SIMULATION OF THREE-DIMENSIONAL FINITE-DIFFERENCE TIME DOMAIN METHOD ON LIMITED MEMORY SYSTEMS

W.J. Buchanan and N.K. Gupta

Napier Polytechnic, UK

1 INTRODUCTION

It is the intention of this paper to discuss techniques in the modelling of Electromagnetic Fields using Finite Difference Time Domain (FDTD) Methods applied to 80x86-based PCs or limited memory computers.
Electrical modelling of structures is currently carried out mainly in the frequency domain using empirical methods or by treating structures as two-dimensional objects. Until recently the application of finite difference methods were applied to super computers such as the Cray XMP and Cray II. The power of modern workstations and microcomputers offer large storage area, large memory capacity and fast CPU times. Thus it is now possible for full time domain simulations to be carried out on complex structures on base model workstations and microcomputers.

2 THEORY

For the sake of completeness an overview of the Finite Difference Time Domain method is given in this section.

2.1 Equations

Taking Maxwell's curl equations and taking central differences the following equations can be generated.

$$\boxed{\begin{aligned}
H^{n+\frac{1}{2}}_{xi,j,k} &= H^{n-\frac{1}{2}}_{xi,j,k} + \frac{\Delta t}{\mu\Delta z}(E^{n}_{yi,j,k} - E^{n}_{yi,j,k-1}) - \frac{\Delta t}{\mu\Delta y}(E^{n}_{zi,j,k} - E^{n}_{zi,j-1,k})\\
H^{n+\frac{1}{2}}_{yi,j,k} &= H^{n-\frac{1}{2}}_{yi,j,k} + \frac{\Delta t}{\mu\Delta x}(E^{n}_{zi,j,k} - E^{n}_{zi-1,j,k}) - \frac{\Delta t}{\mu\Delta z}(E^{n}_{xi,j,k} - E^{n}_{xi,j,k-1})\\
H^{n+\frac{1}{2}}_{zi,j,k} &= H^{n-\frac{1}{2}}_{zi,j,k} + \frac{\Delta t}{\mu\Delta y}(E^{n}_{xi,j,k} - E^{n}_{xi,j-1,k}) - \frac{\Delta t}{\mu\Delta x}(E^{n}_{yi,j,k} - E^{n}_{yi-1,j,k})\\
E^{n+1}_{xi,j,k} &= E^{n}_{xi,j,k} + \frac{\Delta t}{\varepsilon\Delta y}\left(H^{n+\frac{1}{2}}_{zi,j+1,k} - H^{n+\frac{1}{2}}_{zi,j,k}\right) - \frac{\Delta t}{\varepsilon\Delta z}\left(H^{n+\frac{1}{2}}_{yi,j,k+1} - H^{n}_{yi,j,k}\right)\\
E^{n+1}_{yi,j,k} &= E^{n}_{yi,j,k} + \frac{\Delta t}{\varepsilon\Delta z}\left(H^{n+\frac{1}{2}}_{xi,j,k+1} - H^{n+\frac{1}{2}}_{xi,j,k}\right) - \frac{\Delta t}{\varepsilon\Delta x}\left(H^{n+\frac{1}{2}}_{zi+1,j,k} - H^{n}_{zi,j,k}\right)\\
E^{n+1}_{zi,j,k} &= E^{n}_{zi,j,k} + \frac{\Delta t}{\varepsilon\Delta x}\left(H^{n+\frac{1}{2}}_{yi+1,j,k} - H^{n+\frac{1}{2}}_{yi,j,k}\right) - \frac{\Delta t}{\varepsilon\Delta y}\left(H^{n+\frac{1}{2}}_{xi,j+1,k} - H^{n}_{xi,j,k}\right)
\end{aligned}}$$

2.2 Problem formation

The structure to be analysed is fitted into an *xyz* axis. The z-direction being the height, the y-direction the length, and the x-direction the width (see Figure 1). Next the problem is given a cell structure by splitting each axis into a series of nodes. The distance between the nodes in the x-direction is defined as delta X, in the y-direction delta Y and the z-direction delta Z. A Gaussian pulse is applied to the source [2]. This pulse is used because its frequency spectrum is also Gaussian and will provide frequency domain information from zero frequency to the desired cut-off frequency by adjusting the width of the pulse. A raised cosine can also be used [3].

