Technical Report
RAL-TR-97-001

Performance Issues for Frontal Schemes on a Cache-Based High Performance Computer

K A Cliffe I S Duff and J A Scott

January 1997

COUNCIL FOR THE CENTRAL LABORATORY OF THE RESEARCH COUNCILS

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-97-001

Performance issues for frontal schemes on a cache-based high performance computer[1]

K. A. Cliffe[2], I. S. Duff, and J. A. Scott

Abstract

We consider the implementation of a frontal code for the solution of large sparse unsymmetric linear systems on a high performance computer where data must be in the cache before arithmetic operations can be performed on it. In particular, we show how we can modify the frontal solution algorithm to enhance the proportion of arithmetic operations performed using Level 3 BLAS thus enabling better reuse of data in the cache. We illustrate the effects of this on Silicon Graphics Power Challenge machines using problems which arise in real engineering and industrial applications.

Keywords: unsymmetric sparse matrices, frontal solver, direct methods, finite-elements, BLAS, computational kernels.

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

---

[1] Current reports available by anonymous ftp from `matisa.cc.rl.ac.uk` (internet 130.246.8.22) in the directory `pub/reports`. This report is in file cdsRAL97001.ps.gz.

[2] Address: AEA Technology, Harwell, Didcot, Oxon OX11 0RA, England.

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

January 15, 1997.

Contents

1 Introduction

The frontal solution scheme (Irons, 1970, Hood, 1976, Duff, 1984, Duff and Scott, 1996*b*) is a technique for the direct solution of the linear systems of equations

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

where the $\mathtt{n} \times \mathtt{n}$ matrix $\mathbf{A}$ is large and sparse, and $\mathbf{B}$ is an $\mathtt{n} \times \mathtt{nrhs}$ ($\mathtt{nrhs} \geq 1$) matrix of right-hand sides. The method is a variant of Gaussian elimination and involves the factorization of a permutation of $\mathbf{A}$ which can be written as

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

where $\mathbf{P}$ and $\mathbf{Q}$ are permutation matrices, and $\mathbf{L}$ and $\mathbf{U}$ are lower and upper triangular matrices, respectively. The code MA42 developed by Duff and Scott (1996*b*) for the Harwell Subroutine Library (1996) uses a frontal scheme for solving systems of the form (1.1) with $\mathbf{A}$ unsymmetric. MA42 includes an option which allows the assembled matrix $\mathbf{A}$ to be input by rows. However, as illustrated by Duff and Scott (1996*a*), the power of the frontal scheme is more apparent when the matrix $\mathbf{A}$ comprises contributions from the elements of a finite-element discretization. That is, we can express $\mathbf{A}$ as the sum of elemental matrices

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

where $\mathbf{A}^{(l)}$ is nonzero only in those rows and columns that correspond to variables in the l-th element. We shall be concerned with this case in the following. Our aim is to study the performance of a frontal solver on a machine where data must be in the cache before being operated upon.

In Section 2, we discuss salient features of the frontal scheme. In particular, we show how the computation in MA42 is organized to exploit _GEMM, the Level 3 Basic Linear Algebra Subprogram (BLAS) (Dongarra, DuCroz, Duff and Hammarling 1990) that implements dense matrix-matrix multiplication. We show, in Section 3, how we can modify the frontal algorithm to obtain a factorization which requires a larger number of floating-point operations but which is richer in Level 3 BLAS. The main theme of this paper is to see how this trade-off works in practical applications.

We discuss the effect of a cache in Section 4 and indicate the effect of data reuse by looking at the performance of _GEMM on a Silicon Graphics Power Challenge machine. In Section 5, we illustrate the effects of exploiting Level 3 BLAS in the frontal solver through experiments on Power Challenge machines using practical problems. Numerical experiments on an IBM RS/6000 and on a CRAY J932 are also reported on.

Finally, in Section 6, we present some concluding remarks.

2 Frontal solution schemes

2.1 The use of BLAS in frontal schemes

A key feature of the frontal method is that the system matrix $\mathbf{A}$ is never assembled explicitly but the assembly and Gaussian elimination processes are interleaved, with each variable being eliminated as soon as its row and column are fully summed, that is, after the last occurrence in an elemental matrix $\mathbf{A}^{(l)}$. This allows all intermediate working to be performed in a full matrix, termed the *frontal matrix*, whose rows and columns correspond to variables that have not yet been eliminated but have appeared in at least one of the elements that have been assembled.

Using Fortran notation, the innermost loop of a typical frontal method for an elemental problem is of the form

```
do j = 1,frnt
   p1 = pr(j)
   if (p1 .ne. 0.0) then
      do i = 1,frnt
         fa(i,j) = fa(i,j) + pc(i)*p1
      end do
   end if
end do
```

where `fa` is the frontal matrix, pc is the pivot column, pr is the pivot row, and `frnt` is the order of the frontal matrix. This code represents a rank-one update to the matrix that can be performed using the Level 2 BLAS routine `_GER`. After the assembly of an element, if there are `k` fully summed variables which can be eliminated, then `k` calls to `_GER` can be made. However, as we shall illustrate in Section 5, the computation is made more efficient if we avoid updating the frontal matrix until all pivots for the current element have been chosen. If we delay the elimination operations in this way, the Level 3 BLAS routine `_GEMM` can be used. We now discuss in a little more detail how this is achieved in the Harwell Subroutine Library (HSL) code `MA42`.

