Technical Report
RAL-TR-2005-007

Towards a stable static pivoting strategy for the sequential and parallel solution of sparse symmetric indefinite systems

Iain S. Duff and Stéphane Pralet

April 22, 2005

Council for the Central Laboratory of the Research Councils

ISSN 1358-6254

RAL-TR-2005-007

Towards a stable static pivoting strategy for the sequential and parallel solution of sparse symmetric indefinite systems[1]

Iain S. Duff[2] and Stéphane Pralet[3]

ABSTRACT
We consider the direct solution of sparse symmetric indefinite matrices. We develop new pivoting strategies that combine numerical and static pivoting. Furthermore, we propose original approaches that are designed for parallel distributed factorization. We show that our pivoting strategies are numerically robust and that the factorization is significantly faster because of this static/numerical combination. A key point of our parallel implementation is the cheap and reliable estimation of the growth factor. This estimation is based on an approximation of the off-diagonal entries and does not require any supplementary messages.

Keywords: sparse indefinite systems, augmented systems, direct solver for sparse symmetric indefinite matrices,

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

---

[1]Current reports available by anonymous ftp to `ftp.numerical.rl.ac.uk` in directory pub/reports. This report is in file duprRAL2005007.ps.gz. The report also available through the URL `www.numerical.rl.ac.uk/reports/reports.html`. It is also available as the CERFACS Technical Report TR/PA/05/26 and IRIT-ENSEEIHT Technical Report RT/TLSE/05/08.
[2]i.s.duff@rl.ac.uk. This work was supported by the EPSRC Grant GR/S42170.
[3]Stephane.Pralet@enseeiht.fr. CNRS-IRIT, ENSEEIHT, 2 rue Charles Camichel, BP 7122 - F 31071 Toulouse Cedex 7, France.

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

April 22, 2005

Contents

1 Introduction

We study pivoting strategies for computing the LDL^T factorization of a symmetric indefinite matrix where L is a lower triangular matrix and D is a block diagonal matrix with 1×1 and 2×2 blocks. We consider direct methods based on a multifrontal technique although most of our comments and analysis apply to other approaches for direct factorization.

Usually the factorization is computed in two phases. The analysis phase preprocesses the system of equations and is often based purely on matrix structure. The second phase performs the Gaussian elimination. If the numerical tests prevent the selection of some pivots chosen by the analysis, then the factorization can still proceed but there will normally be an increase in both storage and work for the factorization above that required if no pivots are delayed. This effect can be particularly significant on augmented systems. A static pivoting scheme follows closely the pivot selection of the analysis to give generally lower fill-in and factorization times at the potential cost of worse accuracy in the factorization.

The originality of our approach is to mix static and numerical pivoting and to propose a criterion to decide between small 1×1 and small 2×2 pivots. We find that delaying pivots is dangerous in the context of static pivoting. We also propose new pivoting strategies that are numerically robust on our representative test set and do not limit the scalability of parallel distributed factorization. They are based on estimations of growth factors and do not require any supplementary messages.

In Section 2, we give a brief presentation of a multifrontal solver. We also describe an existing static pivoting strategy in the context of LU factorization and the numerical pivoting strategies that are often used in the context of sparse direct methods. Section 3 presents our experimental environment. In Section 4, we describe a new pivoting strategy that combines numerical and static pivoting and is well designed for sequential factorizations. We present experimental results for this strategy. In Section 5, we show that combining static pivoting and delayed pivots severely affects the numerical quality of the factorization. Section 6 presents pivoting strategies that are particularly suited for parallel distributed solvers. In Section 7, we study the influence of the preprocessings of Duff and Pralet (2004) on our static pivoting strategies (both sequential and parallel approaches).

We will use our pivoting strategies with a symmetric multifrontal code, `MA57` Version 3.0.0 (Duff 2004, HSL 2004), on a challenging test set (see Section 3). When nothing is mentioned we force the METIS ordering during the analysis. For the other control parameters we use the default options, in particular we symmetrically scale the matrix (see Duff and Pralet, 2004). Although `MA57` is a sequential code, we will use it to simulate the parallel behaviour of a multifrontal solver like `MUMPS` (Amestoy, Duff, Koster and L'Excellent 2001, Amestoy, Duff and L'Excellent 2000).

1.1 Notation

In the following, s will be the function for the sign of a real number:

$$s(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases},$$

ϵ will denote the machine precision, $|| \; ||_2$, $|| \; ||_\infty$ will denote the sub-multiplicative matrix norms, and $||A||_M$ will denote the norm $\max_{ij} |a_{ij}|$.

For each matrix or submatrix $A = (a_{ij})$, $|A| = (|a_{ij}|)$ and n will denote the order of A. $0 < u \leq 1$ and $\mu = \sqrt{\epsilon}$ will denote real numbers which will be used as thresholds in our pivoting strategies. In practice, we will use $u = 0.01$.

We summarize in Table 1.1 the main pivoting strategies that we develop in this paper.

Name	Pivoting strategy	Section
`numSEQ`	Numerical pivoting of Duff-Reid algorithm	2.2
`mixSEQ`	Numerical pivoting combined with static pivoting	4.1
`numBPAR`	Basic restriction of Duff-Reid algorithm	6.2
`numEPAR`	Adaptation of Duff-Reid algorithm that uses estimations	6.3
`mixPAR`	Combination of `numEPAR` and `mixSEQ`	4.1 and 6.3

Table 1.1: Summary of our main pivoting strategies. The `SEQ` suffix means that the strategy is designed for a sequential code. The `PAR` suffix means that the strategy is designed for a parallel code.

2 Symmetric indefinite multifrontal solvers and numerical pivoting

2.1 Multifrontal approach

For an irreducible matrix, the *elimination tree* (Duff and Reid 1983, Liu 1990) represents the order in which the matrix can be factorized, that is, in which the unknowns from the underlying linear system of equations can be eliminated. Generally, the tree yields only a *partial* ordering which allows some freedom for the order in which pivots can be eliminated. (This tree is in the most general case a forest, but we will assume in our discussions, for the sake of clarity, that it is a connected tree. That is the matrix is irreducible).

One central concept of the multifrontal approach (Duff and Reid 1983) is to group (or *amalgamate*) columns with the same sparsity structure to create *supervariables* or *supernodes* (Duff and Reid 1983, Liu, Ng and Peyton 1993) in order to make use of efficient dense matrix kernels. The amalgamated elimination tree is called the *assembly tree.*

The notion of child nodes that send their contribution blocks to their parents leads to the following interpretation of the factorization process. When a node in the assembly tree is being processed, it assembles the contribution blocks from all its child nodes into its *frontal matrix* (see Figure 2.1). Frontal matrices are always considered as dense matrices and we can make use of efficient BLAS kernels and avoid indirect addressing, see for example, Dongarra, Duff, Sorensen and van der Vorst (1998). Afterwards, the pivotal

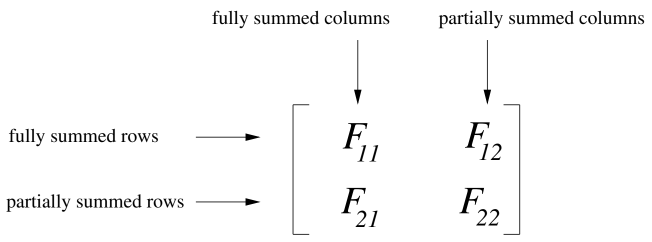


Figure 2.1: A frontal matrix.

variables from the *fully summed* block are eliminated and the Schur complement matrix, $F_{22}-F_{21}F_{11}^{-1}F_{12}$, computed. We call this Schur complement matrix the *contribution block* of the node. The contribution block is then sent to the parent node to be assembled. If some variables are not eliminated because of numerical issues, they are moved to the contribution block F_{22} and sent to the parent node. The effect of this is that the computation $F'_{22} - F'_{21}{F'_{11}}^{-1}F'_{12}$ is performed where F'_{11} is a submatrix of F_{11} of dimension the number of pivots selected at this stage. If this dimension is less than the order of F_{11} then the difference represents the number of pivots *delayed* at this stage.

In the symmetric case, $F_{12} = F_{21}^T$, the matrices F_{11} and F_{22} are symmetric, pivots are chosen from the diagonal as discussed in the following section, and operations and storage are about half that of the general case.

2.2 Numerical pivoting

In the unsymmetric case, at step k of Gaussian elimination, the pivot (p, q) is selected from the fully summed rows and columns and the entries a_{ij} of the remaining submatrix are updated:

$$a_{ij}^{(k+1)} \leftarrow a_{ij}^{(k)} - \frac{a_{ip}^{(k)} a_{qj}^{(k)}}{a_{pq}^{(k)}}.$$

To limit the growth of the entries in the factors and thus to obtain a more accurate factorization, a test on the magnitude of the pivot is commonly used. a_{pq} can be selected if and only if

$$|a_{pq}| \geq u \max_j |a_{pj}| \tag{2.1}$$

where u is a threshold parameter between 0 and 1. This criterion will ensure that the growth factor is limited to $1 + 1/u$.

