Paper

Electromagnetic Field Analysis using a Parallel FDTD Algorithm in Extremely Large Areas for Mobile Communication

Non-member Glen Rodriguez (Toyohashi University of Technology)
Member Yasumitsu Miyazaki (Toyohashi University of Technology)

This paper deals with the simulation of big problems using FDTD method. FDTD consumes a lot of memory and computing time. Most research focuses on processing time, but memory is the first obstacle in simulating big problems. The common FDTD equations are reformulated into a big system of linear equations. A new recursive algorithm is proposed for dealing with this problem, and the parallel version of the algorithm is also introduced. Some numerical comparasions are done, and conclusions about the advantages of the algorithm and the required improvements to make it practical.

Keywords: mobile communication, FDTD method, parallel processing, system of linear equations

1. Introduction

The growing use of wireless communications, especially cellular telephony, has created a new interest in the study of the propagation and scattering of electromagnetic waves. This has driven the research about the characteristics of wave propagation and scattering in big areas. FDTD method should be convenient, but it demands a great amount of memory and computer time. Until now, work has been done to improve the speed by using FDTD parallel algorithms, but no effort has been made in being able to solve problems needing more memory than available in parallel computer enviroments. This limit the size of problems FDTD can deal with. This paper presents the initial research actually done regarding this matter. A big size FDTD solver could allow the simulation of many interesting behaviours in realistic size models. The main objective is to solve the problem of the memory, by adding memory in different computers and lessening the amount of memory needed.

2. Yee's Formulation of FDTD

The Finite Difference Time Domain method (also known as FDTD) is a computational method for solving numerically different electromagnetic phenomena. It is based in the Maxwell equations in differential format. In linear, isotropic nondispersive materials (materials with field independent, direction independent and frequency independent properties) the simple constitutive relationships can be used, obtaining:

$$\nabla \times \mathbf{E} = -\mu \frac{\partial \mathbf{H}}{\partial t} \quad \cdots\cdots (1)$$

$$\nabla \times \mathbf{H} = \varepsilon \frac{\partial \mathbf{E}}{\partial t} + \sigma \mathbf{E} \quad \cdots\cdots (2)$$

With a central difference, leap-frog scheme, the FDTD formulation is obtained. The present research

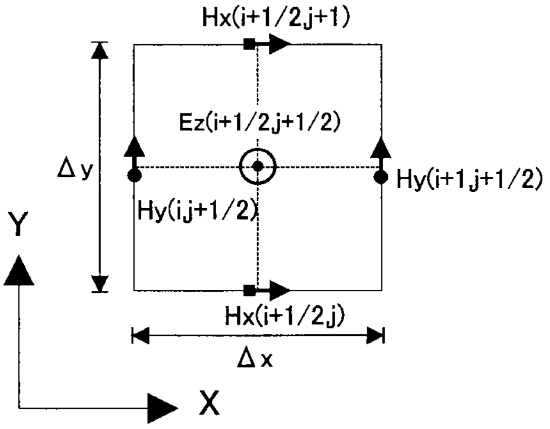


Fig. 1. Fields in the 2-D, TM problem

deals with the 2-D case, which can be obtained from the above mentioned equations. Assuming that there is no variation in the geometry, the materials nor in the electromagnetic excitation field in the z-direction, then all partial derivatives with respect to z become zero. This 2-D problem will use the TM formulation. The Maxwell's curl equations (1) and (2) would be converted into the following (also see fig. (1)):

$$\frac{\partial H_x}{\partial t} = -\frac{1}{\mu}\frac{\partial E_z}{\partial y} \quad \cdots\cdots (3)$$

$$\frac{\partial H_y}{\partial t} = \frac{1}{\mu}\frac{\partial E_z}{\partial x} \quad \cdots\cdots (4)$$

$$\frac{\partial E_z}{\partial t} = \frac{1}{\varepsilon}\left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - \sigma E_z\right) \quad \cdots\cdots (5)$$

The standard formulation of the Yee algorithm for this case is:

$$E^{n+1}_{z(i,j)} = (\frac{1-\sigma_{i,j,k}\Delta t/2\varepsilon_{i,j,k}}{1+\sigma_{i,j,k}\Delta t/2\varepsilon_{i,j,k}})E^{n}_{z(i,j)} +$$
$$\frac{\Delta t/\varepsilon_{i,j,k}}{1+\frac{\sigma_{i,j,k}\Delta t}{2\varepsilon_{i,j,k}}}\cdot\frac{1}{\Delta x}[H^{n+1/2}_{y(i+1/2,j)} - H^{n+1/2}_{y(i-1/2,j)}] -$$
$$\frac{\Delta t/\varepsilon_{i,j,k}}{1+\frac{\sigma_{i,j,k}\Delta t}{2\varepsilon_{i,j,k}}}\cdot\frac{1}{\Delta y}[H^{n+1/2}_{x(i,j+1/2)} - H^{n+1/2}_{x(i,j-1/2)}] \cdots (6)$$

$$H^{n+1/2}_{x(i,j)} = H^{n-1/2}_{x(i,j)} -$$
$$(\frac{\Delta t}{\mu_{i,j,k}\Delta y})[E^{n}_{z(i,j+1/2)} - E^{n}_{z(i,j-1/2)}] \cdots\cdots\cdots (7)$$

$$H^{n+1/2}_{y(i,j)} = H^{n-1/2}_{y(i,j)} +$$
$$(\frac{\Delta t}{\mu_{i,j,k}\Delta x})[E^{n}_{z(i+1/2,j)} - E^{n}_{z(i-1/2,j)}] \cdots\cdots\cdots (8)$$