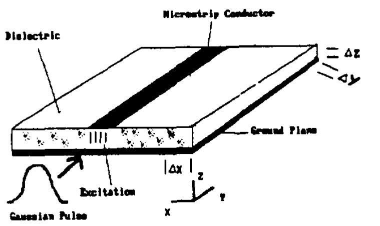


Fig 1: Microstrip line

The electric field applied to the source consists only of a z component and is Gaussian.

$$E_z = \exp-\frac{(t-t_o)^2}{T^2}$$

Simulations are usually carried out for several hundred or even several thousand time steps (or iterations). After a complete time solution a Fourier Transform is taken at a specific node to determine the frequency response of the structure. For more information on the Finite Difference Time Domain method [1-5].

3 METHODS OF SIMULATION

Some methods of increasing speed of simulation and/or increasing memory allocation are given in this section.

3.1 Memory Allocation

Memory allocation on a PC using DOS causes some problems. 80286 processors are able to address 16 Megabytes (4GBytes for 80386) of physical address space and 1 Gigabyte (64 Terabytes for 80386) of virtual address space per task. Unfortunately modem compilers can only access up to a maximum of 1 MByte. It is necessary in most simulations to use the maximum amount of usable memory. 8086 compatible PCs can address up to a maximum of 1 MByte of RAM. Some of this memory is used for applications such as graphics, interrupt vectors, etc. Computers based around the 8086/286/386 have a segmented memory architecture. They are designed to directly address only 64k of memory at a time. Memory address pointers on a 8088 compatible machine can be near pointers or far pointers. The near pointer is a 16 bit address value and can address up to 64k of data, whereas a far pointer is a 32bit address and can address up to 1 MB of memory. The far pointer is made up of a segment address and an offset address (segment:offset). In the large model far pointers are used for both the code and the data, giving both a 1Mb range. It is also possible (in 'C') to force a variable to be a far pointer using the **far** type. The PC uses around 400 kBytes for interrupts, graphics[1], etc leaving around 600 kBytes for the program and data storage.

3.1.1 Memory allocation for FDTD method

The 3D FDTD method uses four three dimensional arrays to store Electric and Magnetic field components. Two of these arrays store the previous time step (for E and H fields), while the other two store the current time interval. Each of these cells has three field components in the *x,y* and *z* directions. The static declaration is poor in both speed and in the size of memory it can use. A much better solution is to use a two-dimensional array of pointers, which are used to point to a block of data. Thus only the offset address of the data added to the pointer gives the memory location. In the example of a 3 dimensional array of 24 by 40 by 10 (for the x,y and z nodes), 24*40 (480) pointers can be allocated. Each of these pointers point to a 10 values (the z-values). The total number of bytes to store all the arrays in this example will be 406,200 (24 * 50 * 7 * 4 * 3). This leaves around 100 kB for the FDTD program.

3.2 Multitasking with segmentation

The FDTD method provides a good application of multitasking.

A technique to use the maximum amount of the memory is to segment the problem then run the FDTD processor on each of the segments as a separate task. Each segment will have access to a maximum of 1MB of memory. For example if a PC has 8MBytes of memory, the problem can be segmented into eight separate tasks. Each of these tasks will wait on boundary conditions if they are not available.

1 608K is available for programs, the rest is used by BIOS, Fixed Disk ROM, Video Buffer, Interrupt Vector Table, Communications Data, Disk handling.

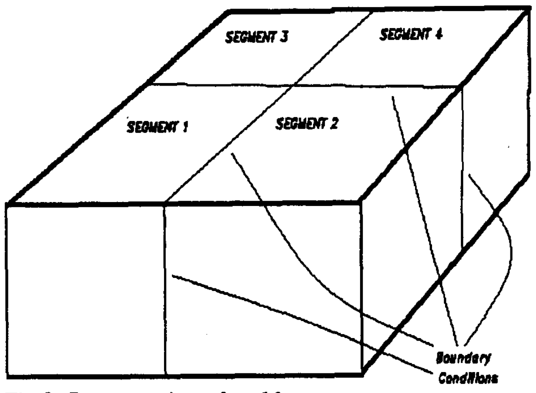
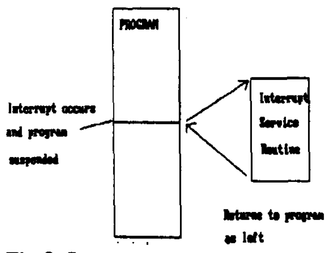


