Reducing the amount of pivoting in symmetric indefinite systems

Dulceneia Becker, Marc Baboulin, Jack Dongarra

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Reducing the amount of pivoting in symmetric indefinite systems

Dulceneia Becker — Marc Baboulin — Jack Dongarra

N° 7621

May 2011

Domaine Informatique/Analyse numérique

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--7621--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
SACLAY – ÎLE-DE-FRANCE

Reducing the amount of pivoting in symmetric indefinite systems

Dulceneia Becker* , Marc Baboulin† , Jack Dongarra‡

Domaine :
Équipe-Projet Grand-Large

Rapport de recherche n° 7621 — May 2011 — 12 pages

Abstract: This paper illustrates how the communication due to pivoting in the solution of symmetric indefinite linear systems can be reduced by considering innovative approaches that are different from pivoting strategies implemented in current linear algebra libraries. First a tiled algorithm where pivoting is performed within a tile is described and then an alternative to pivoting is proposed. The latter considers a symmetric randomization of the original matrix using the so-called recursive butterfly matrices. In numerical experiments, we compare the accuracy of tile-wise pivoting and of the randomization approach with the accuracy of the Bunch-Kaufman algorithm.

Key-words: dense linear algebra, symmetric indefinite systems, $\mathrm{LDL}^{\mathrm{T}}$ factorization, pivoting, tiled algorithms, randomization.

* University of Tennessee, USA (`dbecker7@eecs.utk.edu`).
† Université Paris-Sud and INRIA, France (`marc.baboulin@inria.fr`).
‡ University of Tennessee and Oak Ridge National Laboratory, USA, and University of Manchester, United Kingdom (`dongarra@eecs.utk.edu`).

Centre de recherche INRIA Saclay – Île-de-France
Parc Orsay Université
4, rue Jacques Monod, 91893 ORSAY Cedex
Téléphone : +33 1 72 92 59 00

Réduire le pivotage dans les systèmes symétriques indéfinis

Résumé : Ce papier illustre comment il est possible de réduire les communications dûes au pivotage dans la résolution des systèmes symétriques indéfinis en considérant des approches innovantes qui sont différentes des stratégies implémentées dans les bibliothèques logicielles actuelles d'algèbre linéaire. On décrit tout d'abord un algorithme par pavage où le pivotage est effectué à l'intérieur d'un pavé et l'on propose ensuite une alternative au pivotage. Dans cette dernière, on considère une transformation aléatoire symétrique de la matrice de départ en utilisant des matrices appelées "papillons récursifs". Lors d'expériences numériques, on compare la précision du pivotage par pavé et de l'approche par transformation aléatoire avec l'algorithme de Bunch-Kaufman.

Mots-clés : algèbre linéaire dense, systèmes symétriques indéfinis, factorisation LDL^T , pivotage, algorithmes par pavage, transformation aléatoire.

1 Introduction

A symmetric matrix A is called indefinite when the quadratic form $x^T Ax$ can take on both positive and negative values. By extension, a linear system $Ax = b$ is called symmetric indefinite when A is symmetric indefinite. These types of linear systems are commonly encountered in optimization problems coming from physics of structures, acoustics, and electromagnetism, among others. Symmetric indefinite systems also result from linear least squares problems when they are solved via the augmented system method [7, p. 77].

To ensure stability in solving such linear systems, the classical method used is called the diagonal pivoting method [9] where a block-LDL$^{\mathrm{T}}$ factorization[1] is obtained such as

$$PAP^T = LDL^T \tag{1}$$

where P is a permutation matrix, A is a symmetric square matrix, L is unit lower triangular and D is block-diagonal, with blocks of size 1×1 or 2×2; all matrices are of size $n \times n$. If no pivoting is applied, i.e. $P = I$, D becomes diagonal. The solution x can be computed by successively solving the triangular or block-diagonal systems $Lz = Pb$, $Dw = z$, $L^T y = w$, and ultimately we have $x = P^T y$.

