Rectangular Full Packed Format for Cholesky's Algorithm: Factorization, Solution, and Inversion

FRED G. GUSTAVSON
Umeå University and IBM T.J. Watson Research Center (Emeritus)
JERZY WAŚNIEWSKI
Technical University of Denmark
JACK J. DONGARRA
University of Tennessee, Oak Ridge National Laboratory, and University of Manchester
and
JULIEN LANGOU
University of Colorado Denver

We describe a new data format for storing triangular, symmetric, and Hermitian matrices called *Rectangular Full Packed Format* (RFPF). The standard two-dimensional arrays of Fortran and C (also known as *full format*) that are used to represent triangular and symmetric matrices waste nearly half of the storage space but provide high performance via the use of Level 3 BLAS. Standard packed format arrays fully utilize storage (array space) but provide low performance as there is no Level 3 packed BLAS. We combine the good features of packed and full storage using RFPF to obtain high performance via using Level 3 BLAS as RFPF is a standard full-format representation. Also, RFPF requires exactly the same minimal storage as packed the format. Each LAPACK full and/or packed triangular, symmetric, and Hermitian routine becomes a single new RFPF routine based on eight possible data layouts of RFPF. This new RFPF routine usually consists of two calls to the corresponding LAPACK full-format routine and two calls to Level 3 BLAS routines. This means *no* new software is required. As examples, we present LAPACK routines for Cholesky factorization,

J. Langou's research was supported by the National Science Foundation grant no. NSF CCF-811520.
Authors' addresses: F. G. Gustavson, IBM T.J. Watson Research Center, Yorktown Heights, NY 10598; email: fg2935@hotmail.com; J. Waśniewski, Department of Informatics and Mathematical Modelling, Technical University of Denmark, Richard Petersens Plads, Building 321, DK-2800 Kongens Lyngby, Denmark; email: jw@imm.dtu.dk; J.J. Dongarra, Electrical Engineering and Computer Science Department, University of Tennessee, 1122 Volunteer Blvd, Knoxville, TN 37996-3450; email: dongarra@eecs.utk.edu; J. Langou, Department of Mathematical and Statistical Sciences, University of Colorado Denver, 1250, Fourteenth Street, Room 646, Denver, CO 80202; email: julien.langou@ucdenver.edu.

DOI 10.1145/1731022.1731028 http://doi.acm.org/10.1145/1731022.1731028

Cholesky solution, and Cholesky inverse computation in RFPF to illustrate this new work and to describe its performance on several commonly used computer platforms. Performance of LAPACK full routines using RFPF versus LAPACK full routines using the standard format for both serial and SMP parallel processing is about the same while using half the storage. Performance gains are roughly one to a factor of 43 for serial and one to a factor of 97 for SMP parallel times faster using vendor LAPACK full routines with RFPF than with using vendor and/or reference packed routines.

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra—*Linear systems (direct and interactive methods)*; G.4 [**Mathematical Software**]

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Real symmetric matrices, complex Hermitian matrices, positive definite matrices, Cholesky factorization and solution, recursive algorithms, novel packed matrix data structures, LAPACK, Rectangular Full Packed Format, BLAS, linear algebra libraries

ACM Reference Format:

Gustavson, F. G., Waśniewski, J., Dongarra, J. J., and Langou, J. 2010. Rectangular full packed format for Cholesky's algorithm: Factorization, solution and inversion. ACM Trans. Math. Softw. 37, 2, Article 18 (April 2010), 21 pages.
DOI = 10.1145/1731022.1731028 http://doi.acm.org/10.1145/1731022.1731028

1. INTRODUCTION

A very important class of linear algebra problems deals with a coefficient matrix A that is symmetric and positive definite [Dongarra et al. 1998; Demmel 1997; Golub and Van Loan 1996; Trefethen and Bau 1997]. Because of symmetry, it is only necessary to store either the upper or lower triangular part of the matrix A.

1.1 LAPACK Full and Packed Storage Formats

The LAPACK library [Anderson et al. 1999] offers two different kinds of subroutines to solve the same problem: POTRF[1] and PPTRF both factorize symmetric, positive definite matrices by means of the Cholesky algorithm. A major difference in these two routines is the way they access the array holding the triangular matrix (see Figures 1 and 2).

In the POTRF case, the matrix is stored in one of the lower left or upper right triangles of a full square matrix (Anderson et al. [1999, pages 139 and 140], and IBM [1997], page 64), and[2] the other triangle is wasted (see Figure 1). Because of the uniform storage scheme, blocked LAPACK and Level 3 BLAS subroutines [Dongarra et al. 1990a, 1990b] can be employed, resulting in a fast solution.

In the PPTRF case, the matrix is stored in *packed* storage (Anderson et al. [1999], pages 140 and 141; Agarwal et al. [1994], and IBM [1997], pages 74 and

[1]Four names SPOTRF, DPOTRF, CPOTRF, and ZPOTRF are used in LAPACK for real symmetric and complex Hermitian matrices [Anderson et al. 1999], where the first character indicates the precision and arithmetic versions: S—single precision, D—double precision, C—complex and Z—double complex. LAPACK95 uses one name LA_POTRF for all versions [Barker et al. 2001]. In this article, POTRF and/or PPTRF express any precision, any arithmetic, and any language version of the PO and/or PP matrix factorization algorithms.
[2]In Fortran column major, in C row major.

Lower triangular case Upper triangular case

$$\begin{pmatrix} 1 & & & & & & \\ 2 & 9 & & & & & \\ 3 & 10 & 17 & & & & \\ 4 & 11 & 18 & 25 & & & \\ 5 & 12 & 19 & 26 & 33 & & \\ 6 & 13 & 20 & 27 & 34 & 41 & \\ 7 & 14 & 21 & 28 & 35 & 42 & 49 \end{pmatrix} \quad \begin{pmatrix} 1 & 8 & 15 & 22 & 29 & 36 & 43 \\ & 9 & 16 & 23 & 30 & 37 & 44 \\ & & 17 & 24 & 31 & 38 & 45 \\ & & & 25 & 32 & 39 & 46 \\ & & & & 33 & 40 & 47 \\ & & & & & 41 & 48 \\ & & & & & & 49 \end{pmatrix}$$

Fig. 1. The *full* format array layout of an order N symmetric matrix required by LAPACK.

Lower triangular case Upper triangular case

$$\begin{pmatrix} 1 & & & & & & \\ 2 & 8 & & & & & \\ 3 & 9 & 14 & & & & \\ 4 & 10 & 15 & 19 & & & \\ 5 & 11 & 16 & 20 & 23 & & \\ 6 & 12 & 17 & 21 & 24 & 26 & \\ 7 & 13 & 18 & 22 & 25 & 27 & 28 \end{pmatrix} \quad \begin{pmatrix} 1 & 2 & 4 & 7 & 11 & 16 & 22 \\ & 3 & 5 & 8 & 12 & 17 & 23 \\ & & 6 & 9 & 13 & 18 & 24 \\ & & & 10 & 14 & 19 & 25 \\ & & & & 15 & 20 & 26 \\ & & & & & 21 & 27 \\ & & & & & & 28 \end{pmatrix}$$

Fig. 2. The *packed* format array layout of an order 7 symmetric matrix required by LAPACK.

75), which means that the columns of the lower or upper triangle are stored consecutively in a one-dimensional array (see Figure 2). Now the triangular matrix occupies the strictly necessary storage space but the nonuniform storage scheme means that use of full storage BLAS is impossible and only the Level 2 BLAS packed subroutines [Lawson et al. 1979; Dongarra et al. 1988] can be employed, resulting in a slow solution.

To summarize: LAPACK offers a choice between high performance and wasting half of the memory space (POTRF) versus low performance with optimal memory space (PPTRF).

1.2 Packed Minimal Storage Data Formats Related to RFPF

Recently many new data formats for matrices have been introduced for improving the performance of dense linear algebra (DLA) algorithms.

1.2.1 *Recursive Packed Format (RPF).* [Andersen et al. 2001, 2002]. A new compact way to store a triangular, symmetric or Hermitian matrix called the *Recursive Packed Format* was described in Andersen et al. [2001], as was novel ways to transform RPF to and from standard packed format. New algorithms, called *Recursive Packed Cholesky* (RPC) [Andersen et al. 2001, 2002], that operate on the RPF format were presented. The RPF format operates almost entirely by calling Level 3 BLAS GEMM [Dongarra et al. 1990a, 1990b] but requires variants of algorithms TRSM and SYRK [Dongarra et al. 1990a, 1990b] that are designed to work on RPF. The authors called these algorithms *RPTRSM* and *RPSYRK* [Andersen et al. 2001] and found that they do most of their FLOPS by calling GEMM [Dongarra et al. 1990a, 1990b]. It follows that almost all of the execution time of the RPC algorithm is done in calls to GEMM.

There are three advantages of this storage scheme compared to traditional packed and full storage. First, the RPF storage format uses the minimum amount of storage required for symmetric, triangular, or Hermitian matrices. Second, the RPC algorithm is a Level 3 implementation of Cholesky factorization. Finally, RPF requires no block size tuning parameter. A disadvantage of the RPC algorithm was that it has a high recursive calling overhead as it uses natural recursion or a block size of 1. Gustavson and Jonsson [2000] removed this overhead and added other novel features to the RPC algorithm.

