Efficiency Issues in the Implementation of Finite Difference Time Domain Codes

*Joe LoVetri**
Systems Integration Lab
Institute for Information Technology
National Research Council,
Ottawa, Ontario, Canada, K1A 0R8.

George I. Costache
Department of Electrical Engineering
University of Ottawa,
Ottawa, Ontario, Canada,
K1N 6N5.

Abstract

Implementation techniques which increase performance of time domain finite difference codes for the solution of sparse time domain electromagnetic problems are considered. As the basis of this analysis, Maxwell's equations are expressed as a system of conservation laws. Practical issues, such as computational efficiency and memory requirements, are discussed for the implementation of the finite difference schemes. Advanced programming techniques in the C language are used to implement the finite difference schemes discussed. The example of the penetration of electromagnetic energy through a shield with a thick gap is used to check the efficiency of the methods.

1.0 Maxwell's Equations as Conservation Law System

Maxwell's equations can be cast as a system of conservation laws of the form (see [Shankar 89, 90]),

$$u^j_t + \partial_x E^j + \partial_y F^j + \partial_z G^j = S^j, \quad j = 1, \ldots, n,$$

where the solution vector {u}, the flux vectors {E}, {F}, and {G}, and the source vector {S} are given by

$$\{\mathrm{u}(x,t)\} = \begin{bmatrix}\begin{bmatrix}B_x\\B_y\\B_z\end{bmatrix} & \begin{bmatrix}D_x\\D_y\\D_z\end{bmatrix}\end{bmatrix}^T = \begin{bmatrix}B\\D\end{bmatrix}, \; \{E(x,t)\} = \begin{bmatrix}\begin{bmatrix}0\\(-eD_z)\\(eD_y)\end{bmatrix} & \begin{bmatrix}0\\(mB_z)\\(-mB_y)\end{bmatrix}\end{bmatrix}^T = \begin{bmatrix}\hat{x}\times eD\\-\hat{x}\times mB\end{bmatrix},$$

$$\{F(x,t)\} = \begin{bmatrix}\begin{bmatrix}(eD_z)\\0\\(-eD_x)\end{bmatrix} & \begin{bmatrix}(-mB_z)\\0\\(mB_x)\end{bmatrix}\end{bmatrix}^T = \begin{bmatrix}\hat{y}\times eD\\-\hat{y}\times mB\end{bmatrix},$$

$$\{G(x,t)\} = \begin{bmatrix}\begin{bmatrix}(-eD_y)\\(eD_x)\\0\end{bmatrix} & \begin{bmatrix}(mB_y)\\(-mB_x)\\0\end{bmatrix}\end{bmatrix}^T = \begin{bmatrix}\hat{z}\times eD\\-\hat{z}\times mB\end{bmatrix}, \text{ and } \{S(x,t)\} = \begin{bmatrix}0\\-J\end{bmatrix}.$$

In this way Maxwell's equations can be handled as a vector equation consisting of six coupled partial differential equations. This simplifies the application of different finite difference techniques and allows one to talk about some efficiency considerations relevant to all the methods applicable to the above system.

91CH3036-1/0000-1746 $1.00 ©1991 IEEE

2.0 Finite Difference Algorithms

There are many ways to finite difference the above equations. One of the simplest procedures is the interlaced leap-frog scheme [Yee 66] which is a two step scheme on an interlaced mesh. Note that this interlacing is specific to Maxwell's equations and may not occur for other equations. Other schemes such as the Lax-Wendroff and upwind difference schemes are also available [Shankar 89, 90]. Most of these schemes can be put into the form

$$u_{jkl}^{n+1} = Q(u_{jkl}^{n}),$$

or a similar two step form, for $n \geq 0$ where Q is a polynomial in the backward and forward shift operators E_- and E_+ for each space dimension [LoVetri 90].

2.1 Memory Requirements

In three-dimensional space, memory requirements for a practical sized problem can be formidable. With *brute force* programming methods, total memory requirements, B_T, are estimated as

$$B_T = N_x \times N_y \times N_z \times B_{node} = N_T \times B_{node}$$

where N_x, N_y, and N_z are the number of mesh points in the x, y, and z directions respectively (assuming a rectangular lattice), and B_{node} is the number of bytes of storage space required at each node in the lattice. If a scheme is used where all six components of u are stored at every node, as well as the σ, ε, and μ associated with each node then

$$B_{node} = 6 \times B_{comp} + 3 \times B_{mat}$$

where B_{comp} is the number of bytes required for each component of **u** and B_{mat} corresponds to the bytes required for the material constants. If *double precision* is used for **u** and *single precision* for the material constants then $B_{comp} = 8$ and $B_{mat} = 4$ (for the SPARCstation 1™). Thus for a lattice with $N_x = N_y = N_z = 100$, the total storage requirement for the problem is at least 60 MBytes which is beyond the capabilities of most engineering workstations (e.g. SPARCstation 1 with 16 MBytes of memory). This does not include *overhead* storage, such as program and intermediate variable storage.

2.2 Practical Implementation: Dynamic Memory Allocation

During the solution of a typical electromagnetic interaction problem it is often the case that many of the nodes at a certain time step do not contain a disturbance. This is especially true when the excitation is in the form of a sharp time domain pulse. For example, in the propagation of an electromagnetic pulse through a shield, the disturbance is localized in space at early interaction times and only starts to spread spatially as the interaction progresses.This spatial localization of the disturbance can be taken advantage of when a scheme is implemented by allocating memory to a lattice node only if it contains a disturbance at the current time step <u>or</u> has the possibility of containing a disturbance at the next time step. In the same way, memory is *unallocated* or removed from a lattice node when the node does not contain a disturbance at the current time step <u>and</u> does not have the possibility of containing a disturbance at the next time step. This allocating and unallocating of memory has the effect of *propagating* the memory storage for the nodes through the lattice with the disturbance.

