T-MAG/30/5/1/08925

A Parallel Processing Method in Finite Element Analysis using Domain Division

Kenji Iwano
Vlatko Čingoski
Kazufumi Kaneda
Hideo Yamashita

A Parallel Processing Method in Finite Element Analysis using Domain Division

Kenji Iwano, Vlatko Čingoski†, Kazufumi Kaneda, and Hideo Yamashita
Electric Machinery Laboratory, Faculty of Engineering
Hiroshima University, Kagamiyama 1-4
Higashi-hiroshima, 724 JAPAN

ABSTRACT – **Current parallel processing approaches in finite element analysis can be roughly classified into two categories: In the domain method, an analysis region is divided into subdomains and one *CPU* assigned to each subdomain. Alternatively, one may calculate in parallel the matrix and vector products which arise in the process of solving the set of simultaneous equations. In this paper, we present a hybrid of the above two methods. Iteration to bring values on the subdomain boundaries coincide is not necessary, but the idea of domain division is used.**

I. INTRODUCTION

The availability of high-speed and large-memory computers have enticed researchers to tackle solution of ever more complex and higher dimensioned problems. Numerical techniques have also made startling progress, but computation time has not decreased, if anything, an opposite trend is in evidence. As the rate of improvement in the processing speed of *CPUs* is a limited quantity, it is desirable to develop more efficient algorithms. One approach to reducing computation times is parallel processing; several domain division methods have already been presented [1],[2]. However, in such methods, the values computed at subdomain boundaries by individual *CPUs* generally differ, so iterative calculation to bring these values into coincidence is required. This sacrifices much of the potential speed-up, and the processing time depends upon how the analysis region is divided. On the other hand, a method which does not use the subdomain concept has been proposed [3]. In this method, the products and summations for matrix and vector in iterations of the *ICCG* method are executed in parallel. After incomplete Cholesky decomposition, the large coefficient matrix is divided into smaller sub-matrices. In order to calculate small matrix and vector products in parallel, the coefficients far from the diagonal, that is, on the outside of the small matrices, are ignored. Therefore, as the number of *CPUs* increases, the number of ignored coefficients increases, ultimately requiring a larger number of iterations.

Manuscript received November 1, 1993.

†The author is on leave from Electrotechnical Faculty, University "Sv. Kiril i Metodij", Skopje, Macedonia

In this paper, we propose a new parallel method for *ICCG* method. The analysis region is divided into subdomains, and node numbering is carried out on each subdomain. Then, using an *ICCG* method to solve the set of simultaneous equations, the calculation inside each subdomain can be executed in parallel. Parallel calculation of products of $(\mathbf{LU})^{-1}$ and vectors, previously thought to be difficult, is made possible by using the sub-domain concept. This is in addition to parallel calculation of products and of matrix and vector summations. This method allows 95% of the *ICCG* method processing to be parallelized. The main advantage of the new method is that it is not necessary to make coincident the solution on the boundaries, even though the sub-domain method is used. Here, to examine the proposed method, we discuss the two dimensional finite element method using second order triangular elements.

II. PARALLEL COMPUTATION METHOD

Much of the processing time for finite element analysis is consumed in solving a large system of simultaneous equations, $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$. Two methods, the direct method and the iterative one, are generally used. As the dimensions of a system of simultaneous equations become larger, the iterative method is preferable. For this reason *ICCG*, an iterative method, is generally used in finite element analysis. Fig. 1 shows a flow chart of the *ICCG* method; in the first, pre-processing step, the coefficient matrix is incompletely decomposed by the Cholesky method.

The processing time for this step is about 5% of the total processing time. The main process of *ICCG* method is the iterative calculation. This process spends a great deal of time in computing:

1. The products of matrices and vectors,
2. The product $(\mathbf{LU})^{-1} \cdot \mathbf{b}$, where $\mathbf{L}$ and $\mathbf{U}$ are lower and upper triangular matrices composed from an incomplete decomposition of $\mathbf{A}$,
3. The inner vector product and vector summation.

Items 1 and 2 above each consume about 40% of the total processing time of *ICCG*, while 3 consumes about 15%. Thus the iterative calculation process consumes about 95% of total processing time of *ICCG*.

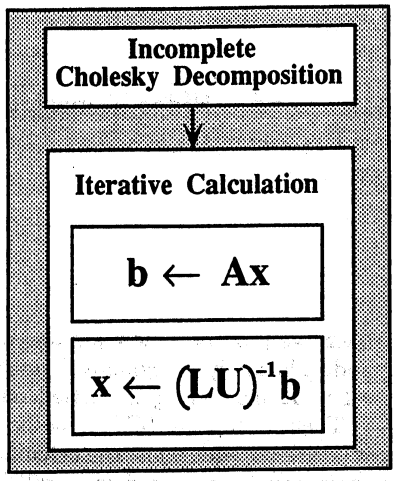


Fig. 1. ICCG Processing Procedure.

