Contemporary Mathematics
Volume **180**, 1994

Analysis and Implementation of DD Methods for Parallel FE Computations

HAI XIANG LIN

ABSTRACT. Applications of finite element methods (FEM) are known to be computation-intensive. Therefore, various forms of parallel numerical algorithms have been studied in order to speed up finite element computations. Many of these recently proposed parallelization methods use domain decomposition (DD) as a framework. In this paper, we analyze the load balance and communication overhead in relation to the number of subdomains and the number of processors. It is shown that for an efficient parallel computation the often used formulation of the partitioning problem needs to be reformulated. An efficient domain decomposer is presented which has been implemented as a parallelization preprocessor for the finite element software package DIANA.

1. Introduction

In [**3**], a methodology is presented for the parallelization of the FEM software packages (e.g. DIANA). Direct methods play a dominent role in commercial FEM software packages, so we first focus on the parallelization of the direct solution of a large sparse system of linear equations. Fig. 1 describes the major modules of the parallelization method. The proposed approach starts by extracting parallelism from the very beginning of a problem formulation: the description of the element model (i.e. element domain). This is accomplished by performing domain decomposition. The next module "ordering" determines a parallel elimination sequence.

Given an elimination sequence the non-zero structure of the matrix factor can be determined through the so called symbolic factorization. The matrix factor is stored as a doubly bordered block diagonal (DBBD) matrix (see Fig. 2c). All block submatrices in Region I are stored as skyline matrices, those in Region II are stored as a collection of sparse vectors, and those submatrices in Region III are stored as dense matrices. We define the operations (e.g. an LDL^T factorization, an update) with respect to an entire block submatrix as a single task. The task dependence graphs of the factorization and triangular solutions are then derived from the non-zero structure of the matrix factor. An efficient macro data flow execution scheme is used to implement the parallel execution of the task dependence graph.

1991 *Mathematics Subject Classification.* 65F05, 65Y05, 65F50.
DIANA is a registered tradmark of TNO Institute.
This paper is in final form and no version of it will be submitted for publication elsewhere.

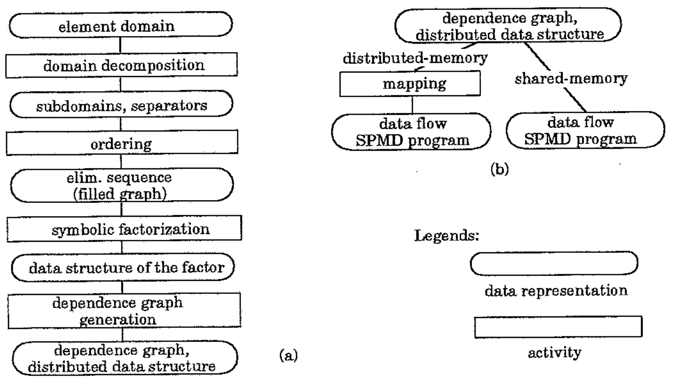


FIGURE 1. A strategy for the parallelization of direct solvers: the preparation phases; (b) Illustration of the PARASOL programs.

In this paper, we will concentrate on domain decomposition techniques. In order to handle regularly and irregularly structured domains with different types of elements, the domain is considered as a (general) connectivity graph. The problem of decomposing an element domain can now be considered as partitioning a graph. The so-called graph partitioning problem for the purpose of parallel computation is usually formulated as: *Divide a graph (element domain) into p (p is the number of processors) subgraphs separated from each other by the separators, such that the subgraphs are of the same size, i.e. consisting of the same number of elements or nodes, and that the number of separator nodes is minimal.*

It is known that smaller separators between the subgraphs imply fewer communication during a parallel computation. The requirement that all subgraphs having equal size are imposed to give a good load balance (we will show that more are required for a good load balance). Minimal separators and all subgraphs having equal sizes are two conflicting goals and the graph partitioning problem is known to be NP-complete. So, in practice heuristics have to be used.

2. A short review of previous work

The one-way and the nested dissection (ND) methods [2] were originally used to order the node elimination sequence for minimum fill-ins in sequential processing; recently they have been succefully applied as DD techniques for parallel computation. The dissection method starts by finding a "peripheral" node of the graph, then a subdivision is made through level-structuring. In case of one-way dissection, an s-partition can be obtained by selecting $(s-1)$ levels in the level structure as the sets of separators. In case of nested dissection, first a bisection is performed by chosing a "middle" level which separates the graph into two subgraphs of about the same size. Applying this bisection procedure on the resulting subgraphs repeatedly, a partition of $s = 2^t$ subgraphs is obtained after t iterations. Finding a good peripheral node usually requires a time of $O(n^2)$, so often a simpler approach is applied. The dissection methods are simple and fast (if a simple peripheral node is used), but do not give a very optimal separator length.

