The Finite Element Solution of Elliptical Systems on a Data Parallel Computer

Scott Hutchinson
Edward Hensel
Steven Castillo
Kim Dalton

CRPC-TR90084
August, 1990

Center for Research on Parallel Computation
Rice University
P.O. Box 1892
Houston, TX 77251-1892

THE FINITE ELEMENT SOLUTION OF ELLIPTICAL SYSTEMS ON A DATA PARALLEL COMPUTER

Scott Hutchinson[1]
Edward Hensel[2]
Steven Castillo[1]
Kim Dalton[1]
[1]Department of Electrical and Computer Engineering,
[2]Department of Mechanical Engineering,
New Mexico State University, Las Cruces, New Mexico 88003
USA

August 14, 1990

Abstract

A study is conducted of the finite element solution of elliptic partial differential equations on a data parallel computer. A nodal assembly technique is introduced which maps a single node to a single processor. The system of equations is first assembled and then solved in parallel using a conjugate gradient algorithm for to unsymmetric, non-positive definite systems. Using this technique and a massively parallel machine, problems in excess of 100k nodes are solved.

Results of electromagnetic scattering, governed by the 2-d scalar Helmholtz equation, are presented for both an infinite cylinder and an airfoil cross-section. Solutions are demonstrated for a wide range of object sizes. A summary of performance data is given for a set of test problems.

1 Introduction

The finite element method is a versatile and general technique for solving partial differential equations in geometrically complex domains. In spite of this versatility, certain domains often present too great a computational task for conventional computers. There will probably always be some problem of interest whose solution is prevented by its shear size and the limitations of available computers. This is one reason for the ever increasing need of greater computational power, both on the leading edge of computer technology and on a per cost basis. In the past 10 years, the use of parallel computers (computers which use multiple processors) has become a cost effective method for increasing computational resources.

This paper looks at the capabilities of a data-parallel supercomputer as applied to the finite element solution of an elliptic partial differential equation – the 2-d scaler wave equation. The particular computer used is Thinking Machines Corporation's Connection Machine 2 (CM-2). The CM-2 employs up to 64k (k = 1024) *simple* processors, each with 64k bits of memory. While the CM-2 has been in existance for nearly 4 years, finite element algorithms for the CM-2 have been few. One reason is that, as with all parallel architectures, techniques which may be mature on serial computers must be altered and sometimes discarded in favor of more applicable algorithms. This paper will show that this is especially true of data parallel architectures if one wishes to approach both peak performance *and* peak efficiency for finite element problems.

Work in finite elements on the Connection Machine has been done by Johnsson and Mathur [9], Hutchinson, et al. [8], Cline, et al [2], Fonseca [3] and others. This paper takes a new approach to the generation of the finite element equations while solution of the equations is done using a parallelized version of the conjugate gradient methods described by Hestenes and Stiefel [5] and Peterson [10].

In what follows, data parallel programming is introduced. Next, the governing

equation is given along with its finite element form. The implementation of the finite element equations using a nodal assembly technique as well as their solution on a data parallel computer is then described. Results and conclusions follow with future research interests.

2 Data Parallel Programming

While developed on the CM-2, the techniques presented here may be useful on a variety of data parallel or SIMD (Single Instruction Multiple Data) computers. A brief introduction to some aspects of data parallel programming will help in understanding the techniques described in the following sections. In data parallel programming, two concepts should be considered when developing algorithms for a data parallel architecture.

First, in a data parallel architecture, one data element is ideally associated with one processor. Computationally, each instruction is performed on each member of a data set concurrently. This is in contrast to coarser-grained parallel computers which typically divide computational domains into large segments and/or distribute tasks among the processors. Because of a data parallel computer's fine-grain nature, the selection of an elementary data item is critical. Subdividing a problem for data-parallel implementation should be done in a manner which produces high *efficiency*. An efficient algorithm is one which delegates the subtasks or subdomains such that at all times, the full power of the machine is brought to bear on the problem, *i.e.*, the idle time of any processor is brought to a minimum. Choosing a data item for which the complete processor set does not participate in each operation can be detrimental to a program's efficiency [8].

Second, parallel computers depend on attacking a problem with multiple processors to gain speed over their serial counterparts. They do, however, possess a time consuming aspect which is absent in serial computers – interprocessor communication. Keeping communication to a minimum and, where necessary, doing it as efficiently as possible, will allow the architecture to realize its fullest capabilities. In minimizing communication, the data item should be chosen such that separate processes are as independent as possible. Dependency usually requires information and, thus, communication. On an architecture whose processors are connected in

a hypercube arrangement, one method of speeding interprocessor communication is to perform it only with a given processor's nearest neighbor. Nearest neighbor communication is faster than random interprocessor communication for two reasons. First, the address of the processor with which communication is taking place need not be calculated or checked and second, there is no possibility of message collision. The data item and associated data structure should lend itself to minimal and efficient communication. Much of what is developed in the following sections was based on these two basic ideas. Data structure and communication are the keys to gaining performance in this type of algorithm development and the two are not related.

3 Problem Formulation

The finite element method is a numerical technique used to obtain an approximation to some boundary value problem

$$L\psi = f \tag{1}$$

over some domain Ω with associated boundary conditions. In finite elements, the domain is discretized and represented by the linear system

$$\mathbf{K}\psi = \mathbf{b} \tag{2}$$

for the unknown, ψ [7].

The Helmholtz wave equation governs the properties of time harmonic wave propagation [4]. It is derived from Maxwell's equations on the interaction of electric and magnetic fields. The 2-d scalar Helmholtz equation takes the form

$$\nabla^2\psi + k^2\psi = 0 \tag{3}$$

Boundary conditions on the problem are usually written as

$$\psi = C_d \text{ on } \Gamma_{inner} \tag{4}$$

$$\frac{\partial\psi}{\partial n} = C_n \text{ on } \Gamma_{outer} \tag{5}$$

Solving for the scattered fields produced by a Transverse Magnetic (TM) polarized (the magnetic field is transverse to the direction of propagation) incident plane-wave and considering a non-homogeneous media, the wave equation is written

$$\nabla\cdot\left(\frac{1}{\mu_r}\nabla E^s\right) + k_0^2\epsilon_r E^s = -\nabla\cdot\left(\frac{1}{\mu_r}\nabla E^i\right) - k_0^2\epsilon_r E^i \tag{6}$$

where the scattered electric field (E^s) has replaced the ψ of Equation (3). Here, E^i represents the incident plane wave as in figure 4 and k_0 is the free-space wave number and is defined as

$$k_0 = \frac{2\pi}{\lambda_0} \tag{7}$$

where λ_0 represents the free-space wavelength. Also, μ_r and ϵ_r are known as the relative permeability and the relative permittivity, respectively, and may be functions of position

$$\mu_r = \mu_r(x, y)$$

$$\epsilon_r = \epsilon_r(x, y)$$

Although not of primary importance, Equation (6) is the governing equation explored in this paper. The development presented here is applicable to a wide class of elliptic partial differential equations.

On Γ_{outer}, the normal derivative of the scattered fields is approximated with a second order Bayliss-Turkel [1] absorbing boundary condition

$$\frac{\partial E^s}{\partial \hat{n}} = \alpha(\rho) E^s + \beta(\rho) \frac{\partial^2 E^s}{\partial \phi^2} \tag{8}$$

where ρ and ϕ are the familiar cylindrical coordinates and

$$\alpha(\rho) = \frac{\frac{-3}{2\rho} + \jmath\left(\frac{3}{8k\rho^2} - k\right)}{1 - \frac{\jmath}{k\rho}} \tag{9}$$

$$\beta(\rho) = \frac{\frac{-\jmath}{2k\rho^2}}{1 - \frac{\jmath}{k\rho}} \tag{10}$$

for $\jmath = \sqrt{-1}$.

