A Frontal Code for the Solution of Sparse Positive-Definite Symmetric Systems Arising from Finite-Element Applications

IAIN S. DUFF and JENNIFER A. SCOTT
Rutherford Appleton Laboratory

We describe the design, implementation, and performance of a frontal code for the solution of large sparse symmetric systems of linear finite-element equations. The code is intended primarily for positive-definite systems, since numerical pivoting is not performed. The resulting software package, MA62, will be included in the Harwell Subroutine Library. We illustrate the performance of our new code on a range of problems arising from real engineering and industrial applications. The performance of the code is compared with that of the Harwell Subroutine Library general frontal solver MA42 and with other positive-definite codes from the Harwell Subroutine Library.

Categories and Subject Descriptors: G.1.0 [**Numerical Analysis**]: General—*Numerical algorithms*; G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*Sparse, structured and very large systems* (direct and iterative methods)

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Finite-element equations, Gaussian elimination, Level 3 BLAS, sparse symmetric linear equations, symmetric frontal method

1. INTRODUCTION

In a previous article [Duff and Scott 1996], we discussed the design and performance of a general frontal code, MA42, for solving sparse unsymmetric systems of equations. It was observed in that article that MA42 is frequently used to solve finite-element problems for which the system matrix is symmetric and positive-definite. However, apart from offering an option of restricting pivoting to the diagonal, MA42 does not exploit symmetry or positive definiteness, and, as a result, the code is more expensive in terms of both storage requirements and operation counts than it need be for this class of problems. Our goal is to design and develop a frontal code

Authors' address: Computational Science and Engineering Department, Rutherford Appleton Laboratory, Atlas Centre, Oxon, OX11 0QX, England; email: i.s.duff@rl.ac.uk; j.a.scott@rl.ac.uk.

specifically for the efficient solution of sparse positive-definite symmetric systems from finite-element applications. The new code is called MA62.

Before discussing the design of MA62, we briefly describe some of the key features of the frontal method. We are concerned with the solution of linear systems

$$\mathbf{AX} = \mathbf{B} \tag{1}$$

where the $n \times n$ matrix $\mathbf{A}$ is large, sparse, and symmetric. $\mathbf{B}$ is an $n \times nrhs$ ($nrhs \geq 1$) matrix of right-hand sides, and $\mathbf{X}$ is the $n \times nrhs$ solution matrix. In this article, we are only interested in the case where the matrix $\mathbf{A}$ is an elemental matrix; that is, it is a sum of finite-element matrices

$$\mathbf{A} = \sum_{l=1}^{m} \mathbf{A}^{(l)}, \tag{2}$$

where each element matrix $\mathbf{A}^{(l)}$ has nonzeros only in a few rows and columns and corresponds to the matrix from element l. The frontal method is a variant of Gaussian elimination that was originally developed by Irons [1970] and was later extended and modified by Hood [1976] and Duff [1981; 1983; 1984]. For the symmetric system (1), the method involves the matrix factorization

$$\mathbf{A} = (\mathbf{PL})\mathbf{D}(\mathbf{PL})^T, \tag{3}$$

where $\mathbf{P}$ is a permutation matrix, $\mathbf{D}$ is a diagonal matrix, and $\mathbf{L}$ is a unit lower triangular matrix. If $\mathbf{A}$ is a positive-definite matrix, the entries of $\mathbf{D}$ are positive. The solution process is completed by performing the forward elimination

$$(\mathbf{PL})\mathbf{Z} = \mathbf{B}, \tag{4}$$

then the diagonal solution

$$\mathbf{DY} = \mathbf{Z}, \tag{5}$$

followed by the back substitution

$$(\mathbf{PL})^T\mathbf{X} = \mathbf{Y}. \tag{6}$$

The main feature of the method is that the contributions $\mathbf{A}^{(l)}$ from the finite elements are assembled one at a time and the storage of the entire coefficient matrix $\mathbf{A}$ is avoided by interleaving assembly and elimination operations. An assembly operation is of the form

$$a_{ij} \Leftarrow a_{ij} + a_{ij}^{(l)}, \tag{7}$$

where $a_{ij}^{(l)}$ is the (i, j)th nonzero entry of the element matrix $\mathbf{A}^{(l)}$. A variable is *fully summed* if it is involved in no further sums of the form (7) and is *partially summed* if it has appeared in at least one of the elements assembled so far but is not yet fully summed. The Gaussian elimination operation

$$a_{ij} \Leftarrow a_{ij} - a_{il}[a_{ll}]^{-1}a_{lj} \tag{8}$$

may be performed as soon as all the terms in the triple product in (8) are fully summed. At any stage during the assembly and elimination processes, the rows and columns corresponding to the fully and partially summed variables are held in a dense matrix, termed the *frontal* matrix. The power of frontal schemes comes from the following observations:

—since the frontal matrix is held as a dense matrix, dense linear algebra kernels can be used during the numerical factorization,

—the matrix factors need not be held in main memory, which allows large problems to be solved using only modest amounts of high-speed memory.

The remainder of this article is organized as follows. In Section 2, the design of `MA62`, including the structure of the package and important features, is discussed. We look at the use of high-level Basic Linear Algebra Subprograms (BLAS) [Dongarra et al. 1990] in Section 3. In Section 4, we illustrate the performance of the new code, and compare it with our general frontal code `MA42` and with other Harwell Subroutine Library [HSL 1996] codes for the solution of symmetric positive-definite systems. Concluding comments are made in Section 5. Full details of the code, including Specification Sheets and more extensive numerical results, may be found in Duff and Scott [1997]. Information on how to obtain the code is given in Section 6.

When discussing the design of `MA62` we will want to illustrate the effects of our design decisions by numerical examples. We therefore end this section by introducing the problems that we will use for performance testing. In all cases, they arise in real engineering and industrial applications. The problems are all unassembled finite-element problems, and a brief description of each is given in Table I. The number of unassembled element problems available in the Harwell-Boeing Collection [Duff et al. 1992] is limited, and all are small by today's standards. Consequently, we have only selected two representative problems, `cegb3306` and `lock3491`, from this collection. Problem `ramage02` was provided by Alison Ramage of the University of Strathclyde [Ramage and Wathen 1993]; `aeac5081` came from Andrew Cliffe of AEA Technology; and `cham` and `tubu` were from Ron Fowler of the Rutherford Appleton Laboratory. The remaining problems were supplied by Christian Damhaug of Det Norske Veritas, Norway. For all the problems, values for the entries of the element matrices were generated using the Harwell Subroutine Library random-number generator `FA01`, and each was made symmetric and diagonally dominant.

