

**MA62 - A frontal code for sparse
positive-definite symmetric systems from
finite-element applications.***

Iain S. Duff and Jennifer A. Scott

Abstract

We describe the design, implementation, and performance of a frontal code for the solution of large sparse symmetric systems of linear finite-element equations. The code is intended primarily for positive-definite systems since numerical pivoting is not performed. The resulting software package, MA62, will be included in Release 13 of the Harwell Subroutine Library (HSL). We illustrate the performance of our new code on a range of problems arising from real engineering and industrial applications. The performance of the code is compared with that of the HSL general frontal solver MA42 and with other positive-definite codes from the Harwell Subroutine Library.

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

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

Running title: Symmetric frontal code

* Current reports available by anonymous ftp from [matisa.cc.rl.ac.uk](ftp://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.

July 11, 1997.

Contents

1	Introduction	1
2	Frontal schemes	2
3	The software package MA62	3
3.1	The user interface	4
3.1.1	Initialization	5
3.1.2	Prepass	5
3.1.3	Symbolic factorization	6
3.1.4	Direct access files	7
3.1.5	Numerical factorization and optional solve	8
3.1.6	Further solves	9
3.2	Internal data structures	9
3.3	The use of high level BLAS	12
3.4	Exploiting zeros in the front	16
4	Numerical results	18
4.1	Test problems	18
4.2	The effect of ordering	19
4.3	The effect of exploiting zeros	21
4.4	The effect of blocking parameters	23
4.5	The performance of MA62 compared with MA42	31
4.6	A comparison of the frontal code MA62 with other HSL codes	34
5	Conclusions	41
6	Availability of the code	42
7	Acknowledgements	42
A	Appendix: Specification Sheets	45

1 Introduction

In this report, we discuss the design and use of a code for the solution of the linear systems of equations

$$\mathbf{A}\mathbf{X} = \mathbf{B} \quad (1.1)$$

where the $n \times n$ matrix \mathbf{A} is large, sparse, and symmetric. \mathbf{B} is an $n \times \text{nrhs}$ ($\text{nrhs} \geq 1$) matrix of right-hand sides and \mathbf{X} is the $n \times \text{nrhs}$ solution matrix. It is assumed that the matrix \mathbf{A} is an elemental matrix, that is, it is a sum of finite-element matrices

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

where each element matrix $\mathbf{A}^{(l)}$ has nonzeros only in a few rows and columns and corresponds to the matrix from element l . The frontal method was originally developed by Irons (1970) for the solution of symmetric positive-definite systems which come from finite-element discretizations in structural analysis. However, it was later realised that the method could be extended and modified to be applicable to a far wider class of problems, including those for which \mathbf{A} is any general unsymmetric matrix (see, for example, Hood, 1976, Duff, 1981). A frontal code for unsymmetric systems, **MA32**, was developed for the Harwell Subroutine Library (1996) by Duff (1981, 1983, 1984). This code was very widely used before being substantially restructured and improved by Duff and Scott (1993, 1996). The upgraded code, **MA42**, which has superseded **MA32** in the Harwell Subroutine Library (HSL), is efficient on a wide range of modern computers and has been used to solve problems from many different application areas.

From the feedback we have received from users, it is apparent that **MA42** is frequently used to solve finite-element problems for which the system matrix \mathbf{A} is symmetric and positive definite. However, apart from offering an option of restricting pivoting to the diagonal, **MA42** does not exploit symmetry or positive definiteness and, as a result, the code is more expensive in terms of both storage requirements and operation counts than it need be for this class of problems. **MA42** is also complicated by an option allowing the input of the assembled matrix \mathbf{A} by rows. Our goal is to design and develop an efficient frontal code specifically for symmetric positive-definite elemental matrices.

Our new code, **MA62**, is now available and will be included in the next release of the Harwell Subroutine Library. Details of how to obtain the code are given in Section 6.

The outline of this report is as follows. In Section 2, we briefly review frontal schemes for symmetric positive-definite elemental matrices. The design of **MA62**, including the user interface, the internal data structures, and the use of high level BLAS, is discussed in Section 3. In Section 4, we illustrate the performance of the new code, and compare it with our general frontal code **MA42** and with other Harwell Subroutine Library codes for the solution of symmetric positive-definite systems. Concluding comments are made in Section 5. Specification Sheets for **MA62** are given in the Appendix.

2 Frontal schemes

The frontal method is a variant of Gaussian elimination and, for the symmetric system (1.1), involves the matrix factorization

$$\mathbf{A} = (\mathbf{PL})\mathbf{D}(\mathbf{PL})^T, \quad (2.1)$$

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

$$(\mathbf{PL})\mathbf{Z} = \mathbf{B}, \quad (2.2)$$

then the diagonal solution

$$\mathbf{D}\mathbf{Y} = \mathbf{Z}, \quad (2.3)$$

followed by the back substitution

$$(\mathbf{PL})^T\mathbf{X} = \mathbf{Y}. \quad (2.4)$$

The main feature of the method is that the contributions $\mathbf{A}^{(l)}$ from the finite-elements (see (1.2)) are assembled one at a time and the construction of the assembled coefficient matrix \mathbf{A} is avoided by interleaving assembly and elimination operations. An assembly operation is of the form

$$a_{ij} \Leftarrow a_{ij} + a_{ij}^{(l)}, \quad (2.5)$$

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 (2.5) 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

$$a_{ij} \Leftarrow a_{ij} - a_{il}[a_{ll}]^{-1}a_{lj} \quad (2.6)$$

may be performed as soon as all the terms in the triple product in (2.6) are fully summed. At any stage during the assembly and elimination processes, the fully and partially summed variables are held in a dense matrix, termed the *frontal* matrix. The power of frontal schemes comes from the following observations:

- since the frontal matrix is held as a dense matrix, dense linear algebra kernels, in particular, the Level 3 Basic Linear Algebra Subprograms (BLAS) (Dongarra, DuCroz, Duff and Hammarling, 1990), can be used during the numerical factorization.
- the matrix factors need not be held in-core, which allows large problems to be solved using only modest amounts of high-speed memory,

The number of floating-point operations and the storage requirements for the frontal method are dependent on the size of the frontal matrix at each stage of the computation. Since the size of the frontal matrix increases when a variable appears for the first time and decreases whenever a variable is eliminated, the order in which the element matrices are input is critical for the efficiency of the method. There has been considerable attention paid to the problem of automatically choosing a good ordering. Many of the proposed algorithms are similar to those for bandwidth reduction of assembled matrices (see, for example, Duff, Reid and Scott, 1989, and the references therein). To give the user the greatest flexibility in deciding how to order the elements, we have chosen not to incorporate element ordering within the MA62 package. However, the Harwell Subroutine Library routine MC43 (Duff et al., 1989) can be used to preorder the elements for MA62.

3 The software package MA62

In this section, we describe the user interface to MA62, its internal data structures, and its use of BLAS kernels. We highlight how it has been possible

to take advantage of symmetry and positive definiteness when designing the code.

3.1 The user interface

An initial decision when designing MA62 was that it should have a user interface which was similar to that of the unsymmetric frontal solver MA42. This was not only because we felt the flexibility of the reverse communication interface used by MA42 remained appropriate for the symmetric positive-definite case, but because we wanted to make it straightforward for a user who was familiar with MA42 to use MA62. The MA62 package has six entries which may be called directly by the user. Each of the subroutines are named according to the naming convention of the Harwell Subroutine Library, with the single precision version having names commencing with MA62 plus one more letter, and double precision versions with the additional sixth letter D. For simplicity, we will use the single precision names throughout this report. The user-callable entries are:

Initialization : MA62I initializes the parameters which control the execution of the package. A single call must be made to MA62I before any other routines in the MA62 package are called.

Prepass: MA62A determines in which element each variable appears for the last time and thus when a variable is fully summed and can be eliminated. MA62A must be called once for each element.

Symbolic factorization : MA62J uses the information from MA62A to determine the amount of real and integer storage required for the factorization. MA62J must be called once for each element, in the same order as in the calls to MA62A.

Direct access files : MA62P sets up direct access files for holding the matrix factors. Use of MA62P is optional. If direct access files are used, a single call to MA62P must be made.

Numerical factorization : MA62B uses the information generated by MA62A and MA62J in the factorization of the matrix (1.2) and, if element right-hand sides $\mathbf{B}^{(l)}$ are specified, MA62B solves the equations (1.1) with

right-hand side(s) $\mathbf{B} = \sum_{k=1}^m \mathbf{B}^{(l)}$. MA62B must be called once for each element, in the same order as in the calls to MA62A and MA62J.

Solve : MA62C uses the factors produced by MA62B to rapidly solve for further right-hand sides. Use of MA62C is optional. If used, a single call to MA62C will solve for the number of right-hand sides specified by the user.

We briefly discuss each of these subroutines. Full details of their argument lists and their calling sequences are given in the Specification Sheets (see Appendix).

3.1.1 Initialization

The user must make a single call to MA62I prior to calling any of the other routines in the MA62 package. MA62I assigns default values to the control parameters held in the arrays ICNTL and CNTL. These parameters control the action of the subroutines within the MA62 package. They include parameters to control the level of diagnostic printing and parameters which specify the number of bytes for a real and an integer word. Full details of the control parameters and their default values are included in the Specification Sheets. Should the user want a control parameter to have a value other than its default, the appropriate parameter should be reset after the call to MA62I. MA62I also initializes the array ISAVE, which is used to hold variables that must be preserved between calls to routines in the MA62 package but are unlikely to be of interest to the user. The array ISAVE is also used both to check the data input by the user and to ensure that the user has called the routines in the MA62 package in the correct sequence.

3.1.2 Prepass

MA62A must be called for each element to specify the variable indices associated with it. This subroutine records, in the array LAST of length `ndf` (where `ndf` is the largest integer used to index a variable), the call at which each variable appears for the last time (becomes fully summed). This information must be passed unchanged to the symbolic factorization and numerical factorization subroutines (MA62J and MA62B, respectively). The elements must be presented to the symbolic and numerical factorization routines in exactly the same order as to MA62A.

