6

The multifrontal method in a parallel environment

I. S. Suff, N. I. M. Gould, M. Lescrenier, and J. K. Reid

Abstract

The multifrontal solution of symmetric and positive-definite finite element problems may be represented by an assembly tree. The leaves correspond to the elements themselves and each interior node corresponds to the substructure consisting of all the elements corresponding to its descendants. The overall matrix may be assembled and factorized by performing merge and elimination operations at each node. These operations may be performed in any order, subject to the restriction that each node must wait for the completion of all operations at its children. This allows parallel execution of the operations at any two nodes that are not such that either is a descendant of the other.

The aim of this work is to investigate ordering algorithms that produce trees that work well in a parallel environment. We show the performance of several algorithms on a variety of finite-element problems using a model of parallel execution that takes account of the available parallelism both between the nodes and at them.

6.1. Introduction

We consider the direct solution of large sets of linear equations

$$Ax = b \tag{6.1}$$

on a computer that allows different processes to execute in parallel. We assume that A is symmetric and positive definite and has the form

$$A = \sum_k B^{(k)}, \tag{6.2}$$

where each $B^{(k)}$ is an element matrix that is positive definite and has entries only in the rows and columns that correspond to its associated variables. Such problems occur naturally in finite-element and partially separable programming calculations (see Griewank and Toint 1982).

We use Gaussian elimination, following a multifrontal scheme (see, for example, Duff and Reid 1983, or Duff *et al.* 1986, pp. 218–21). Here the elimination is represented by an assembly tree in which each leaf corresponds to a finite element and each interior node corresponds to the substructure consisting of all the elements corresponding to its descendants. Each interior node has a generated element matrix that is obtained by adding (assembling) the element matrices of its children and performing elimination operations for the rows and columns corresponding to variables that are involved only in elements that correspond to its descendants. In the case of a leaf node, an elimination is performed for each variable that is involved only in the corresponding element; this is often called 'static condensation'.

Operations at an interior node must wait for the availability of the generated element matrices of all the children of the node. Apart from this, the operations may be performed in any order. Here we focus on the possibility of performing them in parallel. We explore what speed-up is available and consider heuristic algorithms that aim to enhance it.

It should be noted that the multifrontal technique is also applicable to the case of assembled matrices and similar considerations of parallelism also apply in that case. Duff (1986) and Liu (1986) have both investigated the parallelism inherent in the elimination trees associated with multifrontal methods on assembled matrices and Duff (1986) has proposed an algorithm for implementing such an approach on shared-memory multiprocessors. The method has been implemented on an eight-processor Alliant FX/8 and displays a speed-up of about 6 (see Duff 1989).

We are not here concerned with the fine details of implementation. Our aim is to consider the potential for parallel execution by simulating execution on a multiprocessor. The structure of the assembly tree is clearly crucial to its potential for parallelism. We discuss possible choices in Sections 6.2–6.5. We consider the use of a standard minimum-degree algorithm in Section 6.2 and the nested dissection algorithm in Section 6.3. In Section 6.4, we propose new algorithms based on a notion of 'similarity' and, in Section 6.5, consider ways of modifying the minimum-degree algorithm to produce greater potential parallelism. We discuss the assumptions made in our simulation of parallel execution in Section 6.6 and define some of the measures of evaluation for work and communication in Section 6.7. We present the numerical results in Section 6.8 and draw some conclusions from these results in Section 6.9.

6.2. The minimum-degree algorithm

The algorithm of minimum degree (Scheme 2 of Tinney and Walker 1967) has a good reputation as a general-purpose algorithm (see George and Liu 1987, for example). In this context, it is implemented by starting with all the leaf nodes (original elements) and finding a variable of minimum degree. The

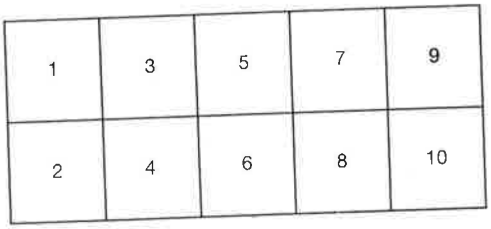

Fig. 6.1 A rectangular region with bilinear elements.

