An unsymmetrized multifrontal LU factorization *

Patrick R. Amestoy† and Chiara Puglisi‡

July 18, 2000

Abstract

A well-known approach to compute the **LU** factorization of a general unsymmetric matrix **A** is to build the elimination tree associated with the pattern of the symmetric matrix $\mathbf{A} + \mathbf{A}^{\mathrm{T}}$ and use it as a computational graph to drive the numerical factorization. This approach, although very efficient on a large range of unsymmetric matrices, does not capture the unsymmetric structure of the matrices. We introduce a new algorithm which detects and exploits the structural unsymmetry of the submatrices involved during the process of the elimination tree. We show that with the new algorithm significant gains both in memory and in time to perform the factorization can be obtained.

Key words. sparse linear equations, unsymmetric matrices, Gaussian elimination, multifrontal methods, elimination tree

AMS subject classification. 65F05, 65F50

1 Introduction

We consider the direct solution of sparse linear equations based on a multifrontal approach. The systems are of the form $\mathbf{Ax} = \mathbf{b}$ where $\mathbf{A}$ is an $n \times n$ unsymmetric sparse matrix. The multifrontal method has been developed by Duff and Reid [11, 12] for computing the solution of indefinite sparse symmetric linear equations using Gaussian elimination and then has been extended to solve more general unsymmetric matrices by Duff and Reid [13].

The multifrontal method belongs to the class of methods that separate the factorization into an analysis phase and a numerical factorization. The analysis phase involves a reordering step, which will reduce the fill-in during numerical factorization and a symbolic phase that

*This work was supported by the Director, Office of Science, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract number DE-AC03-76SF00098.

†amestoy@enseeiht.fr, ENSEEIHT-IRIT, 2 rue Camichel 31071 Toulouse (France) and PRamestoy@lbl.gov, NERSC, Lawrence Berkeley National Laboratory, 1 Cyclotron Rd. Berkeley CA 94720

‡NERSC, Lawrence Berkeley National Laboratory, 1 Cyclotron Rd., Berkeley CA 94720

builds the computational tree, so called **elimination tree** [9, 18, 20], whose structure gives the dependency graph of the multifrontal approach. The analysis phase is generally not concerned with numerical values and is only based on the sparsity pattern of the matrix.

As far as the analysis phase is concerned, the approaches introduced by Duff and Reid for both symmetric and unsymmetric matrices are almost identical. When the matrix is unsymmetric, the structurally symmetric matrix $\mathbf{M} = \mathbf{A} + \mathbf{A}^{\mathrm{T}}$, where the summation is performed symbolically, is used in place of the original matrix $\mathbf{A}$. The elimination tree of the unsymmetric $\mathbf{LU}$ factorization is thus identical to that of the Cholesky factorization of the symmetrized matrix $\mathbf{M}$.

To control the growth of the factors during $\mathbf{LU}$ factorization, partial threshold pivoting is used during the numerical factorization phase. The pivot order, used during the analysis to build the elimination tree might not be respected. Numerical pivoting can then result in an increase in the estimated size of the factors and in the number of operations. To improve the numerical behaviour of the multifrontal approach it is common to involve a step of preprocessing based on the numerical values. In fact if the matrix is not well-scaled, which means that the entries in the original matrix do not have the same order of magnitude, a good prescaling of the matrix can have a significant impact on the accuracy and performance of the sparse solver. In some cases it is also very beneficial to precede the ordering by performing an unsymmetric permutation to place large entries on the diagonal. Duff and Koster [10] have designed algorithms to permute large entries onto the diagonal and have shown that it can very significantly improve the behaviour of multifrontal solvers.

The multifrontal approach by Duff and Reid [13] is used in the Harwell Subroutine Library code `ma41` [2, 3] and in the distributed memory code `MUMPS` developed in the context of the PARASOL project (EU ESPRIT IV LTR project 20160) [4, 5]. Another way to represent the symbolic $\mathbf{LU}$ factorization of a structurally unsymmetric matrix is to use directed acyclic graphs (see for example [14, 15]). These structures more costly and complicated to handle than a tree, capture better the asymmetry of the matrix. Davis and Duff [6] implicitly use this structure to drive their unsymmetric-pattern multifrontal approach.

We explain, in this article, how to use the simple elimination tree structure of the symmetric matrix $\mathbf{M}$ to detect, during the numerical factorization phase, structural asymmetry in the factors. We show that, with the new factorization phase, we very significantly reduce the computational time, the size of the $\mathbf{LU}$ factors and the total memory requirement with respect to the standard multifrontal approach [13]. In Section 2, we first recall the main properties of the elimination tree and describe the standard multifrontal factorization algorithm. We then introduce the new algorithm and use a simple example to show the benefits that can be expected from the new approach. In Section 3, our set of test matrices is introduced. We analyse the performance gains (in terms of size of the factors, memory requirement and factorization time) of the new approach with respect to the standard multifrontal code on our set of test matrices. We add some concluding remarks in Section 4

2 Description of the multifrontal factorization algorithms

Let $\mathbf{A}$ be an unsymmetric matrix and let $\mathbf{M}$ denotes the structurally symmetric matrix $\mathbf{A}+\mathbf{A}^{\mathrm{T}}$. The elimination tree is defined using the structure of the Cholesky factors of $\mathbf{M}$. If the matrix $\mathbf{M}$ is reducible then the tree will be a forest. Liu [18] defines the elimination tree as the transitive reduction of the directed graph of the Cholesky factors of $\mathbf{M}$. The characterization of the elimination tree and the description of its properties are beyond the scope of this article. In our context, we are interested in the elimination tree only as the computational graph for the multifrontal factorization. For a complete description of the elimination tree the reader can consult [18, 19].

In the multifrontal approaches, we actually use an amalgamated elimination tree, referred to as the **assembly tree** [12] which can be obtained from the classical elimination tree. Each node of the assembly tree corresponds to Gaussian elimination operations on a full submatrix, called a **frontal matrix**. The frontal matrix can be partitioned as shown in Figure 1.

