Progress In Electromagnetics Research M, Vol. 43, 1–8, 2015

Improvement of Computational Performance of Implicit Finite Difference Time Domain Method

Hasan K. Rouf*

Abstract—Different solution techniques, computational aspects and the ways to improve the performance of 3D frequency dependent Crank Nicolson finite difference time domain (FD-CN-FDTD) method are extensively studied here. FD-CN-FDTD is an implicit unconditionally stable method allowing time discretization beyond the Courant-Friedrichs-Lewy (CFL) limit. For the solution of the method both direct and iterative solver approaches have been studied in detail in terms of computational time, memory requirements and the number of iteration requirements for convergence with different CFL numbers (*CFLN*). It is found that at higher *CFLN* more iterations are required to converge resulting in increased number of matrix-vector multiplications. Since matrix-vector multiplications account for the most significant part of the computations their efficient implementation has been studied in order to improve the overall efficiency. Also the scheme has been parallelized in shared memory architecture using OpenMP and the resulted improvement of performance at different *CFLN* is presented. It is found that better speed-up due to parallelization always comes at higher *CFLN* implying that the use of FD-CN-FDTD method is more appropriate while parallelized.

1. INTRODUCTION

In the finite difference time domain (FDTD) method, the size of time discretization is limited by the smallest spatial step due to the CFL stability condition [1]. To overcome this drawback of explicit FDTD method recently researchers have focused on developing alternative unconditionally stable schemes which can overcome the upper bound on the time discretization and thereby reduce the total simulation time. Crank Nicolson FDTD (CN–FDTD) [2] is an unconditionally stable method and in this work we deal with the frequency dependent version of it (FD-CN-FDTD method) [3, 4]. Since Crank Nicolson based schemes require the solution of computationally expensive large sparse matrices, extensive studies on different computational aspects are essential to make them a promising affordable alternative to the explicit FDTD method.

In this paper we studied different computational aspects of 3D frequency dependent CN–FDTD (FD-CN-FDTD) method [3]. For the solution of the method both direct and iterative solver approaches have been studied in detail in terms of computational time and memory requirements. Furthermore, two best-known iterative methods, Bi-Conjugate Gradient Stabilised (BiCGStab) and Generalised Minimal Residual (GMRES), were compared in terms of the number of iteration requirements for convergence with different *CFLN*, CPU-time and memory requirements. Here $CFLN \equiv \Delta t/\Delta t_{CFL}$ with Δt being the time discretization used in the simulation and Δt_{CFL} denoting the maximum time discretization allowed by the CFL stability condition. As in each time step of the FD-CN-FDTD method, matrix-vector multiplication needs to be performed a number of times, it has a significant contribution in the overall computational performance. This is because matrix-vector multiplications are performed

Received 24 May 2015, Accepted 1 July 2015, Scheduled 8 July 2015

* Corresponding author: Hasan Khaled Rouf (hasan_khaled@fuji.waseda.jp).

The author is with the Department of Applied Physics, Electronics & Communication Engineering, Chittagong University, Chittagong 4331, Bangladesh.

repeatedly until the solution converges. Also it is found that more iterations are required to converge at higher *CFLN* resulting in increased number of matrix-vector multiplications. Therefore, efficient implementation of matrix-vector multiplications have been studied. We also performed parallelization by using OpenMP in a shared memory architecture and studied the resulted speed-up of performance at different *CFLN*. The resulted operations from the increased number of iteration to converge at higher *CFLN* are more suitable for parallelization. Therefore, better speed-up by parallelization is seen at higher *CFLN* which indicates that the use of FD-CN-FDTD method is more appropriate while parallelized.

The paper is organized as follows: Section 2 briefly describes the FD-CN-FDTD method. Performance of Gaussian elimination based direct methods is presented in Section 3 while Section 4 presents that of iterative methods. How the computational efficiency of FD-CN-FDTD method can be further improved is discussed in Section 5.

