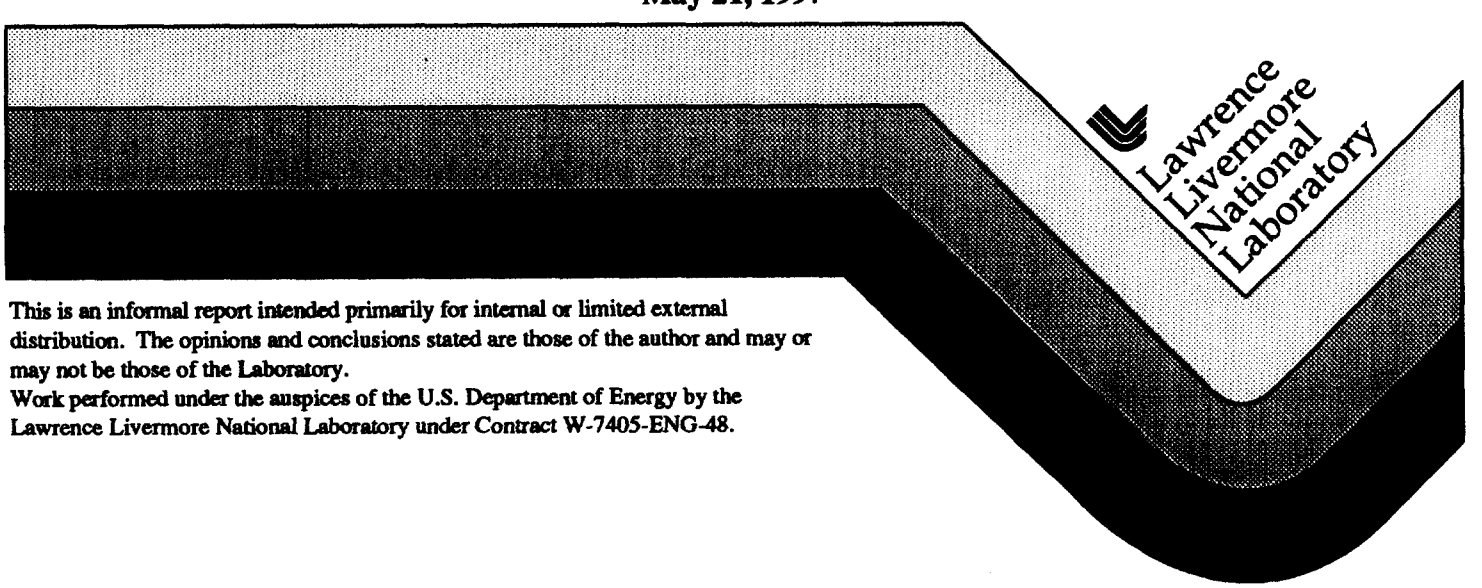
UCRL-ID-127450

Parallel Computation of Electromagnetic Fields

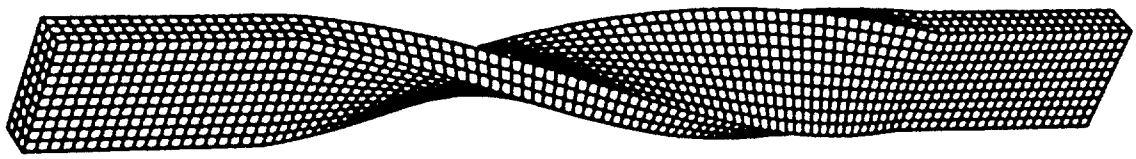
N. K. Madsen

May 21, 1997



Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under Contract W-7405-ENG-48.

Parallel Computation of Electromagnetic Fields



May 21, 1997

Niel Madsen

Lawrence Livermore National Laboratory

Table of Contents

Introduction

The DSI3D code is designed to numerically solve electromagnetics problems involving complex objects by solving Maxwell's curl equations in the time-domain and in three space dimensions. The code has been designed to run on the new parallel processing computers as well as on conventional serial computers.

The DSI3D code is unique for the following reasons:

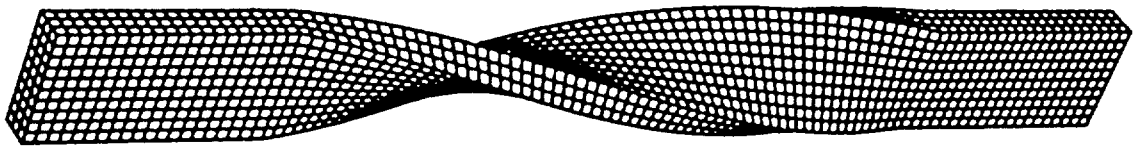
- It runs efficiently on a variety of parallel computers,
- Allows the use of unstructured non-orthogonal grids,
- Allows a variety of cell or element types,
- Reduces to be the Finite Difference Time Domain (FDTD) method when orthogonal grids are used,
- Preserves charge or divergence locally (and globally),
- Is non-dissipative,
- Is accurate for non-orthogonal grids.

This method is derived using a Discrete Surface Integration (DSI) technique. As formulated, the DSI technique can be used with essentially arbitrary unstructured grids composed of convex polyhedral cells. This implementation of the DSI algorithm allows the use of unstructured grids that are composed of combinations of non-orthogonal hexahedrons, tetrahedrons, triangular prisms and pyramids. This algorithm reduces to the conventional FDTD method when applied on a structured orthogonal hexahedral grid.

Overview

Discrete Surface Integration Method

We begin by assuming that we wish to solve Maxwell's curl equations on an irregular three-dimensional domain R that has a boundary surface denoted by S. We will also assume that the domain R has been discretized into convex polyhedrons. The figure below shows a twisted waveguide discretized using hexahedral cells.



Twisted waveguide discretized using distorted hexahedral cells.