There are several pivoting techniques that can be applied to determine P. These methods involve different numbers of comparisons to find the pivot and have various stability properties. As for the LU factorization, the *complete pivoting* method (also called *Bunch-Parlett* algorithm [9]) is the most stable pivoting strategy. It guarantees a satisfying growth factor bound [14, p. 216] but also requires up to $\mathcal{O}(n^3)$ comparisons. The well-known *partial pivoting* method, based on the *Bunch-Kaufman* algorithm [8], is implemented in LAPACK [1] and requires at each step of the factorization the exploration of two columns, resulting in a total of $\mathcal{O}(n^2)$ comparisons. This algorithm has good stability properties [14, p. 219] but in certain cases $\|L\|$ may be unbounded, which is a cause for possible instability [3], leading to a modified algorithm referred to as *rook pivoting* or *bounded Bunch-Kaufman pivoting*. The latter involves between $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ comparisons depending on the number of 2×2 pivots. Another pivoting strategy, called *Fast Bunch-Parlett* strategy (see [3, p. 525] for a description of the algorithm), searches for a local maximum in the current lower triangular part. It is as stable as the rook pivoting but it also requires between $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ comparisons.

With the advent of architectures such as multicore processors [19] and Graphics Processing Unit (GPU), the growing gap between communication and computation efficiency made the communication overhead due to pivoting more critical. These new architectures prompted the need for developing algorithms that lend themselves to parallel execution. A class of such algorithms for shared memory architectures, called *Tiled Algorithms*, has been developed for one-sided dense factorizations[2] [10, 11] and made available as part of the PLASMA [12].

Tiled algorithms are based on decomposing the computation in small tasks in order to overcome the intrinsically sequential nature of dense linear algebra methods. These tasks can be executed out of order, as long as dependencies

[1] Another factorization method is for example the Aasen's method [13, p.163]: $PAP^T = LTL^T$ where L is unit lower triangular and T is tridiagonal.

[2] LDL$^{\mathrm{T}}$ is still under development and shall be available in the future [6].

are observed, rendering parallelism. Furthermore, tiled algorithms make use of a tile data-layout where data is stored in contiguous blocks, which differs from the column-wise layout used by LAPACK, for instance. The tile data-layout allows the computation to be performed on small blocks of data that fit into cache, and hence exploits cache locality and re-use. However, it does not lend itself straightforwardly for pivoting, as this requires a search for pivots and permutations over full columns/rows. For symmetric matrices, the difficulties are even greater since symmetric pivoting requires interchange of both rows and columns. The search for pivots outside a given tile curtails memory locality and data dependence between tiles (or tasks). The former has a direct impact on the performance of serial kernels and the latter on parallel performance (by increasing data dependence among tiles, granularity is decreased and therefore scalability) [18].

In this paper, the possibility of eliminating the overhead due to pivoting by considering randomization techniques is also investigated. These techniques were initially proposed in [16] and modified approaches were studied in [4, 5] for the LU factorization. In this context, they are applied to the case of symmetric indefinite systems. According to this random transformation, the original matrix A is transformed into a matrix that would be sufficiently "random" so that, with a probability close to 1, pivoting is not needed. This transformation is a multiplicative preconditioning by means of random matrices called *recursive butterfly matrices*. The $\mathrm{LDL^T}$ factorization without pivoting is then applied to the preconditioned matrix. One observes that two levels of recursion for butterfly matrices are enough to obtain an accuracy close to that of $\mathrm{LDL^T}$ with either partial (Bunch-Kaufman) or rook pivoting on a collection of matrices. The overhead is reduced to $\sim 8n^2$ operations, which is negligible when compared to the cost of pivoting.

2 Tile-wise Pivoting

Given Equation (1), the tiled algorithm starts by decomposing A in $nt \times nt$ tiles[3] (blocks), where each A_{ij} is a tile of size $mb \times nb$. The same decomposition can be applied to L and D. For instance, for $nt = 3$:

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ L_{31} & L_{32} & L_{33} \end{bmatrix} \begin{bmatrix} D_{11} & & \\ & D_{22} & \\ & & D_{33} \end{bmatrix} \begin{bmatrix} L_{11}^T & L_{21}^T & L_{31}^T \\ & L_{22}^T & L_{32}^T \\ & & L_{33}^T \end{bmatrix}$$

[3]For rectangular matrices, A is decomposed into $mt \times nt$ tiles.