Table I. The Test Problems

Identifier	Order	Number of Elements	Description/discipline
aeac5081	5081	800	Double glazing problem
cegb3306	3222	791	2.5D Framework problem
cham	12834	11070	Part of an engine cylinder
crplat2	18010	3152	Corrugated plate field
fcondp2	201822	35836	Oil production platform
fullb	199187	59738	Full-breadth barge
halfb	224617	70211	Half-breadth barge
lock3491	3416	684	Lockheed cross-cone problem
opt1	15449	977	Part of oil production platform
mt1	97578	5328	Tubular joint
ramage02	16830	1400	3D Navier-Stokes
shipsec1	140874	41037	Full-breadth section of ship
srb1	54924	9240	Space shuttle rocket booster
thread	29736	2176	Threaded connector
trdheim	22098	813	Mesh of the Trondheim fjord
tsyl201	20685	960	Part of oil production platform
tubu	26573	23446	Engine cylinder model

In all the tables of results in which floating-point operation counts ("ops") are quoted, we count all operations (+, –, *, /) equally. Times quoted are CPU times in seconds and include the I/O overhead for the codes that use direct-access files. The experimental results given in this article were obtained using 64-bit floating-point arithmetic on a single processor of a CRAY J932. The vendor-supplied BLAS were used. Numerical results for other computers are given by Duff and Scott [1997].

2. THE SOFTWARE PACKAGE MA62

In this section, we describe the structure of the MA62 package and its main features. We highlight how it has been possible to take advantage of symmetry and positive definiteness when designing the code.

2.1 The structure of MA62

The MA62 code has three distinct phases.

(1) An analyze phase that comprises a prepass and a symbolic factorization. The prepass determines in which element each variable appears for the last time and thus when a variable is fully summed and can be eliminated. The symbolic factorization uses the information from the prepass to determine the amount of real and integer storage required for the factorization.

(2) A factorization phase in which the information from the analyze phase is used in the factorization of the matrix (2). In addition, if element right-hand sides $\mathbf{B}^{(l)}$ are specified, the equations (1) with right-hand side(s) $\mathbf{B} = \Sigma_{l=1}^{m} \mathbf{B}^{(l)}$ are solved.

(3) A solution phase which uses the factors produced by the factorization phase to rapidly solve for further right-hand sides **B**. Use of the solution phase is optional.

We remark that, in general, one of the difficulties facing the user of a frontal code is the need to specify file sizes for the factors and the maximum front sizes required before the computation begins. The symbolic factorization works only with the variable indices associated with the elements and, by assuming each variable may be eliminated as soon as it is fully summed, is able to determine the maximum order of the frontal matrix and the file sizes needed for the factors. For general unsymmetric matrices, the need to incorporate pivoting means the statistics returned by the symbolic factorization are only lower bounds on the front size and file sizes actually needed. But for positive-definite matrices a variable can always be eliminated once it is fully summed, and, since the symbolic factorization is inexpensive, we decided to require the MA62 user to perform a symbolic factorization before the numerical factorization commences. If the user provides the numerical factorization with less space than that determined by the symbolic factorization, it can be detected immediately and the computation terminated with an error message.

2.2 Reverse Communication

The interface to the user is through reverse communication with control returned to the calling program each time an element needs to be input. This provides the user with flexibility when choosing how to hold the element matrices. In particular, the user may choose to generate an element only when it is required. This is useful when solving problems that are too large to allow all the elements to be stored at once.

The only phase of MA62 that does not use reverse communication is the optional solve phase. Here the right-hand-side matrices **B** must be input in assembled form. The code was designed in this way, since the user may already have the right-hand sides in assembled form, and, if not, it was felt that it would be straightforward for the user to perform the assembly, leaving the optional solve with a simple interface.

2.3 Use of Direct-Access Files

The user may store the matrix factors in direct-access files. MA62 uses two direct-access files, one for the reals in the factors and one for the indices of the variables in the factors. The user must specify the stream numbers for the direct-access files and may name the files. The length of the records in each of the direct-access files (which is equal to the length of an associated in-core buffer) is chosen by the user. Use of direct-access files is unnecessary if there is sufficient main memory for the factors. Numerical experiments have shown, for some machines, that the overheads which result from using direct-access files can be considerable, particularly during the

solve phase (see Duff and Scott [1997]). Therefore, in MA62 the call to the routine that initializes the direct-access files is optional.

We remark that the unsymmetric frontal code MA42 uses three direct-access files, since it stores the real values for the **L** and **U** factors of **A** separately. Moreover, since the unsymmetric code uses off-diagonal pivoting to maintain stability, it is necessary to hold both row and column indices of the variables in the frontal matrix. Therefore, MA62 requires substantially less real and integer storage than MA42. This is illustrated in Table VIII in Section 4.1.

2.4 Error Checks

We have incorporated many tests in the program to check that the user supplies consistent data and carries out the calls to the different subroutines in the correct sequence. Since the space required by the numerical factorization cannot exceed that determined by the symbolic factorization, on the assumption that the user supplies consistent data, the only way MA62 can terminate before the factorization is complete is if the matrix **A** is found not to be positive-definite. If a pivot is found to be too small, an error flag is set, and control is returned to the user. Note that although the factorization will not proceed, it need not be the case that the matrix is singular. For example, consider the 2×2 matrix with sparsity pattern

$$\begin{pmatrix} 0 & \times \\ \times & \times \end{pmatrix}.$$

If a pivot is found to be negative, the matrix is not positive-definite. However, provided the absolute value of the pivot is not too small, the computation continues, and information on the number of negative pivots is returned to the user at the end of the computation.

2.5 Element Ordering

The number of floating-point operations and the storage requirements are dependent on the size of the frontal matrix at each stage of the computation. Since the size 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 element matrices are input is critical for efficiency. Considerable attention has been given to the problem of automatically choosing a good ordering. Many of the proposed algorithms are similar to those for bandwidth reduction of assembled matrices (see, for example, Duff et al. [1989], and the references therein). To give the user the greatest flexibility in deciding how to order the elements, we have chosen not to incorporate element ordering within the MA62 package. However, the Harwell Subroutine Library routine MC63 [Scott 1999] can be used to preorder the elements for MA62.