This is an example of a problem type that we wish to consider that could not be easily solved using the conventional orthogonal grid FDTD method. Maxwell's curl equations are given by:

$$\frac{\partial \mathbf{D}}{\partial t} = \nabla \times \mathbf{H} \tag{1}$$

$$\frac{\partial \mathbf{B}}{\partial t} = -\nabla \times \mathbf{E} \tag{2}$$

where for linear isotropic materials the vectors **D**, **E**, **B** and **H** are related by the constitutive relationships

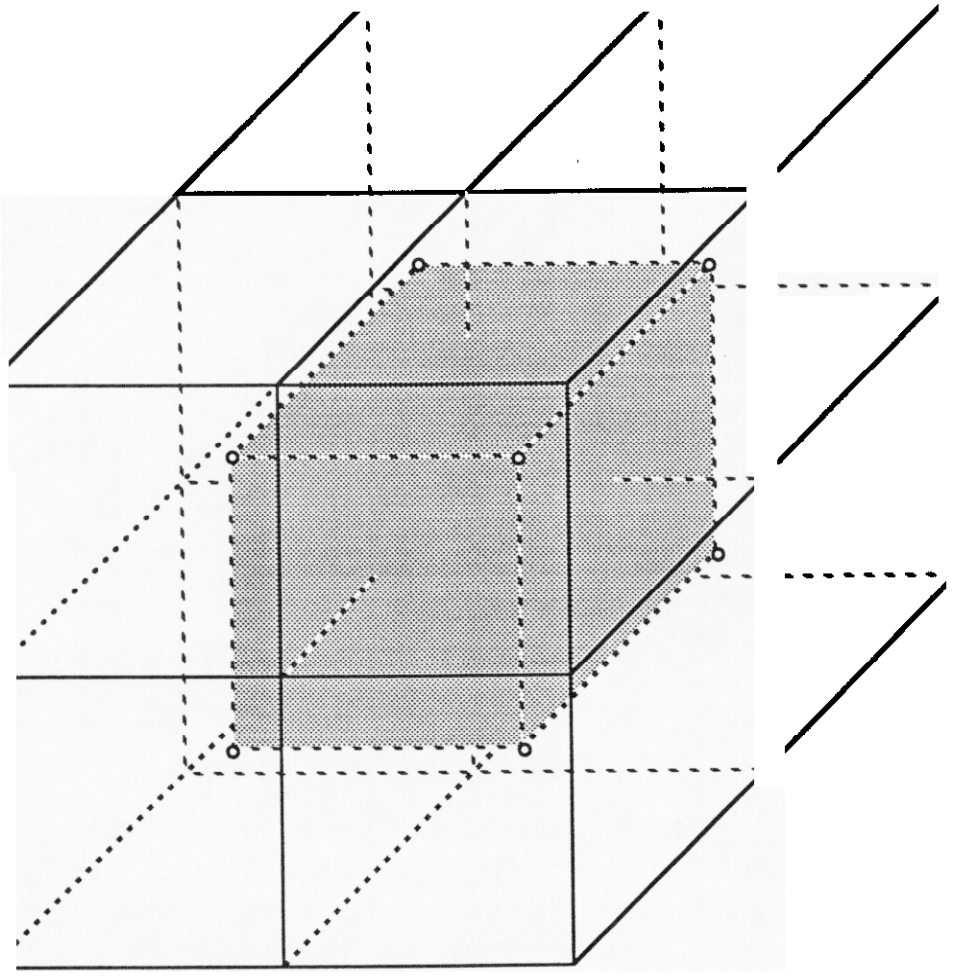
$$\mathbf{D} = \varepsilon \mathbf{E}$$
$$\mathbf{B} = \mu \mathbf{H}$$

The linear isotropic material properties are: e, the permittivity, and m, the permeability. Like the conventional FDTD method, the DSI methods can be generalized to treat more complex materials. However, for the purposes of this discussion, we will assume that the linear isotropic material properties are piecewise constant over the domain R. We will also assume that S is a perfectly conducting

surface, i.e., $E_{tan} = 0$. This is sufficient to guarantee that the problem is well-posed.

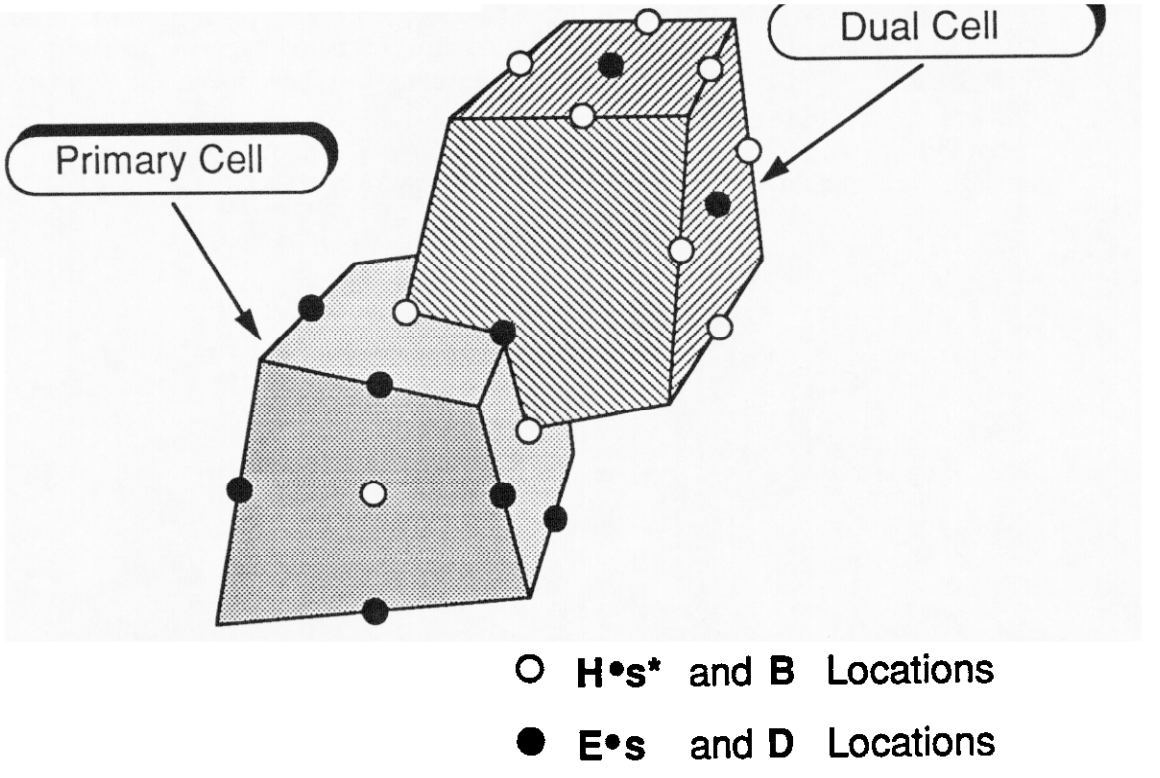
The DSI algorithm[1] is applicable for all unstructured grids formed from convex polyhedral cells. We restrict the choice of cell types to convex polyhedrons whose edges are straight lines. The faces of the polyhedrons are not necessarily planar and we make the assumption that any face in the assembled grid is shared by at most two cells. These are very weak restrictions and allow a great deal of flexibility.