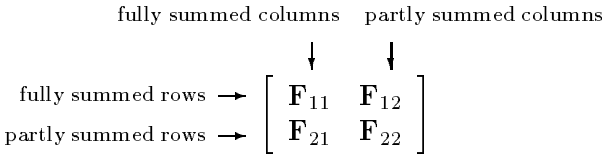

Figure 1: Partitioning of a frontal matrix.

Each frontal matrix factorization involves the computation of a block of columns of $\mathbf{L}$, termed **fully summed columns** of the frontal matrix, a block of rows of $\mathbf{U}$, termed **fully summed rows**, and the computation of a Schur complement matrix $\mathbf{F}_{22}-\mathbf{F}_{21}\mathbf{F}_{11}^{-1}\mathbf{F}_{12}$, called a **contribution block**. The rows (columns) of the $\mathbf{F}_{22}$ block are referred to as **partly summed rows (columns)**.

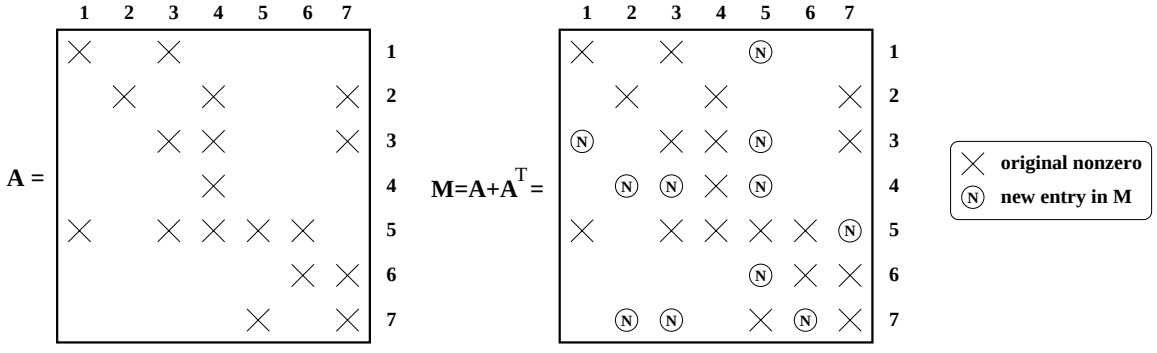

Figure 2: Example of matrix $\mathbf{A}$ and $\mathbf{M} = \mathbf{A} + \mathbf{A}^{\mathrm{T}}$.

The unsymmetric matrix $\mathbf{A}$, on the left-hand side of Figure 2, will be used to illustrate the

main properties of the assembly tree and to introduce the new algorithm. In Figures 2 and 3 an "X" denotes a nonzero position from the original matrix $\mathbf{A}$ and a "Ⓝ" corresponds to a new entry introduced during symmetrization. In Figure 3, we indicate the structure of the filled matrix $\mathbf{M}_F = \mathbf{L} + \mathbf{L}^T$ where $\mathbf{L}$ is the matrix of the Cholesky factor of $\mathbf{M}$. Entries with an "F" corresponds to fill-in entries in the $\mathbf{L}$ factor.

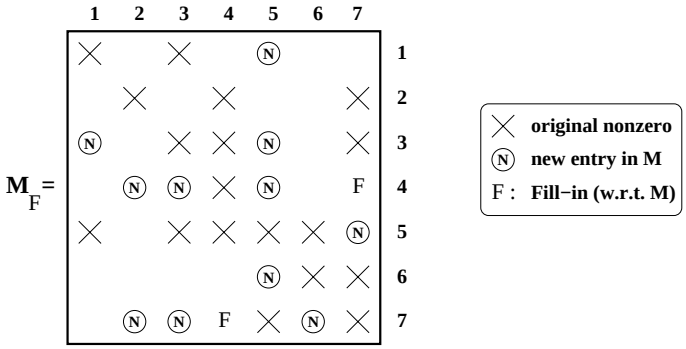

Figure 3: Structure of the Cholesky factors of the matrix $\mathbf{M}$.

The matrix $\mathbf{M}_F$ is used to define the assembly tree (see Figure 4) associated with the multifrontal $\mathbf{LU}$ factorization of the matrix $\mathbf{A}$. From the fact that the factorization is based on the assembly tree associated with the Cholesky factorization of $\mathbf{M}_F$, it results that

$$Struct(\mathbf{M}_F) = Struct(\mathbf{L}) + Struct(\mathbf{U}) \quad \text{and} \quad Struct(\mathbf{L}^T) = Struct(\mathbf{U})$$

where *Struct()* denotes the matrix pattern. Let us denote by **structural zero** a numerical zero that does not result from numerical cancellation. Typically, due to the symmetrization, the matrix $\mathbf{M}$ might contain many structural zeros that will propagate during the numerical factorization phase. What has motivated our work is the following question. Is it possible, during the processing of the assembly tree to efficiently detect and remove structural zeros that appear in matrix $\mathbf{M}_F$ and that are direct or indirect consequence of the symmetrization of matrix $\mathbf{A}$? Although it is not so clear from the structure of the matrix $\mathbf{M}_F$, we will show that blocks of structural zeros can be identified during the processing of the assembly tree.

In the following, we first describe how the assembly tree is exploited during the standard multifrontal algorithm. We then report and analyse the sparsity structure of the frontal matrices involved in the processing of the assembly tree associated with our example matrix. Based on these observations, we will introduce the new factorization algorithm.

The assembly tree is rooted (a node of the tree called the **root** is chosen to give an orientation to the tree) and is processed from the leaf nodes to the root node. If two nodes are adjacent in the tree, then the one nearer the root is the **parent node,** and the other is termed its **child**. Each edge of the assembly tree indicates a data dependency between parent and child. It involves sending a contribution block from the child to the parent. A parent node process will start when the processes associated with all of its children are completed.

