Science & Technology Facilities Council

Technical Report

A note on a simple constrained ordering for saddle-point systems

J. A. Scott

February 2009

RAL-TR-2009-007

Library and Information Services
SFTC Rutherford Appleton Laboratory
Harwell Science and Innovation Campus
Didcot
OX11 0QX
UK
Tel: +44 (0)1235 445384
Fax: +44(0)1235 446403
Email: library@rl.ac.uk

ISSN 1358-6254

A note on a simple constrained ordering for saddle-point systems[1]

by

Jennifer A. Scott

Abstract

A well-known problem with sparse direct solvers is that, if numerical pivoting is required, the number of entries in the computed factors can be significantly greater than the number predicted on the basis of the sparsity pattern alone. In this note, we review a simple constrained ordering recently proposed by Bridson [1] for saddle-point systems. Bridson's approach allows the factorization to be computed without numerical pivoting but numerical experiments show that the computed factors are generally significantly denser than those obtained by prescaling the matrix and then using an unconstrained ordering combined with threshold partial pivoting.

Keywords: large sparse symmetric linear systems, direct solvers, pivoting, scaling.

---

[1] This work was supported by the EPSRC grant EP/E053351/1.

Computational Science and Engineering Department,
Atlas Centre, Rutherford Appleton Laboratory,
Oxon OX11 0QX, England.

1 Introduction

It is well-known that it is challenging to design direct solvers to efficiently and accurately solve linear systems $Kx = b$ when K is of the form

$$K = \begin{pmatrix} A & B^T \\ B & C \end{pmatrix}, \tag{1.1}$$

with A symmetric positive definite, B rectangular, and C symmetric positive semi-definite. Matrices of the form (1.1) are often called saddle-point matrices or, in the special case $C = 0$, KKT matrices, in reference to the Karush-Kuhn-Tucker first-order necessary optimality conditions for the solution of general nonlinear programming problems. KKT matrices arise in equality and inequality constrained nonlinear programming, sparse optimal control, and mixed finite-element discretizations of partial differential equations.

For indefinite systems, direct methods compute a matrix factorization LDL^T of a permutation of K, where L is unit lower triangular and D is block diagonal with blocks of order 1 or 2. The solution process is completed by performing forward and back substitutions (that is, by first solving a lower triangular system and then an upper triangular system). An analyse phase normally chooses the permutation to minimise the number of entries in L and works with the pattern of the matrix only. For stability, it is generally necessary either to incorporate numerical pivoting within the factorization process or to modify the matrix being factorized (or a combination of both). Important disadvantages of numerical pivoting are that it adds a logistical overhead and since pivots that do not satisfy the threshold pivot test are delayed (that is, used later than expected in the elimination order), there can be significant fill in the matrix factor beyond that predicted by the analyse phase. This is particularly true for KKT systems (see, for example, [4, 7]). On the other hand, modifying the matrix generally necessitates some kind of iterative process to try and recover the required accuracy but convergence of such a process is not guaranteed. This has led to interest in constrained orderings that are sufficient to guarantee the existence of the LDL^T factorization.

The aim of a constrained ordering is to find a permutation Q such that QKQ^T can be factorized stably without the need for numerical pivoting and without modifying the entries in K, while still limiting the number of entries in L. This problem has been examined for special classes of matrices by a number of authors. Of practical interest is the class of $\mathcal{F}$ matrices, where each column of B has exactly two entries which sum to zero and $C = 0$. These arise in, for example, Stokes flow problems. Tuma [18] and De Niet and Wubs [3] have presented methods for these problems and report positive results. Most recently, Bridson [1] has proposed a constrained ordering for more general saddle-point problems where A is definite and B has full rank.

In this note, we review the method of Bridson and illustrate its effectiveness for saddle-point problems arising from practical applications. Our experiments show that, while the method generally works in practice, it can lead to many more entries in the matrix factor than are obtained by prescaling of the linear system and then using partial pivoting.

