Finite-Difference Time-Domain Analysis of Microwave Circuit Devices on High Performance Vector/Parallel Computers

Stephen D. Gedney

Abstract— **In this paper, an efficient finite-difference time-domain algorithm for high performance distributed memory vector/parallel computers is presented. The algorithm is developed in a manner which requires only one interprocessor communication per time step. Illustrated examples based on the analysis of microwave circuit devices are presented demonstrating the efficiency and scalability of the finite-difference time-domain algorithm.**

I. Introduction

The finite-difference time-domain (FDTD) algorithm is an ideal algorithm for both vector processors and distributed memory multiprocessor computers. The kernel of the algorithm is a vector update operation that is readily vectorizable. Employment on distributed memory multiprocessor computers requires limited amounts of message passing between neighboring processors, leading to a highly scalable algorithm. By exploiting the architecture's of high performance vector/parallel computers, the problem sizes that can be currently solved using the FDTD are orders of magnitudes larger than problems that could have been treated only two or three years ago. As a result, the FDTD method has been highly effective for numerous applications, including the modeling of microwave circuit devices [1], [2]. As memory densities and CPU speeds continue to increase at extraordinary rates, so will the potential of this technique.

In this paper a parallel FDTD algorithm is presented. The algorithm is based on a spatial decomposition of the orthogonal lattice structure. It is shown that only one interprocessor communication per time iteration is necessary, leading to higher parallel efficiencies than previous algorithms. Test cases based on a simple 3 dB Wilkinson power divider, and an 8-way power divider are presented. It is thus demonstrated that the parallel algorithm presented herein is scalable, and extremely efficient for parallel platforms.

II. The FDTD Algorithm

The finite-difference time-domain algorithm is based on central difference approximations of the spatial and time derivatives of Maxwell's curl equations. This is achieved by projecting orthogonal components of the electric and magnetic field intensity vectors onto the edges of a dual, staggered, orthogonal grid [3], [4]. By staggering the vector fields both in space and time, a second-order accurate explicit time-marching solution is obtained. This leads to the explicit time-domain solution for the discrete electric and magnetic field vectors [3], [4], e.g., (1) and (2) as shown at the bottom of the next page, where, ε_r, μ_r, and σ represent the averaged relative permittivity and permeability, and the averaged absolute conductivity, respectively, along the grid edges, c_o is the speed of light in free space, and the remaining field components are similarly updated. Note that the electric and magnetic field intensities have been normalized as

$$\vec{e} = \frac{\vec{E}}{\sqrt{\eta_o}}, \qquad \vec{h} = \vec{H}\sqrt{\eta_o} \tag{3}$$

where η_o is the free space characteristic wave impedance. The result of this normalization is that the magnitudes of $\vec{e}$ and $\vec{h}$ are of the same order.

The most computationally intensive portion of the FDTD algorithm is the explicit updates of the six vector field components, as expressed in (1) and (2). Each vector update can be implemented using a triple nested loop. A naive implementation of (1) and (2) would require 87 N floating-point operations per time iteration to update the six field components throughout the entire space, where $N = N_x N_y N_z$, and N_x, N_y, and N_z are the grid dimensions. A more traditional approach, is to precompute the material related multipliers, which thus reduces the number of floating-point operations to $45N$ [4]. If the space is assumed to be inhomogeneous in three-dimensions the material related constants are also stored in three-dimensional arrays. Thus, the total memory required to store the six field components and the material constants associated with each of the field components will be $15N$ floating point real numbers. In homogeneous media, or one or two-dimensional inhomogeneities, this memory requirement is relaxed.

The computational time can be further reduced by multiplying the electric and magnetic field intensity vectors by their respective edge lengths, e.g

$$e'_{x_{i+(1/2),j,k}} = e_{x_{i+(1/2),j,k}}\,\Delta x. \tag{4}$$

Subsequently, (1) and (2) are rewritten as

$$\begin{aligned} e'^{n+(1/2)}_{x_{i+(1/2),j,k}} &= D'_{x_{i+(1/2),j,k}} e'^{n-(1/2)}_{x_{i+(1/2),j,k}} + C'_{x_{i+(1/2),j,k}} \\ &\quad \cdot \left\{ \left[h'^{n}_{z_{i+(1/2),j+(1/2),k}} - h'^{n}_{z_{i+(1/2),j-(1/2),k}} \right] \right. \\ &\quad \left. - \left[h'^{n}_{y_{i+(1/2),j,k+(1/2)}} - h'^{n}_{y_{i+(1/2),j,k-(1/2)}} \right] \right\} \end{aligned} \tag{5}$$