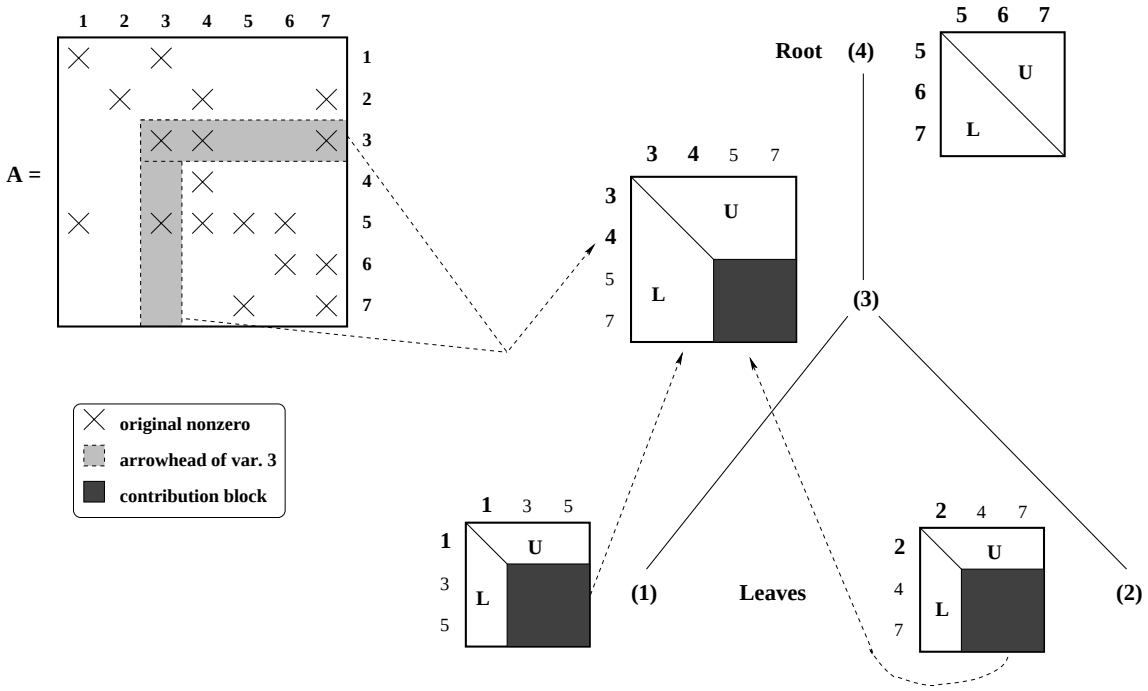

Figure 4: Assembly tree of assembly tree associated with our test matrix.

For example, in Figure 4, node **(3)** must wait for the completion of nodes **(1)** and **(2)** before starting its computations. The subset of variables which can be used as pivots (boldface variables in Figure 4) are the **fully summed** variables of node **(k)**. The contribution blocks of the children and the entries from the original matrix corresponding to the fully summed variables of node **(k)** are used to build the frontal matrix of the node. This will be referred to as the **assembly process**. During the assembly process of a frontal matrix, we need for each fully summed variable j, to access the nonzero elements in the original matrix that are in rows/columns of indices greater than j. A way to efficiently access the original matrix is to store it in arrowheads according to the reordered matrix. For example during the assembly process of node **(3)** the arrowheads of variables *3* and *4* from matrix **A** together with the contribution blocks of nodes **(1)** and **(2)** are used to assemble the frontal matrix of node **(3)**. One should note that, by construction, the list of indices in the partly summed rows is identical to that of the partly summed columns (row and column indices of block $\mathbf{F}_{22}$ in Figure 1). Therefore, during the assembly process, only the list of row indices of the partly summed rows is built. This list is obtained by merging all the row and column indices of the arrowheads of the matrix **A** with the row indices of the contribution blocks of all the sons. Once the structure of the frontal matrix is built, the numerical values from both the arrowheads and the contribution blocks can be assembled at the right place in the frontal matrix. The floating point operations involved during the assembly process will be referred to as **assembly operations** (only additions) whereas floating-point operations involved during the factorization of the frontal matrices will be referred to as **elimination operations**.

Partial threshold pivoting is used to control the element growth in the factors. Note that pivots can be chosen only from within the block $\mathbf{F}_{11}$ of the frontal matrix. The **LU**factors corresponding to the fully summed variables are computed and a new contribution block is produced. When a fully summed variable of node **(k)** cannot be eliminated during the node process because of numerical considerations, then the corresponding arrowhead in the frontal matrix is added to the contribution block and the fully summed variable will be included in the fully summed variables at the parent of node **(k)**. This process creates additional fill-in in the **LU**factors.

In a multifrontal algorithm, we have to provide space for the frontal matrices and the contribution blocks, and to reserve space for storing the factors. We need working space to store both real and integer information. This will be referred to as the **total working space** of the factorization phase. The same integer array can be used to describe a frontal matrix, its corresponding **LU**factors and its contribution block. The management of the integer working array can thus be done in a simple and efficient way. In a uniprocessor environment, it is possible to determine the order in which the assembly tree will be processed. Furthermore, if we process the assembly tree with a depth first search order, we can use a stack to manage the storage of the factors and the contribution blocks. This mechanism is efficient both in terms of total memory requirement and amount of data movement (see [12]). A stack mechanism, starting from the beginning of the real working array, is used to store the **LU** factors. Another stack mechanism starting from the end of the real working array is used to store the contribution blocks. After the assembly phase of a node the working space used by the contribution blocks of its children can be freed and, because the assembly tree is processed with a depth first search order, the contribution blocks will always be at the top of the stack. In the remainder of this paper, the maximum stack size of the contribution blocks will be referred to as the **maximum stack size**.

The standard and new algorithms for multifrontal factorization

During a multifrontal factorization, each frontal matrix can be viewed as the minimum structure to perform the elimination of the fully summed variables and to carry the contribution blocks from all of its sons. In Figure 5, we have a closer look at the frontal matrices involved in the processing of the assembly tree of Figure 4 to identify the structural zeros.

We report, beside each node, the structure of the factorized frontal matrix assuming that the pivots are chosen down the diagonal of the fully summed block and in order (i.e. no numerical pivoting is required). An "X" corresponds to a nonzero entry and a "O" corresponds to a structural zero.

