A COMBINED UNIFRONTAL/MULTIFRONTAL METHOD FOR UNSYMMETRIC SPARSE MATRICES

TIMOTHY A. DAVIS* AND IAIN S. DUFF†

Technical Report TR-95-020, Computer and Information Science and Engineering Department, University of Florida. September, 1995.

Abstract. We discuss the organization of frontal matrices in multifrontal methods for the solution of large sparse sets of linear equations. In the multifrontal method, several frontal matrices are used. Each is used for one or more pivot steps, and the resulting Schur complement is summed with other Schur complements to generate another frontal matrix. Although this means that arbitrary sparsity patterns can be handled efficiently, extra work is required to add the Schur complements together and can be costly because indirect addressing is required. The (uni-)frontal method avoids this extra work by factorizing the matrix with a single frontal matrix. Rows and columns are added to the frontal matrix, and pivot rows and columns are removed. Data movement is simpler, but higher fill-in can result if the matrix cannot be permuted into a variable-band form with small profile. We consider a combined unifrontal/multifrontal algorithm to enable general fill-in reduction orderings to be applied without the data movement of previous multifrontal approaches.

Categories and Subject Descriptors:
G.1.3 **[Numerical Analysis]:** Numerical Linear Algebra - *linear systems (direct methods), sparse and very large systems;*
G.4 **[Mathematics of Computing]:** Mathematical Software - *algorithm analysis, efficiency.*
General Terms: Algorithms, Experimentation, Performance.
Additional Key Words and Phrases: sparse matrices, linear equations, multifrontal methods, frontal methods.

1. Introduction. We consider the direct solution of sets of linear equations $\mathbf{Ax} = \mathbf{b}$, where the coefficient matrix $\mathbf{A}$ is sparse, unsymmetric, and has a general nonzero pattern. In a direct approach, a permutation of the matrix $\mathbf{A}$ is factorized into its LU factors, $\mathbf{PAQ} = \mathbf{LU}$, where $\mathbf{P}$ and $\mathbf{Q}$ are permutation matrices chosen to preserve sparsity and maintain numerical accuracy. Many recent algorithms and software for direct solution of sparse systems are based on a multifrontal approach [17, 2, 10, 27]. In this paper, we will examine a frontal matrix strategy to be used within a multifrontal approach.

Frontal and multifrontal methods compute the LU factors of $\mathbf{A}$ by using data structures that permit regular access of memory and the use of dense matrix kernels in their innermost loops. On supercomputers and high-performance workstations, this can lead to a significant increase in performance over methods that have irregular memory access and which do not use dense matrix kernels.

We discuss frontal methods in Section 2 and multifrontal techniques in Section 3, before discussing the combination of the two methods in Section 4. We consider the influence of the principle parameter in our approach in Section 5, and the performance

* Computer and Information Science and Engineering Department, University of Florida, Gainesville, Florida, USA. (904) 392-1481, email: davis@cis.ufl.edu. Technical reports and matrices are available via the World Wide Web at http://www.cis.ufl.edu/~davis, or by anonymous ftp at ftp.cis.ufl.edu:cis/tech-reports.

† Rutherford Appleton Laboratory, Chilton, Didcot, Oxon. 0X11 0QX England, and European Center for Research and Advanced Training in Scientific Computation (CERFACS), Toulouse, France. email: isd@letterbox.rl.ac.uk. Technical reports, information on HSL, and matrices are available via the World Wide Web at http://www.cis.rl.ac.uk/struct/ARCD/NUM.html, or by anonymous ftp at seamus.cc.rl.ac.uk/pub.

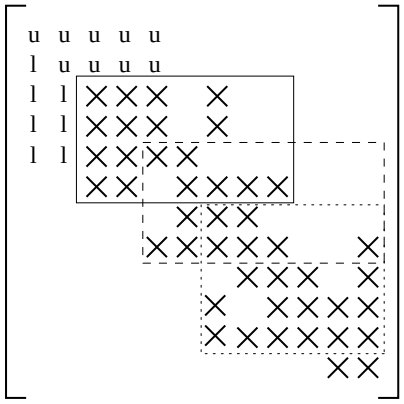


FIG. 2.1. *Frontal method example*

of our new approach in Section 6, before a few concluding remarks and information on the availability of our codes in Section 7.

2. Frontal methods. In a frontal scheme [15, 26, 31, 32], the factorization proceeds as a sequence of partial factorizations and eliminations on full submatrices, called frontal matrices. Although frontal schemes were originally designed for the solution of finite element problems [26], they can be used on assembled systems [15] and it is this version that we study in this paper. For general systems, the frontal matrices can be written as

$$\begin{pmatrix} \mathbf{F}_{11} & \mathbf{F}_{12} & \mathbf{F}_{13} \\ \mathbf{F}_{21} & \mathbf{F}_{22} & \mathbf{F}_{23} \end{pmatrix}, \tag{2.1}$$