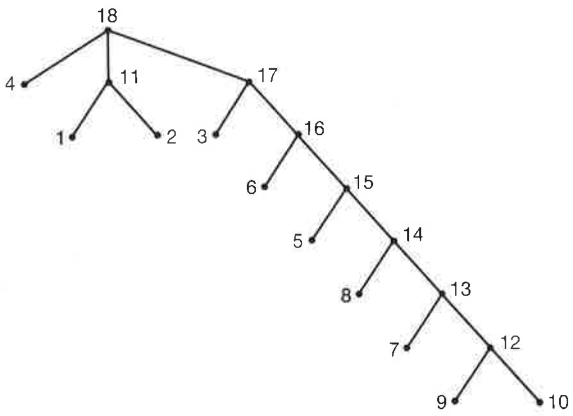

Fig. 6.2 The minimum-degree assembly tree for the problem of Fig. 6.1.

elements in which this variable is involved are grouped as children of a new node. The element at the new node is generated by assembling the elements in the group and eliminating variables not involved in other elements. The degree of the variable is the number of variables in the assembled element before eliminations commence. This is also termed the front size. The general step is similar, working with the remaining variables and the remaining elements and generated elements.

One disadvantage of the algorithm is that it can produce very tall and thin trees. A good illustration of this is the use of bilinear elements on a 2 by 5 grid. Figure 6.1 describes the grid and Fig. 6.2 displays the corresponding minimum-degree assembly tree. Such trees give little scope for exploiting parallelism between nodes.

6.3. The nested dissection algorithm

An alternative is to use trees produced by the nested dissection algorithm, introduced by George (1973) for regular grids. We consider here the generali-

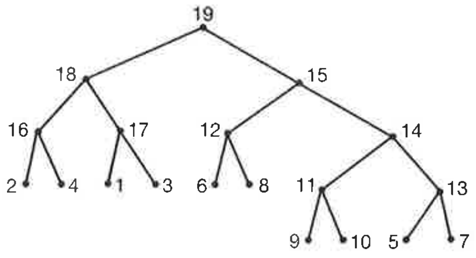

Fig. 6.3 The nested-dissection assembly tree for the problem of Fig. 6.1.

zation of George and Liu (1978). This uses the algorithm of Gibbs *et al.* (1976) to find a set of level sets in the graph of the matrix. If the variables in the middle level set and their edges are removed from the graph, the graph will be split into two approximately equal halves. The variables in the middle level set are assigned last in the pivotal order, and the subgraphs associated with the remaining variables are now analysed separately in the same way and their middle level sets assigned next to last in the order. The algorithm continues until the ordering is complete.

Once the ordering has been constructed, the associated elimination tree may be generated as for the minimum-degree algorithm. Because of the close relationship between the graph of the matrix and the sets of elements associated with the descendants of a tree node, this will be a well-balanced tree with successive levels of the tree having usually 1, 2, 4, 8, ... nodes until the sequence is broken by reaching leaf nodes. The tree for the Fig. 6.1 example is shown in Fig. 6.3.

6.4. An elimination tree based on the similarity of elements

With the aim of introducing more parallelism between nodes of elimination trees, we have experimented with some heuristic algorithms that construct groups of elements, each group involving roughly the same number of variables. Each group will correspond to a frontal matrix and the matrix order will be small if the elements are similar. We use the concept of 'similarity', introduced for partially separable optimization problems by Strodiot and Toint (1976). The similarity between an element and a group of elements is the ratio of the number of common variables (variables that lie both in the given element and in at least one element of the group) to the total number of distinct variables involved. It increases with the number of common variables and decreases with the total number of variables.

We choose the starting element of a group arbitrarily and add elements one by one. Each addition is chosen as the element that is most similar to the current group. To end a group, several stopping criteria have been tested. The first idea was to end when the best similarity between the current group and a remaining element fell below a threshold. We rejected this strategy since it led to groups with very different numbers of elements. After examining several other strategies, we finally chose to limit the maximum size of the group to a user-provided number of elements. Once a group has been chosen, the next group is chosen similarly from the remaining elements, continuing until all elements are grouped.

As in the minimum-degree algorithm, we generate an element matrix for each group by assembling the elements of the group and eliminating variables not involved elsewhere. When all the elements are allocated to groups and the associated assemblies and eliminations have been performed, we are left with a new finite-element problem to which the same algorithm may be applied. We continue until we end with only one group, which is the root of the elimination tree.

On finite-element problems, the proposed algorithm assembles adjacent elements and the variables internal to the assembled elements are eliminated. The remaining nodes then define the generated element. However, we cannot be sure that there are any internal variables (and thus eliminations) and numerical experiments showed that this problem occurs in practical situations. In order to avoid this drawback, we slightly modified the notion of similarity. First of all, we defined the elemental degree of a variable as the number of elements or generated elements which involve it. The similarity between an element and a group of elements may then be defined as the ratio of the number of common variables of minimum elemental degree (amongst the variables of the group) to the total number of variables. We decided also to end a group as soon as at least one elimination is performed in it.

