SOLUTION OF LARGE LINEAR SYSTEMS OF EQUATIONS ON THE MASSIVELY PARALLEL PROCESSOR

Nathan Ida Kapila Udawatta
Electrical Engineering Department
The University of Akron
Akron, Ohio 44325

ABSTRACT

The Massively Parallel Processor, has been designed as a special machine for specific applications in image processing. As a parallel machine, with a large number of processor that can be reconfigured in different combinations it is also applicable to other problems that require a large number of processors. This work investigates the solution of linear systems of equations on the MPP. The solution times achieved are compared to those obtained with a serial machine and the performance of the MPP is discussed.

INTRODUCTION

The advantage of a parallel computer is in its potential ability to solve large problems in realistic solution times. In particular, as the improvements in speed of single processor computers approach intrinsic limits, the appeal of parallel processing becomes more significant. Yet, beyond the fact that some problems have little or no natural parallelism, the performance of such machines is not known with any accuracy. The ideal performance is of course well defined. It depends on the number of processors and their speed. Thus, 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. Thus, 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 a drastic 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. Again, in the limit, when no parallel operation can be performed, the parallel processor is used as a serial computer.

Estimating the performance of a parallel processor for any particular type of problems is not a trivial process. In many cases this cannot be done without actually solving the problem and then evaluating the performance. Thus, the need to evaluate a computer's performance on well documented problems or benchmarks becomes extremely important, particularly with parallel processors. With all this, one can safely assume that some degree of parallelism does exist in most algorithms and therefore an improvement in solution time compared with serial machines can be realized.

The work presented here attempts to establish the usefulness of a parallel processor, the Massively Parallel Processor (MPP), for the solution of systems of linear equations. No attempt is made to either optimize code or to establish exact performance figures. Such an attempt is beyond the scope of this paper and would require considerably more work. Instead, the emphasis is on a particular simple algorithm and on the comparison of performance with a serial computer (the Microvax II). The type of systems solved are those arising from the application of the finite element method to engineering applications. The finite element method is particularly computationally intensive. By using a parallel processor, it is conceivable that considerably faster solution times can be achieved or, alternatively, larger problems can be solved.

The solution method chosen for evaluation is the Gauss elimination method. It is used as representative to direct solution methods and because for the majority of practical applications it is used almost exclusively. The results will therefore be useful for implementation of other similar methods like the Gauss-Jordan or the Choleski decomposition methods.

GAUSS ELIMINATION

Consider the following system of n equations with n unknowns:

$$a_{11}X_1+a_{12}X_2+a_{13}X_3+\ldots\ldots+a_{1n}X_n=C_1 \quad (1a)$$

$$a_{21}X_1+a_{22}X_2+a_{23}X_3+\ldots\ldots+a_{2n}X_n=C_2 \quad (1b)$$

$$a_{31}X_1+a_{32}X_2+a_{33}X_3+\ldots\ldots+a_{3n}X_n=C_3 \quad (1c)$$

$$\vdots$$

$$a_{n1}X_1+a_{n2}X_2+a_{n3}X_3+\ldots\ldots+a_{nn}X_n=C_n \quad (1n)$$

The system in Eq. (1) is assumed to be symmetric, positive definite for the purpose of this work although none of these requirements is necessary in general. The elimination is done in the following order:

Equation 1a is divided by its diagonal coefficient to obtain

$$x_1+a_{12}/a_{11}X_2+a_{13}/a_{11}X_3+\ldots \ldots+a_{1n}/a_{11}X_n=C_1/a_{11} \quad (2)$$

Equation (2) is multiplied by the first coefficient of (1b)

$$a_{21}X_1+a_{21}a_{12}/a_{11}X_2+\ldots\ldots \ldots+a_{21}a_{2n}/a_{11}X_n=C_1a_{21}/a_{11} \quad (3)$$