where all rows are fully summed (that is there is no further contributions to come to the rows in (2.1)) and the first two block columns are fully summed. This means that pivots can be chosen from anywhere in the fully summed block comprising the first two block columns and, within these columns, numerical pivoting with arbitrary row interchanges can be accommodated since all rows in the frontal matrix are fully-summed. We assume, without loss of generality, that the pivots that have been chosen are in the square matrix $\mathbf{F}_{11}$ of (2.1). $\mathbf{F}_{11}$ is factorized, multipliers are stored over $\mathbf{F}_{21}$ and the Schur complement $(\mathbf{F}_{22}\ \mathbf{F}_{23}) - \mathbf{F}_{21}\mathbf{F}_{11}^{-1}(\mathbf{F}_{12}\ \mathbf{F}_{13})$ is formed, all using full matrix kernels. The submatrix consisting of the rows and columns of the frontal matrix from which pivots have not yet been selected is called the contribution block.

At the next stage, further entries from the original matrix are assembled with the Schur complement to form another frontal matrix. The overhead is low since each equation is only assembled once and there is never any assembly of two (or more) frontal matrices.

An example is shown in Figure 2.1, where two pivot steps have already been performed on a 5-by-7 frontal matrix (computing the first two rows and columns of $\mathbf{U}$ and $\mathbf{L}$, respectively). Nonzero entries in $\mathbf{L}$ and $\mathbf{U}$ are shown in lower case. Row 6 has just been assembled into the current 4-by-7 frontal matrix (shown as a solid box). Columns 3 and 4 are now fully-summed and can be eliminated. After this step, rows 7 and 8 must both be assembled before columns 5 and 6 can be eliminated (the dashed box, a 4-by-6 frontal matrix containing rows 5 through 8 and columns 5, 6, 7, 8, 9, and 12 of the active submatrix). The frontal matrices are, of course, stored with columns

packed together so no zero columns are held in the frontal matrix delineated by the dashed box. The dotted box shows the state of the frontal matrix when the next five pivots can be eliminated. To factorize the 12-by-12 sparse matrix in Figure 2.1, a (dense) working array of size 5-by-7 is required to hold the frontal matrix. Note that, in Figure 2.1, the columns are in pivotal order.

One important advantage of the method is that only the frontal matrix need reside in memory. Entries in $\mathbf{A}$ can be read sequentially from disk into the frontal matrix, one row at a time. Entries in $\mathbf{L}$ and $\mathbf{U}$ can be written sequentially to disk in the order they are computed.

The method works well for matrices with small profile, where the profile of a matrix is a measure of how close the nonzero entries are to the diagonal and is given by the expression:

$$\sum_{i=1}^{n} \{ \max_{a_{ij} \neq 0} (i-j) + \max_{a_{ji} \neq 0} (i-j) \},$$

where it is assumed the diagonal is nonzero so all terms in the summation are non-negative. If numerical pivoting is not required, fill-in does not increase the profile ($\mathbf{L} + \mathbf{U}$ has the same profile as $\mathbf{A}$). To reduce the profile, the frontal method is typically preceded by an ordering method such as reverse Cuthill-McKee (RCM) [5, 7, 29], which is typically faster than the sparsity-preserving orderings required by a more general technique like a multifrontal method (such as nested dissection [24] and minimum degree [1, 25]). A degree update phase, which is typically the most costly part of a minimum degree algorithm, is not required in the RCM algorithm. However, for matrices with large profile, the frontal matrix can be large, and an unacceptable level of fill-in can occur.

3. Multifrontal methods. In a multifrontal scheme for a symmetric matrix [2, 8, 9, 10, 17, 18, 28], it is normal to use an ordering such as minimum degree to reduce the fill-in. Such an ordering tends to reduce fill-in much more than profile reduction orderings. The ordering is combined with a symbolic analysis to generate an assembly or computational tree, where each node represents the assembly and elimination operations on a frontal matrix and each edge the transfer of data from child to parent node. When using the tree to drive the numerical factorization, assembly and eliminations at any node can proceed as soon as those at the child nodes have been completed, giving added flexibility for issues such as exploitation of parallelism. As in the frontal scheme, the complete frontal matrix (2.1) cannot normally be factorized but only a few steps of Gaussian elimination are possible, after which the remaining reduced matrix (the Schur complement $(\mathbf{F}_{22}\ \mathbf{F}_{23}) - \mathbf{F}_{21}\mathbf{F}_{11}^{-1}(\mathbf{F}_{12}\ \mathbf{F}_{13})$) needs to be summed (assembled) with other data at the parent node.

In the unsymmetric-pattern multifrontal method, the tree is replaced by a directed acyclic graph (dag) [9], and a contribution block may be assembled into more than one subsequent frontal matrix.

