Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate

YANQING CHEN, TIMOTHY A. DAVIS, WILLIAM W. HAGER,
and SIVASANKARAN RAJAMANICKAM
University of Florida

CHOLMOD is a set of routines for factorizing sparse symmetric positive definite matrices of the form A or AA^T, updating/downdating a sparse Cholesky factorization, solving linear systems, updating/downdating the solution to the triangular system $Lx = b$, and many other sparse matrix functions for both symmetric and unsymmetric matrices. Its supernodal Cholesky factorization relies on LAPACK and the Level-3 BLAS, and obtains a substantial fraction of the peak performance of the BLAS. Both real and complex matrices are supported. CHOLMOD is written in ANSI/ISO C, with both C and MATLAB™ interfaces. It appears in MATLAB 7.2 as `x=A\b` when `A` is sparse symmetric positive definite, as well as in several other sparse matrix functions.

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*Linear systems (direct methods), Sparse structured, and very large systems*; G.4 [**Mathematical Software**]: —*Algorithm design and analysis, Efficiency*

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: Cholesky factorization, linear equations, sparse matrices

ACM Reference Format:
Chen, Y., Davis, T. A., Hager, W. W., and Rajamanickam, S. 2008. Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. ACM Trans. Math. Softw. 35, 3, Article 22 (October 2008), 14 pages. DOI = 10.1145/1391989.1391995. http://10.1145/1391989.1391995.

This work was supported by the National Science Foundation, under grants 0203270, 0620286, and 0619080.
Authors' addresses: T. A. Davis, Department of Computer and Information Science and Engineering, University of Florida, Gainesville, FL; email: davis@cise.ufl.edu; W. W. Hager, Department of Mathematics, University of Florida, Gainesville, FL; email: hager@math.ufl.edu.

1. OVERVIEW

Methods for the direct solution of a sparse linear system $Ax = b$ rely on matrix factorizations. When A is symmetric positive definite, sparse Cholesky factorization is typically used. Supernodal and multifrontal methods are among those that can exploit dense matrix kernels (the BLAS [Lawson et al. 1979; Dongarra et al. 1988, 1990]) and can thus achieve high performance on modern computers. For more background on direct methods for sparse matrices, see Davis [2006]; George and Liu [1981]; and Duff et al. [1986]. Supernodal methods are discussed by Ashcraft and Grimes [1999]; Dobrian et al. [2000]; Hénon et al. [2002]; Ng and Peyton [1993]; Rothberg and Gupta [1991]; and Rotkin and Toledo [2004].

CHOLMOD is a package that provides sparse Cholesky factorization methods and related sparse matrix functions. It includes both a supernodal method and a nonsupernodal up-looking method [Davis 2005] that does not exploit the BLAS. If the original matrix A is replaced by $A \pm WW^T$, where W is n-by-k with $k << n$, the package can update or downdate the factorization while exploiting *dynamic supernodes*. An update or downdate of a conventional supernodal structure is not feasible, since changes in the nonzero pattern of L cause supernodes to merge and split apart. Reorganizing a conventional supernodal structure of L would dominate the run time of the update/downdate. Dynamic supernodes, in contrast, are detected and exploited as the algorithm progresses. For a rank-1 update/downdate, the running time is proportional to the number of entries in L that change; the time for a rank-k update/downdate is proportional to the time for k separate rank-1 update/downdates [Davis and Hager 1999, 2001]. CHOLMOD also exploits dynamic supernodes in the triangular solves, and can update/downdate the corresponding solution to $Lx = b$ after updating/downdating L itself. A detailed discussion of how CHOLMOD uses dynamic supernodes is given in Davis and Hager [2009].

Section 2 summarizes the features in CHOLMOD. The performance of its sparse Cholesky factorizations and nested dissection ordering methods is given in Section 3. Section 4 explains how to obtain the code and the packages it relies on.

2. FEATURES

CHOLMOD is composed of a set of modules, each of which defines a set of objects and/or operations on those objects. CHOLMOD's modules (Core, Cholesky, Check, Demo, MATLAB™, MatrixOps, Modify, Partition, and Supernodal) are described in the following. In addition, CHOLMOD includes a full user guide that documents all of its 134 user-callable functions, a Makefile, and an exhaustive test suite that exercises all of the statements in the code. Nearly all matrix operations can be performed on either symmetric or unsymmetric matrices, except that only symmetric matrices A or AA^T can be factorized and only a symmetric LDL^T factorization can be updated/downdated.

2.1 Core Module

The Core Module defines the five basic objects that CHOLMOD supports:

(1) `cholmod_sparse`: a sparse matrix in compressed sparse column form, either symmetric or unsymmetric (the latter may be rectangular). In the symmetric case, either the upper or lower triangular part is stored. Most of CHOLMOD's functions operate on `cholmod_sparse` matrices.
(2) `cholmod_triplet`: a sparse matrix in *triplet* form (a list of nonzero values with their row and column indices). This is a flexible data structure for the user to generate. Only a few functions are provided (including functions for converting to and from the triplet form).
(3) `cholmod_dense`: a dense matrix in column-oriented form (compatible with MATLAB and Fortran).
(4) `cholmod_factor`: a sparse Cholesky factorization, either supernodal or nonsupernodal, and either LL^T or LDL^T (where D is diagonal).
(5) `cholmod_common`: CHOLMOD parameters, workspace, and statistics.

