Experience with a Recursive Perturbation Based Algorithm for Symmetric Indefinite Linear Systems[⋆]

Anshul Gupta[1], Fred Gustavson[1], Alexander Karaivanov[2], Jerzy Wasniewski[2], and Plamen Yalamov[3]

[1] IBM Watson Research Center, P. O. Box 218, Yorktown Heights, NY 10598, USA, fax: + 1 914 945 3434, `anshul@watson.ibm.com, gustav@watson.ibm.com`
[2] UNI•C, Building 304 - DTU, DK-2800 Lyngby, Denmark, fax: + 45 3587 8990, `alex@uni-c.dk, unijw@uni-c.dk`
[3] University of Rousse, 7017 Rousse, Bulgaria, fax: + 35 982 455 145, `yalamov@ami.ru.acad.bg`

Abstract. We consider recursive algorithms for symmetric indefinite linear systems. First, the difficulties with the recursive formulation of the LAPACK SYSV algorithm (which implements the Bunch-Kaufman pivoting strategy) are discussed. Next a recursive perturbation based algorithm is proposed and tested. Our experiments show that the new algorithm can be about two times faster although performing about the same number of flops as the LAPACK algorithm.

1 Introduction

Recursive algorithms for dense linear algebra problems are proposed and studied in [1, 2, 7]. It is shown that recursion leads to better performance on modern processors. Also, the codes using recursion are very simple, and easy to write in languages that support recursion (e. g. Fortran90).

In [1, 2, 5, 7] recursion is applied to three widely used algorithms, the LU and QR decompositions for general dense matrices, and the Cholesky decomposition for symmetric and positive definite matrices. In the present work we discuss the recursive approach to the decomposition of symmetric but indefinite matrices. It is well-known that such matrices require pivoting as the decomposition algorithms can be unstable, or can break down, even for well-conditioned matrices. Therefore, in LAPACK [3], the method Bunch-Kaufman pivoting is applied. As we will see in the next section the same type of pivoting is possible to apply in the recursive algorithm but doing so makes the algorithm more complicated and time consuming.

In practice there are different approaches to avoid the break down of accuracy in practice. In this paper we propose and test a perturbation approach; i. e., we

[⋆] This research is supported by the UNI•C collaboration with the IBM T.J. Watson Research Center at Yorktown Heights. The last author was partially supported by the Grant I-702/97 from the Bulgarian Ministry of Education and Science.

perturb pivot elements whenever they are small. In this way we move them away from zero, and improve the stability to some extent. Because of the perturbation the obtained decomposition is only accurate to a few digits. Nevertheless, it can be used for the solution of linear systems by adding 1-2 steps of iterative refinement. The cost of iterative refinement is tiny because it only uses triangular solves. Previously, the same approach has been applied to other types of matrices in [4] and [8], for example.

When the matrix of the problem is kept in full storage (like in _SYSV of LAPACK) our algorithm does not need additional memory. At the same time it can be up to three times faster than the corresponding LAPACK subroutine _SYSV.

The outline of the paper is as follows. In Section 2 we explain the difficulties with the recursive algorithm that uses Bunch-Kaufman pivoting. Then in Section 3 the algorithm with the perturbation approach is given. Numerical experience is presented and discussed in Section 4.

2 Recursive Factorization with Pivoting

It is well-known that the Cholesky factorization can fail for symmetric indefinite matrices. In this case some pivoting strategy can be applied (e. g. the Bunch-Kaufman pivoting [6, §4.4]). The algorithm can be given as follows.

```
**LDL$^T$ factorization**
$L = I$ (identity matrix); $k = 1$;
**while** $(k < n)$
    Apply pivoting: choose a $1 \times 1 (s = 0)$, or $2 \times 2 (s = 1)$ pivot block,
    and exchange the corresponding rows and columns;
    $E = A_{k:k+s,k:k+s}$
    $C = A_{k+s+1:n,k:k+s}$
    $B = A_{k+s+1:n,k+s+1:n}$
    $L_{k+s+1:n,k:k+s} = CE^{-1}$
    $D_{k:k+s,k:k+s} = E$
    $A_{k+s+1:n,k+s+1:n} = B - CE^{-1}C^T$
    k = k+s+1
**end**
```

As a result we get

$$PAP^T = LDL^T,$$

where L is unit lower triangular, D is block diagonal with 1×1, or 2×2 blocks, and P is a permutation matrix.