3.1.3 Symbolic factorization

One of the difficulties facing the user of a frontal code is the need to specify file sizes for the factors and the maximum front sizes required before the computation begins. A symbolic factorization works only with the variable indices associated with the elements and, by assuming each variable may be eliminated as soon as it is fully summed, it determines the maximum order of the frontal matrix and the file sizes needed for the factors. For general unsymmetric matrices, the need to incorporate pivoting means the statistics returned by the symbolic factorization are only lower bounds on the front size and file sizes actually needed. But for positive-definite matrices a variable can always be eliminated once it is fully summed and, since the symbolic factorization is inexpensive, we decided to require the user to perform a symbolic factorization before the numerical factorization commences. The symbolic factorization is performed by calling the subroutine MA62J for each element in the same order as they were presented to MA62A. An entry in ISAVE is flagged so that a check can be made at the start of the numerical factorization that MA62J has been called. The real and the integer storage required by the factorization are returned to the user after the last element has been input in the information array INFO. If the user provides the numerical factorization with less space than that determined by MA62J, it can be detected immediately and the computation terminated with an error message.

MA62 allows the user to specify (using the control parameter ICNTL(5)) the minimum number of pivots that will be selected at any stage. Delaying performing eliminations until the number of fully summed variables is at least ICNTL(5) ($\text{ICNTL}(5) \geq 1$) increases the Level 3 BLAS component of the factorization (see Section 3.3), albeit at the cost of more floating-point operations, increased storage for the reals in the factor, and, in general, an increase in the maximum front size and consequently in the in-core storage required. Since the symbolic factorization is cheap to perform, the user may want to look at the effect on the maximum front size and the file sizes of different values of ICNTL(5) before starting the numerical factorization. MA62 has been designed so that this can be done in a straightforward way. Provided the user has performed a complete sequence of calls to MA62J, the user may reset ICNTL(5) and then repeat the sequence of calls to MA62J **without** making any further changes to the input parameters. Having chosen the

minimum pivot block size, the calls to the numerical factorization routine MA62B must immediately follow a sequence of calls to MA62J with the **same** value of ICNTL(5). A component of ISAVE is used to check this.

3.1.4 Direct access files

A key feature of MA62 (and of MA42 and the earlier code MA32) is that it offers the user the option of holding the matrix factors in direct access files. MA62 optionally uses two direct access files, one for the reals in the factors and one for the indices of the variables in the factors. A single call to the subroutine MA62P sets up the direct access files. The user must specify the stream numbers for the direct access files and may optionally name the files.

Corresponding to each direct access file is an in-core buffer (or workspace). Eliminations are performed during the numerical factorization whenever the number of fully summed variables KR is at least ICNTL(5). The eliminations generate KR columns of the matrix factor **(PL)D**. The columns are written to the real in-core buffer and each time columns are written, the following integer data is written (in order) to the integer buffer:

1. The number of integers being written.
2. The number KR of eliminations (that is, the number of columns being written to the real buffer).
3. The current front size FRNT.
4. A list of the indices of the FRNT variables in the current front.
5. The number of integers being written.

The number of integers is held as the first and last entry to allow the real and integer data to be scanned both in the order in which it is written and in reverse order. Reverse order is needed when performing the back substitution (2.4). Once either a buffer is full or all the elimination operations are complete, the contents of the buffer are output to the associated direct access file. Use of direct access files is unnecessary if there is sufficient in-core space for the factors. Numerical results given in Sections 4.5 and 4.6 show that, on some machines, the overheads which result from using direct access files can be considerable, particularly during the solve phase.

We remark that the unsymmetric frontal code MA42 uses three direct access files since it stores the \mathbf{L} and \mathbf{U} factors of \mathbf{A} separately. Moreover, since the unsymmetric code uses off-diagonal pivoting to maintain stability, it is necessary to hold both row and column indices of the variables in the frontal matrix. Therefore, in addition to the saving in real storage, if the minimum pivot block size ICNTL(5) is set to 1, MA62 uses approximately half the integer storage of MA42, and if ICNTL(5) > 1, the savings are even greater. This is illustrated in Table 4.12 in Section 4.5.

3.1.5 Numerical factorization and optional solve

The numerical factorization subroutine MA62B accepts the element matrices $\mathbf{A}^{(l)}$ and, optionally, the corresponding element right-hand sides $\mathbf{B}^{(l)}$, one at a time. Only the upper triangular part of $\mathbf{A}^{(l)}$ needs to be specified by the user but, to facilitate the assembly process, MA62B copies the upper triangular part into the lower triangle. If any variables in the incoming element are internal to the element, they are eliminated within the element and the columns of the matrix factor which are generated are written to the buffers before contributions from the remaining variables are assembled into the frontal matrix. These internal variables are termed *static condensation variables* and the number of such variables is returned to the user in the information array INFO at the end of the numerical factorization. As the element is being assembled, the number KR of variables in the front which are now fully summed is counted. If this number is at least as large as the control parameter ICNTL(5), KR eliminations are performed (see Section 3.3). Otherwise, as long as elements remain to be assembled, control is returned to the user for the next element to be input.

Since the space required by the numerical factorization can not exceed that determined by the symbolic factorization, on the assumption that the user supplies consistent data (and using the array ISAVE we have incorporated many checks for this in the program), the only way MA62 can terminate before the factorization is complete is if the matrix \mathbf{A} is found not to be positive definite. In MA62, each pivot candidate (that is, each fully summed variable) is checked to see that it is of absolute value at least as large as the control parameter CNTL(1) (with default value zero). If a pivot is found to be too small, an error flag is set and control returned to the user. Note that although the factorization will not proceed, it need not be the case that

the matrix is singular. If a pivot is found to be negative, the matrix is not positive definite, but provided it is of absolute value at least `CNTL(1)`, the computation continues and the number of negative pivots is returned to the user in the information array `INFO` at the end of the computation.

3.1.6 Further solves

In common with many other sparse direct codes, `MA62` offers the option of solving for further right-hand side matrices \mathbf{B} , without recalling the numerical factorization routine. On each call to the solve routine `MA62C`, the number of right-hand sides (columns of \mathbf{B}) must be specified in `nrhs`. In contrast to `MA62A`, `MA62J`, and `MA62B`, there is no input by elements to `MA62C` and the right-hand sides must be input in assembled form. This gives `MA62C` a straightforward interface. In `MA62`, the forward elimination and diagonal solution ((2.2) and (2.3)) are combined. For the forward elimination, the real and integer data for the factors is read in the order in which it was written and, for the back substitution, the data is read in reverse order. When `nrhs` > 1, both the forward elimination and back substitution steps use `_GEMM`, the Level 3 BLAS matrix-matrix multiplication kernel (see Section 3.3). When there is only one right-hand side (`nrhs` = 1), Level 2 BLAS are used. It should be noted that \mathbf{B} is involved in this matrix-matrix product and that `_GEMM` becomes more efficient with an increased number of columns in \mathbf{B} . This is illustrated in Section 4.5 (Tables 4.13 to 4.15).

3.2 Internal data structures

The internal data structures used by `MA62` are simplified and modified versions of those used by `MA42` and, earlier, by `MA32` (see Duff, 1981, for a detailed description of the internal arrays used by `MA32`). The user must supply both a real workspace array and an integer workspace array, which are subdivided as follows:

BUFR	FA	FRHS
------	----	------

IBUFR	LHED	LPIV	LASTFT
-------	------	------	--------

We first discuss the real workspace.

BUFR is the in-core buffer for the reals in the factor $(\mathbf{PL})\mathbf{D}$. The length of **BUFR** is chosen by the user. On exit from the symbolic factorization, the required length of the file for the reals in the factor is given by **INFO(7)**. If the user calls the numerical factorization **MA62B** with **nrhs** right-hand sides, the total real storage for the factors and the right-hand sides is $\mathbf{INFO}(7) + \mathbf{nrhs} \times \mathbf{ndf}$ (**ndf** is the largest integer used to index a variable). If direct access files are not being used, **BUFR** must be at least this length. Otherwise, for efficiency, **BUFR** should be chosen so that $\mathbf{INFO}(7) + \mathbf{nrhs} \times \mathbf{ndf} = k_1 \times \mathbf{BUFR}$ with $k_1 \geq 1$ as small as available space permits. After the eliminations following an element assembly, the columns of $(\mathbf{PL})\mathbf{D}$ which have been generated are written to **BUFR** (see Subsection 3.1.4).

FA holds the current frontal matrix. On exit from the symbolic factorization, the maximum front size is given by **INFO(3)**. The frontal matrix is held as a square matrix of order **INFO(3)** but only the data in the upper triangular part is meaningful. Storage for both upper and lower triangular parts is needed to permit the use of Level 3 BLAS (see Section 3.3).

FRHS holds the right-hand sides corresponding to the current frontal matrix. **FRHS** is a matrix of size **INFO(3)** by **nrhs**, where **nrhs** is the number of right-hand sides to be solved for at the same time as the factorization. **FRHS** is not needed if **nrhs** = 0.

We now turn our attention to the integer workspace.

IBUFR is the in-core buffer for the integer data for the factors. The length of **IBUFR** is chosen by the user. On exit from the symbolic factorization, the required length of the file for the integer data is given by **INFO(8)**. If direct access files are not being used, **IBUFR** must be at least this length.

Otherwise, for efficiency, `IBUFR` should be chosen so that `INFO(8) = $k_2 \times \text{IBUFR}$` with $k_2 \geq 1$ as small as available space permits. After the eliminations following an element assembly, the indices of the columns of $(\mathbf{PL})\mathbf{D}$ which have been generated by the eliminations are written to `IBUFR` (see Subsection 3.1.4).

