
**Technical Report**
RAL-TR-97-041

# Exploiting Zeros in Frontal Solvers

**J A Scott**

August 1997

COUNCIL FOR THE CENTRAL LABORATORY OF THE RESEARCH COUNCILS

Enquiries about copyright, reproduction and requests for additional copies of this report should be addressed to:

The Central Laboratory of the Research Councils
Library and Information Services
Rutherford Appleton Laboratory
Chilton
Didcot
Oxfordshire
OX11 0QX
Tel: 01235 445384 Fax: 01235 446403
E-mail library@rl.ac.uk

**ISSN 1358-6254**

RAL-TR-97-041

# Exploiting zeros in frontal solvers.*

by

Jennifer A. Scott

**Abstract**

An important feature of the frontal method for the solution of large sparse systems of linear equations is that the frontal matrix is held as a dense matrix. This allows efficient dense linear algebra kernels, in particular, the Level 3 Basic Linear Algebra Subprograms (BLAS) to be used during the numerical factorization. However, the frontal matrix may contain a significant number of zeros and this, in turn, leads to unnecessary operations being performed and to zeros being stored explicitly in the factors. In this paper we look at how we can take advantage of zeros within the frontal matrix. We illustrate the effects of exploiting zeros in the front on the factorization and solve times, and on the storage requirements for the Harwell Subroutine Library general frontal code `MA42` using a range of problems arising from real engineering and industrial applications.

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

**AMS(MSC 1991) subject classifications:** 65F05, 65F50.

**CR classification system:** G.1.3.

---

* Current reports available by anonymous ftp from `matisa.cc.rl.ac.uk` (internet 130.246.8.22) in the directory `pub/reports`.

Department for Computation and Information,
Atlas Centre, Rutherford Appleton Laboratory,
Oxon OX11 0QX, England.

August 13, 1997.

# Contents

# 1 Introduction

In this report, we are interested in using the frontal method to solve systems of linear equations

$$\mathbf{AX} = \mathbf{B}, \tag{1.1}$$

where the $n \times n$ matrix $\mathbf{A}$ is large and sparse. $\mathbf{B}$ is an $n \times nrhs$ ($nrhs \geq 1$) matrix of right-hand sides and $\mathbf{X}$ is the $n \times nrhs$ solution matrix. The frontal method was originally developed by Irons (1970) for the solution of symmetric positive-definite systems which come from finite-element discretizations in structural analysis. The method was extended to unsymmetric finite-element problems by Hood (1976). For finite-element problems, the system matrix $\mathbf{A}$ is normally envisaged as a sum of finite-element matrices

$$\mathbf{A} = \sum_{l=1}^{m} \mathbf{A}^{(l)}, \tag{1.2}$$

where each matrix $\mathbf{A}^{(l)}$ has nonzeros only in a few rows and columns and corresponds to the matrix from element $l$. The frontal method interleaves an assembly phase and an elimination phase, thereby allowing operations to be confined to a relatively small matrix, termed the *frontal matrix*, that can be regarded as dense. Duff (1983) realised that the method could be extended to general unsymmetric matrices by assembling the matrix $\mathbf{A}$ one row (equation) at a time. In this case, $\mathbf{A}^{(l)}$ corresponds to row $l$ of $\mathbf{A}$. This strategy was first implemented in the early 1980s in the Harwell Subroutine Library (1996) code `MA32`. `MA32` was very widely used before being substantially restructured and improved by Duff and Scott (1993, 1996*b*). The upgraded code, `MA42`, which has superseded `MA32` in the Harwell Subroutine Library (HSL), has been used on a wide range of modern computers to solve problems from many different application areas. It allows input by elements or equations but is principally structured for finite-element problems.

Recent experiments by Duff and Scott (1996*a*) found that, for some problems, the frontal code `MA42` can require substantially more operations and storage for the factors than other HSL codes for solving large sparse systems. In particular, the equation entry to `MA42` was found to be inefficient if the equations were poorly ordered. Closer examination revealed that, in this case, a large number of zeros are held explicitly within the factors. Our aim is to reduce the number of zeros within the factors, while not compromising the efficiency of the code when applied to well-ordered problems. Reducing the zeros will reduce the operation count and the storage required for the matrix factors as well as the time needed for using the factors to solve for different right-hand sides $\mathbf{B}$.

The outline of this report is as follows. In Section 2, we briefly review frontal schemes. We then look, in Section 3, at how we can exploit zeros within the front. The efficient treatment of static condensation variables is also discussed. In Section 4, we illustrate the advantages of exploiting zeros in frontal solvers. Concluding comments are made in Section 5.

