A New Recursive Implementation of Sparse Cholesky Factorization

J. J. Dongarra* P. Raghavan†

Abstract

Consider the Cholesky factorization of a sparse symmetric positive definite matrix, $A = LL^T$. The first two steps use symbolic, graph-theoretic techniques to order A to reduce fill in L, and to determine the exact sparsity structure of L. The factor L is computed in a third "numeric factorization" step. The two leading schemes for numeric factorization are a blocked column-oriented scheme, and a multifrontal implementation. We propose a new recursive implementation that could be viewed as a hybrid of these two schemes. The new scheme seeks to efficiently access the memory hierarchy on modern computers by a simple recursion on a "supernodal tree" associated with L. Consider diagonal blocks in L numbered in post-order on the supernodal tree; now the recursive formulation is equivalent to processing a sequence of dense diagonal blocks in L from the top left to the bottom right. Unlike the multifrontal scheme, the new scheme does not require extra stack storage.

Key words: sparse matrix methods, sparse Cholesky, multifrontal factorization.

AMS subject classifications: 65F05,65F50.

1 Introduction

Consider the solution of a sparse linear system of the form $Ax = b$, where the matrix A is large, sparse, and symmetric positive definite. A "direct" solution method computes the Cholesky factorization $A = LL^T$ and then solves triangular systems $Ly = b$ and $L^T x = y$. The Cholesky factorization causes original zeroes in A to fill-in and become nonzero in L. The fill-in depends on the sparsity structure of A and not on actual numeric values. Consequently, the overall solution process is organized in the form of four steps. The first two steps are symbolic, i.e., use the graph of the matrix A (i) to compute a fill-reducing ordering or permutation of A, and (ii) to compute the nonzero structure of the Cholesky factor of the reordered A. The last two steps are numeric and involve factorization and triangular solution using static data-structures for the Cholesky factor (of the reordered matrix) obtained from step (ii).

The numeric factorization step is the most expensive; the symbolic steps typically require a small fraction of the time required for numeric factorization [1, 7, 6, 8, 9, 13, 15, 18]. A simple column-by-column implementation with columns in L stored using a standard sparse storage scheme is very inefficient on modern computers with deep cache-hierarchies. The inefficiency stems from indirect addressing and disregard for data-locality leading to both a larger number of cache-misses and average clock-cycles per operation. Hence, a "cache-cognizant" implementation of the numeric factorization step is critical for good performance.

Henceforth, let L be the Cholesky factor of the sparse matrix A after the application of a fill-reducing permutation. Cache-efficient numeric factorization schemes are based on exploiting *effectively dense* groups of columns within L. The computation can then be organized as a set of suitable matrix-matrix and matrix-vector operations. Optimized cache-efficient kernels are available for such matrix-matrix and matrix-vector operations through the Basic Linear Algebra Subroutines (BLAS)[3, 11]. The two leading schemes for sparse numeric factorization are a column-block approach and a multifrontal method [4, 5, 17, 20]. The two schemes differ in the data-movement and in the amount of

*Department of Computer Science, The University of Tennessee, Knoxville TN, 37996-1301, dongarra@cs.utk.edu.
†Department of Computer Science, The University of Tennessee, Knoxville TN, 37996-1301, padma@cs.utk.edu.

16th IMACS World Congress (© 2000 IMACS)

