWB signal propagation was implemented in 2005 by Arnaud Thiry within his PhD work [Thi06]. The in-house software was successfully parallelised by João Costa in the following year [Cos06]. The core FD-FDTD program is developed in Fortran 90 with MPI and C subroutines. The FD-FDTD simulation supplies a large amount of output data with $\boldsymbol{D}, \boldsymbol{E}$ and $\boldsymbol{H}$ values for each space grid location x, y, z and time step t. The produced data is stored in a number of textual ASCII files. Where each output file represents a single simulation time step and contains the displacement and electromagnetic field values for the entire grid space.

In the parallel FD-FDTD in-house software implementation the computation domain is divided among the available processors. Currently the workload is spread according to the z axis of the simulation grid space. This means one CPU calculates a certain volume of the FD-FDTD space for the complete simulation time range. The total amount of output files produced equals to $t_{max} \times n_{CPUs}$, e.g. the algorithm execution for 10 time steps performed on 2 processors will result in $10 \times 2 = 20$ output data files. The real-world simulations are run for at least 5000 time steps on 16 CPUs. In this case a single file's size reaches 250 MB and the total amount of disk space required is $5000 \times 16 \times 250 = 19.07$ TB. Moreover the time needed to output a single text file drastically increases and can take up to 30 minutes.

Summarising all the above conditions states that one FD-FDTD simulation can easily extend to many days. The current procedure of data production and storage in ASCII format was considered non-optimal. This paper analyses major scientific data formats available and presents a novel approach for the FD-FDTD data output and storage based on the Hierarchical Data Format 5 (HDF5) functionality.

II. Scientific Data Processing

A. Current Status and Future Forecast

The scientific data volumes are doubling each subsequent year. There is a strong need for interactive data analysis tools, that would be able to cope effectively with the increasing amount of information. Currently there exists a gap in the human-machine interface. A scientist becomes confused by the huge volume of information. He needs to be put back in control of his data. The advanced information analysis utilities are required for better data manipulation. Sophisticated complexity and super-linearity of data analysis algorithms as well as the low I/O bandwidth growth in comparison to the storage space increase are the trends of the contemporary data processing.

Jim Gray et al. [GLNS+05] predict a development of data centre ideology, where a scientific computing resource centre will keep all the necessary data and programs required for data analysis. Every scientist will have a designated workspace area on the data centre's server called *myDB*. Another important notion in the future data processing is a *smart notebook* – a mechanism providing efficient exploration, capture, organisation, analysis, visualisation and publication of data. Using the smart notebook scientists will be able to manipulate their data stored in the computing centre by means of the service-based communication.

Metadata will play an important role in the development of novel information analysis tools. Metadata – is a descriptive information about data containing attributes, names, units, precision, accuracy, data layout and especially data lineage: the way the information was measured, acquired and computed. Many multimedia formats such as JPEG, PDF and MP3 already support metadata. Metadata will provide an intermediate layer bridging the raw data and analysis software guided by scientists.

Special-purpose scientific formats have emerged in the last decade. They provide data models for the representation of numerical data arrays and relationships among them. The data processing based upon these formats usually follows a *file-at-a-time* procedural data analysis. In most cases scientific formats pass the information to programming languages for further processing. This enforces the *filter-then-analyse* approach in operation with data. The raw data has to be gathered in a temporary storage buffer, sorted, filtered for the effective information and then analysed.

On the other hand the commercial world provides a data manipulation solution standardised around the Relational Database Management Systems (RDBMS) and Structured Query Language (SQL). Application of database systems for data processing tasks offers a wide range of benefits including automatic parallelism, indexing, non-procedural data access and look-up capabilities, data independence, federation, replication and back-ups, as well as flexible organisation, automatic design and management

options.

Impedance mismatch existing between the relational database model and the object-oriented model of programming languages will significantly reduce within the next decade. New highly flexible information systems will arise during the confluence of databases, file systems and object-oriented programming languages. This evolution will bring the modern database performace features and scientific format data representation models together and will result in highly efficient information analysis products. It is expected that scientific data formats will be included into commercial database systems as standard data types.