The remaining integer arrays `LHED`, `LPIV`, and `LASTFT` are each of length `INFO(3)` (the maximum frontsize). To discuss these arrays we need some terminology. The *global* index of a variable is the integer given to it by the user in the calls to the subroutines in the package (if the variables are numbered contiguously then the global index of a variable is its index in the assembled matrix \mathbf{A}). The *local* index of a fully or partially summed variable refers to its location in the current frontal matrix. During the factorization, the array `LAST` (which was set by the prepass `MA62A` to hold the call at which each variable becomes fully summed) is used as workspace and it, together with the internal arrays `LHED`, `LPIV`, and `LASTFT`, are used to provide an efficient mapping between global and local indices as follows:

`LHED` holds the global indices of the variables in the current frontal matrix.

`LPIV` holds the local indices of the fully summed variables in the current frontal matrix.

`LAST` is used to hold the local indices of the variables in the current frontal matrix. If the variable with global index `JVAR` is partially summed, its local index is `-LAST(JVAR)`. Otherwise, `LAST(JVAR)` is the assembly step at which `JVAR` becomes fully summed.

`LASTFT` is used to hold the assembly step at which variables in the front become fully summed (the information which was originally held in `LAST`). For each partially summed variable `JVAR` in the front, `LASTFT(-LAST(JVAR))` holds the assembly step at which `JVAR` is fully summed.

To summarize the use of `LHED`, `LAST` and `LASTFT`, suppose on the first call to `MA42B` that `LAST(JVAR) = KELL`. If, at some stage before the assembly of the element `KELL`, `JVAR` is partially summed with local index `L`, then `LHED(L) =`

JVAR, LASTFT(L) = KELL, and LAST(JVAR) = -L. The use of LASTFT to hold information which was originally held in LAST avoids the use of a second array of length ndf.

3.3 The use of high level BLAS

In this section, we describe the use of the BLAS in performing both the numerical factorization and the forward elimination and back substitution steps. We first consider the factorization. After the assembly of an element, the frontal matrix can be written in the form

$$\mathbf{F} = \begin{pmatrix} \mathbf{F}_T & \mathbf{F}_C^T \\ \mathbf{F}_C & \mathbf{F}_U \end{pmatrix}, \quad (3.1)$$

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

$$\begin{pmatrix} \mathbf{F}_T & \mathbf{F}_C^T \\ \mathbf{F}_C & \mathbf{F}_U \end{pmatrix} = \begin{pmatrix} \mathbf{F}_{TL} & \mathbf{0} \\ \mathbf{F}_L & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{D}_T & \mathbf{0} \\ \mathbf{0} & \hat{\mathbf{F}} \end{pmatrix} \begin{pmatrix} \mathbf{F}_{TL}^T & \mathbf{F}_L^T \\ \mathbf{0} & \mathbf{I} \end{pmatrix}, \quad (3.2)$$

where

$$\mathbf{F}_T = \mathbf{F}_{TL} \mathbf{D}_T \mathbf{F}_{TL}^T, \quad (3.3)$$

$$\mathbf{F}_L = \mathbf{F}_C (\mathbf{D}_T \mathbf{F}_{TL}^T)^{-1} \quad (3.4)$$

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

$$\hat{\mathbf{F}} = \mathbf{F}_U - \mathbf{F}_L \mathbf{D}_T \mathbf{F}_L^T. \quad (3.5)$$

Level 3 BLAS kernels can be used in computing \mathbf{F}_L and $\hat{\mathbf{F}}$. From (3.4) it follows that

$$\mathbf{F}_L \mathbf{D}_T = \mathbf{F}_C \mathbf{F}_{TL}^{-T}. \quad (3.6)$$

The Level 3 kernel `_TRSM` can be used to form $\mathbf{F}_L \mathbf{D}_T$ and \mathbf{F}_L^T follows by diagonal scaling. If the number of fully summed variables is KR, the Level 3 kernel `_GEMM` with internal dimension KR may then be used to compute (3.5). However, since only the upper triangular part of $\hat{\mathbf{F}}$ is needed, forming

the whole of $\hat{\mathbf{F}}$ involves many unnecessary operations. With a front size of FRNT , the computation (3.5) requires $2 * \text{KR} * (\text{FRNT} - \text{KR})^2$ floating-point operations. We could form only the upper triangle of $\hat{\mathbf{F}}$ by updating each column using the Level 2 kernel `_GEMV`. This would reduce the number of floating-point operations to $\text{KR} * (\text{FRNT} - \text{KR}) * (\text{FRNT} - \text{KR} + 1)$ but the efficiency gains of using Level 3 BLAS would be lost. To take advantage of Level 3 BLAS while at the same time restricting the number of unnecessary operations, the columns of $\hat{\mathbf{F}}$ may be computed in blocks. Assuming a block size of NB with $\text{FRNT} - \text{KR} = m * \text{NB}$ then, using Fortran 90 section notation,

$$\hat{\mathbf{F}}(1 : \text{K}, \text{K} - \text{NB} + 1 : \text{K}) = \mathbf{F}_U(1 : \text{K}, \text{K} - \text{NB} + 1 : \text{K}) - (\mathbf{F}_L \mathbf{D}_T)(1 : \text{K}, 1 : \text{KR}) * \mathbf{F}_L^T(1 : \text{KR}, \text{K} - \text{NB} + 1 : \text{K}) \quad (3.7)$$

where $\text{K} = \text{K}_1 * \text{NB}$, $\text{K}_1 = 1, 2, \dots, m$. The update (3.7) can be performed using `_GEMM` with interior dimension KR (the Level 2 kernel `_GEMV` is used if $\text{KR} = 1$).

By increasing the minimum number of pivots that are selected at each stage (that is, by increasing the value of the control parameter `ICNTL(5)`), KR is increased and greater efficiency can be gained from using `_GEMM` to perform (3.7). This is discussed further by Cliffe, Duff and Scott (1997) (see also Tables 4.8 to 4.11 in Section 4.4).

In MA62, the size of the blocks used in (3.7) is controlled by the parameter `ICNTL(7)`. In general, $\text{FRNT} - \text{KR} = (m - 1) * \text{ICNTL}(7) + r$ and the final block will have r columns. As a result of numerical experiments, we have chosen 16 as the default value for both `ICNTL(5)` and `ICNTL(7)`. The effects of using different values are illustrated in Section 4.4.

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

$$\begin{pmatrix} \mathbf{F}_{TL} \\ \mathbf{F}_L \end{pmatrix}. \quad (3.8)$$

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

$$\mathbf{W}_1 \leftarrow \mathbf{F}_{TL}^{-1} \mathbf{W}_1 \quad (3.9)$$

followed by

$$\mathbf{W}_2 \leftarrow \mathbf{W}_2 - \mathbf{F}_L \mathbf{W}_1 \quad (3.10)$$

and finally

$$\mathbf{W}_1 \leftarrow \mathbf{D}_T^{-1} \mathbf{W}_1 \quad (3.11)$$

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 leading dimension KR and the active variables of the solution vector \mathbf{X} are put into an array \mathbf{Z}_2 of leading dimension FRNT-KR. The operations

$$\mathbf{Z}_1 \leftarrow \mathbf{Z}_1 - \mathbf{F}_L^T \mathbf{Z}_2 \quad (3.12)$$

and then

$$\mathbf{Z}_1 \leftarrow \mathbf{F}_{TL}^{-T} \mathbf{Z}_1 \quad (3.13)$$

are carried out before \mathbf{Z}_1 is unloaded into \mathbf{X} .

In MA62, \mathbf{F}_L is written to the buffer by columns, and \mathbf{F}_{TL} is written to the buffer in packed form. Provided $\text{KR} > 1$, the forward elimination and back substitutions are performed using the Level 2 BLAS kernels `_GEMV` and `_TPSV` if there is only one right-hand side (`nrhs = 1`), and the Level 3 routine `_GEMM` and the Level 2 routine `_TPSV` if there are multiple right-hand sides. We observe that there is no Level 3 BLAS kernel for solving a triangular system of equations with the matrix held in packed form and multiple right-hand sides. We performed some numerical experiments in which we stored \mathbf{F}_{TL} as a full matrix with only the lower triangular part containing meaningful data and, in the forward elimination and back substitutions, we used the Level 2 kernel `_TRSV` if `nrhs = 1` and the corresponding Level 3 kernel `_TRSM` otherwise. Our results showed that the savings in the real storage which result in storing only the packed lower triangular matrix \mathbf{F}_{TL} are small (generally less than 10 per cent of the real storage requirement). This is illustrated in Table 4.12 in Section 4.5. The triangular solves (3.9) and (3.13) account for a relatively small part of the total solution time. Our experience is that there is little difference in the CPU times when `_TRSM` is used in place of `_TPSV`. As a result,

we have chosen in MA62 to minimize storage requirements by using the packed triangular form.

Although the factorization and solution stages described above are reasonably straightforward, their implementation within MA62 using Level 3 BLAS kernels is non-trivial. Elements are assembled into the frontal matrix \mathbf{FA} until there are at least $\text{ICNTL}(5)$ fully summed variables. The fully summed columns are then permuted to the last columns of \mathbf{FA} and, if supplied, the corresponding rows of the right-hand side array \mathbf{FRHS} are permuted to the end of the array. When an element is assembled into \mathbf{FA} , only the upper triangular part of \mathbf{FA} contains meaningful data, and symmetry must be exploited to perform the column interchanges. We remark that, by permuting the fully summed columns to the last columns of \mathbf{FA} , once the eliminations have been performed and the resulting data written to the in-core buffers \mathbf{BUFR} and \mathbf{IBUFR} , the last columns of \mathbf{FA} can be reset to zero and the next element assembled. If instead we were to permute the fully summed columns to the start of \mathbf{FA} , further costly data movement would be required.