Table II. The Effect of Preordering with MC63

	Max Front Size		RMS Front Size	
Identifier	Before	After	Before	After
aeac5081	150	155	142	110
cegb3306	354	78	246	60
cham	1181	412	769	332
crplat2	1570	336	1184	244
fcondp2	6714	4374	5058	2677
fullb	132321	2826	95181	2021
halfb	171225	2352	121610	1608
lock3491	834	180	583	123
mt1	12144	2436	8196	1335
opt1	2681	774	2068	528
ramage02	1717	1452	1492	1329
shipsec1	93036	3654	52642	2395
srb1	2286	546	1506	318
thread	6942	2877	4410	1962
trdheim	276	324	182	146
tsyl201	1200	540	862	511
tubu	2290	645	1299	408

The importance of good element ordering is illustrated in Tables II and III. In these tables, the maximum front size (Max Front Size) is defined to be

$$\max_{l=1,\,n} f_l,$$

and the root-mean squared front size (RMS Front Size) is given by

$$\left(\sum_{l=1}^{n} (f_l)^2 / n \right)^{1/2},$$

where f_l denotes the number of variables in the front before the lth elimination. In Table III, we give results for MA62 before and after the elements are ordered using MC63. Here and elsewhere the storage includes both the real and integer storage. On the CRAY, integers and reals are stored in 64-bit words, so the storage statistics given in Table III are just the sum of the number of real and the number of integer words needed. The "Solve" times quoted are for a single right-hand side. Note that we were not able to run the largest problems without reordering, because of the memory and CPU times they required. We observe, in general, that reordering significantly reduces the factor and solve times, as well as the operation count and the storage for the factors. fcondp2 is the only problem for which MA62 performs better with the initial ordering than with the MC63 ordering. With the initial ordering, the symbolic factorization predicts the factor storage required will be 991160, and for the MC63 ordering predicts factor storage of 520585. We see, however, that the actual factor storage is 292019 and 488670, respectively. This difference between the predicted and

Table III. The Results of Using the Element Ordering Routine MC63 with the Frontal Solver MA62. Cases marked NS were not solved.

	Factorization Time (seconds)		Solve Time (seconds)		Factor ops ($*10^7$)		Storage in Words ($*10^3$)	
Identifier	Before	After	Before	After	Before	After	Before	After
aeac5081	1.56	1.15	0.09	0.08	12	7	786	586
cegb3306	0.64	0.45	0.03	0.03	3	2	232	219
cham	53.9	13.1	0.95	0.46	766	152	9689	4472
crplat2	126	11.3	1.7	0.49	1919	113	17985	4583
fcondp2	3557	8779	24.8	43.1	57305	140758	292019	488670
fullb	NS	5028	NS	35.6	NS	79695	NS	405607
halfb	NS	3461	NS	29.9	NS	53817	NS	344857
lock3491	4.0	0.99	0.11	0.06	51	7	1123	448
mt1	NS	1118	NS	11.1	NS	17661	NS	125694
opt1	54.5	31.6	0.9	0.7	857	437	10031	7596
ramage02	232	175	2.2	1.8	3819	2852	25547	21847
srb1	550	48.9	5.9	1.7	8604	595	65894	18058
shipsec1	NS	5450	NS	28.7	NS	86304	NS	333791
thread	1925	680	7.0	4.5	32996	11301	92048	55207
trdheim	5.7	4.6	0.37	0.32	52	37	2872	2389
tsyl201	98.1	38.3	1.5	1.0	1528	535	17236	10654
tubu	122	35.6	1.7	1.1	1859	440	17277	10742

actual storage can be accounted for by the exploitation of zeros in the front during the numerical factorization. This is discussed further in Section 3.3.

3. THE USE OF HIGH-LEVEL BLAS

In this section, we describe the use of the BLAS in performing both the numerical factorization and the forward elimination and back substitution steps.

3.1 Use of BLAS during the Factorization

Using Fortran 90 notation, the innermost loop of the frontal method can be written as

```
do j = 1,frnt
  if (pr(j).ne. 0.0) f(1:frnt,j) = f(1:frnt,j) + &
                                   pc(1:frnt)*pr(j)
end do
```

where f is the frontal matrix, pc is the pivot column, pr is the pivot row, and frnt is the order of the frontal matrix. This code represents a rank-one update to the matrix that can be performed using the Level 2 BLAS routine _GER. After the assembly of an element, if there are k fully summed variables which can be eliminated, then k calls to _GER can be made. However, as illustrated by Cliffe et al. [1998], the computation is made significantly more efficient (in terms of the factor and solve times) if we avoid updating the frontal matrix until all pivots for the current element

have been chosen. After the assembly of an element, the frontal matrix has the form

$$\mathbf{F} = \begin{pmatrix} \mathbf{F}_T & \mathbf{F}_C^T \\ \mathbf{F}_C & \mathbf{F}_U \end{pmatrix}, \tag{9}$$

where the submatrices $\mathbf{F}_T$ and $\mathbf{F}_C$ are fully summed. Since the pivots may be chosen from the diagonal of $\mathbf{F}_T$ in order, we can compute the factorization

$$\begin{pmatrix} \mathbf{F}_T & \mathbf{F}_C^T \\ \mathbf{F}_C & \mathbf{F}_U \end{pmatrix} = \begin{pmatrix} \mathbf{F}_{TL} & \mathbf{0} \\ \mathbf{F}_L & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{D}_T & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{F}} \end{pmatrix} \begin{pmatrix} \mathbf{F}_{TL}^T & \mathbf{F}_L^T \\ \mathbf{0} & \mathbf{I} \end{pmatrix}, \tag{10}$$

where

$$\mathbf{F}_T = \mathbf{F}_{TL}\mathbf{D}_T\mathbf{F}_{TL}^T, \tag{11}$$

$$\mathbf{F}_L = \mathbf{F}_C(\mathbf{D}_T\mathbf{F}_{TL}^T)^{-1}, \tag{12}$$

and where the Schur complement $\hat{\mathbf{F}}$ is given by

$$\hat{\mathbf{F}} = \mathbf{F}_U - \mathbf{F}_L\mathbf{D}_T\mathbf{F}_L^T. \tag{13}$$

