Efficient acoustic modelling of large acoustic spaces using finite difference methods

S Durbridge Bowers & Wilkins, West Sussex, UK

A Hill Department of Electronics, Computing & Mathematics, University of Derby, Derby, UK

# INTRODUCTION

Improvements in the flexibility, accuracy and performance of simulation tools could help towards making predictions, performing low-cost exploration (rapid prototyping) and system design workflows easier, faster and more intuitive. Time domain numerical methods used for performing acoustic simulation, could provide useful visual information, as well as reasonably accurate measurement data.

One of the early proponents of work on time domain numerical methods for acoustics was Bootledooren[1]; whose work involved porting the finite difference time domain (FDTD) and finite volume time domain (FVTD) methods from electromagnetic simulation, for use in low frequency acoustic simulation. This work has been followed on by many researchers such as Murphy[2], Bilbao[3] and Hamilton[4], to expand and improve potential use of these methods. In his thesis doctoral thesis[5], Hill presented a simple and effective implementation of the FDTD for low frequency modelling, that was the basis for the work presented below.

Following key work such as that by Trefethen[6], a number of project have begun to explore the Fourier pseudo-spectral time domain method (PSTD), most notably the OpenPSTD project from Eindhoven University of Technology[7]. Caunce and Angus[8] recognised the limitations of implementing the FDTD method on general purpose graphical processing units (GPGPU) that can be used to improve the speed of FDTD solving, and introduced the potential improvements in execution speed by performing spectral differentiation on a GPGPU using the Fourier PSTD method. This study was the basis for the PSTD work in this study.

Meanwhile in the field of microcontroller development, Doerr[9] produced work on the spare finite difference time domain method for electromagnetic simulation. PIC microcontrollers are often modelled as vastly large electromagnetic simulations of networks of channels, that take large computational resources and a lot of time to simulate. The sparse finite difference time domain method presented by Doerr essentially presents a moving window method of reducing the size of the portion of the domain being solved for at any one time.

Methods such as FDTD present benefits for low frequency simulation over other simulation methods such as ray based and direct calculation, as features such as the modal behaviour of the acoustic system being modelled is accounted for within the solving method. This is due to solving the acoustic wave equation in second order partial differential form; ray based methods assume planar radiation of sound waves, and ignore the radial propagation Eigen-modes of general acoustic systems.

Regardless of the method used for solving in the simulation, it should be possible to setup and solve simulation in reasonable time.

However, large scale and high frequency simulations are difficult to perform using these methods, not only because of the large computational resources required to perform the simulations; often the time required to perform these simulations is also severely limiting. This is due to the various conditions for accuracy and stability that must be met when solving partial differential equations numerically. These conditions are difficult to overcome, but if it is possible to reduce the execution time (and ideally the memory requirements) of a simulation, using time domain numerical methods could become more accessible for less specialist users such as the slightly more academic sound engineer, loudspeaker designers and undergraduate students.

The FDTD method in a more basic implementation involves representing an acoustic system such as a room, as a set of matrices that represent points of pressure and points of velocity within the system. The conceptual distance between the points in the system is defined by the stability of the equations being solved, and the highest frequency of interest. The number of points is also dictated by the size of the system. A wave equation is split up into two reciprocating parts, using velocity points to calculate surrounding pressure points, and pressure points to calculate surrounding velocity points. This is performed in a leapfrog fashion in steps over time, the conceptual length of the step is also determined by the highest frequency of interest and the stability or the solving method. More specifically, the Courant-Freidrichs-Lewey (CFL) condition is a condition that dictates the minimum length of time step and spatial step required for a convergent solution. The equation for Courant number in the one dimensional case is given below:

The maximum value of is determined by the stability of the method being used to solve the PDE, and for a simple explicit FDTD simulation is typically 1. As the amount of memory required for a simulation scales with frequency, it also scales with domain size as the constraints are points per distance. It is difficult to perform large simulations up to high frequency, as the amount of memory required to perform a simulation can quickly become greater than that available in non-specialist computer systems. Another fundamental problem with the FDTD method is the requirement to constantly perform non-contiguous memory accesses to perform calculations. Computer memory access (particularly in the CPU cache) is optimised for contiguous accesses in one particular direction. The FDTD method can require the system to access memory in an orthogonal direction to the optimum around 50% of the time, and also requires the system to index into two large blocks of memory simultaneously.