The DSI method is complicated in that it requires the use of a dual grid[1]. There is a one-to-one correspondence between the nodes, edges, faces, and cells of the primary grid to the cells, faces, edges and nodes of the dual grid, respectively. A dual face associated with a primary edge has as its perimeter the dual edges associated with all of the primary faces which share the given primary edge. The dual cell associated with a primary node has as its surface the dual faces which correspond to all of the primary edges which share the primary node. The following figure shows an eight cell hexahedral primary grid and its one interior dual cell.



Primary grid consisting of eight hexahedral cells and its one interior dual cell.

The DSI solution variables are associated with the edges and faces of the primary grid and also with the edges and faces of the dual grid. The quantity associated with a primary cell edge is the projection of the electric field vector onto that edge, i.e., $\mathbf{E}\cdot\mathbf{s}$, where $\mathbf{s}$ is the primary cell edge vector. The magnetic field projection $\mathbf{H}\cdot\mathbf{s}^*$ is associated with a dual cell edge where $\mathbf{s}^*$ is the dual cell edge vector. In addition, with each primary grid face we will associate a full magnetic field vector **B**, and with each dual grid face we will associate a full electric displacement vector **D**. We will denote with an asterisk, *, geometric quantities associated with the dual grid. The following figure depicts these associations.



Discrete electric and magnetic field variable locations relative to the primary and dual grid cells.

We remark that these associations of field quantities with the primary and dual grid locations are entirely reciprocal and that the respective locations of the magnetic and electric field quantities could be interchanged. The particular choice of which field quantities to associate with each grid is best determined by deciding which field quantities one desires to have on the exterior boundary surfaces where the boundary conditions will be imposed. Since we are assuming that our domain R is surrounded by a perfect electric conductor, we will associate the electric and magnetic field quantities as described above. For open region problems, the choice for the location of the field quantities will depend on the particular radiation boundary condition algorithm used.

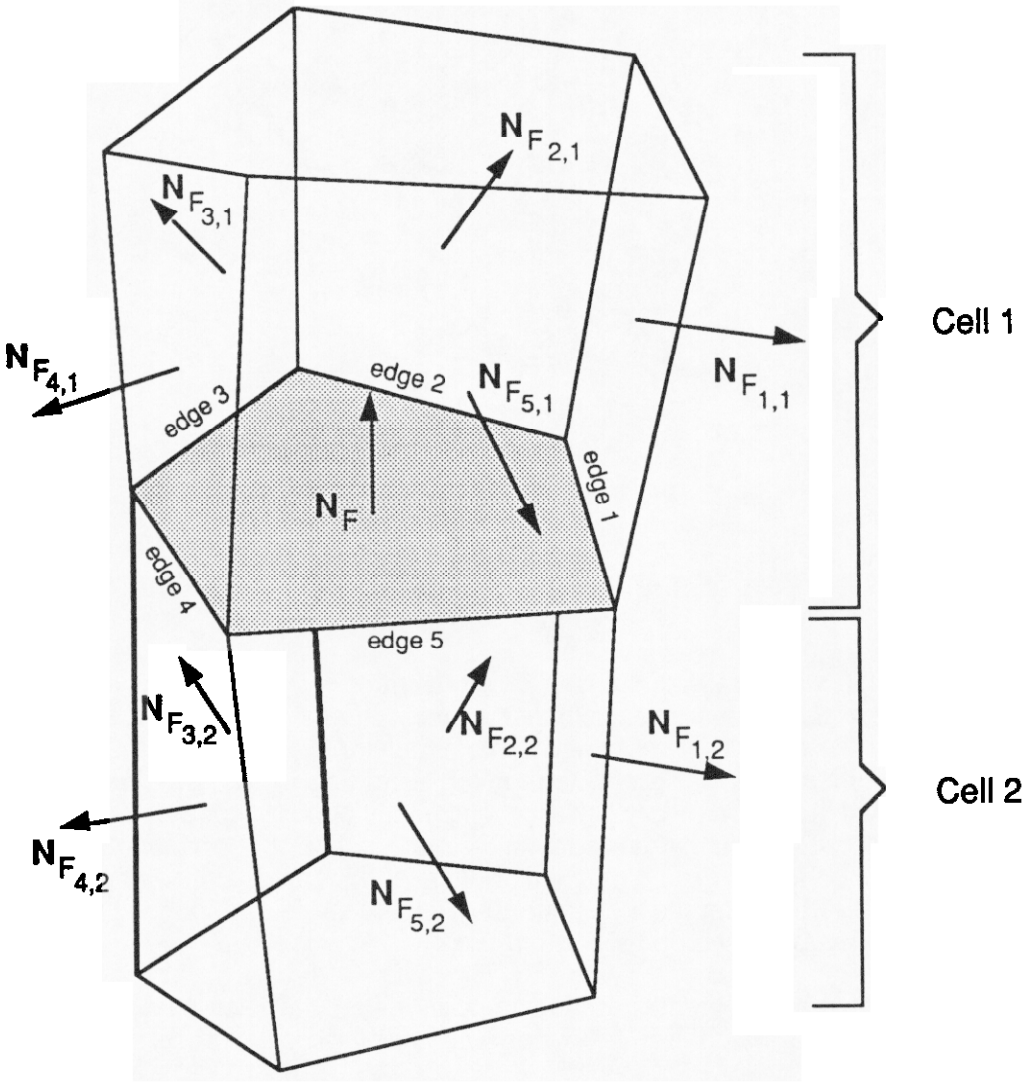
The basic algorithmic process used to advance in time the electric and magnetic field vectors is rather straightforward. The following briefly describes the process for the magnetic field. For a particular primary cell face, we define the area-normal vector to be $\mathbf{N} = \int \mathbf{n}\, dS$, where $\mathbf{n}$ is a unit surface normal defined by the right-hand rule in relation to a specified circulation