From (12), $\mathbf{F}_L\mathbf{D}_T = \mathbf{F}_C\mathbf{F}_{TL}^{-T}$, so the Level 3 kernel _TRSM can be used to form $\mathbf{F}_L\mathbf{D}_T$; $\mathbf{F}_L^T$ is obtained by diagonal scaling. If the number of fully summed variables is KR, the Level 3 kernel _GEMM with internal dimension KR may then be used to compute (13). However, since only the upper triangular part of $\hat{\mathbf{F}}$ is needed, forming the whole of $\hat{\mathbf{F}}$ involves many unnecessary operations. With a front size of FRNT, the computation (13) requires 2 * KR * (FRNT − KR)^2 floating-point operations. We could form only the upper triangle of $\hat{\mathbf{F}}$ by updating each column using the Level 2 kernel _GEMV. This would reduce the number of floating-point operations to KR * (FRNT − KR) * (FRNT − KR + 1), but the efficiency gains of using Level 3 BLAS would be lost. To take advantage of Level 3 BLAS while at the same time restricting the number of unnecessary operations, the columns of $\hat{\mathbf{F}}$ may be computed in blocks. Assuming a block size of NB with FRNT − KR = k * NB then, using Fortran 90 section notation, we have

$$\hat{\mathbf{F}}(1:\mathtt{K}, \mathtt{K}-\mathtt{NB}+1:\mathtt{K}) = \mathbf{F}_U(1:\mathtt{K}, \mathtt{K}-\mathtt{NB}+1:\mathtt{K}) -$$

$$(\mathbf{F}_L\mathbf{D}_T)(1:\mathtt{K}, 1:\mathtt{KR}) * \mathbf{F}_L^T(1:\mathtt{KR}, \mathtt{K}-NB+1:\mathtt{K}), \tag{14}$$

where K = K$_1$ * NB, K$_1$ = 1, 2, . . . , k. The update (14) can be performed using _GEMM with interior dimension KR (the Level 2 kernel _GEMV is used if KR = 1).

In finite-element applications, all the variables (degrees of freedom) associated with a finite-element node become fully summed at the same

Table IV. Storage Statistics and Operation Counts for Different Values of the Minimum Pivot Block Size

Identifier	Minimum Pivot Block	Largest Pivot Block	Maximum Front Size	Factor ops ($*10^6$)	Storage in Words ($*10^3$)	
					Real	Integer
aeac5081	1	10	156	69	530	90
	8	17	161	71	537	48
	16	25	170	77	562	33
	32	41	185	88	600	20
	40	49	195	95	626	18
cegb3306	1	6	78	14	179	33
	8	12	84	15	191	19
	16	18	90	17	204	14
	32	36	108	22	236	8
	40	42	114	24	246	8
opt1	1	38	983	5470	7971	503
	8	38	983	5473	7974	490
	16	39	995	5524	8019	353
	32	67	1009	5657	8137	226
	40	67	1009	5729	8195	193
tubu	1	8	645	4583	10223	8747
	8	14	649	4646	10311	1295
	16	18	657	4729	10415	665
	32	35	673	4906	10628	347
	40	44	681	5000	10738	283

assembly step. Therefore, in general, the pivot block size will be greater than one. However, we can choose to perform further element assemblies and delay performing eliminations until the pivot block is at least some prescribed minimum. By increasing the minimum number of pivots that are selected at each stage, KR is increased, and greater efficiency can be gained from using _GEMM to perform (14). This is discussed further in Cliffe et al. [1998]. When designing MA62, we decided to allow the user to specify the minimum pivot block size. The costs that result from delaying performing eliminations are a higher floating-point operation count, increased storage for the reals in the factor, and, in general, an increase in the maximum front size and consequently in the main memory required. This is illustrated for a subset of our test problems in Table IV. It is apparent that modest increases in the minimum pivot block size have little effect on the size of the largest pivot block and on the maximum front size, and that the real storage requirement and the operation count grow slowly with the pivot block size. Since the symbolic factorization is cheap to perform, the user may want to look at the effect on the maximum front size and the file sizes of different minimum pivot block sizes before starting the numerical factorization. MA62 has been designed to allow this.

In Table V, we give factor and solve times for different minimum pivot block sizes. The factor times indicate that, in general, savings can be

Table V. Factor and Solve Times for Different Minimum Pivot Block Sizes. nrhs denotes the number of right-hand sides.

Identifier	Minimum Pivot Block	Factor Time (seconds)	Solve Time (seconds)		
			nrhs = 1	nrhs = 2	nrhs = 10
aeac5081	1	1.4	0.13	0.18	0.55
	8	1.1	0.09	0.12	0.37
	16	1.1	0.07	0.10	0.31
	32	1.3	0.06	0.08	0.27
	40	1.3	0.06	0.08	0.26
crplat2	1	27.3	1.06	1.39	4.66
	8	22.9	0.76	0.98	3.20
	16	20.5	0.62	0.84	2.60
	32	21.1	0.56	0.72	2.24
	40	20.3	0.55	0.71	2.18
opt1	1	39.2	0.81	1.02	3.27
	8	38.9	0.79	1.02	3.25
	16	37.0	0.70	0.91	2.80
	32	36.5	0.66	0.85	2.58
	40	36.4	0.63	0.82	2.55
tubu	1	62.1	2.91	4.09	11.87
	8	43.3	1.47	1.88	6.08
	16	37.4	1.08	1.37	4.45
	32	35.9	0.94	1.21	3.88
	40	36.5	0.89	1.15	3.56

achieved by choosing the minimum pivot size to be greater than 1, but the precise choice does not appear critical. This is important from a practical point of view because it is possible to get good performance without having to optimize the parameter from run to run. The solve times are also reduced by increasing the minimum pivot size. This is because greater efficiency is gained from the BLAS in the solution phase if KR is increased and because the integer data written to and read from the buffers is reduced, and, as a result, the amount of data movement during the solution phase is reduced (see Section 3.2). On the basis of these results and numerical experiments on other computing platforms (see Duff and Scott [1997]), the default value for the minimum pivot size is set to 16 in MA62. However, we note that if the user is going to perform a large number of solves, it may be beneficial to choose a value larger than this.

In MA62, the size NB of the blocks used in (14) may also be chosen by the user. In Table VI we show the effect of using different block sizes. The timings quoted confirm that it is generally advantageous to exploit Level 3 BLAS (that is, to use a block size greater than 1), albeit at the cost of an increased operation count. The precise choice for NB does not appear important. As a result of our numerical experiments, we have chosen the default value for the block size in MA62 to be 16.