# 2 Frontal schemes

The frontal method is a variant of Gaussian elimination and involves the matrix factorization

$$\mathbf{A} = \mathbf{PLUQ}, \tag{2.1}$$

where **P** and **Q** are permutation matrices, and **L** and **U** are lower and upper triangular matrices, respectively. The solution process is completed by performing the forward elimination

$$(\mathbf{PL})\mathbf{Y} = \mathbf{B}, \tag{2.2}$$

followed by the back substitution

$$(\mathbf{UQ})\mathbf{X} = \mathbf{Y}. \tag{2.3}$$

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

$$a_{ij} \Leftarrow a_{ij} + a_{ij}^{(l)}, \tag{2.4}$$

where $a_{ij}^{(l)}$ is the $(i,j)$th nonzero entry of the element matrix $\mathbf{A}^{(l)}$. A variable is *fully summed* if it is involved in no further sums of the form (2.4) and is *partially summed* if it has appeared in at least one of the elements assembled so far but is not yet fully summed. The Gaussian elimination operation

$$a_{ij} \Leftarrow a_{ij} - a_{il}[a_{ll}]^{-1}a_{lj} \tag{2.5}$$

may be performed as soon as all the terms in the triple product in (2.5) are fully summed.

Since variables can only be eliminated after they are fully summed, the assembly order will determine, to a large extent, the order in which the variables are eliminated. At any stage during the assembly and elimination processes, the fully and partially summed variables are held in an in-core frontal matrix. Suppose that entry is by elements. If after the assembly of an element, there are $k$ fully summed variables and these are permuted to the first rows and columns of the frontal matrix, we can partition the frontal matrix **F** as

$$\mathbf{F} = \begin{pmatrix} \mathbf{F}_T & \mathbf{F}_R \\ \mathbf{F}_C & \mathbf{F}_U \end{pmatrix}, \tag{2.6}$$

where $\mathbf{F}_T$ is a square matrix of order $k$. The rows and columns of $\mathbf{F}_T$, the rows of $\mathbf{F}_R$, and the columns of $\mathbf{F}_C$ are fully summed; the variables in $\mathbf{F}_U$ are not yet fully summed. Pivots may be chosen from anywhere in $\mathbf{F}_T$. For symmetric positive-definite systems, they can be taken from the diagonal in order but in the general case, off-diagonal pivots may be chosen to satisfy a threshold criteria. This is discussed in Duff (1984). The pivot row and column are permuted to the first row and column of (2.6), row 1 of $\mathbf{F}_T$ is scaled by the pivot and columns 2 to $k$ of the permuted frontal matrix updated. Columns 2 to $k$ of the updated matrix $\mathbf{F}_T$ are then searched for the next pivot. When chosen, the pivot row and column are permuted to row 2 and column 2 of the frontal matrix, row 2 of $\mathbf{F}_T$ is scaled by the pivot, and columns 3 to $k$ of the frontal matrix are updated. This process continues until no more pivots can be found. Assuming $k$ ($k > 1$) pivots have been chosen, $\mathbf{F}_R$ is updated using the Level 3 BLAS routine _TRSM

$$\mathbf{F}_R \leftarrow -\mathbf{F}_T^{-1}\mathbf{F}_R \tag{2.7}$$

and, finally, $\mathbf{F}_U$ is updated using the Level 3 BLAS routine _GEMM

$$\mathbf{F}_U \leftarrow \mathbf{F}_U + \mathbf{F}_C\mathbf{F}_R. \tag{2.8}$$

In practice, stability restrictions may only allow $r$ pivots to be chosen ($r < k$) and, in this case, the first $r$ rows of $\mathbf{F}_R$ are updated using _TRSM and then $\mathbf{F}_U$ and the remaining $k-r$ rows of $\mathbf{F}_R$ are updated using _GEMM, with interior dimension $r$. If a single pivot is chosen, Level 2 BLAS routines are used. Further details of how this strategy is implemented within the frontal code MA42 are given by Duff and Scott (1996*b*).

If the assembly is by equations, rather than by elements, an equation is fully summed as soon as it enters the front so that an elimination can be performed once a column becomes fully summed; that is, whenever the equation containing the last occurrence of a variable is assembled. For equation entry, the frontal matrix will thus be rectangular and of the form (2.6) with the rows and columns of $\mathbf{F}_T$ **and** $\mathbf{F}_C$ fully summed, and the rows of $\mathbf{F}_R$ and $\mathbf{F}_U$ fully summed but the columns of $\mathbf{F}_U$ not yet fully summed. Since the rows of $\mathbf{F}_C$ are fully summed, for each fully summed column, pivots can be chosen from anywhere in the column. Therefore the largest entry in the column may be used as the pivot (partial pivoting) and (provided the matrix is non singular) no pivots are delayed by stability considerations.

