(2)

RADC-TR-88-268
Interim Report
November 1988

LARGE MATRIX SOLUTION TECHNIQUES APPLIED TO AN ELECTROMAGNETIC SCATTERING PROBLEM

ARCON Corporation

Margaret B. Woodworth

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441-5700

This report has been reviewed by the RADC Public Affairs Division (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-68-268 has been reviewed and is approved for publication.

APPROVED: Arthur D. Yaghjian

ARTHUR D. YAGHJIAN
Project Engineer

APPROVED: John K. Schindler

JOHN K. SCHINDLER
Director of Electromagnetics

FOR THE COMMANDER: John A. Ritz

JOHN A. RITZ
Directorate of Plans & Programs

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (EECT) Hanscom AFB MA 01731-5000. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notice on a specific document requires that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) N/A	5. MONITORING ORGANIZATION REPORT NUMBER(S) RADC-TR-88-268
6a. NAME OF PERFORMING ORGANIZATION ARCON Corporation 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Rome Air Development Center (EECT)
6c. ADDRESS (City, State, and ZIP Code) 260 Bear Hill Road Waltham MA 02154	7b. ADDRESS (City, State, and ZIP Code) Griffiss AFB NY 13441-5700
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Center 8b. OFFICE SYMBOL (If applicable) EECT	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F19628-86-C-0104
8c. ADDRESS (City, State, and ZIP Code) Hanscom AFB MA 01731-5000	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO.: 62702F; PROJECT NO.: 4600; TASK NO: 15; WORK UNIT ACCESSION NO.: 88

11. TITLE (Include Security Classification)
LARGE MATRIX SOLUTION TECHNIQUES APPLIED TO AN ELECTROMAGNETIC SCATTERING PROBLEM

12. PERSONAL AUTHOR(S)
Margaret B. Woodworth

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Interim	FROM Aug 86 TO Jun 88	November 1988	36

16. SUPPLEMENTARY NOTATION
N/A

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
17	09		Large Matrix Solution; Conjugate Gradient Method; Gaussian Elimination; Scattering from a 3-D Conducting Cube (See Reverse); Biconjugate Gradient Method
17	02		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

There are many numerical applications where one must solve a large system of linear equations. In order to do this, two issues must be addressed: storage of the large matrices and finding an efficient technique to solve this large system of equations. In this report we look at these issues as they apply to a given electromagnetic scattering problem. We solve surface integral equations to find the surface currents and the far-field scattering from a three-dimensional conducting cube. This three-dimensional scattering problem is solved by magnetic-field integral equations (MFIE), augmented magnetic-field integral equations (AMFIE), or, the recently developed, dual-surface magnetic-field integral equations. We also report on various methods for dealing with the shortage of memory - unified extended memory (UEM), virtual memory, mass storage, and direct access fiels. We explain how to use these methods to increase the size of available matrix memory, without vastly increasing the CPU time beyond the time that would be required with central memory alone. We discuss three techniques of matrix solution: Gaussian elimination, the conjugate

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Arthur D. Yaghjian	22b. TELEPHONE (Include Area Code) (617) 377-4239 22c. OFFICE SYMBOL RADC (EECT)

DD Form 1473, JUN 86 Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

gradient, and the biconjugate gradient methods. The computations reported here were done on three representative scientific computers: a Cyber 860 with NOS 2.5 operating system, a Vax 8650 and a Vax 11/780. Both Vaxes have Vax/VMS version 4.7 operating system.

ITEM 17. COSATI CODES (Continued).

Field	Group
20	14

ITEM 18. SUBJECT TERMS (Continued).

Magnetic-Field Integral Equation (MFIE)
Augmented Magnetic-Field Integral Equation (AMFIE)
Dual-surface Magnetic-field Integral Equation

Computer Memory Storage
Central Memory
Virtual Memory
Unified Extended Memory (UEM)
Mass Storage
Direct Access Files

Contents

Table Captions

Figure Legends

FIGURE 1. CPU time (in seconds) to read and write a complex array of a given length to a direct access disk file on the Vax 8650.

FIGURE 2. CPU time per array element (in seconds) to read and write a complex array of a given length to a direct access disk file on the Vax 8650. The same data as Figure 1, except that the time has been divided by the number of elements in the array.

FIGURE 3. CPU time (in seconds) to run, on the Vax 8650, the program which solves scattering from a cube, versus the size of the scatter matrix (N). The program used the "dual-surface" magnetic-field integral equations and solved the matrix by Gaussian elimination.

FIGURE 4. CPU time (in seconds) to run, on the Vax 8650, the program which solves scattering from a cube, versus the size of the scatter matrix (N). The program used the augmented magnetic-field integral equations and solved the matrix by Gaussian elimination.

FIGURE 5. CPU time (in seconds) to run, on the Vax 8650, the program which solves scattering from a cube, versus the size of the scatter matrix (N). The program used the "dual-surface" magnetic-field integral equations and solved the matrix by the conjugate gradient method.

FIGURE 6. CPU time (in seconds) to run, on the Vax 8650, the program which solves scattering from a cube, versus the size of the scatter matrix (N). The program used the augmented magnetic-field integral equations and solved the matrix by the conjugate gradient method.

Large Matrix Solution Techniques Applied to an Electromagnetic Scattering Problem

1. INTRODUCTION

There are many numerical applications where one must solve a large system of linear equations. In order to do this, two issues must be addressed: storage of the large matrices and finding an efficient technique to solve this large system of equations. In this report we look at these issues as they apply to a given electromagnetic scattering problem. We solve surface integral equations to find the surface currents and the far-field scattering from a three-dimensional conducting cube. We discuss various methods of data storage and concentrate on three techniques of matrix solution: Gaussian elimination, the conjugate gradient, and the biconjugate gradient methods.

The computations reported here were done on three representative scientific computers: a Cyber 860 with NOS 2.5 operating system, a Vax 8650 and a Vax 11/780. Both Vaxes have Vax/VMS version 4.7 operating system. The Cyber is a 64 bit machine with 14 decimal digit accuracy. The Vaxes are 32 bit machines with 7 decimal digit accuracy. Double precision on a Vax is equivalent to single precision on the Cyber.

We report on various methods for dealing with the shortage of memory — unified extended memory (UEM), virtual memory, mass storage, and direct access files. We explain how to use these methods to increase the size of available matrix memory, without vastly increasing the CPU time beyond the time that would be required with central memory alone. We will show that UEM is the best choice on the Cyber when the matrix is up to one million words (almost seven times larger than the central memory 131,000 word capacity). UEM is the fastest of the large storage capacity methods

and is easiest to implement. Virtual memory is accessed automatically on the Vaxes and has about the same storage capacity as UEM. Mass storage is only optimal when the Cyber has a mass storage device attached to it; when mass storage is software simulated a direct access disk file is a better choice for very large storage. Mass storage and direct access are more difficult to implement. They require rewriting the program to extract data from the file for use in the program and to deposit data in the file when it is changed, but they have an enormous data storage capacity. Direct access has a capacity of more than 30 million words on either Vax at AFGL on Hanscom Air Force Base.

Many techniques, both iterative and non-iterative, have been developed to solve large systems of equations. Others have reported on the advantages and disadvantages of the various methods (eg. Sarkar et al., 1981). In this report we consider two of the best methods (Gaussian elimination and conjugate gradient), plus a new iterative method (biconjugate gradient). Two other new iterative methods (augmented conjugate gradient and modified conjugate gradient) are also discussed briefly.

Originally, we used Gaussian elimination to solve the large system of equations. However, Gaussian elimination proved too time consuming. Moreover, the large number of arithmetic operations it requires gives opportunity for the accumulation of large round-off errors. After investigating the literature on the solution of large systems of equations, it was decided to implement the conjugate gradient and, later, to test the biconjugate gradient techniques. In this report we compare the results obtained by these three methods of matrix solution. The conjugate gradient method has the advantages that the convergence is generally rapid for well-conditioned matrices, and the accumulated round-off errors can easily be handled. The conjugate gradient method is generally superior to Gaussian elimination for larger systems of equations ($N > 50$). However (unlike Gaussian elimination) the conjugate gradient method and its variations must be repeated for each incident illumination (right-hand vector). The Gaussian elimination technique can solve for any number of right-hand vectors simultaneously. The biconjugate, augmented conjugate, and modified conjugate gradient methods are three variations of the conjugate gradient method, which for our three-dimensional multi-wavelength scattering problem took more iterations to converage than the original conjugate gradient method.