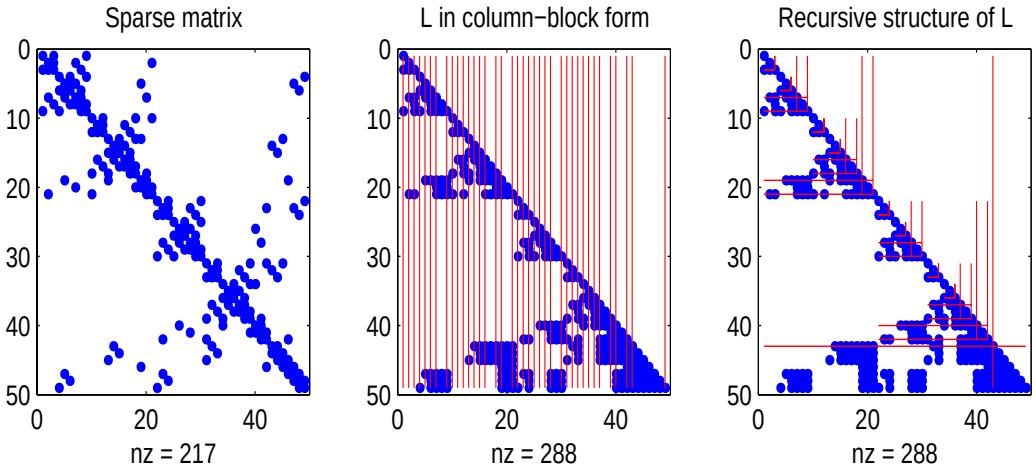

Figure 1: The structure of a sparse matrix and its Cholesky factor. The sparse matrix corresponds to a 7×7, 5-point finite-difference grid reordered to reduce fill; the structure of L can be viewed in the form of a column-block partition or a recursive partition.

temporary storage during factorization. The multifrontal scheme has a recursive control and updates to later columns from earlier columns are propagated through several dense submatrices that are on a stack (i.e., accessed in a "last in first out" manner). In our experience, the multifrontal schemes show slightly higher execution rates than column-block schemes. However, this higher performance comes at the expense significant memory overheads. The latter can be a substantial fraction of the total memory required for L. An intrinsic problem with sparse factorization methods is the non-linear growth of memory requirements as a function of the matrix dimension. This problem is further exacerbated with the extra memory requirements of a multifrontal scheme. We therefore explore a new formulation that tries to combine the best features of both column-block and multifrontal approaches without incurring the memory overheads of the latter.

Section 2 describes column-block and multifrontal schemes. In Section 3 we describe our new method. Section 4 outlines our plan of work.

2 Numeric Factorization Schemes

Consider the structure of the Cholesky factor of a sparse matrix reordered to reduce fill. An example is shown in Figure 1 for the sparse matrix of a model, five-point 7×7 finite-difference grid. This matrix is representative of the class of sparse matrices from finite-element and finite-difference applications and we use it for illustrative purposes throughout this paper. The columns of L can be grouped into "supernodes;" a supernode is a set of consecutive columns that have nested sparsity structure. The lower triangular matrix induced by the columns in a supernode is essentially dense in the subscripts of rows containing nonzeroes in the first (lowest numbered) column. For example, the last set of seven columns in Figure 1 form a supernode. In general, columns of L can be partitioned into supernodes easily [9, 16]. Columns within supernodes are treated as effectively dense submatrices and numeric factorization schemes can thus exploit dense-matrix techniques to compute a sparse Cholesky factorization. The sparsity structure of L can be viewed in terms of column-blocks or alternatively, in a recursive manner as shown in Figure 1.

As mentioned earlier, the two types of cache-efficient numeric factorizations are a column-block scheme and a multifrontal scheme. The two schemes differ in how they compute and apply updates to columns in a given supernode from columns in earlier supernodes. As a consequence of sparsity, columns in a supernode need not be updated by columns in all preceding supernodes. The data-dependence between supernodes is given by a supernodal tree. A supernode s can be updated only by columns in supernodes within the subtree rooted at s; the exact subtree is related to the row-subtree of the "elimination tree" of L [9, 16].