Eq. (3) is subtracted from Eq. (1b) to eliminate the coefficient of X_1 from Eq. (1b). In the next step, Eq. (2) is multiplied by the first coefficient in Eq. (1c). Subtraction as previously results in elimination of the coefficient of X_1 in Eq. (1c). Repeating this for all the remaining coefficients in the first column results in the following system:

$$a_{11}X_1+ a_{12}X_2+ a_{13}X_3+\ldots\ldots+ a_{1n}X_n =C_1 \quad (4a)$$

$$a'_{22}X_2+a'_{23}X_3+\ldots\ldots+a'_{2n}X_n =C'_2 \quad (4b)$$

$$a'_{32}X_2+a'_{33}X_3+\ldots\ldots+a'_{3n}X_n =C'_3 \quad (4c)$$

$$\vdots$$

$$a'_{n2}X_2+a'_{n3}X_3+\ldots\ldots+a'_{nn}X_n =C'_n \quad (4n)$$

In this set, all coefficients were altered except for those in the first row. It is important to note that the operations are done on a whole row at a time, a property which will be exploited later for parallel calculation.

The elimination proceeds by repeating the whole process starting with the second equation. After n-1 such elimination steps, the original system is reduced to an upper triangular system:

$$a_{11}X_1 + a_{12}X_2 + a_{13}X_3 + \cdots\cdots + a_{1n}X_n = C_1 \quad (5a)$$

$$a'_{22}X_2 + a'_{23}X_3 + \cdots\cdots + a'_{2n}X_n = C'_2 \quad (5b)$$

$$a''_{33}X_3 + \cdots\cdots + a''_{3n}X_n = C'_3 \quad (5c)$$

$$\vdots$$

$$a^n_{nn}X_n = C^n_n \quad (5n)$$

The elimination step, which is done in place (i.e. in the same locations the original matrix resides), is followed by a backsubstitution step. This starts by calculating X_n

$$X_n = C^n_n / a^n_{nn} \quad (6a)$$

The rest of the unknowns are calculated as:

$$X_i = [C_i - \sum_{j=i+1}^{n} a_{ij}X_j]/a_{ii} \quad (6b)$$

where $i = n-1, n-2, \ldots\ldots, 1$ and C_i are the modified right hand sides in Eq. (5).

THE MASSIVELY PARALLEL PROCESSOR

For the solution of linear systems, the two most important aspects related to the MPP are the number of memory planes in the ARU and the size of the staging memory available for use. Although the ARU contains 1024 bit planes of memory, the programmer can use only bit planes from 0 to 973. Bit planes from 974 to 1023 are reserved for use by system software (Control and Debug). This limits the number of 128*128 real arrays (32 bit floating point arrays) in the ARU to 30. Without taking into consideration any necessary scratch arrays, the capacity of the ARU is limited to one 640*640 real array or an equivalent size array. In practical terms, since scratch arrays are needed, the limit is lower. A matrix of 512*512 is the limit if increments in size of 128*128 are to be used.

The staging memory has a capacity of 32 Megabytes. This limits the number of stored 128*128 real arrays in the stager to 512.

It is important to note that 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 by assigning the larger array two more dimensions. Blocking of a 512*512 array is given in Fig. 1.

Other aspects of programming on the MPP are not discussed here although they are necessary for implementation. These can be found in a variety of references [3,4,9-15].

IMPLEMENTATION ON THE MPP

Since the minimum data array that can reside in the ARU is 128*128, as a first step the Gaussian elimination was implemented for an array of this size. In addition to the data array, three more real arrays (32 bit) and one boolean array were used as scratch space for the Gaussian elimination. That is, 97 bit planes were used as scratch space.

The algorithm used for Gaussian elimination of an 128*128 array can be written in pseudo code as

```
DO UNTIL # ROWS = 128
BEGIN
NEW  ARRAY <-- PIVOT ROW FROM  ORIGINAL
               ARRAY;
PIVOT ELEMENT ARRAY <-- NEW ARRAY;
MULTIPLIER ARRAY <-- ORIGINAL ARRAY;
NEW  ARRAY <-- NEW ARRAY/PIVOT  ELEMENT
               ARRAY;
NEW   ARRAY  <-- NEW   ARRAY*MULTIPLIER
                 ARRAY;
ORIGINAL ARRAY <-- ORIGINAL ARRAY - NEW
                   ARRAY;
END;
```

The parallel Pascal code for Gaussian elimination of an 128*128 array was used as the building block and was extended for larger arrays. An array of 512*512 was chosen since this is the maximum array that can reside in the ARU. Scanning of the 512*512 array, by a block of 128*128, is given in Fig. 2.