Γ_{inner} is taken to be a perfect electrical conductor, thus, the scattered fields are known on the object to be

$$E^s_{\Gamma_{inner}} = -E^i_{\Gamma_{inner}}. \tag{11}$$

Using these boundary conditions, the wave equation is discretized using a Galerkin method of weighted residuals technique, yielding a standard linear system

$$\sum_{e=1}^{N_e} (\mathbf{K}^{(e)}) \psi^s = \sum_{e=1}^{N_e} \mathbf{f}^{(e)}. \tag{12}$$

The summation here is used to represent an *assembly* of the elemental quantities. This is the standard method employed in most serial finite element programs. As

the next section shows, this assembly is not explicitly carried out in the technique presented here, however it is equivalent and helps in the development which follows. Also, ψ^s represents a vector of scattered field values at the discrete nodal points.

4 Data Parallel Implementation

As with all finite element programs, the algorithm may be divided into the following parts [9]:

- discretization of the problem domain into a mesh
- representation of the discretized problem domain as a computational domain
- generation of the system of equations representing interactions between the discrete points in the computational domain
- solution of the system of equations to produce an approximation to the exact solution at the discrete points in the computational domain

Thus, for each of the above portions, the parallel considerations listed in Section 2 must be taken into account. Using this method, what is termed a *nodal basis* technique has been developed for usage on data parallel architectures. The remainder of this section is a description of how the portions itemized above were implemented in a data parallel fashion on the CM-2 using nodal mapping.

4.1 Discretization

To make use of fast nearest neighbor communication, it is necessary to use a *regular* discretization of the physical domain. To facilitate the regular mesh, the elements used here are first order two-dimensional quadrilaterals. When a physical domain representing two boundaries is discretized using these elements, it may be "wrapped" into a 2-d torus or "O-grid" as in Figures 1 and 2.

For this algorithm, a mesh is generated in a conventional preprocessor and read from disk by the controlling program. Each processor is given its required geometrical information from the front end computer.

4.2 Representation

Subdivision of the computational domain into a finite number of relatively independent domains for parallelization is a significant consideration in any parallel implementation. On a data parallel computer, all active processors must execute the same function on the data set. The problem of choosing the individual data item has received some attention in the literature [9], [3], [8]. The choice of one processor per element has been explored by the authors [8]. However, due in part to the fact that the concept of an element does not naturally exist in the assembled system of equations, this choice presents some problems in terms of parallel efficiency. Johnsson and Mathur [9] also discuss this choice of elementary object but actually implement a scheme based on a processor per node per element. The Johnsson and Mathur method has distinct load balancing advantages when using higher order elements.

As can be seen in Equation (12), Section 3, the system of equations is a nodal representation of the discrete behavior in the computational domain, *i.e.*, there is a single equation for each node in the domain. From this standpoint, it is logical to choose a data structure such that each processor is responsible for a single node. An immediate advantage is that this structure can be preserved throughout the program from discretization through solution. Further, in the case of the 2-d regular mesh, it allows for nearest neighbor or NEWS (North, East, West, South) grid communication throughout the program. A possible problem, however, exists for first order elements in that processors representing boundary nodes may be inactive during phases of the program.

4.3 Nodal Assembly

The mapping employed herein is one in which each processor is responsible for a single node. In terms of the system of equations, each processor both generates and stores a given equation which describes that node's interaction with its

surrounding nodes. This manner of generating the system of equations has been termed *nodal assembly*.

In describing nodal assembly, it is helpful to define a node's neighborhood or *nodal region*. In a regular mesh of quadrilaterals, the nodal region may be represented as in Figure 3. Here, node D has neighbors which may be designated as the $South West$ neighbor, the $South$ neighbor, the $South East$ neighbor, the $East$ neighbor, the $North East$ neighbor, the $North$ neighbor, the $North West$ neighbor and the $West$ neighbor. Note that four of these nodes correspond to the four directions in regular communication grid ($North$, $East$, $West$ and $South$). For an O-grid, there are always four elements surrounding node D, the $South West$ element, the $South East$ element, the $North East$ element and the $North West$ element.

Standard finite element formulations assemble each elemental set of equations to produce the global system of equations. During this process, each row (nodal equation) in the global system receives contributions from its elements, *i.e.*, those elements of which it is a node. For the mesh described above, each row in the global system would have nine non-zero entries, one for each of the eight neighboring nodes and a diagonal value. In nodal assembly, each processor executes instructions which compute the coefficients of each non-zero entry in its row.

An example will illustrate the nodal assembly procedure. Each processor will compute the following nine coefficients

$$SW, S, SE, E, NE, N, NW, W, D$$

which correspond to the left-hand side of the global equation for node D. Each processor also computes the right-hand side value for its equation.

Referring to Figure 3, an elemental stiffness matrix is represented as

$$\mathbf{K}^{(e)} = \begin{bmatrix} K^{(e)}_{1,1} & K^{(e)}_{1,2} & K^{(e)}_{1,3} & K^{(e)}_{1,4} \\ K^{(e)}_{2,1} & K^{(e)}_{2,2} & K^{(e)}_{3,3} & K^{(e)}_{2,4} \\ K^{(e)}_{3,1} & K^{(e)}_{3,2} & K^{(e)}_{3,3} & K^{(e)}_{3,4} \\ K^{(e)}_{4,1} & K^{(e)}_{4,2} & K^{(e)}_{3,3} & K^{(e)}_{4,4} \end{bmatrix} \tag{13}$$

for each of the 4 elements adjacent to node D. For node D, the coefficients above may be computed from the entries in the neighboring elemental stiffness matrices:

$$\begin{aligned} SW &= SW^{(e)}_{3,1} \\ S &= SW^{(e)}_{3,2} + SE^{(e)}_{4,1} \\ SE &= SE^{(e)}_{4,2} \\ E &= SE^{(e)}_{4,3} + NE^{(e)}_{1,2} \\ NE &= NE^{(e)}_{1,3} \\ N &= NE^{(e)}_{1,4} + NW^{(e)}_{2,3} \\ NW &= NW^{(e)}_{2,4} \\ W &= NW^{(e)}_{2,1} + SW^{(e)}_{3,4} \\ D &= SW^{(e)}_{3,3} + SE^{(e)}_{4,4} + NE^{(e)}_{1,1} + NW^{(e)}_{2,2}. \end{aligned} \tag{14}$$

Here, the notation $SE^{(e)}_{4,3}$ refers to the coefficient in row 4, column 3 of the SE element's stiffness matrix.

Thus, once an expression for the elemental stiffness matrices is developed, the equations above may be used to generate row D in the global system of equations. All processors concurrently generate their own equations. Since the elemental stiffness matrices are dependent on the nodal locations of that given element, the implementation described here requires that at some point in the computation each processor obtain the location (x, y) of each of its 8 neighbors in the nodal

region. In this implementation, reading the mesh simply gives each processor its own location as well as its boundary information. To determine its neighbors' locations, 16 nearest neighbor communications are performed (8 x-locations and 8 y-locations).

For boundary nodes, the appropriate computations can be made on those processors. In this portion of the program, the remaining processors are idle. If the required boundary condition computations are laborious, this can affect the parallel efficiency of the program.

