RAL-93-064

MA42 – A new frontal code for solving sparse unsymmetric systems

I. S. Duff and J. A. Scott

ABSTRACT

We describe the design, implementation, and performance of a frontal code for the solution of large sparse unsymmetric systems of linear equations. The resulting software package, MA42, is included in Release 11 of the Harwell Subroutine Library and is intended to supersede the earlier MA32 package. We discuss in detail design changes from the earlier code, indicating the way in which they aid clarity, maintainability, and portability. The new design also permits extensive use of higher level BLAS kernels, which aid both modularity and efficiency. We illustrate the performance of our new code on practical problems on a CRAY Y-MP, an IBM 3090, and an IBM RISC System/6000. We indicate some directions for future development.

Keywords : sparse unsymmetric linear equations, unsymmetric frontal method, Gaussian elimination, finite-element equations, level 3 BLAS.

AMS(MOS) subject classification : 65F05, 65F50.

CR classification system : G.1.3.

Central Computing Department,
Atlas Centre,
Rutherford Appleton Laboratory,
Oxon OX11 0QX.

November 1993.

1 Introduction

We consider the solution of sets of linear equations

$$\mathbf{Ax} = \mathbf{b} \tag{1.1}$$

where the matrix **A** is large and sparse. We do not assume that **A** is symmetric or has any particular structure. This paper describes the design and implementation of a new code for the solution of (1.1) by a direct solution method using a frontal algorithm.

The frontal method (see, for example, Irons (1970), Hood (1976), Duff (1984), Duff, Erisman, and Reid (1986)) is a variant of Gaussian elimination and involves the factorization of a permutation of **A** which can be written as

$$\mathbf{A} = \mathbf{PLUQ}, \tag{1.2}$$

where **P** and **Q** are permutation matrices, and **L** and **U** are lower and upper triangular matrices, respectively. In the frontal method the matrix **A** is normally envisaged as a sum of finite-element matrices

$$\mathbf{A} = \sum_{k=1}^{m} \mathbf{A}^{(k)}, \tag{1.3}$$

where each matrix $\mathbf{A}^{(k)}$ has nonzeros only in a few rows and columns and corresponds to the matrix from element *k*. The basic assembly operation is of the form

$$a_{ij} \leftarrow a_{ij} + a_{ij}^{(k)}. \tag{1.4}$$

The basic Gaussian elimination operation

$$a_{ij} \leftarrow a_{ij} - a_{il}[a_{ll}]^{-1} a_{lj} \tag{1.5}$$

may be performed once each of the terms in the triple product in (1.5) is fully summed (that is, they are involved in no more sums of the form (1.4)). In this way, the factorization can proceed without ever assembling the whole coefficient matrix **A**, and only the active part, the frontal matrix, need be held. The size of the frontal matrix depends upon the ordering of the finite elements so it is important that the elements are suitably ordered (see, for example, Duff, Reid, and Scott (1989)). Duff (1981) extended the frontal method to permit input by equations (rows) as well as elements.

In this paper we are not concerned with the algorithmic details of the frontal method and we refer the reader to the abovementioned papers for this information. Our work is based on a substantial restructuring of an earlier frontal code, MA32 (Duff (1981, 1983)), and indeed was motivated by the requirement to redesign MA32 so that it would run efficiently on a wide range of modern computers and would be in a form suitable for considering further developments in frontal matrix solution, including exploitation of parallelism and the design of codes for complex and symmetric cases.

We present a summary of the main changes we have made and new features we have added in Section 2 and show the overall structure of our new package in Section 3. In Sections 4 to 6, we discuss some of the changes in more detail and illustrate the performance of the new code in Section 7, where we compare it with the earlier code on a CRAY Y-MP, an IBM 3090, and an IBM RS/6000. We consider future work in Section 8 and give a specification sheet for the new code, which is called MA42 in Release 11 of the Harwell Subroutine Library (Anon (1993)), in an appendix.

2 Summary of changes and new features

One of the goals when designing the new code MA42 was the retention of all the functionality of the earlier code MA32 so we first describe briefly characteristics of MA32 that are also present in MA42.

The code MA32 allowed the matrix **A** to be input either by equations or by finite elements. The interface to the user was through reverse communication with control returned to the calling program each time an element or equation needed to be input. The user had to provide the numerical values for each element or equation, together with integer information on the positions of these nonzeros in the assembled matrix. MA32 performed a prepass, on the integer information only, which recorded the element or equation in which each variable appeared for the last time. The assembly of the matrix was then performed one equation or one element at a time and once a variable was fully summed (that is, it had appeared for the last time) it became a candidate for elimination. At any stage during the assembly and elimination the fully or partially summed variables were held in a frontal matrix. The user was required to input the maximum order of this frontal matrix. A judicious choice of assembly order, dependent on the geometry and connectivity of the underlying problem, limited the size of the frontal matrix and thus limited storage requirements and the number of arithmetic operations performed. Static condensation (cases where a variable is internal to an equation or element) was implemented in MA32 so that pivoting and elimination could be performed prior to assembly into the frontal matrix.

A principal feature of MA32 was that it could solve large problems in a predetermined and relatively small amount of main memory. In general, files for the factors **PL** and **UQ** were not held in the main memory. Instead, disk storage was used. During the factorization, data was put into explicitly-held buffers (one for **PL** and one for **UQ**) and, whenever there was insufficient space to accommodate the next column of **PL** or row of **UQ**, the associated buffer was written to a file held on disk. Since the rows of **UQ** were generated in one direction but in the back substitution phase were needed in the reverse order, these files were accessed using direct access. The records in each of the direct access files were of length equal to the associated buffer and this length was chosen by the user. If the user was able to chose the buffers to be very long, direct access files were not necessarily required.

MA32 allowed the user to solve for a specified number of right-hand sides by operating on the right-hand sides (supplied by the user in unassembled form) at the same time as the factorization of the matrix. In addition, MA32 provided a separate entry for the solution of subsequent systems with the same coefficient matrix, and provided two further entries which together could be used to generate the factors of $\mathbf{A}^{\mathrm{T}}$ and solve the transpose equations

$$\mathbf{A}^{\mathrm{T}}\mathbf{x} = \mathbf{b}. \tag{2.1}$$

If the user did not want either to solve for further right-hand sides or to solve transpose systems, MA32 did not store the **PL** factor. The code MA32 also calculated the sign and exponent of the determinant of the matrix.