and

$$\begin{aligned} h'^{n+1}_{x_{i,j+(1/2),k+(1/2)}} &= h'^{n}_{x_{i,j+(1/2),k+(1/2)}} - B'_{x_{i,j+(1/2),k+(1/2)}} \\ &\quad \cdot \left[e'^{n+(1/2)}_{z_{i,j+1,k+(1/2)}} - e'^{n+(1/2)}_{z_{i,j,k+(1/2)}} \right. \\ &\quad \left. + e'^{n+(1/2)}_{y_{i,j+(1/2),k+1}} - e'^{n+(1/2)}_{y_{i,j+(1/2),k}} \right] \end{aligned} \tag{6}$$

where

$$\begin{aligned} B'_{x_{i,j+(1/2),k+(1/2)}} &= \frac{c_o\,\Delta t\,\Delta x}{\Delta y\,\Delta z\,\mu_{i,j+(1/2),k+(1/2)}} \\ D'_{x_{i+(1/2),j,k}} &= \frac{2\varepsilon_{r_{i+(1/2),j,k}} - c_o\,\Delta t\,\sigma_{i+(1/2),j,k}\eta_o}{2\varepsilon_{r_{i+(1/2),j,k}} + c_o\,\Delta t\,\sigma_{i+(1/2),j,k}\eta_o}, \\ C'_{x_{i+(1/2),j,k}} &= 2\,\Delta x\,c_o\,\Delta t/\Delta y\,\Delta z[2\varepsilon_{r_{i+(1/2),j,k}} \\ &\quad + c_o\,\Delta t\,\sigma_{i+(1/2),j,k}\eta_o]. \end{aligned} \tag{7}$$

Similar expressions can be derived for the remaining field components by permuting the indices in a right-handed manner. Based on these expressions, the field updates will require $33N$ floating point operations per time iteration. This results in more than a 60% reduction as compared to a naive approach and a 25% reduction compared to the traditional approach. Furthermore, there are no additional memory

Manuscript received February 7, 1995; revised June 29, 1995.

This work was supported in part by Grant DAAH04-94-G-0243 from the Army Research Office, Research Triangle Park, NC, and by NSF Award ECS-9309179. It was performed in part on the University of Kentucky's 32-node iPSC/860, which was supported by Grants from the National Science Foundation (EMS-9206014), Intel Corp. (iSC022492), and the Army Research Office (DAAH04-93-G-0453).

The author is with the Department of Electrical Engineering, University of Kentucky, Lexington, KY 40506-0046 USA.

IEEE Log Number 9414244.

constraints as compared to the traditional approach. This technique also has the advantage that special cases can be included directly in the update expressions without the need for additional programming logic in the update loops. For example, within a perfect electrical conductor (PEC), the constants C' within the PEC or on a PEC surface can be set to zero. Subsequently, the electric and magnetic fields within the PEC will remain zero throughout the computation without any additional logic. In addition, passive lumped loads [5] can also be modeled directly within the field updates in (5) and (6) by correctly constructing the coefficients C' and D'. This leads to a reduction in the overall CPU time of the FD-TD solution due to improved vectorization and reduced operations, and also simplifies the programming task.

III. Parallel Implementation on Distributed Memory Parallel Computers

It has been demonstrated that the finite-difference time-domain algorithm is ideally suited for implementation on multiple-instruction multiple data (MIMD) [6], [7] and single-instruction multiple-data (SIMD) [8], [9] high performance parallel computers. This is principally due to the regularity of the dual grid and the even distribution of effort in time and space. The algorithm presented in this section is primarily focused on MIMD type-architectures, such as the Intel iPSC/860, Intel Paragon, or a PVM cluster, but can certainly be implemented on SIMD architectures as well. Unlike the previous algorithms presented in [6]–[9], it will be shown that only one interprocessor communication needs be done per time iteration, which increases the parallel efficiency of the algorithm. It will also be demonstrated that it is extremely important to vectorize the core of the FDTD algorithm to achieve optimal performance on today's high performance computers.