4.4 System Solution

A frequently chosen method for solving a sparse system of equations on a data parallel computer is the conjugate gradient iterative technique. This type of solution algorithm has been used in nearly every algorithm which requires the solution of some sparse system of equations on the CM-2. The main reason for this is that the method is mainly a collection of matrix and vector operations – operations for which a data parallel computer is well suited.

The basic conjugate gradient algorithm was given by Hestenes and Stiefel [5] for use on symmetric, positive definite systems. Hestenes and Stiefel also presented a method for any non-singular system of equations. The second technique is chosen for the solution of the wave equation presented here. It is given as

Initialize:

$$\mathbf{r}_0 = \mathbf{b} - \mathbf{K}\psi_0^s \tag{15}$$

$$\mathbf{p}_0 = \mathbf{K}^T\mathbf{r}_0 \tag{16}$$

Iterate:

$$a_i = \frac{|\,\mathbf{K}^T\mathbf{r}_i\,|^2}{|\,\mathbf{K}\mathbf{p}_i\,|^2} \tag{17}$$

$$\psi_{i+1}^s = \psi_i^s + a_i\mathbf{p}_i \tag{18}$$

$$\mathbf{r}_{i+1} = \mathbf{r}_i - a_i \mathbf{K}\mathbf{p}_i \tag{19}$$

$$b_i = \frac{|\,\mathbf{K}^T\mathbf{r}_{i+1}\,|^2}{|\,\mathbf{K}^T\mathbf{r}_i\,|^2} \tag{20}$$

$$\mathbf{p}_{i+1} = \mathbf{K}^T\mathbf{r}_{i+1} + b_i\mathbf{p}_i \tag{21}$$

where the choice of ψ_0^s is arbitrary. Note that the system of equations represented above by $\mathbf{K}$ is symmetric for the problem considered here and so $\mathbf{K}^T$ need not be computed.

The nodal assembly and naming convention presented for the generation of interaction coefficients also lends itself well to the solution phase. During this portion of the program, each processor stores the interaction coefficients and forcing coefficient associated with its represented node. In addition, it now stores its corresponding entry in each of the vectors given in the above equations. This scheme allows for an efficient method for performing the matrix and vector operations given in the solution algorithm, (15)-(21).

In Equations (15)-(21) there are 4 basic matrix and vector operations – 1) vector addition/subtraction, 2) scalar-vector multiplication, 3) vector dot products and 4) matrix-vector products. Each of these operations is carried out in parallel as described below.

- In performing the vector addition/subtraction, each processor, having an entry in the appropriate vectors, computes its result. No communication is required.
- The scalar-vector multiplications are performed by broadcasting to each processor a copy of the front-end serial variable and multiplying within the processor. A one-to-all communication is required for each processor to obtain a copy of the serial variable.
- Computing the vector-dot products is done by first multiplying each vector entry within its processor. A global reduction then sums each processor's

result to a front-end serial variable. All-to-one communication is required to perform this global reduction.

- The matrix-vector products are the most complicated operation performed in the solution algorithm. It first multiplies all the non-zero coefficients stored on each processor with the corresponding vector entries. Since each processor has only its own vector entry, 16 (8 real and 8 imaginary) nearest neighbor communications are necessary to obtain the other 8 vector entries corresponding to the neighbors of node D. Once done, each processor sums the results to its entry in the resultant vector.

Here, there are 3 vector additions/subtractions, 3 scaler-vector multiplications, 3 vector dot products and 2 matrix-vector multiplications per iteration. Since the communication operations described above are few and, on the CM-2, are optimized, the algorithm is highly parallel.

Note that, where required (matrix–vector products), the same interprocessor communication pattern is used as was for the nodal assembly procedure. In addition, the proper data items required by the solution technique (i.e. interaction coefficients and forcing values) are already in place on the processors. No further assembly or communication is needed. These facts are not peculiar to the conjugate gradient method, but are applicable to most iterative solution strategies.

5 Results

The finite element techniques presented above for the solution of the 2-d scalar wave equation have been implemented on the CM-2. The results presented in this section were obtained from a program written in C-PARIS. Double precision (64-bit) computations are now run using floating–point accelerators the CM-2 where available. On other machines, only 32-bit accelerators are available and so any 64-bit computations are done in software. All results here represent single-precision, hardware arithmetic. Timings were obtained using the CM-2 timing facility.

Results for two different geometries are examined – a cylinder and an airfoil. Both geometries are taken as perfectly conducting bodies infinite in the third dimension. In the case of the cylinder, an eigenfunction solution was obtained for program verification purposes with nearly indistinguishable results. In both cases, the magnitude and phase of the total field results are presented.

The meshes in Figures 1 and 2 are much coarser than those used to obtain results. Typical meshes used 32 nodes in the radial direction and up to 2048 nodes in the circumferential direction. The number of nodes in each of the respective directions was chosen to give in excess of 10 nodes per wavelength throughout the problem domain.

There are many ways to evaluate both the utility and the performance of any program. Further, depending on one's viewpoint, the performance may take precedence over the utility or vice-versa. From the engineering point of view, increased performance over existing methods is required but must be coupled with a degree of utility. While both these terms are subject to interpretation, the remainder of this section will endeavor to provide results which demonstrate both qualities.

Table 1 gives geometry and mesh information for the cases presented. The virtual processor ratio gives the number of virtual processors that are assigned to

1 physical processor (see Section 2). The cylindrical case is a small (3-wavelength radius) cylinder wrapped in a dielectric ($\epsilon_r = 5.0$) coating which extends 0.5λ out from the inner conducting cylinder.

The airfoil problem is considered using NACA airfoil number 0010. The incident wave was taken as striking the leading edge of the airfoil at normal incidence. Figures 7 and 8 show the phase and magnitude of the total scattered fields from the airfoil.

The phase plots illustrate that lines of constant phase approach the perfect conducting inner boundary at normal incidence. This follows from the boundary condition that

$$E_{tan} = 0$$

on the boundary. The magnitude plots illustrate the total field for an incident plane wave traveling in the x-direction. The total field becomes zero on the boundary and displays a shadow region behind the conducting body.

Table 2 shows timing data for each case. Total program time is just that and includes the reading of the mesh data and writing of results. For large problems, this is a significant portion of the total program time. This is illustrated by comparison with the fill and solve times. Future use of "in-place" mesh generation on the processors will significantly reduce this time. The fill time represents the time necessary to generate the finite element system of equations. This does not include the calculation of the boundary conditions. The solve time represents the solution time of the conjugate-gradient algorithm. It includes a single complex matrix-vector multiply in the initialization phase of the algorithm as well as two complex matrix-vector multiplies per iteration during the iterative phase. In the cylindrical case, iterations were terminated based on a normalized residual of

$$\frac{|\,\mathbf{r}_i\,|}{|\,\mathbf{b}\,|} < 10^{-4} \tag{22}$$

The cylindrical geometry results in a relatively well conditioned stiffness matrix

with solutions that converge to the criteria above in much fewer iterations than the number of equations.

Convergence for the airfoil problem was extremely slow. This can be partially attributed to the fact that some of the quadrilateral elements in the mesh had large internal angles leading to poor conditioning of the system. This occurs at both the leading and trailing edges of the airfoil as seen in Figure 2. As these angles approach 180 degrees, the two defining edges tend towards a straight line. In the limit as the line becomes straight, the elemental equations become singular.

This problem was run for 131072 iterations (the number of equations in the system) before being terminated. The normalized residual after this number of iterations was

$$\frac{|\,\mathbf{r}_i\,|}{|\,\mathbf{b}\,|} < 1.35 \times 10^{-4} \tag{23}$$

yielding an acceptable result.