The column-block scheme of Ng and Peyton is a left-looking scheme [17]. The structure of L corresponds to columns numbered in a post-ordering of the supernodal-tree. Supernodes are processed from left-to-right in what amounts to a bottom-up computation on the supernodal tree. Consider columns in a supernode s; they are first updated by relevant supernodes to the left of s in L and in the supernodal subtree rooted at s. Next, a dense Cholesky

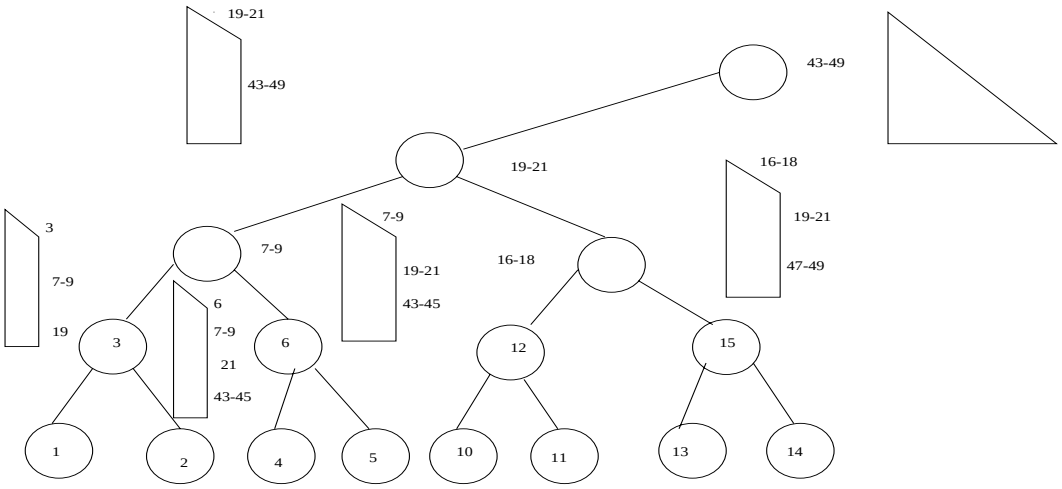

Figure 2: The column-block data-structure associated with nodes in the supernodal tree for the Cholesky factor of the sparse matrix shown in Figure 1.

factorization is computed for the diagonal block in supernode s. Finally, the sub-diagonal part of columns in supernode s is updated by the newly computed diagonal block. The computation involves a "scatter" followed by matrix-matrix BLAS operations. The dense matrix associated with a supernode is typically trapezoidal with more rows than columns. Figure 2 shows a supernodal tree for the example in Figure 1. In this small example several supernodes have only one column. However, for sparse matrices of interest, most supernodes will be considerably larger. In general, for an $N \times N$ sparse matrix associated with a two-dimensional (three-dimensional) finite-element mesh, the number of columns in the largest supernodes is proportional to $N^{1/2}$ ($N^{2/3}$) [12].

The multifrontal scheme of Duff et. al. [4, 5] differs from the column-block scheme in how updates are performed. Consider computation at a supernode s; all updates to columns in s are accumulated into matrices at supernodes whose "parent" is s. That is, updates to the supernode s from *all* earlier supernodes are stored in matrices at supernodes that are immediate descendants of s. This is recursively the case at every node in the supernodal tree. As a consequence, there is extra (temporary) storage associated with each supernode to propagate updates to the parent supernode. Let $struc(s)$ denote the rows containing nonzeroes in the first column of supernode s; the matrix at s is a $|struc(s)| \times |struc(s)|$ dense lower triangular matrix. Let $columns(s)$ be the set of columns contained in supernode s; the part of the matrix associated with $columns(s)$ is called the "new" part and comprised columns of L. The rest of the matrix is called the "update" part and contains the cumulative updates to ancestor supernodes of s; these are not retained after the factorization process is completed. The implementation of multifrontal factorization in a language such as C can be recursive [19]. The computation at a supernode s involves: (i) allocating storage for the triangular matrix at s, (ii) adding updates from matrices at each child node, (iii) computing the Cholesky factorization corresponding to columns in s, and (iv) accumulating updates to later supernodes in the update portion of the matrix s. Most of the computations can be done using a modified form of the dense Cholesky factorization routine available in LAPACK [2]. The implementation can be very efficient; however, this efficiency comes at the expense of additional storage for the update part of the matrices at each supernode. Figure 3 illustrates the data-structure at each supernode for the 7×7 mode grid example.

