ELSEVIER

Comput. Methods Appl. Mech. Engrg. 149 (1997) 289–301

Computer methods in applied mechanics and engineering

A parallel multifrontal algorithm and its implementation

P. Geng, J.T. Oden*, R.A. van de Geijn

Texas Institute for Computational and Applied Mathematics, The University of Texas at Austin, Austin, TX 78712, USA

Abstract

In this paper, we describe a multifrontal method for solving sparse systems of linear equations arising in finite element and finite difference methods.

The method proposed in this study is a combination of the nested dissection ordering and the frontal method. It can significantly reduce the storage and computational time required by the conventional direct methods and is also a natural parallel algorithm. In addition, the method inherits major advantages of the frontal method, which include a simple interface with finite element codes and an effective data structure so that the entire computation is performed element by element on a series of small linear systems with dense stiffness matrices.

The numerical implementation targets both distributed-memory machines as well as conventional sequential machines. Its performance is tested through a series of examples.

1. Introduction

In this paper, we seek for an efficient direct method of solving the system of linear equations,

$$Ax = \boldsymbol{b} \tag{1}$$

where $A = [a_{ij}]_{N\times N}$ is a sparse matrix generated by finite element methods or finite difference methods, $\boldsymbol{b} = \{b_i\}_N$ is a known vector, $\boldsymbol{x} = \{x_i\}_N$ is the unknown solution vector to be solved and N is the total number of unknowns and is also referred to as the total degrees of freedom.

Sparse systems of linear equations are typically solved by one of two different methods—*iterative methods* or *direct methods*. A direct method involves explicit factorization of the sparse matrix A into the product of lower and upper triangular matrices L and U, and it generally requires much more computer time and storage than iterative methods. However, direct methods are important because of their generality and robustness. In many cases, direct methods are preferred because the effort involved in seeking a good preconditioner for an iterative solution often outweighs the cost of direct factorization. Furthermore, direct methods provide effective means for solving systems with the same stiffness matrix A and different right-hand vectors $\boldsymbol{b}$ because the factorization needs to be performed only once.

A direct method is completed in two steps: LU factorization of

$$A = LU \tag{2}$$

followed by solving the triangular systems

$$Ly = \boldsymbol{b} \tag{3}$$

* Corresponding author.

and

$$Ux = y \tag{4}$$

where L is a lower triangular matrix with unit diagonal coefficients and U is an upper triangular matrix.

The factorization of an N by N matrix A is completed in N steps. Setting $A_1 = A$, we have

$$A_1 = \begin{pmatrix} a_{11} & \boldsymbol{u}_1^{\mathrm{T}} \\ \boldsymbol{v}_1 & B_1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \frac{\boldsymbol{v}_1}{a_{11}} & I_{N-1} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & B_1 - \frac{\boldsymbol{v}_1\boldsymbol{u}_1^{\mathrm{T}}}{a_{11}} \end{pmatrix} \begin{pmatrix} a_{11} & \boldsymbol{v}_1^{\mathrm{T}} \\ 0 & I_{N-1} \end{pmatrix}$$

$$= L_1 A_2 U_1$$

$$A_2 = \begin{pmatrix} 1 & 0 \\ 0 & B_1' \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & \begin{pmatrix} a_{22} & \boldsymbol{u}_1^{\mathrm{T}} \\ \boldsymbol{v}_2 & B_2 \end{pmatrix} \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 \\ 0 & \begin{pmatrix} 1 & 0 \\ \frac{\boldsymbol{v}_2}{a_{22}} & I_{N-2} \end{pmatrix} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \begin{pmatrix} 1 & 0 \\ 0 & B_2 - \frac{\boldsymbol{v}_2\boldsymbol{u}_2^{\mathrm{T}}}{a_{22}} \end{pmatrix} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \begin{pmatrix} a_{22} & \boldsymbol{u}_2^{\mathrm{T}} \\ 0 & I_{N-2} \end{pmatrix} \end{pmatrix}$$

$$\vdots$$

until

$$A_N = I_N \tag{5}$$

where $\boldsymbol{v}_i$ and $\boldsymbol{u}_i$ are two vectors of length $N - i$, and I_{N-i} is an $(N - i) \times (N - i)$ identity matrix. In the sequel $B_i' = B_i - \boldsymbol{v}_i\boldsymbol{u}_i^{\mathrm{T}}/a_{ii}$ is known as the part of A remaining to be factored after the first i steps of the factorization have been performed. We also refer to performing the ith step of the factorization as eliminating variable x_i or eliminating the ith row and column of A.

In order to improve numerical stability as well as avoiding $a_{ii} = 0$, we permute certain rows and columns of matrices so that a_{ii} has the maximum absolute value among all coefficients on each remaining matrix A_i. This procedure is known as *pivoting* and (2) should be written as

$$A = PLUQ \tag{6}$$

where P and Q are the matrices representing the accumulation of permutation on the row and column interchanges required by the pivoting.

Another important fact about sparse systems of linear equations is that because the zero entries of the matrix B_i may not be the zero entries of $\boldsymbol{v}_i\boldsymbol{u}_i^{\mathrm{T}}$, some entries that are initially zero in A become nonzero after factorization and those entries are known as *fill-in*. The *fill-in* is inevitable in the factorization of sparse systems, but it can be drastically reduced through reordering unknowns of sparse systems of linear equations. More precisely, let P be a permutation matrix, we can choose P such that the factorization of PAP^{T} has much less *fill-in* than that of A. Clearly, the permutation matrix P used here is different from the one used in (6). In general, the requirement of the pivoting in the factorization process will restrict the ability of using the ordering of unknowns to minimize the amount of *fill-in*. In some direct methods, including the frontal methods and supernodal methods, a limited pivoting can be applied without affecting the performance of the method (see [8,2]) and we will discuss this later.