One can see that, for our example, the frontal matrices have many structural zeros. There are two kinds of structural zeros: those forming a complete zero column (or row), and more isolated zero entries in a nonzero column or row (for example entries (4,3) and (4,7) in the

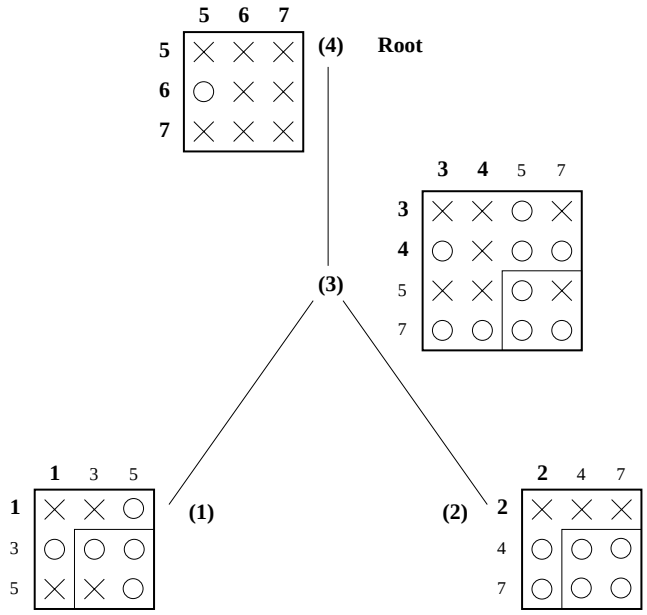

Figure 5: Processing the assembly tree associated with the matrix **A** in Figure 2 with the standard algorithm.

frontal matrix of node **(3)**). If one knows how to detect a partly summed row (or column) with only structural zeros then the corresponding row (or column) can be suppressed from the frontal matrix because this row (or column) will not add any contribution to the father node.

Structural zero rows (or columns) can be detected during the assembly process of a frontal matrix because of the following property: if a row (or column) index does not appear in the row (or column) indices both of the arrowheads of the original matrix and of the contribution blocks of the sons, then this index will correspond to a row (or column) with only structural zeros. This property is used to deduce the assembly process of the new algorithm. Note that if the matrix is not structurally deficient then each fully summed row (or column) must have at least one nonzero entry. Therefore, we can restrict our search for zero rows (columns) to the partly summed rows (columns).

In the new assembly algorithm, the list of indices of the partly summed rows of a frontal matrix is defined as the merge of the row indices in the arrowheads of the fully summed variables of the node with the row indices of the contribution blocks of its sons. The column indices are defined similarly. As it is illustrated in Figure 6, the new assembly process can result in significant modifications in the processing of the assembly tree. For example, on node **(1)**, row 3 and column 5 are suppressed from the frontal matrix; on node **(2)**, all the partly summed rows are suppressed; on node **(3)**, row 7 and column 5 are suppressed. As it can be noticed in Figure 6, frontal matrices naturally become unsymmetric in structure.

We finally indicate in Figure 7 the structure of the **LU** factors obtained with the new algorithm.

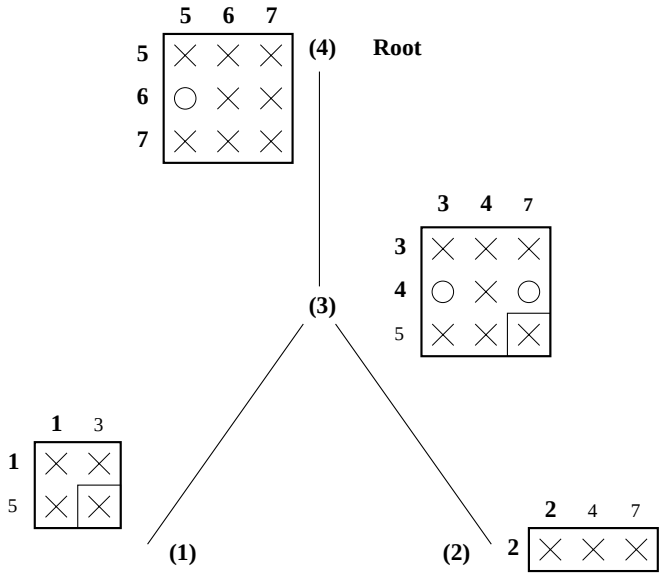

Figure 6: Processing the assembly tree associated with the matrix **A** in Figure 2 with the new algorithm.

This should be compared to the matrix $\mathbf{M}_F$ in Figure 3 showing the structure of the factors obtained with the standard algorithm. It can be seen that nonzero entries corresponding to

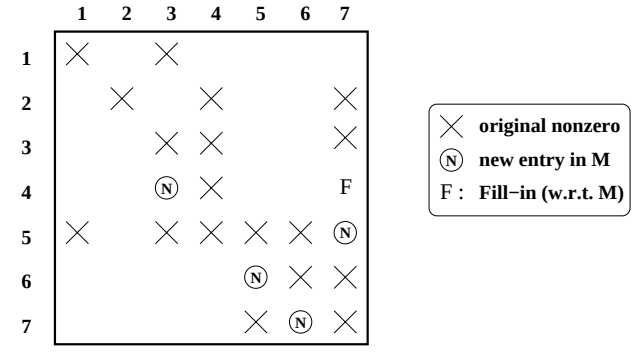

Figure 7: Structure of the **LU**factors obtained with the new algorithm.

fill-in (for example (7,4) in $\mathbf{M}_F$) or introduced during the symmetrization of $\mathbf{M}$ (for example (4,5) in $\mathbf{M}_F$) might be supressed by the new algorithm. On the other hand, the new algorithm will never suppress structural zeros in a block of fully summed variables (for example (4,3) in node (**3**) of Figure 5). On our small example, the total number of entries in the factors reduces from 31 to 23.

Comparing Figures 5 and 6, one can notice that the new algorithm might also lead to a significant reduction in both the number of operations involved during the assembly process and the maximum stack size. The latest combined with a reduction in the size of the factors will result

in a reduction in the total working space. On our example the number of assembly operations drops from 30 to 20 (18 entries from **A** plus 2 from the contribution blocks). The maximum stack size reduces from 8 to 1 (obtained in both cases after stacking the contribution blocks of nodes **(1)** and **(2)**).