1.2.2 *Square Block Packed Format (SBPF) [Gustavson 2003].* SBPF was described in Section 4 of Gustavson [2003]. A strong point of SBPF is that it requires minimum block storage and all its blocks are contiguous and of equal size. If one uses SBPF with kernel routines then data copying is mostly eliminated during Cholesky factorization.

1.2.3 *Block Packed Hybrid Format (BPHF) [Andersen et al. 2005; Gustavson et al. 2007b].* The authors considered an efficient implementation of the Cholesky solution of symmetric positive-definite full linear systems of equations using packed storage. The authors took the same starting point as that of LINPACK [Dongarra et al. 1979] and LAPACK [Anderson et al. 1999], with the upper (or lower) triangular part of the matrix being stored by columns. Following LINPACK [Dongarra et al. 1979] and LAPACK [Anderson et al. 1999], the authors overwrote the given matrix by its Cholesky factor. Andersen et al. [2005] used the BPHF where blocks of the matrix are held contiguously. The article compared BPHF versus conventional full format storage, packed format, and the RPF for the algorithms. BPHF is a variant of SBPF in which the diagonal blocks are stored in packed format and so its storage requirement is equal to that of packed storage.

We mention that for packed matrices SBPF and BPHF have become the format of choice for multicore processors when one stores the blocks in register block format [Gustavson et al. 2007]. Recently, there have been many article published on new algorithms for multicore processors. This literature is extensive. So we only mention two projects, PLASMA [Buttari et al. 2009] and FLAME [Chan et al. 2007], and refer the interested reader to the literature for additional references.

In regard to other references on new data structures, the survey article by Elmroth et al. [2004] gives an excellent overview. However, since 2005 at least two new data formats for Cholesky type factorizations have emerged, one in Herrero [2006] and RFPF in Gustavson and Waśniewski [2007]. RFPF is the subject matter of this article. In the next subsection, we highlight its main features.

1.3 A Novel Way of Representing Triangular, Symmetric, and Hermitian Matrices in LAPACK

LAPACK has two types of subroutines for triangular, symmetric, and Hermitian matrices called the *packed* and *full format* routines. LAPACK has about 300 of these kind of subroutines. So, in either format, a variety of problems can be solved by these LAPACK subroutines. From a user point of view, RFPF can replace both these LAPACK data formats. Furthermore, and this is important, using RFPF does not require any new LAPACK subroutines to be written. Using RFPF in LAPACK only requires the use of already existing LAPACK and BLAS routines. RFPF strongly relies on the existence of the BLAS and LAPACK routines for the full storage format.

1.4 Overview of the Article

First we introduce the RFPF in general, in Section 2. Second, we show how to use RFPF on symmetric and Hermitian positive definite matrices, for example, for the factorization (Section 3), solution (Section 4), and inversion (Section 5) of these matrices. Section 6 describes LAPACK subroutines for the Cholesky factorization, Cholesky solution, and Cholesky inversion of symmetric and Hermitian positive definite matrices using RFPF. Section 7 indicates that the stability results of using RFPF is unaffected by this format choice as RFPF uses existing LAPACK algorithms which are already known to be stable. Section 8 describes a variety of performance results on commonly used platforms both for serial and parallel SMP executions. These results show that the performance of LAPACK full routines using RFPF versus LAPACK full routines using standard format for both serial and SMP parallel processing is about the same while using half the storage. Also, performance gains are roughly 1 to a factor of 43 for serial and 1 to a factor of 97 for SMP parallel times faster using vendor LAPACK full routines with RFPF than with using vendor and/or reference packed routines. Section 9 explains how some new RFPF routines have been integrated in LAPACK. LAPACK software for the Cholesky algorithm (factorization, solution and inversion) using RFPF was released with LAPACK Version 3.2 in November 2008. Section 10 gives a short summary and brief conclusions.

2. DESCRIPTION OF RECTANGULAR FULL PACKED FORMAT

We describe Rectangular Full Packed Format (RFPF). It transforms a standard Packed Array `AP` of size $NT = N(N+1)/2$ to a full two-dimensional (2D) array whose size is also NT. This means that the performance of LAPACK's [Anderson et al. 1999] packed format routines becomes equal to or better than their full array counterparts. RFPF is a rearrangement of a standard full format rectangular array `SA` of size `LDA*N` where `LDA` $\geq N$. `LDA` is the *leading dimension* of the rectangular array `SA`. For more explanations on the leading dimension and its use, we refer the reader to the LAPACK users' guide [Anderson et al. 1999]. Array `SA` holds a triangular part of a symmetric, triangular, or Hermitian matrix A of order N. The rearrangement of array `SA` is equal to a compact full format rectangular array `AR` of size `LDA1` $*$ `N1` $= NT$, and hence array `AR` like array `AP` uses minimal storage. (The specific values of `LDA1` and `N1` can vary depending on various cases and they will be specified later in this text.) Array `AR` will hold a full rectangular matrix A_R obtained from a triangle of matrix A. Note also that the transpose of the rectangular matrix A_R^T resides in the transpose of array `AR` and hence also represents A. Therefore, Level 3 BLAS [Dongarra et al. 1990a,

A of LAPACK full data format
LDA=N $= 7$, memory needed
LDA $\times$ N $= 49$

$$\left(\begin{array}{cccc|ccc}
a_{1,1} & \diamond & \diamond & \diamond & \diamond & \diamond & \diamond \\
a_{2,1} & a_{2,2} & \diamond & \diamond & \diamond & \diamond & \diamond \\
a_{3,1} & a_{3,2} & a_{3,3} & \diamond & \diamond & \diamond & \diamond \\
a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & \diamond & \diamond & \diamond \\
a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & a_{5,5} & \diamond & \diamond \\
a_{6,1} & a_{6,2} & a_{6,3} & a_{6,4} & a_{6,5} & a_{6,6} & \diamond \\
a_{7,1} & a_{7,2} & a_{7,3} & a_{7,4} & a_{7,5} & a_{7,6} & a_{7,7}
\end{array}\right)$$

Matrix A

A_R of Rectangular full packed

$$\left(\begin{array}{cccc}
a_{1,1} & a_{5,5} & a_{6,5} & a_{7,5} \\
a_{2,1} & a_{2,2} & a_{6,6} & a_{7,6} \\
a_{3,1} & a_{3,2} & a_{3,3} & a_{7,7} \\
a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \\
\hline
a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} \\
a_{6,1} & a_{6,2} & a_{6,3} & a_{6,4} \\
a_{7,1} & a_{7,2} & a_{7,3} & a_{7,4}
\end{array}\right)$$

Matrix A_R

Fig. 3. The Cholesky factorization algorithm using the Rectangular Full Packed Format (RFPF) if N is odd, `uplo` = `'lower'`, and `trans` = `'no transpose'`.

1999b] can be used on array `AR` or its transpose. In fact, with the equivalent LAPACK algorithm which uses the array `AR` or its transpose, the performance is slightly better than the standard LAPACK algorithm which uses the array `SA` or its transpose. Therefore, this offers the possibility of replacing all packed or full LAPACK routines with equivalent LAPACK routines that work on array `AR` or its transpose. For examples of transformations of a matrix A to a matrix A_R see the figures in Section 6.

RFPF is closely related to the HFP format see Gunnels and Gustavson [2004], which represents A as the concatenation of two standard full arrays whose total size is also NT. Let A be an order N symmetric matrix. Break A into a block 2–by–2 form

$$A = \begin{bmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} \text{ or } A = \begin{bmatrix} A_{11} & A_{12} \\ A_{12}^T & A_{22} \end{bmatrix}, \tag{1}$$

where A_{11} and A_{22} are symmetric. Note that we only need to store the lower triangles of A_{11} and A_{22} as well as the full matrix $A_{21} = A_{12}^T$ when we are interested in a lower triangular formulation.

When $N = 2k$ is even, the lower triangle of A_{11} and the upper triangle of A_{22}^T can be concatenated together along their main diagonals into a $(k + 1)$–by–k dense matrix (see the figures where N is even in Section 6). The off-diagonal block A_{21} is k–by–k, and so it can be appended below the $(k + 1)$–by–k dense matrix. Thus, the lower triangle of A can be stored as a single $(N+1)$–by–k dense matrix A_R. In effect, each block matrix A_{11}, A_{21}, and A_{22} is now stored in "full format." This means all entries of matrix A_R in array `AR` of size `LDA1` $= N + 1$ by `N1` $= k$ can be accessed with constant row and column strides. So the full power of LAPACK's block Level 3 codes are now available for RFPF, which uses the minimum amount of storage. Finally, matrix A_R^T which has size k–by–$(N + 1)$ is represented in the transpose of array `AR` and hence has the same desirable properties. There are eight representations of RFPF. The matrix A can have either odd or even order N, or it can be represented either in standard lower or upper format or it can be represented by either matrix A_R or its transpose A_R^T, giving $2^3 = 8$ representations in all. In Figure 3, we represent the case for N odd, `uplo` = `'lower'`, and `trans` = `'no transpose'`. All eight cases or representations are presented in Section 6.