Without pivoting, once the ordering is determined, the precise locations of all *fill-in* entries in L and U can be predicted in advance. The process by which the nonzero structures of L and U is determined in advance is called *symbolic factorization* [7].

A heuristic method which has been found to be very effective in finding efficient orderings is the *minimum degree algorithm* [10]. At each step, this method selects the row with the least number of nonzero entries as the next row to be eliminated. The minimum degree algorithm is easy to implement and produces reasonably good orderings over a remarkably broad range of problem classes. However, because the minimum degree algorithm is a heuristic method and there is lack of the theoretical foundation for it, its success is not well understood and there is no robust and efficient way to deal with possible variability in the quality of the orderings which it produces. Furthermore, the minimum degree algorithm inherently is a sequential process, and it will be difficult to develop a parallel direct sparse solver based on this method.

Another effective ordering algorithm is the nested dissection [1]. The nested dissection algorithm is an ordering method based on a sequence of nested dissections on the domain. By dividing a given domain into two subdomains, we are able to reorder unknowns of the system in such a way that

$$A = \begin{pmatrix} A_{11} & 0 & A_{13} \\ 0 & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{pmatrix}. \tag{7}$$

Where A_{11} and A_{22} are the matrix blocks corresponding to the interior nodes of each subdomain, A_{33} is the matrix block corresponding to the nodes on the interface between two subdomains, and A_{13}, A_{23}, A_{31} and A_{32} are coupling matrix blocks between interior nodes and nodes on interface boundaries. The significance of the matrix partitioning given in (7) is that the zero blocks are preserved in the factorization so that the *fill-in* is limited. This idea can be applied recursively, i.e. we break each subdomain into smaller and smaller pieces to limit *fill-in* on diagonal matrix blocks. Furthermore, the successive dissection on each subdomain will automatically place most unknowns associated with the interior nodes before the unknowns associated with the nodes on the interface boundaries so that *fill-in* in the off-diagonal matrix blocks such as A_{13}, A_{23}, A_{31} and A_{32} is also limited. In [1], George proved that a nested dissection ordering for a uniform mesh on a square domain can reduce the arithmetic operation counts from the usual $O(N^2)$ to $O(N^{3/2})$ and reduce the memory requirement from $O(N^{3/2})$ to $O(N \log_2 N^{1/2})$. In addition, two major matrix blocks A_{11} and A_{22} are completely decoupled and can be proceeded separately on different processors, thus the nested dissection algorithm is a natural parallel algorithm which not only enable us to limit *fill-in* but also promote concurrency at each level of the nested dissection.

Despite its theoretical superiority, the nested dissection algorithm gets little application in practical computation. A major difficulty which the nested dissection method runs into in its implementation is that the nested dissection requires that the original domain be divided into several hundreds to thousands of subdomains, and then a special data structure or house-keeping scheme is required to have an order of eliminating interior nodes first and interface boundary nodes next. Furthermore, such data structure or housekeeping scheme must be easy to be implemented and possesses a simple interface with existing finite element codes.

In this paper, we will demonstrate that the above difficulties can be overcome by a modified frontal method or, more precisely, a multifrontal method. The paper will be organized as follows. We first give a simple description on the frontal method and the method used to control factorization in practice; next we discuss certain details of the implementation on the conventional sequential machines as well as parallel machines; finally the experimental results are presented.

2. The overall strategy

The frontal method was originally proposed by Irons [9] in 1970 and later Duff [3] introduced the multifrontal method to solve the problems with indefinite linear systems. In this study, we extend their work and develop a method which enable us to eliminate unknowns in a nested dissection ordering.

To demonstrate the frontal method, the four-element and nine-node mesh shown in Fig. 1 will be considered. We assume that there is only one unknown associated with each node and all the equations are stored in the order of ascending node number.

The stiffness matrix and the right-hand side vector arising in finite element methods can be naturally expressed in forms of

$$A = \sum_{m=1}^{M} A^m \tag{8}$$

and

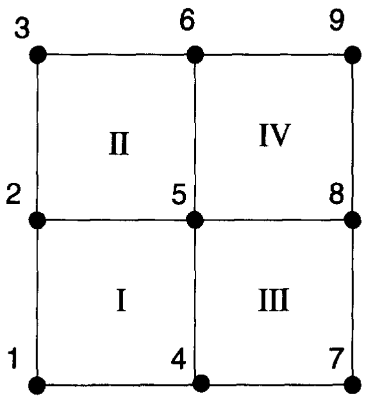


Fig. 1. The mesh of 4 finite elements and the node numbers.

$$b = \sum_{m=1}^{M} b^m \tag{9}$$

where A^m and b^m represent the contribution from a single finite element i and M is the total number of elements. Because most entries in A^m and b^m are zero, we can pack each A^m into a small full matrix (known as *the element stiffness matrix*) and each b^m into a small full vector (known as *the element load vector*), and then a vector of indices is created to label where each entry of the packed matrix and vector fits into the global matrix A and the vector $\boldsymbol{b}$. The process of adding the element stiffness matrices and load vectors into their global matrix and vector is known as assembly.

