Performance of a Characteristic-Based, 3-D, Time-Domain Maxwell Equations Solver on the Intel Touchstone Delta

J. S. Shang[1] & Kueichien C. Hill[2]
Wright Laboratory, Wright-Patterson Air Force Base, Ohio 45433-7913
and
Donald A. Calahan[3]
University of Michigan, Ann Arbor, Michigan

Abstract[4]

A characteristic-based, windward[5] numerical procedure for solving three-dimensional Maxwell equations in the time domain has been successfully ported to the Intel Touchstone Delta multicomputer. The numerical results by concurrent computation duplicated the earlier simulations of an oscillating electric dipole on a vector processor and compared well with the exact solutions. The parallelized code is scalable up to 512 nodes and incurs only up to 7.6% performance degradation. The sustained data processing rate is clocked at 6.551 Gigaops. However, the data I/O process is unscalable on the shared memory system.

Nomenclature

E	Electric field intensity
H	Magnetic field intensity
i, j, k	Indices of discretized grids
J	Electric current vector
L	One-dimensional difference operator
n	Index of time level
t	Time
W	One-dimensional characteristic
x, y, z	Cartesian coordinates
r, θ, ϕ	Spherical coordinates
ϵ	Electric permittivity
μ	Magnetic permeability
λ	Eigenvalue

1 Introduction

The improvement of numerical efficiency is one of the urgent needs of computational electromagnetics (CEM) in aircraft signature technology. In this area of applications, the CEM simulations are generally more computationally intensive than computational fluid dynamics (CFD) problems. A part of the reason is that the numerical accuracy requirement for CEM tends to be more stringent than for CFD. For instance, a desirable predictive dynamic range can be as high as 60 dB over broad viewing ranges [1]. Typically for wave propagation, a suitable numerical resolution requires each wavelength to be supported by at least an order of ten grid points or more. Thus, for a scatterer with high refraction indices and material complexities, this more than ten grid points per wavelength requirement translates into the need of a computing system with astronomical computing speed and memory size.

The heavy demand on computer speed and memory for CEM simulations can be illustrated by the following example. For a fixed incident angle, the numerically generated signature of a modern fighter configuration at 1 GHz requires approximately fifty million grid points[6] to produce a ten grid points per wavelength resolution. For each grid point, the values of at least three coordinates, six field components, and nine direction cosines of a general curvilinear coordinate system need to be stored [2,3,4,5]. This results in a total of nearly one billion memory allocations. A typical CEM code operating on a 100 Megaops (10^6 floating point and integer operations) single vector processor can process data at approximately a rate of three-tenths of a microsecond per grid point per time step. At this rate, a fighter configuration will require almost five hours of computing just to advance the solution to a new time level. In order to complete a signature simulation, usually multiple look angles and hundreds of time steps are required. As a result, the large computer memory requirement and solution time has rendered the use of conventional data processors for solving CEM problems impractical.

The goal of this paper is to provide an efficient way of solving the 3D time domain Maxwell's equations in differential form by combining novel numerical algo-

[1] Senior Scientist. Fellow AIAA

[2] Electronics Engineer, Signature Technology Office

[3] Professor, Electrical Engineering & Computer Science Department

[4] This paper is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

[5] The windward difference approximation is achieved by using the one-sided stencil for computing the difference quotient according to the sign of the coefficient.

[6] Based on 15mx10mx8m size

rithms and concurrent computing technology. In the recent years, the CFD community has made significant progress in the area of algorithm development [6,7,8]. Numerical algorithms for solving hyperbolic equations[7] from the CFD discipline have been adopted for solving three-dimensional Maxwell equations in the time domain [2,3,4,5]. Among these, the characteristic-based algorithm is found to be most efficient and appropriate to duplicate the wave motions that are governed by the time-dependent Maxwell equations [1,2,3]. This numerical scheme is derived from the eigenvalue and the eigenvector structure of the hyperbolic equations system. The procedure is to diagonalize the three- dimensional governing equations into three one-dimensional Riemann problems [3,4,5,6]. Although this new numerical procedure has potential to reduce the required computing resources by allowing larger time steps and fewer discretized mesh points in CEM simulations, substantial progress in CEM for practical applications can finally be achieved by incorporating the massively parallel computing technique to deliver the needed computing resources.

Recently, through remarkable progress in microchip and interconnect data link technology, a host of single address, shared memory, and multiple address message-passing parallel computers becomes available for data processing. These scalable multi-processors or multi-computers, in theory, are capable of providing essentially unlimited computing resources for scientific simulations. However, the effective use of these massively parallel computers still rests squarely on balancing the work load and keeping the communication between computing nodes to an absolute minimum [9,10]. These requirements are intrinsically related to the numerical algorithms and hardware architectures. In the present research effort, attempts were made to map a characteristic-based algorithm onto a message-passing parallel computer for computational electromagnetics.

2 Analysis

The characteristic-based fractional-step algorithms have been demonstrated to be very efficient in solving three-dimensional Maxwell equations in the time domain [3,4,5]. In this method, the time-dependent Maxwell equations can be written in the flux vector form [11,12] given by

$$\frac{\partial U}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} + \frac{\partial H}{\partial z} = -J \qquad (1)$$

[7]A partial differential equations $AUxx+BUxy+CUyy+DUx+EUy=F$ with $B^2-4AC>0$, or all eigenvalue of these coefficient matrices are real.

where

$$U = \left\{ \begin{array}{c} B_x \\ B_y \\ B_z \\ D_x \\ D_y \\ D_z \end{array} \right\} \qquad F = \left\{ \begin{array}{c} 0 \\ -D_z/\epsilon \\ D_y/\epsilon \\ 0 \\ B_z/\mu \\ -B_y/\mu \end{array} \right\}$$

$$G = \left\{ \begin{array}{c} D_z/\epsilon \\ 0 \\ -D_x/\epsilon \\ -B_z/\mu \\ 0 \\ B_x/\mu \end{array} \right\} \qquad H = \left\{ \begin{array}{c} -D_y/\epsilon \\ D_x/\epsilon \\ 0 \\ +B_y/\mu \\ -B_x/\mu \\ 0 \end{array} \right\} \qquad J = \left\{ \begin{array}{c} 0 \\ 0 \\ 0 \\ J_x \\ J_y \\ J_z \end{array} \right\}$$