2. FREQUENCY DEPENDENT CN-FDTD METHOD

In FD-CN-FDTD scheme [3] frequency dependence of single-pole Debye media has been incorporated by auxiliary differential equation method [5]. In material independent form, Maxwell's curl equations are: $\nabla \times \mathbf{E} = -\dfrac{\partial \mathbf{B}}{\partial t}$ and $\nabla \times \mathbf{H} = \dfrac{\partial \mathbf{D}}{\partial t}$, where $\mathbf{E}$, $\mathbf{H}$, $\mathbf{D}$ and $\mathbf{B}$ are the electric field, magnetic field, electric flux density and magnetic flux density, respectively. In frequency domain, the constitutive relationships for isotropic, linear, non-magnetic, single-pole Debye electrically-dispersive media are: $\mathbf{B} = \mu_0 \mathbf{H}$ and $\mathbf{D} = \epsilon_0(\epsilon_\infty + \dfrac{\epsilon_\mathrm{S} - \epsilon_\infty}{1 + \jmath\omega\tau_\mathrm{D}} - \jmath\dfrac{\sigma}{\omega\epsilon_0})\mathbf{E}$, where ϵ_0 and μ_0 are the free-space permittivity and permeability, and ϵ_S is the static permittivity, ϵ_∞ the optical permittivity, τ_D the relaxation time, σ the static conductivity and ω the angular frequency. The previous equation can be re-written as

$$(\jmath\omega)^2\tau_\mathrm{D}\mathbf{D} + \jmath\omega\mathbf{D} = (\jmath\omega)^2\epsilon_0\epsilon_\infty\tau_\mathrm{D}\mathbf{E} + \jmath\omega(\epsilon_0\epsilon_\mathrm{S} + \sigma\tau_\mathrm{D})\mathbf{E} + \sigma\mathbf{E} \tag{1}$$

Mapping frequency domain $(\jmath\omega)^m$ into time domain $\frac{\partial^m}{\partial t^m}$ Eq. (1) can be written as a differential equation in time domain:

$$\tau_\mathrm{D}\frac{\partial^2\mathbf{D}}{\partial t^2} + \frac{\partial\mathbf{D}}{\partial t} = \epsilon_0\epsilon_\infty\tau_\mathrm{D}\frac{\partial^2\mathbf{E}}{\partial t^2} + (\epsilon_0\epsilon_\mathrm{S} + \sigma\tau_\mathrm{D})\frac{\partial\mathbf{E}}{\partial t} + \sigma\mathbf{E} \tag{2}$$

Crank-Nicolson method [6] is applied to Maxwell's curl equations, the constitutive relationships and Eq. (2). Algebraic manipulation of the resultant equations yields an equation with only electric field E^{n+1} terms:

$$\begin{aligned}
&{E_x}^{n+1} - \frac{\xi_1}{\xi_4}\left(\frac{\Delta t}{2}\right)^2\frac{1}{\mu}\frac{\partial^2 {E_x}^{n+1}}{\partial y^2} + \frac{\xi_1}{\xi_4}\left(\frac{\Delta t}{2}\right)^2\frac{1}{\mu}\frac{\partial^2 {E_y}^{n+1}}{\partial x\partial y} + \frac{\xi_1}{\xi_4}\left(\frac{\Delta t}{2}\right)^2\frac{1}{\mu}\frac{\partial^2 {E_z}^{n+1}}{\partial z\partial x} - \frac{\xi_1}{\xi_4}\left(\frac{\Delta t}{2}\right)^2\frac{1}{\mu}\frac{\partial^2 {E_x}^{n+1}}{\partial z^2} \\
&= \frac{\xi_1}{\xi_4}{D_x}^n + \frac{\xi_1}{\xi_4}\frac{\Delta t}{2}\frac{\partial {H_z}^n}{\partial y} + \frac{\xi_1}{\xi_4}\left(\frac{\Delta t}{2}\right)^2\frac{1}{\mu}\frac{\partial^2 {E_x}^n}{\partial y^2} - \frac{\xi_1}{\xi_4}\left(\frac{\Delta t}{2}\right)^2\frac{1}{\mu}\frac{\partial^2 {E_y}^n}{\partial x\partial y} - \frac{\xi_1}{\xi_4}\frac{\Delta t}{2}\frac{\partial {H_y}^n}{\partial z} - \frac{\xi_1}{\xi_4}\left(\frac{\Delta t}{2}\right)^2\frac{1}{\mu}\frac{\partial^2 {E_z}^n}{\partial z\partial x} \\
&+ \frac{\xi_1}{\xi_4}\left(\frac{\Delta t}{2}\right)^2\frac{1}{\mu}\frac{\partial^2 {E_x}^n}{\partial z^2} + \frac{\xi_1}{\xi_4}\left(\frac{\Delta t}{2}\right)^2\frac{\partial {H_z}^n}{\partial y} - \frac{\xi_1}{\xi_4}\left(\frac{\Delta t}{2}\right)^2\frac{\partial {H_y}^n}{\partial z} + \frac{\xi_2}{\xi_4}{D_x}^n + \frac{\xi_3}{\xi_4}{D_x}^{n-1} - \frac{\xi_5}{\xi_4}{E_x}^n - \frac{\xi_6}{\xi_4}{E_x}^{n-1}
\end{aligned} \tag{3}$$