From this point, the constant coefficients of these equations will be defined as: C_a, C_x and C_y are the coefficients in eq. (6), D_y is the coefficient in eq. (7) and D_x is the coefficient in eq. (8). These constant coefficients only depend on the material m in each cell, for example: $D_x(m) = \frac{\Delta t}{\mu(m)\Delta x}$ if the material in the cell is material number m.

3. Reformulation as a Field System of Linear Equations

The Yee's formulation, with the appropiate modifications, can be expressed as a matricial equation. The system is shown in eq. (9)–(11), X_i is a vector with all the variables E_z, H_x, H_y in time step i, and in most simulations $X_0 = \vec{0}$. The coefficient matrix A can be expressed as a block banded matrix with two block components, matrix I and the sparse matrix A_1, see eq.(3), and the constants' matrix b consist of small matrices ϕ_i where $\phi_i = [0, 0, ..., f_k(i\Delta x), ..., 0]'$, $0 < i \leq T$, f_k is the E_z field in the emitter k at time step i.

$$X_t = [H_{x\,1,1,t}; H_{y\,1,1,t}; E_{z\,1,1,t}; ...; E_{z\,p,q,t}]' \cdots (9)$$

Let be A_1 fixed matrix $\Rightarrow X_{t+1} = A_1 X_t$ $\cdot\cdot$ (10)

Equation (10) shows the well proven fact that, with the exception of the boundary conditions, all the variables H_x, H_y and E_z of time step t+1 are dependant only of the variables of time step t. And the set of operations that is done from one time step to the next always is the same, represented by the multiplication to matrix A. If the eq. (10) is expressed for all t=0,1,...T-1, we get a serie of operations $X_1 = A_1X_0$, $X_2 = A_1X_1$, ... , $X_T = A_1X_{T-1}$, that can be expressed as a system of linear equations:

$$AX = b \text{ where } \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (11)$$

$$A = \begin{bmatrix} I & 0 & \cdots & 0 & 0 \\ -A_1 & I & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & I & 0 \\ 0 & 0 & \cdots & -A_1 & I \end{bmatrix} \cdots\cdots\cdots (12)$$

$$X = \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ X_{T-1} \\ X_T \end{bmatrix} \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (13)$$

$$b = \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{T-1} \\ \phi_T \end{bmatrix} \cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots\cdots (14)$$

In eq.(9), p is the number of cells in the x direction, and q is the number of cells in the y direction. In eq.(11), T is the number of time steps through the simulation, and $\phi_1 = A_1X_0$.

If we use the equations (7) and (8), matrix A would not be triangular, and that would mean a process of factorization should be necessary. To avoid these burden, we must change both eq. (7) and (8) in order to get a triangular matrix. For cells that are not in the boundaries of the simulation domain, we substitute the values of the various E^{n+1}_z using eq.(6), and then eq.(7) and (8) become as eq. (15),(16). Where m_1 is the material of cell (i,j), m_2 is the material of cell (i,j-1) and m_3 is the material of cell (i-1,j). In the case of cells near a boundary, the application of the 1st order Mur ABC generates 13 other equations of the same type[10].

$$\begin{aligned} H^{n+1}_x(i,j) &= H^n_x(i,j) - D_y(m_1)[C_a(m_1)E^n_z(i,j) \\ &+C_x(m_1)(H^n_y(i+1,j) - H^n_y(i,j)) \\ &-C_y(m_1)(H^n_x(i,j+1) - H^n_x(i,j)) \\ &-C_a(m_2)E^n_z(i,j-1) \\ &-C_x(m_2)(H^n_y(i+1,j-1) - H^n_y(i,j-1)) \\ &+C_y(m_2)(H^n_x(i,j) - H^n_x(i,j-1))] \cdots\cdots (15) \end{aligned}$$

$$\begin{aligned} H^{n+1}_y(i,j) &= H^n_y(i,j) - D_x(m_1)[C_a(m_1)E^n_z(i,j) \\ &+C_x(m_1)(H^n_y(i+1,j) - H^n_y(i,j)) \\ &-C_y(m_1)(H^n_x(i,j+1) - H^n_x(i,j)) \\ &-C_a(m_3)E^n_z(i-1,j) \\ &-C_x(m_3)(H^n_y(i,j) - H^n_y(i-1,j)) \\ &+C_y(m_3)(H^n_x(i-1,j+1) - H^n_x(i-1,j))] \quad (16) \end{aligned}$$