A key feature of the frontal method is that the matrix factors need not be held in-core. This allows large problems to be solved using only modest amounts of high-speed memory. The minimum in-core memory requirement is dependent on the maximum order of the frontal matrix, so the order in which the elements or equations are input is critical. MA42 optionally allows the use of direct access files for holding the matrix factors. As the rows and columns of the factors are generated they are written to buffers, which are held in-core. MA42 uses three buffers: one each for the reals in **PL** and **UQ** and one for the row and column indices of the variables in the factors. Once a buffer becomes full, it is written to the corresponding direct access file. The size of the buffers is chosen by the user but, for efficiency, the buffers should be chosen as large as in-core memory permits.

# 3 Exploiting zeros

## 3.1 Exploiting zeros within the frontal matrix

During the factorization, the frontal matrix may contain some zero entries. Treating the frontal matrix as a dense matrix results in unnecessary operations being performed with these zeros. As we discussed in the previous section, high level BLAS are used, so that the cost of these operations may not be prohibitive. If, however, the frontal matrix contains a significant number of zeros, it can be advantageous to exploit these zeros. In particular, there are many zeros in the front if the elements (or equations) are poorly ordered. To see how we can exploit the zeros, suppose the frontal matrix has been permuted to the form (2.6). By performing further row and column permutations, the frontal matrix can be expressed in the form

$$\mathbf{F} = \begin{pmatrix} \mathbf{F}_T & \mathbf{F}_{R_1} & \mathbf{0}_1 \\ \mathbf{F}_{C_1} & \mathbf{F}_{U_{11}} & \mathbf{F}_{U_{12}} \\ \mathbf{0}_2 & \mathbf{F}_{U_{21}} & \mathbf{F}_{U_{22}} \end{pmatrix}, \tag{3.1}$$

where $\mathbf{0}_1$ and $\mathbf{0}_2$ are zero matrices of order $k \times k_1$ and $k_2 \times k$, respectively. Assuming the current frontal matrix is of order $lfrnt \times kfrnt$ (with $lfrnt = kfrnt$ for element entry), and $k$ is the number of fully summed variables, $k_1$ and $k_2$ satisfy $0 \leq k_1 \leq lfrnt - k$ and $0 \leq k_2 \leq kfrnt - k$.

In place of (2.7) and (2.8), we now need only perform the updates

$$\mathbf{F}_{R_1} \leftarrow -\mathbf{F}_T^{-1}\mathbf{F}_{R_1} \tag{3.2}$$

and

$$\mathbf{F}_{U_{11}} \leftarrow \mathbf{F}_{U_{11}} + \mathbf{F}_{C_1}\mathbf{F}_{R_1}. \tag{3.3}$$

In practice, for element entry, $r \leq k$ pivots are chosen and again _TRSM and then _GEMM with interior dimension $r$ are used. When writing to the buffers, $\mathbf{F}_{R_1}$ and $\mathbf{F}_{C_1}$, rather than $\mathbf{F}_R$ and $\mathbf{F}_C$, are stored, resulting in savings in both the real and integer factor storage.

If more than one pivot is chosen, the updated matrices $\mathbf{F}_{R_1}$ and $\mathbf{F}_{C_1}$ may still contain some zeros. However, the experiments which we report in Table 4.4 (Section 3) show that, in general, the number of zeros remaining in the factors is small (for element problems, typically less than 10 per cent of the total number of entries in the factors). We do not, therefore, attempt to exploit zeros within $\mathbf{F}_{R_1}$ and $\mathbf{F}_{C_1}$.

We now discuss how the factorization phase of MA42 (that is, MA42B) can be modified to exploit zeros in the front. 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 the frontal matrix. When the next pivot is chosen, it is permuted to the last but one row and column of the frontal matrix, and so on. For element entry, if $r < k$ pivots are chosen for the current element, the remaining $k-r$ fully summed columns are permuted to columns $lfrnt-k$, $lfrnt-k-1$,..., $lfrnt-r+1$ of $\mathbf{F}$.