2 Simple constrained ordering

We use the terminology of Bridson [1] and divide the nodes of the adjacency graph of the matrix K into two disjoint sets: those that correspond to the diagonal entries of A are known as A-nodes and the remaining nodes as C-nodes. The ordering constraint proposed by Bridson [1] is extremely simple: a C-node can only be ordered after all its A-node neighbours in the graph of K have been ordered. Bridson shows that, provided A is definite and B is of full row rank, with this ordering the LDL^T factorization exists. Moreover, the pivots associated with the A-nodes are guaranteed to be positive and those associated with C-nodes are guaranteed to be negative. By rescaling, $L \leftarrow L|D|^{1/2}$ and $D \leftarrow sign(D) = diag(\pm 1)$, the diagonal matrix is fully determined in advance by the structure of the problem, independent of numerical values. Bridson refers to this as the *signed Cholesky* factorization of K. It allows him to modify a Cholesky factorization

code to perform the factorization of the indefinite matrix K with **no** numerical pivoting. A stability analysis is lacking but Bridson reports that numerical experiments indicate the constrained ordering is generally sufficient to avoid numerical pivoting. The hope is that, if an initial ordering is chosen to reduce fill in L, the constrained ordering will be sufficiently close that the additional fill will be modest. If it is close to the initial ordering, the benefits include a potentially faster factorization and, importantly, the analyse phase of the direct solver can accurately predict the size of the factors and other data structures required during the numerical factorization.

Bridson proposes two approaches to computing a constrained ordering. The first modifies the minimum degree algorithm (or one of its variants) to incorporate the constraint within it. An alternative approach is to post process a given fill-reducing ordering to satisfy the constraint. If a C-node is the next node in the supplied ordering it is only included in the modified ordering once all its A-node neighbours have been ordered (that is, a C-node is postponed until after all its A-node neighbours). For many large problems, orderings based on nested dissection are frequently recommended in preference to those based on minimum degree (see, for example [8]). The advantages of the post processing approach are that it can be applied to any fill-reducing ordering and it is very cheap and straightforward to implement. Since Bridson reports that neither approach consistently outperforms the other and we want to be able to modify both approximate minimum degree (AMD) and nested dissection orderings computed using the graph partitioning package Metis [14, 15], in our experiments we use post processing

We remark that if, in the constrained ordering, an A-node is followed by one of its C-node neighbours, the two may be flagged as a 2×2 candidate pivot. In particular, if $C = 0$, the candidate pivot will be a tile pivot (see [4]). Note also that when an A-node is ordered there may be more than one C-node neighbour waiting to be ordered. We have chosen to include these C-nodes in the constrained ordering in the same relative order as originally. Our numerical results were not found to be very sensitive to this choice.

3 Numerical experiments

3.1 Test problems and test environment

We have recently designed and developed a new sparse direct solver for symmetric positive definite and indefinite problems. The code, which is called `HSL_MA77`, is available within the mathematical software library HSL [13]. `HSL_MA77` implements a multifrontal algorithm and offers the option of holding the matrix data, the computed factors, and some of the intermediate work arrays in direct-access files, thus allowing the solution of much larger problems than would otherwise be possible. The algorithms used by `HSL_MA77` together with details of the user interface and numerical experiments that illustrate its performance are presented in [16, 17]. `HSL_MA77` offers the user a number of options, some of which can be used to tune the performance on a particular platform or class of problems. Unless stated otherwise, in our tests we use `HSL_MA77` with its default settings. The pivot sequence passed to `HSL_MA77` is computed using either an AMD algorithm or Metis (which ordering is used for each problem is chosen on the basis of the order of the system and its sparsity as proposed by Duff and Scott [8]).

Our test problems are listed in Table 3.1 They are all available from the University of Florida Sparse Matrix Collection [2] and are a subset of those used in the study by Gould, Hu and Scott [10]. Those in the top part of the table are KKT systems while those in the lower part are saddle-point problems with C of the form $-\delta I$, where δ is a small positive constant.

