PARALLEL IMPLEMENTATION OF FIELD SOLUTION ALGORITHMS

Nathan Ida and Jian-She Wang

Electrical Engineering Department
The University of Akron
Akron, Ohio 44325 U.S.A.

ABSTRACT

The application of a parallel computer for the solution of field problems is described. The initial implementation included a parallel solver as well as all other stages of finite element codes for 2-D and 3-D magnetostatic and eddy current problems. The equation solver is described in detail as the part that will benefit most from parallelization. The Massively Parallel Processor (MPP) was used and solution times almost three orders of magnitude faster than for sequential computers were obtained for dense matrices, with little explicit parallelization.

INTRODUCTION

The advantage of a parallel computer is in its potential ability to solve large problems in realistic solution times. As the improvements in speed of single processor computers approach intrinsic limits, parallel processing becomes more appealing. The performance of such machines is not known with any accuracy. The ideal performance depends on the number of processors and their speed. Upper and lower limits on computation speed can always be obtained. This does not take into account a variety of considerations like I/O and other degradation factors. In the ideal case, the speedup achieved through parallel processing is equal to the number of processors [1]. A variety of factors influence the performance to reduce the speedup considerably. Among these factors, the competition of processors for hardware and the interaction between the parallel processes are the most important [2]. Obviously, the algorithm to be executed has significant influence on the performance. Ideally, the algorithm has intrinsic parallelism such that there is no need to idle processors. In reality, this is not the case and there are always serial operations to be performed. In the limit, when no parallel operation can be performed, the parallel processor is used as a serial computer.

The type of systems considered here are those arising from the application of the finite element method to engineering applications. The finite element method is particularly computationally intensive, yet its various parts are either intrinsically parallel or can be parallelized. By using a parallel processor, it is conceivable that considerably faster solution times can be achieved or, alternatively, larger problems can be solved.

Since the element definition, matrix assembly and the postprocessing phases of a finite element code are essentially parallel operations, little is needed for efficient implementation. The solution of the system of equations is different. It normally consumes most of the execution time. Parallelization of this stage will influence the outcome more than any other stage in the finite element solution. For simplicity, the Gauss elimination process will be described here although the implementation of two iterative methods (an SOR and an ICCG algorithm) is currently in progress. The Massively Parallel Processor (MPP) was used in this work and although reference is made to the MPP, the results presented are typical of parallel computers.

THE MASSIVELY PARALLEL PROCESSOR

The MPP [3,4] has 16,384 individual processors arranged in an 128*128 array. Each processing element (PE) has a local associated memory (1K) and communicates with its nearest neighbors to the North, South, East and West. The machine has a large staging memory (32 Mbyte) and is controlled by a front-end (sequential) computer. The primary power of the machine is its ability to process a plane of data (128*128 by 1 bit) in a single machine cycle (100 nsec). The fact that a plane consists of a single bit also allows flexible word lengths. The MPP is typical of parallel computers in its flexibility to change the outline of the basic array arrangement. Currently, the MPP is programmable in Parallel PASCAL. For the solution of linear systems , the two most important aspects related to the MPP are the number of memory planes in the ARray Unit (ARU) and the size of the staging memory available for use. The ARU contains 1024 bit planes of memory. Only bit planes from 0 to 973 are usable for programming. This limits the number of 128*128 real arrays (32 bit) in the ARU to 30. The staging memory is limited to 512 real arrays (128*128).

The parallel Pascal callable I/O procedures can transfer only one 128*128 array in or out of the ARU at any one time. This makes it necessary for any array larger than 128*128 to be blocked into sub-arrays of 128*128. Blocking of a 512*512 array is given in Fig. 1. As with any computer, a number of intrinsic functions are available. Extensive use has been made of row and column broadcasting functions and masked operations.

A PARALLEL GAUSS ELIMINATION ALGORITHM

The elimination procedure consists of operations which are intrinsically parallel. During every step of the elimination, operations are done on a whole row at a time. This is the basic property used to implement the Gaussian elimination on the parallel computer.