Upon this decomposition and using the same principle as the Schur complement, a series of tasks can be set to calculate each L_{ij} and D_{ii}:

$$
\begin{aligned}
[L_{11}, D_{11}] &= \mathrm{LDL}(A_{11}) && (2)\\
L_{21} &= A_{12}(D_{11}L_{11}^T)^{-1} && (3)\\
L_{31} &= A_{13}(D_{11}L_{11}^T)^{-1} && (4)\\
\tilde{A}_{22} &= A_{22} - L_{21}D_{11}L_{21}^T && (5)\\
[L_{22}, D_{22}] &= \mathrm{LDL}(\tilde{A}_{22}) && (6)\\
\tilde{A}_{32} &= A_{32} - L_{31}D_{11}L_{21}^T && (7)\\
L_{32} &= \tilde{A}_{32}(D_{22}L_{22}^T)^{-1} && (8)\\
\tilde{A}_{33} &= A_{33} - L_{31}D_{11}L_{31}^T - L_{32}D_{22}L_{32}^T && (9)\\
[L_{33}, D_{33}] &= \mathrm{LDL}(\tilde{A}_{33}) && (10)
\end{aligned}
$$

$\mathrm{LDL}(X_{kk})$ at Equations (2), (6) and (10) means the actual $\mathrm{LDL^T}$ factorization of tile X_{kk}. These tasks can be executed out of order, as long as dependencies are observed, rendering parallelism (see [6] for more details).

Following the same approach, for $PAP^T = LDL^T$, Equation (1), i.e. with pivoting, the tasks for $nt = 3$ may be described as:

$$
\begin{aligned}
[L_{11}, D_{11}, P_{11}] &= \mathrm{LDL}(A_{11}) && (11)\\
L_{21} &= P_{22}^T A_{21} P_{11}(D_{11}L_{11}^T)^{-1} && (12)\\
L_{31} &= P_{33}^T A_{31} P_{11}(D_{11}L_{11}^T)^{-1} && (13)\\
\tilde{A}_{22} &= A_{22} - (P_{22}L_{21})D_{11}(P_{22}L_{21})^T && (14)\\
[L_{22}, D_{22}, P_{22}] &= \mathrm{LDL}(\tilde{A}_{22}) && (15)\\
L_{32} &= P_{33}^T \tilde{A}_{32} P_{22}(D_{22}L_{22}^T)^{-1} && (16)\\
\tilde{A}_{33} &= A_{33} - (P_{33}L_{31})D_{11}(P_{33}L_{31})^T - (P_{33}L_{32})D_{22}(P_{33}L_{32})^T && (17)\\
[L_{33}, D_{33}, P_{33}] &= \mathrm{LDL}(\tilde{A}_{33}) && (18)
\end{aligned}
$$

Equations (11) to (18) are similar to Equations (2) to (10), except that the permutation matrix P_{kk} has been added. This permutation matrix P_{kk} generates a cross-dependence between equations, which is not an issue when pivoting is not used. For instance, in order to calculate

$$L_{21} = P_{22}^T A_{21} P_{11} \left(D_{11}L_{11}^T\right)^{-1} \tag{19}$$

P_{22} is required. However, to calculate

$$[L_{22}, D_{22}, P_{22}] = \mathrm{LDL}\left(A_{22} - (P_{22}L_{21})D_{11}(P_{22}L_{21})^T\right) \tag{20}$$

L_{21} is required. To overcome this cross-dependence, instead of actually calculating L_{21}, $P_{22}L_{21}$ is calculated instead, since the equations can be rearranged such as $P_{22}L_{21}$ is always used and therefore L_{21} is not needed. Hence, Equations (12), (13) and (16) become, in a general form,

$$P_{ii}L_{ij} = A_{ij}P_{jj}\left(D_{jj}L_{jj}^T\right)^{-1} \tag{21}$$

After P_{ii} is known, L_{ij}, for $1 \geq j \geq i-1$, can be calculated such as

$$L_{ij} = P_{ii}^T L_{ij} \tag{22}$$

This procedure may be described as in Algorithm 1, where A is a symmetric matrix of size $n \times n$ split in $nt \times nt$ tiles A_{ij}, each of size $mb \times nb$.