The index vectors created in the frontal method are referred to as *the element destination vectors*. Instead of adding the local stiffness matrices and load vectors into a global system, the frontal method adds the local matrices and vectors into a small dense linear system which we refer to as a front. The size of the element destination vector is equal to the number of nodes on the element. The combination of all element destination vectors is *the destination vector*. Initially, the components of each element destination vector is set to be *the nicknames* of nodes on the element, i.e.

$$\text{the nickname} = \text{the node number} \times M_e + n_e \tag{10}$$

and M_e is an integer which must be larger than the maximum number of unknowns associated with each node and n_e is the number of unknowns associated with the corresponding node.

The initial form of the destination vector is also called as *the nickname vector*. Each nickname component contains two pieces of information, the node number and the number of unknowns associated with each node and the nickname vector provides the information about the node connectivity on each element and the number of unknowns associated with each node.

For the example shown in Fig. 1, we have assumed that $n_e = 1$ and will use $M_e = 10$ thus the nickname vector should be

$$\begin{array}{llllll} (11, & 41, & 51, & 21, & & \text{for the element I} \\ 21, & 51, & 61, & 31, & & \text{for the element II} \\ 41, & 71, & 81, & 51, & & \text{for the element III} \\ 51, & 81, & 91, & 61) & & \text{for the element IV}\,. \end{array} \tag{11}$$

The major function of the symbolic factorization is to convert each component of the destination vector from (10) into a form of

$$\text{the location index} \times 10 \times M_e + n_e \times 10 + \text{flag}\,. \tag{12}$$

Here, *the location index* gives the location where each entry of the element matrix and load vector fits into the front and

$$\text{flag} = \begin{cases} 0: & \text{for not the last occurrence of the node} \\ 1: & \text{for the last occurrence of the node} \end{cases} \tag{13}$$

gives the information whether the corresponding entries at the frontal matrix and vector have been fully assembled or not. Clearly, *the last occurrence of the node* or flag = 1 means *yes* and otherwise means *no*. For example, after the symbolic factorization, the vector shown in (11) should be converted into

$$\begin{array}{llllll} (111, & 210, & 310, & 410, & \text{for the element I} \\ 311, & 210, & 410, & 511, & \text{for the element II} \\ 211, & 411, & 510, & 111, & \text{for the element III} \\ 111, & 311, & 411, & 411) & \text{for the element IV}\,. \end{array} \tag{14}$$

The frontal method is operated in an element by element basis. For the example considered here, the first front should be

$$\begin{pmatrix} a^{\rm I}_{11} & a^{\rm I}_{14} & a^{\rm I}_{15} & a^{\rm I}_{12} \\ a^{\rm I}_{41} & a^{\rm I}_{44} & a^{\rm I}_{45} & a^{\rm I}_{42} \\ a^{\rm I}_{51} & a^{\rm I}_{54} & a^{\rm I}_{55} & a^{\rm I}_{52} \\ a^{\rm I}_{21} & a^{\rm I}_{24} & a^{\rm I}_{25} & a^{\rm I}_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_4 \\ x_5 \\ x_2 \end{pmatrix} = \begin{pmatrix} b^{\rm I}_1 \\ b^{\rm I}_4 \\ b^{\rm I}_5 \\ b^{\rm I}_2 \end{pmatrix} \tag{15}$$

where the superscript denotes the element number from which the matrix and the right-hand side vector entries were derived and the subscripts are the global positions of the corresponding coefficients. The flags at the corresponding element destination vector (the first part of vector in (14)) tell us that the first row and column (corresponding to the Node 1) have been fully assembled and can be eliminated from (15). After elimination, the state of the front becomes

$$\begin{pmatrix} a_{44} & a_{45} & a_{42} \\ a_{54} & a_{55} & a_{52} \\ a_{24} & a_{25} & a_{22} \end{pmatrix} \begin{pmatrix} x_4 \\ x_5 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_4 \\ b_5 \\ b_2 \end{pmatrix} \tag{16}$$

where

$$a_{ij} = a^{\rm I}_{ij} - a^{\rm I}_{i1} a^{\rm I}_{1j} / a^{\rm I}_{11}$$

and

$$b_i = b^{\rm I}_i - a^{\rm I}_{i1} b^{\rm I}_1 / a^{\rm I}_{11}\,.$$

Next, we add the second element to (16) to form a new front. The location indices in the second element destination vector label where each entry of the local stiffness matrix and load vector fits into the new front. After assembling, the front should be

$$\begin{pmatrix} a_{44} & a_{45} & a_{42} & & \\ a_{54} & a_{55} + a^{\rm II}_{55} & a_{52} + a^{\rm II}_{52} & a^{\rm II}_{56} & a^{\rm II}_{53} \\ a_{24} & a_{25} + a^{\rm II}_{25} & a_{22} + a^{\rm II}_{22} & a^{\rm II}_{26} & a^{\rm II}_{23} \\ & a^{\rm II}_{65} & a^{\rm II}_{62} & a^{\rm II}_{66} & a^{\rm II}_{63} \\ & a^{\rm II}_{35} & a^{\rm II}_{32} & a^{\rm II}_{36} & a^{\rm II}_{33} \end{pmatrix} \begin{pmatrix} x_4 \\ x_5 \\ x_2 \\ x_6 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_4 \\ b_5 \\ b_2 \\ b_6 \\ b_3 \end{pmatrix}. \tag{17}$$

Similarly, from the flags in the destination vector, we know that the second and third rows and columns can be eliminated from (17) and after elimination, the front is ready for the next element. The *LU* factorization and forward substitution are complete after the same procedure is repeated for all elements. The backward substitution is performed in a reversed order (starts at the last element) and the procedure is similar.