Therefore, if these three operations can be executed in parallel on multiple $CPUs$, the total computation time will be reduced. Computing the matrix and vector products in parallel is trivial, because the products of rows of the matrix and the vector are independent. However, handling the operation $(\mathbf{LU})^{-1} \cdot \mathbf{b}$ in parallel is usually very difficult, because this operation is composed of forward elimination and backward substitution. In these processes, the calculation on a row is influenced by the elements in other rows which have already been calculated. Therefore, in the case of a band structured coefficient matrix as shown in Fig. 2(a), parallel computation is impossible. But if the matrix structure is like Fig. 2(b), the elements in submatrices **A** and **B** do not influence each other, and the calculation of these submatrices can be performed in parallel. In finite element analysis, the analysis region is usually composed of only one region, therefore the coefficient matrix does not exhibit a form like Fig. 2(b). But if the region is divided into multiple subdomains, and the node numbering is done first for interior nodes in each subdomain and next for boundary nodes of subdomains, then the coefficient matrix assumes the form shown in Fig. 2(c). In this case, the subdomains are not independent of each other, because they are connected by the boundary nodes of subdomains, but interior nodes in subdomains are independent of each other; therefore, the interior subdomains can be dealt with in parallel. Boundaries must be handled after the parallel operation on the interior subdomains has been finished. If the number of nodes on the boundaries becomes large, parallel processing for the boundary nodes becomes desirable. This is in fact possible: by dividing the nodes on the boundaries into several groups, the submatrix **B** in Fig. 2(c) becomes similar to the global matrix just shown in Fig. 2(c). Parallel processing for the boundaries is then possible, after the parallel processing for the interior subdomains has been finished. Some boundary nodes are not included into any group. The number of these nodes is small, so they can be processed by one CPU.

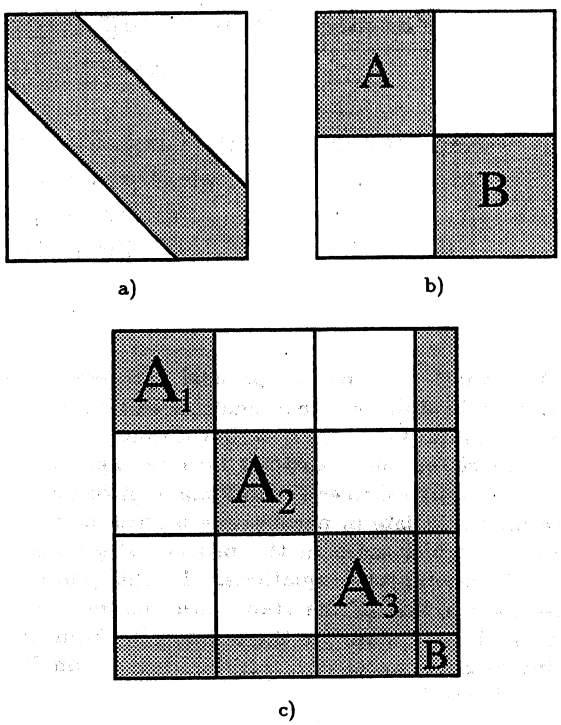


Fig. 2. Coefficient matrix.

Before describing our parallel processing algorithm for the computation $(\mathbf{LU})^{-1} \cdot \mathbf{b}$ we define some terms. If n $CPUs$ are available on a computer, the analysis region is first divided into n subdomains. The set of interior nodes in the i-th subdomain is defined as $\mathbf{A}_i$, $(i = 1 \sim n)$. The nodes on each boundary are divided into n groups, which are defined as $\mathbf{B}_i$, $(i = 1 \sim n)$, and the set of nodes connecting each $\mathbf{B}_i$ is called **C** (see *Fig.* 3).

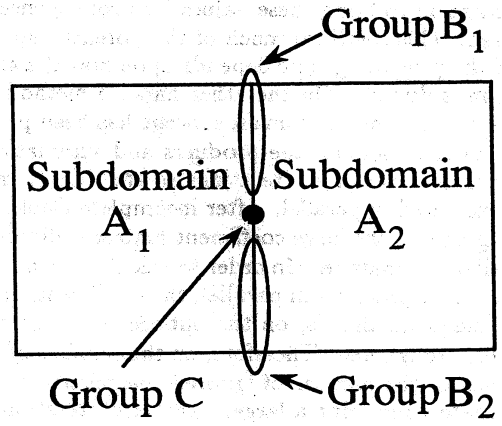


Fig. 3. Grouping of nodes on boundary.