In the symmetric indefinite case, we have to perform 1×1 and 2×2 pivoting if we want to keep the symmetry while maintaining stability. Pivot selection can be done using the Bunch-Parlett (Bunch and Parlett 1971) or Bunch-Kaufman (Bunch and Kaufman 1977) algorithm or a variation proposed by Ashcraft, Grimes and Lewis (1998) that uses rook pivoting. In the context of sparse matrices, the criterion of the Duff-Reid algorithm (Duff and Reid 1983), as modified by Duff and Reid (1996)) can be used to ensure a growth factor lower than $1 + 1/u$ at each step of Gaussian elimination. A 1×1 diagonal pivot can be selected if and only if it satisfies the inequality (2.1). A 2×2 pivot $P = \begin{pmatrix} a_{pp} & a_{pq} \\ a_{qp} & a_{qq} \end{pmatrix}$ can be selected if and only if it satisfies:

$$|P^{-1}| \begin{pmatrix} \max_{k\neq p,q} |a_{pk}| \\ \max_{k\neq p,q} |a_{qk}| \end{pmatrix} \leq \begin{pmatrix} 1/u \\ 1/u \end{pmatrix} \tag{2.2}$$

where u is a threshold between 0 and $\frac{1}{2}$ (the limit $\frac{1}{2}$ is needed to be sure that a pivot can always be chosen when all the frontal matrix is fully summed). During the factorization, it may be impossible to choose some fully summed variables as pivots because of the threshold tests. Elimination of these variables must then be delayed to the parent. This has the effect of causing extra fill-in and thus increases the memory and the number of operations. Too many delayed pivots can severely slow down the factorization and could even prevent the factorization because there is insufficient memory to accommodate the increased fill-in.

2.3 Static pivoting

By a static code we mean a code in which the factorization respects in some sense the ordering of the analysis. The factorization does not necessarily follow the analysis exactly and some slight variations are allowed. For example in a multifrontal context, it is sufficient that the factorization decisions are compatible with the assembly tree (numerical pivoting can be performed within a front). Here the analysis predicts exactly the memory needed and the number of operations of the factorization because it selects pivots from the fully summed variables for each node of the elimination tree and it does not postpone pivots. A static approach was proposed by Li and Demmel (1998) in the context of LU factorization. During Gaussian elimination *"small perturbations"* are added to limit the growth of the factors in order to enhance the backward stability of the algorithm (see, for example, Theorem 11.4 of Higham (2002), that gives an upper bound of the residual after one iterative refinement). By *"small perturbations"* we mean that the perturbed matrix is close enough to the original so that the factorization of the perturbed matrix is close to the factorization of the original matrix. If this is the case, then we would hope to have a cheap and sufficiently accurate solution using the computed factors. In the framework of our study, we want a static approach with the following features:

(F1) It is easy to decide whether to add a perturbation or not. In particular, in a parallel symmetric indefinite solver, this decision must not involve any limiting extra communication cost or synchronization.

(F2) The perturbations are restricted to the block of fully summed rows/columns in each front.

Given a symmetric matrix A, not necessarily positive definite, approaches to compute a modified Cholesky factorization of $A + E$ with E as small as possible (in a modified Cholesky $||E||_2 \geq -\min \lambda_i(A)$) have been developed (Cheng and Higham 1998, Eskow and Schnabel 1991, Gill and Murray 1974). The size of this perturbation is larger than we want so we do not use modified Cholesky approaches. Furthermore, the approach of Cheng and Higham (1998) is not adapted for sparse matrices because the pattern of E can be significantly different from the pattern of A.

Modifying the diagonal of the matrix instead of performing numerical pivoting was introduced by Stewart (1974). In his approach, the magnitude of the perturbations can be quite large. In the context of *LU* factorization with static pivoting, `SuperLU_DIST` (Li and Demmel 2003) adds small perturbations δ to the diagonal entries when the pivot a_{ii} is too small and the inequality (2.1) is transformed into the constraint

$$(\mathcal{C}_{1\times1}) \ : \ |a_{ii} + \delta| \geq \mu \ ||A||_M$$

At each step of Gaussian elimination, one has to solve the problem

$$(\mathcal{P}_{1\times1}) \begin{cases} \min |\delta| \\ w.r.t. \ (\mathcal{C}_{1\times1}) \end{cases}$$

Property 2.1 *A solution of* $(\mathcal{P}_{1\times1})$ *is given by:*

$$\delta = s(a_{ii}) \max(\tau - |a_{ii}|, 0)$$

where s is the sign function and $\tau = \mu||A||_M$.

`SuperLU_DIST` performs static pivoting using Property 2.1. Note that no pivoting is performed within the supernodes.

More recently Schenk and Gärtner (2004) have combined static pivoting approaches and Bunch-Kaufman pivoting strategies. Nevertheless their approach may significantly degrade the precision of the solution and slow down the solution phase because of the increased number of iterative refinement steps.

3 Experimental environment

Our experiments are conducted on one node of a COMPAQ Alpha Server SC45 at CERFACS. There are 4 GBytes of memory shared between 4 EV68 processors per node and we disable three of the processors so that we can use all the memory of the node with the remaining single processor. We use the Fortran 90 compiler, f90 version 5.5 with the -O option. If the factorization needs more than 4 GBytes or requires more than 30 minutes CPU time, we consider that it is not successful.

We conduct our experiments on a number of challenging and sometimes badly conditioned test problems. The matrices are available from `ftp.numerical.rl.ac.uk/pub/matrices/symmetric/indef/` and most of them (all except the cvxqp3 matrix) are a subset of the matrices collected by Gould and Scott (2004) for testing symmetric sparse solvers. To select our matrices we ran our pivoting strategies on the matrices from Gould and Scott (2004) and kept the difficult matrices that illustrate the characteristics of our pivoting strategies. (We use “difficult” in the sense that they often required iterative refinement steps to get a small residual.)

Some of the matrices come from the Maros and Meszanos quadratic programming collection (M2) (Maros and Meszaros 1999), and the CUTEr optimization test set (CUTEr) (Gould, Orban and Toint 2002). Some problems were generated by Andy Wathen (AW), Mario Arioli (MA) and Miroslav Tuma (MT). These problems are described in Table 3.2. These test matrices correspond to augmented matrices of the form

$$\mathcal{K}_{H,A} = \begin{pmatrix} H & A \\ A^T & 0 \end{pmatrix}.$$

Matrix	n	nnz	$\lambda+$	$\lambda-$	Origin
BRAINPC2	27607	96601	13807	13800	Biological model (CUTEr)
BRATU3D	27792	88627	15625	12167	3D Bratu problem on the unit cube (CUTEr)
CONT-201	80595	239596	40397	40198	KKT matrix–Convex QP (M2)
CONT-300	180895	562496	90597	90298	KKT matrix–Convex QP (M2)
cvxqp3	17500	62481	10000	7500	Convex QP (CUTEr)
DTOC	24993	34986	9997	9997	Discrete-time optimal control (CUTEr)
mario001	38434	114643	23130	15304	Stokes equation (MA)
NCVXQP1	12111	40537	7111	5000	KKT matrix–nonconvex QP (CUTEr)
NCVXQP5	62500	237483	28534	33966	KKT matrix–nonconvex QP (CUTEr)
NCVXQP7	87500	312481	37500	50000	KKT matrix–nonconvex QP (CUTEr)
SIT100	10262	34094	7143	3119	Straz pod Ralskem mine model (MT)
stokes128	49666	295938	33281	16385	Stokes equation (MA)
stokes64	12546	74242	8449	4097	Stokes equation (AW)

Table 3.2: Symmetric indefinite matrices. $\lambda+$: number of positive eigenvalues. $\lambda-$: number of negative eigenvalues.

To perform an error analysis of the solution we compute the sparse component-wise backward error using the theory and measure developed by Arioli, Demmel and Duff (1989). The scaled residual of the i^{th} equation is

$$\Delta_i = \frac{|r_i|}{(|A||x| + |b|)_i},$$

where $r = b - Ax$ and x is the computed solution, except if the denominator is too small (in our code the threshold for this is $1000 \times \epsilon$). In this case, we use

$$\Delta_i = \frac{|r_i|}{((|A||x|)_i + ||A_i||_\infty ||x||_\infty)_i},$$

where A_i represents the i^{th} row of A. We apply iterative refinement in all our approaches. At each step k of the iterative refinement, we compute the current backward error

$berr^{(k)} = \max_i \Delta_i$. We stop if $berr^{(k)} < 10^{-15}$ or if $berr^{(k)} > 0.9 \times berr^{(k-1)}$ (the convergence rate is too slow) or $k = 20$ (the maximum number of iterations has been reached). We choose a minimum convergence rate of 0.9 to have a complete set of results at each iteration. Note that this convergence rate will not change our discussion and that our approaches are also valid with usual convergence rates (for example 0.5). We could have decided to use other iterative methods (for example MINRES) but this is out of the scope of this paper.

4 Mixing numerical pivoting and static pivoting

4.1 Algorithm

In this section, we present an approach which combines numerical checking for stability with 1×1 static pivoting. We decided to use only 1×1 perturbations because they are easier to implement and we want to clearly identify the impact of this mixed approach. Some promising experiments with 2×2 perturbations can be found in Pralet (2004), but they were applied in a different context; the analysis fixes the 1×1 and 2×2 pivots and appropriate perturbations were applied during the factorization if necessary. In this present paper, the pivots are dynamically chosen and we have found that the use of 2×2 perturbations does not give a significant improvement. In our algorithm, we perturb the original entries only in extreme cases where pivots are very small in magnitude. That is why the use of 2×2 perturbations as well as 1×1 perturbations does not affect the precision of the solution. We will again mention 2×2 perturbations in our conclusions and suggest that they can be used in a different context, in particular if we allow large perturbations.

Let us consider a frontal matrix from the elimination tree. It contains two kinds of variables, the fully summed variables (FSV) which correspond to the pivot block that we want to eliminate and the partially summed variables (PSV) on which the Schur complement will be computed.

Our mixed approach is based on two phases. In the first phase, we perform numerical pivoting in the block of fully summed variables until no remaining variables satisfy the numerical criterion. In the second phase, we eliminate the remaining fully summed variables adding 1×1 perturbations if necessary.

Moreover, we can relax the threshold tests of Section 2.2. Instead of using the inequalities (2.1) and (2.2) for our stability criteria, we only consider entries in a subset K, where $FSV \subset K \subset FSV \cup PSV$. In Section 4.2 we perform experiments with $K = FSV \cup PSV$ and we use $K = FSV$ in Section 6. since this will allow us to design a scalable approach for parallel computation.

