Science & Technology Facilities Council

Technical Report

RAL-TR-2012-009

A study of pivoting strategies for tough sparse indefinite systems

JD Hogg, JA Scott

July 2012

ISSN 1358- 6254

A study of pivoting strategies for tough sparse indefinite systems

Jonathan Hogg and Jennifer Scott[1]

ABSTRACT

The performance of a sparse direct solver is dependent upon the pivot sequence that is chosen during the analyse phase. In the case of symmetric indefinite systems, it may be necessary to modify this sequence during the factorization to ensure numerical stability. Delaying pivots can have serious consequences in terms of time as well as the memory and flops required for the factorization and subsequent solves. This study focuses on hard-to-solve sparse symmetric indefinite problems for which standard threshold partial pivoting leads to a large number delayed pivots. We perform a detailed review of pivoting strategies that are aimed at reducing delayed pivots without compromising numerical stability. Extensive numerical experiments are performed on a set of tough problems arising from practical applications.

Keywords: sparse matrices, sparse linear systems, indefinite symmetric systems, direct solvers, threshold partial pivoting, static pivoting, matching, scaling.

AMS(MOS) subject classifications: 65F05, 65F50

---

[1] Computational Science and Engineering Department, Rutherford Appleton Laboratory, Harwell Oxford, Oxfordshire, OX11 0QX, UK.
Correspondence to: jennifer.scott@stfc.ac.uk
This work was supported by EPSRC grant EP/I013067/1.

July 20, 2012

1 Introduction

The accurate and efficient solution of sparse symmetric indefinite linear systems has long been an important area of interest since such systems arise in a wide range of practical applications, including incompressible flow problems, electromagnetic scattering, eigenvalue problems and augmented systems within linear and nonlinear optimization problems. A key difference between a sparse direct solver for the solution of symmetric positive-definite systems and one for symmetric indefinite systems is that the latter needs to incorporate pivoting to maintain numerical stability. Not only does this contribute significantly to the complexity of the development of the solver, pivoting adds overheads when the solver is run. Firstly, in the search for a suitable pivot at each stage of the factorization. Secondly, in the handling of pivot candidates that are found to be unsuitable, leading to additional flops (in both the factorize and solve phases) and fill-in of the factors beyond that predicted by the analyse phase. We have recently developed new task-based sparse direct solvers for the efficient solution of positive-definite and indefinite linear systems on multicore machines [21, 24, 28]. In the indefinite case, the need for pivoting means there is less scope for achieving parallelism. In particular, the tasks are based on block columns whereas in the positive-definite case, the tasks are block-based and thus finer grained. Our goal is to develop an efficient communication-avoiding pivoting strategy that, while fast, is as stable as standard threshold partial pivoting. The intention is that this would allow us to work with block tasks, switching to block column tasks only when careful monitoring of growth suggests stability is a concern. The success of this approach will be dependent on being able to precompute pivot sequences that can be used with little or no modification during the factorization, without compromising numerical stability and the accuracy of the computed solution, and retaining sparsity in the factors. Thus our interest lies in techniques that ensure stability but lead to few, if any, delayed pivots (that is, pivots that during the factorization can only be used stably later than their position in the supplied pivot sequence), possibly at the cost of a (modest) increase in the factor size and flop count.

A direct solution of the sparse symmetric indefinite linear system

$$Ax = b$$

involves factorizing A into the form

$$A = LDL^T,$$

where D is a diagonal matrix with 1×1 and 2×2 pivot blocks and L is a sparse unit lower triangular matrix. In practice, a more general factorization of the form

$$SAS^{-1} = PLD(PL)^T \tag{1.1}$$

is computed, where S is a diagonal scaling matrix and P is a permutation matrix (or, more generally, a product of permutation matrices) that holds the pivot order. In some implementations, (1.1) is further generalised to

$$SAS = P(LDL^T + E)P^T, \tag{1.2}$$

where E is a diagonal matrix with small entries. It is the choice of S, P and E that determines the sparsity of L as well as the accuracy and stability of the numerical factorization. The aim of this study is to examine different choices for S, P and E and, using a range of hard-to-solve symmetric indefinite linear systems, illustrate how well they work in practice.

Before the factorization commences, a pivot sequence (elimination order) is computed using one of the many algorithms available for this (for example, a variant of nested dissection [18] or minimum degree [1, 33, 46]). This sequence is used by the analyse phase to set up the (provisional) data structures for the factorization. If at a given stage of the factorization there are p pivot candidates but only $p_1 < p$ pivots are selected as being suitable, the remaining $p - p_1$ candidates are called *delayed*. The effect of this is that at a later stage in the factorization, the number of candidate pivots will be greater than predicted for the supplied pivot sequence; the data structures set up during the analyse phase will have to be modified to accommodate this, more operations will be performed in the computation of the factors and L will be less sparse than if the pivot sequence was used without modification. These problems that result from delayed pivots are well-known and over the last two decades, a number of techniques have been proposed to limit delays and papers exploring this issue written. We review and summarise these approaches and, using a collection of hard-to-solve symmetric indefinite problems arising from practical applications, we report on how well they work in practice. The paper is organised as follows. In Section 2, we briefly look at the importance of prescaling the matrix A before the factorization begins. In Section 3, we consider threshold partial pivoting and discuss variants that are designed to weaken the standard test criteria to reduce delayed pivots without, it is hoped, substantially effecting stability. We then consider, in Section 4, strategies to choose 2×2 pivot blocks during the analyse phase of the solver. Extensive numerical experiments are reported on in Section 5. Finally, we summarise our findings and recommendations in Section 6.

We observe that it is not our intention to attempt to exhaustively describe and review the pivoting strategies that are used in each of the modern state-of-the-art sparse indefinite solvers, although where appropriate, we indicate in which solver(s) a strategy is used. Our key contribution is a systematic review of the techniques that have been proposed to improve the performance of indefinite solvers and, using a set of tough problems that these methods are aimed at, we examine and compare their effectiveness. Experts in the development of sparse direct solvers undoubtedly understand the issues well but we have been unable to find any comprehensive results that bring together and compare different strategies. Furthermore, our contact with users has led us to believe that they are unaware of the potential consequences of selecting a particular strategy and do not appreciate that some approaches may sacrifice robustness for speed.

2 Scaling the linear system

The aim of scaling is to choose a diagonal matrix S such that the matrix $\hat{A} = SAS^{-1}$ is numerically easier to factorize than A. This will often mean more accurate results obtained faster than if no scaling is used and, in extreme cases, by reducing the number of delayed pivots, enables a factorization to be computed where previously memory (or time) limitations made it impossible. How to find a good scaling S is still an open question, but a number of scalings have been proposed and are widely used. The use of scalings (and combinations of scalings) with the well-known symmetric indefinite solver `MA57` [16] have been reported on by Hogg and Scott [22]. Based on that work, we restrict our attention here to the maximum matching algorithm of `MC64` [10] and the equilibration algorithm offered by `MC77` [38].

2.1 `MC64`

`MC64` finds a maximum matching of an unsymmetric matrix such that the largest entries are moved on to the diagonal [10]; 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 [11], 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 recent years, `MC64` has been widely used in conjunction with both direct and iterative methods. For example, it is used in the sparse LU factorization solver SuperLU [31, 32], where it is particularly advantageous to put large entries on the diagonal because the pivoting strategy implemented in SuperLU does not allow delayed pivots. The symmetrized version was developed following the success of `MC64` on unsymmetric systems; it is used by default within the factorize phase of `MA57` and is available within the factorize phases of `HSL_MA97` [26] and MUMPS [34].

2.2 MC77

Equilibration is a particular form of scaling in which the rows and columns of the matrix are modified so that they have approximately the same norm. The package MC77 uses an iterative procedure [38] to attempt to make all row and column norms of the matrix unity for a user-specified geometric norm $\|\cdot\|_p$. The infinity norm is the default within MC77. It produces a matrix whose rows and columns have maximum entry of exactly one and has good convergence properties. The one norm produces a matrix whose row and column sums are exactly one (a doubly stochastic matrix) and is, in some sense, optimal. Recently, Ruiz and Uçar [39] considered combining the use of the infinity and one-norms. Specifically, they propose performing one step of $\infty-$norm scaling followed by a few steps of $1-$norm scaling.

Equilibration scaling is available within factorize phases of HSL_MA77 [35, 36], HSL_MA97 and MUMPS. Within HSL_MA77, it is implemented out-of-core, avoiding the need to hold the system matrix in main memory.

3 Pivoting strategies

3.1 Threshold partial pivoting

In the case of symmetric linear systems, 1×1 and 2×2 pivoting must be performed if symmetry is to be kept while stability is retained. Stability of the factorization of symmetric indefinite systems was considered in detail in the paper by Ashcraft, Grimes and Lewis [4]. They showed that bounding the size of the entries of L, together with a backward stable scheme for solving 2×2 linear systems, suffices to show backward stability for the entire solution process. They found that the widely used strategy of Bunch and Kaufmann [6] does not have this property whereas the threshold pivoting technique first used by Duff and Reid [13] in their original multifrontal solver does.

Duff and Reid choose the pivots one-by-one, with the aim of limiting the size of the entries $L_{i,j}$in L:

$$|l_{i,j}| < u^{-1}, \tag{3.1}$$

where the threshold u is a user-set value in the range $0 \leq$u≤ 1.0. In the case where u is zero, this is interpreted as requiring that the entries be finite. Suppose q is the number of rows and columns of D found so far (that is, the number of 1×1 pivots plus twice the number of 2×2 pivots). Let $a_{i,j}$, with $i > q$ and $j > q$, denote an entry of the matrix after it has been updated by all the permutations and pivot operations so far. For a 1×1 pivot in column $j = q + 1$, the requirement for inequality (3.1) corresponds to the threshold test

$$|a_{q+1,q+1}| > u \max_{q+1<i\leq n} |a_{i,q+1}|. \tag{3.2}$$

The original test used by Duff and Reid [13] for 2×2 pivots proved unnecessarily severe. Instead, following the work of Duff et al. [17], an appropriate test for a 2×2 pivot is

$$\left| \begin{pmatrix} a_{q+1,q+1} & a_{q+1,q+2} \\ a_{q+1,q+2} & a_{q+2,q+2} \end{pmatrix}^{-1} \right| \begin{pmatrix} \max_{q+2<i\leq n} |a_{i,q+1}| \\ \max_{q+2<i\leq n} |a_{i,q+2}| \end{pmatrix} < \begin{pmatrix} u^{-1} \\ u^{-1} \end{pmatrix}, \tag{3.3}$$

where the absolute value notation for a matrix refers to the matrix of corresponding absolute values. In the case where u is zero, this is interpreted as requiring that the pivot be nonsingular. While this test is not used universally by modern sparse direct solvers, it has been incorporated into all recent sparse indefinite solvers within the HSL mathematical software library [29] (including MA57, HSL_MA77, HSL_MA86 [24] and HSL_MA97) and is thus the one that is used in this study.

3.2 Choice of pivot threshold parameter

The choice of the pivot threshold parameter u not only influences the number of candidate pivots that are rejected and hence delayed, but also the stability of the factorization. A large u means a tight bound on the size of the entries of L at the possible cost of a large number of delays whereas a smaller value potentially reduces the number of delays but, because the pivot test is less strict, the factorization may be less accurate and it may be necessary to perform refinement during the solve phase to recovery the required accuracy. Frequently-used choices are $u = 0.1$ or 0.01. These are held, on the basis of extensive numerical experience, to generally provide a good compromise between stability and sparsity. However, in some application areas it is common to use a much smaller threshold, despite the attendant risk of numerical instability. For example, the well-known optimization package Ipopt [47] optionally uses one of the HSL indefinite solvers `MA27` [14] and `MA57`. For both solvers, the default setting for u within Ipopt is $u = 10^{-8}$. If, at some stage of the computation, this is found to give an unstable factorization (detected via an unexpected inertia or large backward error following solution), the factorization is recomputed with a larger of value of u, and this process repeated as necessary until either a stable factorization is achieved or the maximum allowable value for u is reached (which, by default, in Ipopt is 0.0001).

