ELECTROMAGNETIC FIELD MODELING ON SUPERCOMPUTERS

Nathan Ida

Abstract

The relatively new area of problem solving on supercomputers offers exciting new possibilities for numerical solution of 3-D electromagnetic field problems. The solution of systems of equations with tens of thousands of unknowns and bandwidths in the thousands is all but impossible on conventional computers. The availability of new vector supercomputers such as the CRAY 1 and the CYBER 205 brings both hope and new challenges. The faster machine cycle, vector instructions and parallel computing combined with advanced I/O devices promises to speed up the solution of large problems considerably. On the other hand, efficient programming on such machines is far from being trivial and in most cases, one has to compromise portability of the code. In addition, these machines require a front end computer and are not interactive, complicating and lenghtening the program development process.

This work presents an overview of the work done to date on electromagnetic field modeling on supercomputers. Some data on field solution on an attached array processor is also included since array processors are a special class of supercomputers and for comparison purposes.

Using some very large eddy current problems the various aspects of computing are presented together with results showing dramatic improvements in solution times. The problem of code portability, vector complex computation and program conversion are also addressed and, finally, the need for new algorithms, specifically designed for vector computing is presented.

Introduction

The very existence of supercomputers is in response to the need for large scale computation in many diverse areas. Accordingly, since their appearance some significant problems in numerical weather modeling[1], wave propagation[2], petroleum reservoir simulation[3] and statistical modeling in semiconductor physics[4], to name but a few were addressed. Other areas of research have greatly benefited from this powerful tool[5-7].

At the same time only limited emphasis was given to electromagnetic field modeling although, it seems, this should be a very appropriate application. Modeling of fields is unique in that it requires a large number of mesh points with relatively dense meshes. In addition, especially in eddy current problems, skin depth considerations and infinite boundaries must be taken into account increasing the number of unknowns in the solution region considerably. The need for accurate 3-D solutions in moving coil applications[8] compounds the problem to the point where only an increase by orders of magnitude in computer performance can render the solution feasible.

The author is with the Electrical Engineering Department, The University of Akron, Akron, Ohio 44325.

Considerable amount of ground work has been done in other fields or is progressing. Thus, for example, large structural analysis programs such as NASTRAN have been converted to run on vector computers[9]. Some of the algorithms crucial to efficient implementation of large scale field programs such as the Gauss elimination and the Conjugate Gradient algorithms have been vectorized[10-12]. Although no one can expect these algorithms to be optimal at such an early stage, the basis for cost-effective solution of very large problems exists. There also exist whole libraries with vectorized routines that should allow exploration of this field with little user effort[13,14].

Vector Versus Scalar Computing

In conventional architectures, each instruction is fetched from memory and then executed for a limited number of operands. Since in most computer architectures a single memory address per instruction is allowed, operations on an array can be viewed as repetitively applying the same instruction to each element of the array. In contrast, a vector operation is the application of a single instruction to all the elements of an array (or pairs of arrays). The issuance of such an instruction performs the vector operation, specifies the starting addresses of the vectors involved and their length. For example, consider the following:

```
      DO 100 I=1,N
      DO 100 J=1,N
      C(I,J)=A(J,I)+B(J,I)
100   CONTINUE
```

In a scalar mode, this results in N*N instructions, each calculating one element of C. In vector notation, on the CYBER 205, this can be written as

```
C(1,1;N*N) = A(1,1;N*N) + B(1,1;N*N)
```

and only a single instruction is necessary to perform this calculation.

Similarly, vector intrinsic functions can be used. The following code segments perform identical operations

```
SCALAR      DO 100 I =1,N
            A(I)=SQRT(B(I))+C(I)
        100 CONTINUE

VECTOR      A(1;N)=VSQRT(B(1;N))+C(1;N)
```

where VSQRT is the equivalent vector intrinsic function to the scalar function SQRT and performs the same operation on all the elements of array B in a single instruction.

On the CYBER 205 one can assign (dynamically) each of the arrays A, B and C as descriptors

0018-9464/85/1100-2490$01.00©1985 IEEE

```
DESCRIPTOR AD,BD,CD
```

followed by appropriate ASSIGN statements

```
ASSIGN AD, A(1;0)
ASSIGN BD, B(1;0)
ASSIGN CD, C(1;0)
```

and the vector statement becomes:

```
CD=AD+BD
```

The operations above resulted in a new vector of the same length as the input vectors. The vectorization process can be extended to situations where a single scalar is calculated such as in the case of a dot product:

```
      A=0
      DO 100 I=1,42300
      A=A+X(I)*Y(I)
  100 CONTINUE
```

This can be vectorized as

```
K=42300
A=Q8SDOT(X(1;K),Y(1;K))
```

where the vector function Q8SDOT was used to issue the necessary machine instructions.