Our new tests are more precisely defined as follows. First we define

$$g_1(i) = \frac{\max_{k\in K\setminus\{i\}} |a_{ik}|}{|a_{ii}|}, \tag{4.3}$$

and

$$g_2(i,j) = \left\| |P^{-1}| \begin{pmatrix} \max_{k\in K\setminus\{i,j\}} |a_{ik}| \\ \max_{k\in K\setminus\{i,j\}} |a_{jk}| \end{pmatrix} \right\|_\infty . \tag{4.4}$$

During the first phase, a 1×1 pivot a_{ii} is considered to be stable if and only if

$$g_1(i) \le 1/u, \tag{4.5}$$

and a 2×2 pivot P is considered to be stable if and only if

$$g_2(i,j) \le 1/u. \tag{4.6}$$

Algorithm 1 summarizes our static pivoting strategy.

Algorithm 1 Relaxed numerical pivot selection combined with static pivoting

```
Phase 1: Eliminate as many $1\times 1$ and $2\times 2$ pivots as possible which satisfy inequalities (4.5) and (4.6)
respectively using the Duff-Reid algorithm with threshold $u$.
Phase 2:
while $FSV \neq \emptyset$ do
  Let $i \in FSV$.
  if $\#FSV = 1$ then
    if $|a_{ii}| < \mu\ ||A||_M$ then $a_{ii} = s(a_{ii})\mu\ ||A||_M$
    Perform elimination using $i$ as a $1\times 1$ pivot.
    return
  end if
  Let $j = arg\max_{k\in FSV\setminus\{i\}} |a_{ik}|$ and $P$ be the $2\times 2$ block associated with $i$ and $j$.
  Choose between $1\times 1$ or $2\times 2$ pivoting:
  if $\min\{g_1(i), g_2(i,j)\} < 1/\mu$ then /* Case 1 */
    if $g_2 < g_1$ then
      Perform elimination using $(i,j)$ as a $2\times 2$ pivot.
    else
      Perform elimination using $i$ as a $1\times 1$ pivot.
    end if
  else if $\min\{1/|a_{ii}|, ||P^{-1}||_\infty\} < 1/(\mu\ ||A||_M)$ then /* Case 2 */
    if $1/|a_{ii}| > ||P^{-1}||_\infty$ then
      Perform elimination using $(i,j)$ as a $2\times 2$ pivot.
    else
      Perform elimination using $i$ as a $1\times 1$ pivot.
    end if
  else /* Case 3 */
    $a_{ii} = s(a_{ii})\mu\ ||A||_M$
    Perform elimination using $i$ as a $1\times 1$ pivot.
  end if
end while
```

During the second phase we use the threshold $\mu = \sqrt{\epsilon}$. We tried different thresholds in Pralet (2004) and compared the precision of the solution after applying iterative refinement. We remark that choosing $10^{-7} \le \mu \le 10^{-10}$ seems to be a good compromise between small perturbations and small growth factors (larger values would give a more stable factorization). The different values of μ lead to similar behaviour in terms of number of iterations and precision of the solution. The static phase perturbs the diagonal of the matrix if the pivot is too small with respect to the initial values in A (smaller than $\mu\ ||A||_M$).

The choice between a 1×1 and a 2×2 pivot is done in three stages. Firstly (Case 1 of Algorithm 1), if we can eliminate a pivot and ensure a growth factor lower than $1 + 1/\mu$ then we select the one with the lower growth factor. Secondly (Case 2 of Algorithm 1), if we cannot ensure a growth factor lower than $1 + 1/\mu$ then we compare the quantities $1/|a_{ii}|$ and $||P^{-1}||_\infty$. This second comparison is guided by the growth factor that would appear if we suppose that the largest off-diagonal entry is bounded by $||A||_M$. Finally, if no pivot can be chosen, a perturbed 1×1 pivot is selected (Case 3 of Algorithm 1).

We also tried to smooth the transition between the first and the second phase. Before the second phase we performed numerical pivoting with smaller values of u. More precisely we defined a parameter u_{min}. While u is larger than u_{min}, we decrease the u value (for example $u = u/10$) and restart phase 1. We did not observe gains from such an approach and thus prefer to focus on the simple version of our pivoting strategies (Algorithm 1) and to keep u fixed at 0.01.

4.2 Experimental results

In this section we discuss the influence of static pivoting with $K = FSV \cup PSV$. Approaches with $K = FSV$ will be discussed in Section 6.

The $K = FSV \cup PSV$ approach will be referred to as the `mixSEQ` algorithm because checking the stability over the partially summed rows is not well designed for existing parallel implementations (it involves a bottleneck in terms of communications or synchronization). Hence this approach does not fully agree with the feature (F1) presented in Section 2.3.

In the rest of the paper, `numSEQ` will refer to a minor modification of the numerical pivoting strategy of Duff-Reid (Duff and Reid 1983). The only difference between our `numSEQ` strategy and the Duff-Reid strategy is that, when we detect that all fully summed rows are numerically zero ($< 10^{-20} \times ||A||_M$), we perturb the corresponding diagonal entry and eliminate it. It does not change the rest of the factorization too much because the magnitude of the updates is bounded by $(10^{-20} \times ||A||_M)^2/(\mu \times ||A||_M) \approx 10^{-32}||A||_M$.

When we perturb a 1×1 pivot (Case 3 of Algorithm 1 and the above exception in the Duff-Reid algorithm) this pivot is called a *tiny pivot*. Note we did not get any tiny pivots with the `numSEQ` strategy except on the DTOC matrix. It is structurally singular and `numSEQ` applies 4999 perturbations which corresponds exactly to the dimension of the null space.

Table 4.3 compares the precision of the solution for an LDL^T factorization with numerical pivoting and an LDL^T factorization with the `mixSEQ` pivoting strategy. Thanks to numerical pivoting, no iterative refinement is needed for a backward error smaller than $\sqrt{\epsilon}$, whereas with `mixSEQ` the backward error without any iterative refinement is often larger than $\sqrt{\epsilon}$. Thus we advise performing one or two steps of iterative refinement when

Matrix	`numSEQ` pivoting strategy		`mixSEQ` pivoting strategy			
	it. 0	it. 1	it. 0	it. 1	it. 2	tiny
BRAINPC2	1.6e-15	1.0e-15	2.1e-08	5.7e-15	9.8e-16	12932
BRATU3D	2.0e-09	1.7e-16	9.2e-06	2.2e-11	1.7e-16	8429
CONT-201	8.8e-11	1.6e-16	1.0e-05	9.4e-09	4.9e-09	27470
CONT-300	7.6e-11	1.9e-16	2.1e-05	2.7e-09	2.5e-09	67864
cvxqp3	5.2e-11	2.7e-16	8.5e-06	1.2e-12	3.4e-16	6277
DTOC	2.1e-16	2.7e-20	8.3e-07	2.1e-13	1.9e-15	9790
mario001	6.3e-15	1.3e-16	3.1e-08	2.5e-13	1.3e-16	10305
NCVXQP1	4.6e-14	1.7e-17	4.9e-13	3.2e-15	2.6e-17	3619
NCVXQP5	2.0e-11	2.0e-16	2.0e-08	6.7e-11	2.7e-14	8402
NCVXQP7	9.6e-10	2.2e-16	4.9e-06	1.4e-12	2.2e-16	31043
SIT100	4.4e-15	1.4e-16	2.0e-08	5.8e-15	1.5e-16	1388
stokes128	1.1e-14	5.5e-16	4.2e-14	2.0e-15	1.7e-15	12738
stokes64	4.3e-15	1.5e-15	1.6e-13	2.3e-14	2.2e-14	3106

Table 4.3: Component-wise backward error and number of tiny pivots.

using the `mixSEQ` strategy. This slight degradation of the precision is due to the tiny pivots. On average, the `mixSEQ` strategy needs one iteration more to get the same precision as the `numSEQ` strategy.

Although the number of tiny pivots can be very large (see last column of Table 4.3), our `mixSEQ` approach generally succeeds in reducing the backward error. The CONT-* matrices are the only ones for which iterative refinement does not converge to the machine precision, even with several iterations.

Table 4.4 shows the main advantage of using static pivoting: the `mixSEQ` factorization is always faster. Delaying pivots increases the number of operations and thus tends to slow down the factorization phase. Static pivoting decreases the size of the factors and thus decreases the time for backward and forward substitution. Nevertheless the solution phase is often more costly with the `mixSEQ` strategy because it requires more iterative refinement steps and thus more backward and forward substitutions and matrix-vector multiplications.

Note that on the DTOC matrix `numSEQ` generates factors that have 25 times more entries. This matrix is structurally singular and so the numerical pivoting approach has to store large dense blocks (many of whose entries are zero) until the factorization detects the singularity (it has to postpone 29478 variables). It also explains why we observe a maximum front size of 2526 with `numSEQ` but only 21 with the `mixSEQ` strategy.