3 A New Recursive Formulation of Numeric Factorization

Our new formulation can be viewed as hybrid of column-block and multifrontal schemes. It attempts to formulate computations in terms of symmetric dense diagonal blocks computed from the top-left to the bottom-right in L. It can also be restated as a recursion on the supernodal tree or the recursive sparse structure of L. Our main goal is to obtain the high-performance of multifrontal schemes without paying the penalty of stack-memory overheads. As shown in Table 1, the memory overhead can be a significant fraction of the storage required for L. On average, the overhead is greater than one half the storage for L for matrices in the test-suite in Table 1. These matrices were ordered using the well-known Multiple Minimum degree scheme [14] and in several instances the overhead is as large as the storage for L.

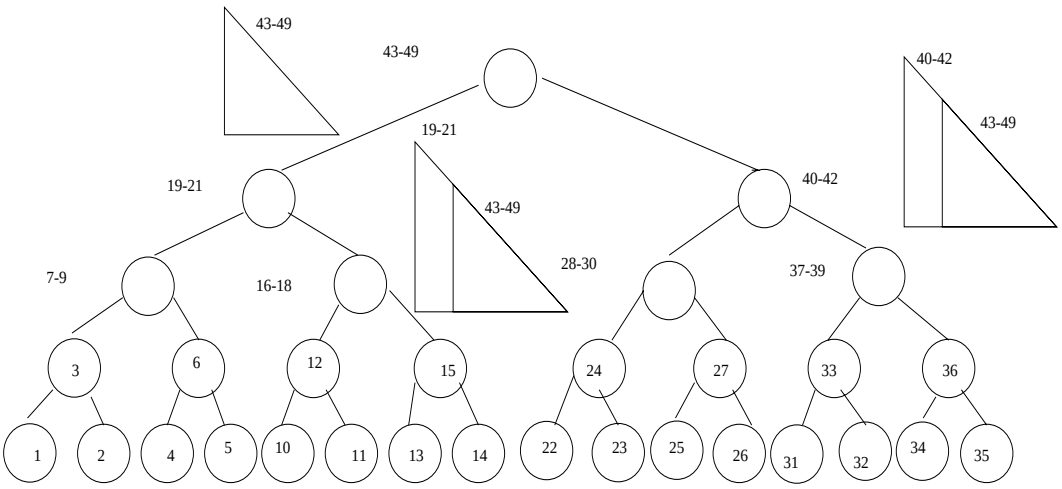

Figure 3: The dense triangular matrices associated with nodes in the supernodal tree in multifrontal factorization.

We also conjecture that a recursive control coupled with our new data-structures may lead to a more efficient implementation . Our conjecture is based on recent results for dense Cholesky factorization; a recursive implementation was shown to outperform a left-looking blocked implementation because of greater temporal locality of data coupled with enhanced reuse of data in cache [10].

Owing to the sparsity of L, our recursive scheme differs substantially from the one for dense Cholesky. Our recursion uses the natural recursive structure of L. The sparsity of L is a consequence of nested-dissection orderings as well as suitable supernodal post-orderings after computing fill-reducing permutations using greedy schemes. In general, the structure can be represented as follows:

$$\begin{bmatrix} L_{11} & 0 & {L_{s1}}^T \\ 0 & L_{22} & {L_{s2}}^T \\ L_{s1} & L_{s1} & L_{ss} \end{bmatrix}$$

The L_{11} block is $n_1 \times n_1$, the L_{22} block is $n_2 \times n_2$, and L_{ss} block is $n_s \times n_s$, where $n_1 + n_2 + n_s = N$ for sparse matrices of dimension N. Typically n_s is of the order of $N^{1/2}$ or $N^{2/3}$. The matrix L_{ss} is dense but the recursion continues with blocks L_{11} and L_{22}. Our new scheme can be stated recursively as follows:

- Recursively compute L_{11} and L_{22}.
- Compute L_{1s} as $L_{11} \times L_{s1}^T = A_{s1}^T$.
- Compute L_{2s} as $L_{22} \times L_{s2}^T = A_{s2}^T$.
- Update L_{ss} by L_{1s} and L_{2s}.
- Compute a dense Cholesky factorization of L_{ss}.