Unfortunately, our experimental results (Section 6.8) showed that, although this algorithm increased the amount of parallelism, it also increased the fill-in and the number of operations by an unacceptable amount.

The similarity tree for the Fig. 6.1 example is shown in Fig. 6.4. Here we end a group as soon as it contains 2 elements.

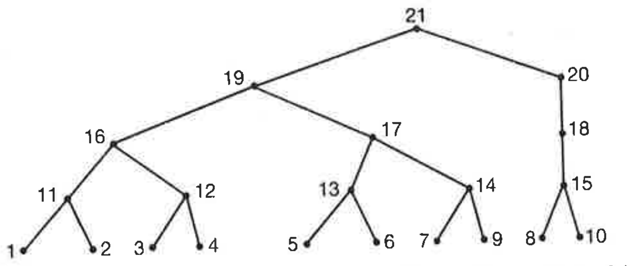

Fig. 6.4 The similarity assembly tree for the problem of Fig. 6.1.

6.5. Variants of the minimum-degree algorithm

We therefore returned to the minimum-degree algorithm with the intent of removing its main disadvantage, the production of tall and thin trees. We have introduced a variant which aims to retain the sparsity benefits of minimum-degree while generating an assembly tree with better potential for parallelism. The algorithm works in distinct stages corresponding to levels of the assembly tree. Within a stage, we first choose a variable of minimum degree and assemble the elements in which it lies. Each successive variable is chosen to minimize the degree among candidates not associated with elements already assembled in the stage. We could continue the stage only as long as the chosen variable minimizes the degree of all variables, which would yield a minimum-degree algorithm. However, this often results in very few nodes being generated in a stage, contrary to our aim of having plenty for the sake of parallelism. We have therefore chosen to continue until a threshold for the degree is reached, choosing the threshold to be a fixed factor times the greater of the threshold of the previous stage and the minimum degree at the start of the stage. A larger factor (for example, 1.5) can give an undesirably large increase in the total number of operations, but a small factor (for example, 1.1) can

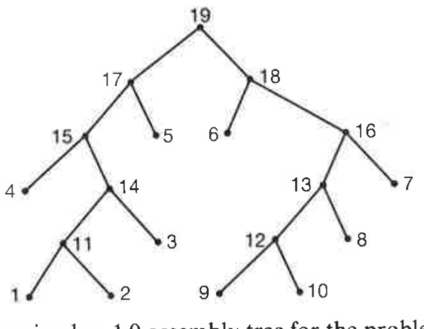

Fig. 6.5 The min. deg. 1.0 assembly tree for the problem of Fig 6.1.

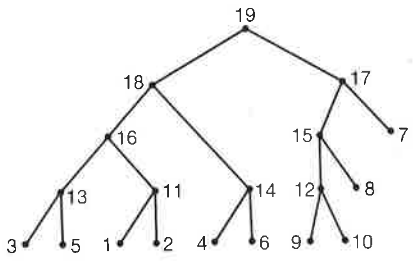

Fig. 6.6 The min. deg. 1.5 assembly tree for the problem of Fig. 6.1.

benefit parallelism without much increase in the total number of operations (see Section 6.8).

The trees for the Fig. 6.1 example when this algorithm is applied with two values for the factor are shown in Figs. 6.5 and 6.6. Notice that Fig. 6.5 has 6 tree levels, whereas Fig. 6.2 has 8 levels. This tree is obtained with factors 1.0, 1.1, and 1.2, and the associated Gaussian elimination involves 172 operations (divides and multiply–add pairs). With the factor 1.5, the number of tree levels is reduced to 5, but the number of operations increases to 201.

6.6. The parallel environment

Large problems give rise to nodes with large element matrices and it will be worthwhile to exploit parallelism at the nodes as well as between the nodes. Most authors (see, for example, George *et al.* 1986) suggest assigning blocks of rows (or columns) to processors and treating each row operation as indivisible. This also has the merit of allowing good vectorization on vector hardware. We have chosen to aim for all blocks to have roughly the same number of elements, this number being set externally by a parameter s that can be tuned.

Each element matrix is symmetric and we assume that only the upper-triangular part is stored. Our algorithm for dividing it into block rows is to make each block except the last have the property that deleting the last row or adding another row beyond it would make the number of entries in the block further from s. Note that if the frontal matrix has s or fewer entries, we have a single block and treat all the assembly and elimination operations at the node as a single task.