3. CHOLESKY FACTORIZATION USING RECTANGULAR FULL PACKED FORMAT

The Cholesky factorization of a symmetric and positive definite matrix A can be expressed as

$$\begin{aligned} A = LL^T \text{ or } A = U^TU(\text{in the symmetric case}),\\ A = LL^H \text{ or } A = U^HU(\text{in the Hermitian case}), \end{aligned} \tag{2}$$

where L and U are lower triangular and upper triangular matrices, respectively.

Break the matrices L and U into 2–by–2 block form in the same way as was done for the matrix A in Equation (1):

$$L = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \text{ and } U = \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}. \tag{3}$$

We now have

the symmetric case:

$$LL^T = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22}^T \end{bmatrix} \text{ and } U^TU = \begin{bmatrix} U_{11}^T & 0 \\ U_{12}^T & U_{22}^T \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix},$$

and the Hermitian case:

$$LL^H = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} L_{11}^H & L_{21}^H \\ 0 & L_{22}^H \end{bmatrix} \text{ and } U^HU = \begin{bmatrix} U_{11}^H & 0 \\ U_{12}^H & U_{22}^H \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}, \tag{4}$$

where L_{11}, L_{22}, U_{11}, and U_{22} are lower and upper triangular submatrices, and L_{21} and U_{12} are square or almost square submatrices.

Using Equations (2) and equating the blocks of Equations (1) and Equations (4) give us the basis of a 2–by–2 block algorithm for Cholesky factorization using RFPF. We can now express each of these four block equalities by calls to existing LAPACK and Level 3 BLAS routines. In the real lower case, we obtain three block equations: $L_{11}L_{11}^T = A_{11}$, $L_{21}L_{11}^T = A_{21}$, and $L_{21}L_{21}^T + L_{22}L_{22}^T = A_{22}$. The first and second of these block equations are handled by calling LAPACK's POTRF routine $L_{11} \leftarrow \texttt{chol}(A_{11})$ and by calling Level 3 BLAS TRSM via $L_{21} \leftarrow A_{21}L_{11}^{-T}$. In both these block equations, the Fortran equality of replacement ($\leftarrow$) is being used so that the lower triangle of A_{11} is being replaced by L_{11} and the nearly square matrix A_{21} is being replaced by L_{21}. The third block equation breaks into two parts: $A_{22} \leftarrow A_{22} - L_{21}L_{21}^T$ and $L_{22} \leftarrow \texttt{chol}(A_{22})$, which are handled by calling Level 3 BLAS SYRK or HERK and by calling LAPACK's POTRF routine. At this point, we can use the flexibility of the LAPACK library. In RFPF, A_{22} is in upper format (upper triangle) while, in standard format, A_{22} is in lower format (lower triangle). Due to symmetry, both formats of A_{22} contain equal values. This flexibility allows LAPACK to accommodate both formats. Hence, in the calls to SYRK or HERK and POTRF we set `uplo = 'U'` even though the rectangular matrix of SYRK and HERK comes from a lower triangular formulation. The corresponding LAPACK code is given in Figure 4. (The data layout for the array `AR` used here when N is odd can be seen in Figure 3.)

```
        **Cholesky Factorization Algorithm** $(n1 = \lceil N/2 \rceil, n2 = N - n1)$
1) factor $L_{11}L_{11}^T = A_{11}$
   call POTRF$('L', n1, AR, N, info)$
2) solve $L_{21}L_{11}^T = A_{21}$ for $L_{21}$
   call TRSM$('R','L','T','N', n2, n1, one, AR, N, AR(n1+1,1), N)$
3) update $L_{22} := A_{22} - L_{21}L_{21}^T$ as $U_{22} := A_{22} - L_{21}L_{21}^T$
   call SYRK/HERK$('U','N', n2, n1, -one, AR(n1+1,1), N, one, AR(1,2), N)$
4) factor $L_{22}L_{22}^T = L_{22}$ as $U_{22}^T U_{22} = U_{22}$
   call POTRF$('U', n2, AR(1,2), N, info)$
```

Fig. 4. The Cholesky factorization algorithm using the Rectangular Full Packed Format (RFPF) if N is odd, `uplo = 'lower'`, and `trans = 'no transpose'`.

The values of L are stored over the values of A. The code does not use any temporary arrays. We have named the new LAPACK routine *PFTRF*. We have chosen the prefix PF to designate a symmetric/Hermitian positive definite matrix stored in a RFPF array to fit with LAPACK's use of PO for full storage and PP for packed storage.

4. SOLUTION

In Section 3 we obtained the 2–by–2 Cholesky factorization (3) of matrix A. Now, we can solve the equation $AX = B$:

—If A has lower triangular format then

$$\begin{aligned} LY = B \text{ and } L^T X = Y \text{ (in the symmetric case)},\\ LY = B \text{ and } L^H X = Y \text{ (in the Hermitian case)}. \end{aligned} \tag{5}$$

—If A has an upper triangular format then

$$\begin{aligned} U^T Y = B \text{ and } UX = Y \text{ (in the symmetric case)},\\ U^H Y = B \text{ and } UX = Y \text{ (in the Hermitian case)}. \end{aligned} \tag{6}$$

B, X, and Y are either vectors or rectangular matrices. B contains the right-hand side (RHS) values. X and Y contain the solution values. B, X, and Y are vectors when there is one RHS and matrices when there are many RHSs. The values of X and Y are stored over the values of B.

Expanding (5) and (6) using (3) give the forward substitution equations

$$\begin{gathered} \text{in the symmetric case:}\\ \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} \text{ and } \begin{bmatrix} U_{11}^T & 0 \\ U_{12}^T & U_{22}^T \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix},\\ \text{and in the Hermitian case:}\\ \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix} \text{ and } \begin{bmatrix} U_{11}^H & 0 \\ U_{12}^H & U_{22}^H \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}, \end{gathered} \tag{7}$$

```
                 Cholesky Solution Algorithm,
where $B(LDB, nr)$ and $LDB \geq N$ (here $LDB = N$) :
Solve $LY = B$
1.1) solve $L_{11}Y_1 = B_1$
    call TRSM$('L','L','N','N', n1, nr, one, AR, N, B, N)$
1.2) update $B_2 = B_2 - L_{21}Y_1$
    call GEMM$('N','N', n2, nr, n1, -one, AR(n1+1,1), N, B, N, one, B(n1+1,1), N)$
1.3) solve $L_{22}Y_2 = B_2$  as $U_{22}^TY_2 = B_2$
    call TRSM$('L','U','T','N', n2, nr, one, AR(1,2), N, B(n1+1,1), N)$
Solve $L^TX = Y$
2.1) solve $L_{22}^TX_2 = Y_2$  as $U_{22}Y_2 = B_2$
    call TRSM$('L','U','N','N', n2, nr, one, AR(1,2), N, B(n1+1,1), N)$
2.2) update $Y_1 = Y_1 - L_{21}^TX_2$
    call GEMM$('T','N', n1, nr, n2, -one, AR(n1+1,1), N, B(n1+1,1), N, one, B, N)$
2.3) solve $L_{11}^TX_1 = Y_1$
    call TRSM$('L','L','T','N', n1, nr, one, AR, N, B, N)$
```

Fig. 5. The Cholesky solution algorithm using the Rectangular Full Packed Format (RFPF) if N is odd, `uplo = 'lower'`, and `trans = 'no transpose'`.

and the back substitution equations

$$\text{in the symmetric case:}$$
$$\begin{bmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22}^T \end{bmatrix}\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} \text{ and } \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix}, \tag{8}$$
$$\text{and in the Hermitian case:}$$
$$\begin{bmatrix} L_{11}^H & L_{21}^H \\ 0 & L_{22}^H \end{bmatrix}\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} \text{ and } \begin{bmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{bmatrix}\begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix}.$$

The Equations (7) and (8) give the basis of a 2×2 block algorithm for Cholesky solution using the RFPF format. We can now express these two sets of two block equalities by using existing Level 3 BLAS routines. As an example, the first set of these two block equalities is $L_{11}Y_1 = B_1$ and $L_{21}Y_1 + L_{22}Y_2 = B_2$. The first block equality is handled by Level 3 BLAS TRSM: $Y_1 \leftarrow L_{11}^{-1}B_1$. The second block equality is handled by Level 3 BLAS GEMM and TRSM: $B_2 \leftarrow B_2 - L_{21}Y_1$ and $Y_2 \leftarrow L_{22}^{-1}B_2$. The backsolution routines are similarly derived. One gets $X_2 \leftarrow L_{22}^{-T}Y_2$, $Y_1 \leftarrow Y_1 - L_{21}^TX_2$ and $X_1 \leftarrow L_{11}^{-T}Y_1$.

The corresponding LAPACK code is given in Figure 4. In RFPF, L_{22} is stored as an upper triangular matrix while it represents a lower triangular one. (See Figures 3, 7, and 8.) Hence, in the calls to TRSM using L_{22} (see lines 1.3 and 2.1 in Figure 5), we set `uplo = 'U'`.

New LAPACK-like routine PFTRS performs these two solution computations for the eight cases of RFPF. PFTRS calls a new Level 3 BLAS TFSM in the same way that POTRS calls TRSM.

5. INVERSION

We consider the following three stage procedures:

(1) Factorize the matrix A and overwrite A with either L or U by calling PFTRF; see Section 3.