To take advantage of zeros we now need to perform further row permutations to collect the zeros in the fully summed columns into a block held in the first rows and, similarly, column permutations to collect the zeros in the fully summed rows into a block held in the first columns. To collect the zeros in the fully summed columns into a block, the number $l$ of rows with zeros in all the pivotal columns is initially set to 0. Rows 1 to $kfrnt-k+1$ of the pivotal columns are scanned in reverse order. Let $i$ be the index of the row currently being scanned. There are two possibilities: either there is a nonzero entry in at least one of the pivotal columns or all the entries in the pivotal columns are zero. In the first case, no action is needed and, assuming $i > 1$, we now scan row $i-1$. In the second case, we increment $l$ by one and, starting with row $l$, we search for a row with index at most $i-1$ with a nonzero entry in at least one of the pivotal columns. If we find such a row, it is interchanged with row $i$ and we then scan row $i-1$. Otherwise, there are no more rows to add to the zero block. Column permutations to collect the zeros in the fully summed rows into a block are then performed in a similar manner, before performing the elimination operations.

In this way, we can exploit zeros without altering the internal data structures of MA42. The overheads are the searching for zeros and, once found, the additional row and column permutations.

## 3.2 Efficient static condensations

Many finite-element discretizations include variables which are internal to the element. These so-called *static condensation* variables can be eliminated without reference to any other elements. In general, the order of an element is very much smaller than the frontsize so the relative cost of eliminating static condensation variables is negligible. MA42 offers the user the option of performing static condensations. If this option is chosen, each incoming element or equation is checked for internal variables. For the equation entry, the presence of more than one internal variable results in a structurally singular matrix and, in this case, a warning message is issued and, unless the user chooses to continue (in which case,

the equation is assembled straight into the frontal matrix), the computation terminates. Otherwise, pivots are chosen from the internal variables and, as each pivot is chosen, the corresponding row and column are permuted to the end of the incoming element matrix $\mathbf{A}^{(l)}$. The elimination operations are performed within the element (or equation) using the BLAS. Once all possible eliminations have been performed, the partially eliminated element or equation is assembled into the frontal matrix. The rows of **UQ** and columns of **PL** corresponding to the static condensation variables are then written to the in-core buffers.

For equation entry, if the equations are well-ordered, the current frontsize may not be significantly greater than the number of variables in an incoming equation. However, this may not be the case if the equations are poorly ordered and, for element entry, the order of an element is generally much smaller than the order of the frontal matrix. Thus, by writing the rows of **UQ** and columns of **PL** corresponding to the static condensation variables into the frontal matrix before writing to the buffers, zeros may be written to the buffers. In particular, if entry is by elements and the current frontsize is *frnt* and the incoming element has *nvar* variables, $frnt - nvar$ zero entries are written to the **L** and **U** buffers for each static condensation variable in the element. If there are a significant number of static condensation variables in the problem, this can lead to a large number of zeros being stored explicitly within the factors (see Table 4.3 in Section 4). This is avoided if we can write the rows of **UQ** and columns of **PL** corresponding to the static condensation variables directly to the buffers. For the equation entry, this can be done using the existing `MA42` data structures. However, for the element entry, if off-diagonal pivoting is used (and this is the default), an extra array of length *nvar* is required. We now explain this.

For each element (or equation), the user must specify in an array *ivar* of length *nvar* a list of the variables belonging to it. The reals must be specified in an array *avar*. For equation entry there is at most one static condensation variable and a single permutation is needed to permute this variable to the end of *ivar* and *avar*. Assuming the stability criteria is satisfied, the first $nvar - 1$ entries of *avar* are then scaled by the pivot, before the pivot and scaled array *avar* are written to the real buffers. The integer buffer requires the row and column indices of the variables. The row index is the current equation number and the column indices are in *ivar*. Thus for equation entry, no additional arrays are needed. However, the same is not true for element entry. In this case, once a pivot has been chosen from amongst the internal variables, row and column permutations within the element must generally be performed. If off-diagonal pivoting is used, both the row and column indices of the variables within the element are needed. If we are to avoid writing to the frontal matrix before writing to the buffers, we need a second array *jvar* of length *nvar* to enable *ivar* to be used to hold the rows indices and *jvar* the column indices. This extra array could be an internal work array. Alternatively, if the user were allowed to specify both the row and column indices of the incoming element, the code could be generalised to allow the input of rectangular elements. We anticipate that in a future release of the Harwell Subroutine Library, `MA42` will be superseded by a new frontal code which will use an additional array *jvar* for static condensations and which will allow rectangular elements to be input.