The above partial differential equation system is hyperbolic, and constitutes an initial value problem. Since the three coefficient matrices can only be diagonalized one at a time [6,7], the three-dimensional equation is split into three one-dimensional formulations. For each one-dimensional formulation, the solving procedure starts with solving the eigenvalues and eigenfunctions of that particular spatial direction. The eigenvalues for all three coefficient matrices turn out to be $\{-c,-c,0,0,c,c\}$, where c is the wave propagating speed. The signs of these eigenvalues actually control the direction of the information flow as the wave propagates through the computational domain [3,4,5,6]. Based on the eigenvectors, the field components are then expressed in terms of characteristic variables for that spatial direction. Once expressed in the characteristic form, the one-dimensional formulation becomes that of the Riemann problem [5,6]. After applying the same procedure to the other two dimensions, the characteristic-based fractional-step or time-splitting method can now be summarized in the following formulas:

$$w^{n+2} = L_x L_y L_z L_z L_y L_x w^n \qquad (2)$$

$$L_x : \frac{\partial w_{x,i}}{\partial t} + \lambda_i \frac{\partial w_{x,i}}{\partial x} = 0 \qquad i = 1,2,\ldots,6 \qquad (3)$$

$$L_y : \frac{\partial w_{y,i}}{\partial t} + \lambda_i \frac{\partial w_{y,i}}{\partial y} = 0 \qquad i = 1,2,\ldots,6 \qquad (4)$$

$$L_z : \frac{\partial w_{z,i}}{\partial t} + \lambda_i \frac{\partial w_{z,i}}{\partial z} = 0 \qquad i = 1,2,\ldots,6 \qquad (5)$$

where $w_x, w_y,$ andw_z are the one-dimensional characteristic variables associated with the $x, y,$ and z directions; L is the one-dimensional difference operator; λ_i are the eigenvalues of the partial differential equation system; and n is the index of time level[8].

[8]The fractional Step method sweeps all spatial derivatives twice in a symmetric sequence, thus advances the time step accordingly

The second-order accurate windward difference approximation can be easily constructed by using a single forward or backward difference approximation according to the signs of the eigenvalues. The ability to associate the sign of the eigenvalue with the direction of wave propagation is a very important feature of the present numerical procedure. The windward difference approximation for L_x is given by

$$\frac{\partial w}{\partial t} = \frac{w^{n+1}_{i,j,k} - w^{n}_{i,j,k}}{\Delta t} \quad (6)$$

$$\frac{\partial w}{\partial x} = \frac{-3w^{*}_{i,j,k} + 4w^{*}_{i+1,j,k} - w^{*}_{i+2,j,k}}{2\Delta x} \quad \lambda < 0 \quad (7)$$

$$\frac{\partial w}{\partial x} = \frac{3w^{*}_{i,j,k} - 4w^{*}_{i-1,j,k} + w^{*}_{i-2,j,k}}{2\Delta x} \quad \lambda > 0 \quad (8)$$

where $*$ takes the values $n+1$ and n respectively for implicit and explicit procedures; i, j, k are the indices of discretized grids. The windward difference approximations for L_y and L_z are similar to that for L_x. They can be obtained by replacing x in Equations (10) and (11) with y and z, and applying the windward difference to the j and k indices, respectively.

From Equation (5), the time-splitting solving procedure requires six one directional numerical sweeps to update the solution to the next time level. Since the formulation for each numerical sweep is identical, the strategy to map each one directional sweep to a massively parallel computer is identical. Thus, a code based on the time-splitting algorithm can be easily tuned for maximum performance by counting and adjusting arithmetic operations for each numerical sweep.

3 Data Partition Schemes and Fine Tuning for Maximum Parallel Performance

The partition of data structure plays a key role in achieving high parallel efficiency. On a message passing or distributed memory multicomputer system, the performance of concurrent computing is closely tied to memory bandwidth and memory latency. Thus, the basic strategy in achieving high parallel efficiency of any numerical procedure is to minimize data movements between nodes. In this effort, a computer code based on the characteristic-based fractional-step algorithm was mapped onto the Intel Touchstone Delta at Caltech using different data partition schemes to explore the parallel performance of the code.