Now let us look at the recursive formulation of this algorithm. This is given below. The recursion is done on the second dimension of matrix A; i. e., the algorithm works on full columns as in LU factorization.

```
**Recursive Symmetric Indefinite Factorization (RSIF)** of $A_{1:m,1:n}$
$k = 1$
**if** $(n = 1)$
  Define the pivot: 1×1, or 2×2.
  Apply interchanges if necessary
  $k = k + 1$, or $k = k + 2$
  If the pivot is 2×2: FLAG=1
**else**
  $n1 = n/2$
  $n2 = n - n1$
  **RSIF** of $A_{:,k:k+n1-1}$
  **if** (FLAG = 1)
    $n1 = n1 + 1$
    $n2 = n - n1$
  **end**
  **update** $A_{:,k:k+n2-1}$
  **RSIF** of $A_{:,k:k+n1-1}$
**end**
```

Since matrix A is square, we must set $m = n$ when we first call RSIF. The advantage of the recursive formulation is that the updating step is a matrix-matrix operation, and BLAS Level 3 subroutine can be used. Thus if the algorithm is properly implemented some speedup can be expected from this formulation. But this does not occur.

The fact that the recursive algorithm does not fully update the lower right part of A forces us to incorporate updating and downdating of the exchanged by the pivoting strategy columns. Such a step may bring more computation depending on the position of the columns. Additionally, the computation must be done as a Level 2 computation as we are updating and downdating single columns. To summarize: the application of the Bunch-Kaufman pivoting strategy forces one to do Level 2 computations (in some cases undoing a previously done computation at a Level 3 rate).

3 Perturbation Approach

An alternative to pivoting is our perturbation approach. It is applied in cases when there can be a large growth of elements (or breakdown), and pivoting is not desirable for some reason. The reason for avoiding the Bunch-Kaufman pivoting is that the performance suffers; this was illustrated in Section 2. This approach is applied in [4] to a parallel algorithm (where pivoting is not desirable because it adds more communication between the processors), and in [8] to an algorithm for inversion of Toeplitz matrices (where pivoting spoils the Toeplitz structure, and slows down the algorithm).

The idea of the perturbation approach is simple. Usually, growth of elements (or breakdown) happens when we pivot with a small number (or zero). Therefore, we add a small number δ to each pivot a,

$$a = a + \mathrm{Sgn}(a)\delta, \quad \mathrm{Sgn}(a) = \begin{cases} \mathrm{sign}(a), & a \neq 0, \\ 1, & a = 0, \end{cases}$$

in case $|a|$ is small, more precisely, $|a| < \delta$.

With this approach the factorization for symmetric indefinite matrices looks as follows:

Perturbed Recursive Symmetric Indefinite Factorization (PRSIF)

```
of $A(1:n, 1:n)$
**if** $(n = 1)$
  **if** $(|A_{1,1}| < \delta)$
    $A_{1,1} = A_{1,1} + \mathrm{Sgn}(A_{1,1})\delta$
  **end**
  $D_{1,1} = A_{1,1}$
**else**
  $p = n/2$
  **PRSIF** of $A_{1:p,1:p} = L_1 D_1 L_1^T$
  **solve** $X D_1 L_1^T = A_{p+1:n,1:p}$ for $X$
  **update** $A_{p+1:n,p+1:n} = A_{p+1:n,p+1:n} - X D_1 X^T$
  **PRSIF** of $A_{p+1:n,p+1:n} = L_2 D_2 L_2^T$
**end**
```

Let us note that the algorithm can be easily modified so that the case $n = 1$ is changed to $n \leq n_0$. In such a case we do not go to the deepest level of recursion, and decompose the block of size $\leq n_0$ with some appropriate algorithm. The numerical experiments at the end of the paper are done in this way (with $n_0 = 64$ which is the best choice for our architecture).

