# New Data Storage Formats for Dense Matrices Lead to Variety of High-Performance Algorithms (Solving Linear Systems of Equations)

**Jerzy Waśniewski**

**Emeritus Senior Research Professor**

**Department of Informatics & Mathematical Modeling**

**Technical University of Denmark**

**DTU, Bldg. 305**

**DK - 2800 Lyngby, Denmark**

**e-mail: jw@imm.dtu.dk**

**http://www.imm.dtu.dk/~jw/Lectures/ppam05.pdf**

**August 14, 2005**

# Outline of my Talk:

1. Introduction
2. First Step
   - Cholesky and Gauss
3. Symm. Pos. Def. Matrices
   (a) LAPACK storages
   - Full storage
   - Packed storage
   - Performance results

   (b) New data formats
   - Recursive storage
     – Algorithm
     – Recursive BLAS
     – Performance results
   - Hybrid storage
     – Performance results
4. Symm. Indef. Matrices
   (a) Perturbation Approach
   (b) Packed Block Storage
   - Data format
   - Performance results
5. More results

# Introduction

# Starting references:

**Fred G. Gustavson: "Recursive Leads to Automatic Variable Blocking for Dense Linear-Algebra Algorithms", IBM Journal of Research and Development, Volume 41, Number 6, November 1997.**

**Sivan Toledo: "Locality of Reference in *LU* Decomposition with Partial Pivoting", SIAM Journal on Matrix Analysis and Applications, Vol. 18, No. 4, 1997.**

# Cholesky and Gauss Factorizations

# $LL^T = U^T U$ and $LU$

# Full storage data format

# First step of our work

## $LL^T = U^T U$ Cholesky Decomposition

$$AX = B$$

- $A$ **– a symmetric or Hermitian, positive definite**
- $X$ **and** $B$ **– rectangular matrices or vectors**

⋆ $A = U^T U$**, if upper triangular part of** $A$ **is given**

⋆ $A = L\,L^T$**, if lower triangular part of** $A$ **is given**

⋆ $U$ **– an upper triangular matrix**

⋆ $L$ **– a lower triangular matrix**

⋆ $L = U^T$ **and** $U = L^T$

⋆ **The factored form of** $A$ **is then used to** $A\,X = B$

## $LU$ Gauss Decomposition

$$AX = B$$

- $A$ **is a general real or complex matrix**
- $X$ **and** $B$ **are rectangular matrices or vectors**

⋆ $A = P^T L\, U$

⋆ $P$ **is a permutation matrix**

⋆ $U$ **is an upper triangular matrix**

⋆ $L$ **is a lower triangular matrix**

⋆ **The factored form of** $A$ **is then used to solve** $A\, X = B$

# $LL^T = U^T U$ Cholesky Factorization

# Full storage data format

# The first step of our work

## $LL^T = U^T U$ Cholesky Decomposition

$$AX = B$$

- $A$ **– a symmetric or Hermitian, positive definite**
- $X$ **and** $B$ **– rectangular matrices or vectors**

⋆ $A = U^T U$**, if upper triangular part of** $A$ **is given**

⋆ $A = L\,L^T$**, if lower triangular part of** $A$ **is given**

⋆ $U$ **– an upper triangular matrix**

⋆ $L$ **– a lower triangular matrix**

⋆ $L = U^T$ **and** $U = L^T$

⋆ **The factored form of** $A$ **is then used to** $A\,X = B$

**Cholesky:** $A = LL^T$

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

$$A = \begin{pmatrix} L_{11} & \\ L_{21} & L_{22} \end{pmatrix} \times \begin{pmatrix} L_{11}^T & L_{21}^T \\ & L_{22}^T \end{pmatrix}$$

$$A = \begin{pmatrix} A_{11} & A_{21} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11}L_{11}^T & L_{11}L_{21}^T \\ L_{21}L_{11}^T & L_{21}L_{21}^T + L_{22}L_{22}^T \end{pmatrix}$$

$$A_{11} = L_{11}L_{11}^T, \quad L_{21}L_{11}^T = A_{21} \quad \text{and} \quad \hat{A}_{22} = L_{22}L_{22}^T$$