An alternate but equivalent formulation can be stated in terms of a recursion on the supernodal tree. Unlike column-block and multifrontal schemes which use a single matrix at each supernode, we associate a linked list of dense matrices with each supernode. The recursion will be implemented using the supernodal tree and and the sequence of dense matrices at each supernode. The update sequence for a supernode s will be maintained using information from "row-subtrees" much as in left-looking column-block Cholesky. For the 7×7 model grid matrix, the recursive sparsity structure of L is shown in the rightmost matrix in Figure 1; our data-structures are shown Figure 4.

4 Plan of Work

We are in the process of implementing our new recursive formulation. We will report on the performance of our scheme on a variety of modern high-performance uniprocessor architectures such as the Intel Pentium, Sun Ultra-Sparc and RISC processors from IBM and SGI. We will also present comparisons with column-block and multifrontal implementations.

Table 1: Memory overheads for a multifrontal factorization of sparse matrices ordered using Multiple Minimum Degree; on average, the overhead is 58.8% of the memory required for storing the Cholesky factor.

matrix	rank	$\|A\|$ (10^3)	$\|L\|$ (10^6)	L-memory (10^6 bytes)	Stack-memory (10^6 bytes)	(%) L-memory
nasa2910	2910	88.60	.204	1.84	1.23	67
bcsstk24	3562	81.74	.294	2.51	1.65	66
bcsstk28	4410	111.72	.366	3.11	1.57	51
crystk01	4875	160.38	.991	8.26	8.47	102
bcsstk38	8032	181.75	.764	6.60	4.65	70
msc10848	10848	620.31	2.078	17.32	10.15	59
bcsstk17	10974	219.81	1.030	8.90	5.61	63
bcsstk18	11948	80.52	.680	6.26	4.83	77
vibrobox	12328	157.01	2.002	16.93	20.38	120
crystk02	13965	491.27	4.867	40.08	38.69	96
crystm02	13965	168.44	1.584	13.81	4.61	33
bcsstk25	15439	133.84	1.436	12.91	7.44	58
msc23052	23052	582.87	2.864	24.08	11.32	47
bcsstk36	23052	583.10	2.764	23.29	11.98	51
crystk03	24696	887.94	11.930	97.52	106.20	21
crystm03	24696	304.23	3.789	32.41	9.23	28
bcsstk37	25503	583.24	2.867	24.30	6.95	29
bcsstk35	30237	740.20	2.920	24.80	6.75	27
ct20stif	52329	1326.31	1.0695	88.97	59.89	67
nasasrb	54870	1366.10	11.961	99.77	45.69	46
cfd1	70656	948.12	32.435	267.32	148.58	56
mean						58.8