The order of unknowns being eliminated in the frontal method is determined by the sequence of C^m and b^m being added into the fronts; and for the example considered here, the unknowns are eliminated in an order of (by the node numbers)

$$(1, 2, 3, 4, 7, 5, 8, 9, 6)$$

which is not a nested dissection ordering. Next, we discuss a modified frontal method which can eliminate unknowns at a nested dissection ordering and the same example will be used to demonstrate the solving procedure.

The new method requires a series of nested dissection on the original domain and a set of extra auxiliary elements to control the sequence of elimination on each subdomain. As shown in Fig. 2, the first level of the dissection divides the original domain given in Fig. 1 into two subdomains which are separated by the boundary consisting of the nodes

$$(4, 5, 6)\,. \tag{18}$$

The second level of the dissection further divides the domain into four subdomains and they are separated by the boundaries consisting of the nodes

$$(4,5,2),\ (2,5,6),\ (4,8,5) \quad \text{and} \quad (5,8,6)\,, \tag{19}$$

respectively. The factorization starts from the higher level to the lower level and at each level, the frontal solver works independently on each subdomain and is restricted only to eliminate the unknowns associated with interior nodes. In order to do this, we create the following sets of the nickname vectors:

the second-level dissection:

$$\begin{array}{ll} (11,41,51,21, & 41,51,21) \\ (21,51,61,31, & 21,51,61) \\ (41,71,81,51, & 41,81,51) \\ (51,81,91,61, & 51,81,61) \end{array} \tag{20}$$

the first-level dissection:

$$\begin{array}{lll} (41,51,21, & 21,51,61 & 41,51,61) \\ (41,71,51, & 51,81,61 & 41,51,61) \end{array} \tag{21}$$

and finally, the zero-level dissection:

$$(41,51,61) \tag{22}$$

For convenience, here we consider the original domain as the subdomain of the zero-level dissection.

Each nickname vector in (20) is composed of the element nickname vectors for the original elements on the subdomain plus a so-called control element. The control elements consist of the node on the separation boundaries (one of the node vectors given in (19)). Then, the control elements at the second-level dissection becomes the new elements at the first-level dissection. Each nickname vector at the first level is composed of two parts: the nickname vector of the control elements from the second-level dissection and the nickname vector of its own control element which also consists of the nodes on the separation boundary (the node vector given in

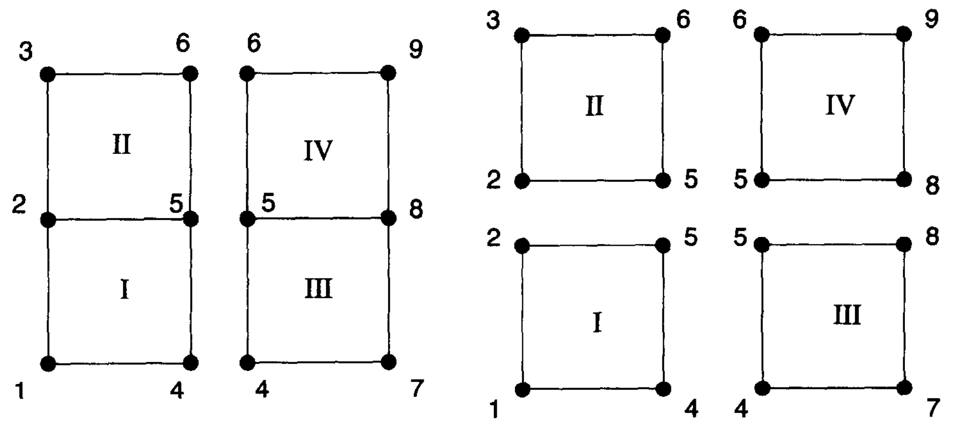


Fig. 2. The two levels of the nested domain decomposition on the mesh given in Fig. 1. After the second domain decomposition, the multifrontal method proposed in this study can eliminate the unknowns in a nested dissection ordering.

(18)). Similarly, the control element at the first-level dissection becomes the new element of the zero-level dissection and there is no need of the control element for the zero-level dissection.

Because of using the control elements, we can proceed with each nickname vector independently during the symbolic factorization but still have the flags in the destination vectors being set correctly. After the symbolic factorization, the vectors given (20, (21) and (22) are converted into

the second-level dissection:

$$\begin{array}{ll} (111, 210, 310, 410, & 111, 211, 311) \\ (110, 210, 310, 411, & 111, 211, 311) \\ (110, 211, 310, 410, & 111, 211, 311) \\ (110, 210, 311, 410, & 111, 211, 311) \end{array}$$

the first-level dissection:

$$\begin{array}{lll} (110, 210, 310, & 311, 210, 410, & 111, 211, 311) \\ (110, 210, 310, & 310, 211, 410, & 211, 111, 311) \end{array}$$

and finally the zero-level dissection:

$$(111, 121, 131)\,.$$

Then, the last element vector (the control element) on each destination vector will be skipped after the symbolic factorization. The real computation is performed in seven steps and each step will be controlled by the destination vectors

$$\begin{array}{ll} (111, 210, 310, 410) & \text{step 1} \\ (110, 210, 310, 411) & \text{step 2} \\ (110, 211, 310, 410) & \text{step 3} \\ (110, 210, 311, 410) & \text{step 4} \\ (110, 210, 310, 311, 210, 410) & \text{step 5} \\ (110, 210, 310, 310, 211, 410) & \text{step 6} \\ (111, 121, 131) & \text{step 7} \end{array} \tag{23}$$

The computation on the different subdomain can proceed independently and the flags in the destination will only allow the elimination of the unknowns associated with the interior nodes. During the steps 1–4, the unknowns associated with the nodes 1, 3, 7 and 9 are eliminated. Each step will open a new working front and there will be four unfinished working fronts at the end of step 4. Steps 5 and 6 will combine the unfinished fronts from the previous steps and eliminate the unknowns associated with nodes 2 and 8. Steps 5 and 6 open two new fronts. Then, those two unfinished fronts are added together at the final step and the unknowns associated with nodes 4, 5 and 6 are eliminated. The unknowns here are eliminated in an order of (by the node numbers)

$$(1, 3, 7, 9, 2, 8, 4, 5, 6)$$

which is a nested dissection ordering.

