A Parallel Planar Generalized Yee-Algorithm For the Analysis of Microwave Circuit Devices

Stephen D. Gedney
Department of Electrical Engineering
University of Kentucky
Lexington, KY 40506-0046

and

Faiza Lansing
Spacecraft Telecommunications Equipment Section
Jet Propulsion Laboratory
Pasadena, CA 91109

California Institute of Technology

Abstract. The planar generalized Yee (PGY) algorithm is an extension of the generalized Yee-algorithm and the discrete surface integral (DSI) methods, which are based on explicit time-marching solutions of Maxwell's equations. Specifically, by exploiting the planar symmetries of printed microwave circuit devices, great savings in both CPU time and memory can be achieved. Since the PGY algorithm is an explicit method, it has a high degree of parallelism. To this end, a highly scalable parallel algorithm based on a spatial decomposition of the general unstructured mesh is presented. Two spatial decompositions are compared, the recursive inertia partitioning (RIP) algorithm and the Greedy algorithm. The Greedy algorithm provides optimal load balance, whereas the RIP algorithm more effectively minimizes shared boundary interface lengths. Through numerical example, it is demonstrated that the Greedy algorithm provides superior speedups. It is also demonstrated that the parallel PGY algorithm is a highly scalable algorithm.

This work was supported under JPL/NASA Contract 959534 with the University of Kentucky
Submitted to the International Journal on Numerical Modeling (Electronic Networks, Devices and Fields)

1. Introduction

The generalized Yee-algorithm [1, 2], and the Discrete Surface Integral (DSI) algorithm [3] have proven to be highly robust and accurate techniques for the analysis of microwave circuit devices. These methods arc explicit time-marching schemes derived from the discretization of Ampere's and Faraday's laws in their integral form [1-3]. To this end, the vector fields are projected onto the edges of a dual, staggered grid which is assumed to be unstructured and irregular. This is in contrast to the traditional finite difference time-domain (FDTD) method [4] which is based on regular and orthogonal grids, or the non-orthogonal FDTD method [5-8], which is based on irregular structured grids. The advantage of using unstructured grids, is that highly complex geometries can be accurately modeled with the aide of automatic grid generation techniques. Unfortunately, a disadvantage of generalized Yee and DSI algorithms is that the numerical grid must be stored. Moreover, the sparse matrices associated with them must also be stored [1, 2], greatly limiting, the size of the problem which can be solved.

The memory requirements of the generalized Yee-algorithm and the DSI algorithm can be greatly relaxed by exploiting symmetries in the model. A large class of microwave circuits can be said to have *planar symmetry,* which is recognized for three-dimensional geometries that can be uniquely described by a projection onto a two-dimensional plane. It was shown in [9] that by exploiting this symmetry, (he entire three-dimensional circuit geometry can then be described uniquely by a two-dimensional grid. Subsequently, the grid used to analyze the three-dimensional problem can be described by an unstructured two-dimensional grid in a transverse plane and as a regular grid in the third-dimension. Thus only the two-dimensional grid need be stored. Furthermore, it is shown below that the sparse update matrices of this planar generalized Yee-(PGY) algorithm need only be constructed for the two-dimensional grid. This greatly relaxes the memory requirements of the algorithm to the extent that it is actually as memory efficient as the FDTD algorithm.

The PGY-algorithm is an efficient computational method, however, since it is based on a volume discretization the computational demand can increase substantially with the size of the problem geometry. For example, the number of floating point operations per time iteration will increase as $O(N)$, where N is the number of unknowns in the discrete volume. However, the number of time iterations required to reach a steady-state will also increase as $O(\sqrt{N})$ (the number of time iterations will be inversely proportional to the minimum edge length in the model as well). The analysis of single component circuits can typically be modeled on conventional workstations or sequential computers in reasonable amounts of time. On the other hand, the analysis of multi-component circuits requires much greater resources.

It is becoming much more evident that distributed parallel computing is a highly cost effective means of achieving supercomputing performance. With the rapid increase in computational power of RISC microprocessors, as well as the increase in speeds of local area networks, highly cost effective supercomputing can be achieved through loosely coupled distributed parallel systems using message passing protocols such as PVM (Parallel Virtual Machine). Tightly coupled distributed memory multiprocessor computers, such as the 1 ntel Paragon, the Cray T3D, the IBM SP2, or the Convex SPP1 000, will provide higher performance, principally due to the fact of having faster dedicated networks interconnecting processors with greatly reduced latency times. These architectures have already demonstrated the potential of 100 GFLOPS performance for practical scientific applications.

The PGY-algorithm is extremely well suited for nigh performance distributed memory computing. Since the method is explicit, the kernel of the algorithm consists of a series of matrix-vector products. These operations are conveniently parallelized using a spatial decomposition of the unstructured mesh. It is shown that by treating the matrix as a subassembly of matrices, where each sub matrix is associated with each spatial subdomain, interprocessor communication can be minimized, resulting in a highly scalable parallel algorithm. Furthermore, due to the regularity of the grid along the vertical direction, vector/pipelining can be exploited to further increase the floating point speed of the algorithm,

The focus of this paper is on the development of an efficient implementation of the PGY-algorithm on high performance parallel computers. Section 11 presents the PGY-algorithm.

Section 111 discusses the parallelism inherent within the algorithm, and presents an efficient parallel algorithm based on a spatial decomposition of the three-dimensional mesh. Section IV presents the inherent vectorism in the PGY-algorithm, and the advantages of exploiting the vector/pipelining of RISC based parallel computers. Finally, some numerical examples are presented in Section V, which illustrate the efficiency and the scalability of the parallel PGY-algorithm.

11. 'Ike Planar Generalized Yee-Algorithm

The planar generalized Yee-algorithm is based on a direct solution of the time-dependent Maxwell's equations in their integral form. The electric and magnetic field intensities are initially normalized as

$$\begin{aligned} \vec{e} &= \vec{E} / \sqrt{\eta_o} \\ \vec{h} &= \vec{H}\sqrt{\eta_o} \end{aligned} \tag{1}$$

where η_o is the characteristic wave impedance in free space. Faraday's law and Ampère's law arc then expressed in their integral form as

$$\oint_C \vec{e} \cdot d\vec{\ell} = -\frac{1}{c_o}\frac{\partial}{\partial t}\oiint_S \mu_r \vec{h} \cdot d\vec{S} \tag{2}$$

$$\oint_C \vec{h} \cdot d\vec{\ell} = \frac{1}{c_o}\frac{\partial}{\partial t}\oiint_S \varepsilon_r \vec{e} \cdot d\vec{S} + \eta_o \oiint_S \sigma\vec{e} \cdot d\vec{S} \tag{3}$$

where co is the free space velocity of light, μr and εr are the relative permeability and permittivity, respectively, and σ is the absolute conductivity. The principle advantage of this normalization is that the magnitudes of $\vec{e}$ and h will be of the same order, reducing rounding error. Furthermore, it is much more convenient to work with the relative permittivity and permeabilities rather than their absolute values.