The first four objects can be real or complex (both double precision), or pattern only (with no numerical values). CHOLMOD supports two forms of complex matrices and factors: C/C++/Fortran-style, and the split-style used in MATLAB. The Core Module provides functions to allocate, reallocate, and free these objects. The Core can also copy all but the `cholmod_common` object, and convert between different object types.

Lower case letters are used for subsets (`f`) or permutation vectors (`p`); `alpha` and `beta` are scalars. Sparse matrix operators (for `cholmod_sparse`) in the Core Module include (in MATLAB notation) `A*A'`, `A(:,f)*A(:,f)'`, `alpha*A+beta*B`, `tril`, `triu`, `diag`, `A'`, `A(p,f)'`, and `A(p,p)'`. The Core Module includes a function to sort row indices within each sparse column vector and a function to extract entries within a band.

The `cholmod_factor` object contains either a symbolic or numeric factorization, in either LL^T or LDL^T form, and either supernodal or nonsupernodal. The Core Module includes a function that converts a `cholmod_factor` between these various forms.

2.2 Cholesky Module

The Cholesky Module provides functions for computing or using a sparse Cholesky factorization. These functions compute the elimination tree of `A` or `A*A'` and its postordering, row/column counts of `chol(A)` or `chol(A*A')`, the symbolic factorization of `A` or `A*A'`, an interface to a supernodal Cholesky factorization, nonsupernodal LL^T and LDL^T factorizations (up-looking), incremental LL^T and LDL^T factorizations (one row at a time), a *symbolic refactorization* (to remove entries after an update/downdate), and solutions to upper/lower triangular systems (supernodal, dynamic supernodal, and dense, sparse, and/or multiple right-hand sides). The module also provides an interface to the AMD [Amestoy et al. 1996] and COLAMD [Davis et al. 2004] ordering methods. The methods used in this Module are discussed by Liu

[1990]; Gilbert et al. [1994, 2001]; George and Liu [1981]; and Davis [2005, 2006].

If the Supernodal Module is installed, the sparse Cholesky factorization automatically selects between a nonsupernodal up-looking factorization, and a supernodal BLAS-based factorization. An up-looking Cholesky factorization computes L one row at a time, and is much faster than the supernodal method for very sparse matrices, such as tridiagonal matrices. The selection is done during symbolic analysis, when $|L|$ and the floating-point operation count are found.[1] If the operation count divided by $|L|$ is greater than or equal to 40, then the supernodal method is selected. Otherwise, the nonsupernodal method is selected. This ratio is a coarse metric of how much memory-reuse a supernodal method will be able to exploit during numeric factorization. If it is high, then the supernodes will tend to be large and the BLAS routines will be efficient. If it is low, there will be little scope for efficient exploitation of cache within the BLAS kernels. The use of this ratio was selected analytically, based on a model of cache memory behavior, but the default threshold of 40 was selected based on performance measurements on a large range of sparse matrices arising in real applications (see Section 3).

If the Partition Module is not installed, CHOLMOD uses a minimum degree ordering (AMD or COLAMD). Otherwise, it automatically selects between minimum degree and nested dissection. The default strategy is to first try AMD. If AMD finds an ordering where the floating point operation divided by $|L|$ is less than 500, or if $|L| < 5a$ where a is the number of nonzeros in the lower triangular part of A, then the AMD ordering is used and no other ordering is attempted. Otherwise, METIS Karypis and Kumar [1998] or CHOLMOD's nested dissection ordering (NESDIS, described in Section 2.8) is tried, and the best ordering (the one with the lowest $|L|$) is used.

CHOLMOD can also be requested to try up to nine different orderings and select the best one found (user-provided, natural, AMD/COLAMD, METIS, NESDIS, and the latter three with varying nondefault parameter selections). This is useful in the case where a sequence of matrices with identical nonzero patterns must be factorized, as often occurs in nonlinear solvers, optimization methods, eigensolvers, and many other applications. These options are made available to the user as simple-to-modify nondefault parameter settings; the user need not call CHOLMOD multiple times.

Note that MATLAB 7.2 includes neither the Partition Module nor METIS. It thus always uses AMD for its ordering in `x=A\b` when `A` is sparse and symmetric positive definite.

Our sparse Cholesky update/downdate methods do not remove entries from the factor L, but only add them. After many update/downdates, it may be useful to prune entries that would not appear in L if it were computed from scratch. We do this with what we call a symbolic refactorization; it is just the same as a symbolic factorization, except that it takes as input, the A that L should be a factor of, and prunes entries that need not appear in L.

[1] $|L|$ denotes the number of nonzeros in L, or `nnz(L)` in MATLAB notation.