Algorithm 1 Tiled LDL$^{\mathrm{T}}$ Factorization with Tile-wise Pivoting

for $k = 1$ to nt **do**
  [L_{kk} , D_{kk} , P_{kk}] = LDL(A_{kk})
  for $j = k+1$ to nt **do**
    $L_{jk} = A_{jk} P_{jj} (D_{kk} L_{kk}^T)^{-1}$
  end for
  for $i = k+1$ to nt **do**
    $A_{ii} = A_{ii} - L_{ik} D_{kk} L_{ik}^T$
    for $j = k+1$ to $i - i$ **do**
      $A_{ij} = A_{ij} - L_{ik} D_{kk} L_{jk}^T$
    end for
  end for
  for $i = 1$ to $j - 1$ **do**
    $L_{ki} = P_{kk}^T L_{ki}$
  end for
end for

The permutation matrices P_{kk} of Algorithm 1 are computed during the factorization of tile A_{kk}. If pivots were searched only inside tile A_{ii}, the factorization would depend only and exclusively on A_{kk}. However, for most pivoting techniques, pivots are searched throughout columns, which make the design of efficient parallel algorithm very difficult [18].

The tile-wise pivoting restricts the search of pivots to the tile A_{kk} when factorizing it, i.e. if LAPACK [1] routine `xSYTRF` was chosen to perform the factorization, it could be used as it is. In other words, the same procedure used to factorize an entire matrix A is used to factorize the tile A_{kk}. This approach does not guarantee the accuracy of the solution; it strongly depends on the matrix to be factorized and how the pivots are distributed. However, it guarantees numerical stability of the factorization of each tile A_{kk}, as long as an appropriate pivoting technique is applied. For instance, LDL$^{\mathrm{T}}$ without pivoting fails as soon as a zero is found on the diagonal, while the tile-wise pivoted LDL$^{\mathrm{T}}$ does not, as shown in Section 4. Note that pivoting is applied as part of a sequential kernel, which means that the pivot search and hence the permutations are also serial.

3 An Alternative to Pivoting in Symmetric Indefinite Systems

A randomization technique that allows pivoting to be avoided in the LDL$^{\mathrm{T}}$ factorization is described. This technique was initially proposed in [16] in the context of general linear systems where the randomization is referred to as Random Butterfly Transformation (RBT). Then a modified approach has been described in [5] for the LU factorization of general dense matrices and we propose here to adapt this technique specifically to symmetric indefinite systems. It consists of a multiplicative preconditioning U^TAU where the matrix U is chosen among a particular class of random matrices called *recursive butterfly matrices*. Then $\mathrm{LDL^T}$ factorization without pivoting is performed on the symmetric matrix U^TAU and, to solve $Ax = b$, $(U^TAU)y = U^Tb$ is solved instead, followed by $x = Uy$. We study the random transformation with *recursive* butterfly matrices, and minimize the number of recursion steps required to get a satisfying accuracy. The resulting transformation will be called Symmetric Random Butterfly Transformation (SRBT). We define two types of matrices that will be used in the symmetric random transformation. These definitions are inspired from [16] in the particular case of real-valued matrices.

Definition 1 *A butterfly matrix is defined as any n-by-n matrix of the form:*

$$B = \frac{1}{\sqrt{2}} \begin{pmatrix} R_0 & R_1 \\ R_0 & -R_1 \end{pmatrix}$$

where $n \geq 2$ and R_0 and R_1 are random diagonal and nonsingular $n/2$-by-$n/2$ matrices.

Definition 2 *A recursive butterfly matrix of size n and depth d is a product of the form*

$$W^{<n,d>} = \begin{pmatrix} B_1^{<n/2^{d-1}>} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & B_{2^{d-1}}^{<n/2^{d-1}>} \end{pmatrix} \times \cdots \times$$
$$\begin{pmatrix} B_1^{<n/4>} & 0 & 0 & 0 \\ 0 & B_2^{<n/4>} & 0 & 0 \\ 0 & 0 & B_3^{<n/4>} & 0 \\ 0 & 0 & 0 & B_4^{<n/4>} \end{pmatrix} \times \begin{pmatrix} B_1^{<n/2>} & 0 \\ 0 & B_2^{<n/2>} \end{pmatrix} \times B^{<n>}$$

where $B_i^{<n/2^{k-1}>}$ are butterfly matrices of size $n/2^{k-1}$ with $1 \leq k \leq d$.