After the assembly of an element, if the `k` fully summed variables are permuted to the leading rows and columns, the frontal matrix can be expressed in the form

$$\begin{pmatrix} \mathbf{F}_{11} & \mathbf{F}_{12} \\ \mathbf{F}_{21} & \mathbf{F}_{22} \end{pmatrix}, \tag{2.1}$$

where $\mathbf{F}_{11}$ is a square matrix of order `k`. The rows and columns of $\mathbf{F}_{11}$, the rows of $\mathbf{F}_{12}$, and the columns of $\mathbf{F}_{21}$ are fully summed; the variables in $\mathbf{F}_{22}$ are not yet fully summed. Pivots may be chosen from anywhere in $\mathbf{F}_{11}$. The columns of $\mathbf{F}_{11}$ are searched for a pivot and, when chosen, the pivot row and column are permuted to the first row and column of (2.1). Row 1 of the permuted matrix $\mathbf{F}_{11}$ is scaled by the pivot and columns 2 to `k` of the permuted frontal matrix are updated by `k` $-$ 1 calls to the Level 1 BLAS routine `_AXPY`. Columns 2 to `k` of the updated matrix $\mathbf{F}_{11}$ are then searched for the next pivot. When chosen, the pivot row and column are permuted to row 2 and column 2 of (2.1), row 2 of $\mathbf{F}_{11}$ 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 pivots have been chosen, $\mathbf{F}_{12}$ is then updated using the Level 3 BLAS routine _TRSM

$$\mathbf{F}_{12} \leftarrow -\mathbf{F}_{11}^{-1}\mathbf{F}_{12}, \tag{2.2}$$

and, finally, $\mathbf{F}_{22}$ is updated using the Level 3 BLAS routine _GEMM

$$\mathbf{F}_{22} \leftarrow \mathbf{F}_{22} + \mathbf{F}_{21}\mathbf{F}_{12}. \tag{2.3}$$

In practice, for a general matrix $\mathbf{A}$, stability restrictions may only allow r pivots to be chosen ($\mathtt{r} < \mathtt{k}$) and, in this case, the first r rows of $\mathbf{F}_{12}$ are updated using _TRSM and then the remaining k - r rows of $\mathbf{F}_{12}$, together with $\mathbf{F}_{22}$ are updated using _GEMM. Further details of how this strategy is implemented within the frontal code MA42 are given by Duff and Scott (1996*b*).

Once all the eliminations have been performed, the upper triangular part of $\mathbf{F}_{11}$ (which we denote by $\mathbf{F}_U$) and $\mathbf{F}_{12}$ are stored for the **UQ** factor and the lower triangular part of $\mathbf{F}_{11}$ (denoted by $\mathbf{F}_L$) and $\mathbf{F}_{21}$ are stored for the **PL** factor. The triangular matrices $\mathbf{F}_U$ and $\mathbf{F}_L$ are held in packed form. To exploit the block structure, MA42 uses direct addressing in the solution phase. At each stage of the forward elimination, all the active components of the partial solution matrix $\mathbf{Y}$ (where $(\mathbf{PL})\mathbf{Y} = \mathbf{B}$) are put into an array $\mathbf{W} = (\mathbf{W}_1, \mathbf{W}_2)^T$, with $\mathbf{W}_1$ of dimension frnt - r by nrhs and $\mathbf{W}_2$ of dimension r by nrhs, where frnt is the current front size, r is the number of pivots chosen and nrhs of the number of right-hand sides which are being solved (the second dimension of $\mathbf{B}$). The operations

$$\mathbf{W}_2 \leftarrow -\mathbf{F}_L^{-1}\mathbf{W}_2 \tag{2.4}$$

followed by

$$\mathbf{W}_1 \leftarrow \mathbf{W}_1 + \mathbf{F}_{21}\mathbf{W}_2 \tag{2.5}$$

are performed before $\mathbf{W}$ is unloaded into $\mathbf{Y}$. Similarly, during the back substitution, all the active components of the partial solution matrix $\mathbf{Y}$ are put into an array $\mathbf{Z}_1$ of leading dimension r and the active variables of the solution matrix $\mathbf{X}$ are put into an array $\mathbf{Z}_2$ of leading dimension frnt - r. The operations

$$\mathbf{Z}_1 \leftarrow \mathbf{Z}_1 - \mathbf{F}_{12}\mathbf{Z}_2 \tag{2.6}$$

and then

$$\mathbf{Z}_1 \leftarrow \hat{\mathbf{F}}_U^{-1}\mathbf{Z}_1 \tag{2.7}$$

are carried out before $\mathbf{Z}_1$ is unloaded into $\mathbf{X}$ ($\hat{\mathbf{F}}_U$ is the triangular matrix $\mathbf{F}_U$ with units on the diagonal). Provided $\mathtt{r} > 1$, the forward elimination and back substitution are performed using the Level 2 BLAS kernels _GEMV and _TPSV if there is only one right-hand side ($\mathtt{nrhs} = 1$), and the Level 3 routine _GEMM and the Level 2 routine _TPSV if there are multiple right-hand sides (there is no Level 3 BLAS kernel for solving a triangular system of equations with the matrix held in packed form and multiple right-hand sides). We remark that the interior dimension in the call to _GEMM (or _GEMV) is r during the forward elimination and frnt - r during the back substitution. At most stages of the solution phase, frnt - r is larger than r and, in general, the Mflop rate for the forward elimination is therefore lower than for the back substitution.