3 Results and performance analysis

We describe in Table 1 the set of test matrices (order, number of nonzero entries, structural symmetry and origin). We define the structural symmetry as the percentage of the number of nonzeros matched by nonzeros in symmetric locations over the total number of entries. A symmetric matrix has a value of 100. Although, our performance analysis will focus on matrices with a relatively small structural symmetry, all classes of unsymmetric matrices are represented in this set. The selected matrices come from the forthcoming Rutherford-Boeing Sparse Matrix Collection [8] [1], Tim Davis collection[2], and SPARSEKIT2[3].

The Harwell Subroutine Library [16] code `ma41` has been used to obtain the results for the standard multifrontal method. The factorization phase of `ma41` has then been modified with the new algorithm. The `ma41` code has a set of parameters to control its efficiency. We have used the default values for our target computer. Approximate minimum degree ordering [1] has been used to reorder the matrix. As we have mentioned in the Introduction, it is often quite beneficial for very unsymmetric matrices to precede the ordering by performing an unsymmetric permutation to place large entries on the diagonal and then scaling the matrix so that the diagonal entries are all of modulus one and the off-diagonals have modulus less than or equal to one. We use the Harwell Subroutine Library code `mc64` [10] to perform this preordering and scaling on all matrices of structural symmetry smaller than 55. When `mc64` is not used, our matrices are always row and column scaled (each row/column is divided by its maximum value). All results presented in this section, have been obtained on one processor (R10000 MIMPS RISC 64-bit processor) of the SGI Cray Origin 2000 from Parallab (University of Bergen, Norway). The processor runs at a frequency of 195 Mhertz and has a peak performance of 400 Mflops per second.

[1] Web page `http://www.cse.clrc.ac.uk/Activity/SparseMatrices/`
[2] Web page `http://www.cise.ufl.edu/ davis/sparse/`
[3] Web page `http://iftp.cs.umn.edu/pub/sparse/`

Matrix name	Order	No. entries	StrSym	Origin (Discipline)
AV4408	4408	95752	0	Vavasis (Partial diff. eqn.) [21]
AV41092	41092	1683902	0	Vavasis (Partial diff. eqn.) [21]
BBMAT	38744	1771722	54	Rutherford-Boeing (CFD)
CAVITY15	2597	76367	94	SPARSEKIT2 (CFD)
CAVITY26	4562	138187	95	SPARSEKIT2 (CFD)
EX11	16614	1096948	100	SPARSEKIT2 (CFD)
FIDAPM11	22294	623554	100	SPARSEKIT2 (CFD)
GOODWIN	7320	324784	64	Davis (CFD)
LHR02	2954	37206	1	Davis (Chemical engineering)
LHR14C	14270	307858	1	Davis (Chemical engineering)
LHR17C	17576	381975	0	Davis (Chemical engineering)
LHR34C	35152	764014	0	Davis (Chemical engineering)
LHR71C	70304	1528092	0	Davis (Chemical engineering)
LNS_3937	3937	25407	87	Rutherford-Boeing (CFD)
OLAF1	16146	1015156	100	Davis (Structural engineering)
ONETONE1	36057	341088	10	Davis (Circuit simulation)
ONETONE2	36057	227628	15	Davis (Circuit simulation)
ORANI678	2529	90158	7	Rutherford-Boeing (Economics)
PSMIGR_1	3140	543162	48	Rutherford-Boeing (Demography)
RAEFSKY5	6316	168658	4	Davis (Structural engineering)
RAEFSKY6	3402	137845	2	Davis (Structural engineering)
RDIST1	4134	94408	6	Rutherford-Boeing (Chemical engineering)
RIM	22560	1014951	65	Davis (CFD)
SHERMAN5	3312	20793	78	Rutherford-Boeing (Oil reservoir simul.)
SHYY41	4720	20042	77	Davis (CFD)
SHYY161	76480	329762	77	Davis (CFD)
TWOTONE	120750	1224224	28	Davis (Circuit simulation)
UTM3060	3060	42211	56	SPARSEKIT2
UTM5940	5940	83842	56	SPARSEKIT2
WANG4	26068	177196	100	Rutherford-Boeing (Semiconductor)

Table 1: Test matrices. StrSym denotes the structural symmetry.

In the following graphs, we report the performance ratios of the new factorization algorithm over the standard algorithm. Matrices are sorted by increasing structural symmetry of the matrix to be factored, i.e. after application of the column permutation when `mc64` is used. We use the same matrix order in the graphs and in the complete set of results provided in Tables 2 and 3. In this way, one can easily find, given a point in the graph, its corresponding entry in the tables.

On the complete set of test matrices, we first analyse in Figure 8 what is probably of main concern for the user of a sparse solver, i.e. the time to factor the matrix and the total working space (as defined in the previous section). In Figure 8, we divide the matrices into three categories: matrices of structural symmetry smaller than 50 for which the time reduction is between 20% and 80%, matrices whose structural symmetry is between 50 and 80 for which the time reduction is between 3% and 20% and nearly structurally symmetric matrices for which there is almost no difference between the standard and new version. It is interesting to notice that even on symmetric matrices the added work to detect asymmetry does not affect much the

performance of the factorization. In the remainder of this paper, we will not report results on almost structurally symmetric matrices (symmetry greater than 80).

