PREDICTING SCATTERING OF ELECTROMAGNETIC FIELDS USING FD-TD ON A CONNECTION MACHINE

Andrew T. Perlik, Member, IEEE
MRJ, Inc.
10455 White Granite Drive
Oakton, Virginia 22124
(703) 934-9232

Torstein Opsahl
Center for Innovative Technology
13873 Park Center Road
Herndon, Virginia 22071

Allen Taflove, Senior Member, IEEE
Department of Electrical Engineering and Computer Science
Technological Institute, Northwestern University
Evanston, Illinois 60201

Abstract

The Finite Difference-Time Domain (FD-TD) numerical technique for solving Maxwell's equations is mapped onto a massively parallel Single Instruction Multiple Data (SIMD) architecture. A Connection Machine™ was chosen over other contemporary SIMD machines as the most promising candidate. The fundamental FD-TD algorithm developed by Taflove is decomposed into its serial and parallel segments. Connection Machine implementation is discussed in detail including processor assignment, processor utilization, run time, problem size, and future directions.

Introduction

FD-TD is an explicit time stepping finite difference formulation of Maxwell's curl equations that was developed in the mid-1980's by Allen Taflove [1]. It has broad applicability to the study of electromagnetic scattering by three dimensional objects because the scatterers can be closed or open, conducting, dielectric, inhomogeneous, or anisotropic. The algorithm has been validated against experimental data on objects that exhibit scattering physics at edges, corners, and cavity penetration. Agreements to within 1 dB of experimental data have been reported for scatterers that physically span 9 wavelengths and have a bistatic scatter dynamic range of 40 dB [1]. Consequently, FD-TD is certainly among the most robust scattering predictive algorithms that are currently available. If a parallel computer algorithm could be developed that could handle larger scatterers than serial code and/or run faster than serial codes, then this new development could have broad applicability to the electromagnetics community.

FD-TD Algorithm

FD-TD models the propagation of a plane wave in a finite volume of space containing the electromagnetic scatterer, Figure 1. A cubic cell spatial lattice grids the volume under study. Although simplistic, attempts to generalize the gridding procedure to a non-uniform, scatterer surface conformal one have not been totally satisfactory because both the computational dynamic range and algorithm run time degrade severely. Alternative gridding techniques still remain an area for continued research. An exploded view of a typical FD-TD cell appears in Figure 2, the well known Yee lattice. Note that the E and H field components are computed on a half cell staggered grid. This formulation guarantees second order accuracy in the difference equations that are being solved. Two computational regions are defined, a total field region and a scattered field region. The total field region completely encloses the scatterer. In it, the total field gets updated computationally in order to preserve a large computational dynamic range. Scattered fields are needed to obtain far field information. The traveling wave incident fields get added at the interface of these two computational regions respecting causality.