**where** $\hat{A}_{22} := A_{22} - L_{21}L_{21}^T$

**Cholesky:** $A = LL^T$

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

**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**
  - $L := \sqrt{A_{11}}$

**End recursion**

**Cholesky:** $A = U^T U$

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

**Do recursion**

- **if** $n > 1$ **then**
  - $U_{11} :=$ **rcholesky of** $A_{11}$
  - $U_{11}^T U_{12} = A_{12} \rightarrow$ **RTRSM**
  - $\hat{A}_{22} := A_{22} - U_{12}^T U_{12} \rightarrow$ **RSYRK**
  - $U_{22} :=$ **rcholesky of** $\hat{A}_{22}$
- **otherwise**
  - $U := \sqrt{A_{11}}$

**End recursion**

# Cholesky

```
IF (N == 1) THEN; A(1,1) = SQRT(A(1,1))
ELSE IF( N > 0 )THEN; H=N/2
 IF( LSAME(LUPLO,'L') )THEN
  CALL RCF( A(1:H,1:H), LUPLO, LINFO)
  CALL RTRSM( A(1:H,1:H), A(H+1:N,1:H), &
    UPLO=LUPLO, SIDE='R', TRANSA='T')
  CALL RSYRK( A(H+1:N,1:H), A(H+1:N,H+1:N), &
    ALPHA=-ONE, UPLOC=LUPLO )
  CALL RCF( A(H+1:N,H+1:N), LUPLO, LINFO)
 ELSE
  CALL RCF( A(1:H,1:H), LUPLO, LINFO )
  CALL RTRSM( A(1:H,1:H), A(1:H,H+1:N), TRANSA='T')
  CALL RSYRK( A(1:H,H+1:N), &
    A(H+1:N,H+1:N), ALPHA=-ONE, TRANSA='T')
  CALL RCF( A(H+1:N,H+1:N), LUPLO, LINFO)
 ENDIF
ENDIF
```

