INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING, VOL. 39, 3999–4020 (1996)

ELEMENT RESEQUENCING FOR USE WITH A MULTIPLE FRONT ALGORITHM

J. A. SCOTT

Department of Computation and Information, Atlas Centre, Rutherford Appleton Laboratory, Oxon OX11 0QX, U.K.

SUMMARY

The multiple front algorithm is an extension of the frontal method to allow parallelism to be exploited in the solution process. The finite-element domain is partitioned into a number of subdomains and a frontal decomposition is performed on each subdomain separately. For a given partitioning of the domain, the efficiency of the multiple front algorithm depends on the ordering of the elements within each subdomain. We look at the limitations of existing element reordering algorithms when applied to a subdomain and consider how these limitations may be overcome. Extensive numerical experiments are performed on a range of practical problems and, on the basis of the results, we propose a new element resequencing algorithm for use with a multiple front algorithm.

KEY WORDS: sparse matrices; frontal methods; Gaussian elimination; finite-element equations

1. INTRODUCTION

In this paper, we are concerned with the solution of $n \times n$ linear systems of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{1}$$

where $\mathbf{A}$ is a large sparse matrix arising from finite-element analysis. The matrix $\mathbf{A}$ is a sum of elemental matrices

$$\mathbf{A} = \sum_{l=1}^{m} \mathbf{A}^{[l]} \tag{2}$$

Each matrix $\mathbf{A}^{[l]}$ has entries only in the principal submatrix corresponding to the variables in element l and represents contributions from this element. This principal submatrix is assumed to be dense. The matrix $\mathbf{A}$ may be unsymmetric but the form (2) implies that the sparsity pattern is symmetric with non-zero diagonal entries. One possible direct solution method for (1), and the one which is still frequently the method of choice in many structural engineering applications, is the frontal method (see, for example, Irons,[1] Hood,[2] and Duff.[3]).

The efficiency of a frontal scheme, in terms of both storage and computation time, is dependent upon the ordering of the elements. This is because, in the frontal method, the system matrix $\mathbf{A}$ is never assembled explicitly but the assembly and Gaussian elimination processes are interleaved, with each variable being eliminated as soon as its row and column are fully summed, that is, after its last occurrence in a matrix $\mathbf{A}^{[l]}$. This allows all intermediate working to be performed in a full matrix, termed the *frontal matrix*, whose rows and columns correspond to variables that have not yet been eliminated but occur in at least one of the elements that have been assembled. Since the

CCC 0029–5981/96/233999–22
© 1996 by John Wiley & Sons, Ltd.

Received 28 March 1995
Revised 3 January 1996

order of the frontal matrix increases when a variable appears for the first time and decreases whenever a variable is eliminated, the order in which the elements is input is critical.

To formalize this, let us introduce some notation. If m is the number of finite elements in the domain and $fsize_i$ is the number of variables in the frontal matrix after the assembly of the ith element, the maximum frontsize $fmax$ is defined to be

$$fmax = \max_{1 \leqslant i \leqslant m} \{fsize_i\} \tag{3}$$

and the root mean-square (r.m.s) frontsize $frms$ is defined to be

$$frms = \left(\frac{1}{m}\sum_{i=1}^{m} fsize_i^2\right)^{\frac{1}{2}} \tag{4}$$

In a frontal algorithm, the maximum amount of storage required for the frontal matrix during the Gaussian elimination is dependent upon the maximum frontsize. Moreover, the number of floating-point operations performed after the assembly of the ith element and before the assembly of the $(i+1)$th element is dependent upon $fsize_i^2$. The elements should therefore be numbered in such a way as to reduce $fmax$ and $frms$.

In recent years, many algorithms for automatically ordering finite elements have been proposed in the literature. These include the methods described by Akin and Pardue,[4] Bykat,[5] Razzaque,[6] Pina,[7] Sloan and Randolph,[8] Fenves and Law,[9] Sloan,[10] Shephard *et al.*,[11] Duff *et al.*,[12] Kaveh,[13] Koo and Lee,[14] Medeiros *et al.*,[15] and Paulino *et al.*[16] Each of these algorithms is designed to reorder *all* the elements in the finite-element domain (they are global reordering schemes). However, in a multiple front method, the finite-element domain is partitioned into a (small) number of subdomains and a frontal decomposition is performed on each subdomain separately.[17,18] For a multiple front method, a global reordering algorithm is unlikely to provide efficient element orderings. It is the need to develop element ordering schemes which are efficient when applied to a subdomain that motivates the current study.

The remainder of this paper is organized as follows. We review the multiple front algorithm in Section 2 and in Section 3 we introduce some basic concepts from graph theory. In Section 4 we discuss the main features of the element reordering algorithm used by the Harwell Subroutine Library[19] (HSL) routine MC43, which is designed to resequence elements for use with the HSL frontal code MA42.[20,21] We also look at why this algorithm is unsuitable for reordering elements for use with the multiple front algorithm. In Section 5 we propose two new schemes for ordering elements in a subdomain and in Section 6 we report on the performance of these methods when used on a range of practical problems. Finally, in Section 7 concluding remarks and comments are made.

2. THE MULTIPLE FRONT ALGORITHM

The power of frontal schemes comes from the fact that they are able to solve quite large problems with modest amounts of high-speed memory and the fact that they can use dense linear algebra kernels, in particular the Level 3 Basic Linear Algebra Subprograms (BLAS)[39] for the numerical factorization. However, there are two main deficiences with frontal schemes. The first is that far more arithmetic may be done than is required by the numerical factorization. This is illustrated by the matrix LOCK3491 from the Harwell–Boeing Collection.[25,26] If the original problem is reordered using the HSL routine MC43, $95*10^6$ floating-point operations (flops) are required by the HSL frontal solver MA42 to factor the matrix, but if the HSL multifrontal code MA37 is used, the flop count reduces to $40*10^6$. The second deficiency of the frontal method is that there is little scope for parallelism, other than that which can be obtained within the higher level BLAS. One way of overcoming these problems is to extend the basic frontal algorithm to use multiple fronts. The use of multiple fronts depends upon being able to decouple the underlying 'domain' and eliminate variables within each subdomain independently. The logical conclusion of this approach is the so-called multifrontal algorithm (see, for example, Duff and Reid,[40]), where many fronts are started simultaneously. The variables that are not immediately eliminated are combined to form new elements or fronts which are then merged with other new elements or original elements according to an assembly tree. The initial fronts (which may number nearly half the total) and the assembly tree are identified by a flop-reducing ordering such as minimum degree or nested dissection. In this study we use the term multiple fronts for algorithms for solving systems of finite-element equations which partition the finite-element domain into a number of subdomains and perform a frontal decomposition on each subdomain using an element-by-element ordering in a somewhat similar fashion to Benner *et al.*[27] and Zhang and Liu.[41] We reserve the term multifrontal for algorithms for solving systems of equations (which may or may not arise from a finite-element application) where a symbolic analysis to construct an elimination tree is first performed using a standard flop-reducing ordering. In the remainder of this section we briefly discuss the use of multiple fronts.

Consider a finite-element domain which has been partitioned into (non-overlapping) subdomains. Variables which belong to a single subdomain are termed *internal variables* and variables which lie on the interface boundaries of the subdomains are called *interface variables*. In general, each subdomain will contain some elements whose variables are all internal variables and some elements with both internal and interface variables. Elements containing only internal variables are called *internal elements* and those with one or more interface variables are called *interface elements*. Internal variables may be eliminated as soon as they are fully summed (provided, of course, that stability criteria are satisfied) but interface variables cannot be eliminated within a subdomain since they are shared by more than one subdomain. In a multiple front algorithm, a frontal solver is applied to each subdomain separately. Since the factorizations of the subproblems are independent, this can be done in parallel. At the end of the assembly and elimination processes for the subdomains, for each subdomain i there will remain a subdomain frontal matrix $\mathbf{F}_i$ and a corresponding frontal right-hand-side vector $\mathbf{c}_i$ (or matrix, if there are multiple right-hand sides) satisfying

$$\mathbf{F}_i \mathbf{y}_i = \mathbf{c}_i \tag{5}$$

If there are *nsub* subdomains, *nsub* equations of the form (5) are generated and these may be assembled to give

$$\mathbf{F}\mathbf{y} = \mathbf{c} \tag{6}$$

where the order of the matrix $\mathbf{F}$ is the number of interface variables (plus any internal variables not eliminated for stability reasons). By treating each of the subdomain frontal matrices $\mathbf{F}_i$ as an elemental matrix, the system (6) may also be solved by a frontal scheme. Once (6) has been solved, the results for the interface variables must be passed back to the subdomains so that back-substitution for the internal variables can be performed. The back-substitutions on the subdomains may also be performed in parallel. For further details, see Duff and Scott,[17,18] who also provide numerical examples illustrating the speed-ups which can be obtained when the multiple front algorithm is implemented in a parallel environment.