3.3 Relaxed threshold pivoting

The idea behind relaxed threshold pivoting is to relax the threshold parameter during the factorization when no pivot satisfying the threshold tests for the input u is available. In this way, the pivot threshold used during the computation may be smaller than at the start of the factorization, reducing the number of delays but hopefully without seriously compromising stability. Relaxed pivoting strategies have been explored by Duff and Pralet [12] (see also [37]). The factorization commences with the user-supplied threshold u. If at some stage no 1×1 or 2×2 candidate pivot satisfies the test (3.2) or (3.3), pivots are accepted using a weaker threshold, provided this is at least a user-defined minimum u_{min}. The hope is that this will lead to a more stable factorization that requires less refinement than that which results from using a smaller u throughout the computation.

3.4 Restricted pivoting

Threshold partial pivoting requires all the entries below the diagonal in the candidate pivot column(s) to be searched when checking the threshold tests (3.2) and (3.3). This is expensive, particularly in a parallel implementation. One way of avoiding this is to limit the search to the p candidate rows, so that (3.2) is replaced by

$$|a_{q+1,q+1}| > u \max_{q+1<i\leq p} |a_{i,q+1}|,$$

and (3.3) is modified in the same way. The hope is that, for a well-scaled matrix, restricting the search will save time without leading to the acceptance of pivots that would otherwise have been rejected. Because of the greater potential for numerical instability, refinement is more likely to be needed to recover accuracy. Note that time is saved not only in the search for pivots but also because entries in the candidate pivot columns that lie in rows $p+1$ to n do not need to be updated after each pivot has been chosen. This allows block update operations (using high level BLAS kernels) to be employed, thereby improving efficiency.

3.5 Static pivoting

A *static pivoting* strategy refers to one that allows the factorization to respect the elimination ordering passed to it from the analyse phase. The factorization does not necessarily follow the analysis exactly and some slight variations are allowed. For example, in a multifrontal algorithm it is sufficient that the factorization decisions are compatible with the assembly tree (numerical pivoting can be performed within a front). The key point is that pivots are not delayed so that the analysis predicts exactly the memory and operations needed to performed the factorization and thus the data structures can remain static. A static approach for LU factorization was proposed by Li and Demmel [31] for the SuperLU solver. During Gaussian elimination small perturbations are added to the diagonal to prevent pivots from becoming too small and failing the threshold test. The computed factorization is thus not of A but of a perturbed matrix $A + E$, where E is a diagonal matrix.

For symmetric indefinite systems, Schenk and Gärtner [42] combine static pivoting with Bunch-Kaufman and restricted pivoting strategies. Their strategy is implemented within the solver PARDISO [41]. A similar approach is available as an option within WSMP [19]. In this case, the coefficient matrix is perturbed whenever numerically acceptable 1×1 and 2×2 pivots cannot be found within a diagonal supernode block (checks on potential pivots are only made within the supernode block). This strategy has been shown to perform well in many applications. An important downside is that the solve phase can require an increased number of refinement steps to achieve the requested accuracy. Furthermore, in some tough cases, it is not possible to recovery accuracy using iterative refinement (see, for example, the results reported in [26]) and it may be necessary to try a more expensive procedure, such as Flexible GMRES (FGMRES) [2, 40] (but, again, convergence is not guaranteed). A further disadvantage is that, since a perturbed system is solved, the computed inertia of the original matrix A may not be reliable and in some applications accurate knowledge of the inertia is required (for example, to ensure local convexity in a non-linear interior point method).

Duff and Pralet [12] propose combining threshold partial pivoting with static to try and minimise both the number of perturbations that are added and the amount of refinement required. Their strategy (or variants of it) is available as an option within a number of solvers, including `MA57`, `HSL_MA77`, `HSL_MA86`, and `MUMPS`. The static pivoting strategy used in this study follows that of Duff and Pralet. If no 1×1 or 2×2 candidate pivot satisfies the test (3.2) or (3.3) even after relaxing the value of u, the 1×1 pivot that is nearest to satisfying the test is accepted. If its absolute value is less than another user-set threshold *static*, it is given the value that has the same sign but absolute value *static* (see [37]).

4 Block pivot strategies

Standard algorithms for computing a pivot sequence, including minimum degree and nested dissection, compute a sequence of 1×1 pivots. For many indefinite problems, it is necessary to use 2×2 pivots during the factorization and so the supplied pivot sequence will be modified. To try and minimise modifications, it is therefore natural to try and construct a tentative pivot sequence that contains 2×2 pivots. In this section, we discuss a number of approaches that have been proposed for this.

4.1 Reusing a pivot sequence

If a sequence of linear systems

$$A_k x_k = b_k$$

is to be solved in which the matrix A_k varies from the previous matrix A_{k-1} by a relatively small amount (in terms of both the sparsity pattern and the values of the entries), an obvious possibility is to pass the pivot sequence actually used by the factorization phase for A_{k-1} to the analyse phase for A_k. This is a strategy that could be used, for example, within a non-linear optimization package, such as Ipopt.

We have performed experiments in which we set $A_k = A_{k-1} = A$. The results are reported on in Section 5.9.

4.2 Constraint pivot orderings

Many tough indefinite problems are of the form

$$A = \begin{pmatrix} H & B^T \\ B & C \end{pmatrix}, \tag{4.4}$$

with H symmetric positive definite, B rectangular, and C symmetric positive semi-definite. Matrices of the form (4.4) 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. We refer to a matrix of the form (4.4) with $H = C = 0$ as an OXO matrix.

The problem of find a permutation P such that PAP^T can be factorized stably without the need for numerical pivoting and without modifying the entries in A, while still limiting the number of entries in L, has been examined for special classes of KKT 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. Tůma [45] and De Niet and Wubs [8] have presented methods for these problems and report positive results. For more general saddle-point problems, Bridson [5] has proposed a constrained ordering as follows. The nodes of the adjacency graph $\mathcal{G}$ of the matrix A are split into two disjoint sets: those that correspond to the diagonal entries of H are known as H-nodes and the remaining nodes as C-nodes. The ordering constraint proposed by Bridson is extremely simple: a C-node can only be ordered after all its H-node neighbours in $\mathcal{G}$ have been ordered. Bridson shows that, provided H is definite and B is of full row rank, with this ordering the LDL^T factorization exists. Moreover, the pivots associated with the H-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 A. It allows him to modify a Cholesky factorization code to perform the factorization of the indefinite matrix A with **no** threshold pivoting. A stability analysis is lacking but Bridson reports that numerical experiments indicate the constrained ordering is generally sufficient to avoid numerical pivoting; this was supported by additional experiments given by Scott [44]. 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.

We note that WSMP offers a limited form of constrained ordering. This allows the user to specify that the last $n1 < n$ columns are to be pivoted on last. This is recommended in the WSMP documentation for problems that have a few zero (or near-zero) entries on the diagonal. For such matrices, the factorization is performed without pivoting. By ordering the $n - n1$ rows and columns with zero diagonal entries to the end, the user ensures (unless there is numerical cancellation) that these diagonal entries are nonzero by the time they are pivoted on.

4.3 `MA47` orderings

`MA47` [15] is a sparse symmetric solver that is specifically designed for solving indefinite systems and, in particular, for matrices that have some zeros on the diagonal. A key feature is that the analyse phase may choose tentative block pivots. A variant of the Markowitz criterion recommended in [17] is used to extend the strategy of minimum degree to block pivots. As with other minimum degree algorithms, the pivots are chosen during the analyse using the sparsity structure of A alone (without the assumption that the diagonal is implicitly present that is made by standard fill-reducing ordering algorithms).

`MA47` distinguishes between block pivots with different sparsity patterns. Specifically, pivots may be

(i) of the form

$$\begin{pmatrix} 0 & A_1 \\ A_1^T & 0 \end{pmatrix},$$

with A_1 square, which is called an *oxo* pivot;

(ii) of the form

$$\begin{pmatrix} A_2 & A_1 \\ A_1^T & 0 \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} 0 & A_1 \\ A_1^T & A_2 \end{pmatrix},$$

with A_1 square, which is called an *tile* pivot; or

(iii) of any other form, which is called *full*.

Tile and oxo pivots are termed *structured* pivots and their structure is taken into account within the analyse and factorize phases. As far as we are aware, MA47 is the only sparse symmetric solver that exploits the structure of block pivots. However, the analyse phase of MA47 can be used to compute a tentative pivot sequence containing 2×2 structured pivots that may be passed to other solvers; we report on this in Section 5.5.

4.4 Matching elimination orderings

In the unsymmetric case, the matching algorithm moves large entries on to the diagonal of the matrix. In the symmetric case, we need to preserve symmetry but a symmetric permutation leaves the diagonal unchanged. Thus the aim is to permute a large off-diagonal entry $a_{i,j}$ close to the diagonal so that the 2×2 block

$$\begin{pmatrix} a_{i,i} & a_{i,j} \\ a_{i,j} & a_{j,j} \end{pmatrix}$$

is potentially a good 2×2 candidate pivot. Duff and Gilbert [9] noticed that the cycle structure of the permutation $P_{\mathcal{M}}$ associated with the unsymmetric matching $\mathcal{M}$ can be exploited to obtain such a permutation P_s. This has been explored further by Duff and Pralet [11] and, amongst others, Schenk et al. [20, 42, 43], and symmetric maximum matchings are (optionally) used within the sparse solvers HSL_MA97, MUMPS, PARDISO and WSMP.

A maximum weighted matching $\mathcal{M}$ is first computed. Assume initially that A is not found to be structurally singular. Any diagonal entries that are in the matching are immediately considered as potential 1×1 pivots and are held in a set $\mathcal{M}_1$. A set $\mathcal{M}_2$ of potential 2×2 pivots is then built by expressing the computed permutation $P_{\mathcal{M}}$ in terms of its component cycles. Because of the scaling, all the entries in the cycles of $P_{\mathcal{M}}$ are 1 in absolute value so a structural criterion is used to select the potential 2×2 pivots. A cycle of length 1 corresponds to an entry a_{ii} in the matching. A cycle of length 2 corresponds to two nodes i and j, where a_{ij} and a_{ji} are both in the matching. k potential 2×2 pivots can be extracted from even cycles of length $2k$ or from odd cycles of length $2k + 1$. The idea is to select k entries a_{ij} and their symmetric counterpart and to discard the other matched entries. In practice, most of the cycles in the matching permutation are of length 1 or 2 [11]. Where there are longer cycles, Duff and Pralet discuss possible ways of extracting 2×2 pivots, based on the sparsity patterns of the rows of A (in particular, they seek to pair up rows that have as similar a structure as possible). Since long cycles do not occur often, we adopt the more straightforward approach of taking the first two entries as the first 2×2 pivot, the next two as the next 2×2 pivot, and so on, until if the cycle is a cycle of odd length, a single entry remains, which is added to the set $\mathcal{M}_1$.

To combine the resulting permutation with a fill-reducing ordering (such as nested dissection or minimum degree), the graph of $P_s A P_s^T$ is compressed and the ordering applied to the compressed graph. In the compression step, the union of the sparsity structure of the two rows and columns corresponding to a potential 2×2 pivot is built and used as the structure of a single row and column in the compressed matrix. Having applied a fill-reducing ordering to the compressed graph, the resulting permutation is expanded to a permutation P_f for the original matrix. The final permutation is the product $P = P_f P_s$. The rows/columns corresponding to a potential 2×2 pivot are reordered consecutively and, in our implementation, a negative flag is used to indicate a 2×2 pivot. This is because some solvers and, in particular, HSL_MA77, are able to take advantage of 2×2 pivots during the analyse phase and our experience is that this can offer a (usually small) time advantage over omitting the flags (see also [17]).