All reported experiments were performed using double precision reals on a Dell Precision T5400 with two Intel E5420 quad core processors running at 2.5 GHz backed by 8 GB of RAM. We used the Goto BLAS [9] and gfortran-4.3 compiler with the -O3 option. All reported times are CPU timings in seconds. For each test on an individual problem we impose a CPU time limit of 300 seconds.

The right-hand side b is chosen so that the solution is $x_i = 1$ for all i. We measure accuracy of the

Table 3.1: Test problems. n denotes the order of K, m the order of C, nz the number of entries in K (in thousands).

	n	m	nz
bloweya	30.0	10.0	150.0
cont-201	80.6	40.2	438.8
cont-300	180.9	90.2	988.2
cvxqp3	17.5	7.5	115.0
darcy003	389.9	155.7	2097.6
k1_san	67.8	20.8	560.0
ncvxqp1	12.1	5.0	74.0
ncvxqp3	75.0	25.0	500.0
ncvxqp5	62.5	12.5	425.0
olesnik0	88.3	27.2	744.3
qpband	20.0	5.0	45.0
sit100	10.2	3.1	61.0
stokes128	50.0	16.4	558.6
turon_m	189.9	56.1	1690.9
c-59	41.2	13.6	480.5
c-62	41.7	17.5	559.3
c-68	64.8	28.3	566.0
c-70	68.9	29.6	659.0

computed solution using the scaled residual:

$$\frac{\|Kx - b\|}{\|K\|\|x\| + \|b\|}$$

with the infinity norm $\|x\|_\infty = \max_i |x_i|$ and its induced matrix norm $\|K\|_\infty = \max_i \sum_j |(K)_{ij}|$.

Unless stated otherwise, in all our experiments, the constrained ordering uses $u = 0.0$ and the unconstrained ordering uses $u = 0.01$ (the default setting for `HSL_MA77`).

3.2 Initial results

In Tables 3.2 and 3.3 we present results for the unconstrained and constrained orderings. For the unconstrained ordering, we report the predicted and actual numbers of entries in the factor (which we denote by $nz(Lp)$ and $nz(La)$, respectively) and the number of delayed eliminations Note that if a variable is delayed at more than one step of the computation, it will be counted more than once. For the constrained ordering, the predicted and actual statistics are the same. For both orderings we give the factorization time and, in Table 3.3, we report the scaled residuals before and after two steps of iterative refinement. Incomplete results are given for the unconstrained ordering for problems `bloweya`, `ncvxqp3`, `ncvxqp5`, and `ncvxqp7` since our CPU time limit of 5 minutes was exceeded. Where the unconstrained ordering did complete, we observe that for many of the problems (including `cont-300`, `cvxqp3`, and `c-62`), a large number of eliminations are delayed and this results in $nz(La)$ being significantly larger than $nz(Lp)$. For a number of test problems (particularly those for which the unconstrained ordering leads to a large number of delayed eliminations) the constrained ordering outperforms the unconstrained ordering in terms of the sparsity of L and factorization time but for others (such as `turon_m`, `c-68` and `c-70`) the converse is true.

In terms of accuracy, for several problems (including `cvxqp3` and `c-59`) using the constrained ordering (and hence no pivoting) leads to significantly larger scaled residuals than are obtained using the unconstrained ordering. However, with the exception of `bloweya`, after two steps of iterative refinement the scaled residual is order 10^{-15} or less.

Table 3.2: Comparison of the unconstrained and constrained orderings when used with the direct solver HSL_MA77. NS indicates our CPU time limit was exceeded.