2.2 The effect of reordering

The order of the frontal matrix increases when a variable appears for the first time and decreases when it is eliminated. Consequently the order in which the elements are assembled has a crucial effect on the performance of the frontal solver. Ordering routines have been developed for frontal solvers and use similar logic to bandwidth minimization. The HSL code `MC43` offers the user the choice of basing the ordering on the element structure or on the usual sparse matrix pattern (Duff, Reid and Scott, 1989). These two approaches are termed direct and indirect element reordering, respectively. The results presented by Duff et al. (1989) show that there is little difference in the quality of the ordering from the two approaches but, as observed by Duff and Scott (1996*a*), the former is generally faster if the problem has fewer elements than nodes. In the numerical experiments reported on in Section 5, the direct element reordering algorithm is used.

2.3 The use of direct access files

Another principal feature of the frontal method is that by holding the **PL** and **UQ** factors in direct access files, large problems can be solved using a relatively small amount of in-core memory. A lower bound on the in-core memory required can be obtained by performing a symbolic factorization, which is an option offered by the code `MA42`. This is only a lower bound because numerical pivoting during the factorization may increase the memory requirements. `MA42` uses three direct access files, one each for the reals in **PL** and **UQ** and one for the row and column indices of the variables in the factors. Corresponding to each of the direct access files is a buffer (or workspace), which is held in-core. During the factorization, each time a block of pivots is chosen and the frontal matrix (2.1) updated, a record is written to each of the buffers. Once a buffer becomes full (or the final eliminations have been performed), it is written to the appropriate direct access file. Use of direct access files is not needed if sufficient in-core storage is available.

In the integer buffer, each record holds lists of the (global) row and column indices of the variables in the front. Each variable enters and leaves the front once only. By storing the row and column indices of all the variables in the front in each record, more integer storage than necessary is used by `MA42`. In practice, the repetition of the storage of variable indices in `MA42` does not require a prohibitively large amount of storage because, as explained earlier, blocks of pivots are used and a record is only written once a block of pivots has been chosen. In our experience, for elemental problems the required integer storage is in the range $15n$ to $50n$ and the number of integers stored is less than a quarter the number of reals stored (detailed results are given by Duff and Scott, 1993 and in Section 5 below).

3 Modification for Level 3 BLAS enrichment

We saw, in Section 2.1, that if $\mathbf{r}$ pivots are chosen after the assembly of an element into the frontal matrix, the code `MA42` uses the Level 3 BLAS routine `_GEMM` with interior dimension $\mathbf{r}$ to update the frontal matrix prior to the next element assembly. If $\mathbf{r}$ is small, there may be little advantage gained by using Level 3 BLAS. We can increase the Level 3 BLAS component by delaying updating the frontal matrix until the number of pivot candidates is at least some prescribed minimum, say r_{min}. Suppose, at some stage, that the number of fully summed variables is k, then the maximum number of pivots which we can choose is k. If $k < r_{min}$ and not all the elements have been assembled, we do not look for pivots but repeat the process of assembling another element into the frontal matrix until either the number of fully summed variables exceeds r_{min} or there is insufficient storage allocated for the frontal matrix to accommodate the next element. We then go ahead and choose as many pivots as possible and update the frontal matrix, before assembling the next element.

Delaying the search for pivots until the number of fully summed variables is at least r_{min} ($r_{min} > 1$) will have several effects on the factorization. Firstly, the total number of calls to the Level 3 BLAS routine _GEMM will decrease but the average interior dimension will increase since, on most of the calls, the interior dimension will be at least r_{min} (numerical considerations may prevent all the potential pivots from being chosen). Secondly, when looking for pivots there will generally be a larger number of fully summed variables to test as potential candidates. Once a pivot is chosen, each of the fully summed columns not yet selected as a pivot column is updated using the Level 1 BLAS routine _AXPY. Therefore, the number of calls to _AXPY will increase. This increase can be restricted by making greater use of Level 2 BLAS. We now explain how this can be achieved. As discussed in Section 2.1, once MA42 has chosen a pivot, all the remaining fully summed columns are updated using calls to _AXPY, and then these columns are searched in turn for the next pivot. The process is repeated until no further pivots can be found. An alternative approach is to delay updating the i-th fully summed column until it is to be searched for a possible pivot. Assuming columns 1 to $i-1$ have already been successfully used as pivot columns, column i is updated using the Level 2 BLAS routines _TRSV and _GEMV. There is a problem with this approach if column i is updated using the Level 2 BLAS kernels and then is found to be unsuitable for use as a pivot column. In this case, column $i+1$ is updated using _TRSV and _GEMV and then searched. If column $i+1$ is chosen as a pivot column, column i is again updated, but since it has already been updated for the first $i-1$ pivots, _AXPY is used to perform a single update, and then column i is retested. Keeping track of which of the fully summed columns have been updated by which of the pivot columns adds to the complexity of this approach. It also requires that the fully summed columns are permuted to lie in a block before the search for pivots begins, whereas MA42 limits the amount of swapping of rows and columns by holding the positions of the fully summed variables and delaying permuting the pivot rows and columns into a block until all the pivots following an assembly have been chosen. Furthermore, since our numerical experiments show that the cost of the calls to the Level 1 BLAS kernels is much less than the total cost of the Level 3 BLAS calls (see Table 5.4), using Level 2 BLAS in place of Level 1 BLAS would not have a dramatic effect on the total factorization time and so we have not used the Level 2 BLAS implementation in our numerical experiments.