In the case where the maximum matching algorithm detects that A is structurally singular, the unmatched rows and corresponding columns are removed and the matching algorithm reapplied to the nonsingular part. The unmatched rows/columns are held as potential 1×1 pivots; their unsuitability as pivots will be picked up during the factorization phase of the solver. Further details of the matching algorithm in the rank deficient case are given in [27].

A by-product of computing a matching-based ordering is a scaling for A and when employing a matching-based ordering we always combine it with using this scaling. To employ a matching algorithm within the analyse phase it is necessary for the analyse phase to have available the numerical values of the entries of A (and thus the analyse phase no longer depends solely on the structure of A). This means that, if the user wants to factorize more than one matrix with the same sparsity pattern but different numerical values, it may be necessary to recompute the ordering and the scaling. This can add a significant overhead when compared with an analyse phase that uses the sparsity structure only. However, if scaling factors need to be computed prior to the factorization of each matrix, then the additional cost associated with the matching-based ordering will generally be modest compared with the total solution time.

5 Numerical experiments

5.1 Test environment

As already indicated, the HSL mathematical software library contains a number of sparse solvers that are designed for symmetric indefinite systems. In this study, we have chosen to use the code `HSL_MA77` [35, 36] (Version 5.8.0 is used). `HSL_MA77` implements a multifrontal algorithm and includes the possibility of holding the matrix data, the computed factors, and some of the intermediate work arrays in files held on disk, thus allowing the solution of much larger problems than would otherwise be possible. `HSL_MA77` offers the user a number of options. Importantly for this study, these include the use of relaxed pivoting and threshold pivoting. Furthermore, the user is required to supply the elimination ordering, allowing us to experiment with constraint and matching-based elimination orderings. An option also exists to supply a scaling.

All our experiments are performed using double precision reals on a Dell Precision T5400 with two Intel E5420 quad core processors running at 2.5 GHz. The ifort compiler (Version 12.0) with option -O3 and MKL BLAS (Version 10.2) are used. The right-hand side b is chosen so that the solution is $x_i = 1$ for all i. We measure accuracy of the computed solution using the scaled residual:

$$\frac{\|Ax - b\|}{\|A\|\|x\| + \|b\|}$$

with the infinity norm $\|x\|_\infty = \max_i |x_i|$ and its induced matrix norm $\|A\|_\infty = \max_i \sum_j |a_{i,j}|$. The computed solution is only accepted if the scaled residual is less than 10^{-14} and the forward error is less than 10^{10}; where necessary, iterative refinement or refinement using FGMRES is performed.

The test problems used in this study are taken from in the University of Florida Sparse Matrix Collection [7] and are listed in Tables 5.1 and 5.2. For each problem, we give its order n and number of entries $nz(A)$ and, for the singular problems, the structural rank *srank*. In addition, we report the number of entries nzL in L (in each table, the problems are listed in increasing order of nzL) and the number of floating-point operations (flops) required to compute L that are returned by the analyse phase of `HSL_MA77`. These are computed using the default settings for `HSL_MA77`; the elimination ordering is computed using the nested dissection ordering package METIS [30]. The statistics from the analyse phase are the predicted nzL and predicted flop count and the statistics that would be returned by the factorization phase if the supplied ordering could be used without modification.

The test problems were chosen from amongst the many available within the Collection because when the factorize phase is run (again, with default settings and, in particular, the default threshold parameter $u = 0.01$) there are a large number of delayed pivots. The statistics returned by the factorize phase of `HSL_MA77` are reported in Table 5.3. Here *ndelay* is the number of eliminations that were delayed. If a candidate pivot is delayed more than once, it will be counted the number of times it is delayed. We see that, for many of our chosen examples, the actual number of entries in L is more than an order of magnitude greater than was predicted by the analyse phase and the difference between the predicted and actual number of flops can be significantly greater. A notable example is `GHS_indef/bloweybl`. For this problem, the actual flop count is five orders of magnitude greater than predicted. However, in some cases, although the number of delays is relatively high, they do not lead to a large increase in nzL or the flop

Table 5.1: Test problems. n denotes the order of A, $nz(A)$ is the number of entries in A, nzL is the predicted number of entries in L and $nflop$ is the predicted number of flops required to compute L.

Identifier	n	$nz(A)$	nzL	$nflop$
TSOPF/TSOPF_FS_b162_c1	10798	608540	1.3930×10^6	2.0509×10^8
TSOPF/TSOPF_FS_b39_c7	28216	730080	1.5755×10^6	1.0796×10^8
Schenk_IBMNA/c-64	51035	717841	1.6525×10^6	1.2588×10^8
GHS_indef/ncvxqp1	12111	73963	1.6839×10^6	7.2793×10^8
QY/case39	40216	1042160	2.2885×10^6	1.5130×10^8
GHS_indef/boyd2	466316	1500397	2.5854×10^6	1.5582×10^7
GHS_indef/stokes128	49666	558594	2.9813×10^6	3.6881×10^8
GHS_indef/cvxqp3	17500	122462	3.1398×10^6	1.7670×10^9
TSOPF/TSOPF_FS_b162_c3	30798	1801300	4.2194×10^6	6.3970×10^8
TSOPF/TSOPF_FS_b39_c19	76216	1977600	4.4010×10^6	2.8666×10^8
GHS_indef/cont-201	80595	438795	4.7815×10^6	8.6513×10^8
TSOPF/TSOPF_FS_b162_c4	40798	2398220	5.5772×10^6	8.3825×10^8
GHS_indef/bratu3d	27792	173796	6.2769×10^6	4.4174×10^9
TSOPF/TSOPF_FS_b39_c30	120216	3121160	7.0473×10^6	4.7541×10^8
GHS_indef/darcy003	389874	2101242	8.1587×10^6	5.5664×10^8
Schenk_IBMNA/c-62	41731	559343	8.4634×10^6	8.0165×10^9
TSOPF/TSOPF_FS_b300	29214	4400122	1.0603×10^7	4.3977×10^9
TSOPF/TSOPF_FS_b300_c1	29214	4400122	1.0603×10^7	4.3977×10^9
GHS_indef/cont-300	180895	988195	1.1744×10^7	2.9559×10^9
GHS_indef/ncvxqp5	62500	424966	1.2052×10^7	9.7223×10^9
GHS_indef/turon_m	189924	1690876	1.3723×10^7	4.2270×10^9
GHS_indef/d_pretok	182730	1641672	1.4581×10^7	5.0572×10^9
GHS_indef/ncvxqp3	75000	499964	1.9007×10^7	2.0692×10^{10}
TSOPF/TSOPF_FS_b300_c2	56814	8767466	2.1433×10^7	8.9629×10^9
GHS_indef/ncvxqp7	87500	574962	2.4731×10^7	3.0939×10^{10}
TSOPF/TSOPF_FS_b300_c3	84414	13135930	3.3105×10^7	1.4253×10^{10}

Table 5.2: Structurally singular test problems. n denotes the order of A, $srank$ is the structural rank of A, $nz(A)$ is the number of entries in A, nzL is the predicted number of entries in L and $nflop$ is the predicted number of flops required to compute L. † indicates OXO matrix.

Identifier	n	$srank$	$nz(A)$	nzL	$nflop$
GHS_indef/dtoc†	24993	19994	69972	2.1314×10^5	2.0247×10^6
GHS_indef/bloweybl	30003	30002	120000	2.6137×10^5	2.5881×10^6
Marini/eurqsa	7246	7245	46142	2.9351×10^5	2.6890×10^7
GHS_indef/aug2d†	29008	19208	76832	5.4805×10^5	2.1346×10^7
GHS_indef/aug3d†	24300	11664	69984	1.2007×10^6	2.3221×10^8
Pajek/Reuters911	13332	10685	296076	4.2982×10^6	6.3985×10^9
Newman/cond-mat-2003	31163	29174	240058	6.3105×10^6	8.0092×10^9
Newman/cond-mat-2005	40421	37968	351386	1.2563×10^7	2.4313×10^{10}

Table 5.3: The number of delayed pivots (*ndelay*), actual number of entries and flops for L with the ratio against predicted values, and number of steps (*nitr*) of iterative refinement. Default settings (pivot threshold $u = 0.01$), METIS ordering and no scaling.

Identifier	*ndelay*	*nzL*		*nflop*		*nitr*
		Actual	Ratio	Actual	Ratio	
TSOPF/TSOPF_FS_b162_c1	8437	2.25×10^6	1.62	6.80×10^8	3.32	0
TSOPF/TSOPF_FS_b39_c7	14563	2.53×10^6	1.61	4.72×10^8	4.38	0
Schenk_IBMNA/c-64	22832	4.23×10^6	2.56	3.04×10^9	24.12	0
GHS_indef/ncvxqp1	124164	1.18×10^7	7.01	2.67×10^{10}	36.62	0
QY/case39	19012	3.41×10^6	1.49	5.71×10^8	3.78	0
GHS_indef/boyd2	27079	7.89×10^7	30.53	2.98×10^{11}	19112	0
GHS_indef/stokes128	39083	4.92×10^6	1.65	9.54×10^8	2.59	0
GHS_indef/cvxqp3	312790	3.57×10^7	11.38	1.65×10^{11}	93.55	0
TSOPF/TSOPF_FS_b162_c3	29472	8.02×10^6	1.90	4.32×10^9	6.76	0
TSOPF/TSOPF_FS_b39_c19	40727	8.25×10^6	1.87	3.98×10^9	13.90	0
GHS_indef/cont-201	70024	8.57×10^6	1.79	2.60×10^9	3.00	1
TSOPF/TSOPF_FS_b162_c4	36807	1.19×10^7	2.13	8.87×10^9	10.58	0
GHS_indef/bratu3d	59105	1.18×10^7	1.89	1.23×10^{10}	2.77	1
TSOPF/TSOPF_FS_b39_c30	64326	1.49×10^7	2.11	1.28×10^{10}	27.00	0
GHS_indef/darcy003	43702	8.92×10^6	1.09	5.99×10^8	1.08	1
Schenk_IBMNA/c-62	131085	3.40×10^7	4.01	9.92×10^{10}	12.37	0
TSOPF/TSOPF_FS_b300	21544	1.29×10^7	1.22	6.57×10^9	1.49	0
TSOPF/TSOPF_FS_b300_c1	46582	1.68×10^7	1.58	1.30×10^{10}	2.95	0
GHS_indef/cont-300	148992	2.15×10^7	1.83	9.84×10^9	3.33	1
GHS_indef/ncvxqp5	544258	1.14×10^8	9.50	9.00×10^{11}	92.58	1
GHS_indef/turon_m	18929	1.44×10^7	1.05	4.33×10^9	1.02	0
GHS_indef/d_pretok	23904	1.54×10^7	1.06	5.22×10^9	1.03	0
GHS_indef/ncvxqp3	1661198	3.79×10^8	19.96	5.97×10^{12}	288.3	1
TSOPF/TSOPF_FS_b300_c2	97464	3.61×10^7	1.68	3.41×10^{10}	3.80	0
GHS_indef/ncvxqp7	3784535	8.29×10^8	33.51	2.05×10^{13}	663.9	1
TSOPF/TSOPF_FS_b300_c3	200549	6.25×10^7	1.89	8.35×10^{10}	5.86	0
GHS_indef/dtoc	47628	1.13×10^7	53.20	3.29×10^{10}	16229	0
GHS_indef/bloweybl	72303	5.11×10^7	195.4	3.38×10^{11}	130408	1
Marini/eurqsa	14678	2.19×10^6	7.46	1.75×10^9	65.16	0
GHS_indef/aug2d	64859	6.47×10^6	11.81	4.72×10^9	221.3	0
GHS_indef/aug3d	291547	3.83×10^7	31.92	1.34×10^{11}	578.9	0
Pajek/Reuters911	46221	2.32×10^7	5.39	8.87×10^{10}	13.86	0
Newman/cond-mat-2003	81018	1.26×10^7	2.00	2.34×10^{10}	2.93	1
Newman/cond-mat-2005	135871	2.37×10^7	1.89	5.98×10^{10}	2.46	0

count (see, for instance, GHS_indef/darcy003 where the increase is less than 10%). Note that, with the exception of the Newman problems, the singular examples are saddle-point systems (4.4) with $C = 0$ or C diagonal. For all our test problems, when `HSL_MA77` is run with default settings, at most one step of iterative refinement is required to achieve the required accuracy.