5 Combination of static pivoting and delayed pivots

A quite natural idea to improve the precision of the solution is to allow some delayed pivots up to a certain predetermined limit. This criterion could be based on the memory increase or on the increase in the size of the factors or on the increase in the number of operations or on the increase in the volume of communications if the factorization is performed in a

Matrix	Number delayed numSEQ	Factorization time numSEQ	Factorization time mixSEQ	Time for forward and backward substitution numSEQ	Time for forward and backward substitution mixSEQ	MV time	Size of the factors numSEQ	Size of the factors mixSEQ
BRAINPC2	14267	0.18	0.11	0.018	0.014	0.003	656765	322971
BRATU3D	90052	34.2	9.24	0.255	0.125	0.003	11484379	5569194
CONT-201	71296	5.51	1.94*	0.195	0.127*	0.008	8820367	4304559
CONT-300	183306	21.1	6.08*	0.547	0.306*	0.033	23838606	10714425
cvxqp3	30519	9.73	3.08	0.099	0.048	0.002	4740141	2301836
DTOC	29478	29.1	0.41	0.064	0.006	0.001	4714248	187639
mario001	15463	0.28	0.23	0.024	0.022	0.008	817056	575373
NCVXQP1	12463	2.69	1.29	0.039	0.024	0.001	2235743	1327920
NCVXQP5	16703	25.7	23.0	0.326	0.279	0.015	13365963	11205204
NCVXQP7	195973	195.	71.6	0.874	0.498	0.021	37683838	19367210
SIT100	2710	0.13	0.11	0.007	0.004	0.001	483383	417147
stokes128	18056	1.14	1.06	0.096	0.076	0.016	3437116	2753749
stokes64	4292	0.33	0.29	0.012	0.013	0.002	736428	577581

Table 4.4: Factorization and solution time (in seconds), number of delayed pivots and size of the factors. `numSEQ`: Duff-Reid pivoting strategy. `mixSEQ`: combination of static and numerical pivoting. * means that the `mixSEQ` numerical quality is not similar to the `numSEQ` quality. Number delayed: number of delayed pivots. MV time: matrix-vector multiplication time (in seconds).

distributed memory environment. Algorithm 2 presents an example of combining delayed pivots and static pivoting where we use a very simple criterion to limit the number of delayed pivots: we do not delay more than $\alpha \times n$ pivots where α is a parameter.

Algorithm 2 Combination of static pivoting and delayed pivots

```
INPUT:
  $\alpha$: control on number of delayed pivots

  Let $d = \alpha \times n$ be the number of delayed pivots allowed.
  for each node of the assembly tree do
    if less than $d$ variables have already been delayed then
      use Duff-Reid algorithm with threshold $u$.
    else
      use Algorithm 1.
    end if
  end for
```

Table 5.5 shows that combining delayed pivots and static pivoting may be dangerous. This combination often requires more steps of iterative refinement to obtain a small residual. For example with $\alpha = 10$, `MA57` requires 18 iterative refinement steps to get a component-wise backward error smaller than $\sqrt{\epsilon}$ on CONT-201 and obtains a backward error equal to 4.7×10^{-8} after 10 steps on the cvxqp3 matrix. We think that the poor convergence of the iterative refinement process and sometimes the degradation in the precision is due to the accumulation of both rounding errors and perturbations. When delayed pivots are not allowed then the static perturbations and their influence on rounding errors are localized to the contribution blocks predicted by the analysis. If we allow delayed pivots, some pivots can be postponed and it is possible that they remain unacceptable in ancestor nodes because of the numerical pivoting test until we switch to static pivoting mode. In that case after switching to static mode and when a delayed pivot (possibly postponed over several generations) is still small, it is perturbed. Then this elimination contaminates a larger contribution block than if it had been eliminated at an earlier node.

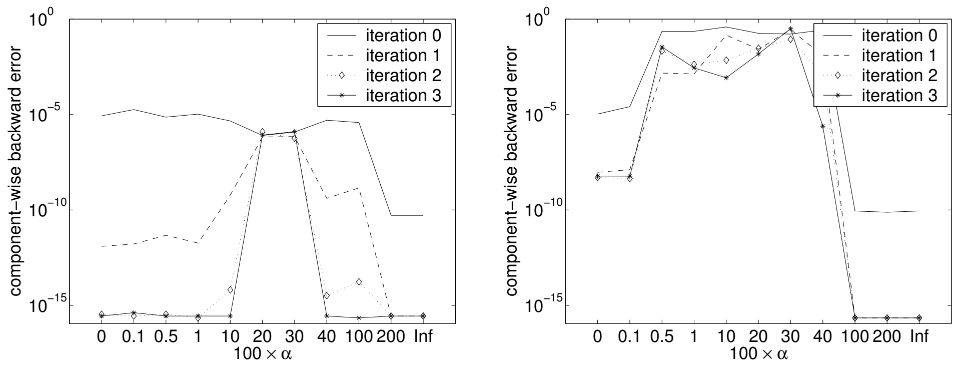
We tried a wide range of rules to decide how to delay the elimination of a variable and remark that all strategies were very unstable. For example, we tried a strategy that, before postponing the elimination of a variable i, checks to see if a large off-diagonal entry in row i and in the fully summed variables of the parent node exists, but these strategies did not improve our basic combination of delayed pivots and static pivoting.

	Iteration 0			Iteration 1		
Matrix	`mixSEQ`	$\alpha = 0.01$	$\alpha = 0.1$	`mixSEQ`	$\alpha = 0.01$	$\alpha = 0.1$
BRAINPC2	2.1e-08	2.3e-08	2.0e-08	5.7e-15	8.7e-15	3.0e-14
BRATU3D	9.2e-06	2.9e-01	1.4e-01	2.2e-11	4.6e-07	7.1e-06
CONT-201	1.0e-05	2.2e-01	3.8e-01	9.4e-09	1.3e-03	1.4e-01
CONT-300	2.1e-05	1.8e-01	1.5e-01	2.7e-09	5.0e-02	1.7e-02
cvxqp3	8.5e-06	1.0e-05	4.6e-06	1.2e-12	1.8e-12	5.9e-10
DTOC	8.3e-07	8.3e-07	3.7e-10	2.1e-13	2.1e-13	1.3e-19
mario001	3.1e-08	2.7e-08	3.2e-08	2.5e-13	2.5e-13	2.1e-13
NCVXQP1	4.9e-13	4.6e-13	6.0e-13	3.2e-15	1.7e-14	1.4e-14
NCVXQP5	2.0e-08	1.1e-08	3.6e-08	6.7e-11	5.1e-11	6.2e-11
NCVXQP7	4.9e-06	4.1e-07	1.6e-06	1.4e-12	1.2e-07	7.1e-11
SIT100	2.0e-08	1.4e-08	8.6e-08	5.8e-15	6.4e-15	3.3e-14
stokes128	4.2e-14	4.2e-14	4.2e-14	2.0e-15	1.9e-15	3.2e-15
stokes64	1.6e-13	1.6e-13	1.6e-13	2.3e-14	1.0e-13	2.8e-14

Table 5.5: Component-wise backward error. α is the input parameter of Algorithm 2.

We show in Figure 5.2 the precision of the solution while increasing the number of delayed pivots allowed (α parameter of Algorithm 2). This approach with $\alpha = 0$ is equivalent to the `mixSEQ` strategy and $\alpha = \infty$ (Inf in Figure 5.2) corresponds to the `numSEQ` strategy. For the same number of iterations, even for α very small, the backward error may be larger with delayed pivots than with the `mixSEQ` strategy (for example see the residual values for CONT-201).

It seems also quite difficult to predict the behaviour of the precision. For example, there is a small window, $10 \leq 100\alpha \leq 40$, in which Algorithm 2 does not return an accurate solution for cvxqp3. The failure window is completely different for CONT-201, $0.1 \leq 100\alpha \leq 100$. It seems that for the middle values of α, we first delay pivots and then force static pivoting on fronts whose size has significantly increased and that contain many small entries. The elimination of pivots is then often associated with a relatively large growth factor and is very dangerous because their updates affect a large number of uneliminated variables. If they had been eliminated near the bottom of the tree in a smaller front, they would not have degraded the quality of the factorization as much.



(a) cvxqp3 (b) CONT-201

Figure 5.2: Influence of the number of delayed pivots on the precision of the solution.

6 Pivoting strategies in parallel distributed environments

6.1 Parallel distributed approaches

We first discuss the parallelism that is exploited by distributed multifrontal solvers, focusing on the `MUMPS` approach and using the terminology from that work (Amestoy et al. 2001, Amestoy et al. 2000).

A pair of nodes in the assembly tree where neither is an ancestor of the other can be factorized independently from each other, in any order or in parallel. Consequently, independent branches of the assembly tree can be processed in parallel, and we refer to this as *tree parallelism* or *type 1 parallelism.* It is obvious that, in general, tree parallelism can be exploited more efficiently in the lower part of the assembly tree than near the root node. Additional parallelism is then created using distributed memory versions of blocked algorithms to factorize the frontal matrices (see, for example, Amestoy et al., 2000, Dongarra et al., 1998).

The lower triangular part of the frontal matrix is partitioned and each part of it is assigned to a different process. The so called *master process* is responsible for the block of fully summed variables and will also decide which processes (the so called *slave processes*) will be involved in the parallel activity associated with this node. We refer to this as *type 2 parallelism* and call the nodes involved *type 2 nodes.*

Figure 6.3 shows an example of the parallelization of a type 2 node and the consequent communication scheme. After eliminating a set of pivots, the master sends

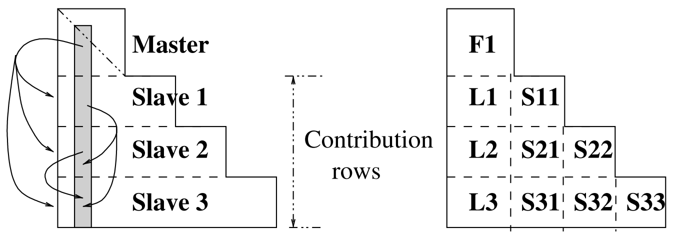


Figure 6.3: An example of parallelization of a type 2 node.

the corresponding factors to its slaves. Then the slaves have to communicate with each other to transmit the rest of the fully summed rows. For example, Slave 2 needs to receive L1 from Slave 1 to update the S21 block.

After computing its part of the contribution block a slave will communicate with the master and the slaves of the parent node. Thus slaves of Figure 6.3 will send the rows and columns that are fully summed at the parent node to the master of the parent node.