4. Combining the two methods. Let us now consider an approach that combines some of the best features of the two methods. Assume we have chosen a pivot and determined a frontal matrix as in a normal multifrontal method. At this stage, a normal multifrontal method will select as many pivots as it can from the fully summed part of the frontal matrix, perform the eliminations corresponding to these pivots, store the contributions to the factors, and store the remaining frontal matrix for later assembly at the parent node of the assembly tree. The strategy we use in

our combined method is, after forming the frontal matrix, to hold it as a submatrix of a larger working array and allow further pivots to be chosen from anywhere within the frontal matrix. If such a potential pivot lies in the non-fully summed parts of the frontal matrix then it is necessary to sum its row and column before it could be used as a pivot. This is possible so long as its fully summed row and column can be accommodated within the larger working array. In this way, we avoid some of the data movement and assemblies of the multifrontal method. Although the motivation is different, the idea of continuing with a frontal matrix for some steps before moving to another frontal matrix is similar to recent work in implementing frontal schemes within a domain decomposition environment, for example [23], where several fronts are used within a unifrontal context. However, in the case of [23], the ordering is done a priori and no attempt is made to use a minimum degree ordering. Another case where one continues with a frontal matrix for several assemblies is when relaxed supernode amalgamation is used [17]. However, this technique delays the selection of pivots and normally causes more fill-in and operations than our technique. Indeed, the use of our combined unifrontal/multifrontal approach does not preclude implementing relaxed supernode amalgamation as well. In comparison with the unifrontal method where the pivoting is determined entirely from the order of the assemblies, the combined method requires the selection of pivots to start each new frontal matrix. This implies either an a priori symbolic ordering or a pivot search during numerical factorization. The pivot search heuristic requires the degrees of rows and columns, and thus a degree update phase. The cost of this degree update is clearly a penalty we pay to avoid the poor fill-in properties of conventional unifrontal orderings.

We now describe how this new frontal matrix strategy is applied in UMFPACK Version 1.1 [8, 10], in order to obtain a new combined method. However, the new frontal strategy can be applied to any multifrontal algorithm. UMFPACK does not use actual degrees but keeps track of upper bounds on the degree of each row and column. The symmetric analogue of the approximate degree update in UMFPACK has been incorporated into an approximate minimum degree algorithm (AMD) as discussed in [1], where the accuracy of our degree bounds is demonstrated.

Our new algorithm consists of several major steps, each of which comprises several pivot selection and elimination operations. To start a major step, we choose a pivot, using the degree upper bounds and a numerical threshold test, and then define a working array which is used for pivoting and elimination operations so long as these can be performed within it. When this is no longer possible, the working array is stored and another major step is commenced.

To start a major step, the new algorithm selects a few (by default 4) columns from those of minimum upper bound degree and computes their patterns and true degrees. A pivot row is selected on the basis of the upper bound on the row degree from those rows with nonzero entries in the selected columns. Suppose the pivot row and column degrees are r and c, respectively. A k-by-l working array is allocated, where $k = \lfloor Gc \rfloor$ and $l = \lfloor Gr \rfloor$ (G typically has a value between 2 and 3, and is fixed for the entire factorization). The active frontal matrix is c-by-r but is stored in a k-by-l working array. The pivot row and column are fully assembled into the working array and define a submatrix of it as the active frontal matrix. The approximate degree update phase computes the bounds on the degrees of all the rows and columns in this active frontal matrix and assembles previous contribution blocks into the active frontal matrix. A row i in a previous contribution block is assembled into the active frontal matrix if

1. the row index i is in the nonzero pattern of the current pivot column, and
2. if the column indices of the remaining entries in the row are all present in the nonzero pattern of the current pivot row.

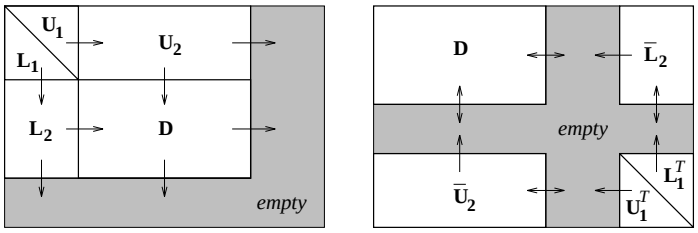
Columns of previous contribution blocks are assembled in an analogous manner.

The major step then continues with a sequence of minor steps at each of which another pivot is sought. These minor steps are repeated until the factorization can no longer continue within the current working array, at which point a new major step is started. When a pivot is chosen in a minor step, its rows and columns are fully assembled into the working array and redefine the active frontal matrix. The approximate degree update phase computes the bounds on all rows and columns in this active frontal matrix and assembles previous contribution blocks into the active frontal matrix. The corresponding row and column of $\mathbf{U}$ and $\mathbf{L}$ are computed, but the updates from this pivot are not necessarily performed immediately. For efficient use of the Level 3 BLAS it is better to accumulate a few updates (typically 16 or 32, if possible) and perform them at the same time. To find a pivot in this minor step, a single candidate column from the contribution block is first selected, choosing one with least value for the upper bound of the column degree, and any pending updates are applied to this column. The column is assembled into a separate work vector, and a pivot row is selected on the basis of upper bound on the row degrees and a numerical threshold test. Suppose the candidate pivot row and column have degrees are r' and c', respectively. Three conditions apply:

1. If $r' > l$ or $c' > k$, then factorization can no longer continue within the active frontal matrix. Any pending updates are applied. The LU factors are stored. The active contribution block is saved for later assembly into a subsequent frontal matrix. The major step is now complete.
2. If $r' \leq l - p$ and $c' \leq k - p$, then the candidate pivot can fit into the active frontal matrix without removing the p pivots already stored there. Set $p \leftarrow p + 1$. Factorization continues within the active frontal matrix by commencing another minor step.
3. Otherwise, if $l - p < r' \leq l$ or $k - p < c' \leq k$, then the candidate pivot can fit, but only if some of the previous p pivots are shifted out of the current frontal matrix. Any pending updates are applied. The LU factors corresponding to the pivot rows and columns are removed from the front and stored. The active contribution block is left in place. Set $p \leftarrow 1$. Factorization continues within the active frontal matrix by commencing another minor step.

We know of no previous multifrontal method that considers case 3 (in particular, UMFPACK Version 1.1 does not). Case 1 does not occur in frontal methods, which are given a working array large enough to hold the largest frontal matrix (unless numerical pivoting causes the frontal matrix to grow unexpectedly). Taking simultaneous advantage of both cases 1 and 3 can significantly reduce the memory requirements and number of assembly operations, while still allowing the use of orderings that reduce fill-in.

Figure 4.1 illustrates how the working array is used in UMFPACK Version 1.1 and the combined method. The matrices $\mathbf{L}_1$, $\mathbf{L}_2$, $\mathbf{U}_1$, and $\mathbf{U}_2$ in the figure are the columns and rows of the LU factors corresponding to the pivots eliminated within this frontal matrix. The matrix $\mathbf{D}$ is the contribution block. The matrix $\bar{\mathbf{U}}_2$ is $\mathbf{U}_2$ with rows in reverse order, and $\bar{\mathbf{L}}_2$ is $\mathbf{L}_2$ with columns in reverse order. The arrows denote how these matrices grow as new pivots are added. When pivots are removed from the working array in Figure 4.1(b), for case 4 above, the contribution block does



(a) UMFPACK working array (b) Combined method working array

FIG. 4.1. *Data structures for the active frontal matrix*

TABLE 5.1
Test matrices.

name	n	$\|\mathbf{A}\|$	sym.	discipline	comments
gre_1107	1107	5664	0.000	discrete simul.	computer system
gemat11	4929	33185	0.001	electric power	linear programming basis
orani678	2529	90158	0.071	economics	Australia
psmigr_1	3140	543162	0.479	demography	US county-to-county migration
lns_3937	3937	25407	0.850	fluid flow	linearized Navier-Stokes
shyy161	76480	329762	0.726	fluid flow	viscous fully-coupled Navier-Stokes
hydr1	5308	23752	0.004	chemical eng.	dynamic simulation
rdist1	4134	94408	0.059	chemical eng.	reactive distillation
lhr04	4101	82682	0.015	chemical eng.	light hydrocarbon recovery
lhr71	70304	1528092	0.002	chemical eng.	light hydrocarbon recovery

not need to be repositioned. A similar data organization is employed by MA42 [22].

5. Numerical experiments. We discuss some experiments on the selection of a value for G in this section and compare the performance of our code with other sparse matrix codes in the next section.

We show in Table 5.1 the test matrices we will use in our experiments in this section and the next. The table lists the name, order, number of entries ($|\mathbf{A}|$), symmetry, the discipline from which the matrix comes, and additional comments. The symmetry, or more correctly the structural symmetry, is the number of *matched* off-diagonal entries over the total number of off-diagonal entries. An entry, a_{ij} $(j \neq i)$, is matched if a_{ji} is also an entry.

We have performed our experiments on the effect of the value of G on a SUN SPARCstation 20 Model 41, using the Fortran compiler (f77 version 1.4, with -O4 and -libmil options), and the BLAS from [11] and show the results in Table 5.2. The table lists the matrix name, the growth factor G, the number of frontal matrices (major steps), the numerical factorization time in seconds, the total time in seconds, the number of nonzeros in the LU factors, the memory used, and the floating-point operation count.

The number of frontal matrices is highest for $G = 1$ and decreases as G increases although, because the effect is local, this decrease may not be monotonic. Although the fill-in and operation count is typically the lowest when the minimum amount of memory is allocated for each frontal matrix ($G = 1$), the factorization time is often high because of the additional data movement required to assemble the contribution blocks and the fact that the dense matrix kernels are more efficient for larger frontal matrices.