Here ξ_1, ξ_2, ξ_3, ξ_4, ξ_5, ξ_6 are space dependent and functions of the Debye parameters and time discretization Δt as defined in [3]. Cyclic permutation of x, y and z in Eq. (3) yields the remaining two $\mathbf{E}$-field equations. By applying them to each grid position in the computational space a system of linear equations of $\mathbf{Au} = \mathbf{c}$ is set up. $\mathbf{A}$ is very large and highly sparse coefficient matrix, $\mathbf{u}$ represents a vector with the electric field components to be solved, and $\mathbf{c}$ is the excitation vector. When the problem space is homogeneous, $\mathbf{A}$ is symmetric and otherwise asymmetric. The system of equations $\mathbf{Au} = \mathbf{c}$ is solved to find the $\mathbf{E}$ field. Then $\mathbf{D}$ and $\mathbf{H}$ fields are calculated in an explicit manner from the $\mathbf{E}$ field.

3. SOLUTION BY DIRECT METHODS

Sparse matrix system $\mathbf{Au} = \mathbf{c}$ is solved by either *direct* or *iterative* methods. Flowchart of the FD-CN-FDTD method in both approaches is shown in Figure 1. At each time step of the FD-CN-FDTD

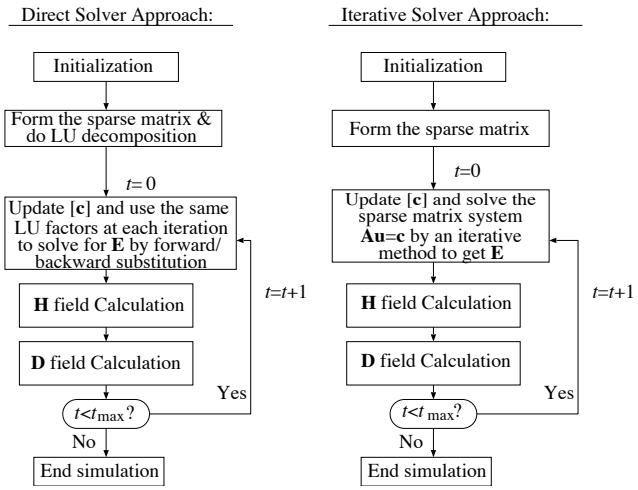


Figure 1. Flowchart of the FD-CN-FDTD method.

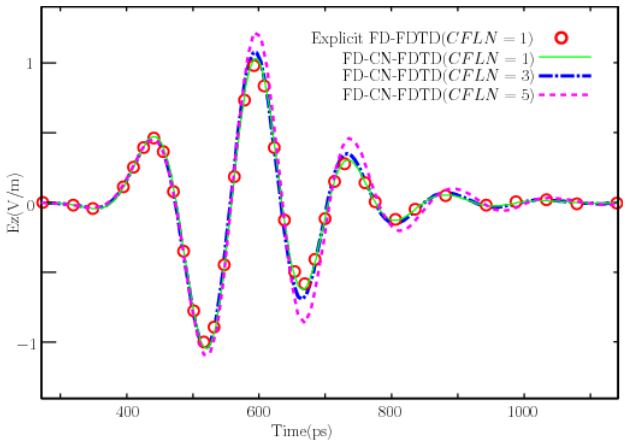


Figure 2. Observations from explicit FDTD and FD-CN-FDTD schemes.

algorithm, a new vector **c** is calculated for the right hand side, while **A** is fixed. Direct solvers factorizes **A** (which dominates the computational time) only once before the beginning of the FDTD iteration. Once factorized, the same factors are repeatedly used at each time step to obtain **u**. For this reason, this method showed the potential to become more computationally efficient than the iterative methods when a large number of FDTD iterations are needed, since at each time step only forward and backward solutions are required.