5.2 Effect of scaling

The results in this section illustrate how scaling helps in the solution of tough indefinite problems. The number of delayed pivots with no scaling, `MC64` scaling and `MC77` scaling (using the Ruiz and Uçar [39] approach of one step of $\infty-$norm scaling followed by a three steps of $1-$norm scaling) are given in Table 5.4, with a comparison of numbers of entries in the factors and floating point operations in Table 5.5. As we would expect, the effects of scaling are highly problem-dependent. For some problems (including Marini/eurqsa, GHS_indef/boyd2 and Schenk_IBMNA/c-62), scaling with `MC64` reduces the number of delays to (close to) zero. For these problems, `MC77` also reduces the number of delays but less dramatically. For some problems, `MC77` is competitive but, in general, `MC64` gives better results than `MC77`. This is consistent with the experiences reported in [22]. However, for a significant proportion of the test set, despite scaling with `MC64`, there are still a large number of delays. Indeed, in some cases, `MC64` gives no worthwhile reduction in the number of delays (for example, the OXO matrices and GHS_indef/cont-x matrices). Based on these results, all the experiments in the remainder of this paper use `MC64` scaling.

Table 5.4: The number of delayed pivots (*ndelay*) and number of steps (*nitr*) of iterative refinement with no scaling and with `MC64` and `MC77` scaling (pivot threshold $u = 0.01$).

Identifier	*ndelay*			*nitr*		
	None	`MC64`	`MC77`	None	`MC64`	`MC77`
TSOPF/TSOPF_FS_b162_c1	8437	5087	7072	0	1	1
TSOPF/TSOPF_FS_b39_c7	14563	11238	12366	0	1	1
Schenk_IBMNA/c-64	22832	752	1948	0	1	1
GHS_indef/ncvxqp1	124164	10359	31703	0	1	0
QY/case39	19012	14806	16631	0	1	1
GHS_indef/boyd2	27079	0	6276	0	1	1
GHS_indef/stokes128	39083	9274	9274	0	0	2
GHS_indef/cvxqp3	312790	26152	34336	0	1	1
TSOPF/TSOPF_FS_b162_c3	29472	17497	24233	0	1	1
TSOPF/TSOPF_FS_b39_c19	40727	29668	32005	0	1	1
GHS_indef/cont-201	70024	70021	69975	1	1	1
TSOPF/TSOPF_FS_b162_c4	36807	21778	30705	0	1	1
GHS_indef/bratu3d	59105	58888	59391	1	1	1
TSOPF/TSOPF_FS_b39_c30	64326	47578	53073	0	1	0
GHS_indef/darcy003	43702	43702	43702	1	1	1
Schenk_IBMNA/c-62	131085	604	70018	0	1	0
TSOPF/TSOPF_FS_b300	21544	20596	20519	0	1	1
TSOPF/TSOPF_FS_b300_c1	46582	24511	42112	0	1	1
GHS_indef/cont-300	148992	148976	148856	1	1	1
GHS_indef/ncvxqp5	544258	11869	13252	1	1	1
GHS_indef/turon_m	18929	18931	18929	0	0	0
GHS_indef/d_pretok	23904	21399	21030	0	0	0
GHS_indef/ncvxqp3	1661198	65603	88408	1	1	1
TSOPF/TSOPF_FS_b300_c2	97464	52593	89385	0	1	1
GHS_indef/ncvxqp7	3784535	272409	369372	1	1	1
TSOPF/TSOPF_FS_b300_c3	200549	116630	186277	1	1	1
GHS_indef/dtoc	47628	44520	44520	0	0	0
GHS_indef/bloweybl	72303	3117	4499	1	1	0
Marini/eurqsa	14678	306	7821	0	0	0
GHS_indef/aug2d	64859	64563	64978	0	0	0
GHS_indef/aug3d	291547	295845	268683	0	0	0
Pajek/Reuters911	46221	45745	45713	0	0	0
Newman/cond-mat-2003	81018	75956	76880	1	0	0
Newman/cond-mat-2005	135871	128402	131590	1	0	0

5.3 Relaxing the pivot test

In Table 5.6 we report results for using a small threshold parameter $u = 10^{-8}$ and for relaxed threshold pivoting, with $u_{min} = 10^{-8}$ (the computation starts with $u = 0.01$ and relaxes the threshold during the computation, provided it remains at least u_{min}). In our tests, using a small value of u is only moderately successful in reducing the number of delays and for some problems, iterative refinement fails to achieve the requested accuracy. Using relaxed threshold pivoting also has a limited effect on the number of delays and gives similar results to using $u = 10^{-8}$. FGMRES is used when iterative refinement does not converge. FGMRES was proposed in [2, 3] as an alternative to iterative refinement; it was shown that, for some problems, it is able to compute backward stable solutions when iterative refinement fails to converge. We use the restarted FGMRES algorithm described in [23]. This is essentially as given in [2] but it additionally

Table 5.5: Comparison of the `MC64` and `MC77` scalings. The numbers are ratios of the number of entries in L and number of floating point operations to those predicted by the analyse phase.

Identifier	nzL			$nflop$		
	None	`MC64`	`MC77`	None	`MC64`	`MC77`
TSOPF/TSOPF_FS_b162_c1	1.62	1.31	1.48	3.32	1.82	2.59
TSOPF/TSOPF_FS_b39_c7	1.61	1.40	1.42	4.38	2.56	2.58
Schenk_IBMNA/c-64	2.56	1.02	1.05	24.12	1.05	1.12
GHS_indef/ncvxqp1	7.01	1.35	2.01	36.62	1.68	4.06
QY/case39	1.49	1.36	1.39	3.78	2.63	2.76
GHS_indef/boyd2	30.53	1.00	4.83	19112	1.00	1344
GHS_indef/stokes128	1.65	1.10	1.10	2.59	1.09	1.09
GHS_indef/cvxqp3	11.38	1.56	1.69	93.55	2.16	2.64
TSOPF/TSOPF_FS_b162_c3	1.90	1.44	1.62	6.76	2.78	3.99
TSOPF/TSOPF_FS_b39_c19	1.87	1.49	1.40	13.90	5.38	3.07
GHS_indef/cont-201	1.79	1.79	1.79	3.00	3.00	3.00
TSOPF/TSOPF_FS_b162_c4	2.13	1.49	1.78	10.58	3.37	6.01
GHS_indef/bratu3d	1.89	1.88	1.90	2.77	2.75	2.81
TSOPF/TSOPF_FS_b39_c30	2.11	1.61	1.43	27.00	9.81	3.80
GHS_indef/darcy003	1.09	1.09	1.09	1.08	1.08	1.08
Schenk_IBMNA/c-62	4.01	1.01	2.25	12.37	1.02	3.60
TSOPF/TSOPF_FS_b300	1.22	1.20	1.20	1.49	1.45	1.44
TSOPF/TSOPF_FS_b300_c1	1.58	1.29	1.46	2.95	1.75	2.37
GHS_indef/cont-300	1.83	1.83	1.83	3.33	3.33	3.32
GHS_indef/ncvxqp5	9.50	1.11	1.12	92.58	1.17	1.19
GHS_indef/turon_m	1.05	1.05	1.05	1.02	1.02	1.02
GHS_indef/d_pretok	1.06	1.05	1.05	1.03	1.03	1.03
GHS_indef/ncvxqp3	19.96	1.32	1.40	288.3	1.61	1.84
TSOPF/TSOPF_FS_b300_c2	1.68	1.35	1.56	3.80	2.03	2.94
GHS_indef/ncvxqp7	33.51	1.59	1.80	663.9	2.26	3.09
TSOPF/TSOPF_FS_b300_c3	1.89	1.38	1.61	5.86	2.26	3.54
GHS_indef/dtoc	53.20	45.24	45.24	16229	12476	12476
GHS_indef/bloweybl	195.4	1.14	1.19	130408	1.24	1.34
Marini/eurqsa	7.46	1.06	3.21	65.16	1.17	14.20
GHS_indef/aug2d	11.81	11.62	12.30	221.3	222.1	258.0
GHS_indef/aug3d	31.92	32.59	27.61	578.9	592.4	449.5
Pajek/Reuters911	5.39	5.37	5.37	13.86	13.74	13.74
Newman/cond-mat-2003	2.00	1.95	1.96	2.93	2.79	2.82
Newman/cond-mat-2005	1.89	1.85	1.87	2.46	2.35	2.39

Table 5.6: The number of delayed pivots (*ndelay*) and number of refinement solves (*nitr*) with pivot threshold $u = 0.01$ (the default), $u = 10^{-8}$ and relaxed pivoting with $u_{min} = 10^{-8}$. $*$ indicates FGMRES was required to achieve convergence to the required accuracy.

Identifier	*ndelay*			*nitr*			u_{min}
	$u = 0.01$	$u = 10^{-8}$	$u_{min} = 10^{-8}$	$u = 0.01$	$u = 10^{-8}$	$u_{min} = 10^{-8}$	used
TSOPF/TSOPF_FS_b162_c1	5087	3961	3986	1	2	2	1.11×10^{-8}
TSOPF/TSOPF_FS_b39_c7	11238	8527	8552	1	2	2	2.72×10^{-8}
Schenk_IBMNA/c-64	752	734	734	1	1	1	1.01×10^{-8}
GHS_indef/ncvxqp1	10359	9267	9313	1	1	1	1.93×10^{-8}
QY/case39	14806	11011	11024	1	2	2	1.10×10^{-8}
GHS_indef/boyd2	0	0	0	1	1	1	1.00×10^{-2}
GHS_indef/stokes128	9274	9274	9274	0	0	0	1.00×10^{-2}
GHS_indef/cvxqp3	26152	26126	26145	1	1	1	3.12×10^{-3}
TSOPF/TSOPF_FS_b162_c3	17497	12089	12114	1	2	3	1.01×10^{-8}
TSOPF/TSOPF_FS_b39_c19	29668	22610	22647	1	2	2	1.10×10^{-8}
GHS_indef/cont-201	70021	66002	66002	1	24$*$	12$*$	1.01×10^{-8}
TSOPF/TSOPF_FS_b162_c4	21778	15926	15917	1	2	3	1.01×10^{-8}
GHS_indef/bratu3d	58888	41829	41869	1	44$*$	36$*$	1.00×10^{-8}
TSOPF/TSOPF_FS_b39_c30	47578	37293	37277	1	3	3	1.05×10^{-8}
GHS_indef/darcy003	43702	43702	43702	1	1	1	1.00×10^{-2}
Schenk_IBMNA/c-62	604	583	583	1	1	1	2.92×10^{-5}
TSOPF/TSOPF_FS_b300	20599	19033	19040	1	5	7	1.02×10^{-8}
TSOPF/TSOPF_FS_b300_c1	24511	18928	19317	1	4	3	1.00×10^{-8}
GHS_indef/cont-300	148976	141091	141091	1	12$*$	12$*$	1.00×10^{-8}
GHS_indef/ncvxqp5	11869	10636	10636	1	1	1	2.83×10^{-5}
GHS_indef/turon_m	18931	18928	18928	0	0	0	1.43×10^{-3}
GHS_indef/d_pretok	21399	18012	18012	0	0	0	4.32×10^{-5}
GHS_indef/ncvxqp3	65603	64876	64853	1	2	1	3.74×10^{-5}
TSOPF/TSOPF_FS_b300_c2	52593	41836	41853	1	5	5	1.00×10^{-8}
GHS_indef/ncvxqp7	272409	270790	270812	1	2	1	8.02×10^{-7}
TSOPF/TSOPF_FS_b300_c3	116630	99361	99361	1	5	5	1.01×10^{-8}
GHS_indef/dtoc	44520	12391	20792	0	1	0	2.00×10^{-4}
GHS_indef/bloweybl	3117	3117	3117	1	1	1	1.00×10^{-2}
Marini/eurqsa	306	267	250	0	2	1	1.09×10^{-8}
GHS_indef/aug2d	64563	64563	64563	0	0	0	1.00×10^{-2}
GHS_indef/aug3d	295845	295845	295845	0	0	0	1.00×10^{-2}
Pajek/Reuters911	45745	45745	45745	0	0	0	1.00×10^{-2}
Newman/cond-mat-2003	75956	75836	75742	0	1	0	1.00×10^{-6}
Newman/cond-mat-2005	128402	129744	129696	0	1	3	4.50×10^{-8}