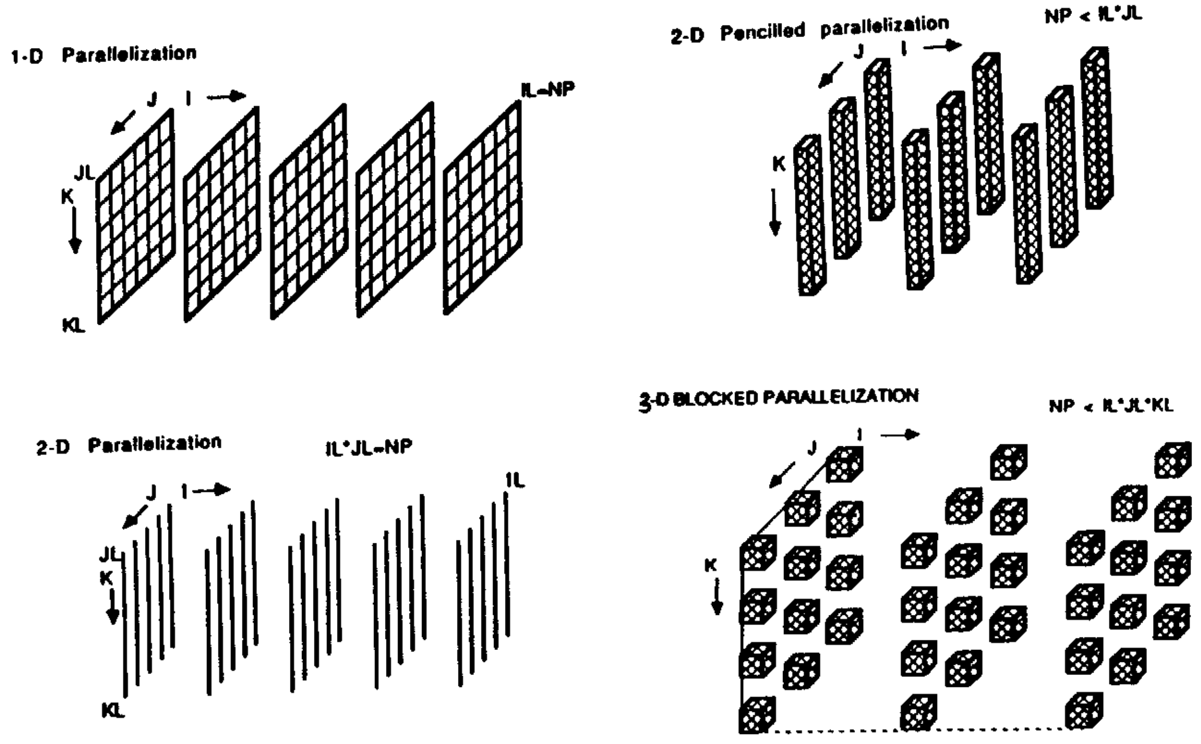

Figure 1: Hierarchy of data partition

The Delta multicomputer consists of a total of 576 heterogeneous nodes [13]. They are designated as numeric, service, gateway, and disk nodes to perform the computation, frame buffer, network link, and disk string functions respectively. A total of 512 numeric nodes are available for parallel computing. Each of the numeric nodes has a peak rate of 80 MFlops for single-precision and 60 MFlops for double-precision, and is interconnected by a two-dimensional mesh network. On this ensemble nodal architecture, the data flow and data management lead to four different approaches for the controlling of data movements between nodes. The most elementary approach to data partition is the one-dimensional parallelization in which the outermost do loop of the numerical sweeps is assigned to a number of numeric nodes. The other data partition schemes include the two-dimensional page structure by partitioning three dimensional space into cross-sectional planes, the pencil grouping, and finally, the three-dimensional block parallelization. A graphic depiction of these data partition schemes is given in Figure 1. The computational domain is assumed to consist of IL, JL, and KL discretized mesh points in each direction of the three dimensional space; the computation is assigned to NP numeric nodes.

As mentioned in the beginning of this section, minimizing the number of message passings is by far most important in achieving high parallel efficiency. In addition, load balancing is also very important in reducing the performance disparity among nodes since the performance of the slowest node actually determines the completion of a numerical simulation. Last, but not least, an ultimate data assigning sequence which takes advantage of the nearest neighbor message-passing priority is essential in reducing the unnecessary message routing length and, therefore, improve the over all parallel performance. In the following discussion, the number of message passings, load balancing, and sequence assigning issues will be addressed.

Although four data partition schemes are identified in mapping the characteristic-based code onto the Delta, only the one-dimensional parallelization and the pencil grouping scheme were investigated in this effort. The one-dimensional scheme is the most straightforward data partition for the time-splitting, windward procedure [5]. In this scheme, the computational domain of $IL \times JL \times KL$ mesh points is divided into either IL, JL, or KL grid planes. Without loss of generality, the computational domain is assumed to be divided into IL grid planes. Thus, each grid plane contains $JL \times KL$ mesh points. The computation associated with the $JL \times KL$ mesh points of each grid plane is assigned to a numeric node. At each time level, each node performs eight message passings, which include messages sent to and received from its nearest four neighboring nodes. In all, the total number of message passings is given by the value of $2 \times (4 \times IL - 6)$. With four bytes per message, the message length associated with each message passing is determined by $JL \times KL \times 4$. For example, a $(node \times 48 \times 48)$ mesh system will have a message length of 9,216 bytes.

Since each node performs the same amount of arithmetic operations in the one-dimensional partition, load balance is automatically achieved. This type of data partition can be viewed as a form of task partitioning.

In one-dimensional parallelization scheme, the number of numeric nodes required is equal to the number of grid planes, 'IL'. To locate the desired number of numeric nodes, a Delta partition (m, n) with m rows and n columns should contain 'IL' nodes. Every node in this partition has a logic number assigned to it which a programmer can access through the function call "mynode()". The returned value of "mynode()" ranges from 0 to $nm - 1$ for the (m, n) partition. Since IL equals $nm - 1$, an easy way of mapping the grid planes to the nodes is to assign grid plane number one to node 0, grid plane number two to node 1, and etc. In other word, for the ith grid plane, the computation associated with that grid plane is performed by numeric node $i - 1$. Unfortunately, this logical sequence assignment does not take advantage of the nearest neighbor message-passing priority and incurs unnecessary delay in data movement. Particularly, the message passing in Delta is row biased. If a grid plane is located at or next to a boundary of the Delta partition, the messages of this grid plane will have to be routed through 'n' or '$n - 1$' nodes before reaching its destination grid plane located in a different row of the Delta partition.

One way to avoid this delay is to use the serpentine sequence in assigning the grid planes to the numeric nodes, such that any message need not travel more than two nodes to reach its destination grid plane [11]. The serpentine arrangement can be easily achieved by the following program instruction:

$$i = \begin{cases} \text{mynode}() + 1 & x = \text{even number} \\ (2 * x + 1) * n - \text{mynode}() & x = \text{odd number} \end{cases}$$

where the row number $x = 0, 1, 2, ...m - 1$. The example of a $(4, 5)$ Delta partition with its physical layout and logical node numbers given by mynode() is shown in Table 1 (a). The indices of the grid planes mapped to the $(4, 5)$ partition using logical sequence and serpentine sequence are shown in table 1 (b) and (c), respectively.

