The design of 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 the extensive use of higher level BLAS kernels within MA42 and illustrate the performance on a range of practical problems on a CRAY Y-MP, an IBM 3090, and an IBM RISC System/6000. We examine extending the frontal solution scheme to use multiple fronts to allow MA42 to be run in parallel. We indicate some directions for future development.

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

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

CR classification system : G.1.3.

Computing and Information Systems Department,
Atlas Centre,
Rutherford Appleton Laboratory,
Oxon OX11 0QX.

April 1996.

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 further developments in frontal matrix solution, including exploitation of parallelism and the design of codes for complex and symmetric cases. The restructured frontal code is called MA42 in Release 11 of the Harwell Subroutine Library (Anon 1993) and full details of the code, including specification sheets, may be found in Duff and Scott (1993).

The MA42 code may be split into three distinct phases.

(i) An analyse phase (MA42A) which determines when each variable is fully summed.

(ii) A factorization phase (MA42B) which factorizes the matrix into its upper and lower triangular factors using Gaussian elimination. If right-hand side vectors (in unassembled form) are included in the calls to MA42B, forward substitution on these right-hand sides is performed at the same time as the matrix factorization and, once the factorization is complete, MA42B calls an internal subroutine (MA42D) to perform the back-substitution.

(iii) A solution phase for further right-hand sides and for the solution of transpose systems

$\mathbf{A}^T\mathbf{x} = \mathbf{b}$ (MA42C). This phase is optional. MA42C calls an internal subroutine MA42E to perform the forward substitution on the right-hand sides and then calls the internal subroutine MA42D to perform the back-substitution and complete the solution.

Key features of MA42 are:

- The matrix **A** may be input either by finite elements or by equations.

- The interface to the user is through reverse communication with control returned to the calling program each time an element or equation needs to be input.

- Efficiency of the code (in terms of storage and computation time) is dependent upon the order in which the elements or equations are input.

- Large problems can be solved in a predetermined and relatively small amount of main memory. In general, files for the factors **PL** and **UQ** are not held in the main memory. Instead, during the factorization, data is put into explicitly held buffers and, whenever a buffer is full, it is written to a direct access file held on disk.

- The length of the records in each of the direct access files (which is equal to length of the associated in-core buffer) is chosen by the user.

- If the user is able to choose very long buffers, direct access files are not necessarily required so in MA42 the call to the routine that initializes the direct access files is optional.

- The reals and integers for the factors are held in separate direct access files.

- An optional symbolic factorization of the matrix computes lower bounds on the maximum size of the frontal matrix and estimates of the sizes of the files for holding the factors.

- MA42 switches to a symbolic factorization if the user fails to allocate sufficient space to the files for the factors. Information to enable success on a subsequent run with identical data is returned.

- If the user does not want either to solve for further right-hand sides or to solve transpose systems, MA42 need not store the **PL** factor.

- The sign and exponent of the determinant of the matrix **A** is computed.

- Level 2 and Level 3 Basic Linear Algebra Subprograms (BLAS) are exploited both in the inner loops of the factorization and when performing forward and back-substitutions (see Section 2).

- The code is portable and written in standard Fortran 77.

- Common blocks are not used and are replaced by information and control arrays. Default values for the control variables are set by a call to the initialization subroutine MA42I.

- An option to continue the computation if the matrix is found to be singular is offered. 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.

The remainder of this paper is organised as follows. In Section 2 we consider the restructuring of the code and, in particular, the exploitation of high level BLAS. In Section 3 we illustrate the performance of MA42 and compare it with the earlier code MA32 on a CRAY Y-MP, an IBM 3090, and an IBM RS/6000. We consider the use of MA42 in a parallel environment in Section 4 and finally indicate possible future work in Section 5.

2 Code restructuring and the use of BLAS

The Basic Linear Algebra Subprograms (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). They also give a complete list of the BLAS, including the operations they perform and their calling sequences.

The BLAS are divided 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).

When designing our new frontal code a prinicpal objective was to enable maximum use of Level 2 and Level 3 BLAS. There are two main places in MA42 where we exploit 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 following subsections. 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 _SWAP, _SCAL, and _COPY 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 used.