Because there exist more than one working front at certain stages of the computation, the method discussed is a multifrontal method. A special issue arising in this method is that the extra storage must be allocated to save those unfinished fronts during computation. Consider the cubic mesh shown in Fig. 5. To reduce the maximum number of unfinished working fronts, instead of the procedure as we discussed above (also as shown in Fig. 6(a)), the factorization is actually performed in a sequence as shown in Fig. 6(b) and it can reduce the maximum number of unfinished fronts from P to $\log_2 P$.

Most importantly, in the sequence given in Fig. 6(b), the storage of the working fronts can be simply handled as a stack. To clarify this, consider Fig. 6(b). After the first step, we push the front into the stack and then start step 2. At the end of step 2, we pop the front saved in the first step out of the stack, add it to the current front and do the factorization for step 3. At the end of step 3, the current front is pushed into the stack and also at the end of step 4. At this moment, there are two fronts in the stack and under the *last in and first out* stack rule, the front saved at the end of step 4 will be the first one to be popped and it is exactly what we want at the end of step 5. The front saved at step 3 is the next one to be popped out and it is also exactly what we want at the end of step 6. We then combine the fronts from steps 3 and 6 together and complete the work. The memory required by the stack is dynamically allocated and reaches its maximum at the end of the step

$$\sum_{i=0}^{l} 2^i - l + 1$$

where $P = 2^l$ and then the entire memory will be released before the end of factorization. The same stack algorithm is also used to save and update the right-hand sides of fronts during the forward and backward substitution. The major advantages of the stack algorithm proposed here is that it can avoid the complexity of saving a list of addresses pointing to memory location of fronts and greatly simplifies the implementation.

3. Performance

In this section, we will discuss the performance of the method proposed and demonstrate that after a proper sequel of nested domain decomposition, the frontal width is reduced greatly and so are the storage and operational counts.

The stiffness matrices arising in the finite element or finite difference methods are always structurally symmetric. Matrix $A = [a_{ij}]_{N\times N}$ is said to be structurally symmetric if $a_{ij} \neq 0$ follows that $a_{ji} \neq 0$ for $1 \leq i, j \leq N$. For a structurally symmetric matrix, the number and position of nonzero entries in the vectors v_k and u_k in (2) are identical and the number of arithmetic operations required to factor a structurally symmetric matrix A can be calculated by

$$\theta = \sum_{k=1}^{N-1} (2\mu_k^2 + \mu_k) \approx 2 \sum_{k=1}^{N-1} \mu_k^2 \tag{24}$$

and the total number of nonzero entries in L and U matrices will be

$$\eta = \sum_{k=1}^{N} (2\mu_k - 1) \approx 2 \sum_{k=1}^{N} \mu_k \,. \tag{25}$$

where μ_k is the number of nonzero entries in v_k or u_k. In the frontal method, we also call μ_k the frontal width.

In [1], George proved that for the problem with a uniform mesh of linear elements on a square domain, the nested dissection method could reduce the memory requirement from $O(N^{3/2})$ to $O(N^{1/2} \log_2 N)$ and the number of arithmetic operations from $O(N^2)$ to $O(N^{3/2})$. The George's analysis requires that the original domain be dissected iteratively until each new subdomain contains at most $2 \times 2 = 4$ elements. In this study, we consider the performance of the method in a more realistic basis requiring only

$$P \gg 1 \quad \text{and} \quad \frac{M}{P} \gg 1 \tag{26}$$

where P is the total number of subdomains after the final level of the dissection and M is the total number of elements.

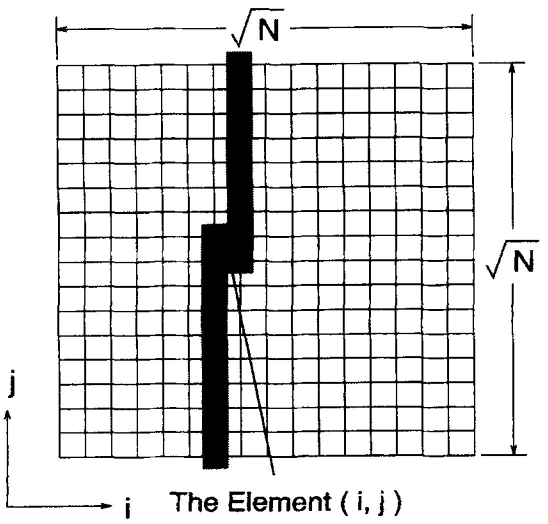
For the problem given in Fig. 3, the frontal width (i.e. μ_k) of using the conventional single frontal method on average is about $\sqrt{N}$ and by using (24) and (27), we have

$$\theta = 2 \sum_{k=1}^{N-1} \mu_k^2 = 2 \sum_{k=1}^{N-1} (\sqrt{N})^2 = 2N^2 \tag{27}$$

and

$$\eta = 2 \sum_{k=1}^{N} \mu_k = 2 \sum_{k=1}^{N} \sqrt{N} = 2N^{3/2} \,. \tag{28}$$