An alternative is to use a minimum-degree algorithm to perform a (sequential) ordering first. Then, a partitioning is performed on the elimination tree corresponding to the resulting ordering ([6], [1]). This method generally gives better results than the dissection methods in terms of separator length. However, the resulting

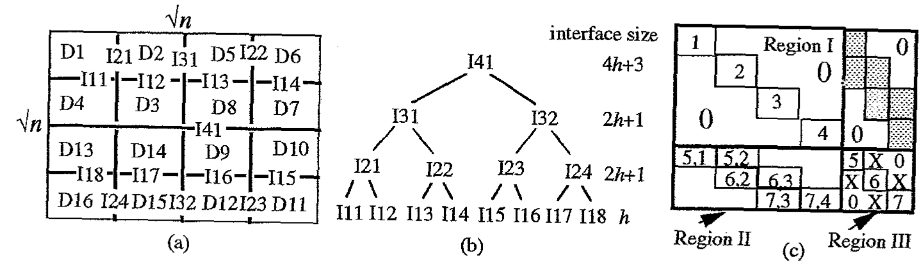


FIGURE 2. (a) A decomposition into 16 subdomains; (b) The elimination sequence of interfaces according to ND-ordering; (c) The DBBD matrix.

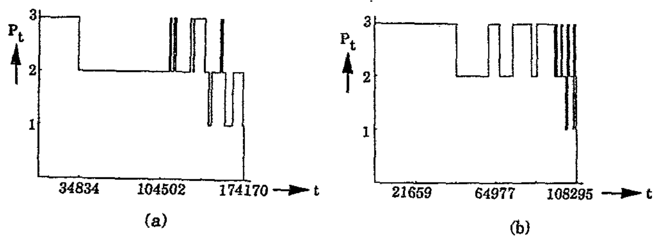


FIGURE 3. Execution profile of an example problem divided into (a) 3 subdomains; (b) 6 subdomains. $p = 3$. P_t is the number of tasks executed in parallel at a time.

subgraphs can vary considerably in size. Moreover, for a graph with n nodes a minimum-degree algorithm has a time complexity of $O(n^2)$ as compared to that of $O(n)$ to $O(n \cdot log(n))$ for the dissection methods. This means that the method using a minimum-degree algorithm has a higher time complexity than the solution of the sparse matrix system (e.g. $O(n^{3/2})$ for a $\sqrt{n} \times \sqrt{n}$ 2-D grid).

Another popular method is the spectral bisection (SB) method [7]. The SB method uses the second smallest eigenvector of the Laplacian matrix as the separator for a bisection of the graph. In terms of optimality the SB method is more robust than the dissection methods. However, it again has a time complexity of $O(n^2)$. So, speeding up the partitioning by means of parallel processing becomes an important issue. The parallel SB algorithm on the CM-5 is such an example.

3. Analysis of load balance in relation to the number of subdomains

In this section, we analyze the load balance and parallel efficiency as a function of subdomain's size and the number of processors. Because it is very hard to make a complexity analysis of a general element domain, we consider a $\sqrt{n} \times \sqrt{n}$ mesh and generalize the results to other problems. The $\sqrt{n} \times \sqrt{n}$ element mesh is subdivided into s subdomains each of size $\sqrt{n/s} \times \sqrt{n/s}$ as shown in Fig. 2a. We consider the factorization of $A = L \cdot D \cdot L^T$ with A being obtained through an ordering equivalent to the ND-ordering for the interfaces. We consider parallel sparse factorization of the type of medium to large granularity. Operations on the same block in the DBBD matrix form a single task and are scheduled on a single processor. In order to make the analysis simpler, we assume that the parallel factorization algorithm executes in a "lock-step" fashion. That is, all processors start with executing a group of tasks at the same time, they will wait for all the processors have completed their current group of tasks before entering a communication phase. After a communication phase, all processors will start to execute the next group of tasks.

The procedure of computing the LDL^T factor of the DBBD-matrix according to the ND-ordering can be described as follows.