(2) Compute the inverse of either L or U. Call these matrices W or V and overwrite either L or U with them. We wrote a new routine named TFTRI available in LAPACK version 3.2 for this purpose.
(3) Calculate either the product $W^T W$ or VV^T and overwrite either W or V with them.

As in Sections 3 and 4 we examined 2–by–2 block algorithms for steps two and three above. In Section 3 we obtained either matrices L or U in RFPF. Like LAPACK inversion algorithms for POTRI and PPTRI, this is our starting point for our LAPACK inversion algorithm PFTRI using RFPF. The LAPACK inversion algorithms for POTRI and PPTRI also follow from steps 2 and 3 above by first calling in the full case LAPACK TRTRI and then calling LAPACK LAUUM.

Take the inverse of Equation (2) and obtain

$$\begin{aligned} A^{-1} &= W^T W \text{ or } A^{-1} = VV^T \text{(in the symmetric case)},\\ A^{-1} &= W^H W \text{ or } A^{-1} = VV^H \text{(in the Hermitian case)}, \end{aligned} \tag{9}$$

where W and V are lower and upper triangular matrices.

Using the 2–by–2 blocking for either L or U in Equations (3), we obtain the following 2–by–2 blocking for W and V:

$$W = \begin{bmatrix} W_{11} & 0 \\ W_{21} & W_{22} \end{bmatrix} \text{ and } V = \begin{bmatrix} V_{11} & V_{12} \\ 0 & V_{22} \end{bmatrix}. \tag{10}$$

From the identities $WL = LW = I$ and $VU = UV = I$ and the 2–by–2 block layouts of Equations (10), we obtain three block equations for W and V which can be solved using LAPACK routines for TRTRI and Level 3 BLAS TRMM. To illustrate the procedure, we consider the three block equations $L_{11}W_{11} = I$, $L_{21}W_{11} + L_{22}W_{21} = 0$, and $L_{22}W_{22} = I$. The first and third of these block equations are handled by LAPACK TRTRI routines as $W_{11} \leftarrow L_{11}^{-1}$ and $W_{22} \leftarrow L_{22}^{-1}$. In the second inverse computation, we use the fact that L_{22} is equally represented by its transpose L_{22}^T, which is U_{22} in RFPF. So we get $V_{22} \leftarrow U_{22}^{-1}$. The second block equation leads to two calls to Level 3 BLAS TRMM via $W_{21} \leftarrow -L_{21}W_{11}$ and $W_{21} \leftarrow W_{22}W_{21}$. In the second call, compute $W_{21} \leftarrow V_{22}^T W_{21}$ as the V_{22} matrix is in upper format when using RFPF. In the last two block equations, the Fortran equality of replacement ($\leftarrow$) is being used so that $W_{21} \leftarrow -W_{22}L_{21}W_{11}$ is replacing L_{21}.

Now we turn to part 3 of the three stage LAPACK procedure above. For this we use the 2–by–2 blocks layouts of Equations (10) and the matrix multiplications indicated by following block Equations (12) giving

$$\begin{gathered} \text{the symmetric case:}\\ W^T W = \begin{bmatrix} W_{11}^T & W_{21}^T \\ 0 & W_{22}^T \end{bmatrix} \begin{bmatrix} W_{11} & 0 \\ W_{21} & W_{22} \end{bmatrix} \text{ and } VV^T = \begin{bmatrix} V_{11} & V_{12} \\ 0 & V_{22} \end{bmatrix} \begin{bmatrix} V_{11}^T & 0 \\ V_{12}^T & V_{22}^T \end{bmatrix},\\ \text{and the Hermitian case:}\\ W^H W = \begin{bmatrix} W_{11}^H & W_{21}^H \\ 0 & W_{22}^H \end{bmatrix} \begin{bmatrix} W_{11} & 0 \\ W_{21} & W_{22} \end{bmatrix} \text{ and } VV^H = \begin{bmatrix} V_{11} & V_{12} \\ 0 & V_{22} \end{bmatrix} \begin{bmatrix} V_{11}^H & 0 \\ V_{12}^H & V_{22}^H \end{bmatrix}, \end{gathered} \tag{11}$$

```
Cholesky Inversion Algorithm
Inversion
1) invert $W_{11} = L_{11}^{-1}$
   call TRTRI$('L','N', n1, AR, N, info)$
2) multiply $W_{21} = -L_{21}W_{11}$
   call TRMM$('R','L','N','N', n2, n1, -one, AR, N, AR(n1+1,1), N)$
3) invert $W_{22} = L_{22}^{-1}$  as $V_{22} = U_{22}^{-1}$
   call TRTRI$('U','N', n2, AR(1,2), N, info)$
4) multiply $W_{21} = W_{22}W_{21}$  as $W_{21} = V_{22}^{T}W_{21}$
   call TRMM$('L','U','T','N', n2, n1, one, AR(1,2), N, AR(n1+1,1), N)$
Triangular matrix multiplication
1) triang. mult. $W_{11} = W_{11}^{T}W_{11}$
   call LAUUM$('L', n1, AR, N, info)$
2) update $W_{11} = W_{11} + W_{21}^{T}W_{21}$
   call SYRK/HERK$('L','T', n1, n2, one, AR(n1+1,1), N, one, AR, N)$
3) multiply $W_{21} = W_{22}W_{21}$  as $W_{21} = V_{22}W_{21}$
   call TRMM$('L','U','N','N', n2, n1, one, AR(1,2), N, A(n1+1,1), N)$
4) triang. mult. $W_{22} = W_{22}^{T}W_{22}$  as $V_{22} = V_{22}V_{22}^{T}$
   call LAUUM$('U', n2, AR(1,2), N, info)$
```

Fig. 6. The Cholesky inversion algorithm using the Rectangular Full Packed Format (RFPF) if N is odd, `uplo` = `'lower'`, and `trans` = `'no transpose'`.

where W_{11}, W_{22}, V_{11}, and V_{22} are lower and upper triangular submatrices, and W_{21} and V_{12} are square or almost square submatrices. The values of the indicated block multiplications of W or V in Equations (12) are stored over the block values of W or V.

Performing the indicated 2–by–2 block multiplications of Equations (12) leads to three block matrix computations. To illustrate the procedure, we consider the three block computations $W_{11}^{T}W_{11} + W_{21}^{T}W_{21}$, $W_{22}^{T}W_{21}$, and $W_{22}^{T}W_{22}$. Additionally, we want to overwrite the values of these block multiplications on their original block operands. Block operand W_{11} only occurs in the (1, 1) block operand computation and hence can be overwritten by a call to LAPACK LAUUM, $W_{11} \leftarrow W_{11}^{T}W_{11}$, followed by a call to Level 3 BLAS SYRK or HERK, $W_{11} \leftarrow W_{11} + W_{21}^{T}W_{21}$. Block operand W_{21} now only occurs in the (2, 1) block computation and hence can be overwritten by a call to Level 3 BLAS TRMM, $W_{21} \leftarrow W_{22}^{T}W_{21}$. Finally, block operand W_{22} can be overwritten by a call to LAPACK LAUUM, $W_{22} \leftarrow W_{22}^{T}W_{22}$. Note that in the RFPF format $W_{22} \leftarrow L_{22}^{-1}$ will be $V_{22} \leftarrow U_{22}^{-1}$, also supported by LAPACK TRTRI, as RFPF stores A_{22} in the upper format instead of in the lower format. LAPACK POTRI supports both the UPLO formats. For this reason, no new LAPACK software is needed to support the RFPF format. The corresponding LAPACK code is given in Figure 6. The new LAPACK routine, PFTRI, performs this computation for the eight cases of RFPF.

6. GENERALIZATIONS

6.1 RFP Data Format: The Eight Variants

This section contains two figures.

(1) Figure 7 shows the transformation from full to RFPF of all "no transpose" cases.
(2) Figure 8 depicts all eight cases of RFPF.

The data format for A has `LDA` $= N$. Matrix A_R has `LDAR` $= N$ if N is odd and `LDAR` $= N + 1$ if N is even and $n1$ columns where $n1 = \lceil N/2 \rceil$. Hence, matrix A_R always has `LDAR` rows and $n1$ columns. Matrix A_R^T always has $n1$ rows and `LDAR` columns and its leading dimension is equal to $n1$. Matrix A_R always has `LDAR` $\times\, n1 = NT = N(N+1)/2$ elements, as does matrix A_R^T.

The order N of matrix A in both figures is six or seven.