The first step towards the efficient data processing is a joint application of scientific data format and operation parallelism provided by the Message Passing Interface (MPI).

B. Modern Formats

There was a number of data formats developed especially for scientific use:

Network Common Data Form (NetCDF) was designed at the Unidata Program Center and aimed for manipulation of atmospheric science data.

Flexible Image Transport System (FITS) was implemented at NASA Goddard Space Flight Center with the main intention to store astronomical data sets.

Portable Binary Data Format (PDB) was developed at the Lawrence Livermore National Laboratory (LLNL) for general usage in physical research. Later it was adopted by the U.S. Department of Energy.

Hierarchical Data Format (HDF) was created by the National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana-Champaign (UIUC). It was first applied for the internal data exchange. In the later time NASA used it for the Earth Observing System programme.

III. Hierarchical Data Format 5

Hierarchical Data Format 5 (HDF5) was selected for the application in the in-house FD-FDTD software. Bearing comparable and better performance characteristics with the other scientific formats [Pou02], HDF5 excels in higher flexibility and much broader feature support [Gro02].

HDF5 places no limit upon data file size and number of objects in a file. The format provides C, C++, Java and Fortran application programming interfaces (APIs). The versatile data model allows to express other formats in terms of HDF5. Data stored in this format might be described by common and user-defined metadata. Virtual File Layer (VFL) offers efficient standard, parallel and network I/O as well as diverse storage media support. A number of advanced operations on data is possible during the I/O process: data type and location change, subsetting and optional separation of meta- from raw data. There exist data compression, extensibility and chunking strategies for the information storage. HDF5 provides a wide selection of pre-defined data types.

Fortran 90 programming language support, parallel data access capabilities and flexibility of data structure design were the most important features that have influenced the research group's decision to apply HDF5 for the FD-FDTD data manipulation activities.

IV. Output File Design in HDF5

Storing FD-FDTD data in textual ASCII form was considered extremely inefficient. A new file structure in terms of HDF5 has to be designed to replace the

The original ASCII file represented one time step of the FD-FDTD simulation and contained the displacement $D^n{}_{(x,y,z)}$ and electromagnetic field values $E^n{}_{(x,y,z)}$ and $H^n{}_{(x,y,z)}$ for the entire grid space. Table I shows the column-wise organisation of the initial output file. The first 3 columns are obligatory since they specify the exact location in the grid space. They hold x, y and z cartesian coordinates. The following 9 columns intended to store the displacement and electromagnetic field values are optional. The number of output field values as well as the spacial positions are set according to the simulation requirements.

Space Coordinates			Field Values		
x	y	z	E_x	$\cdots$	H_z
1	1	90	-0.15396E+10	$\cdots$	-0.15041E+12
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$
189	467	95	0.13878E+08	$\cdots$	0.13895E+10

TABLE I
Output Data File Structure

The major building blocks of the HDF5 format are:

Group – is a container structure organising datasets and other groups.

Dataset – is an n-dimensional data array of any type. This is the main entity for data storage.

Datatype – is a structure specifying the nature and character of raw data in a dataset.

Dataspace – is an entity setting the data layout and indicating its intended usage.

Format's groups and datasets are similar to the Unix notions of directories and files. It is also possible to create links between HDF5 objects to enable data sharing.

A complete volume of displacement and electromagnetic field values for one time step is kept in a single HDF5 *dataset*. Cartesian coordinates x, y, z describing the spatial domain of the simulation are represented by a three-dimensional *dataspace*. While there might be up to 9 different field values assigned to a single spacial location, a user-defined one-dimensional array *datatype* is created to hold the calculated field parameters. The original idea of separating the stored data according to the time step value was left intact. A single HDF5 data file contains all the information referring to one time step. There was an option to output the entire spatial and time domain data into one file, but it was rejected because of a potentially large resulting file size. A new file consists of a default *root* group "/" with a custom *group* "MyGroup" created within it. This is done for flexibility reasons to allow future ordering and relocation of raw data inside a file. A string attribute "MyAttribute" containing the textual data description concludes the HDF5 file contents.