Although the new code MA42 retains these features and employs many of the internal data structures used in MA32, the structure of the package has been radically altered. We show the constituent subroutines of MA42 and their dependencies in Section 3 and discuss the restructuring in more detail in Section 4. The main reasons for the changes were to increase the modularity and

portability of the code, to make the code more readable and easier to maintain, and to facilitate future developments of the kind discussed in Section 8. Another key reason for the restructuring was to allow greater use of Level 2 and Level 3 Basic Linear Algebra Subprograms (BLAS), both in the inner loops of the factorization and when performing forward and back-substitutions. Expressing the frontal algorithm in terms of the BLAS provides a means of achieving high performance portable Fortran code.

In MA42 we have removed all references to common blocks. We feel this is an aid to portability and will make the design of a version for parallel machines easier. Variables that in MA32 were in common blocks are now held as components of three integer arrays (`ICNTL`, `INFO`, and `ISAVE`) and two real arrays (`CNTL` and `RINFO`). The arrays `CNTL` and `ICNTL` hold variables that in MA32 were set in a BLOCK DATA subprogram. In MA42 these variables are given default values by the initialization routine MA42ID (see Section 3). These arrays control the action of routines in the MA42 package and are termed control arrays. Should the user want the control variables to have values other than the defaults, the appropriate variables should be reset after the call to MA42ID. On successful exit from the final call to the factorization routine MA42BD, the elements of the arrays `INFO` and `RINFO` provide information regarding the execution of the code. This information includes, for example, the amount of real and integer storage used by the factors. The integer array `ISAVE` is used to hold variables which must be preserved between calls to routines in the MA42 package but are unlikely to be of interest to the user. Full details of `ICNTL`, `INFO`, `ISAVE`, `CNTL`, and `RINFO` are given in the specification sheet (see appendix).

The structure of the direct access files used by MA42 has been significantly altered. The reals and integers for the factors are held in separate direct access files. This makes reading to and writing from the direct access files more straightforward and increases the portability of the code. The portability is further enhanced by only writing a buffer to its associated direct access file when either it is completely full or the factorization is complete. This is more efficient since it prevents the fragmentation of data, reducing the number of records used. The new structure also allows systems of the form (2.1) to be solved as soon as the matrix $\mathbf{A}$ has been factorized. In the earlier MA32 code, the storage scheme for the factors did not permit this. Instead, the factors of $\mathbf{A}^{\mathrm{T}}$ had first to be explicitly generated from those of $\mathbf{A}$ and then stored separately. The structure of the direct access files in MA42 is discussed further in Section 6.

Another important change in MA42 is that the basic logical unit is a block of eliminations, rather than the single or double pivot operation used in MA32. Using blocks of eliminations enables Levels 2 and 3 BLAS to be exploited more fully. We examine the use of BLAS in Section 5.

In addition to retaining all the functionality of MA32, MA42 offers the user some new facilities. These include an optional symbolic factorization of the matrix to obtain lower bounds on the maximum frontsizes and estimates of the file sizes required by the factors. This should greatly facilitate the user's job of choosing appropriate sizes for the direct access files and for the associated buffers. In addition, whereas MA32 terminated all computation with an error message when the matrix was detected to be singular, MA42 offers the user the option of continuing the computation. In this case, an estimate of the deficiency of the matrix is returned to the user at the end of the factorization and in the solution vector (or solution matrix for multiple right-hand sides) components corresponding to zero pivots are set to zero.

3 Structure of the MA42 package

The MA42 package consists of 16 subroutines, 6 of which may be called directly by the user. The subroutines are named according to the naming convention of the Harwell Subroutine Library. The single-precision version subroutines all have names that commence with MA42 and have one more letter. The corresponding double-precision versions have the same names with an additional letter D. There is a well defined 5-level hierarchical structure to the package. We give the function of each (double-precision version) subroutine in the following list, where they are grouped according to their level in the hierarchy. The BLAS called by MA42 are also listed.

User called subroutines

MA42ID Initialization of control parameters.

MA42AD Entry of index lists for elements/equations. This subroutine determines when each variable appears for the last time and must be called once for each element/equation.

MA42JD Symbolic factorization of the matrix. This subroutine is optional. If used, it must be called once for each element/equation.

MA42PD Initialization of buffers and direct access files. This subroutine is optional.

MA42BD Entry of elements/equations and, optionally, right-hand sides. This subroutine performs the matrix factorization and, optionally, solves systems of the form (1.1). It must be called once for each element/equation.

MA42CD This subroutine solves for further right-hand sides or solves for systems of the form $\mathbf{A}^{\mathrm{T}}\mathbf{X}=\mathbf{B}$. This subroutine is optional.

Auxiliary subroutines

MA42DD Performs back-substitution on modified right-hand sides.

MA42ED Performs forward-substitution on right-hand sides.

MA42FD Performs the assembly of the elements/equations and factorization of the matrix.

Assembly and factorization subroutines

MA42MD Adds contributions from the incoming element/equation into the frontal matrix and, if right-hand sides are supplied, into the corresponding frontal right-hand side vector.

MA42ND Selects pivots and performs Gaussian elimination operations.

MA42OD Performs static condensation.

Output to buffers subroutines

MA42GD Writes either a block of rows of **UQ** or a block of columns of **PL** to a buffer.

MA42HD Writes the row and column indices of the factors to a buffer.

Subroutines implementing direct access

MA42KD Checks for sufficient space in the direct access files to write out the buffers.

MA42LD Performs direct access read and write operations.

BLAS routines called by MA42

IDAMAX	Level 1 routine. Returns the index of the entry in a vector of maximum modulus.
DAXPY	Level 1 routine. Adds a multiple of one vector to another.
DGER	Level 2 routine. Performs a rank-one update of a matrix.
DTRSV	Level 2 routine. Solves a triangular system of equations.
DGEMV	Level 2 routine. Adds a multiple of a matrix-vector product to another vector.
DTPSV	Level 2 routine. Solves a single triangular system of equations with the matrix held in packed form.
DTRSM	Level 3 routine. Solves a triangular system of equations with multiple right-hand sides.
DGEMM	Level 3 routine. Adds a multiple of a matrix-matrix product to another matrix.