In the multiple front algorithm, once an interface variable has entered the subdomain frontal matrix it cannot be eliminated. Therefore, an element ordering scheme which yields an efficient global ordering may give very inefficient orderings when applied to a single subdomain. The aim of this study to develop a new element ordering scheme which is efficient when used with the multiple front algorithm.

3. GRAPHS AND FINITE-ELEMENT DOMAINS

Associating graphs with finite elements is useful when developing element reordering algorithms. In this section, we briefly recall some basic definitions from elementary graph theory which are of relevance to this paper.

An *undirected graph* G is defined to be a pair (V, E), where V is a finite set of *nodes* (or *vertices*), and E is a finite set of *edges* defined as unordered pairs of distinct nodes. A *labelling* (or *ordering*) of a graph $G = (V, E)$ with n nodes is a bijection of $\{1, 2, \ldots, n\}$ onto V. The integer i $(1 \leqslant i \leqslant n)$ assigned to a node in V by a labelling is called the *label* (or *number*) of that node. Two nodes i and j in G are said to be *adjacent* if $(i, j) \in E$. The *degree* of a node $i \in G$ is defined to be the number of nodes in G which are adjacent to i, and the *adjacency list* for i is the list of these adjacent nodes. A *path of length* k in G is an ordered set of distinct nodes $(i_0, i_1, \ldots, i_k)$, where $(i_{j-1}, i_j) \in E$ for $1 \leqslant j \leqslant k$. Two nodes are *connected* if there is a path joining them. A graph G is *connected* if each pair of distinct nodes is connected. Otherwise, G is disconnected and consists of two or more *components.*

The *distance* between nodes i and j in a connected graph G (or in a component of a disconnected graph) is denoted by $d(i, j)$, and is defined to be the number of edges on the shortest path connecting them. The *diameter* $D(G)$ of G is the maximum distance between any pair of nodes. That is,

$$D(G) = \max \{d(i, j) \mid i, j \in V\}$$

Nodes at opposite ends of a diameter of G are known as *peripheral* nodes. A *pseudo-diameter* $\delta(G)$ is defined by any pair of nodes i and j in G for which $d(i, j)$ is close to $D(G)$. A pseudo-diameter may be slightly less than, or equal to, the true diameter and is found by some heuristic algorithm. Nodes defining a pseudo-diameter are termed *pseudo-peripheral* nodes.

A *level structure rooted at a node* $r \in V$ is defined as the partitioning of V into levels $l_1, l_2, \ldots, l_{h(r)}$ such that

(i) $l_1 = \{r\}$
(ii) for $i > 1$, l_i is the set of all nodes that are adjacent to nodes in l_{i-1} but are not in $l_1, l_2, \ldots, l_{i-1}$.

The level structure rooted at node r may be expressed as the set $L(r) = \{l_1, l_2, \ldots, l_{h(r)}\}$, where $h(r)$ is the total number of levels and is termed the *depth.* The number of nodes on level i is the *width of level* i and is denoted by $|l_i|$. The *width* of the level structure $L(r)$ is given by

$$w(r) = \max_{1 \leqslant i \leqslant h(r)} |l_i|.$$

A *list* is an ordered set. A *priority queue* is a list from which deletions or extractions are made on the basis of a priority function.

A *finite-element domain* is a collection of finite elements in which the elements are joined at their common boundaries and vertices. Finite-element nodes may lie at vertices, along the sides, on the faces, or within the element itself. Associated with each finite-element node is a set of one or more variables corresponding to the freedoms at that node. A convenient way of associating a graph with a finite-element domain consists of choosing the nodes of the graph to be the finite elements and using the interconnection of the finite elements to determine the edges. Relabelling the nodes of this element connectivity graph is equivalent to reordering the elements of the associated finite-element domain and an algorithm which does this is termed a *direct* element reordering algorithm.[12]

There are several possible ways to use the interconnection of the finite elements to determine the edges of an element connectivity graph. Bykat[5] defines two elements to be adjacent to one another whenever they share a common edge and describes his algorithm in detail for planar triangular elements. Fenves and Law[9] generalize this definition to problems in n dimensions, $n = 1, 2, 3$. They define two elements in n dimensions to be adjacent whenever they possess a common boundary of $(n - 1)$ dimensions. Thus in three dimensions two volumetric elements are adjacent if they share a common two-dimensional boundary face; in two dimensions planar elements are interconnected by one-dimensional boundary lines; and one-dimensional finite elements are adjacent if they have a common finite-element node. Fenves and Law noted that the adjacency of elements cannot always be completely represented by this definition of adjacent elements, since n-dimensional elements are not necessarily connected through $(n - 1)$-dimensional boundaries. In addition, adjacent finite elements do not necessarily have the same geometric boundaries. In such examples, the element connectivity graph may become disconnected, and each component must be numbered independently. This contributes to the difficulties associated with attempting to use this definition of element adjacency.

To try and avoid these difficulties, Kaveh[42] defines two elements of dimension n_1 and n_2, respectively, to be adjacent if they have a common face which is of dimension $\min(n_1, n_2) - 1$. Duff *et al.*[12] adopt an even simpler definition: they define two elements to be adjacent to each other whenever they have one or more variables in common. Using this definition, it is not difficult to generate the associated element connectivity graph; the user does not need to provide information on the different types of elements in the grid other than a list of the variables associated with each finite element. From their numerical experiments, Duff *et al.* report that using this definition to generate the element connectivity graph and then employing their direct element reordering algorithm did not, in general, lead to a significant increase in the frontsizes compared with those obtained using the element connectivity graph resulting from the adjacency definition of Fenves and Law. The definition of adjacency introduced by Duff *et al.* has recently been used by Paulino *et al.*,[16] who term the resulting connectivity graph the *element communication graph*. Throughout the remainder of this paper we will use the Element Communication (EC) graph.

4. ELEMENT REORDERING USING MC43

In this section we look at the key features of the direct element reordering algorithm used by the Harwell Subroutine Library code MC43 and illustrate the shortcomings of the algorithm if it is used in conjunction with the multiple front method. The MC43 element reordering algorithm exploits the profile reduction algorithm of Sloan.[10] It has three distinct steps:

(1) selection of a pair of pseudo-peripheral elements (nodes),
(2) element relabelling,
(3) computation of the maximum frontsize.

In the first step, for each component of the Element Communication (EC) graph, a pair of pseudo-peripheral elements (nodes) is located. It has been found that, because these elements tend to yield rooted level structures which are deep and narrow, they make good starting elements for profile and wavefront reduction algorithms (see Gibbs[22] and Sloan and Randolph.[8]). The procedure used in MC43 to locate pseudo-peripheral elements is a modification of that described by Gibbs *et al.*[23] and George and Liu.[24] A starting element $s \in G$ of minimum degree is chosen, and the rooted level structure $L(s)$ is generated. The last level set $l_{h(s)}$ is 'shrunk' by retaining only elements with distinct degrees, ties being broken arbitrarily. The level structures rooted at each element in this reduced set (selected in order of increasing degree) are then computed. If, for some $r \in l_{h(s)}$, the depth of $L(r)$ exceeds $h(s)$, r replaces s as the starting element, and the procedure is repeated. If no such element r is found, and e is the element in $l_{h(s)}$ whose associated level structure has the smallest width, the elements s and e are chosen as pseudo-peripheral elements. A 'short circuiting' strategy by which wide level structures are rejected as soon as they are detected is incorporated. This algorithm for locating s and e has also recently been used by Medeiros *et al.*[15]

In the second step of the algorithm, the elements in each component of the EC graph are renumbered to obtain a profile which is smaller than that given by the original labelling of the graph. The pseudo-peripheral elements s and e serve as starting and end elements for the relabelling within their component. The rooted level structure $L(e)$ is generated and the distance $d(e, i)$ of each element i from the end element e is computed. The starting element s is relabelled as element one and a list of elements that are eligible to receive the next label is formed. At each stage in the relabelling process the list of eligible elements comprises those elements which are either adjacent to a element which has been relabelled or are adjacent to a element which is itself adjacent to a relabelled element. The next element to be given a new number is the element among all eligible elements with the highest priority, where the priority P_i of element i is defined to be

$$P_i = -W_1 * ngain(i) + W_2 * d(e, i) - W_3 * nadj(i). \tag{7}$$

Here W_1, W_2 and W_3 are integer weights, $ngain(i)$ is the number of variables element i will introduce into the front less the number that can then be eliminated, and $nadj(i)$ is the number of elements adjacent to element i which have not yet been relabelled. The basic idea of the algorithm is that, during the reordering process, elements which will make only a small increase to the frontsize (or will decrease the frontsize) and which are at a large distance from the end element are labelled first. The third term means preference is also given to elements which have only a small number of unlabelled neighbours. The values assigned to the weights determines the importance of each of these criteria. As a result of numerical experimentation, in MC43 the weights have the values $W_1 = 10$, $W_2 = 5$, and $W_3 = 1$. Since $W_3 = 1$ is much smaller than W_1 and W_2, the number of unlabelled neighbours is essentially only used to resolve ties (see Duff *et al.*[12] for further discussion). We remark that although the EC graph does not take into account the number of variables in each element, the priority P_i of element i given by (7) does depend on the number of variables it has.

