Mathematical Modelling and Analysis
Volume 4, 1999, pages 7–17
© 1999 Technika

LAWRA – LINEAR ALGEBRA WITH RECURSIVE ALGORITHMS

B. S. ANDERSEN[1], F. GUSTAVSON[2], A. KARAIVANOV[3], J. WAŚNIEWSKI[4] and P. Y. YALAMOV[1,5]

[1,3,4] *Danish Computing Centre for Research and Education (UNI•C), Technical University of Denmark*
DTU, Building 304, DK-2800 Lyngby, Denmark
[2] *IBM T.J. Watson Research Center*
P.P. Box 218, Yorktown Heights, NY 10598, USA
[5] *Center of Applied Mathematics and Informatics, University of Rousse*
7017 Rousse, Bulgaria
E-mail: [1]bjarne.stig.andersen@uni-c.dk, [2]gustav@watson.ibm.com
E-mail: [3]alex@uni-c.dk, [4]jerzy.wasniewski@uni-c.dk
E-mail: [5]yalamov@ami.ru.acad.bg

Received September 24, 1999

ABSTRACT

Recursion leads to automatic variable blocking for dense linear-algebra algorithms. The recursive way of programming algorithms eliminates using BLAS level 2 during the factorization steps. For this and other reasons recursion usually speeds up the algorithms. The Cholesky factorization algorithm for positive definite matrices and *LU* factorization for general matrices are formulated. Different storage data formats and recursive BLAS are explained in this paper. Performance graphes of packed and recursive Cholesky algorithms are presented.

1. INTRODUCTION

This work is a continuation of the work of Gustavson and Toledo described in [5; 9]. These papers describe the application of recursion to the numerical dense linear algebra algorithms. Recursion leads to automatic variable blocking for the dense linear-algebra algorithms. This leads to modifications of the LAPACK [1] algorithms. LAPACK's level-2 version routines are transformed into level-3 codes by using recursion.

Fortran 90 allows recursion (see [8]). The programs are very concise and the recursion part is automatic as it is handled by the compiler. The intermediate subroutines obey the Fortran 90 standard too (see [3]).

Section 2 shows the recursive Cholesky factorization algorithm. Section 3 formulates the recursive algorithm of Gaussian elimination without pivoting and LU factorization with partial pivoting. Section 4 explains two recursive BLAS: RTRSM and RSYRK.

2. CHOLESKY FACTORIZATION

We would like to compute the solution to a system of linear equations $AX = B$, where A is real symmetric or complex Hermitian and, in either case, positive definite matrix, X and B are rectangular matrices or vectors. The Cholesky decomposition can be used to factor A, $A = L\,L^T$ or $A = U^TU$, where U is an upper triangular matrix and L is a lower triangular ($L = U^T$). The factored form of A is then used to solve the system of equations $A\,X = B$.

A recursive algorithm of Cholesky factorization is described in detail in [10; 5]. Here we give the final recursive algorithms for the lower triangular and upper triangular cases. We assume that A is n by n.

Recursive Algorithm 2.1. *Cholesky recursive algorithm if lower triangular part of A is given (rcholesky):*

```
Do recursion
• if $n > 1$ then
  • $L_{11} :=$ rcholesky of $A_{11}$
  • $L_{21}L_{11}^T = A_{21} \;\rightarrow$ RTRSM
  • $\hat{A}_{22} := A_{22} - L_{21}L_{21}^T \;\rightarrow$ RSYRK
  • $L_{22} :=$ rcholesky of $\hat{A}_{22}$
• otherwise
  • $L := \sqrt{A}$
End recursion
```

Recursive Algorithm 2.2. *Cholesky recursive algorithm if upper triangular part of A is given (rcholesky):*

```
Do recursion
• if $n > 1$ then
  • $U_{11} :=$ rcholesky of $A_{11}$
  • $U_{11}^TU_{12} = A_{12} \;\rightarrow$ RTRSM
  • $\hat{A}_{22} := A_{22} - U_{12}^TU_{12} \;\rightarrow$ RSYRK
  • $U_{22} :=$ rcholesky of $\hat{A}_{22}$
• otherwise
  • $U := \sqrt{A}$
End recursion
```

The matrices A_{11}, A_{12}, A_{21}, A_{22}, L_{11}, L_{21}, L_{22}, U_{11}, U_{12} and U_{22} are submatrices of A, L and U respectively.