Performing additional assemblies before choosing pivots will lead to an increase in the average and maximum front sizes. The number of operations used to perform the matrix factorization will also rise, with many operations being performed on zeros. The real storage required to hold the matrix factors will increase but, since fewer records will be written to the buffers, the repetition of the storage of the row and column indices will be reduced and the integer storage will consequently decrease.

There will also be effects on the solution phase. In the forward elimination, if `nrhs` > 1 (respectively, `nrhs` $= 1$), the interior dimension of the calls to `_GEMM` (respectively, `_GEMV`) will increase. The interior dimension for the back substitution is `frnt - r`, where `frnt` is the order of the frontal matrix and `r` the number of pivots chosen. Our new strategy will lead to an increase in `frnt` and in `r` although, in general, the increase in `frnt` will be greater than the increase in `r`. Therefore, at most stages of the back substitution, the interior dimension will also increase. During the forward elimination and back substitution there will be a smaller number of calls to the Level 2 routine `_TPSV`, but the order of the matrix in each call will increase. Fewer records will be written to the buffers and, as a result, the time taken by the use of direct addressing during the solution phase will decrease. Since the amount of data which must be copied from the partial solution matrix into the arrays used for direct addressing is related to the number of right-hand sides, the time saved will depend on the number of right-hand sides, `nrhs`.

4 The reuse of cache

In this section, we discuss the performance of BLAS kernels on cache-based machines. We present a very simple model for such machines with a multiply-add pipe and derive a formula that gives an upper bound on the performance of the Level 3 BLAS routine `DGEMM` in terms of a number of parameters that characterize the machine. This result is compared with the observed performance of a Silicon Graphics Power Challenge XL with 75 MHz R8000 processors.

In our model, we assume that the machine has a clock speed of C MHz and that, if data is in the cache, f floating-point multiply-add pairs can be performed in each clock period. We also suppose that the size of the cache line is c words and that the latency of the cache is l clocks. We assume that the memory to cache operations cannot be overlapped with the floating-point operations (the cache is a blocking cache), although after the first word of the cache line is accessed computation can be overlapped with the transfer of subsequent words into the cache line.

Now consider using the Level 3 BLAS routine `_GEMM` to perform the operation

$$\mathbf{C} \leftarrow \alpha\mathbf{AB} + \beta\mathbf{C}, \tag{4.1}$$

where $\mathbf{A}$ and $\mathbf{B}$ are matrices of order $m \times r$ and $r \times m$, respectively. We are interested in the case where $m \gg r$ and m is sufficiently large that $\mathbf{C}$ will not fit in the cache.

The number of operations required by (4.1) is rm^2 floating-point multiply-add pairs plus a further $m^2 + mr$ floating-point multiplications. The total number of memory to cache operations is $m^2 + 2mr$. In practice, this is likely to be an underestimate because it may be necessary to load $\mathbf{A}$ and/or $\mathbf{B}$ from memory to cache several times during the operation. Thus the estimate we derive here for the speed of the operation will be greater than that actually observed.

The time (in clocks) taken for the memory to cache operations is

$$(m^2 + 2mr)l/c.$$

The time (in clocks) taken for the floating-point operations is

$$(rm^2 + m^2 + mr)/f.$$

We then estimate the speed of _GEMM (in Mflops) to be

$$C((2r+1)m^2 + mr)/[(m^2 + 2mr)l/c + ((r+1)m^2 + mr)/f].$$

That is,

$$fC((2r+1)m^2 + mr)/[m^2(lf/c + r + 1) + mr(2lf/c + 1)].$$

Using our assumption $m \gg r$, this simplifies to

$$2fC(r + 1/2)/(lf/c + r + 1).$$

For the Power Challenge workstation with 75 MHz R8000 processors and using double precision arithmetic the parameters have the following values:

$$\begin{aligned} C &= 75, \\ f &= 2, \\ c &= 16, \\ l &\approx 56. \end{aligned}$$

This leads to an approximate speed of $300(r + 1/2)/(r + 8)$ Mflops for the DGEMM operation with interior dimension r. In Figure 1 the estimated and observed speeds of DGEMM (in Mflops) are plotted against the interior dimension r. For these results, $m = 1000$ was used.