# 4 Numerical results

In this section, we first describe the problems that we use for testing the effects of exploiting zeros within the frontal solver and then present our numerical results. In all cases, the test problems arise in real engineering and industrial applications. For each test problem, if

numerical values were provided with the matrix, these values are used in our experiments. Otherwise, values for the entries were generated using the Harwell Subroutine Library (HSL) random number generator FA01. A brief description of each of the unassembled

| Identifier | Degrees of freedom | Static condensation variables | Number of elements | Description/discipline |
|---|---|---|---|---|
| CEGB3306 | 3222 | 0 | 791 | 2.5D framework problem |
| CEGB2919 | 2859 | 387 | 128 | 3D cylinder with flange |
| LOCK2232 | 2208 | 0 | 944 | Lockheed tower problem |
| TRDHEIM | 22098 | 7740 | 813 | Mesh of the Trondheim fjord |
| CRPLAT2 | 18010 | 192 | 3152 | Corrugated plate field |
| OPT1 | 15449 | 701 | 977 | Part of oil production platform |
| TSYL201 | 20685 | 52 | 960 | Part of oil production platform |
| AEAC6398 | 6398 | 3020 | 600 | Flow of a Newtonian fluid past a cylinder in a channel |
| AEA87000 | 87000 | 35553 | 5800 | Flow of a Newtonian fluid in a pipe with a sudden symmetric expansion |

Table 4.1: The element test problems

element problems is given in Table 4.1. The number of unassembled element problems available in the Harwell-Boeing Collection (Duff, Grimes and Lewis, 1992) is limited and all are small by today's standards. Consequently, we have only selected three representative problems, CEGB3306, CEGB2859, and LOCK2232, from this Collection. The problems AEAC6398 and AEA87000 are from Andrew Cliffe of AEA Technology and were chosen because they contain a large number of static condensation variables. The remaining problems (TRDHEIM, CRPLAT2, OPT1, and TSYL201) were supplied by Christian Damhaug of Det Norske Veritas, Norway. These problems are much larger than those in the Harwell-Boeing Collection and vary widely in the proportion of variables which are static condensation variables. For all the element problems, unless stated otherwise, we reorder the elements using the direct element reordering algorithm offered by the HSL routine MC43 before the frontal solver is used.

The assembled test problems, for which the MA42 equation entry is used, are listed in Table 4.2. The first three are taken from the Harwell-Boeing Collection and the other problems were supplied by Tim Davis, University of Florida. SHERMAN3 and ONETONE2 were chosen as they have a significant number of static condensation variables. ONETONE2 was also selected because Duff and Scott (1996*a*) observed that, compared with other HSL sparse solvers, MA42 performed poorly for this problem. For the same reason, WEST2021 and PSMIGR 3 were chosen. For unsymmetric problems, the Harwell Subroutine Library does not contain the equivalent of a profile minimizer but, for problems with a nearly symmetric structure, an ordering for the equation entry to MA42 can be obtained by applying the HSL profile reducing code MC40 to the pattern of $\mathbf{A}+\mathbf{A}^T$. To enable comparisons to be made between problems for which we have a good ordering and those for which only the original ordering is available, we include SHERMAN3 and WANG3 since they have (almost) symmetric sparsity patterns. For these problems, MC40 is used.

The experimental results given in this paper were obtained using 64-bit floating-point arithmetic on the following platforms:

- A Sun Ultra 1 using the epcf90 Fortran compiler, with compiler option -O.

| Identifier | Order | Static condensation variables | Number of entries | Description/discipline |
|---|---|---|---|---|
| SHERMAN3* | 5005 | 2109 | 20033 | Oil reservoir simulation |
| WEST2021 | 2021 | 34 | 7353 | Chemical engineering |
| PSMIGR 3 | 3140 | 0 | 543162 | Population migration |
| WANG3* | 26064 | 0 | 177168 | 3D semiconductor device simulation |
| ONETONE2 | 36057 | 1919 | 227628 | Harmonic balance method |

Table 4.2: The assembled test problems. * denotes reordered using MC40.

- A single processor of a CRAY J932 using the CRAY Fortran compiler cf77-7 with compiler option -Zv and the vendor-supplied BLAS.

All times are CPU times in seconds and include the i/o overhead for the use of direct access files for storing the matrix factors. In our tables of results, the solve times are for a single right-hand side ($nrhs = 1$). We remark that the savings in the solve times which result from our modifications to MA42 are even more worthwhile if $nrhs > 1$.