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad L = \begin{pmatrix} L_{11} & \\ L_{21} & L_{22} \end{pmatrix} \text{ and } U = \begin{pmatrix} U_{11} & U_{12} \\ & U_{22} \end{pmatrix}$$

The sizes of the submatrices are: for A_{11}, L_{11} and U_{11} is $h \times h$, for A_{21} and L_{21} is $(n-h) \times h$, for A_{12} and U_{12} is $h \times (n-h)$, and for A_{22}, L_{22} and U_{22} is $(n-h) \times (n-h)$, where $h = n/2$. Matrices L_{11}, L_{22}, U_{11} and U_{22} are lower and upper triangular respectively. Matrices A_{11}, A_{12}, A_{21}, A_{22}, L_{21} and U_{12} are rectangular.

The RTRSM and RSYRK are recursive BLAS of _TRSM and _SYRK respectively. _TRSM solves a triangular system of equations. _SYRK performs the symmetric rank k operations (see 2.2).

2.1. Full and Packed Storage Data Format

The Cholesky factorization algorithm can be programmed either in "full storage" or "packed storage". For example the LAPACK subroutine POTRF works on full storage, while the routine PPTRF is programmed for packed storage. Here we are interested only in full storage and packed storage holding data that represents dense symmetric positive definite matrices. We will compare our recursive algorithms to the LAPACK POTRF and PPTRF subroutines.

The POTRF subroutine uses the Cholesky algorithm in full storage. A storage for the full array A must be declared even if only $n \times (n+1)/2$ elements of array A are needed and, hence $n \times (n-1)/2$ elements are not touched. The PPTRF subroutine uses the Cholesky algorithm on packed storage. It only needs $n \times (n+1)/2$ memory words. Moreover the POTRF subroutine works fast while the PPTRF subroutine works slow. Why? The routine POTRF is constructed with BLAS level 3, while the PPTRF uses the BLAS level 2. These LAPACK data structures are illustrated by the figs. 1 and 2 respectively.

Lower triangular case

$$\begin{pmatrix} 1 & & & & & & \\ 2 & 9 & & & & & \\ 3 & 10 & 17 & & & & \\ 4 & 11 & 18 & 25 & & & \\ 5 & 12 & 19 & 26 & 33 & & \\ 6 & 13 & 20 & 27 & 34 & 41 & \\ 7 & 14 & 21 & 28 & 35 & 42 & 49 \end{pmatrix}$$

Upper triangular case

$$\begin{pmatrix} 1 & 8 & 15 & 22 & 29 & 36 & 43 \\ & 9 & 16 & 23 & 30 & 37 & 44 \\ & & 17 & 24 & 31 & 38 & 45 \\ & & & 25 & 32 & 39 & 46 \\ & & & & 33 & 40 & 47 \\ & & & & & 41 & 48 \\ & & & & & & 49 \end{pmatrix}$$

Figure 1. The mapping of 7× 7 matrix for the LAPACK Cholesky Algorithm using the full storage.

Lower triangular case

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

Upper triangular case

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

Figure 2. The mapping of 7×7 matrix for the LAPACK Cholesky Algorithm using the packed storage.