Once all the elements have been renumbered, MC43 computes the maximum frontsize for the new ordering. If this is larger than the maximum frontsize for the initial ordering, the user is warned that no reduction in the maximum frontsize was achieved and the initial ordering is retained. The value of the maximum frontsize returned by MC43 is useful if the HSL frontal solver MA42 is to be employed since it assists the user in choosing the size of the frontal matrix required and the sizes of the files which will hold the **LU** factors of **A**.

Experience has shown that, when used with MA42, the orderings generated by MC43 are efficient.[12,20,21] There are, however, weaknesses in each step of the MC43 algorithm if it is employed to resequence the elements in a subdomain for the multiple front algorithm. When locating pseudo-peripheral elements, MC43 does not distinguish between internal and interface elements. As a result, MC43 may choose as a starting element an element containing interface variables. In general, this will be a poor choice since the element numbering should start with elements lying away from the interface boundaries so as to delay the introduction of interface variables into the subdomain front for as long as possible. To illustrate this consider an $N \times 2N$ rectangular domain of rectangular elements. Suppose the elements are initially ordered pagewise parallel to the side of length N and the domain is then partitioned into two, with elements 1 to N^2 in subdomain 1 and elements $N^2 + 1$ to $2N$ in subdomain 2. If MC43 is applied to each subdomain (with the ith element in subdomain 2 corresponding to element $N^2 + i$ in the original domain) the initial orderings will be returned. That is, the ordering for subdomain 1 will start at the exterior boundary and work towards the interface boundary while for subdomain 2 the elements along the interface boundary will be reordered first and those on the exterior boundary last. MC43 is not able to distinguish between these two orderings but, in the multiple front algorithm, the ordering for subdomain 2 will be much less efficient than that for subdomain 1. For example, if $N = 8$ and there are five variables at the corners, mid-points of the sides, and centre of each element, numerical experimentation shows that in the multiple front algorithm the two subdomains have r.m.s. frontsizes of 111·5 and 188·1, respectively, and require $1{\cdot}86*10^7$ and $5{\cdot}77*10^7$ floating-point operations, respectively.

In the second step of the MC43 algorithm, the reordering is based on the priorities of the elements computed using (7). As we have already seen, an element has a high priority if (a) the net gain it makes to the frontsize is small, (b) it is at a large distance from the end element, and (c) it has a small number of neighbours which have not already been relabelled. Since MC43 is a global algorithm, when calculating the net gain to the frontsize, it is assumed that any variable appearing for the last time may be eliminated. For a subdomain this is no longer true since interface variables may not be eliminated within the subdomain. Failure to take this into account can lead to interface elements being assigned a high priority and being relabelled early in the reordering algorithm. Furthermore, MC43 may select as the end element e an element lying a long way from the interface boundary. In this case, $d(e, i)$ will be large if i is an interface element and will again lead to a high priority for interface elements. This is unlikely to yield an efficient ordering.

In the final step, MC43 returns the original and new maximum frontsizes to the user. When a frontal scheme is applied to a domain which has not been partitioned, in general the maximum frontsizes will occur after some, but not all, the elements have been assembled. However, when the domain is partitioned into subdomains, the maximum frontsize may occur after all the elements have been assembled, and the remaining front then comprises the interface variables (plus any internal variables which have not been eliminated for stability reasons). Since MC43 is unable to distinguish between interface and internal nodes, the maximum frontsizes it returns may be significantly smaller than the frontsizes required by the multiple front method. As already noted, if MC43 finds that the maximum frontsize for the ordering it generates is larger than for the original ordering, it will reject the new ordering and retain the original ordering. But in the multiple front algorithm, the rejected ordering can be more efficient then the accepted ordering. To illustrate this we took the test problem AEAC5081 and partitioned the domain into 16 subdomains using the code Chaco (see Section 6 for details of the test problems and Chaco). MC43 was applied to subdomain 12. The maximum frontsize for the initial ordering was 61. MC43 generates an ordering with a maximum frontsize of 47 so MC43 accepts the new ordering. In the multiple front algorithm, the new ordering gives maximum and r.m.s. frontsizes of 131 and 91·1, respectively, while the rejected (original) ordering yields frontsizes of 100 and 72·1. Thus, for this subdomain, the ordering which was rejected by MC43 provides a more efficient ordering than the one which was accepted.

From the above discussion it is clear that the global reordering algorithm implemented by the code MC43 is not suitable for reordering the elements in a subdomain. Other global reordering schemes applied to a subdomain suffer similar problems to those experienced by MC43 because they too have no concept of internal and interface elements. To be able to use a multiple front algorithm effectively we need to develop a reordering scheme which takes account of interface variables; this is the subject of the rest of this paper.

5. REORDERING ELEMENTS IN A SUBDOMAIN

Algorithms to reorder elements in a subdomain require knowledge of the interface variables. The interface variables also need to be known when the frontal code MA42 is used to implement the multiple front algorithm outlined in Section 2. We have developed a Harwell Subroutine Library package, MA52, which allows MA42 to be used to implement a multiple front algorithm. MA52 can also be used to generate the list of interface variables for a subdomain. Following the terminology of Duff and Scott,[17,18] this list is termed the *guard element*.

Intuitively, reordering of a subdomain should begin well away from the interface boundary. We consider two approaches for finding a suitable starting element and we compare the performance of the two approaches in Section 6. Throughout Sections 5.1–5.4 we will assume that the EC graph is connected; in Section 5.5 we will consider the case of it having more than one component.

5.1. Approach I

In Approach I we locate starting and end elements s and e using a modified version of the algorithm used by MC43. The steps in finding s and e are as follows.

(1) Generate a list of interface variables. Flag all elements containing interface variables as interface elements. The unflagged elements are internal elements.
(2) Select an element s of minimum degree (that is, an element with the smallest number of adjacent elements), if necessary breaking ties by giving preference to internal elements.
(3) Generate the rooted level structure $L(s) = \{l_1, l_2, \ldots, l_{h(s)}\}$.
(4) Sort the elements in the last level $l_{h(s)}$ in ascending order of their degrees.
(5) Shrink $l_{h(s)}$ to a list $\bar{l}$ by retaining only one element of each degree, if necessary breaking ties by giving preference to internal elements.
(6) Set $w = \infty$.
(7) For each element $r \in \bar{l}$ in order of increasing degree, generate $L(r)$. If $h(r) > h(s)$ and $w(r) \leqslant w$, set $s = r$ and **go to 4**. Else if $w(r) \leqslant w$, set $e = r$ and $w = w(r)$.
(8) If s is an internal element **go to 9**. Else if s is an interface element and e is an internal element then set $s = e$ and $e = s$ and **go to 9**. Else s and e are interface elements. Generate $L(s)$ and consider the elements in the middle level set $l_{h(s)/2}$. If there are no internal elements **go to 9**. Else choose s to be the internal element in $l_{h(s)/2}$ of minimum degree, breaking ties arbitrarily.
(9) Accept s and e as starting and end elements, respectively.

We remark that the choice for s made in step 8 in general ensures that we begin the element renumbering away from an interface boundary. The case when the elements s and e found in step 7 are both interface elements may occur if the subdomain has interface variables on all sides. Choosing the starting element to lie in the middle set $l_{h(s)/2}$ is then a way of starting the element reordering with an element at an approximate centre of the subdomain.

5.2. *Approach II*

In our second approach, we treat the guard element g for the subdomain as an extra element. Suppose there are *nelt* finite elements in the subdomain and let g be labelled element $nelt + 1$. Let the element communication graph for this augmented set of elements be denoted by EC(g). Recall that in an element communication graph, two elements are adjacent if they have a variable in common. Since the variables in the guard element are the interface variables, in the EC(g) graph the guard element is adjacent to all the interface elements. By considering the rooted level structure $L(g)$, we can locate elements which lie away from the interface boundary and which should make a good choice for the starting element for reordering elements in the subdomain.