6.2 Limiting the areas for pivot checking

The diagonal block of the fully summed part of a type 2 node is stored on a single processor, the master. Furthermore, the master does not have local access to the other rows of the front, which are sent directly from the slaves of its child nodes to its own slaves. To avoid extra communications and, even worse, synchronizations we set $K = FSV$ for type 2 nodes in Algorithm 1. This **B**asic **PAR**allel strategy with **NUM**erical pivoting will be referred to as `numBPAR`. Because the `numBPAR` strategy means that it is not necessary to communicate between processes to perform the threshold tests, this strategy is more friendly for the parallel distributed implementation of `MUMPS` (Amestoy et al. 2001) and more generally for other distributed solvers, for example `SuperLU_DIST` (Li and Demmel 2003).

Moreover, we are obliged to do some slight variations while computing the quantities g_1 and g_2 of equations (4.3) and (4.4) to avoid mistakes because we do not have access to the non fully summed part. That is why we always suppose that the largest off-diagonal entry is greater than $\mu||A||_M$ in order to avoid the selection of a pivot whose fully summed part is small whereas its partially summed part is large. Thus for every type 2 node we define

$$g_1(i) = \frac{\max\{\max_{k\in FSV\setminus\{i\}} |a_{ik}|, \mu\ ||A||_M\}}{|a_{ii}|}, \tag{6.7}$$

and

$$g_2(i,j) = \left|\left| |P^{-1}| \begin{pmatrix} \max\{\max_{k\in FSV\setminus\{i,j\}} |a_{ik}|, \mu\ ||A||_M\} \\ \max\{\max_{k\in FSV\setminus\{i,j\}} |a_{jk}|, \mu\ ||A||_M\} \end{pmatrix} \right|\right|_\infty . \tag{6.8}$$

These quantities are used in the `numBPAR` strategy to adapt the numerical pivoting in the Duff-Reid algorithm on type 2 nodes to parallel distributed environments. Note that a similar strategy is already used in the symmetric code of `MUMPS` in the selection of 1×1 pivots (Version 4.3 of `MUMPS` does not perform 2×2 pivoting), but has never been presented.

6.3 Cheap estimation of growth factors

We will see in Section 6.4 that the `numBPAR` approach is not robust. In this section, we propose an approximation of the off-diagonal information that does not limit the scalability and that will significantly improve the numerical robustness of the factorization.

For each fully summed variable of a node p, each slave of its children sends to the master of p the maximum entry that it has computed in its contribution block. Then the master of the parent node will approximate the maximum entry in each column by the maximum quantity that it has received from the slaves of its child nodes. Note that it is only an approximation because the child contributions are summed and no account is taken of numerical growth as the eliminations at node p are performed.

More formally, let c be a child node and p be its parent. PSV_c (resp. FSV_c) and PSV_p (resp. FSV_p) denote the partially (resp. fully) summed variables of the child and the parent respectively. Thus we have $PSV_c \subset (FSV_p \cup PSV_p)$. Let $CB = (cb_{ij})$ be the part of the contribution block computed by a slave s of c. If we define R_s (resp. C_s) as the set of partially summed rows (resp. columns) that belong to s, then $R_s \subset PSV_c$ (resp. $C_s \subset PSV_c$). Note that $R_s \neq PSV_c$ if the child node has more than one slave.

If the slave s is in charge of a row $i \in FSV_p$, it computes the entries cb_{ik}, $k \in C_s$ and $k \leq i$. If it is in charge of a column $i \in FSV_p$, it computes the entries cb_{ki}, $k \in R_s$ and $k \geq i$. Thus slave s can compute the largest element (in magnitude) in its rows and columns that appear in FSV_p and send them to the master of node p. We set

$$\begin{array}{l} m_i^r(s) = \max_{k \in C_s \setminus FSV_p, k \leq i} |cb_{ik}|, \ \forall i \in R_s \cap FSV_p, \\ m_i^r(s) = 0, \ \forall i \in FSV_p \setminus R_s, \end{array} \tag{6.9}$$

and

$$\begin{array}{l} m_i^c(s) = \max_{k \in R_s \setminus FSV_p, k \geq i} |cb_{ki}|, \ \forall i \in C_s \cap FSV_p, \\ m_i^c(s) = 0, \ \forall i \in FSV_p \setminus C_s. \end{array} \tag{6.10}$$

Finally we define

$$m_i(s) = \max\{m_i^r(s), m_i^c(s)\}, \text{ for each } i \in FSV_p. \tag{6.11}$$

The areas checked for the computation of these quantities are illustrated in Figure 6.4; each slave accesses the shaded areas to compute its m_i quantities and communicates them to the father while sending the black blocks of its contributions. The other parts of the contribution blocks (shaded and blank) are sent directly to the slaves of the parent node.

While receiving information about maximum off-diagonal entries, the master of the parent estimates the maximum off-diagonal entry in a fully summed row i by the quantity M_i:

$$M_i = \max\left\{ \max_{c \text{ child of } p} \left\{ \max_{s \text{ slave of } c} \{m_i(s)\} \right\}, \max_{k \in PSV_p} |a_{ik}^{(0)}| \right\} \tag{6.12}$$

where $a^{(0)}$ denote the original entries of $\mathbf{A}$ that are assembled at the parent node.

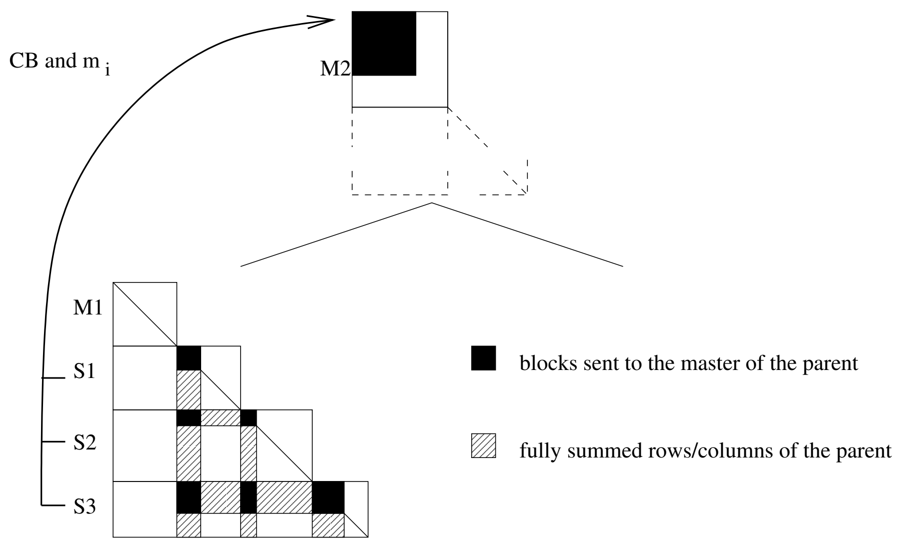

CB and m i
M2
M1
S1
S2
S3
blocks sent to the master of the parent
fully summed rows/columns of the parent

Figure 6.4: Illustration of the areas accessed to estimate the m_i quantities of equation (6.11) and of the blocks that are sent from a slave of a child to the master of the parent.

For each pivoting strategy (numerical pivoting or combination of numerical and static pivoting) the growth factor quantities of equations (6.7) and (6.8) are adapted. For type 2 nodes we define

$$g_1(i) = \frac{\max\{\max_{k \in FSV_p} |a_{ik}|, M_i, \mu \; ||A||_M\}}{|a_{ii}|}, \tag{6.13}$$

and

$$g_2(i,j) = \left|\left| |P^{-1}| \begin{pmatrix} \max\{\max_{k \in FSV_p} |a_{ik}|, M_i, \mu \; ||A||_M\} \\ \max\{\max_{k \in FSV_p} |a_{ik}|, M_j, \mu \; ||A||_M\} \end{pmatrix} \right|\right|_\infty . \tag{6.14}$$

These quantities are then used to adapt the Duff-Reid algorithm on type 2 nodes to parallel distributed environments. This **NUM**erical pivoting strategy based on **E**stimations of off-diagonal entries in a **PAR**allel framework will be referred to as `numEPAR`. We will also use these quantities to adapt Algorithm 1 for type 2 nodes. This static pivoting strategy will be referred to as `mixPAR`.

6.4 Experimental results

In our experiments, we consider that a node is a type 2 node if it is large enough (in practice if $\#PSV > 400$) and it is not a leaf node nor the root node.

6.4.1 Parallel approaches with numerical pivoting

Table 6.6 shows that the basic parallel adaptation of the Duff-Reid algorithm, `numBPAR`, is not robust. We observe numerical failures on the CONT-201, CONT-300 and BRATU3D matrices. Furthermore, there is a significant degradation of the precision compared to the `numSEQ` approach. We see that `numEPAR` is robust (no numerical failures) on our challenging

	Iteration 0			Iteration 1		
Matrix	`numSEQ`	`numBPAR`	`numEPAR`	`numSEQ`	`numBPAR`	`numEPAR`
BRAINPC2	1.6e-15	1.6e-15	1.6e-15	1.0e-15	1.0e-15	1.0e-15
BRATU3D	2.0e-09	4.8e-01	7.8e-09	1.7e-16	4.6e-01	1.7e-16
CONT-201	8.8e-11	1.3e-01	3.0e-10	1.6e-16	1.2e-01	2.3e-16
CONT-300	7.6e-11	1.3e-01	1.4e-10	1.9e-16	7.7e-02	1.7e-16
cvxqp3	5.2e-11	4.8e-06	2.2e-10	2.7e-16	2.5e-06	2.7e-16
DTOC	2.1e-16	2.1e-16	2.1e-16	2.7e-20	2.7e-20	2.7e-20
mario001	6.3e-15	6.3e-15	6.3e-15	1.3e-16	1.3e-16	1.3e-16
NCVXQP1	4.6e-14	9.2e-14	5.2e-14	1.7e-17	3.0e-17	2.7e-17
NCVXQP5	2.0e-11	2.5e-10	5.8e-11	2.0e-16	2.0e-16	2.0e-16
NCVXQP7	9.6e-10	5.9e-08	1.3e-09	2.2e-16	1.2e-07	2.2e-16
SIT100	4.4e-15	4.4e-15	4.4e-15	1.4e-16	1.4e-16	1.4e-16
stokes128	1.1e-14	1.1e-14	1.1e-14	5.5e-16	5.5e-16	5.5e-16
stokes64	4.3e-15	4.3e-15	4.3e-15	1.5e-15	1.5e-15	1.5e-15

Table 6.6: Component-wise backward error of strategies with numerical pivoting. `numSEQ`: sequential approach. `numBPAR`: basic parallel approach. `numEPAR`: parallel approach using estimations.

test set. Even if the first solution may not be as good as with the `numSEQ` strategy, we see that, after one step of iterative refinement, the backward errors of `numSEQ` and `numEPAR` are similar. That is why we recommend automatically performing one iterative refinement step with the `numEPAR` strategy and then checking the accuracy of the solution.