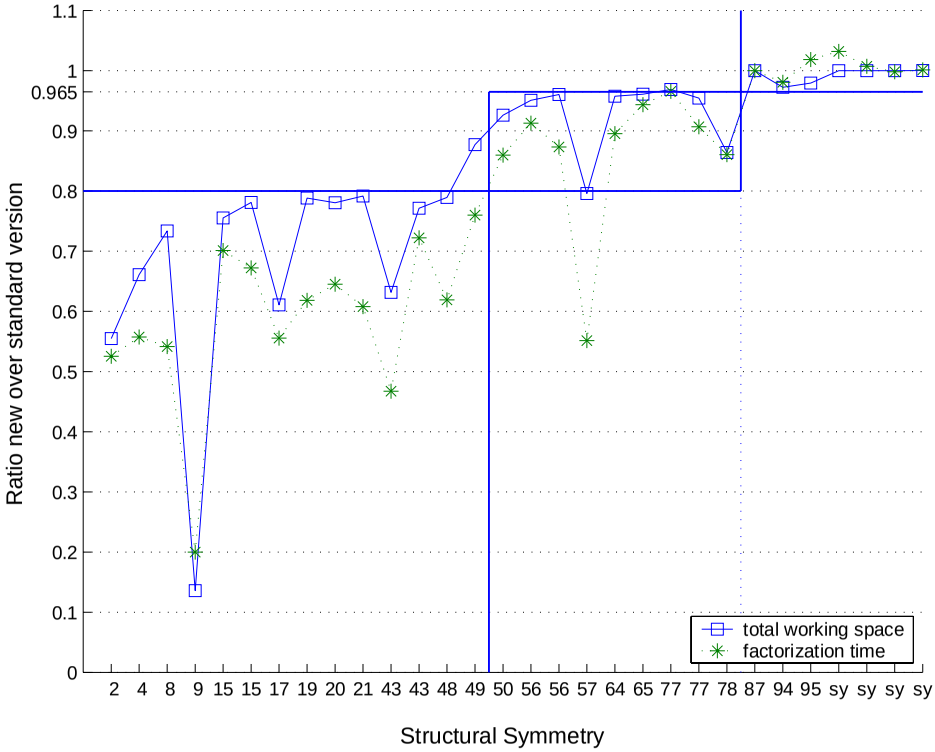

Figure 8: Study of the factorization time and the total working space. *sy* (in abscissa) corresponds to structurally symmetric matrices.

In Figure 9, we relate the gain in the factorization time with the reduction in the number of elimination operations and in the number of assembly operations. Although the number of operations due to the assembly is always much smaller that the number of operations involved during factorization (see Tables 2 and 3), the assembly process can still represent an important part of the time spent in the factorization phase (see for example [2]). This is illustrated in Figure 9 where we see that the high reduction in the number of assembly operations (more than 50%) significantly contributes to reducing the factorization time. Note that on a relatively large matrix (TWOTONE) of symmetry 57 still significant gains in time and in number of assembly operations (more than 40%) can be obtained. In Figure 10, we relate the total working space reduction to the size of the factors and to the maximum stack size. Although a reduction in the maximum size of the stack might not always introduce a reduction in the total working space, we see that in practice it is often the case. An extreme example of this reduction is matrix ORANI678 of symmetry 9 (see Table 2) for which maximum stack size is reduced by more than one order of magnitude (5482454 to 457312). Finally, we notice that a large reduction in the maximum stack size (Figure 10) will generally correspond to a large reduction in the number of assembly operations (Figure 9).

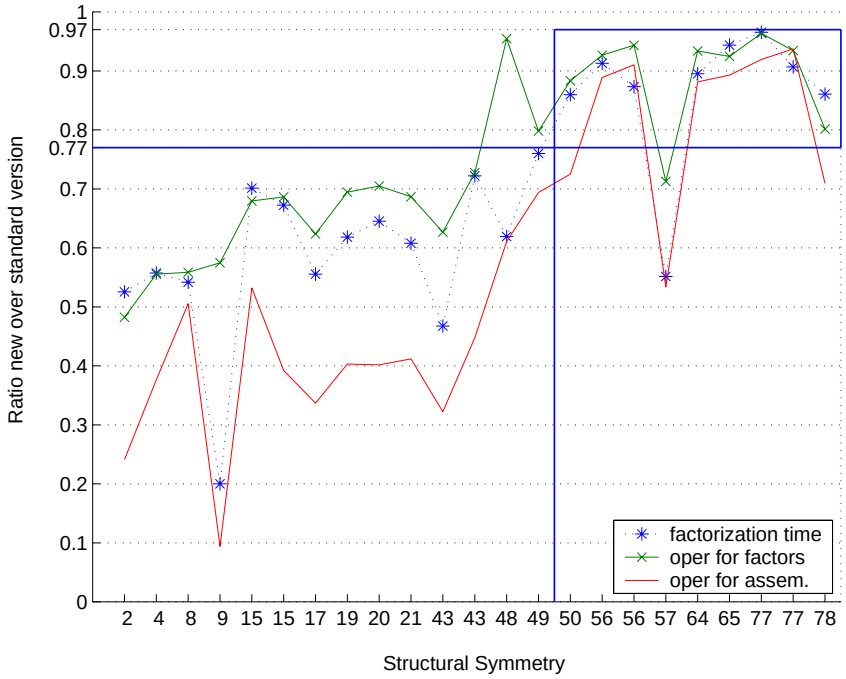

Figure 9: Impact of the reduction in the number of floating point operations on the time.

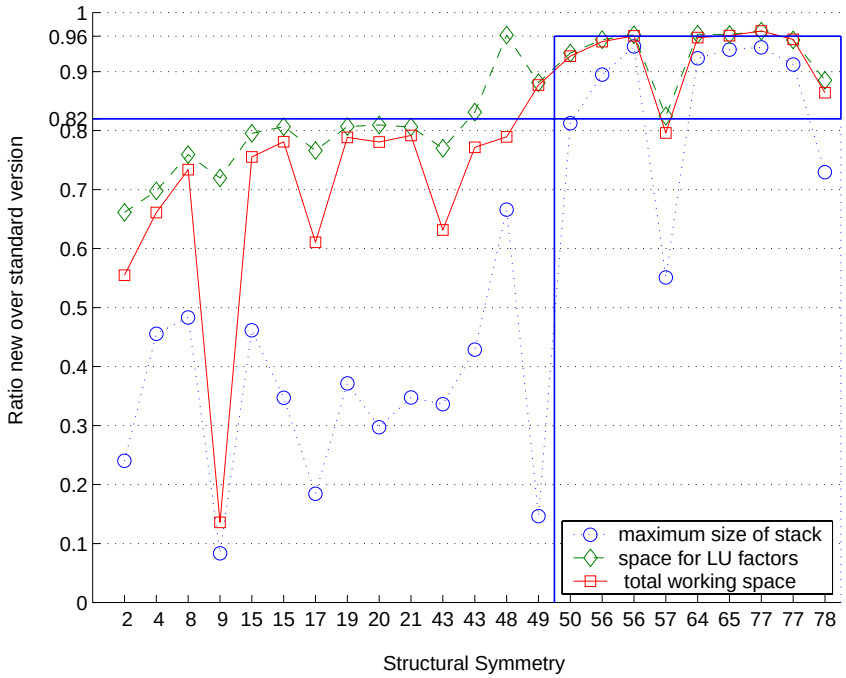

Figure 10: Correlation between the factor size, the maximum stack size, and the total working space.