Using this idea, the steps in Approach II for choosing a starting element s are as follows:

(1) Generate the guard element g.
(2) Generate the element communication graph EC(g).
(3) Generate the rooted level structure $L(g) = \{l_1, l_2, \ldots, l_{h(g)}\}$.
(4) Shrink $l_{h(g)}$ to a list $\bar{l}$ by retaining only one element of each degree (breaking ties arbitrarily).
(5) Set $h = 1$.
(6) For each element $r \in \bar{l}$ generate $L(r)$. If $h(r) > h$, set $h = h(r)$, $s = r$.

In this way, the starting element is chosen to be at a maximum distance from the interface boundary. For a subdomain with interface variables on all sides, Approach II provides a straightforward way of locating an element at the approximate centre of the subdomain. An advantage of this approach is that it avoids the need to locate pseudo-peripheral elements and, except when all the elements in the subdomain are interface elements, the starting element will always be an internal element. The end element is taken to be the guard element, that is, $e = g$. The distance $d(e, i)$ is the distance of element i from the interface boundary. A disadvantage of Approach II is that (s, e) may not provide a good estimate to the diameter of the element communication graph. Furthermore, the EC(g) graph has more edges than the EC graph, the number of extra edges being dependent upon the number of interface variables. The EC(g) graph therefore takes longer to generate than the EC graph and can lead to Approach II being slower than Approach I. However, in our experience, the difference in the times for Approaches I and II was insignificant compared with the time taken to solve the underlying finite-element problem using the multiple front algorithm (see Tables II–V in Section 6).

5.3. *Element relabelling*

Approaches I and II are procedures for choosing starting and end elements s and e for a subdomain. The algorithm we use to relabel the remaining elements in the subdomain is based on that used by MC43 but is modified to take into account interface variables. A priority queue is used where now the priority P_i of element i given by

$$P_i = -W_1 * ngain(i) + W_2 * d(e, i) - W_3 * nadj(i) - W_4 * nint(i) \qquad (8)$$

Here W_1, W_2, W_3 and W_4 are integer weights. The quantity *ngain*(i) is the number of variables element i will introduce into the front less the number of internal variables that can then be eliminated. Since interface variables cannot be eliminated within a subdomain, if element i contains one or more interface variables, *ngain*(i) will be higher than for the same element in a single domain problem and the element will therefore have a lower priority. As in (7), *nadj*(i) is the number of elements adjacent to element i which have not yet been relabelled. The quantity *nint*(i) is the number of interface variables the element will introduce into the front and is non-zero only if i is an interface element. The aim of (8) is to give a high priority to internal elements. For Approach II, in which the end element e is the guard element, $d(e, i)$ will be large if element i is well away from the interface boundary and this will lead to these elements being given a high priority. Moreover, for Approach II, *nint*(i) will be non-zero only if $d(e, i)$ is equal to 1 and so for this approach we set $W_4 = 0$. On the basis of our numerical experiments, we suggest choosing the weights to have values $W_1 = 12$, $W_2 = 6$, $W_3 = 1$, and (for Approach I only) $W_4 = 2$. As in MC43, the number of unlabelled neighbours only acts to resolve ties. These weights were used in the numerical experiments reported on in Section 6. We found that, in general, small changes to these weight values had no significant effect on the quality of the orderings obtained but, if a weight was omitted or the ratios between the weights were substantially altered, for some of the subdomains in our test problems, the resulting orderings had significantly larger frontsizes. Examples illustrating this are included in Section 6 (see Table VI).

5.4. Selecting the element order

In MC43, the maximum frontsize is computed for the original and new element orderings. If the new ordering has a maximum frontsize which is no smaller than the maximum frontsize of the original ordering, the original ordering is retained. For the subdomain problem, returning the maximum frontsize which takes into account the interface variables is useful since the frontal solver will need a frontal matrix of at least this size. However, it is not necessarily the appropriate criteria to use to choose between the original and new orderings. As we have already mentioned, for a subdomain, the maximum frontsize may occur after all the elements have been assembled and, in our experience, the reordering algorithms often failed to produce an ordering with a maximum frontsize which was smaller than that for the original ordering. An alternative criteria is to use the r.m.s. frontsize so that if the new ordering provides a smaller r.m.s. frontsize than the original ordering, the new ordering is chosen.

In general, we have found that the choice we have made between the original user-supplied ordering and the new element orderings based on the r.m.s. frontsize has been the correct one. That is, when the frontal solver has been applied to the subdomain, the number of floating-point operations required by the accepted ordering has been fewer than the number required by the rejected ordering. However, even if stability considerations do not cause pivots to be delayed, it is possible that the accepted ordering may still result in the multiple frontal solver requiring more floating-point operations than would be needed by the rejected ordering. When computing the maximum and r.m.s. frontsizes for a given ordering, it is straightforward to also compute the number of floating-point operations the Harwell frontal solver MA42 will require when applied to the subdomain (assuming stability considerations do not cause variables to remain in the front once they are fully summed). In our numerical experiments we compute the number of floating-point operations and choose the ordering for which this is smallest. In this way, when combined with the multiple front algorithm, the element ordering we use never requires more work than the original ordering.

5.5. Coping with more than one component

For the single domain problem, if the EC graph has more than one component, each component is reordered separately. The maximum frontsize and the r.m.s. frontsize are independent of the order in which the components are renumbered. However, for the subdomain problem we need to consider the case of more than one component more carefully since the quality of the element ordering will depend upon the order in which the components are renumbered. To illustrate this consider, for example, a subdomain with an associated EC graph with two components, a and b. Suppose the elements in component a are passed to the frontal solver first. Let $\mathbf{F}_a$ denote the frontal matrix after the last element in component a has been assembled. $\mathbf{F}_a$ is of order at least $nsub_a$, where $nsub_a$ is the number of interface variables lying within component a. After some of the elements in component b have been assembled the frontal matrix will be of the form

$$\mathbf{F} = \begin{pmatrix} \mathbf{F}_a & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{F}} \end{pmatrix} \tag{9}$$

The frontal solver MA42 treats the frontal matrix as dense and so is unable to exploit the zero blocks in (9). Thus unnecessary work will be performed and, unless $nsub_a$ is small, it will generally be more efficient to split the subdomain into two subdomains, corresponding to the two components a and b. If the subdomain is not partitioned, we renumber the components in increasing order of the number of interface variables they contain. Note that, if there are any components with no interface elements, these components can be renumbered first using the standard MC43 algorithm.

In Approach II, the EC(g) graph is generated. Although the EC graph may have more than one component containing interface variables, the EC(g) graph has only one such component. We could apply the two-step algorithm described in Sections 5.2 and 5.3 directly to this single component graph but this can lead to the reordering 'jumping' about between the components (that is, some of the elements in the first component may not have been renumbered before some of the elements in another component are renumbered). As might be expected, numerical experimentation shows that this can result in poor orderings. Instead, when the EC(g) graph has more than one component, we modify Approach II so that, when constructing rooted level structures $L(r)$ using the EC(g) graph, we only include elements lying in the same component of the EC graph as the root r. This is equivalent to splitting the guard element into *ncomp* guard elements, where *ncomp* is the number of components in the EC graph. Each of these *ncomp* guard elements is a list of the interface variables lying in the corresponding component of the EC graph. The improvements this can give may be illustrated using problem LOCK1074 from the Harwell–Boeing sparse matrix collection.[25,26] We partitioned the finite-element domain into 4 using the code Chaco (see Section 6 for details) and generated the EC graph for each subdomain. The EC(g) graphs for subdomains 3 and 4 were found to have 3 and 2 components, respectively. Applying Approach II to the single component EC(g) graphs for these subdomains gave r.m.s. frontsizes of 185·8 and 196·3, respectively. However, using the proposed modification, the r.m.s. frontsizes were reduced to 118·4 and 114·7, respectively.

6. THE TEST PROBLEMS

In this section, we report the results of using our proposed subdomain element resequencing algorithms on a range of test problems. All our numerical experiments were performed on a CRAY Y-MP81 using single precision arithmetic and all the reported CPU timings are in seconds. In our experiments, we have only used a single processor of the CRAY but since the reordering of the elements in each subdomain are independent, this can be done in parallel and, as already discussed, the frontal method can be applied to each subdomain in parallel. Throughout this section, all computed frontsizes allow for the interface variables, that is, they are the frontsizes for the multiple front algorithm. For the MC43 element ordering the frontsizes are computed outside the MC43 code.