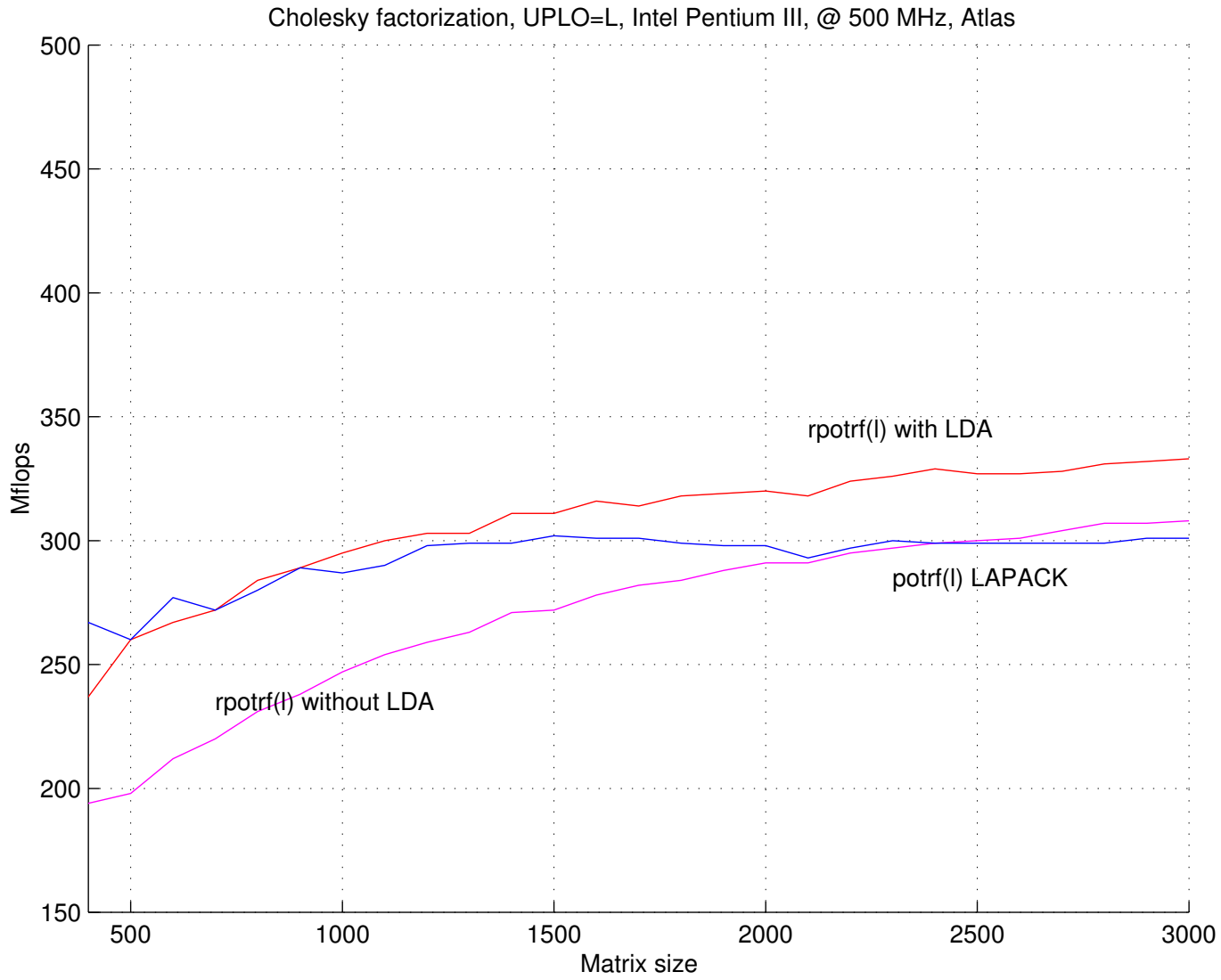
## Cholesky

```
RECURSIVE SUBROUTINE RPOTRF( A, UPLO, INFO)
 USE LA_PRECISION, ONLY: WP => DP
 USE LA_AUXMOD, ONLY: ERINFO, LSAME
 USE F90_RCF, ONLY: RCF => RPOTRF, &
    RTRSM, RSYRK
 IMPLICIT NONE
 CHARACTER(LEN=1), OPTIONAL, INTENT(IN) ::  UPLO
 INTEGER, OPTIONAL, INTENT(OUT) ::  INFO
 REAL(WP), INTENT(INOUT) ::  A(:,:)
    ...   ...  Locals declaration ...   ...
    ...   ...  Testing arguments ...   ...
    ...   ...  Recursive block ...   ...
 CALL ERINFO(LINFO, SRNAME, INFO)
END SUBROUTINE RPOTRF
```

## Cholesky

```
RECURSIVE SUBROUTINE RPOTRF( A, LDA, UPLO, INFO)
 USE LA_PRECISION, ONLY: WP => DP
 USE LA_AUXMOD, ONLY: ERINFO, LSAME
 USE F90_RCF, ONLY: RCF => RPOTRF, &
    RTRSM, RSYRK
 IMPLICIT NONE
 CHARACTER(LEN=1), OPTIONAL, INTENT(IN) ::  UPLO
 INTEGER, OPTIONAL, INTENT(OUT) ::  INFO
 INTEGER, INTENT(IN) ::  LDA
 REAL(WP), INTENT(INOUT) ::  A(:,LDA)
    ...   ...  Locals declaration ...   ...
    ...   ...  Testing arguments ...   ...
    ...   ...  Recursive block ...   ...
 CALL ERINFO(LINFO, SRNAME, INFO)
END SUBROUTINE RPOTRF
```

# Cholesky


Cholesky factorization, UPLO=L, Intel Pentium III, @ 500 MHz, Atlas
Mflops
Matrix size
500
450
400
350
300
250
200
150
500
1000
1500
2000
2500
3000
rpotrf(l) with LDA
potrf(l) LAPACK
rpotrf(l) without LDA