	Size of the factors			Number of delayed pivots		
Matrix	`numSEQ`	`numBPAR`	`numEPAR`	`numSEQ`	`numBPAR`	`numEPAR`
BRATU3D	11484379	9672751	11249260	90052	49205	87167
CONT-201	8820367	8918700	8829464	71296	71415	71389
CONT-300	23838606	23595744	23928663	183306	182422	183641

Table 6.7: Size of the factors and number of delayed pivots with different numerical pivoting strategies.

Table 6.7 compares the size of the factors and the number of delayed pivots between the `numSEQ`, `numBPAR` and `numEPAR` pivoting strategies. We focus on the BRATU3D and CONT-* matrices because they reveal the weaknesses of the `numBPAR` strategy.

Firstly we observe that `numSEQ` and `numEPAR` have a similar number of entries in the factors and number of delayed pivots. This supports the idea that the `numEPAR` strategy takes a good numerical decision even if it only has an approximate view of the off-diagonal entries. For each node of the assembly tree, Figures 6.5 and 6.6 represent the number of pivots that are delayed (right-hand side) and the maximum of the quantities g_1 and g_2 of equations (4.3) and (4.4) with the growth factors computed over $FSV \cup PSV$ (left-hand side) on the BRATU3D and CONT-201 matrices. Thus the left-hand side figures show the real growth factors while the pivoting strategy may take its decision according to a different estimation (for example over FSV and estimates for entries in PSV). With the numSEQ strategy the estimation and the actual values are exactly the same. That is why we always have the actual growth factors smaller than 10^{-2}. With numBPAR and numEPAR, the actual value may be underestimated. That is why we observe growth factors greater than 10^{-2}. Figures 6.5 and 6.6 confirm that the numSEQ and numEPAR strategies have a similar behaviour, in the sense that they postpone approximately the same number of pivots at each node. Furthermore the numEPAR strategy succeeds in bounding the growth factor by 2×10^4 and 4×10^3 at each step of Gaussian elimination on BRATU3D and CONT-201 respectively.

The behaviour of the numBPAR strategy is significantly different. We see, in Table 6.7, that the number of delayed pivots decreases significantly on the BRATU3D matrix. A consequence is the decrease in the number of nonzeros in the factors (9.7×10^6 for numBPAR versus 11.5×10^6 for numSEQ). In Figure 6.5, we observe growth factors greater than $1/\sqrt{\epsilon}$ on the BRATU3D matrix. It explains why the solution is not accurate. We also see a similar number of delayed pivots with numSEQ and numEPAR whereas numBPAR makes bad numerical decisions because it delays fewer pivots.

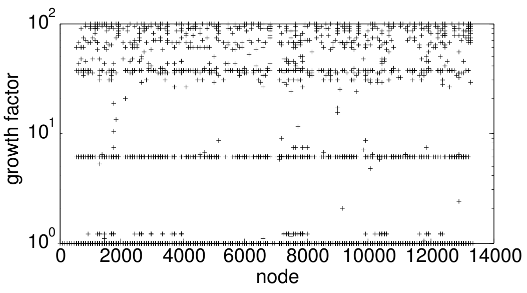
On the CONT-201 and CONT-300 matrices, numBPAR delays approximately the same number of variables and computes factors with a similar number of nonzeros. Figure 6.6 shows that numBPAR does not guarantee a reasonable growth factor on CONT-201 (growth factors nearly equal to $1/\epsilon$ for some nodes of the assembly tree). We see also that the number of delayed pivots are significantly different between the numBPAR strategy and the two other strategies, numSEQ and numEPAR, for some nodes of the tree.

6.4.2 Parallel approaches combining numerical and static pivoting

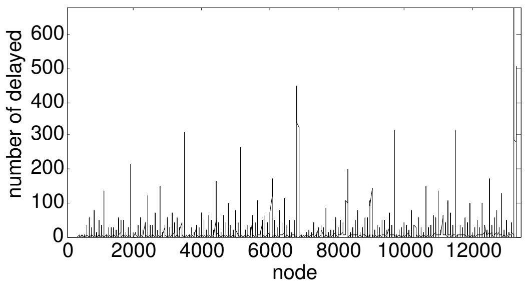
Table 6.8 shows that there are no significant differences between the sequential and the parallel version of our combination of numerical and static pivoting. We still need two iterations to converge to the machine precision on most of the matrices.

7 Influence of preprocessing

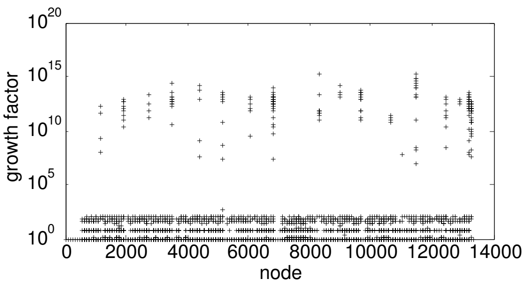
In Duff and Pralet (2004), we presented preprocessing techniques to improve the quality of preselected pivots. Our preprocessing uses a maximum weighted matching and sparsity ordering techniques. Of the techniques presented, we saw that METIS combined with the MC64SYM scaling was the best ordering on symmetric indefinite matrices in terms of CPU factorization time but that it sometimes caused many pivots to be delayed. We also proposed an ordering based on METIS that increases the number of nonzeros in the factors slightly but that clearly improves the numerical stability for the preselected pivots. In this section, we study the influence of this preprocessing in a static pivoting context. We use these two orderings for the present experiments: the METIS ordering and an ordering that we call METIS on the compressed graph (denoted by CMP). Our CMP ordering has four main phases:



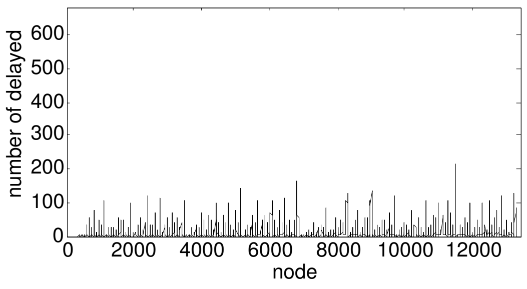
(a) Growth factor on BRATU3D using `numSEQ` pivoting strategy.



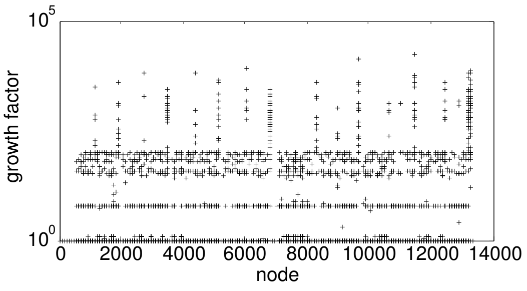
(b) Number of delayed pivots on BRATU3D using `numSEQ` pivoting strategy.



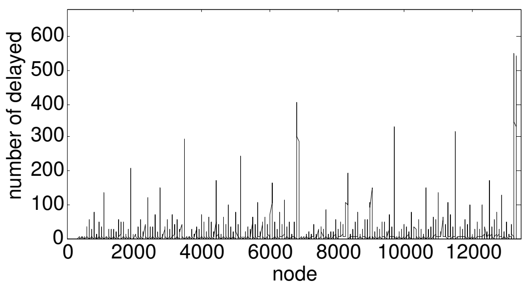
(c) Growth factor on BRATU3D using `numBPAR` pivoting strategy.



(d) Number of delayed pivots on BRATU3D using `numBPAR` pivoting strategy.

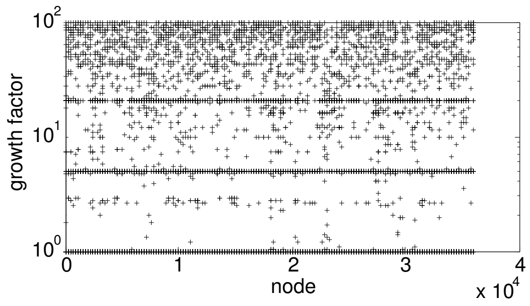


(e) Growth factor on BRATU3D using `numEPAR` pivoting strategy.

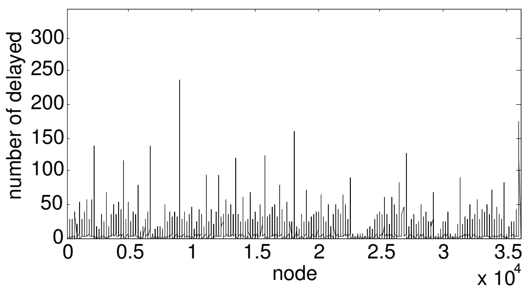


(f) Number of delayed pivots on BRATU3D using `numEPAR` pivoting strategy.