Using similar analysis, we can estimate the speed of a rank-one update (DGER) to be $300/8 = 37.5$ Mflops. Note that this is less than the estimated speed of 50 Mflops which is given by our DGEMM formula with $r = 1$.

5 The performance of the modified frontal code

In this section, we illustrate the effects of using the Level 3 BLAS enriched version of the frontal code MA42 when solving a range of problems arising from real engineering and industrial applications. We first present results for two examples which arise from groundwater flow calculations undertaken by AEA Technology. Although practical applications can often call for significantly larger models, these problems are typical of the problems which AEA Technology wants to solve using its code NAMMU (Hartley, Jackson and Watson, 1996). NAMMU uses a frontal solver and it is important that the frontal solver is as efficient as possible. The first problem, GFLOW2D, is a two-dimensional coupled groundwater flow salt transport calculation. The problem has 20000 9-noded quadrilateral elements with a total of 80200 degrees of freedom. The second problem, GFLOW3D, is a three-dimensional

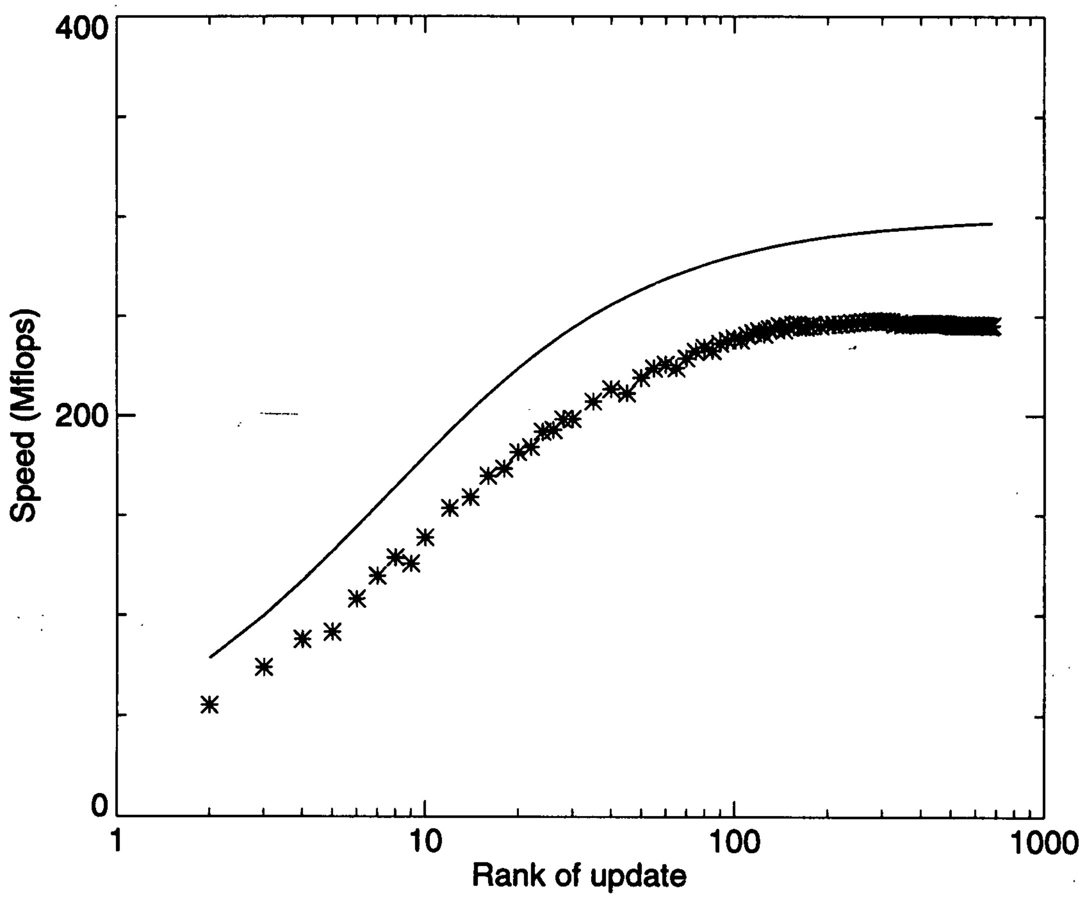

Figure 1: The estimated (continuous line) and observed speeds (stars) of DGEMM as a function of the interior dimension (rank of update) on an SGI Power Challenge workstation.

groundwater flow problem with pressure interpolated using a mixture of 27-noded triquadratic brick elements and 18-noded prism elements. The problem has 8820 elements with 73943 degrees of freedom. The results quoted in Table 5.1 were obtained using a Silicon Graphics Power Challenge XL with 4 75MHz R8000 processors and a cache size of 4 Mbytes, running IRIX 6.2. All runs were performed on a single processor using double precision arithmetic and the vendor-supplied BLAS.

In all the tables of results in this section in which the number of floating-point operations ("flops") are quoted, we count all operations (+,-,*,/) equally and assume that there are no zeros in the frontal matrices. All CPU timings are given in seconds.