Numerical tests were performed with the direct solver using sparse Gaussian elimination in the FD-CN-FDTD method. A cubic computational region of size $(30 \times 30 \times 30)$ cells was considered. Half of it was filled with medium 1 ($\epsilon_S = 71.66$, $\epsilon_\infty = 34.58$, $\sigma = 0.49\,\mathrm{S/m}$ and $\tau_D = 5.65\,\mathrm{ps}$) and the other half with medium 2 ($\epsilon_S = 87.34$, $\epsilon_\infty = 49.13$, $\sigma = 0.69\,\mathrm{S/m}$ and $\tau_D = 26.89\,\mathrm{ps}$). A z-directed dipole was placed at (10, 15, 15) in medium 1, with a time evolution of a modulated Gaussian pulse centered at 3 GHz. Signals were recorded 10 cells away at (20, 15, 15) in medium 2. A uniform spatial sampling was taken ($\Delta x = \Delta y = \Delta z = 10^{-3}\,\mathrm{m}$). The first 600 time steps of the E_z field component at the observation point are shown in Figure 2 computed both with standard explicit frequency dependent (FD-)FDTD with $CFLN = 1$, and with FD-CN-FDTD with $CFLN = 1$, 3, 5. Good agreement between the signals from the unconditionally stable FD-CN-FDTD method beyond the CFL limit ($CFLN > 1$) and explicit FD-FDTD was observed. For this numerical problem, CPU time required for LU decomposition was 633 minutes and average CPU time per iteration was 6.489 seconds when $CFLN = 1$ on dual AMD Opteron 280 with 8GB of memory. When the FDTD space was filled with homogeneous material of media 1 or when $CFLN > 1$ there were no significant differences in these values.

Although direct solvers are robust and reliable, they are computationally more expensive than iterative solvers, unless parallelized, and require excessively large memory. For example, a 30^3 cells computational space (like the one described in the previous numerical test) solved by the direct solver in double precision using sparse Gaussian elimination requires 2.4 GB of memory whereas iterative solvers like BiCGStab and GMRES require only 62 MB and 65 MB, respectively. Thus for practical problems iterative solvers should be used.

4. SOLUTION BY ITERATIVE METHODS

In the iterative solver approach, an initial estimation of the solution is made, and by repeated use of a certain algorithm (depending on the iterative method used) this initial estimation approaches the desired solution. Although there exist many iterative methods, a certain method may work well for one problem but not for another. To choose an effective iterative method from the best known methods for which extensive computational experience has been gathered, the flowchart of Figure 3 can be used [7]. In this work, GMRES(m) that restarts every m iterations [8] and BiCGStab of [9] were used.

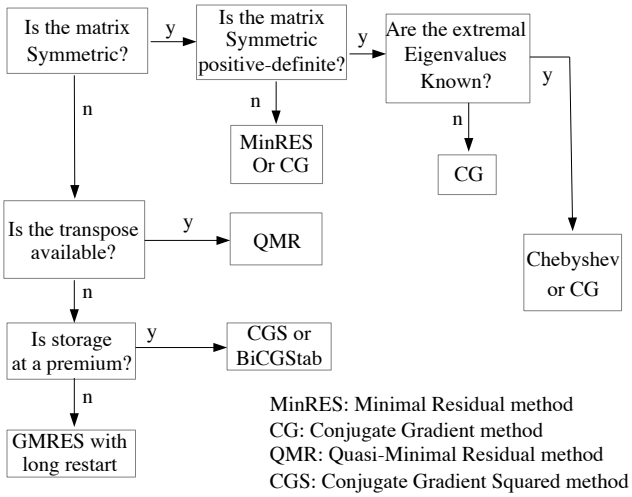
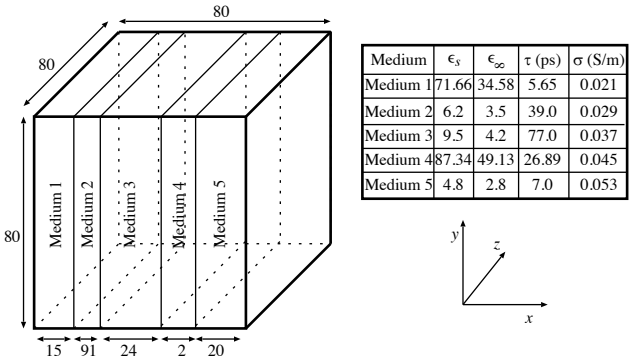


Figure 3. Choosing an effective iterative method.



Medium	ϵ_s	ϵ_∞	τ (ps)	σ (S/m)
Medium 1	71.66	34.58	5.65	0.021
Medium 2	6.2	3.5	39.0	0.029
Medium 3	9.5	4.2	77.0	0.037
Medium 4	87.34	49.13	26.89	0.045
Medium 5	4.8	2.8	7.0	0.053

Figure 4. Computational environment for numerical tests.