## Reference


- **J. Waśniewski, B.S. Andersen and F. Gustavson. "Recursive Formulation of Cholesky Algorithm in Fortran 90". Proceedings of the Workshop on Applied Parallel Computing, Large Scale Scientific and Industrial Problems, PARA'98, 1998, Umeå, Sweden, Lecture Notes in Computer Science Number 1541, Springer, June, pages 574–578.**

# *LU* Gauss Factorization

## $LU$ Gauss Decomposition

$$AX = B$$

- $A$ **is a general real or complex matrix**
- $X$ **and** $B$ **are rectangular matrices or vectors**

⋆ $A = P^T L U$

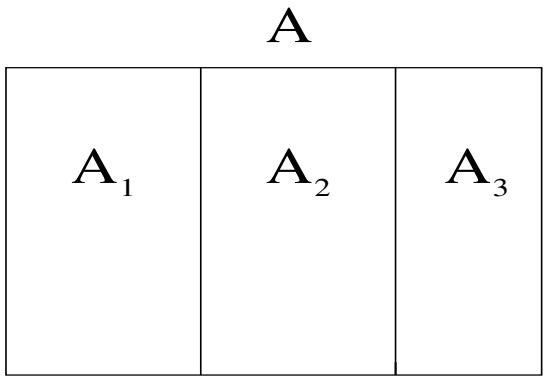
⋆ $P$ **is a permutation matrix**

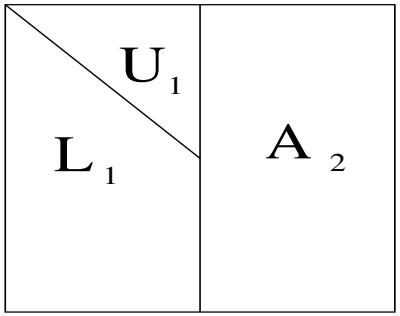
⋆ $U$ **is an upper triangular matrix**

⋆ $L$ **is a lower triangular matrix**

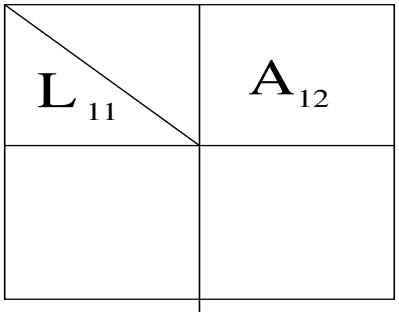
⋆ **The factored form of** $A$ **is then used to solve** $A X = B$