1. For each $A_{i,i}$ in Region I, compute the LDL^T factor: $A_{i,i} \longrightarrow L_{i,i} \cdot D_{i,i} \cdot L_{i,i}^T$, for $i = 1, ..., s$. All the submatrices in Region I can be factorized independently.
2. Compute the factors in Region II: $L_{i,j} = A_{i,j} \cdot L_{i,i}^{-T} \cdot D_{i,i}^{-1}$ for $j = 1, ..., s$, and (interface segment) $i \in$ Adj{subdomain j}, followed by the updates in Region III (Schur-complement): $A_{k1,k2} = A_{k1,k2} - L_{k1,k} \cdot D_{k,k} \cdot L_{k2,k}^T$ for $k = 1, ..., s$ and interface segments $k2, k \in$Adj{subdomain $k1$}.
3. The interface segments induced by the last (recursive) bisection are eliminated first. Level $d = log(s)$ comprises the interfaces of the last bisections and level 1 the interface of the first bisection. The elimination sequence of the interfaces corresponds to an elimination tree in Fig. 2b. At each level l, the elimination can be performed simultaneously, followed by the updates: $A_{k1,k2} = A_{k1,k2} - L_{k1,k} \cdot D_{k,k} \cdot L_{k2,k}^T$ for $k1$ being an interface segment at level l and interface segments $k2, k \in$Adj{interface segment $k1$}.

Let each subdomain consisting of h^2 interior nodes. For $p = s = h^2$ (hence $n = s \cdot h^2 = h^4$), it has been derived in [3] that the parallel efficiency is

$$\eta(p = s) \approx \frac{829/42}{8 \cdot n^{1/2}/p + 26 + (log(p) - 44)/2 \cdot p^{1/2} - 70/p} \approx 0.6 \tag{3.1}$$

When $p = s/2 = h^2/2$, the parallel efficiency becomes:

$$\eta(p = s/2) \approx \frac{829/42}{29 + 14 \cdot \sqrt{2} \cdot p^{-1/2}} \approx 0.68 \tag{3.2}$$

In the calculation of the above efficiencies, the sequential time of the (complete) nested dissection ordering is used, which is known to be optimal in terms of the order of the number of floating point operations [2]. It has been shown in [3] that for $s \gg h \geq 5$ (i.e. n is very large), the parallel efficiencies are $\eta \approx 0.76$ for $p = s$ and $\eta \approx 0.92$ for $p = s/2$.

The results shown in (3.1) and (3.2) do not include the communication overhead, so they can be interpreted as the degree of load balance. It can be concluded that the load balance is significantly improved by partitioning a domain into $2 \cdot p$ instead of p subdomains. This is also illustrated in Fig. 3. Fig. 3a and 3b show the execution profiles of a problem partitioned into 3 and 6 subdomains respectively. The number of processors is 3. The execution profile shows the number of parallel tasks as a function of the time. The work load among the 3 processors in Fig. 3b is clearly beter balanced than in Fig. 3a.

When the parallel factorization is performed on a distributed memory system, there will be communication overhead. The load balance is increased by increasing the number of subdomains, however, the communication overhead will increase as well. For example, on a 2-D grid of $\sqrt{p} \times \sqrt{p}$ processors, the communication time is $T_{comm} = O(c_1 \cdot p \cdot h^2 + c_2 \cdot \sqrt{s} \cdot h^2)$. For a constant p, when s increases from p to $2 \cdot p$, the increase in communication time is bounded by a factor of $\sqrt{2}$. Thus, if T_{comm} is about 10% of the total execution time for $s = p$, then T_{comm} is less than 14% for $s = 2 \cdot p$. So, for a non-communication-dominant computation the improvement

```
Initialization: Mark all elements in $G$ as unpartitioned;
While (not all elements in $G$ are partitioned) do
    Select an unpartitioned element $el$ with the minimum degree;
    Determine a set $S$ of elements by a level-structuring algorithm using $el$ as starting
    point; and which holds that all elements in $S$ have not been partitioned previously and
    $min\_size \leq |S| \leq max\_size$;
    Determine the set of newly introduced separators $I$;
    Improve $I$ by means of the bipartite matching technique and adjust the $S$ accordingly;
    Add the set $S$ as the new subdomain;
    Mark all elements in $S$ in the graph as partitioned;
od;
Define the interface segments in the resulting partition of the graph;
Improve the subdomain connectivity of the resulting partition through adjusting the interface
segments;
```

FIGURE 4. A sketch of the graph partition algorithm GP.

in load balance (between 8% and 16%) outweighs the increase in communication overhead. We conclude that it is generally better to partition a domain into $k \cdot p$ subdomains with k is a small integer larger than 1 (say 2 or 3).

4. A generalized graph partitioning algorithm

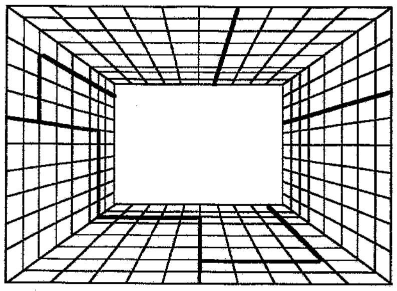
From the analysis in Section 3, we reformulate the partitioning problem as follows: *Divide a graph into $k \cdot p$ subgraphs, such that the size of the subgraphs is between* **min_size** *and* **max_size** *and that the number of separator nodes is minimal.*

The relaxation of the requirement of all subdomains having equal size creates the possibility for reducing the length of the separators. Load balance can still be achieved through combining smaller and larger subdomains together to a processor during the task assignment. Experiments show that choosing the values of *min_size* and *max_size* to differ about 10% from the mean value n/p gives good results.