Matrix (Str.Sym.)	Version	LU	stack	Total space	Operations in Elimin.	Assemb.	Facto. Time
RAEFSKY6	Stnd	1509016	606458	2017106	4.795E+08	4.348E+06	2.17
(**2**)	New	998064	145575	1119135	2.313E+08	1.049E+06	1.14
RAEFSKY5	Stnd	1757680	378792	2095082	3.746E+08	3.874E+06	1.83
(**4**)	New	1226376	172619	1385099	2.081E+08	1.459E+06	1.02
AV41092	Stnd	13947192	3875082	15955575	7.747E+09	4.688E+07	37.20
(**8**)	New	10589987	1872716	11706493	4.328E+09	2.369E+07	20.15
ORANI678	Stnd	422713	5482454	5803903	9.012E+07	8.122E+06	2.30
(**9**)	New	304199	457312	788170	5.179E+07	7.633E+05	0.46
AV4408	Stnd	551354	227787	677254	6.872E+07	1.451E+06	0.46
(**15**)	New	438530	105157	511453	4.670E+07	7.721E+05	0.32
LHR14C	Stnd	2167304	415090	2439502	2.092E+08	7.944E+06	1.77
(**15**)	New	1748270	144025	1905487	1.436E+08	3.115E+06	1.19
LHR02	Stnd	235048	137125	345145	1.332E+07	7.683E+05	0.18
(**17**)	New	180065	25268	210778	8.306E+06	2.588E+05	0.10
LHR34C	Stnd	5613656	755710	6249187	6.282E+08	2.064E+07	5.24
(**19**)	New	4529304	280891	4926554	4.362E+08	8.321E+06	3.24
LHR17C	Stnd	2813418	639172	3204801	3.089E+08	1.085E+07	2.65
(**20**)	New	2277484	189988	2501349	2.177E+08	4.359E+06	1.71
LHR71C	Stnd	11615170	729857	12711920	1.402E+09	4.304E+07	13.16
(**21**)	New	9364949	253508	10063754	9.628E+08	1.773E+07	8.00
TWOTONE	Stnd	22086166	15899616	34489449	2.933E+10	2.171E+08	183.84
(**43**)	New	17004794	5344194	21782217	1.838E+10	6.993E+07	85.92
ONETONE1	Stnd	4713485	3348215	6212037	2.282E+09	2.675E+07	14.54
(**43**)	New	3918207	1434965	4792473	1.660E+09	1.198E+07	10.50
PSMIGR_1	Stnd	6316254	12896617	18554060	9.313E+09	8.326E+07	84.07
(**48**)	New	6075412	8587331	14646034	8.889E+09	5.087E+07	52.05
RDIST1	Stnd	258096	53767	279999	8.150E+06	5.054E+05	0.13
(**49**)	New	227436	7865	245576	6.504E+06	3.507E+05	0.10

Table 2: Comparison of the standard (Stnd) and the new algorithms on matrices of structural symmetry < 50.

Matrix (Str.Sym.)	Version	LU	stack	Total space	Operations in Elimin.	Operations in Assemb.	Facto. Time
BBMAT	Stnd	44111480	8351266	48035816	3.676E+10	2.283E+08	185.75
(**50**)	New	41081573	6785219	44491215	3.247E+10	1.655E+08	159.68
UTM3060	Stnd	324640	78679	806970	2.683E+07	6.973E+05	0.22
(**56**)	New	309700	70390	775115	2.486E+07	6.198E+05	0.20
UTM5940	Stnd	701496	131839	2799224	6.640E+07	1.529E+06	0.51
(**56**)	New	675079	124245	2227948	6.264E+07	1.392E+06	0.45
ONETONE2	Stnd	2253553	898540	385816	5.085E+08	7.628E+06	3.88
(**57**)	New	1858514	495096	366856	3.624E+08	4.070E+06	2.14
GOODWIN	Stnd	1264140	307777	1385604	1.612E+08	2.841E+06	1.05
(**64**)	New	1217726	283920	1326773	1.505E+08	2.504E+06	0.94
RIM	Stnd	4127204	833290	4371615	5.648E+08	9.194E+06	3.37
(**65**)	New	3973769	780826	4200819	5.221E+08	8.209E+06	3.18
SHYY161	Stnd	7437816	377535	290589	9.945E+08	1.178E+07	6.56
(**77**)	New	7204304	355293	277352	9.583E+08	1.083E+07	6.33
SHYY41	Stnd	251336	28523	8137032	1.036E+07	6.337E+05	0.14
(**77**)	New	239696	26015	7882722	9.681E+06	5.940E+05	0.14
SHERMAN5	Stnd	167412	61227	217769	1.284E+07	4.414E+05	0.13
(**78**)	New	148176	44670	188168	1.029E+07	3.131E+05	0.10
LNS_3937	Stnd	285517	89578	335285	1.920E+07	5.482E+05	0.20
(**87**)	New	284874	89390	335328	1.914E+07	5.463E+05	0.21
CAVITY15	Stnd	202629	33004	230111	1.033E+07	3.453E+05	0.10
(**94**)	New	197553	31796	223789	9.896E+06	3.320E+05	0.12
CAVITY26	Stnd	394164	58589	447293	2.433E+07	6.877E+05	0.22
(**95**)	New	386700	57061	438054	2.358E+07	6.670E+05	0.25
EX11	Stnd	11981558	3960507	13614400	6.678E+09	3.836E+07	27.76
(**100**)	New	11981558	3960507	13614943	6.678E+09	3.836E+07	28.65
FIDAPM11	Stnd	15997220	4863371	19069150	9.599E+09	4.705E+07	39.78
(**100**)	New	15997220	4863371	19071689	9.599E+09	4.705E+07	40.07
OLAF1	Stnd	5880174	1506068	6794919	1.965E+09	1.685E+07	8.91
(**100**)	New	5880174	1506068	6795508	1.965E+09	1.685E+07	8.89
WANG4	Stnd	11561486	5063375	15900237	1.048E+10	4.087E+07	46.19
(**100**)	New	11561486	5063375	15906325	1.048E+10	4.087E+07	46.25