After permuting the fully summed columns to the end, the pivots are generated in reverse order on the diagonal of the factors. We now discuss in detail what this means for the forward elimination operation (3.9). As already mentioned, the triangular parts of the factor matrix are held in packed form. We store the lower triangular matrix \mathbf{F}_{TL} by rows and store the negative of the pivot entries at the end of each row. Thus, if $\mathbf{KR} = 3$ and the lower triangular part of the matrix \mathbf{F}_{TL} is given by

$$\begin{pmatrix} piv_3 & & & & \\ a & piv_2 & & & \\ b & c & piv_1 & & \end{pmatrix},$$

it is stored as

$$-piv_3 \ a \ -piv_2 \ b \ c \ -piv_1.$$

In equation (3.9) we wish to compute

$$\begin{pmatrix} 1 & & & \\ c & 1 & & \\ b & a & 1 & \end{pmatrix}^{-1} \mathbf{W}_1, \quad (3.14)$$

but since we have the pivots in reverse order, we are computing

$$\tilde{\mathbf{P}}\mathbf{L}^{-1}\tilde{\mathbf{P}}^T\tilde{\mathbf{P}}\mathbf{W}_1,$$

where $\tilde{\mathbf{P}}$ is the reverse permutation $\{\text{KR}, \text{KR} - 1, \dots, 1\}$ and \mathbf{L} is the matrix in (3.14). Our coefficient matrix is therefore of the form

$$\begin{pmatrix} 1 & a & b \\ & 1 & c \\ & & 1 \end{pmatrix}^{-1}.$$

Thus, using the data held in the form above, in MA62 the packed triangular solve Level 2 BLAS routine `_TPSV` is called with its control parameters set to “Upper”, “Unit”, and “No transpose”.

3.4 Exploiting zeros in the front

During the factorization, the frontal matrix may contain some zero entries. Treating the frontal matrix as a dense matrix results in unnecessary operations being performed with these zeros. As we have shown, high level BLAS are used, so that the cost of these operations may not be prohibitive. If, however, the frontal matrix contains a significant number of zeros, it can be advantageous to exploit these zeros. In particular, there are many zeros in the front if the elements are poorly ordered. To see how we can exploit the zeros, suppose the frontal matrix has been permuted to the form (3.1). By performing further row and column permutations, the frontal matrix can be expressed in the form

$$\mathbf{F} = \begin{pmatrix} \mathbf{F}_T & \mathbf{F}_{C_1}^T & \mathbf{0} \\ \mathbf{F}_{C_1} & \mathbf{F}_{U_1} & \mathbf{F}_{U_2}^T \\ \mathbf{0} & \mathbf{F}_{U_2} & \mathbf{F}_{U_3} \end{pmatrix}, \quad (3.15)$$

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

$$\begin{pmatrix} \mathbf{F}_T & \mathbf{F}_{C_1}^T & \mathbf{0} \\ \mathbf{F}_{C_1} & \mathbf{F}_{U_1} & \mathbf{F}_{U_2}^T \\ \mathbf{0} & \mathbf{F}_{U_2} & \mathbf{F}_{U_3} \end{pmatrix} = \begin{pmatrix} \mathbf{F}_{TL} & \mathbf{0} & \mathbf{0} \\ \mathbf{F}_{L_1} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{pmatrix} \begin{pmatrix} \mathbf{D}_T & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \widehat{\mathbf{F}}_{U_1} & \mathbf{F}_{U_2}^T \\ \mathbf{0} & \mathbf{F}_{U_2} & \mathbf{F}_{U_3} \end{pmatrix} \begin{pmatrix} \mathbf{F}_{TL}^T & \mathbf{F}_{L_1}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{pmatrix}, \quad (3.16)$$

where

$$\mathbf{F}_{L_1} = \mathbf{F}_{C_1} (\mathbf{D}_T \mathbf{F}_{TL}^T)^{-1}, \quad (3.17)$$

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

$$\widehat{\mathbf{F}}_{U_1} = \mathbf{F}_{U_1} - \mathbf{F}_{L_1} \mathbf{D}_T \mathbf{F}_{L_1}^T. \quad (3.18)$$

If $\text{KR} > 1$, the matrix \mathbf{F}_{L_1} may still contain some zeros. However, the experiments which we report in Table 4.3 (Section 4.3) show that, in general, the number of zeros remaining in the factors is small (less than 10 per cent of the total number of entries in the factors). Furthermore, \mathbf{F}_{L_1} rather than \mathbf{F}_L ($\mathbf{F}_L = (\mathbf{F}_{L_1}^T, \mathbf{0})^T$), is written to the buffer, resulting in savings in both the real and integer factor storage.

In MA62, zeros in the front are exploited if the default value $\text{ICNTL}(9) = 1$ is used. With this setting, once an element has been assembled and the fully summed columns permuted to the last columns of the frontal matrix, row permutations are performed to collect the zeros in the fully summed columns into a block held in the first rows. We now explain how this is achieved in MA62. The number K of rows with zeros in all the pivotal columns is initially set to 0. Rows 1 to $\text{FRNT} - \text{KR} + 1$ 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 $\text{I} > 1$, we now scan row $\text{I}-1$. In the second case, we increment K by one and, starting with row K , we search for a row with index at most $\text{I}-1$ 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 $\text{I}-1$. Otherwise, there are no more rows to add to the zero block and we are ready to perform the elimination operations.

In general, when zeros in the front are exploited, the real and integer storage used to hold the factors will be less than that predicted by the symbolic factorization, which assumes that the frontal matrices are dense. The information on the storage actually used is returned to the user in the information array `INFO`. Zeros in the front are not exploited if the control parameter `ICNTL(9)` is set by the user to 0. Results which demonstrate how beneficial it can be to exploit zeros in the front are given in Section 4.3.

4 Numerical results

4.1 Test problems

Identifier	Order	Number of elements	Description/discipline
CEGB3306	3222	791	2.5D Framework problem
CEGB2919	2859	128	3D cylinder with flange
CEGB3024	2996	551	2D reactor core section
LOCK1074	1038	323	Lockheed gyro problem
LOCK2232	2208	944	Lockheed tower problem
LOCK3491	3416	684	Lockheed cross-cone problem
RAMAGE01	1476	128	3D Navier-Stokes
AEAC5081	5081	800	Double glazing problem
TRDHEIM	22098	813	Mesh of the Trondheim fjord
CRPLAT2	18010	3152	Corrugated plate field
OPT1	15449	977	Part of oil production platform
TSYL201	20685	960	Part of oil production platform
RAMAGE02	16830	1400	3D Navier-Stokes

Table 4.1: The test problems

In this section, we describe the problems that we use for testing the performance of **MA62**. In all cases, they arise in real engineering and industrial applications. The problems are all unassembled finite-element problems and a brief description of each is given in Table 4.1. The first seven problems are from the Harwell-Boeing Collection (Duff, Grimes and Lewis, 1992), the **RAMAGE01** and **RAMAGE02** problems were provided by Alison Ramage of the University of Strathclyde (Ramage and Wathen, 1993), the problem **AEAC5081** is from Andrew Cliffe of AEA Technology, and the remaining problems (**TRDHEIM**, **CRPLAT2**, **OPT1**, and **TSYL201**) were supplied by Christian Damhaug of Det Norske Veritas, Norway. For all the problems, values for the entries of the element matrices were generated using the Harwell Subroutine Library (HSL) random number generator **FA01** and each was made symmetric and diagonally dominant. In all the experiments, apart

from those reported on in Sections 4.2 and 4.3, the elements were ordered using the direct element reordering algorithm offered by the HSL routine MC43 before the frontal solvers were called.

All the HSL linear equation solvers used in our numerical experiments have control parameters with default values. Unless stated otherwise, we use these defaults in each case.

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

- A single processor of a CRAY J932 using the CRAY Fortran compiler cf77-7 with compiler option `-Zv` for the Fortran 77 codes and f90 with default options for the Fortran 90 codes.
- An IBM RS/6000 3BT using the IBM Fortran compilers xlf and xlf90, with compiler option `-O`.
- A DEC 7000 using the DEC Fortran compilers f77 and f90, with default compiler options.

On each machine the vendor-supplied BLAS were used. All times are CPU times in seconds and include the i/o overhead for the codes which use direct access files. In all the tables in which the number of floating-point operations (“ops”) are quoted, we count all operations (+, -, *, /) equally.

4.2 The effect of ordering

As explained in Section 2, the order in which the elements are presented to the frontal solver has a significant effect on its performance. If we denote by f_l the number of variables in the front before the l th elimination, then an important measure, particularly for computing the amount of in-core storage required, is the maximum front size

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

A prediction of the work involved in the frontal algorithm can be obtained from the root-mean squared front size (rms front size) defined by

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

Identifier	Max front size in MA62		rms front size in MA62		Number of ops (*10 ⁷)		Factorize time (seconds)	
	Before	After	Before	After	Before	After	Before	After
CEGB3306	366	90	248.3	65.5	3.1	1.7	0.6	0.4
CEGB2919	357	306	216.5	191.0	8.3	10.2	0.9	1.0
CEGB3024	162	146	113.7	90.0	2.7	2.6	0.5	0.5
LOCK1074	822	138	519.2	84.1	6.6	0.9	0.7	0.2
LOCK2232	1278	84	749.2	56.2	5.1	0.8	1.1	0.3
LOCK3491	846	231	582.6	138.7	51.2	5.3	3.9	0.8
RAMAGE01	470	377	345.6	279.3	18.3	12.1	1.4	1.1
AEAC5081	163	170	125.7	98.1	12.4	7.7	1.5	1.1
TRDHEIM	288	360	184.2	174.4	51.8	50.2	5.9	5.7
CRPLAT2	1570	550	1183.6	378.6	1918.9	262.4	124.4	20.3
OPT1	2688	995	2070.9	610.9	857.2	552.4	57.2	37.0
TSYL201	1209	549	862.7	511.6	1528.4	554.2	97.1	38.8
RAMAGE02	1730	1457	1498.1	1295.0	3811.9	2852.3	231.5	175.7

Table 4.2: The results of using the MC43 ordering with MA62 (CRAY J932). The root mean-squared front size is denoted by “rms front size”.