HDF5 library features as data chunking, shuffling and compression were enabled to provide better data manipulation and disk space usage. Chunking separates the raw data kept in a single dataset into a number of equal parts. This stimulates fast partial access to the information and more importantly provides a mechanism for parallel operation upon the data. Shuffling is an auxiliary option, which allows to reach better compression levels. Similarly to the interleaving approach it re-orders the byte sequence of data providing a higher compression ration. Since the displacement and electromagnetic field information is of numeric character it has a relatively high locality, which makes shuffling especially useful.

Each HDF5 output file is compressed with a GNU zlib algorithm. Moderate encoding level of 6 is a trade-off between the speed of output production and amount of storage space usage. When applying shuffling a possible concern might be the additional time required for excessive byte permutations. However, a series of practical experiments conducted by the HDF Group in [Gro04] states that shuffling and bzip2 compression operation needs less amount of time than the sole bzip2 data encoding. This work also indicates 5% compression improvement for 64-bit float data and 10% for 32-bit.

V. Modification of FD-FDTD Software

Pseudocode in the Algorithm V.1 indicates the main steps of the HDF5 output file production. The notation used within the pseudocode is as follows. Parameter i identifies the HDF5 Fortran interface. An HDF5 file, group, dataspace and dataset are represented with variables f_{h5}, g, s and d respectively. Custom file and memory datatypes are t_{file} and t_{mem}. Parameter l specifies the dataset creation property list, while $r_{chunk}, r_{shuffle}$ and r_{zlib} are the chunking, shuffling and compression filters applied to it. Parameters $wdata$ and $rdata$ are the auxiliary input and output raw data buffers. Finally $\boldsymbol{D}$, $\boldsymbol{E}$ and $\boldsymbol{H}$ denote the displacement and electromagnetic field values.

1: **initialise** HDF5 Fortran interface i
2: **create** new file f_{h5}, group g, 3D dataspace s
3: **create** file array datatype t_{file}, memory array datatype t_{mem}
4: **modify** dataset creation property list l to use filters $r_{chunk}, r_{shuffle}, r_{zlib}$
5: **create** dataset $d \in g \in f_{h5} \leftarrow s, t_{file}, l$
6: **create** output and input data buffers $wdata, rdata$
7: **fill in** $wdata \leftarrow \boldsymbol{D}, \boldsymbol{E}, \boldsymbol{H}$
8: **write** $wdata \rightarrow f_{h5}$ using t_{mem}
9: **read** $f_{h5} \rightarrow rdata$ using t_{mem}
10: **close** l, s, d, g, f_{h5}, i

Algorithm V.1: HDF5 Output File Production

The novel output algorithm proceeds in the following manner. The HDF5 Fortran interface is initialised at the beginning of the data output process. A new HDF5 file f_{h5} is created for each time step of on each CPU. A group g is constructed within the default root group in the file. Three-dimensional dataspace s sets the data layout for a future dataset d. Two user-defined one-dimensional array datatypes t_{file} and t_{mem} are prepared next. The platform-independent file datatype t_{file} is intended for HDF5 file reading and writing operations, whereas the memory datatype t_{mem} is different for each specific computer architecture. It supports the data transfer operations in memory.

A default dataset creation propertty list l has to be modified to pass the raw data through chunking, shuffling and compression filters $r_{chunk}, r_{shuffle}$ and r_{zlib} before output. Finally the dataset d, the main container keeping the simulation data, is created using the dataspace s, file array datatype t_{file} and the altered dataset creation property list l. An output data buffer $wdata$ is filled in with the calculated $\boldsymbol{D}$, $\boldsymbol{E}$ and $\boldsymbol{H}$ values prior to the final data writing into the HDF5 file. The memory datatype t_{mem} has to be applied for the writing procedure. The intermediate data buffering is required for a more efficient data output. In case of bufferisation the entire simulation data set is written into the file in one operation, which is faster than multiple output procedures for each grid space location.