When the element matrix has $b > 1$ blocks, we treat any set of assembly or elimination operations on a block of rows as a task. Assembly operations are not needed at leaf nodes. At a node that is not a leaf, the matrix is assembled as b separate tasks. Assuming that each element matrix is held by rows in pivotal order, each task need only access those parts of the matrices of the children that it needs.

Elimination operations at a node must wait for the assembly to be complete. The elimination operations for the first pivot are then performed on the first block. Next b tasks are spawned, to perform operations for the second pivot on the first block and the operations for the first pivot on the rest. Once these are all complete, we spawn another b tasks for the third pivot on the first block and the second pivot on the other blocks. This continues until all the eliminations have been completed. Also, if the number of eliminations exceeds the number of rows in the first block, the early blocks will not be involved in the later pivotal steps so the number of tasks spawned will be reduced.

Once the elimination operations at a node have been completed, a check is made to see if its parent now has element matrices for all its children. If so, the assembly task (or tasks) for the parent node is spawned.

The tasks within the nodes all compete for processors on the basis of a queue of work. When a processor is available, the first task ready for execution is commenced. When a task is spawned, it joins the queue at the end.

We are concerned only with the potential for parallel execution and not with the fine details of the implementation. We assume that we have a multiprocessor on which the cost of an elementary step of Gaussian elimination (a division or a multiply–add pair) is γ, the cost of the assembly of an entry of a child element matrix into its parent element matrix is α, and the cost of creating or spawning a multiprocessor task is σ.

This model is directly applicable to a shared-memory computer, but it also gives a good approximation for the performance on a distributed-memory computer if α and σ are chosen appropriately. For the latter, many assemblies require data from another processor, so α should include the cost of data transfer. On the other hand, most eliminations require only the pivot row to be transferred and this is applied to several non-pivot rows, so we do not need to modify γ. Many of the tasks require data from one other processor, so the value of σ should allow for the start-up cost of a data transfer from another processor.

6.7. Operation counts

On a serial processor, the number of elimination operations (divides and multiply–add pairs) at a node whose front size is f and whose order following the eliminations at the node is g is

$$f(f-1)(f+4)/6 - g(g-1)(g+4)/6, \tag{6.3}$$

and a measure of the total work for elimination can be obtained by summing these quantities over all the nodes. We do this in all our computational experiments.

In the assembly operation that follows these eliminations, all the entries of the reduced matrix are summed into the matrix of the parent node, so the number of assembly operations is

$$g(g+1)/2, \tag{6.4}$$

and we sum these over all the nodes to measure the total assembly work.

For a parallel implementation, we assume that a fixed number of processors p is available, take a specified value for the parameter s (the number of entries that we aim to have in each block row) and values for the parameters γ, α, and

σ (the costs of an elimination operation, an assembly operation, and spawning a task, respectively) and then model the whole assembly and elimination process, reporting the total cost for comparison with the corresponding serial figure.

6.8. Experimental results

We have conducted experiments on nine test problems, all supplied to John Reid for experiments in connection with his TREESOLV package (Reid 1984) and now included in the set of sparse problems collected by Duff *et al.* (1989). Their properties are summarized in Table 6.1. The first four were supplied by A. J. Donovan of C.E.G.B., London, the next four by P. S. Jensen of Lockheed Palo Alto Research Laboratory, and the last by T. A. Manteuffel of Sandia Laboratories.

We chose values for the parameters α, γ, and σ based on our knowledge of commonly used machines, and then conducted experiments on the sensitivity of our model to these parameters. On many machines (for example, IBM 3084 and CRAY 2), accessing entries from store costs about the same as an addition which is in turn similar to a multiplication. Thus, for our initial experiments we choose $\alpha = 1$ and $\gamma = 2$. Our choice of σ is based on the overhead of a subroutine call which we estimate to be about 100 times that of our basic operation.

Table 6.1 Details of test problems.

No. of variables	No. of elements	Max. element size	Description
2919	128	60	Three-dimensional cylinder with flange.
3024	551	16	Two-dimensional reactor core section.
3306	791	12	Framework problem, essentially in two dimensions.
2802	108	60	Turbine blade.
1074	323	24	A toros with protusions, representing a gyro.
700	324	18	A porch structure.
2232	944	12	A tower with short legs.
3491	684	24	A cylindrical cone attached at its base to crossed beams.
5976	784	20	A cylindrical shell with irregularities, reduced by grouping three variables at each point into one.