Two similar methods to FDTD that may execute faster are the PSTD and SFDTD methods. The PSTD method follows a similar form to the FDTD methods in most respects. The differentiation in the method is performed in the frequency domain; each domain matrix is multiplied by the impulse response of an ideal differentiator in the frequency domain, before being used to calculate the new values of the reciprocating field. While this method has the potential to be much faster than FDTD by leveraging the speed of optimised memory access and discrete Fourier transforms, this method requires a PML to overcome Gibbs phenomenon and can suffer from aliasing due to the non-periodic nature of the system being simulated.

The SFDTD method involves windowing around the portions of the domain that have above a threshold of energy. This window is then used as a guide, and only the necessary portions of the domain are computed. This method is still very much in early development and there is little literature in acoustics that have explored this method. As such, a robust and well validated implementation of SFDTD in acoustics has yet to be reported. Further, the method may only be useful for speed improvements before the level of the diffuse field is relatively high i.e. when the early strong reflections are propagating across the domain. This method would also ideally use a high order FDTD stencil that doesn’t suffer from numeric dispersion. - WHAT

Figures backing up the problems

The aim of this paper is to explore the improvements of execution speed of the PSTD and SFDTD methods, over the FDTD method. In the following section of this paper, a series of simulation test cases are described. Following this, the results of the acoustic output and the execution speed of the simulation methods are compared. Finally, the execution speed profile of each method is reviewed, highlighting where the speed bottlenecks occur in each method. The work described in this paper was undertaken using the Matlab language and IDE, as part of an MSc project at the University of Derby.

# Implementation Experiment

In order to test the execution speed of the FDTD, PSTD and SFDTD methods, all three were implemented as functions in Matlab. Code development and speed testing was executed on a PC with the following characteristics:

* Operating System: Windows 10
* CPU: i5 4960k Overclocked to 4.5GHz and 1.227V
* RAM: 16GB DDR3 ram at 3875 MHz
* Motherboard: Asus Gryphon Armour Edition with Z97 chipset
* GPU: Nvidia GTX 1070

This computer system uses standard, easily available consumer grade parts and was configured using inbuilt automatic tools, thus requiring little specialist configuration knowledge.

Initially the FDTD solving method was implemented as a function, based on the work by Hill[5]. First a 2D version, and then a 3D version was implemented. The differentiation in the FDTD method is performed by indexing into discrete points of pressure and velocity potential matrices, and calculating new local values of each based on the old and surrounding values of the related variable at each point in the domain. Following this, a test was executed to check that a stimulus is propagated across the domain without great distortion, spectral shifting or unstable behaviour. The domain setup was a 5m wide by 4m deep by 3m tall rectangle, and had partially absorbing boundaries with an absorption coefficient of 0.45. The maximum analysis frequency was 5kHz. The stimulus used was three sets of 10 cycles of 1kHz windowed tone burst, with a rest period of 3 times the length of the tone. The tone burst stimulus lasted for 0.1s, following which there was 0.1s of silence to allow for decay of the reverberation. The signal source was position as close to 1m away from a corner of the domain as possible. 5 points of the domain (near the corners and the centre) were ‘recorded’ for the length of the simulation.

The FIGURE below shows the normalise source and receiver signals in the time domain, and in the frequency domain using Welch’s power spectral density estimation method built into matlab. The output shows that the frequency of the propagations across the domain is the same as the stimulus, and no high level oscillatory components are present. The time domain behaviour of the simulation appears to show sensible propagation delay between measurement points, with decay that would suggest the simulation is convergent. Using the inbuilt code profiler tools in Matlab, was possible to analyse the performance of the FDTD method and determine where the speed was restricted. The figure below shows the speed of the lines of code in the solving function. The slowest parts of the solving method are the parts where the differentiation is occurring, where the system is having to perform multiple memory accesses to separate large matrices. Managing or reducing these accesses may help speed up solving performance.