In Table 4.2, we show the results of using the element ordering code `MC43` with our symmetric positive-definite frontal solver, `MA62` (default settings are used for all control parameters). We note that the original order is the one provided by the application which, in most instances, the originator of the problem believed to be a “good” element ordering. For some problems, reordering with `MC43` gives a significant reduction in the maximum and rms front sizes and this is reflected in the reduced factorize times and operation counts. Having generated a new ordering, `MC43` compares the maximum front size of the new ordering with that of the original ordering, and then returns to the user the ordering with the smallest maximum front size. However, it is possible that by doing this `MC43` rejects the ordering with the smallest rms front size (which is generally the ordering that gives the smallest operation count and factorize time when used with the frontal solver). In all our experiments we have therefore made a minor alteration to `MC43` so that the ordering with the smallest rms front size is selected, even if the maximum front size is increased. We see the effect of this on test problems `AEAC5081` and `TRDHEIM`. We note that the effect of using Level 3 BLAS means that the poorer orderings have a higher Megaflop rate so that the ratio of times, before and after ordering, is not as high as the operation count ratio. For the problem `CEGB2919`, the maximum and rms front sizes are reduced by using `MC43` but the operation count and the factorize time are smaller for the original element ordering. This is because for this problem more zeros are exploited in the front for the original ordering than for the `MC43` ordering. If zeros are not exploited (`ICNTL(9)` is set to 0), the element ordering generated by `MC43` is more efficient than the original ordering (see Table 4.3 below).

4.3 The effect of exploiting zeros

We discussed, in Section 3.4, the option offered by `MA62` of exploiting zeros within the front. Zeros are ignored if `ICNTL(9) = 0` and are exploited if `ICNTL(9) = 1`. In Table 4.3, we show the effect of exploiting zeros. We have chosen some of the test problems which were initially not well ordered (see Table 4.2) and have run these problems with `ICNTL(9) = 0` and 1, both with and without preordering using `MC43`. We see that, in general, if the elements are not well ordered, substantial savings are achieved in the factor storage, the number of operations, and the factorize time by exploiting zeros in the front. Once the elements have been ordered, the savings which result from

Identifier	Ordered	ICNTL(9)	Real Storage (Kwords)	Integer Storage (Kwords)	Number of ops (*10 ⁷)	Factorize time (seconds)
CEGB3306	N	0	765 (661)	45	21.5	1.8
	N	1	218 (113)	14	3.1	0.6
	Y	0	207 (51)	14	1.7	0.4
	Y	1	204 (47)	14	1.7	0.4
CEGB2919	N	0	582 (99)	37	12.5	1.1
	N	1	403 (5)	17	8.3	0.9
	Y	0	467 (4)	24	10.2	1.0
	Y	1	446 (4)	20	10.2	1.0
LOCK1074	N	0	486 (324)	22	29.5	2.0
	N	1	219 (57)	10	6.6	0.7
	Y	0	86 (8)	5	0.9	0.2
	Y	1	85 (6)	5	0.9	0.2
LOCK2232	N	0	1418 (1282)	78	127.3	8.5
	N	1	206 (71)	13	5.1	1.1
	Y	0	122 (22)	8	0.9	0.3
	Y	1	114 (13)	8	0.8	0.3
LOCK3491	N	0	1804 (808)	100	118.4	8.1
	N	1	1062 (69)	59	51.3	3.9
	Y	0	456 (97)	27	7.3	0.9
	Y	1	380 (22)	23	5.3	0.8
CRPLAT2	N	0	20734 (3822)	1125	2545.9	163.3
	N	1	17053 (131)	940	1918.9	124.4
	Y	0	6556 (124)	37	267.4	20.4
	Y	1	6490 (59)	37	262.4	20.3
OPT1	N	0	29331 (19865)	1124	6357.8	384.8
	N	1	9628 (207)	355	857.2	57.2
	Y	0	8142 (214)	358	571.1	37.8
	Y	1	7984 (90)	341	552.4	37.0
TSYL201	N	0	16693 (248)	709	1546.3	97.9
	N	1	16521 (88)	701	1528.4	97.1
	Y	0	10416 (48)	485	554.2	38.6
	Y	1	10405 (48)	483	554.2	38.8

Table 4.3: The effect of exploiting zeros in the front (CRAY J932). If ICNTL(9) = 0, zeros are not exploited; if ICNTL(9) = 1, zeros are exploited. The figures in parentheses are the number of zeros (in thousands) which are held explicitly in the factors.

exploiting zeros are much smaller. Indeed, if the savings in the factor storage and operation counts are very small, the overheads of searching for zeros and increased data movement to accumulate the zeros into blocks can increase the factorize time. On the DEC 7000, we found that for the problem TSYL201 the factorize time without exploiting zeros was 95 seconds but this increased to 101 seconds when zeros were exploited. However, in general, the factorize and solve times are reduced by taking advantage of zeros, so we have chosen the default setting to be $\text{ICNTL}(9) = 1$.

4.4 The effect of blocking parameters

In this section, we examine how the performance of the frontal code MA62 is affected by the parameters which control the minimum pivot block size and the size of the blocks used in updating the frontal matrix ($\text{ICNTL}(5)$ and $\text{ICNTL}(7)$, respectively). In Tables 4.4 to 4.7 results are given for a subset of our test problems for a range of values of $\text{ICNTL}(7)$. In these tests we use minimum pivot block sizes of 1 (so that variables are eliminated as soon as they become fully summed) and 16. Zeros in the front are exploited ($\text{ICNTL}(9) = 1$). The timings quoted confirm that it is generally advantageous to exploit Level 3 BLAS, albeit at the cost of an increased operation count. As a result of our numerical experiments, we have chosen the default value for $\text{ICNTL}(7)$ in MA62 to be 16.

In Tables 4.8 to 4.11 results are given for the test problems CEGB3306, AEAC5081, CRPLAT2, and OPT1 for a range of values of $\text{ICNTL}(5)$ ($\text{ICNTL}(7) = 16$). It is apparent that modest increases in the minimum pivot block size have little effect on the size of the largest pivot block and on the maximum front size, and that the real storage requirement and the operation count grow slowly with $\text{ICNTL}(5)$. The factor times indicate that, in general, modest savings can be achieved by allowing $\text{ICNTL}(5)$ to be greater than 1 but the precise choice of the minimum pivot block size parameter does not appear crucial. This is important from a practical point of view since it is possible to get good performance without having to optimize the parameter from run to run. In general, the solve times are reduced by using $\text{ICNTL}(5) > 1$. This is because, by increasing $\text{ICNTL}(5)$, the integer data written to and read from the buffers is reduced and, as a result, the amount of data which must be copied from the partial solution matrix into the arrays used for direct addressing is reduced. Since the amount of data

Identifier	ICNTL(7)			
	(ICNTL(5) = 1)			
	1	8	16	32
CEGB3306	11.0	12.3	13.8	17.0
LOCK2232	4.3	4.9	5.8	7.7
LOCK3491	40.4	42.9	46.1	53.7
RAMAGE01	110.3	112.9	116.2	123.3
AEAC5801	60.9	64.7	69.5	80.4
TRDHEIM	458.7	473.1	491.5	532.2
CRPLAT2	2447.3	2492.2	2545.6	2658.0
	(ICNTL(5) = 16)			
CEGB3306	14.1	15.8	17.0	20.3
LOCK2232	6.4	7.1	8.0	10.3
LOCK3491	47.5	50.0	53.4	61.6
RAMAGE01	115.1	117.8	121.0	128.8
AEAC5801	68.7	72.5	77.3	88.3
TRDHEIM	469.7	483.6	502.1	542.7
CRPLAT2	2525.8	2570.7	2624.2	2736.6

Table 4.4: The number of operations required to perform the numerical factorization for different values of the blocking parameter ICNTL(7). The operation counts are in millions of floating-point operations.

Identifier	ICNTL(7) (ICNTL(5) = 1)			
	1	8	16	32
CEGB3306	0.59	0.52	0.51	0.53
LOCK2232	0.43	0.39	0.37	0.40
LOCK3491	1.21	1.03	1.01	1.01
RAMAGE01	1.39	1.21	1.19	1.21
AEAC5801	1.71	1.48	1.40	1.44
TRDHEIM	6.32	5.72	5.72	5.83
CRPLAT2	33.31	27.84	27.34	27.60
	ICNTL(5) = 16			
CEGB3306	0.49	0.48	0.45	0.48
LOCK2232	0.38	0.35	0.34	0.36
LOCK3491	0.93	0.85	0.84	0.88
RAMAGE01	1.21	1.08	1.08	1.09
AEAC5801	1.30	1.17	1.20	1.23
TRDHEIM	6.72	6.01	6.07	6.17
CRPLAT2	24.01	21.12	20.91	21.14

Table 4.5: The time in seconds for the numerical factorization for different values of the blocking parameter ICNTL(7) (CRAY J932).

Identifier	ICNTL(7) (ICNTL(5) = 1)			
	1	8	16	32
CEGB3306	0.41	0.33	0.30	0.27
LOCK2232	0.20	0.16	0.16	0.13
LOCK3491	1.17	0.88	0.93	0.94
RAMAGE01	1.78	1.82	1.79	1.79
AEAC5801	1.69	1.30	1.30	1.39
TRDHEIM	7.12	7.35	6.82	7.02
CRPLAT2	40.67	40.55	38.90	37.68

Table 4.6: The time in seconds for the numerical factorization for different values of the blocking parameter ICNTL(7) (IBM RS/6000).