around the perimeter of the cell face. Using (2) for the primary grid face, F, we define the time derivative of the normal component of the magnetic field to be:

$$\frac{d\mathbf{B}^k}{dt}\cdot\mathbf{N}_F \equiv \int_F\left(\frac{\partial \mathbf{B}^k}{\partial t}\cdot\mathbf{n}\right)dS = -\int_F\left(\nabla\times\mathbf{E}^k\right)\cdot\mathbf{n}\,dS$$

$$= \oint_F \mathbf{E}^k\cdot d\mathbf{l} \qquad (3)$$

Equation (3) allows us to obtain the time derivative of the normal component of the magnetic field on a primary face, F, from the electric field projections onto the perimeter edges of that face. The last integral in (3) is easily computed numerically by summing these edge projections. This can be done for each primary cell face. From these projected time derivatives, the full vector value of $d\mathbf{B}_F^k/dt$ may be computed. We will denote by $F_{i,j}$, the face of cell j (other than F) which shares edge i. The following figure depicts these associations for a dual edge associated with a primary face defined by five primary edges.



The primary grid faces used to time advance a magnetic field vector.

At each of the P nodes of face F, we will unfold N_c vector values $d\mathbf{B}_{ij}^k/dt$ by solving the 3×3 system of equations:

$$\begin{aligned}
\frac{d\mathbf{B}_{ij}^k}{dt}\cdot\mathbf{N}_F &= -\oint_F \mathbf{E}^k\cdot d\mathbf{l} \\
\frac{d\mathbf{B}_{ij}^k}{dt}\cdot\mathbf{N}_{F_{i,j}} &= -\oint_{F_{i,j}} \mathbf{E}^k\cdot d\mathbf{l} \\
\frac{d\mathbf{B}_{ij}^k}{dt}\cdot\mathbf{N}_{F_{m,j}} &= -\oint_{F_{m,j}} \mathbf{E}^k\cdot d\mathbf{l}
\end{aligned} \tag{4}$$

where $i=1,\ldots,P$, $j=1,\ldots,N_c$, and $m=(i \bmod P)+1$. For this primary face, F, which is shared by N_c primary cells, there are $P\,N_c$ different values of $d\mathbf{B}_{ij}^k/dt$, which will now be averaged or interpolated to form a single $d\mathbf{B}_F^k/dt$ vector for the face. The particular averaging or interpolation we use is given by:

$$\frac{d\mathbf{B}_F^k}{dt} = \frac{\sum_{j=1}^{N_c}\sum_{i=1}^{P}|w_{ij}|\dfrac{d\mathbf{B}_{ij}^k}{dt}}{\sum_{j=1}^{N_c}\sum_{i=1}^{P}|w_{ij}|} \tag{5}$$

where the weight

$$w_{ij} = N_F\cdot\left(N_{F_{i,j}}\times N_{F_{i+1,j}}\right),$$

represents the volume of the j^{th} local coordinate system at node i of face F. We note also that the weight w_{ij} is the determinant of the system of equations (4).

The full **B** vector for each primary face, F, may now be advanced in time using the time-centered leapfrog algorithm:

$$\mathbf{B}^{k+\frac{1}{2}} = \mathbf{B}^{k-\frac{1}{2}} + \Delta t\frac{d\mathbf{B}^k}{\mathbf{dt}}, \tag{6}$$

where Dt is the specified time step size. Finally, a time advanced value of the projection of the magnetic field onto the dual edge, $\mathbf{s}^*$, which penetrates the primary cell face, F, is easily obtained using:

$$\left(\mathbf{H}\cdot\mathbf{s}^*\right)^{k+\frac{1}{2}} = \frac{\mathbf{B}^{k+\frac{1}{2}}\cdot\mathbf{s}^*}{\mu}, \tag{7}$$

where μ is an appropriate permeability value.

To time advance the electric field vectors which are associated with each dual cell face, we proceed in a manner which is exactly "dual" to the magnetic field procedure described above.

Equations (3)-(7) for the magnetic field quantities, and similar dual equations for the electric field quantities constitute the divergence conserving DSI approximation method.

When orthogonal hexahedral based grids are used, many simplifications occur in the above described algorithm. And in fact, the DSI algorithm reduces to be the canonical Yee[2] FDTD algorithm.