Fig 2: Segmentation of problem

Multitasking splits into two categories non-preemtive and pre-emptive. In a non-preemtive system the task must voluntarily give up the processor before the next task can run. The running task is said to be pre-empted by the task waiting to run. Operating systems for the PC such as DOS and Windows 3.0 are non-preemtive. In a pre-emptive system the task is 'kicked' off the processor by another task. The scheduling used in this application is the round robin time slicing algorithm. Tasks which are ready to run are put in a queue. On a particular event, in this case the timer tick interrupt, the next task in the queue pre-empts the running task and the task pre-empted is put to the end of the 'ready' queue. DOS does not handle multitasking. It is required that processes are allocated a certain amount of time on the processor and therefore it is logical to use the PCs built-in timer routines to keep track of time. By extending the clock-tick Interrupt Service Routine (ISR) it is possible to keep the routine doing its old job of updating the system clock and refreshing memory, but add the feature of task switching control.


PROGRAM
Interrupt occurs and program suspended
Interrupt Service Routine
Returns to program as left
Fig 3: Interrupt pauses program

Turbo C and C++ supports the redirection of the interrupt vector table through the **getvect()** and **setvect()** commands. This makes task switching much easier. By redirecting the existing ISR routine for the timer tick interrupt to a free location. In its place put the customised ISR which calls the old ISR by calling the relevant interrupt. It then goes on to call the task switching routine which controls task switching.

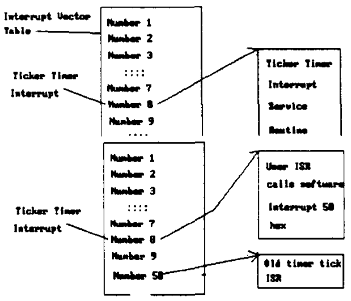


Fig 4: Redefinition of Interrupt Vector Table

This is illustrated in fig 8. The old timer ISR will now have interrupt vector number 50 hex, the new ISR will have interrupt vector number 8[2]. With each tick, the new ISR will be called.

3.3 Use of Maths Co-processor

Currently 80286/80386-based PCs only have a Maths Co-processor as an option. It is possible to use software floating point arithmetic, but this is relatively slow compared to a 80287/80387 Maths Co-processor. It is difficult to make comparison between different systems but bench-mark timings for a standard simulation are given in table 1. The grid used was a 25,45,7 and the number of iterations was 100.

Computer type	Co-processor	Time taken (minutes:secs)
Intel 80386DX-based, 25MHz	None	35:00
Intel 80386SX-based, 16MHz	Intel 80387	6:37
Intel 80386SX-based, 16MHz	None	51:50

Table 1: Comparison of run times for 100 iterations

2 Interrupt 8 is the System Timekeeper and is a hardware interrupt. The system keeps time by dividing a 1.19318-MHz clock signal by 65536. This produces 'ticks' at a rate of 18.2 per second. Each tick generates a type 8 interrupt.

The maximum current clock speed available for a 80386-type processor is 40MHz. No machine with an Intel 80486 processor was tested, although these processors are at least twice as fast as the comparable Intel 80386 processor. Another advantage of the Intel 80486 processor is that it has a Maths Co-processor in-built (although the 80486SX does not).

3.4 Subgridding method

A technique of subgridding can be employed to reduce memory storage [3]. This method splits the problem into a coarse grid with a large step size and a fine grid with a small step size is introduced only around discontinuities. The cell dimensions will thus vary in size.

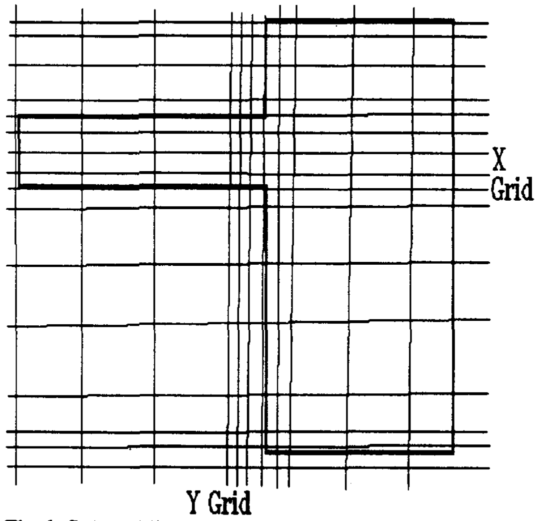


Fig 6: Subgridding method for Microstrip Antenna

3.5 Shielded Area/Volume

All conductors are treated as perfect, thus tangential Electric field components on a zero thickness conductor will equal zero. If the conductors are assumed to have a finite thickness all electric fields within the volume and all tangential electric field components on the surface will be equal to zero. These electric fields thus do not have to be calculated or stored. Problems which have a large amount of metal can benefit most from this method [6].