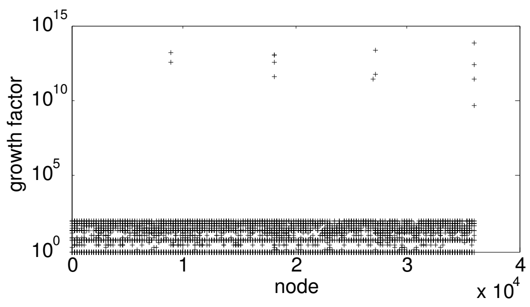
Figure 6.5: Influence of the pivoting strategy on the growth factor and the number of delayed pivots on BRATU3D.



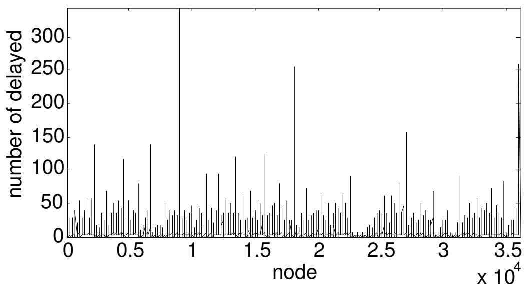
(a) Growth factor on CONT-201 using `numSEQ` pivoting strategy.



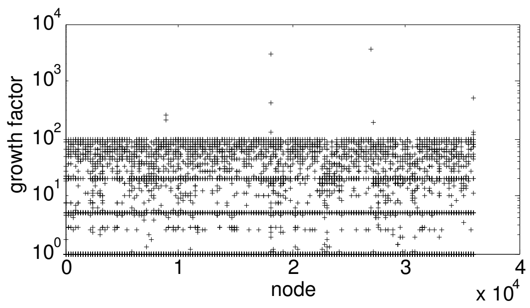
(b) Number of delayed pivots on CONT-201 using `numSEQ` pivoting strategy.



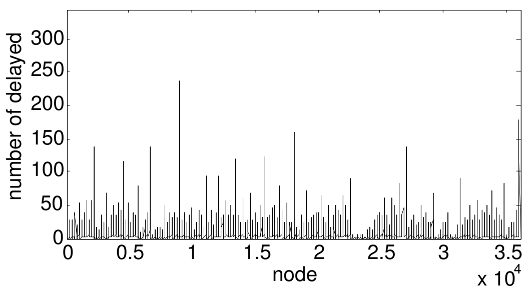
(c) Growth factor on CONT-201 using `numBPAR` pivoting strategy.



(d) Number of delayed pivots on CONT-201 using `numBPAR` pivoting strategy.



(e) Growth factor on CONT-201 using `numEPAR` pivoting strategy.



(f) Number of delayed pivots on CONT-201 using `numEPAR` pivoting strategy.

Figure 6.6: Influence of the pivoting strategy on the growth factor and the number of delayed pivots on CONT-201.

	Iteration 0		Iteration 1		Iteration 2	
Matrix	`mixSEQ`	`mixPAR`	`mixSEQ`	`mixPAR`	`mixSEQ`	`mixPAR`
BRAINPC2	2.1e-08	2.1e-08	5.7e-15	5.7e-15	9.8e-16	9.8e-16
BRATU3D	9.2e-06	1.7e-05	2.2e-11	1.3e-10	1.7e-16	2.3e-16
CONT-201	1.0e-05	1.8e-05	9.4e-09	9.4e-09	4.9e-09	4.5e-09
CONT-300	2.1e-05	1.9e-05	2.7e-09	2.6e-09	2.5e-09	3.4e-09
cvxqp3	8.5e-06	8.0e-06	1.2e-12	9.3e-13	3.4e-16	2.7e-16
DTOC	8.3e-07	8.3e-07	2.1e-13	2.1e-13	1.9e-15	1.9e-15
mario001	3.1e-08	3.1e-08	2.5e-13	2.5e-13	1.3e-16	1.3e-16
NCVXQP1	4.9e-13	3.3e-13	3.2e-15	4.4e-15	2.6e-17	6.1e-17
NCVXQP5	2.0e-08	7.5e-08	6.7e-11	1.6e-11	2.7e-14	1.5e-14
NCVXQP7	4.9e-06	4.3e-06	1.4e-12	2.0e-12	2.2e-16	2.7e-16
SIT100	2.0e-08	2.0e-08	5.8e-15	5.8e-15	1.5e-16	1.5e-16
stokes128	4.2e-14	4.2e-14	2.0e-15	2.0e-15	1.7e-15	1.7e-15
stokes64	1.6e-13	1.6e-13	2.3e-14	2.3e-14	2.2e-14	2.2e-14

Table 6.8: Component-wise backward error of strategies with static pivoting. `mixSEQ`: sequential approach. `mixPAR`: parallel approach using estimations of equation (6.12) and pivoting Algorithm 1.

1. we compute a symmetric weighted matching to select 1×1 and 2×2 pivots,
2. we compress variables that belong to the same 2×2 pivot into a single supervariable and merge their adjacency structures,
3. we apply METIS to the compressed graph and obtain a permutation,
4. we expand the permutation to the initial expanded matrix.

7.1 Sequential approaches

Table 7.9 compares our combination of static and numerical pivoting when using the above two orderings. We see that the preselection of 2×2 pivots using a symmetric weighted matching significantly decreases the number of tiny pivots and improves the precision of the solution. This influence of pivot preselection using maximum weighted matching techniques has also been observed in the context of `SuperLU_DIST` (Li and Demmel 2003). We see that, in most of the cases, the approach with an ordering on the compressed graph needs one iteration fewer than an approach based on the METIS ordering.

Generally the approach on the compressed graph increases the number of operations and the fill-in in the factors. Thus it increases also the factorization time (see Table 7.10). Note that the compressed approach is better on CONT-201 and CONT-300 in terms of fill-in in the factors and factorization time because the compression detects cliques and then improves the quality of the fill-in reducing phase (METIS). The `CMP` approach also significantly improves the residual for the CONT matrices.

We also compare the solution times in Table 7.10. Generally the `CMP` strategy increases the size of the factors and thus increases the time for backward and forward substitution.

	Iteration 0		Iteration 1		Iteration 2	Tiny pivots	
Matrix	METIS	`CMP`	METIS	`CMP`	METIS	METIS	`CMP`
BRAINPC2	2.1e-08	4.3e-13	5.7e-15	9.5e-16	9.8e-16	12932	0
BRATU3D	9.2e-06	7.1e-08	2.2e-11	1.4e-15	1.7e-16	8429	284
CONT-201	1.0e-05	8.6e-14	9.4e-09	2.0e-16	4.9e-09	27470	0
CONT-300	2.1e-05	3.4e-13	2.7e-09	1.8e-16	2.5e-09	67864	0
cvxqp3	8.5e-06	5.3e-06	1.2e-12	3.2e-14	3.4e-16	6277	30
DTOC	8.3e-07	6.1e-16	2.1e-13	2.6e-20	1.9e-15	9790	4999
mario001	3.1e-08	5.9e-08	2.5e-13	3.1e-15	1.3e-16	10305	29
NCVXQP1	4.9e-13	2.1e-14	3.2e-15	6.7e-15	2.6e-17	3619	10
NCVXQP5	2.0e-08	6.4e-12	6.7e-11	2.0e-16	2.7e-14	8402	0
NCVXQP7	4.9e-06	2.1e-08	1.4e-12	1.8e-14	2.2e-16	31043	46
SIT100	2.0e-08	1.2e-07	5.8e-15	1.3e-14	1.5e-16	1388	6
stokes128	4.2e-14	5.9e-14	2.0e-15	4.9e-14	1.7e-15	12738	27
stokes64	1.6e-13	1.6e-13	2.3e-14	2.2e-13	2.2e-14	3106	5

Table 7.9: Influence of the preprocessing on the component-wise backward error and on the number of perturbed pivots. METIS: `mixSEQ` with METIS ordering. `CMP`: `mixSEQ` with a preprocessing based on symmetric weighted matching and on METIS.

	Factorization time		Time for forward and backward substitution	
Matrix	METIS	`CMP`	METIS	`CMP`
BRAINPC2	0.11	0.11	0.014	0.015
BRATU3D	9.24	8.56	0.125	0.124
CONT-201	1.94	1.44	0.127	0.121
CONT-300	6.08	4.38	0.306	0.294
cvxqp3	3.08	9.46	0.048	0.110
DTOC	0.41	62.6	0.006	0.133
mario001	0.23	0.31	0.022	0.029
NCVXQP1	1.29	5.44	0.024	0.055
NCVXQP5	23.0	39.8	0.279	0.425
NCVXQP7	71.6	199.	0.498	0.988
SIT100	0.11	0.13	0.004	0.005
stokes128	1.06	1.63	0.076	0.111
stokes64	0.29	0.38	0.013	0.017

Table 7.10: Influence of the preprocessing on the factorization and solution time (in seconds). METIS: `mixSEQ` with METIS ordering. `CMP`: `mixSEQ` with a preprocessing based on symmetric weighted matching and on METIS.

Nevertheless this effect is compensated by the decrease in the number of iterative refinement steps: on average the `CMP` strategy requires one backward and forward substitution and one matrix-vector multiplication fewer than the METIS strategy.

7.2 Parallel approaches

In this section, we study the influence of the preprocessing on parallel approaches. As in the sequential case, our preprocessing improves the numerical robustness and we get small backward errors after one iterative refinement step (see Table 7.11). Our preprocessing is beneficial for both the numerical pivoting strategies and the combination of numerical and static pivoting in particular for the CONT matrices.