Parallel Processing and Problem Partitioning

To efficiently use multiple processors on a parallel computer to solve any one problem, it is required to split or partition the problem among the available processors. With the problem partitioned, each processor can produce the solution for its part of the problem. In order for this process to be efficient, there are to fundamental goals any partitioning process.

- Each processor should have an equal workload.
- The amount of inter-processor communication required should be minimized.

For DSI3D, this partitioning ultimately divides up the primary and dual edge variables $\mathbf{E} \cdot \mathbf{s}$ and $\mathbf{H} \cdot \mathbf{s}^*$. For some structured grids it is intuitively obvious how to partition the problem so that the above stated goals are achieved. For unstructured grids and particularly for tetrahedral based grids adequate partitioning schemes are very non-intuitive and difficult to construct. Fortunately, recent work by Karypis and Kumar[7] and earlier work by H. Simon[8] has produced some automatic approaches for solving this problem.

The new approach is embodied in the codes called METIS and PARMETIS[7]. The basic approach comes from methods used to partition graphs which are associated with solution processes for sparse matrix problems. The details of the METIS approach as well as comparisons with other approaches may be found in the above references. For DSI3D we have chosen to use the grid cells as the basic quantities to be partitioned. This is primarily done because there are significantly fewer cells than edges and faces in the primary and dual grids. This results in a smaller and more efficient partitioning process.

First, a graph of the connectivity of the cells is constructed. We choose to define that two cells are connected as meaning that they share a face (other possibilities are that they share a node or an edge). The nodes of the graph represent the grid cells and the graph edges which connect two nodes represent that the two cells share a common face. This graph is input to METIS which splits the graph into "n" pieces where n is the number of processors we desire to use to solve the electromagnetics problem. METIS accomplishes this by coarsening the graph by forming "maximal sets" by identifying sets of graph edges which have no nodes in common. These edges are "removed" from the graph by coalescing the two nodes which are associated with each identified edge. Thus, the graph size is reduced effectively by a factor of 2. This is done successively until a resulting graph of small size is obtained. This small graph is then partitioned into n pieces using a spectral method[8]. The coarsening process is then reversed and the smaller partition pieces are expanded repeatedly until one reaches the original graph. This multi-level graph partitioning strategy has resulted in partitioning speed increases of roughly a factor of 100.

Boundary Conditions

Perfect Electric Conductor (PEC) boundary conditions are some of the most commonly used boundary conditions when modeling problems involving metallic objects. PEC conditions require that the total tangential electric field be set to zero on the PEC surface. This is easily accomplished because the tangential electric field components, $\mathbf{E} \cdot \mathbf{s}$, live on the boundary and we simply set $\mathbf{E} \cdot \mathbf{s} = E_{tan} = 0$ for the total field.

Perfect Magnetic Conductor (PMC) symmetry plane boundary conditions are another useful boundary condition for certain types of problems. They frequently allow a problem to be solved using a grid that is half as big as the full grid for the problem. These conditions require that the total tangential magnetic field be set to zero on the specified surface. We desire that these conditions be applied on the surface of the primary grid which is usually where tangential electric field information is specified. We accomplish this by using reflection principles and images which are inherent with this type of planar boundary.

Radiation Boundary Conditions (RBC) are an absolute necessity for computing electromagnetic field values in open region problems. These conditions allow one to truncate the grid at a reasonable distance from the scatterer and thus avoid having to discretize extremely large volumes of space. These conditions are designed to allow scattered EM wave to leave the computational region without significant reflections. Achieving good RBC conditions is usually a very nontrivial process and much research has been performed looking for better and better RBC algorithms.

The following RBCs are presently included in the code:

1) First order Higdon[3] (Mur[5])
2) First order Liao[4]
3) First order Taylor[6]
4) Modified second order Mur[5]
5) Second order Higdon
6) Second order Taylor
7) Second order Taylor-Higdon[6]

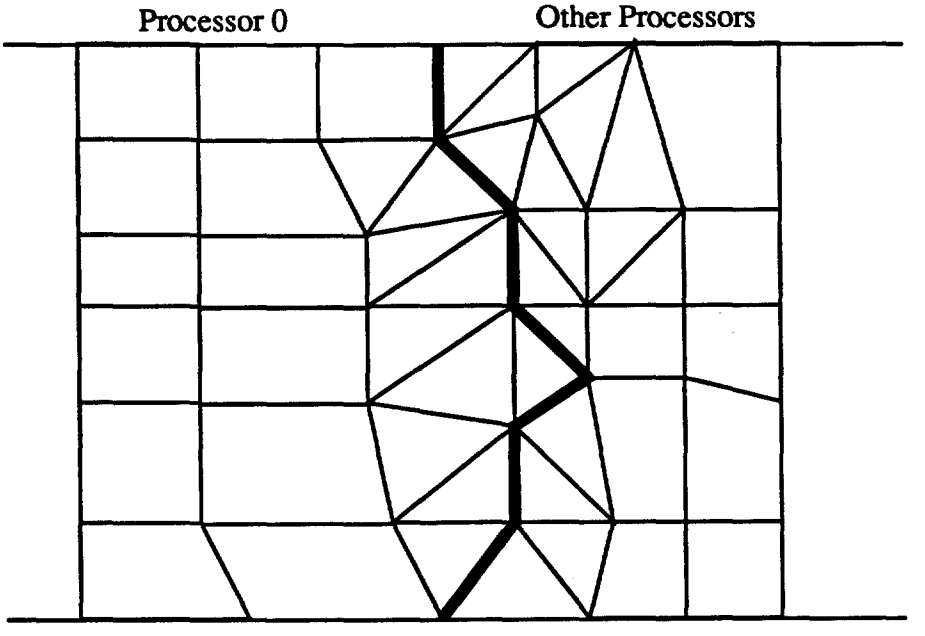
The modified form of the second order Mur RBC uses only stencil components which are normally directed into the problem space [6].