Table VI. The Number of Operations (in millions) and Time (in seconds) Required to Perform the Numerical Factorization for Different Block Sizes

	Block Size NB			
Identifier	1	8	16	32
aea5801	68.7/ 1.30	72.5/ 1.17	77.3/ 1.20	88.3/ 1.23
cegb3306	14.1/ 0.49	15.8/ 0.48	17.0/ 0.45	20.3/ 0.48
crplat2	2526/ 24.0	2571/ 21.1	2624/ 20.9	2737/ 21.1
lock3491	47.5/ 0.93	50.0/ 0.85	53.4/ 0.84	61.6/ 0.88
trdheim	469/ 6.7	484/ 6.0	502/ 6.0	543/ 6.2
tubu	4572/ 44.6	4644/ 37.8	4729/ 37.6	4908/ 38.2

Although the numerical factorization described above is straightforward, its implementation within MA62 using Level 3 BLAS kernels is nontrivial. Elements are assembled into the frontal matrix FA until the number of fully summed variables is at least the minimum pivot block size chosen by the user. The fully summed columns are then permuted to the last columns of FA, and, if supplied, the corresponding rows of the right-hand-side array FRHS are permuted to the end of the array. When an element is assembled into FA, only the upper triangular part of FA contains meaningful data, and symmetry must be exploited to perform the column interchanges. We remark that, by permuting the fully summed columns to the last columns of FA, once the eliminations have been performed and the resulting data are written to the in-core buffers, the last columns of FA can be reset to zero and the next element assembled. If, instead, we were to permute the fully summed columns to the start of FA, further costly data movement would be required.

3.2 Use of BLAS during the Solution Phase

The other main use of high-level BLAS is in the solution phase. From (10), the columns of $\mathbf{PL}$ corresponding to the KR variables eliminated at the same stage are

$$\begin{pmatrix} \mathbf{F}_{TL} \\ \mathbf{F}_{L} \end{pmatrix}. \tag{15}$$

We use direct addressing in the solution phase to exploit this block structure. At each stage of the forward elimination, all the active components of the partial solution vectors $\mathbf{Y}$ (where $(\mathbf{PL})\mathbf{DY} = \mathbf{B}$) are gathered into an array $\mathbf{W} = (\mathbf{W}_1, \mathbf{W}_2)^T$, with $\mathbf{W}_1$ of dimension KR $\times$ nrhs and $\mathbf{W}_2$ of dimension FRNT $-$ KR $\times$ nrhs. The operations

$$\mathbf{W}_1 \leftarrow \mathbf{F}_{TL}^{-1}\mathbf{W}_1 \tag{16}$$

followed by

$$\mathbf{W}_2 \leftarrow \mathbf{W}_2 - \mathbf{F}_L\mathbf{W}_1 \tag{17}$$

and finally

$$\mathbf{W}_1 \leftarrow \mathbf{D}_T^{-1}\mathbf{W}_1 \tag{18}$$

are performed before $\mathbf{W}$ is scattered into $\mathbf{Y}$. Similarly, if $\mathbf{Z} = (\mathbf{Z}_1, \mathbf{Z}_2)^T$, with $\mathbf{Z}_1$ and $\mathbf{Z}_2$ of leading dimension KR and FRNT-KR, respectively, during the back substitution, the active components of the partial solution vector $\mathbf{Y}$ are gathered into $\mathbf{Z}$. The operations

$$\mathbf{Z}_1 \leftarrow \mathbf{Z}_1 - \mathbf{F}_L^T\mathbf{Z}_2 \tag{19}$$

and then

$$\mathbf{Z}_1 \leftarrow \mathbf{F}_{TL}^{-T}\mathbf{Z}_1 \tag{20}$$

are carried out before $\mathbf{Z}$ is scattered into $\mathbf{X}$.

In MA62, $\mathbf{F}_L$ is stored by columns, and $\mathbf{F}_{TL}$ is stored in packed form. Provided KR > 1, the forward elimination and back substitutions are performed using the Level 2 BLAS kernels _GEMV and _TPSV if there is only one right-hand side (nrhs = 1), and the Level 3 routine _GEMM and the Level 2 routine _TPSV if there are multiple right-hand sides. We observe that there is no Level 3 BLAS kernel for solving a triangular system of equations with the matrix held in packed form and multiple right-hand sides. We tried storing $\mathbf{F}_{TL}$ as a full matrix (with only the lower triangular part containing meaningful data) and using the Level 2 kernel _TRSV if nrhs = 1 and the corresponding Level 3 kernel _TRSM otherwise. We found the savings from using the packed form are small (generally less than 10% of the real storage requirement). This is illustrated in Table VIII in Section 4.1. The triangular solves (16) and (20) account for a relatively small part of the total solution time. Our experience is that, for a small number of right-hand sides, there is little difference in the CPU times when _TRSM is used in place of _TPSV. As a result, in MA62 the default is to use the packed triangular form, but an option exists for not doing this.

3.3 Exploiting Zeros in the Front

During the factorization, the frontal matrix may contain some zero entries. Treating the frontal matrix as a dense matrix results in unnecessary operations being performed with these zeros. As we have shown, high-level BLAS are used, so that the cost of these operations may not be prohibitive. If, however, the frontal matrix contains a significant number of zeros, it can be advantageous to exploit these. In particular, there are many zeros in the front if the elements are poorly ordered. To see how we can exploit the zeros, suppose the frontal matrix has been permuted to the form (9). By performing further row and column permutations, the frontal matrix can be expressed in the form

$$\mathbf{F} = \begin{pmatrix} \mathbf{F}_T & \mathbf{F}_{C_1}^T & \mathbf{0} \\ \mathbf{F}_{C_1} & \mathbf{F}_{U_1} & \mathbf{F}_{U_2}^T \\ \mathbf{0} & \mathbf{F}_{U_2} & \mathbf{F}_{U_3} \end{pmatrix}, \tag{21}$$

where the zero matrix is of order KR by K for some K with $0 \leq$ K $\leq$ FRNT $-$ KR (FRNT is the current front size and where KR is the number of fully summed variables). The factorization then becomes

$$\begin{pmatrix} \mathbf{F}_T & \mathbf{F}_{C_1}^T & \mathbf{0} \\ \mathbf{F}_{C_1} & \mathbf{F}_{U_1} & \mathbf{F}_{U_2}^T \\ \mathbf{0} & \mathbf{F}_{U_2} & \mathbf{F}_{U_3} \end{pmatrix} = \begin{pmatrix} \mathbf{F}_{TL} & \mathbf{0} & \mathbf{0} \\ \mathbf{F}_{L_1} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{D}_T & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \widehat{\mathbf{F}_{U_1}} & \mathbf{F}_{U_2}^T \\ \mathbf{0} & \mathbf{F}_{U_2} & \mathbf{F}_{U_3} \end{pmatrix} \begin{pmatrix} \mathbf{F}_{TL}^T & \mathbf{F}_{L_1}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{pmatrix}, \tag{22}$$

where

$$\mathbf{F}_{L_1} = \mathbf{F}_{C_1}(\mathbf{D}_T\mathbf{F}_{TL}^T)^{-1}, \tag{23}$$

and $\widehat{\mathbf{F}_{U_1}}$ is given by

$$\widehat{\mathbf{F}_{U_1}} = \mathbf{F}_{U_1} - \mathbf{F}_{L_1}\mathbf{D}_T\mathbf{F}_{L_1}^T. \tag{24}$$

