Data Format Selection for an I/O-intensive large-scale FDTD

Maksims Abaļenkovs[1], Fumie Costen[1], Craig Lucas[2], and Anthony Brown[1]
[1] School of Electrical and Electronic Engineering,
[2] Research Computing Services, The University of Manchester, U.K.

Introduction

The development of Ultra Wide Band (UWB) systems requires numerical simulation to examine the waveform distortion in the time domain during propagation in a wide range of dispersive media. Finite Difference Time Domain (FDTD) methods are the most suitable for UWB system modelling, offering the capability of analysing arbitrarily-complex, wideband problems. The main difficulties in use of FDTD are the long execution time and high memory requirements when the simulation contains geometrical features that are electrically-small relative to a large physical space. One of the approaches to tackle these problems is the parallelisation of FDTD. The standard explicit FDTD is inherently highly parallelisable on distributed memory architectures. It is adaptable even when bandwidth between Processor Elements (PEs) is not dedicated for computation. This holds because the communication time is small relative to the computation time for the large FDTD space. However, with the performance improvement of CPU and memory, the data file production of the large scale FDTD simulation dominate the total elapsed time. This paper discusses possible approaches to the reduction of the data Input/Output (I/O) time from the perspective of the computational environment and the output format. This work also provides a guideline on the appropriate data format, depending on the computational environment, the size of the FDTD space and amount of the data required.

Methods to Improve I/O Performance

Given the weak I/O performance of the currently available High Performance Computing (HPC) environments relative to the performance of CPU and memory, the simulation time for an I/O-intensive large-scale FDTD is mainly spent on the data file production [1] in order to detail the wave propagation at each FDTD grid at each time step. Identical problems are experienced by other scientific numerical simulations such as [2]. One of the promising exercises to improve I/O performance is the adoption of a high-level I/O library for a scientific data format built on top of the Message Passing Interface (MPI) - IO interface such as ROMIO [3] configured to operate on top of Parallel File Systems (PFS) or the Parallel Virtual File System (PVFS) [4] which provides a high performance I/O infrastructure [5]. Similar approach has been taken by other works such as [6]. Although PVFS seems to provide the most appropriate computational environment for the I/O-intensive FDTD simulation, the HPC facilities, to which the authors have had an access, do not have local disks dedicated to each PE for data storage, PVFS nor ROMIO. Without PFS, the data production onto the shared disk by each client causes significant load-imbalance. Under the given computational environment available to the authors, the in-house FDTD code is re-implemented so that only one I/O server can produce output data. This approach achieves high scalability and is portable to any HPC system with weak I/O. However, the total elapsed time significantly varies depending on the data format. Therefore this paper assesses Hierarchical Data Format (HDF5) as a scientific data format among others due to its high flexibility in data

storage and manipulation. This work also discusses the HDF5 merit over ASCII and binary formats which are commonly used in Computational Electromagnetics.

Computational environment

Many wide-access Beowulf clusters do not have scientific format libraries installed. The clusters accessible to the authors did not have HDF5 installed. Therefore the experiments are performed in the local cluster. 2^{n_e} Processor Elements (PE) are used for FDTD calculation where n_e is the integer from 0 to 3. 2^3 PEs are composed of 2 single core AMD Athlon 64 4000+ and 3 dual core 4200+. Each node equips 4GB of memory and is connected with the Gigabit Ethernet to a Gigabit switch. Each PE computes the FDTD space of $N_x \times N_y \times N_z$ grid points. Since each core in the dual core node has only 2 GB of memory, FDTD subspace greater than 200^3 grids is not allocated to a single PE to avoid the usage of the swap region. Therefore, the value of $N_x = N_y = N_z$ is varied from 30 to 190 with the increment of 10. The number of grids at interface of PEs is set to $N_x \times N_y$ for all numerical experiments. Thus 2^{n_e} PEs accommodate $N_x \times N_y \times (N_z \times 2^{n_e})$ FDTD grids in total. The point source excitation is placed at the centre of the first FDTD subspace. The 6 fields of E_x, E_y, E_z, H_x, H_y and H_z at each FDTD grid point are outputted together with the Cartesian coordinate at each time step upto 10 time steps. One output file holds the entire data from one PE. The 3D dataspace and 1D array of 6 elements are used for storing the simulation data in HDF5 files. The data in a file is chunked and shuffled to enable the zlib compression. An intermediate I/O-buffer keeps the entire dataset processed by a single PE before saving the data to a file.