The ARU contains arrays of 128*128 bit-planes. Any array operation is highly parallelized. For example, an array multiplication operation involves all elements multiplied simultaneously. Initially, the discussion will focus on an array 128*128 in size as this is the smallest size the MPP can handle. Thus, reference to an array means an 128*128 array. For a system of equations of the form $[A]\{X\}=\{B\}$, the parallel implementation of the Gauss elimination algorithm begins by first loading the matrix A into one array and the right hand side into the first column of a second array. All other coefficients in this array are zero.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} & b_2 \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} & b_3 \\ & \vdots & \vdots & & \vdots & \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} & b_n \end{bmatrix} \quad (1)$$

0018-9464/88/0100-0291$01.00©1988 IEEE

$$\begin{bmatrix} b_1 & 0 & 0 & \cdots & 0 \\ b_2 & 0 & 0 & \cdots & 0 \\ b_3 & 0 & 0 & \cdots & 0 \\ \cdot & \cdot & \cdot & & \cdot \\ b_n & 0 & 0 & & 0 \end{bmatrix} \qquad (2)$$

To eliminate the first column of the matrix, a pivot row array and a pivot element array are created by using fast row and column-broadcasting intrinsic routines.

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ \cdot & \cdot & \cdot & & \cdot \\ a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \end{bmatrix} \qquad (3)$$

$$\begin{bmatrix} a_{11} & a_{11} & a_{11} & \cdots & a_{11} \\ a_{11} & a_{11} & a_{11} & \cdots & a_{11} \\ a_{11} & a_{11} & a_{11} & \cdots & a_{11} \\ \cdot & \cdot & \cdot & & \cdot \\ a_{11} & a_{11} & a_{11} & \cdots & a_{11} \end{bmatrix} \qquad (4)$$

A pivot column array is created from the array in (1) as

$$\begin{bmatrix} a_{11} & a_{11} & a_{11} & \cdots & a_{11} \\ a_{21} & a_{21} & a_{21} & \cdots & a_{21} \\ a_{31} & a_{31} & a_{31} & \cdots & a_{31} \\ \cdot & \cdot & \cdot & & \cdot \\ a_{n1} & a_{n1} & a_{n1} & \cdots & a_{n1} \end{bmatrix} \qquad (5)$$

The pivot column array in (5) is then divided by the pivot element array in (4) and then multiplied by the pivot row array in (3) to create a modifier array

$$\begin{bmatrix} a_{11}a_{11}/a_{11} & a_{11}a_{12}/a_{11} & a_{11}a_{13}/a_{11} & \cdots & a_{11}a_{1n}/a_{11} \\ a_{21}a_{11}/a_{11} & a_{21}a_{12}/a_{11} & a_{21}a_{13}/a_{11} & \cdots & a_{21}a_{1n}/a_{11} \\ a_{31}a_{11}/a_{11} & a_{31}a_{12}/a_{11} & a_{31}a_{13}/a_{11} & \cdots & a_{31}a_{1n}/a_{11} \\ \cdot & \cdot & \cdot & & \cdot \\ a_{n1}a_{11}/a_{11} & a_{n1}a_{12}/a_{11} & a_{n1}a_{13}/a_{11} & \cdots & a_{n1}a_{1n}/a_{11} \end{bmatrix} \qquad (6)$$

This array, with the exception of the pivot row, is subtracted from the original array in (1). The result is the following new coefficient array

$$\begin{bmatrix} a_{11} & + & a_{12} & + & a_{13} & + & \cdots & + & a_{1n} \\ & & a'_{22} & + & a'_{23} & + & \cdots & + & a'_{2n} \\ & & a'_{32} & + & a'_{33} & + & \cdots & + & a'_{3n} \\ & & \cdot & & \cdot & & & & \cdot \\ & & a'_{n2} & + & a'_{n3} & + & \cdots & + & a'_{nn} \end{bmatrix} \qquad (7)$$

Using one parallel array multiplication, one divide and one add operation the first column of (6) was eliminated.

The modification of the right hand side during elimination is similar. The array in (5) is first divided by the pivot element in (4) and then multiplied by the right hand pivot element array in (2) to generate a right hand side modifier array

$$\begin{bmatrix} a_{11}b_1/a_{11} & 0 & 0 & \cdots & 0 \\ a_{21}b_1/a_{11} & 0 & 0 & \cdots & 0 \\ a_{31}b_1/a_{11} & 0 & 0 & \cdots & 0 \\ \cdot & \cdot & \cdot & & \cdot \\ a_{n1}b_1/a_{11} & 0 & 0 & \cdots & 0 \end{bmatrix} \qquad (8)$$

This is then subtracted from the original right-hand side array in (2) to obtain the new right-hand side array. As previously, the pivot row is not modified.

$$\begin{bmatrix} b_1 & 0 & 0 & \cdots & 0 \\ b_2-a_{21}b_1/a_{11} & 0 & 0 & \cdots & 0 \\ b_3-a_{31}b_1/a_{11} & 0 & 0 & \cdots & 0 \\ \cdot & \cdot & \cdot & & \cdot \\ b_n-a_{n1}b_1/a_{11} & 0 & 0 & \cdots & 0 \end{bmatrix} \qquad (9)$$

The modification of the right-hand side in every elimination step needs one parallel array addition, and one multiplication operation.