These results show that our strategy of allocating more space than necessary for the frontal matrix and choosing pivots that are not from the initial fully summed

TABLE 5.2
Effect of G on the performance of the combined method

Matrix	G	fronts	factor (sec)	total (sec)	$\|\mathbf{L}+\mathbf{U}\|$ (10^6)	memory (10^6)	op count (10^6)
gre_1107	1.0	926	0.51	0.89	0.048	0.160	2.6
	1.5	697	0.32	0.66	0.060	0.157	3.8
	2.0	406	0.43	0.74	0.079	0.196	6.7
	2.5	323	0.70	1.01	0.100	0.224	9.0
	3.0	295	0.55	0.74	0.088	0.194	6.9
	5.0	159	0.69	1.09	0.120	0.239	9.5
	7.0	122	0.75	0.85	0.121	0.235	9.5
gemat11	1.0	1885	0.39	0.84	0.052	0.306	0.6
	1.5	1281	0.37	0.76	0.058	0.293	0.7
	2.0	988	0.37	0.70	0.067	0.287	0.8
	2.5	873	0.39	0.69	0.077	0.299	1.0
	3.0	776	0.33	0.66	0.087	0.312	1.3
	5.0	586	0.41	0.70	0.119	0.353	2.0
	7.0	538	0.41	0.77	0.148	0.381	2.8
orani678	1.0	2400	2.08	4.86	0.113	0.932	6.2
	1.5	2332	1.19	3.93	0.121	0.947	7.0
	2.0	2324	1.41	3.79	0.122	0.959	7.2
	2.5	2326	1.37	3.96	0.122	0.883	7.2
	3.0	2322	1.41	3.97	0.123	0.886	8.1
	5.0	2298	1.24	3.91	0.128	0.854	8.4
	7.0	2280	2.82	5.87	0.153	0.852	16.0
lns_3937	1.0	3208	12.82	17.16	0.474	1.161	95.5
	1.5	2699	5.63	7.80	0.421	0.976	68.6
	2.0	2011	5.68	9.02	0.494	1.170	84.1
	2.5	1834	4.60	6.62	0.442	1.047	60.0
	3.0	1717	7.01	9.29	0.551	1.425	90.0
	5.0	1446	6.26	9.30	0.591	1.750	96.4
	7.0	1454	11.76	15.14	0.697	2.005	134.1
hydr1	1.0	4749	0.57	1.69	0.078	0.356	1.4
	1.5	4204	0.50	1.61	0.088	0.352	1.6
	2.0	3947	0.52	1.57	0.112	0.384	2.7
	2.5	3726	0.67	1.46	0.122	0.398	3.3
	3.0	3399	0.67	1.58	0.129	0.414	3.7
	5.0	3325	0.72	1.53	0.153	0.440	3.9
	7.0	3231	0.73	1.71	0.153	0.497	4.7
rdist1	1.0	2436	3.87	6.25	0.359	0.887	21.3
	1.5	1255	2.84	4.47	0.446	0.969	27.0
	2.0	306	1.12	1.85	0.278	0.668	10.6
	2.5	316	1.21	1.85	0.304	0.694	11.5
	3.0	295	1.23	1.69	0.333	0.702	10.4
	5.0	160	2.15	2.21	0.493	0.890	14.6
	7.0	160	3.17	2.04	0.620	1.038	14.7
lhr04	1.0	3097	3.05	5.64	0.267	0.813	14.5
	1.5	2714	2.16	4.56	0.303	0.965	18.1
	2.0	2223	1.90	4.03	0.313	0.824	16.4
	2.5	2185	3.09	5.12	0.457	1.051	32.4
	3.0	2111	1.78	3.56	0.356	0.895	18.9
	5.0	2153	2.51	4.10	0.444	0.995	26.4
	7.0	2180	3.70	4.55	0.497	1.195	28.9

block can give substantial gains in execution time with sometimes a small penalty in storage, although occasionally the storage can be less as well.

6. Performance. In this section, we compare the performance of the combined unifrontal/multifrontal method (MA38) with the unsymmetric-pattern multifrontal

method (UMFPACK Version 1.1 [8, 10]), a general sparse matrix factorization algorithm that is not based on frontal matrices (MA48, [19, 20]), the frontal method (MA42, [15, 22]), and the symmetric-pattern multifrontal method (MUPS [2]). All methods can factorize general unsymmetric matrices and all use dense matrix kernels to some extent [12]. We tested each method on a single processor of a CRAY C-98, although MUPS is a parallel code. Version 6.0.4.1 of the Fortran compiler (CF77) was used. Each method (except MA42, which we discuss later) was given 95 Mw of memory to factorize the matrices listed in Table 5.1. Each method has a set of input parameters that control its behavior. We used the recommended defaults for most of these, with a few exceptions that we now indicate.

By default, three of the five methods (MA38, UMFPACK V1.1, and MA48) preorder a matrix to block triangular form (always preceded by finding a maximum transversal [14]), and then factorize each block on the diagonal [16]. This can reduce the work for unsymmetric matrices. We did not perform the preordering, since MA42 and MUPS do not provide these options.

One matrix (lhr71) was so ill-conditioned that it required scaling prior to its factorization. The scale factors were computed by the Harwell Subroutine Library routine MC19A [6]. Each row was then subsequently divided by the maximum absolute value in the row (or column, depending on how the method implements threshold partial pivoting). No scaling was performed on the other matrices.

By default, MUPS preorders each matrix to maximize the modulus of the smallest entry on the diagonal (using a maximum transversal algorithm). This is followed by a minimum degree ordering on the nonzero pattern of $\mathbf{A} + \mathbf{A}^T$.