As a result of this algorithm we get $\tilde{A} = \tilde{L}\tilde{D}\tilde{L}^T$, where $\tilde{D}$ is diagonal, $\tilde{L}$ is unit lower triangular, and we put a tilde because of the perturbations.

Because of adding perturbations it is possible that we might change the input matrix A dramatically. So, the question is, how much does LDL^T change when we add perturbations. To answer this question we consider the product $LDL^T = A$. For any diagonal entry we have

$$A_{i,i} = D_{i,i} + \sum_{j=1}^{i-1} L_{i,j}^2. \tag{1}$$

If a perturbation is necessary then we add $\pm\delta$ to $D_{i,i}$, and we have

$$\tilde{A}_{i,i} = D_{i,i} \pm \delta + \sum_{j=1}^{i-1} L_{i,j}^2. \tag{2}$$

By comparing (1) and (2) we see that

$$\tilde{A}_{i,i} = A_{i,i} \pm \delta.$$

For the whole algorithm we operate not on the original matrix A but on $\tilde{A}$, where

$$\tilde{A} = A + \Delta A, \quad |\Delta A| \leq \delta I, \tag{3}$$

i. e. $\varDelta A$ is diagonal. In the example problems we chose 1-2 perturbations seemed to be enough, so, only 1-2 entries of $\varDelta A$ were nonzero and equal to $\pm\delta$. Thus if δ is small the changes in A are small, too. If A is well-conditioned this will lead to small changes in A^{-1} because we are only doing 1–2 small rank corrections. So, the perturbation approach is expected to work well for well-conditioned matrices.

Because of the changes in A the LDL^T factorization is no longer accurate but we can use this factorization for solution of linear systems $AX = B$. The idea is to apply iterative refinement [6, §3.5.3]:

```
**solve** $\tilde{A}X^{(0)} = B$;
**for** $k = 1, 2, \ldots$ until convergence do
    $R^{(k-1)} = B - AX^{(k-1)}$;
    **solve** $(A + \varDelta A)\tau^{(k)} = R^{(k-1)}$;
    $X^{(k)} = X^{(k-1)} + \tau^{(k)}$;
**end**
```

The perturbed factorization is used when solving the linear systems above. Thus the iterative refinement needs $O(n^2)$ additional operations, and does not essentially increase the total operations count.

For the iterative refinement we need to keep the original matrix A. However, we do not need additional memory (except for one vector of size n where we store the diagonal of A) for this because we assume that matrix A is kept in full storage, and the original matrix A stays untouched by the algorithm in the upper triangle of the array. Thus, the algorithm essentially does not require more memory.

At present the perturbation δ is difficult to estimate theoretically. From our practical experience we suggest that the best value for δ is $\delta = \sqrt{\rho_0}$, where ρ_0 is the machine roundoff unit. In a Cholesky factorization $A = LL^T$ and $L_{ii} = \sqrt{\ldots}$. This analogy suggests why $\sqrt{\rho_0}$ works. With this value of δ 1-2 iterative refinement steps usually produce satisfactory results. In case δ is not chosen properly, we will have a large residual R. Since we compute the residual explicitly, we can notice such a situation, and produce a warning message.