These boundary conditions have been chosen because their stencils allow efficient communication on MPP machines and allow the user a moderate amount of freedom in generating the grid. The Higdon and Mur boundary conditions are one-way wave operators and the Taylor and Liao boundary conditions are Taylor series approximations [6].

This simple sounding approach actually works quite well in practice. There is mathematical theory that states that if the original grid is connected that each of the produced sub-partitions will also be connected. Stated another way, each processor will be working on a single local sub-piece of the grid rather than on a few cells here and a few more from someplace distant in the grid. There is also theory that shows that the partitioning is close to optimal as far as the communication requirements to. It also is guaranteed to produce almost the same number of cells for each processor to process.

Communication Strategy

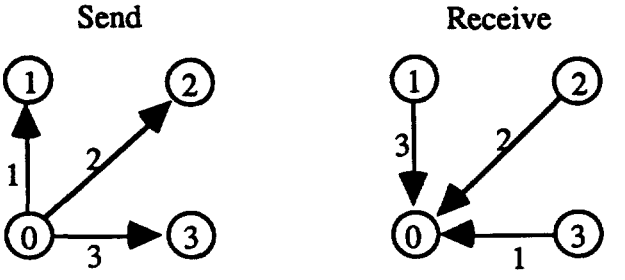
Once each processor on the parallel computer has its sub-piece of the global grid defined, the processor will be responsible for computing and/or updating all of the edge-based variables which are associated with its piece of the grid. We note that there will be some duplication due to the fact that faces and edges which exist on a partition boundary will necessarily be shared with the other processor(s) which "owns" the cell(s) on the other side of the partition boundary. In order for a processor to be able to update all of its variables it will need have access to all of the other cells which share nodes on the partition boundary. The next figure shows a 2D grid with a partition boundary depicted as the heavy solid line.



Grid with processor partition depicted with heavy solid line

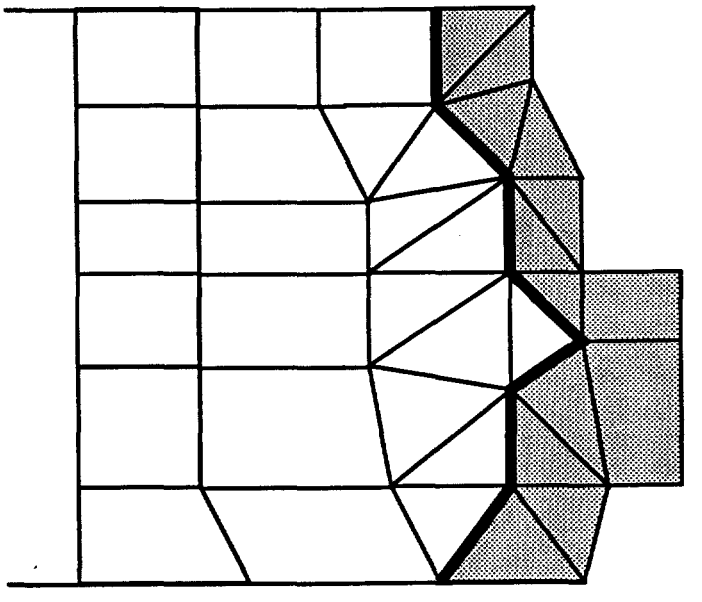
As each processor does not know in the beginning what cells or processors it may be surrounded by, each processor must discover these relationships by communicating with the other processors. The first step in this process is that each processor identifies all of its own boundary nodes and elements. These nodes may be nodes which lie on the real physical problem boundary or the partition boundary or both. This boundary-element data is then successively sent to every other processor. At the same time as a processor sends its boundary data it also receives the boundary data from another processor. A processor then compares its own boundary data with that obtained from another processor looking for matches between the boundary node sets. When a match occurs, then data is stored indicating which elements need to be sent to the other processor and also which elements

need to be received from the other processor. This process is repeated n-1 times (where n is the number of processors being used) as if the processors were connected in a "ring-like" manner. After completion of the boundary identification stage, each processor then actually sends all of its elements to the individual processors who need to have them. Again at the same time, a processor is receiving from the other processors all of the elements it requires.



Ringlike send / receive communication order for processor 0 to discover neighboring cells belonging

At the conclusion of this boundary element exchange stage, each processor will now have all of its original elements plus a layer of new elements which share its grid partition boundaries. At this point each processor has all of the geometry data which will be required to perform time step updates of all of its original edge and dual edge values. As a final step in setting up the communication tables which will be required in the time stepping update process, each processor must create a final communication table which lists all of the actual edge and dual edge field values it must send/receive to/from other processors. This is readily generated from the previously generated boundary element exchange data.



Extra cells (shaded) obtained by processor 0 from neighboring processors.

There is a somewhat subtle issue relating to obtaining the same results each time the code is run for the same problem which must be addressed. Because more than one processor will update edge values which lie on the grid partition boundaries, slightly different values for the same edge may be generated by the different processors sharing the edge. This possibility arises because even though they have identical geometry information (node coordinates and elements definitions), they may use a different ordering of the arithmetic operations to compute a new edge value. Round-off errors will then cause slightly different values to be produced. When these different values are sent to other processors through the message passing communication system, it is usually not feasible to control the exact order in which messages arrive at another processor. As a result of this asynchronous behavior, the same code can produce different answers each time it is run even for the same problem.

This potential randomness can be overcome by developing a unique "ownership" strategy for all of the problem variables. We do this in a simple but effective way by decreeing that an edge value is "owned" exclusively by the lowest numbered processor which shares the edge. This processor's value will then be the only value that will be used by any other processor. This strategy insures that with a constant partition of the grid and a constant number of processor being utilized for the solution process, identical results will be obtained each time the problem is run. Changing the grid partitioning or the number of processors being used may still give slightly different results.