6.2 From Full Packed Format Algorithm to RFPF Algorithm

We now consider the performance benefits of the RFP format in the context of using LAPACK routines on triangular matrices stored in RFP format. Let X be a Level 3 LAPACK routine that operates either on standard packed or full format. X has a full 2-by-2 Level 3 LAPACK block algorithm, call it *FX*. Write a *simple related partition algorithm* (SRPA) with partition sizes $n1$ and $n2$. Apply the new SRPA on the new RFP data structure. The new SRPA almost always has four major steps consisting entirely of calls to the full format LAPACK routine X in two steps and calls to level 3 BLAS in the remaining two steps. We give below an illustration in the case where N is odd, `uplo` is `'lower'`, and `trans` is `'no transpose'`.

```
call FX('L',n1,AR,ldar) ! step 1
call L3BLAS('parms',n1,n2,AR,ldar,AR(n1+1,1),ldar) ! step 2
call L3BLAS('parms',n1,n2,AR(n1+1,1),ldar,AR(1,2),ldar) ! step 3
call FX('U',n2,AR(1,2),ldar) ! step 4
```

7. STABILITY OF THE RFPF ALGORITHM

The RFPF Cholesky factorization (Section 3), Cholesky solution (Section 4), and Cholesky inversion (Section 5) algorithms are equivalent to the traditional algorithms in the books [Dongarra et al. 1998; Demmel 1997; Golub and Van Loan 1996; Trefethen and Bau 1997]. The whole theory of the traditional Cholesky factorization, solution, inversion, and BLAS algorithms carries over to these three Cholesky and BLAS algorithms described in Sections 3, 4, and 5. The error analysis and stability of these algorithms was very well described in Higham [1996]. The difference between LAPACK algorithms PO, PP, and RFPF[3] is how inner products are accumulated. In each case a different order is used. They are all mathematically equivalent, and stability analysis shows that any summation order is stable.

8. A PERFORMANCE STUDY USING RFP FORMAT

The LAPACK library [Anderson et al. 1999] routines POTRF/PPTRF, POTRI/PPTRI, and POTRS/PPTRS were compared with the RFPF routines

[3]full, packed and rectangular full packed.

7.1 The matrices A of order N and A_R of size $LDAR$ by $n1$, here $N = 7$.

7.1.1 Full Format **7.1.2 RFPF**

$$\left[\begin{array}{cccc|ccc} a_{1,1} & \diamond & \diamond & \diamond & \diamond & \diamond & \diamond \\ a_{2,1} & a_{2,2} & \diamond & \diamond & \diamond & \diamond & \diamond \\ a_{3,1} & a_{3,2} & a_{3,3} & \diamond & \diamond & \diamond & \diamond \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & \diamond & \diamond & \diamond \\ \cline{1-4} a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & a_{5,5} & \diamond & \diamond \\ a_{6,1} & a_{6,2} & a_{6,3} & a_{6,4} & a_{6,5} & a_{6,6} & \diamond \\ a_{7,1} & a_{7,2} & a_{7,3} & a_{7,4} & a_{7,5} & a_{7,6} & a_{7,7} \end{array}\right], \quad \left[\begin{array}{cccc} a_{1,1} & a_{5,5} & a_{6,5} & a_{7,5} \\ a_{2,1} & a_{2,2} & a_{6,6} & a_{7,6} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{7,7} \\ \hline a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \\ a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} \\ a_{6,1} & a_{6,2} & a_{6,3} & a_{6,4} \\ a_{7,1} & a_{7,2} & a_{7,3} & a_{7,4} \end{array}\right]$$

7.2 The matrices A of order N and A_R of size $LDAR$ by $n1$, here $N = 6$.

7.2.1 Full format **7.2.2 RFPF**

$$\left[\begin{array}{ccc|ccc} a_{1,1} & \diamond & \diamond & \diamond & \diamond & \diamond \\ a_{2,1} & a_{2,2} & \diamond & \diamond & \diamond & \diamond \\ a_{3,1} & a_{3,2} & a_{3,3} & \diamond & \diamond & \diamond \\ \cline{1-3} a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & \diamond & \diamond \\ a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} & a_{5,5} & \diamond \\ a_{6,1} & a_{6,2} & a_{6,3} & a_{6,4} & a_{6,5} & a_{6,6} \end{array}\right], \quad \left[\begin{array}{ccc} a_{4,4} & a_{5,4} & a_{6,4} \\ \cline{1-1} a_{1,1} & a_{5,5} & a_{6,5} \\ a_{2,1} & a_{2,2} & a_{6,6} \\ a_{3,1} & a_{3,2} & a_{3,3} \\ \hline a_{4,1} & a_{4,2} & a_{4,3} \\ a_{5,1} & a_{5,2} & a_{5,3} \\ a_{6,1} & a_{6,2} & a_{6,3} \end{array}\right]$$

7.3 The matrices A of order N and A_R of size $LDAR$ by $n1$, here $N = 7$.

7.3.1 Full format **7.3.2 RFPF**

$$\left[\begin{array}{ccc|cccc} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & a_{1,6} & a_{1,7} \\ \diamond & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & a_{2,6} & a_{2,7} \\ \diamond & \diamond & a_{3,3} & a_{3,4} & a_{3,5} & a_{3,6} & a_{3,7} \\ \cline{4-7} \diamond & \diamond & \diamond & a_{4,4} & a_{4,5} & a_{4,6} & a_{4,7} \\ \diamond & \diamond & \diamond & \diamond & a_{5,5} & a_{5,6} & a_{5,7} \\ \diamond & \diamond & \diamond & \diamond & \diamond & a_{6,6} & a_{6,7} \\ \diamond & \diamond & \diamond & \diamond & \diamond & \diamond & a_{7,7} \end{array}\right], \quad \left[\begin{array}{cccc} a_{1,4} & a_{1,5} & a_{1,6} & a_{1,7} \\ a_{2,4} & a_{2,5} & a_{2,6} & a_{2,7} \\ a_{3,4} & a_{3,5} & a_{3,6} & a_{3,7} \\ \hline a_{4,4} & a_{4,5} & a_{4,6} & a_{4,7} \\ \cline{1-1} a_{1,1} & a_{5,5} & a_{5,6} & a_{5,7} \\ a_{1,2} & a_{2,2} & a_{6,6} & a_{6,7} \\ a_{1,3} & a_{2,3} & a_{3,3} & a_{7,7} \end{array}\right]$$

7.4 The matrices A of order N and A_R of size $LDAR$ by $n1$, here $N = 6$.

7.4.1 Full format **7.4.2 RFPF**

$$\left[\begin{array}{ccc|ccc} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} & a_{1,6} \\ \diamond & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} & a_{2,6} \\ \diamond & \diamond & a_{3,3} & a_{3,4} & a_{3,5} & a_{3,6} \\ \cline{4-6} \diamond & \diamond & \diamond & a_{4,4} & a_{4,5} & a_{4,6} \\ \diamond & \diamond & \diamond & \diamond & a_{5,5} & a_{5,6} \\ \diamond & \diamond & \diamond & \diamond & \diamond & a_{6,6} \end{array}\right], \quad \left[\begin{array}{ccc} a_{1,4} & a_{1,5} & a_{1,6} \\ a_{2,4} & a_{2,5} & a_{2,6} \\ a_{3,4} & a_{3,5} & a_{3,6} \\ \hline a_{4,4} & a_{4,5} & a_{4,6} \\ \cline{1-1} a_{1,1} & a_{5,5} & a_{5,6_{19}} \\ a_{1,2} & a_{2,2} & a_{6,6} \\ a_{1,3} & a_{2,3} & a_{3,3} \end{array}\right]$$

Fig. 7. Eight two-dimensional arrays for storing the matrices A and A_R that are needed by the LAPACK subroutine POTRF (full format) and PFTRF RFPF, respectively. The leading dimension LDA is N for LAPACK, and LDAR for RFPF. LDAR $= N$ for N odd, and $N + 1$ for N even. Here N is 7 or 6. The memory needed is $LDA \times N$ for full format and $LDAR \times n1 = (N + 1)N/2$ for RFPF, here 49 and 36 for full format and 28 and 21 for RFPF. The column size of RFPF is $n1 = \lceil N/2 \rceil$, here 4 and 3.

8.1 RFPF for the matrices of rank odd, here $N = 7$ and $n1 = 4$

Lower triangular

$LDAR = N$

$$\left[\begin{array}{cccc}
a_{1,1} & a_{5,5} & a_{6,5} & a_{7,5} \\
a_{2,1} & a_{2,2} & a_{6,6} & a_{7,6} \\
a_{3,1} & a_{3,2} & a_{3,3} & a_{7,7} \\
a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \\
\hline
a_{5,1} & a_{5,2} & a_{5,3} & a_{5,4} \\
a_{6,1} & a_{6,2} & a_{6,3} & a_{6,4} \\
a_{7,1} & a_{7,2} & a_{7,3} & a_{7,4}
\end{array}\right]$$

transpose, $lda = n1$

$$\left[\begin{array}{cccc|ccc}
a_{1,1} & a_{2,1} & a_{3,1} & a_{4,1} & a_{5,1} & a_{6,1} & a_{7,1} \\
a_{5,5} & a_{2,2} & a_{3,2} & a_{4,2} & a_{5,2} & a_{6,2} & a_{7,2} \\
a_{6,5} & a_{6,6} & a_{3,3} & a_{4,3} & a_{5,3} & a_{6,3} & a_{7,3} \\
a_{7,5} & a_{7,6} & a_{7,7} & a_{4,4} & a_{5,4} & a_{6,4} & a_{7,4}
\end{array}\right]$$