For the multifrontal method, we suppose that after the l levels of the nested dissection, the original domain is divided into $P = 2^l$ square subdomains with $n_s = \sqrt{N/P} + 1$ nodes on each side. Because the elimination of the



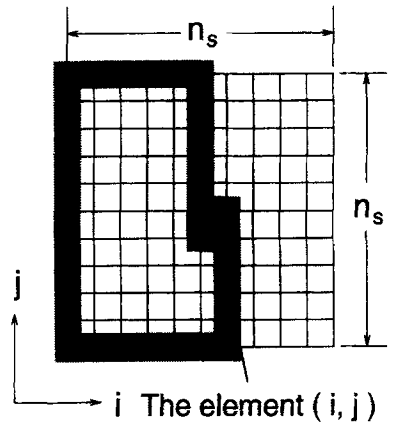


Fig. 3. The two-dimensional example. The uniform mesh of linear elements on a square domain. Here the front moves from the right side to the left. The frontal width (the number of nodes covered by the shadow) on average is equal to $\sqrt{N}$.

Fig. 4. The frontal width on a subdomain. We suppose that the original square domain is divided into P square subdomains. On average, there are $n_s = (\sqrt{N/P}+1)^2$ nodes on each side of the subdomains, and the frontal width or μ_k^s for an interior node on the element (i, j) is $2(n_s+1)+2i+1$ (the number of nodes covered by the shadow).

unknowns associated with the boundary nodes must be held till all the unknowns on the interior nodes are eliminated, the frontal width associated with a given interior node (as shown in Fig. 4) is equal to

$$\mu_k^s = 2(n_s+1) + 2\lfloor k/n_s \rfloor + 1 \approx 2(n_s + \lfloor k/n_s \rfloor) \tag{29}$$

where $\lfloor r \rfloor$ represents the largest integer which is less than r. It is very difficult to derive a general formulation to evaluate the frontal width associated with the nodes on the boundaries or interface boundaries. However, in assumption (26), we can simply use (29) to evaluate the frontal width associated with the boundary nodes and then the total number of arithmetic operations and the storage required to factor A can be estimated by

$$\hat{\theta} = 2P \sum_{k=1}^{n_s^2-1} (\mu_k^s)^2 = 2P \sum_{k=1}^{n_s^2-1} [2(n_s + \lfloor k/n_s \rfloor)]^2 \approx \frac{56}{3P} N^2 \tag{30}$$

and

$$\hat{\eta} = 4P \sum_{k=1}^{n_s^2} (n_s + \lfloor k/n_s \rfloor) \approx 6 \times \frac{N^{3/2}}{\sqrt{P}} = \frac{6}{\sqrt{P}} N^{3/2}, \tag{31}$$

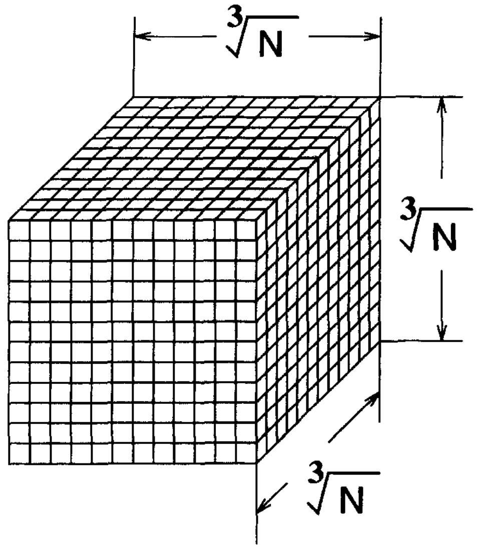
respectively. The experimental results indicate that the estimated values $\hat{\theta}$ and $\hat{\eta}$ will be close to the real θ and η if the assumption (26) holds.

Comparing (30) and (31) with (27) and (28), we can conclude that for the problem shown in Fig. 3, the method proposed here can reduce the operational counts and the nonzero entries in the L and U matrices to

$$\frac{28}{3P} \text{ and } \frac{3}{\sqrt{P}} \tag{32}$$

of their original values, respectively. In addition, a similar analysis indicates that for the three-dimensional example shown in Fig. 5, the multifrontal method can reduce the operational counts and the number of the nonzero entries in the L and U matrices from $2N^{7/3}$ and $2N^{5/3}$ to

$$\frac{52}{3P^{4/3}} \text{ and } \frac{4}{P^{2/3}} \tag{33}$$



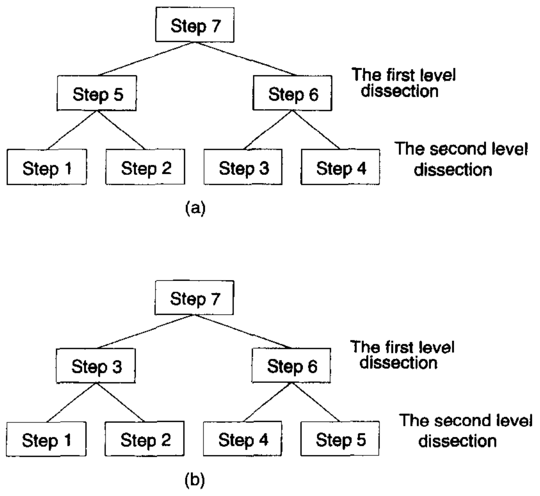

Step 7
Step 5
Step 6
The first level dissection
Step 1
Step 2
Step 3
Step 4
The second level dissection
(a)
Step 7
Step 3
Step 6
The first level dissection
Step 1
Step 2
Step 4
Step 5
The second level dissection
(b)

Fig. 5. The three-dimensional example. The uniform mesh of linear elements on a cubic domain.