Identifier	ICNTL(7) (ICNTL(5) = 1)			
	1	8	16	32
CEGB3306	0.84	0.82	0.78	0.80
LOCK2232	0.46	0.43	0.44	0.46
LOCK3491	1.89	1.79	1.77	1.80
RAMAGE01	3.07	2.65	2.47	2.50
AEAC5801	2.78	2.59	2.51	2.60
TRDHEIM	16.32	14.07	13.40	13.71
CRPLAT2	77.19	64.83	60.32	62.13

Table 4.7: The time in seconds for the numerical factorization for different values of the blocking parameter ICNTL(7) (DEC 7000).

copied is related to the number of right-hand sides, the time saved increases with the number `nrhs` of right-hand sides. On the basis of our numerical experiments on the different machines, in MA62 the default value for ICNTL(5) is 16. However, we note that if the user is going to perform a large number of solves, it may be beneficial to choose a value of ICNTL(5) larger than this default. However, since using ICNTL(5) > 1 reduces the repetition of the storage of the variable indices, increasing ICNTL(5) can give substantial savings in the integer storage used.

Identifier	ICNTL(5)	Largest pivot block	Maximum front size	Factor flops (*10 ⁶)	Storage (Kwords)	
					Real	Integer
CEGB3306	1	6	78	14	179	33
	8	12	84	15	191	19
	16	18	90	17	204	14
	32	36	108	22	236	8
	40	42	114	24	246	8
AEAC5081	1	10	156	69	530	90
	8	17	161	71	537	48
	16	25	170	77	562	33
	32	41	185	88	600	20
	40	49	195	95	626	18
CRPLAT2	1	24	544	2546	6383	1050
	8	24	544	2583	6435	540
	16	24	550	2624	6491	370
	32	42	568	2751	6655	195
	40	50	574	2795	6708	170
OPT1	1	38	983	5470	7971	503
	8	38	983	5473	7974	490
	16	39	995	5524	8019	353
	32	67	1009	5657	8137	226
	40	67	1009	5729	8195	193

Table 4.8: Storage statistics and operation counts for different values of the minimum pivot block size (ICNTL(5)).

Identifier	ICNTL(5)	Factor Time (seconds)	Solve Time (seconds)		
			nrhs=1	nrhs=2	nrhs=10
CEGB3306	1	0.5	0.07	0.10	0.28
	8	0.5	0.04	0.06	0.18
	16	0.4	0.03	0.05	0.14
	32	0.5	0.03	0.04	0.12
	40	0.6	0.03	0.04	0.12
AEAC5081	1	1.4	0.13	0.18	0.55
	8	1.1	0.09	0.12	0.37
	16	1.1	0.07	0.10	0.31
	32	1.3	0.06	0.08	0.27
	40	1.3	0.06	0.08	0.26
CRPLAT2	1	27.3	1.06	1.39	4.66
	8	22.9	0.76	0.98	3.20
	16	20.5	0.62	0.84	2.60
	32	21.1	0.56	0.72	2.24
	40	20.3	0.55	0.71	2.18
OPT1	1	39.2	0.81	1.02	3.27
	8	38.9	0.79	1.02	3.25
	16	37.0	0.70	0.91	2.80
	32	36.5	0.66	0.85	2.58
	40	36.4	0.63	0.82	2.55

Table 4.9: Factor and solve times for different minimum pivot block sizes on the CRAY J932. `nrhs` denotes the number of right-hand sides.

Identifier	ICNTL(5)	Factor Time (seconds)	Solve Time (seconds)		
			nrhs=1	nrhs=2	nrhs=10
CEGB3306	1	0.3	0.02	0.08	0.25
	8	0.3	0.01	0.06	0.16
	16	0.3	0.03	0.03	0.17
	32	0.4	0.03	0.03	0.18
	40	0.4	0.06	0.06	0.18
AEAC5081	1	1.3	0.10	0.13	0.49
	8	1.2	0.09	0.13	0.41
	16	1.2	0.08	0.11	0.33
	32	1.4	0.09	0.10	0.38
	40	2.0	0.07	0.11	0.36
CRPLAT2	1	38.9	1.13	1.54	7.94
	8	42.1	1.12	1.62	6.09
	16	32.2	0.95	1.46	4.60
	32	32.2	0.92	1.33	4.04
	40	32.2	0.87	1.44	3.82
OPT1	1	81.2	1.12	1.93	6.11
	8	81.4	1.17	1.80	5.89
	16	75.7	1.23	1.82	5.16
	32	75.6	0.94	1.74	4.65
	40	75.1	1.08	1.51	4.64

Table 4.10: Factor and solve times for different minimum pivot block sizes on the IBM RS/6000. nrhs denotes the number of right-hand sides.

Identifier	ICNTL(5)	Factor Time (seconds)	Solve Time (seconds)		
			nrhs=1	nrhs=2	nrhs=10
CEGB3306	1	0.7	0.36	0.40	0.60
	8	0.7	0.31	0.34	0.49
	16	0.8	0.34	0.38	0.50
	32	0.9	0.38	0.40	0.52
	40	0.9	0.38	0.40	0.52
AEAC5081	1	2.5	1.01	1.14	1.64
	8	2.3	0.88	0.99	1.34
	16	2.3	0.91	1.01	1.31
	32	2.4	0.94	1.03	1.31
	40	2.7	0.99	1.07	1.35
CRPLAT2	1	60.3	14.7	16.4	20.7
	8	51.5	13.6	14.9	17.5
	16	50.1	13.2	14.0	16.9
	32	50.0	12.8	14.9	16.6
	40	51.4	13.2	14.3	16.6
OPT1	1	97.6	16.4	18.6	21.5
	8	97.0	16.3	17.8	21.3
	16	94.6	15.9	17.7	20.7
	32	90.5	15.9	17.4	20.3
	40	90.2	16.4	17.4	20.1

Table 4.11: Factor and solve times for different minimum pivot block sizes on the DEC 7000. `nrhs` denotes the number of right-hand sides.

4.5 The performance of MA62 compared with MA42

Our aim in designing and developing MA62 was to produce a code that would be much more efficient than the general unsymmetric frontal solver MA42 when used to solve symmetric positive finite-element systems. To assess how successful we have been, in Table 4.12 we compare the storage requirements and the operation counts for the two codes. For MA62, we give the real storage requirements for storing the lower triangular matrix \mathbf{F}_{TL} in packed form and as a full matrix in which only the lower triangular entries are meaningful (see Section 3.3). In Tables 4.13, 4.14, and 4.15 we compare the timings for MA62 with those for MA42 on the CRAY J932, the IBM RS/6000, and the DEC 7000, respectively. The timings quoted for MA42 were obtained using the option of restricting pivoting to the diagonal. Without this option, the code checks more entries when searching for pivots but in our experiments, it had no significant effect on the factorization time. MA62 was run with the default settings for all the control parameters. Throughout the remainder of this paper, the ‘Analyse’ times quoted for both MA42 and MA62 include the time to order the elements using MC43, to perform the prepass of the integer data and to perform the symbolic factorization. The analyse times for both codes are essentially the same since both codes work only with the integer data.

For the factorization phase, we see from these tables that MA62 is always significantly faster than MA42 and, for the larger problems, MA62 can be more than twice as fast as MA42. The solve times for MA62 are almost always less than for MA42, particularly for a large number of right-hand sides. This reduction is a result of using a minimum pivot block size of 16 in MA62 ($\text{ICNTL}(5) = 16$) and also of exploiting zeros in the front. If we set $\text{ICNTL}(5) = 1$ and $\text{ICNTL}(9) = 0$ (zeros not exploited), the solve times for the two codes are similar. We remark that the HSL Release 12 version of MA42 does not have an option for specifying the minimum pivot block size, although a version of MA42 used by AEA Technology in their code NAMMU for groundwater flow calculations (Hartley, Jackson and Watson, 1996) does include this option. The current version of MA42 does not take advantage of zeros in the front but recent experiments (Scott, 1997) have shown that the performance of MA42, particularly in terms of the factor storage and operation count, can be enhanced by exploiting zeros. For example, for the problem LOCK3491, by exploiting the zeros, the real factor storage is cut from 882

Identifier	Code	Factor Storage (Kwords)		Factor ops (*10 ⁶)
		Real	Integer	
CEGB3306	MA42	375		22.6
	MA62	204	(231)	17.2
CEGB2919	MA42	1012		181.1
	MA62	445	(477)	101.9
CEGB3024	MA42	477		38.2
	MA62	236	(261)	26.3
LOCK1074	MA42	159		12.7
	MA62	85	(94)	8.9
LOCK2232	MA42	219		10.8
	MA62	114	(133)	8.9
LOCK3491	MA42	882		119.5
	MA62	380	(411)	73.1
RAMAGE01	MA42	759		219.6
	MA62	388	(406)	121.0
AEAC5801	MA42	1065		120.4
	MA62	562	(610)	77.3
TRDHEIM	MA42	6663		866.5
	MA62	2154	(2318)	502.1
CRPLAT2	MA42	12915		4980.1
	MA62	6490	(6644)	2674.6
OPT1	MA42	16674		11047.1
	MA62	7984	(8165)	5710.9
TSYL201	MA42	20905		10723.5
	MA62	10405	(10630)	5542.2
RAMAGE02	MA42	41792		55870.1
	MA62	20996	(21204)	28523.2

Table 4.12: A comparison of the operation count and storage requirements for MA42 and MA62 on symmetric positive-definite unassembled finite-element systems. The numbers in parentheses are the real storage needed if packed form for \mathbf{F}_{TL} is not used.