Upper triangular

$LDAR = N$

$$\left[\begin{array}{cccc}
a_{1,4} & a_{1,5} & a_{1,6} & a_{1,7} \\
a_{2,4} & a_{2,5} & a_{2,6} & a_{2,7} \\
a_{3,4} & a_{3,5} & a_{3,6} & a_{3,7} \\
\hline
a_{4,4} & a_{4,5} & a_{4,6} & a_{4,7} \\
a_{1,1} & a_{5,5} & a_{5,6} & a_{5,7} \\
a_{1,2} & a_{2,2} & a_{6,6} & a_{6,7} \\
a_{1,3} & a_{2,3} & a_{3,3} & a_{7,7}
\end{array}\right]$$

transpose, $lda = n1$

$$\left[\begin{array}{ccc|cccc}
a_{1,4} & a_{2,4} & a_{3,4} & a_{4,4} & a_{1,1} & a_{1,2} & a_{1,3} \\
a_{1,5} & a_{2,5} & a_{3,5} & a_{4,5} & a_{5,5} & a_{2,2} & a_{2,3} \\
a_{1,6} & a_{2,6} & a_{3,6} & a_{4,6} & a_{5,6} & a_{6,6} & a_{3,3} \\
a_{1,7} & a_{2,7} & a_{3,7} & a_{4,7} & a_{5,7} & a_{6,7} & a_{7,7}
\end{array}\right]$$

8.2 RFPF for the matrices of rank even, here $N = 6$ and $n1 = 3$.

Lower triangular

$LDAR = N + 1$

$$\left[\begin{array}{ccc}
a_{4,4} & a_{5,4} & a_{6,4} \\
\hline
a_{1,1} & a_{5,5} & a_{6,5} \\
a_{2,1} & a_{2,2} & a_{6,6} \\
a_{3,1} & a_{3,2} & a_{3,3} \\
\hline
a_{4,1} & a_{4,2} & a_{4,3} \\
a_{5,1} & a_{5,2} & a_{5,3} \\
a_{6,1} & a_{6,2} & a_{6,3}
\end{array}\right]$$

transpose, $lda = n1$

$$\left[\begin{array}{cccc|ccc}
a_{4,4} & a_{1,1} & a_{2,1} & a_{3,1} & a_{4,1} & a_{5,1} & a_{6,1} \\
a_{5,4} & a_{5,5} & a_{2,2} & a_{3,2} & a_{4,2} & a_{5,2} & a_{6,2} \\
a_{6,4} & a_{6,5} & a_{6,6} & a_{3,3} & a_{4,3} & a_{5,3} & a_{6,3}
\end{array}\right]$$

Upper triangular

$LDAR = N + 1$

$$\left[\begin{array}{ccc}
a_{1,4} & a_{1,5} & a_{1,6} \\
a_{2,4} & a_{2,5} & a_{2,6} \\
a_{3,4} & a_{3,5} & a_{3,6} \\
\hline
a_{4,4} & a_{4,5} & a_{4,6} \\
a_{1,1} & a_{5,5} & a_{5,6_{19}} \\
a_{1,2} & a_{2,2} & a_{6,6} \\
a_{1,3} & a_{2,3} & a_{3,3}
\end{array}\right]$$

transpose, $lda = n1$

$$\left[\begin{array}{ccc|cccc}
a_{1,4} & a_{2,4} & a_{3,4} & a_{4,4} & a_{1,1} & a_{1,2} & a_{1,3} \\
a_{1,5} & a_{2,5} & a_{3,5} & a_{4,5} & a_{5,5} & a_{2,2} & a_{2,3} \\
a_{1,6} & a_{2,6} & a_{3,6} & a_{4,6} & a_{5,6} & a_{6,6} & a_{3,3}
\end{array}\right]$$

Fig. 8. Eight two-dimensional arrays for storing the matrices A_R and A_R^T in RFPF. The leading dimension *LDAR* of A_R is N when N is odd and $N + 1$ when N is even. For the matrix A_R^T, it is $n1 = \lceil N/2 \rceil$. The memory needed for both A_R and A_R^T is $(N + 1)/2 \times N$. This amount is 28 for $N = 7$ and 21 for $N = 6$.

PFTRF, PFTRI, and PFTRS for Cholesky factorization (PxTRF), Cholesky inverse (PxTRI), and Cholesky solution (PxTRS), respectively. In the previous sentence, the character "x" can be "O" (full format), "P" (packed format), or "F" (RFPF). In all cases, long real precision arithmetic (also called *double precision*) was used. Results were obtained on several different computers using everywhere the vendor Level 3 and Level 2 BLAS. The sequential performance results were done on the following computers:

—*Sun Fire E25K* (`newton`). UltraSPARC IV+ dual-core node (1800-MHz; 2-MB shared L2-cache; 32-MB shared L3-cache). The theoretical peak for a node is 6.0 GFlop/s.

—*SGI Altix 3700* (`freke`). Intel Itanium2 node (1.5-GHz; 6-MB L3-cache). The theoretical peak for a node is 3.6 GFlop/s.

—*DMI NEC SX-6*. (Vector register length: 256; 8 CPU's per node.) The theoretical peak for a node is 8.0 GFlop/s per CPU or 64 GFlops/s for the whole node composed of 8 CPU.

—*Intel Tigerton* (`zoot`). Quad-socket quad-core Intel Tigerton 2.4-GHz (16 total cores) node with 32 GB of memory. The theoretical peak is equal to 9.6 GFlop/s per core or 153.2 GFlop/s for the whole node, composed of 16 cores. We used Intel MKL 10.0.1.014.

The performance results are given in Figures 9 to 14.

Figure 9 (double precision) presents results for the Sun UltraSPARC IV+ computer. Figure 10 (double precision) presents results for the SGI Altix 3700 computer. Figure 11 (double precision) presents results for the NEC SX-6 computer. Figure 12 (double precision) presents results for the quad-socket quad-core Intel Tigerton computer using reference LAPACK-3.2.0.4.[4] Figure 13 (double precision) presents results for the quad-socket quad-core Intel Tigerton computer using the vendor LAPACK library (MKL-10.0.1.14).

Figure 14 shows the SMP parallelism of these subroutines on the IBM Power4 (clock rate: 1300 MHz; two CPUs per chip; L1 cache: 128-kB (64-kB/CPU) instruction, 64-kB two-way (32-kB/CPU) data; L2 cache: 1.5-MB eight-way shared between the two CPUs; L3 cache: 32 MB eight-way shared (off-chip); TLB: 1024 entries) and SUN UltraSPARC-IV (clock rate: 1350 MHz; L1 cache: 64-kB four-way data, 32-kB four-way instruction, and 2 kB Write, 2-kB Prefetch; L2 cache: 8-MB; TLB: 1040 entries) computers, respectively. They compare SMP times of PFTRF, vendor POTRF and reference PPTRF.

The RFPF packed results greatly outperformed the packed and more often than not were better than the full results. Note that our timings do *not* include the cost of sorting any LAPACK data formats to RFPF data formats and vice versa. We think that users will input their matrix data using RFPF. Hence, this is our rationale for not including the data transformation times.

For all our experiments, we used vendor Level 3 and Level 2 BLAS. For all our experiments except Figures 12 and 14, we use the provided the vendor libraries for LAPACK and BLAS.

[4]From `netlib.org`.

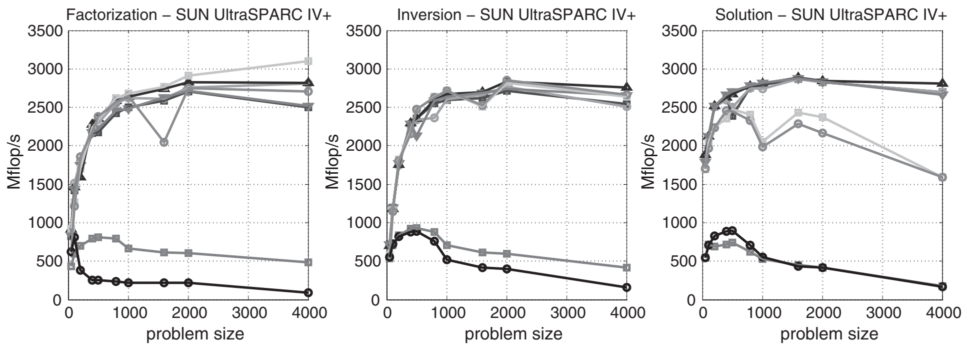

Fig. 9. Performance in Mflop/s of Cholesky factorization/inversion/solution on SUN UltraSPARC IV+ computer, long real arithmetic. For the solution phase, $nrhs = \max(100, n/10)$. ■: PF_N_U; ●: PF_N_L; ▲: PF_T_U; ▼: PF_T_L; ■: PO_U; ●: PO_L; ■: PP_U; ●: PP_L.

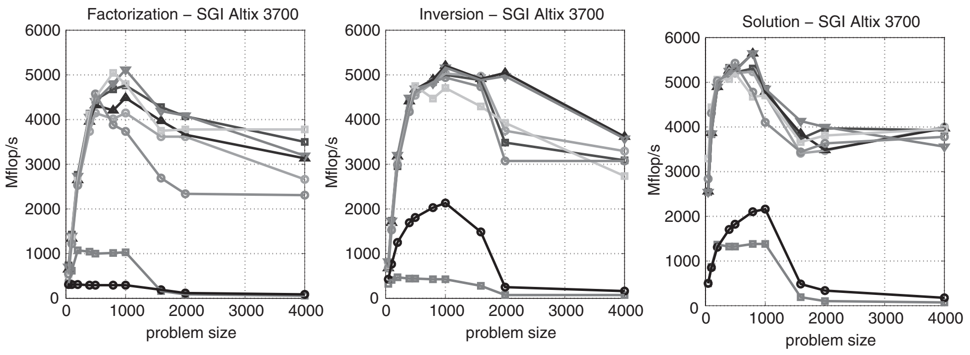

Fig. 10. Performance in Mflop/s of Cholesky factorization/inversion/solution on SGI Altix 3700, Intel Itanium 2 computer, long real arithmetic. For the solution phase, $nrhs = \max(100, n/10)$. ■: PF_N_U; ●: PF_N_L; ▲: PF_T_U; ▼: PF_T_L; ■: PO_U; ●: PO_L; ■: PP_U; ●: PP_L.