The parallel algorithm for taking the product of $(\mathbf{LU})^{-1}$ and a vector **b** is as follows:

- Forward elimination ($\mathbf{y} = \mathbf{L}^{-1} \cdot \mathbf{b}$)
  - Step 1: Parallel processing of the set of interior nodes $\mathbf{A}_i$, $(i = 1 \sim n)$ for each subdomain on n *CPU*s.
  - Step 2: Parallel processing of the group of boundary nodes $\mathbf{B}_i$, $(i = 1 \sim n)$ on n *CPUs*.
  - Step 3: Processing of the set of nodes $\mathbf{C}$ on one *CPU*.
- Backward substitution ($\mathbf{x} = \mathbf{U}^{-1} \cdot \mathbf{y}$)
  - Step 4: Processing of the set of nodes $\mathbf{C}$ on one *CPU*.
  - Step 5: Parallel processing of the group of boundary nodes $\mathbf{B}_i$, $(i = 1 \sim n)$ on n *CPUs*.
  - Step 6: Parallel processing of the set of interior nodes $\mathbf{A}_i$, $(i = 1 \sim n)$ for each subdomain on n *CPU*s.

In this paper, the matrix/vector product is also calculated in parallel.

III. Numerical Examples

We implemented the proposed method on the *Symmetry S81*, produced by *Sequent Co.*. The *Symmetry* is a *MIMD* type parallel computer containing 14 *CPUs* and 32 *Mbyte* of main memory. In this computer, multiple child processes are automatically generated during execution, and each child process is executed by a separate *CPU*. The memory includes both distributed and shared memory. Distributed memory is accessible from a single *CPU*, and shared memory from all *CPUs*.

The two dimensional magnetostatic models shown in Fig. 4 are used, and the three meshes shown in Table I, with various numbers of nodes and elements, are examined.

Table I
Number of nodes for each model

Case	Model 1	Model 2
A	3033	3514
B	6185	5438
C	9505	9117

The results of test runs are presented in Fig. 5, where the computation time for the *ICCG* method is shown. The value of S_0 is defined by

$$S_0 = \frac{T_n}{T_1} \cdot 100, \tag{1}$$

where T_1 and T_n are processing times on one *CPU* and n *CPUs*, respectively. For comparison, the results using Nour's method [1] are also shown in Fig. 5(a).

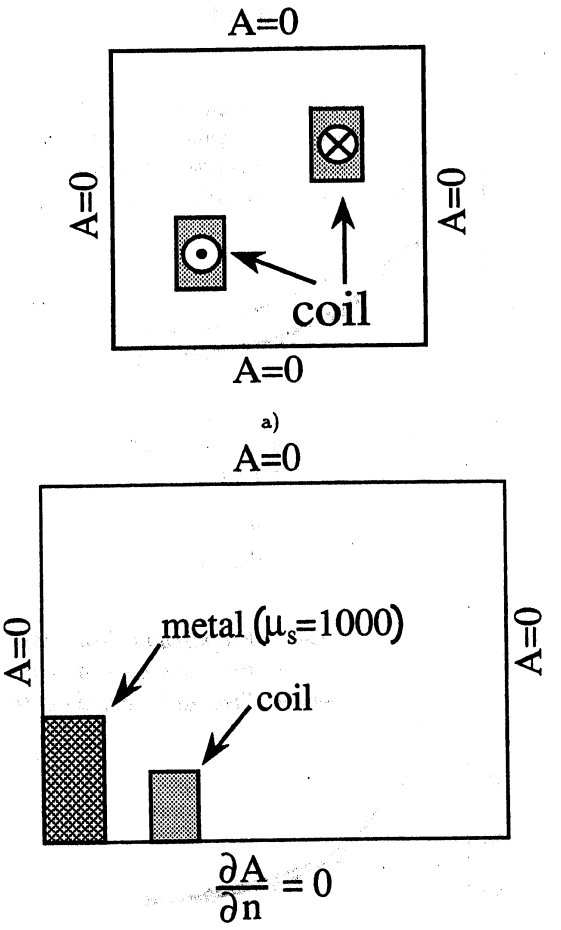


Fig. 4. Models.

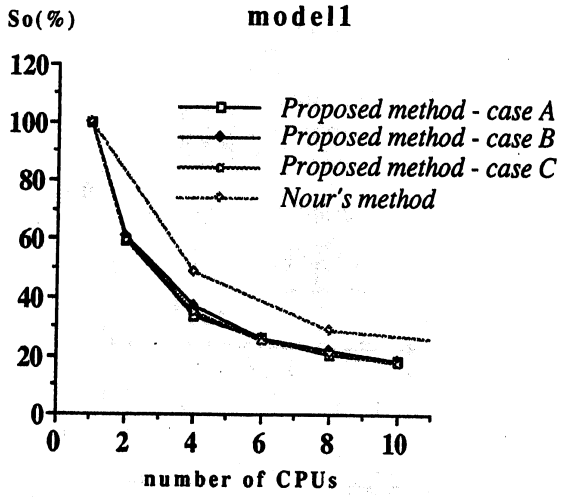
As is obvious from Fig. 5, the value of S_0 decreases as the number of *CPUs* increases. At the same time, the speed-up decreases, because the ratio for incomplete Cholesky decomposition, where parallel processing is not done, increases. As is also obvious from Fig. 5, the ratio of the processing time with n processors to the processing time with one processor (S_0) does not increase with the number of nodes. Our method is superior to Nour's [1].