2.1 The use of BLAS in frontal Gaussian elimination

In a typical frontal method the innermost loop is of the form

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

where `F` is the frontal matrix, `PC` is the pivotal column, `PR` is the scaled pivotal row, and `KFRNT` and `LFRNT` are, respectively, the number of rows and columns in the front. This code performs a rank-one update to the matrix `F`. 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), who showed that optimal performance could be obtained on the CRAY-2 using this device with CAL coding of the kernels. 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.

To improve the portability of the code, it would be possible to use the Level 2 BLAS routine _GER to perform one step of Gaussian elimination and the Level 3 routine _GEMM to perform two steps. However, by restructuring the code we would like to avoid the possibility of wasting work when looking for a second pivot and, more importantly, we would like to avoid the limitation of only allowing 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 **F** has the form

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

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). 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 **F**. 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 _AXPY. 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 **F**, 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 **F**. 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 _TRSM 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 _GEMM (or the LEVEL 2 routine _GER if only one pivot has been chosen). The first r rows and columns of **F** 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 **F**. When the next pivot is chosen, it is permuted to the last but one row and column of **F**, 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 **F**. The last r rows of the frontal matrix (columns `1` to `LFRNT-k`) are updated using _TRSM and finally, the remaining rows and columns are updated using _GEMM (_GER 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 performance achieved by MA42 through the use of Level 3 BLAS in the elimination phase can be impressive. We illustrate this using a model problem which, although artificial, is designed to simulate those actually occurring in some CFD calculations. The elements are

nine-node rectangular elements with nodes at the corners, mid-points of the sides, and centre of the element. A parameter to the element generation routine determines the number of variables per node. We have chosen this parameter to be five for our numerical experiments. The elements are arranged in a rectangular grid and a range of grid dimensions is used. In Table 2.1 we illustrate the performance of MA42 on a single processor of a CRAY Y-MP8I vector supercomputer, whose peak performance is 333 Mflop/s and on which the Level 3 BLAS matrix-matrix multiply routine SGEMM runs at 313 Mflop/s on sufficiently large matrices. It is important to realize that the Mflop rates given in Table 2.1 include all overheads for the out-of-core working used by MA42.

Table 2.1. Performance (in Mflop/s) of MA42 on CRAY Y-MP on test problem.

Dimension of element grid	16×16	32×32	48×48	64×64	96×96
Max order frontal matrix	195	355	515	675	995
Total order of problem	5445	21125	47045	83205	186245
Mflop/s	145	208	242	256	272

2.2 The solution phase and the use of BLAS

The other 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 _DOT and _AXPY were used. 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

$$\begin{pmatrix}\mathbf{F}_{\mathrm{L}}\\ \mathbf{F}_{\mathbf{C}}\end{pmatrix} \quad \text{and} \quad \begin{pmatrix}\mathbf{F}_{\mathbf{U}} & \mathbf{F}_{\mathrm{R}}\end{pmatrix},$$

where, if r is the number of rows or columns in the block, $\mathbf{F}_{\mathrm{L}}$, $\mathbf{F}_{\mathbf{U}}$ are $\mathtt{r}\times\mathtt{r}$ lower and upper triangular matrices respectively, $\mathbf{F}_{\mathbf{C}}$ is of order $(\mathtt{KFRNT-r})\times\mathtt{r}$, and $\mathbf{F}_{\mathrm{R}}$ is of order $\mathtt{r}\times(\mathtt{LFRNT-r})$. 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 $\mathbf{y}$ ($\mathbf{PLy}=\mathbf{b}$) are put into an array $\mathbf{w}=(\mathbf{w}_1,\mathbf{w}_2)^{\mathrm{T}}$, with $\mathbf{w}_1$ of length r and $\mathbf{w}_2$ of length KFRNT–r. Operations of the form

$$\mathbf{w}_1 \leftarrow \mathbf{F}_{\mathrm{L}}^{-1}\mathbf{w}_1$$

and

$$\mathbf{w}_2 \leftarrow \mathbf{w}_2 - \mathbf{F}_{\mathbf{C}}\mathbf{w}_1$$

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

$$\mathbf{z}_1 \leftarrow \mathbf{z}_1 - \mathbf{F}_{\mathrm{R}}\mathbf{z}_2$$

and

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

are performed before $\mathbf{z}_1$ is unloaded into $\mathbf{x}$. Similar operations are performed when solving the transpose system. In MA42, the forward and back-substitutions are performed using the Level 2 BLAS routines _GEMV and _TPSV if there is only one right-hand side and the Level 3 routine _GEMM and the Level 2 routine _TPSV if there are multiple right-hand sides (there

is no Level 3 BLAS routine for solving triangular systems of equations where the matrix is held in packed form with 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 the next section.