Result Analysis

The FDTD calculation is composed of computation, communication and datafile production. The computation time is accumulated for 10 time steps at each PE. The standard deviation of the computation time of 2^{n_e} PEs is under 10 percent of the average computation time. The highest computation time among ones from 2^{n_e} PEs is used as a total computation time of the calculation for 10 time steps. The time to communicate the field values at the interface between PEs is about $1/6$ of computation time, independent of the size of the FDTD space allocated to each PE. An FDTD computation setting is used three times to produce data in ASCII, binary and HDF5 formats. One computation is dedicated to one of these formats. Symbols $\times$, $+$, $*$, and $\Box$ in Figure 1 show the measured time spent for the HDF5 data production using 2^{n_e} PEs with $n_e = 0, 1, 2, 3$, respectively. As is seen, the measured time is the function of the FDTD grids allocated to each PE and the total number of PEs. The data fitting is performed for a fixed number of PEs as follows: first, the Bezier smoothing noted as $Bez\left(n_e, N_x^3\right)$ is applied to the measured data. Secondly the data fitting noted as $F\left(n_e, N_x^3\right)$ is performed to the Bezier curve. Four curves in Figure 1 are defined by the function $F\left(n_e, N_x^3\right)$ for the different number of PEs 2^{n_e}. The accuracy of the data fitting is calculated as $\left(\frac{1}{4 \cdot 17} \sum_{n_e} \sum_{N_x^3} \left(\frac{F\left(n_e, N_x^3\right) - Bez\left(n_e, N_x^3\right)}{Bez\left(n_e, N_x^3\right)}\right)^2\right)^{\frac{1}{2}}$ and error rate of 8 % is obtained from Figure 1. This function is used to estimate the HDF5 data production time for 2^4 PEs and 2^8 PEs which is presented in Figure 2. Similar data fitting is carried out to the measured time to produce data in ASCII and binary formats. These two cases