uses an adaptive restart parameter that was found in numerical experiments to be more efficient than using a fixed restart parameter (that is, in general, it reduced the number of iterations required). Our choice of initial restart parameter of 4 is based on the results given in [23]. Note that the results are sensitive to this choice: using a larger value can lead to a larger total number of solves because we only test the termination conditions when the algorithm is restarted.

In Table 5.7 we report ratios of the number of entries in L and the number of flops required to compute the factorization for $u = 0.01$ and $u = 10^{-8}$ against the predicted values.

In our tests on static pivoting, the parameter *static* (see Section 3.5) is set to $\|\hat{A}\|\sqrt{\epsilon}$, where ϵ is the machine precision and $\hat{A}$ is the scaled matrix. There are no delayed pivots so the number of entries in the computed factor and the flop counts are as given in columns 3 and 5 of Table 5.3. In Table 5.8, we report the number of solves performed using iterative refinement and FGMRES to recover the required accuracy (with a limit of 100 solves); the number of diagonal entries that were perturbed is also given. We see that, if the required accruacy is achieved after a small number of steps of iterative refinement, then using iterative refinement is more efficient than FGMRES. But FGMRES is more robust and, if a large number of steps of iterative refinement is needed, FGMRES can require fewer solves. However, for two

Table 5.7: Ratios of the number of entries in L and the number of flops for $u = 0.01$ and $u = 10^{-8}$ and for $u_{min} = 10^{-8}$ against the predicted values.

Identifier	*nzL*			*nflop*		
	$u = 0.01$	$u = 10^{-8}$	$u_{min} = 10^{-8}$	$u = 0.01$	$u = 10^{-8}$	$u_{min} = 10^{-8}$
TSOPF/TSOPF_FS_b162_c1	1.31	1.20	1.20	1.82	1.45	1.46
TSOPF/TSOPF_FS_b39_c7	1.40	1.24	1.24	2.56	1.64	1.64
Schenk_IBMNA/c-64	1.02	1.02	1.02	1.05	1.05	1.05
GHS_indef/ncvxqp1	1.35	1.32	1.32	1.68	1.61	1.61
QY/case39	1.36	1.21	1.21	2.63	1.56	1.55
GHS_indef/boyd2	1.00	1.00	1.00	1.00	1.00	1.00
GHS_indef/stokes128	1.10	1.10	1.10	1.09	1.09	1.09
GHS_indef/cvxqp3	1.56	1.56	1.56	2.16	2.16	2.16
TSOPF/TSOPF_FS_b162_c3	1.44	1.23	1.23	2.78	1.52	1.52
TSOPF/TSOPF_FS_b39_c19	1.49	1.22	1.22	5.38	1.56	1.56
GHS_indef/cont-201	1.79	1.72	1.72	3.00	2.75	2.75
TSOPF/TSOPF_FS_b162_c4	1.49	1.25	1.24	3.37	1.60	1.60
GHS_indef/bratu3d	1.88	1.53	1.53	2.75	1.81	1.81
TSOPF/TSOPF_FS_b39_c30	1.61	1.22	1.22	9.81	1.56	1.56
GHS_indef/darcy003	1.09	1.09	1.09	1.08	1.08	1.08
Schenk_IBMNA/c-62	1.01	1.01	1.01	1.02	1.02	1.02
TSOPF/TSOPF_FS_b300	1.20	1.19	1.19	1.45	1.41	1.41
TSOPF/TSOPF_FS_b300_c1	1.29	1.19	1.19	1.75	1.41	1.41
GHS_indef/cont-300	1.83	1.77	1.77	3.33	3.11	3.11
GHS_indef/ncvxqp5	1.11	1.11	1.11	1.17	1.16	1.16
GHS_indef/turon_m	1.05	1.05	1.05	1.02	1.02	1.02
GHS_indef/d_pretok	1.05	1.04	1.04	1.03	1.02	1.02
GHS_indef/ncvxqp3	1.32	1.32	1.32	1.61	1.60	1.60
TSOPF/TSOPF_FS_b300_c2	1.35	1.21	1.21	2.03	1.50	1.50
GHS_indef/ncvxqp7	1.59	1.58	1.58	2.26	2.25	2.25
TSOPF/TSOPF_FS_b300_c3	1.38	1.19	1.19	2.26	1.43	1.43
GHS_indef/dtoc	45.24	1.79	3.60	12475	4.44	49.82
GHS_indef/bloweybl	1.14	1.14	1.14	1.24	1.24	1.24
Marini/eurqsa	1.06	1.05	1.05	1.17	1.15	1.14
GHS_indef/aug2d	11.62	11.62	11.62	222.1	222.1	222.1
GHS_indef/aug3d	32.59	32.59	32.59	592.4	592.4	592.4
Pajek/Reuters911	5.37	5.37	5.37	13.74	13.74	13.74
Newman/cond-mat-2003	1.95	1.95	1.95	2.79	2.80	2.79
Newman/cond-mat-2005	1.85	1.86	1.85	2.35	2.37	2.36

Table 5.8: The number of refinement solves performed using iterative refinement (IR) and FGMRES following static pivoting. *nptb* is the number of perturbed diagonal entries. - indicates required accuracy not achieved.

Identifier	*nptb*	*nitr*	
		IR	FGMRES
TSOPF/TSOPF_FS_b162_c1	2034	4	4
TSOPF/TSOPF_FS_b39_c7	5846	6	4
Schenk_IBMNA/c-64	734	4	4
GHS_indef/ncvxqp1	1310	2	4
QY/case39	7163	11	8
GHS_indef/boyd2	0	2	4
GHS_indef/stokes128	4988	-	-
GHS_indef/cvxqp3	2833	8	8
TSOPF/TSOPF_FS_b162_c3	6254	5	4
TSOPF/TSOPF_FS_b39_c19	14491	6	8
GHS_indef/cont-201	8569	-	24
TSOPF/TSOPF_FS_b162_c4	8569	4	4
GHS_indef/bratu3d	8569	4	4
TSOPF/TSOPF_FS_b39_c30	23895	11	8
GHS_indef/darcy003	19865	18	12
Schenk_IBMNA/c-62	294	3	4
TSOPF/TSOPF_FS_b300	5627	8	8
TSOPF/TSOPF_FS_b300_c1	5642	10	8
GHS_indef/cont-300	38946	-	52
GHS_indef/ncvxqp5	2720	-	40
GHS_indef/turon_m	9026	-	76
GHS_indef/d_pretok	8645	-	60
GHS_indef/ncvxqp3	7966	-	-
TSOPF/TSOPF_FS_b300_c2	11648	9	8
GHS_indef/ncvxqp7	13741	64	20
TSOPF/TSOPF_FS_b300_c3	16089	-	40
GHS_indef/dtoc	5273	0	0
GHS_indef/bloweybl	3117	1	4
Marini/eurqsa	95	-	12
GHS_indef/aug2d	12436	2	4
GHS_indef/aug3d	14980	2	4
Pajek/Reuters911	4869	18	32
Newman/cond-mat-2003	3307	3	4
Newman/cond-mat-2005	4555	-	28

of our test problems (GHS_indef/stokes128 and GHS_indef/ncvxqp3), we were not able to recovery the required accuracy after using static pivoting.

Using a small pivot threshold or using static pivoting can result in many more calls to the solve phase as refinement is needed to recover accuracy. Traditionally, the factorize phase of a sparse direct solver has generally required the greatest portion of the total execution time. It typically involves the majority of the floating-point operations (often as many as 98% of the flops are performed by the factorize phase), and is computation bound; the other phases are memory bound. Over the past decade or so, the increase in computational capacity (flops per second) has vastly outstripped the increase in memory bandwidth (bytes per second). The result of this for sparse direct solvers has been an increase in the proportion of the computation time taken by the solve phase (see [25] for further discussion and computational results). Thus, although static pivoting can eliminate delays, there is a potentially costly penalty to pay of subsequently requiring many solves and, since the solve does not parallelise well, this can become a bottleneck.

5.4 Constrained ordering

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 H-node neighbours have been ordered (that is, a C-node is postponed until after all its H-node neighbours). 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. Bridson reports that neither approach consistently outperforms the other and so in our experiments we apply post processing to the nested dissection ordering computed by METIS.

We remark that if, in the constrained ordering, an H-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 [17]). Note also that when an H-node is ordered there may be more than one C-node neighbour waiting to be ordered. As in [44], we include these C-nodes in nested dissection order.

Table 5.9 presents results for the constrained ordering with $u = 0.0$ (no pivoting). Only a subset of our test problems are included since this approach is limited to KKT systems with H definite. With $u = 0.0$ there are no delays and the actual number of entries in L and actual number of flops are equal to the predicted values. Comparing the ratios with those in Table 5.7, we see that the penalty for no delays is denser factors and higher flop counts than for the corresponding unconstrained ordering. For some problems, including Schenk_IBMNA/c-64, the ratios for the constrained ordering are much greater than for the METIS ordering.

Table 5.9: Results for the constrained ordering with pivot threshold $u = 0.01$. The ratios of the number of entries in L and flop counts to those predicted for the METIS ordering is given together with the number of refinement solves.

Identifier	*nzL*	*nflop*	*nitr*
Schenk_IBMNA/c-64	72.74	6519	1
GHS_indef/cvxqp3	3.51	9.91	22
GHS_indef/cont-201	1.74	2.75	1
GHS_indef/darcy003	3.08	11.78	0
Schenk_IBMNA/c-62	4.81	14.66	1
GHS_indef/cont-300	1.79	2.89	1
GHS_indef/turon_m	3.89	11.26	0
GHS_indef/d_pretok	3.93	11.23	0

5.5 `MA47` ordering

Results for the ordering computed using the analyse phase of `MA47` are given in Table 5.10 (`MC64` scaling is used and threshdold $u = 0.01$). We see that there are no delays for the OXO matrices and for many of the remaining test problems there are fewer delays than with the METIS ordering using the same settings (see column 2 of Table 5.3). However, for large problems, minimum degree-based orderings are generally not as effective as those based on nested dissection and so the number of entries in L and the flop counts are, for some cases, significantly larger for the `MA47` ordering. We experimented with using $u = 10^{-8}$. This led to some reduction in the number of delays but for a number of the test problems FGMRES refinement was needed to recover accuracy.