The efficiency of vector computers relies on the parallel execution of pairwise operations that make up a vector operation. The CYBER 205 employs pipelining to achieve this. In a pipelined structure, each pipeline operates on different elements in the data. A pipelined adder, for example, is divided into segments, each of which performs specific operations in each machine cycle. After a start-up time require to "fill the pipeline", this adder will produce a sum for each machine cycle. This structure, immediately suggests that maximum efficiency on such a machine is achieved for long vectors, where the start-up time adds little to the overall execution time. An interesting question in this regard is: what is a "long vector"?. The answer obviously depends on the operation to be performed. The start-up time on the CYBER 205 is about 50 machine cycles (depending on the operation performed) and any vector of length less than about 10 should be considered less efficient than equivalent scalar operations. The CRAY 1 in contrast, has a start-up time of less than 10 cycles for most operations and therefore is faster for short vectors.

Vector computers also have scalar processors for unvectorizable portions of the code. These processors by themselves are significantly faster than other sequential computers and allow a relatively painless transfer of programs from scalar to vector environments. Automatic vectorization by the compiler adds to this aspect by recognizing the simplest of the loop structures as vector operations and issuing appropriate machine instructions.

Program Conversion

Since in most cases it is more feasible to convert an existing program to run on a vector computer rather than to develop a new one, the effort involved in such an undertaking must be considered. Since vector computers have scalar processors it is possible to proceed in three steps.

1. Conversion of the program to run on the vector machine with little or no explicit vectorization.
2. Explicit vectorization. Parts of the program may require significant changes in order to take advantage of the available vector instructions.
3. Algorithm changes. This step is in fact equivalent to developing special routines for vector computing.

Step 1: No Explicit Vectorization

In this step, only a minimum of changes to the original programs are usually required, depending on the scalar computer on which the program was originally written. This may include changes to PROGRAM statements, OPEN and CLOSE statements and some control statements used to transfer files from and to the front end computer. Thus, for example, in transferring programs from a CYBER 720 to the CYBER 205, only the PROGRAM statement and some Hollerith strings needed changes while transfer of VAX programs also required changes in OPEN and CLOSE statements. Overall, these are very minor changes.

At this point, the program will run more or less as a scalar program on a vector machine. The CYBER 205 does vectorize some of the loops in the program. Only the simplest of the loops (loops without any kind of branching or recursiveness) are automatically vectorized by the compiler but this, and the optimization by the scalar processor are sufficient to significantly reduce the solution time.

This relatively simple and all-important step in the program conversion must be performed whether any further vectorization will be undertaken or not. Similarly, although other machines will require different changes, these are in essence of the type described above unless the language used on the supercomputer is different than the language of the program. For example, the adaptation of a VAX 11/780 program to run on an FPS-164 attached array processor required only minor changes in OPEN statements.

Step 2: Explicit Vectorization

After the program is running and an initial assessment of its performance has been done, one can proceed to vectorize those parts of the program that the compiler cannot vectorize including, perhaps, treatment of loops that vectorize under unsafe conditions (variable limits). Minor changes in algorithms are also appropriate at this stage provided they do not require rewriting and extensive testing. An important part in program modification is the treatment of I/O statements, memory allocation and calls to subroutines. Also, as a rule, operations should be performed columnwise rather than the more conventional rowwise operations.

The first step in the process is to recognize those parts of the program that will benefit most from vectorization. In it's simplest form this decision can be based on previous experience or on specific knowledge of the algorithms involved. In a finite element program, the elimination and backsubstitution routines are the primary targets for vectorization. If specific software exists, capable of accurately timing subroutines or loops in the program (i.e. the SPY routine on the CYBER 205) better decisions can be made. As an example to this aspect consider Table 1. An eddy current program was timed using the SPY routine on the CYBER 205 before and after vectorization. The element assembly routine (ASSEMB), elimination (UDU1) and backsubstitution (UDU2) are listed separately. It is clear that most of the vectorization effort should be directed at subroutine UDU1

with UDU2 as the next priority. Any improvement in other routines would be marginal at this stage. Inspection of the timing after vectorization reveals a more uniform time distribution between the various routines indicating better performance. As a second step, other routines may be vectorized to improve performance.

Step 3: Algorithmic changes

This last step in the program conversion is less defined than the previous two. It is also one in which maximum performance can be achieved. The main trust should be in changes in algorithms or perhaps, use of different algorithms that are better suited to vector computing. As in the previous stage, it would be natural to begin with those parts in the program that use large portions of time (total and/or CPU). The elimination and backsubstitution algorithms, or in more general terms, the solution algorithm are certainly the prime candidates for this type of program conversion.