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. Explicit row and column indices were not held but dynamic changes to the front were stored. One of the disadvantages of this storage scheme was that, if the user of MA32 wished to solve transpose systems, a separate subprogram had to be called after **A** had been factorized to create files holding the factors of $\mathbf{A}^T$ and then a further subprogram was called to complete the solution. To make the code MA42 easier to read and to maintain, as well as more portable, when designing the code we chose 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. This use of three direct access files greatly simplifies the user interface for solving transpose systems. In MA42, when calling the solution phase (MA42C) the user has only to set a logical flag to indicate whether linear systems with the system matrix **A** or $\mathbf{A}^T$ are to be solved.

3 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 3.1. 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-MP8I, 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 used to obtain a good element ordering. All CPU timings given in this section are in seconds. The double precision versions of the codes were used 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 used.

In Table 3.2 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 chosen at a significant number of stages of the factorization and this limits the amount of integer storage used.

The performance times of the factorization routines MA32B and MA42B are compared for the above mentioned machines in Table 3.3. From the table we see that, for each of the test problems on each of the machines, MA42B performs better than MA32B. The gain in using MA42B in preference to MA32B is particularly significant for the IBM 3090, where MA42B may sometimes take nearly a quarter of the time of MA32B. The differences in the

Table 3.1. 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

improvements between the machines is a consequence of the performance of the BLAS routines _GEMM and _TRSM on the architectures of the different machines.

In Table 3.4 the performance times of the solve routines MA32C and MA42C are compared. The results show that, for some problems when the number of right-hand sides is small, MA32C may perform better than MA42C on the IBM 3090 and IBM RS/6000. However, if MA42 is able to use large pivot blocks (problems 1, 3, and 9), MA42C gives improvements over MA32C for any number of right-hand sides. Moreover, as the number of right-hand sides increases, the cost of running MA42C increases at a slower rate than the cost of running MA32C. This results from the use of the BLAS routine _GEMM in MA42C, 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, MA42C may perform less well than MA32C but for these problems the computational times are

Table 3.2. 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

Table 3.3. Performance times (in seconds) of MA32B and MA42B on various machines

Problem	Cray Y-MP		IBM 3090		IBM RS/6000	
	MA32B	MA42B	MA32B	MA42B	MA32B	MA42B
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

unlikely to be as important so we propose that MA32 should be superseded by MA42 for all applications.

4 The use of MA42 in parallel

One of the main deficiences of the frontal solution scheme is that there is little scope for parallelism other than that which can be obtained within the high level BLAS. One way of attempting to overcome this is to extend the basic frontal algorithm to use multiple fronts. The use of multiple fronts depends upon being able to decouple the underlying ‘domain’ and eliminate variables within each subdomain independently. The logical conclusion of this approach is the so-called multifrontal algorithm (see, for example, Duff and Reid 1983), where many fronts are started simultaneously. The variables that are not immediately eliminated are combined to form new elements or fronts which are then merged with other new elements or original elements according to an assembly tree. The initial fronts and the assembly tree are identified by a flop reducing ordering such as minimum degree or nested dissection. We will adopt the term multiple fronts for algorithms for solving systems of finite-element equations which partition the finite-element domain into a (small) number of

Table 3.4. Performance times (in seconds) of MA32C and MA42C on various machines

Problem	Number of right-hand sides	Cray Y-MP MA32C	Cray Y-MP MA42C	IBM 3090 MA32C	IBM 3090 MA42C	IBM RS/6000 MA32CD	IBM RS/6000 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

subdomains and perform a frontal decomposition on each subdomain using an element-by-element ordering in a somewhat similar fashion to Benner, Montry, and Weigland (1987) and Zhang and Lui (1991).