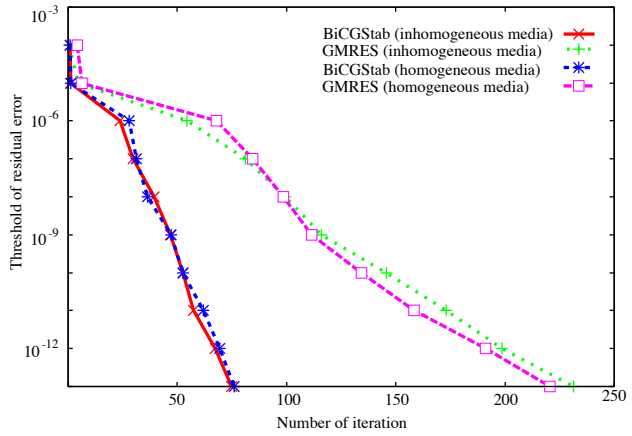


Figure 5. Residual error versus number of average iteration required by BiCGStab and GMRES for homogeneous and inhomogeneous cases ($CFLN = 20$).

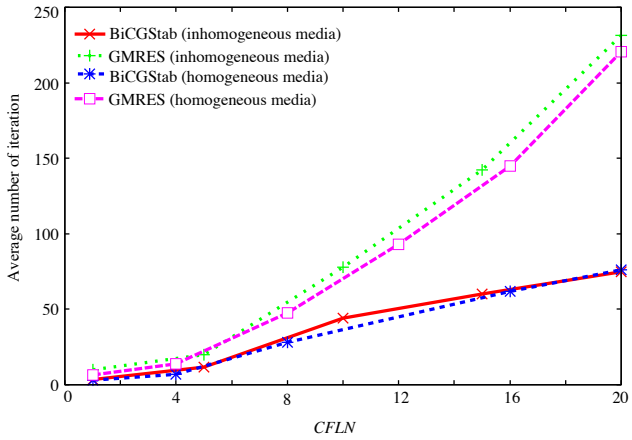


Figure 6. Average number of iterations required for different $CFLN$ with residual error $= 10^{-13}$.

We compare the performances of BiCGStab and GMRES for two cases. The first one consists of an inhomogeneous medium, in a cubic space of size 80^3 cells, with 5 different media as shown in Figure 4. The second one involves the same cubic space of the previous case, now filled with a homogeneous medium with Debye parameters $\epsilon_S = 6.2$, $\epsilon_\infty = 3.5$, $\sigma = 0.029\,\mathrm{S/m}$ and $\tau_D = 39.0\,\mathrm{ps}$. In both cases a z-directed dipole hard source with a time variation given by a Gaussian pulse centered at 6.9 GHz, with 4.94 GHz bandwidth ($-3\,\mathrm{dB}$ decay) was placed at the centre of the computational space. Spatial sampling was uniform: $\Delta x = \Delta y = \Delta z = \Delta = 10^{-3}\,\mathrm{m}$. The time step is taken equal or above the CFL stability condition of the explicit FDTD. The level of accuracy in waveform compared with explicit frequency-dependent FDTD is the same as the one presented in Figure 2. Figure 5 shows the convergence pattern for $CFLN = 20$, plotting the residual error as a function of the average number of iterations required to achieve a specified accuracy. For example, to make the residual error lower than 10^{-8} BiCGStab requires about 45 iterations whereas GMRES requires about 97 iterations in both homogeneous and inhomogeneous cases. The convergence rate of the solvers is weakly affected by homogeneity. For a modest value of $CFLN$ iteration numbers would certainly be lowered than those showed in Figure 5 ($CFLN = 20$).

Figure 6 shows how the average number of iterations, required by BiCGStab and GMRES to converge, increases with the CFL number. Stopping criteria in this case was 10^{-13} and the reason for selecting this small value of convergence tolerance is, in FD-CN-FDTD, unlike frequency-independent CN-FDTD, $\mathbf{D} = \epsilon\mathbf{E}$ is used and therefore $\mathbf{D}$ can have a value of such small order because of ϵ (permitivitty). GMRES stagnates when convergence tolerance is below 10^{-13} while BiCGStab can work even at a lower convergence tolerance. Both solvers require more iterations to converge as *CFLN* goes up but the rate of increase of iteration numbers with *CFLN* is higher for GMRES than for BiCGStab. Homogeneity does not affect significantly this rate, particularly, for BiCGStab. The change of iteration number with *CFLN* for convergence tolerance values from 10^{-12} to 10^{-3} can be assumed from Figure 5. In FD-CN-FDTD, the total number of iterations required to complete the simulation decreases with *CFLN*, but the increase of computational costs per iteration with *CFLN*, as shown in Figure 6, can undermine this positive effect unless the solution is very efficient. Figure 7 plots the CPU time required by BiCGStab and GMRES as a function of *CFLN*. The case of stopping criterion of 10^{-13} was employed and simulated to reach a fixed time instant by letting the code run for $1200/CFLN$ time steps on a dual AMD Opteron 280 with 8 GB of memory. Observe that the CPU-time decreases with *CFLN*, for both solvers, although GMRES requires more CPU time than BiCGStab. It should be noted that for a much smaller problem (of size 30^3 cells), as mentioned in Section 3, the direct solver requires 633 minutes for LU decomposition. Therefore, BiCGStab and GMRES are far more efficient than the direct solver. Table 1 presents the memory required by the two solvers for three different computational space sizes. GMRES always requires more memory than BiCGStab.