Note that this definition requires that n is a multiple of 2^{d-1} which can always be obtained by "augmenting" the matrix A with additional 1's on the diagonal. Note also that Definition 2 differs from the definition of a recursive butterfly matrix given in [16], which corresponds to the special case where $d = \log_2 n + 1$, i.e. the first term of the product expressing $W^{<n,d>}$ is a diagonal matrix of size n.

For instance, if $n = 4$ and $d = 2$, then the recursive butterfly matrix $W^{<4,2>}$ is defined by

$$W^{<4,2>} = \begin{pmatrix} B_1^{<2>} & 0 \\ 0 & B_2^{<2>} \end{pmatrix} \times B^{<4>}$$

$$= \frac{1}{2} \begin{pmatrix} r_1^{<2>} & r_2^{<2>} & 0 & 0 \\ r_1^{<2>} & -r_2^{<2>} & 0 & 0 \\ 0 & 0 & r_3^{<2>} & r_4^{<2>} \\ 0 & 0 & r_3^{<2>} & -r_4^{<2>} \end{pmatrix} \begin{pmatrix} r_1^{<4>} & 0 & r_3^{<4>} & 0 \\ 0 & r_2^{<4>} & 0 & r_4^{<4>} \\ r_1^{<4>} & 0 & -r_3^{<4>} & 0 \\ 0 & r_2^{<4>} & 0 & -r_4^{<4>} \end{pmatrix}$$

$$= \frac{1}{2} \begin{pmatrix} r_1^{<2>}r_1^{<4>} & r_2^{<2>}r_2^{<4>} & r_1^{<2>}r_3^{<4>} & r_2^{<2>}r_4^{<4>} \\ r_1^{<2>}r_1^{<4>} & -r_2^{<2>}r_2^{<4>} & r_1^{<2>}r_3^{<4>} & -r_2^{<2>}r_4^{<4>} \\ r_3^{<2>}r_1^{<4>} & r_4^{<2>}r_2^{<4>} & -r_3^{<2>}r_3^{<4>} & -r_4^{<2>}r_4^{<4>} \\ r_3^{<2>}r_1^{<4>} & -r_4^{<2>}r_2^{<4>} & -r_3^{<2>}r_3^{<4>} & r_4^{<2>}r_4^{<4>} \end{pmatrix},$$

where $r_i^{<j>}$ are real random entries.

The objective here is to minimize the computational cost of the RBT defined in [16] by considering a number of recursions d such that $d \ll n$, resulting in the transformation defined as follows.

Definition 3 *A symmetric random butterfly transformation (SRBT) of depth d of a square matrix A is the product:*

$$A_r = U^T A U$$

where U is a recursive butterfly matrix of depth d.

Remark 1 Let A be a square matrix of size n, the computational cost of a multiplication $B^T A B$ with B butterfly of size n is $M(n) = 4n^2$. Then the number of operations involved in the computation of A_r by an SRBT of depth d is

$$C(n,d) = \sum_{k=1}^{d} \left((2^{k-1})^2 \times M(n/2^{k-1})\right) = \sum_{k=1}^{d} \left((2^{k-1})^2 \times 4(n/2^{k-1})^2\right)$$
$$= \sum_{k=1}^{d} \left(4n^2\right) = 4dn^2$$

Note that the maximum cost in the case of an RBT as described in [16] is

$$C(n, \log_2 n + 1) \simeq 4n^2 \log_2 n.$$

We can find in [16] details on how RBT might affect the growth factor and in [5] more information concerning the practical computation of A_r as well as a packed storage description and a condition number analysis. Note that, since we know that we do not pivot when using SRBT, the $\mathrm{LDL^T}$ factorization without pivoting can be performed with a very efficient tiled algorithm [6].

4 Numerical Experiments

Experiments to measure the accuracy of each procedure described in the previous sections were carried out using Matlab version 7.12 (R2011a) on a machine with a precision of $2.22 \cdot 10^{-16}$. Table 1 presents accuracy comparisons of linear systems solved using the factors of A calculated by $\mathrm{LDL^T}$ with: no pivoting (NP), partial pivoting (PP), tile-wise pivoting (TP), and no pivoting preceded by the Symmetric Random Butterfly Transformation (SRBT).