Thus, for example, the program used to obtain the results in Table 3 uses a SKYLINE storage algorithm and a segmented elimination and backsubstitution method. The skyline algorithm is very efficient but it requires extensive searching and branching - two highly nonvectorizable operations. A simpler algorithm, such as using a constant bandwidth storage scheme has a larger storage overhead but the vectors are of constant length and less searching is required. This may prove to be more efficient overall.

Similarly, the Gauss elimination algorithm may not be the best approach. There are more than ample indications that the Conjugate Gradient algorithm in one form or another holds the key to truly efficient and fast solution programs. As they are today there are still many difficulties in vectorization of these highly recursive algorithms but some have been vectorized with significant improvements in timing.

Some Results

To put the foregoing arguments in perspective consider first Table 2. It represents the initial conversion of two programs to run on an FPS-164 array processor connected to a (dedicated) VAX 11/780. Improvements of more than a factor of 4.5 were realized for the larger problem with no explicit vectorization and only the absolutely necessary changes in the program itself. The improvement factor can be increased up to 15 (for the particular processor used) using explicit vecorization[15]. This however is too low a factor to have a significant impact on field computation.

Table 3 represents the conversion of two programs from scalar environment (VAX 11/780 and/or CYBER 720) to a vector environment (CYBER 205) without any explicit vectorization. The first program is a mesh generator (20,000 triangular elements, 10,201 nodes). The second is a 3-D program running an eddy current problem with 735 equations(245 nodes, 144 elements) and a bandwidth of 174. Four resolution steps were performed. Clearly, the improvement in the second case is significantly higher indicating a problem with relatively long vectors and extensive matrix operations. More significant and dramatic are the improvements possible with explicit vectorization of the code. Table 4 shows the performance of a large eddy current problem on a VAX 11/780 as compared to a CYBER 205 (columns No. 1 and No. 2). If one compares the total time, the vectorized code runs faster by a factor of more than 160. The true advantage of vector machines can be seen in column No. 3 of Table 4. This problem could not be run at all on the VAX 11/780 or the CYBER 720. These solution times are quite remarkable considering the fact that only limited explicit vectorization was performed, without any changes in algorithms.

It is quite clear that further improvements can be obtained, perhaps by an order of magnitude, through algorithmic changes.

Program Portability

As a rule, vectorized programs are not portable, not even from one vector computer to another. Programs written on a CYBER 205 will not run on a CRAY and vice versa, although, depending on the extent of vectorization, the changes necessary may not be extensive. Similarly, because of the fact that the programs must be written, edited and modified on a front end computer, these may not even be portable between sites with identical vector machines. The main impediment to portability is the fact that the vector instructions are not part of a standard language. The new FORTRAN 8X proposal promises to change this somewhat but the problem of portability should remain for the foreseeable future. The user must weigh this aspect before engaging in vectorization of programs.

Another problem, not directly related to portability is the fact that almost none of the vector operations (on the CYBER 205) accepts complex variables. This is important in eddy current applications since it requires appropriate changes in the elimination/backsubstitution routines to take advantage of vector operations.

Future Developments

The use of supercomputers in numerical modeling of electromagnetic field problems is, in a sense, inevitable. Before these computers become well accepted it is necessary to explore fully the existing algorithms and develop new ones as required. The well known, trusted algorithms such as Gauss elimination should be reevaluated. Frontal methods should, most certainly be abandoned since their memory optimization interferes strongly with vectorization due to extensive branching and extensive I/O operations. The field of iterative algorithms, used only sparingly in electromagnetic field problems should in fact offer considerable improvements in solution times despite the complications in vectorization.

Conclusions

Conversion of existing programs from scalar to vector environments offers new possibilities in 3-D field analysis. Larger problems can be solved at reduced solution times and costs. The amount of time invested in the conversion process determines the improvement obtained, but even for negligible vectorization an improvement of more than an order of magnitude in solution time is achieved. Further vectorization can improve this factor considerably but, to take full advantage of the machine's features it is necessary to make changes in algorithms and to consider new algorithms. In particular, iterative algorithms should be investigated.

References

[1] D. L. Williamson and P. N. Swarztrauber, A numerical weather prediction model - computational aspects on the CRAY-1, Proceedings of the IEEE, Vol. 72, No.1, January 1984, pp. 56-67.

[2] O. G. Johnson, Three-dimensional wave equation computation on vector computers, Proceedings of the IEEE, Vol. 72, No. 1, Jan. 1984, pp. 90-95.