From all the above, we can conclude that BiCGStab outperforms GMRES in computational efficiency. This finding is in contrary to that of [10] which reports that GMRES is the fastest for the frequency-independent CN-FDTD scheme presented there. The work of [10] is based on Maxwell's curl equations in material-independent form while FD-CN-FDTD additionally involves Eq. (2) which has 2nd order time derivative terms. The FD-CN-FDTD involves nine field components in place of six for CN-FDTD, and the sparsity pattern of the former has more bands than the latter. Apart from this, the concerned problem of simulation, implementation, optimization and parameters tuning have an obvious influence in concluding which solver is the most efficient.

Table 1. Memory required by BiCGStab and GMRES for different computational spaces.

Computational Size (cells)	40^3	60^3	80^3
BiCGStab	145 MB	487 MB	1.1 GB
GMRES	151 MB	507 MB	1.2 GB

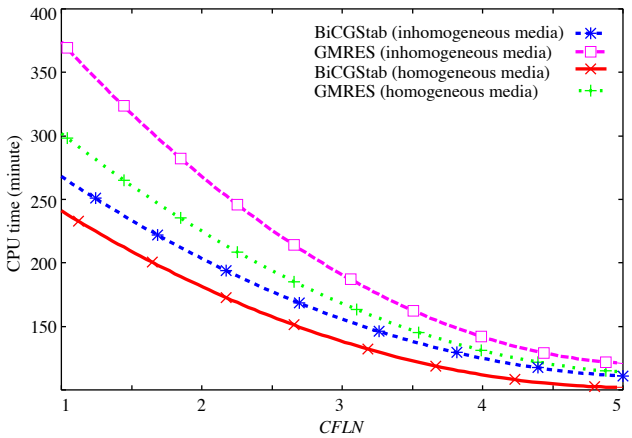


Figure 7. CPU time required by BiCGStab and GMRES for different *CFLN*.

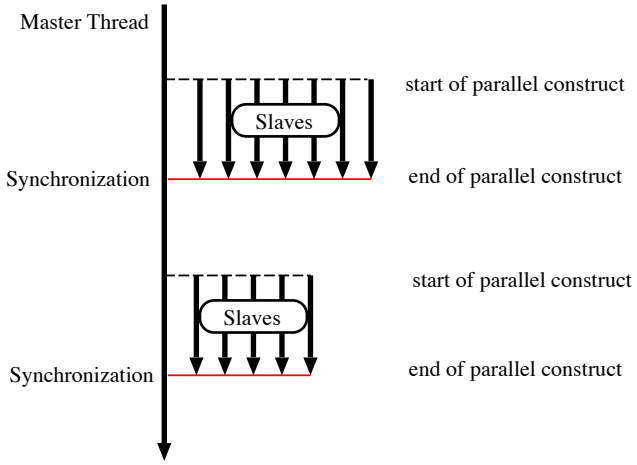


Figure 8. Execution model of OpenMP.

5. IMPROVEMENT OF COMPUTATIONAL EFFICIENCY

Repetitive matrix-vector multiplications are required in each simulation step before getting the solution and such multiplications account for the most significant portion of the computations during the solution by iterative methods. Therefore, we need to opt for an efficient matrix-vector multiplication technique in order to improve the overall efficiency. To solve the sparse system, $\mathbf{Au} = \mathbf{c}$ we used Harwell Subroutine Library (HSL) packages [11] which use their own routine `mc-65` for matrix-vector multiplications. For efficient implementation we further studied other sparse matrix-vector multiplication subroutines, for example, `amux` from SPARSEKIT [12] and observed improved performance. For the numerical tests described in Section 4 (Figure 4), when the simulation is run for 1200 time steps with $CFLN = 1$ using Intel Fortran Compiler on AMD Athlon 64 X2 4200+ Dual Core Processor, the performance of matrix-vector multiplication subroutines, `mc-65` and `amux` are shown in Table 2. In Table 2, the performance is shown in terms of the percentage of total CPU time used by a particular subroutine and total CPU time required by the whole FD-CN-FDTD code. `amux` accounted for 42.4% of the total time spent to run the code in comparison to 47.8% for `mc-65`. Depending on the choice of matrix-vector multiplication subroutine, the percentage of time used by the BiCGStab solver subroutine, `mi26ad`, is also affected. The most noticeable observation in Table 2 is the reduction of total CPU time when `amux` is used instead of `mc-65`. `amux` further simplifies the implementation as it can do the matrix-vector multiplication when the sparse matrix is stored in usual compressed sparse row (CSR) format, while `mc-65` further requires the matrix in HSL's own format (`HSL_ZD11`).