It is clear from the results presented in Table 5.1 for $r_{min} = s$ and $r_{min} = 1$ that there are considerable benefits to be gained from the standard MA42 strategy of delaying the elimination of pivots until all possible pivots following an assembly have been chosen. The benefits are greater for the three dimensional problem than for the two dimensional problem. The reason for this is that each of the three dimensional elements has significantly more degrees of freedom. This means that the number of

Identifier	r_{min}	Maximum front size	Largest pivot block	Factor flops ($*10^{10}$)	Factor time (seconds)
GFLOW2D	s	308	1	1.46	208
	1	309	7	1.46	129
	10	318	14	1.50	111
	15	323	20	1.52	108
	20	328	26	1.56	109
	30	338	37	1.60	112
	40	348	44	1.67	115
GFLOW3D	s	1636	1	16.7	6369
	1	1636	26	16.7	1688
	10	1641	26	16.8	1264
	20	1651	43	16.9	1119
	40	1675	59	17.1	1057
	80	1702	105	17.7	1058
	320	1936	345	21.5	4150

Table 5.1: Performance of different pivot block sizes for groundwater flow problems. $r_{min} = s$ denotes all pivot blocks are of size 1.

variables which become fully summed at each stage tends to be larger, resulting in larger pivot blocks and better performance of the BLAS kernel _GEMM when updating the frontal matrix.

It is also clear there are additional benefits to be gained from the Level 3 BLAS enrichment modification. In both two and three dimensions, the operation count and the total factorization time does not appear to be very sensitive to the value of r_{min}. This suggests that, in practice, it is not necessary to choose the value carefully and it is likely that good performance will be achieved for a wide range of problems with values for r_{min} of about 15 and 40 for two and three-dimensional problems, respectively.

We now present, in more detail, results for test problems from a range of other application areas. A brief description of each of the problems is given in Table 5.2. For these problems only the sparsity pattern of the matrix was available and values for the matrix entries were generated using the Harwell Subroutine Library pseudo-random number generator FA04. The experimental results in Tables 5.3 and 5.4 were obtained on a 6 processor Silicon Graphics Power Challenge with the MIPS R10000 chip running at 195 MHz. The runs were performed on a single processor and again double precision arithmetic and the vendor-supplied BLAS were used. In each case, the elements were preordered using MC43 before the frontal solver was used.

In Table 5.3, the size of the largest pivot block used, the maximum front size, the total number of floating-point operations for factorizing the matrix, and the real and integer storage are shown for $r_{min} = s$ and for values of r_{min} in the range 1 to 40. The real storage is for holding both the **PL** and the **UQ** factors (although, in practice, MA42 only requires **PL** to be stored if the user wishes to solve for subsequent right-hand sides or to solve transpose systems $\mathbf{A}^T\mathbf{X} = \mathbf{B}$). It is apparent that modest increases in r_{min} have little effect on the size of the largest pivot block and on the

Identifier	Order	Number of elements	Description/discipline
RAMAGE02	16830	1400	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

Table 5.2: The test problems

maximum front size, and that the real storage requirement and the operation count grow slowly with r_{min}. However, since large values of r_{min} reduce the repetition of the storage of the row and column indices, increasing r_{min} can give substantial savings in the amount of integer storage used. Conversely, if only single pivots are chosen ($r_{min} = s$), there is much repetition in the integer storage.

Table 5.4 presents the CPU times for the calls to the Level 1 and Level 3 BLAS kernels, and the total time for the matrix factorization, together with the time taken to solve for 1, 2, and 10 right-hand sides. The total factorization time and the solve times include all the overheads for the out-of-core working. We again observe that if no Level 3 BLAS are used ($r_{min} = s$), the factorization is significantly more expensive than if the frontal matrix is updated at each stage using as many pivots as are available (that is, as in the standard version of MA42, $r_{min} = 1$). In the latter case, the calls to the Level 1 BLAS kernels account for a small part of the total factorization cost. As r_{min} is increased, the Level 1 BLAS account for a larger proportion of the factorization time until a point is reached where the savings in the Level 3 BLAS time is more than offset by the increase in the Level 1 BLAS time. The value of r_{min} at which this occurs is problem-dependent, but our results suggest that, in general, it is advantageous to use a value of about 16. However, if we want to solve for a large number of right-hand sides, it can be beneficial to use an even larger value of r_{min}.

The results in Table 5.4 were all obtained on an SGI Power Challenge machine. We have also performed some experiments on a subset of our test problems on an IBM RS/6000 3BT and on a single processor of a CRAY J932. The results are given in Tables 5.5 and 5.6, respectively. In each case, the vendor-supplied BLAS are used. We see that, on the RS/6000, there are considerable savings to be made by not forcing all pivot blocks to be of size 1, and further modest savings in the factorization and solve times can result from choosing r_{min} to be greater than 1. The Level 1 BLAS perform well on the CRAY and this is reflected in our results since, on this machine, the difference between the times for factorizing the matrix with $r_{min} = s$ and $r_{min} \geq 1$ are less significant. However, because of the significant savings in both the time taken to read the integer data from the direct access file and the time used by the direct addressing in the solution phase, the solve times are substantially reduced by allowing $r_{min} \geq 1$.