In Table VII, we illustrate the effects of exploiting zeros. Results are included both for the original element order and for the element order generated by MC63. We see that, in general, if the elements are not well ordered, substantial savings are achieved in the factor storage, the number of operations, and the factorize time by exploiting zeros in the front. Once the elements have been ordered, the savings which result from exploiting zeros are much smaller. Indeed, if the savings in the factor storage and operation counts are very small, the overheads of searching for zeros and increased data movement to accumulate the zeros into blocks can result in a (small) increase in the factorize time. We were not able to solve problem fcondp2 within our CPU time limit of six hours if the matrix was not ordered and the zeros in the front were not exploited. For this problem, the storage figures given are those returned by the symbolic factorization. The symbolic factorization takes no account of the zeros in the front. As a result, the file sizes for the factors predicted by the symbolic factorization are exact only if zeros in the front are not exploited. If they are (and this is the default in MA62) the predicted storage for the factors is an upper bound on the storage actually needed.

If KR, the number of fully summed variables, is greater than 1, the matrix $\mathbf{F}_{L_1}$ in (22) may still contain some zeros. However, the results reported in Table VII show that the number of zeros remaining in the factors is small (generally less than 10% of the total number of entries in the factors if the elements have been ordered), so we do not attempt to exploit these remaining zeros.

Table VII. The Effect of Exploiting Zeros in the Front. The figures in parentheses are the number of zeros (in thousands) which are held explicitly in the factors. The case marked NS was not solved because the CPU limit of six hours was exceeded.

Identifier	MC63 Used	Zeros Exploited	Real Storage (*10^3)		Integer Storage in Words (*10^3)	Number of ops (*10^7)	Factorize Time (seconds)
cegb3306	No	No	765	(661)	45	21.5	1.8
	No	Yes	218	(113)	14	3.1	0.6
	Yes	No	207	(51)	14	1.7	0.4
	Yes	Yes	204	(47)	14	1.7	0.4
crplat2	No	No	20734	(3822)	1125	2546	163
	No	Yes	17053	(131)	940	1919	124
	Yes	No	6556	(124)	37	267	20
	Yes	Yes	6490	(59)	37	262	20
fcondp2	No	No	944244	(NS)	46915	NS	NS
	No	Yes	278475	(4006)	13544	57305	3557
	Yes	No	493218	(33257)	27367	151787	9443
	Yes	Yes	462977	(3016)	25693	140758	8779
opt1	No	No	29331	(19865)	1124	6358	385
	No	Yes	9628	(207)	355	857	57
	Yes	No	8142	(214)	358	571	38
	Yes	Yes	7984	(90)	341	552	37
tsyl201	No	No	16693	(248)	709	1546	98
	No	Yes	16521	(88)	701	1528	97
	Yes	No	10416	(48)	485	554	38
	Yes	Yes	10405	(48)	483	554	39

4. NUMERICAL RESULTS

In this section, we compare the performance of MA62 with that of other codes in the Harwell Subroutine Library (HSL). All the HSL linear equation solvers used in our numerical experiments have control parameters with default values. Unless stated otherwise, we use these defaults in each case.

4.1 The Performance of MA62 Compared with MA42

Our aim in designing and developing MA62 was to produce a code that would be much more efficient than the general unsymmetric frontal solver MA42 when used to solve symmetric positive-definite finite-element systems. To assess how successful we have been, in Table VIII we compare the storage requirements and the operation counts for the two codes. For MA62, we give the real storage requirements for storing the lower triangular matrix $\mathbf{F}_{TL}$ in packed form and as a full matrix in which only the lower triangular entries are meaningful (see Section 3.2). In Table IX, we compare the timings for MA62 with those for MA42. The timings quoted for

Table VIII. A Comparison of the Operation Count and Storage Requirements for MA42 and MA62 on Symmetric Positive-Definite Unassembled Finite-Element Systems. The numbers in parentheses are the real storage needed if the packed form for $\mathbf{F}_{TL}$ is not used.

		Factor Storage in Words (*10^3)			
Identifier	Code	Real		Integer	Factor ops (*10^6)
aea5801	MA42	1046		180	116
	MA62	553	(599)	33	74
cegb3306	MA42	377		68	22
	MA62	205	(233)	14	17
cham	MA42	8217		7175	2771
	MA62	4203	(4300)	270	1517
crplat2	MA42	8506		1376	2014
	MA62	4333	(4489)	250	1126
lock3491	MA42	832		142	106
	MA62	376	(405)	22	52
opt1	MA42	14866		1136	8415
	MA62	7269	(7411)	327	4373
ramage02	MA42	41792		3495	55870
	MA62	20996	(21204)	851	28523
tsyl201	MA42	20462		1002	10355
	MA62	10185	(10404)	469	5355

MA42 were obtained using the option of restricting pivoting to the diagonal. Without this option, the code checks more entries when searching for pivots, but in our experiments, it had no significant effect on the factorization time. MA62 was run with the default settings for all the control parameters. The "Analyze" times quoted for both MA42 and MA62 include the time to order the elements using MC63. For both codes, the analyze times are essentially the same, since only the integer data are used. Direct-access files were used to hold the matrix factors.

For the factorization phase, we see that MA62 is always significantly faster than MA42, and, for the larger problems, MA62 can be more than twice as fast as MA42. The solve times for MA62 are also less than for MA42, with the savings increasing with the number of right-hand sides. This reduction is a result of using a minimum pivot block size of 16 in MA62 and of exploiting zeros in the front. We remark that the HSL Release 12 version of MA42 does not have an option for specifying the minimum pivot block size, although a version of MA42 used by AEA Technology in their code NAMMU [Hartley et al. 1996] for groundwater flow calculations does include this option. The problem cham has a single variable at each finite-element node, and this results in many small pivot blocks. If we use a minimum pivot block size of 16, the real factor storage for MA42 increases

Table IX. A Comparison of MA42 and MA62 on Symmetric Positive-Definite Unassembled Finite-Element Systems. The numbers in parentheses are the times needed if packed form for $\mathbf{F}_{TL}$ is not used.