For MA42, we first preordered the matrix to reduce its profile using a version of Sloan's algorithm [30, 21]. MA42 is able to operate both in-core and out-of-core, using direct access files. It has a finite-element entry option, for which it was primarily designed. We used the equation entry mode on these matrices, although some matrices are obtained from finite-element calculations. We tested MA42 both in-core and out-of-core. The CPU time for the out-of-core factorization was only slightly higher than the in-core factorization, but the memory usage was much less. We thus report only the out-of-core results. Note that the CPU time does not include any additional system or I/O device time required to read and write the direct access files. The only reliable way to measure this is on a dedicated system. On a CPU-bound multiuser system, the time spent waiting for I/O would have little effect on total system throughput.

The symbolic analysis phase in MA42 determines a minimum front size (k-by-l, which assumes no increase due to numerical pivoting), and minimum sizes of other integer and real buffers (of total size b, say). We gave the numerical factorization phase a working array of size $2k$-by-$2l$, and a buffer size of $2b$, to allow for numerical pivoting. In our tables, we show how much space was actually used. The disk space taken by MA42 is not included in the memory usage.

The results are shown in Table 6.1. For each matrix, the tables list the numerical factorization time, total factorization time, number of nonzeros in $\mathbf{L}+\mathbf{U}$ (in millions), amount of memory used (in millions of words), and floating-point operation count (in millions of operations) for each method. The total time includes preordering, symbolic analysis and factorization, and numerical factorization. The time to compute the scale factors for the lhr71 matrix is not included, since we used the same scaling algorithm for all methods. For each matrix, the lowest time, memory usage, or operation count is shown in bold. We compared the solution vectors, $\mathbf{x}$, for each method. We found

that all five methods compute the solutions with comparable accuracy, in terms of the norm of the residual. We do not give the residual in Table 6.1.

All five codes have factorize-only options that are often much faster than the combined analysis+factorization phase(s), and indeed the design criterion for some codes (for example MA48) was to minimize factorize time even if this caused an increase in the initial analyse time. For MA42, however, the analysis is particularly simple so normally its overhead is much smaller than for the other codes.

MUPS is shown as failing twice (on shyy161 and lhr71 which are both nearly singular). In both cases, numerical pivoting caused an increase in storage requirements above the 95 Mw allocated. MA42 failed to factorize the shyy161 matrix because of numerical problems. It ran out of disk space during the numerical factorization of the lhr71 matrix (about 5.8 Gbytes were required). For this matrix, we report the estimates of the number of entries in the LU factors and the memory usage, as reported by the symbolic analysis phase of MA42.

Since the codes being compared all offer quite different capabilities and are designed for different environments, the results should not be interpreted as a direct comparison between them. However, what we would like to highlight is the improvements that our new technique brings to the UMFPACK code and that the new code is at least comparable in performance with other sparse matrix codes. The peak performance of MA38 is 607 Mflops for the numerical factorization of the psmigr_1 matrix (compared with the theoretical peak of about 1 Gflop for one C-90 processor). Our new code requires less storage than the original UMFPACK code and sometimes it requires less than half the storage. In execution time for both analyse/factor and factorize only, it is generally faster (sometimes by nearly a factor of two) and when it is slower than the original UMFPACK code it is so by at most about 13% (for the hydr1 matrix). Over all the codes, MA38 has the fastest analyse+factorize time for five out of the ten matrices. Except for one matrix (lns_3937) it never takes more than twice the time of the fastest method. Its memory usage is comparable to the other in-core methods. MA42 can typically factorize these matrices with the least amount of core memory. MA42 was originally designed for finite-element entry and so is not optimized for equation entry. The experiments are run on only one computer which strongly favors direct addressing and may thus disadvantage MA48 which does not make such heavy use of the Level 3 BLAS.

7. Summary. We have demonstrated how the advantages of the frontal and multifrontal methods can be combined. The resulting algorithm performs well for matrices from a wide range of disciplines. We have shown that the combined unifrontal/multifrontal method gains over our earlier code because it avoids more indirect addressing and generally performs eliminations on larger frontal matrices. Other differences between UMFPACK Version 1.1 and the new code (MA38, or UMFPACK Version 2.0) include an option of overwriting the matrix $\mathbf{A}$ with its LU factors, printing of input and output parameters, a removal of the extra copy of the numerical values of $\mathbf{A}$, iterative refinement with sparse backward error analysis [4], more use of Level 3 BLAS within the numerical factorization routine, and a simpler calling interface. These features improve the robustness of the code and result a modest decrease in memory usage.

The combined unifrontal/multifrontal method is available as the Fortran 77 codes, UMFPACK Version 2.0 in Netlib [13],[1] and MA38 in Release 12 of the Harwell

[1] UMFPACK Version 2.0, in Netlib, may only be used for research, education, or benchmarking

Subroutine Library [3].[2]

8. Acknowledgements. We would like to thank Nick Gould, John Reid, and Jennifer Scott from the Rutherford Appleton Laboratory for their helpful comments on a draft of this report.