	Unconstrained				Constrained	
	$nz(Lp)$ $*10^6$	$nz(La)$ $*10^6$	ndelay $*10^3$	factor time	$nz(L)$ $*10^6$	factor time
bloweya	0.21	NS	NS	NS	0.27	0.05
cont-201	4.64	11.05	88.3	1.45	9.89	1.57
cont-300	11.75	21.43	148.8	3.41	21.01	4.30
cvxqp3	3.13	35.72	59.7	62.6	11.03	15.0
darcy003	8.31	9.18	46.1	1.09	28.28	3.46
k1_san	3.27	3.46	7.0	0.35	11.02	1.26
ncvxqp1	1.68	11.80	312.1	20.2	5.54	4.56
ncvxqp3	18.99	NS	NS	NS	57.46	105
ncvxqp5	12.04	NS	NS	NS	23.82	19.9
olesnik0	4.58	4.86	9.2	0.50	15.64	1.96
qpband	0.05	0.05	0	0.01	0.05	0.01
sit100	0.45	0.47	0.8	0.04	1.27	0.13
stokes128	2.93	4.77	31.0	0.46	5.96	0.54
turon_m	13.71	14.35	19.1	1.72	53.35	10.7
c-59	5.11	5.19	0.6	2.08	36.23	57.7
c-62	8.21	34.23	108.9	80.6	35.80	50.1
c-68	8.88	9.52	6.3	6.00	102.8	271
c-70	4.89	5.00	1.2	1.30	56.99	81.2

Table 3.3: Comparison of the scaled residuals for unconstrained and constrained orderings when used with the direct solver HSL_MA77. Scaled residuals are reported before and after 2 steps of iterative refinement. NS indicates our CPU time limit was exceeded.

	Unconstrained		Constrained	
	Before	After	Before	After
bloweya	NS	NS	$5.6 * 10^{-12}$	$1.2 * 10^{-12}$
cont-201	$3.6 * 10^{-11}$	$1.2 * 10^{-16}$	$2.7 * 10^{-14}$	$1.2 * 10^{-16}$
cont-300	$4.7 * 10^{-11}$	$1.2 * 10^{-16}$	$3.6 * 10^{-14}$	$1.6 * 10^{-16}$
cvxqp3	$6.3 * 10^{-16}$	$2.0 * 10^{-16}$	$1.1 * 10^{-9}$	$1.9 * 10^{-16}$
darcy003	$2.1 * 10^{-14}$	$1.3 * 10^{-16}$	$1.8 * 10^{-15}$	$1.1 * 10^{-16}$
k1_san	$1.1 * 10^{-15}$	$4.0 * 10^{-17}$	$3.3 * 10^{-15}$	$5.8 * 10^{-17}$
ncvxqp1	$1.0 * 10^{-18}$	$2.5 * 10^{-17}$	$7.3 * 10^{-17}$	$2.8 * 10^{-17}$
ncvxqp3	NS	NS	$4.4 * 10^{-11}$	$2.3 * 10^{-16}$
ncvxqp5	NS	NS	$2.1 * 10^{-9}$	$2.3 * 10^{-16}$
olesnik0	$1.1 * 10^{-15}$	$2.3 * 10^{-17}$	$1.0 * 10^{-16}$	$8.9 * 10^{-17}$
qpband	$1.0 * 10^{-16}$	0.0	$3.4 * 10^{-17}$	$3.3 * 10^{-17}$
sit100	$6.0 * 10^{-15}$	$2.0 * 10^{-15}$	$7.0 * 10^{-16}$	$1.5 * 10^{-15}$
stokes128	$9.3 * 10^{-16}$	$2.3 * 10^{-16}$	$4.2 * 10^{-15}$	$6.5 * 10^{-15}$
turon_m	$9.7 * 10^{-16}$	$1.4 * 10^{-16}$	$7.2 * 10^{-15}$	$1.2 * 10^{-15}$
c-59	$1.9 * 10^{-16}$	$2.1 * 10^{-16}$	$1.5 * 10^{-11}$	$3.9 * 10^{-16}$
c-62	$6.2 * 10^{-16}$	$1.3 * 10^{-17}$	$4.8 * 10^{-11}$	$1.2 * 10^{-17}$
c-68	$3.9 * 10^{-17}$	$3.2 * 10^{-17}$	$4.5 * 10^{-14}$	$3.3 * 10^{-17}$
c-70	$3.1 * 10^{-15}$	$9.9 * 10^{-19}$	$3.1 * 10^{-13}$	$1.1 * 10^{-17}$