4 RESULTS

A patch antenna was analysed using the techniques discussed. This layout of the structure is shown in figure 7. An example of the 3D time slices are shown in figures 8-11. The linear grid used was 30,56,14.

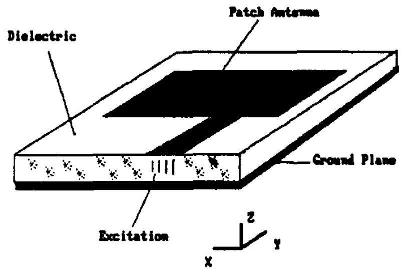


Figure 7: Patch antenna structure

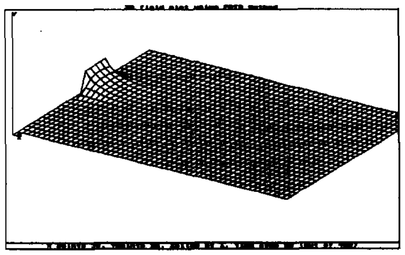


Fig 8: Response after 50 time steps

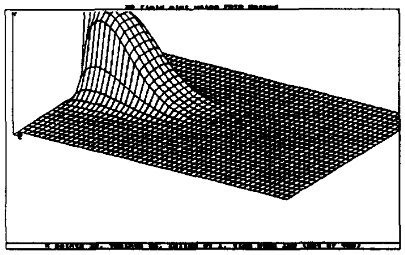


Fig 9: Response after 100 time steps

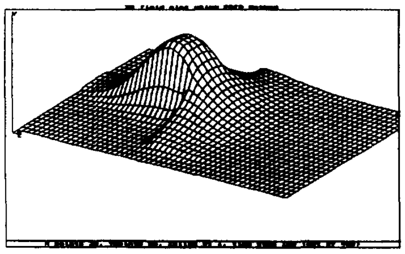


Fig 10: Response after 150 time steps

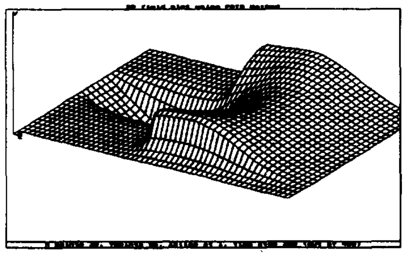


Fig 11: Response after 200 time steps

5 CONCLUSIONS

The multitasking kernel has worked well giving a good working environment with several processes running in parallel each using a 1Mbyte segment of memory. Multitasking using Transputers would obviously decrease run-times through true concurrency. A maths co-processor is essential when simulating. A factor of 8 improvement is speed was obtained. This is due to the amount of floating point arithmetic. The subgridding method is dependent on the generated grid. This grid can be staggered in both *i,j* and *k* directions. A useful addition to the program would be an automatic grid generator which detects discontinuities and this generates a finer grid around it. Problems which have a large amount of metal can benefit most from the shielded volume method. A major drawback of the FDTD processor is that it can only address up to a maximum of 1MBytes. To take full advantage of the 80x86 memory management facilities new 80286/80386 code needs to be written to interface with the existing code. This will take advantage of the 80x86 virtual memory facility.

References

1. A Taflone,"Finite Difference Time Domain Method for Electromagnetic Scattering and interaction problems",IEEE Trans. Electromagnetic Compatibly, vol. EMC-22,pp191-202,Aug 1980.
2. X Zhang, J Fang and K Mei,"Calculations of the Dispersive Characteristics of Microstrips by the TDFD Method",IEEE Trans. MTT, vol. 26,pp 263-267,Feb 1988.
3. T.Shitata, T Havashi and T.Kimura,"Analysis of microstrip circuits using 3D full-wave electromagnetic field analysis in the time domain",IEEE Trans. MTT,vol. 36,pp1064-1070,June 1988.
4. V Svetlana, K Lee and K Mei,"A subgridding Method for the TDFD Method to Solve Maxwell's Equations",IEEE Trans. MTT, vol 39, No 3, 1991.
5. D Sheen, S Ali, M Abouzahra and J Kong,"Application of the 3D FDTD Method to the Analysis of Planar Microstrip Circuits",IEEE Trans. MTT, vol. 38,No7, July 1990.
6. Furse C and Mathur S,"Improvements to the FDTD Method for Calculating the Radar Cross Section of a Perfectly Conducting Target",IEEE Trans. MTT, vol. 38, No.7, July 1990,pp919-927.