Table 3 gives MFlop (Million Floating point operations) performance characteristics of the program. Each floating point operation, whether a multiplication, division, addition or subtraction, is counted. Achieved as well as extrapolated ratings are given for various portions of the program. Extrapolated ratings are obtained by extrapolating the times on the given processor ratio to the same ratio on a full 64k CM-2. This extrapolation assumes a perfectly parallel algorithm while the solve portion actually contains some front-end communication which does not scale exactly. However, previous research [2] shows that this communication does not greatly effect the time per iteration and so the scaled results represent a very good approximation.

The largest MFlop rating achieved during the solve portion of the program used the highest virtual processor ratio. This occurs because, although the calculations in each virtual processor are slowed due to a sharing of the physical resources, interprocessor communication becomes a simple memory swap for virtual processors sharing a physical processor. This memory swap is faster than

nearest neighbor communication between physical processors and so communication time is reduced. In effect, doubling the virtual processors doubles the number of floating point operations while the execution time is somewhat less than double, resulting in more MFlops. Note here that this is not necessarily the optimum program execution time. Using a higher virtual processor ratio than necessary will cause an increase in execution time even though the MFlop rating will go up. For the fill portion of the program, the Mflop rating remains somewhat constant (depending on the front-end availability and other factors) as the virtual processor ratio is increased. This is because the fill portion contains no interprocessor communication and so doubling the number of floating point operations a given processor must execute doubles the execution time.

6 Conclusions and Future Research

Using the nodal assembly technique, a finite element program is implemented on a data parallel computer in a manner which allows the use of the same data structure throughout the program, from fill through solution. Discretization using this mapping is also possible. Nodal assembly mapping provides for a relatively efficient program.

More importantly, the computational advantages described above allow the solution of a larger class of problems. Specifically, the method permits the finite element solution of scattering problems with geometries much larger than previously possible. This is also true of other finite element problems such as those governed by Laplace's and Poisson's equations.

From the work presented here, conclusions may be drawn from several areas. First, with respect to the nodal assembly implementation on the CM-2:

- Nodal assembly allows the mapping of one finite element node onto one virtual processor. This mapping is maintained throughout the program.
- Using first order quadrilaterals and a regular mesh, the mapping may be configured in a NEWS grid, allowing nearest-neighbor communication.
- The mapping described above will permit a maximum virtual processor ratio of 16 under the current CM-2 memory limitations. On a 64k processor machine, this allows a maximum of 1048576 nodes.
- Nodal assembly is inefficient when handling boundary conditions. This is because only processors on a given boundary are active during this portion of a program.
- Nodal basis mapping is well suited for use with a conjugate-gradient iterative solution. All the matrix and vector operations can be computed with a

high level of concurrency. Nearest neighbor communication is again used in performing the matrix-vector products.

Second, considering the scattering-problem capability of the nodal assembly as implemented on the CM-2:

- The regular grid allows for the solution of a variety of shapes of interest in the physical world.
- Taking the outer boundary to be a radial distance of 2 wavelengths from the inner conductor and at least 10 nodes per wavelength, finite element solutions for cylinders on the order of 100 wavelengths are possible.

With respect to the CM-2, several things need be said. First, although all these examples were run on a machine with only a 32 bit floating point accelerator, 64 bit accelerators are now available to allow double–precision floating–point calculations in hardware. Thinking Machines estimates that double precision computations in hardware take approximately twice as long as do single precision hardare computations. On machines with only 32-bit floating point accelerators, 64-bit computations (software) and are about 20 times slower than the 64-bit hardware results[13].

Second, the individual processor memory has been increased from 64k bits to 256k bits on some machines. This would effectively allow the solution of problems with 4 times the number of nodes. The 64k bits of processor memory allowed for a maximum virtual processor ratio of 16 for 1,048,576 nodes. The new memory size would allow a virtual processor ratio of 64 for 4,194,304 nodes.

Further research into data parallel techniques and their use in the solution of scattering problems is on-going. Currently, a study of the limitations of the *ten nodes per wavelength* rule of thumb is taking place. Other areas which require investigation include an extension to 3-dimensional finite elements, non–uniform meshes which result from the discretization of multiply connected problem do-

mains, effectiveness of the absorbing boundary condition for both 2- and 3- dimensional problems and convergence properties of the conjugate gradient algorithm when applied to complex geometries. Also, with respect to mesh generation, several areas require investigation. These include parallel mesh generation, mesh refinement techniques as well as control over specific element geometries.

It is well known that the use of a pre–conditioner with a conjugate gradient solver can decrease the number of iterations required to achieve some required convergence criteria [14]. The algorithm used here does not employ any pre–conditioning as this work was mainly directed towards the development of a suitable mapping technique. Others [2], [9] have shown that a simple diagonal preconditioner may be used on a data parallel computer without to much difficulty. However, more research into the use of preconditioned conjugate gradients on data parallel computers is needed.

Work on an nodal assembly algorithm for non–uniform meshes is currently under progress. It is expected that this algorithm will lack some of the performance of the one presented here. This is due to the fact that the nodes can no longer be mapped onto the simple nearest neighbor communication grid and that, in the case of the CM-2, all interprocessor communication will require the slower router mechanism. Some idea of the comparison in solution times for conjugate gradient algorithms using the two interprocessor communication schemes can be obtained from [2] who shows a significant degredation in performance.

Parallel mesh generation would allow the each processor in a nodal–basis data structure to compute its own location as well as its nodal region properties. This means that the controlling program need only read in data which describes the overall domain geometry. The amount of data would be much less than that required by the current method which gives the location of each node. This results in less disk storage and less program time spent doing disk I/O.

Mesh refinement techniques allow the program to alter the node distribution in

the physical domain. This permits more nodes to be allocated in regions where the solution is expected to vary rapidly and fewer nodes in regions where the expected solution is relatively constant. Thus, a better approximation to the exact solution is obtained for a given number of nodes.

7 Acknowledgements

This work was supported by NSF grant # EET-8812958.

Computational resources were provided by Los Alamos National Laboratories, Los Alamos, New Mexico and by the Northeast Parallel Architectures Center (NPAC) at Syracuse University, which is funded by and operates under contract to, DARPA, and the Air Force Systems command, Rome Air Development Center (RADC), Griffiss AFB, NY, under contract number F30602-88-C-0031.

References

[1] A. Bayliss and E. Turkel, "Radiation boundary conditions for wave-like equations," *Communications on Pure and Applied Mathematics*, 33,707-725.

[2] R. E. Cline *et al.*, "Towards the development of engineering production codes for the Connection Machine," *Proceedings of the Fourth Annual Conference on Hypercubes, Concurrent Computers and Applications,* to be published, Monterey CA, 1989.

[3] A. Fonseca, "An adapted finite element method for massively parallel processors", *Proceedings of the Fourth Annual Conference on Hypercubes, Concurrent Computers and Applications,* to be published, Monterey CA, 1989.

[4] R. F. Harrington, *Time Harmonic Electromagnetic Fields,* McGraw-Hill: San Francisco, 1961.

[5] M. R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solution of linear systems," *J. Res. Nat. But. Standards,* vol. 49, pp.409-436, 1952.

[6] W. Daniel Hillis, *The Connection Machine,* MIT Press, Cambridge, MA, 1985.

[7] K. H. Huebner and E. A. Thornton, *The Finite Element Method for Engineers,* John Wiley & Sons, 1982.