After n-1 elimination steps, the original coefficient matrix is reduced to an upper triangular system. The whole process requires 2(n-1) add operations, 2(n-1) multiply operations and (n-1) divide operations, where n=128.

The parallel algorithm for Gaussian elimination of an 128*128 order system of linear equations can be written in pseudo code as

```
DO ROW_INDEX = 0 TO 127
BEGIN
    P ARRAY <-- PIVOT ROW FROM ORIGINAL ARRAY;
    Q ARRAY <-- PIVOT ELEMENT FROM P ARRAY;
    R ARRAY <-- PIVOT COLUMN FROM ORIGINAL ARRAY;
    R ARRAY <-- R ARRAY / Q ARRAY;
    P ARRAY <-- P ARRAY * R ARRAY;
    ORIGINAL ARRAY <-- ORIGINAL ARRAY - P ARRAY;
    Q ARRAY <-- PIVOT RIGHT-HAND ELEMENT FROM
                RIGHT-HAND SIDE ARRAY;
    R ARRAY <-- Q ARRAY * Q ARRAY;
    RIGHT-HAND ARRAY <-- RIGHT-HAND ARRAY - R ARRAY;
END;
```

In this case, in addition to 2 data arrays, one stores the coefficient matrix, and the right-hand side. Three more real arrays P, Q, R (32 bit planes each) and one boolean array (1 bit plane) are needed as scratch space.

BACKSUBSTITUTION

The solution of the triangular system obtained by elimination can be easily performed on a sequential machine but is more complicated on a parallel computer. Handling of the summation operation necessary for backsubstitution is highly inefficient because of its sequential nature. A SUM function is available but it is about 20 times slower than a multiply operation and acts on the whole 128*128 array.

Instead, another algorithm is used. It is outlined as:

$$x_i = b_i/a_{ii}, \qquad b_k = b_k - a_{ki}x_i \qquad (10)$$

where i=n, n-1, ..., 1 and k=i-1, i-2, ..., 1. This algorithm states that, once an unknown is backsubstituted, the upper triangular system is reduced in order by one by elimination of a column of coefficients associated with that unknown, and modifying the right hand side. First, a pivot column array is created as (ith step)

$$\begin{bmatrix} a_{1i} & a_{1i} & a_{1i} & \cdots & a_{1i} \\ a_{2i} & a_{2i} & a_{2i} & \cdots & a_{2i} \\ a_{3i} & a_{3i} & a_{3i} & \cdots & a_{3i} \\ \cdot & \cdot & \cdot & & \\ a_{ii} & a_{ii} & a_{ii} & \cdots & a_{ii} \\ 0 & 0 & 0 & \cdots & 0 \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \qquad (11)$$

A pivot element array is created as

$$\begin{bmatrix} a_{ii} & a_{ii} & a_{ii} & \cdots & a_{ii} \\ a_{ii} & a_{ii} & a_{ii} & \cdots & a_{ii} \\ \cdot & \cdot & \cdot & & \cdot \\ a_{ii} & a_{ii} & a_{ii} & \cdots & a_{ii} \end{bmatrix} \qquad (12)$$

The right hand side is divided by the pivot element array (masked operation) to solve the ith unknown

$$\begin{bmatrix} b_1 & 0 & 0 & \cdots & 0 \\ b_2 & 0 & 0 & \cdots & 0 \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ x_i & 0 & 0 & \cdots & 0 \\ \cdot & \cdot & \cdot & & \cdot \\ x_n & 0 & 0 & & 0 \end{bmatrix} \quad (13)$$

A right hand side pivot element is created

$$\begin{bmatrix} x_i & 0 & 0 & \cdots & 0 \\ x_i & 0 & 0 & \cdots & 0 \\ x_i & 0 & 0 & \cdots & 0 \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ x_i & 0 & 0 & \cdots & 0 \end{bmatrix} \quad (14)$$

The multiplication of the pivot column array by the array in (14) results in a modifier array with nonzero terms in the first column

$$\begin{bmatrix} a_{1i}x_i & 0 & 0 & \cdots & 0 \\ a_{2i}x_i & 0 & 0 & \cdots & 0 \\ a_{3i}x_i & 0 & 0 & \cdots & 0 \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ a_{ii}x_i & 0 & 0 & \cdots & \cdot \\ 0 & 0 & 0 & \cdots & 0 \\ \cdot & \cdot & \cdot & \cdots & \cdot \\ 0 & 0 & 0 & \cdots & 0 \end{bmatrix} \quad (15)$$