Lower triangular case

$$\left(\begin{array}{ccc|cccc} 1 & & & & & & \\ 2 & 4 & & & & & \\ 3 & 5 & 6 & & & & \\ \hline 7 & 11 & 15 & 19 & & & \\ 8 & 12 & 16 & 20 & 21 & & \\ 9 & 13 & 17 & 22 & 24 & 26 & \\ 10 & 14 & 18 & 23 & 25 & 27 & 28 \end{array}\right)$$

Upper triangular case

$$\left(\begin{array}{ccc|cccc} 1 & 2 & 4 & 7 & 10 & 13 & 16 \\ & 3 & 5 & 8 & 11 & 14 & 17 \\ & & 6 & 9 & 12 & 15 & 18 \\ \hline & & & 19 & 20 & 22 & 25 \\ & & & & 21 & 23 & 26 \\ & & & & & 24 & 27 \\ & & & & & & 28 \end{array}\right)$$

Figure 3. The mapping of 7×7 matrix for the LAPACK Cholesky Algorithm using the packed recursive storage.

2.2. The Recursive Storage Data Format

We introduce a new storage data format, the recursive storage data format, using the recursive algorithms 2.1 and 2.2. Like packed data format this recursive storage data format requires $n \times (n+1)/2$ storage for the upper or lower part of the matrix. The recursive storage data format is illustrated in fig. 3. A buffer of the size $p \times (p-1)/2$, where $p = [n/2]$ (integer division), is needed to convert from the LAPACK packed storage data format to the recursive packed storage data format and back. No buffer is needed if data is given in recursive format.

We can apply the BLAS level 3 using the recursive packed storage data data format. The performance of the recursive Cholesky algorithm with the recursive packed data format reaches the performance of the LAPACK POTRF algorithm. A graph with the performance between different packed storages is presented in fig. 4.

The graph in fig. 4 presents seven curves. From the bottom: The first two curves represent LAPACK POTRF performance results for upper and lower case respectively. The third curve represents IBM ESSL performance results

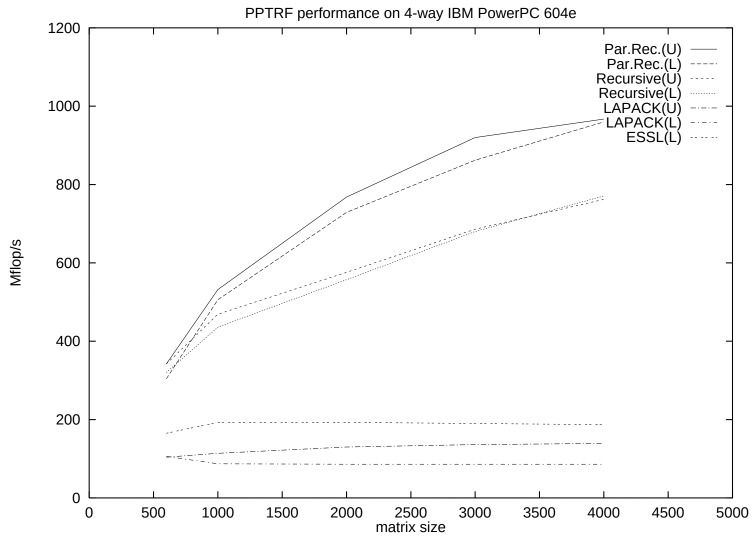

Figure 4. The performance graphs between different packed storage data formats of the Cholesky factorization algorithm run on the 4-way IBM PowerPC 604e computer, using the double precision arithmetic.

of the lower case. The last four curves give the performance of variants of the recursive algorithm. The conversion time from LAPACK packed data format to the recursive packed data format and back is included here. The fourth and fifth curves give performance of the L and U variants without any compiler directives. The sixth and seventh curves give performance of the L and U variants using compiler parallelizing directives. The SMP parallel ESSL DGEMM was used by the last five algorithms.

The same good results were obtained on other parallel supercomputers, for example on Compaq α DS–20 and SGI Origin 2000.

The forward and back substitutions perform better than LAPACK too. The recursive RTRSM is used here.

3. *LU* FACTORIZATION

We would like to compute the solution to a system of linear equations $AX = B$, where A is a real or complex matrix, and X and B are rectangular matrices or vectors. Gaussian elimination with row interchanges is used to factor A as $LU = PA$, where P is a permutation matrix, L is a unit lower triangular matrix, and U is an upper triangular matrix. The factored form of A is then used to solve the system of equations $AX = B$.