The symbolic refactorization step is also used to convert a supernodal L (which has extra numerically-zero entries due to relaxed supernodal amalgamation) into a concise nonsupernodal L where these extra entries do not appear.

In an incremental LL^T or LDL^T factorization, only some of the rows of L are computed. Additional rows of L are computed incrementally, as needed. We use this method in LPDASA when solutions to subproblems are required [Davis and Hager 2007b, 2007a].

2.3 Check Module

The Check Module checks the validity of the five CHOLMOD objects, and prints their contents. It can also read a sparse matrix from a file, in triplet or Matrix Market form [Boisvert et al. 1997].

2.4 Demo Module

The Demo Module provides sample main programs to illustrate how CHOLMOD can be used in a user's application. The CHOLMOD User Guide has more details.

2.5 MATLAB Module

The MATLAB Module is CHOLMOD's interface to MATLAB, providing most of CHOLMOD's functionality to the MATLAB environment. Note that MATLAB 7.2 already includes much of CHOLMOD itself, as built-in functions (namely, the Core, Cholesky, MATLAB, and Supernodal Modules). Built-in functions that rely on CHOLMOD include backslash (`x=A\b` and `x=b/A` when `A` is sparse and symmetric positive definite), `chol`, `etree`, and `symbfact`.

CHOLMOD provides additional functions to the MATLAB user that are not built into MATLAB, including nested dissection orderings (METIS and NESDIS), and the ability to update and downdate a sparse LDL^T factorization. Its sparse-matrix-times-dense-matrix function, `sdmult`, is significantly faster than the corresponding operation in MATLAB 7.2. Its variant of the `sparse` function is much faster than the MATLAB built-in version, sometimes by a factor of 10 or more.

2.6 MatrixOps Module

The MatrixOps Module provides functions for the `cholmod_sparse` object, including `[A,B]`, `[A;B]`, `alpha*A*X+beta*Y` and `alpha*A'*X+beta*Y`, where `A` is sparse and `X` and `Y` are dense, `A*B`, and `A(i,j)`, where `i` and `j` are arbitrary integer vectors. It can also compute norms of sparse or dense matrices. It can perform row and column scaling, and can drop small entries from a sparse matrix.

2.7 Modify Module

The Modify Module provides functions that can add or delete a row of `L` from an LDL^T factorization and compute the dynamic supernodal update/downdate of

an LDL^T factorization. The solution to $Lx = b$ can also be updated/downdated when L is modified. Details of the methods used are given by Davis and Hager [1999, 2001, 2009]. For background on update/downdate methods, see Gill et al. [1974]; Hager [1989], and Stewart [1979, 1998].

2.8 Partition Module

The Partition Module provides graph-partitioning based orderings. The graph partitioning methods are currently based on METIS. Thus, this Module can only be used in CHOLMOD if METIS is also available.

The Partition Module includes an interface to three constrained minimum degree ordering methods: CAMD, CCOLAMD, and CSYMAMD. These methods are constrained versions of AMD, COLAMD, and SYMAMD, respectively. In a constrained minimum degree ordering method, each node is in one of up to n constraint sets. All nodes in constraint set zero are ordered first, followed by all nodes in set one, and so on. These ordering methods are most useful when combined with nested dissection [Pellegrini et al. 2000; Rothberg and Eisenstat 1998].

The Module provides a direct interface to METIS functions for computing a node separator of an undirected graph (METIS_NodeComputeSeparator), and for computing the nested dissection ordering of an undirected graph (METIS_NodeND). The latter recursively partitions the graph via node separators until the subgraphs are small. Small subgraphs are ordered independently with minimum degree (MMD, [Liu 1985]), not with a constrained minimum degree ordering.

CHOLMOD also includes its own nested dissection ordering, NESDIS. It uses METIS_NodeComputeSeparator to partition the graph recursively, in the same manner as METIS_NodeND. Unlike METIS_NodeND, which orders each subgraph independently, NESDIS uses the separators as ordering constraints for CAMD or CCOLAMD (for symmetric and unsymmetric matrices, respectively). This can sometimes result in a better ordering at the expense of a modest increase in ordering time. Section 3 compares the performance of CHOLMOD's ordering methods.

2.9 Supernodal Module

The Supernodal Module provides supernodal symbolic and numeric factorization (LL^T only) of A or AA^T, and a conventional supernodal triangular solver. The functions in this Module are normally accessed indirectly by the user, through the Cholesky Module.

3. PERFORMANCE

This section highlights the performance of CHOLMOD's sparse Cholesky factorization methods, and compares its nested dissection ordering with METIS.