The inclusion of point-source antennas in the grid is done using the soft source approach. A soft source is a superposition of the source waveform (in the electric or magnetic components) with the fields existing in the source region. Soft sources let propagating signals naturally pass through the source region without abnormal reflections. For example, if the signal is expressed as an electric field, the equation for the component should be:

$$\begin{aligned} E^{n+1}_z(i,j) &= C_a[E^n_z(i,j)] \\ &+C_x[H^n_y(i+1,j) - H^n_y(i,j)] \\ &-C_y[H^n_x(i,j+1) - H^n_x(i,j)] + f(n\Delta t) \cdots (17) \end{aligned}$$

Where the function f is the signal expressed in the same units as E (V/m), operating in this special point (i,j). In this research, the antenna has been modeled as a single point soft source. In our case, we will define the source as a current J_{src}, so our function f becomes:

$$E^n_{src} = f(t) = f(n\Delta t) = \frac{\Delta t}{\varepsilon} J_{src}(n\Delta t) \cdots\cdots (18)$$

4. Algorithm of Field Analysis

As mentioned before, a problem with the parallel FDTD using spatial partition is the inconvenience for large problems because it has to communicate a lot of data across processors at the same time. But the other problem is the need for memory; this kind of parallelization does not change the need of about 130 Gb of memory for a 2-D problem with size 100,000 × 100,000 cells (120 Gb for components or field variables and around 10 Gb for material, boundary conditions, etc.). Even using parallel processing and the memory in all processors, this amount is far from the possibilities of most researchers. At the present time, most computers (PC and workstations) normally have a 256 Mb memory, and they can upgrade to 1 Gb, but it is very expensive. That means we would need more than 100 very expensive computers, or 10 high performance computers to get the neccesary memory. The memory available in all the facilities where our programs run, are only 18.5 Gb.

After considering this, a different approach was created, that is, the formulation of the FDTD method like a huge system of linear equations. At first, it will be explained as a sequential algorithm.

From this point, the unknown components of **E** and **H** will be represented as x variables, and the following notation is defined: $x_{t,i,j,c}$ represents $H^t_x(i,j)$ if c=0, $H^t_y(i,j)$ if c=1 and $E^t_z(i,j)$ if c=2. Other notation will be $x_{\mathcal{L}}$ which is equivalent to some $x_{t,i,j,c}$, where $\mathcal{L} = (t-1)(3pq) + 3qx + 3y + c$; this formula has a inverse, that is, given some $\mathcal{L}$, the values of t, x, y and c can be calculated by division and modules. The equation solver deal with the second notation.

The sparse equations (the rows of A) are represented by arrays, containing information about the column, the row and the coefficient (value in A or any related matrix). Some of these arrays are shown in fig. (2), where the array "column" stores the number of the column associated with the "coefficient" inside the matrix, and the array "value" stores the value of the variable after it is solved. These arrays are big and are stored across the processors. The row and some auxiliar arrays are smaller and do not require special storage. This special method to store sparse matrizes does not keep any zero (most of the elements of the matrix are zeros) but the price is that the access to any element of the matrix is not direct.

The substitution is carried from the equation for $x_{\mathcal{L}} = x_{T,i_0,j_0,c_0}$ (greatest time step T), many times, changing the original equation of the following type, where each x^a_b are the equivalent to some variable $x_{T-a,i,j,c}$ and c^a_b are constant coefficients:

$$x_{\mathcal{L}} + c^1_1 x^1_1 + c^1_2 x^1_2 + ... + c^1_{z_1} x^1_{z_1} = b^1_{\mathcal{L}} \cdots\cdots (19)$$

	processor 1		processor 2			processor P	
INDEX	1	2	3	4	...	z−1	z
COLUMN	sp(1)	sp(2)	sp(3)	sp(4)	...	sp (z−1)	sp(z)
COEFFICIENT	c(1)	c(2)	c(3)	c(4)	...	c(z−1)	c(z)
VALUE	v(1)	v(2)	v(3)	v(4)	...	v(z−1)	v(z)

Fig. 2. Sparse arrays in the algorithm

Into an equation of type:

$$x_{\mathcal{L}} + c^2_1 x^2_1 + c^2_2 x^2_2 + ... + c^2_{z_2} x^2_{z_2} = b^2_{\mathcal{L}} \cdots\cdots (20)$$

Where all x^2_b are variables of the kind $x_{T-2,i,j,c}$, as explained. It must be noted that the number of terms in the last equation is greater than in the first: $z_{k+1} > z_k$. This process is carried many times, until the equation becomes:

$$x_{\mathcal{L}} + c^n_1 x^n_1 + c^n_2 x^n_2 + ... + c^n_{z_n} x^n_{z_n} = b^n_{\mathcal{L}} \cdots\cdots (21)$$

For some "n", the size of the array ($z_n = 6n^2 + 4n - 1$ for H_x or H_y and $6n^2 - 2n - 1$ for E_z) representing this equation becomes very large, and then we take the variables $x_{T-n,i,j,c}$ one by one, create a new array and process this variable in the same way as we processed $x_{\mathcal{L}}$. Each stage of processing is called a level, where the stage processing $x_{\mathcal{L}}$ until getting all the $x_{T-n,i,j,c}$ is called level 1; the stage beginning with the processing of any $x^n_b$is level 2, and so on. Each level uses a distinct array to represent the respective equations.