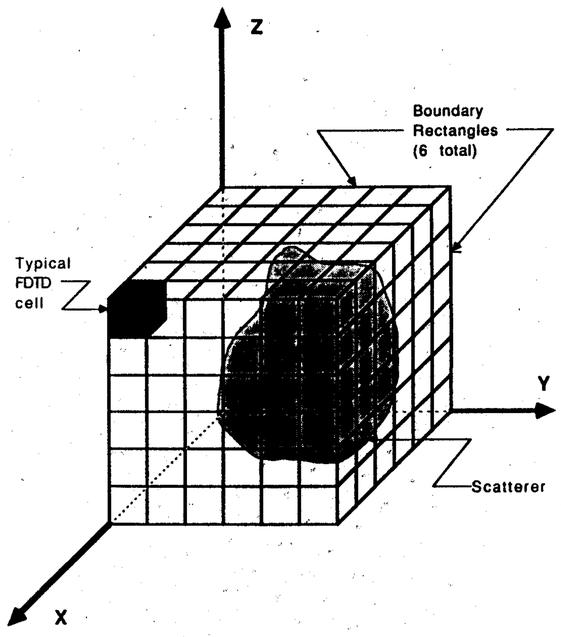

Figure 1. FD-TD Volumetric Grid Enclosing a Scatterer

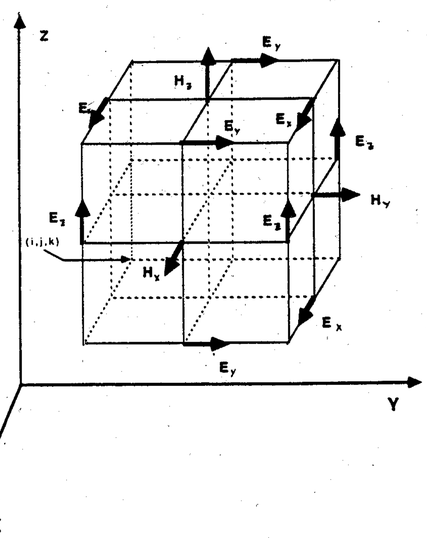

Figure 2. Yee FD-TD Lattice

A problem solution is obtained by time stepping the difference equations. As time evolves the incident wave travels through space exciting the scatterer. Time stepping continues until steady state is achieved for all field components in the grid. At steady state all field components are sinusoidal in time with converged magnitudes and converged relative phases.

Update equations for the volumetric grid decompose into four categories: discretized Maxwell's equations for grid interior points, Mur update equations for E components at the interior points of the boundary rectangles, interpolation and extrapolation equations for the edges of the boundary rectangles, and equations for the incident fields. Mur update equations [2] are

0018-9464/89/0700-2910$01.00

used to computationally terminate the volumetric grid. Basically, they are discrete versions of a one-way wave equation and are designed to "absorb" outgoing plane waves. In order to update a Mur boundary point to a new time, all nearest neighbor spatial values are needed. Consequently, the Mur updates do not apply for fields along the edges of the volumetric grid. Edges are updated heuristically using an interpolation and extrapolation scheme on nearest neighbor data.

The Connection Machine

The Connection Machine (CM) is a massively parallel SIMD computer manufactured by Thinking Machines Corporation in Cambridge, Massachusetts. CMs are hosted by serial computers that broadcast instructions to it, and the same instruction is executed by each processor on data in its own memory. A commercially available CM-2 contains a maximum of 65,536 processors, each having 65,536 bits of dedicated memory (63,995 bits are user addressable). Instructions are bit oriented giving the programmer the unusual, but extremely useful flexibility to match word length to suit desired dynamic range or to match interprocessor message lengths. Floating point coprocessors are available, but at the present time they support only 32 bit single precision arithmetic. An elapsed time for a complete cycle of arithmetic (i.e., retrieve operands, perform the arithmetic operation, and store the result) of 40 μsec is easily achievable, using non-optimized non-pipelined code. Provided that all 65k processors are productively computing, 1.6 gigaflops (65k/40 μs) establishes a lower limit on the machine capability. Typical performance for algorithms developed at MRJ, Inc. is equivalent to a serial machine performance of 5 gigaflops, and peak performance rates equivalent to 28 gigaflops on a serial machine have been demonstrated.

General processor-to-processor communication is available; however, for the algorithm capabilities incorporated into the present model, only nearest neighbor processor communication is needed. All processors can get data from their nearest neighbors simultaneously. For a 32 bit message the elapsed time ranges from 30 μsec to 140 μsec depending on algorithm design implying that a net memory transfer rate of 70 gigabits is possible on a full machine.

On-line disk storage (data vault) is available up to 80 gigabytes for a 65k CM-2, but at present it is not compatible with all hosts. Software will be available in the near future to remedy the situation.

The host computer plays an integral part in executing algorithms. Not only does it broadcast instructions to the CM, but it can also read data out, perform computations, and write data back to selected processors. This affords the user with flexibility to use both serial and parallel computational capability, as needed.

Programs can be written on small (8k) machines and can be run on larger machines without changing the code. This is accomplished by using the good coding practice of exploiting the operating software supplied machine constants. The concept of virtual processors is also supported. Each physical processor can partition its memory by factors of 2 and assign to each partition a virtual processor ID. An 8k machine can, therefore, act like a 65k machine provided that each processor needs only 1/8 of a processor's physical memory. Run time degrades with virtualization.