3.3 Effect of scaling

The experiments so far were performed using the matrix data as supplied. However, a number of recent studies (including those of Duff and Pralet [6] and Hogg and Scott [12]) have highlighted the potential benefits of scaling on the performance of direct solvers. In addition to sometimes reducing the scaled residual, prescaling the system matrix can help from a purely computational standpoint by reducing the number of delayed pivots and hence the memory required by the solver, the size of the computed factors, and total solution time. How to find a good scaling is still an open question, but a number of scalings have been proposed and are being successfully used. Here we use the symmetrized version of the HSL routine `MC64` [5]. `MC64` finds a maximum matching of an unsymmetric matrix such that the largest entries are moved on to the diagonal; this leads to an unsymmetric scaling such that the scaled matrix has all ones on the diagonal and the remaining entries are of modulus less than or equal to one. The approach can be symmetrized by the method of Duff and Pralet [6], which essentially amounts to initially ignoring the symmetry of the matrix and then averaging the relevant row and column scalings from the unsymmetric permutation.

In Tables 3.4 and 3.5 we repeat the results of the previous section but for prescaled K. We only include results for the problems on which scaling has a significant effect when the unconstrained ordering is used (the size of the factors and the times for the constrained ordering are unchanged since they are independent of the numerical values of the entries in K). We observe that, for this subset, prescaling substantially reduces both the number of delayed pivots and the difference between $nz(Lp)$ and $nz(La)$; this in turn leads to important reductions in the factorization times. Notable examples include the `ncvxqp` matrices that were previously not solved within the CPU time limit (although for these problems, the actual number of entries in L still exceeds the prediction by more than 10 per cent). If we compare the unconstrained and constrained orderings, we see that for well-scaled matrices, the unconstrained ordering is generally faster and produces much sparser factors than the constrained ordering. This suggests that the modifications made to the initial ordering to ensure a C-node is not ordered before its A-node neighbours leads to a substantially different ordering which can be of much poorer quality.

Table 3.4: Comparison of the unconstrained and constrained orderings when used with the direct solver `HSL_MA77` and K is prescaled.

	Unconstrained				Constrained	
	$nz(Lp)$ $*10^6$	$nz(La)$ $*10^6$	ndelay $*10^3$	factor time	$nz(L)$ $*10^6$	factor time
bloweya	0.21	0.24	3.3	0.18	0.27	0.05
cvxqp3	3.13	4.88	26.0	2.28	11.03	15.0
ncvxqp1	1.68	2.26	10.3	0.77	5.53	4.57
ncvxqp3	18.99	25.16	65.3	16.2	57.46	105
ncvxqp5	12.04	13.41	11.8	5.29	23.82	19.9
c-62	8.21	8.36	0.6	3.13	35.80	50.3
c-68	8.88	9.26	3.3	5.80	102.8	274

If we look at the scaled residuals in Table 3.5, there appears to be little difference in the quality of the solution produced using the different orderings. For the constrained ordering, for our test examples scaling does not consistently lead to smaller initial residuals. However, it is very straightforward to construct examples for which the constrained ordering fails to give any accuracy in the computed solution without suitable prescaling. For example, large entries in the matrix B can lead to instability.

Table 3.5: Comparison of the scaled residuals for unconstrained and constrained orderings when used with the direct solver HSL_MA77 with K prescaled. Scaled residuals are reported before and after 2 steps of iterative refinement.