We compare the following algorithms:

(1) minimum degree;
(2) nested dissection, using the implementation in SPARSPAK (Chu *et al.* 1984);
(3) the two algorithms based on element similarity described in Section 6.4; and
(4) the variant of minimum degree described in Section 6.5, with thresholds 1.0, 1.1, 1.2, and 1.5.

With $\alpha = 1$ and $\gamma = 2$, the costs without multiprocessing are shown in Table 6.2. The best algorithm (from the cost point of view) on each problem is labelled with * and those within 10 per cent of the best are labelled with #. Clearly, as one would expect, the minimum-degree algorithm preserves the sparsity well and is best or near to best on all problems.

Since the largest front size is over 600, we take s to be 650 or more so that we can run all the problems. The figure of 650 for s fits well with our estimate for σ of 100, because it means that the overheads of task initiation do not dominate.

The total parallel costs with four processors available, and $s = 650$, $\alpha = 1$, $\gamma = 2$, and $\sigma = 100$ are shown in Table 6.3. The minimum-degree algorithm is always the best or within 10 per cent of the best and there seems to be little justification for using another algorithm. The speed-ups (ratios of figures in Tables 6.2 and 6.3) are shown in Table 6.4, and it can be seen that the processors are almost always being used efficiently. We note that a number of shared-memory computers, including the CRAY X-MP48 and CRAY 2, have four processors.

We take 32 processors as an indicative higher-end number for a shared-memory computer, and the figures shown in Tables 6.5 and 6.6 correspond to those in Tables 6.3 and 6.4, using the same values of the parameters other than the number of processors p. It is now not so easy to choose a 'best buy'. The minimum-degree algorithm suffers sometimes from poor parallelization and its variants are needed to recover this. However, either it or a variant is always within 10 per cent of the best. The nested dissection algorithm is best on two problems and within 10 per cent of the best on another, which represents a performance on these problems that is comparable with any single one of the minimum degree variants. The similarity algorithms are never best or within 10 per cent of the best. Notice, however, that those algorithms obtained the best speed-ups (for which they were designed), leading then to a better use of parallel processors. Unfortunately, their costs in sequential mode are so high that even with such good speed-ups, they do not compete with the variants of the minimum degree algorithm.

Table 6.2 Costs in serial mode in millions of units. *best # within 10%

No. of variables	2919	3024	3306	2802	1074	700	2232	3491	5976
No. of elements	128	551	791	108	323	324	944	684	784
Max. element size	60	16	12	60	24	18	12	24	20
Min. degree	63.79 #	6.47*	2.32*	28.00*	5.17 #	1.57*	2.99*	20.11*	78.53*
Min. deg. 1.0	62.69*	7.46	2.42 #	29.18 #	5.38 #	1.71 #	4.01	20.19 #	94.71
Min. deg. 1.1	63.25 #	8.10	2.45 #	35.47	4.95*	1.80	4.18	24.72	95.26
Min. deg. 1.2	74.89	8.52	2.61	38.50	5.50	2.10	4.40	27.14	95.01
Min. deg. 1.5	143.51	12.93	2.76	47.02	5.32	2.89	4.35	33.31	148.79
Nested diss.	88.92	12.95	28.92	38.12	5.75	3.59	17.35	40.91	79.34 #
Similarity 1	112.98	15.44	15.43	63.21	7.82	6.25	50.76	59.44	136.57
Similarity 2	130.41	15.06	31.56	63.21	8.66	5.07	28.50	88.89	144.91

Table 6.3 Costs in parallel mode with four processors in millions of units ($s = 650, \alpha = 1, \gamma = 2, \sigma = 100$).
*best #within 10%

No. of variables	2919	3024	3306	2802	1074	700	2232	3491	5976
No. of elements	128	551	791	108	323	324	944	684	784
Max. element size	60	16	12	60	24	18	12	24	20
Min. degree	17.63 #	2.04*	0.79*	8.17*	1.49 #	0.53*	1.35 #	5.76*	21.62*
Min. deg. 1.0	17.25*	2.27	0.82 #	8.38 #	1.60	0.57 #	1.32 #	5.76*	26.02
Min. deg. 1.1	17.48 #	2.37	0.81 #	9.88	1.45*	0.61	1.27*	6.86	26.05
Min. deg. 1.2	20.50	2.43	0.85 #	10.53	1.61	0.68	1.31 #	7.51	25.99
Min. deg. 1.5	39.33	3.66	0.89	12.79	1.54 #	0.91	1.30 #	9.15	40.68
Nested diss.	24.40	3.62	8.24	10.37	1.65	1.09	4.83	11.26	21.78 #
Similarity 1	30.80	4.34	4.36	17.25	2.21	1.82	14.05	16.22	37.21
Similarity 2	35.48	4.26	8.87	17.25	2.41	1.49	7.84	24.14	39.55

Table 6.4 Speed-up with four processors ($s = 650, \alpha = 1, \gamma = 2, \sigma = 100$).