With multiple fronts, we partition the underlying finite-element domain into subdomains and perform a frontal decomposition on each subdomain separately. Since the factorizations of the subproblems are independent, this can be done in parallel. At the end of the assembly and elimination processes for the subdomains, there will remain m variables, where m is the number of variables on the interface boundaries of the subdomains. These variables are called

interface variables. They cannot be eliminated within the subdomains since they are shared by more than one subdomain. In practice, there will also remain variables which were not eliminated within the subdomain because of stability considerations. For each subdomain a relationship of the form

$$\mathbf{F}_i \mathbf{y}_i = \mathbf{c}_i \tag{4.1}$$

is thus formed, where $\mathbf{F}_i$ is the remaining frontal matrix for subdomain i after the final assembly, and $\mathbf{c}_i$ is the corresponding frontal right-hand side vector (or matrix). If there are *nsub* subdomains, *nsub* equations of the type (4.1) are generated and these can be assembled to give

$$\mathbf{F}\mathbf{y} = \mathbf{c}, \tag{4.2}$$

where $\mathbf{F}$ is of order $m \times m$. The system (4.2) may also be solved using a frontal solver. Once (4.2) has been solved, the results for the interface variables must be passed to the subdomains so that back-substitution can be performed. The back-substitutions on the subdomains may also be performed in parallel. Some experiments using this approach have been reported by Zhang and Lui (1991) for structural finite-element problems on an Alliant FX/80.

We would like to use MA42 to implement the multiple front algorithm. If MA42 is applied to a subdomain, the elimination of interface variables must be prevented during the factorization phase (MA42B). To do this, during the prepass phase (MA42A) the interface variables must be marked as being not fully summed after the entry of all the elements in the subdomain. This can be done quite simply by making a final call to MA42A with an extra element containing all the interface variables. The extra element will be termed the 'guard element' and there will be one guard element associated with each subdomain. A new routine MA52A has been developed to generate guard elements. After the extra call to MA42A, MA42B is called for each element in the subdomain but not for the guard element. Since MA42B is not called for the guard element, variables in the guard element (that is, the interface variables) will remain in the front after the assembly and eliminations for the final element in the subdomain. The frontal matrix and corresponding frontal right-hand side vectors are held within the work arrays used by MA42B. A new subroutine was needed to extract the frontal matrix $\mathbf{F}_i$ and frontal right-hand sides $\mathbf{c}_i$ from these arrays and return them to the user in the form of an element matrix and element right-hand side. This new subroutine also had to be able to write any remaining data in the in-core buffers to the files holding the factors. We have written MA52B to perform these tasks.

The interface problem (4.2) can be solved using MA42. Once this is done, a routine to take the results for the interface variables and perform back substitution on the subdomain is needed. When designing the MA42 code, we decided that the routines to perform forward and back-substitution, namely MA42D and MA42E, should be internal subroutines. This simplifies the user interface since it reduces the number of subroutine calls the user has to make. However, for the multiple front algorithm we want to call forward or back-substitution directly. The routine we have written to do this is MA52C. This routine provides an interface to MA42D and MA42E. A user-supplied parameter determines whether forward or back-substitution is performed and, for back-substitution, whether it follows a call to MA42B or to MA42C for the interface problem. MA52C must be called for each subdomain but the calls may be made in parallel.

Some preliminary experiments have been performed using MA42 and MA52 to implement the multiple front algorithm and these are reported by Duff and Scott (1994a, 1994b). In Table 4.1 we illustrate the performance of MA42 for our model problem (see Section 2.1) in two parallel environments: on an eight processor shared-memory CRAY Y-MP8I and on a network of five DEC Alpha workstations using PVM (Geist, Beguelin, Dongarra, Jiang, Manchek, and Sunderam 1993). In each case we divide the original problem into a number of subdomains equal to the number of processors being used. It is difficult to get meaningful times in either environment because we cannot guarantee to have a dedicated machine. The

Table 4.1. Multiprocessor performance of MA42 on CRAY Y-MP and DEC Alpha farm on model problem.