The recursive algorithm of the Gauss LU factorization is described in detail in [2; 5]. We give two recursive algorithms here. They are listed in figs. 5 and 6.

The matrices A_1, A_2, A_3, A_{12}, A_{22}, L_1, L_{11}, L_{21}, L_{22}, U_1, U_3, U_{12} and U_{22} are submatrices of A, L and U respectively, $a_{11} \in A_1$.

Recursive Algorithm 3.1. *Recursive LU factorization without pivoting (rgausslu):*

```
Do recursion
• if $min(m,n) > 1$ then
  • $(L_1, U_1)$ = rgausslu of $A_1$
  • $L_{11}U_{12} = A_{12}$ $\rightarrow$ **RTRSM**
  • $\hat{A}_{22} = A_{22} - L_{21}U_{12}$ $\rightarrow$ **_GEMM**
  • $(L_{22}, U_{22})$ = rgausslu of $\hat{A}_{22}$
• otherwise
  • $L_1 := A_1/a_{11}$ and $U_1 = a_{11}$
End recursion
• if $n > m$ then
  • $LU_3 = A_3$ $\rightarrow$ **RTRSM**
```

Figure 5.

Recursive Algorithm 3.2. *Recursive LU=PA factorization with partial pivoting (rgausslu):*

```
Do recursion
• if $\min(m,n) > 1$ then
  • $(P_1, L_1, U_1)$ = rgausslu of $A_1$
  • Forward pivot $A_2$ by $P$ $\rightarrow$ **_LASWP**
  • $L_{11}U_{12} = A_{12}$ $\rightarrow$ **RTRSM**
  • $\hat{A}_{22} := A_{22} - L_{21}U_{12}$ $\rightarrow$ **_GEMM**
  • $(P_2, L_{22}, U_{22})$ = rgausslu of $\hat{A}_{22}$
  • Back pivot $A_1$ by $P_2$ $\rightarrow$ **_LASWP**
  • $P = P_2P_1$
• otherwise
  • pivot $A_1$
  • $L_1 := A_1/a_{11}$ and $U_1 := a_{11}$
End recursion
• if $n > m$ then
  • Forward pivot $A_3$ by $P$ $\rightarrow$ **_LASWP**
  • $LU_3 = A_3$ $\rightarrow$ **RTRSM**
```

where P_1 and P_2 are permutation matrices.

Figure 6.

4. RECURSIVE BLAS AND THEIR PARALLELISM

Two recursive BLAS (Basic Linear Algebra Subprograms, see [12]) subroutines are used in our recursive Cholesky and recursive LU algorithms: RTRSM and RSYRK. These two routines will be explained below.

4.1. RTRSM

RTRSM is a recursive formulation of _TRSM, where _ is a precision and arithmetic indicator: S, D, C or Z. _TRSM subroutine solves one of the matrix equation

$$AX = \alpha B, A^T X = \alpha B, XA = \alpha B, or XA^T = \alpha B,$$

where α is a scalar. X and B are $(m \times n)$ rectangular matrices. A is a unit, or non-unit, upper or lower $(m \times m)$ triangular matrix. The matrix X is overwritten on B. We have 16 different triangular equations because A and A^T can be either upper or lower triangular matrices, and the diagonal is normal or unit.

We will introduce the recursive formulation only for one case $AX = \alpha B$, where A is lower triangular. The other cases will be similar.

The matrices A, B, and X can be partitioned into smaller submatrices, thus

$$\begin{pmatrix} A_{11} & \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{pmatrix} = \alpha \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

The matrices $A_{11} = A(1:h, 1:h)$, $A_{21} = A(h+1:m, 1:h)$, $A_{22} = A(h+1:m, h+1:m)$, $B_{11} = B(1:h, 1:p)$, $B_{12} = B(1:h, p+1:n)$, $B_{21} = B(h+1:m, 1:p)$, $B_{22} = B(h+1:m, p+1:n)$, $X_{11} = X(1:h, 1:p)$, $X_{12} = X(1:h, p+1:n)$, $X_{21} = X(h+1:m, 1:p)$ and $X_{22} = X(h+1:m, p+1:n)$ are submatrices of A, B and X respectively.