The dataset reading from a file is optional. Similarly to the writing operation the information is read from a file using the memory datatype t_{mem} and is stored into an input buffer $rdata$. Reading is

currently used for the verification purposes of correct file infill. Access to the current dataset creation property list l, dataspace s, dataset d, group g, file f_{h5} and the Fortran interface i should be terminated in the end of data output to release the computing resources.

The FD-FDTD in-house software was modified to accommodate the HDF5 output procedure described in the Algorithm V.1. To ease the program debugging each HDF5 file production subroutine is supplied with a status message. The program capability to output files in ASCII format was left intact. It is possible to specify the desired data output format before the simulation begin. The dual output capability will be important for the output format performance experiments.

VI. Future Work

A. Project Development

An extensive testing procedure is required to measure the FD-FDTD output performace. The existing ASCII and HDF5 formats should be compared according to the output speed and disk space usage. This could be done gradually increasing the amount of simulation output. A program could be executed producing no output at all, E_z only, $E_{x,y,z}$ and $D_{x,y,z}$, $E_{x,y,z}$ and $H_{x,y,z}$.

The FD-FDTD data has to be post-processed to produce either static or animated electromagnetic signal representation in time domain. This is done for the verification of simulation correctness. The post-processing utilities will be developed in Fortran using the HDF5 library for parallel data access.

Since the FD-FDTD simulations demands large amount of resources it is usually run on High Performance Computing (HPC) systems. The new FD-FDTD software has to be ported to Horace and IBM BlueGene/L (BGL) supercomputers the research group works with. The successfull code porting depends on the HDF5 library installation onto the HPC machines. Significant difficulties were encountered while attempting to setup HDF5 on the BGL system.

Another possible project improvement might be a development of intelligent output algorithm similar to the optimised implementation of the Lanczos method [SHS07]. Where only valid $\boldsymbol{D}$, $\boldsymbol{E}$ and $\boldsymbol{H}$ field parameters will be output. In this case the output time will be saved on all zero field values.

B. Conclusion

A novel efficient data storage approach for the FD-FDTD electromagnetic simulation was developed within this project. The HDF5 scientific data format was selected for keeping the computational raw data. The preference to HDF5 was given because of its Fortran interface, extensive selection of data handling features and most importantly parallel access support. This research shares the practise and experience of HDF5 application to electromagnetic computation purposes which could be widespread to any other scientific area requiring storage and efficient manipulation of large amounts of raw data.

References

[Cos06] João Costa. Application of high performance computing to FDTD for UWB systems, September 2006.

[GLNS+05] Jim Gray, David T. Liu, Maria Nieto-Santisteban, Alex Szalay, David J. DeWitt, and Gerd Heber. Scientific data management in the coming decade. *ACM SIGMOD Record*, 34(4):34–41, January 2005.

[Gro02] The HDF Group. HDF5 wins 2002 R & D 100 award. Technical report, The National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, 2002.

[Gro04] The HDF Group. Performance evaluation report: gzip, bzip2 compression with and without shuffling algorithm. Technical report, The National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign, 2004.

[Pou02] Elena Pourmal. FITSIO, HDF4, NetCDF, PDB and HDF5 performance, some benchmarks results. *Science Data Processing Workshop*, February 2002.

[SHS07] Jürgen Schnack, Peter Hage, and Heinz-Jürgen Schmidt. Optimized implementation of the Lanczos method for magnetic systems. *Journal of Computational Physics*, 2007.

[Thi06] Arnaud Thiry. *Efficient FDTD for Broadband Systems*. PhD thesis, University of Manchester, School of Computer Science, Kilburn Building, Manchester M13 9PL, United Kingdom, May 2006.

[Yee66] Kane S. Yee. Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media. *Antennas and Propagation, IEEE Transactions on [legacy, pre-1988]*, 14(3):302–307, 1966.