		Time (seconds)				
				Solve		
Identifier	Code	Analyze	Factorize	nrhs = 1	nrhs = 10	
aea5801	MA42	0.3	1.8	0.14	0.58	
	MA62	0.3	1.1	0.08	0.32	(0.26)
cegb3306	MA42	0.2	0.7	0.07	0.28	
	MA62	0.2	0.4	0.04	0.16	(0.12)
cham	MA42	4.6	28.8	2.62	12.1	
	MA62	4.6	13.1	0.46	1.9	(1.7)
crplat2	MA42	1.0	20.4	0.81	3.3	
	MA62	1.0	11.3	0.49	2.0	(1.8)
lock3491	MA42	0.2	1.5	0.16	0.46	
	MA62	0.2	1.1	0.05	0.23	(0.21)
opt1	MA42	2.2	58.7	0.84	3.4	
	MA62	2.2	31.6	0.70	2.8	(2.6)
ramage02	MA42	1.3	379.5	2.31	9.3	
	MA62	1.3	175.7	1.78	6.8	(6.6)
tsyl201	MA42	0.9	70.0	1.00	3.8	
	MA62	0.9	38.3	0.90	3.7	(3.5)

from $8217 * 10^3$ to $8407 * 10^3$ words, and the operation count increases from $2.77 * 10^9$ to $2.89 * 10^9$. However, the integer storage is cut from $7175 * 10^3$ to $538 * 10^3$ words, while the factorization time reduces from 28.9 to 23.6 seconds, and the solve times for 1 and 10 right-hand sides are reduced to 0.7 and 2.1 seconds, respectively. The current version of MA42 also does not take advantage of zeros in the front, but recent experiments by Scott [1997] have shown that the performance of MA42, particularly in terms of the factor storage and operation count, can be enhanced by exploiting zeros. For example, for the problem lock3491, by exploiting the zeros, the real factor storage for MA42 is cut from $832 * 10^3$ to $705 * 10^3$ words; the integer storage is reduced from $142 * 10^3$ to $121 * 10^3$ words; and the operation count is cut from $106 * 10^8$ to $77 * 10^6$. We anticipate that a version of MA42 which exploits zeros and allows the user to specify the minimum pivot block size will be included in a future release of the Harwell Subroutine Library.

4.2 A Comparison of the Frontal Code MA62 with Other HSL Codes

In this section, we compare the performance of the frontal code MA62 with other codes in the Harwell Subroutine Library that are also designed for solving symmetric positive-definite systems, namely the multifrontal code MA27 and the code VBAN, which is a development version of a new HSL code MA55.

The code MA27 uses the multifrontal algorithm of Duff and Reid [1982]. In the analyze phase, pivots are selected from the diagonal using the minimum-degree criterion. During the factorization, this pivot sequence may be modified to maintain numerical stability, and 2×2 diagonal block pivots can also be used. By this means, MA27 can stably factorize symmetric indefinite problems. However, if the matrix is known to be positive-definite, the user can set a parameter in the calling sequence so that a logically simpler path in the code is followed. In all our tests using MA27, this option was used. MA27 does not exploit BLAS kernels.

Our colleague John Reid at the Rutherford Appleton Laboratory is currently developing a variable-band code for the solution of systems of equations whose matrix is symmetric and positive-definite. It does no interchanges and takes advantage of variation in bandwidth. The code optionally uses a direct-access file to store the matrix factor. The intention is that the new code MA55 will replace an older HSL code MA36. The development code is written in Fortran 90 and uses only Level 1 BLAS, but MA55 will use blocking and Level 3 BLAS. We have called the development code used in our experiments VBAN in the tables and in the following text.

The codes are compared in Table X. The elemental matrices are assembled before MA27 and VBAN are called. The cost of this preprocessing is not included. Since the efficiency of VBAN depends upon the equations being ordered for a small profile, the assembled matrix is ordered using the HSL profile reduction code MC61 [Reid and Scott 1999] prior to calling VBAN, and the time taken to do this is given as the "Analyze" time for VBAN. For MA27, the "Analyze" time is that taken to select the pivot sequence using the minimum-degree criterion and prepare the data structures for the subsequent numerical factorization.

Note that the "In-Core" storage figures quoted in Table X are the minimum main memory requirements for performing the matrix factorization and solving the linear system $\mathbf{Ax} = \mathbf{b}$. We remark that if the minimum memory is used, the performance of the codes will often be considerably degraded, since either a large number of data compressions must be performed or because a large number of records must be written to direct-access files.

Our experiments show that MA62 requires less time for the factorization than VBAN and MA27, although it can need more floating-point operations. As expected, we see that the minimum degree ordering generally performs a better job of reducing the number of entries in the factors than our "band" orderings. Both VBAN and MA62 store their factors in direct-access files and consequently usually require much less in-core storage than MA27. How-

Table X. A Comparison of MA27, VBAN, and MA62 on Symmetric Positive-Definite Finite-Element Systems. Cases marked NS were not solved because of insufficient memory.

		Time (seconds)			Factor ops	Storage in Words ($*10^3$)	
Identifier	Code	Analyze	Factorize	Solve	($*10^6$)	In-Core	Factors
aeac5081	MA27	1.3	3.1	0.08	44	526	430
	VBAN	1.3	3.1	0.08	70	84	565
	MA62	0.3	1.1	0.07	75	28	586
cham	MA27	12.2	81.3	0.29	1916	5895	4983
	VBAN	8.4	31.7	0.42	1407	443	4131
	MA62	4.6	13.1	0.46	1517	193	4472
crplat2	MA27	5.3	40.2	0.30	16248	4554	3815
	VBAN	5.2	44.4	0.64	2321	2309	6319
	MA62	1.0	11.3	0.49	1126	113	4583
mt1	MA27	50.1	581	2.01	14726	36575	37374
	VBAN	NS	NS	NS	NS	NS	NS
	MA62	7.3	1117	11.11	176614	6007	125694
opt1	MA27	11.9	77.1	0.32	3649	7741	5975
	VBAN	11.8	62.5	0.69	3917	3097	7048
	MA62	2.2	31.6	0.70	4374	665	7596
ramage02	MA27	20.4	783.0	1.05	44989	30569	21297
	VBAN	18.8	338.3	2.16	29923	3692	21787
	MA62	1.3	175.9	1.82	28523	2123	21847
shipsec1	MA27	41.0	940	2.64	25653	50186	43480
	VBAN	NS	NS	NS	NS	NS	NS
	MA62	10.0	5450	28.7	863043	14745	333791
srb1	MA27	15.3	116	0.85	2156	14991	11466
	VBAN	15.9	127	1.76	5560	2372	16873
	MA62	3.1	49	1.76	5951	311	18058
thread	MA27	5.8	858	1.22	25890	36082	28978
	VBAN	30.2	976	4.97	115015	53171	54011
	MA62	5.9	680	4.48	113011	8381	55207
trdheim	MA27	10.8	17.7	0.27	211	2893	2002
	VBAN	9.6	17.9	0.39	471	565	2970
	MA62	0.7	4.6	0.32	366	109	2389
tsyl201	MA27	13.6	90.0	0.40	4285	8922	7069
	VBAN	12.2	87.5	1.03	5267	2029	10238
	MA62	0.9	38.3	0.97	5355	382	10654
tubu	MA27	26.9	113.4	0.55	2388	9093	8351
	VBAN	17.8	92.6	1.09	5271	3513	10988
	MA62	8.5	35.6	1.09	4000	477	10742