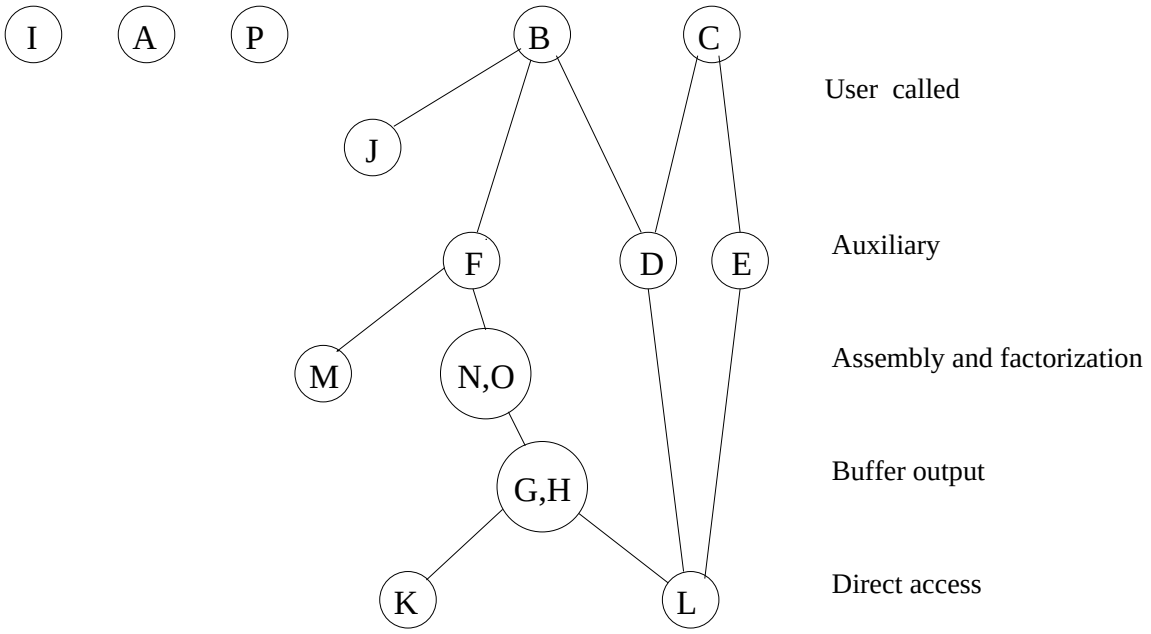

Figure 3.1 Structure of the calls

We illustrate this hierarchy with the call tree in the diagram in Figure 3.1. In this figure, a subroutine at a higher level calls one at the same or a lower level if there is a line joining the two. Subroutines in the MA42 package are identified only by their distinguishing letter, thus D stands for subroutine MA42DD, for example.

4 Code restructuring

Since the release of MA32 in 1981, the user interface remained largely unchanged but the code underwent substantial modifications as new features were added and enhancements were made. Some of these are described in Duff (1983). The changes which were made to the original code, many of which were in response to users' requests, lead to the code MA32 becoming complicated and difficult to maintain or modify. As a result, further changes and developments were almost impossible without substantial restructuring of the code. In this section we consider how we have restructured the code.

After the prepass on the integer information, the user of MA32 had to call the main factorization routine MA32BD once for each element or equation. The subroutine MA32FD was then called by MA32BD and this subroutine performed both the assembly of the elements or equations into the frontal matrix and the matrix factorization. MA32FD first checked the incoming element or equation to see if it could be accommodated within the frontal matrix. If there was not enough room but the number of already fully assembled variables exceeded the number of variables entering the front for the first time, sufficient room was made by forcing eliminations. This involved searching the fully assembled variables for those which came closest to satisfying the threshold stability criterion and then pivoting on these entries. After each forced elimination, the resulting row of **UQ** and column of **PL** were written directly to the buffers.

Having done any necessary forced eliminations, MA32FD performed static condensation on the incoming element or equation. In this phase, variables internal to the incoming element or equation were eliminated within that element or equation. In general, the order of the element or equation is very much smaller than the frontsize so the relative cost of eliminating internal variables is negligible. For the equation entry, the presence of more than one internal variable results in a structurally singular matrix and, if this happened, it was trapped and flagged by MA32 and the computation terminated. Otherwise, pivots were chosen from the internal variables. As each pivot had to satisfy the threshold stability criterion, not all the internal variables were necessarily eliminated. As each pivot was chosen and the elimination performed, the row of **UQ** and column of **PL** generated were again written directly to the buffers. The partially eliminated element or equation was then assembled into the frontal matrix and the fully assembled variables searched for pivots. When a pivot was found, an elimination was performed and the corresponding row of **UQ** and column of **PL** were written to the buffers. Efficiency was improved by performing two steps of Gaussian elimination together whenever possible using a rank-two update of the frontal matrix.

From this brief description we see that the operations performed by MA32FD can be divided into a number of consecutive phases.

(1) Forced eliminations.

(2) Static condensation.

(3) Assembly of partially eliminated element or equation into frontal matrix.

(4) Selection of pivots among fully assembled variables and performance of Gaussian elimination operations.

To improve the modularity and readability of the code, the restructured code MA42FD calls separate subroutines to perform element or equation assembly (MA42MD), pivot selection and

Gaussian elimination operations (MA42ND), and static condensation (MA42OD). Phases (1) and (4) involve pivot searches and eliminations, and in MA42 both phases call MA42ND. Phase 2 corresponds to MA42OD and phase 3 to MA42MD. We remark that in the static condensation phase of MA42 (MA42OD), if entry is by equations and an equation is found to have more than one internal variable, the user has the option of terminating the computation (the action taken by MA32) or of continuing. If the computation continues, static condensation is not performed on equations with more than one internal variable and such equations are assembled directly into the frontal matrix. At some stage zero pivots will be encountered and in the solution vector components corresponding to these zero pivots are set to zero.

In MA32, during phases (1), (2), and (4), whenever a pivot was chosen and used to perform an elimination, the resulting row of **UQ** and column of **PL** were immediately written to the buffers. When restructuring the code we observed that, in phases (1) and (4), it is possible to delay updating the part of the frontal matrix corresponding to variables not chosen as pivots until all pivots for that phase have been chosen. This allows the use of Level 3 BLAS to perform a block of Gaussian elimination operations. We discuss this in greater detail in Section 5. Following the block of Gaussian elimination operations, writing to the buffers is delayed until the end of each of the phases (1), (2), and (4), so that in MA42FD, instead of writing one row and one column at a time to the buffers, a block of rows and a block of columns corresponding to all the eliminations performed within that phase are output together (see (5.2)). The main advantages of writing blocks of rows and columns are that it allows the exploitation of Level 2 and Level 3 BLAS in the solution phase and reduces the amount of integer information which must be stored. The use of BLAS in performing Gaussian elimination operations and in the solution phase is discussed further in the next section.

5 Use of BLAS

A principle objective when restructuring the code was to allow maximum use of Level 2 and Level 3 Basic Linear Algebra Subprograms (BLAS). The BLAS are an aid to clarity, portability, modularity, and maintenance of software. Efficiency is achieved by using tailored implementations of the BLAS. The BLAS are now generally accepted and are widely used in mathematical software. The purpose of the BLAS and their advantages are reviewed by Dongarra, Duff, Sorensen, and van der Vorst (1991). Dongarra *et. al.* also give a complete list of the BLAS, including the operations they perform and their calling sequences.