Multiplying the matrix A by X gives:

$$\begin{pmatrix} A_{11}X_{11} & A_{11}X_{12} \\ A_{21}X_{11} + A_{22}X_{21} & A_{21}X_{12} + A_{22}X_{22} \end{pmatrix} = \begin{pmatrix} \alpha B_{11} & \alpha B_{12} \\ \alpha B_{21} & \alpha B_{22} \end{pmatrix}.$$

We have got two independent groups of triangular systems:

$$\begin{array}{cc} A_{11}X_{11} = \alpha B_{11} & A_{11}X_{12} = \alpha B_{12} \\ A_{22}X_{21} = \alpha B_{21} - A_{21}X_{11} & A_{22}X_{22} = \alpha B_{22} - A_{21}X_{12} \end{array}$$

We could do a double recursion on m and n; i. e. on h and p. However, we do not do the recursion on p. This results in the following algorithm:

Recursive Algorithm 4.1. *Recursive algorithm for the $AX = B$ operation (one group only), where A is a lower triangular matrix (rtrsm):*

```
    Do recursion
```

```
• if $m > 1$ then
  • $A_{11}X_1 = \alpha B_1 \ \rightarrow$ **RTRSM**
  • $\hat{B}_2 := \alpha B_2 - A_{21}X_1 \ \rightarrow$ **_GEMM**
  • $A_{22}X_2 = \alpha \hat{B}_2 \ \rightarrow$ **RTRSM**
• otherwise
  • $a_{11}X_1 = \alpha B_1$
End recursion
```

4.2. RSYRK

RSYRK is a recursive formulation of _SYRK, where _ is a precision and arithmetic indicator: S, D, C or Z. _SYRK performs one of the symmetric rank k operations:

$$C := \alpha AA^T + \beta C \text{ or } C := \alpha A^T A + \beta C,$$

where α and β are scalars. A is a rectangular matrix $(m \times n)$. C is a square symmetric matrix.

We will introduce the recursive formulation only for one of the four cases of _SYRK:

$$C := \alpha AA^T + \beta C.$$

The matrices A and C can be partitioned into smaller submatrices:

$$\begin{pmatrix} C_{11} & \\ C_{21} & C_{22} \end{pmatrix} = \beta \begin{pmatrix} C_{11} & \\ C_{21} & C_{22} \end{pmatrix} + \alpha \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}^T.$$

The matrices $A_{11} = A(1:h, 1:p)$, $A_{12} = A(1:h, p+1:n)$, $A_{21} = A(h+1:m, 1:p)$, $A_{22} = A(h+1:m, p+1:n)$, $C_{11} = C(1:h, 1:h)$, $C_{21} = C(h+1:m, 1:h)$, $C_{22} = C(h+1:m, h+1:m)$ are submatrices of A and C, respectively. The recursion could be done again on two variables, h and p, but we do recursion on h only.

In terms of the partitioning we have three independent formulas:

$$\begin{array}{l} C_{11} = \beta C_{11} + \alpha A_{11}A_{11}^T + \alpha A_{12}A_{12}^T, \\ C_{21} = \beta C_{21} + \alpha A_{21}A_{11}^T + \alpha A_{22}A_{12}^T, \\ C_{22} = \beta C_{22} + \alpha A_{21}A_{21}^T + \alpha A_{22}A_{22}^T. \end{array}$$

These three computations can be computed in parallel. We now formulate a recursive algorithm as follows:

Recursive Algorithm 4.2. *Recursive algorithm for the $C := \alpha AA^T + \beta C$ symmetric rank k operations (rsyrk):*

```
Do recursion
```

```
• if $m \geq 1$ then
  Perform computation $C_{11}$:
  • $\hat{C}_{11} := \beta C_{11} + \alpha A_{11} A_{11}^T \rightarrow$ **RSYRK**
  • $C_{11} := \hat{C}_{11} + \alpha A_{12} A_{12}^T \rightarrow$ **RSYRK**
  Perform computation $C_{21}$:
  • $\hat{C}_{21} := \beta C_{21} + \alpha A_{21} A_{11}^T \rightarrow$ **_GEMM**
  • $C_{21} := \hat{C}_{21} + \alpha A_{22} A_{12}^T \rightarrow$ **_GEMM**
  Perform computation $C_{22}$:
  • $\hat{C}_{22} := \beta C_{22} + \alpha A_{21} A_{21}^T \rightarrow$ **RSYRK**
  • $C_{22} := \hat{C}_{22} + \alpha A_{22} A_{22}^T \rightarrow$ **RSYRK**
End recursion
```

4.3. A fast _GEMM algorithm

The _ of _GEMM is a precision and arithmetic indicator: S, D, C or Z. _GEMM subroutine does the following operations

$$C := \alpha AB + \beta C, C := \alpha AB^T + \beta C, C := \alpha A^T B + \beta C,$$

$$C := \alpha A^T B^T + \beta C, \text{ or } C := \alpha AB^C + \beta C, C := \alpha A^C B + \beta C$$

$$\text{and } C := \alpha A^C B^C + \beta C,$$

where α and β are scalars. A, B and C are rectangular matrices. A^T, B^T, A^C and B^C are transpose and conjugate matrices respectively.

The GEMM operation is very well documented and explained in [6; 12]. We can see that work is done by _GEMM for both our BLAS RTRSM 4.1 and RSYRK 4.2. The speed of our computation depends very much from the speed of a good _GEMM. Good _GEMM implementations are usually developed by computer manufacturers. The model implementation of _GEMM can be obtained from netlib [12]; it works correctly but slowly. However, an excellent set of high performance BLAS, called _GEMM based BLAS was developed by Bo Kågström at the University of Umeå in Sweden, see for example [7]. A key idea behind _GEMM based BLAS was to cast all BLAS algorithms in terms of the simple BLAS _GEMM. Recently, the Innovative Computing Laboratory at University of Tennessee in Knoxville developed a system called ATLAS (see 4.3.1) which can produce a fast _GEMM program.

4.3.1. ATLAS

Automatically Tuned Linear Algebra Software (ATLAS) [11]. ATLAS is an approach for the automatic generation and optimization of numerical software for processors with deep memory hierarchies and pipelined functional units. The production of such software for machines ranging from desktop workstations to embedded processors can be a tedious and time consuming task. ATLAS has been designed to automate much of this process. So, having a fast

_GEMM means our RTRSM and RSYRK routines will be fast. The ATLAS GEMM is often better than _GEMM developed by the computer manufacture. What is important, the ATLAS software is available to every body, free of charge. Every personal computer can have a good GEMM.

REFERENCES

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

[2] B.S. Andersen, F. Gustavson, J. Waśniewski and P. Yalamov. Recursive formulation of some dense linear algebra algorithms, In: *Proceedings of the* 9^{th} *SIAM Conference on Parallel Processing for Scientific Computing, PPSC99*, B. Hendrickson, K.A. Yelick, C.H. Bischof, I.S. Duff, A.S. Edelman, G.A. Geist, M.T. Heath, M.A. Heroux, C. Koelbel, R.S. Schrieber, R.F. Sincovec, and M.F. Wheeler (Eds.), San Antonio, TX, USA, March 24-27, 1999, SIAM, Scientific Computing, CDROM.