NOTE: NOISEFLOOR OF THE MEASUREMENT IS HIDDEN

NOTE: PLOT SCALE

The PSTD method was implemented as a set of Matlab functions in a similar way to the FDTD method, and was based on the work by Caunce & Angus[8]. To implement partially absorbing boundary conditions, work by Spa *et al*[10]. The differentiation in the PSTD method is performed by performing a discrete Fourier transform on 1 dimension of the domain, and multiplying the frequency domain spatial data with the impulse response of an ideal differentiator. The inverse discrete Fourier transform of the differentiated spatial domain data is then used to calculate the new values of the reciprocating field. The differentiation is performed singularly in all spatial dimensions of interest. This method of differentiation may be preferential to the FDTD method, because the differentiation for calculating any one point includes differentiation of all points in the domain that are linearly coupled. This not only increases the order of accuracy of the method, but use of optimised libraries for the Fourier transform and SIMD can be leveraged by the compiler, to increase the speed of computations for the differentiation. The same simulation test as that described above for the FDTD method, was used with the PSTD implementation. The figure below shows the output of the method in the same format. The frequency domain response of the system gives a centre frequency of wave propagation at 1kHz, the same as the stimulus tone. The width and shape of the window is however not an ideal hump, and some aliasing appears in the response. Further, the rate and level of signal decay would indicate that for the same desired absorption coefficients, the absorption is greater than on the FDTD method. Although the quality of the output of the system is questionable, it would appear that the overall performance is acceptable enough to use this algorithm for speed testing.

The SFDTD method was implemented as a set of Matlab functions, based on the same work by Hill mentioned above, and with inspiration from the work of Doerr[9]. Though Doerr’s work in computational electromagnetics is interesting, it uses a list based method for accounting for window shape and position is perhaps not appropriate for an elastic wave system where a diffusely fluctuating field is desirable for calculation. The approach taken for implementing SFDTD in this study was to create a normalising indexing window, based on the absolute pressure of the domain.

The normalised shape of the domain (around a threshold) was then smoothed using a Gaussian image filtering technique, to ensure that the window surrounding points of high enough pressure within the domain are also used for differentiation. This allows wave fronts to propagate naturally across the domain unimpeded by the window itself. The window is used to restrict the number of points within the domain that are calculated, to those around which there is sufficient energy. Due to time constraints with this study, little work was done to optimise the threshold value and 40dB was used throughout the study. Further work should be undertaken to determine ideal smoothing window shapes and threshold values. The next step was to solve the example test using the SFDTD method. The results are displayed below. Due to much of the configuration being identical to the FDTD implementation, the results of the SFDTD simulation were quite similar to the FDTD solution. Same set of domains and maximum frequency used for reasonable comparison. The Amplitude of the reflected wave fronts, at least for early reflections are relatively high compared to the threshold of the window.

The FIGURE below shows the computation time transition for a set of 2D simulations using the SFDTD method. This figure should highlight the potential behaviour of the SFDTD method, reducing early computation times before a steady state diffuse field is calculated. Using this method to reduce computation time may be appropriate when calculating the early reflection behaviour of the acoustic system, when the wave fronts that are propagating are distinct.

# SPEED TEST EXPERIMENT – HOW of the thing measuring

To examine the execution speed performance of each time domain method, each method was used to solve a series of increasingly large rectangular 3D domains. The execution speed for each of the 2000 time-step iterations of each method was measured using the inbuilt Tic/Toc functionality of Matlab. A set of 5 domain sizes were used, the size and number of cells for each time domain method are given in table 1.