The parallel algorithm is based on a spatial decomposition of the regular grid structure. To this end, the original domain is spatially decomposed into contiguous subdomains [6]–[9]. The subdomains are rectangular in shape, nonoverlapping, sharing common surfaces only, and of equal size. The boundaries, or surfaces, shared by subdomains are chosen by taking slices along edges of the primary grid in the x, y, and z-directions. Thus, the discrete electric fields are tangential and the discrete magnetic fields are normal to the shared boundary surfaces. Each sub domain is then mapped onto independent processors of the parallel computer. The electric fields are updated using (5). Each edge is updated using the magnetic field vectors normal to the four faces sharing this edge. However, on a shared boundary, each processor has in local memory at most three of the four faces needed to update the electric field, and it becomes necessary to retrieve the needed data via interprocessor communication. On the other hand, the magnetic field vectors normal to the shared interface are updated using (6). Since each processor has the updated value of the tangential electric fields on the shared interface, the normal magnetic field can be updated independently on each processor, and interprocessor communication is not needed. Rather, it is more expedient to simply update the normal magnetic fields redundantly on each processor sharing the face.

Tangential electric field vectors on exterior absorbing boundaries must be updated using an absorbing boundary condition (ABC). The choice of an absorbing boundary condition can affect both the accuracy as well as the efficiency of the parallel FDTD algorithm. The widely used Mur's second-order ABC [4] requires transverse derivatives, which will result in additional interprocessor communication on shared boundaries. On the other hand, Liao's ABC [10] or the second-order dispersive boundary condition (DBC) [11], are based on normal derivatives only. Based on the decomposition described above, these operators will not require any additional interprocessor communication, since all the necessary information needed to compute the update is resident on each processor. The only consequence, is that each processor sharing a field value will perform the update redundantly.

IV. Vectorization

Almost all of today's distributed memory parallel computers utilize RISC processors as central processing units (CPU's). Most RISC processors rely on pipelining to achieve maximum floating point operation speeds and high speed cache to reduce memory access time. The optimization of many RISC processors can be achieved in the same manner as one would vectorize a program. Vectorization is realized on the innermost loops of any multi-dimensional loop structure and can be achieved in an optimal manner when: 1) the inner loops are truly vector operations and are not corrupted by function calls, or branch statements, 2) the inner-most column index corresponds to the index of the inner loop, and 3) the length of the inner loop is equal to or greater than the optimal vector length (typically determined by the vector length of a vector processor, or the cache size of a pipelined processor).

The triple loops updating e_x and h_x based on (5) and (6), are implemented as

```
do 10 k = 2, nz - 1
   do 10 j = 2, ny - 1
      do 10 i = 1, nx - 1
         ex(i, j, k) = dx(i, j, k)*ex(i, j, k) + cx(i, j, k)*
         ·[hz(i, j, k) - hz(i, j - 1, k) - hy(i, j, k)
         + hy(i, j, k - 1)]
10    continue
```

$$e_{x_{i+(1/2),j,k}}^{n+(1/2)} = \frac{\frac{\varepsilon_{r_{i+(1/2),j,k}}}{c_o\,\Delta t} - \frac{\eta_o\sigma_{i+(1/2),j,k}}{2}}{\frac{\varepsilon_{r_{i+(1/2),j,k}}}{c_o\,\Delta t} + \frac{\eta_o\sigma_{i+(1/2),j,k}}{2}} e_{x_{i+(1/2),j,k}}^{n-(1/2)} + \frac{1}{\frac{\varepsilon_{r_{i+(1/2),j,k}}}{c_o\,\Delta t} + \frac{\eta_o\sigma_{i+(1/2),j,k}}{2}} \cdot \left\{ \frac{h_{z_{i+(1/2),j+(1/2),k}}^{n} - h_{z_{i+(1/2),j-(1/2),k}}^{n}}{\Delta y} - \frac{h_{y_{i+(1/2),j,k+(1/2)}}^{n} - h_{y_{i+(1/2),j,k-(1/2)}}^{n}}{\Delta z} \right\} \tag{1}$$

$$h_{x_{i,j+(1/2),k+(1/2)}}^{n+1} = h_{x_{i,j+(1/2),k+(1/2)}}^{n} - \frac{c_o\,\Delta t}{\mu_{r_{i,j+(1/2),k+(1/2)}}} \cdot \left\{ \frac{e_{z_{i,j+1,k+(1/2)}}^{n+(1/2)} - e_{z_{i,j,k+(1/2)}}^{n+(1/2)}}{\Delta y} - \frac{e_{y_{i,j+(1/2),k+1}}^{n+(1/2)} - e_{y_{i,j+(1/2),k}}^{n+(1/2)}}{\Delta z} \right\} \tag{2}$$