[8] S. A. Hutchinson, S. P. Castillo and E. Hensel, "Solving 2-d problems on the Connection Machine using the finite element method," *Proceedings of the 5th Annual Review of Progress in Applied Computational Electromagnetics,* Monterey CA, 1989.

[9] S.L. Johnsson and K. Mathur, *Data Structures and Algorithms for the Finite Element Method on a Data Parallel Supercomputer,* Technical Report CS89-1, Thinking Machines Corp., 1988.

[10] A. F. Peterson, *On the Implementation and Performance of Iterative_Methods for Computational Electromagnetics,* Ph.D. Dissertation, University of Illinois, Urbana, IL, 1986.

[11] A. F. Peterson and S. P. Castillo, "Differential equation methods for electromagnetic scattering from inhomogeneous cylinders," *IEEE Transactions on Antennas Propagation,* vol. AP-37, pp.601-607, 1989.

[12] Thinking Machines Corp. *Connection Machine Model CM-2 Technical Summary,* Version 5.1, Thinking Machines Corp., 1989.

[13] Thinking Machines Corp. *Paris Reference Manual,* Version 5.2 Release Notes, Thinking Machines Corp., 1990.

[14] *Matrix Computations.* The John Hopkins University Press, 1985.

List of Figures

List of Tables

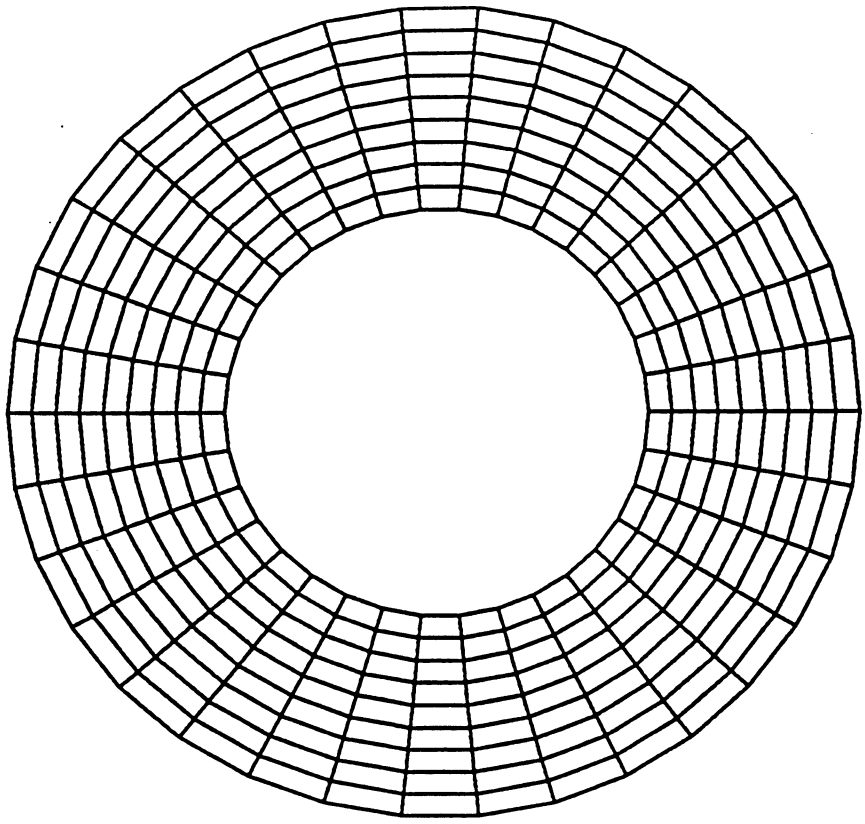

Figure 1: 2-d Torus Mesh.

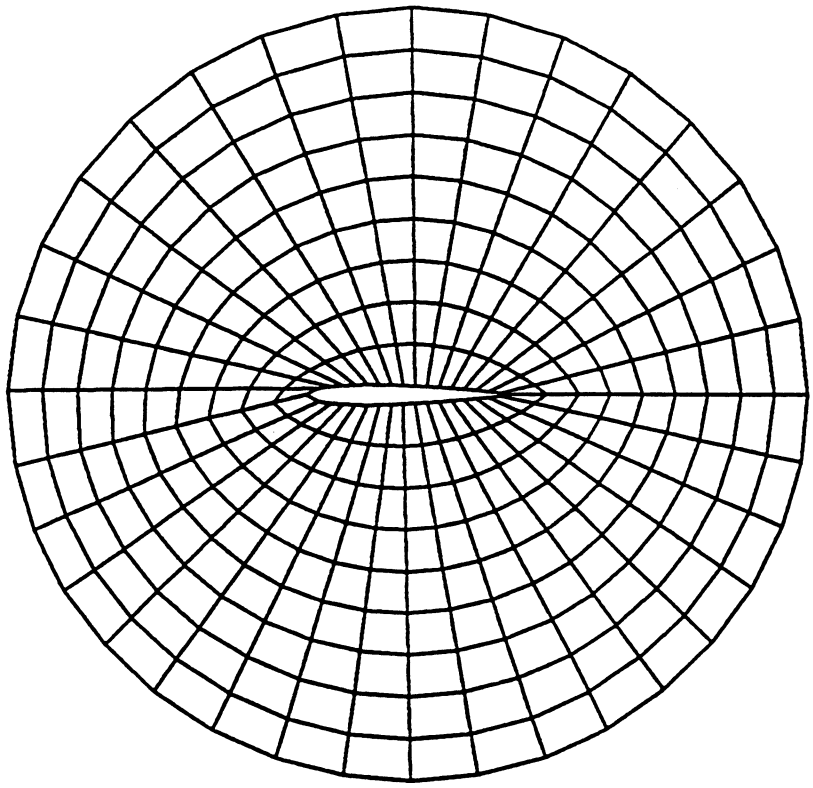

Figure 2: 2-d Airfoil Mesh.