We first present results to illustrate the importance of implementing static condensation efficiently when the problem has a large number of static condensation variables. Our results are given in Table 4.3. We see that, in general, for problems with

| Identifier | MA42 version | Real storage (Kwords) | | Integer storage (Kwords) | Factorize time (seconds) | Solve time (seconds) |
|---|---|---|---|---|---|---|
| CEGB2919 | Standard | 1015 | (88) | 72 | 10.6 | 0.6 |
| | Modified | 927 | (0.1) | 57 | 10.6 | 0.5 |
| TRDHEIM | Standard | 6686 | (1642) | 535 | 51.9 | 3.8 |
| | Modified | 5048 | (4) | 352 | 48.9 | 2.8 |
| OPT1 | Standard | 16697 | (706) | 1196 | 588.8 | 10.2 |
| | Modified | 16248 | (258) | 352 | 588.0 | 10.0 |
| AEAC6398 | Standard | 1474 | (589) | 290 | 6.3 | 0.9 |
| | Modified | 992 | (106) | 169 | 6.0 | 0.6 |
| AEA87000 | Standard | 32114 | (11608) | 4432 | 192.9 | 22.2 |
| | Modified | 20702 | (195) | 2553 | 181.8 | 14.0 |
| SHERMAN3 | Standard | 597 | (210) | 594 | 2.7 | 0.8 |
| | Modified | 597 | (210) | 594 | 2.7 | 0.8 |
| ONETONE2 | Standard | 24246 | (21019) | 13102 | 100.4 | 24.8 |
| | Modified | 23346 | (19718) | 11801 | 100.2 | 21.0 |

Table 4.3: The effect of improving the efficiency of the static condensation phase (Sun Ultra 1). Standard denotes the HSL Release 12 version of MA42 and modified denotes that an extra array *jvar* is used for static condensations (see Section (3.2)). The figures in parentheses are the number of zeros (in thousands) which are held explicitly in the factors.

a large number of static condensation variables, using an extra array *jvar* as described in Section 3.2 substantially reduces the number of zeros held explicitly within the factors. In particular, for the unassembled problems we see that less than 10 per cent of the entries in the factors are zeros. The reduction in the real and integer storage for the factors leads to savings in the solve times. The small savings in the factorize times for some of the problems can be attributed to the reduction in data movement which results from writing directly from the element arrays to the buffers. For problem SHERMAN3 (which was

reordered using MC40), the standard and modified codes yield the same factors. Further investigation shows that, when reordered, the static condensation variables are all in rows of length 1 (that is, each static condensation variable is the only variable in its row), and these rows are the first rows to be assembled. For problem ONETONE2, a large number of zeros remain in the factors, which suggests that the problem is poorly ordered.

In Tables 4.4 to 4.7, we show the effects of exploiting zeros in the front. When zeros are exploited, we use the implementation of static condensation described in Section 3.2; otherwise, the HSL Release 12 version of MA42 is used. Problems AEAC6398 and AEA87000 had been preordered using MC43 before they were supplied to us but for the remaining unassembled problems, we give results on the Sun Ultra both for the supplied order and for the ordering obtained using MC43. On the CRAY J932, the results are for the reordered elements.

| Identifier | Ordered | Zeros exploited | Real storage (Kwords) | | Integer storage (Kwords) | Factorize time (seconds) | Solve time (seconds) |
|---|---|---|---|---|---|---|---|
| CEGB3306 | N | N | 1528 | (1292) | 260 | 5.9 | 0.9 |
| | N | Y | 261 | (26) | 48 | 3.5 | 0.2 |
| | Y | N | 390 | (186) | 71 | 3.8 | 0.3 |
| | Y | Y | 372 | (0.7) | 68 | 1.8 | 0.2 |
| CEGB2919 | N | N | 1154 | (313) | 87 | 10.2 | 0.8 |
| | N | Y | 843 | (2) | 52 | 8.7 | 0.5 |
| | Y | N | 1015 | (88) | 72 | 10.6 | 0.6 |
| | Y | Y | 927 | (0.1) | 57 | 10.5 | 0.5 |
| LOCK2232 | N | N | 2827 | (2552) | 433 | 17.4 | 1.7 |
| | N | Y | 287 | (12) | 46 | 10.8 | 0.2 |
| | Y | N | 220 | (33) | 40 | 0.9 | 0.2 |
| | Y | Y | 188 | (0.2) | 35 | 0.9 | 0.1 |
| TRDHEIM | N | N | 7566 | (2174) | 598 | 56.1 | 4.5 |
| | N | Y | 5403 | (11) | 383 | 52.5 | 3.0 |
| | Y | N | 6686 | (1642) | 535 | 51.9 | 3.8 |
| | Y | Y | 5048 | (4) | 352 | 50.3 | 2.8 |
| CRPLAT2 | N | N | 45068 | (7646) | 6381 | 3422.4 | 31.1 |
| | N | Y | 39011 | (59) | 5530 | 3844.2 | 27.6 |
| | Y | N | 12937 | (1480) | 2120 | 259.4 | 8.6 |
| | Y | Y | 12791 | (2) | 2095 | 257.3 | 8.5 |
| OPT1 | N | N | 61453 | (42332) | 4372 | 2550.9 | 55.6 |
| | N | Y | 19925 | (106) | 383 | 999.4 | 14.6 |
| | Y | N | 16697 | (706) | 352 | 588.8 | 10.2 |
| | Y | Y | 16000 | (10) | 352 | 594.9 | 10.2 |
| TSYL201 | N | N | 33668 | (624) | 1638 | 1662.1 | 22.0 |
| | N | Y | 33056 | (12) | 1568 | 1684.7 | 22.0 |
| | Y | N | 20934 | (97) | 1021 | 578.6 | 13.4 |
| | Y | Y | 20837 | (0.4) | 1005 | 576.9 | 12.9 |
| AEAC6398 | Y | N | 1474 | (589) | 290 | 6.3 | 0.9 |
| | Y | Y | 906 | (21) | 276 | 5.7 | 0.5 |
| AEA87000 | Y | N | 32114 | (11608) | 4432 | 192.9 | 22.2 |
| | Y | Y | 20702 | (195) | 2553 | 185.2 | 14.0 |