	Unconstrained		Constrained	
	Before	After	Before	After
bloweya	$3.9 * 10^{-14}$	$3.0 * 10^{-14}$	$1.1 * 10^{-12}$	$2.9 * 10^{-17}$
cvxqp3	$5.7 * 10^{-11}$	$2.2 * 10^{-16}$	$9.4 * 10^{-10}$	$1.9 * 10^{-16}$
ncvxqp1	$6.3 * 10^{-14}$	$1.5 * 10^{-17}$	$1.3 * 10^{-16}$	$1.9 * 10^{-17}$
ncvxqp3	$5.3 * 10^{-9}$	$2.5 * 10^{-16}$	$7.9 * 10^{-11}$	$2.4 * 10^{-16}$
ncvxqp5	$2.7 * 10^{-11}$	$2.4 * 10^{-16}$	$1.5 * 10^{-9}$	$2.3 * 10^{-16}$
c-62	$3.0 * 10^{-14}$	$2.4 * 10^{-18}$	$9.2 * 10^{-12}$	$4.4 * 10^{-18}$
c-68	$5.0 * 10^{-17}$	$4.0 * 10^{-17}$	$1.5 * 10^{-14}$	$3.9 * 10^{-17}$

3.4 Relaxing the constraint

Since waiting to order a C-node until all its A-node neighbours have been ordered can lead to significantly denser factors, an obvious approach is to try relaxing the constraint. In particular, we could require that at least one of the A-node neighbours of the C-node has already been ordered. During the modification of the initial ordering, this will clearly have the effect of postponing fewer C-nodes and those that are postponed will be brought back into the ordering sooner. However, there is now no guarantee that the LDL^T factorization will exist without pivoting (numerical tests quickly confirm the factorization will generally breakdown) and thus the main advantage of the constrained ordering has been lost. We have rerun our experiments using the relaxed constrained ordering with partial pivoting (threshold $u = 0.01$). We found that $nz(Lp)$ may increase beyond that of the unconstrained ordering but the changes we observed were small and led to only small changes in $nz(La)$. Thus for some examples the difference between the predicted and actual number of entries in L is reduced but the reduction is modest. Furthermore, the total number of delayed eliminations is not necessarily reduced. Indeed, it can increase significantly, suggesting some pivots are delayed for a large number of steps of the factorization. Our findings are illustrated by the results reported in Table 3.6 (here the matrix K is prescaled using the symmetrized MC64).

Table 3.6: Comparison of the unconstrained and relaxed constrained orderings when used with the direct solver HSL_MA77 with partial pivoting ($u = 0.01$).

	Unconstrained			Relaxed Constrained		
	$nz(Lp)$ $*10^6$	$nz(La)$ $*10^6$	ndelay $*10^3$	$nz(Lp)$ $*10^6$	$nz(La)$ $*10^6$	ndelay $*10^3$
cvxqp3	3.13	4.88	26.0	3.75	4.78	62.4
ncvxqp1	1.68	2.26	10.3	1.86	2.24	18.6
ncvxqp3	18.99	25.16	65.3	21.00	25.02	151.0
ncvxqp5	12.04	13.41	11.8	12.79	13.40	16.1
c-62	8.21	8.36	0.6	8.40	8.43	0.4
c-68	8.88	9.26	3.3	9.78	9.90	3.5

4 Concluding remarks

We have examined the performance of the constrained ordering proposed by Bridson. Although we were able to compute the factorization of the saddle point systems without numerical pivoting, the computed

factors were almost always denser than those obtained by prescaling the matrix with MC64 and then using an unconstrained ordering combined with partial pivoting.

One of the motivations for the work of Bridson was that it was simpler to modify an existing Cholesky factorization to perform a signed Cholesky factorization than to develop a general purpose indefinite solver that employs partial pivoting. Our situation is different since our solver HSL_MA77 is already designed to efficiently solve both positive definite and indefinite problems. Moreover, Reid and Scott [16] report that, in general, the code loses little performance in treating a positive definite problem as indefinite. Nevertheless, strategies that do not use pivoting are of interest for parallel factorizations on multicore architectures (see, for example, [11] and the references therein). On a multicore machine it may be faster as well as more convenient to use an ordering that does not require pivoting even if it leads to denser factors and so it may be advantageous to exploit the approach of Bridson (and to develop other constraint ordering strategies) in a parallel sparse code.