For reference, Fig. 6 demonstrates that the processing time increases with the number of nodes and elements for a fixed number of *CPUs*. In Fig. 6, the value on the ordinate axis is the processing time in units of the processing time for case *A*.

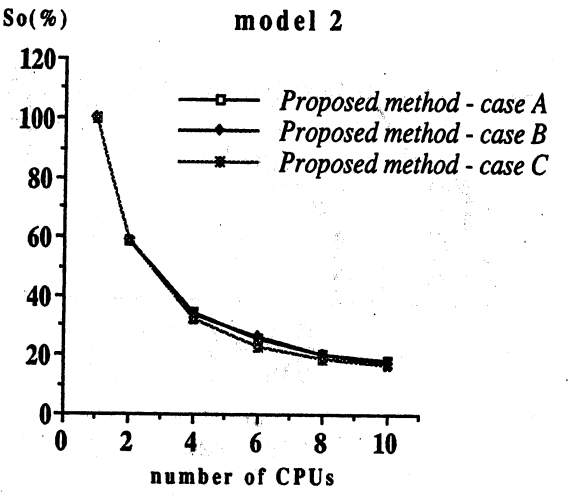
In order to examine the proposed method more in detail, the speed-up ratio given by the following equation is considered.

$$e = \frac{T_1}{T_n \cdot n} \cdot 100 \tag{2}$$

Table II shows the speed-up ratio of the three cases for two models. In Table II, the speed-up ratios are from 85% to 89% when two *CPUs* are used, and from



a)



b)

Fig. 5. Relationship between number of *CPUs* and S_0

70% to 83% when ten *CPUs* are used, very respectable speed-ups. The speed-up ratio falls short of 100% because the percentage of the total processing time spent in *Steps* 3 and 4 increases as the number of *CPUs* increases.

Table II
Speed-up ratio

	model 1			model 2		
CPU	A	B	C	A	B	C
2	86.6	85.4	86.6	88.9	88.3	88.4
4	83.0	75.0	80.0	81.1	82.4	86.5
6	75.8	75.6	77.7	77.8	73.9	88.1
8	76.7	71.7	74.7	78.2	79.7	85.4
10	70.2	70.2	75.0	75.1	78.0	83.7

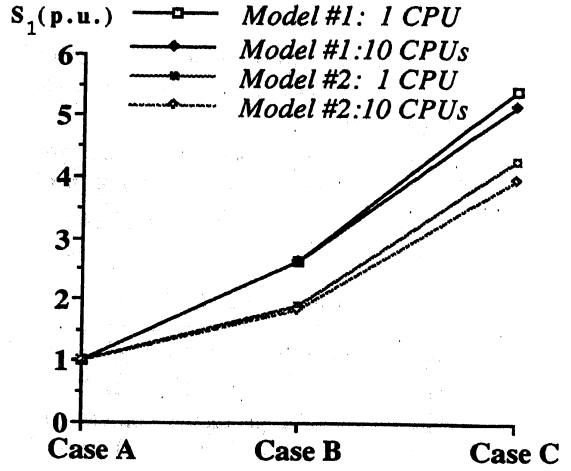


Fig. 6. Increase of processing time with nodes & elements.

IV. Conclusion

In this paper, a parallel algorithm to solve a set of simultaneous equations using the *ICCG* method is proposed. Iterative calculation to make the values on the boundary of subdomains coincide is not necessary. The principal characteristic of the proposed method is that the ratio of the processing time using multiple CPUs to the processing time with one processor does not increase with the number of nodes and number of elements in the analyses. The best performances by the proposed method can be achieved for approximately the same number of nodes for each subdomain, which procedure is under development.

References

[1] B. Nour-Omid, A. Raefsky, G. Lyzenga: "Solving finite element equation on concurrent computers," *Parallel Computations and their Impact on Mechanics, ed. A. K. Noor, ASME AMD-Vol. 86*, (1987), pp. 209-227.

[2] C. Farhat and E. Wilson: "Concurrent iterative solution of large finite element systems," *Comm. Appl. Numer. Meth.*, Vol. 3, (1987), pp. 319-326.

[3] K. Ueyama, H. Kaneko, & K. Umezu: "Parallel processing technique," *Proceedings of the 3rd Seminar concerning numerical analysis for electromagnetic field*, Okayama, Jan. 23-24, (in Japanese), (1992), pp. 18-23.