The BLAS are subdivided into three levels. The Level 1 BLAS, which is the original set of BLAS, perform low-level operations such as dot-products and the adding of a multiple of one vector to another (Lawson, Hanson, Kincaid, and Krogh 1979). The Level 1 BLAS permit efficient implementation on scalar machines, but the ratio of floating-point operations to data movement is too low to achieve effective use of most vector or parallel hardware. Even on scalar machines, the cost of a subroutine call may be prohibitively high when vector lengths are short. To obtain better performance on computers that use vector processing, an additional set of BLAS, the Level 2 BLAS, was designed for a small set of frequently used matrix-vector operations (Dongarra, Du Croz, Hammarling, and Hanson 1988). Most of the common algorithms used in linear algebra can be coded so that the bulk of the computation is performed by calls to the Level 2 BLAS; efficiency can then be

obtained through optimisation within the BLAS. Unfortunately, for machines having a memory hierarchy, the Level 2 BLAS do not have a ratio of floating-point operations to data movement that is high enough to make efficient use of data that reside in cache or local memory. For these architectures, it is often preferable to partition matrices into blocks and to perform the computation using matrix-matrix operations on the blocks. The Level 3 BLAS are targeted at the matrix-matrix operations required for these purposes (Dongarra, Du Croz, Duff, and Hammarling 1990).

We wanted the code MA42 to be readable, portable, and efficient on high-performance computers. Our aim therefore was to maximise the use of Level 3 BLAS. There are two main places in MA42 where we employ Level 3 BLAS. The first is in the innermost loop of the frontal method where the Gaussian elimination operations are performed, and the second is in the solution phase. We describe the use of Level 3 BLAS in each of these phases in more detail in the rest of this section. We remark that, in addition to using Level 2 and Level 3 BLAS, it would have been possible in MA42 to make extensive use of the Level 1 BLAS routines DSWAP, DSCAL, and DCOPY for interchanging two vectors, scaling a vector, and copying one vector to another, respectively. However, from our numerical experiments we found that, while they marginally improved the readability of the code, use of these routines generally lead to an unacceptable increase in the total computation time so they have not been employed.

We first consider the use of BLAS in the Gaussian elimination phase. The innermost loop of a typical frontal method is of the form

```
      DO 20 L = 1,LFRNT
         P1 = PR(L)
         IF (P1.EQ.ZERO) GO TO 20
         DO 10 K = 1,KFRNT
            FA(K,L) = FA(K,L) + PC(K)*P1
10       CONTINUE
20    CONTINUE
```

where FA is the frontal matrix, PC is the pivot column, PR is the pivot row, and KFRNT and LFRNT are, respectively, the number of rows and columns in the front. This code represents a rank-one update to the matrix FA. To achieve greater efficiency, whenever possible the old code MA32 performed two steps of Gaussian elimination together using a rank-two update. This is described by Dave and Duff (1987). The implementation of a rank-two update required more preparatory work since two pivots had to be chosen and tested for stability. The first pivot was tested as usual using a stability threshold criterion. The second pivot was tested after updating its column (and row) using the first pivot. If the second pivot did not satisfy the stability threshold criterion, some work was wasted. In MA32 the Harwell Subroutine Library routines MC32AD and MC31AD were used to perform one and two steps of Gaussian elimination, respectively. For the CRAY-2, CRAY Assembler Language (CAL) versions of these routines were available. The CAL versions were more than twice as fast on the CRAY-2 as the equivalent vectorized Fortran code.

To improve the portability of the code, it would be possible to use the Level 2 BLAS routine DGER to perform one step of Gaussian elimination and the Level 3 routine DGEMM to perform two steps (that is, DGER and DGEMM could be used in place of MC32AD and MC31AD, respectively). However, in MA42 we would like to avoid the possibility of wasting work when looking for a second

pivot and, more importantly, we do not want to restrict ourselves to rank-one and rank-two updates: we would like to be able to delay updating the part of the frontal matrix corresponding to variables not chosen as pivots until all pivots for the current element or equation have been chosen. We now discuss how this can be achieved. After the assembly of an element or equation into the frontal matrix, if all the fully assembled variables are permuted to the first rows and columns, the frontal matrix **FA** has the form

$$\mathbf{FA} = \begin{pmatrix} \mathbf{F}_1 & \mathbf{F}_2 \\ \mathbf{F}_3 & \mathbf{F}_4 \end{pmatrix}, \tag{5.1}$$

where $\mathbf{F}_1$ is a square matrix of order k and $\mathbf{F}_4$ is of order $k_1 \times k_2$, with k_1 equal to k_2 for element entry and to 0 for equation entry. Note that $k + k_1$ is equal to KFRNT (the current number of rows in the front), and $k << k_1$, in general. The rows and columns of $\mathbf{F}_1$, the rows of $\mathbf{F}_2$, and the columns of $\mathbf{F}_3$ are fully summed; the variables in $\mathbf{F}_4$ are not yet fully summed. Pivots may be chosen from anywhere in $\mathbf{F}_1$. The columns of $\mathbf{F}_1$ are searched in order for a pivot and the first entry in $\mathbf{F}_1$ to satisfy the stability threshold criterion is selected as the pivot. The pivotal row and column are permuted to the first row and column of **FA**. Row 1 of the permuted matrix $\mathbf{F}_1$ is then scaled by the pivot and columns 2 to k of the permuted frontal matrix are updated using the Level 1 BLAS routine DAXPY. Columns 2 to k of the updated matrix $\mathbf{F}_1$ are then searched for the next pivot (starting with the first unsearched column and searching cyclically). When found, the pivotal row and column are permuted to row 2 and column 2 of **FA**, row 2 of $\mathbf{F}_1$ is scaled by the pivot, and columns 3 to k of the frontal matrix are updated. This process of selecting pivots and updating the fully summed columns continues until no more pivots satisfying the stability criterion can be found in the fully summed part of **FA**. At this point, if r pivots have been chosen ($r \leq k$), the first r rows of $\mathbf{F}_2$ are updated using the Level 3 BLAS routine DTRSM and, finally, the remaining $k–r$ rows of $\mathbf{F}_2$ and the rows and columns of $\mathbf{F}_4$ are updated using the Level 3 BLAS routine DGEMM (or the LEVEL 2 routine DGER if only one pivot has been chosen). The first r rows and columns of **FA** are then written to the buffers.