No. of variables	2919	3024	3306	2802	1074	700	2232	3491	5976
No. of elements	128	551	791	108	323	324	944	684	784
Max. element size	60	16	12	60	24	18	12	24	20
Min. degree	3.62	3.17	2.94	3.43	3.46	2.96	2.22	3.50	3.63
Min. deg. 1.0	3.63	3.29	2.94	3.48	3.36	2.98	3.02	3.50	3.64
Min. deg. 1.1	3.62	3.42	3.01	3.59	3.43	2.95	3.30	3.60	3.66
Min. deg. 1.2	3.65	3.51	3.07	3.66	3.42	3.10	3.34	3.61	3.66
Min. deg. 1.5	3.65	3.53	3.11	3.68	3.45	3.18	3.36	3.64	3.66
Nested diss.	3.64	3.57	3.51	3.67	3.48	3.28	3.60	3.63	3.64
Similarity 1	3.67	3.56	3.54	3.66	3.53	3.43	3.61	3.66	3.67
Similarity 2	3.68	3.53	3.56	3.66	3.59	3.40	3.63	3.68	3.66

Table 6.5 Costs in parallel mode with 32 processors in millions of units ($s = 650, \alpha = 1, \gamma = 2, \sigma = 100$).
*best #within 10%

No. of variables	2919	3024	3306	2802	1074	700	2232	3491	5976
No. of elements	128	551	791	108	323	324	944	684	784
Max. element size	60	16	12	60	24	18	12	24	20
Min. degree	2.83#	0.87	0.32	2.21	0.38*	0.28*	0.89	1.24	3.23#
Min. deg. 1.0	2.61*	0.74	0.33	2.01	0.46	0.29#	0.62	1.22	3.79
Min. deg. 1.1	2.76#	0.64	0.29#	1.78	0.39#	0.32	0.34	1.09*	3.65
Min. deg. 1.2	2.97	0.46*	0.27*	1.58#	0.43	0.28*	0.33	1.16#	3.63
Min. deg. 1.5	5.63	0.70	0.27*	1.79	0.38*	0.32	0.29*	1.39	5.76
Nested diss.	3.60	0.63	1.54	1.44*	0.40#	0.40	0.80	1.76	3.13*
Similarity 1	4.28	0.74	0.77	2.44	0.44	0.44	2.17	2.28	5.08
Similarity 2	4.89	0.79	1.53	2.44	0.45	0.38	1.17	3.22	5.48

Table 6.6 Speed-up with 32 processors ($s = 650, \alpha = 1, \gamma = 2, \sigma = 100$).

No. of variables	2919	3024	3306	2802	1074	700	2232	3491	5976
No. of elements	128	551	791	108	323	324	944	684	784
Max. element size	60	16	12	60	24	18	12	24	20
Min. degree	22.56	7.40	7.28	12.67	13.52	5.54	3.35	16.28	24.35
Min. deg. 1.0	24.01	10.07	7.46	14.49	11.67	5.79	6.44	16.48	24.96
Min. deg. 1.1	22.88	12.57	8.60	19.91	12.64	5.69	12.41	22.69	26.13
Min. deg. 1.2	25.21	18.40	9.53	24.30	12.80	7.52	13.36	23.40	26.14
Min. deg. 1.5	25.48	18.60	10.32	26.29	13.85	8.95	15.07	24.02	25.85
Nested diss.	24.67	20.63	18.75	26.43	14.53	9.03	21.58	23.29	25.31
Similarity 1	26.38	20.92	20.03	25.92	17.61	14.32	23.37	26.07	26.88
Similarity 2	26.68	19.03	20.64	25.92	19.27	13.24	24.33	27.63	26.45

Table 6.7 Cost in parallel mode and speed-up with 32 processors for the min. deg. 1.0 algorithm on the first problem ($\alpha = 1, \gamma = 2, \sigma = 500$).

s	350	650	1000	2000	5000	10000	50000
Cost	4.00	3.40	3.34	3.92	6.82	11.87	26.38
Speed-up	15.67	18.46	18.76	15.95	9.19	5.28	2.38