ever, VBAN sometimes requires a lot more in-core storage than MA62. This happens if there is just a single row of high bandwidth toward the end of the reordered matrix. For the simple variable-band scheme used by VBAN, this requires that many previous rows needed to update this are held in main memory. Problem thread illustrates this. For this problem, the in-core storage is almost as great as the factor storage. The frontal code does not suffer from this problem; the only effect is to add one to the front size for most of the computation. There were two problems, mt1 and shipsec1, that we were unable to solve using VBAN because of insufficient memory.

5. CONCLUSIONS

We have designed and developed a frontal code for solving systems of symmetric positive-definite unassembled finite-element equations. The code optionally uses direct-access files to hold the matrix factors and makes full use of Level 3 BLAS kernels in its innermost loop and in the solution phase. We have shown, that, as well as needing approximately half the real storage for the matrix factors as the general frontal code MA42, the code can be more than twice as fast as MA42. We have seen that the frontal method, with a good element ordering, can provide a very powerful approach to the solution of large sparse systems, compared with other HSL codes, although our conclusions may have to be modified when multifrontal and variable band codes for positive-definite systems that exploit the Level 3 BLAS become available.

6. AVAILABILITY OF THE CODE

MA62 is written in standard Fortran 77. The code, together with the ordering routines MC61 and MC63, is now available and will be included in the next release of the Harwell Subroutine Library. MA27 and MA42 are available in Release 12 of the HSL. Anybody interested in using any of these codes should contact the HSL Manager: Katie North, AEA Technology plc, Culham Science Centre, Abingdon, Oxfordshire OX14 3ED, England, tel. +44 (0) 1235 463797, fax +44 (0) 1235 463649, email hsl@aeat.co.uk, who will provide licencing information. Information on the Library is also available at http://www.cse.clrc.ac.uk/Activity/HSL.

ACKNOWLEDGMENTS

We are grateful to Andrew Cliffe, Christian Damhaug, Ron Fowler, and Alison Ramage for the test problems, and to our colleague John Reid for allowing us to use the code VBAN.

REFERENCES

CLIFFE, K. A., DUFF, I. S., AND SCOTT, J. A. 1998. Performance issues for frontal schemes on a cache-based high performance computer. *Int. J. Num. Methods Eng. 42*, 127–143.

DONGARRA, J. J., DU CROZ, J. J., HAMMARLING, S., AND DUFF, I. S. 1990. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Softw. 16*, 1 (Mar.), 1–17.

DUFF, I. S. 1981. MA32—A package for solving sparse unsymmetric systems using the frontal method. Rep. AERE R10079. Her Majesty's Stationery Office, London, UK.

DUFF, I. S. 1983. Enhancements to the MA32 package for solving sparse unsymmetric equations. Rep. AERE R11009. Her Majesty's Stationery Office, London, UK.

DUFF, I. S. 1984. Design features of a frontal code for solving sparse unsymmetric linear systems out-of-core. *SIAM J. Sci. Comput. 5*, 270–280.

DUFF, I. S. AND REID, J. A. 1982. MA27—A set of Fortran subroutines for solving sparse symmetric sets of linear equations. Rep. AERE R10533. Her Majesty's Stationery Office, London, UK.

DUFF, I. S. AND SCOTT, J. A. 1996. The design of a new frontal code for solving sparse, unsymmetric systems. *ACM Trans. Math. Softw. 22*, 1, 30–45.

DUFF, I. S. AND SCOTT, J. A. 1997. MA62—A new frontal code for sparse positive-definite symmetric systems from finitie-element applications. Tech. Rep. RAL-TR-97-012. Ruthorford Appleton Laboratory, Didcot, Oxon, England.

DUFF, I. S., GRIMES, R. G., AND LEWIS, J. G. 1992. Users' guide for the Harwell-Boeing sparse matrix collection (Release 1). Tech. Rep. RAL-92-086. Rutherford Appleton Lab., Didcot, Oxon, United Kingdom.

DUFF, I., REID, J., AND SCOTT, J. 1989. The use of profile reduction algorithms with a frontal code. *Int. J. Numer. Method. Eng. 28*, 2555–2568.

HARTLEY, L. J., JACKSON, C. P., AND WATSON, S. P. 1996. NAMMU (release 6.3) user guide. Tech. Rep. AEA-ES-0138. Harwell Laboratory, AEA Technology, Didcot, Oxon, United Kingdom.

HOOD, P. 1976. Frontal solution program for unsymmetric matrices. *Int. J. Num. Methods Eng. 10*, 379–400.

HSL. 1996. *Harwell Subroutine Library: A Catalogue of Subroutines (Release 12)*. AEA Technology, Didcot, Oxon, United Kingdom.

IRONS, B. M. 1970. A frontal solution program for finite element analysis. *Int. J. Numer. Method. Eng. 2*, 5–32.

RAMAGE, A. AND WATHEN, A. J. 1993. Iterative solution techniques for the Navier-Stokes equations. Tech. Rep. AM-93-01. School of Mathematics, University of Bristol, Bristol, UK.

REID, J. AND SCOTT, J. 1999. Ordering symmetric sparse matrices for small profile and wavefront. *Int. J. Numer. Method. Eng. 45*, 1737–1755.

SCOTT, J. A. 1997. Exploiting zeros in frontal solvers. Tech. Rep. RAL-TR-98-041. Rutherford Appleton Lab., Didcot, Oxon, United Kingdom.

SCOTT, J. 1999. On ordering elements for a frontal solver. *Commun. Numer. Methods Eng. 15*, 309–323.

Received: September 1998; revised: July 1999; accepted: September 1999