When implementing this strategy in MA42, because of the overheads involved in swapping and sorting operations, the fully summed rows and columns are not permuted to the first rows and columns of the frontal matrix before the pivot selection begins. Instead, the fully summed columns are searched for a pivot and, once a pivot has been found, its row and column are permuted to the last row and column of **FA**. When the next pivot is chosen, it is permuted to the last but one row and column of **FA**, and so on. Having chosen all r pivots for the current element or equation, if $r < k$, the remaining $k–r$ fully summed columns are permuted to columns LFRNT-r, LFRNT-r-1,..., LFRNT-k+1 of **FA**. The last r rows of the frontal matrix (columns 1 to LFRNT-k) are updated using DTRSM and finally, the remaining rows and columns are updated using DGEMM (DGER if $r = 1$). Since the pivotal rows and columns are always permuted to the end of the frontal matrix, once they have been written to the buffers, they can be reset to zero before the next element or equation is assembled, without the need for further row and column permutations.

The other main use of Level 2 and Level 3 BLAS in MA42 is in the solution phase. In MA32, each column of **PL** and each row of **UQ** was stored separately and, in the forward and back-substitution phases, one column of **PL** or one row of **UQ** was read in at a time and the Level 1 BLAS routines DDOT and DAXPY were employed. In MA42, the blocks of columns of **PL** and blocks of rows of **UQ** corresponding to variables eliminated at the same stage are of the form

$$\begin{pmatrix}\mathbf{F}_L\\ \mathbf{F}_C\end{pmatrix} \text{ and } (\mathbf{F}_U \;\; \mathbf{F}_R), \tag{5.2}$$

where, if r is the number of rows or columns in the block, $\mathbf{F}_L$, $\mathbf{F}_U$ are $r \times r$ lower and upper triangular matrices respectively, $\mathbf{F}_C$ is of order (KFRNT-r)$\times$r, and $\mathbf{F}_R$ is of order r$\times$(LFRNT-r). To exploit this block structure during the solution phase we use direct addressing. During the forward substitution, all the active components of the partial solution vector **y** (**PLy**=**b**) are put into an array $\mathbf{w}=(\mathbf{w}_1,\mathbf{w}_2)^T$, with $\mathbf{w}_1$ of length r and $\mathbf{w}_2$ of length KFRNT–r. Operations of the form

$$\mathbf{w}_1 \leftarrow \mathbf{F}_L^{-1}\mathbf{w}_1$$

and

$$\mathbf{w}_2 \leftarrow \mathbf{w}_2 - \mathbf{F}_C\mathbf{w}_1$$

are performed before **w** is unloaded into **y**. Similarly, during the back-substitution, all the active components of the partial solution vector **y** are put into an array $\mathbf{z}_1$ of length r and the active variables of the solution vector **x** are put into an array $\mathbf{z}_2$ of length LFRNT–r. Operations of the from

$$\mathbf{z}_1 \leftarrow \mathbf{z}_1 - \mathbf{F}_R\mathbf{z}_2$$

and

$$\mathbf{z}_1 \leftarrow \mathbf{F}_U^{-1}\mathbf{z}_1$$

are performed before $\mathbf{z}_1$ is unloaded into **x**. Similar operations are performed when solving the transpose system. In MA42, the forward and back-substitutions are performed using the Level 2 BLAS routines DGEMV and DTPSV if there is only one right-hand side and the Level 3 routine DGEMM and the Level 2 routine DTPSV if there are multiple right-hand sides (there is no Level 3 BLAS routine for solving a triangular system of equations with the matrix held in packed and multiple right-hand sides). Use of the high level BLAS will be particularly efficacious in cases where the number of variables eliminated at the same stage is large (that is, where r is large) and where the number of right-hand sides is large. Numerical results which demonstrate this are included in Section 7.

6 Structure and size of direct access files

In MA32, two direct access files were used, one for **PL** and one for **UQ**. The reals and integers for each factor were stored in the same direct access file and an integer was stored in a real word (although in an early version of the code for IBM machines, two integers were stored in one real word). As each row of **UQ** or column of **PL** was generated, it was written to an buffer and, if it could not be accommodated in the remaining space in that buffer, the buffer was written to a direct access file. A row or column was not permitted to span buffers. This restriction resulted in some fragmentation of the data, which incurred a storage overhead. On some machines it also limited the size of problem MA32 could solve. For example, on the IBM 3090 under CMS the maximum record length is 32,760.

To make the code MA42 easier to read and to maintain, as well as more portable, we have chosen to use three direct access files, one each for the reals in **PL** and **UQ** and one for the row and column indices of the variables in the factors. In addition, MA42 allows a block of pivotal rows or columns to span more than one buffer and only writes a buffer to the associated direct access file when it is completely full.

Since direct access files may not be needed if the user is able to make the buffers very long, we decided when designing MA42 to make the call to the routine that initializes the direct access data sets (MA42PD) optional. We feel that this simplifies the user interface when MA42 is used to solve problems that are small enough not to require direct access files. The user may also choose not to store the **PL** factor, which is only needed for the solution of further right-hand sides or systems of the form $\mathbf{A}^{\mathrm{T}}\mathbf{x}=\mathbf{b}$.

When writing to the buffers we had to consider the form in which we want the data during the forward and back substitutions. To facilitate the use of DTPSV and DGEMV (or DGEMM) in the solution phase, each record in the real buffers stores the rectangular part of the pivot block ($\mathbf{F}_{\mathbf{C}}$ and $\mathbf{F}_{\mathrm{R}}$ in (5.2)) first, followed by the triangular part ($\mathbf{F}_{\mathbf{U}}$ and $\mathbf{F}_{\mathrm{L}}$ in (5.2)) in packed form and, for the **UQ** buffer, the modified right-hand side vectors. In the integer buffer, the entries in each record are as follows:

(1) The length of the record.
(2) The size of the pivot block.
(3) The current number of rows and columns in the frontal matrix.
(4) Lists of the (global) row and column indices of the variables in the fro
(5) The length of the record.

The length of the record is held as the first and the last entries in the integer record so that the real and integer records can be scanned both in the order they were written and in reverse order. We need to be able to scan both **PL** and **UQ** in reverse order so that the factors of **A** can be used to solve transpose systems, without generating the factors of $\mathbf{A}^{\mathrm{T}}$ explicitly. This greatly simplifies the user interface for the solution of transpose systems of the form (2.1). When using MA32 for such systems, after using MA32BD to factorize the matrix **A**, a subprogram (MA32RD) had to be called to create files holding the factors of $\mathbf{A}^{\mathrm{T}}$ before a further subprogram to solve the transpose system (MA32UD) was called. In MA42, when calling the solution phase (MA42CD) the user has only to set a logical flag to indicate whether systems of the form (1.1) or (2.1) are to be solved.