Using the C language, this process is implemented by first only allocating memory to an array of *pointers* to the data structures which hold the information for each lattice node. Thus, if 10000 nodes make up a 100X100 finite difference lattice, then a 100X100 array of pointers to the nodal data structure is allocated, requiring a total of 40 KBytes of memory (a pointer requires 4 bytes of memory). The nodal data structure consists of the data types required to represent the solution vector at each node and to keep the material constants for the node's cell. For the three-dimensional Maxwell's equations each data structure consists of six double precision variables. Since the material properties are usually constant with time, it is best to define a separate data structure for the three single precision variables required to hold the material constants for the lattice. These data structures are illustrated in Fig. 1.

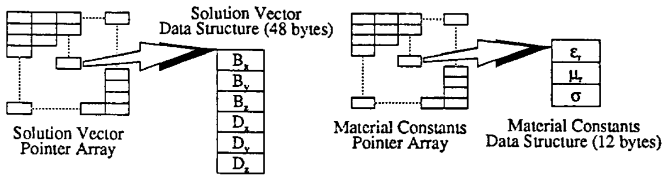

Figure 1. Example of solution vector and material constants data structures

The next step is to input the initial conditions and material constants into the lattice. This is done by first allocating the structure memory and then storing the appropriate data. Only cells with non-free-space material constants have their material constant data structure memory allocated. Once the initial conditions are input, the finite difference algorithm is applied to find the disturbance at the next time step. This consists of performing three steps:

- *allocate nodes which are neighbors to a non-zero node,*
- *execute the finite difference algorithm, and*
- *unallocate nodes which are zero <u>and</u> have all neighbors as zero.*

In this context, the term neighbor is dependent on the domain of dependence (or the spatial bandwidth) of the finite difference scheme. For example, in the leap-frog and Lax-Wendroff schemes, a disturbance at one node can propagate only to the nearest neighbors of that node in each spatial direction. On the other hand, in the upwind scheme, a disturbance at one node can propagate to its *two* nearest neighbors in each spatial direction.

In order to save on the time required to compute the disturbance at each time step, an obvious technique is to not perform the finite difference algorithm at unallocated nodes. If the solution vector data structure of a node is unallocated then it has already been determined a priori that a disturbance cannot propagate to this node at the next time step, thus it is justified to skip the calculation altogether. In order that every node in the lattice does not have to be checked as to whether it is *in* (i.e. allocated) or *out* (i.e. unallocated) before a calculation is performed, all the nodes which are *in* are kept in a *linked list* [Horspool 86]. The finite difference algorithm is then only performed on the nodes in the linked list. For example, at time zero only the nodes with given initial conditions along with their neighbors are in the linked list. In this way the calculations at each time step are performed quicker when there are only a few nodes *in* the lattice. This is usually the case at the beginning of a problem when the disturbance is fairly localized. Every time a discontinuity is met in the lattice the disturbance is *spread* spatially and more nodes will be *in*, having the effect of slowing the next computation time.

3.0 Performance Results and Conclusions

The two dimensional leap frog method was run over 80 time steps on a 100X100 node mesh for the problem of a sharp Gaussian pulse impinging on a thick shield with a gap. Results of the memory required at each time step along with the total CPU time required are shown below in Fig. 2 for both the brute force method and the dynamic allocation (sparse technique). As the pulse fills the inside of the shield the number of allocated nodes increase and the slope of the Total CPU time graph increases.

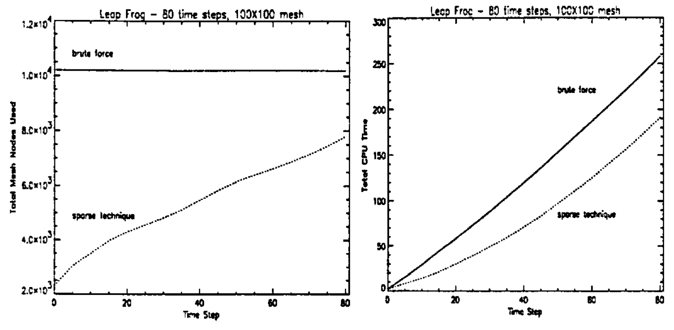

Figure 2. Memory requirement (mesh nodes allocated) and CPU time (sec.) comparisons

Thus, this sparse technique can be effectively used for problems with spatially localized propagating disturbances through a finite difference mesh. As the disturbance begins to fill the mesh (e.g. steady state problems) the method may be less efficient.

References

[Horspool 86] Horspool, R. N., **C: Programming in the Berkeley UNIX Environment**, Prentice-Hall Canada Inc., Scarborough, Ont., 1986.

[LoVetri 90] LoVetri, J., Wartak, M. S., *Practical Comparison of Finite Difference Time Domain Schemes*, submitted this symposium proceedings, 1990.

[Shankar 89] Shankar, V., Hall, W. F., Mohammadian, A. H., *A Time-Domain Differential Solver for Electromagnetic Scattering Problems*, Proceedings of the IEEE, Vol. 77, No. 5, pp 709 - 721, May, 1989.

[Shankar 90] Shankar, V., Mohammadian, A. H., Hall, W. F., *A Time-Domain Finite-Volume Treatment for the Maxwell Equations*, Electromagnetics, Vol. 10, No. 1-2, pp 127 - 145, 1990.

[Yee 66] Yee, S. K., *Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equations in Isotropic Media*, IEEE Trans. on Ant. and Prop., Vol. AP-14, No. 3, pp 302 -307, May, 1966.