In the pencil grouping scheme employed here, the calculations associated with two dimensions of the com-

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19

(a)

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

(b)

1	2	3	4	5
10	9	8	7	6
11	12	13	14	15
20	19	18	17	16

(c)

Table 1: (a) Logical node numbers of a $(4,5)$ Delta partition given by mynode(). (b) The grid planes mapped to the $(4,5)$ partition using Logical sequence. (c) The grid planes mapped to the $(4,5)$ partition using Serpentine sequence.

putational domain, for instance $IL \times JL$, is assigned to an (m,n) partition on the Delta. In other words, each numeric node performs the calculations associated with $(IL \times JL \times KL)/mn$ grid points. If the value of $(IL \times JL \times KL)/mn$ turns out to be an integer, each node will perform the same amount of arithmetical operations and load balance is automatically achieved. If the value is non-integer, some nodes will have more grid points to calculate than some others. In this case, an optimum assignment must be sought to balance the load. Although in some cases load balance is not automatically achieved, a pencil grouping scheme provides greater flexibility than the one-dimensional scheme in assigning grid points to the numeric nodes.

Since in the pencil grouping scheme, only the data near the surface of the pencil must be transferred to the adjacent pencils, the data flow is reduced compared with the one-dimensional partition scheme. Thus, a larger pencil size will reduce the overall data flow and enhance the concurrent performance. The number of message passings depends on the size of each pencil partition.

4 Scope of Numerical Experiments

All numerical solutions reported in this paper were processed on the Delta system. The numerical simulations were focused on a three-dimensional oscillating electric dipole. Although the physics of the oscillating dipole is well known, it is chosen because its theoretical solution [11,12] can be used for validation. Besides, the exact solution contains a singular behavior at the center of the dipole. The leading term of this singularity appears as the inverse cubic power of the radial distance from the dipole [12]. Therefore, it offers a serious challenge to the accuracy of the numerical simulation and the robustness of the solving procedure. To determine the suitable mesh spacing to capture the singular behavior of the dipole, a three-level mesh spacing refinement study was conducted. The numerical results were compared with the theoretical solutions to assess the accuracy of the simulations. Details comparisons will be reported in the next section.

In addition to the validation with the exact solutions, the results generated by the parallel codes for the one-dimensional parallelization and the pencil grouping schemes on a $(48 \times 48 \times 48)$ mesh system were compared with the previous numerical results generated on a single vector processor [5]. The parallel results are identical to the serial results.

As mentioned before, the three keys to improve parallel performance is to balance the load, shorten message routine path lengths, and minimize the number of message passings between nodes. Since for 1D parallelization, the load balance is already achieved, performance evaluation was concentrated on the latter two. In the next section, parallel performance of the logical and serpentine sequence mapping schemes will be compared using mesh systems of $(node \times 48 \times 48)$, where $node$ equals 4, 8, 16, 32, 64, 256, 384, and 512; performance comparison for different number of message passings will be carried out using $(node \times 48 \times 48)$ and $(node \times 96 \times 96)$ mesh systems. For the pencil grouping parallelization scheme, the three keys to improve parallel performance still apply. The pencil grouping performance will be reported as well.

All the parallel performances were evaluated based on the timing data obtained from the execution phase and data output phase of the codes. The execution phase includes updating the fields at all grid points from the first time level up to the last time level of the simulation; the data output phase includes writing the calculated field results and the coordinates of the grid system to the disk for post processing. The program initialization time of the present code is negligible compared to the program execution and data output time; therefore, it will not be reported here. (In the following text, data output is sometime referred to as data I/O since the output performance is tied into the I/O performance of the computer system.)

The execution times were collected at the 480th time level. The duration of 480 time steps was selected for all calculations such that the wave traversed more than ten times the distance across the entire computational domain. This time duration is deemed sufficient for the numerical procedure to yield meaningful results and repeatable timing data. From the execution time, the data processing rate and scale factor are derived. At each time level and grid point, both one-dimensional and the pencil grouping approach must perform at least 492 mixed integer and real number arithmetic operations. The data processing rate is computed by the total number of arithmetic operations performed during the 480 time steps divided by the maximum and the minimum time elapsed of all nodes in use. These minimum and maximum values bracket the range of performance among nodes. In other word, the data processing rate in Gigaop (10^9 arithmetic operations per second) is calculated by multiplying the numbers of iterations, arithmetic operations, and total grid points, then dividing by the executing time ($480 \times 492 \times (IL \times JL \times KL)/$ (execution time $\times 10^9$)). This calculated data rate is a conservative estimate because the number 492 does not include 47 basic library function calls and nine system message passing calls at every time step. These library function calls include mostly trigonometry and square root calculations. Together with the system message passing calls, they may account for a significant amount of the execution time. The scale factor of the code is defined by the time elapsed for each array of nodes used then normalized by the execution time required for four nodes. The execution time for the four-node calculation was used as timing basis because in the present code, at least four grid points are required to specify the boundary conditions at the dipole and at the far field.

The data output phase of the one-dimensional parallelization scheme requires 78 arithmetic operations and 38 system message passing calls. Most of the 38 system calls are either synchronous or asynchronous message passing instructions to generate one unformatted field data file and three unformatted grid files. The field data file contains all six components of the electromagnetic field at every grid point; and the grid files contain the three coordinates of the grid system. The longest and the shortest time periods required to output those unformatted files for 4-node up to 512-node simulations were recorded. From these timing data, the scale factor for each simulation was again normalized by the 4-node result to determine the possible performance degradation of data management.