Dimension of element grid	Order of problem	Number of subdomains	CRAY Y-MP Wall clock time (secs)	CRAY Y-MP Speedup	DEC Alpha farm Wall clock time (secs)	DEC Alpha farm Speedup
16 x 16	5445	1	1.88	–	23.2	–
		2	1.12	1.7	29.4	0.8
		4	0.77	2.4	29.9	0.8
		8	0.92	2.0	–	–
32 x 32	21125	1	16.67	–	350.6	–
		2	9.95	1.7	250.8	1.4
		4	4.17	4.0	110.5	3.2
		8	3.97	4.2	–	–
48 x 48	47045	1	98.75	–	1460.3	–
		2	64.57	1.5	1043.0	1.4
		4	30.65	3.2	457.5	3.2
		8	15.25	6.5	–	–

times are, however, for lightly loaded machines.

The results on the CRAY are encouraging and show good speedup for large problem sizes. In general, the efficiency of the multiple front algorithm increases with the size of the problem. For small problems, the amount of computation on each subdomain is small and the ratio of internal variables to interface variables is also small. As a result, the communication time dominates the computation time and only a relatively small amount of work is actually performed in parallel. Results on the Alpha farm are comparable with those on the CRAY and indicate that, for larger problems, the overheads of PVM and communication costs do not dominate and good speedups are possible. We should add that it is important that disk i/o is local to each processor. The default on our Alpha system was to write all files centrally and this increased data traffic considerably, gave poor performance, and varied greatly depending on system loading.

In the future we plan to extend this work on using MA42 to implement a multiple frontal algorithm. In particular, we intend to perform experiments using real applications, and we will look at the issues of loading balancing and obtaining efficient element orderings within the subdomains.

5 Further extensions to our frontal codes.

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 mentioned, the reals and integers are stored separately in MA42. 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 $\mathbf{Ax} = \mathbf{b}$, $\mathbf{A}^T\mathbf{x} = \mathbf{b}$, or $\mathbf{A}^H\mathbf{x} = \mathbf{b}$ ($\mathbf{A}^H$ 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 reverse communication 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 MA43A 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.

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.

6 Availability of the code

MA42, MA43, and their complex versions ME42, ME43 are included in Release 11 of the Harwell Subroutine Library (Anon 1993). We plan to include MA52 in Release 12 of HSL. Anyone interested in using HSL routines should contact the HSL Manager: Ms L Thick, Harwell Subroutine Library, AEA Technology, Building 8.19, Harwell, Oxfordshire, OX11 0RA, England, tel (44) 235 432688, fax (44) 235 432989, or e-mail libby.thick@aea.orgn.uk, who will provide details of price and conditions of use.

7 Acknowledgments

We would like to acknowledge the partial financial support of AEA Technology. We would also to thank Andrew Cliffe of AEA Technology for supplying test problem 10 and Alison Ramage of the University of Strathclyde for supplying test problem 9. We are grateful to our colleagues at the Atlas Centre for assistance with running our codes in a multiprocessor environment and to Andrew Cliffe and to Hon Yau and Mark Sawyer of the Edinburgh Parallel Computing Centre for helpful discussions on the multiple frontal algorithm.

8 References

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.

Dongarra, J. J., Duff, I. S., Sorensen, D. C. and van der Vorst, H. A. (1991). Solving linear systems on vector and shared memory computers. SIAM Press.

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

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.

Duff, I. S., and Scott, J. A. (1993). MA42 – A new frontal code for solving sparse unsymmetric systems. Rutherford Appleton Laboratory Report RAL-93-064.

Duff, I. S., and Scott, J. A. (1994a). The use of multiple fronts in Gaussian elimination. Proceedings of the Fifth SIAM Conference on Applied Linear Algebra, editor J. Lewis, SIAM Press, 567-571.

Duff, I. S., and Scott, J. A. (1994b). The use of multiple fronts in Gaussian elimination. Rutherford Appleton Laboratory Report RAL-94-040.

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V. (1993). PVM 3 User's Guide and Reference Manual. Technical Report ORNL/TM-12187, Engineering Physics and Mathematics Division, Oak Ridge National Laboratory, Tennessee.

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.

Zhang, W. P. and Lui, E. M. (1991). A parallel frontal solver on the Alliant FX/80. *Computers and Structures* **38**, 203-215.