Identifier	Code	Time (seconds)			
		Analyse	Factorize	Solve	
				nrhs = 1	nrhs = 10
CEGB3306	MA42	0.2	0.7	0.07	0.28
	MA62	0.2	0.4	0.03	0.14
CEGB2919	MA42	0.1	1.7	0.07	0.26
	MA62	0.1	1.0	0.05	0.21
CEGB3024	MA42	0.2	0.8	0.09	0.36
	MA62	0.2	0.5	0.04	0.17
LOCK1074	MA42	0.1	0.3	0.02	0.10
	MA62	0.1	0.2	0.01	0.05
LOCK2232	MA42	0.3	0.5	0.05	0.18
	MA62	0.3	0.3	0.02	0.09
LOCK3491	MA42	0.2	1.6	0.11	0.43
	MA62	0.2	0.8	0.05	0.20
RAMAGE01	MA42	0.1	1.8	0.05	0.20
	MA62	0.1	1.0	0.04	0.14
AEAC5801	MA42	0.3	1.9	0.14	0.59
	MA62	0.3	1.1	0.07	0.30
TRDHEIM	MA42	0.6	8.8	0.49	2.00
	MA62	0.6	5.7	0.33	1.42
CRPLAT2	MA42	1.2	45.3	1.13	4.78
	MA62	1.2	20.3	0.62	2.60
OPT1	MA42	0.8	76.0	0.91	3.65
	MA62	0.8	37.0	0.70	2.80
TSYL201	MA42	0.7	71.8	1.00	3.88
	MA62	0.7	38.8	0.90	3.60
RAMAGE02	MA42	1.4	389.6	2.29	9.32
	MA62	1.4	175.7	1.66	6.58

Table 4.13: A comparison of MA42 and MA62 on symmetric positive-definite unassembled finite-element systems (CRAY J932)

to 751 Kwords; the integer storage is reduced from 151 to 123 Kwords; and the operation count is cut from 120×10^6 to 79×10^6 .

We also observe that the solution phase of the frontal solvers is much slower on the DEC 7000 than on the other machines. Further investigation shows that, on this machine, the overheads for out-of-core working are considerable and reading the factors from the direct access files accounts for most of the solve time. If direct access files are not used, the MA62 solve time for a single right-hand side for TRDHEIM is cut from 5.3 to 0.6 seconds. For TSYL201, the corresponding times are 22.5 and 2.4 seconds, respectively. On the DEC, the use of direct access files also has a significant effect on the factor time. If we hold the factors in-core, then the MA62 factor time for TRDHEIM reduces from 13.4 to 9.0 seconds and for TSYL201 it is cut from 101 to 70 seconds. Similarly, without the use of direct access files, the MA42 factorization time for TRDHEIM reduces from 26 to 14 seconds and for TSYL201 it is cut from 173 to 116 seconds.

4.6 A comparison of the frontal code MA62 with other HSL codes

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

The code MA27 uses the multifrontal algorithm of Duff and Reid (1982). In the analyse phase, pivots are selected from the diagonal using the minimum degree criterion. During the factorization, this pivot sequence may be modified to maintain numerical stability, and 2×2 diagonal block pivots can also be used. By this means, MA27 can stably factorize symmetric indefinite problems. However, if the matrix is known to be positive definite, the user can set a parameter in the calling sequence so that a logically simpler path in the code is followed. In all our tests using MA27, this option was used.

Our colleague John Reid at the Rutherford Appleton Laboratory is currently developing a variable-band code for the solution of systems of equations whose matrix is symmetric and positive-definite. It does no interchanges and takes advantage of variation in bandwidth. The code

Identifier	Code	Time (seconds)			
		Analyse	Factorize	Solve	
				nrhs = 1	nrhs = 10
CEGB3306	MA42	0.03	0.4	0.05	0.25
	MA62	0.04	0.3	0.03	0.17
CEGB2919	MA42	0.03	2.2	0.11	0.37
	MA62	0.01	1.6	0.04	0.23
CEGB3024	MA42	0.04	0.6	0.07	0.28
	MA62	0.03	0.4	0.01	0.17
LOCK1074	MA42	0.03	0.2	0.01	0.08
	MA62	0.02	0.1	0.01	0.05
LOCK2232	MA42	0.07	0.3	0.03	0.16
	MA62	0.05	0.2	0.02	0.08
LOCK3491	MA42	0.04	1.7	0.06	0.39
	MA62	0.05	1.0	0.04	0.23
RAMAGE01	MA42	0.02	2.6	0.07	0.28
	MA62	0.02	1.7	0.04	0.21
AEAC5801	MA42	0.07	1.8	0.11	0.53
	MA62	0.06	1.2	0.08	0.38
TRDHEIM	MA42	0.11	10.8	0.54	2.8
	MA62	0.12	7.4	0.32	1.7
CRPLAT2	MA42	0.26	58.3	1.5	8.4
	MA62	0.24	33.0	1.0	4.6
OPT1	MA42	0.18	113.6	1.4	7.4
	MA62	0.18	77.2	1.2	5.1
TSYL201	MA42	0.14	107.4	1.6	8.2
	MA62	0.16	68.5	1.6	7.6
RAMAGE02	MA42	0.28	508.4	3.8	19.7
	MA62	0.28	347.3	2.7	12.9

Table 4.14: A comparison of MA42 and MA62 on symmetric positive-definite unassembled finite-element systems (IBM RS/6000)

Identifier	Code	Time (seconds)			
		Analyse	Factorize	Solve	
				nrhs = 1	nrhs = 10
CEGB3306	MA42	0.04	1.0	0.4	0.6
	MA62	0.04	0.7	0.3	0.5
CEGB2919	MA42	0.02	4.2	0.9	1.2
	MA62	0.02	2.4	0.7	1.0
CEGB3024	MA42	0.04	1.6	0.5	0.9
	MA62	0.04	1.0	0.4	0.6
LOCK1074	MA42	0.02	0.6	0.2	0.3
	MA62	0.02	0.4	0.1	0.2
LOCK2232	MA42	0.05	0.8	0.2	0.4
	MA62	0.05	0.5	0.2	0.3
LOCK3491	MA42	0.05	3.5	0.9	1.4
	MA62	0.05	1.6	0.6	0.9
RAMAGE01	MA42	0.02	4.1	0.7	1.0
	MA62	0.02	2.4	0.6	0.8
AEAC5801	MA42	0.06	4.0	1.0	1.7
	MA62	0.06	2.2	0.9	1.3
TRDHEIM	MA42	0.10	26.2	7.0	9.4
	MA62	0.11	13.4	5.3	6.8
CRPLAT2	MA42	0.25	105.3	15.3	22.4
	MA62	0.26	50.1	13.2	16.9
OPT1	MA42	0.15	176.4	18.6	24.7
	MA62	0.15	94.6	15.9	21.7
TSYL201	MA42	0.12	173.4	23.1	30.0
	MA62	0.12	101.8	22.5	27.3
RAMAGE02	MA42	0.28	863.4	49.2	65.6
	MA62	0.28	435.6	46.1	57.8

Table 4.15: A comparison of MA42 and MA62 on symmetric positive-definite unassembled finite-element systems (DEC 7000)

Identifier	Code	Time (seconds)			Factor ops (*10 ⁶)	Storage (Kwords)	
		Analyse	Factorize	Solve		In-core	Factors
CEGB3306	MA27	0.4	0.6	0.03	2.1	114	81
	VBAN	0.5	1.2	0.05	11.6	133	187
	MA62	0.2	0.5	0.03	17.0	8	218
CEGB2919	MA27	1.9	3.6	0.04	57.3	590	384
	VBAN	2.6	4.0	0.08	110.6	402	526
	MA62	0.1	1.0	0.05	101.9	94	465
CEGB3024	MA27	0.6	1.0	0.04	7.1	175	146
	VBAN	0.8	1.3	0.05	17.7	106	219
	MA62	0.2	0.5	0.04	26.3	21	252
LOCK1074	MA27	0.3	0.5	0.01	4.9	109	71
	VBAN	0.4	0.5	0.02	6.0	29	77
	MA62	0.1	0.2	0.01	8.7	19	90
LOCK2232	MA27	0.4	0.6	0.02	2.7	133	83
	VBAN	0.5	0.6	0.03	4.6	11	99
	MA62	0.3	0.3	0.02	8.0	7	122
LOCK3491	MA27	0.9	2.0	0.04	20.2	336	254
	VBAN	1.3	3.0	0.07	65.4	322	428
	MA62	0.2	0.8	0.05	53.3	53	403
RAMAGE01	MA27	1.3	4.0	0.03	94.0	549	345
	VBAN	2.6	3.3	0.05	122.7	243	401
	MA62	0.1	1.0	0.04	121.0	142	405
AEAC5081	MA27	1.3	3.1	0.08	44.4	526	430
	VBAN	1.5	3.7	0.10	69.8	95	564
	MA62	0.3	1.1	0.07	77.3	29	594
TRDHEIM	MA27	10.8	17.7	0.27	211.0	2893	2002
	VBAN	15.3	19.6	0.46	459.0	798	2958
	MA62	0.6	5.6	0.33	502.1	130	2262
CRPLAT2	MA27	5.3	40.2	0.30	1623.8	4554	3815
	VBAN	9.4	54.9	0.77	2475.8	2276	6406
	MA62	1.2	20.3	0.62	2624.2	302	6527
OPT1	MA27	11.9	77.1	0.32	3648.9	7741	5975
	VBAN	20.7	74.2	0.86	4116.5	3315	7215
	MA62	0.8	37.0	0.70	5523.8	990	8325
TSYL201	MA27	13.6	90.0	0.40	4285.0	8922	7069
	VBAN	22.3	96.4	1.18	5262.0	2079	10231
	MA62	0.7	38.8	0.90	5542.2	301	10888
RAMAGE02	MA27	20.4	783.0	1.05	44988.9	30569	21297
	VBAN	39.1	338.3	2.16	29922.7	3692	21787
	MA62	1.4	175.6	1.66	28523.2	2123	21847

Table 4.16: A comparison of MA27, VBAN and MA62 on symmetric positive-definite finite-element systems (CRAY J932).