Table 3: Comparison of the standard (Stnd) and the new algorithms on matrices of structural symmetry >= 50

4 Concluding remarks

We have described a modification of the standard multifrontal **LU**factorization algorithm that can lead to a significant reduction in both the computational time and the total working space. The standard multifrontal algorithm [13] for unsymmetric matrices is based on the assembly tree of a symmetrized matrix and involves frontal matrices symmetric in structure. Therefore, it produces **LU** factors such that the matrix $\mathbf{F} = \mathbf{L} + \mathbf{U}$ is symmetric in structure. This approach is currently used in the context of two publically available packages (`ma41` [2, 3] and `MUMPS` [5, 4]) and has the advantage, with respect to other unsymmetric factorization algorithm [6, 7, 17], of having the **LU** factorization based on the processing of an assembly tree, while the other approaches explicitly or implicitly use a more complex to handle graph structure.

We have demonstrated that, based on the same assembly tree, one can derive a new multifrontal algorithm that will introduce asymmetry in the frontal matrices and in the matrix of the factors **F**. The detection of the asymmetry is only based on structural information and is not costly to compute as it has been illustrated on structurally symmetric matrices, for which both algorithms behave similarly. On a set of unsymmetric matrices, we have shown that the new algorithm will reduce both the factor size and the number of operations by a significant factor. We have also observed that the reduction in the number of indirect memory access operations involved during the assembly process is generally much higher than the reduction in the number of elimination operations. Finally, we have noticed that the gain in the maximum stack size is also relatively high and is comparable to the gain in the number of assembly operations.

	Space for			Operations		
	LU	Stack	Total	Elim	Assemb.	Time
$0 \leq$ Structural symmetry < 50						
mean	0.79	0.35	0.69	0.67	0.41	0.59
median	0.80	0.35	0.76	0.68	0.40	0.61
$50 \leq$ Structural symmetry < 80						
mean	0.93	0.85	0.93	0.89	0.82	0.86
median	0.95	0.91	0.95	0.93	0.89	0.90

Table 4: Performance ratios of the new algorithm over the standard algorithm.

To conclude, we report in Table 4 a summary of the results (mean and median) obtained on the test matrices with structural symmetry smaller than 80. For very unsymmetric matrices (structural symmetry smaller that 50), we obtain an average reduction of 31% in the total working space and of 41% in the factorization time. The maximum stack size and the number of assembly operations are reduced by respectively 65% and 59%. Finally, it is interesting to observe that, even on fairly symmetric matrices ($50 \leq$ structural symmetry < 80), it can still be worth trying to identify and exploit asymmetry during the processing of the assembly tree.

Acknowledgements. We want to thank Horst Simon and Esmond Ng who gave us the opportunity to work at NERSC (LBNL) for one year. We are grateful to Sherry Li for helpful comments on an early version of this paper.

References

[1] P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17:886–905, 1996.

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

[3] P. R. Amestoy and I. S. Duff. Memory management issues in sparse multifrontal methods on multiprocessors. *Int. J. of Supercomputer Applics.*, 7:64–82, 1993.

[4] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. Technical Report RT/APO/99/2, ENSEEIHT-IRIT, 1999. (submited to SIAM Journal on Matrix Analysis and Applications).

[5] P. R. Amestoy, I.S. Duff, and J.-Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Methods Appl. Mech. Eng.*, pages 501–520, 2000.

[6] T. A. Davis and I. S. Duff. An unsymmetric-pattern multifrontal method for sparse LU factorization. *SIAM Journal on Matrix Analysis and Applications*, 18:140–158, 1997.

[7] T. A. Davis and I. S. Duff. A combined unifrontal/multifrontal method for unsymmetric sparse matrices. *ACM Transactions on Mathematical Software*, 25(1):1–19, 1999.

[8] I. Duff, R. G. Grimes, and J. G. Lewis. The Rutherford-Boeing Sparse Matrix Collection. Technical Report RAL-TR-97-031, Rutherford Appleton Laboratory, 1997. Also Technical Report ISSTECH-97-017 from Boeing Information & Support Services and Report TR/PA/97/36 from CERFACS, Toulouse.

[9] I. S. Duff. Full matrix techniques in sparse Gaussian elimination. In G.A. Watson, editor, *Numerical Analysis Proceedings, Dundee1981*, Lecture Notes in Mathematics 912, pages 71–84, Berlin, 1981. Springer-Verlag.

[10] I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. Technical Report RAL-TR-1999-030, Rutherford Appleton Laboratory, 1999.

[11] I. S. Duff and J. K. Reid. MA27—a set of Fortran subroutines for solving sparse symmetric sets of linear equations. Technical Report R.10533, AERE, Harwell, England, 1982.

[12] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear systems. *ACM Transactions on Mathematical Software*, 9:302–325, 1983.

[13] I. S. Duff and J. K. Reid. The multifrontal solution of unsymmetric sets of linear systems. *SIAM Journal on Scientific and Statistical Computing*, 5:633–641, 1984.

[14] S. C. Eisenstat and J. W. H. Liu. Exploiting structural symmetry in unsymmetric sparse symbolic factorization. *SIAM Journal on Matrix Analysis and Applications*, 13:202–211, 1992.

[15] J. R. Gilbert and J. W. Liu. Elimination structures for unsymmetric sparse lu factors. *SIAM Journal on Matrix Analysis and Applications*, 14:334–352, 1993.

[16] HSL. A collection of Fortran codes for large scale scientific computation, 2000.

[17] X. S. Li and J. W. Demmel. A scalable sparse direct solver using static pivoting. In *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*, San Antonio, Texas, March 22–24, 1999.

[18] J. W. H. Liu. The role of elimination trees in sparse factorization. *SIAM Journal on Matrix Analysis and Applications*, 11:134–172, 1990.

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

[20] R. Schreiber. A new implementation of sparse Gaussian elimination. *ACM Transactions on Mathematical Software*, 8:256–276, 1982.

[21] S.A. Vavasis. Stable finite elements for problems with wild constraints. *SIAM Journal on Numerical Analysis*, 33:809–916, 1996.