# *LU* Gauss Decomposition

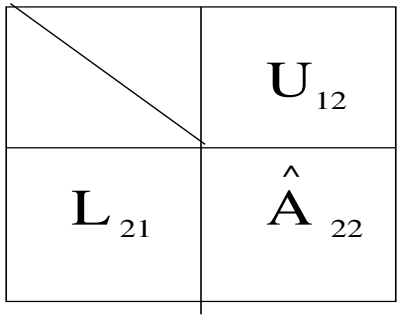


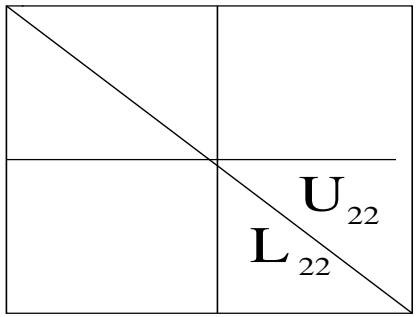

$$(L_1, U_1) = LU(A_1)$$



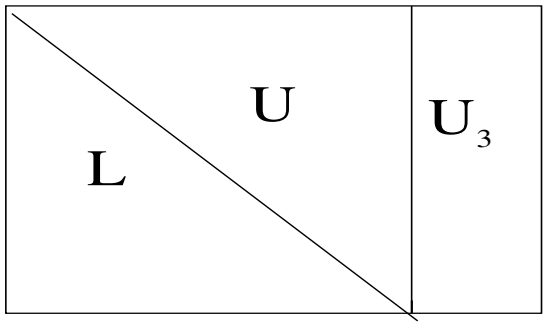

$$L_{11} U_{12} = A_{12}$$




$$\hat{A}_{22} := A_{22} - L_{21} U_{12}$$




$$(L_{22}, U_{22}) := LU(\hat{A}_{22})$$




$$L\, U_3 = A_3$$

**The Recursive Algorithm without pivoting**

## $LU$ Gauss Decomposition

**Do recursion**

- **if** $n > 1$ **then**
  - $(L_1, U_1)$ **:= rgausslu**$(A_1)$
  - $L_{11}U_{12} = A_{12} \rightarrow$ **RTRSM**
  - $\hat{A}_{22} := A_{22} - L_{21}U_{12} \rightarrow$ **GEMM**
  - $(L_{22}, U_{22})$ **:= rgausslu**$(\hat{A}_{22})$
- **otherwise**
  - $L_1 := A_1/a_{11}$

**End recursion**

- **if** $n > m$ **then**
  - $L\,U_3 = A_3 \rightarrow$ **RTRSM**

**The Recursive Algorithm without pivoting**

## $LU$ Gauss Decomposition

**The Recursive Algorithm with partial pivoting:**

**Do recursion**

- **if** $n > 1$ **then**
  - $(P_1, L_1, U_1) =$ **rgausslu**$(A_1)$
  - **Forward pivot** $A_2$ **by** $P_1 \rightarrow$ **LASWP**
  - $L_{11}U_{12} = A_{12} \rightarrow$ **R TRSM**
  - $\hat{A}_{22} := A_{22} - L_{21}U_{12} \rightarrow$ **GEMM**
  - $(P_2, L_{22}, U_{22}) =$ **rgausslu**$(\hat{A}_{22})$
  - **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.**

## References


- **B.S. Andersen, F. Gustavson, A. Karaivanov, J. Waśniewski, and P.Y. Yalamov. "LAWRA – Linear Algebra with Recursive Algorithms". Proceedings of the Conference on Parallel Processing and Applied Mathematics, PPAM'99, September, 1999, Kazimierz Dolny, Poland. Published by the Technical University of Częstochowa, pages 63–76.**
- **Presented at the SIAM Conference on Parallel Processing and Scientific Computing, 1999, San Antonio, USA.**
- **Presented at the PARA2000 Workshop on Applied Parallel Computing, 2000, Bergen, Norway.**

$$LL^T = U^T U$$

# Cholesky Factorization

# LAPACK Algorithms and Data Formats

# Symmetric/Hermitian matrix

$$\mathbf{n} = 7, \quad \mathbf{lda} = 9, \quad \mathbf{memory\ needed} = \mathbf{lda} \times \mathbf{n} = 63$$

$$\begin{pmatrix}
a_{1,1_1} & \diamond & \diamond & \diamond & \diamond & \diamond & \diamond \\
a_{2,1_2} & a_{2,2_{11}} & \diamond & \diamond & \diamond & \diamond & \diamond \\
a_{3,1_3} & a_{3,2_{12}} & a_{3,3_{21}} & \diamond & \diamond & \diamond & \diamond \\
a_{4,1_4} & a_{4,2_{13}} & a_{4,3_{22}} & a_{4,4_{31}} & \diamond & \diamond & \diamond \\
a_{5,1_5} & a_{5,2_{14}} & a_{5,3_{23}} & a_{5,4_{32}} & a_{5,5_{41}} & \diamond & \diamond \\
a_{6,1_6} & a_{6,2_{15}} & a_{6,3_{24}} & a_{6,4_{33}} & a_{6,5_{42}} & a_{6,6_{51}} & \diamond \\
a_{7,1_7} & a_{7,2_{16}} & a_{7,3_{25}} & a_{7,4_{34}} & a_{7,5_{43}} & a_{7,6_{52}} & a_{7,7_{61}} \\
\oslash & \oslash & \oslash & \oslash & \oslash & \oslash & \oslash \\
\oslash & \oslash & \oslash & \oslash & \oslash & \oslash & \oslash
\end{pmatrix}$$

**The mapping of 7× 7 real symmetric or complex Hermitian matrix for the LAPACK algorithm using the full storage. Lower triangular case.**