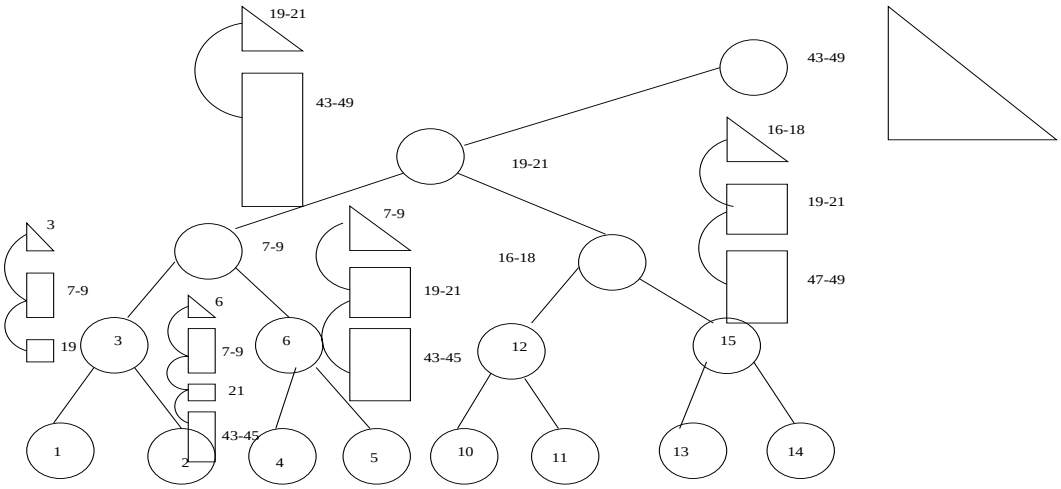

Figure 4: In our recursive formulation, the data-structure associated with a node in the supernodal tree consists of a sequence of dense matrices.

References

[1] P. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl.*, 17:886–905, 1996.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. SIAM Publications, 2nd edition, 1995.

[3] J. J. Dongarra, J.D. Croz, S. Hammarling, and I. S. Duff. An extended set of basic linear algebra subprograms. *ACM Trans. Math. Software*, 14:1–17, 1988.

[4] I.S. Duff and J.K. Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Trans. Math. Software*, 9:302–325, 1983.

[5] I.S. Duff and J.K. Reid. The multifrontal solution of unsymmetric sets of linear equations. *SIAM J. Sci. Stat. Comput.*, 5:633–641, 1984.

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

[7] J. A. George. Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.*, 10:345–363, 1973.

[8] J. A. George and J. W-H. Liu. An automatic nested dissection algorithm for irregular finite element problems. *SIAM J. Numer. Anal.*, 15:1053–1069, 1978.

[9] J.R. Gilbert, E. Ng, and B.W. Peyton. An efficient algorithm to compute row and column counts for sparse Cholesky factorization. *SIAM J. Matrix Anal. Appl.*, 15:1075–1091, 1994.

[10] F. G. Gustavson. Recursion leads to automatic variable blocking for dense linear-algebra algorithms. *IBM Journal of Research and Development*, 46(6), 1997.

[11] C. L. Lawson, R.J. Hanson, D.R. Kincaid, and F.T. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Software*, 5:308–323, 1979.

[12] R. J. Lipton, D. J. Rose, and R. E. Tarjan. Generalized nested dissection. *SIAM J. Numer. Anal.*, 16:346–358, 1979.

[13] J. W-H. Liu. Modification of the minimum degree algorithm by multiple elimination. *ACM Trans. Math. Software*, 11:141–153, 1985.

[14] J. W-H. Liu. Modification of the minimum degree algorithm by multiple elimination. *ACM Trans. Math. Software*, 11:141–153, 1985.

[15] Joseph W. H. Liu. The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. Appl.*, 11:134–172, 1990.

[16] J.W.H. Liu, E. Ng, and B.W. Peyton. On finding supernodes for sparse matrix computations. *SIAM J. Matrix Anal. Appl.*, 14:242–252, 1993.

[17] E. G. Ng and B. W. Peyton. Block sparse Cholesky algorithms on advanced uniprocessor computers. *SIAM J. Sci. Comput.*, 14:1034–1056, 1993.

[18] E. G. Ng and P. Raghavan. The performance of greedy ordering heuristics for sparse Cholesky factorization. *SIAM J. Matrix Anal. Appl.*, 20(4):902–914, 1999.

[19] P. Raghavan. DSCPACK: A Domain-Separator Cholesky Package for solving sparse linear systems on uniprocessor, multiprocessors and NOWs, 1999. Available upon request.

[20] E. Rothberg and A. Gupta. An evaluation of left-looking, right-looking, and multifrontal approaches to sparse Cholesky factorization on hierarchical-memory machines. *Int. J. High Speed Comput.*, 5:537–593, 1993.