We include comparisons with reference LAPACK for the quad-socket quad-core Intel Tigerton machine in Figure 12. In this case, the vendor LAPACK library packed storage routines significantly outperformed the LAPACK reference implementation from netlib. In Figure 13, you find the same experiments on the same machine but, this time, using the vendor library (MKL-10.0.1.014). We think that MKL is using the reference implementation for inverse Cholesky (packed and full format). For Cholesky factorization, we see that both packed and full format routines (PPTRF and POTRF) were tuned. But even in this case, our RFPF storage format results were better.

When we compared RFPF with full storage, the results were mixed. However, both codes were rarely far apart. Most of the performance ratios were between 0.95 to 1.05 overall. But note that the RFPF performance was more uniform over its versions (four presented; the other four were for n odd). For LAPACK full (two versions), the performance variation was greater. Moreover, in the case of the inversion on quad-socket quad-core Tigerton (Figures 12 and 13), RFPF clearly outperformed both variants of the full format.

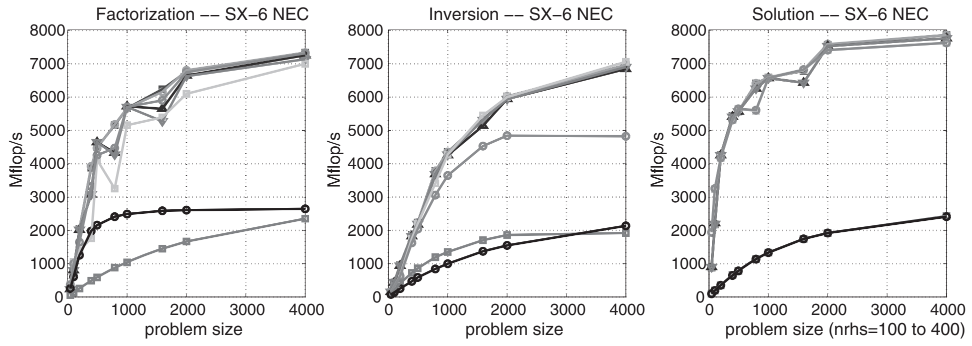

Fig. 11. Performance in Mflop/s of Cholesky factorization/inversion/solution on SX-6 NEC computer, long real arithmetic. For the solution phase, $nrhs = \max(100, n/10)$. ■: `PF_N_U`; ●: `PF_N_L`; ▲: `PF_T_U`; ▼: `PF_T_L`; ■: `PO_U`; ●: `PO_L`; ■: `PP_U`; ●: `PP_L`.

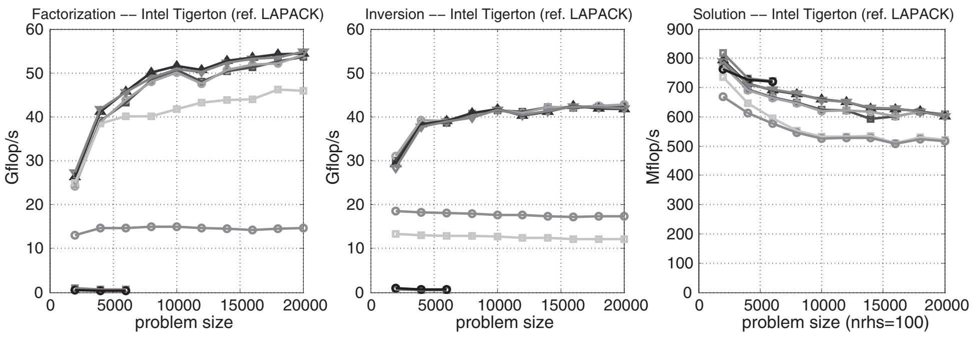

Fig. 12. Performance of Cholesky factorization/inversion/solution on quad-socket quad-core Intel Tigerton computer, long real arithmetic. We used reference LAPACK-3.2.0 (from netlib) and MKL-10.0.1.014 multithreaded BLAS. For the solution phase, *nrhs* was fixed to 100 for any n. Due to time limitations, the experiment was stopped for the packed storage format inversion at $n = 4000$. ■: `PF_N_U`; ●: `PF_N_L`; ▼: `PF_T_U`; ▲: `PF_T_L`; ■: `PO_U`; ●: `PO_L`; ■: `PP_U`; ●: `PP_L`.

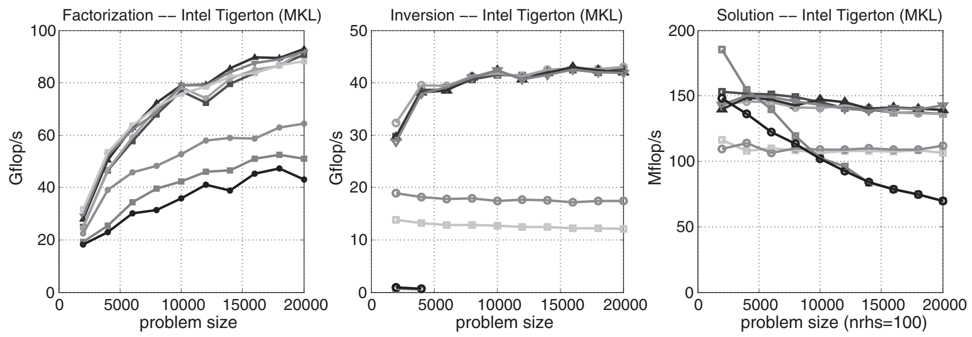

Fig. 13. Performance of Cholesky factorization/inversion/solution on quad-socket quad-core Intel Tigerton computer, long real arithmetic. We used MKL-10.0.1.014 multithreaded LAPACK and BLAS. For the solution phase, *nrhs* was fixed to 100 for any n. Due to time limitations, the experiment was stopped for the packed storage format inversion at $n = 4000$. ■: `PF_N_U`; ●: `PF_N_L`; ▲: `PF_T_U`; ▼: `PF_T_L`; ■: `PO_U`; ●: `PO_L`; ■: `PP_U`; ●: `PP_L`.

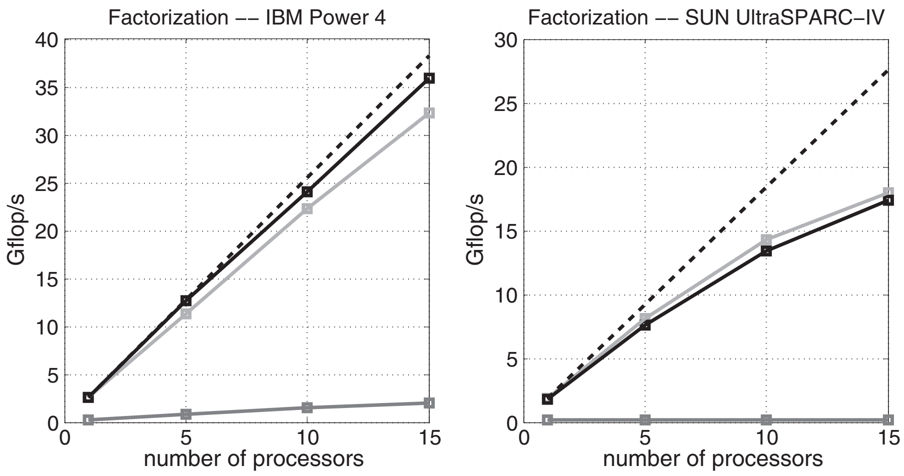

Fig. 14. Performance in Gflop/s of Cholesky factorization on IBM Power 4 (left) and SUN UltraSPARC-IV (right) computer, long real arithmetic, with a different number of processors, testing the SMP parallelism. The implementation of PPTRF of sunperf did not show any SMP parallelism. UPLO = "L." $N = 5{,}000$ (strong scaling experiment). The dashed line represents perfect scalability from one to 15 processors. ■: `PF_L`; ■: `PO_L`; ■: `PP_L`;

9. INTEGRATION IN LAPACK

As mentioned in the introduction, as of release 3.2 (November 2008), LAPACK supports a preliminary version of RFPF. Ultimately, the goal would be for RFPF to support as many functionalities as full format or standard packed format do. The 44 routines included in release 3.2 for RFPF are given in Table I. The names for the RFPF routines follow the naming nomenclature used by LAPACK. We have added the format description letters: PF for symmetric/Hermitian positive definite RFPF (PO for full, PP for packed); SF for symmetric RFPF (SY for full, SP for packed); HF for Hermitian RFPF (HE for full, HP for packed), and TF for triangular RFPF (TR for full, TP for packed).

Currently, for the complex case, we assume that the transpose complex-conjugate part is stored whenever the transpose part is stored in the real case. This corresponds to the theory developed in this present article. In the future, we will want to have the flexibility to store the transpose part (as opposed to transpose complex conjugate) whenever the transpose part is stored in the real case. In particular, this feature will be useful for complex symmetric matrices.