The algorithm is recursive in nature, that is, a process that repeat itself. The pseudocode of the algorithm called "solution" is as shown in fig. (3). The initial call would use the parameters (T,i,j,c). The implementation as a recursive program in most programming languages (C, Fortran) is inefficient, so it was implemented as an equivalent non-recursive algorithm controlled for many conditions and counters, simulating the flow of the recursion.

5. Parallel Algorithm of FDTD

5.1 Previous research and the most common Parallel FDTD Most common parallelization is achieved by spatial partition or decomposition. There is a lot of research in this kind of parallelization (7)~(9). We have also done experiments with 3D problems and spatial parallelization, simulating the scattering of a small sphere, and the table (1). shows the results in time and speed-up. The 180^3 simulation was too big for only one processor, so the speed up has been calculated respect the speed using 4 processors and then multiplied by 4. It was run using a network of workstations attached with a LAN Ethernet 100 Mbps and the message passing library MPI (Message Passing Interface), over TCP/IP. It can be observed that, for any number of processors, there are some very small problems that need very few computations and relatively more communication per

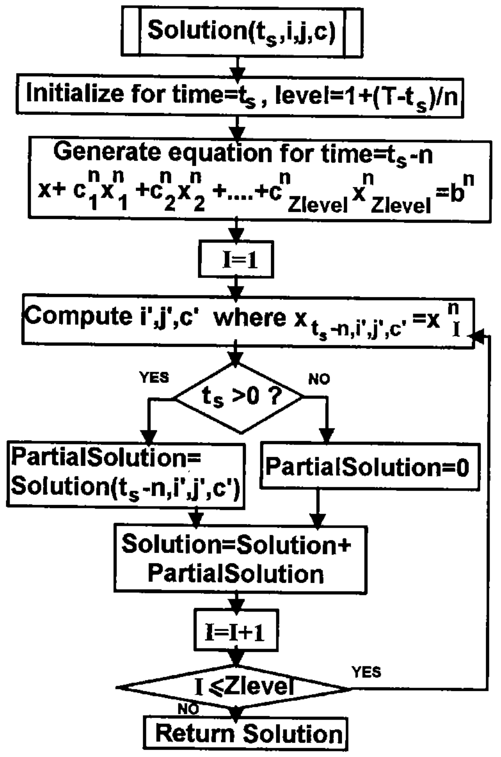

Solution(t_s,i,j,c)
Initialize for time=t_s, level=1+(T-t_s)/n
Generate equation for time=t_s-n
x+ c_1^n x_1^n +c_2^n x_2^n +....+c^n_Zlevel x^n_Zlevel =b^n
I=1
Compute i',j',c' where x_{t_s-n,i',j',c'} =x^n_I
YES
t_s >0 ?
NO
PartialSolution= Solution(t_s-n,i',j',c')
PartialSolution=0
Solution=Solution+ PartialSolution
I=I+1
I ≤Zlevel
YES
NO
Return Solution

Fig. 3. Flowchart of the recursive algorithm

time step. These problems have a very bad speed-up. There is also a range of medium size problems, and they have a good speed-up. Finally, there are relatively big problems where again the speed-up is not good, because each time step all processors try to transfer and to receive a huge amount of data almost at the same time; this causes a busy network, collision of IP packets and slowdowns in the process.

From this data, it can be shown that the most common parallel FDTD method used until now can not scale conveniently for large problems in a network of workstations using a slow network as the ethernet 100Mbps. For better networks (for example, Myrinet) better speed-up for big problems should be expected. So, a secondary objective should be to improve the scaling and, if possible, the processing time.

There are super-linear speed-up but also very poor speed-up. We think that the better explanations for these unexpected results are:

(1) Virtual memory access when running in only one processor, because the problem with 144^3 cells (more than 71 Mb) is big for a workstation with 128 Mb of physical memory. Generally, the free physical memory after the loading of the operating system and other basic software is between 50 and 60 Mb in this workstation. But when the work is distributed among 4 or more processors, virtual memory is not required.

(2) For the small problem of 60^3 cells, we suspect that there is cache miss because the memory required is 5.1 Mb. In only one processor, the data cache size available is only 1 Mb and the RAM should be accessed 6 times by time step. But the group of four processors has to store only 1.3 Mb per processor, and it has to access the RAM only 2 times by time step.

(3) For small problems but more processors (60^3 cells and 16 processors), the speed up is worse because the ratio computation/communication is low.

(4) For the bigger problem, the cause of the poor speed up appears to be the network. The speed of the networks is 100 Mbps, and when he problem is big (180^3 cells = 139.9 Gb) and we use 16 processors, the amount of data being interchanged between one processor and the surrounding processors is $6 \times 4 \times (4 \times 45 \times 90 + 2 \times 90 \times 90)$=777.6 Kb. That is, a total of 12 Mbytes = 96 Mbits is interchanged per time step. This generates can create a bottleneck in the network, but packet collisions, timeouts, etc.