Table 5.10: Results for the `MA47` ordering with pivot threshold $u = 0.01$. The number of delays, the ratios of the predicted and actual number of entries in L and flop counts to the predicted values using METIS ordering, and the number of refinement solves are given. NS indicates not solved within a time limit of an hour.

Identifier	*ndelay*	*nzL*		*nflop*		*nitr*
		Predicted	Actual	Predicted	Actual	
TSOPF/TSOPF_FS_b162_c1	225	9.56	9.67	217.8	221.4	1
TSOPF/TSOPF_FS_b39_c7	883	15.96	17.22	572.6	672.7	1
Schenk_IBMNA/c-64	42	1.00	1.00	0.89	0.92	1
GHS_indef/ncvxqp1	10731	12.22	12.36	99.77	101.71	1
QY/case39	4800	13.75	15.14	416.2	520.4	1
GHS_indef/boyd2	0	1.00	1.00	1.00	1.00	1
GHS_indef/stokes128	2548	1.07	1.12	1.29	1.37	0
GHS_indef/cvxqp3	18383	1.56	3.63	3.32	13.28	1
TSOPF/TSOPF_FS_b162_c3	441	26.97	27.09	1769	1782	1
TSOPF/TSOPF_FS_b39_c19	4640	26.20	29.01	2950	3551	1
GHS_indef/cont-201	185801	47.45	55.88	3166	4134	1
TSOPF/TSOPF_FS_b162_c4	532	35.86	36.08	3143	3179	1
GHS_indef/bratu3d	250517	15.67	16.03	155.6	163.0	1
TSOPF/TSOPF_FS_b39_c30	10135	37.37	41.72	6832	8235	1
GHS_indef/darcy003	18909	1.26	1.32	2.48	2.57	1
Schenk_IBMNA/c-62	421	1.42	1.48	2.57	2.76	1
TSOPF/TSOPF_FS_b300	1898	9.28	9.37	206.4	209.2	0
TSOPF/TSOPF_FS_b300_c1	2430	9.28	9.39	206.4	210.1	1
GHS_indef/cont-300	NS	95.48	NS	9998	NS	NS
GHS_indef/ncvxqp5	NS	40.39	NS	772.2	NS	NS
GHS_indef/turon_m	15101	3.21	3.28	14.55	14.65	0
GHS_indef/d_pretok	17715	2.85	2.92	11.27	11.40	0
GHS_indef/ncvxqp3	NS	49.56	NS	1077	NS	NS
TSOPF/TSOPF_FS_b300_c2	NS	18.05	NS	793.7	NS	NS
GHS_indef/ncvxqp7	NS	52.10	NS	1188	NS	NS
TSOPF/TSOPF_FS_b300_c3	NS	26.12	NS	1671	NS	NS
GHS_indef/dtoc	0	0.88	0.88	0.80	0.80	0
GHS_indef/bloweybl	3330	0.79	0.92	0.64	0.82	0
Marini/eurqsa	796	1.40	1.48	2.30	2.66	1
GHS_indef/aug2d	0	7.92	7.92	31.81	31.81	0
GHS_indef/aug3d	0	9.85	9.85	26.22	26.22	0
Pajek/Reuters911	4462	16.48	16.49	74.52	74.60	0
Newman/cond-mat-2003	47830	21.96	22.02	119.4	119.8	0
Newman/cond-mat-2005	106538	21.82	21.89	113.4	114.0	0

5.6 Matching-based ordering

Results for the matching-based ordering are given in Tables 5.11 and 5.12 for $u = 0.01$ and $u = 10^{-8}$, respectively. The implementation used is that provided by `HSL_MC80`, which in turn exploits `MC64` to obtain the matching and scaling. In our tests, METIS is applied to the compressed graph. We see that the predictions for the matching-based ordering can be substantially greater than for the METIS ordering. In many instances, the predicted nzL is between 50 and 100% greater for the matching ordering. However, the matching-ordering results in substantially fewer delayed pivots and for some problems, eliminates delays altogether (for example, GHS_indef/cont-201 and GHS_indef/bratu3d). This is important as it enables the solver to obey scheduling based on the analyse phase more accurately. If we run the matching ordering with $u = 10^{-8}$, with the exception of the singular problems, the number of delays is reduced to 0 (or close to 0) and refinement is not required. For some problems, such as GHS_indef/ncvxqp1 and GHS_indef/stokes128, the METIS ordering produces sparser factors (despite having a large number of delays) but for others, including GHS_indef/cont-300, the matching-based ordering produces the sparsest factors.

Table 5.11: Results for the matching-based ordering with pivot threshold $u = 0.01$. In each case, refinement is not needed to achieve the required accuracy. The number of delays, and the ratios of the predicted and actual number of entries in L and flop counts to the predicted values using METIS ordering are given.

Identifier	$ndelay$	nzL		$nflop$	
		Predicted	Actual	Predicted	Actual
TSOPF/TSOPF_FS_b162_c1	348	1.24	1.26	1.52	1.58
TSOPF/TSOPF_FS_b39_c7	1356	1.46	1.50	1.88	1.98
Schenk_IBMNA/c-64	0	1.25	1.25	1.70	1.70
GHS_indef/ncvxqp1	25	1.70	1.70	2.83	2.84
QY/case39	5925	1.55	1.63	2.26	2.53
GHS_indef/boyd2	0	1.00	1.00	1.00	1.00
GHS_indef/stokes128	5	1.59	1.59	2.23	2.23
GHS_indef/cvxqp3	64	1.82	1.82	3.08	3.08
TSOPF/TSOPF_FS_b162_c3	1264	1.40	1.43	1.96	2.03
TSOPF/TSOPF_FS_b39_c19	9010	1.49	1.56	2.15	2.44
GHS_indef/cont-201	0	0.95	0.95	0.87	0.87
TSOPF/TSOPF_FS_b162_c4	1410	1.21	1.23	1.46	1.52
GHS_indef/bratu3d	0	0.97	0.97	0.96	0.96
TSOPF/TSOPF_FS_b39_c30	5699	1.40	1.46	1.82	2.22
GHS_indef/darcy003	119	1.75	1.75	3.95	3.95
Schenk_IBMNA/c-62	0	1.71	1.71	2.09	2.09
TSOPF/TSOPF_FS_b300	1017	1.22	1.24	1.48	1.54
TSOPF/TSOPF_FS_b300_c1	1269	1.24	1.27	1.53	1.61
GHS_indef/cont-300	0	0.95	0.95	0.87	0.87
GHS_indef/ncvxqp5	112	1.64	1.64	2.36	2.36
GHS_indef/turon_m	129	1.36	1.37	1.52	1.52
GHS_indef/d_pretok	313	1.38	1.38	1.48	1.48
GHS_indef/ncvxqp3	220	1.84	1.84	2.90	2.90
TSOPF/TSOPF_FS_b300_c2	2533	1.21	1.24	1.47	1.55
GHS_indef/ncvxqp7	195	1.75	1.75	2.77	2.77
TSOPF/TSOPF_FS_b300_c3	6764	1.15	1.19	1.34	1.44
GHS_indef/dtoc	48399	1.06	26.38	1.12	3760
GHS_indef/bloweybl	0	1.04	1.04	1.04	1.04
Marini/eurqsa	252	1.64	1.67	3.20	3.36
GHS_indef/aug2d	119457	1.50	23.06	2.51	1106
GHS_indef/aug3d	517235	1.78	17.85	2.63	145.4
Pajek/Reuters911	846509	2.87	5.19	3.61	10.06
Newman/cond-mat-2003	137023	1.96	2.33	2.66	3.46
Newman/cond-mat-2005	258421	1.91	2.24	2.27	2.94

Table 5.12: Comparison of the METIS and the matching-based orderings for pivot threshold $u = 10^{-8}$. The number of delays, the ratios of the actual number of entries in L and flop counts to the predicted values using METIS ordering, and the number of refinement solves are given. $*$ indicates FGMRES used for refinement.

Identifier	METIS				matching			
	$ndelay$	nzL	$nflop$	$nitr$	$ndelay$	nzL	$nflop$	$nitr$
TSOPF/TSOPF_FS_b162_c1	3961	1.20	1.45	2	0	1.24	1.52	0
TSOPF/TSOPF_FS_b39_c7	8527	1.24	1.64	2	0	1.46	1.88	0
Schenk_IBMNA/c-64	734	1.02	1.05	1	0	1.24	1.72	0
GHS_indef/ncvxqp1	9267	1.32	1.61	1	25	1.70	2.84	0
QY/case39	11011	1.21	1.56	2	0	1.55	2.26	0
GHS_indef/boyd2	0	1.00	1.00	1	0	1.00	1.00	0
GHS_indef/stokes128	9274	1.10	1.09	0	5	1.59	2.23	0
GHS_indef/cvxqp3	26145	1.56	2.16	1	23	1.82	3.08	0
TSOPF/TSOPF_FS_b162_c3	12126	1.23	1.52	2	0	1.40	1.96	0
TSOPF/TSOPF_FS_b39_c19	22610	1.22	1.56	2	0	1.49	2.15	0
GHS_indef/cont-201	66002	1.72	2.75	24$*$	0	0.95	0.87	0
TSOPF/TSOPF_FS_b162_c4	15926	1.25	1.60	3	0	1.21	1.46	0
GHS_indef/bratu3d	41829	1.53	1.81	44$*$	0	0.97	0.96	0
TSOPF/TSOPF_FS_b39_c30	37293	1.22	1.56	3	0	1.40	1.82	0
GHS_indef/darcy003	43702	1.09	1.08	1	119	1.75	3.95	0
Schenk_IBMNA/c-62	583	1.01	1.02	1	0	1.71	2.09	0
TSOPF/TSOPF_FS_b300	19033	1.19	1.41	5	0	1.22	1.48	0
TSOPF/TSOPF_FS_b300_c1	19310	1.19	1.41	4	0	1.24	1.53	0
GHS_indef/cont-300	141089	1.77	3.11	12$*$	0	0.95	0.87	0
GHS_indef/ncvxqp5	10636	1.11	1.16	1	0	1.64	2.36	0
GHS_indef/turon_m	18928	1.05	1.02	0	127	1.37	1.52	0
GHS_indef/d_pretok	18012	1.04	1.02	0	301	1.38	1.48	0
GHS_indef/ncvxqp3	64876	1.32	1.60	2	119	1.84	2.90	0
TSOPF/TSOPF_FS_b300_c2	41836	1.21	1.50	5	0	1.21	1.47	0
GHS_indef/ncvxqp7	270790	1.58	2.25	2	35	1.75	2.77	0
TSOPF/TSOPF_FS_b300_c3	99361	1.19	1.43	5	0	1.15	1.34	0
GHS_indef/dtoc	12391	1.79	4.44	0	15675	1.84	3.35	0
GHS_indef/bloweybl	3117	1.14	1.24	0	0	1.04	1.04	0
Marini/eurqsa	267	1.05	1.15	2	91	1.66	3.27	0
GHS_indef/aug2d	64563	11.62	222.1	0	119457	23.06	1106	0
GHS_indef/aug3d	295845	32.59	592.4	0	517235	17.85	145.4	0
Pajek/Reuters911	45745	5.37	13.74	0	846524	5.19	10.06	0
Newman/cond-mat-2003	75836	1.95	2.80	0	135449	2.32	3.45	0
Newman/cond-mat-2005	129744	1.86	2.37	0	254168	2.24	2.93	0

5.7 Matching-based ordering with restricted pivoting

In Section 3.4, we considered restricting the search for pivots to the rows of the candidate pivot columns. Experimentation found that, for our tough indefinite problems (scaled by `MC64`), iterative refinement and FGMRES were not, in general, able to recover the required accuracy. However, if combined with a matching-based ordering, the approach was successful for many of our problems; the results with and without static pivoting are reported in Table 5.13. For completeness, we also include results for the matching ordering used with static pivoting (but without restricting the pivot search). Note that when static pivoting is used there are no delays. We see that, if restricted pivoting is used, a small number of the non-singular problems failed to achieve the required accuracy. For the singular problems, incorporating static pivoting can greatly improve performance but the downside is that, if diagonal entries are perturbed, accuracy of the computed inertia is reduced.