Fig. 6. The maximum number of working fronts required by the factorization sequence (a) is equal to P ($P = 4$ for the example given here) and it happens after the first P steps are performed. The sequence given in (b) reduces the maximum number of working fronts to $\log_2 P$ and most importantly, the storage of working fronts can be simply handled as a stack.

of their original values, respectively.

Finally, the symbolic factorization of the conventional frontal method is an $O(N^2)$ operation where N is the length of the destination vector. However, the method proposed in this study breaks the original destination vector into P independent vectors and reduce the symbolic factorization to an operation of

$$P \times O\left(\frac{N^2}{P^2}\right) = O\left(\frac{N^2}{P}\right)$$

which will drastically reduce the cost of the symbolic factorization.

4. Experiments

The multifrontal method proposed in this study has been implemented on the sequential machines as well as the distributed memory machines. The performance of the method was tested through a series of experiments on the two- and three-dimensional examples and we here give a summary of those results.

4.1. Performance

Since the frontal width for the equations is determined after the symbolic factorization, the storage and the operational counts can be exactly calculated by using (24) and (25). We calculated the nonzero entries of the stiffness matrix after the factorization and the operational counts of the *LU* factorization for the two- and three-dimensional examples as shown in Figs. 3 and 5 with a different number of linear or quadratic elements. Table 1 shows the result of a typical computation performed on the two-dimensional test problem with the mesh of linear elements and Table 2 shows the result for the three-dimensional test problem. Here we are able to reduce the storage by 70% and the operational counts of the *LU* factorization by nearly 90% by using the proper numbers of fronts. The calculated results for the mesh of quadratic elements are similar and are not presented.

For the real computation, we solve a two-dimensional Laplace problem on the two-dimensional test problem with a different size of mesh. Table 3 compares the computational time of using different numbers of fronts for

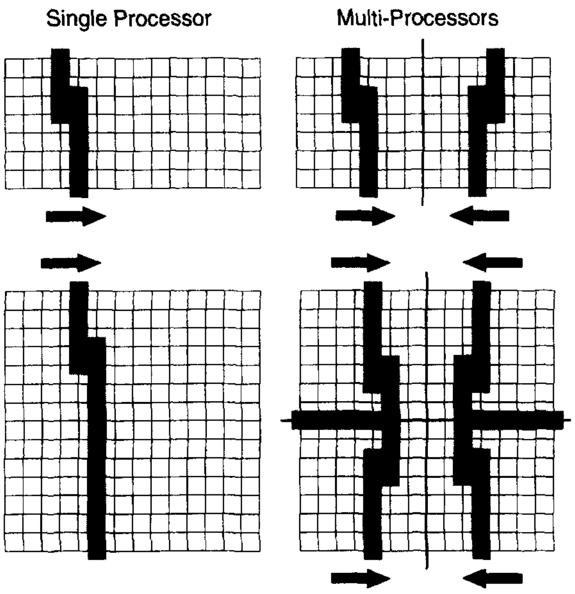


Fig. 7. The arrangements of the fronts for the tests performed on the phase II work. Because a perfect loading balance can be expected in this experiment, we have the parallel efficiency near to 100%.

Table 1
The comparison of storage and operational counts in *LU* factorization for the different number of fronts (the two-dimensional uniform mesh of $128 \times 128 = 16\,384$ linear elements and 16 641 degrees of freedom) where *P* represents the number of fronts

No. of fronts (P)	No. of nonzero entries	Operational counts
$P = 1$	4.29×10^6	5.646×10^8
$P = 16$	2.82×10^6 (65.8%)	2.666×10^8 (47.2%)
$P = 64$	1.79×10^6 (47.7%)	1.284×10^8 (21.9%)
$P = 256$	1.28×10^6 (29.9%)	0.853×10^8 (15.1%)

Table 2
The comparison of storage and operational counts in *LU* factorization for the different number of fronts (the three-dimensional uniform mesh of $32 \times 32 \times 32 = 32\,768$ linear elements and 35 937 degrees of freedom) where *P* represents the number of fronts

1	77.2×10^6	85.2×10^9
64	27.9×10^6 (36.1%)	17.9×10^9 (21.0%)
256	23.0×10^6 (29.7%)	16.2×10^9 (19.0%)
512	22.3×10^6 (28.8%)	16.0×10^9 (18.8%)

Table 3
The comparison of the time spent on each step of computation among the different numbers of fronts. We here solve a two-dimensional Laplace problem on the uniform mesh of $64 \times 64 = 4096$ quadratic elements and 16 649 degrees of freedom and the test was performed on an IBM R6000 3BT machine. Here, *P* also represents the number of fronts

P	Symbolic factorization (s)	Factorization and forward substitution (s)	Backward substitution (s)
1	87.2	118.7	89.6
16	7.0 (8.0%)	118.7	89.6
64	3.5 (4.0%)	25.0 (21.1%)	5.7 (6.3%)
128	3.4 (4.0%)	23.1 (19.5%)	5.7 (6.3%)
256	4.4 (5.1%)	24.0 (20.2%)	7.6 (8.5%)

a test performed on a mesh of $64 \times 64 = 4096$ quadratic elements (16 641 degrees of freedom). As shown in Table 3, the best results are obtained when the 64 fronts are used for this size of the problems, and by using 64 fronts, we can reduce the computational time on the symbolic factorization and the backward substitution by 95% and the time of the factorization and forward substitution by nearly 80%.