2. MEMORY STORAGE

To find N unknowns with N equations, the matrix must have N x N elements. Often the whole matrix need not be stored if the matrix has certain properties such as being banded, symmetric, or sparse, or if the elements of the matrix are easy to compute as they are used. However, sometimes this is not the case and very large matrices must be stored in memory. The problem is compounded if

the matrix is complex, over-determined, or both. When solving a large system of equations one can easily run out of central memory storage. In addition, the programs which solve these large systems of equations may require considerable memory themselves. Even if the matrix is within the central memory storage limits, others on the system may object to having large amounts of central memory tied up for the long period of time it takes to solve large matrices.

In solving augmented surface integral equations, the size of the matrices was further increased to a $\frac{3}{2}$N x N over-determined matrix. Although the product of the Hermitian conjugate of the matrix with itself is taken to produce an N x N matrix, there must be enough memory to initially store the entire over-determined matrix.

Large systems of equations often exceed central memory. Central memory storage can be augmented in the following ways: unified extended memory, virtual memory, mass storage, and direct access file storage.

2.1 Unified Extended Memory

On the Cyber 860 the most straightforward, least time consuming method for storing large matrices is unified extended memory (UEM). The program instructions and the rest of the program data are stored in central memory as usual. UEM is accessible in the following way:

1. Put the large matrix in a common block by itself.

2. Designate the common block level 2 (UEM) every place in the program where it appears with a LEVEL statement before the COMMON statement, eg:

LEVEL 2, /NAME/

where NAME is the name of the common block which should be in UEM. Any other common block should be designated level 1 (central memory).

3. Before compiling or running enter RLF, EC = m to specify the extended memory field length needed. The value of m is the extended memory required divided 1000 (in octal). The value of m may be small for compiling. To find the proper m for a run look under STATISTICS in the compilation print out. Use the octal value for ESC LABELLED COMMON LENGTH divided by 1000 (octal). Round up any fraction.

4. Compile the program in FORTRAN 5. Include the parameter LCM = G (extended memory = giant mode addressing).

At AFGL, the maximum amount of UEM on the Cyber during the week is 500,000 (decimal) words, and 1,000,000 (decimal) words on the weekends. A large job may be batched in and held until the weekend. The maximum amount of central memory is 131,000 words. Some of this space is used to store the program instructions and variables, so not all this space is available to the large matrix. Using UEM the maximum matrix size can be increase seven fold. This can almost triple the maximum number of unknowns. However, at AFGL the amount of UEM will be reduced soon due to a change in the operating system. The largest matrix we evaluated with UEM was 840 x 560 complex, or 940,800 words with 560 unknowns. The largest matrix we evaluated with central memory was only 288 x 192 complex or 110,592 words with 192 unknowns.

2.2 Virtual Memory

On a Vax, virtual memory is accessed automatically and no changes need be done to the program. Virtual memory allows the use of matrices of comparable size as UEM on the Cyber. Virtual memory on the Vaxes will accommodate up to about one million words. The exact amount of virtual memory depends on the quotas set at the computer site.

2.3 Direct Access and Mass Storage

In some cases more detailed solutions with larger matrices are needed. For instance we wanted to solve for 1200 unknowns involving a 1800 x 1200 complex matrix (4,320,000 words). This is more than four times as large as what can be stored with UEM or virtual memory. So, a method other than central, unified extended, or virtual memory storage of the matrix is required. Two alternative methods that provide much larger storage capacity are mass storage on the Cyber, and direct access files on both the Cyber and the Vaxes.

In both methods the matrix is divided into its rows and the rows are stored in a directly accessible file on disk. For instance if element (5, 9) is to be modified, row 5 is copied into an array in memory and the 9th element changed, then row 5 is copied back into the file. For both the conjugate gradient method and Gaussian elimination, the extra CPU time required to manipulate the direct

access files approximately doubles the total CPU time, provided programs are written to optimize memory use. Usually programs are written to optimize calculation time, therefore changes should be made to minimize data file access time which now can be the dominant factor in the total CPU time. All the operations that can be done on a row at once should be done. Some duplication of calculations may greatly improve file manipulation CPU time. We had a case where the original program made use of symmetry in the matrix to avoid re-calculation of the elements under the diagonal by assigning Q(n, m) = Q(m, n) for n < m. However, the CPU time required for this section of the program was proportional to N cubed, because reading in the row from the file took longer than recalculating the value of the element. When the program was rewritten to calculate every element, ignoring symmetry and thus avoiding unnecessary read statements, the time required was proportional to N squared — a major saving in time. The amount of central memory required in our disk-storage program decreases to 3 N; a vast improvement over N^2 memory, that allows much larger matrices to be solved. The major limiting factors are now disk space and computer time, rather than central memory.

Appreciable changes must be made to the programs in order to use mass storage and direct access files. These changes are similar for both methods and will be discussed in the following two sections. In both cases we dimension three, one-dimensional arrays with sizes equal to the number of columns in the original matrix. These arrays are working space for the manipulation of the matrix elements. The actual number of arrays needed depends on the intricacy of the program.

For Gaussian elimination, once the matrix solution is complete, our subroutine has the solution vector in the last column of the matrix, that is, the last element in each row. In order to minimize the number of read statements, we read each row once, and put the solution elements into an array in central memory for further manipulation.

The temporary file containing the matrix tends to be extremely large. If possible, it should be placed in scratch space and deleted as soon as the solution vector is found.

2.3.1 Direct Access

To use direct access, do the following:

1. Open the file with "direct" access and record length (RECL) equal to row size. (If the array is double precision or complex, the record length should be double the number of columns (NC).) The maximum record length on the Vaxes is 8191 words. An example of an OPEN statement is:

```
OPEN (1, ACCESS = 'DIRECT', RECL = 2 * NC)
```

On the Vaxes, you should also include the parameter INITIALSIZE = ISIZE, where ISIZE is the number of blocks of memory needed to store the matrix. ISIZE = RECL * NR / 128 (record length times number of rows divided by 128 elements per block). Declaring the initial size causes the necessary blocks of memory to be set aside when the data file is opened. This causes the memory blocks to be grouped together contiguously, rather than scattered all over the disk in little sections. Contiguous blocks of memory can be read and written more quickly than scattered blocks.

2. Calculate the elements in a row and write each row to the direct access file, filling the matrix row by row.

```
WRITE (1, REC = M) Q
```

where M is the row number and Q is the array work space containing row M. When reusing the array work space, make sure that either the array is read in from the file or that each element is set to a value. The default is no longer zero.

3. To make changes on a row once it exists, first read the row in, modify the elements, then rewrite the row out to the file. For example:

```
READ (1, REC = I) Q1
READ (1, REC = J) Q2
      ⋮
Q2(5) = Q2(5) − X * Q1(5)
      ⋮
WRITE (1, REC = J) Q2
```

A problem might occur in the above example if I = J. Q1 and Q2 would represent the same row, but would have different elements. This situation can easily occur within a DO loop, and should be checked for with an IF-THEN block with alternative program instructions which avoid duplication of a row in memory. (If there are no changes in a row, the row does not have to be rewritten. The conjugate and biconjugate gradient methods never modify the elements of the matrix.)

2.3.2 Mass Storage

To use mass storage:

1. Open the file with the OPENMS statement.

```
CALL OPENMS (1, INDEX, NR+1, 0)
```

where 1 is the unit number, INDEX is an integer array with dimension NR+1 which contains the mass storage index, NR is the number of rows in the matrix, and 0 signifies that the index is by number rather than by name.

2. Calculate the elements in a row and write each row to the direct access file, filling the matrix row by row.

```
CALL WRITMS (1, Q, NC, M, 0)
```

where 1 is the unit number, Q is the array containing row M, NC is the array size (i.e. number of columns in the matrix), M is the row number, and 0 signifies write a new record. Use -1 instead of 0 when the row is to be changed in the mass storage file.

3. To make changes to a row once it exists, first read the row in, modify the elements, then rewrite the row out to the file. For example:

```
CALL READMS (1, Q1, NC, I)
CALL READMS (1, Q2, NC, J)
        ⋮
Q2(5) = Q2(5) − X * Q1(5)
        ⋮
CALL WRITMS (1, Q2, NC, J, -1)
```

A problem might occur in the above example if I = J. Q1 and Q2 would represent the same row, but would have different elements. This situation can easily occur within a DO loop, and should be checked for with an IF-THEN block with alternative program instructions which avoid duplication of a row in memory. (If there are no changes in a row, the row does not have to be rewritten. The conjugate and biconjugate gradient methods never modify the elements of the matrix.)