Faraday's and Ampère's laws are expressed in a discrete form by mapping $\vec{e}$ and $\vec{h}$ into a discrete three-dimensional space. The mapping consists of projecting the vector fields onto the edges of a dual grid, composed of two staggered grids, referred to as the primary and secondary grids. Each grid is a three-dimensional grid that is described as being regular along the vertical direction (assumed to be the z-direction), and is unstructured in the horizontal direction.

Conceptually, this grid can be generated by extruding a two-dimensional unstructured grid in the vertical direction, and segmenting it at discrete heights, as illustrated in Fig. 1. The secondary grid is staggered within the primary grid such that its vertices lie at the centroids of the primary grid cells, and the edges of the secondary grid connect the centroids by passing through the faces of the primary grid.

The electric and magnetic fields are then decomposed into orthogonal components

$$\vec{e} = \vec{e}_t + \hat{z}e_z,$$
$$\vec{h} = \vec{h}_t + \hat{z}h_z. \qquad (4)$$

Subsequently, the transverse electric and magnetic fields are mapped onto the horizontal edges of the primary and secondary grids, respectively. Likewise, the vertical electric and magnetic fields are mapped onto the vertical edges of the primary and secondary grids, respectively. The vector fields are assumed to be constant along their respective. edge lengths, as well as over the dual face through which they pass.

Based on the above discretization, Faraday's and Ampère's laws are. then mapped into the discrete space. The time derivative is then approximated using a central difference expression, which is second-order accurate if the fields are staggered in time. This leads to [9]

$$h_{z_j}^{n+1}(k) = h_{z_j}^{n}(k) - \frac{c_o dt}{A_p}\frac{1}{\mu_r(k)}\left[\sum_{i=1}^{N_{pj}} e_{t_i}^{n+\frac{1}{2}}(k)\ell_i\right] \qquad (5)$$

$$b_{t_i}^{n+1}(k+\tfrac{1}{2}) = b_{t_i}^{n}(k+\tfrac{1}{2}) - \frac{c_o dt}{dz|\vec{\ell}_i|}\left[\left(e_{z_m}^{n+\frac{1}{2}}(k+\frac{1}{2}) - e_{z_{m+1}}^{n+\frac{1}{2}}(k+\frac{1}{2})\right)dz + \left(e_{t_i}^{n+\frac{1}{2}}(k+1) - e_{t_i}^{n+\frac{1}{2}}(k)\right)\ell_i\right] \qquad (6)$$

$$e_{z_m}^{n+\frac{3}{2}}(k+\frac{1}{2}) = \left[\left(\frac{\varepsilon_{r_m}(k+\frac{1}{2})}{c_o dt} \quad \frac{\sigma_m(k+\frac{1}{2})\eta_o}{2}\right) e_{z_m}^{n+\frac{1}{2}}(k+\frac{1}{2})\right.$$
$$\left. - \frac{1}{A_s}\left[\sum_{i=1}^{N_{sj}} h_{t_i}^{n+1}(k+\frac{1}{2})\ell_i\right]\right]\frac{1}{\left(\frac{\varepsilon_{r_m}(k+\frac{1}{2})}{c_o dt} + \frac{\sigma_m(k+\frac{1}{2})\eta_o}{2}\right)} \qquad (7)$$

$$d_{t_i}^{n+\frac{3}{2}}(k) = \frac{1}{\left(\frac{1}{c_o dt} + \frac{\sigma_i (k+\frac{1}{2})\eta_o}{2\varepsilon_{r_i}(k+\frac{1}{2})}\right)} \left(\frac{1}{c_o dt} - \frac{\sigma_i (k+\frac{1}{2})\eta_o}{2\varepsilon_{r_i}(k+\frac{1}{2})}\right) d_{t_i}^{n+\frac{1}{2}}(k) - \frac{1}{dz\ell_i} \cdot \left[\left(h_{z_j}(k) \quad h_{z_{j+1}}(k)\right) dz + h_{t_i}^{n+1}(k-\frac{1}{2}) h_{t_i}^{n+1}(k+\frac{1}{2})\right) \ell_i \right]\Big] \qquad (8)$$

where d_t and b_t are the flux densities in the transverse plane, At is the time increment, n is the time index, k is the index along z, A_p and A_s arc the areas of the primary and secondary grid faces, respectively, N_{pi} and N_{sj} are the number of edges bounding the i-th primary and the j-th secondary grid faces, respectively, and the ℓ_i are the length vectors of the primary or secondary grid edges. 'I'he material parameters ε_r, μ_r and σ are assumed to be piecewise homogeneous in both the z-direction as well as in the transverse direction. At the. interface of two unlike medium, the parameters are assigned an average value, as described in Appendix A of [1].

Based on **(2)** and (3), it is recognized that the flux densities updated in (6) and (8) are normal to the faces. However, the corresponding field intensities on the dual edges passing through these faces arc not necessarily normal to the faces. As a result, the flux densities must be projected onto the edges before the dual fields can be updated. Since only one component of the field is locally known, an auxiliary operator must be introduced to perform the projection. To this end, the projection operators implemented in [1] are used to project the fields onto the dual edge passing through the face.