Table 1. Results for spatial-parallel FDTD
(*) Estimated using 4 processors' time

	16 cpu		12 cpu		4 cpu	
Size	Time	Speedup	Time	Speedup	Time	Speedup
180^3	1h11m	*3.67	1h34m	*2.76	1h5m	n/a
144^3	25.8m	18.20	12.9m	36.56	32.2m	14.61
60^3	4.6m	4.60	1.32m	16.16	2.96m	7.20

5.2 Parallel FDTD solving the big System of Linear Equations

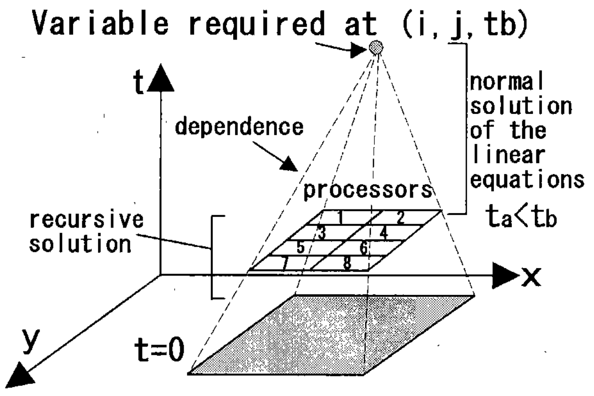
The parallelization is done by distributing the variables in the sparse arrays between the different processors. Each processor stores one subset of variables, as shown in fig. (2). As the solution of the system of equations progresses, the eq. (21) for solving variable $x_{\mathcal{L}}$ can be expressed as eq. (22), with p= number of processors, where the each small summation inside of the big summation is performed by a separate processor, and later the parallel program gathers all the partial additions.

$$\begin{aligned} x_{\mathcal{L}} &= -(c_1^n x_1^n + c_2^n x_2^n + \ldots + c_{z_n}^n x_{z_n}^n) + b_{\mathcal{L}}^n \\ &= -\sum_{k=1}^{z_n} c_k^n x_k^n + constant \\ &= -\sum_{r=1}^{p} \Big(\sum_{k=1}^{z_n^p} c_k^n x_k^n\Big) + constant \quad \cdots\cdots\cdots \quad (22) \end{aligned}$$

The solution process and the point when the parallelization is done is shown in fig. (4), where the vertical axis is the time, and the initial process is to solve the value of $x_{t_b,i,j,c}$; as it is shown, after n time steps, the solution is broken down into the solution of many other variables $x_{t_a,i,j,c}$, where $t_a = t_b - n$, and at this point the distribution of variables begins. The communication between processors is low compared with the common parallelization of the FDTD method. For this parallelization to be effective, it is better to distribute the work load proportionally to the speed of each processors. This is an important advantage of this method when used in Network of Workstations, where not all the processors are the same type. If the speed of each processor is S_i, for $1 \leq i \leq p$, then each processor should be in charge of approximately:

$$\frac{S_i}{\sum_{i=1}^{p} S_i} z_{level} \text{ variables} \cdots\cdots\cdots\cdots\cdots\cdots \quad (23)$$

Each processor has the information about the speed of all the processors in the parallel environment. Then, for any summation, they can calculate the lower and upper index of their respective sub-summation. Finally, the master processor gets the partial summations and having the constant being calculated beforehand, gives the solution for $x_{\mathcal{L}}$.



Variable required at (i, j, tb)
t
dependence
normal solution of the linear equations
processors
ta<tb
recursive solution
1
2
3
4
5
6
7
8
x
y
t=0

Fig. 4. Parallelization and partition of the data

This algorithm has been improved using some techniques in order to avoid repeating some calculations, by storing some results from one step of the calculations in one level to the next calculation of the same level, and copying these results instead of recalculating them. For example, in solving the variable $x_{\mathcal{L}1} = E_z^T(i,j)$, and after that, the variable $x_{\mathcal{L}2} = E_z^T(i,j+1)$, for some time step $T >> 1$, using our algorithm with a number of time steps calculated at each level n=100, it would be necessary to run the recursive solution at first for many variables, around 6×10^4 for each $x_{\mathcal{L}}$, but all these variables are the same except around 400. The improvement has been to store the values of the common variables of $x_{\mathcal{L}1}$ with $x_{\mathcal{L}2}$, then from $x_{\mathcal{L}3}$ with $x_{\mathcal{L}3}$, etc. Some improvement has been achieved, but yet, this algorithm is slower than normal FDTD. The goal of further revisions of the algorithm is to improve the processing time.

In our algorithm, the number of components of cells of the FDTD simulation (the field variables) are not changed, but not all the cells are calculated at the same time and the algorithm reduces the memory for the working variables. The algorithm can work without all the 120 Gb of field variables, only with a subset of field variables at different time steps. The algorithm only needs to store data for no more than $6n^2 + 4n - 1$ variables x per level, and the number of levels is T/n, where T=greatest time step in the computation (that is, duration of the time in the simulation). Each variable x is stored using sparse matrices techniques, and requires basically 3 variables in the program, for column, coeffcient and value (see fig. (2)). With single precision floating-point variables, total basic memory use is in the order of $24Tn$. For simulations as large as $10^5 \times 10^5$ cells, a reasonable duration of the simulated wave propagation should be 10^6 time steps. If each level computes n=50 time steps, it should be necessary to have 20000 levels, and each level should be able to store a little bit more than $4 \times 3 \times (6n^2 + 4n - 1) = 182388$ bytes. Making some conservative approximations about the extra variables, this 182388 bytes increase to 200 kb and this algorithm needs:

$$\text{Memory required} = 20000 \times 200 \text{ kb} = 4 \text{ Gb} \quad (24)$$