5 Discussion of Results

5.1 Mesh refinement study

The originally planned mesh refinement included mesh systems of $(48 \times 48 \times 48)$, $(96 \times 96 \times 96)$, and a finest mesh system of $(192 \times 192 \times 192)$. However, as the number of processors increased beyond 96, the disparity in execution times among processors grew to a factor of 11.6 for 108 nodes, and 14.43 for 144 nodes. In other words, the observed scalable performance of the present computer program for the cases of $(48 \times 48 \times 48)$ and $(96 \times 96 \times 96)$ was not sustainable for the mesh system of $(192 \times 192 \times 192)$. Therefore, the mesh refinement study was forced to reduce the scope.

Three mesh systems of $(24\times24\times24)$, $(48\times48\times48)$, and $(96 \times 96 \times 96)$ were used for the numerical resolution study instead. For this reduced scope of numerical experiment, the coarse mesh system has only 24 grid points in each coordinate. The mesh point density is sufficient to resolve the wave motion [1,2] away from the dipole, but it may be deficient in simulating the singular field behavior near the dipole [11,12].

The mesh refinement results for the radial component of the electric field are presented in Figure 2. The calculation from the $(24 \times 24 \times 24)$ mesh system revealed significant numerical oscillations in an attempt to overcome the large truncation error near the dipole. As shown in Figure 2, the results exhibits a steady improvement as the mesh systems becomes finer. However, the finest mesh is still not fine enough to overcome the stringent demand for accurate simulation near the dipole. Nevertheless, the present numerical procedure has demonstrated the robustness in treating the problem containing a singular behavior.

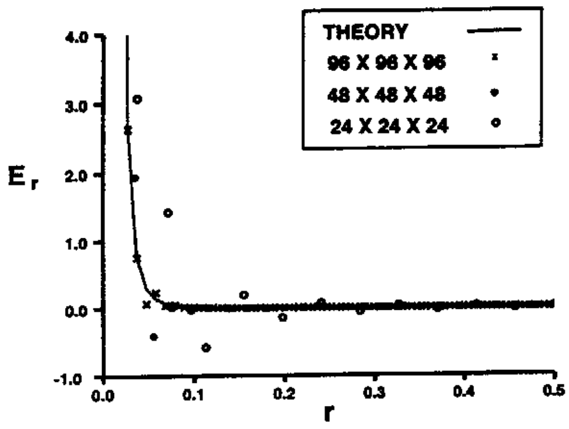

Figure 2: Comparison of computed radial components of the electric fields with theory

Figure 3 depicts the numerical results of the circumferential component of electric intensity on the three mesh systems. The numerical result obtained on the finest mesh attained the best agreement with the analytical result. The maximum deviation from the theoretical results is merely a fraction of one percent. Another desired feature of the characteristic-based formulation also stands out. At a non-dimensionalized radial distance of 0.14 and beyond, all three solutions agree well with the theory and show no wave reflection from the truncated numerical boundary.

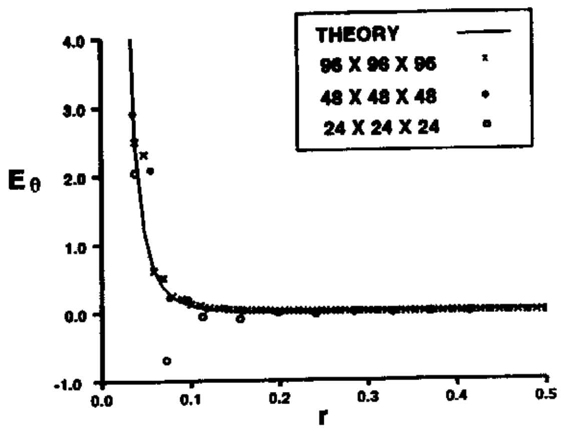

Figure 3: Comparison of computed circumferential components of the electric fields with theory

The comparison of magnetic azimuthal component of the three mesh systems is given in Figure 4. Although the leading term singularity of the magnetic field has a lower order asymptote than that of the electric field, the numerical solutions of the magnetic field generated on the three mesh systems behave similarly to those of the circumferential electric field component. The numerical resolution produced by the finest mesh system, however, is able to suppress the numerical oscillation near the dipole. Finally, the reflected wave from the truncated numerical boundary is completely absent.

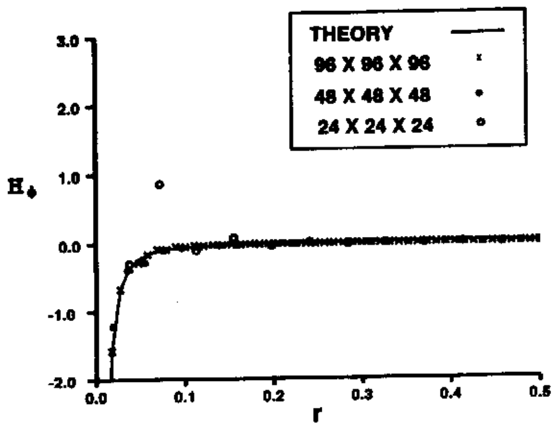

Figure 4: Comparison of computed azimuthal components of the magnetic fields with theory

5.2 Serpentine sequence vs. logical sequence

In this experiment, the one-dimensional parallelization scheme and the grid system (*node* × 48 × 48) were used so that every node performs exactly the same amount of computations and writes out the same amount of outputs.

Table 2 shows the data processing rates in Gigaops. The performance difference between the logical sequence and the serpentine message path arrangements begins to appear for the number of nodes beyond 128. Since the serpentine arrangement took advantage of the nearest neighbor priority in message passing hierarchy, the performance degradation is less than that of the logical sequence. In fact, to complete a 512-node computation using serpentine procedure, the slowest node required only 3.1 percent more execution time than the fastest node; whereas, in the logical sequence arrangement, the slowest node is 10.4 percent slower compared with the fastest node. For the same 512-node simulation, the fastest node operating on the serpentine arrangement attained a data processing rate of 5.966 Gigaops; while the fastest node on the logical sequence arrangement only attained 5.412 Gigaops.