[3] J. Dongarra and J. Waśniewski. High Performance Linear Algebra Package – LAPACK90, In: *Advances in Randomized Parallel Computing, Kluwer Academic Publishers, Combinatorial Optimization Series*, P.M. Pardalos and S. Rajasekaran (Eds.), 1999 and available as the LAPACK Working Note (Lawn) Number 134: http://www.netlib.org/lapack/lawns/lawn134.ps

[4] G.H. Golub and C.F. Van Loan. Matrix Computations (third edition) Johns Hopkins University Press, Baltimore, MD, 1996.

[5] F. Gustavson. Recursive Leads to Automatic Variable Blocking for Dense Linear-Algebra Algorithms. *IBM Journal of Research and Development*, **41** (6), November 1997.

[6] F. Gustavson, A. Henriksson, I. Jonsson, B. Kågström and P. Ling. Recursive Blocked Data Formats and BLAS' for Dense Linear Algebra Algorithms, In: *Proceedings of the* 4^{th} *International Workshop, Applied Parallel Computing, Large Scale Scientific and Industrial Problems, PARA'98*, B. Kågström, J. Dongarra, E. Elmroth, and J. Waśniewski (Eds.), Umeå, Sweden, June 1998, Springer, Lecture Notes in Computer Science Number 1541, 195–206.

[7] B. Kågström, P. Ling and C. Van Loan. *GEMM-based level 3 BLAS: High-performance model implementations and performance evaluation benchmark.* ACM Trans. Math. Software, 1997.

[8] S. Metcalf and J. Reid. *Fortran 90/95 Explained.* Oxford, New York, Tokyo, Oxford University Press, 1996.

[9] S. Toledo. *Locality of Reference in LU Decomposition with Partial Pivoting.* SIAM Journal on Matrix Analysis and Applications, Vol. 18, No. 4, 1997.

[10] J. Waśniewski, B.S. Andersen, and F. Gustavson. Recursive Formulation of Cholesky Algorithm in Fortran 90, In: *Proceedings of the* 4^{th} *International Workshop, Applied Parallel Computing, Large Scale Scientific and Industrial Problems, PARA'98*, B. Kågström, J. Dongarra, E. Elmroth, and J. Waśniewski (Eds.), Umeå, Sweden, June 1998, Springer, Lecture Notes in Computer Science Number 1541, pp. 574–578.

[11] R.C. Whaley and J. Dongarra. Automatically Tuned Linear Algebra Software (ATLAS), In: *Ongoing Projects, The Innovative Computing Laboratory, Distributed Network Computing, Numerical Linear Algebra, Software Repositories, and Performance Evaluation*, http://www.netlib.org/atlas/, Knoxville, Tennessee, USA, 1999.

[12] BLAS (Basic Linear Algebra Subprograms), In: *Ongoing Projects, The Innovative Computing Laboratory, Distributed Network Computing, Numerical Linear Algebra, Software Repositories, and Performance Evaluation*, http://www.netlib.org/blas/, Knoxville, Tennessee, USA, 1999.

LAWRA – REKURSYVINIAI TIESINĖS ALGEBROS ALGORITMAI

B. ANDERSEN, F. GUSTAVSON, A. KARAIVANOV, J. WASNIEWSKI, P. YALAMOV

Rekursyviniai algoritmai leidžia automatiškai parinkti optimalų bloko dydį realizuojant tiesinės algebros algoritmus su pilnomis matricomis. Naudojant rekursyvinį programavimą išvengiama BLAS bibliotekos antrojo lygio paprogramių naudojimo vykdant faktorizacijos ciklą. Dėl šios ir kitų priežasčių rekursyviniai algoritmai dažniausiai yra greitesni už standartinius tiesinės algebros algoritmus. Straipsnyje pateikti Choleckio ir LU išskaidymo rekursyviniai algoritmai. Apibrėžti skirtingi rekursyviniai duomenų saugojimo formatai ir aprašytas naujas BLAS bibliotekos projektas. Pateikiami naujojo rekursyvinio Choleckio išskaidymo algoritmo efektyvumo tyrimo rezultatai, kurie buvo atlikti su įvairių tipų kompiuteriais.