Identifier	Code	Time (seconds)		
		Analyse	Factorize	Solve
CEGB3306	MA27	0.09	0.15	0.01
	VBAN	0.14	1.60	1.28
	MA62	0.04	0.33	0.06
CEGB2919	MA27	0.31	2.54	0.05
	VBAN	0.49	6.14	2.97
	MA62	0.01	1.56	0.04
CEGB3024	MA27	0.11	0.30	0.02
	VBAN	0.17	1.80	1.23
	MA62	0.03	0.42	0.01
LOCK1074	MA27	0.04	0.16	0.01
	VBAN	0.07	0.75	0.54
	MA62	0.02	0.12	0.01
LOCK2232	MA27	0.08	0.18	0.02
	VBAN	0.11	0.73	0.58
	MA62	0.05	0.23	0.02
LOCK3491	MA27	0.16	0.82	0.04
	VBAN	0.30	4.32	2.50
	MA62	0.05	0.99	0.04
RAMAGE01	MA27	0.18	3.95	0.04
	VBAN	0.43	5.55	2.42
	MA62	0.02	1.68	0.04
AEAC5081	MA27	0.24	1.80	0.07
	VBAN	0.38	5.33	3.17
	MA62	0.06	1.35	0.08
TRDHEIM	MA27	2.02	10.83	0.28
	MA62	0.12	7.41	0.32
CRPLAT2	MA27	1.15	65.4	0.49
	MA62	0.24	33.0	1.06
OPT1	MA27	2.37	146.3	0.76
	MA62	0.18	77.2	1.19
TSYL201	MA27	2.27	169.3	0.89
	MA62	0.16	68.5	1.63
RAMAGE02	MA62	0.25	347.3	2.67

Table 4.17: A comparison of MA27, VBAN and MA62 on symmetric positive-definite finite-element systems (IBM RS/6000).

Identifier	Code	Time (seconds)		
		Analyse	Factorize	Solve
CEGB3306	MA27	0.07	0.14	0.01
	VBAN	0.10	0.64	0.11
	MA62	0.04	0.78	0.35
CEGB2919	MA27	0.30	1.99	0.05
	VBAN	0.36	4.05	0.31
	MA62	0.02	2.40	0.75
CEGB3024	MA27	0.09	0.30	0.02
	VBAN	0.13	0.85	0.16
	MA62	0.04	0.98	0.40
LOCK1074	MA27	0.04	0.19	0.01
	VBAN	0.05	0.29	0.06
	MA62	0.02	0.41	0.15
LOCK2232	MA27	0.06	0.16	0.01
	VBAN	0.08	0.27	0.06
	MA62	0.05	0.48	0.19
LOCK3491	MA27	0.15	0.78	0.03
	VBAN	0.22	2.50	0.26
	MA62	0.05	1.60	0.62
RAMAGE01	MA27	0.18	3.11	0.05
	VBAN	0.31	4.15	0.23
	MA62	0.02	2.41	0.65
AEAC5081	MA27	0.21	1.51	0.06
	VBAN	0.27	2.81	0.34
	MA62	0.06	2.28	0.92
TRDHEIM	MA27	1.98	9.3	0.43
	VBAN	2.40	18.8	2.06
	MA62	0.11	13.4	5.31
CRPLAT2	MA27	1.03	49.1	0.91
	VBAN	2.12	129.1	4.56
	MA62	0.26	52.1	13.2
OPT1	MA27	2.15	140.0	1.49
	VBAN	3.64	212.8	5.44
	MA62	0.15	93.7	15.9
TSYL201	MA27	2.43	167.9	1.78
	VBAN	4.62	212.7	7.24
	MA62	0.12	100.8	22.5
RAMAGE02	MA27	3.56	2970.0	5.81
	VBAN	8.58	2008.4	17.6
	MA62	0.28	435.7	46.1

Table 4.18: A comparison of MA27, VBAN and MA62 on symmetric positive-definite finite-element systems (DEC 7000).

optionally uses a direct access file to store the matrix factor. The intention is that the new code **MA55** will replace an older HSL code **MA36**. At present, the development code is written in Fortran 90 and it only uses Level 1 BLAS. A Fortran 77 version of **MA55** will be made available in the future. It is intended that the **MA55** code will use blocking and Level 3 BLAS. We have called the development code used in our experiments **VBAN** in the tables and in the following text.

The codes are compared on the CRAY J932, the IBM RS/6000, and the DEC 7000 in Tables 4.16 to 4.18. We were unable to run **VBAN** and **MA27** on the largest problems on the IBM RS/6000 because of insufficient memory. The elemental matrices are assembled before **MA27** and **VBAN** are called. The cost of this preprocessing is not included. Since the efficiency of **VBAN** depends upon the equations being ordered for a small profile, the assembled matrix is ordered using the HSL profile reducing code **MC40** prior to calling **VBAN**, and the time taken to do this is given as the “Analyse” time for **VBAN**. For **MA27**, the “Analyse” time is that taken to select the pivot sequence using the minimum degree criterion and prepare the data structures for the subsequent numerical factorization. It is interesting that this more complicated **MA27** analyse is usually faster than the reordering for **VBAN**. This highlighted for us some deficiencies in the **MC40** ordering code which we are now attempting to rectify. Similar deficiencies are also present in the **MC43** code but are masked because the direct element reordering algorithm works with the connectivity pattern of elements rather than variables.

Note that the “In-core” storage figures quoted in Table 4.16 are the minimum in-core storage requirements for performing the matrix factorization and solving the linear system $\mathbf{Ax} = \mathbf{b}$. This figure includes both real and integer storage. On the CRAY, both integers and reals are stored in 64-bit words, so the storage statistics given in Table 4.16 are just the sum of the number of real and the number of integer words needed. We remark that if this minimum in-core storage is used, the performance of the codes will often be considerably degraded since either a large number of data compressions must be performed or a large number of records written to direct access files. The length (in real words) of the in-core files used by both **MA62** and **VBAN** was 25000.

Our experiments show that, with the exception of some of the smaller problems on the DEC 7000, **MA62** requires less time for the factorization than **VBAN** although it needs more floating-point operations. In most cases, we see

that the minimum degree ordering as expected performs a much better job of reducing the number of entries in the factors than our “band” orderings; sometimes the factor storage for MA27 is about half that of the better of the other two codes. In general, the number of entries in the factors is slightly less for VBAN than for MA62. Both VBAN and MA62 store their factors in direct access files and so, as expected, usually require much less “In-core” storage than MA27. However, VBAN sometimes requires a lot more in-core storage than MA62. This will happen if there is just a single row of high bandwidth towards the end of the reordered matrix. For the simple variable-band scheme used by VBAN, this would require that many previous rows needed to update this be held in memory. The frontal code does not suffer from this problem; the only effect is to add one to the front size for most of the computation. One possible remedy is to develop better orderings for the variable-band scheme and this is currently being studied.

We again observe that on the DEC 7000 the overheads for out-of-core working are high. Since MA27 does not have these overheads, on this machine if the factors are held in direct access files, for some of the test problems (particularly the smaller problems) MA27 has the fastest factorize time and for all the problems it has the fastest solve time for a single right-hand side. We remark that MA27 is designed for assembled problems; we are not aware of any software which implements a multifrontal algorithm for symmetric problems and which accepts matrices in elemental form.

5 Conclusions

We have designed and developed a frontal code for solving systems of symmetric positive-definite unassembled finite-element equations. The code optionally uses direct access files to hold the matrix factors and makes full use of Level 3 BLAS in its innermost loop and in the solution phase. We have shown that, as well as needing approximately half the real storage for the matrix factors as the general frontal code MA42, the code can be more than twice as fast as MA42. Compared with other HSL codes, we have seen that the frontal method can provide a very powerful approach for the solution of large sparse systems. We notice that, although other approaches may result in much less fill-in, if the factors are held in direct access files, the frontal code generally requires far less main memory. The performance

of the factorization and solution phases of the frontal codes is significantly affected by the efficiency of the i/o. If the i/o does not add a large overhead, the frontal scheme is generally faster than other the approaches considered, although this conclusion may have to be modified when multifrontal and variable band codes for positive definite systems that exploit the Level 3 BLAS become available.

6 Availability of the code

MA62 is written in standard Fortran 77. The code will be included in Release 13 of the Harwell Subroutine Library. MA27 and MA42, as well as the ordering routines MC40 and MC43, are available in Release 12 of the HSL. Anybody interested in using any of these codes should contact the HSL Manager: Scott Roberts, AEA Technology, Building 552 Harwell, Didcot, Oxfordshire OX11 0RA, England, tel. +44 (0) 1235 434988, fax +44 (0) 1235 434136, email Scott.Roberts@aeat.co.uk, who will provide licencing information.

7 Acknowledgements

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

References

- K. A. Cliffe, I. S. Duff, and J. A. Scott. Performance issues for frontal schemes on a cache-based high performance computer. Technical Report RAL-TR-97-001, Rutherford Appleton Laboratory, 1997.
- J. J. Dongarra, J. DuCroz, I. S. Duff, and S. Hammarling. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Trans. Mathematical Software*, **16**(1), 1–17, 1990.
- I. S. Duff. MA32 - a package for solving sparse unsymmetric systems using the frontal method. Report AERE R10079, Her Majesty's Stationery Office, London, 1981.

- 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. K. Reid. MA27 – A set of Fortran subroutines for solving sparse symmetric sets of linear equations. Report AERE R10533, Her Majesty's Stationery Office, London, 1982.
- 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. The design of a new frontal code for solving sparse unsymmetric systems. *ACM Trans. Mathematical Software*, **22**(1), 30–45, 1996.
- 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.
- I. S. Duff, J. K. Reid, and J. A. Scott. The use of profile reduction algorithms with a frontal code. *Inter. Journal on Numerical Methods in Engineering*, **28**, 2555–2568, 1989.
- L.J. Hartley, C.P. Jackson, and S.P. Watson. NAMMU (Release 6.3) User Guide. Technical Report AEA-ES-0138, AEA Technology, 1996.
- 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.

- A. Ramage and A. J. Wathen. Iterative solution techniques for the Navier-Stokes equations. Technical Report AM-93-01, School of Mathematics, University of Bristol, 1993.
- J. A. Scott. Exploiting zeros in frontal solvers. Technical Report to appear, Rutherford Appleton Laboratory, 1997.

A Appendix: Specification Sheets

The specification sheets can be obtained on request from the authors