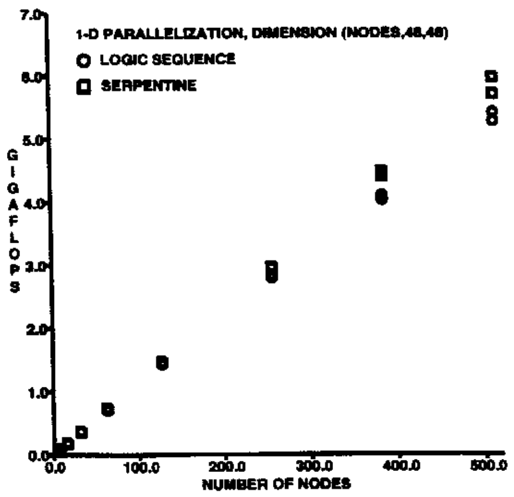

Figure 5: Data processing rates on the Delta

The data processing rates of the serpentine and logical sequences are plotted in Figure 5. The improved data processing rate of the serpentine sequence over that of the logical sequence is clearly demonstrated. As can be seen from the figure, the performance of the logical sequence, which did not honor the data passing priority of the nearest neighbor hierarchy, started to lag behind that of the serpentine sequence when more than 128 numeric nodes were used. From this experiment, the serpentine procedure appears to be more effective for the distributed memory system. However, for both sequences, an increasing disparity of the data processing

	Serpentine		Logical Sequence	
Number of nodes	Maximum	Minimum	Maximum	Minimum
4	0.046	0.046	0.046	0.046
8	0.092	0.092	0.092	0.092
16	0.184	0.184	0.184	0.169
32	0.368	0.368	0.369	0.369
64	0.738	0.738	0.736	0.707
128	1.472	1.472	1.445	1.430
246	2.967	2.952	2.829	2.798
384	4.490	4.413	4.105	4.028
512	5.966	5.704	5.412	5.274

Table 2: Comparison of data rates between serpentine and logical sequences of an (*node* × 48 × 48) mesh system

rate among nodes was noted as the number of nodes increased beyond 64.

The scale factors of the two mapping schemes are demonstrated in Figure 6. All data processing rates are normalized by the value of the 4-node simulation. The superior scalable property of the serpentine sequence over that of the logical sequence is obvious. The scalability of the present code up to 512 nodes is perceived within a performance degradation of less than 3.1 percent. In fact, the significant degradation only appeared when all 512 numeric nodes were employed.

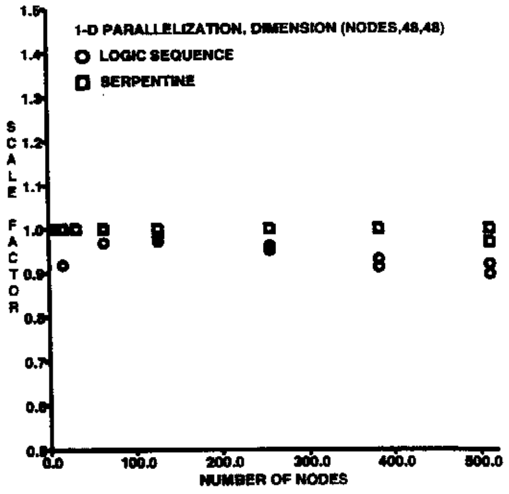

Figure 6: Scalability of executing time on the Delta

Figure 7 presents the data output time in seconds. The data output time varied widely from 0.24 to 32.20 seconds for the serpentine sequence, and from 0.28 to 162.80 seconds for the logical sequence. The shortest waiting time over the entire range of nodes used was only 0.24 seconds. Unfortunately, the aggregated time required by the slowest node in computation and data output actually determines the completion of a numerical simulation. From Figure 7, the data output time increases almost linearly with the number of nodes in use. For the 512-node calculations, the serpentine sequence used 32.2 seconds while the logical sequence needed 162.8 seconds to output the same amount of data.

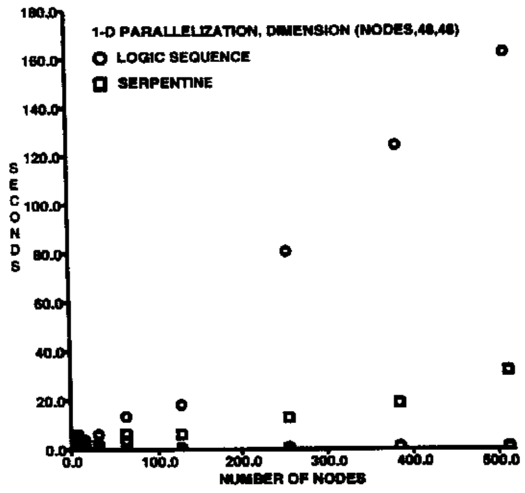

Figure 7: Data I/O time on the Delta

Figure 8 depicts the widely varying data I/O performance among nodes for the entire range of available nodes. Again the performance degrades rapidly as the number of nodes in use increases. The logical sequence yields the maximum parallel performance discrepancy among nodes. The ratio between the most and least efficient nodes is as high as 580.4. The serpentine sequence reduces the disparity to a value about 134.2. If this behavior is a common trend for all distributed memory computer systems, this deficiency shall be a pacing item for research in concurrent computing.

Concurrent performance, similar to those discussed above, were also observed for (96 × 96 × 96) and (96 × 192 × 192) computations. The data processing rate per node was maintained in the range of 12.89 to 11.65 Megaops. These data processing rates are far below the nominal 60 to 80 Megaops performance level of the i860 microprocessor [13]. Additional performance improvement of the present numerical procedure using the Delta system is still possible.