Table 1: Component-wise backward error for $\mathrm{LDL^T}$ solvers on a set of test matrices.

Matrix	Cond A	NP	PP	TP	SRBT (IR)
`condex`	$1 \cdot 10^{2}$	$5.57 \cdot 10^{-15}$	$6.94 \cdot 10^{-15}$	$7.44 \cdot 10^{-15}$	$6.54 \cdot 10^{-15}$ (0)
`fiedler`	$7 \cdot 10^{5}$	Fail	$2.99 \cdot 10^{-15}$	$7.43 \cdot 10^{-15}$	$9.37 \cdot 10^{-15}$(0)
`orthog`	$1 \cdot 10^{0}$	$8.40 \cdot 10^{-1}$	$1.19 \cdot 10^{-14}$	$5.31 \cdot 10^{-1}$	$3.51 \cdot 10^{-16}$ (1)
`randcorr`	$3 \cdot 10^{3}$	$4.33 \cdot 10^{-16}$	$3.45 \cdot 10^{-16}$	$4.40 \cdot 10^{-16}$	$5.10 \cdot 10^{-16}$ (0)
`augment`	$5 \cdot 10^{4}$	$7.70 \cdot 10^{-15}$	$4.11 \cdot 10^{-15}$	$8.00 \cdot 10^{-15}$	$2.59 \cdot 10^{-16}$ (1)
`prolate`	$6 \cdot 10^{18}$	$8.18 \cdot 10^{-15}$	$8.11 \cdot 10^{-16}$	$2.62 \cdot 10^{-15}$	$2.67 \cdot 10^{-15}$ (0)
`toeppd`	$1 \cdot 10^{7}$	$5.75 \cdot 10^{-16}$	$7.75 \cdot 10^{-16}$	$6.99 \cdot 10^{-16}$	$2.38 \cdot 10^{-16}$ (0)
`ris`	$4 \cdot 10^{0}$	Fail	$3.25 \cdot 10^{-15}$	$8.81 \cdot 10^{-1}$	$6.05 \cdot 10^{-1}$ (10)
$\lvert i-j\rvert$	$7 \cdot 10^{5}$	$2.99 \cdot 10^{-15}$	$2.99 \cdot 10^{-15}$	$7.43 \cdot 10^{-15}$	$1.15 \cdot 10^{-14}$ (0)
`max(i,j)`	$3 \cdot 10^{6}$	$2.35 \cdot 10^{-14}$	$2.06 \cdot 10^{-15}$	$5.08 \cdot 10^{-15}$	$1.13 \cdot 10^{-14}$ (0)
`Hadamard`	$1 \cdot 10^{0}$	$0 \cdot 10^{0}$	$0 \cdot 10^{0}$	$0 \cdot 10^{0}$	$7.29 \cdot 10^{-15}$ (0)
`rand0`	$2 \cdot 10^{5}$	$1.19 \cdot 10^{-12}$	$7.59 \cdot 10^{-14}$	$1.69 \cdot 10^{-13}$	$1.64 \cdot 10^{-15}$ (1)
`rand1`	$2 \cdot 10^{5}$	Fail	$1.11 \cdot 10^{-13}$	$2.07 \cdot 10^{-11}$	$1.77 \cdot 10^{-15}$ (1)
`rand2`	$1 \cdot 10^{5}$	Fail	$5.96 \cdot 10^{-14}$	$6.41 \cdot 10^{-13}$	$1.77 \cdot 10^{-15}$ (1)
`rand3`	$8 \cdot 10^{4}$	$4.69 \cdot 10^{-13}$	$7.60 \cdot 10^{-14}$	$4.07 \cdot 10^{-13}$	$1.92 \cdot 10^{-15}$ (1)

NP: $\mathrm{LDL^T}$ with No Pivoting
PP: $\mathrm{LDL^T}$ with Partial Pivoting
TP: $\mathrm{LDL^T}$ with Tile-wise Pivoting
SRBT: Symmetric Random Butterfly Transformation followed by $\mathrm{LDL^T}$ without pivoting
IR: Number of iterations for iterative refinement in SRBT

The partial pivoting corresponds to the Bunch-Kaufman algorithm as it is implemented in LAPACK. Note that for all experiments the rook pivoting method achieves the same accuracy as the partial pivoting and therefore is not listed.