2.4 Matrix Storage Time

The different data storage methods took different amounts of time to run due to their differing efficiencies. The clock run time is composed of three factors: CPU time, input/output (I/O) manipulation time, and competition from other jobs on the computer. The CPU time and clock time are explicitly known for each run, but I/O manipulation time and percent of time shared with other users are not quantified. Only the I/O manipulation units are quantified. Complicated I/O processing also requires more central processing than central memory storage, thus also increasing CPU time. None of the machines at AFGL are tuned for efficient I/O processing. Usually this is the slowest part of large jobs, so it is a major contributor to clock run time and CPU run time for I/O intensive jobs.

The effect on the total CPU of the different matrix storage techniques depends on the specific routine used. In this section, the different matrix storage times are compared for augmented matrices solved with our Gaussian elimination routine. The CPU time of the Gaussian elimination routine is proportional to N cubed for each of the methods of data storage. There are also differences in the speed of the CPUs of the various computers. The Vax 8650 has a 1.3 megaflop, 32-bit Linpack performance rating, and the Cyber 860 has a 2.1 megaflop, 64-bit Linpack rating. We found that with our program the Cyber 860 was the fastest, 40% to 50% faster than the Vax 8650, and 6 to 7 times faster than the Vax 11/780. The Vax 8650 was 4 to 5 times faster than the Vax 11/780.

Central memory has an advantage over other data storage methods in that it is very fast accessing information. Other forms of data storage require more computer effort to store and retrieve data. The further removed from central memory, the slower the data storage and retrieval.

Unified extended memory increases CPU time by about 25%. For instance on the Cyber, one run with a 288 x 192 complex matrix (110,592 words) took 101 CPU seconds using central memory only, and 126 CPU seconds when the matrix was stored in UEM.

Mass storage is only optimal when a mass storage device is attached to the Cyber. In our case the mass storage utility is only software simulated, and approximately doubles CPU time over central memory, but it also increases data manipulation cost by more than 300 times. For a 162 x 108 matrix the data manipulation cost increased from 2,228 units using central memory to 728,945 units using mass storage. The CPU time is comparable to that of direct access, and the total run cost is 50% greater.

Use of a direct access disk file nearly doubles CPU time on both the Cyber and the Vaxes. On the Cyber, for a 162 x 108 matrix, data storage costs increased by 80 times over central memory to 176,655 units (one quarter of the cost of mass storage). On the Vax 11/780 with a 288 x 192 matrix

CPU time was 616 seconds with central memory and 1215 seconds with direct access. On the Vax 8650, we determined the CPU time required to read and write a complex array to a direct access disk file as a function of row length (see Figure 1). Figure 2 is a plot of CPU time required to operate on each element of the row. When N > 200, the CPU time per element approaches a constant (about 2×10^{-6} sec/element). When N < 50, CPU time per row is nearly independent of N (about 2×10^{-4} sec/row). See Section 3.4 for more information on CPU run time.

3. MATRIX SOLUTION TECHNIQUES

Both the conjugate gradient method and the Gaussian elimination techniques belong to the class of iterative methods known as the method of conjugate directions. This class of iterative method always converges to the solution at a rate of geometric progression in a finite number of steps, provided round-off errors are negligible. The method of conjugate directions starts with a basis set of linearly independent vectors and from them generates a set of conjugate direction vectors. The conjugate gradient method uses the series of residual vectors as the basis set, and Gaussian elimination uses the set of unit basis vectors, i.e. (1, 0,..., 0), (0, 1, 0,..., 0), ..., (0,..., 0, 1). (See Hageman and Young (1981) Section 7.2, pp. 139–142). We define

$$\mathbf{A}\,\mathbf{X} = \mathbf{Y}$$

where **A** is an N x N matrix and **X** is the unknown to be solved given **Y**. Both **X** and **Y** are vectors with dimension N.

The biconjugate, augmented conjugate, and modified conjugate gradient methods each solve two complementary equations simultaneously. For an explanation of these three methods see Sarkar (1987) and Sarkar et al. (1988).

3.1 Gaussian Elimination

Gaussian elimination is a well-known, extensively used method of matrix solution. A solution may always be obtained in a finite number of (N) steps as long as the matrix is not singular and round-off errors are negligible. For a well-conditioned matrix with N < 50, Gaussian elimination is a very reliable method of solving a system of equations. However, the round-off error builds up for ill-conditioned matrices. Also round-off errors may be prohibitive for large systems of equations in which a large number of arithmetic operations are required. There appears no simple way of correcting the round-off errors that accumulate in the solution vector using Gaussian elimination. The accuracy of

the solution is not checked unless we multiply the solution vector times the original matrix and compare the result to the original Y vector. Many algorithms of Gaussian elimination, like ours, do not keep the original matrix in order to save memory storage space, especially if the matrix is large.

We had little trouble with round-off errors using Gaussian elimination, even for large matrices. For single precision the Cyber 860 has a 64 bit word length (14 significant figures) and the Vax 11/780 has a 32 bit word length (7 significant figures). When we compared two runs with the two word lengths for 300 unknowns (a 450 x 300 complex matrix) we had identical results to 5 or 6 significant figures. The results for 432 unknowns (a 648 x 432 complex matrix) were the same to 2 or 3 decimal places on the 32 bit Vax and the 64 bit Cyber. That is a difference of about 1%. This was sufficient accuracy for our surface integral equation applications, which required an accuracy of only 2 or 3 significant figures. However, round-off errors probably become more serious for larger matrices solved by Gaussian elimination.

3.2 The Conjugate Gradient Method

The conjugate gradient method is an effective iterative method for solving a large system of equations. It converges in a finite number of steps for any initial guess, as long as the matrix is not singular and round-off errors are kept negligible. Other iterative methods (eg. Newton's method and the biconjugate gradient method) may not converge for all initial guesses. We used an initial guess of $X_0 = 0$, the null vector. The flexibility of accepting any initial estimate permits the user to end the iteration and start again using the current estimate of the solution as a new initial guess. This restarting eliminates any round-off error which has accumulated in the iterative terms such as the residual vector. The restarting should be done when the round-off builds up — more often for poorly conditioned and large matrices, and less often for well-conditioned and small matrices. Restarting requires an extra matrix multiplication, slows down the convergence, and increases the run time. It should be done only when necessary to reduce round-off errors. We found, for our matrices, on the 32-bit Vax, that it was sufficient to restart only once, before the final step, even when N was equal to 3468.

We use a version of the conjugate gradient method discussed in Sarkar and Arvas (1985). Start with an initial guess X_0 and define

$$R_0 = Y - A X_0$$

$$P_0 = G_0 = A^* R_0$$

Iterate the following for $i = 1, 2, \ldots$

$$a_i = \frac{\| G_{i-1} \|^2}{\| A P_{i-1} \|^2}$$

$$X_i = X_{i-1} + a_i P_{i-1}$$

$$R_i = R_{i-1} + a_i A P_{i-1}$$

$$G_i = A^* R_i$$

$$b_i = \frac{\| G_i \|^2}{\| G_{i-1} \|^2}$$

$$P_i = G_{i-1} + b_i P_{i-1}$$

where we define $\| C \|^2 = \langle C, C \rangle = \sum_j c_j \overline{c_j}$, the inner product of any vector C with itself, and $\overline{c_j}$ is the complex conjugate of vector element c_j. X, Y, R, and G are vectors; a and b are scalars. A^* denotes the transpose, complex conjugate of matrix A. R is the residual vector. This method minimizes R, giving a sequence of least squares solutions to A X = Y, such that X_i gets closer to X_{true} with each iteration. Iteration continues until some convergence criterion. We use $\| R_i \| / \| Y \| < 10^{-6}$. This is the most widely used method of conjugate gradient solution. Three other methods of conjugate gradients are also discussed in Sarkar and Arvis (1985).

One can save CPU time, I/O time, and memory storage by avoiding explicitly forming the A^* matrix. (Forming the A^* matrix requires N^2 reads and N writes.) Above we have $G = A^* R$. This product can be found indirectly by the following routine:

```
      DO 10 J = 1, N
        READ (1, REC = J) A
        DO 10 K = 1, N
10        G(K) = G(K) + CONJG(A(K)) * R(J)
```

where the array A is the j^{th} row of the matrix A.