We first use a model problem in which the elements are nine-node rectangular elements with nodes at the corners, mid-points of the sides, and centre of the element. A parameter to the element generation routine determines the number of variables per node. We have chosen this parameter to be five in our numerical experiments. The elements are arranged in a rectangular grid of size $4N \times 4N$ and are initially ordered pagewise. The number of variables is $5(4N + 1)^2$. The grid is partitioned into k equal subdomains of order $4N \times 2N$ if $k = 2$, $2N \times 2N$ if $k = 4$, $N \times 2N$ if $k = 8$, and $N \times N$ if $k = 16$. If k is equal to 2 or 4, it is straightforward to order the elements efficiently within each subdomain. However, if the number of subdomains is 8 or 16, even for this simple problem it is less apparent how to do this. In Tables I(a)–I(c) we present results for MC43, Approach I, and Approach II applied to the 4-, 8-, and 16-subdomain problem, respectively, with $N = 12$. For each subdomain, the maximum frontsize *fmax* and the r.m.s. frontsize *frms* for the different element orderings are given and the ordering which gives the smallest r.m.s. frontsize is in bold typeface. The CPU times required by the element-reordering algorithms for 2, 4, 8, and 16 subdomains are shown in Table II. In addition, the number of floating-point operation (flops) and the CPU time taken by the multiple front algorithm are given. The number of floating-point operations includes the floating-point operations for the frontal algorithm in each subdomain together with the floating-point operations needed to solve the interface problem (6) using the HSL code MA42.

From Tables I(a)–I(c) we see that the orderings generated by Approaches I and II have smaller r.m.s. frontsizes than the MC43 ordering and, in general, they also have smaller maximum frontsizes. For each subdomain in the 4- and 16-subdomain problems, Approach II generates an element ordering with a smaller r.m.s. frontsize than Approach I. In some subdomains in the 8-subdomain problem, Approach I does slightly better than Approach II. However, the quality of the ordering generated by Approach I appears to be more dependent on the initial ordering than the Approach II ordering is. For example, for the 4-subdomain problem, Approach II produces orderings with the same maximum and r.m.s. frontsizes for each of the subdomains but Approach I gives orderings which have slightly different r.m.s. frontsizes in each subdomain. From Table II we see that MC43 is cheaper to use than Approaches I and II, but the small extra cost entailed in using one of the new approaches is more than justified by the savings which the resulting element orderings give in the CPU timings for the multiple front algorithm. We also observe the amount of work can be reduced by partitioning the domain into more subdomains. However, as the number of subdomains increases, the number of interface variables increases and the work involved in solving the interface problem also increases and this rapidly dominates the computation costs. One possible way of increasing efficiency further is by nesting the multiple front algorithm (see, for example, Benner *et al.*[27]).

In addition to the model problem, a range of problems arising from practical applications have been used to test and assess the quality of the element reordering algorithms discussed in this paper. The problems range in size from 360 to 23446 elements. A brief description of each test problem is given in Table III. Problems with origin HB were taken from the widely used Harwell–Boeing sparse matrix collection.[25,26] Those with origin AEAT were supplied by Andrew Cliffe of AEA Technology, Harwell Laboratory; those with origin DNVR came from A. C. Damhaug of Det Norske Veritas Research, Norway; and those with origin RALPAR were supplied by R. F. Fowler of the Rutherford Appleton Laboratory. For the RALPAR problems, only element connectivity information was available (that is, lists of the finite-element nodes belonging to each element were supplied but not the variables associated with each of the nodes). Thus, the results we present for the RALPAR problems may only be regarded as an indication of the relative performance of the different reordering algorithms. They will be a good indication of performance if the number of variables per finite-element node is relatively constant. For the

Table I(a). A comparison of MC43, Approach I, and Approach II for the 4-subdomain model problem

	MC43		Approach I		Approach II	
Subdomain	*fmax*	*frms*	*fmax*	*frms*	*fmax*	*frms*
1	505	393·7	505	393·7	505	**367·9**
2	515	403·1	505	394·2	505	**367·9**
3	745	623·1	505	395·2	505	**367·9**
4	750	632·7	505	394·7	505	**367·9**

Table I(b). A comparison of MC43, Approach I, and Approach II for the 8-subdomain model problem

	MC43		Approach I		Approach II	
Subdomain	*fmax*	*frms*	*fmax*	*frms*	*fmax*	*frms*
1	385	**277·8**	385	**277·8**	385	**277·8**
2	625	**417·1**	625	**417·1**	625	**417·1**
3	625	**417·1**	625	**417·1**	625	**417·1**
4	395	286.6	385	**278·2**	395	286·6
5	505	390·9	405	**277·8**	385	294·4
6	735	523·0	625	**417·1**	625	**417·1**
7	735	523·0	625	**417·1**	625	**417·1**
8	510	400·0	385	**278·2**	385	294·4

Table I(c). A comparison of MC43, Approach I, and Approach II for the 16-subdomain model problem

	MC43		Approach I		Approach II	
Subdomain	*fmax*	*frms*	*fmax*	*frms*	*fmax*	*frms*
1	265	210·3	265	210·3	265	**198·3**
2	385	**280·7**	385	**280·7**	385	**280·7**
3	385	**280·7**	385	**280·7**	385	**280·7**
4	275	219·3	265	211·1	265	**198·3**
5	385	325·9	385	**280·7**	385	**280·7**
6	500	390·2	500	385·7	500	**366·8**
7	500	390·2	500	385·7	500	**366·8**
8	395	334·7	405	321·7	385	**280·7**
9	385	325·9	385	**280·7**	385	**280·7**
10	500	390·2	500	385·7	500	**366·8**
11	500	390·2	500	385·7	500	**366·8**
12	395	334·7	405	321·7	385	**280·7**
13	385	321·0	285	210·3	265	**198·3**
14	495	385·2	385	**280·7**	385	**280·7**
15	495	385·2	385	**280·7**	385	**280·7**
16	390	330·2	385	210·3	265	**198·3**

Table II. CPU times (seconds) for reordering the elements together with floating-point operation counts (flops) and CPU times (seconds) for solving the model problem using the multiple front algorithm

	MC43			Approach I			Approach II		
Number of subdomains	Recorder CPU time	Total flops	Solve CPU time	Recorder CPU time	Total flops	Solve CPU time	Recorder CPU time	Total flops	Solve CPU time
2	0·50	1·82E + 10	82·46	0·63	1·7EE + 10	79·45	0·55	**1·70E + 10**	77·68
4	0·51	1·85E + 10	83·62	0·66	1·04E + 10	51·09	0·56	**9·05E + 09**	45·26
8	0·55	1·21E + 10	57·66	0·71	**9·07E + 09**	45·11	0·61	9·26E + 09	46·22
16	0·65	8·95E + 09	44·63	0·83	7·51E + 09	38·49	0·71	**7·00E + 09**	36·10

Table III. The test problems

Problem identifier	Origin	Description	Number of variables	Number of elements
CEGB3024	HB	2D cross-section of a reactor core	2996	551
CEGB3306	HB	Framework problem from structural engineering	3222	791
LOCK2232	HB	Framework model of a launch umbilical tower	2208	944
LOCK3491	HB	2D vehicle model	3416	684
AEAC5081	AEAT	Double glazing problem	5081	800
RFLOW1	AEAT	Flow problem	9731	1715
OPT1	DNVR	Part of a condeep cylinder	15449	977
TRDHEIM	DNVR	Trondheim fjord model	22098	813
TSYL201	DNVR	Part of a condeep cylinder	20685	960
JETN	RALPAR	3D pipe model	548	360
CHAM	RALPAR	Part of an engine cylinder	12834	11070
TUBU	RALPAR	Engine cylinder model	26573	23446

other problems, a list of the unknowns for each element in the finite-element domain was available. These lists did not include the constrained variables lying on boundaries with Dirichlet boundary conditions. Using these lists, the element orderings produced by the reordering algorithms may differ slightly from those which would be obtained if complete lists of the variables associated with each element in the finite-element domain were available.

Before we could test the element reordering algorithms it was necessary to partition the underlying finite-element domains into subdomains. This problem has itself been the subject of much research in recent years and a variety of methods have been discussed in the literature. These methods include the greedy algorithm of Farhat,[28] bandwidth minimization,[29] Keringhan and Lin methods,[30] the inertial bisection method (see, for example, Simon[31]), recursive graph partitioning,[32] spectral partitioning methods,[33,34] and multilevel methods.[35] For the RALPAR problems, a partitioning of the finite-element domain into subdomains was supplied. This partitioning was obtained using the recursive graph bisection algorithm with the Keringhan and Lin refinement implemented in the Rutherford Appleton Laboratory code RALPAR code.[36,37]

For the HB and DNVR problems and problem AEAC5081, the finite-element domain was partitioned using the Chaco package from Sandia National Laboratories.[38] Spectral bisection with the Keringhan and Lin refinement was used. For the RFLOW1 problem, a partitioning of the domain was supplied with the problem.

In all our tests on the problems described in Table III, we have chosen to partition the domain into at most 16 subdomains. This is consistent with the number of subdomains we anticipate when using the multiple front algorithm. Our experience has been that, as the number of subdomains increases (and consequently the number of interface variables increases), it becomes increasingly important to take into account the interface variables when reordering the elements.