A theoretical calculation of timing for an array of 1024*1024 was also done. The 1024*1024 array was blocked into four sub-arrays of 512*512 and at any given time one 512*512 sub-array resided in the ARU. The actual time needed for loading/unloading the ARU with a 512*512 real array is 590 ms. The time needed to eliminate 2 columns from a 512*512 real array is 10 ms. A real array of 512*512 occupies 512 bit planes. This leaves the ARU with 14 real arrays to be used as scratch space. Of these, one plane is necessary to use as a boolean array for the mask plane. This makes it possible to save 14 "coefficient" arrays in the ARU. By saving 14 coefficient arrays it is possible to eliminate 2 columns from any of the four 512*512 sub-arrays. the Gaussian elimination on the 1024*1024 real array can be written in pseudo code as

```
DO L = 1 To 2
 BEGIN
  DO K = 1 TO 256
   BEGIN
    DO I = L TO 2
     DO J = L TO 2
      BEGIN
       IF L = 1 THEN
        BEGIN
         ELIMINATE 2 COLUMNS FROM
         SUB-ARRAY;
          UNLOAD SUB-ARRAY TO STAGER
          WHILE LOADING NEW SUB-ARRAY
          TO ARU;
         END;
        ELIMINATE 512 COLUMNS FROM
        LAST SUB-ARRAY;
       END;
      END;
     END;
```

RESULTS

The solution times for Gaussian elimination of an 128*128, 512*512 and 1024*1024 real arrays on the MPP are summarized and compared with the solution times for the same arrays on a Microvax-II. The performance of the MPP compared to a serial computer is illustrated in Fig. 3.

Arrays below the size of 27*27 can be solved on the Microvax-II computer and obtain the same performance as on

the MPP. The array size below which it is more economical to use a serial computer, depends on the type of the serial computer.

But, as seen in Table 1 and Fig. 3, the performance of the parallel computer (MPP) improves with the increase of the size of the data array and drops sharply once the data array exceeds the memory space of the ARU. Taking into account the actual times involved, the MPP is much faster than a serial machine.

It is important to note that the algorithms used for the MPP were not optimized, since the study involved only understanding the performance of the MPP in solving large systems of linear equations and not the precise evaluation of the MPP performance. Fig. 3 clearly shows the tendency of the behavior of the MPP. In addition to the algorithms being nonoptimized, no attempt was made to taylor the algorithm to a particular type of problem. For example, consider the matrices generated by finite element modeling which are banded (and, in many cases, symmetric).

For this type of matrices great advantage can be taken of the fact that the matrix has a limited bandwidth. The shaded areas in Fig. 4 show the sub-arrays (128*128) that could be used while solving (scanning) a 512*512 banded matrix with a bandwidth which is less than 128.

The performance can also be changed by using different blocking techniques for larger arrays. It will be interesting to see the performance when a 1024*1024 array is blocked not as four arrays of 512*512, but four 1024*256 arrays. In this way, the optimal blocking of large arrays for particular problems can be chosen.

The experience gained here clearly shows not only 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.

From the results presented above it can be seen that an 1024*1024 array (for example) will solve a 1024*1024 matrix in about 48 ms which means a speedup of about 69,320 against the speedup of 6.6 for an 128*128 array.

At the same time, an 128*128 array with large memory (large enough to contain the matrix) can solve a 1024*1024 matrix in 4.21 s, resulting in a speedup of 823 compared to 6.6 with a 1k memory.

As it stands now, the tendency seems to be towards larger arrays but it will be interesting to study the performance of these machines with larger memories which are by nature less expensive than processors.

CONCLUSIONS

The results presented in this work show that the Massively Parallel Processor 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. Not surprisingly, the performance deteriorates somewhat when parts of the matrix need to be brought in from the stager. The deterioration in performance is quite dramatic when the matrix does not fit into the stager and there is a need to perform considerable

I/O. With all this, the solution is always faster than for serial computers.

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, September 1980.

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

5. P. B. Hansen, "The programming language concurent Pascal", IEEE Transactions on Software Engineering, Vol. SE-1, No. 2, June 1975.