3.3 The Biconjugate Gradient Method

A second method of iterative matrix solution is the biconjugate gradient method which is a variation on the conjugate gradient method. For an explanation of the biconjugate gradient method see Sarkar (1987). We found that the biconjugate gradient method is very sensitive to the initial guess and the structure of the matrix. Unlike the conjugate gradient method, the biconjugate gradient method does not converge for all initial guesses. We could not use an initial guess of $X_0 = 0$, because this resulted in a division by zero. We tried two different initial guesses. One guess was $X_0 = Y$, the right-hand vector. This converged very slowly. We also tried the results of one step of the conjugate gradient method as the initial guess for the biconjugate gradient method. This guess produced faster convergence than the $X_0 = Y$ guess. This initial X is found in the following way:

$$G = A^* R$$

$$X_0 = \frac{\| G \|^2}{\| A G \|^2} G$$

3.4 Time Comparison

The three methods for solving a system of equations were applied to a problem in electromagnetics. The problem involved finding the scattering from a three-dimensional conducting cube. We used two methods to find the surface current, from which the scattered field is determined. Conventional magnetic-field integral equations (MFIE) for a perfectly conducting body introduce spurious solutions at the resonant frequencies of the interior cavity formed by the scattering surface, thus yielding unreliable solutions. To overcome the spurious results, we apply one of the following two modifications: 1) the augmented magnetic-field integral equations (AMFIE) (Yaghjian, 1981), or 2) the "dual-surface" magnetic-field integral equations (Tobin et al., 1987). The cube surface is divided up into a large number of patches and the two components of surface current are found for each patch, yielding a total of N unknown complex currents. The matrix used for AMFIE is an over-determined $\frac{3}{2}$N x N matrix. It is multiplied by its Hermitian conjugate to produce an N x N symmetric, square matrix. Unfortunately, this also squares the condition number. The "dual-surface" integral equation adds the effect of an imaginary interior surface to yield an even-determined N x N non-symmetric matrix. The N x N matrices formed in these ways are then solved by Gaussian elimination, the

conjugate, or the biconjugate gradient methods. The perimeter of the cube in wavelengths, λ, is given by $4S/\lambda$, where S is the side-length of the cube.

The programs in this section are all run on a Vax 8650 for consistency in comparing run times. The total CPU time is also affected by the method of data storage. In all the cases in this section, the files are stored on direct access disk files, because most of the matrices are too large to store in any other way.

The number of computer operations provides a rough estimate of efficiency of the algorithm. Table 1 below compares the number of operations required by our subroutines for the three methods (only the highest order terms are included). The read and write statements are required because of the use of direct access files. For large N, the CPU time required by each read or write is proportional to N, so their contribution to the total CPU time is of the same order as the other operations. Note, our version of Gaussian elimination has a round-off error check in the inner most DO loop. This accounts for the IF operations and half of the multiplications with Gaussian elimination.

TABLE 1: *Number of complex operations required for the three methods of matrix solution.*

METHOD	NUMBER OF OPERATIONS						
	*	+	−	LOOPS	WRITE	READ	IF
Gaussian Elim. (total)	$\frac{2}{3}N^3$	0	$\frac{1}{3}N^3$	$\frac{1}{3}N^3$	$\frac{1}{2}N^2$	$\frac{3}{2}N^2$	$\frac{1}{3}N^3$
Conjugate Grad. (per iteration)	$2N^2$	$2N^2$	0	$2N^2$	0	2N	0
Biconjugate Grad. (per iteration)	$2N^2$	$2N^2$	0	N^2	0	N	0

We determined the CPU time required for arithmetic and file operations on the Vax 8650. The file operation times are shown in Figures 1 and 2. The arithmetic times are listed below in Table 2. It is interesting to note that as expected: a complex add takes about twice as long as a real add, a complex multiply takes about four times as long as a real multiply, and a complex divide takes about twelve times as long as a real divide. The CPU time for the logical comparison is approximate and depends on the number of true and false results. The complex arithmetic and file operation times can be inserted into Table 1 to produce expected CPU times.

TABLE 2: *CPU time required for various operations on the Vax* 8650.

OPERATION	CPU (10^{-6} sec)
Real Add	.34
Real Multiply	.51
Real Divide	1.12
Complex Add	.84
Complex Subtract	.96
Complex Multiply	2.03
Complex Divide	13.09
Logical IF	15.09
DO Loop	.67

First we consider the well-known methods of matrix solution: Gaussian elimination and the conjugate gradient method. Using the read and write times for N = 1000 (2.30 x 10^{-6} N and 2.41 x 10^{-6} N, respectively) we estimate the Gaussian elimination CPU time as 11.6 x 10^{-6} N^3, and the conjugate gradient CPU time as 11.7 x 10^{-6} N^2 I, where I is the number of iterations. Therefore, in order for the conjugate gradient method to be more efficient than Gaussian elimination, the conjugate gradient method must converge sufficiently in less than N iterations, which is usually true unless the matrix is highly ill-conditioned. (Note, the logical comparison contributes to nearly half the Gaussian elimination time, and the read and write times contribute to about 40% of both the Gaussian elimination and the conjugate gradient CPU times. If the matrix can be stored and manipulated entirely within the central memory of the computer, i.e. without requiring direct access storage, and if we remove the round-off error check in the Gaussian elimination subroutine, then the conjugate gradient method would be the more efficient routine if it converged within N/6 iterations.)

We compared our CPU estimates with some actual runs. We compared two runs where N = 1200 and $4S/\lambda = 20$, with "dual-surface" magnetic-field integral equations. The Gaussian elimination run used 6 hr, 21 m, 58 s (22918 s) of CPU. The conjugate gradient run used 42 m, 25 s (2545 s) of CPU in 122 iterations. The Gaussian elimination run took 22918 / 2545 $\approx$ 9 times longer than the conjugate gradient run. Using the above estimates we would expect a Gaussian elimination run time of 20,000 CPU sec and a conjugate gradient run time of 2050 CPU sec. The Gaussian elimination estimate is 90% of the actual run time and the conjugate gradient estimate is 80% of the actual run time. These estimates are quite good because we do expect the matrix solution part of the program to take about 90% of the total run time. The estimate of the conjugate gradient run time excludes the CPU time needed to find the complex conjugates, which may account for the estimated run time being slightly low. According to the estimated run times, the Gaussian elimination run should be 1200 / 122

$\approx$ 10 times longer than the conjugate gradient run. This is quite close to the actual value of 9 times longer.

If the matrix is not ill-conditioned, the conjugate gradient method will converge within $M < N$ steps where M is the number of independent eigenvalues of A. Matrices which have dominant diagonals tend to be well-conditioned and have quite a few eigenvalues close together. The conjugate gradient method will converge quite rapidly to a chosen accuracy for these matrices, even for a large system of equations. The number of independent eigenvalues in a matrix depends on the condition number of the matrix and the structure of the matrix. For a given matrix size, the higher the frequency, the more iterations are required to reach convergence. As frequency is increased for the same number of patches, the surface current changes more rapidly from patch to patch and the matrix elements associated with the patches differ greatly from each other, thus slowing convergence. Table 3a and 3b list the number of iteration (I) required for convergence with AMFIE and "dual-surface" matrices, respectively. The AMFIE matrices have been squared, thus their condition number has also been squared and they take many more iterations to solve.

TABLE 3a: *Number of iterations (I) required for convergence, AMFIE case .*

N	$4S/\lambda$	patches/λ^2	I	I/N
192	6.03	28.2	206	1.07
432	3.02	252.6	149	.35
432	6.03	63.4	243	.56
768	12.06	28.2	489	.64

TABLE 3b: *Number of iterations (I) required for convergence, dual-surface case.*

N	$4S/\lambda$	patches/λ^2	I	I/N
432	.96	2500.0	31	.07
48	3.02	28.1	35	.73
108	3.02	63.2	39	.36
192	3.02	112.3	42	.22
432	3.02	252.6	45	.10
192	6.03	28.2	61	.32
432	6.03	63.4	62	.14
768	6.03	112.6	61	.08
432	9.60	25.0	83	.19
768	9.60	44.4	82	.11
1200	9.60	69.4	88	.07
768	12.06	28.2	90	.12
1200	12.06	44.0	92	.08
1728	12.06	63.4	93	.05
1200	20.00	16.0	122	.10
1728	20.00	23.0	118	.07
3468	20.00	46.2	119	.03
3468	27.00	25.4	141	.04

A significant conclusion here is that the number of iterations at a given frequency do not depend strongly on the number of patches per square wavelength. Since the CPU time is proportional to the total number of patches, we conclude that CPU time is minimized by choosing the least number of patches sufficient for the desired solution accuracy. We found that the minimum number of unknowns required by magnetic-field surface integral equations to compute reasonably accurate far-fields for a perfectly conducting cube of surface area $\mathcal{A}$, is roughly 50 $\mathcal{A}/\lambda^2$, where λ^2 is the wavelength squared. There are two components of the surface current in each patch, so this is equivalent to 25 patches per λ^2. (For the cube, x-y symmetry was used to further reduce the total number of unknowns by a factor of four.)