In MA42, each record in the integer buffer includes the row and column indices for each of the variables in the current front. This differs from the storage used in MA32 where only the indices of the rows added to the front since the previous elimination were stored. More precisely, in MA32 the integer information stored for each column of **PL** was a pointer to the start of the next column, the current frontsize, the row index of the pivot, the change in the frontsize since the last elimination, and a list of the indices of the rows added to the front since the last elimination. Since each variable entered and exited the front once only, the indices for each variable were stored only once and the total number of integers stored for the **PL** factor was 5*n* (*n* is the order of the matrix **A**). For each row of **UQ**, MA32 stored 5 integers (a pointer to the start of the previous row, the row and column indices of the pivot, the current frontsize, and the local column index of the pivot) and so a total of 5*n* integers were also stored for the **UQ** factor. Full details of the integer information stored by MA32 are given by Duff (1981). In MA42, the indices of some variables may be stored more than once but, as MA42 holds blocks of rows and columns, this repetition of indices will be reduced. In practice, we have found the integer storage used is in the range 15–50*n* and, for our test problems, the number of integers stored is less than a quarter the number of reals stored. In Table 7.2 we present more detailed results comparing the storage requirements for MA32 with those of MA42 for some practical problems.

One of the difficulties facing the user of MA32 or MA42 is the need to specify file sizes for the factors and to specify the maximum frontsizes required before the computation begins. For a complicated or unfamiliar problem choosing appropriate sizes may not be easy. For finite-element problems, a lower bound on the maximum frontsize can be obtained by calling the Harwell Subroutine Library routine MC43 (see Duff, Reid, and Scott (1989)) and this can be of great assistance to the user. If the user of MA32 failed to allocate sufficient space to the files for the factors, the code continued with a symbolic factorization only. Provided the symbolic frontsize did not exceed the square of the maximum order allowed by the user for the frontal matrix, after all the elements or equations had been input, MA32 returned information to enable success on a subsequent run with identical data.

In addition to offering such a facility, we felt it would be useful to include an optional subroutine which could be called by the user to perform a symbolic factorization before the file sizes were specified for the numerical factorization. In the MA42 package this routine is called MA42JD. The symbolic factorization assumes that a variable may be eliminated as soon as it is fully summed so, in order that pivots may be chosen to avoid numerical stability, the user should allow file sizes somewhat larger than those returned by MA42JD. In MA42, the routine MA42AD generates an integer array of length n which records, for each variable, the call at which it appears for the last time. The symbolic factorization is then performed using this single integer array and the integer information for each element or equation. As a variable enters the front for the first time, unless the variable occurs only in the incoming element or equation, its entry in the integer array is negated and the symbolic frontsize is incremented by one. When a variable has appeared for the last time, the symbolic frontsize is reduced by one and the sign of the corresponding entry in the integer array is restored. No action is taken for variables which only appear in the incoming element or equation (static condensation candidates). Once all the elements or equations have been input, MA42JD provides the user with lower bounds on the maximum frontsize and estimates of the filesizes required to hold the matrix factors. The use of flags in the symbolic factorization phase of MA42 means that, unlike MA32, its success does not depend upon the user allocating sufficient workspace.

7 Performance of MA42

In this section we report the results of using the new code MA42 on a set of test problems. The test problems all arise from practical applications. A brief description of each problem is given in Table 7.1. Problems 1-8 were taken from the Harwell-Boeing sparse matrix collection (Duff, Grimes, and Lewis (1989,1992)). Problem 9 arises from a Lagrange-Galerkin mixed finite element approximation of the Navier-Stokes and continuity equations on a unit cube (see Ramage and Wathen (1993) for details). Problem 10 comes from a finite element discretisation of a natural convection problem in two dimensions. The element matrices for this problem were generated using the finite element package ENTWIFE (see Winters (1985)).

MA42 has been tested on a Cray Y-MP, an IBM 3090-400E, and an IBM RISC System/6000 Model 550 and its performance has been compared with that of MA32. In each test, the Harwell Subroutine Library routine MC43 was employed to obtain a good element ordering. All CPU timings given in this section are in seconds. The double precision versions of the codes were employed on the IBM 3090 and the IBM RS/6000. On the Cray Y-MP a single processor was used and the codes were run in single

precision. On each of the machines, the implementations of the BLAS provided by the manufacturer were employed.

Table 7.1. The test problems
(CEGB = Central Electricity Generating Board; LOCK = Lockheed Palo Alto Research Laboratory; NV3D = Navier-Stokes in 3D; AEAT = AEA Technology, Harwell.)

Problem	Origin	Description	Number of variables	Number of elements	Elements
1	CEGB	3D model of a turbine blade	2694	108	20-node / 60-freedom bricks
2	CEGB	Framework problem from structural engineering	3222	791	2-node / 12-freedom bars
3	CEGB	3D model of a cylinder with a flange	2859	128	20-node / 60-freedom bricks
4	CEGB	2D cross-section of a reactor core	2996	551	8-node / 16-freedom quadrilaterals
5	LOCK	2D cradle assembly problem	1038	72 158 94	2-node / 6-freedom bars 2-node / 12-freedom bars 3-node / 18-freedom triangles
6	LOCK	2D model of a component used in ocean-mining	691	72 158 94	2-node / 6-freedom bars 2-node / 12-freedom bars 3-node / 18-freedom triangles
7	LOCK	2D model of part a vehicle	3416	72 10 48 556	2-node / 6-freedom bars 2-node / 12-freedom bars 3-node / 18-freedom triangles 4-node / 24-freedom quadrilaterals
8	LOCK	Framework model of a launch umbilical tower	2208	944	2-node / 12-freedom bars
9	NV3D	3D Navier-Stokes and continuity equations	1476	128	20-node / 68-freedom bricks
10	AEAT	Double glazing problem	5081	800	6-noded triangular elements

In Table 7.2 we compare the storage used by MA32 in real words with the real and integer storage used by MA42. The ratio of the real storage to the integer storage for MA42 and the largest number of pivots chosen at a single stage (the largest pivot block size) are also given. From the table we see that, as expected, the real storage requirements for MA42 are less than those for MA32. More interestingly, problems 1, 3, and 9 use a comparatively small amount of integer storage. These problems are 3D problems with 20-node brick elements with either 60 or 68 degrees of freedom. Two adjacent brick elements have a common 2D face so that, if a judicious element ordering is used, as the assembly proceeds several nodes will, in general, appear for the last time at the same assembly step. As a result, for 3D problems of this type with multiple freedoms at the nodes, a large number of pivots may be

Table 7.2. Storage used by MA32 and MA42

Problem	MA32	MA42			
	Real words	Real words	Integer words	Storage ratio	Largest pivot block size
1	508962	481854	33241	14.5	32
2	418302	386814	68766	5.6	11
3	1048140	1019550	71997	14.2	49
4	516592	486736	110713	4.4	14
5	172470	162102	26941	6.0	13
6	98736	91862	18591	4.9	12
7	245232	223188	40140	5.6	12
8	923636	889626	150797	5.9	24
9	778558	763928	67781	11.3	29
10	1492160	1440986	243666	5.9	13

chosen at a significant number of stages of the factorization and this limits the amount of integer storage used.