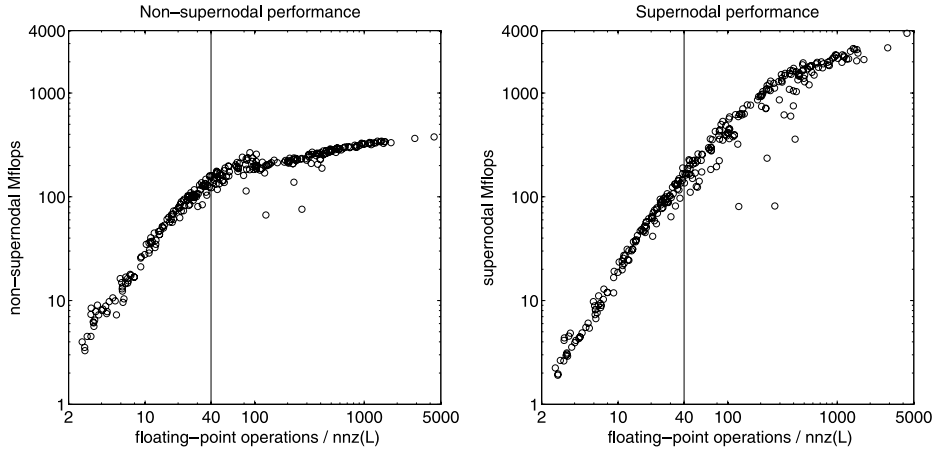

Fig. 1. CHOLMOD supernodal and nonsupernodal performance.

3.1 Sparse Cholesky factorization

3.1.1 *MATLAB Backslash.* MATLAB 7.1 uses a left-looking nonsupernodal method and the SYMMMD minimum degree ordering [Gilbert et al. 1992]. For large matrices, sparse backslash when `A` is symmetric positive definite, is typically five to ten times faster in MATLAB 7.2 as compared with MATLAB 7.1, because of CHOLMOD's supernodal factorization and AMD ordering. CHOLMOD's up-looking nonsupernodal factorization, coupled with the better and faster AMD ordering, provides a speedup even for small or very sparse matrices. For large matrices, a speedup of up to 40 has been observed. For the ND/ND3K matrix, a 3D mesh, the time for `x=A\b` reduces from 178.6 seconds in MATLAB 7.1 to 10.7 seconds in MATLAB 7.2. With the Partition Module and the METIS nested dissection ordering, the time to solve $Ax = b$ with CHOLMOD drops to 8.5 seconds. Backslash does not use METIS, nor can it be configured to do so by the MATLAB user. METIS is available only if the MATLAB user installs both CHOLMOD and METIS, and uses the CHOLMOD mexFunction instead of backslash.

3.1.2 *CHOLMOD Supernodal/Nonsupernodal Tradeoff.* Figure 1 shows the performance of CHOLMOD's nonsupernodal up-looking method and its supernodal method, as a function of the ratio of floating-point operations over the number of nonzeros in L. Figure 2 compares the relative performance of these two methods. The computer listed in the first row of Table I was used for Figures 1 and 2, using the Goto BLAS [Goto and van de Geijn 2008]. The test set of 320 matrices consists of all matrices in the UF Sparse Matrix Collection [Davis 1994] that are either positive definite or binary with a symmetric nonzero pattern. For the binary matrices, numerical values were constructed to ensure the matrices were positive definite; off-diagonal entries were set to -1, and the diagonal entry a_{ii} was set to one plus the number of nonzero entries in column i. Random matrices were excluded. The x-axis of each plot is

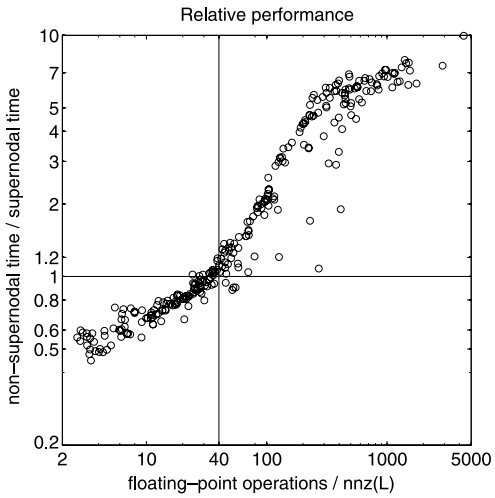

Fig. 2. CHOLMOD relative supernodal and nonsupernodal performance.

the floating-point operation count over $|L|$, which is the metric used to automatically select between the two methods. Run times include ordering, analysis, factorization, and solution of the resulting triangular systems. This is all the work done by `x=A\b`, except for the `mldivide` meta-algorithm that selects which method to use (LU, Cholesky, QR, banded solver, etc.). Each circle in the figure is a single matrix.

These results indicate that the floating-point count per nonzero in L is a remarkably accurate method for predicting both the absolute and relative performance of these two methods. The relative performance, and more importantly the threshold of 40 flops/$|L|$ used to automatically select between the up-looking and supernodal method, may of course differ on different computers. The value of 40 is the default value of a parameter that the user can modify. Reasonable thresholds on a range of other computers are shown in Table I, based on the same test set. On all platforms listed, the flops/$|L|$ ratio proves to be an accurate performance predictor.