The total CPU time is equal to $\mathcal{T}$ (sec) = c N^d, where c and d should be constants. For the estimates above, we assumed d = 3 for Gaussian elimination (GE), and d = 2 (plus a factor for the number of iterations) for the conjugate gradient method (CG). Figures 3 - 6 show plots of log CPU time versus N for different cases. We construct Table 4 of c and d for the total CPU times over the whole range of N. Table 5 gives c and d for each increment in N, showing how c and d vary with N. c has only one significant figure; d has two significant figures. The conjugate gradient results are for

the total run, not per iteration as in Table 1. The complete computer program includes: filling the matrix, Hermitian multiplication (if AMFIE), solving the matrix for the surface currents, and finding the scattered field. The matrix solution part takes up about 90% or more of the total CPU time for large N.

TABLE 4: *Constants c and d found over the whole range of N.*

Matrix type	Matrix solution	c (10^{-6} sec)	d
dual-surface	GE	30	2.9
AMFIE	GE	80	2.8
dual-surface	CG	80	2.4
AMFIE	CG	7	3.1

TABLE 5: *Constants c and d found over small ranges of N.*

Matrix type	Matrix solution	N (range)	c (10^{-6} sec)	d
dual-surface	GE	432- 768	40	2.8
		768-1200	20	3.0
AMFIE	GE	192- 300	90	2.8
		300- 432	100	2.8
		432- 768	50	2.9
dual-surface	CG	432- 768	6	2.8
		768-1200	200	2.3
		1200-1728	300	2.3
		1728-3468	200	2.3
AMFIE	CG	432- 768	7	3.1

To minimize the run time we must minimize c and d, especially the exponent d. It is clear that for large N, the "dual-surface" matrix solved with the conjugate gradient method is the quickest. The value of d for the conjugate gradient method improves as N increases apparently approaching 2; but d gets worse for Gaussian elimination as N increases, approaching 3. The AMFIE matrix does poorly with the conjugate gradient method because the Hermitian conjugate multiplication has a time consuming N^3 dependence, and the matrix is much less well-conditioned, converging slowly, sometimes with more than N iterations. This causes a more than cubic dependence on N. The two Gaussian elimination cases have exponents comparable to the expected nearly cubic dependence in N.

Next we discuss our work with the biconjugate gradient method. We found that the biconjugate gradient method is very sensitive to the initial guess and the structure of the matrix. Sarkar (1987) states that the biconjugate gradient method converges up to 17 times faster than the conjugate gradient method for a Toeplitz matrix. However, we found that the biconjugate gradient rate of convergence is strongly dependent on the structure of the matrix. Toeplitz matrices are highly structured. Table 6 compares the rate of convergence of the two methods on the Vax 8650 for different frequencies for scattering from a cube. We used the result of a single step of the conjugate gradient method as the initial guess of X for the biconjugate gradient method. When the number of patches per square wavelength is large, the biconjugate gradient method converges at about the same rate as the conjugate gradient method. However, when the number of patches per square wavelength is small, (i.e. at high frequencies) the biconjugate gradient method takes up to 5 times longer than the conjugate gradient method. At low frequencies the surface current does not change much from patch to patch, thus the resultant matrices are more structured than high frequency matrices and relatively easier for the biconjugate gradient method to solve (provided the total number of patches remains fixed). The conjugate gradient method is not as sensitive to matrix structure as the biconjugate gradient method. Since we wanted to save CPU time by using the minimal number of patches as the frequency increased, we found the biconjugate gradient method less efficient than the conjugate gradient method. Moreover, the biconjugate gradient method did not show a faster convergence rate during the early iterations, so that switching back to the conjugate gradient method after a number of biconjugate gradient iterations did not reduce computer time over a pure conjugate gradient method.

TABLE 6: *Comparison of biconjugate gradient and conjugate gradient convergence over a range of frequencies for a given N, dual-surface case.*

			Biconjugate Grad.		Conjugate Grad.	
N	$4S/\lambda$	patches/λ^2	I	CPU	I	CPU
432	.96	2500.0	27	1m 32s	31	1m 54s
432	3.02	252.6	43	2m 9s	45	2m 18s
432	6.03	63.4	284	11m 31s	62	3m 8s
432	9.60	25.0	548	21m 23s	83	4m 29s

3.5 The Augmented and Modified Conjugate Gradient Methods

A third method of iterative matrix solution is the augmented conjugate gradient method. All the variations of the conjugate gradient method solve Hermitian systems of equations. We get around

this requirement for the conjugate gradient method in section 3.2 by essentially solving $A^* A X = A^* Y$, where A is not necessarily Hermitian. This, however, squares the condition number and may cause slow convergence. Sarkar et al. (1988) introduce an augmented form of the conjugate gradient method which solves

$$\begin{bmatrix} 0 & A \\ A^* & 0 \end{bmatrix} \begin{bmatrix} C \\ X \end{bmatrix} = \begin{bmatrix} Y \\ \overline{Y} \end{bmatrix}$$

or

$$[B] \quad [T] = [Z]$$

They have augmented the matrix equation $A X = Y$ with the adjoint equation $A^* C = \overline{Y}$ to produce the Hermitian system $B T = Z$. Here $\overline{Y}$ is the complex conjugate of Y, and C is an unknown vector which is never explicitly needed. Thus the A matrix is not squared, but the number of equations is doubled. For details see Sarkar et al. (1988).

We used their computer routine GACG (generalized augmented conjugate gradient) to solve our scattering problem. We found GACG very slowly converging and like the biconjugate gradient method, very sensitive to the starting guess. We also did some work with their modified conjugate gradient method which begins with the same Hermitian system as the augmented conjugate gradient method, but solves it in a slightly different way. We found similar problems with slow convergence and sensitivity to the starting guess. For our electromagnetic 3-dimensional scattering problem, we found the original conjugate gradient method described in section 3.2 to be the best.

4. CONCLUSIONS

Central memory is the most efficient way of storing and manipulating data. However, when solving large systems of equations one can easily run out of central memory storage space. Virtual memory is accessed automatically on the Vax and has about the same storage capacity as unified extended memory (UEM) on the Cyber. We demonstrated that UEM is the best choice on the Cyber when the matrix is up to four or eight times the central memory capacity. UEM is the fastest of the large storage capacity methods and is easy to implement; only minor modification of the program is required. Direct access (Cyber and Vax) is more difficult to implement. It requires rewriting the program to extract data from the file for use in the program and to deposit data in the file when the data is changed. In our case, it took nearly twice the CPU time as with central memory, but it also enormously increases the storage capacity by using disk space. A direct access file will handle huge

data sets which can not be handled by any other method. Mass storage (Cyber) is comparable to direct access in its difficulty of implementation and in its data handling capacity. However, mass storage is only optimal when the Cyber has a mass storage device attached to it; when it is software simulated (as in our case) a direct access disk file is a less costly choice for very large storage.

For a well-conditioned, large ($N > 50$) system of equations, based on our surface integral equations applied to the cube, the conjugate gradient method proved superior to Gaussian elimination for three main reasons: 1) The conjugate gradient method generally converges to a solution much more rapidly than Gaussian elimination; for a large N x N matrix, the exponential time dependence in N approaches about 2 rather than the nearly cubic dependence for Gaussian elimination. 2) The Gaussian elimination with a round-off error check has a logical comparison in the innermost loop, which is very time consuming. 3) Accumulated round-off errors are easily eliminated with the conjugate gradient method, although we found no serious round-off problems with Gaussian elimination for complex matrices as large as 648 x 432 solved with 32-bit accuracy. Of course, it should be noted that the conjugate gradient method must be repeated for each right-hand source vector, whereas Gaussian elimination can be applied to several right-hand vectors simultaneously. Thus our conclusions stated here applied to a single aspect angle of the scatterer (i.e. only one right-hand vector).

The biconjugate gradient method proved to be very sensitive to the initial guess and to the structure of the matrix being solved. We found that for our three-dimensional multi-wavelength problem, it converged much more slowly than the conjugate gradient method, regardless of the initial guess, or whether it was combined judiciously with the conjugate gradient method. The augmented and modified conjugate gradient methods had similar defects. In our case, we found that the original conjugate gradient method of matrix solution described in section 3.2 to be the most efficient for large matrices.

REFERENCES

Hageman, L.A. and D.M. Young, (1981), "Applied iterative methods," Academic Press, New York.

Sarkar, T.K., (1987), "On the application of the generalized biconjugate gradient method," J. EM Waves and Application, Vol. 1, pp. 223–242.