Figure 6 shows that in solving the sparse matrix system, more iterations are required to converge at higher *CFLN*. This means that the number of matrix-vector multiplications would increase at higher *CFLN*. For the same numerical test mentioned above, the performance of `amux` and `mi26ad` subroutines at different *CFLN* is shown in Table 3. These two are the most computationally expensive subroutines used in the implementation of the FD-CN-FDTD method. Table 3 shows that with the increase of *CFLN*, the percentage of total CPU time used by both of the subroutines increases. Since these routines consume much of the CPU time, while doing parallelization (in the next paragraph) they should be adeptly taken care of. As preconditioners are usually used by matrix-vector multiplication, this study is also relevant if a suitable preconditioner is used during the solution.

By doing parallelization, the requirements of large memory and long CPU time by the FD-CN-FDTD method can be tackled efficiently. We performed parallelization using OpenMP in a shared

Table 2. Performance with matrix-vector multiplication subroutines *mc-65* and *amux* ($CFLN = 1$).

Subroutine	% of total CPU time used by the subroutine	Total CPU time
`mc-65`	47.8	84 min 31 sec
`mi26ad`	18.5	
`amux`	42.4	70 min 49 sec
`mi26ad`	25.3	

Table 3. Performance of the two most computationally expensive subroutines at different *CFLN*.

CFLN	% of total CPU time	
	`amux`	`mi26ad`
1	42.4	25.3
2	44.4	28.1
3	48.7	30.7
4	51.8	31.3
5	56.4	32.5

Table 4. Speed-up by the OpenMP code on 32 cores at different $CFLN$.

$CFLN = 1$		
Code type	CPU time	Speed-up
Serial	39 hr 53 min	1
OpenMP	15 hr 11 min	2.63
$CFLN = 3$		
Code type	CPU time	Speed-up
Serial	37 hr 38 min	1
OpenMP	6 hr 26 min	5.85
$CFLN = 5$		
Code type	CPU time	Speed-up
Serial	34 hr 32 min	1
OpenMP	4 hr 3 min	8.53

memory architecture. In the parallel execution of the code, OpenMP uses the fork-join model as shown in Figure 8. An OpenMP programme begins as a single thread of execution. When it encounters a parallel construct, it creates ('forks') the required number of threads and becomes the master thread of all the threads. Programme statements within the parallel construct are executed in parallel by each thread of the team of threads. At the end of the parallel construct, threads synchronize back ('join') and only the master thread continues the execution. There can be any number of parallel regions and different number of threads in each parallel region, as shown in Figure 8.

Numerical modelling of the human body in the FD-CN-FDTD method, as described in [13, 14], was implemented in OpenMP for the cases of $CFLN = 1$, 3 and 5. The geometrical features of the human body were read from a 2-mm resolution male voxel model and for each voxel (representing a certain tissue) the corresponding single–pole Debye parameters were mapped. As the section above the upper chest was modelled, the size of the FD-CN-FDTD computational space was $320 \times 160 \times 220$ voxels from the top of the head. The source excitation was z-directed modulated Gaussian pulse centred at 3 GHz. Table 4 shows the achieved improvement by the OpenMP parallelization over the serial code when it was run for $1050/CFLN$ time steps. With the OpenMP code, the CPU time has been greatly reduced when $CFLN = 1$ and as expected, the most significant reduction in CPU time has been achieved when $CFLN$ is higher. However, the scaling is not perfect for the number of threads (32) the code uses because in OpenMP, usually, it is hard to obtain perfect speed-ups even when the parallelization is done correctly [15]. Understanding the details of underlaying hardware and using vendor-supplied parallel mathematical operation libraries may improve the performance to some extent.

An interesting observation in Table 4 is the increase of speed-up by OpenMP parallelization at higher $CFLN$. From the speed-up by OpenMP for $CFLN = 1$, 3 and 5 are 2.63, 5.85 and 8.53, respectively, it is found that better speed-up always comes at higher $CFLN$, indicating that the use of the FD-CN-FDTD method is more appropriate while parallelized. This may be resulted from the fact that the operations involved in the increased iteration numbers to converge at higher $CFLN$, as shown in Figure 6, are possibly more suitable for parallelization.