Table 5.13: Results for the matching-based ordering used with restricted pivoting, static pivoting, and restricted pivoting plus static pivoting. The number of delays, the ratios of the actual number of entries in L and flop counts to the predicted values using METIS ordering, and the number of refinement solves are given. *nptb* is the number of perturbed diagonal entries. - indicates required accuracy not achieved. $*$ indicates FGMRES used for refinement.

Identifier	matching + restricted				matching + static				matching + restricted + static			
	ndelay	*nzL*	*nflop*	*nitr*	*nptb*	*nzL*	*nflop*	*nitr*	*nptb*	*nzL*	*nflop*	*nitr*
TSOPF/TSOPF_FS_b162_c1	0	1.24	1.52	1	0	1.24	1.52	1	0	1.24	1.52	1
TSOPF/TSOPF_FS_b39_c7	0	1.46	1.88	1	0	1.46	1.88	1	0	1.46	1.88	1
Schenk_IBMNA/c-64	0	1.25	1.70	0	0	1.25	1.70	0	0	1.25	1.70	0
GHS_indef/ncvxqp1	37	1.84	3.24	1	2	1.84	3.24	1	1	1.84	3.24	1
QY/case39	0	1.55	2.26	1	0	1.55	2.26	1	0	1.55	2.26	1
GHS_indef/boyd2	0	1.00	1.00	1	0	1.00	1.00	1	0	1.00	1.00	1
GHS_indef/stokes128	4	1.59	2.23	32$*$	5	1.59	2.23	12$*$	-	-	-	-
GHS_indef/cvxqp3	87	1.81	2.84	1	2	1.81	2.84	1	2	1.81	2.84	1
TSOPF/TSOPF_FS_b162_c3	0	1.40	1.96	1	0	1.40	1.96	1	0	1.40	1.96	1
TSOPF/TSOPF_FS_b39_c19	0	1.49	2.15	1	0	1.49	2.15	1	0	1.49	2.15	1
GHS_indef/cont-201	0	0.95	0.87	1	0	0.95	0.87	1	0	0.95	0.87	1
TSOPF/TSOPF_FS_b162_c4	0	1.21	1.46	1	0	1.21	1.46	1	0	1.21	1.46	1
GHS_indef/bratu3d	0	0.97	0.96	0	0	0.97	0.96	0	0	0.97	0.96	0
TSOPF/TSOPF_FS_b39_c30	0	1.40	1.82	1	0	1.40	1.82	1	0	1.40	1.82	1
GHS_indef/darcy003	-	-	-	-	109	1.75	3.95	2	-	-	-	-
Schenk_IBMNA/c-62	0	1.71	2.09	1	0	1.71	2.09	1	0	1.71	2.09	1
TSOPF/TSOPF_FS_b300	0	1.22	1.48	1	0	1.22	1.48	1	0	1.22	1.48	1
TSOPF/TSOPF_FS_b300_c1	0	1.24	1.53	1	0	1.24	1.53	1	0	1.24	1.53	1
GHS_indef/cont-300	0	0.95	0.87	1	0	0.95	0.87	1	0	0.95	0.87	1
GHS_indef/ncvxqp5	0	1.64	2.36	1	0	1.64	2.36	1	0	1.64	2.36	1
GHS_indef/turon_m	-	-	-	-	110	1.36	1.52	1	-	-	-	-
GHS_indef/d_pretok	-	-	-	-	275	1.38	1.48	1	-	-	-	-
GHS_indef/ncvxqp3	31	1.87	2.96	1	3	1.87	2.96	1	3	1.87	2.96	1
TSOPF/TSOPF_FS_b300_c2	0	1.21	1.47	1	0	1.21	1.47	1	0	1.21	1.47	1
GHS_indef/ncvxqp7	34	1.82	2.91	1	3	1.82	2.91	1	3	1.82	2.91	1
TSOPF/TSOPF_FS_b300_c3	0	1.15	1.34	1	0	1.15	1.34	1	0	1.15	1.34	1
GHS_indef/dtoc	48399	26.38	3759.72	0	4996	1.06	1.12	1	4996	1.06	1.12	1
GHS_indef/bloweybl	0	1.04	1.03	1	0	1.04	1.03	1	0	1.04	1.03	1
Marini/eurqsa	36	1.65	3.23	26	24	1.64	3.20	1	6	1.64	3.20	31
GHS_indef/aug2d	119457	23.06	1106.25	0	11052	1.50	2.51	10	9733	1.50	2.51	1
GHS_indef/aug3d	517235	17.85	145.36	0	14457	1.78	2.63	1	12415	1.78	2.63	1
Pajek/Reuters911	-	-	-	-	2666	2.80	3.43	1	2600	2.80	3.43	1
Newman/cond-mat-2003	-	-	-	-	1085	1.94	2.56	1	997	1.94	2.56	10
Newman/cond-mat-2005	-	-	-	-	1436	1.97	2.49	1	1285	1.97	2.49	8

5.8 Sparse direct solver timings

So far, we have concentrated on looking at how different strategies impact on the number of delayed pivots and on the factor size and flop count. In this section, we illustrate how the effect on computation time. Here we use Version 2.0.0 of the recent multifrontal solver `HSL_MA97` [26]. `HSL_MA97` is used rather than `HSL_MA77` because the latter is an out-of-core solver and as such its solve phase is comparatively expensive (the factor L has to be read in once for the forward substitution and once for the back substitution); there are also other overheads associated with the out-of-core design that result in `HSL_MA77` being slower than `HSL_MA97`. Furthermore, `HSL_MA97` is a parallel code. Table 5.14 reports complete solution times (which includes the time for scaling, ordering and refinement) for `HSL_MA97` run on 8 processors with default settings ($u = 0.01$), with and without scaling with `MC64` and for the matching-based ordering with $u = 10^{-8}$. Times are wall clock times in seconds. We see that, for some problems (notably GHS_indef/bloweybl and GHS_indef/ncvxqp5), scaling with `MC64` dramatically reduces the time. For other problems, the cost of the scaling results in an increase in the time (for example, GHS_indef/aug2d and TSOPF/TSOPF_FS_b300). Although the matching-based ordering produces denser factors, for many problems it gives the fastest time. As it also produces few delayed pivots, this would appear to be an attractive approach.

5.9 Reusing the pivot sequence

In Section 4.1, we discussed reusing the pivot sequence from the first factorization when factorizing a second matrix with the same structure and similar values. In Table 5.15, we report factorization times (in this case, the same matrix values are used on both the first and second factorizations). For some problems (such as GHS_indef/ncvxqp7) the second factorization is, as we would like, substantially faster than the first. However, for other examples (including GHS_indef/ncvxqp3), although there are few delays, the second factorization is slower than the first. The reason for this is that, when the pivot sequence from the first factorization is input to the second analyse phase, the assembly tree can be very different from that computed by the first analyse phase. In particular, we observe that the former can have significantly fewer nodes while being much deeper. As a result, many of the nodes in the second analyse have a single child and, in this case, it may be worthwhile to develop alternative node amalgamation strategies to try and improve performance.

6 Summary of findings

For many problems from a wide range of applications, if a nested dissection or minimum degree ordering is used with a sparse indefinite solver and the solver is run with its default settings, few pivots will be delayed. In this study, we have concentrated solely on the problems where the standard approach leads to a large number of delays: we have reviewed techniques designed to overcome this issue and we have employed a set of hard-to-solve symmetric indefinite systems to examine how well these techniques perform in practice. Our key findings (which are only for these tough problems) are:

- Scaling (and, in particular, the `MC64` scaling) can significantly reduce the number of delays and so is recommended but is not sufficient on its own to ensure only a few delays. All our remaining findings are based on prescaling the system using `MC64`.
- Relaxing the pivot threshold parameter has limited effect on the number of delays and, for some problems a large number of steps of FGMRES may be needed to recover accuracy.
- Static pivoting leads to no delays but a significant number of solves can be needed to recover accuracy and, in a few cases, we were not able to achieve the requested accuracy using FGMRES. Furthermore, knowledge of the inertia may be lost.
- The constrained ordering of Bridson without pivoting is not sufficiently general in that it requires the $(1,1)$ block to be definite. If this is satisfied, in our experiments the approach works with no delays,

Table 5.14: The complete solution times (in seconds) for `HSL_MA97` run with METIS ordering (with and without `MC64` scaling) and the matching ordering ($u = 0.01$ and $u = 10^{-8}$). - indicates insufficient memory to perform factorization. Bold indicates the fastest time (± 0.01 seconds).

Identifier	METIS			Matching	
	No scale	`MC64`	`MC64`	`MC64`	`MC64`
	$u = 0.01$	$u = 0.01$	$u = 10^{-8}$	$u = 0.01$	$u = 10^{-8}$
TSOPF/TSOPF_FS_b162_c1	0.34	0.34	0.41	**0.30**	**0.31**
TSOPF/TSOPF_FS_b39_c7	0.42	0.44	0.51	**0.38**	**0.38**
Schenk_IBMNA/c-64	1.93	1.43	1.44	**1.05**	**1.05**
GHS_indef/ncvxqp1	7.67	**0.64**	0.69	1.12	1.23
QY/case39	**0.64**	0.93	1.00	1.03	1.04
GHS_indef/boyd2	56.93	47.38	47.35	**37.69**	**37.68**
GHS_indef/stokes128	**0.53**	0.57	0.60	**0.54**	0.56
GHS_indef/cvxqp3	47.76	**1.59**	1.72	2.37	2.34
TSOPF/TSOPF_FS_b162_c3	1.49	**1.12**	1.30	**1.12**	**1.13**
TSOPF/TSOPF_FS_b39_c19	1.62	1.39	1.55	1.19	**1.17**
GHS_indef/cont-201	0.86	0.90	2.70	0.45	**0.43**
TSOPF/TSOPF_FS_b162_c4	2.11	1.52	1.64	**1.32**	**1.32**
GHS_indef/bratu3d	2.51	2.61	3.47	**0.73**	**0.72**
TSOPF/TSOPF_FS_b39_c30	3.18	2.44	2.68	2.04	**1.97**
GHS_indef/darcy003	3.12	3.66	3.75	**2.70**	**2.70**
Schenk_IBMNA/c-62	30.03	2.34	**2.21**	7.26	7.27
TSOPF/TSOPF_FS_b300	2.75	3.34	3.26	2.62	**2.51**
TSOPF/TSOPF_FS_b300_c1	4.15	3.10	3.53	3.01	**2.75**
GHS_indef/cont-300	2.27	2.47	6.52	**1.02**	**1.02**
GHS_indef/ncvxqp5	167.8	**3.64**	3.84	5.59	5.55
GHS_indef/turon_m	2.21	2.43	2.66	2.13	**2.11**
GHS_indef/d_pretok	2.19	2.36	2.41	2.07	**2.03**
GHS_indef/ncvxqp3	-	**16.17**	16.20	24.70	25.88
TSOPF/TSOPF_FS_b300_c2	7.81	**5.67**	6.87	6.23	6.28
GHS_indef/ncvxqp7	-	46.10	46.90	37.00	**36.78**
TSOPF/TSOPF_FS_b300_c3	12.42	**8.93**	10.37	9.62	9.63
GHS_indef/dtoc	2.11	2.73	0.85	1.82	**0.82**
Marini/eurqsa	0.62	**0.07**	**0.08**	0.09	0.09
GHS_indef/bloweybl	51.76	0.20	0.22	0.11	0.11
GHS_indef/aug2d	0.65	0.70	0.73	**0.53**	**0.54**
GHS_indef/aug3d	**11.38**	11.40	13.91	38.95	38.94
Pajek/Reuters911	25.83	39.43	**15.20**	137.7	134.8
Newman/cond-mat-2003	**6.49**	6.79	6.56	20.03	21.62
Newman/cond-mat-2005	21.42	**20.44**	20.62	67.23	67.51

Table 5.15: Factorization times (in seconds) for `HSL_MA97`. METIS ordering is used for the first factorization and the second factorization uses the actual pivot order returned by the first factorization (`MC64` scaling and the pivot threshold $u = 0.01$).