6. "MPP I/O control unit", Goodyear Aerospace Corporation, GER-16679, April 1983.

7. "MPP main control unit", Goodyear Aerospace Corporation, GER-16659, April 1983.

8. "MPP PE control unit", Goodyear Aerospace Corporation, GER-16650, June 1983.

9. "MPP Staging memory", Goodyear Aerospace Corporation, GER-16964, March 1981.

10. "MPP staging memory manager", Goodyear Aerospace Corporation, GER-17062, April 1983.

11. "MPP user's guide", NASA Goddard Space Flight Center, January 1986.

12. "Parallel Pascal callable I/O procedure", NASA Goddard Space Flight Center, January 1986.

13. "Parallel Pascal language reference manual", MUD-210, Version 2.

14. A. P. Reeves, "Parallel Pascal: an extended Pascal for parallel computers", Journal of Parallel and Distributed Computing 1, Vol. 64, 1980.

15. "Theory of the MPP hardware operation", Goodyear Aerospace Corporation, GER-17143, April 1983.

Table 1. Processing speeds of Microvax-II and MPP

Array	Microvax	MPP	Speedup
16*16	10 ms	48.07 ms	0.2
27*27	50 ms	48.07 ms	1.0
128*128	5500 ms	48.07 ms	114
512*512	340 sec	1.272 sec	267
1024*1024	57.76 min	8.12 min	6.6

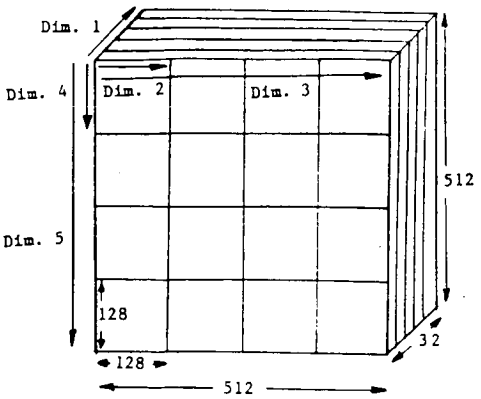

Figure 1. Blocking and dimensioning of 512*512 array into sub-arrays 128*128 insize.

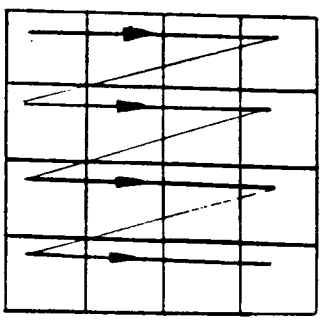

Figure 2. Mode of scanning a 512*512 array in blocks of 128*128.

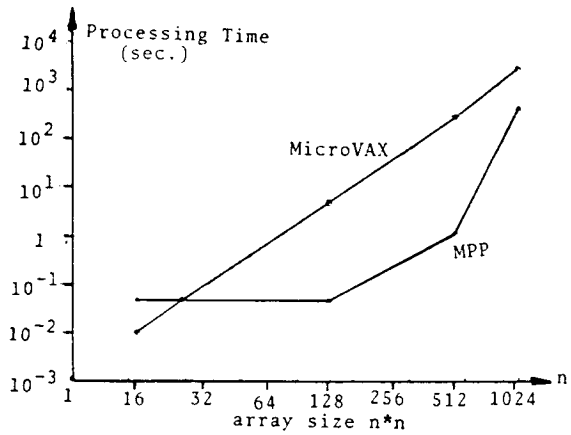

Figure 3. Processing time of the MPP and the Microvax-II vs. array size.

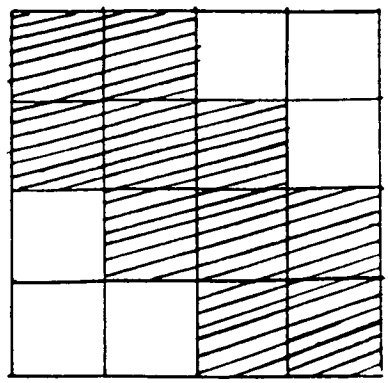

Figure 4. Possible scanning of a banded matrix with bandwidth of 128 or less.