In Tables IV(a)–IV(d) we present results for the different element ordering algorithms for the HB, AEAT, DNVR, and RALPAR test problems, respectively. For each problem, the maximum and r.m.s. frontsizes (*fmax* and *frms*) are given for each subdomain for the original user-supplied element ordering, the MC43 ordering, and the Approach I and Approach II orderings. The ordering which gives the smallest r.m.s. frontsize is in bold typeface. CPU timings for reordering the elements together with floating-point operation counts and CPU times for solving the test problems using the multiple front algorithm are given in Table V. The RALPAR examples are not included in Table V since lists of variables associated with the finite-element nodes were not available for these examples. For problems for which only the matrix sparsity pattern was available (the HB problems and problem AEAC5081), values for the matrix entries were generated using the Harwell Subroutine Library random number generator FA04.

From Tables IV(a)–IV(d) we see that, in general, Approach II provides the element ordering with the smallest r.m.s. frontsize. As a result, the Approach II element orderings are generally the most efficient when combined with the multiple front algorithm (Table V). The only test problem for which Approach II does not give the best results is LOCK3491; for this problem, Approach I gives slightly better results. In most examples, MC43 does provide a better ordering (in terms of both the maximum and r.m.s. frontsizes) than the original ordering, but this is not guaranteed. For example, for subdomains 1 and 2 of problem TRDHEIM, the MC43 ordering has larger maximum and r.m.s. frontsizes than the original ordering and when used with the multiple front algorithm, requires more floating-point operations than the original ordering. As expected, Approach I, which is essentially the MC43 algorithm with modifications to allow for interface variables, almost always produces element orderings which are an improvement on those generated using MC43 but tie-breaking means that an improvement is again not guaranteed.

The sizes of the reductions in the r.m.s. frontsizes and in the number of floating-point operations which are achieved using the reordering algorithms are obviously dependent on the original user-supplied element ordering. It is clear that for some of the test problems, the original ordering was reasonable and the reordering algorithms were only able to produce modest savings. From Table V we see that for problem AEAC5081, Approaches I and II gave savings in the floating-point operation counts of little more than 10 and 20 per cent, respectively. However, for many of the test problems the user was not able to provide a good initial ordering and for these problems the reordering algorithms gave substantial savings. For example, for problem OPT1, Approaches I and II reduced the floating-point operation count by about 68 and 75 per cent, respectively. We performed some additional experiments in which the initial element ordering was arbitrary (that is, the user-supplied ordering was randomly permuted). The savings achieved using the reordering algorithms were impressive. For an arbitrary element order, the total number of floating-point operations required by the multiple front algorithm for problem TRDHEIM was $2{\cdot}04*10^{10}$, but Algorithm II reduced this number by about 96 per cent to $8{\cdot}25*10^{8}$.

Table IV(a). A comparison of the original, MC43, Approach I, and Approach II element orderings for the HB test problems

Problem	Sub-domain	Number of elements	Number of interface variables	Original		MC43		Approach I		Approach II	
				fmax	*frms*	*fmax*	*frms*	*fmax*	*frms*	*fmax*	*frms*
CEGB3024	1	67	74	112	88·5	108	76·7	84	67·1	80	**60·4**
	2	66	82	92	**62·4**	104	77·2	92	**62·4**	92	**62·4**
	3	65	90	128	100·3	96	75·0	96	73·9	92	**72·7**
	4	71	42	98	75·1	88	66·4	68	43·0	62	**39·5**
	5	72	46	102	73·4	72	52·1	76	60·1	63	**48·0**
	6	74	66	98	77·7	80	58·7	80	56·7	78	**47·7**
	7	69	112	134	95·1	144	98·4	126	90·6	118	**86·4**
	8	67	72	106	86·5	90	69·3	86	70·0	80	**64·9**
CEGB3306	1	199	114	186	136·8	138	112·8	120	86·6	122	**58·1**
	2	197	120	258	189·7	126	72·8	126	66·4	126	**59·7**
	3	197	108	210	172·1	114	66·1	114	62·8	114	**58·5**
	4	198	102	222	170·5	126	87·5	108	62·6	109	**56·7**
LOCK2232	1	117	150	276	198·9	156	101·7	156	103·5	156	**93·5**
	2	118	120	294	216·9	144	108·8	138	105·1	126	**83·7**
	3	118	108	246	193·7	126	94·6	126	85·3	114	**76·4**
	4	117	90	234	167·0	138	100·0	108	69·7	96	**47·8**
	5	117	120	210	164·2	138	100·7	138	95·3	126	**66·3**
	6	117	78	300	211·0	126	93·8	120	90·1	84	**62·0**
	7	123	48	246	159·6	102	76·4	78	52·2	66	**44·1**
	8	117	162	300	216·4	186	124·6	168	110·6	168	**93·5**
LOCK3491	1	148	108	171	134·9	234	188·2	171	134·9	147	**119·3**
	2	179	161	318	219·6	276	201·0	167	**102·9**	167	124·7
	3	176	199	428	289·8	297	200·2	232	141·9	230	**138·5**
	4	181	210	294	223·8	210	118·5	186	**115·7**	176	130·1

Table IV(b). A comparison of the original, MC43, Approach I, and Approach II element orderings for the AEAT test problems

Problem	Sub-domain	Number of elements	Number of interface variables	Original		MC43		Approach I		Approach II	
				fmax	*frms*	*fmax*	*frms*	*fmax*	*frms*	*fmax*	*frms*
AEAC5081	1	48	139	149	113·8	149	113·8	149	113·8	142	**109·0**
	2	50	152	165	124·1	165	124·1	165	124·1	169	**117·4**
	3	53	76	79	62·8	79	58·9	79	56·4	79	**53·2**
	4	49	154	164	124·4	164	109·9	164	109·8	164	**106·0**
	5	49	154	157	134·3	164	109·1	164	109·8	164	**106·8**
	6	49	133	136	102·6	136	102·6	136	102·6	136	**85·4**
	7	50	110	120	**85·4**	120	**85·4**	120	**85·4**	120	**85·4**
	8	49	119	122	100·6	122	100·6	136	95·8	136	**94·0**
	9	49	119	129	108·0	129	108·0	129	108·0	136	**94·0**
	10	50	110	141	106·8	141	106·8	134	105·1	127	**88·7**
	11	50	119	129	109·6	129	109·6	129	100·1	129	**85·6**
	12	51	90	100	72·1	131	91·1	100	72·1	100	**71·7**
	13	51	90	129	111·0	121	89·0	121	80·4	100	**71·7**
	14	54	76	123	102·5	82	56·3	79	53·5	79	**49·5**
	15	50	138	141	**98·4**	141	**98·4**	141	**98·4**	141	**98·4**
	16	48	139	149	112·8	149	112·8	149	112·8	142	**107·0**
RFLOW1	1	740	668	698	438·9	678	474·3	698	438·9	672	**392·8**
	2	575	1149	1462	929·8	1456	869·4	1456	869·1	1468	**747·1**
	3	400	783	827	501·6	827	487·2	827	486·8	815	**412·4**

Table IV(c). A comparison of the original, MC43, Approach I, and Approach II element orderings for the DNVR test problems

Problem	Sub-domain	Number of elements	Number of interface variables	Original		MC43		Approach I		Approach II	
				fmax	*frms*	*fmax*	*frms*	*fmax*	*frms*	*fmax*	*frms*
OPT1	1	243	904	1638	1206·2	941	692·5	925	497·4	925	**479·1**
	2	236	533	1161	800·9	539	311·4	539	539	**304·2**	313·7
	3	244	590	1362	1104·1	967	693·4	876	692·1	602	**369·7**
	4	254	889	1479	1169·7	1016	801·1	979	**760·1**	952	788·0
TRDHEIM	1	101	180	216	**174·4**	282	206·2	216	**174·4**	216	**174·4**
	2	102	216	336	227·3	414	328·0	228	166·2	228	**153·6**
	3	101	174	306	231·5	324	225·5	210	**156·8**	294	165·8
	4	101	258	414	301·7	360	270·2	366	214·0	306	**191·5**
	5	102	192	300	251·1	300	251·1	294	238·7	240	**180·1**
	6	102	210	246	**203·8**	246	**203·8**	246	**203·8**	246	**203·8**
	7	102	66	150	109·7	150	109·7	150	106·7	162	**104·6**
	8	102	108	234	193·4	234	193·4	156	111·1	150	**107·8**
TSYL201	1	120	543	651	505·2	594	448·7	564	455·0	564	**441·5**
	2	120	582	633	477·1	708	512·3	603	**409·0**	603	**409·0**
	3	120	732	915	688·4	753	533·3	753	523·7	753	**515·2**
	4	120	582	633	477·1	708	512·3	603	**409·0**	603	**409·0**
	5	120	576	759	558·7	597	472·6	597	477·1	597	**441·2**
	6	120	582	708	644·7	633	548·9	603	**409·0**	603	**409·0**
	7	120	732	768	582·3	753	530·7	753	**512·5**	753	519·3
	8	120	582	633	477·1	708	512·3	603	**409·0**	603	**409·0**