Identifier	First	Second
TSOPF/TSOPF_FS_b162_c1	0.12	0.15
TSOPF/TSOPF_FS_b39_c7	0.11	0.10
Schenk_IBMNA/c-64	0.41	0.56
GHS_indef/ncvxqp1	0.52	0.68
QY/case39	0.44	0.41
GHS_indef/boyd2	0.38	0.36
GHS_indef/stokes128	0.18	0.17
GHS_indef/cvxqp3	1.38	2.05
TSOPF/TSOPF_FS_b162_c3	0.38	0.48
TSOPF/TSOPF_FS_b39_c19	0.35	0.34
GHS_indef/cont-201	0.28	0.24
TSOPF/TSOPF_FS_b162_c4	0.52	0.52
GHS_indef/bratu3d	2.26	3.14
TSOPF/TSOPF_FS_b39_c30	0.63	0.63
GHS_indef/darcy003	0.75	0.70
Schenk_IBMNA/c-62	1.76	2.76
TSOPF/TSOPF_FS_b300	1.83	1.66
TSOPF/TSOPF_FS_b300_c1	1.58	2.15
GHS_indef/cont-300	0.96	0.89
GHS_indef/ncvxqp5	2.96	3.53
GHS_indef/turon_m	0.58	0.63
GHS_indef/d_pretok	0.57	0.63
GHS_indef/ncvxqp3	15.08	18.64
TSOPF/TSOPF_FS_b300_c2	2.45	3.55
GHS_indef/ncvxqp7	44.59	29.57
TSOPF/TSOPF_FS_b300_c3	3.87	4.64
GHS_indef/dtoc	2.60	1.99
GHS_indef/bloweybl	0.03	0.02
Marini/eurqsa	0.04	0.03
GHS_indef/aug2d	0.54	0.61
GHS_indef/aug3d	11.09	4.87
Pajek/Reuters911	39.13	50.77
Newman/cond-mat-2003	6.50	18.62
Newman/cond-mat-2005	19.98	45.41

although the fill-in of L and flop counts are generally significantly greater than for the unconstrained METIS ordering.

- The MA47 ordering leads to no delays for the OXO matrices in our test set and, for many problems, the number of delays was less than for a METIS ordering. However, for large problems, the MA47 ordering is less efficient than the METIS ordering (denser factors and higher flop counts).
- Matching-based orderings can substantially reduce the number of delays for non-singular problems. In many cases, the computed factors were sparser than if METIS was used.

Thus, if delays occur, we always recommend scaling. If delays remain a problem and the efficiency of the underlying factorization algorithm depends on there being few (if any) delays, a matching-based ordering should be used (for OXO matrices, MA47 ordering could also be considered). If the inertia is not required, incorporating static pivoting removes all delays and, when used with the matching ordering, gave the required accuracy after refinement. We emphasize, however, that an approach based on matchings is only recommended for tough problems such as those we have reported on in this study. If standard threshold partial pivoting gives no delays, scaling may be unnecessary and using a matching-based ordering is not desirable as it can be expensive to compute and result in denser factors and higher flop counts.

References

[1] P. Amestoy, T. Davis, and I. Duff, *An approximate minimum degree ordering algorithm*, SIAM J. on Matrix Analysis and Applications, 17 (1996), pp. 886–905.

[2] M. Arioli and I. S. Duff, *Using FGMRES to obtain backward stability in mixed-precision*, Electronic Transactions on Numerical Analysis, 33 (2009), pp. 31–44.

[3] M. Arioli, I. S. Duff, S. Gratton, and S. Pralet, *A note on GMRES preconditioned by a perturbed LDL^T decomposition with static pivoting*, SIAM J. on Scientific Computing, 29 (2007), pp. 2024–2044.

[4] C. Ashcraft, R. G. Grimes, and J. G. Lewis, *Accurate symmetric indefinite linear equation solvers*, SIAM J. on Matrix Analysis and Applications, 20 (1999), pp. 513–561.

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

[6] J. R. Bunch and L. Kaufman, *Some stable methods for calculating inertia and solving symmetric linear systems*, Mathematics of Computation, 31 (1977), pp. 1634–179.

[7] T. Davis and Y. Hu, *The University of Florida Sparse Matrix Collection*, ACM Transactions on Mathematical Software, 38 (2011). Article 1, 25 pages.

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

[9] I. Duff and J. Gilbert, *Maximum-weighted matchingand block pivoting for symmetric indefinite systems.* in Abstract book of Householder Symposium XV, June 17-21, 2002, pp.73–75.

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

[11] I. Duff and S. Pralet, *Strategies for scaling and pivoting for sparse symmetric indefinite problems*, SIAM J. on Matrix Analysis and Applications, 27 (2005), pp. 313 – 340.

[12] ——, *Towards a stable mixed pivoting strategy for the sequential and parallel solution of sparse symmetric indefinite systems*, SIAM J. on Matrix Analysis and Applications, 29 (2007), pp. 1007–1024.

[13] I. Duff and J. Reid, *The multifrontal solution of indefinite sparse symmetric linear systems*, ACM Transactions on Mathematical Software, 9 (1983), pp. 302–325.

[14] ——, *MA27: A set of fortran subroutines for solving sparse symmetric sets of linear equations*, Technical Report Report R-10533, Computer Science and Systems Division, AERE, 1992.

[15] ——, *MA47,: a Fortran code for direct solution of indefinite sparse symmetric linear systems*, Technical Report RAL-95-001, Rutherford Appleton Laboratory, 1995.

[16] I. S. Duff, *MA57– a new code for the solution of sparse symmetric definite and indefinite systems*, ACM Transactions on Mathematical Software, 30 (2004), pp. 118–154.

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

[18] A. George, *Nested dissection of a regular finite-element mesh*, SIAM J. on Numerical Analysis, 10 (1973), pp. 345–363.

[19] A. Gupta, *WSMP: Watson sparse matrix package (Part-I: Direct solution of symmetric sparse systems)*, Tech. Rep. RC 21886, IBM T. J. Watson Research Center, Yorktown Heights, NY, November 2000. `http://www.cs.umn.edu/~agupta/wsmp`.

[20] M. Hagemann and O. Schenk, *Weighted matchings for preconditioning symmetric indefinite linear systems*, SIAM J. on Scientific Computing, 28 (2006), pp. 403–420.

[21] J. Hogg, J. Reid, and J. Scott, *Design of a multicore sparse Cholesky factorization using DAGs*, SIAM J. on Scientific Computing, 32 (2010), pp. 3627–3649.

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

[23] ——, *A fast and robust mixed precision solver for the solution of sparse symmetric linear systems*, ACM Transactions on Mathematical Software, 37 (2010). Article 17, 24 pages.

[24] ——, *An indefinite sparse direct solver for large problems on multicore machines*, Technical Report RAL-TR-2010-011, Rutherford Appleton Laboratory, 2010.

[25] ——, *A note on the solve phase of a multicore solver*, Technical Report RAL-TR-2010-007, Rutherford Appleton Laboratory, 2010.

[26] ——, *HSL_MA97: a bit-compatible multifrontal code for sparse symmetric systems*, Technical Report RAL-TR-2011-024, Rutherford Appleton Laboratory, 2011.

[27] ——, *Finding weighted matchings for rank-deficient sparse symmetric matrices*, Technical Report RAL-P-2012-003, Rutherford Appleton Laboratory, 2012.

[28] ——, *New parallel sparse direct solvers for engineering applications*, Technical Report RAL-P-2012-001, Rutherford Appleton Laboratory, 2012.

[29] HSL, *A collection of Fortran codes for large-scale scientific computation*, 2011. `http://www.hsl.rl.ac.uk/`.

[30] G. Karypis and V. Kumar, *METIS: A software package for partitioning unstructured graphs, partitioning meshes and computing fill-reducing orderings of sparse matrices - version 4.0*, 1998. `http://www-users.cs.umn.edu/~karypis/metis/`.

[31] X. S. Li and J. W. Demmel, *Making sparse Gaussian elimination scalable by static pivoting*, in Proceedings of the 1998 ACM/IEEE conference on Supercomputing, IEEE Computer Society, 1998, pp. 1–17.

[32] ——, *SuperLU DIST: A scalable distributed-memory sparse direct solver or unsymmetric linear systems*, ACM Transactions on Mathematical Software, 29 (2003), pp. 110–140.

[33] J. Liu, *Modification of the minimum-degree algorithm by multiple elimination*, ACM Transactions on Mathematical Software, 11 (1985), pp. 141–153.

[34] MUMPS, *MUMPS: a multifrontal massively parallel sparse direct solver*, 2011. `http://graal.ens-lyon.fr/MUMPS/`.

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

[36] ——, *An out-of-core sparse Cholesky solver*, ACM Transactions on Mathematical Software, 36 (2009). Article 9, 33 pages.

[37] ——, *Partial factorization of a dense symmetric indefinite matrix*, ACM Transactions on Mathematical Software, 38 (2011). Article 10, 19 pages.

[38] D. Ruiz, *A scaling algorithm to equilibrate both rows and columns norms in matrices*, Technical Report RAL-TR-2001-034, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2001.

[39] D. Ruiz and B. Uçar, *A symmetry preserving algorithm of matrix scaling*, Technical Report RR-7552, INRIA, 2011.

[40] Y. Saad, *A flexible inner-outer preconditioned GMRES algorithm*, SIAM J. on Scientific Computing, 14 (1993), pp. 461–469.

[41] O. Schenk and K. Gärtner, *Solving unsymmetric sparse systems of linear equations with PARDISO*, Journal of Future Generation Computer Systems, 20 (2004), pp. 475–487.

[42] ——, *On fast factorization pivoting methods for symmetric indefinite systems*, Elec. Trans. Numer. Anal, 23 (2006), pp. 158–179.

[43] O. Schenk, A. Wächter, and M. Hagemann, *Matching-based preprocessing algorithms to the solution of saddle-point problems in saddle-point problems in large-scale non-convex interior-point optimization*, Comput. Optim. Appl., 36 (2007), pp. 321–341.

[44] J. Scott, *A note on a simple constrained ordering for saddle-point systems*, Technical Report RAL-TR-2009-007, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2009.

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

[46] W. Tinney and J. Walker, *Direct solutions of sparse network equations by optimally ordered triangular factorization*, Proc. IEEE, 55 (1967), pp. 1801–1809.

[47] A. Wächter and L. T. Biegler, *On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming*, Mathematical Programming, 106(1) (2006). 25–57.