Table 6.8 Cost in parallel mode and speed-up with 32 processors for the fourth problem for various σ ($s = \max(650, \sigma)$, $\alpha = 1$, $\gamma = 2$).
*best #within 10%

	Cost			Speed-up		
σ	10	100	1000	10	100	1000
Min. degree	2.07	2.21	4.23	13.56	12.67	6.62
Min. deg. 1.0	1.88	2.01	3.73	15.52	14.49	7.83
Min. deg. 1.1	1.66	1.78	3.12	21.35	19.91	11.36
Min. deg. 1.2	1.47#	1.58#	2.55	26.21	24.30	15.10
Min. deg. 1.5	1.66	1.79	2.74	28.28	26.29	17.15
Nested diss.	1.34*	1.44*	2.23*	28.43	26.43	17.10
Similarity 1	2.28	2.44	3.69	27.74	25.92	17.14
Similarity 2	2.28	2.44	3.69	27.74	25.92	17.14

Of course, depending on the computer, the appropriate values for the parameters will vary. We have therefore run a number of our tests with different values to assess the sensitivity.

Different values of the parameter γ were tested. The choice of $\gamma = 1$ instead of $\gamma = 2$, for instance, approximately halves the cost of the different algorithms since the elimination operations are dominant. There was little effect on the speed-ups, so that the conclusions of this section still hold for other values of γ.

We then tuned the parameter s. Table 6.7 shows the evolution with s of the cost in parallel mode (and the related speed-up) of the minimum-degree 1.0 algorithm, run on the first problem with 32 processors.

We noticed that for all the algorithms, the speed-ups are sensitive to the value of s. A very large s causes each node of the elimination tree to be regarded as atomic (no parallelism inside the nodes), and too little parallelism is achieved. Small values of s must be used to increase the parallelism inside the nodes, allowing speed-ups nearly equal to the number of processors. Nonetheless, we must ensure that s is greater than the maximum front size and is not too small with respect to σ, the cost of spawning a task.

Table 6.8 presents numerical experiments with different values of σ for the fourth problem using 32 processors. An increase of the cost of spawing a task means a reduction of parallelism for all the algorithms.

The relative performance is not greatly affected, but the minimum-degree and minimum-degree 1.0 algorithms, with their tall and thin trees, deteriorate more than the others as σ increases. From previous experiments, we know that s must be chosen dependent on σ, which is why we choose, for our experiments, $s = \max(650, \sigma)$. Large values of σ would represent the context of distributed memory machines, since this parameter could include the costs of initialization of data transfers between the processors.

Finally, we varied the number of processors to analyse the efficiency of our algorithms. Tables 6.9 and 6.10 present the costs and speed-ups of the different algorithms for p ranging from 4 to 256.

We notice that the minimum-degree algorithm takes less advantage of large p, which indicates that much more parallelism is needed for a greater number of processors. The variants of the minimum-degree algorithm seem better and,

Table 6.9 Costs in parallel mode (in millions of units) for the cylinder problem ($s = 650, \alpha = 1, \gamma = 2, \sigma = 100$). *best #within 10%

p	4	8	16	32	64	128	256
Min. degree	17.63#	9.12#	4.87#	2.83#	1.88	1.71	1.69
Min. deg. 1.0	17.25*	8.86*	4.68*	2.61*	1.67*	1.39#	1.38#
Min. deg. 1.1	17.48#	9.03#	4.82#	2.76#	1.78#	1.52	1.50
Min. deg. 1.2	20.50	10.46	5.44	2.97	1.82#	1.38*	1.33*
Min. deg. 1.5	39.33	20.08	10.42	5.63	3.30	2.23	1.84
Nested diss.	24.40	12.50	6.55	3.60	2.18	1.53	1.40#
Similarity 1	30.80	15.65	8.06	4.28	2.44	1.63	1.38#
Similarity 2	35.48	18.00	9.24	4.89	2.78	1.81	1.47

Table 6.10 Speed-up for the cylinder problem ($s = 650, \alpha = 1, \gamma = 2, \sigma = 100$).

p	4	8	16	32	64	128	256
Min. degree	3.62	6.99	13.09	22.56	33.97	37.41	37.81
Min. deg. 1.0	3.63	7.07	13.41	24.01	37.57	44.99	45.57
Min. deg. 1.1	3.62	7.00	13.11	22.88	35.51	41.56	42.04
Min. deg. 1.2	3.65	7.16	13.76	25.21	41.18	54.24	56.37
Min. deg. 1.5	3.65	7.14	13.78	25.48	43.51	64.47	77.85
Nested diss.	3.64	7.11	13.58	24.67	40.75	58.10	63.47
Similarity 1	3.67	7.22	14.01	26.38	46.24	69.34	82.04
Similarity 2	3.68	7.25	14.11	26.68	46.90	72.13	88.78

from all our experiments, minimum-degree 1.0 and minimum-degree 1.2 seem to be good candidates.