# Symmetric/Hermitian matrix

$$\mathbf{n} = 7, \quad \mathbf{lda} = 9, \quad \mathbf{memory\ needed} = \mathbf{lda} \times \mathbf{n} = 63$$

$$\begin{pmatrix}
a_{1,1_1} & a_{1,2_{10}} & a_{1,3_{19}} & a_{1,4_{28}} & a_{1,5_{37}} & a_{1,6_{46}} & a_{1,7_{55}} \\
\diamond & a_{2,2_{11}} & a_{2,3_{20}} & a_{2,4_{29}} & a_{2,5_{38}} & a_{2,6_{47}} & a_{2,7_{56}} \\
\diamond & \diamond & a_{3,3_{21}} & a_{3,4_{30}} & a_{3,5_{39}} & a_{3,6_{48}} & a_{3,7_{57}} \\
\diamond & \diamond & \diamond & a_{4,4_{31}} & a_{4,5_{40}} & a_{4,6_{49}} & a_{4,7_{58}} \\
\diamond & \diamond & \diamond & \diamond & a_{5,5_{41}} & a_{5,6_{50}} & a_{5,7_{59}} \\
\diamond & \diamond & \diamond & \diamond & \diamond & a_{6,6_{51}} & a_{6,7_{60}} \\
\diamond & \diamond & \diamond & \diamond & \diamond & \diamond & a_{7,7_{61}} \\
\oslash & \oslash & \oslash & \oslash & \oslash & \oslash & \oslash \\
\oslash & \oslash & \oslash & \oslash & \oslash & \oslash & \oslash
\end{pmatrix}$$

**The mapping of 7× 7 real symmetric or complex Hermitian matrix for the LAPACK algorithm using the full storage. Upper triangular case.**

# Symmetric/Hermitian matrix

$$\mathbf{n} = 7, \quad \mathbf{memory\ needed} = \mathbf{n} \times (\mathbf{n}+1)/2 = 28$$

$$\begin{pmatrix}
a_{1,1_1} & & & & & & \\
a_{2,1_2} & a_{2,2_8} & & & & & \\
a_{3,1_3} & a_{3,2_9} & a_{3,3_{14}} & & & & \\
a_{4,1_4} & a_{4,2_{10}} & a_{4,3_{15}} & a_{4,4_{19}} & & & \\
a_{5,1_5} & a_{5,2_{11}} & a_{5,3_{16}} & a_{5,4_{20}} & a_{5,5_{23}} & & \\
a_{6,1_6} & a_{6,2_{12}} & a_{6,3_{17}} & a_{6,4_{21}} & a_{6,5_{24}} & a_{6,6_{26}} & \\
a_{7,1_7} & a_{7,2_{13}} & a_{7,3_{18}} & a_{7,4_{22}} & a_{7,5_{25}} & a_{7,6_{27}} & a_{7,7_{28}}
\end{pmatrix}$$

**The mapping of 7$\times$ 7 real symmetric or complex Hermitian matrix for the LAPACK algorithm using the packed storage. Lower triangular case.**

## Example; packed data storage

$$A = \begin{pmatrix} \underline{a_{11}} & a_{12} & a_{13} & a_{14} \\ a_{21} & \underline{a_{22}} & a_{23} & a_{24} \\ a_{31} & a_{32} & \underline{a_{33}} & a_{34} \\ a_{41} & a_{42} & a_{43} & \underline{a_{44}} \end{pmatrix}$$

$a_{ij} = \mathsf{conjg}(a_{ji})$ for $i, j = 1, \cdots, 4.$

**UPLO = 'U'**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $a_{11}$ | $a_{12}$ | $a_{22}$ | $a_{13}$ | $a_{23}$ | $a_{33}$ | $a_{14}$ | $a_{24}$ | $a_{34}$ | $a_{44}$ |