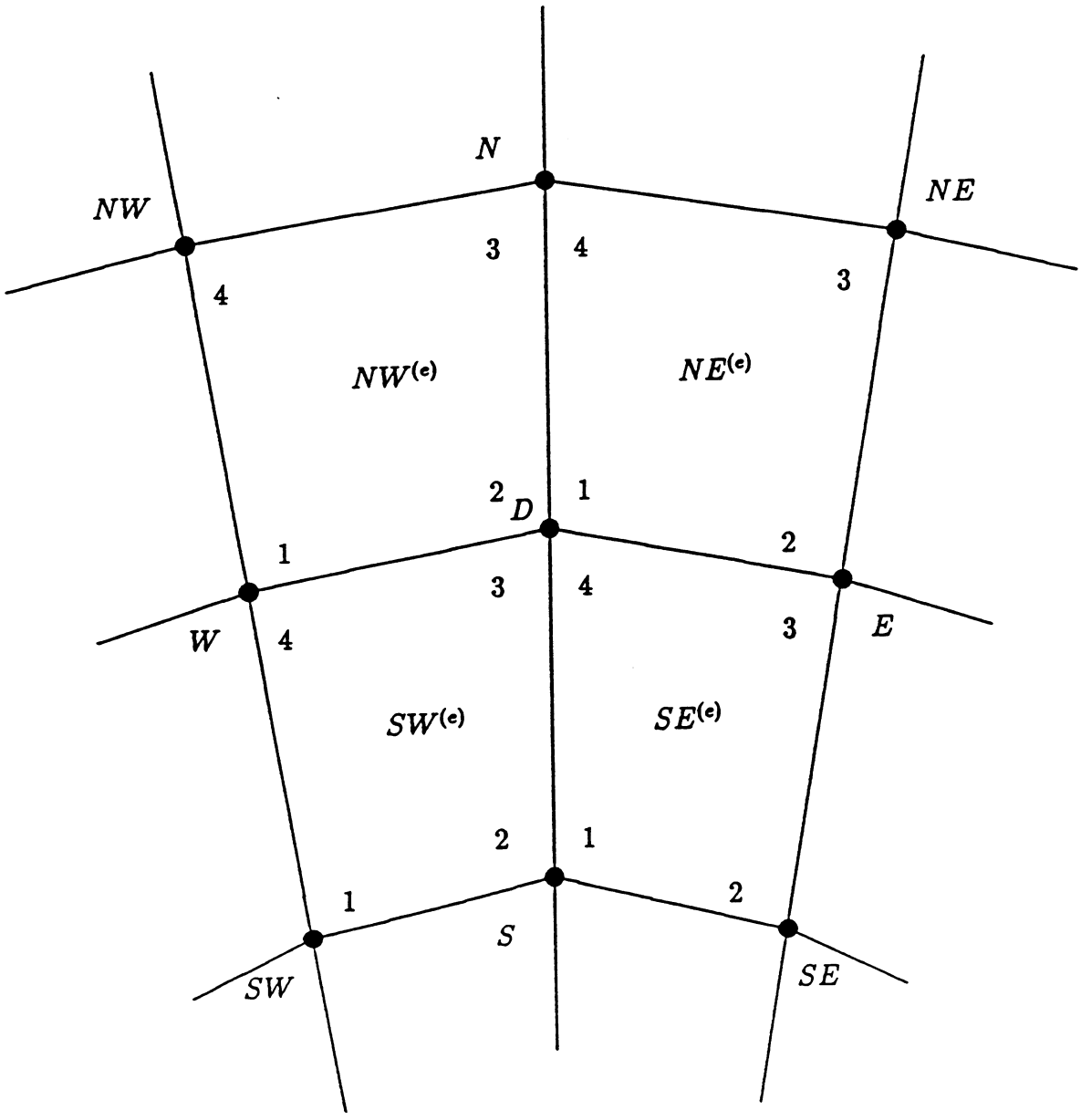

Figure 3: Nodal Region of node "D" indicating nearest neighbor nodes and adjacent elements

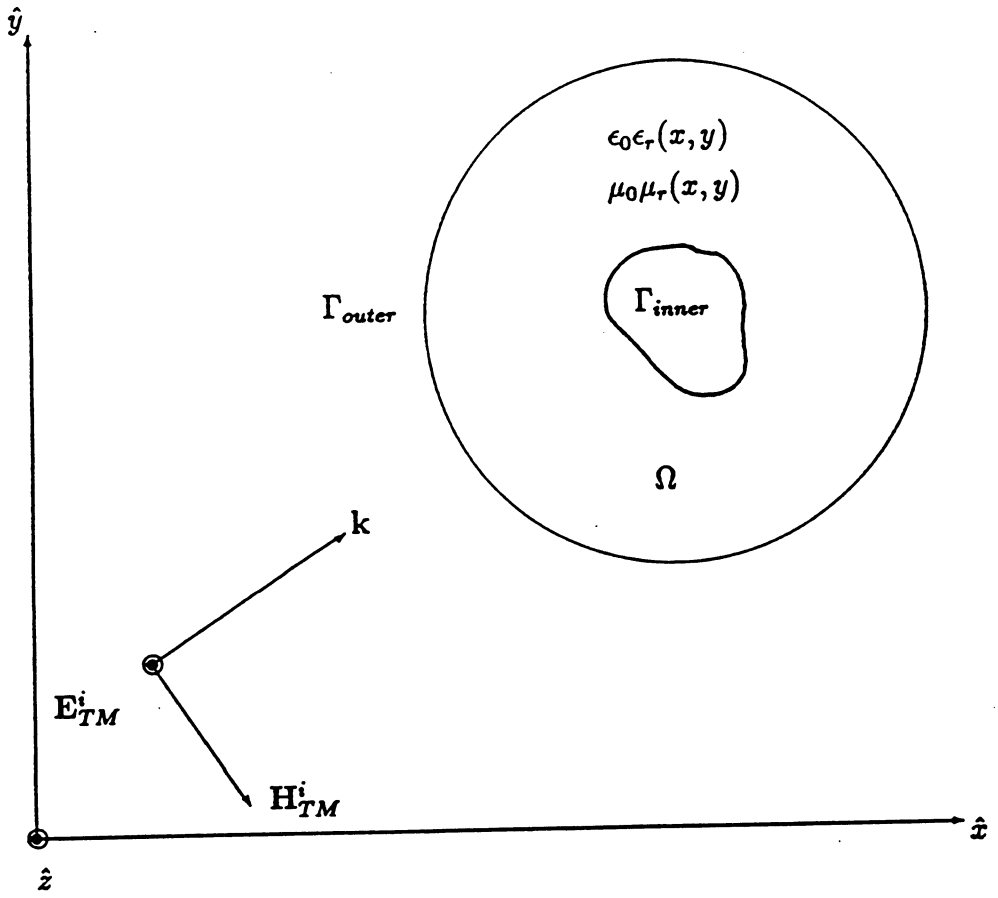

Figure 4: Open region scattering problem.

Table 1: Test cases

Case	Inner Radius	Outer Radius	Num. Nodes			V.P. Ratio
			Circum.	Radial	Total	
Cylinder	3 λ^*	5 λ	512	32	16384	2
NACA no. 0010 Airfoil	5 λ^{**}	9.5 λ	1024	128	131072	8

$*$ – a perfect electrical conductor wrapped with a 0.5λ dielectric coating, $\epsilon_r = 5.0$
$**$ – inner radius refers to chord length

Table 2: Timings

Case	Time (sec.)				Number Iter.
	Total	Fill	Solve	Per Iter.	
Cylinder	112.07	0.06	36.62	0.018	2030
Airfoil	16732.02	0.61	12504.27	0.0954	131072*

* – Program terminated after the number of iterations equaled the number of equations with the normalized residual $< 1.35 \times 10^{-4}$

Table 3: MFlop ratings

Case	Achieved MFlops		Extrapolated MFlops	
	Fill	Solve	Fill	Solve
Cylinder	318	103	2545	823
Airfoil	250	155	2003	1242