Identifier	r_{min}	Largest pivot block	Maximum front size	Factor flops ($*10^6$)	Storage (Kwords) Real	Integer
RAMAGE02	s	1	1453	55910	41808	41892
	1	32	1453	55910	41808	3496
	8	30	1453	55952	41826	3128
	16	45	1458	56462	42033	1702
	32	54	1474	57082	42275	1074
	40	54	1484	57392	42397	912
AEAC5081	s	1	154	202	1431	1456
	1	12	154	202	1431	243
	8	16	157	205	1441	129
	16	26	166	223	1502	86
	32	42	182	245	1573	58
	40	50	190	264	1630	53
TRDHEIM	s	1	277	961	7551	5232
	1	36	277	961	7551	597
	8	36	277	961	7551	597
	16	42	289	985	7661	550
	32	61	308	1073	8039	469
	40	68	315	1128	8248	452
CRPLAT2	s	1	538	5065	13012	13089
	1	19	539	5065	13012	2133
	8	24	545	5141	13116	1101
	16	27	550	5221	13225	754
	32	44	568	5466	13553	399
	40	49	574	5552	13662	346
OPT1	s	40	984	10764	16466	16215
	1	40	984	10764	16466	1190
	8	39	984	10771	16471	1163
	16	45	996	10875	16573	863
	32	59	1012	11204	16800	628
	40	68	1016	11268	16939	565
TSYL201	s	62	543	10741	20919	20925
	1	62	543	10741	20919	1021
	8	62	543	10743	20921	1017
	16	62	551	10759	20944	985
	32	61	572	11202	21369	541
	40	73	579	11257	21424	534

Table 5.3: Storage requirements for different pivot block sizes. $r_{min} = s$ denotes all pivot blocks are of size 1.

Identifier	r_{min}	Factor Time (seconds) Total	BLAS 3	BLAS 1	Solve Time (seconds) nrhs=1	nrhs=2	nrhs=10
RAMAGE02	s	2845.7	0.00	2724.80	14.50	17.98	47.69
	1	547.8	433.56	5.14	9.76	10.98	18.22
	8	527.4	411.26	5.42	9.89	10.26	17.77
	16	447.1	326.42	11.21	9.47	10.01	16.05
	32	422.5	292.18	19.18	9.60	10.38	15.51
	40	442.4	300.11	25.66	9.84	10.52	15.32
AEAC5081	s	3.5	0.00	2.26	0.63	0.77	1.64
	1	1.6	0.97	0.08	0.20	0.23	0.47
	8	1.5	0.85	0.10	0.17	0.20	0.42
	16	1.6	0.85	0.18	0.17	0.19	0.38
	32	1.9	0.91	0.32	0.17	0.18	0.34
	40	2.0	0.96	0.38	0.17	0.19	0.35
TRDHEIM	s	17.3	0.00	10.75	2.49	3.05	6.33
	1	7.8	3.82	0.49	1.41	1.49	2.42
	8	7.7	3.84	0.47	1.39	1.40	2.40
	16	7.7	3.81	0.61	1.14	1.24	2.17
	32	8.3	3.73	1.21	1.21	1.30	2.12
	40	8.9	3.76	1.60	1.24	1.33	2.21
CRPLAT2	s	235.6	0.00	212.00	5.02	5.86	15.71
	1	57.0	46.52	0.54	2.94	2.83	5.44
	8	43.4	31.64	0.96	2.69	2.85	4.48
	16	40.3	28.20	1.47	2.32	2.71	4.25
	32	38.6	24.97	3.12	2.07	2.21	3.58
	40	38.2	24.26	3.76	2.07	2.22	3.49
OPT1	s	538.4	0.00	493.86	5.98	7.27	19.28
	1	92.7	71.24	2.38	2.78	3.10	5.48
	8	92.0	70.53	2.43	2.99	2.96	5.32
	16	83.8	61.43	3.67	2.78	3.35	5.48
	32	81.6	54.70	5.55	2.87	3.31	4.92
	40	82.5	53.03	7.49	2.75	3.32	5.00
TSYL201	s	606.1	0.00	555.63	8.83	10.08	26.61
	1	82.7	58.30	3.08	4.21	4.20	6.74
	8	83.4	58.63	3.02	3.77	3.96	6.56
	16	83.3	57.86	3.17	3.63	4.06	6.76
	32	75.7	50.01	6.41	3.20	3.46	5.65
	40	75.9	49.77	6.42	3.18	3.76	5.51

Table 5.4: Performance for different pivot block sizes. $r_{min} = s$ denotes **all** pivot blocks are of size 1. **nrhs** denotes the number of right-hand sides.