On dual-core processors, a high flops/$|L|$ ratio is required to warrant the use of a multi-threaded BLAS, as compared to a single-threaded BLAS (about 300 on the Intel T2500, and 500 on the AMD Opteron). This is due to performance bugs in the Intel MKL and AMD ACML multithreaded BLAS libraries. This bug also results in a slowdown in the sparse component of the MATLAB `bench` when multithreading is enabled (MATLAB Versions 7.4 and 7.5) [Moler 2007]. Future versions of the BLAS should attain the same flops/$|L|$ threshold of about 40, since a multithreaded BLAS should never be significantly slower than a single-threaded BLAS for any size matrix.

Note that the Pentium 4M and Intel Core Duo T2500 [Gochman et al. 2006] are laptop processors, where power and thermal constraints are met at the expense of floating-point performance. All other computers in Table I use desktop or server processors.

Table I. Performance of CHOLMOD Sparse Cholesky Factorization on a Range of Computers

Computer	threshold	up-looking peak GFlops	supernodal peak GFlops
Intel Pentium 4			
3.2GHz, 512KB cache, 4GB RAM			
Goto BLAS	40	0.38	3.76
Intel Pentium 4M			
2GHz, 512KB cache, 1GB RAM			
Intel MKL BLAS	40	0.20	1.77
Goto BLAS	40	0.20	1.80
Intel Core Duo T2500 (2-core)			
2GHz, 2MB cache, 2GB RAM			
Intel MKL BLAS (1 thread)	40	0.41	1.34
Goto BLAS (1 thread)	40	0.41	1.47
Goto BLAS (2 threads)	120	0.41	2.47
AMD Opteron 252 (2-core)			
2.6GHz, 1MB cache, 8GB RAM			
ACML BLAS (1 thread)	30	0.38	2.56
Goto BLAS (1 thread)	30	0.38	3.93
Goto BLAS (2 threads)	80	0.38	6.65
Sun UltraSparcII V9+vis (4-core)			
450GHz, 4MB cache, 4GB RAM			
ATLAS BLAS (1 thread)	45	0.06	0.43

3.1.3 *Performance Comparisons by Gould, Hu, and Scott.* Full details of an independent performance comparison between CHOLMOD and ten other solvers are given in [Gould et al. 2007, 2006]. With 87 symmetric positive definite test matrices, CHOLMOD had the fastest total run time (including analysis, factorization, and solution of the triangular systems) for 42 matrices, and its run time was either fastest or within 10% of the fastest for 73 out of 87 matrices.

Gould, Hu, and Scott define a matrix as *large* if it has dimension larger than 50,000; there are 42 large matrices in their test set. Considering just the larger matrices, it was either the fastest or within 10% of the fastest for 40 out of 42 matrices. The two remaining matrices[2] are the two worst-case outliers in Figure 1. These two matrices are from an interior point linear programming problem and contain many dense rows and columns that cause problems with many solvers. CHOLMOD's run time was 2.2 and 3.8 times the fastest run time for these two matrices, respectively. The bulk of the time was spent in AMD. Modifying AMD's default dense row/column control parameter can greatly reduce the ordering time for these two matrices.

CHOLMOD used the least amount of memory (or within 10% of the least) for 35 of the 42 larger matrices. Scott and Hu [2007] also discuss the design considerations for a sparse direct code and give details on the user interface and design of CHOLMOD.

In many applications, the factorization and solve time is more important than the total time, since the analysis is done just once for a sequence of matrices. In this case, CHOLMOD's default parameters can be modified to

[2]The GUPTA1 and GUPTA2 matrices.

instruct it to call all three ordering methods available to it: AMD, METIS, and NESDIS. Alternatively, user-provided orderings can be given to CHOLMOD. For 30 of the 42 larger matrices using its default ordering strategy, CHOLMOD called both AMD and METIS and took the best ordering. For the other 12, AMD gave a reasonable ordering and so by default, METIS was not attempted. If METIS were used for these 12 matrices, the fill-in would be reduced for seven matrices and significantly reduced by more than 10% for only two matrices.

Ignoring the ordering and analysis time and considering just the factor+solve time, CHOLMOD was fastest for just 5 out of the 42 larger matrices, but it was within 10% of the fastest for 21 matrices. The total CHOLMOD factor+solve time for all 42 matrices was 2009 seconds, which is the second lowest among the 11 methods. This total time could be reduced if CHOLMOD were to be instructed to use all of its ordering methods and pick the best. The lowest total factor+solve time was obtained by WSMP [Gupta et al. 2001, 1997] at 1910 seconds, or just 5% lower than CHOLMOD. The WSMP analysis phase is designed to minimize the factor+solve time at the expense of ordering time. If the total analysis+factor+solve time is considered, CHOLMOD takes 2320 seconds for all 42 matrices, whereas WSMP takes 3062 seconds (32% more than CHOLMOD) because of its costly ordering and analysis phase.

These results by Gould, Hu, and Scott point out some of the strengths and weaknesses of CHOLMOD relative to other solvers. Comparing the performance of solvers is fraught with pitfalls: the matrix test sets used may not reflect a user's application (even though the test matrices arise in real applications) and the default parameters and ordering options for different solvers are tuned for different goals. However, with these caveats, these results do indicate that CHOLMOD has competitive performance relative to other solvers.