Sarkar, T.K. and E. Arvas, (1985), "On a class of finite step iterative methods (conjugate directions) for the solution of an operator equation arising in electromagnetics," IEEE Trans. on Antennas and Propagation, Vol. AP-33, pp. 1058–1066.

Sarkar, T.K., A.R. Djordjevic, and E. Arvas, (1981), "On the choice and weighting functions in the numerical solution of operator equations," IEEE Trans. on Antennas and Propagation, Vol. AP-33, pp. 989–996.

Sarkar, T.K., X. Yang, and E. Arvas, (1988), "A limited survey of various conjugate gradient methods for solving complex matrix equations arising in electromagnetic wave interactions," Wave Motion, Vol. 10, pp. 527-546.

Tobin, A.R., A.D. Yaghjian, and M.M. Bell, (1987), "Surface integral equations for multi-wavelength, arbitrarily shaped, perfectly conducting bodies," Digest of the National Radio Science Meeting, Boulder, CO, p.9.

Yaghjian, A.D., (1981), "Augmented electric- and magnetic-field integral equations," Radio Sci., Vol. 16, pp. 987–1001.

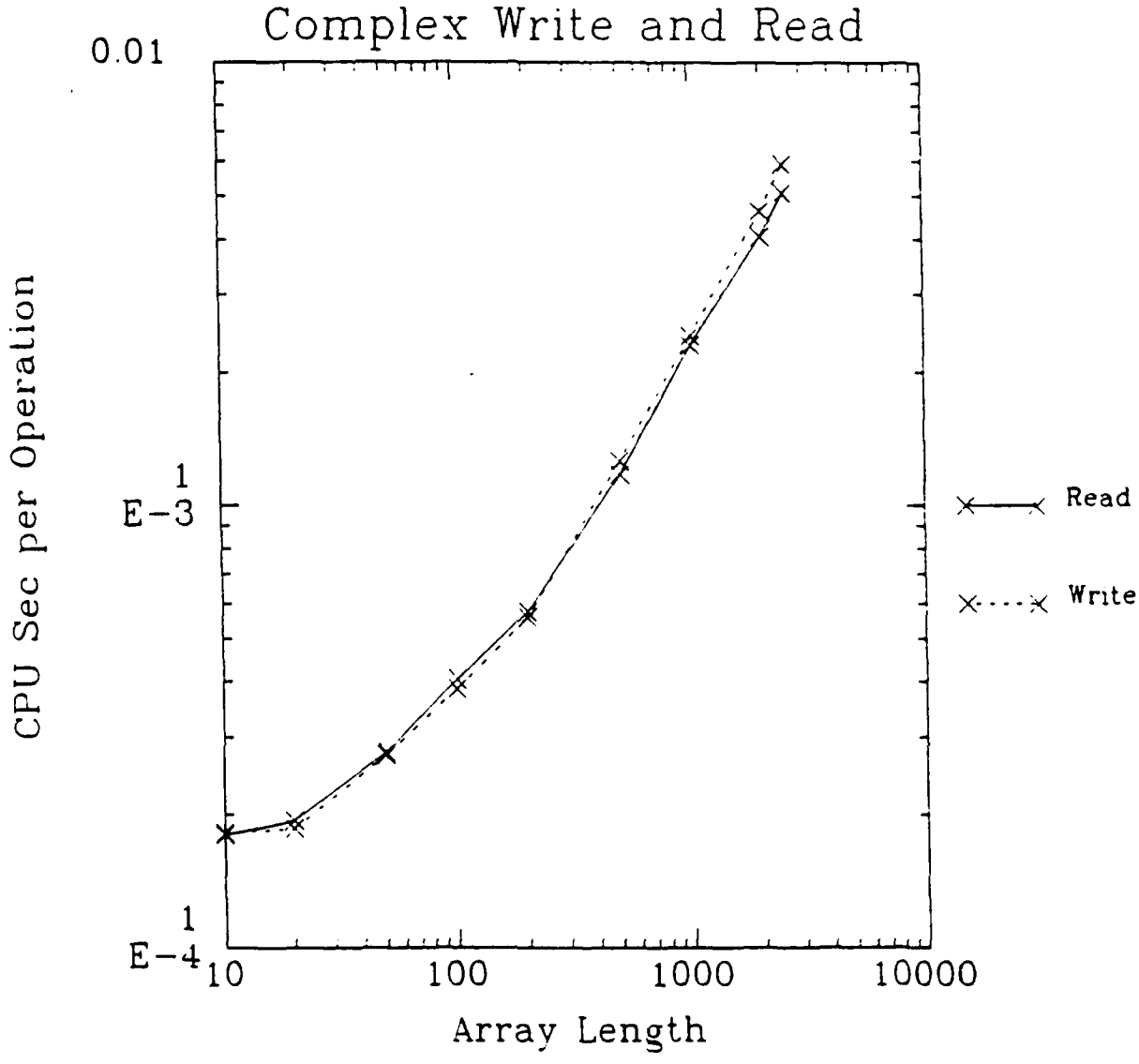

FIGURE 1. *CPU time (in seconds) to read and write a complex array of a given length to a direct access disk file on the Vax* 8650.

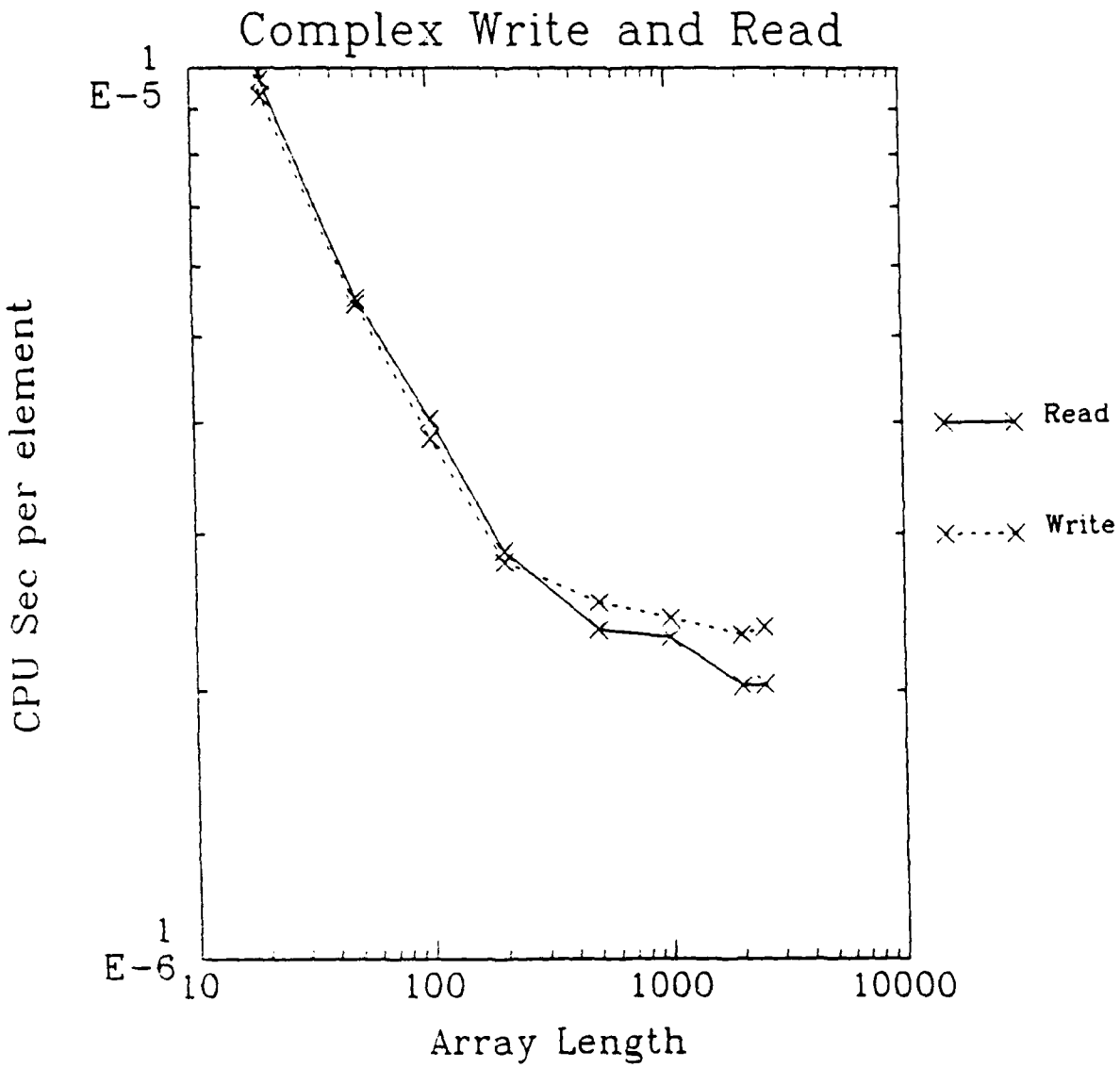

FIGURE 2. CPU time per array element (in seconds) to read and write a complex array of a given length to a direct access disk file on the Vax 8650. The same data as Figure 1, except that the time has been divided by the number of elements in the array.