TABLE I
MFLOPS PER PROCESSOR VERSUS INNER LOOP LENGTH ($Ny = 110$, $Nz = 40$)

Nx	i860 (MFLOPS)	Cray-YMP(MFLOPS)
200	20.0	208
128	16.9	213
100	14.9	200
50	12.7	165
25	11.0	100
10	8.1	53.4

$$\begin{aligned}&\text{do } 20\ k = 1,\ nz - 1\\ &\quad \text{do } 20\ j = 1,\ ny - 1\\ &\quad\quad \text{do } 20\ i = 1,\ nx\\ &\quad\quad\quad hx(i,\ j,\ k) = hx(i,\ j,\ k) + bx(i,\ j,\ k)^*\\ &\quad\quad\quad \cdot [ey(i,\ j,\ k+1) - ey(i,\ j,\ k) - ez(i,\ j+1,\ k)\\ &\quad\quad\quad + ez(i,\ j,\ k)]\\ &20 \quad \text{continue.}\end{aligned} \tag{8}$$

Note that there is no extraneous logic embedded in either of these triple loops. Rather, special conditions such as perfect conductors, impedance boundaries, or lumped loads are modeled directly within the coefficients bx, cx, and dx, preserving the inner vector loops.

The j and k indices of the e_x-update in (8) are looped over the ranges $(2,\ ny-1)$ and $(2,\ nz-1)$, respectively. These indices are chosen in this manner such that at $j = 1$ or $j = ny$, e_x is either on an absorbing boundary, or a shared boundary if it is an interior partition. There is no reason to introduce additional logic that will upset the vectorization of the inner loop, and these fields are simply updated within a separate subroutine. Also note that h_x is looped over the entire range of i, j, and k. This is due to the fact that all the necessary information is available on each processor to completely update h_x whether it is on a shared boundary or on an absorbing boundary.

Table I illustrates the effects of the inner vector length on the floating point speed. These results were generated using the loops in (8) (case #1) on a single i860 processor and a single Cray-YMP processor. As expected, longer vector lengths result in faster floating point speeds, up to the optimal vector length (which was found to be ~128 for the Cray-YMP, and ~200 for the i860). This leads to the conclusion that better floating point performance is realized by maximizing the length of the inner loop. Unfortunately, as the problem size is decreased the floating point speeds realized will decrease. It will be shown in the following section, that this leads to a degradation in the measured speedups when doing a fixed speedup study.

V. NUMERICAL RESULTS

The parallel FDTD algorithms described in the previous section were implemented on a number of platforms. The programs were initially developed on the 32-node iPSC/860 at the University of Kentucky and then ported directly to the Intel Delta at the California Institute of Technology, as well as a number of sequential computers, including the JPL Cray-YMP. The program has been fully validated through a large number of numerical examples [12].

The purpose of this section is to present the computational efficiency of the vector/parallel algorithm. The numerical results have been obtained using the 512-node Intel Delta and a Cray-YMP, for a benchmark comparison. Each processing node of the Intel Delta hosts an i860 RISC processor with 12 Mb of RAM. The Intel Delta has a high-speed network interconnecting nodes with a two-dimensional mesh topology. The primary advantage of the mesh topology is that the routing hardware is greatly simplified resulting in much larger communication bandwidths (200 Mb/sec). The efficiency of the parallel code will be measured by the *speedup* of the parallel