Let $\vec{N}_p$ be the normal area vector to a primary grid transverse face, and $\hat{s}$ be the unit vector along the dual grid edge passing through the face (Fig. 2). Using (6) the magnetic flux densities projected onto the normals of all the primary grid transverse faces are updated, Subsequently, for each face, a general flux density vector $\vec{b}$ is introduced. From (6), $\vec{b} \cdot \vec{N}_p$ is known at each edge. In Fig. 2, the edge identified is bound by vertices 1 and 2, which is identified by the index $i = 1,2$. Each vertex is also shared by two additional edges which share a common cell. Let j represent one of these edges, where j = $1,2$. The normal area vector to the j-th edge associated with the i-th vertex is $N_{p_{i,j}}$. Subsequently, we define a general flux density vector $\vec{b}_{i,j}$ to be the local estimate of the magnetic flux vector associated with the i-th vertex and the j-th edge, where $\vec{b}_{i,j}$ is computed by solving the two-dimensional system of equations

$$\begin{aligned} \vec{b}_{i,j} \cdot \vec{N}_D &= \vec{b} \cdot \vec{N}_D \\ \vec{b}_{i,j} \cdot \vec{N}_{D_{i,j}} &= \vec{b} \cdot \vec{N}_{D_{i,j}} \end{aligned} \qquad (9)$$

where the right-hand-side is known from (6). Subsequently, introducing the weighting coefficient $w_{i,j}$ $\hat{z} \cdot (\vec{N}_D \times \vec{N}_{D_{i,j}})$, the flux density projected onto the dual edge is expressed as

$$\vec{b} \cdot \hat{s} = \frac{\sum_{i=1}^{2} \sum_{j=1}^{2} w_{i,j} (\vec{b}_{i,j} \cdot \hat{s})}{\sum_{i=1}^{2} \sum_{j=1}^{2} w_{i,j}}. \qquad (1o)$$

The field updates are then computed using (6)-(10). However, it is realized that computing the parameters for these equations requires a significant number of floating point operations, leading to a highly inefficient algorithm. However, by employing standard finite-element type techniques, the computational efficiency can be greatly enhanced by treating these linear operators as sparse matrices. To this end, (6)-(10) can be expressed in reduced form as

$$\left[H_z^{n+1}\right]_k = \left[H_z^n\right]_k + \overline{\overline{A}}_{h_z} \left[E_t^{n+1/2}\right]_k \qquad (11)$$

$$\left[B_t^{n+1}\right]_{k+1/2} = \left[B_t^n\right]_{k+1/2} + \overline{\overline{A}}_{h_t} \begin{bmatrix} E_{z_{k+1/2}}^{n+1/2} \\ E_{t_{k,k+1}}^{n+1/2} \end{bmatrix} \qquad (12)$$

$$\left[H_t^{n+1}\right]_{k+1/2} = \overline{\overline{A}}_{h_t} \left[B_t^{n+1}\right]_{k+1/2} \qquad (13)$$

$$\left[E_z^{n+3/2}\right]_{k+1/2} = \overline{\overline{D}}_{e_z} \left[E_z^{n+1/2}\right]_{k+1/2} + \overline{\overline{A}}_{e_z} \left[H_t^{n+1}\right]_{k+1/2} \qquad (14)$$

$$\left[D_t^{n+3/2}\right]_k = \overline{\overline{D}}_{e_t} \left[D_t^{n+1/2}\right]_k + \overline{\overline{A}}_{e_t} \begin{bmatrix} H_{z_{k,k+1}}^{n+1} \\ H_{t_{k+1/2}}^{n+1} \end{bmatrix} \qquad (15)$$

$$\left[E_t^{n+3/2}\right]_k = \overline{\overline{A}}_{e_p} \left[D_t^{n+3/2}\right]_k \qquad (16)$$

where the subscript k refers to the discrete height along the z-direction, D_t and B_t are the flux densities, the $\overline{\overline{D}}$'s are diagonal matrices, and the $\overline{\overline{A}}$'s are sparse matrices. Note that these matrices are only associated with the two-dimensional grid since they are the same for all values of k (inhomogeneities in material parameters are. easily built into these expressions). As a result, the additional memory required to store these matrices is nominal.

process on a single processor of the computer is P. In fact, due to *serialism*, the speedup will always be less than P. Thus the principle objective in designing a parallel algorithm is to minimize the amount of serialism in the parallel algorithm. Serialism can be defined as computation that would be better done on a uniprocessor system than a parallel system. It also includes additional computation that must be performed by the parallel algorithm, but is not required by a sequential algorithm. Some examples of tasks that lead to serialism are: 1) load imbalances, 2) interprocessor communication, 3) latency, 4) synchronization, and 5) redundant computation. in a parallel system, the total computational time to complete a global task is equal to the time required by the *slowest* processor to complete its local task. Therefore, it is important to evenly distribute the work effort among all the processors, namely the work load must be balanced. In contrast, load imbalances lead to processor idle time and reduce parallel efficiency.

Interprocessor communication, i.e., message passing, introduces additional effort that is required by the parallel algorithm which is not performed by the sequential algorithm, and it leads to the degradation of the parallel efficiency. The computational time required to perform an interprocessor communication can be divided into two parts, 1) the time it takes to initiate the communication, which is referred to as the *latency time*, and 2) the time/byte required to transmit a data packet between two remote processors. In most tightly coupled distributed-memory multiprocessor computers, the time required to transmit a byte of data is on the order of nanoseconds. However, the latency time is typically on the order of tens of microseconds. Therefore, it is much more profitable to send a small number of large data packets rather than a large number of small data packets. Furthermore, since single floating point operations are performed on the orders of nanoseconds, it is extremely important to maximize the ratio of the time a processor spends performing floating point operations to the amount of time spent performing interprocessor communication, Unfortunately, as the number of processors increases for a fixed problem size, this ratio inevitably decreases, leading to the degradation of the parallel efficiency.

Synchronization can also lead to serialism in a parallel algorithm. This leads to processors which are left idle while waiting for information from a remote processor. A good example of this is a recursive algorithm, where a process on my processor is dependent on data from another processor, whereas, another processor is dependent on my data still to be processed, and so on. Even though the work load may be balanced, the parallel efficiency can be extremely poor due to the required synchronization.

Finally, redundant computation is computation done on a number of processors concurrently that could have been done by a single processor. Often, if the time spent doing redundant computation is small, it is cheaper in terms of overall CPU time to do it redundantly rather than to have to perform an additional interprocessor communication. Again, the reason being that the latency time to initiate the communication will be on the order of microseconds, compared to the nanoseconds required to perform the floating-point operations. However, a substantial amount of redundant computation can lead to degradation in parallel efficiency.