10. SUMMARY AND CONCLUSIONS

This article describes RFPF as a standard minimal full format for representing both symmetric and triangular matrices. Hence, from a user point of view, these matrix layouts are a replacement for both the standard formats of DLA, namely, full and packed storage. These new layouts possess three good features: they are efficient, they are supported by Level 3 BLAS and LAPACK full format routines, and they require minimal storage.

Table I. LAPACK 3.2 RFPF Routines

Functionality	Routine Names and Calling Sequence
Cholesky factorization	CPFTRF DPFTRF SPFTRF ZPFTRF (TRANSR, UPLO, N, A, INFO)
Multiple solve after PFTRF	CPFTRS DPFTRS SPFTRS ZPFTRS (TRANSR, UPLO, N, NR, A, B, LDB, INFO)
Inversion after PFTRF	CPFTRI DPFTRI SPFTRI ZPFTRI (TRANSR, UPLO, N, A, INFO)
Triangular inversion	CTRTRI DTRTRI STRTRI ZTRTRI (TRANSR, UPLO, DIAG, N, A, INFO)
Sym/Herm matrix norm	CLANHF DLANSF SLANSF ZLANHF (NORM, TRANSR, UPLO, N, A, WORK)
Triangular solve	CTFSM DTFSM STFSM ZTFSM (TRANSR,SIDE,UPLO,TRANS,DIAG,M,N,ALPHA,A,B,LDB)
Sym/Herm rank-*k* update	CHFRK DSFRK SSFRK ZHFRK (TRANSR,UPLO,TRANS,N,K,ALPHA,A,LDA,BETA,C)
Conv. from TP to TF	CTPTTF DTPTTF STPTTF ZTPTTF (TRANSR,UPLO,N,AP,ARF,INFO)
Conv. from TR to TF	CTRTTF DTRTTF STRTTF ZTRTTF (TRANSR,UPLO,N,A,LDA,ARF,INFO)
Conv. from TF to TP	CTFTTP DTFTTP STFTTP ZTFTTP (TRANSR,UPLO,N,ARF,AP,INFO)
Conv. from TF to TR	CTFTTR DTFTTR STFTTR ZTFTTR (TRANSR,UPLO,N,ARF,A,LDA,INFO)

ACKNOWLEDGMENTS

The authors would like to thank the four referees and the Associate Editor for their valuable and numerous suggestions. They made our article more readable by pointing out that a reorganization, a consistent mathematical treatment, and more examples would lead to better understanding of our article by TOMS readers. Two of the referees proved to be very meticulous. Their attention to details allowed us to better clarify both the mathematical and programming aspects of our article.

The results in this article were obtained on six computers: an IBM, a SGI, two SUNs, a NEC, and an Intel Tigerton. The IBM machine belongs to the Center for Scientific Computing at Aarhus, the two SUN machines to the Danish Technical University, the NEC machine to the Danish Meteorological Institute, and the Intel Tigerton machine to the Innovative Computing Laboratory at the University of Tennessee.

We would like to thank Bernd Dammann for consulting on the SUN systems; Niels Carl W. Hansen for consulting on the IBM and SGI systems; and Bjarne Stig Andersen for obtaining the results on the Itanium and NEC computers. We thank IBMers John Gunnels, who worked earlier on the HFPF format, and JP Fasano, who was instrumental in getting the source code released by the IBM Open Source Committee. We thank Allan Backer for discussions about an older version of this manuscript.

REFERENCES

Agarwal, R. C., Gustavson, F. G., and Zubair, M. 1994. Exploiting functional parallelism on power2 to design high-performance numerical algorithms. *IBM J. Res. Develop. 38*, 5, 563–576.

Andersen, B. S., Gunnels, J. A., Gustavson, F., and Waśniewski, J. 2002. A recursive formulation of the inversion of symmetric positive definite matrices in packed storage data format. In *Proceedings of the 6th International Conference on Applied Parallel Computing*, J. Fagerholm, J. Haataja, J. Järvinen, M. Lyly, and P. R. V. Savolainen, Eds. Lecture Notes in Computer Science, vol. 2367, Springer, Berlin, Germany, 287–296.

Andersen, B. S., Gustavson, F. G., Reid, J. K., and Waśniewski, J. 2005. A fully portable high performance minimal storage hybrid format Cholesky algorithm. *ACM Trans. Math. Softw. 31*, 201–227.

Andersen, B. S., Gustavson, F. G., and Waśniewski, J. 2001. A recursive formulation of cholesky facorization of a matrix in packed storage. *ACM Trans. Math. Softw. 27*, 2, 214–244.

Anderson, E., Bai, Z., Bischof, C., Blackford, L. S., Demmel, J., Dongarra, J. J., Du Croz, J., Greenbaum, A., Hammarling, S., McKenney, A., and Sorensen, D. 1999. *LAPACK Users' Guide*, 3rd Ed. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Barker, V. A., Blackford, L. S., Dongarra, J. J., Croz, J. D., Hammarling, S., Marinova, M., Waśniewski, J., and Yalamov, P. 2001. *LAPACK95 Users' Guide*, 1st ed. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Buttari, A., Langou, J., Kurzak, J., and Dongarra, J. 2009. A class of parallel tiled linear algebra algorithms for multicore architectures. *Parall. Comput. 35*, 38–53.

Chan, E., Quintana-Ortí, E., Quintana-Ortí, G., and van de Geijn, R. 2007. Super-matrix out-of-order scheduling of matrix operations for SMP and multi-core architectures. In *Proceedings of the 19th ACM Symposium on Parallelism in Algorithms and Architecture*. 116–125.

Demmel, J. W. 1997. *Applied Numerical Linear Algebra*. SIAM, Philadelphia, PA.

Dongarra, J. J., Bunch, J. R., Moler, C. B., and Stewart, G. W. 1979. *Linpack Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Dongarra, J. J., Du Croz, J., Duff, I. S., and Hammarling, S. 1990a. Algorithm 679: A set of Level 3 basic linear algebra subprograms. *ACM Trans. Math. Softw. 16*, 1, 18–28.

Dongarra, J. J., Du Croz, J., Duff, I. S., and Hammarling, S. 1990b. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Softw. 16*, 1, 1–17.

Dongarra, J. J., Du Croz, J., Hammarling, S., and Hanson, R. J. 1988. An extended set of Fortran basic linear algebra subroutines. *ACM Trans. Math. Softw. 14*, 1, 1–17.

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

Elmroth, E., Gustavson, F. G., Kagstrom, B., and Jonsson, I. 2004. Recursive blocked algorithms and hybrid data structures for dense matrix library software. *SIAM Rev. 46*, 1, 3–45.

Golub, G. and Van Loan, C. F. 1996. *Matrix Computations* 3rd Ed. Johns Hopkins University Press, Baltimore, MD.

Gunnels, J. A. and Gustavson, F. G. 2004. A new array format for symmetric and triangular matrices. In *Proceedings of the International Conference on Applied Parallel Computing (PARA'04)* J. W. J. J. Dongare and K. Madsen, eds. Lecture Nots in Computer Science, vol. 3732. Springer-Verlag, Berlin, Heidelberg, Germany, 247–255.

Gustavson, F. G. 2003. High performance linear algebra algorithms using new generalized data structures for matrices. *IBM J. Res. Develop. 47*, 1, 823–849.

Gustavson, F. G., Gunnels, J., and Sexton, J. 2007a. Minimal data copy for dense linear algebra factorization. In *Proceedings of the International Conference on Applied Parallel Computing, (PARA'06)*. Lecture Nots in Computer Science, vol. 4699, Springer-Verlag, Berlin Heidelberg, Germany, 540–549.

Gustavson, F. G. and Jonsson, I. 2000. Minimal storage high performance Cholesky via blocking and recursion. *IBM J. Res. Develop. 44*, 6, 823–849.

Gustavson, F. G., Reid, J. K., and Waśniewski, J. 2007b. Algorithm 865: Fortran 95 subroutines for Cholesky factorization in blocked hybrid format. *ACM Trans. Math. Softw. 33*, 1, 5.

Gustavson, F. G. and Waśniewski, J. 2007. Rectangular full packed format for LAPACK algorithms timings on several computers. In *Proceedings of the International Conference on Applied Parallel Computing (PARA'06)*. Lecture Nots in Computer Science, vol. 4699. Springer-Verlag, Berlin Heidelberg, Germany, 570–579.

Herrero, J. R. 2006. A framework for efficient execution of matrix computations. Ph.D. dissertation. Universitat Politècnica de Catalunya, Bancelona, Spain.

HIGHAM, N. J. 1996. *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, PA.
IBM. 1997. *Engineering and Scientific Subroutine Library for AIX*, Version 3, Volume 1. IBM, Yorktown Heights, NY. IBM. Pub. number SA22–7272–0.
LAWSON, C. L., HANSON, R. J., KINCAID, D., AND KROGH, F. T. 1979. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Softw. 5*, 308–323.
TREFETHEN, L. N. AND BAU, D. 1997. *Numerical Linear Algebra*. SIAM, Philadelphia, PA.

Received January 2009; revised July 2009; accepted November 2009