Mapping FD-TD onto the CM

FD-TD was mapped onto the CM with two guiding principles: maximize the size of the volumetric grid that can be analyzed and minimize run time. Development took place under the constraints that no on-line disk storage was available and that all floating point arithmetic would be single precision (32 bit) so that the floating point coprocessors could be used. The first constraint arose because both development risk and time were high when the programming was started. Data vault software was new, unproven, and was available under an operating system that was less flexible than the one desired. Floating point arithmetic using the coprocessors reduced run time by a factor of 10--sufficiently attractive to warrant their use. Contrary to what might be expected, however, no compromise in desired accuracy is incurred. Single precision arithmetic was demonstrated to support a computational dynamic range of 40 dB and is believed to support a range of over 60 dB. The real trade was against volumetric grid size because using less than 32 bits would free memory for more grid points.

Step 1

The FD-TD algorithm is examined and broken into its algorithmic serial and parallel computational steps. Start by identifying the computational chronology at each time step:

update incident fields
update interior H fields
update interior E fields
update rectangular boundary edges
update rectangular boundary interiors
test for steady state

Each computational step above depends on the result of the previous one. Unless the basic algorithm is changed, these steps are sequential on both serial and parallel computers. All spatial points can theoretically be updated simultaneously in each computational step. In the second step, for example, all three components of H can be updated simultaneously as well.

Step 2

Identify the role(s) to be played by each processor for each parallel step. This step is critical for robust parallel code development. Performance both in run time and problem size could vary by orders of magnitude among implementations. Returning to the second computational step above, minimum execution time would result if each H field component and each E field component at a grid cell were assigned to individual processors. Such an assignment would greatly sacrifice problem size since each FD-TD cell would require six processors. Instead, the decision was made to assign each processor the role of updating the three H components and the three E components assigned to each FD-TD cell, Figure 3. The trade here was in favor of problem size at the expense of run time. Total storage requirements needed to support computations for each FD-TD cell sums to 1,536 bits, significantly less than the 63,995 user addressable bits. Each processor can, therefore, by virtualized by a factor of 39, the remaining storage being reserved for stack space.

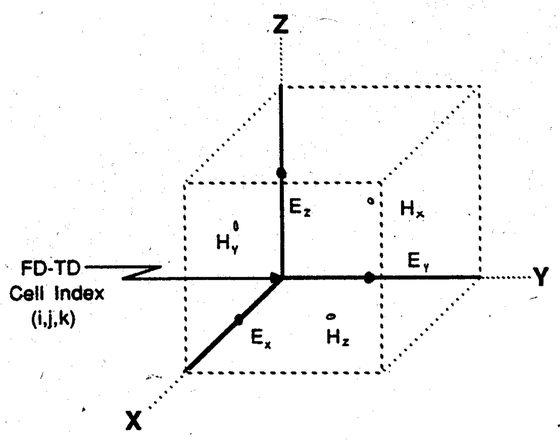

Figure 3. Fields Assigned to a CM Processor

Figure 4 summarizes the essential breakdown between the serial and parallel processing steps. Some steps have been left out for clarity, i.e., update of incident fields, convergence testing, and computation of peak and phase for scattered fields, but the breakdown in Figure 4 captures the philosophy.