For the small model of next section (100 × 100 cells), normal FDTD requires 120 Kb of memory. Our algorithm uses a level with n=30 time steps, with 1 level, then we have $4 \times 3 \times (6n^2 + 4n - 1) = 66228$ bytes (plus extra variables, around 70 kb) per level and:

$$\text{Memory required} = 1 \times 70 \text{ kb} = 70 \text{ kb} \cdots\cdots \quad (25)$$

This amount of memory for the small model is 70/120=58.3% of the memory required by the common parallelization of the FDTD method, but it is only 4/130=3.08% of the memory for the big model of size $100,000 \times 100,000$. These amounts of memory are the memory directly used by the algorithm, not the total memory needed in the computers, because there is some amount of memory not available for computations (OS memory, devices, daemons, services); not available memory depends on the machine and OS configuration. The principle in which this method is based is to "decrease something by increasing the computational bandwidt" (See (4), (11)). In other areas of numerical simulation, this principle has been used to decrease the number of operations and the processing time by modifying the algorithm, including a new process that reduces the number of time steps or increases the time of the cell, but that requieres some pre or post-processing. In our case, this principle means that memory used can be reduced by a modification in the FDTD algorithm.

6. Numerical Field Results

For the simulation of electromagnetic propagation, we assumed an source antenna as a point source, with a sinusoidal current in the z direction as follows:

$$J_z^{i,n} = J_{max}^i sin 2\pi f t; \text{ for } t < t_{max} \cdots\cdots\cdots \quad (26)$$

where t is the time elapsed, J_{max}^i is the amplitude and f is frequency. The current stops after some $t_{max} = 30\Delta t$. The parameters are:

- Frequency of source : 850 or 950 MHz
- Cell size, δ: 0.02 m
- Time increment Δt: 37.7 ps
- Relative permittivity(building): 3.0
- Conductivity of building, σ: 0.005 S/m
- Current amplitude: 1000.0 A/m
- Pulse duration: 1.81 ns (for Gaussian pulse only)

To test the correctness of out algorithm, we simulated a small 2-D problem, as shown in fig. (5), with propagation from the antenna at point (20,50) emitting a signal like eq. (26), f=950 MHz, $J_{max}^i = 1000$, t_{max}=30Δt=1.131ns, and a block of concrete in the center. The objective for the continuation of this research is to simulate big models like shown in fig. (6), with the same parameters as the small one, the same kind of dielectric (concrete) and rectangular shapes. But for doing this, further improvements must be made in the algorithm.

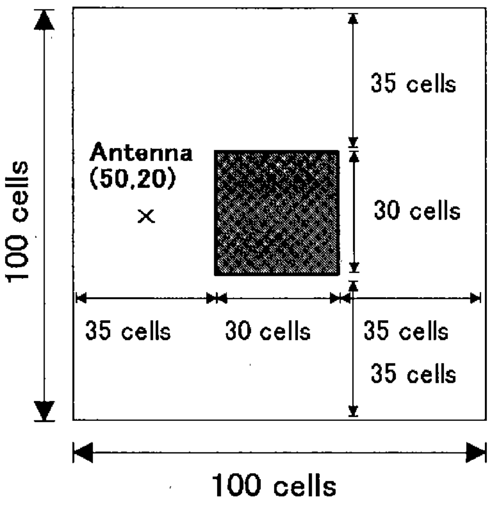


Fig. 5. Small model

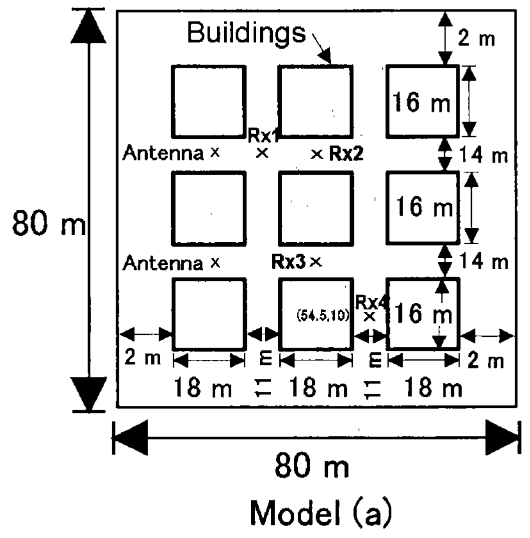
The distribution of buildings in the medium and big models was chosen in order to compare with results in (2). The buildings are hollow structures, the walls have a thickness of 25 cm. and the inside is free space. Only the medium size problem has been similated in that paper, and the time it took to simulate it was about one week, with the common, non-parallel FDTD. Those results can be used as reference.