Table 7.3. Performance of MA32BD and MA42BD on various machines

Problem	Cray Y-MP		IBM 3090		IBM RS/6000	
	MA32BD	MA42BD	MA32BD	MA42BD	MA32BD	MA42BD
1	0.61	0.43	5.41	2.12	2.29	1.82
2	0.57	0.46	3.64	2.06	1.74	1.55
3	1.55	1.07	22.01	5.73	9.22	5.86
4	0.66	0.54	5.63	2.56	2.45	2.20
5	0.23	0.18	1.84	0.92	0.88	0.82
6	0.13	0.11	0.86	0.56	0.44	0.44
7	0.33	0.28	1.76	1.34	0.94	0.91
8	1.18	0.92	12.85	5.34	5.40	4.95
9	1.35	1.09	25.51	6.79	11.86	7.06
10	1.86	1.54	25.67	9.02	11.81	7.95

The performance of the main factorization routines MA32BD and MA42BD are compared for the above mentioned machines in Table 7.3. From the table we see that, for each of the test problems on each of the machines, MA42BD performs better than MA32BD. The gain in using MA42BD in preference to MA32BD is particularly significant for the IBM 3090, where MA42BD may take as little as a quarter of the time of MA32BD. The differences in the improvements between the machines is a consequence of the performance of the BLAS routines DGEMM and DTRSM on the architectures of the different machines.

In Table 7.4 the performances of the solve routines MA32CD and MA42CD are compared. The results show that, for some problems when the number of right-hand sides is small, MA32CD may perform better than MA42CD on the IBM 3090 and IBM RS/6000. However, if MA42 is able to use large pivot blocks (problems 1, 3, and 9), MA42CD gives improvements over MA32CD for any number of right-hand sides. Moreover, as the number of right-hand sides increases, the cost of running

MA42CD increases at a slower rate than the cost of running MA32CD. This results from the use of the BLAS routine DGEMM in MA42CD, which is more efficient for a large number of right-hand sides. We conclude that for problems which allow large pivot blocks or problems with a large number of right-hand sides, significant gains are made by using the MA42 package in place of MA32. For some small problems MA42CD may perform less well than MA32CD but for these problems the computational times are unlikely to be as important so we propose that MA32 should be superseded by MA42 for all applications.

Table 7.4. Performance of MA32CD and MA42CD on various machines

Problem	Number of right-hand sides	Cray Y-MP		IBM 3090		IBM RS/6000	
		MA32CD	MA42CD	MA32CD	MA42CD	MA32CD	MA42CD
1	1	0.04	0.02	0.21	0.18	0.22	0.14
	2	0.06	0.03	0.28	0.23	0.25	0.23
	10	0.26	0.09	0.89	0.62	0.50	0.45
	100	2.50	0.69	9.63	5.14	5.82	3.74
2	1	0.04	0.03	0.18	0.20	0.13	0.23
	2	0.07	0.05	0.25	0.28	0.15	0.21
	10	0.29	0.15	0.76	0.85	0.37	0.66
	100	2.83	1.15	8.45	8.65	5.54	6.75
3	1	0.05	0.03	0.41	0.34	0.36	0.36
	2	0.08	0.05	0.56	0.46	0.43	0.42
	10	0.33	0.15	1.70	1.13	0.99	0.89
	100	3.20	1.17	18.38	8.44	11.31	6.86
4	1	0.04	0.04	0.21	0.25	0.21	0.23
	2	0.07	0.07	0.29	0.35	0.21	0.32
	10	0.28	0.19	0.89	1.23	0.47	0.87
	100	2.73	1.40	9.90	10.80	6.24	9.41
5	1	0.01	0.01	0.07	0.08	0.05	0.08
	2	0.02	0.02	0.10	0.11	0.06	0.12
	10	0.10	0.05	0.18	0.20	0.15	0.20
	100	0.94	0.40	3.20	2.80	2.14	2.42
6	1	0.01	0.01	0.04	0.05	0.03	0.07
	2	0.02	0.01	0.06	0.07	0.05	0.08
	10	0.06	0.04	0.31	0.30	0.08	0.15
	100	0.60	0.28	1.87	1.96	1.21	1.64
7	1	0.03	0.02	0.11	0.13	0.06	0.14
	2	0.05	0.04	0.15	0.18	0.08	0.12
	10	0.19	0.09	0.48	0.53	0.27	0.46
	100	1.85	0.74	4.89	5.30	3.19	3.98
8	1	0.05	0.05	0.37	0.38	0.33	0.29
	2	0.05	0.05	0.50	0.54	0.33	0.42
	10	0.36	0.23	1.58	1.50	0.85	1.18
	100	3.40	1.70	16.76	13.40	10.01	11.18
9	1	0.02	0.02	0.30	0.27	0.29	0.28
	2	0.04	0.04	0.40	0.36	0.32	0.35
	10	0.14	0.11	1.28	0.84	1.06	0.78
	100	0.85	1.35	13.25	6.72	8.75	1.58
10	1	0.06	0.07	0.59	0.62	0.47	0.56
	2	0.10	0.12	0.81	0.87	0.73	0.67
	10	0.40	0.36	2.56	2.20	1.33	1.78
	100	3.77	2.72	27.28	19.27	17.83	17.18

8 Extensions and future work

Since one of the main reasons for designing and coding MA42 was to provide a code which would be relatively easy to develop further, we indicate in this final section our development plans.

Although the code MA32 was widely used for more than a decade to solve real linear systems of equations, there was no comparable code in the Harwell Subroutine Library for solving systems of complex linear equations. One of the obstacles to creating a complex version of MA32 was the storage of reals and integers in the same buffer. As we have already discussed, in MA42 the reals and integers are stored separately. Moreover, MA42 is designed to be both readable and modular, and to exploit BLAS routines. These design features make creating a complex version of the code straightforward. The complex version of the code is called ME42 and can be used to solve the systems $\mathbf{Ax} = \mathbf{b}$, $\mathbf{A}^{\mathrm{T}}\mathbf{x} = \mathbf{b}$, or $\mathbf{A}^{\mathrm{H}}\mathbf{x} = \mathbf{b}$ ($\mathbf{A}^{\mathrm{H}}$ is the conjugate transpose of $\mathbf{A}$). ME42 is also included in Release 11 of the Harwell Subroutine Library.