3.2 Ordering Methods in CHOLMOD

The results in this section demonstrate the improvement in fill-in that can be obtained by combining nested dissection and minimum degree ordering.

The default ordering strategy for CHOLMOD is to first try AMD (or COLAMD for the Cholesky factorization of AA^T). Let f be the floating point operation count as determined by AMD. Let a be the number of nonzeros in the lower triangular part of the symmetric matrix A. Then if $f/|L| \geq 500$ and $|L| \geq 5a$, METIS is tried and the best ordering is used—AMD or METIS, whichever gives the lowest $|L|$. If one or both of these conditions do not hold, then AMD has returned an ordering with low fill-in and flop count, and METIS is not attempted at all. This strategy is useful because while METIS can give good orderings, it is very costly in terms of run time and thus should be used only when AMD reports a high fill-in.

Our test set consists of 137 symmetric positive definite matrices. These are all matrices in the UF Sparse Matrix Collection [Davis 1994] that are either symmetric positive definite, or binary with symmetric nonzero pattern. Binary matrices were given values to make them positive definite, since symmetric binary matrices in the test set normally arise from symmetric positive definite

matrices whose values have not been contributed to the collection.[3] Random matrices and matrices with fewer than one million entries were also excluded. Six large matrices were too large to be solved on the computer used for the test and were also excluded (we used the computer in the first row of Table I). Of these 137 matrices, 108 come from applications with a 2D or 3D geometry. Nested dissection orderings (METIS and NESDIS) are typically better than minimum degree (AMD) for 3D problems, and often tie in quality with AMD for 2D problems.

Five different methods were used:

(1) The default strategy (AMD, then try METIS if warranted).
(2) Always try both AMD and METIS, and take the best ordering.
(3) AMD alone.[4] AMD failed for one of the 137 matrices.
(4) METIS alone.
(5) NESDIS alone. NESDIS failed for one of the 137 matrices.

The run time and flop count performance profiles of these five methods are shown in Figure 3.

In the left plot, the x-axis is the ratio of the total run time (ordering, symbolic factorization, numeric factorization, and forward/backsolve), divided by the best total run time for all of the five methods. The y-axis is the fraction of matrices for which a given method obtained that relative run time, or better. For example, the point $(2, 0.84)$ on the run-time plot for AMD means that for 84% of the matrices, the total run time when using AMD was no higher than twice the lowest run time of any of the five methods. The default method is the absolute fastest for only 20% of the matrices, but its run time is no more than 10% higher than the fastest method for 78% of the matrices. Our default strategy clearly dominates the run time of the other four strategies.

The right plot in Figure 3 is a performance profile for the same experiment, except the x-axis is the flop count relative to the best flop count for any method (AMD, METIS, or NESDIS). NESDIS can sometimes find slightly better orderings than METIS, but as can be seen in Figure 3, the total run time with NESDIS is higher because NESDIS is slower than METIS. AMD is a very fast method, but it does not always give the best orderings.

We can conclude from these results that the default ordering strategy is the best approach. The faster AMD/COLAMD minimum degree ordering routines should be used first. If CHOLMOD finds that AMD/COLAMD does not lead to a reasonable ordering quality, then the slower but sometimes better METIS or NESDIS should be tried. NESDIS can sometimes give slightly better orderings than METIS, but at a cost of higher run time. Thus, NESDIS is not used in the default ordering for CHOLMOD. CHOLMOD can be configured via parameter settings to try many different orderings and select the best one, which is useful

[3]Refer to Section 3.1.2 for how binary matrices were given numerical values.
[4]This is the ordering method used in MATLAB's backslash because (as of Version 7.5) MATLAB does not include the METIS ordering.