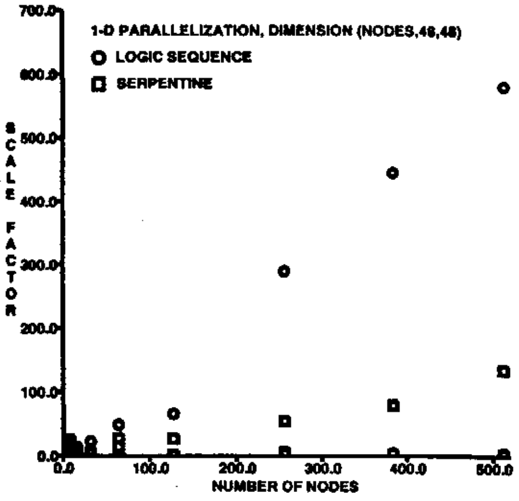

Figure 8: Non-scalability of data I/O on the Delta

5.3 Comparison between (*node* × 48 × 48) and (*node* × 96 × 96) mesh systems

For the one-dimensional parallelization scheme, the vector length is doubled and the message passing length is quadrupled from the coarse grid of (*node* × 48 × 48) to the refined grid of (*node* × 96 × 96). Since the serpentine mapping scheme was proven to give better parallel performance, it was used for the experiment reported in this subsection.

For the (*node* × 48 × 48) systems, the execution time for each node varied from 45.61 to 52.90 seconds regardless of whether the computations were conducted on 4 or 512 numeric nodes. For the (*node* × 96 × 96) case, the execution time ranges from 158.37 to 173.16 seconds. The data processing rates based on the above execution time is given in Table 3. The maximum data processing rate of the refined mesh system is 9.8 percent greater than the coarser mesh computation; and the highest achieved value was 6.551 Gigaops. Apparently at a message passing length of 36,864 bytes and 8 message passings per node per time step, the higher data processing rate by a greater vector length has not been hampered by the limiting memory bandwidth. As a result, the deviation between the fastest and the slowest execution rates is less than one percent.

The execution times expressed in scale factors for the (*node* × 96 × 96) and (*node* × 48 × 48) systems are shown in Figure 9. Small variations of data processing rates for both grid systems were observed over the entire range of nodes used. The results of the coarse grid computations exhibited a wider performance deviation among nodes than the finer grid calculations. The latter, however, also degraded noticeably from the 4-node to the 8-node result. The difference is about 4.6 percent. As the number of nodes increases to 512, the degradation of scalability of the present procedure reached the maximum value of 7.6 percent. But the major portion of the performance deficiency is incurred at the 4-node to 8-node transition. In all, the present algorithms mapped to Delta system has demonstrated an acceptable scalability of data processing rate up to a grid system consisting of 512 × 96 × 96 grid points.

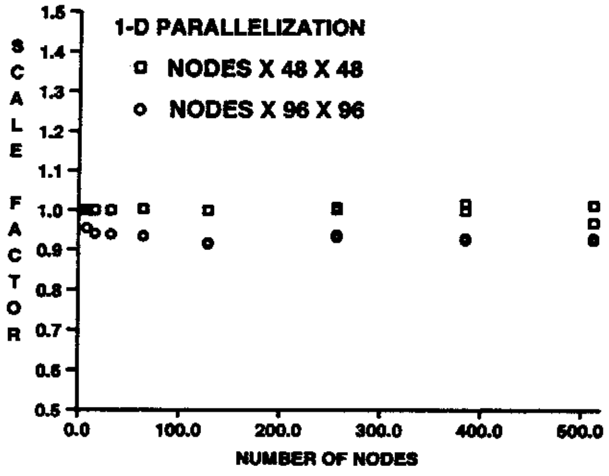

Figure 9: Execution time scale factors for the coarse and fine mesh systems

The data output time varies from 0.2369 to 32.20 for the coarse grid system and from 0.1324 seconds for the 16-node calculation to 30.02 seconds for the 512-node simulation for the finer grid system. Figure 10 shows the scale factors of the output timing results for the coarse and refined mesh systems. Clearly, the I/O performance degrades nearly linearly as the number of nodes in use increases. The worst I/O performance occurred when all 512 nodes were engaged. In this case, the fastest and the slowest nodes used 11.51 and 226.8 times, respectively, the time required to output the same amount of data as the fastest node of the 16-node computation. This observation further asserts that the scalable I/O performance shall be a pacing item for research in concurrent computing.

5.4 Pencil grouping scheme

Owing to similar peculiarities of the Delta compiler observed for the one-dimensional parallelization, only a limited number of timing data from the pencil partition is obtained at present time. In short, the serpentine sequence did improve the performance over that

	node × 96 × 96		*node* × 48 × 48	
Number of nodes	Maximum	Minimum	Maximum	Minimum
4	0.055	0.055	0.046	0.046
8	0.105	0.105	0.092	0.092
16	0.207	0.207	0.184	0.184
32	0.413	0.413	0.368	0.368
64	0.823	0.822	0.738	0.738
128	1.616	1.609	1.472	1.472
246	3.305	3.278	2.967	2.952
384	4.915	4.884	4.490	4.413
512	6.551	6.505	5.966	5.704

Table 3: Comparison of data rates between *node* × 96 × 96 and *node* × 48 × 48 mesh systems

Delta	$IL \times JL$ (Pencil Dimension)													
(m,n)	192x1	96x2	64x3	48x4	32x6	24x8	16x12	12x16	8x24	6x32	4x48	3x64	2 x96	1x192
(6,32)	13.4	9.73	10.6	10.7	12.1	13.1	14.2	15.7	16.8	16.8	16.7	14.3	13.2	8.94
(8,24)	12.6	10.6	11.6	12.1	13.0	13.9	15.7	16.8	17.3	16.4	16.9	14.0	13.2	9.55
(12,16)	12.5	10.6	12.0	13.1	14.0	15.7	17.0	18.2	17.3	16.8	16.5	14.2	13.3	9.75
(16,12)	13.1	11.3	13.1	13.9	15.4	16.2	18.4	17.5	17.1	16.2	14.3	13.2	13.2	10.1

Table 4: Execution rates (Megaops) for a (192 × 192 × 192) grid system by pencil grouping scheme

of the logical sequence as the one-dimensional case. The timing information for a (192 × 192 × 192) mesh system mapped to the Delta using pencil grouping scheme with various combinations of pencil dimension ($IL \times JL$) and Delta partition (m, n) is tabulated in Table 4. The variation of data processing rates among the different pencil partitions for a fixed Delta partition is significant. In general, slower performance is observed when the pencil dimension and Delta partition are at the extreme situations when the pencil grouping scheme degenerates into one-dimensional parallelization. For example, the pencil dimensions of (192 x 1) or (1 x 192) on a Delta partition of (6,32) has a slower data processing rate of 13.2 or 8.94 Megaops, respectively.