4 Numerical Experiments

The experiments are produced on an IBM SMP node which has four CPUs (we use all of them in the experiments). The codes are written in Fortran90. We use double precision (roundoff unit $\approx$ 2.22E-16). Thus, our choice is δ = 1E-8. We compare our PRSIF algorithm with iterative refinement (denoted by RPSYSV) to the LAPACK SYSV algorithm. In order to see the advantage of the recursive algorithm we coded also a version of the SYSV algorithm in which the pivoting part was replaced by a perturbation part in the same way as for RPSYSV. This algorithm is denoted by PSYSV. This means that in PSYSV we also use iterative refinement. The number of iterative refinement steps in RPSYSV and PSYSV was fixed to be 1. Since IR doubles the number of digits and the precision we work with is approximately $\delta = 1e - 8$ the accuracy we obtain in our examples

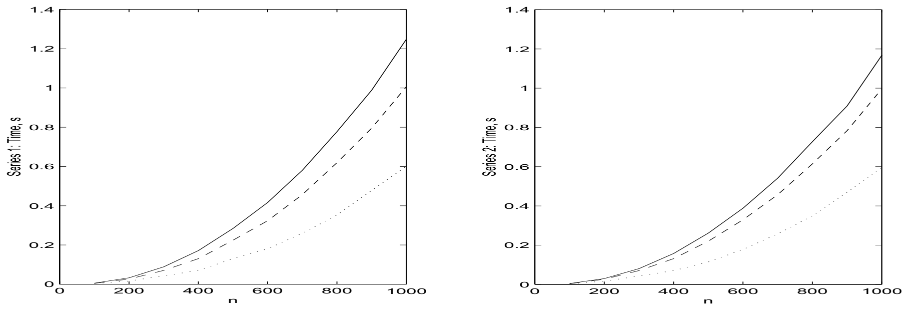

Fig. 1. Timing results: SYSV(-), PSYSV(--), RPSYSV(···)

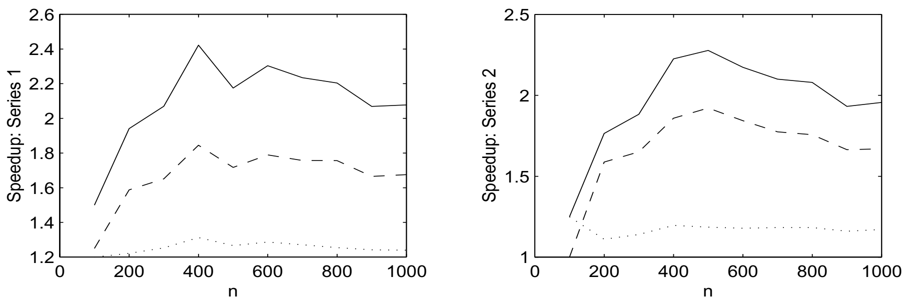

Fig. 2. Speedup results: SYSV/RPSYSV(-), PSYSV/RPSYSV(--), SYSV/PSYSV(···)

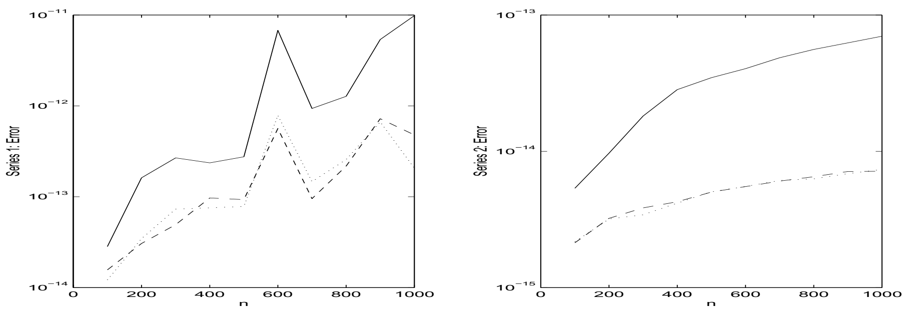

Fig. 3. Error results: SYSV(-), PSYSV(--), RPSYSV(···)

is close to $1e-16$. We present results for the time, the speedup, and the forward relative error $\|\tilde{x}-x\|_\infty/\|x\|_\infty$ of the solution x for the three algorithms.