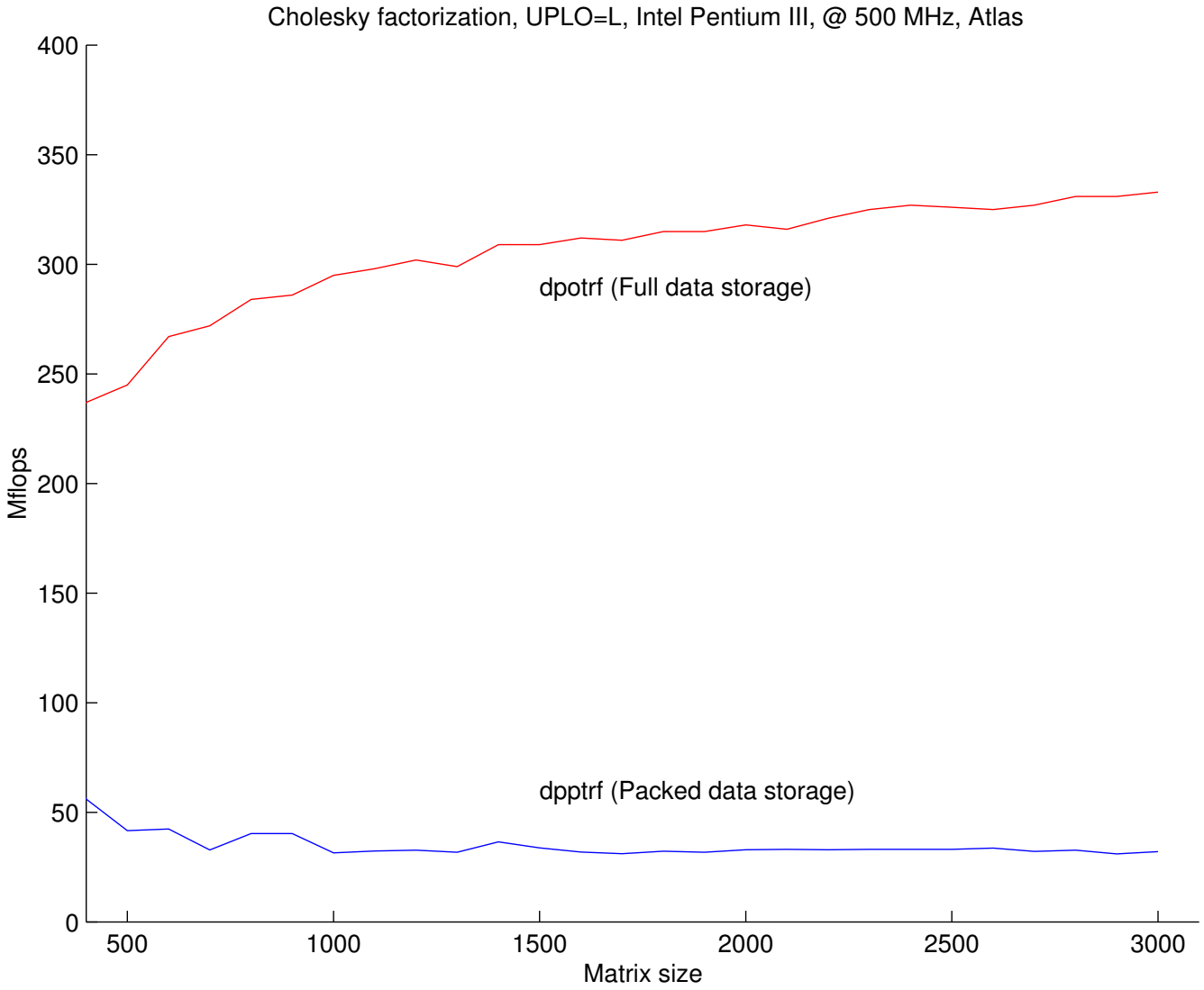
**UPLO = 'L'**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $a_{11}$ | $a_{21}$ | $a_{31}$ | $a_{41}$ | $a_{22}$ | $a_{32}$ | $a_{42}$ | $a_{33}$ | $a_{43}$ | $a_{44}$ |