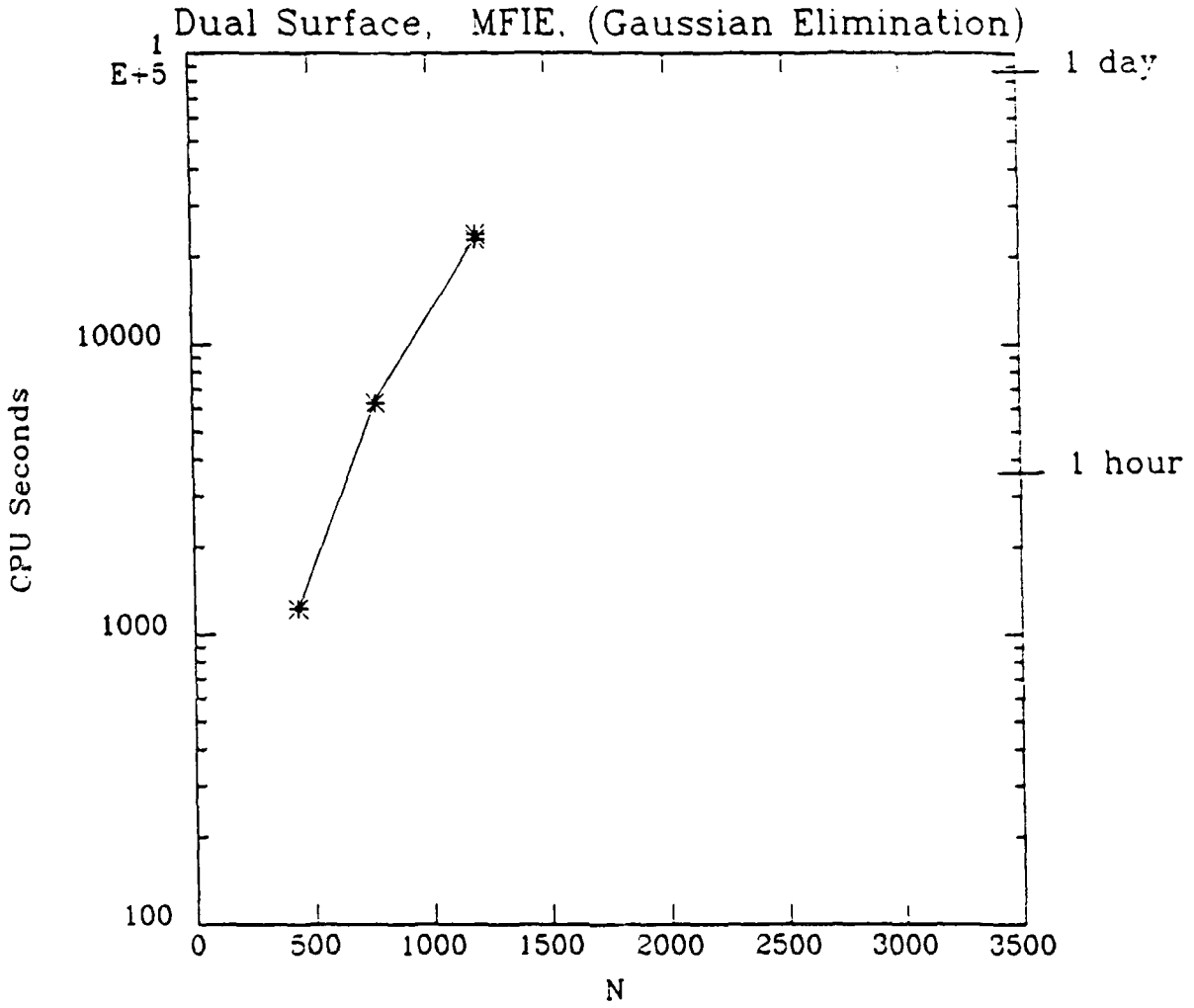

FIGURE 3. CPU time (in seconds) to run, on the Vax 8650, the program which solves scattering from a cube, versus the size of the scatter matrix (N). The program used the "dual-surface" magnetic-field integral equations and solved the matrix by Gaussian elimination.

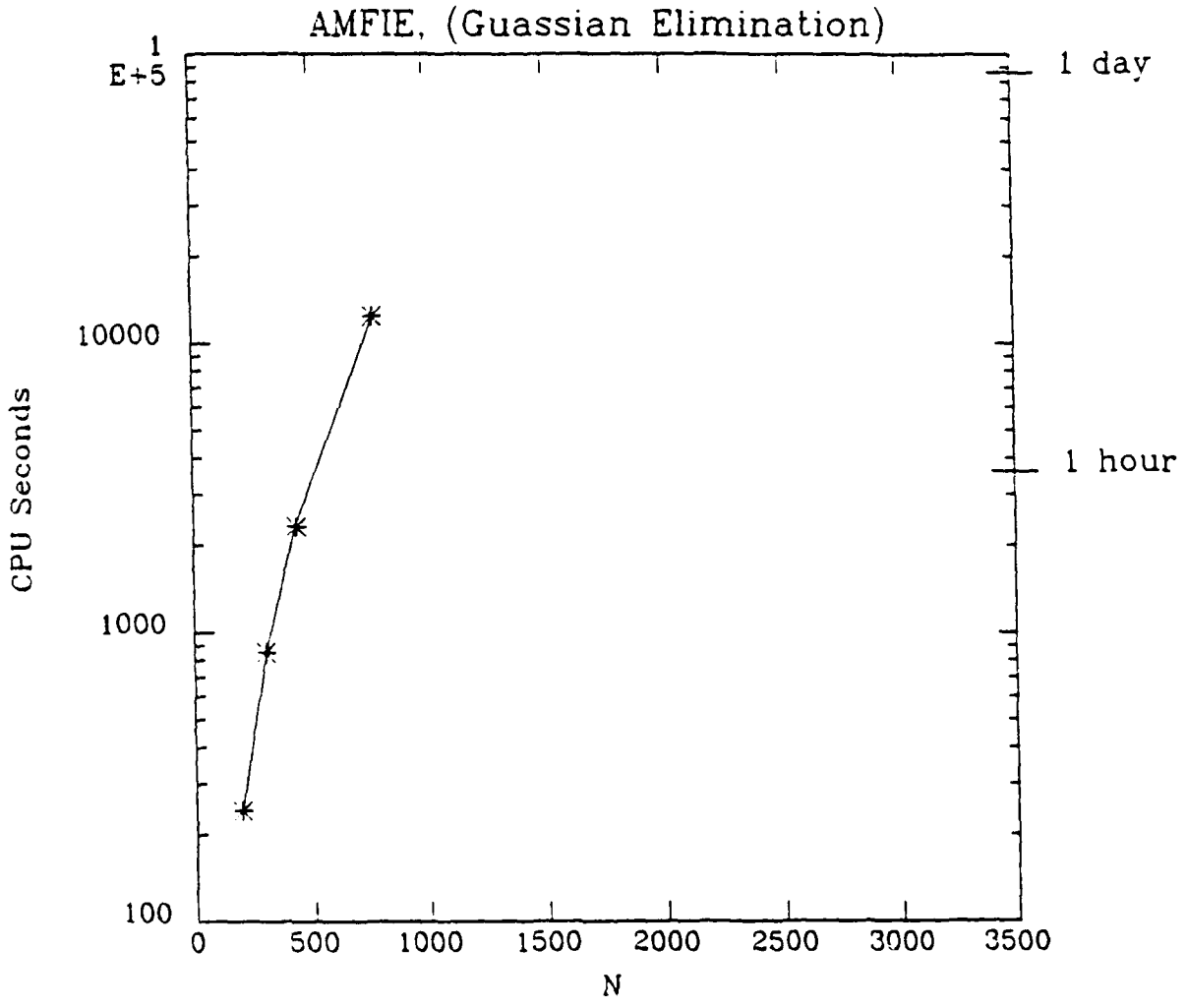

FIGURE 4. CPU time (in seconds) to run, on the Vax 8650, the program which solves scattering from a cube, versus the size of the scatter matrix (N). The program used the augmented magnetic-field integral equations and solved the matrix by Gaussian elimination.