Fig. 4 depicts a scheme of the graph partitioning algorithm. The algorithm starts with determining an initial partition of a subdomain by means of a variant of the dissection methods (similar to the Cuthill-McKee ordering). Then, in order to shorten the length of the (newly) introduced separators between the current subdomain and the unpartitioned parts, a bipartite maximum matching algorithm [**6**] is applied. This procedure repeats until the entire graph is partitioned. Finally, the separators are improved for a smaller subdomain interconnectivity [**3**].

5. Results

Fig. 5 shows two examples of domain decomposition produced by the algorithm GP. The time complexity of the algorithm GP is $O(n \cdot log(n))$, so the time required to partition a domain is typically less than 10% of the factorization time. Experiments on a Convex shared-memory vector computer with 4 processors show that typically efficiencies between 75% and 95% are obtained [**4**]. In Table 1 some simulation results of the parallel solver on a distributed memory computer are shown. The four problems are as follows. (1) A plate decomposed into 3 subdomains and the execution is simulated for 3 processors; (2) Mesh A, decomposed into 64 subdomains and simulated for 32 processors; (3) Mesh B, as in (2), but now simulated for 64 processors; (4) A stopcock, decomposed into 9 subdomains and simulated for 9 processors. T is the duration time of a parallel factorization (incl. the communication time) simulated using the timing characteristics of the Intel iPSC/2.


(a)

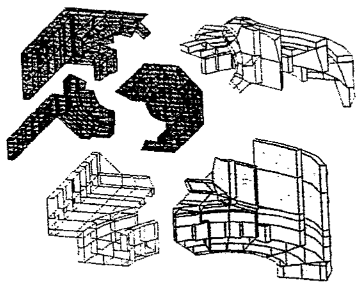

(b)

FIGURE 5. (a) A masonry wall partitioned into 5 subdomains; (b) A stopcock partitioned into 6 subdomains.

Problem	Plate	Mesh A	Mesh B	Stopcock
α	64.8	73.8	57.0	79.7
β	14.3	7.7	15.2	11.8
T	133706	7572911	4840176	226948925

TABLE 1. Some simulation results of the parallel execution of the task graphs resulted from DD. T is in time unit equivalent to a floating point operation.

α is the ratio between the average busy time per processor and T. β is the ratio between the average communication time per processor and T. α and β indicate the load-balance and communication overhead respectively.

Conclusions We have analyzed the DD technique for parallel direct solution of FE systems. It is shown that a better load-balance is achieved if the number of subdomains is k times ($k > 1$) the number of processors. We have reformulated the graph partitioning problem to allow a better resolution of the trade-offs between balanced subdomains and minimal separators. The results obtained from the parallelization of DIANA show that it is an efficient method for parallel FE computations.

REFERENCES

1. C.C. Ashcraft, *The domain/segment partition for the factorization of sparse symmetric positive definite matrices*, Eng. Comp. & Anal. TR ECA-TR-148, Boeing Compter Service, Nov. 1990.
2. A. George and J.W.H. Liu, *Computer solution of large sparse positive difinite systems*, Prentice-Hall, 1981.
3. H.X. Lin, *A methodology for parallel direct solution of finite element systems*, Ph.D. thesis, Delft University of Technology, 1993.
4. H.X. Lin and H.J. Sips, *Parallel direct solution of large sparse systems in finite element computations*, Proc. 1993 Int. Conf. on Supercomputing, Tokyo, July, 1993, pp. 261-270.
5. J.W.H. Liu, *A graph matching algorithm by node separators*, ACM Trans. Math. Softw. 15 (1989), 198 - 219.
6. J.W.H. Liu, *The role of elimination trees in sparse factorization*, SIAM J. Matrix Anal. Appl. 11 (1990), 134-172.
7. A. Pothen, H.D. Simon and K-P. Liou, *Partitioning sparse matrices with eigenvectors of graphs*, SIAM J. Matrix Anal. Appl. 11 (1990), 430-452.

DEPARTMENT OF APPLIED MATHEMATICS AND INFORMATICS, DELFT UNIVERSITY OF TECHNOLOGY, MEKELWEG 4, NL-2628 CD, DELFT, THE NETHERLANDS
E-mail address: lin@pa.twi.tudelft.nl