This is now subtracted from the original right hand side array in (2). After n=128 steps, the right hand side array contains the n unknowns in its first column. The solution process needs (n) adds, (n-1) multiplies and (n) divide operations. This algorithm can be written in pseudo code as follows:

```
DO ROW_INDEX = 127 DOWNTO 0
  BEGIN
    P ARRAY <-- PIVOT COLUMN FROM ORIGINAL ARRAY;
    Q ARRAY <-- PIVOT ELEMENT FROM P ARRAY;
    RIGHT HAND ARRAY <-- RIGHT HAND ARRAY / Q ARRAY;
    R ARRAY <-- PIVOT UNKNOWN FROM RIGHT HAND ARRAY;
    P ARRAY <-- P ARRAY * R ARRAY;
    RIGHT HAND ARRAY <-- RIGHT HAND ARRAY - P ARRAY;
END;
```

BLOCK GAUSS ELIMINATION METHOD

For the solution of any system with order larger than 128, its coefficient matrix must be blocked in subarrays of 128*128. A 512 order problem is chosen here since this is the largest one that the ARU can handle. In Fig. 1, the 512*512 coefficient matrix is blocked into 4*4 subarrays. The right-hand vector is stored in the first column of 4 corresponding subarrays although this is not necessary as one 128*128 array can store one 16384 column vector. The parallel PASCAL code previously developed for a 128 order problem is extended to the current problem with the scanning order shown in Fig. 1.

RESULTS

The solution times for Gaussian elimination of an 128*128 and 512*512 real arrays (resulting from the finite element application to a 3-D magnetostatic problem) on the MPP are summarized and compared with the solution times for the same arrays on a Microvax-II. This is shown in Table 1. The speedup is obviously larger for larger matrices and reaches a maximum for a 512*512 matrix (the largest matrix that can fit in the ARU). For this type of matrices, advantage can be taken of the fact that the matrix has a limited bandwidth. The shaded areas in Fig. 2 show the sub-arrays (128*128) that could be used while solving a 512*512 banded matrix with a semi-bandwidth of 128 or less. The solution of finite element matrices resulting from eddy current applications is similar except for the need for solution in complex variables. The parallel Pascal language does not support complex notation but this can be done explicitly. Table 3 summarizes results for two eddy current problems using a parallel Gauss-Jordan and Gauss Elimination methods. In general, the ratio between complex and real variables solution is about 4, reflecting the fact that the number of operations in complex variables is roughly four times larger. This table also shows that for small arrays, the Gauss-Jordan method is faster. For arrays larger than 512*512, Gauss elimination is faster. Partitioning of large systems in blocks of 128*128 allows solution of systems as large as the memory of the machine will accommodate. For a 32 Mbyte machine this is about 2800*2800 or any other combination (i.e. a 32000*128 banded matrix will also fit). The solution times in Table 4 show the efficiency of parallel solution in absolute terms and in comparison with a CRAY X-MP/48.

The experience gained here not only shows that the solution of large systems of equations is possible and faster than that possible on serial machines but also that there is an alternative between using large arrays or smaller arrays with large memory. The structure of the MPP allows the user to fit a matrix as large as the memory of the stager. Clearly, if the matrix can be fitted in the ARU , the solution will be faster than for the case where parts of the matrix need to be retrieved from the stager or from the front end computer.

PARALLELIZATION OF OTHER ALGORITHMS

For efficient implementation finite element codes on parallel computers, other parts of the code need to be implemented. These include the assembly stage (including integration) and postprocessing calculation such as calculation of global quantities, current distributions, fields, etc. Because all of these can be computed on an elemental basis, they can be considered to be essentially parallel. The approach taken here was to partition the finite element mesh into parts that are exclusive of each other and process the elements in each part separately. This is shown schematically in Fig. 3. The elements marked do not share any common nodes and therefore can be processed in parallel. Repeating this eight times will complete the processing of the mesh in Fig. 1. Obviously, some meshes may be more complicated and may requires more steps. A similar idea can be used for the postprocessing stage of the finite element program.