Table IV(d). A comparison of the original, MC43, Approach I, and Approach II element orderings for the RALPAR test problems

Problem	Sub-domain	Number of elements	Number of interface variables	Original		MC43		Approach I		Approach II	
				fmax	*frms*	*fmax*	*frms*	*fmax*	*frms*	*fmax*	*frms*
JETN	1	90	61	93	62·2	65	47·6	64	46·3	65	**44·9**
	2	90	61	81	58·9	63	45·7	63	44·8	62	**43·2**
	3	90	77	114	81·6	95	75·7	77	**46·8**	80	49·4
	4	90	57	96	69·8	84	59·5	64	47·1	57	**39·7**
CHAM	1	1383	342	449	372·0	377	278·4	397	287·5	343	**259·8**
	2	1383	403	640	486·7	419	301·2	418	303·8	410	**275·0**
	3	1384	395	484	406·1	454	319·1	462	318·1	446	**309·8**
	4	1384	489	571	441·0	575	391·6	546	372·3	549	**368·7**
	5	1384	505	565	450·8	585	398·8	547	370·0	512	**357·0**
	6	1384	373	490	404·7	420	294·2	385	288·7	374	**257·4**
	7	1384	529	610	503·4	581	386·8	572	393·7	530	**369·1**
	8	1384	513	595	461·0	548	378·3	577	384·4	514	**364·1**
TUBU	1	5861	296	707	534·7	540	367·1	540	362·2	536	**353·7**
	2	5861	430	657	477·0	759	551·1	471	**317·8**	556	338·8
	3	5862	691	1740	1306·6	1017	789·3	743	599·7	725	**541·9**
	4	5862	736	1179	978·6	916	734·0	739	451·1	737	**386·0**

Table V. CPU times (seconds) for reordering the elements together with floating-point operation counts (flops) and CPU times (seconds) for solving the test problems using the multiple front algorithm

	Original		MC43			Approach I			Approach II		
Problem	Total flops	Solve CPU time	Reorder CPU time	Total flops	Solve CPU time	Reorder CPU time	Total flops	Solve CPU time	Reorder CPU time	Total flops	Solve CPU time
CEGB3024	3·70E + 07	0·82	0·07	2·75E + 07	0·76	0·08	2·34E + 07	0·71	0·08	**2·04E + 07**	0·69
CEGB3306	1·82E + 08	1·69	0·07	4·97E + 07	0·90	0·08	3·39E + 07	0·81	0·08	**2·39E + 07**	0·74
LOCK2232	1·44E + 08	1·46	0·10	4·65E + 07	0·90	0·11	3·75E + 07	0·85	0·12	**2·48E + 07**	0·75
LOCK3491	3·62E + 08	3·31	0·09	2·13E + 08	2·21	0·10	**1·09E + 08**	1·67	0·10	1·14E + 08	1·70
AEAC5081	1·44E + 08	2·41	0·14	1·34E + 08	2·32	0·18	1·30E + 08	2·31	0·17	**1·12E + 08**	2·20
RFLOW1	8·98E + 09	54·01	0·14	8·08E + 09	51·30	0·18	7·98E + 09	50·42	0·21	**6·85E + 09**	44·21
OPT1	3·46E + 10	148·70	0·29	1·39E + 10	66·27	0·36	1·12E + 10	57·81	0·39	**8·61E + 09**	47·94
TRDHEIM	1·24E + 09	12·56	0·22	1·40E + 09	13·23	0·29	8·17E + 08	10·49	0·28	**6·95E + 08**	9·85
TSYL201	1·40E + 10	69·54	0·27	1·12E + 10	62·12	0·34	9·79E + 09	53·52	0·34	**9·57E + 09**	52·19

Table VI. The sensitivity of Approach II to the priority function weights W_1, W_2, W_3

Weights				TUBU		RFLOW1	
W_1	W_2	W_3	Subdomain	*fmax*	*frms*	*fmax*	*frms*
12	6	1	1	536	**353·8**	672	**392·4**
			2	556	338·8	1468	**747·1**
			3	725	**542·0**	815	**412·3**
			4	737	**386·1**		
10	5	1	1	544	354·5	672	**392·4**
			2	558	348·0	1468	**747·1**
			3	721	542·2	815	**412·3**
			4	744	390·5		
12	6	2	1	538	359·1	672	392·6
			2	449	**318·5**	1468	747·6
			3	749	552·6	815	**412·3**
			4	807	401·7		
12	6	0	1	593	380·0	672	391·8
			2	539	353·8	1462	748·8
			3	740	544·7	815	501·6
			4	737	387·7		
2	1	2	1	592	447·2	672	400·1
			2	657	477·0	1468	764·0
			3	942	724·1	818	414·5
			4	792	431·7		
1	0	0	1	707	534·7	672	**392·4**
			2	476	346·1	1468	750·0
			3	822	588·5	815	**412·3**
			4	825	407·2		
0	1	0	1	707	534·7	698	438·9
			2	657	477·0	1462	930·0
			3	1740	1306·6	827	501·6
			4	1179	978·7		
1	1	1	1	620	438·5	672	399·2
			2	657	477·0	1468	763·1
			3	885	640·9	818	413·6
			4	800	427·8		

We remark that we found the Chaco code was well able to partition the finite-element domains into subdomains having an (approximately) equal number of elements. The Chaco code attempts to minimise the total number of interface variables and this will reduce both the amount of data which must be transferred between processors when the multiple front algorithm is run in parallel and the amount of work needed to solve the interface problem. However, Chaco can produce partitions in which the subdomains all have a very different number of interface variables and this can lead to a wide variation in the subdomain frontsizes and hence to poor load balancing when the multiple front algorithm is implemented in a parallel environment. Consider the test problem TRDHEIM. We partitioned the domain into 8 subdomains using Chaco and found that each subdomain had 101 or 102 elements but the number of interface variables ranged from 66 for subdomain 7 to 258 for subdomain 4 (Table IV(c)). After reordering the elements with Approach II, the number of floating-point operations required by the frontal solver for these two subdomains were $3{\cdot}33*10^7$ and $1{\cdot}12*10^8$, respectively. If all 8 processors on the CRAY Y-MP8I are used to solve the problem (one for each subdomain), processor 7 completes its work in 0·93 s while processor 4 takes 1·30 s. Domain partitioning algorithms which attempt to balance the number of interface variables between subdomains are required for the multiple front algorithm and we plan to investigate this in the future.

In Table VI we illustrate the sensitivity of Approach II to the choice of the weights used in the priority function (8). Results are given for test problems TUBU (4 subdomains) and RFLOW1 (3 subdomains). For each subdomain, the smallest r.m.s. frontsizes are in bold typeface. These two problems were selected since they illustrate that for some subdomains the element ordering can be sensitive to the choice of weights, while other subdomains are much less sensitive. We see that the maximum and r.m.s. frontsizes for subdomain 1 of RFLOW1 are relatively insensitive to the changes in the weights we considered. However, for subdomain 3 we see that setting W_3 equal to 0 leads to much larger frontsizes and this illustrates the importance of including the third weight in the priority function. For problem TUBU, small changes to the recommended values of 12, 6, and 1 for W_1, W_2, and W_3, respectively, have some effect on the frontsizes but the effects are small. However, for this problem, making substantial changes to the ratios between the weights (such as giving the weights equal values) results in significant increases in the frontsizes in some of the subdomains.

7. CONCLUSIONS

In this study we have proposed two algorithms for reordering elements for use with a multiple front algorithm. This problem is more complicated than the problem of resequencing elements for a frontal solver on a single domain since it is necessary to distinguish between variables which can be eliminated once they are fully summed and interface variables that cannot be eliminated within the subdomain. The two approaches we have considered involve two different ways of locating a suitable starting element s for the reordering procedure. Once a starting element has been selected, both methods use a modification of the method of Sloan[10] to reorder the remaining elements. Our first method (Approach I) for choosing s is based on finding pseudo-peripheral nodes of the element communication graph. The second method (Approach II), introduces an artificial element, the guard element, and uses this extra element to find an element lying as far from the interface boundary as possible and uses this to start the reordering. We have tested both approaches on a range of problems and compared their performance with that of the HSL code MC43, which is designed for single domain problems. Both approaches give significant improvements over MC43 and Approach II was almost always the method of choice. On the basis of our findings, a code MC53 implementing Approach II has been developed and is included in the Harwell Subroutine Library.[19] Since we saw from Table VI that the quality of the element ordering can be dependent on the weights used in the priority function, these weights are passed to MC53 as control parameters. They are given the default values of 12, 6, and 1 (which are the values we recommend on the basis of our numerical experiments) but the user may choose to reset one or more of these values.