All matrices are of size 1024×1024, either belonging to the Matlab gallery or the Higham's Matrix Computation Toolbox [14] or generated using Matlab function `rand`. Matrices $|i-j|$, $max(i,j)$ and `Hadamard` are defined in the experiments performed in [16]. Matrices `rand1` and `rand2` correspond to random matrices with entries uniformly distributed in $[0,1]$ with all and $1/4$ of the diagonal elements set to 0, respectively. Matrices `rand0` and `rand4` are also random matrices, where the latter has its diagonal elements scaled by $1/1000$.

For all test matrices, we suppose that the exact solution is $x = [1\,1\ldots 1]$ and we set the right-hand side $b = Ax$. In Table 1, the 2-norm condition number of each matrix is listed. Note that we also computed the condition number of the randomized matrix which, similarly to [5], is of same order of magnitude as cond A and therefore is not listed. For each $\mathrm{LDL^T}$ solver, the component-wise backward error is reported. The latter is defined in [15] and expressed as

$$\omega = \max_i \frac{|A\hat{x} - b|_i}{(|A| \cdot |\hat{x}| + |b|)_i},$$

where $\hat{x}$ is the computed solution.

Similarly to [16], the random diagonal matrices used to generate the butterfly matrices described in Definition 1 have diagonal values $exp(\frac{r}{10})$ where r is randomly chosen in $[-\frac{1}{2}, \frac{1}{2}]$ (matlab instruction `rand`). The number of recursions used in the SRBT algorithm (parameter d in Definition 3) has been set to 2. Hence, the resulting cost of SRBT is $\sim 8n^2$ operations (see Remark 1). To improve the stability, iterative refinement (in the working precision) is added when SRBT is used. Similarly to [2, 17], the iterative refinement algorithm is called while $\omega > (n+1)u$, where u is the machine precision. The number of iterations (IR) in the iterative refinement process is also reported in Table 1.

For all matrices, except `orthog` and `ris` with TP and `ris` with SRBT, the factorization with both tile-wise pivoting and randomization achieves satisfactory results. Iterative refinement turns out to be necessary in a few cases when using SRBT but with never more than one iteration (except for `ris` for which neither TP nor SRBT have achieved accurate results). For matrix `prolate`, all methods result in a small backward error. However, the solution cannot be accurate at all due to the large condition number. Note that when matrices are orthogonal (`orthog`) or proportional to an orthogonal matrix (`Hadamard`), $\mathrm{LDL^T}$ must not be used. Also, `toeppd` is positive definite and would normally be solved by Cholesky and not $\mathrm{LDL^T}$. These three test cases have been used only for testing purposes. In the case of the integer-valued matrix `Hadamard` , SRBT destroys the integer structure and transforms the initial matrix into a real-valued one. For the four random matrices, TP achieves results slightly less accurate than SRBT. However, in these cases iterative refinement added to TP would enable us to achieve an accuracy similar to SRBT.

TP and SRBT are always more accurate than NP but they both failed to produce results as accurate as PP for at least one of the test matrices. Nevertheless, despite the reduced number of test cases, they cover a reasonable range of matrices, including those with zeros on the diagonal. Test case `rand1` has only zeros on the diagonal and was accurately solved by both techniques. This case fails at the very first step of the $\mathrm{LDL^T}$ method without pivoting. Test case `orthog` has been solved accurately with SRBT but not with TP. For this particular case, when the pivot search is applied on the full matrix, rows/columns 1 and n are permuted, then rows/columns 2 and $n-1$ are permuted, and so forth. In others, the pivots are spread far apart and the tile-wise pivoting cannot reach them, i.e. there are not *good enough* pivots within each tile.

5 Conclusion and Future Work

A tiled $\mathrm{LDL^T}$ factorization with tile-wise pivoting and a randomization technique to avoid pivoting in the $\mathrm{LDL^T}$ factorization have been presented. The tile-wise pivoting consists of choosing a pivoting strategy and restraining the pivot search to the tile being factored. The randomization technique, called Symmetric Random Butterfly Transformation (SRBT), involves a multiplicative preconditioning which is computationally very affordable and negligible compared to the communication overhead due to classical pivoting algorithms.