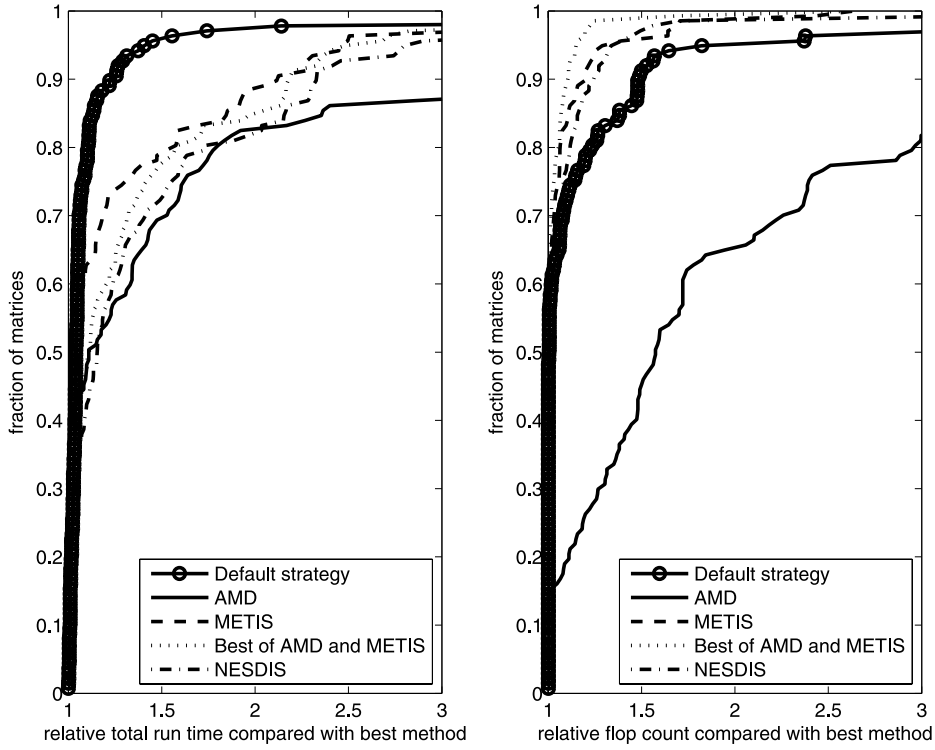

Fig. 3. Performance profiles of five CHOLMOD ordering strategies.

for the scenario where multiple matrices with identical nonzero patterns are to be factorized.

4. AVAILABILITY AND DEPENDENCIES

In addition to appearing as a Collected Algorithm of the ACM, CHOLMOD is available as a built-in function in MATLAB version 7.2 or later. It requires the ordering functions AMD, COLAMD, CAMD, and CCOLAMD, the dense matrix libraries LAPACK [Anderson et al. 1999] and the BLAS [Lawson et al. 1979; Dongarra et al. 1988, 1990], and optionally uses METIS [Karypis and Kumar 1998] for its nested dissection orderings.

REFERENCES

AMESTOY, P. R., DAVIS, T. A., AND DUFF, I. S. 1996. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl. 17*, 4, 886–905.

ANDERSON, E., BAI, Z., BISCHOF, C. H., BLACKFORD, S., DEMMEL, J. W., DONGARRA, J. J., DU CROZ, J., GREENBAUM, A., HAMMARLING, S., MCKENNEY, A., AND SORENSEN, D. C. 1999. *LAPACK Users' Guide*, 3rd ed. SIAM, Philadelphia.

ASHCRAFT, C. C. AND GRIMES, R. G. 1999. SPOOLES: an object-oriented sparse matrix library. In *Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing*. SIAM, Philadelphia.

BOISVERT, R. F., POZO, R., REMINGTON, K., BARRETT, R., AND DONGARRA, J. J. 1997. The Matrix Market: A Web resource for test matrix collections. In *Quality of Numerical*

Software, Assessment and Enhancement, R. F. Boisvert, Ed. Chapman & Hall, London, 125–137. (http://math.nist.gov/MatrixMarket).

DAVIS, T. A. 1994. University of Florida sparse matrix collection. www.cise.ufl.edu/research/sparse. NA Digest, vol 92, no. 42.

DAVIS, T. A. 2005. Algorithm 849: A concise sparse Cholesky factorization package. *ACM Trans. Math. Softw. 31*, 4, 587–591.

DAVIS, T. A. 2006. *Direct Methods for Sparse Linear Systems*. SIAM, Philadelphia, PA.

DAVIS, T. A., GILBERT, J. R., LARIMORE, S. I., AND NG, E. G. 2004. A column approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw. 30*, 3, 353–376.

DAVIS, T. A. AND HAGER, W. W. 1999. Modifying a sparse Cholesky factorization. *SIAM J. Matrix Anal. Appl. 20*, 3, 606–627.

DAVIS, T. A. AND HAGER, W. W. 2001. Multiple-rank modifications of a sparse Cholesky factorization. *SIAM J. Matrix Anal. Appl. 22*, 997–1013.

DAVIS, T. A. AND HAGER, W. W. 2007a. Dual multilevel optimization. *Math. Program. 112*, 2 (Nov.), 403–425.

DAVIS, T. A. AND HAGER, W. W. 2007b. A sparse proximal implementation of the LP Dual Active Set Algorithm. *Math. Program. 112*, 2 (Nov.), 275–301.

DAVIS, T. A. AND HAGER, W. W. 2009. Dynamic supernodes in sparse Cholesky update/downdate and triangular solves. *ACM Trans. Math. Softw.*, to appear.

DOBRIAN, F., KUMFERT, G. K., AND POTHEN, A. 2000. The design of sparse direct solvers using object oriented techniques. In *Advances in Software Tools for Scientific Computing*, H. P. Langtangen, A. M. Bruaset, and E. Quak, Eds. Lecture Notes in Computational Science and Engineering, vol. 10. Springer-Verlag, Berlin, 89–131.

DONGARRA, J. J., DU CROZ, J., DUFF, I. S., AND HAMMARLING, S. 1990. A set of level-3 basic linear algebra subprograms. *ACM Trans. Math. Softw. 16*, 1, 1–17.

DONGARRA, J. J., DU CROZ, J., HAMMARLING, S., AND HANSON, R. J. 1988. An extended set of Fortran basic linear algebra subprograms. *ACM Trans. Math. Softw. 14*, 18–32.