It is thus important to design a parallel algorithm which is load balanced, has a minimal number of interprocessor communications, minimizes the size of the data packets being communicated, is asynchronous in operation, and minimizes the amount of redundant or serial computation. To this end, the parallel PGY algorithm is based on a spatial decomposition of the three-dimensional grid into contiguous, non-overlapping subdomains. The partitioning of the two-dimensional unstructured mesh is being performed using one of two different techniques: 1) The Recursive inertia Partitioning (RIP) algorithm [11], which is a power of two algorithm that is ideal for hypercube computers, and 2) the Greedy algorithm [12], which a non-power of two algorithm providing a more general decomposition. Both algorithms are quite simple to implement, and are computationally efficient even for large meshes. 'l'he RIP algorithm has the advantage that it minimizes the number of grid edges on shared boundaries, however, it typically has 10 % load imbalances in the partition. The Greedy algorithm has the advantage that it partitions the mesh in a manner that is ideally load balanced, however, the number of edges on shared boundaries is typically greater than that yielded by the RIP algorithm. Furthermore, the

[illegible]

Greedy algorithm can sometimes result in disjoint sub domains [12]. Nevertheless, in Section V, it is illustrated that the load balancing is more important than minimizing shared boundary lengths, and the Greedy algorithm leads to improved parallel efficiencies. The spatial decomposition along the third, regular dimension is done using a trivial partitioning scheme of the regular grid.

Once the mesh is decomposed into subdomains, one subdomain is assigned to each processor. The matrices in (11)-(16) are then expressed as a subassembly of matrices, where each sub matrix represents the updates of the fields within each subdomain. subsequently, the matrix vector products are simply expressed as

$$\overline{\overline{A}}x = \sum_{i=1}^{P} \overline{\overline{A}}_i x_i \tag{18}$$

where P is the total number of processors.

This approach has a number of advantages, The spatial decomposition is done in a manner such that each processor performs roughly the same number of floating point operations each time iteration, leading to a balanced parallel algorithm. Secondly, only the local matrices and field vectors need be stored on each processor. This algorithm maximizes the ratio of computation to communication leading to a highly scalable algorithm. This can be seen by further decomposing $\overline{\overline{A}}_i$ as

$$\overline{\overline{A}}_i = \overline{\overline{A}}_i^{\text{int}} + \sum_{j=1}^{N_{shared}} \overline{\overline{A}}_{i,j}^{shared} \tag{19}$$

where $\overline{\overline{A}}_i^{\text{int}}$ are the rows of A_i associated with all field vectors inter-nal to the i-th processor's subdomain, $\overline{\overline{A}}_{i,j}^{shared}$ are the rows of A_i associated with all field vectors in the i-th processor's subdomain that lie on the boundary shared with the j-th domain, and N_{shared} is the number of processors that share boundaries with the i-th processor. Subsequently, local to each processor, (18) is actually performed as

$$\overline{\overline{A}}_i x_i = \overline{\overline{A}}_i^{\text{int}} x_i + \sum_{j=1}^{N_{shared}} \left(\overline{\overline{A}}_{i,j}^{shared} x_i\right) + \sum_{j=1}^{N_{shared}} R_x\left(\overline{\overline{A}}_{j,i}^{shared} x_j\right) \tag{20}$$

where R_x is the receive operator, receiving the vector of data from the jth processor. The first two expressions on the right-hand-side of (20) are done completely in parallel on each processor, and the final term requires interprocessor communication. Since the RIP and the Greedy algorithms both attempt to minimize the lengths of the shared boundaries, the ratio of computation to communication is high, leading to a highly scalable algorithm as demonstrated in Section V.

IV. The Vector algorithm

Almost all of today's distributed memory parallel computers utilize RISC processors as central processing units (CPUs). Many of the RISC processors rely on vector pipelining to achieve maximum floating point operation speeds. They also rely on a high speed cache to reduce memory access time. As a result, to optimize the processor floating point operation speeds, dominant computational tasks must be vectorized. 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, logical statements, or indirect addressing, 2) the inner-most column index of lnulti-dimensionsl arrays corresponds to the index of the inner loop, 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 or a super-scalar processor).

The matrices in (11)-(16) are assumed to be stored in a compressed format, namely, only the non-zero entries are actually stored and pointers are used to identify the row and column number of each entry. Thus, the matrix vector product performed in (20) is performed using indirect addressing, i.e., the effective index of the vector x is determined by either a pointer, or an integer array, and x is addressed in a random fashion. As a result, poor floating point operation will result when performing the linear operations in (20) on RISC processors that have architectures employing either vector units, pipelined floating point units, or even superscalar floating point pipelined units. This appears to be predominately due to the inability to stream the vectors into the local high-speed cache, and subsequently through the floating-point control unit.

Vectorization can be achieved, by exploiting the regularity of the sparse update matrices along the vertical direction. Specifically, the operations dependent on the discrete index k corresponding to the vertical z-direction can be placed on the inner loop of the matrix-vector product operation. As a result, the indirectly addressed variables become constants within the inner loop, and are addressed only in the outer loop. This leads to a very efficient operation that has increased floating point speeds on a RISC processor. As an example, Fig. 3 illustrates the FORTRAN loop that is used to perform the update of the interior vertical electric-field intensities, specifically from (14). It is assumed that the. secondary grid cells in the transverse plane are arbitrary polygons, and each row has a random number of non-zero elements. To this end, the pointer *iez* points to the first non-zero entry of the i-th row in the vectors *aez* and jez , jez is the column indices of each entry of *aez*, and *aez* contains the non-zero entries of the matrix. Also *epsz is* the inverse of the relative permittivity local to the vertical edge. The inner loop, loops through the vertical index k, and within this loop jez and *aez* are constants.

By exploiting the vectorization of the RISC processor, a substantial speedup can be achieved. As an example, the CPU speed .vs. the vector loop lengths were measured on an Intel i860 microprocessor. The i860 is a 64-bit RISC processor, It has a peak performance of 80 MFLOPS (Millions of Floating Point Operations per Second) single precision and 60 MFLOPS double precision at a 40-MHz clock cycle. The i860 has a 4 Kbyte instruction cache organized as a two-way set-associative memory with 32 bytes per cache block. It also has a separate data cache which is a two-way set-associative memory of 8 Kbytes. There are also two floating-point units within the processor, namely, a multiplier unit and an adder unit, which can be used separately or simultaneously under the coordination of the floating-point control unit. This design supports dual operation floating point instructions such as “add-and-multiply” and “subtract-and-multiply” by using both the adder and multiplier units in parallel. Vector operations can be pipelined through the floating point units yielding maximal performance.

The benchmarked performance of the FORTRAN loop in Fig. 3 versus the vector length *nz is* illustrated in Fig, 4 (Case 1). This was performed using single precision floating-point arithmetic. This is compared to the case in which the two inner loops in Fig. 3 are reversed (Case 2). Interestingly, as *nz* is increased, the performance of this loop decreases. Overall, this illustrates that loops with indirect addressing cannot take advantage of vector pipelining. This is a problem that plagues many finite. element codes. An additional benchmark was performed to vary the value of *nnod* (in Fig. 4 *nnod = 1000).* In both cases, the performance is effectively independent of the length of *nnod.* These results are not illustrated here.