Vectorization

When using nonorthogonal unstructured grids, a heavy penalty in computational efficiency can occur if sufficient care is not taken in structuring the computations and data. This will occur primarily because the inner-most loops involve a great deal of indirect addressing and the data may be accessed in rather random patterns. The basic DSI3D time stepping loops consist of dot products of vector coefficients with a vector of solution variables. These dot products can and will vary in length and involve indirect addressing. These conditions usually defeat most vectorizing compilers.

We have found that most of this difficulty can be easily overcome by reordering and regrouping the data so that much of it lies consecutively in memory. We will rearrange the data so that all of the data for dot products of the same length lie consecutively in memory. The next diagram shows the original inner loop for updating the electric field edge projected values. Next to it is shown the new inner loop restructured to gain more efficiency.

The short, variable length inner loops that perform the dot products limit performance on vector computers

```
do i = 1, hLen
   hdot = 0.
   do j = ipth(i), ipth(i+1) - 1
      hdot = hdot + coefh(j)
                  *eds(npth(j))
   enddo
   hds(i) = hds(i) - dt * hdot
enddo
```

Sorting the variable update order by loop length allows do-loop orders to be exchanged, leaving very long inner loops

```
do L = 1, maxLLen
   i1 = firstHolLen(L)
   i2 = firstHofLen(L+1) - 1
   j1 = firstHCofLen(L) - i1
   j2 = firstHCofLen(L+1) - i1 - 1
   do j = j1, j2, HLengthCount(L)
      do i = i1, i2
         hds(i) = hds(i) + coefh(i+j)
                         * eds(npth(i+j))
      enddo
   enddo
enddo
```

The performance improvement using this reordering and rearranging technique is dramatic. On a Cray C-90 single cpu, the performance on the original loop was about 15-20 megaflops. The rearranged data loop now performs at about 307 megaflops. On the Meiko CS-2 parallel processing computer the performance improves from about 5 megaflops to about 35 megaflops on each processor. We have even observed that the data restructuring also improves (to a lesser extent) the performance on scalar CPUs for various other machines.

The following table shows the basic computational speed(mflops) achieved on a variety of platforms.

Machine	Short Inner Loops	Long Inner Loops
SGI-R4000	3.3	4.3
Meiko Scalar	2.6	4.0
Meiko Vector	2.7	35.0
Paragon	2.6	4.3
Cray T3D	5.4	6.7
Cray C90	20.0	307.0

Parallel Scaling

There are differing approaches as to how best to determine how efficient a particular code or algorithm performs on a parallel computer. For single cpu computers, the situation is reasonably straightforward - you can simply measure the length of time it takes to solve the problem. Of course you must take into account things such as how much memory is required, how the local cache is being utilized, how much input/output is required, etc.

For parallel computers all of the above mentioned items are issues and in addition one must assess how much time penalty accrues as a result of the communication that must occur between processors. Life is simplest if all processors are doing exactly the same thing and they have exactly the same amount of work to perform. In reality this is almost never the case, different materials or different boundary conditions in different parts of the problem

are likely to cause some processors to have more computations to perform than others. Some processors may have more communications to perform than others.

For electromagnetic problems posed in the time domain, communication between processors must occur at least once and often twice per time step taken. RBC boundary condition computational demands are notoriously much greater than those for closed region problems involving only PEC boundaries. Significant load imbalances can easily occur. In general, the most dominant effects for efficiency many problems result from the initial mesh partitioning. This partition determines the relative amount of data that must be communicated versus the amount of typical floating point computation that must be performed by the processor. Geometrically speaking, the partitioned sub-domains should have a very low surface to volume ratio - the surface being proportional to the amount of communication that must take place and the volume proportional to the amount of floating point computation.

For a fixed size problem, trying to solve it using more and more processors inevitably leads to larger surface to volume ratios for the sub-domains - and hence - inefficiency. In the scaling results (speedup vs. processors) that are presented below, this is clearly demonstrated. The goal in solving any particular problem should be to determine an optimal number of processors to use to solve the problem. By performing scaling studies of this type one can determine what a reasonable number of processors to use may be. This will be determined by the user by his specifying an efficiency level that is acceptable for his problem.

The problem used for determining the speedup was a twisted waveguide problem with a grid structure similar to the grid pictures above. Three different problem sizes were solved with increasing numbers of processors. As is clearly evident, the larger problem is solved with greater efficiency than the smaller problems.

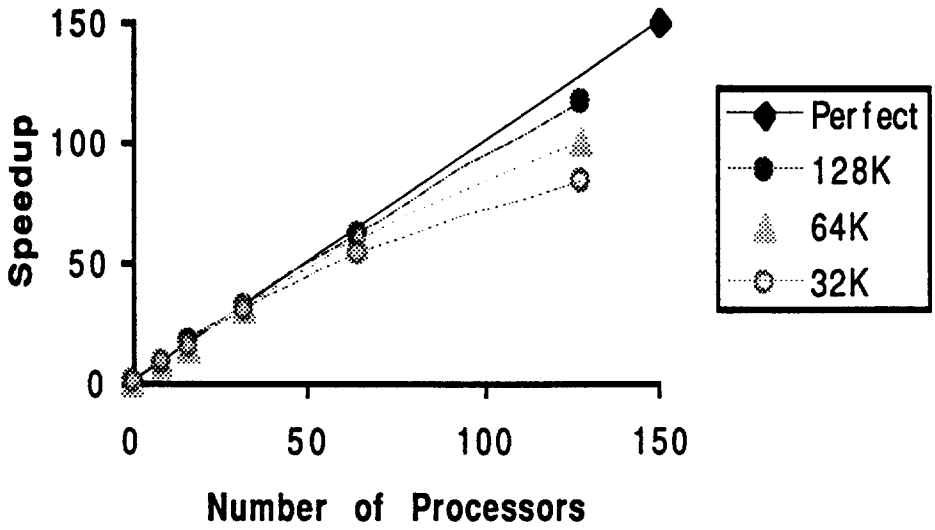


Figure: Plot of speedup versus number of processors

In general we know that having more zones per problem generally results in better efficiencies. Another way to assess how many processors to use to solve a problem is to consider the data (for the previously described problems) presented in the scatter-graph below which shows the "cost" per solution unknown versus the number of zones per processor.