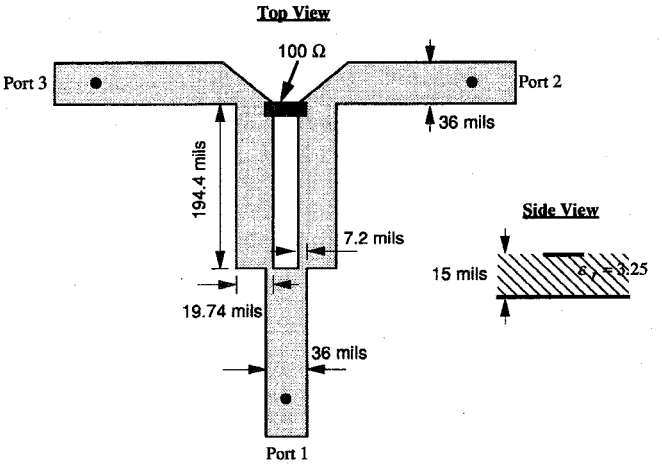

Fig. 1. 3 dB, in-phase Wilkinson power divider printed on a 15 mil TMM substrate ($\varepsilon_r = 3.25$).

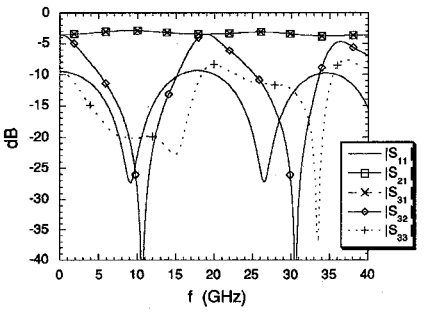

Fig. 2. Magnitude of the S-parameters of the Wilkinson power divider illustrated in Fig. 1.

algorithm. To this end, two types of speedups are referenced: fixed speedup and scaled speedup. Fixed speed up (S_f) is a measure of the speedup of a fixed sized problem as the number of processors is increased, whereas, scaled speedup (S_s) is a measure of the speedup of a problem whose size scales with the number of processors. Finally, the parallel efficiency is expressed as $\eta(P) = S(P)/P \times 100\%$. Speedup is typically used to measure the degree of parallelism in a code, or more specifically, the relationship between parallelism and the serialism in a computer program. Unfortunately, fixed speedups are asymptotically bound by the algorithm serialism, thus the amount of serialism in an algorithm must be very small to obtain reasonable fixed speedups over large numbers of processors. Scaled speedups have the advantage that the problem size can be chosen such that it maximizes the capacity of each processor. Large speedups are expected from this perspective since the percentage of serialism is constant as the problem is scaled.

As an example, the microstrip-line Wilkinson power divider illustrated in Fig. 1 was simulated using the FDTD method. The power divider is printed on a 15 mil TMM substrate ($\varepsilon_r = 3.25$), and is designed for 3 dB, equal phase power division at 9 GHz. For isolation, a 100 Ω chip resistor was placed across the apex of the power divider, which was modeled as a distributed lumped load [5]. The mitered bends were modeled via a staircase approximation of the regular orthogonal grid. Fig. 2 illustrates the S-parameters computed using the parallel FDTD program. This model was based on a 99 × 151 × 20 mesh, where $dx = dy = 3.6$ mils, and $dz = 3.75$ mils. The simulation required 2500 time iterations ($dt = 0.15$ ps).

Table II presents the time required to complete 2500 time-iterations on the Intel Delta versus the number of processors using the proposed

TABLE II
FIXED SPEEDUP STUDY FOR WILKINSON POWER DIVIDER ON THE INTEL DELTA

P	$nx \times ny$	CPU s	$S_f(P)$	$\eta(P)$
1	99 x 151	2327.	1	100 %
2	99 x 76	1187.	1.93	98 %
4	99 x 38	629.3	3.69	92 %
8	99 x 19	337.1	6.90	86 %
16	50 x 19	190.1	12.23	76 %
32	50 x 10	110.8	21.00	65 %
64	25 x 10	69.5	33.48	52 %
128	13 x 10	48.0	48.47	38 %

FDTD algorithm. The benchmarked times are recorded in CPU-seconds, which is identical to the wall clock time. The last two columns of Table II are the fixed speedup and the parallel efficiency, respectively. The speedup recorded for this small problem realizes 38% efficiency over 128 processors. The observed loss of parallel efficiency as the number of processors is increased has multiple causes. Initially, as the original problem is distributed over 128 processors, it reduces from 99 × 151 × 20 on a single processor to roughly 13 × 10 × 20 on each of 128 processors. As a result, the ratio of floating point operations to interprocessor communication greatly decreases. Slight load imbalances also become more predominant for smaller problem sizes since the CPU time is based on the *slowest* processor. Furthermore, as discussed in Section IV, as the problem size decreases, the FLOPS rate per processor decreases. From Table I, the 99 × 151 × 20 sized problem is being performed at 14.9 MFLOPS on the single processor partition, whereas the 12 × 10 × 20 sized problem is being performed at approximately 8.1 MFLOPS per processor on the 128-processor partition. This is a 55% reduction in CPU speed. For benchmark comparison this problem was also executed on a Cray-YMP. This simulation required 413 CPU's on a single processor, which is equivalent to roughly ten i860 processors.