DUFF, I. S., ERISMAN, A. M., AND REID, J. K. 1986. *Direct Methods for Sparse Matrices*. OUP, Oxford, UK.

GEORGE, A. AND LIU, J. W. H. 1981. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall, Englewood Cliffs, New Jersey.

GILBERT, J. R., LI, X. S., NG, E. G., AND PEYTON, B. W. 2001. Computing row and column counts for sparse QR and LU factorization. *BIT 41*, 4, 693–710.

GILBERT, J. R., MOLER, C., AND SCHREIBER, R. 1992. Sparse matrices in MATLAB: design and implementation. *SIAM J. Matrix Anal. Appl. 13*, 1, 333–356.

GILBERT, J. R., NG, E. G., AND PEYTON, B. W. 1994. An efficient algorithm to compute row and column counts for sparse Cholesky factorization. *SIAM J. Matrix Anal. Appl. 15*, 4, 1075–1091.

GILL, P. E., GOLUB, G. H., MURRAY, W., AND SAUNDERS, M. A. 1974. Methods for modifying matrix factorizations. *Math. Comp. 28*, 126, 505–535.

GOCHMAN, S., MENDELSON, A., NAVEH, A., AND ROTEM, E. 2006. Introduction to the Intel core duo processor architecture. *Intel Techn. J. 10*, 2, 89–98.

GOTO, K. AND VAN DE GEIJN, R. 2008. High performance implementation of the level-3 BLAS. *ACM Trans. Math. Softw. 35*, 1.

GOULD, N. I. M., HU, Y., AND SCOTT, J. A. 2006. Complete results from a numerical evaluation of sparse direct solvers for the solution of large sparse, symmetric linear systems of equations. Tech. Rep. Internal report 2005-1 (revision 2), CCLRC, Rutherford Appleton Laboratory. (Mar.)

GOULD, N. I. M., HU, Y., AND SCOTT, J. A. 2007. A numerical evaluation of sparse direct solvers for the solution of large sparse, symmetric linear systems of equations. *ACM Trans. Math. Softw. 33*, 2 (June), 32 pages.

GUPTA, A., JOSHI, M., AND KUMAR, V. 2001. WSMP: A high-performance serial and parallel sparse linear solver. Tech. Rep. RC 22038 (98932), IBM T.J. Watson Research Center.

GUPTA, A., KARYPIS, G., AND KUMAR, V. 1997. Highly scalable parallel algorithms for sparse matrix factorization. *IEEE Trans. Para. Distrib. Syst. 8*, 5, 502–520.

HAGER, W. W. 1989. Updating the inverse of a matrix. *SIAM Rev. 31*, 2, 221–239.

HÉNON, P., RAMET, P., AND ROMAN, J. 2002. PaStiX: A high-performance parallel direct solver for sparse symmetric definite systems. *Para. Comput. 28*, 2, 301–321.

KARYPIS, G. AND KUMAR, V. 1998. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput. 20*, 359–392.

LAWSON, C. L., HANSON, R. J., KINCAID, D. R., AND KROGH, F. T. 1979. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Softw. 5*, 308–323.

LIU, J. W. H. 1985. Modification of the minimum-degree algorithm by multiple elimination. *ACM Trans. Math. Softw. 11*, 2, 141–153.

LIU, J. W. H. 1990. The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. Appl. 11*, 1, 134–172.

MOLER, C. June 2007. Cleve's corner: parallel MATLAB: multiple processors and multiple cores. *MATLAB News and Notes*.

NG, E. G. AND PEYTON, B. W. 1993. Block sparse Cholesky algorithms on advanced uniprocessor computers. *SIAM J. Sci. Comput. 14*, 5, 1034–1056.

PELLEGRINI, F., ROMAN, J., AND AMESTOY, P. R. 2000. Hybridizing nested dissection and halo approximate minimum degree for efficient sparse matrix ordering. *Concurrency: Pract. Exp. 12*, 68–84.

ROTHBERG, E. AND EISENSTAT, S. C. 1998. Node selection strategies for bottom-up sparse matrix orderings. *SIAM J. Matrix Anal. Appl. 19*, 3, 682–695.

ROTHBERG, E. AND GUPTA, A. 1991. Efficient sparse matrix factorization on high-performance workstations—exploiting the memory hierarchy. *ACM Trans. Math. Softw. 17*, 3, 313–334.

ROTKIN, V. AND TOLEDO, S. 2004. The design and implementation of a new out-of-core sparse Cholesky factorization method. *ACM Trans. Math. Softw. 30*, 1, 19–46.

SCOTT, J. A. AND HU, Y. 2007. Experiences of sparse direct symmetric solvers. *ACM Trans. Math. Softw. 33*, 3.

STEWART, G. W. 1979. The effects of rounding error on an algorithm for downdating a Cholesky factorization. *J. Inst. Math. Appl. 23*, 203–213.

STEWART, G. W. 1998. *Matrix algorithms, Volume 1: Basic decompositions*. SIAM, Philadelphia.

Received September 2006; revised March 2008; accepted March 2008