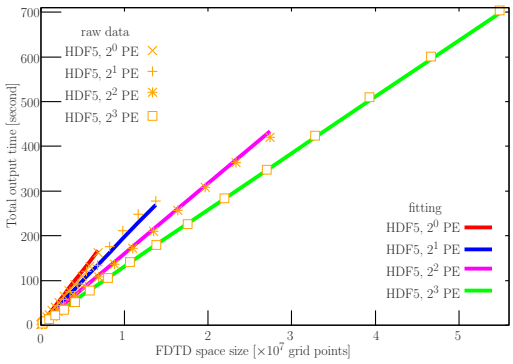

Figure 1: Data fitting

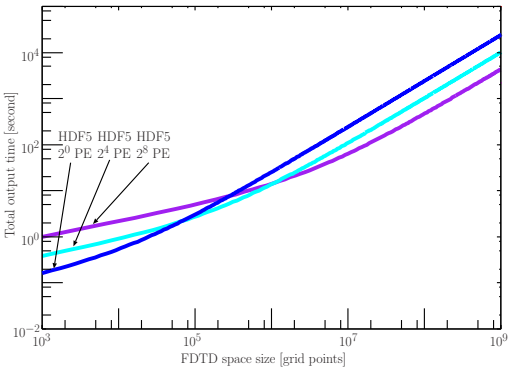

Figure 2: Time for HDF5 data production

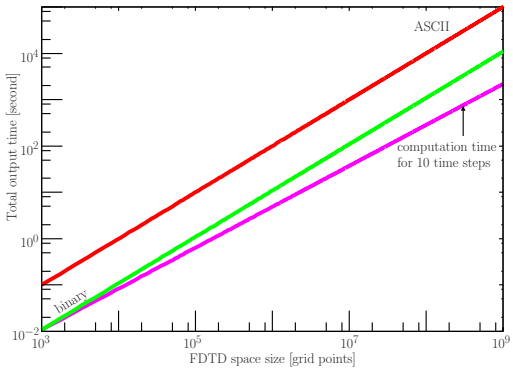

Figure 3: Time for data production in ASCII and binary

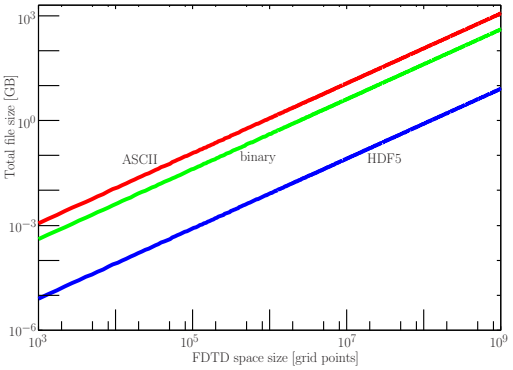

Figure 4: File size in various output formats

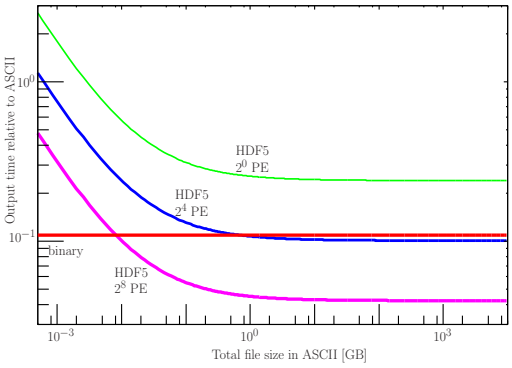

Figure 5: Ratio of data production time compared with ASCII format

are the function of N_x^3 exclusively and the fitted curves are shown in Figure 3 together with the computation time for 10 time steps as a reference. The amount of data produced from each output format is plotted in Figure 4. Figure 2 and Figure 4 tell that HDF5 datafiles larger than 1 MB can be efficiently produced as a collection of more than 150 files with the filesize less than 6 KB whilst HDF5 data smaller than 1 MB should be created as few files as possible for high efficiency. Using Figure 2 and Figure 3, the time to produce binary data and HDF5 data relative to the time to produce ASCII data is calculated in Figure 5. Independent of the total file size produced, a cluster with less than 2^4 PEs produces HDF5 data more slowly than binary data. When more than 2^4 PEs are available, HDF5 data production is more efficient than binary data production only when the corresponding data file size in ASCII is over 1 GB.

Conclusion

One of the bottle-necks of the large-scale FDTD calculation is the data I/O. Although PFS improves the I/O performance it is not always available. Under the computational environment without PVFS and local disks for each PE, limitation on the number of nodes which can produce the output data improves the scalability. HDF5 is compared with the ASCII format and the binary formats from the perspective of the computational efficiency. It becomes clear that HDF5 is beneficial when more than 2^4 PEs are used and more than 1 GB of data in ASCII format are produced. The effect of the number of files on the datafile production time as seen in Figure 2 and Figure 3 is under investigation for the presentation in the conference. Since it is difficult to install and maintain scientific format libraries, they are usually not present on wide-access HPC systems. This means high portability of the code using any scientific format is currently not achievable.

References

[1] R. Maaskant, M. Ivashina, R. Mittra, W. Yu, and N. Huang, "Parallel FDTD modeling of a focal plane array with vivaldi elements on the highly parallel LOFAR BlueGene/L Supercomputer," in *IEEE AP-S Int. Symp.*, 2006.

[2] R. Ross, D. Nurmi, A. Cheng, and M. Zingale, "A case study in application I/O on Linux clusters," in *Supercomputing*, 2001.

[3] R. Thakur, W. Gropp, and E. Lusk, "On implementing MPI-IO portably and with high performance," in *Workshop on Input/Output in Parallel and Distributed Systems*, 1999, pp. 23–32.

[4] P. H. Carns, W. B. Ligion, R. B. Ross, and R. Thakur, "PVFS: A parallel file system for Linux clusters," in *Linux Showcase and Conf.*, 2000, pp. 317–327.

[5] J. Li, W. Liao, A. Choudhary, and V. Taylor, "I/O analysis and optimization for an AMR cosmology application," in *IEEE Int. Conf. Cluster Computing*, 2002, pp. 119–126.

[6] Y. Chen, J. Nieplocha, I. Foster, and M. Winslett, "Optimizing collective I/O performance on parallel computers: A multisystem study," in *Supercomputing*, 1997, pp. 28–35.