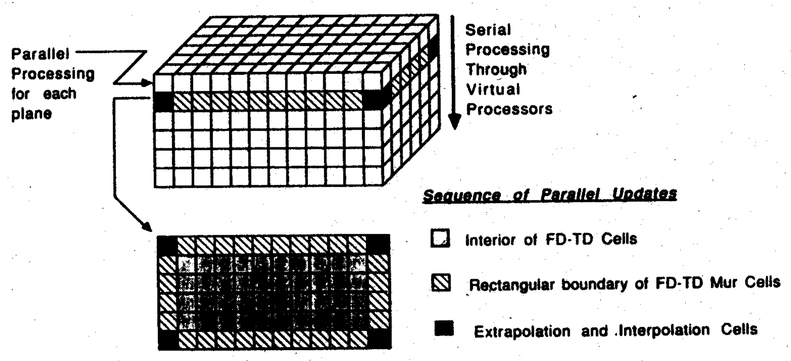

Figure 4. Overview of Essential Parallel and Serial Steps

Assessment of Parallel Implementation

Each FD-TD cell needs only 1,536 bits for data storage. Creating virtual processors is clearly a good idea; otherwise, the excess memory per processor would not get used. Now consider processing for an individual plane. Let each plane be n cells by m cells. Then (n-2)(m-2) cells are interior ones, all getting updated simultaneously. Edge processors update next. Only four processors are active independent of grid size. Clearly, this is poor utilization of the processors since only a small number are being used. The true measure of its impact on overall performance is the relative amount of time consumed compared to the other grid update functions for each plane. Table 1 provides the data needed. Edge updates require nearly half as much time to compute as the interior points do. Updating the edges on the CM instead of the host was a poor choice, one that will be remedied in the next version of the software. Face updates are computed using 2(n+m)-8 processors, on the order of the square root of the number of available processors. Again referring to Table 1, the conclusion is drawn that a significant amount of time is required to update the face processors relative to the interior ones. This does not imply that face updates should be performed in the host, however, because the I/O between the host and the CM for the root of CM processors needed to service the faces may be too expensive. Algorithms for face update run time reductions are currently being examined.

Table 1. Normalized Elapsed Time for Major Parallel Program Segments

PROGRAM SEGMENT	NORMALIZED RUN TIME
interior update	1 unit
edge update	.496 units
face update	.402 units

The three program segments discussed above constitute the core code that gets executed for each plane in the FD-TD grid. The execution time required to service those segments that employ only a fraction of the number of available processors, the edge and face updates, is approximately 47 percent of the total time required to service a plane. This suggests that the run time improvement for the current algorithm is at best a factor of 2. Run time improvements by factors of 10 or more will require a fundamental algorithm redesign.

Assessment of Algorithm Performance

Performance assessment will touch on the following topics: problem size as a function of machine size, run time as a function of machine size, and comparison between serial and parallel codes. The most succinct method for conveying results is to express them in terms of the number of FD-TD cells that fit into the CM's random access memory (RAM). Otherwise, results would be coupled to scatterer size and the desired solution accuracy, since for fixed scatterer size solution accuracy is a function of the thickness of the scattered field computational region. With the present code an 8k processor CM can evaluate 38 planes of 62x126 cells giving a total FD-TD volume of 296,856 cells. A 65k processor CM can evaluate 38 planes of 254x254 cells for a total of 2,451,608 FD-TD cells. The same code will run on both machines at the same execution time of 0.045 seconds/time step/virtual plane. Total run time per time step is 38 times larger, 1.7 seconds/time step. Run time is flat with the increase in machine size because the architecture is SIMD. Run time does change as the number of virtual planes change because the physical processors serially service the virtual processors that they represent. Benchmarks were run to show that reducing the number of virtual planes by a factor of two also reduces the run time by a factor of two. The largest scatterer studied and reported in the literature as far as we are aware is the 9λ cross tee plate reported by Taflove [1] using a Cray based code. It was embedded in a 258,048 cell lattice. By comparison the CM code can evaluate a computational lattice 9.5 times larger in volume on a full CM-2 in RAM. Furthermore, the CM code as currently written is somewhat more general because the lattice cells can be rectangular parallelpipeds instead of just cubes. If the CM code is restructured in the same way, lattice size would double by another factor of two in volume and execution time per virtual level would fall by roughly 10 percent.

Summary

The advent of massively parallel processors is still in its infancy; the CM, in particular, being commercially available for only two years and being based on technology that is 10 years old. Already, we are compiling data that suggests parallel codes are capable of exceeding lattice sizes and executing faster than even Cray based codes for lattice optimally matched to the CM's memory. The applicability of parallel processing to grid based techniques appears to have a bright future.

References

[1] A. Taflove and K. R. Umashankar, "Analytical Models for Electromagnetic Scattering," Final Technical Report RADC-TR-85-87 by ITT Research Institute, Chicago, IL, to Rome Air Development Center, Hanscom AFB, MA 01731 on Procurement Number F19628-82-C-0140, May 1985.

[2] G. Mur, "Absorbing Boundary Conditions for the Finite-Difference Approximation of the Time-Domain Electromagnetic-Field Equations," IEEE Trans. Electromag. Compat., Vol. EMC-23, Nov. 1981, pp. 377-382.

A. Taflove was supported in part for this research by NSF Grant No. ASC-8811273.