Table 4.4: The effect on unassembled problems of exploiting zeros in the front (Sun Ultra 1). The figures in parentheses are the number of zeros (in thousands) which are held explicitly in the factors.

From Table 4.4, we see that, in general, if the elements are not well ordered, worthwhile savings are achieved in the real and integer factor storage and in the factorize and solve

| Identifier | Zeros Exploited | Factorize time (seconds) | Solve time (seconds) |
|---|---|---|---|
| CEGB3306 | N | 1.0 | 0.07 |
| | Y | 1.1 | 0.07 |
| CEGB2919 | N | 2.3 | 0.07 |
| | Y | 2.4 | 0.06 |
| LOCK2232 | N | 0.6 | 0.04 |
| | Y | 0.6 | 0.04 |
| TRDHEIM | N | 11.6 | 0.49 |
| | Y | 12.0 | 0.38 |
| CRPLAT2 | N | 53.6 | 1.16 |
| | Y | 56.7 | 1.12 |
| OPT1 | N | 88.3 | 0.94 |
| | Y | 87.7 | 0.84 |
| TSYL201 | N | 85.0 | 1.03 |
| | Y | 88.7 | 1.03 |

Table 4.5: The effect on ordered unassembled problems of exploiting zeros in the front (CRAY J932).

times by exploiting zeros in the front. Once the elements have been ordered, the savings which result from exploiting zeros are much smaller. By comparing Tables 4.3 and 4.4, we see that once the problems CEGB2919, TRDHEIM and AEA87000 have been ordered, the reduction in the number of zeros within the factors comes from using the array *jvar* when implementing static condensation: no further reduction is achieved by exploiting zeros in the front. Moreover, for the problem CEGB2919, the factor storage and the factorize time are less when zeros are exploited for the original ordering than for the **MC43** ordering. This is because, for this problem, more zeros are exploited when the original ordering is used. The results in Tables 4.4 and 4.5 also show that, if the number of zeros in the

| Identifier | Zeros exploited | Real storage (Kwords) | | Integer storage (Kwords) | Factorize time (seconds) | Solve time (seconds) |
|---|---|---|---|---|---|---|
| SHERMAN3 | N | 597 | (313) | 594 | 2.7 | 0.76 |
| | Y | 387 | (0.006) | 393 | 2.7 | 0.54 |
| WEST2021 | N | 545 | (512) | 229 | 1.1 | 0.44 |
| | Y | 60 | (27) | 20 | 0.5 | 0.07 |
| PSMIGR 3 | N | 9433 | (1973) | 1913 | 847.8 | 7.5 |
| | Y | 7865 | (405) | 1032 | 811.2 | 6.9 |
| WANG3 | N | 38480 | (12792) | 37504 | 971.4 | 48.1 |
| | Y | 25687 | (0) | 25068 | 956.8 | 34.1 |
| ONETONE2 | N | 24246 | (21019) | 13102 | 100.4 | 24.8 |
| | Y | 4356 | (700) | 1311 | 46.8 | 3.3 |

Table 4.6: The effect on assembled problems of exploiting zeros in the front (Sun Ultra 1). The figures in parentheses are the number of zeros (in thousands) which are held explicitly in the factors.

front is a small proportion of the total number of nonzeros, the overheads of searching for them and the increased data movement to accumulate them into blocks can increase the

factorize time. For example, problem AEA87000 has a factorization time of 182 seconds when static condensation is implemented efficiently (Table 4.3) but this increases to 185 seconds on the Sun Ultra when we also look for zero blocks. Unfortunately, we do not know how many zeros there are in the front until we search for them. However, since the solve time is not increased and may be reduced by exploiting zeros, an increase in the factorize time may be considered tolerable. In particular, if the factors are to be used to solve for a large number of right-hand sides an increase in the factorize time could be more than compensated for by the storage and solve time savings. This has been our experience when using **MA42** for eigenvalue calculations (Scott and Lehoucq, 1997).