CONCLUSIONS

The results presented in this work show that a parallel computer can be used for finite element analysis with relative ease and great benefit. It is particularly suited to the solution of problems which can fit in the ARU. The speedup obtained compared to serial computers is in the hundreds. The key to efficient solvers on parallel machines is parallelization of algorithms and the use of a large enough array so as to minimize I/O. Improved scheduling and parallelization of this and other algorithms should result in very fast , efficient solution methods for 3-D field problems.

REFERENCES

[1] K. Hwang, F. A. Briggs, "Computer architecture and Parallel Processing", McGraw-Hill Book Company, New York, 1964.

[2] T. Axerold, " Comparing the performance of parallel computers",Compcon, spring 1985.

[3] K.E. Batcher, "Design of a massively parallel processor ", IEEE Transactions on Computers, Vol. C-29, No.9, pp. 837-840, September 1980.

[4] K. E. Batcher, "Architecture of the MPP", IEEE Computer Society on Computer Architecture for Pattern Analysis and Image Database Management Proceedings, pp. 170-174, October 1983.

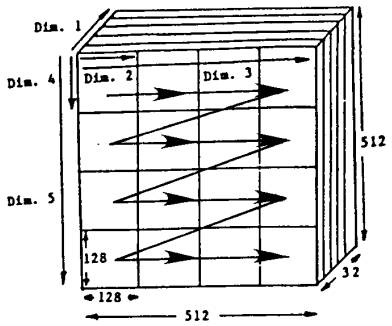

Figure 1. Blocking and dimensioning of a 512*512 array into sub-arrays 128*128 in size.

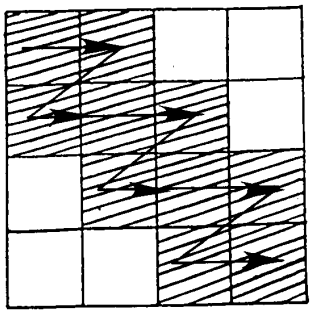

Figure 2. Possible scanning of a banded matrix with bandwidth of 128 or less.

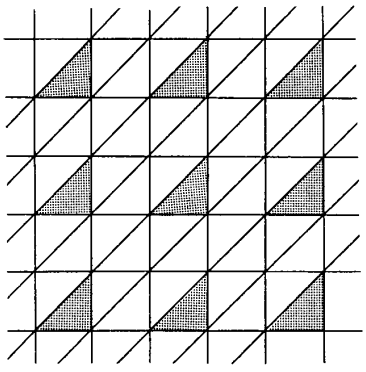

Figure 3. Parallel processing of elemental quantities.

Table 1. Solution times on the Microvax II and the MPP for a 128*128 and a 512*512 system.

N	MicroVax II	MPP	Speedup
128: Elimination	7.57 sec	77.95 ms	97
Solution	120 ms	49.3 ms	2.4
Total	7.59 sec	129.27 ms	59
512: Elimination	52 min 45 sec	1.231 sec	2572
Solution	2.01 sec	256.15 ms	5.6
Total	52 min 47 sec	1.588 sec	1994

Table 2. Solution times for various systems on the MPP. All have a semibandwidth of 128.

N	Elimination	Solution	Total Time
1024	1.556sec	0.651sec	2.206sec
2048	3.300sec	1.371sec	4.671sec
3072	5.015sec	2.077sec	7.093sec
4096	6.731sec	2.784sec	9.515sec
8192	13.594sec	5.611sec	19.205sec
12288	20.458sec	8.438sec	28.896sec
16384	27.321sec	11.264sec	38.264sec

Table 3. Comparison of real and complex variable solutions on the MPP with Gauss-Jordan and Gauss Elimination methods.

N	Real System	Complex System	Ratio
Gauss-Jordan			
128: Elimination	77.88 ms	334.89 ms	4.3
Solution	6.43 ms	12.43 ms	1.9
Total	84.31 ms	347.32 ms	4.1
256: Elimination	343.68 ms	1513.80 ms	4.4
Solution	12.28 ms	24.88 ms	2.0
Total	355.96 ms	1538.68 ms	4.3
Gauss Elim.			
128: Elimination	77.95 ms	334.96 ms	4.3
Solution	49.30 ms	244.58 ms	4.96
Total	128.27 ms	579.55 ms	4.52
256: Elimination	277.76 ms	1196.30 ms	4.3
Solution	125.61 ms	576.48 ms	4.59
Total	403.37 ms	1772.77 ms	4.4

Table 4. Solution of large, real, dense matrices on the MPP.

Size	Elim.	Solution	Total-MPP	CRAY X-MP
1024x1024	7.809sec	1.327sec	9.136sec	36.523sec
2048x2048	51.353sec	4.638sec	55.990sec	-