V. Numerical Results

A FORTRAN program based on the planar generalized Yee-algorithm has been developed on a 32-node Intel iPSC/860 hypercube. This same program has been direct] y ported to the 512-node Intel Delta Supercomputer, a 2-processor Cray - YMP, and an HP 720 workstation. The program is interfaced with a commercial CAD software package (SDRC I-DEAS) running on an HP 720 workstation. The CAD software is used to design and build the circuit models. It is also used to generate the two-dimensional unstructured mesh via automatic grid generation techniques. The node-based two-dimensional mesh is subsequently partitioned on the workstation using either the Greedy or the RIP algorithms. Since the generation of the meshes of very large models can be extremely time-consuming and memory intensive, an automatic mesh refinement technique has been implemented within the parallel algorithm. The refinement is done in a fairly trivial manner, as illustrated in Fig. 5, and can be done completely in parallel. Furthermore, since it is a global refinement, a mesh partitioning does not need to be repeated.

A second-order absorbing boundary condition (ABC) known as the dispersive boundary condition [13] is used to update the fields on truncation boundary walls, minimizing any nonphysical reflections. To maintain the second-order accuracy of the boundary condition, the two-dimensional mesh is padded with two layers of rectangular cells (this is done prior to the partitioning to maintain load balance of the parallel algorithm).

A number of numerical simulations have been run to validate the code and to demonstrate its robustness [9]. As an example, consider a circular cylindrical via through a ground plane, as illustrated in Fig. 6. The via connects two 50 Ω microstrip lines using a cylindrical post passing through a circular hole in a ground plane. 'l'he two-dimensional mesh representing this geometry is illustrated in Fig. 7. The mesh models the three-dimensional geometry as it is projected onto a two-dimensional plane. Since there are conductors and dielectrics at different heights in the three-dimensional model, each cell is assigned a material identification number. Correspondingly, a table is built which identifies the discrete heights, thickness and properties of conductor strips and material slabs.

Both the RIP algorithm [11] and the Greedy algorithm [12] were used to perform the spatial decomposition of the mesh, As described in Section 111, the Greedy algorithm provides a better load balance, while the RIP algorithm better minimizes the lengths of the shared boundaries. Figures *8* and 9 illustrate the spatial decomposition of the two-dimensional quadrilateral mesh illustrated in Fig. 7 into 32 subdomains using the RIP and Greedy algorithms, respectively. A means of comparing the load balancing of the two algorithms, is to compare the maximum number of cells and nodes in a subdomain to the minimum number. This is illustrated in Table 1. It is seen that the Greedy algorithm yields much more optimal load balancing overall. However, the number of edges on shared boundaries are greater in the mesh decomposed by the Greedy algorithm than that decomposed by the RIP algorithm. In fact, observing Fig. 9, it is seen that one of the subdomains is even disjoint, e.g., it is bound by more than one closed surface. Initially, there is some uncertainty as to whether the optimal load balance will result in a more efficient decomposition, Or the minimized shared boundary lengths.

The via was analyzed on a 32-node iPSC/860. Each node of the iPSC/860 hosts a 40 MHz i860 RISC processor and 16 Mb of memory. The two-dimensional mesh in Fig. 7 was used, and consists of 4867 quadrilateral cells. The three-dimensional mesh was 40 cells high along the vertical direction. The full simulation required 4000 time iterations. The CPU times required to perform the simulation versus the number of processors are illustrated in Table 2, comparing the times that resulted from using the RIP and the Greedy algorithms. Clearly, the Greedy algorithm results in improved CPU times. Figure 10 illustrates the speedups of the parallel algorithm, again based on the RIP and Greedy decompositions. "l'he speedup here is defined as being the ratio of the CPU time required to execute the problem on P processors to that required by a single processor. This is also compared to the ideal case of a linear speedup. Excellent speedups are observed over the 32 processors. Finally, the magnitude of the S-parameters are illustrated in Fig. 11. These results are compared with [he measured results presented in [81 and those computed using an orthogonal grid TLM method {Eswarappa, 1994 #89}.

VI. Summary

In this paper, the parallel planar generalized Yee (PGY) algorithm was presented. Initially, it was shown that by exploiting the planar symmetries of printed microwave circuit devices, great savings in both CPU time and memory can be. achieved. It was also shown that significant speedups in floating point operation speeds can be achieved by exploiting inherent vectorism in the PGY algorithm due to the regularity of the grid along one dimension.

The parallel PGY algorithm presented was based on a spatial decomposition of the general unstructured mesh. By treating the update matrices as subassemblies of matrices, a very efficient parallel scheme was obtained. Two spatial decompositions were compared, the recursive inertia partitioning (RIP) algorithm and the Greedy algorithm. The Greedy algorithm provides optimal load balance, whereas the RIP algorithm more effectively minimizes shared boundary interface lengths. Through a numerical example, it was demonstrated that the Greedy algorithm provides superior speedups. From this, it can be concluded that load balancing is extremely important, and will lead to scalable parallel algorithms. Finally, it is demonstrated that the parallel PGY algorithm has a high level of parallel efficiency and provides the means to efficiently and accurately solve practical engineering problems.

References

[1] S. Gedney, F. Lansing and D. Rascoe, “A generalized Yet-algorithm for the analysis of MMIC devices,” *IEEE Transactions on Microwave Theory and Techniques, vol.* submitted for review, pp. “1993.

[2] S. Gedney and F. Lansing, "Full wave analysis of printed microstrip devices using a generalized Yee-algorithm,“ in *IEEE Antennas and Propagation Symposium Digest.* Ann Arbor, Ml: 1993.

[3] N. Madsen, "Divergence preserving discrete surface integral methods for Maxwell's equations using nonorthogonal unstructured grids, " UCRL-JC- 109787, Technical Report, LLNL, February 1992.

[4] A. Taflove and K. Umashankar, "Finite-difference time-domain (FDTD) modeling of electromagnetic wave scattering and interaction pro blems," *IEEE A ntennas and Propagation Magazine, vol. 30,* pp. 5-20, April 1988.

[5 J.-F. Lee, R. Palendech and R. Mittra, "Modeling three-dimensional discontinuities in waveguides using nonorothogonal FDTD algorithm, " *IEEE Transactions on Microwave Theory and Techniques, vol. 40,* pp. 346-352, February 1992.