The above example was also simulated using a parallel FDTD method which utilizes the interprocessor communication scheme suggested in [6]–[9]. The field updates were based on (5), (6), and (8) so that the degradation in efficiency as compared to the method described in this paper is only due to additional communication. A speedup of 37 over 128 processors was recorded on the Intel Delta ($\eta = 29\%$) using this approach, as compared to 48.5 ($\eta = 38\%$) recorded in Table II. Since the Intel Delta is a tightly coupled distributed computing environment and has relatively small latencies and high network bandwidths, the speedup is only reduced by 25%. However, in a loosely coupled distributed computing environment, such as a workstation cluster, the loss of efficiency will be more catastrophic due to large latencies and low network bandwidths.

A scaled speedup study was also performed on the Intel Delta. This was done by scaling the problem size with the number of processors. Maintaining a 99 × 151 × 20 grid on each processor, the number of processors was scaled from 1–256 (the partitions were kept as square as possible). The CPU times per time iteration and the scaled speedups are recorded in Table III. Excellent speedups are realized. Note that there is an initial jump in CPU time from one to two processors, which is principally due to interprocessor communication. This communication time is again increased for 16 processors, when the central processors in the mesh partition are communicating about all four boundaries. Beyond 16 processors, the scaled speedup remained linear.

A final example is the simulation of a larger device, specifically the 8-way power divider illustrated in Fig. 3. The 8-way power divider consisted of three stages of the Wilkinson power dividers in Fig. 1. The FDTD model was based on a 475 × 343 × 20 mesh, where $dx = dy = 3.6$ mils, and $dz = 3.75$ mils. The simulation

TABLE III
SCALED SPEEDUP STUDY ON THE INTEL DELTA
(99 ×151× 20 LATTICE PER PROCESSOR)

P	FDTD	$S_s(P)$	$\eta(P)$
1	.930 s	1	–
8	.978 s	7.6	95 %
16	.997 s	14.9	93 %
32	.997 s	29.8	93 %
64	.997 s	59.5	93 %
128	.997 s	119.1	93 %
256	.997 s	237.7	93 %

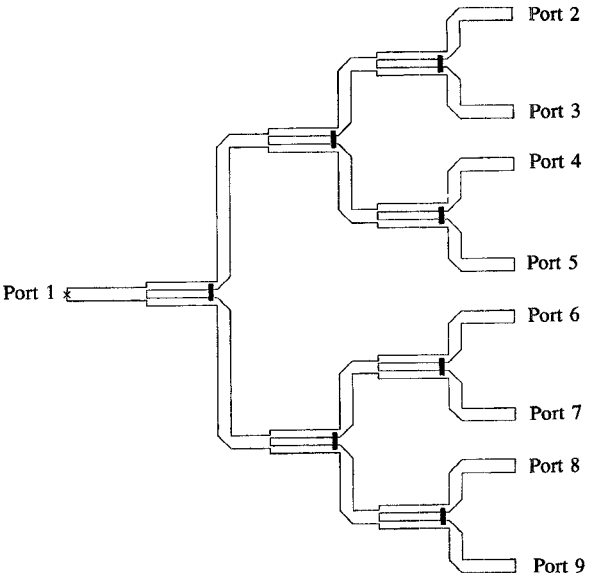

Fig. 3. 8-way Wilkinson power divider network printed on a 15 mil substrate ($\varepsilon_r = 3.25$).