All the HSL codes referred to in this report are part of HSL 2007. Use of HSL requires a licence. Licences are available without charge to individual academic users for their personal (non-commercial) research and for teaching; for other users, a fee is normally charged. Details of how to obtain a licence together with information on all HSL packages are available at www.cse.clrc.ac.uk/nag/hsl/.

References

[1] R. Bridson. An ordering method for the direct solution of saddle-pont matrices. Preprint available from http://www.cs.ubc.ca/∼rbridson/kktdirect/.

[2] Tim Davis. The University of Florida Sparse Matrix Collection. Technical Report, University of Florida, 2007. http://www.cise.ufl.edu/ davis/techreports/matrices.pdf.

[3] A.C. de Niet and F.W. Wubs. Numerically stable LDL^T-factorization of F-type saddle point matrices. *IMA Journal of Numerical Analysis*, 29:208–234, 2009.

[4] I.S. Duff, N.I.M. Gould, J.R. Reid, J.A. Scott, and K. Turner. Factorization of sparse symmetric indefinite matrices. *IMA Journal of Numerical Analysis*, 11:181–2044, 1991.

[5] I.S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. Matrix Analysis and Applications*, 22(4):973–996, 2001.

[6] I.S. Duff and S. Pralet. Strategies for scaling and pivoting for sparse symmetric indefinite problems. *SIAM Journal on Matrix Analysis and Applications*, 27:313 – 340, 2005.

[7] I.S. Duff and S. Pralet. Towards a stable static pivoting strategy for the sequential and parallel solution of sparse symmetric indefinite systems. *SIAM Journal on Matrix Analysis and Applications*, 29:1007–1024, 2007.

[8] I.S. Duff and J.A. Scott. Towards an automatic ordering for a symmetric sparse direct solver. Technical Report RAL-TR-2006-001, Rutherford Appleton Laboratory, 2006.

[9] Kazushige Goto and Robert van de Geijn. High performance implementation of the level-3 BLAS. *ACM Trans. Mathematical Software*, 35:4:1–4:14, 2008.

[10] N.I.M. Gould, J.A. Scott, and Y. Hu. A numerical evaluation of sparse direct solvers for the solution of large, sparse, symmetric linear systems of equations. *ACM Trans. Mathematical Software*, 33, 2007. Article 10, 32 pages.

[11] J.D. Hogg. A DAG-based parallel Colesky factorization for multicore systems. Technical Report RAL-TR-2008-029, Rutherford Appleton Laboratory, 2008.

[12] J.D. Hogg and J.A. Scott. The effects of scalings on the performance of a sparse indefinite solver. Technical Report RAL-TR-2008-007, Rutherford Appleton Laboratory, 2008.

[13] HSL. A collection of Fortran codes for large-scale scientific computation, 2007. See http://www.cse.scitech.ac.uk/nag/hsl/.

[14] G. Karypis and V. Kumar. METIS: Unstructured graph partitioning and sparse matrix ordering system. Technical Report TR 95-035, University of Minnesota, 1995.

[15] G. Karypis and V. Kumar. METIS - family of multilevel partitioning algorithms, 1998. See http://glaros.dtc.umn.edu/gkhome/views/metis.

[16] J.K. Reid and J.A. Scott. An efficient out-of-core sparse symmetric indefinite direct solver. Technical Report RAL-TR-2008-024, Rutherford Appleton Laboratory, 2008.

[17] J.K. Reid and J.A. Scott. An out-of-core sparse Cholesky solver. *ACM Trans. Mathematical Software*, 36(2), 2009. To appear.

[18] M. Tuma. A note on the LDL^T decomposition of matrices from saddle-point problems. *SIAM J. Matrix Analysis and Applications*, 23:903–925, 2002.