6. CONCLUSION

Different solution techniques and computational aspects of three-dimensional unconditionally stable FD-CN-FDTD method have been studied in this paper. For the solution, both direct and iterative solver approaches have been studied in detail in terms of computational time and memory requirements. Direct methods have the potential to become more computationally efficient if the problem size is very small and a large number of FDTD iterations are needed. Because after factorization of the coefficient matrix at the beginning at each time step only forward and backward solutions are required. However, they can not be used for large practical problems since they are computationally more expensive and require

excessively large memory. Therefore, two best-known iterative methods, GMRES and BiCGStab, were studied in terms of the number of iteration requirements for convergence with different *CFLN*, CPU-time and memory requirements. As in each time step of the FD-CN-FDTD method, matrix-vector multiplications need to be done repeatedly until the solution converges, it has a significant contribution in the overall computational performance. Also it is found that more iterations are required to converge at higher *CFLN* which results in increased number of matrix-vector multiplications. Therefore, efficient implementation of matrix-vector multiplications has been studied in order to improve the overall efficiency. FD-CN-FDTD code was parallelized by using OpenMP in shared memory architecture and it was found that better speed-up always comes at higher *CFLN* indicating that the use of the FD-CN-FDTD method is more appropriate while parallelized.

REFERENCES

1. Taflove, A. and S. Hagness, *Computational Electrodynamics: The Finite-difference Time-domain Method*, 3rd Edition, Artech House, Boston, MA, 2005.
2. Yang, Y., R. Chen, and E. Yung, "The unconditionally stable Crank Nicolson FDTD method for three-dimensional Maxwell's equations," *Microwave and Optical Technology Letters*, Vol. 48, 1619–1622, 2006.
3. Rouf, H. K., F. Costen, and S. G. Garcia, "3-D Crank-Nicolson finite difference time domain method for dispersive media," *Electronics Letters*, Vol. 45, No. 19, 961–962, 2009.
4. Rouf, H. K., F. Costen, S. G. Garcia, and S. Fujino, "On the solution of 3-D frequency dependent Crank-Nicolson FDTD scheme," *Journal of Electromagnetic Waves and Applications*, Vol. 23, No. 16, 2163–2175, 2009.
5. Joseph, R., S. Hagness, and A. Taflove, "Direct time integration of Maxwell's equations in linear dispersive media with absorption for scattering and propagation of femtosecond electromagnetic pulses," *Optics Letters*, Vol. 16, No. 18, 1412–1414, 1991.
6. Crank, J. and P. Nicolson, "A practical method for numerical evaluation of solutions of partial differential equations of the heat-conduction type," *Mathematical Proceedings of the Cambridge Philosophical Society*, Vol. 43, 50–67, 1947.
7. Barrett, R., M. Berry, et al., *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM Press, Philadelphia, 1993.
8. Saad, Y. and M. H. Schultz, "GMRES: A generalized minimal residual method for solving nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, Vol. 7, 856–869, 1986.
9. Van der Vorst, H., "BiCGSTAB: A fast and smoothly converging variant of BiCG for the solution of nonsymmetric linear systems," *SIAM Journal on Scientific and Statistical Computing*, Vol. 13, 631–644, 1992.
10. Yang, Y., R. Chen, et al., "Application of iterative solvers in 3D Crank-Nicolson FDTD method for simulating resonant frequencies of the dielectric cavity," *Asia-Pacific Microwave Conference*, 1–4, 2007.
11. Numerical Analysis Group, Rutherford Appleton Laboratory Harwell Subroutine Library, (HSL), 2007 for Researchers, http://www.hsl.rl.ac.uk/hsl2007/hsl20074researchers.html.
12. Saad, Y., "SPARSKIT: A basic toolkit for sparse matrix computations (Version 2)," Research Institute for Advanced Computer Science, NASA Ames Research Center, 1994, http://www-users.cs.umn.edu/ saad/software/SPARSKIT/sparskit.html.
13. Rouf, H. K., "Unconditionally stable finite difference time domain methods for frequency dependent media," Ph.D. Thesis, The University of Manchester, UK, 2010.
14. Rouf, H. K., F. Costen, and M. Fujii, "Modelling EM wave interactions with human body in frequency dependent Crank Nicolson method," *Journal of Electromagnetic Waves and Applications*, Vol. 25, Nos. 17–18, 2429–2441, 2011.
15. Chandra, R., R. Menon, et al., *Parallel Programming in OpenMP*, Morgan Kaufmann, 2001.