7.1. Availability of the code

The codes MA42, MA52, MC43, and MC53 are all written in standard FORTRAN 77 and are included in the current release of the Harwell Subroutine Library.[19] Anyone interested in using

HSL routines should contact the HSL Manager: Dr. S. J. Roberts, Harwell Subroutine Library, AEA Technology, Building 552, Harwell, Oxfordshire, OX11 ORA, England, tel. +44 (0) 1235 434714, fax +44 (0) 1235 434136, or e-mail Scott.Roberts@aeat.co.uk, who will provide details of price and conditions of use.

ACKNOWLEDGMENTS

I would like to thank Andrew Cliffe, Christian Damhaug, and Ron Fowler for supplying me with some of the test problems used in this study. I am also grateful to Iain Duff of the Rutherford Appleton Laboratory for his interest and to Alex Pothen of Old Dominion University for some interesting email discussions.

REFERENCES

1. B. M. Irons, 'A frontal solution program for finite-element analysis', *Int. j. numer. methods eng.*, **2**, 5–32 (1970).
2. P. Hood, 'Frontal solution program for unsymmetric matrices', *Int. j. numer. methods eng.*, **10**, 379–400 (1976).
3. I. S. Duff, 'Enhancements to the MA32 package for solving sparse unsymmetric matrices', *Harwell Report AERE R11009*, HMSO, London, 1983.
4. J. E. Akin and R. M. Pardue, 'Element resequencing for frontal solutions', in J. R. Whiteman (ed.), *Mathematics of Finite Elements and Applications*, Academic Press, New York, 1975.
5. A. Bykat, 'A note on an element ordering scheme', *Int. j. numer. methods eng.*, **11**, 194–198 (1977).
6. A. Razzaque, 'Automatic reduction of frontwidth for finite element analysis', *Int. j. numer. methods eng.*, **15**, 1315–1324 (1980).
7. H. L. Pina, 'An algorithm for frontwidth reduction', *Int. j. numer. methods eng.*, **17**, 1539–1546 (1981).
8. S. W. Sloan and M. F. Randolph, 'Automatic element reordering for finite-element analysis with frontal schemes', *Int. j. numer. methods eng.*, **19**, 1153–1181 (1983).
9. S. J. Fenves and K. H. Law, 'A two-step approach to finite element ordering', *Int. j. numer. methods eng.*, **19**, 891–911 (1983).
10. S. W. Sloan, 'An algorithm for profile and frontsize reduction of sparse matrices', *Int. j. numer. methods eng.*, **23**, 239–251 (1986).
11. M. S. Shephard, P. L. Baehmann and K. R. Griece, 'The versatility of automatic mesh generators based on tree structures and advanced geometric constructs', *Commun. appl. numer. methods*, **4**, 379–392 (1988).
12. I. S. Duff, J. K. Reid and J. A. Scott, 'The use of profile algorithms with a frontal code', *Int. j. numer. methods eng.*, **28**, 2555–2568 (1989).
13. A. Kaveh, 'A connectivity coordinate system for node and element ordering', *Comput. Struct.*, **41**, 1217–1223 (1991).
14. B. U. Koo and B. C. Lee, 'An efficient profile reduction algorithm based on the frontal ordering scheme and graph theory', *Comput. Struct.*, **44**, 1339–1347 (1992).
15. S. R. P. Medeiros, P. M. Pimenta and P. Goldenberg, 'An algorithm for profile and frontsize reduction of sparse matrices with a symmetric structure', *Eng. Comput.*, **10**, 257–266 (1993).
16. G. H. Paulino, I. F. M. Menezes, M. Gattass and S. Mukherjee, 'Node and element resequencing using the Laplacian of a finite element graph: part I—general concepts and algorithm', *Int. j. numer. methods eng.*, **37**, 1511–1530 (1994).
17. I. S. Duff and J. A. Scott, in J. Lewis (ed.), *Proc. 5th SIAM Conf. on Applied Linear Algebra*, SIAM Press, Philadelphia, PA, 1994, pp. 567–571.
18. I. S. Duff and J. A. Scott, 'The use of multiple fronts in Gaussian elimination', *Rutherford Appleton Laboratory Report RAL-94-040*, 1994.
19. Harwell Subroutine Library, 'A catalogue of subroutines (Release 12)', Advanced Computing Department, Harwell Laboratory, Oxfordshire, England.
20. I. S. Duff and J. A. Scott, 'MA42—a new frontal code for solving sparse unsymmetric systems', *Rutherford Appleton Laboratory Report RAL-93-064*, 1993.
21. I. S. Duff and J. A. Scott, 'The design of a new frontal code for solving sparse unsymmetric systems', *ACM Trans. Math. Softw.*, **22**, 30–45 (1996).
22. N. E. Gibbs, 'A hybrid profile reduction algorithm', *ACM Trans. Math. Softw.*, **2**, 378–387 (1976).
23. N. E. Gibbs, W. G. Poole Jr. and P. K. Stockmeyer, 'An algorithm for reducing the bandwidth and profile of a sparse matrix', *SIAM J. Numer. Anal.*, **13**, 236–250 (1976).
24. A. George and W. H. Liu, 'An implementation of a pseudoperipheral node finder', *ACM Trans. Math. Softw.*, **5**, 284–295 (1979).
25. I. S. Duff, R. G. Grimes and J. G. Lewis, 'Sparse matrix test problems', *ACM Trans. Math. Softw.*, **15**, 1–14 (1989).
26. I. S. Duff, R. G. Grimes and J. G. Lewis, 'User's Guide for the Harwell–Boeing Sparse Matrix Collection', *Rutherford Appleton Laboratory Report RAL-92-086*, 1992.

27. R. E. Benner, G. R. Montry and G. G. Weigand, 'Concurrent multifrontal methods: shared memory, cache, and frontwidth issues', *Int. J. Supercomput. Appl.*, **1**, 26–44 (1987).
28. C. Farhat, 'A simple and efficient automatic FEM domain decomposer', *Comput. Struct.*, **28**, 579–602 (1998).
29. J. G. Malone, 'Automatic mesh decomposition and concurrent finite element analysis for hypercube multiprocessor computer', *Comput. Methods Appl. Mech. Eng.*, **70**, 20–58 (1998).
30. B. Keringhan and S. Lin, 'An efficient heuristic procedure for partitioning graphs', *Bell System Tech. J.*, **29**, 291–307 (1970).
31. H. D. Simon, 'Partitioning of unstructured problems for parallel processing', in *Proc. Conf. on Parallel Methods on Large Scale Structural and Physics Applications*, Pergamon Press, Oxford, 1991.
32. R. D. Williams, 'Performance of dynamics load balancing algorithms for unstructured mesh calculations', *Concurrency, Practice, Experience*, **3**, 457–482 (1991).
33. A. Pothen, H. D. Simon and K. P. Liou, 'Partitioning sparse matrices with eigenvectors of graphs', *SIAM J. Matrix Anal. Appl.*, **11**, 430–452 (1990).
34. B. Hendrickson and R. Leland, 'An improved spectral graph partitioning algorithm for mapping parallel computations', *Technical Report SAND92-1460*, Sandia National Laboratories, 1992.
35. B. Hendrickson and R. Leland, 'A multilevel algorithm for partitioning graphs', *Technical Report SAND93-1301*, Sandia National Laboratories, 1993.
36. C. Greenough and R. F. Fowler, 'Partitioning methods for unstructured finite element meshes', *Rutherford Appleton Laboratory Report RAL-94-091*, 1994.
37. C. Greenough and R. F. Fowler, 'RALPAR—The RAL mesh partitioning program', *Rutherford Appleton Laboratory Report RAL-94-092*, 1994.
38. B. Hendrickson and R. Leland, 'The Chaco user's guide, version 1.0', *Technical Report SAND92-2339*, Sandia National Laboratories, 1993.
39. J. J. Dongarra, J. J. Du Croz, I. S. Duff and S. Hammarling, 'A set of Level 3 Basic Linear Algebra Subprograms', *ACM Trans. Math. Software*, **16**, 1–17 (1990).
40. I. S. Duff and J. K. Reid, 'The multifrontal solution of indefinite sparse symmetric linear systems', *ACM Trans. Math. Software*, **9**, 302–325 (1983).
41. W. P. Zhang and E. M. Liu, 'A parallel frontal solver on the Alliant FX/80', *Comput. Struct.*, **38**, 203–215 (1991).
42. A. Kaveh, 'A note on a two-step approach to element ordering', *Int. j. numer. methods eng.*, **20**, 1553–1554 (1984).