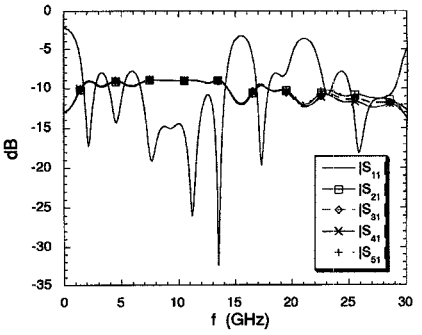

Fig. 4. S-parameters of the power divider illustrated in Fig. 3. (a) Magnitude of the S-parameters. [By symmetry $S_{21} = S_{91}$, $S_{31} = S_{81}$, $S_{41} = S_{71}$, and $S_{51} = S_{61}$.]

required 6000 time iterations ($dt = 0.15$ ps), which required 3700 CPU's on 16 processors of the Delta (1950 CPU's on 32 processors). The amplitudes of the S-parameters computed for this device are illustrated in Fig. 4. It is demonstrated that −9 dB power division is achieved at each output port at resonance. The ports were also of equal phase over the spectrum, although this is not illustrated here.

VI. CONCLUSION

In this paper, a parallel FDTD algorithm was presented for the analysis of microwave circuits and devices. The algorithm, which requires only one interprocessor communication per time iteration, results in high parallel efficiencies. Benchmarked analyses presenting

the fixed and scaled speedups of the parallel FDTD algorithm were provided, demonstrating the efficiency and the scalability of the algorithm on the highly parallel Intel Delta. The existing parallel FDTD algorithm is capable of supporting upwards of one billion degrees of freedom on the Intel Delta, and is capable of solving a problem of this magnitude in quite reasonable amounts of time. With the rate of advances of RISC processors and dynamic random access memory, high performance computers will be capable of handling 10's of billions of degrees of freedom in a fraction of the time using highly scalable algorithms such as the FDTD in the near future.

ACKNOWLEDGMENT

This research was performed in part using the Intel Touchstone Delta System operated by the California Institute of Technology on behalf of the Concurrent Supercomputing Consortium. Access to this facility as well as to the Cray-YMP was provided by the Jet Propulsion Laboratory, Pasadena, CA.

REFERENCES

[1] D. Sheen, S. Ali, M. Abouzahra, and J. A. Kong, "Application of the three-dimensional finite-difference time-domain method to the analysis of planar microstrip circuits," *IEEE Trans. Microwave Theory Tech.*, vol. 38, pp. 849–856, July 1990.

[2] X. Zhang, J. Fang, K. Mei, and W. Lin, "Calculation of the dispersive characteristics of microstrips by a time-domain finite-difference method," *IEEE Trans. Microwave Theory Tech.*, vol. 36, pp. 263–267, Jan. 1988.

[3] A. Taflove and M. E. Brodwin, "Numerical solution of steady-state electromagnetic scattering problems using the time-dependent Maxwell's equations," *IEEE Trans. Microwave Theory Tech.*, vol. 23, pp. 623–630, Aug. 1975.

[4] K. Kunz and R. Luebbers, *The Finite-Difference Time-Domain Method for Electromagnetics.* Boca Raton, FL: CRC Press., 1993.

[5] M. Piket-May, A. Taflove, and J. Baron, "FD-TD modeling of digital signal propagation in 3-D circuits with passive and active loads," *IEEE Trans. Microwave Theory Tech.*, vol. 42, pp. 1514–1523, Aug. 1994.

[6] J. Patterson, T. Cwik, R. Ferraro, N. Jacobi, P. Liewer, T. Lockhart, G. Lyzenga, J. Parker, and D. Simoni, "Parallel computation applied to electromagnetic scattering and radiation analysis," *Electromagnetics*, vol. 10, pp. 21–40, Jan.–June 1990.

[7] R. D. Ferraro, "Solving partial differential equations for electromagnetic scattering problems on coarse-grained concurrent computers," in *Computational Electromagnetics and Supercomputer Architecture,* T. Cwik and J. Patterson, Eds. vol. 7; also in *Progress in Electromagnetics Research,* J. A. Kong, Ed. Cambridge, MA: EMW Publishing, 1993, vol. 7, pp. 111–154.

[8] A. Perlik, T. Opsahl, and A. Taflove, "Predicting scattering of electromagnetic fields using the FD-TD on a connection machine," *IEEE Trans. Magn.*, vol. 25, pp. 2910–2912, July 1989.