Identifier	r_{min}	Factor Time (seconds)	Solve Time (seconds) nrhs=1	nrhs=2	nrhs=10
AEAC5081	s	9.9	0.51	0.62	2.64
	1	3.3	0.14	0.21	0.71
	8	2.8	0.09	0.15	0.45
	16	3.1	0.05	0.13	0.44
	32	3.5	0.15	0.18	0.45
	40	4.0	0.14	0.12	0.42
CRPLAT2	s	216.0	3.73	5.83	33.80
	1	69.4	1.27	1.84	8.09
	8	60.2	1.17	1.70	6.31
	16	58.6	1.15	1.46	5.39
	32	62.5	1.01	1.46	4.44
	40	63.7	0.89	1.45	4.25
OPT1	s	455.6	4.19	7.34	40.63
	1	115.5	1.39	2.00	7.18
	8	115.3	1.47	2.08	6.99
	16	107.1	1.16	1.89	5.98
	32	110.1	1.25	1.76	5.46
	40	112.3	1.21	1.75	5.19

Table 5.5: Performance for different pivot block sizes on an IBM RS/6000. $r_{min} = s$ denotes **all** pivot blocks are of size 1. **nrhs** denotes the number of right-hand sides.

Identifier	r_{min}	Factor Time (seconds)	Solve Time (seconds) nrhs=1	nrhs=2	nrhs=10
AEAC5081	s	5.3	0.79	1.12	3.43
	1	4.4	0.21	0.28	0.86
	8	3.9	0.15	0.19	0.57
	16	3.9	0.15	0.19	0.56
	32	3.9	0.15	0.19	0.57
	40	4.0	0.15	0.19	0.56
CRPLAT2	s	59.9	4.99	6.80	23.69
	1	54.8	1.23	1.57	5.14
	8	47.6	0.87	1.10	3.51
	16	45.0	0.74	0.93	2.89
	32	44.1	0.62	0.77	2.38
	40	44.7	0.63	0.79	2.34
OPT1	s	109.6	5.40	7.46	25.98
	1	94.1	1.18	1.38	4.40
	8	93.2	1.17	1.39	4.34
	16	84.7	0.84	1.08	3.46
	32	80.2	0.78	1.02	2.97
	40	81.2	0.79	0.98	2.92

Table 5.6: Performance for different pivot block sizes on CRAY J932. $r_{min} = s$ denotes **all** pivot blocks are of size 1. **nrhs** denotes the number of right-hand sides.

6 Concluding remarks

We have shown how the frontal method can be implemented to enhance the use of Level 3 BLAS and, using a range of practical problems, we have illustrated that, on cache-based machines, this leads to very good performance in terms of Mflops. The implementation of the frontal method which uses only pivot blocks of size 1 does reasonably well on vector machines but performs poorly on cache-based machines. The plot given in Figure 1 of the speed of DGEMM against the interior dimension indicates that the choice of the minimum pivot block size parameter is not 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.

A disadvantage of frontal schemes is that they usually perform many more operations than are necessary for the numerical factorization and the factors normally have many more entries than those obtained by other techniques. This is illustrated by Duff and Scott (1996*a*). However, in practice we have found that the convenience of being able to specify memory requirements in advance and being able to hold the factors out-of-core more than compensates for this. As a result, we have made extensive use of MA42 and its predecessor, MA32, for more than 15 years. For problems in three dimensions, other techniques are clearly needed, but for two dimensional problems, ease of use and performance mean the frontal method remains our method of choice.

Clearly, it is important that we implement our algorithms to make effective use of machines which have a hierarchical memory structure. The techniques which we have discussed in this paper for making better reuse of data in the cache are applicable to other direct solvers.

7 Availability of software

MA42 and the element ordering routine MC43 are included in Release 12 of the Harwell Subroutine Library. A complex frontal solver, ME42, as well as a frontal solver for symmetric positive-definite systems, MA62, are also available. These codes are all written in standard Fortran 77; a Fortran 90 version of MA42 is also included in Release 12 of the HSL. Anyone wishing to use the codes should contact the HSL Manager: Dr. S. J. Roberts, Harwell Subroutine Library, AEA Technology, Building 552, Harwell, Oxfordshire, OX11 0RA, England, tel. +44 (0) 1235 434714, fax +44 (0) 1235 434136, or e-mail `Scott.Roberts@aeat.co.uk`, who will provide details of price and conditions of use.

Acknowledgements

We are grateful to Alison Ramage of the University of Strathclyde for supplying the test problem RAMAGE02, and to Christian Damhaug of DNVR, Norway, for problems CRPLAT2, OPT1, TRDHEIM, and TSYL201. We are also extremely grateful to our colleague Jean-Yves L'Excellent at CERFACS for obtaining the results presented in Tables 5.3 and 5.4.

References

J.J. Dongarra, J. DuCroz, I. S. Duff, and S. Hammarling. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, **16**(1), 1–17, 1990.

I. S. Duff. Design features of a frontal code for solving sparse unsymmetric linear systems out-of-core. *Parallel Computing*, **5**, 55–64, 1984.

I. S. Duff and J. A. Scott. MA42 – a new frontal code for solving sparse unsymmetric systems. Technical Report RAL-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-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 Transactions on Mathematical Software*, **22**, 30–45, 1996*b*.

I. S. Duff, J. K. Reid, and J. A. Scott. The use of profile algorithms with a frontal solver. *International 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)*. AEA Technology, Harwell Laboratory, Harwell, UK, 1996.

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

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