REFERENCES

[1] P. Amestoy, T. A. Davis, and I. S. Duff, *An approximate minimum degree ordering algorithm*, SIAM Journal on Matrix Analysis and Applications, (to appear). (Also University of Florida technical report TR-94-039).

[2] P. R. Amestoy and I. S. Duff, *Vectorization of a multiprocessor multifrontal code*, Int. J. Supercomputer Appl., 3 (1989), pp. 41–59.

[3] Anon, *Harwell Subroutine Library. A Catalogue of Subroutines (Release 11)*, Theoretical Studies Department, AEA Industrial Technology, 1993.

[4] M. Arioli, J. W. Demmel, and I. S. Duff, *Solving sparse linear systems with sparse backward error*, SIAM Journal on Matrix Analysis and Applications, 10 (1989), pp. 165–190.

[5] W. M. Chan and A. George, *A linear time implementation of the reverse Cuthill-Mckee algorithm*, BIT, 20 (1980), pp. 8–14.

[6] A. R. Curtis and J. K. Reid, *On the automatic scaling of matrices for Gaussian elimination*, Journal of the Institute of Mathematics and its Applications, 10 (1972), pp. 118–124.

[7] E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, in Proceedings 24th National Conference of the Association for Computing Machinery, New Jersey, 1969, Brandon Press, pp. 157–172.

[8] T. A. Davis, *Users' guide to the unsymmetric-pattern multifrontal package (UMFPACK, version 1.1)*, Tech. Report TR-95-004, CIS Dept., Univ. of Florida, Gainesville, FL, 1995. For a copy of UMFPACK Version 1.1, send e-mail to `netlib@ornl.gov` with the one-line message `send umfpack.shar from linalg`.

[9] T. A. Davis and I. S. Duff, *Unsymmetric-pattern multifrontal methods for parallel sparse LU factorization*, Tech. Report TR-91-023, CIS Dept., Univ. of Florida, Gainesville, FL, 1991.

[10] ———, *An unsymmetric-pattern multifrontal method for sparse LU factorization*, SIAM Journal on Matrix Analysis and Applications, (to appear). (Also University of Florida technical report TR-94-038).

[11] M. J. Daydé and I. S. Duff, *A block implementation of Level 3 BLAS for RISC processors*, Tech. Report To appear, CERFACS, Toulouse, France, 1995.

[12] J. J. Dongarra, J. J. Du Croz, I. S. Duff, and S. Hammarling, *A set of Level 3 Basic Linear Algebra Subprograms.*, ACM Transactions on Mathematical Software, 16 (1990), pp. 1–17.

[13] J. J. Dongarra and E. Grosse, *Distribution of mathematical software via electronic mail*, Comm. ACM, 30 (1987), pp. 403–407.

[14] I. S. Duff, *On algorithms for obtaining a maximum transversal*, ACM Transactions on Mathematical Software, 7 (1981), pp. 315–330.

[15] ———, *Design features of a frontal code for solving sparse unsymmetric linear systems out-of-core*, SIAM Journal on Scientific and Statistical Computing, 5 (1984), pp. 270–280.

[16] I. S. Duff and J. K. Reid, *An implementation of Tarjan's algorithm for the block triangularization of a matrix*, ACM Trans. Math. Softw., 4 (1978), pp. 137–147.

[17] ———, *The multifrontal solution of indefinite sparse symmetric linear systems*, ACM Transactions on Mathematical Software, 9 (1983), pp. 302–325.

[18] ———, *The multifrontal solution of unsymmetric sets of linear equations*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 633–641.

by academic users and by the U.S. Government. For a copy, send email to `netlib@ornl.gov` with the message `send index from linalg`. Other users may use UMFPACK only for benchmarking purposes.

[2] The Harwell Subroutine Library is available from AEA Technology, Harwell; the contact is John Harding, Harwell Subroutine Library, B 552, AEA Technology, Harwell, Didcot, Oxon OX11 0RA, telephone (44) 1235 434573, fax (44) 1235 434340, email john.harding@aeat.co.uk, who will provide details of price and conditions of use.

[19] ———, *MA48, a Fortran code for direct solution of sparse unsymmetric linear systems of equations*, Tech. Report RAL-93-072, Rutherford Appleton Laboratory, Didcot, Oxon, England, Oct. 1993.

[20] I. S. Duff and J. K. Reid, *The design of MA48, a code for the direct solution of sparse unsymmetric linear systems of equations*, ACM Transactions on Mathematical Software, To appear (1995), pp. ???–???

[21] I. S. Duff, J. K. Reid, and J. A. Scott, *The use of profile reduction algorithms with a frontal code*, Int. Journal of Numerical Methods in Engineering, 28 (1989), pp. 2555–2568.

[22] I. S. Duff and J. A. Scott, *MA42 – a new frontal code for solving sparse unsymmetric systems*, Tech. Report RAL 93-064, Rutherford Appleton Laboratory, 1993. Shortened version to appear in ACM Trans Math Softw.