4.2. The parallel implementation

The computation on each front is completely independent until it touches other fronts and this inherent parallelism can be exploited to develop a successful parallel sparse solver. Furthermore, because the frontal method conducts all its computation on a series of dense linear systems, its parallel computation should be considered as a special application of the parallel solver for the dense linear system, and then the work on the parallel implementation is simplified through a use of a parallel dense solver package which we developed in a previous research [6].

Most research on parallel multifrontal methods (see e.g. [4,5,11]) merely uses the multifrontal method as a mean of parallel computation and considers only the case of $P = P_1$ where P is the number of the fronts and P_1 represents the number of processors. In this study, we not only use the multifrontal method for the parallel computation but also use it as a means to minimize the *fill in* and our implementation must be established on a more general basis with $P \geqslant P_1$ and $P_1 \geqslant 1$. The entire implementation work was planned in three phases:

(1) the sequential multi-frontal method ($P \geqslant 1$ and $P_1 = 1$),

(2) the multi-processors with the single front on each processor ($P = P_1$ and $P_1 \geqslant 1$), and

(3) the general parallel multifrontal method ($P \geqslant P_1$ and $P_1 \geqslant 1$).

Phase II implementation was completed on an Intel IPSC/860 machine and Fig. 7 shows some experimental results. In this experiment, we compared the computational time spent on solving the same size of the problems by using different numbers of processors. As shown in Fig. 7, the uniform meshes of 32×16 quadratic elements (the total degrees of freedom is 2275) and 32×32 quadratic elements (the total degrees of freedom is 4225) are used. For the mesh with 32×16 quadratic elements, the single processor took about 76 s for solving a Laplace problem and it takes only 40 s for solving the same problem with two processors. It takes about 191 s for the single processor to solve a Laplace problem on the mesh of 32×32 quadratic elements and takes about 51 s for the same problem solved using four processors. The experimental result simply indicated that the multifrontal method is able to give a good parallel efficiency in case there is a good loading balance among different processors.

A more extensive experiment on the parallel computation will be performed after the phase III implementation is completed. This part of the work has been moved to the IBM SP2 machine and the new implementation will be based on MPI (Message Passing Interface). After completing this part of the work, we will be able to provide a portable parallel multifrontal solver for solving problems in any parallel machine.

5. Future work and conclusion

The frontal method is originally designed for solving problems with positive linear systems and there is no choice of pivoting in the method. Although numerous problems with indefinite linear systems have been successfully solved by the frontal method, there is no theoretical justification for doing this. Considering the reality of the engineering computation, it is necessary to provide some pivoting function in any direct solver if one expects it to be used to solve the problems with indefinite linear systems. In [8], Hood proposed that because the frontal method does everything in a series of linear systems with dense stiffness matrices, one can attain limited pivoting choices with small modification on the original frontal method. The Hood's algorithm is easy to implement and is largely a retained performance. In [3], Duff further proposed that one can combine a few different fronts together to have larger fronts and achieve better choices of pivoting. The Duff's method will inevitably sacrifice some efficiency but can provide better numerical stability for problems with indefinite linear systems.

The major unfinished work in this research is the implementation of pivoting in the solver. The final version of our solver should be able to apply the Hood's pivoting method automatically during the computation and provide the option of doing pivoting in the way proposed in [3].

The objective of this research is to develop an efficient sparse solver for practical applications and the solver should be able to work on the conventional sequential machines as well as the parallel machines. The generality, robustness and efficiency of the frontal method has been proved by many years' industrial and research applications. Our task here is to develop a practical approach which can combine the classic frontal method with the latest development, including the parallel computation, ordering and pivoting.

Acknowledgement

The support of the Office of Naval Research under Grant N00014-95-1041 is gratefully acknowledged.

References

[1] A. George, Nested dissection of a regular finite element mesh, SIAM J. Numer. Anal. 10 (1972) 345–363.
[2] J.W. Demmel, S.C. Eisenstat, J.R. Gilbert, X.S. Li and J.W.H. Liu, 1995, A supernodal approach to sparse partial pivoting, Private communication.
[3] I.S. Duff and J.K. Reid, The multifrontal solution of indefinite sparse symmetric linear equations, ACM Trans. Math. Software 9 (1973) 302–325.
[4] I.S. Duff, Parallel implementation of multifrontal scheme, Parallel Comput. 3 (1986) 193–204.
[5] C. Geist, Solving finite element problems with parallel multifrontal schemes, in: M.T. Heath, ed., Hypercule Multiprocessors (SIAM, Philadelphia, PA, 1987) 656–661.
[6] P. Geng, J.T. Oden and R.A. van de Geijn, Massively parallel computation for acoustical scattering problems using boundary element methods, J. Sound Vib. 191 (1) (1996) 145–165.
[7] M.T. Heath, E. Ng and B.W. Peyton, Parallel algorithms for sparse linear systems, SIAM Rev. 33 (1991) 420–460.
[8] P. Hood, Frontal solution program for unsymmetric matrices, Int. J. Numer. Methods Engrg. 10 (1976) 379–399.
[9] B. Irons, A frontal solution of program for finite element analysis, Int. J. Numer. Methods Engrg. 2 (1970) 5–32.
[10] D.J. Rose, A graph-theoretic study of the numerical solution of sparse positive definite system of linear equations, in: R.C. Read, ed., Graph Theory and Computing (Academic Press, New York, 1972).
[11] W.P. Zhang and E.M. Lui, A parallel frontal solver on the Alliant FX/80, Comput. Struct. 38 (1991) 203–215.