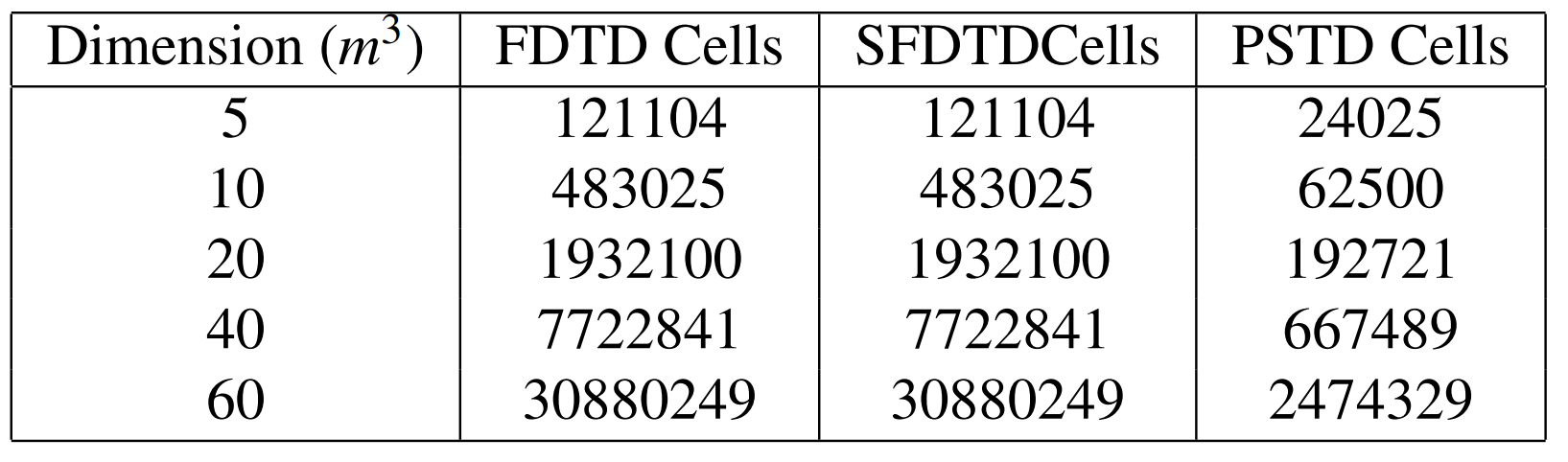


Table 1Set of domain sizes and domain cells for each time domain method

These domain sizes were chosen by choosing a scaling factor up to the maximum domain size and variables that could fit in the computer’s memory. Ideally, some future work on very large domain sizes would handle temporary data storage in binary files on hard-disk, allowing a simulation system to only have necessary large files in memory at any one time.

When running each simulation with the different time domain methods, the supporting code around the simulation kept a similar format and only the execution of the time step solving was measured. No plotting was performed during the speed tests; due to the single threaded nature of Matlabs internal engine, this would have significant performance implications on the overall speed of the simulation execution. The maximum frequency of interest for the simulations was 500Hz, which was chosen due to the size constraints of arrays in memory as per the domain sizes described above.

## Results

* PSTD Results
* Profiler output
* Explanation of what the results show
* Focus on the execution time profile
* SFDTD Results
* Profiler output
* Explanation of what the results show
* Focus on the execution time profile
* The execution speed comparison
  + The two figures below show the mean time step execution speed for each time domain method and domain size. The first figure shows time on a linear scale, and the second figure shows time on a logarithmic scale. – WHAT
  + These results would suggest that the PSTD method gives significantly faster execution times than the FDTD and SFDTD methods. This may be because of both the capacity to leverage optimised computation methods, and the slightly more relaxed domain attributes required for a simulation i.e. number of points required. – WHY
  + However, implementing partially absorbing boundary conditions, handling obstacles, and minimizing aliasing may all be non-trivial work to undertake, where the FDTD and SFDTD methods may be easier to modify and scale for different problems. – WHAT
  + The results also show that on average, the SFDTD method reduced average computation times only the largest domain size, and increased computation time for all other domain sizes. – WHAT
  + This is probably due to the non-optimised implementation of the method and the threshold of the window. Early in the simulation. Before the stimulus has much effect on the domain, the number of extra calculations being undertaken to create the window may be large enough to offset any benefits that such a window might give in terms of total domain used in computation. – WHAT