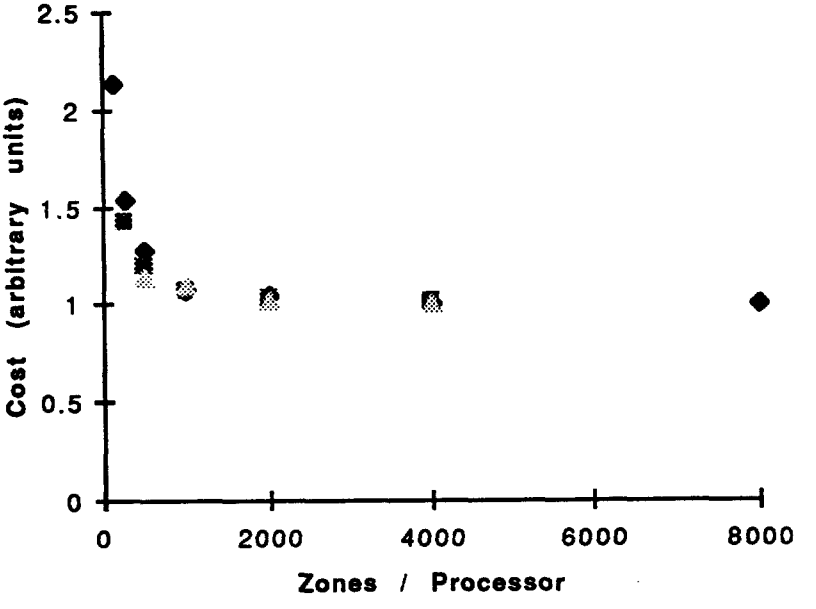


Figure showing the cost/unknown versus the zones per processor.

Clearly, problem setups involving larger numbers of unknowns per processor are more efficient. The differences between these problems are not as great as can frequently occur for other types of problems (particularly those involving RBCs). For a broad range of time domain electromagnetics problems we have found that having a minimum of several thousand grid cells per processor will result in acceptable performance for the computers that we have experienced.

Post-Processing

Post-processing for DSI3D is somewhat different than for many other codes. This is in part due to the fact that full electric and magnetic field vectors are almost never computed in the code as a part of the basic solution algorithm. Rather, projected field values onto primary grid and dual grid edges are the basic quantities computed and used in DSI3D. Also, since DSI3D has been designed as a parallel computer code, the data does not reside all in one place in that it is distributed across many processors. Another fact is that there is very little in the way of parallel graphics software available on almost any of the parallel computers. Work is in progress in this area and hopefully parallel graphics capabilities will exist and allow increased speeds and capabilities for data display.

The philosophy that we have adopted is that we will do all graphics post-processing not on the parallel computer but rather on workstations. This has several important implications. First, since the performance and capacity on most workstations is rather limited as compared to the parallel computer, it may not be possible to visualize all of the data together in one place at one time. Second, we will be required to develop software which will run on the parallel computer which will form and extract the data we desire to visualize.

We have developed a parallel post-processing tool called DSI3DIO which performs the required tasks. When DSI3D is run on a parallel computer, it will produce restart dump files which contain the edge and dual edge data. These files will exist on the separate processors and/or their local disk space. When DSI3DIO runs, it will read the original grid files and extract sup-pieces of the original grid as specified by the user. These sub-pieces may be planar cuts through the grid or particular material types. For the grid sub-pieces, DSI3DIO then uses the data in the restart files to build or interpolate full vector field data which will exist at the nodes of the grid sub-pieces. The code will then produce a reduced ascii grid file (which contains only the boundary cells of the region specified by the user) and ascii files containing the vector field data for this reduced grid. Currently, one can form electric field vectors, magnetic field vectors and Poynting vector field data. This data which will be much smaller than the entire problem data is then moved to a workstation and can be visualized with many different graphics software packages.

References

1. N.K. Madsen, "Divergence Preserving Discrete Surface Integral Methods for Maxwell's Curl Equations Using Non-orthogonal Unstructured Grids", *J. Comp. Phys.*, **119**, pp. 34-45, 1995.

2. K.S. Yee, "Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media,"*IEEE Trans. Antennas Propag.* **AP-14**(3), pp. 302-307, (1966).

3. R.L. Higdon, "Numerical Absorbing Boundary Conditions for the Wave Equations", *Math. of Comput.*, Vol. 49, pp. 65-91, July 1987.

4. Z. Liao, H.L. Wong, B. Yang and Y. Yuan, "A Transmitting Boundary for Transient Wave Analysis", *Scientia Sinica (Series A)*, Vol. 27, pp. 1063-1076, 1984.

5. G. Mur, "Absorbing Boundary Conditions for the Finite-Difference Approximation of the Time-Domain Electromagnetic Field Equations", *IEEE Trans. Electromagn. Compat.*, Vol. 23, pp. 377-382, 1981.

6. D. Steich, "Local Outer Radiating Boundary Conditions for the Finite-Difference Time-Domain Method Applied to Maxwell's Equations," Ph.D. dissertation, The Pennsylvania State University, May 1995.

7. G. Karypis and G. Kumar, "Parallel multilevel k-way partitioning scheme for irregular graphs", Technical Report TR 96-036, Department of Computer Science, U. of Minn., 1996.

8. H. Simon, "Partitioning of Unstructured Problems for Parallel Processing", *Computing Systems in Engineering*, Vol. 2, No. 2/3, pp. 135-148, 1991.

Technical Information Department • **Lawrence Livermore National Laboratory**
University of California • Livermore, California 94551