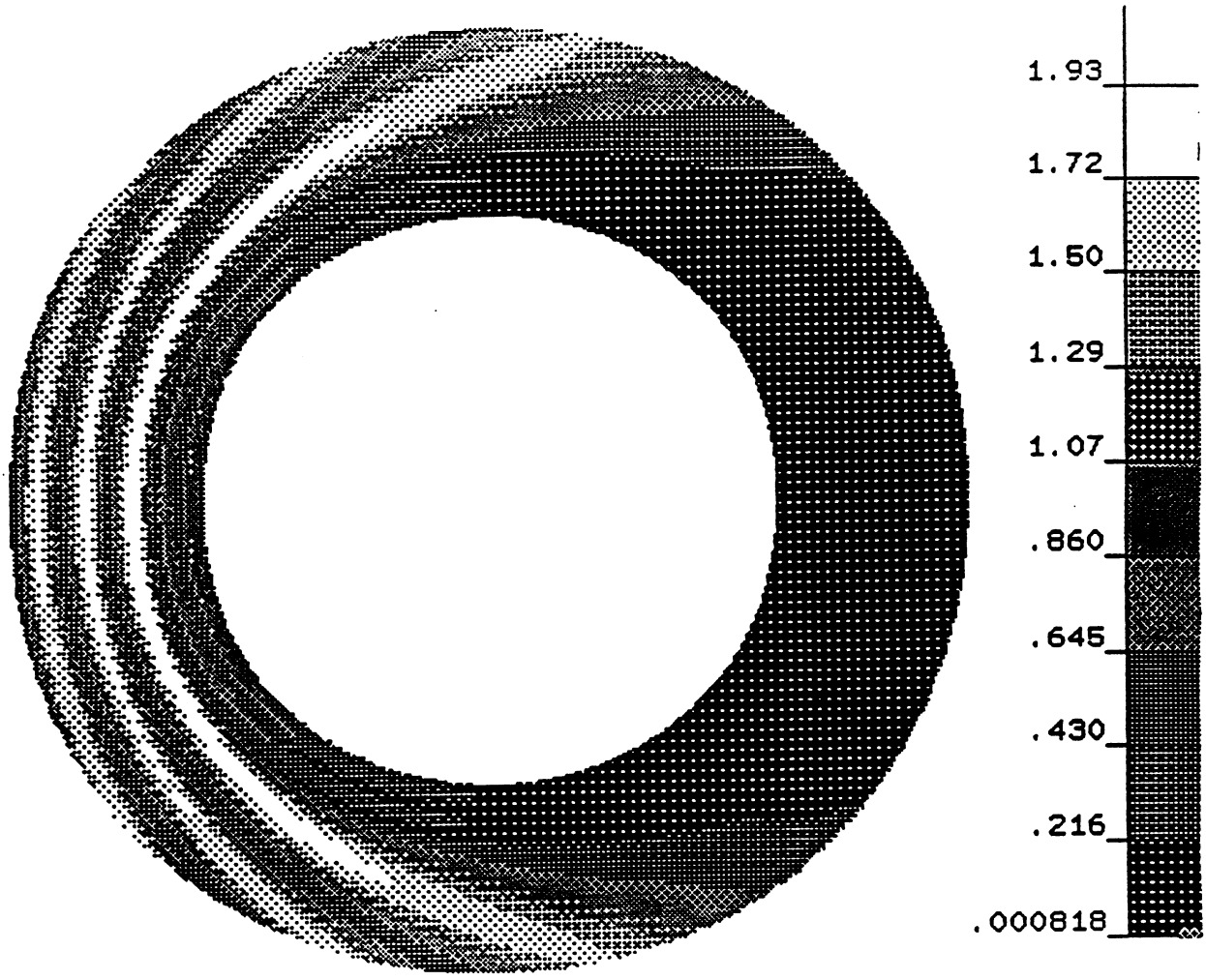

Figure 5a : Cylidrical magnitude: Total field magnitude for scattering from a perfect electric conducting cylinder with $a = 3\lambda$ and $b = 5\lambda$ and a 0.5λ dielectric coating around the cylinder, $\epsilon_r = 5.0$

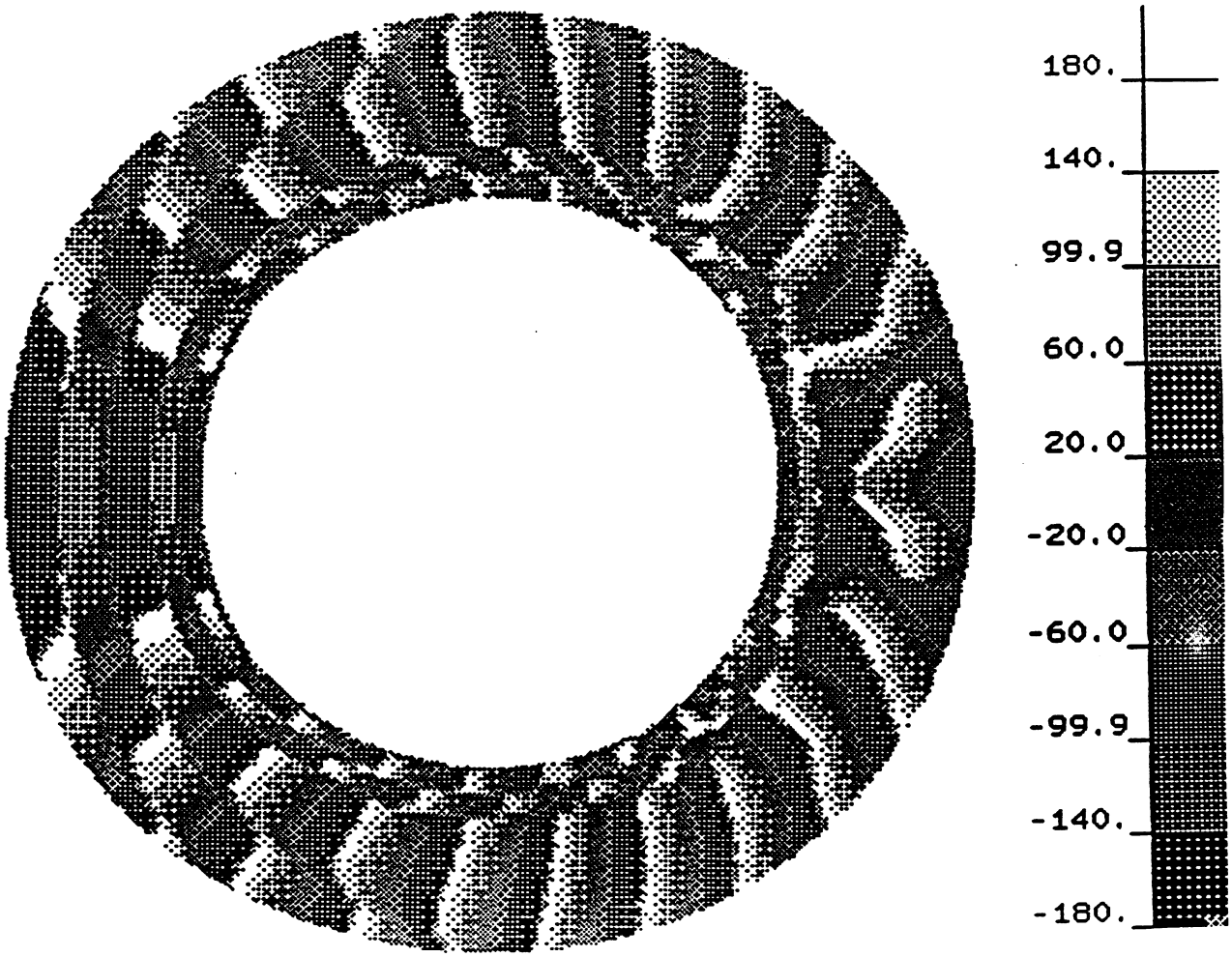

Figure5.b :Cylindrical phase: Total field phase for scattering from a perfect electric conducting cylinder with $a = 3\lambda$ and $b = 5\lambda$ and a 0.5λ dielectric coating around the cylinder, $\epsilon_r = 5.0$