[3] R. P. Kendal, J. s. Nolen and P. L. Stanat, The impact of vector processors on petroleum reservoir simulation, Proceedings of the IEEE, Vol. 72, No. 1, January 1984, pp. 85-89.

[4] K.C. Bowler and G. S. Pawley, Molecular dynamics and Monte Carlo simulations in solid-state and elementary particle physics, Proceedings of the IEEE, Vol. 72, No. 1, January 1984, pp. 42-54.

[5] W. Fichter, L. W. Nagel, B. Penumali, W. P. Petersen and J. L. D'arcy, "The impact of supercomputers on IC technology development and design," Proceedings of the IEEE, Vol. 72, No. 1, January 1984, pp. 96-112.

[6] D. Fuss and G. G. Tull, "Centralized supercomputer support for magnetic fussion energy research," Proceedings of the IEEE, Vol. 72, No. 1, January 1984, pp. 32-41.

[7] N. Schmidt and O. G. Johnson, "A vector elastic model for the CYBER 205," Supercomputer Application Symposium, Oct. 31 - Nov. 1, 1984, (Purdue University and Control Data Corporation).

[8] N. Ida, A finite element model for 3-D eddy current NDT phenomena, Submitted for Publication in IEEE Transactions on Magnetics.

[9] J. F. Gloudeman, The impact of supercomputers on finite element analysis, Proceedings of the IEEE, Vol. 72, No. 1, January 1984, pp. 80-84.

[10] J. Ortega and E. Poole, "Incomplete Choleski Conjugate Gradient onthe CYBER 203/205," Supercomputer Applications Symposium, Oct. 31 - Nov. 1, 1984, (Purdue University and Control Data Corporation)

[11] D. Barkai, K. J. M. Moriarty and C. Rebbi, "A modified conjugate gradient solver for very large systems," Supercomputer Applications Symposium, Oct. 31 - Nov. 1, 1984, (Purdue University and Control Data Corporation)

[12] R. Schieber, " A new implementation of sparse Gaussian Elimination," ACM Transactions on Mathematical Software, Vol. 8, No. 1, September 1982, pp. 256-276.

[13] Control Data Corporation, "MAGEV Library Utility," 1980.

[14] J. J. Dongarra, C. B. Moller and G. W. Stuart, "LINPAK Users' Guide," SIAM, Philadelphia, PA. 1979.

[15] R. C. Young, "Application of a floating point system APL190L array processor to finite element analysis," Report prepared for the Department of Energy, Contract DE-AT03-76ET35301, April 1982.

Table 1. CPU time distribution between the various subroutines of an eddy current program before and after explicit vectorization.

Subroutine	Before Vectorization	After Vectorization
ASSEMB	4.2%	18.0%
UDU1	67.5%	47.2%
UDU2	21.8%	18.9%
All others	6.5%	16.9%

Table 2. Comparison of performance of a VAX 11/780 to an FPS-164 Array processor attached to a VAX 11/780 for three different eddy current problems.

	VAX Alone CPU	FPS-164 CPU	VAX/FPS-164 CPU Ratio
Problem 1	675.41 Sec.	185.96 Sec.	3.632
Problem 2	8 Hrs. 19 Min.	1 Hr. 50 Min.	4.53
Problem 3	3 Hrs. 19 Min.	39 Min.	3.56

Problem 1, 729 equations, bandwidth=276.
Problem 2, 4074 equations, bandwidth=318.
Problem 3, As problem 2 but only backsubstitution was performed for 10 probe positions (no elimination).

Table 3. Comparison of performance between scalar and vector computers.

	VAX 11/780	CYBER 720	CYBER 205	IMPROV. RATIO VAX/205	IMPROV. RATIO 720/205
PROB. 1	581 sec.	-	109.47 sec.	5.3	-
PROB. 2	306.38 sec.	587.275 sec.	17.22 sec.	17.8	34.0

TABLE 4. Performance of the CYBER 205 and the VAX 11/780 for two large eddy current problems.

	12,513 variables VAX 11/780	12,513 variables CYBER 205	19,716 variables CYBER 205
Mesh Gen.	4 Min. 57 Sec.	16 Sec.	28 Sec.
Elimination	12 Hrs. 20 Min.		
Backsubstitut. (one step)	22 Min.		
Backsubstit. *	8 Hrs. 48 Min.		
Total	21 Hrs. 13 Min.	29 Min. 54 Sec.	66 Min. 16 Sec.
Clock Time	Appr. 82 Hrs.	30 Min. 10 Sec.	66 Min. 44 Sec.

* 24 backsubstitution steps. The third column includes 31 backsubstitution steps.