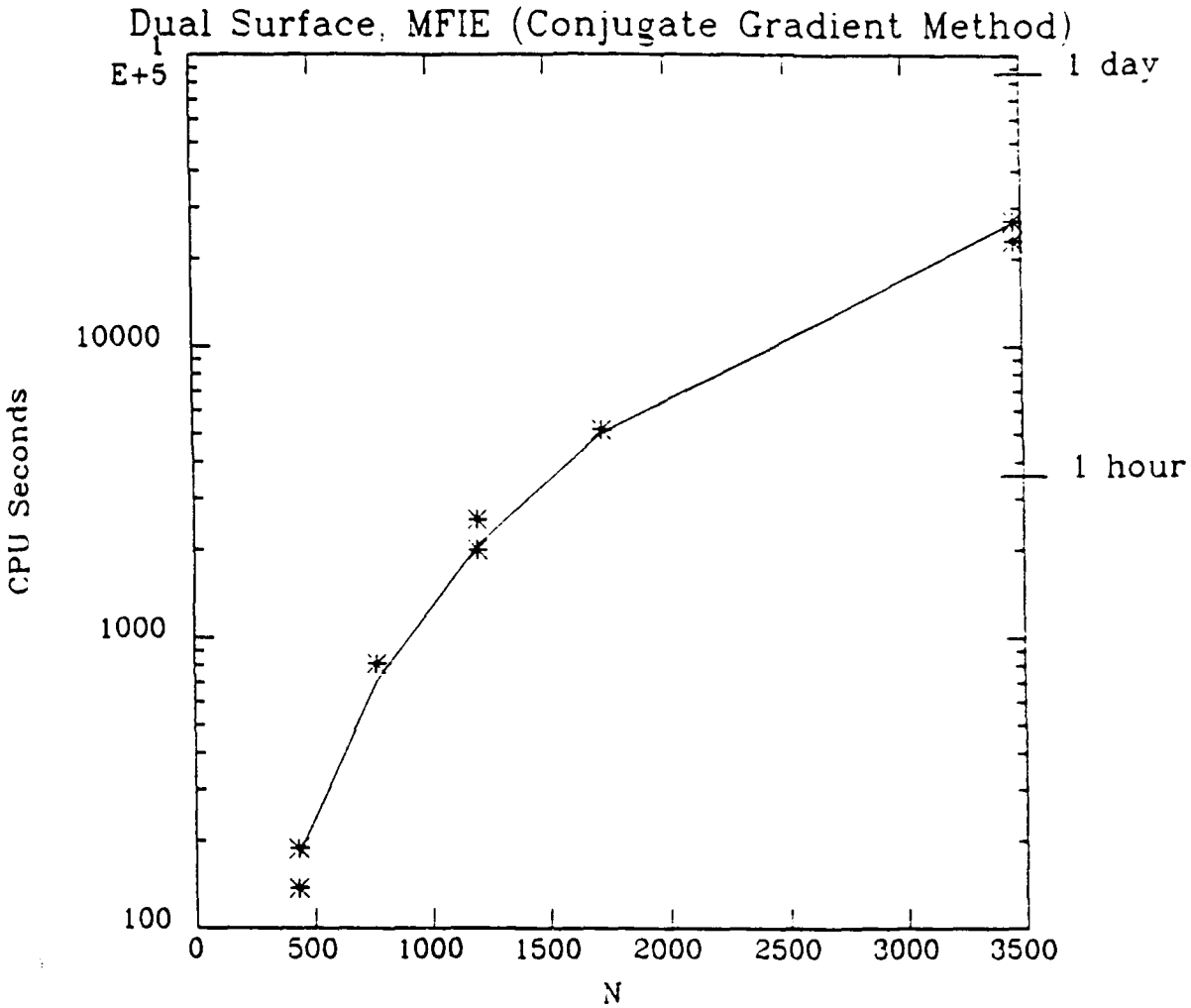

FIGURE 5. CPU time (in seconds) to run, on the Vax 8650, the program which solves scattering from a cube, versus the size of the scatter matrix (N). The program used the "dual-surface" magnetic-field integral equations and solved the matrix by the conjugate gradient method.

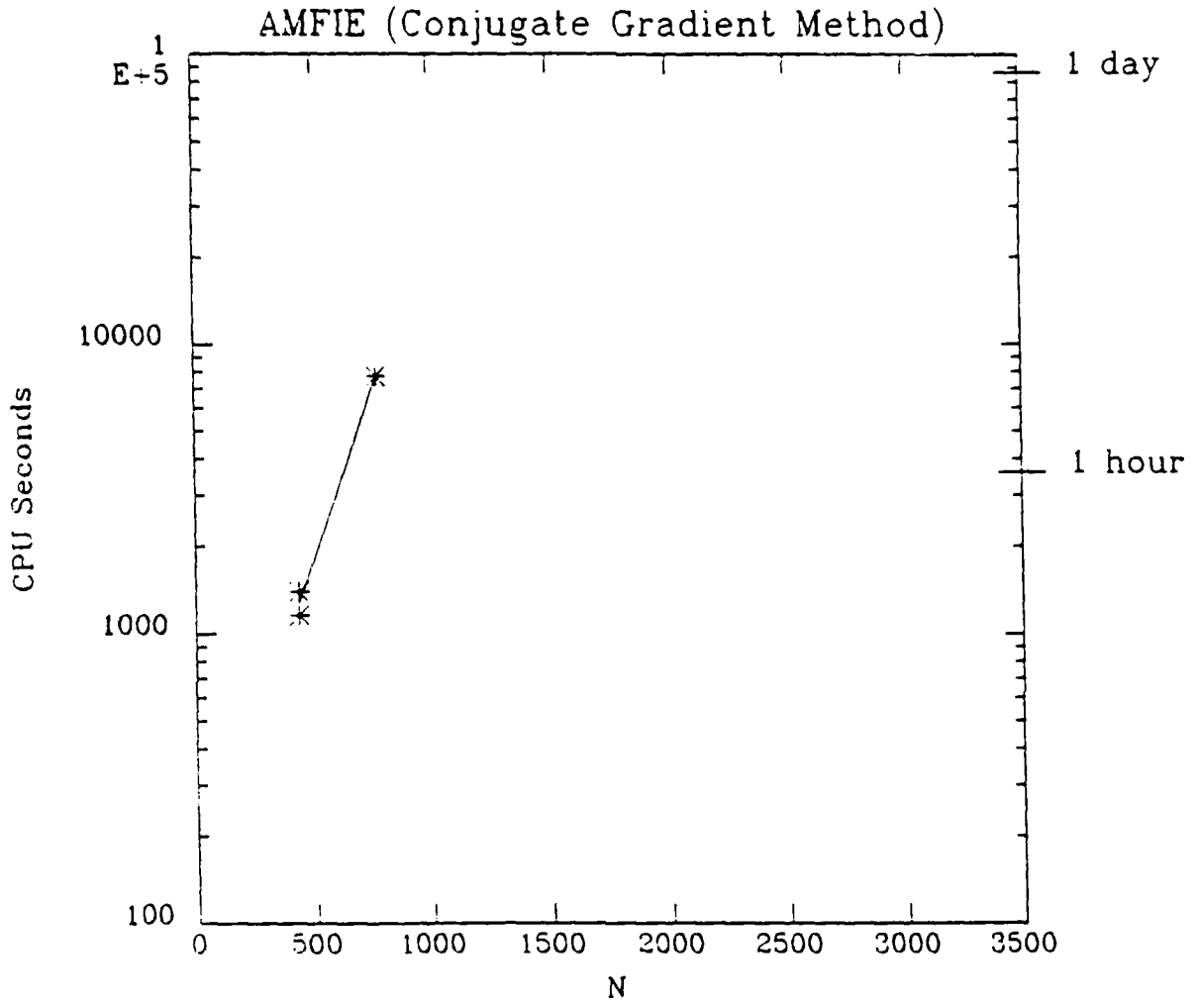

FIGURE 6. CPU time (in seconds) to run, on the Vax 8650, the program which solves scattering from a cube, versus the size of the scatter matrix (N). The program used the augmented magnetic-field integral equations and solved the matrix by the conjugate gradient method.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control, Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. The areas of technical competence include communications, command and control, battle management, information processing, surveillance sensors, intelligence data collection and handling, solid state sciences, electromagnetics, and propagation, and electronic, maintainability, and compatibility.