The propagation of the wave by the small cube made of concrete is shown from fig. (7) to fig. (9) for different times. The propagation through the concrete at a different speed is shown in all these figures, and the reflections are more notorious at time=$60\Delta t$=2.262ns. In fig. (10), the normal, sequential, FDTD method is compared side by side, showing that there are very good agreement between both results.

7. Conclusion and future directions

We have introduced a new parallel 2D FDTD algorithm for the simulation of big areas, beginning with the mathematical foundations of our algorithm and the parallel solution of huge sparse systems of linear equations. It is important to remark that our algorithm, from the point of view of the formulas, is exactly equivalent to the Yee formulation, but expressed in matricial way and with an aditional process to change the matrix into a triangular one. The algorithm allows us to define soft sources of the emmiting antenna.

The main advantage of this algorithm is that it needs less memory than the normal parallel FDTD algorithm, only 3% of memory. It is very convenient for parallel environments with different kinds of processors, because the load balancing is very simple and effective, while the normal FDTD algorithm generally must divide the simulation in similar domains, being more restricted to a completely homogeneous systems, with the same processor speed and memory. Our method has been validated against a small model; the behaviour of the wave corresponds to the expected, and the results are also



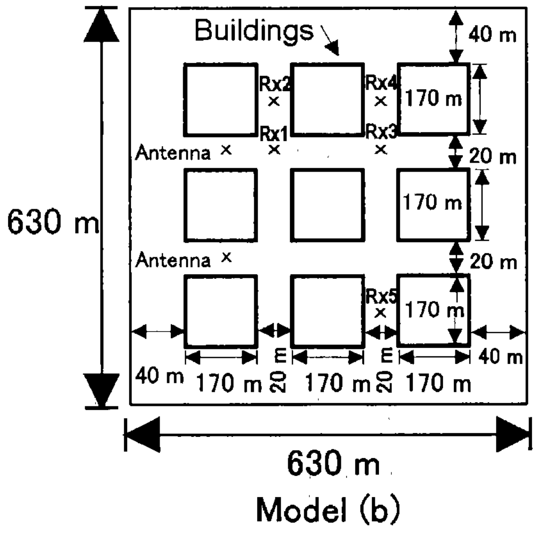


Fig. 6. (a) Medium and (b) Big urban area models

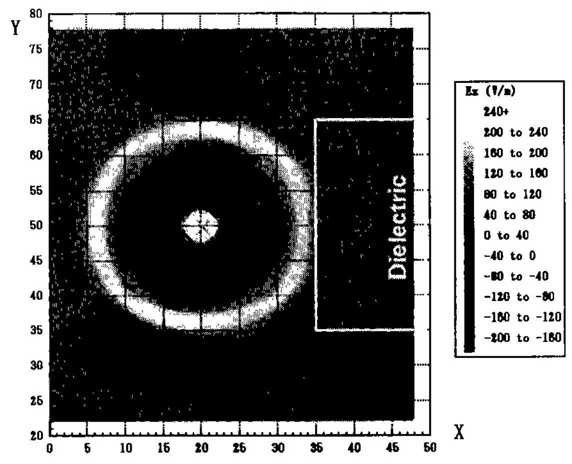


Fig. 7. E_z in time=30Δt=1.131ns

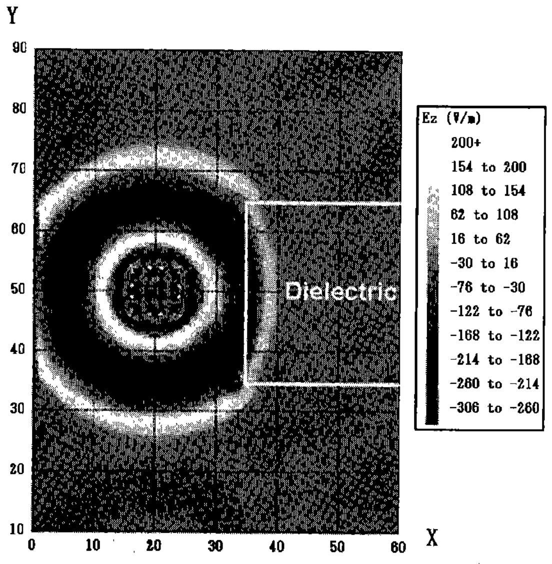


Fig. 8. E_z in time=45Δt=1.6965ns

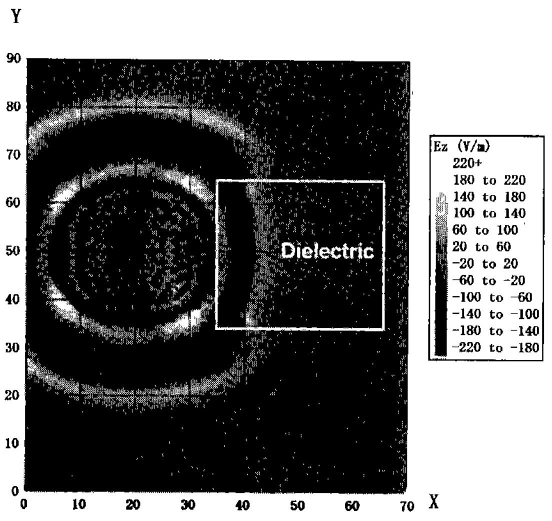


Fig. 9. E_z in time=60Δt=2.262ns

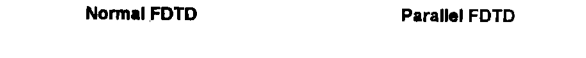


Fig. 10. Comparison for E_z in time t=2.262ns

consistent when they are compared to a sequential, normal FDTD simulation.