# CONCLUSION & Further Work

The results of this study may give an indication of some potential to improve execution speed of time domain methods more generally, when using spectral methods for differentiation. A window based method of reducing domain computation area may improve the execution speed in very large simulations, but further work is required to prove this. The steps of differentiation are likely to be the parts of the time domain methods presented that are slowest. Addressing the speed of differentiation by using different matrix sizes, indexing methods or strategies may improve the execution speed of finite difference methods.

The experiment above may have given some idea as to execution speeds of the methods, but has some significant limitations. The implementations of each method presented are certainly not mature, and required a significant amount of improvement and experimentation to provide high quality results, for both measuring acoustic behaviour and optimal performance. None of the methods presented acoustic behaviour that was easy to validate, and the number of time steps used in the final speed tests were quite small. A change in the initial conditions of the speed tests may well have given very different execution speed results, due to the SFDTD methods window function taking a relatively long time to compute.

While this work shows some good evidence that improvements can be made to the execution speed of time domain finite difference style methods, there is a significant amount of further work required to solidify and validate these results. This work may include, but is not limited to:

* Experimenting with the process of SFDTD window calculation.
* Determining the optimal window threshold for the SFDTD method.
* Experimenting with methods to reduce overall number of points required to represent a domain i.e. domain decomposition.
* Examination and improvement of the output of the PSTD method, including the performance of the PML
* Investigation into obstacles and better partially absorbing boundary conditions in the PSTD method

Although significantly more work needs to be done to improve the speed of execution of time domain methods, this work shows potential for the improvement of these speeds. Further development of these methods could provide simply scalable and intuitive tools for simulating and analysing acoustic propagation, without the need for specialist computing equipment.

# References

[1] J. De Poorter and D. Botteldooren, “Acoustical finite-difference time-domain simulations of subwavelength geometries,” *J. Acoust. Soc. Am.*, vol. 104, no. 3, pp. 1171–1177, 1998.

[2] D. T. Murphy, A. Southern, and L. Savioja, “Source excitation strategies for obtaining impulse responses in finite difference time domain room acoustics simulation,” *Appl. Acoust.*, vol. 82, pp. 6–14, 2014.

[3] S. Bilbao, “Optimized FDTD schemes for 3-D acoustic wave propagation,” *IEEE Trans. Audio, Speech Lang. Process.*, vol. 20, no. 5, pp. 1658–1663, 2012.

[4] B. Hamilton and S. Bilbao, “Fourth-Order and Optimised Finite Difference Schemes for the 2-D Wave Equation,” *Research.Ed.Ac.Uk*, vol. 2, no. 2, pp. 1–8, 2013.

[5] A. J. Hill, “Analysis , Modeling and Wide-Area Spatiotemporal Control of Low-Frequency Sound Reproduction,” University of Essex, 2012.

[6] L. N. Trefethen, “Spectral Methods in Matlab,” *Lloydia Cincinnati*, vol. 10, p. 184, 2000.

[7] M. Hornikx, T. Krijnen, and L. Van Harten, “OpenPSTD: The open source pseudospectral time-domain method for acoustic propagation,” *Comput. Phys. Commun.*, vol. 203, pp. 298–308, 2016.

[8] J. A. S. Angus and A. Caunce, “A GPGPU Approach to Improved Acoustic Finite Difference Time Domain Calculations,” in *128th Audio Engineering Society Convention*, 2010.

[9] C. Doerr, “SPARSE FINITE-DIFFERENCE TIME DOMAIN SIMULATION,” US 2014/0365188 A1, 2014.

[10] C. Spa, J. Escolano, and A. Garriga, “Semi-empirical boundary conditions for the linearized acoustic Euler equations using Pseudo-Spectral Time-Domain methods,” *Appl. Acoust.*, vol. 72, no. 4, pp. 226–230, 2011.