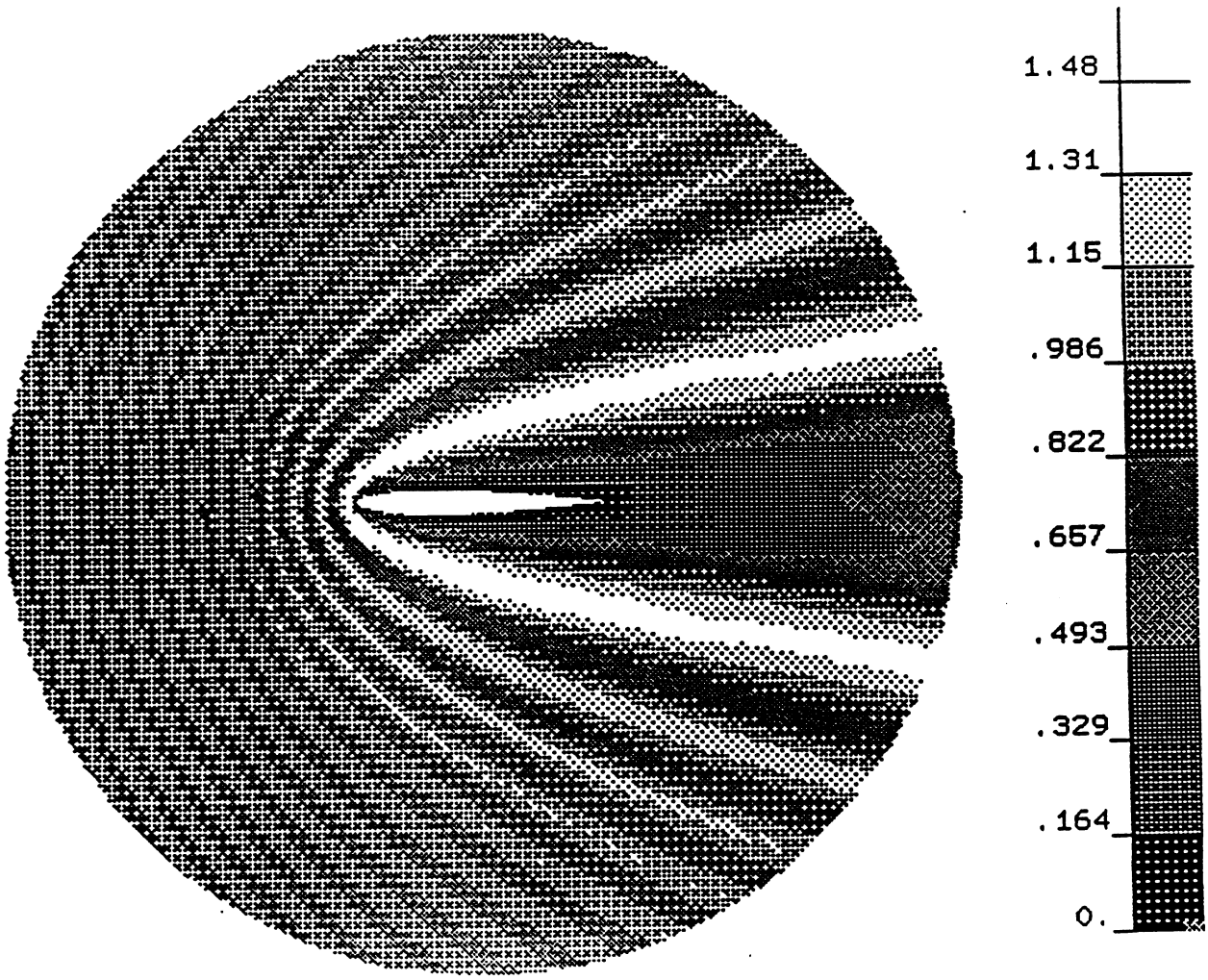

Figure 6a :Airfoil magnitude: Total field magnitude for scattering from a perfect electric conducting airfoil with chord length = 5λ and $b = 9.5\lambda$. NACA number is 0010

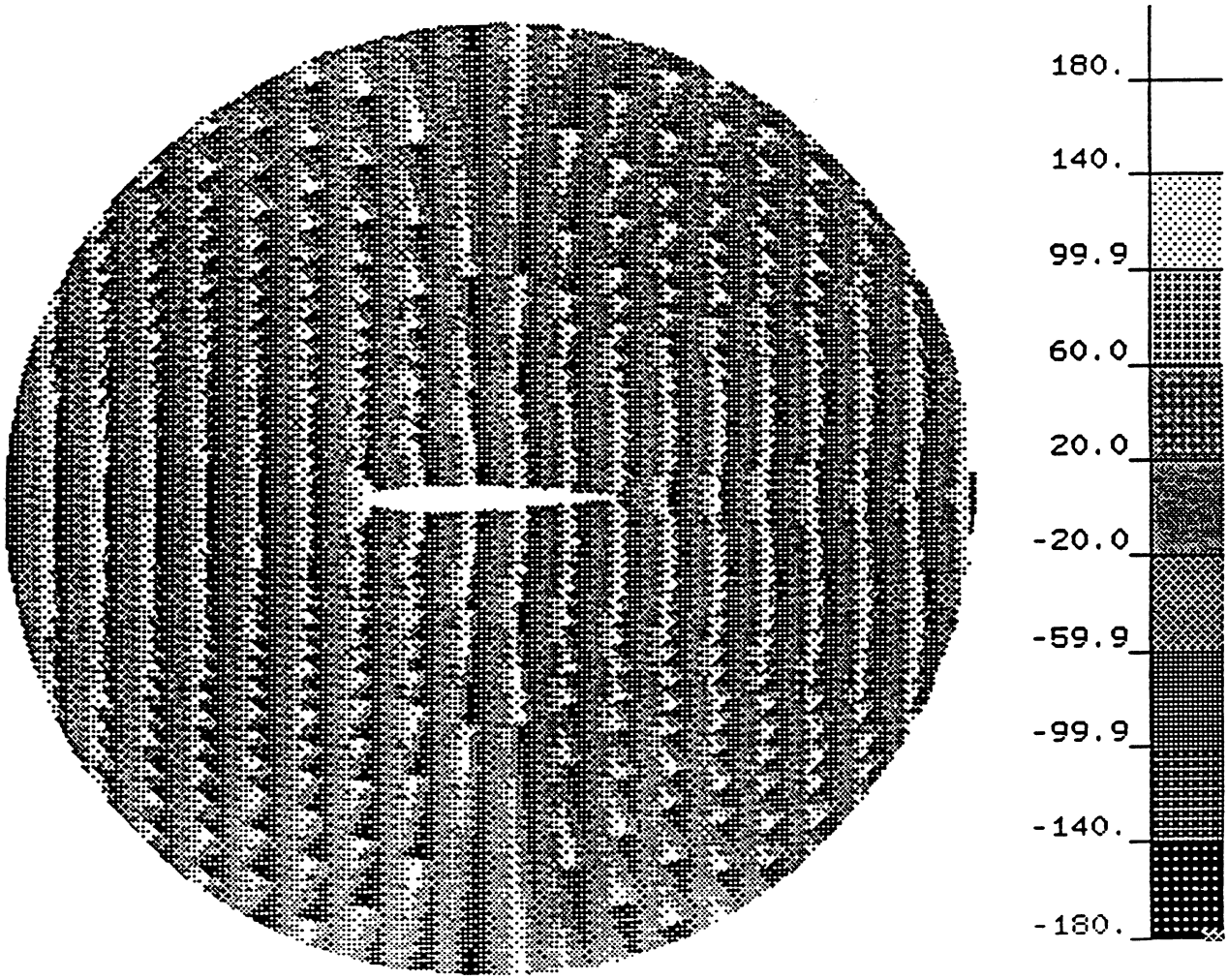

Figure6b :Airfoil phase: Total field phase for scattering from a perfect electric conducting airfoil with chord length = 5λ and $b = 9.5\lambda$. NACA number is 0010