| Identifier | Zeros exploited | Factorize time (seconds) | Solve time (seconds) |
|---|---|---|---|
| SHERMAN3 | N | 1.8 | 0.49 |
| | Y | 2.6 | 0.41 |
| WEST2021 | N | 1.2 | 0.14 |
| | Y | 0.9 | 0.07 |
| PSMIGR 3 | N | 145.2 | 0.80 |
| | Y | 107.7 | 0.54 |
| WANG3 | N | 210.1 | 12.59 |
| | Y | 180.5 | 8.09 |
| ONETONE2 | N | 84.3 | 5.56 |
| | Y | 37.9 | 1.83 |

Table 4.7: The effect on assembled problems of exploiting zeros in the front (CRAY J932).

For the assembled problems, exploiting zeros in the front successfully reduces the number of zeros held explicitly within the factors. Moreover, if the equations are not well-ordered (for example, WEST2021 and PSMIGR 3), there are significant savings in the factorize and solve times and in the factor storage. As remarked earlier, Duff and Scott (1996*a*) observed that **MA42** performed poorly for problem ONETONE2, for which we do not have a good ordering available. Our results show that exploiting zeros significantly enhances the performance of the frontal solver for this problem. When a problem is well-ordered (SHERMAN3 and WANG3), there is either a modest saving in the factorize time or an increase in the factorize time caused by the extra data movement, but there can still be worthwhile savings in the solve times.

# 5 Conclusions

We have looked at the need to exploit zeros in frontal solvers and have implemented two modifications to our general frontal solver **MA42** which can substantially improve its performance. In particular, the performance is improved if the problem has a large number of static condensation variables or is poorly ordered. This has indicated to us the desirability of incorporating zero detection strategies within a new version of our general frontal code. The modified code would allow the user to request that zeros in the front are exploited and would also permit the input of rectangular elements. Our results have also highlighted the need for a good ordering for assembled problems. By comparing our results with those of Duff and Scott (1996*a*), we see that, even with the improvements we have made, for most assembled problems, approaches other than the frontal method are likely to be faster and require less factor storage. However, the frontal code does offer

the advantage of holding the factors out-of-core and is thus far superior in terms of main memory. Finally, we remark that we have recently developed a frontal code **MA62** (Duff and Scott, 1997) for solving systems of symmetric positive-definite unassembled finite-element equations and this code optionally exploits zeros in the front.

# 6 Acknowledgements

I would like to thank my colleague Iain Duff at Rutherford for his interest, for helpful discussions, and for his comments on this report.

# References

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

I.S. Duff. Design features of a frontal code for solving sparse unsymmetric linear systems out-of-core. *SIAM J. Scientific and Statistical Computing*, **5**, 270–280, 1984.

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

I.S. Duff and J.A. Scott. A comparison of frontal software with other sparse direct solvers. Technical Report RAL-TR-96-102, Rutherford Appleton Laboratory, 1996*a*.

I.S. Duff and J.A. Scott. The design of a new frontal code for solving sparse unsymmetric systems. *ACM Trans. Mathematical Software*, **22**(1), 30–45, 1996*b*.

I.S. Duff and J.A. Scott. MA62 – a new frontal code for sparse positive-definite symmetric systems from finite-element applications. Technical Report RAL-TR-97-012, Rutherford Appleton Laboratory, 1997.

I. S. Duff, R. G. Grimes, and J. G. Lewis. Users' guide for the Harwell-Boeing sparse matrix collection (Release I). Technical Report RAL-TR-92-086, Rutherford Appleton Laboratory, 1992.

Harwell Subroutine Library. *A Catalogue of Subroutines (Release 12).* Advanced Computing Department, AEA Technology, Harwell Laboratory, Oxfordshire, England, 1996.

P. Hood. Frontal solution program for unsymmetric matrices. *Inter. Journal on Numerical Methods in Engineering*, **10**, 379–400, 1976.

B.M. Irons. A frontal solution program for finite-element analysis. *Inter. Journal on Numerical Methods in Engineering*, **2**, 5–32, 1970.

J.A. Scott and R.B. Lehoucq. Implicitly restarted arnoldi methods and eigenvalues of the discretized navier stokes equations. Technical Report to appear, Rutherford Appleton Laboratory, 1997.