[9] A. Perlik and S. Moraites, "Electromagnetic wave analysis using FD-TD and its implementation on the connection machine," in *Computational Electromagnetics and Supercomputer Architecture,* M. Morgan, Ed., vol. 7; also in *Progress in Electromagnetics Research,* J. A. Kong, Ed. Cambridge, MA: EMW Publishing, 1993, vol. 7, pp. 266–308.

[10] W. C. Chew, *Waves and Fields in Inhomogeneous Media.* New York: Van Nostrand, 1990.

[11] V. Betz and R. Mittra, "Comparison and evaluation of boundary conditions for the absorption of guided waves in an FDTD simulation," *IEEE Microwave and Guided Wave Lett.*, vol. 2, pp. 499–401, Dec. 1992.

[12] Stephen D. Gedney, "Finite-difference time-domain analysis of microwave circuit devices on high performance parallel computers," Univ. of Kentucky, Lexington, KY. Elec. Eng. Tech. Rep. EE-1-94, Feb. 1994.

Scattering from a Circular Dielectric Post Embedded in a Grounded Dielectric Sheet Waveguide

E. Sawado, K. Ishibashi, and K. Hatakeyama

***Abstract*—A systematic method for obtaining the scattered electric field in a grounded dielectric sheet waveguide is presented. It is shown that point-matching method can be used for an explicit calculation of the integral equation to estimate the scattering from a circular dielectric post embedded in the grounded sheet. Magnitude of the reflection coefficient as a function of dielectric constant is given.**

I. INTRODUCTION

In 1968 Schwinger published the lecture notes on the problem of electromagnetic scattering by a circular dielectric rod in a rectangular waveguide [1]. It was described that, for some special relations between the frequency, the dielectric constant, and the radius of the rod, the reflection coefficient becomes equal to zero [2], [3]. This problem has been a subject of interest to researchers for many years. In particular, attention was focused on the dip shown on the curve of reflection coefficient, illustrating as a function of the dielectric constant of the post. This phenomenon is due to volume resonance of the post.

The aim of this paper is to present a theory of scattering by a dielectric post (dielectric constant; $\epsilon_k = 2\epsilon_o \sim 500\epsilon_o$) embedded in a grounded dielectric sheet (dielectric constant $\epsilon_p = \kappa\epsilon_o$, the permittivity $\kappa = 2$). The point-matching method was used for the numerical estimation of magnitude of the reflection coefficient for the dielectric post with various permittivities. Fig. 1 shows the cross section of this structure. The structure is assumed to be uniform and infinite in both x and z directions. It is also assumed that substrate material is lossless.

For TE mode propagation, the electric field E_y is a solution of

$$\frac{\partial^2 E_y}{\partial x^2} + \frac{\partial^2 E_y}{\partial z^2} + k_0^2 E_y = 0 \qquad 0 < x \tag{1}$$

where $k_o^2 = \omega^2 \epsilon_o \mu_o$.

Mathematically the problem of relating a field to its source is that of integrating an inhomogeneous differential equation. Letting $j\omega\mu_o\delta(x - x')\delta(z - z')$ be the source function, we have the form

$$\frac{\partial^2 E_y}{\partial x^2} + \frac{\partial^2 E_y}{\partial z^2} + \kappa k_o^2 E_y = -j\omega\mu_o\delta(x - x')\delta(z - z') \qquad -T < x < 0 \tag{2}$$

where κ is the relative permittivity. We find the solution for E_y by means of Laplace transform

$$g(x, \gamma) = \int_{-\infty}^{\infty} E(x, z)e^{\gamma z}\, dz. \tag{3}$$

Multiplying both side of (1) and (2) by $e^{\gamma z}$, and integrating from $-\infty$ to $+\infty$, we have

$$\frac{\partial^2 g}{\partial x^2} + (\gamma^2 + k_o^2)g = 0 \qquad 0 < x \tag{4}$$

Manuscript received February 24, 1995; revised June 29, 1995.

E. Sawado is with the Tokyo Metropolitan University, Minami-Osawa, Hachioji, Tokyo, 192-03, Japan.

K. Ishibashi is with the Tokai University, Department of Mechanical Engineering, Tomigaya, Shibuya-ku, Tokyo, 151, Japan.

K. Hatakeyama is with the NEC Corporation, Miyazaki, Miyamae-ku, Kawasaki, Kanagawa, 216, Japan.

IEEE Log Number 9414233.