[6] J. F. Lee, "Finite difference time domain algorithm for non-orthogonal grids and its application to the solution of electromagnetic scattering problems," *Archiv fur Elektronik und Uebertragungstechnik*, vol. 46, pp. 328-335, No. 51992.

[7 P. Harms, J.-F. Lee and R. Mittra, "A study of the non orthogonal FDTD method versus the conventional FDTD technique for computing resonant frequencies of cylindrical cavities," *IEEE Transactions on Microwave Theory and Techniques, vol. 40,* pp. 741-746, April 1992.

18. P. Harms, J.-F. Lee and R. Mittra, "Characterizing the cylindrical via discontinuity," *IEEE Transactions on Microwave Theory and Techniques,* vol. *41,* pp. 153-156, January 1993.

[9] S. Gedney and F. Lansing, "A Generalized Yee-Algorithm For the Analysis of Three-Dimensional Microwave Circuit Devices with Planar Symmetry," *IEEE Transactions on Microwave Theory and Techniques,* vol. pp. submitted for review 1994.

[10] S. Gedney, "Finite-Difference Time-Domain Anal ysis of Microwave Circuit Devices on High Performance Vector/Parallel Computers," *IEEE Transactions on Microwave Theory and Techniques,* vol. pp. submitted for review 1994.

[11] B. Nour-Omid, A. Raefsky and G. Lyzenga, "Solving finite element equations on concurrent computers," in *Symposium on Parallel Computation and their Impact on Mechanics.* Boston: 1987.

[12] C. Farhat and M. Lesoinne, "Automatic partitioning of unstructured meshes for the parallel solution of problems in computational mechanics," *International Journal on Numerical Methods in Engineering,* vol. 36, pp. 745-764, 1993.

[13] 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 Letters, vol. 2,* pp. 499-401, December 1992.

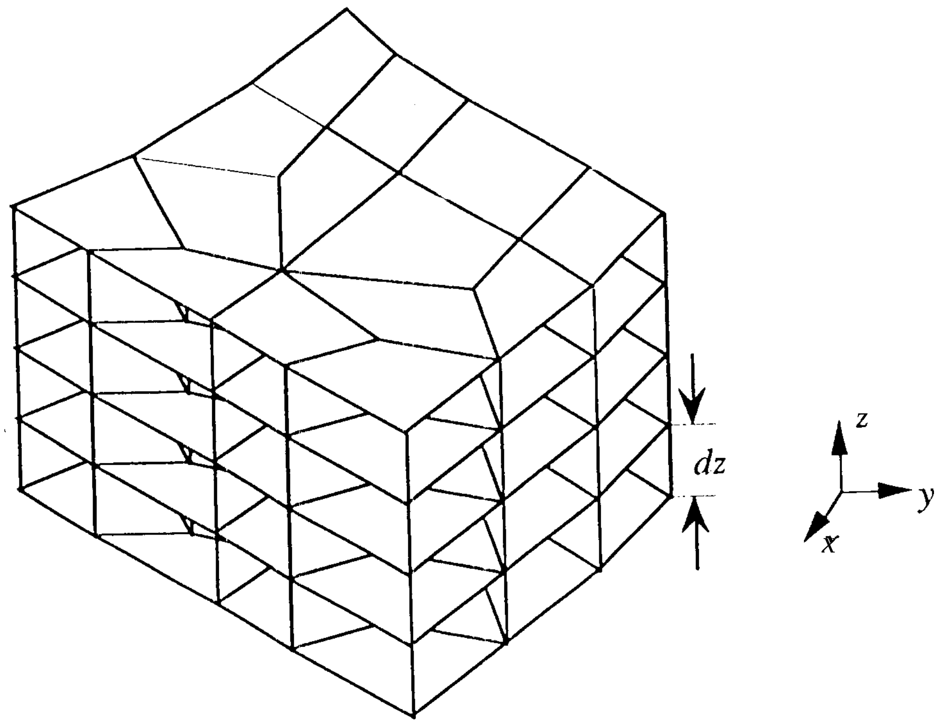

Fig. 1 An example of the primary grid described by similar two-dimensional unstructured grids cascated in the vertical z-direction in a regular sense.

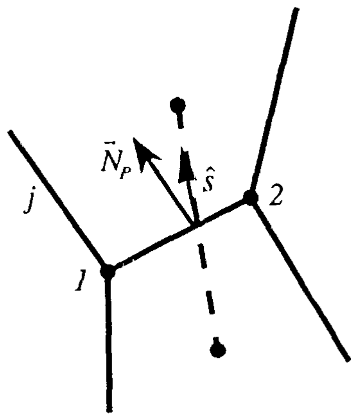

Fig. 2 Normal to a transverse primary face, and a dual edge passing through the face.

```
          do 10 i = 1,nnod
              do 10 j = iez(i),iez(i+1)-1
                  j1 = jez(j)
                  aj = aez(j)*codt
                  do 10 k = 1,nz-1
                       ez(k,i) = ez(k,i)+ aj*ht(k,j1)*epsz(k,i)
10            continue
```

Fig. 3 FORTRAN loop performing the update of the vertical electric field using (14).

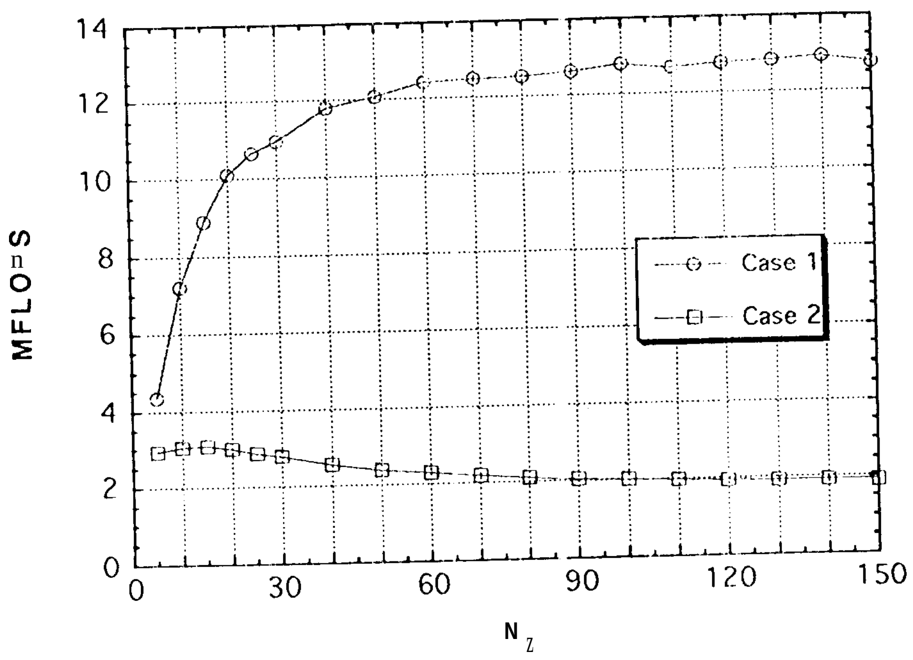

Fig. 4 MFLOPS .vs. number of cells in the vertical direction recorded on a single 40 MHz i860 RISC processor. Case 1 refer's to the loop in Fig. 2, and Case 2 is the same update with the two inner loops switched.