[23] I. S. Duff and J. A. Scott, *The use of multiple fronts in Gaussian elimination*, in Proceedings of the Fifth SIAM Conference on Applied Linear Algebra, J. Lewis, ed., Philadelphia, 1994, SIAM Press, pp. 567–571.

[24] A. George and J. W. H. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Englewood Cliffs, New Jersey: Prentice-Hall, 1981.

[25] ———, *The evolution of the minimum degree ordering algorithm*, SIAM Review, 31 (1989), pp. 1–19.

[26] B. M. Irons, *A frontal solution program for finite element analysis*, International Journal for Numerical Methods in Engineering, 2 (1970), pp. 5–32.

[27] J. W. H. Liu, *The multifrontal method for sparse matrix solution: Theory and Practice*, SIAM Review, 34 (1992), pp. 82–109.

[28] J. W. H. Liu, *The multifrontal method for sparse matrix solution: Theory and practice*, SIAM Review, 34 (1992), pp. 82–109.

[29] J. W. H. Liu and A. H. Sherman, *Comparative analysis of the Cuthill-Mckee and the reverse Cuthill-Mckee ordering algorithms for sparse matrices*, SIAM Journal on Numerical Analysis, 13 (1976), pp. 198–213.

[30] S. W. Sloan, *An algorithm for profile and wavefront reduction of sparse matrices*, International Journal of Numerical Methods in Engineering, 23 (1986), pp. 239–251.

[31] S. E. Zitney, J. Mallya, T. A. Davis, and M. A. Stadtherr, *Multifrontal vs. frontal techniques for chemical process simulation on supercomputers*, Comput. Chem. Eng., (1995, to appear).

[32] S. E. Zitney and M. A. Stadtherr, *Supercomputing strategies for the design and analysis of complex separation systems*, Ind. Eng. Chem. Res., 32 (1993), pp. 604–612.

TABLE 6.1
Results

Matrix	method	factor (sec)	total (sec)	$\|\mathbf{L}+\mathbf{U}\|$ (10^6)	memory (10^6)	op count (10^6)
gre_1107	MA38	.07	0.30	.08	.2	6.7
	V1.1	.07	0.30	.09	.3	9.7
	MA48	.11	0.38	.07	.3	8.1
	MA42	.16	0.21	.23	.1	11.0
	MUPS	.13	0.38	.19	.4	26.6
gemat11	MA38	.20	.46	.07	.3	.8
	V1.1	.18	.45	.08	.4	1.0
	MA48	.18	.54	.05	.4	.7
	MA42	15.43	15.95	8.37	2.2	185.2
	MUPS	.27	.57	.14	.4	2.8
orani678	MA38	.53	1.36	.12	1.0	7.2
	V1.1	.53	2.07	.12	1.1	7.4
	MA48	.32	1.01	.15	.8	14.2
	MA42	9.01	9.62	5.47	4.1	166.9
	MUPS	.61	218.69	.39	13.3	87.6
psmigr_1	MA38	15.64	22.67	6.37	25.2	9489.8
	V1.1	15.62	33.99	6.36	26.4	10194.8
	MA48	14.92	28.86	6.40	20.9	10465.3
	MA42	29.78	45.32	9.77	10.4	13588.4
	MUPS	14.04	323.15	6.21	26.9	9002.4
lns_3937	MA38	.42	1.78	.49	1.2	84.1
	V1.1	.45	1.89	.50	1.4	84.8
	MA48	1.00	3.37	.69	2.2	280.4
	MA42	.49	.65	.83	.1	25.0
	MUPS	.71	1.73	.92	1.2	185.8
shyy161	MA38	10.15	26.13	9.52	16.8	3472.5
	V1.1	13.37	31.76	12.58	21.9	7513.4
	MA48	54.47	193.23	9.14	29.8	6315.7
	MA42	-	-	-	-	-
	MUPS	-	-	-	-	-
hydr1	MA38	.27	1.10	.11	.4	2.7
	V1.1	.24	1.05	.15	.6	4.5
	MA48	.28	.81	.08	.4	.9
	MA42	7.65	8.01	6.22	1.1	68.7
	MUPS	.57	1.21	.24	.5	10.7
rdist1	MA38	.36	.80	.28	.7	10.6
	V1.1	.47	1.53	.49	1.4	37.1
	MA48	1.37	4.78	.41	1.6	27.2
	MA42	.93	1.68	1.11	.3	53.1
	MUPS	.33	2.01	.28	.7	10.3
lhr04	MA38	.55	1.79	.31	.8	16.4
	V1.1	.56	2.51	.39	1.5	30.6
	MA48	1.27	4.25	.34	1.3	25.8
	MA42	3.14	4.21	3.54	.7	169.0
	MUPS	1.03	9.89	1.10	2.3	300.3
lhr71	MA38	11.37	43.47	10.18	20.4	1175.5
	V1.1	12.26	53.80	10.49	30.2	1294.5
	MA48	51.60	171.66	10.08	36.1	1338.4
	MA42	-	-	472.16	36.2	156358.8
	MUPS	-	-	-	-	-