The efficiency, which is often defined as the ratio of the speed-up to the number of processors, gives us an idea of the effective use (or not) of the available processors. We see that the above algorithms lead to efficiencies higher than 0.5 for 32 or 64 processors, which shows that the multifrontal methods are well suited for shared-memory machines with such a number of processors. The low efficiency for 256 processors, however, seems to prove that the algorithms are not well adapted for distributed memory machines with very large numbers of processors unless the problem size is greater than that of any of our tests.

6.9. Conclusions

Our numerical experiments show that on shared-memory machines with only four processors, there is little justification for using an algorithm other than minimum degree.

However, for shared-memory machines with a greater number of processors (up to 128 for instance), the minimum-degree algorithm does not provide enough parallelism for an effective use of the available computational units. We observe that variants of the minimum-degree algorithm, like minimum-degree 1.0 or minimum-degree 1.2, are better in this context.

Finally, it seems difficult using this current approach to implement an efficient multifrontal method for machines with a much greater number of processors (more than 128 for instance) since, up to now, we do not have at our disposal an algorithm that generates a sufficient number of tasks which can be executed in parallel. Indeed it is likely that the problem would have to be considerably larger than those we have tested for high-level parallelism to be useful.

Acknowledgements

The project has been financially supported by the Belgian National Fund for Scientific Research and Harwell Laboratory (U.K.A.E.A.). This support is gratefully acknowledged. The authors thank Philip Sweetenham for his helpful comments on the paper.

References

Chu, E., George, A., Liu, J., and Ng, E. (1984). *SPARSPAK: Waterloo sparse matrix package user's guide for SPARSPAK-A*. Report CS-84-36, Department of Computer Science, University of Waterloo, Ontario, Canada.

Duff, I. S. (1986). Parallel implementation of multifrontal schemes. *Parallel Computing* **3**, 193–204.

Duff, I. S. (1989). *Multiprocessing a sparse matrix code on the Alliant FX/8.* J. Comp. Appl. Math. **27**, 229–39.

Duff, I. S. and Reid, J. K. (1983). The multifrontal solution of indefinite sparse symmetric linear systems. *ACM Trans. Math. Soft.* **9**, 302–25.

Duff, I. S., Erisman, A. M., and Reid, J. K. (1986). *Direct methods for sparse matrices.* Oxford University Press, London.

Duff, I. S., Grimes, R. G., and Lewis, J. G. (1989). *Sparse matrix test problems.* ACM Trans. Math. Soft. **15**, 1–14.

George, A. (1973). Nested dissection of a regular finite-element mesh. *SIAM J. Numer. Anal.* **10**, 345–63.

George, A. and Liu, J. W. H. (1978). An automatic nested dissection algorithm for irregular finite-element problems. *SIAM J. Numer. Anal.* **15**, 1053–69.

George, A. and Liu, J. W. H. (1987). *The evolution of the minimum degree ordering algorithm.* Report ORNL/TM-10452, Oak Ridge National Laboratory, Tennessee.

George, A., Heath, M. T., and Liu, J. W. H. (1986). Parallel Choleski factorization on a shared-memory multiprocessor. *Lin. Algebra and Appl.* **77**, 165–87.

Gibbs, N. E., Poole, W. G., Jr., and Stockmeyer, P. K. (1976). An algorithm for reducing the bandwidth and profile of a sparse matrix. *SIAM J. Numer. Anal.* **13**, 236–50.

Griewank, A. and Toint, Ph. L. (1982). Partitioned variable metric updates for large structured optimization problems. *Numer. Math.* **39**, 119–37.

Liu, J. W. H. (1986). Computational models and task scheduling for parallel sparse Cholesky factorization. *Parallel Computing* **3**, 327–42.

Reid, J. K. (1984). *TREESOLVE. A Fortran package for solving large sets of linear finite-element equations.* Report CSS 155, Computer Science and Systems Division, Harwell Laboratory.

Strodiot, J.-J. and Toint Ph. L. (1976). *An algorithm for unconstrained minimization of large scale problems by decomposition in subspaces.* Report 76/1 of the FNDP, Namur (Belgium).

Tinney, W. F. and Walker, J. W. (1967). Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proc. IEEE* **55**, 1801–9.