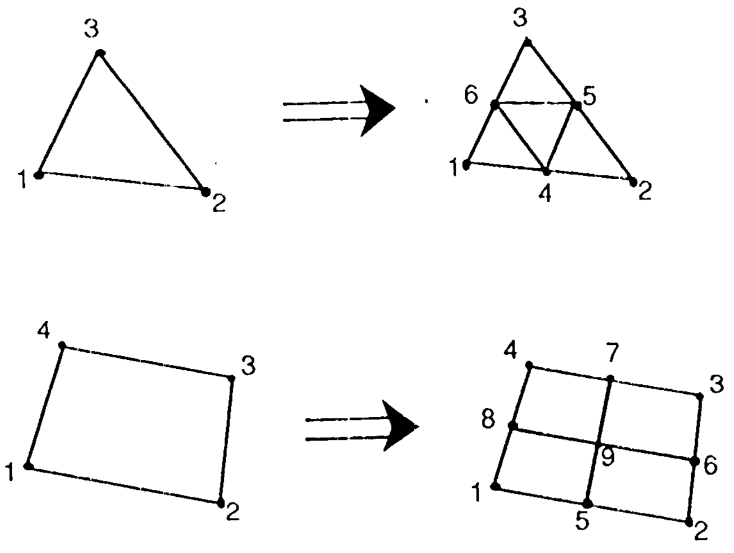

Fig. 5 Refinement of Triangular and quadrilateral elements.

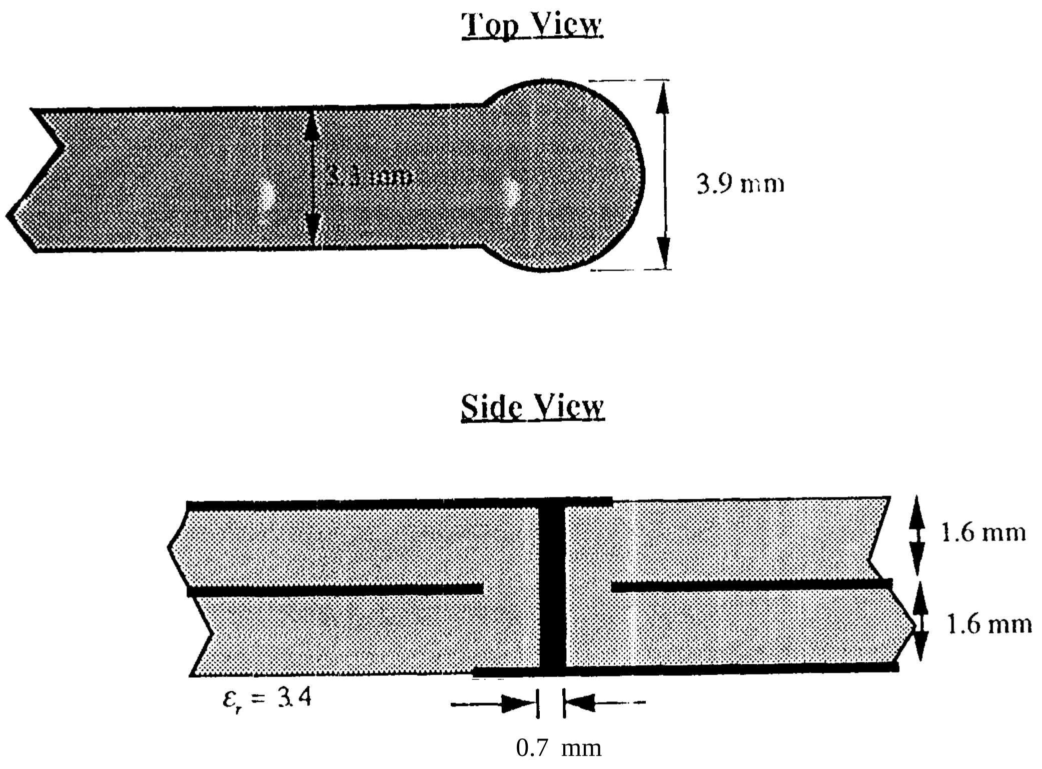

Fig. 6 Geometry of a cylindrical via through a PEC ground plane.

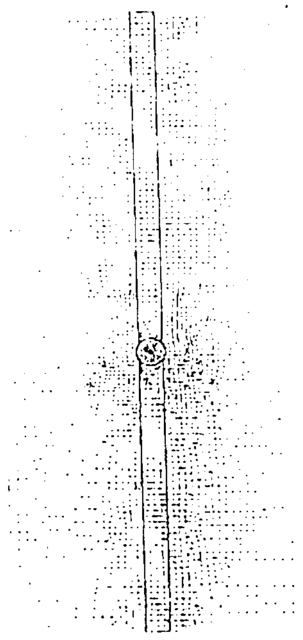

Fig. 7 Two-dimensional mesh representing the cylindrical via

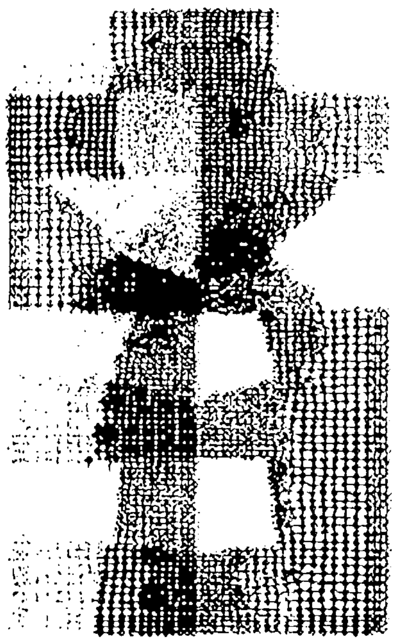

Fig. 8 Spatial decomposition of the two-dimensional mesh using the RIP algorithm (32 sub domains).