In MA42 (and ME42) the interface to the user is through reverse communication. This means that control is returned to the user each time an assembly operation is required. The user must regard the coefficient matrix as being of the form (1.3). For finite-element calculations, the matrices $\mathbf{A}^{(k)}$ are the element matrices. For an assembled matrix (non-element problem), $\mathbf{A}^{(k)}$ is nonzero only in row k ($\mathbf{A}^{(k)}$ holds row k of $\mathbf{A}$). The subroutines MA42AD, MA42JD (if used), and MA42BD must be called once for each $\mathbf{A}^{(k)}$, and all the calls to each subroutine must be completed before the next subroutine is called. This interface is convenient for finite-element problems, but for non-element problems it can be too complicated and may deter potential users. To simplify the user interface in this case, we have developed a separate code MA43 (with a complex version ME43) available when direct access files are not required. The user has only to specify the matrix $\mathbf{A}$ once using the standard sparse matrix format (see, for example, Duff, Erisman, and Reid (1986)) and to provide the code with sufficient workspace. A symbolic factorization performed by MA43AD assists the user in determining the amount of workspace that will be required by the factorization and, in the event of failure due to insufficient space being allocated, the user is given a revised estimate of the space needed. The code does all the work in checking the input data for errors and presenting the matrix in the correct form to MA42. In the future we plan to develop a further code which will simplify the user interface in a similar manner both for finite-element problems and for non-element problems when direct access files are needed. We envisage that the user will supply the element matrices or fully assembled matrix once and these will then be (optionally) written to a direct access file and read into the MA42 subroutines as necessary.

In the work discussed previously, we have only been concerned about implementation on uniprocessors or on a single processor of a parallel machine. Through the use of high level BLAS, we have benefited from the vectorization capabilities of the computers, the effect being particularly evident when solving multiple right-hand sides. Many vendors of multiprocessors provide multiprocessed BLAS and it is trivial for us to exploit these in such an environment. We plan to investigate this but realize that the gains from using this as the only avenue for parallel exploitation may give rather limited benefits. One way of developing a more parallel code is to adopt a multifrontal code (Duff and Reid (1983)) which has natural parallelism because of sparsity (Duff (1986)) and can exploit vectorization in a similar way to that described in this paper (Amestoy and Duff (1989)). There

is, however, an intermediate route (for example, Benner, Montry, and Weigland (1987)) which we prefer to call a multiple frontal solver approach, which is similar to using domain decomposition on the overall problem and employing a frontal solver in each subdomain. We will examine this further and believe that the modularity of our redesigned frontal solver will greatly facilitate this task.

We have had some demand from users for a frontal code for symmetric problems and indeed the MA32 code was frequently used when the matrix was symmetric since the speed of the innermost loop sometimes outweighed the penalty of ignoring symmetry. It would have been very difficult to modify MA32 to exploit symmetry but we will shortly be developing such a code based on the MA42 design.

9 Availability of the code

MA42 is written in standard FORTRAN 77. The code is included in Release 11 of the Harwell Subroutine Library and anyone interested in using the code should contact the HSL Manager: Ms L Thick, Theoretical Studies Department, AEA Technology, Building 424.4, Harwell, Oxfordshire, OX11 0RA, England, tel (44) 235 432688, fax (44) 235 436579, or e-mail libby.thick@aea.orgn.uk, who will provide details of price and conditions of use.

10 Acknowledgments

We would like to acknowledge the partial financial support of AEA Technology. We would also to thank Andrew Cliffe of AEA Technology for useful discussions and supplying test problem 10, and Alison Ramage of the University of Strathclyde for supplying test problem 9.

11 References

Amestoy, P. R. and Duff, I. S. (1989). Vectorization of a multiprocessor multifrontal code. *Int. J. Supercomputer Applics.* **3** (3), 41-59.

Anon (1993). Harwell Subroutine Library. A catalogue of subroutines (Release 11).

Benner, R. E., Montry, G. R., and Weigand,G. G. (1987). Concurrent multifrontal methods: shared memory, cache, and frontwidth issues. *Int. J. Supercomputer Applics.* **1**, 26-44.

Dave, A K. and Duff, I. S. (1987). Sparse matrix calculations on the CRAY-2. *Parallel Computing* **5**, 55-64.

Dongarra, J. J., Du Croz, J., Duff, I. S., and Hammarling, S. (1990). A set of level 3 Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.* **16**, 1-17.

Dongarra, J. J., Du Croz, J., Hammarling, S., and Hanson, R. J. (1988). An extended set of Fortran Basic Linear Algebra Subprograms. *ACM Trans. Math. Softw.* **14**, 1-17.

Duff, I. S. (1981). MA32 – A package for solving sparse unsymmetric systems using the frontal method. AERE R10079, HMSO, London.

Duff, I. S. (1983). Enhancements to the MA32 package for solving sparse unsymmetric equations. AERE R11009, HMSO, London.

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

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

Duff, I. S. and Reid, J. K. (1983). The multifrontal solution of indefinite sparse symmetric linear systems. *ACM Trans. Math. Softw.* **9**, 302-325.

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. Softw.* **15**, 1-14.

Duff, I. S., Grimes, R. G., and Lewis, J. G. (1992). Users' guide for the Harwell-Boeing Sparse Matrix Collection (Release 1). Rutherford Appleton Laboratory Report RAL-92-086.

Duff, I. S., Reid, J. K., and Scott, J. A. (1989). The use of profile reduction algorithms with a frontal code. *Int. J. Numer. Meth. Engng.* **28**, 2555-2568.

Hood, P. (1976). Frontal solution program for unsymmetric matrices. *Int. J. Numer. Meth. Engng.* **10**, 379-400.

Irons, B. M. (1970). A frontal solution program for finite-element analysis. *Int. J. Numer. Meth. Engng.* **2**, 5-32.

Lawson, C. L., Hanson, R. J., Kincaid, D. R., and Krogh, F. T. (1979). Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Softw.* **5**, 308-323.

Ramage, A. and Wathen, A. J. (1993) Iterative solution techniques for the Navier-Stokes equations. School of Mathematics, University of Bristol Report AM-93-01

Winters, K. H. (1985) ENTWIFE user manual (release 1). AERE R11577, HMSO, London.

12 APPENDIX: Specification sheets for the MA42 package

Please contact the authors directly for the specification sheets.