The main disavantage of our method is the time it takes for practical values of T. That is, the processing time grows (apparently) in an exponential way with the duration T of the simulation. This increased processing time is in exchange for saving memory, because the algorithm does not store all the data it computes permanently; instead it stores it and later must discard it to make space for other data, but some data already calculated is needed some time after it was discarded. Another reason for the slowness is the binary search it executes as part of the operations in a storage structure for sparse arrays. Some steps have been taken to address this problem, but have not been enough. That is the reason for not presenting simulation of medium or big areas. More research will be done in the future to cope with this problem. There are some promising alternatives like the hybridization of the normal FDTD method inside of the process that solves the linear equation, or the factorization of the matrix A in many simpler matrices.

(Manuscript received March 26, 2001, revised August 16, 2001)

References

(1) K.S.Yee, "Numerical Solution of initial boundary value problems involving Maxwell's equations in isotropic media", IEEE Trans. Ant. & Prop., Vol.14 N.4, 1966, pp.302–307

(2) P.Selormey, Y.Miyazaki,"Electromagnetic Interference Characteristics by Group of Buildings in High-speed Mobile Communication" EMCJ99, 1999, pp. 71–78.

(3) G.Rodriguez, Y.Miyazaki, "Analysis of Electromagnetic Scattering in Large Areas using a Parallel FDTD method", Proc. 2000 Japan-China Joint Meeting on Optical Fiber Science and Electromagnetic Theory, OFSET 2000, Dec.2000, pp.163–166

(4) A.Fijany, M.Jensen and others, "A massive Parallel Computation Strategy for FDTD", IEEE Trans. Ant. & Prop. Vol.43 N.12, Dec.1995, pp.1441–1449

(5) M.Osano, K.Nakajima, M.Tanimoto, "A Partially Solving Method (PSM): A new efficient method for a large system of linear equations", Bulletin of the Electrotechnical Laboratory, Vol.60, No.2, Feb.1996, pp.93–101.

(6) K.Morgan, P.Brookes et al, "Parallel processing for the simulation of problems involving scattering of electromagnetic waves", comput. Methods Appl. Mech. Engrg. 152, 1998, pp.157–174

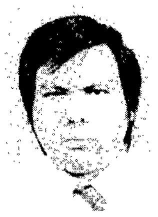
(7) D.Rodohan et al, "A Distributed Implementation of the Finite Difference Time-Domain (FDTD) Method", Int. Journal of Numerical modelling: Electronic Networks, Devices and Fields, Vol.8, 1995, pp.283–291

(8) U.Anderson, "Parallelization of a 3D FD-TD code", Proc. 4th International Workshop of Applied and Parallel Comp. '98, LNCS pp.12–19, 1998

(9) Z.Liu et al, "Techniques for Implementation of the FDTD Method on a CM-5 Parallel Computer", IEEE Ant. & Prop. Magazine, Vol.37 N.5, Oct.1995, pp.64–71

(10) G.Mur, "Absorving boundary conditions for the finite-difference approximation of the time-domain electromagnetic field equations", IEEE Trans. Electromagnetic Compatibility, Vol.23 N.4, Nov.1981, pp.377–382

(11) E.Miller, "Solving bigger problems- By Decreasing the operation count and Increasing the Computational Bandwidth", Proceedings of the IEEE, Vol.79, No.10, Oct.1991, pp.1493–1504



Glen Rodriguez (Non-member) Was born in Lima, Peru, on March 27, 1974. He received the B.Sc. degree in Systems Engineering from National University of Engineering, Peru, in 1994, and the M.E. degree in Information and Computer Science Engineering from Toyohashi University of Technology in 2001. His research interest are Parallel Processing, Mobile Communication and Electromagnetic Waves Propagation and Scattering.

Yasumitsu Miyazaki (Member) Was born in Nagoya, Japan, on February 4, 1941. He received the B.E. M.E. and D.E. degrees in Electronic Engineering from Nagoya University, Nagoya, in 1963, 1965 and 1969, respectively. He has engaged in research in the field of electromagnetic waves including waves in milimeter waveguides, optical fibers, integrated optics, electromagnetic scattering and difraction. He also studies biological phenomena of electromagnetic fields and optical neural computing. He is presently Professor of the Department of Information Engineering, Toyohashi University of Technology, since 1981. From January 1973 to January 1975, he stayed in Institute of High Frequency-Technics, Technical University of Braunschweig, West Germany, and engaged in research on electromagnetic fields for optical communications. In 1996, he was a guest Professor in IHFT of Technical University of Berlin, Germany. He received Yonezawa Memorial Paper award in 1970. He is a member of IEEE, IEE of Japan, Japanese Society of Medical Electronics and Biological Engineering and the Japan Society of Applied Physics.