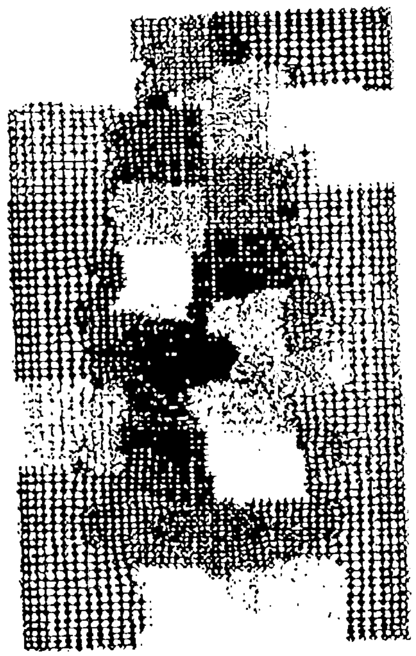

Fig. 9 Spatial decomposition of the two-dimensional mesh using the Greedy algorithm (32 sub domains).

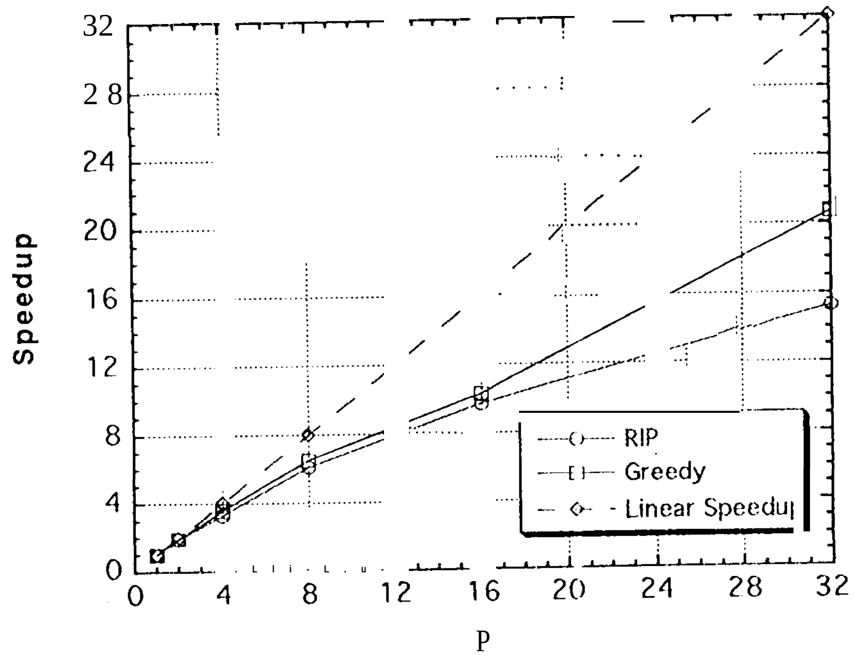

Fig. 10 Speedup of the parallel PGY algorithm over 32 processors of an iPSC/860 using the RIP and the Greedy spatial decomposition methods.

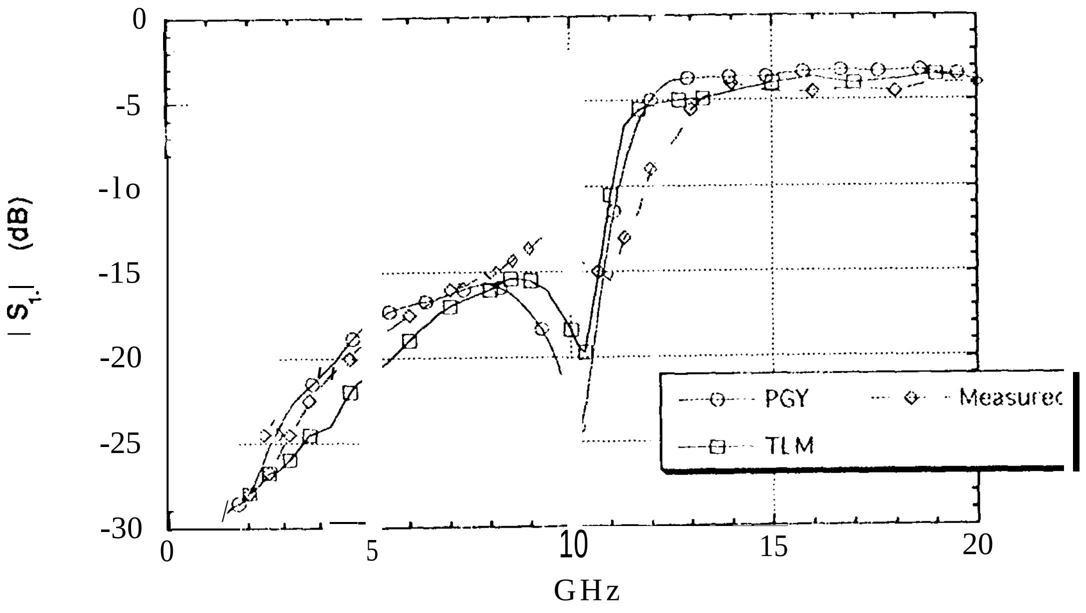

Fig. 11 Magnitude of the S-parameters for the cylindrical via through a ground plane computed using the planar generalized Yee-algorithm.

Table 1.
Load Balance of the RIP and Greedy Algorithms

	RIP		Greedy	
P	$N_{c_{max}} : N_{c_{min}}$	$N_{n_{max}} : N_{n_{min}}$	$N_{c_{max}} : N_{c_{min}}$	$N_{n_{max}} : N_{n_{min}}$
1	4867:4867	4988:4988	4867 : 4867	4988:4988
2	2542:2325	2629:2414	2434:2433	2541:2541
4	1353 : 1142	1421 : 1210	1217:1216	1313:1289
8	661:571	743:621	609:608	670:662
16	374:284	421:322	319 : 289	384:341
32	203 : 131	236:151	153: 151	190:177

Table 2.
CPU Times Recorded .vs. # of Processors (P) on iPSC/860
for the Microstrip Via (4000 time iterations)

P	RIP	Greedy
1	7936.0	7936.()
2	4224.0	4144.()
4	2406.0	2219.0
8	1301.0	12) 6.()
16	815.00	769.00
32 32	5 2 0 . 0 0	384.00