In the first series of experiments (denoted by Series 1) a random matrix is generated by the LAPACK subroutine DLAGGE.

In the second series of experiments (denoted by Series 2) matrix A is of the following type[1]):

$$A = \begin{pmatrix} \Delta & C \\ C^T & I \end{pmatrix},$$

where Δ is diagonal with small entries, C has random elements with large entries, and I is the identity. The entries of Δ are chosen to be less than δ. In this way the PRSIF algorithm is forced to make perturbations.

The results are presented in Fig. 1–3 for different sizes of matrix A.

The experiments show that 1) the recursive algorithm (in the best comparison) is about two times faster on average although performing almost the same number of flops as the LAPACK subroutine, and 2) the error in the recursive algorithm is similar to the error produced by the LAPACK subroutine.

As we mentioned the recursion is stopped when $n_0 = 64$. We tested also other choices of n_0. The performance degrades when decreasing or increasing n_0. The change is slight when we choose values of n_0 close to 64 (e. g. 56,60,68, or 72). When n_0 is significantly different (e. g. 8, or 200) the performance can be much worse. The corresponding blocking factor for LAPACK is chosen to be 32. This is the best choice (with highest performance) for most matrix sizes on our architecture.

Let us note that essentially we have two types of block operations in Algorithm PRSIF: 1) triangular solves with L_1, and 2) updating $A_{p+1:n,p+1:n}$. For the first operation we use the ESSL BLAS-3 subroutine which is compiled for the four CPUs. The second block operation is implemented by ourselves because there is no appropriate BLAS operation for symmetric matrices (this operation is included in the next version of BLAS). In the LAPACK subroutine the ESSL BLAS-3 routines are used wherever possible. The difference between the recursive and LAPACK algorithm is that the recursive algorithm works on larger and larger blocks while the LAPACK algorithm works on blocks with a fixed size. As a result the advantage of BLAS operations is better utilized by the recursive algorithm.

The performance of PSYSV is better than SYSV because perturbation is used instead if row and column exchanges in the Bunch-Kaufman pivoting which are slower. But the experiments show that RPSYSV has a significantly better performance than SYSV and PSYSV. The influence of recursion on the performance is especially illustrated by the difference between PSYSV and RPSYSV where the only difference is the recursion.

The timing behavior of RPSYSV is quite promising. At present we do not have theory about the application of the perturbation idea, and so we need to do more research to justify when its application will give accurate results.

[1] Suggested to us by John Reid.

References

[1] Andersen, B., Gustavson, F., Waśniewski, J.: A recursive formulation of the Cholesky factorization operating on a matrix in packed storage form, in. Parallel Processing for Scientific Computing, Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing, San Antonio, TX , USA, March 24-27, 1999

[2] Andersen, B., Gustavson, F., Waśniewski, J., Yalamov, P.: Recursive formulation of some dense linear algebra algorithms, in. Parallel Processing for Scientific Computing, Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing, San Antonio, TX , USA, March 24-27, 1999

[3] Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., Sorensen, D.: LAPACK Users' Guide Release 2.0. SIAM, Philadelphia, 1995

[4] Balle, S., Hansen, P.: A Strassen-type matrix inversion algorithm for the Connection Machine. Report UNIC-93-11, October 1993

[5] Elmroth, E., Gustavson, F.: New serial and parallel recursive QR factorization algorithms for SMP systems. In: Applied Parallel Computing, (Eds. B. Kågstrm̈ et. al.), Lecture Notes in Computers Science, v. 1541, Springer, 1998.

[6] Golub, G., Van Loan, C.: Matrix Computations, 3rd edition. John Hopkins University Press, Baltimore, 1996

[7] Gustavson, F.: Recursion leads to automatic variable blocking for dense linear-algebra algorithms. IBM J. Res. Develop. 41(1997), pp. 737–755

[8] Hansen, P., Yalamov, P.: Stabilization by perturbation of a $4n^2$ Toeplitz solver. Preprint N25, Technical University of Russe, January 1995