A general performance improvement for an ($IL \times JL$) pencil dimension is noted when IL nearly equals JL. The best performance for a given Delta partition is observed when IL equals m and JL equals n, as underlined in Table 4. The timing data of the pencil partition has revealed a parallel efficiency gain of 37% over that of the one-dimensional partition.

6 Conclusions

A fractional-step, upwind numerical procedure for solving the three-dimensional, time-domain Maxwell equations has been ported to the Intel Touchstone Delta multicomputer. The concurrent computations duplicated the results from earlier numerical results and compared well with theory. For the mesh systems of (*node* × 48 × 48) and (*node* × 96 × 96), the numerical procedure is scalable up to 512 numeric nodes with only up to 7.6 percent performance degradation. The fastest data processing rate is 6.551 Gigaops and the sustained overall performance is clocked at 5.704 Gigaops. Further increased data processing rate is still possible.

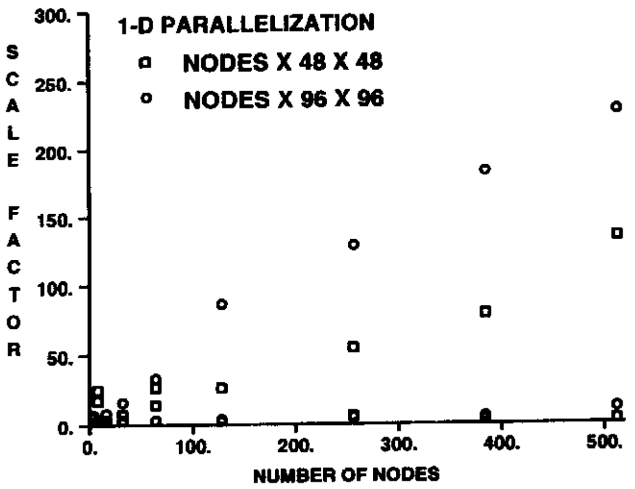

Figure 10: I/O scale factors for the coarse and fine mesh systems

The scalability of concurrent computing is sustained up to a simulation eight times the size of the baseline case. The scalable performance failed at the fourth level of grid point enrichment ($192 \times 192 \times 192$). Although the architecture of the Touchstone Delta multicomputer and its usefulness are impressive, consistent performance in scaling up for massive data bases remains as a necessary research emphasis. The scalable data I/O is also identified as a pacing item for intense research for attaining high performance computation.

The one-dimension parallelization has been shown as a suitable data partition procedure for a characteristic-based, windward difference algorithm in solving the time dependent, three dimensional Maxwell equations. Further parallel efficiency improvement by pencil structure shows developable potential.

Acknowledgement The computing resource was provided by the Intel Touchstone Delta System operated by Caltech on behalf of the Concurrent Supercomputing Consortium. Access to this facility was granted by Dr G. Weigand of ARPA. Stimulating and fruitful discussions with Stephen J. Scherr and B.G. Vikstrom are also appreciated.

References

1. Taflove, A., Re-Inventing Electromagnetics: Supercomputing Solution of Maxwell's Equations Via Direct Time Integration on Space Grids, Computing Sys. in Engineering, Vol. 3, Nos 1-4, 1992, pp. 153-168.

2. Shankar, V., Hall, W. F., and Mohammadian, A. H., A Time-Domain Differential Solver for Electromagnetic Scattering Problems, Proceedings of IEEE, Vol. 77, No. 5, May 1989.

3. Shang, J. S., Characteristic-Based Methods for the Time-Domain Maxwell Equations, AIAA Paper 91-0606, presented at AIAA meeting held in Reno, NV, January 1991

4. Shang, J. S., A Characteristic-Based Algorithm for Solving 3-D, Time-Domain Maxwell Equations, AIAA Paper 92-0452, presented at AIAA meeting held in Reno, NV, January 1992.

5. Shang, J. S., A Fractional-Step Method for Solving 3-D, Time- Domain Maxwell Equations, AIAA Paper 93-0461, presented at AIAA meeting held in Reno, NV, January 1993.

6. Roe, P. L., Characteristic-Based Schemes for the Euler Equations, Ann Rev. Fluid Mech., Vol 18, 1986, pp 337-365.

7. Steger, J. L., and Warming, R. F., Flux Vector Splitting of the Inviscid Gasdynamics Equations with Application to Finite Difference Methods, J. Comp. Phys. Vol. 40, No. 2, February 1987, pp 263-293.

8. MacCormack, R. W., and Candler, G. V., The Solution of the Navier-Stokes Equations Using Gauss-Seidel Line Relaxation, Computer & Fluid Vol. 17, No. 1, 1989, pp 135-150.

9. Wesley, R., Wu, E., and Calahan, D. A., A Massively-Parallel Navier-Stokes Implementation, AIAA Paper 89-1940CP, 1989.

10. Scherr, S. J., Implementation of an Explicit Navier-Stokes Algorithm on a Distributed Memory Parallel Computer, AIAA Paper 93-0063, January 1993.

11. Harrington, R. R., Time-Harmonic Electromagnetic Fields, McGraw-Hill Book Co., New York, 1961.

12. Jordan, E. C., Electromagnetic Waves and Radiation System, Prentice-Hall, Inc., NJ, 1960.

13. Intel Corporation, Touchstone Delta System User's Guide, Publication 312125-001, October 1991.