	Iteration 0				Iteration 1			
Matrix	`numEPAR`	`mixPAR`	`numEPAR_P`	`mixPAR_P`	`numEPAR`	`mixPAR`	`numEPAR_P`	`mixPAR_P`
BRAINPC2	1.6e-15	2.1e-08	3.4e-14	4.3e-13	1.0e-15	5.7e-15	2.3e-15	9.5e-16
BRATU3D	7.8e-09	1.7e-05	1.0e-14	6.9e-08	1.7e-16	1.3e-10	1.7e-16	1.6e-15
CONT-201	3.0e-10	1.8e-05	1.1e-13	1.2e-13	2.3e-16	9.4e-09	1.8e-16	1.7e-16
CONT-300	1.4e-10	1.9e-05	5.9e-14	2.4e-13	1.7e-16	2.6e-09	1.7e-16	1.7e-16
cvxqp3	2.2e-10	8.0e-06	1.8e-10	5.3e-06	2.7e-16	9.3e-13	2.7e-16	3.2e-14
DTOC	2.1e-16	8.3e-07	1.5e-18	6.1e-16	2.7e-20	2.1e-13	9.0e-21	2.6e-20
mario001	6.3e-15	3.1e-08	2.9e-15	5.9e-08	1.3e-16	2.5e-13	1.3e-16	3.1e-15
NCVXQP1	5.2e-14	3.3e-13	1.9e-14	1.6e-14	2.7e-17	4.4e-15	3.7e-17	6.7e-15
NCVXQP5	5.8e-11	7.5e-08	5.1e-12	6.8e-12	2.0e-16	1.6e-11	2.0e-16	2.0e-16
NCVXQP7	1.3e-09	4.3e-06	9.0e-11	1.6e-08	2.2e-16	2.0e-12	2.2e-16	2.0e-14
SIT100	4.4e-15	2.0e-08	8.4e-15	1.2e-07	1.4e-16	5.8e-15	1.7e-16	1.3e-14
stokes128	1.1e-14	4.2e-14	8.3e-15	5.9e-14	5.5e-16	2.0e-15	1.0e-14	4.9e-14
stokes64	4.3e-15	1.6e-13	5.1e-15	1.6e-13	1.5e-15	2.3e-14	3.0e-15	2.2e-13

Table 7.11: Component-wise backward error of strategies with numerical pivoting. `numEPAR`: numerical pivoting strategy without preprocessing. `mixPAR`: static pivoting strategy without preprocessing. `numEPAR_P`: numerical pivoting strategy with preprocessing. `mixPAR_P`: static pivoting strategy with preprocessing.

8 Conclusions

We have presented different choices for static pivoting and have identified their main characteristics. We have implemented these within the `MA57` code. Our new pivoting strategies that combine numerical and static pivoting seem to address a large class of problems and to be significantly faster than approaches with standard numerical pivoting. Table 8.12 summarizes our recommended default number of iterative refinement steps for each pivoting strategy. Concerning classes of problem on which `mixPAR` or `mixSEQ` strategies does not compute a precise enough solution, we recommend first preprocessing the matrix using a compressed graph and a maximum weighted matching, and then if necessary using the numerical pivoting strategies, `numEPAR` or `numSEQ`.

Our parallel approaches can easily be generalized to the unsymmetric case. It would improve the precision of `SuperLU_DIST` and allow 2D partitioning in `MUMPS`.

Even if the precision of the solution is good, further improvements can be obtained. Firstly, there are still problems on which our combination of numerical and static pivoting

Name	numSEQ	mixSEQ	numEPAR	mixPAR
It. Step	0/0	1/1	1/0	2/1

Table 8.12: Number of recommended iterative refinement steps before checking precision of the solution. It. Step: number of iterative refinement steps. x/y means that we recommend x steps if no preprocessing is used and y steps if preprocessing and an ordering on the compressed graph is used.

is less accurate than approaches that perform standard threshold pivoting. Secondly, the number of iterative refinement steps could be decreased. This problem becomes all the more critical when we have many right-hand sides. In our future work, we would like to decrease the number of iterative refinement steps by trying to include 2×2 perturbations. Hence, it might be useful either to decide a priori where the perturbation could be applied or to discover large entries dynamically during the factorization. More generally we would like to try off-diagonal perturbations to prevent future numerical problems. Such decisions would be taken in a look-ahead style as in the Schnabel-Eskow modified Cholesky factorization.

A much higher value for μ would result in a far more stable factorization but perhaps with a modified matrix further from the original. We intend to study the trade-off whose optimal value might well be quite different if more sophisticated iterative methods than iterative refinement were used (for example MINRES). We think that in this framework the use of 2×2 perturbations may have a strong influence.

Acknowledgments

We would like to thank John Reid and Jennifer Scott of the Rutherford Appleton Laboratory for their comments on a draft version of this manuscript. We are also very grateful to Serge Gratton for discussions on various aspects of this work.

References

Amestoy, P. R., Duff, I. S. and L'Excellent, J.-Y. (2000), 'Multifrontal parallel distributed symmetric and unsymmetric solvers', *Comput. Methods in Appl. Mech. Engng.* **184**, 501–520.

Amestoy, P. R., Duff, I. S., Koster, J. and L'Excellent, J.-Y. (2001), 'A fully asynchronous multifrontal solver using distributed dynamic scheduling', *SIAM J. Matrix Analysis and Applications* **23**(1), 15–41.

Arioli, M., Demmel, J. W. and Duff, I. S. (1989), 'Solving sparse linear systems with sparse backward error', *SIAM J. Matrix Analysis and Applications* **10**, 165–190.

Ashcraft, C., Grimes, R. G. and Lewis, J. G. (1998), 'Accurate symmetric indefinite linear equation solvers', *SIAM J. Matrix Analysis and Applications* **20**(2), 513–561.

Bunch, J. R. and Kaufman, L. (1977), 'Some stable methods for calculating inertia and solving symmetric linear systems', *Mathematics of Computation* **31**, 162–179.

Bunch, J. R. and Parlett, B. N. (1971), 'Direct methods for solving symmetric indefinite systems of linear equations', *SIAM J. Numerical Analysis* **8**, 639–655.

Cheng, S. H. and Higham, N. J. (1998), 'A modified Cholesky algorithm based on a symmetric indefinite factorization', *SIAM J. Matrix Analysis and Applications* **19**(4), 1097–1112.

Dongarra, J. J., Duff, I. S., Sorensen, D. C. and van der Vorst, H. A. (1998), *Numerical Linear Algebra for High-Performance Computers*, SIAM Press, Philadelphia.

Duff, I. S. (2004), 'MA57 – A code for the solution of sparse symmetric indefinite systems', *ACM Trans. Math. Softw.* **30**(2), 118–144.

Duff, I. S. and Pralet, S. (2004), Strategies for scaling and pivoting for sparse symmetric indefinite problems, Technical Report RAL-TR-2004-020, Rutherford Appleton Laboratory, Oxfordshire, England.

Duff, I. S. and Reid, J. K. (1983), 'The multifrontal solution of indefinite sparse symmetric linear systems', *ACM Trans. Math. Softw.* **9**, 302–325.

Duff, I. S. and Reid, J. K. (1996), 'Exploiting zeros on the diagonal in the direct solution of indefinite sparse symmetric linear systems', *ACM Trans. Math. Softw.* **22**(2), 227–257.

Eskow, E. and Schnabel, R. B. (1991), 'Algorithm 695: Software for a new modified Cholesky factorization', *ACM Trans. Math. Softw.* **17**, 306–312.

Gill, P. E. and Murray, W. (1974), 'Newton-type methods for unconstrained and linearly constrained optimization', *Mathematical Programming* **28**, 311–350.

Gould, N. I. M. and Scott, J. A. (2004), 'A numerical evaluation of HSL packages for the direct solution of large sparse, symmetric linear systems of equations', *ACM Transactions on Mathematical Software* **30**(3), 300–325.

Gould, N. I. M., Orban, D. and Toint, P. L. (2002), CUTEr (and SifDec), a constrained and unconstrained testing environment, revisited, Technical Report RAL-TR-2002-009, Rutherford Appleton Laboratory.

Higham, N. J. (2002), *Accuracy and Stabilty of Numerical Algorithms*, 2. edn, SIAM, Philadelphia.

HSL (2004), 'HSL 2004: A collection of Fortran codes for large scale scientific computation'. `www.cse.clrc.ac.uk/nag/hsl`.

Li, X. S. and Demmel, J. W. (1998), Making sparse Gaussian elimination scalable by static pivoting, *in* 'Proceedings of Supercomputing', Orlando, Florida.

Li, X. S. and Demmel, J. W. (2003), 'SuperLU_DIST: A scalable distributed-memory sparse direct solver or unsymmetric linear systems', *ACM Trans. Math. Softw.* **29**(2), 110–140.

Liu, J. W. H. (1990), 'The role of elimination trees in sparse factorization', *SIAM J. Matrix Analysis and Applications* **11**, 134–172.

Liu, J. W. H., Ng, E. G. and Peyton, W. (1993), 'On finding supernodes for sparse matrix computations', *SIAM J. Matrix Analysis and Applications* **14**, 242–252.

Maros, I. and Meszaros, C. (1999), 'A repository of convex quadratic programming problems', *Optimization Methods and Software* **11-12**, 671–681.

Pralet, S. (2004), Constrained orderings and scheduling for parallel sparse linear algebra, Phd thesis, Institut National Polytechnique de Toulouse. CERFACS Technical Report, TH/PA/04/105.

Schenk, O. and Gärtner, K. (2004), On fast factorization pivoting methods for sparse symmetric indefinite systems, Technical Report CS-2004-004, CS Department, University of Basel.

Stewart, G. W. (1974), 'Modifying pivot elements in Gaussian elimination', *Mathematics of Computation* **28**(126), 537–542.