Both techniques give accurate results on most test cases considered in this paper, including pathological ones. However, further development of the tile-wise pivoting is required in order to increase its robustness. In particular, techniques such as search by pairs of tiles, also called incremental pivoting, have to be investigated for symmetric indefinite factorizations. Also, to improve stability, the solution obtained after randomization should be systematically followed by iterative refinement in fixed precision (one iteration is sufficient in general). The algorithms presented in this paper shall be integrated into PLASMA, which will allow performance comparisons of the $\mathrm{LDL^T}$ solvers and more extensive testing using the matrices available as part of LAPACK.

References

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK User's Guide*. SIAM, 1999. Third edition.

[2] M. Arioli, J. W. Demmel, and I. S. Duff. Solving sparse linear systems with sparse backward error. *SIAM J. Matrix Anal. and Appl.*, 10(2):165–190, 1989.

[3] C. Ashcraft, R. G. Grimes, and J. G. Lewis. Accurate symmetric indefinite linear equation solvers. *SIAM J. Matrix Anal. and Appl.*, 20(2):513–561, 1998.

[4] M. Baboulin, J. Dongarra, and S. Tomov. Some issues in dense linear algebra for multicore and special purpose architectures. In *Proceedings of the 9th International Workshop on State-of-the-Art in Scientific and Parallel Computing (PARA'08)*.

[5] M. Baboulin, J. Dongarra, J. Herrmann, and S. Tomov. Accelerating linear system solutions using randomization techniques. *Lapack Working Note 246* and *INRIA Research Report 7616*, May 2011.

[6] D. Becker, M. Faverge, and J. Dongarra. Towards a Parallel Tile LDL Factorization for Multicore Architectures. Technical Report ICL-UT-11-03, Innovative Computing Laboratory, University of Tennessee, Knoxville, TN, USA, April 2011.

[7] Å. Björck. *Numerical Methods for Least Squares Problems*. Society for Industrial and Applied Mathematics, 1996.

[8] J. R. Bunch and L. Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Math. Comput.*, 31:163–179, 1977.

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

[10] A. Buttari, J. Langou, J. Kurzak, and J. Dongarra. Parallel tiled QR factorization for multicore architectures. *Concurrency Computat.: Pract. Exper.*, 20(13):1573–1590, 2008.

[11] A. Buttari, J. Langou, J. Kurzak, and J. Dongarra. A class of parallel tiled linear algebra algorithms for multicore architectures. *Parallel Comput. Syst. Appl.*, 35:38–53, 2009.

[12] J. Dongarra, J. Kurzak, J. Langou, J. Langou, H. Ltaief, P. Luszczek, A. YarKhan, W. Alvaro, M. Faverge, A. Haidar, J. Hoffman, E. Agullo, A. Buttari, and B. Hadri. PLASMA Users' Guide, Version 2.3. Technical Report, Electrical Engineering and Computer Science Department, Univesity of Tennessee, Knoxville, TN, Sep 2010.

[13] G. H. Golub and C. F. van Loan. *Matrix Computations*. The Johns Hopkins University Press, 1996. Third edition.

[14] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, 2002. Second edition.

[15] W. Oettli and W. Prager. Compatibility of approximate solution of linear equations with given error bounds for coefficients and right-hand sides. *Numerische Mathematik*, 6:405–409, 1964.

[16] D. S. Parker. Random butterfly transformations with applications in computational linear algebra. Technical Report CSD-950023, Computer Science Department, UCLA, 1995.

[17] R. D. Skeel. Iterative refinement implies numerical stability for Gaussian elimination. *Math. Comput.*, 35:817–832, 1980.

[18] P. E. Strazdins. Issues in the design of scalable out-of-core dense symmetric indefinite factorization algorithms. In *Proceedings of the 2003 international conference on computational science: PartIII*, ICCS'03, pages 715–724, Springer-Verlag.

[19] H. Sutter. The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software. *Dr. Dobb's Journal*, 30(3), 2005.

Centre de recherche INRIA Saclay – Île-de-France
Parc Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 Orsay Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399