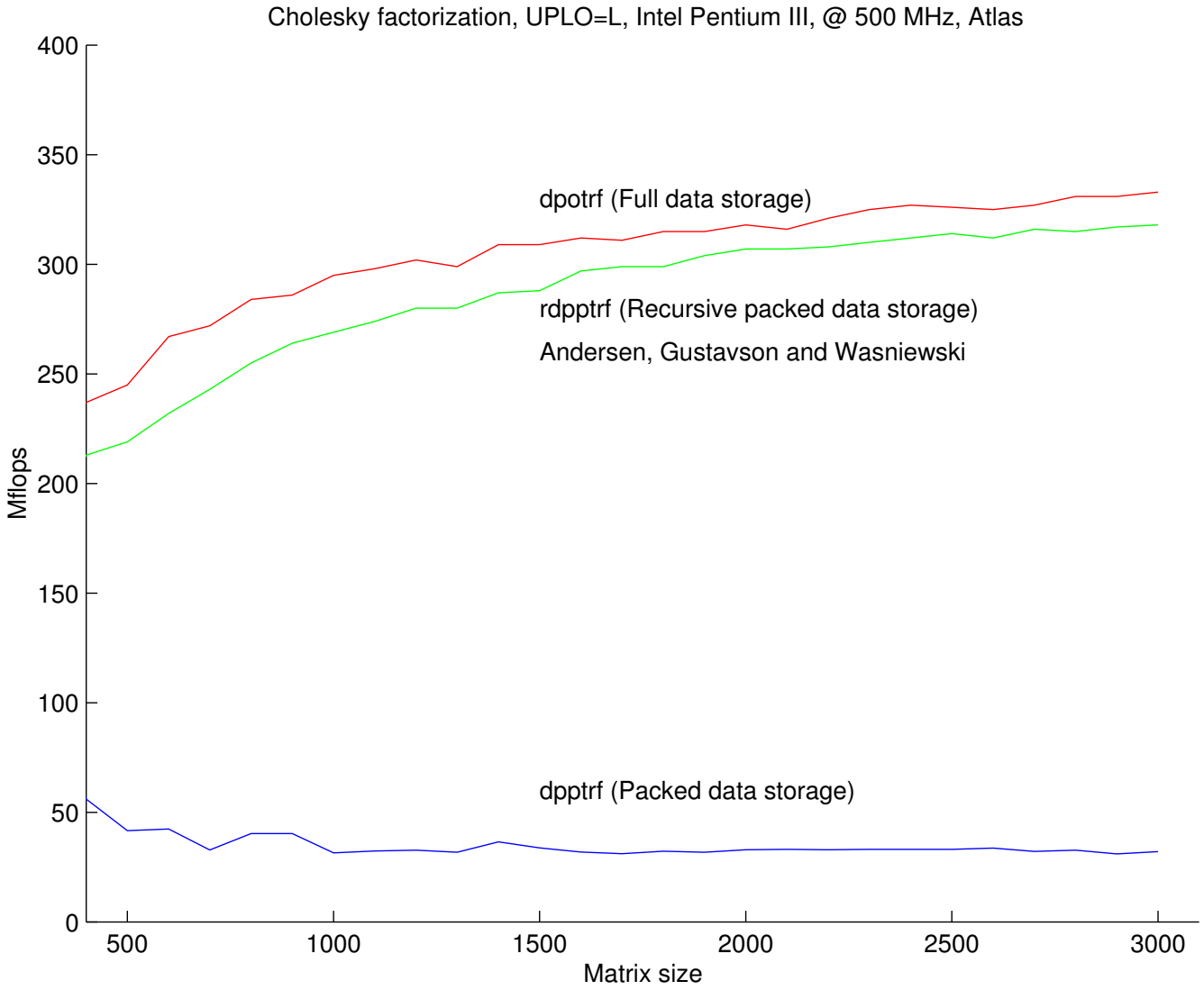
$size(\mathbf{AP}) = n(n+1)/2 = 10$

**If** $n = 1000$ **then** $size(\mathbf{AP}) = 500500$**, saving** $499500$

# Cholesky, a packed and full data storage



# Cholesky, a packed and full data storage

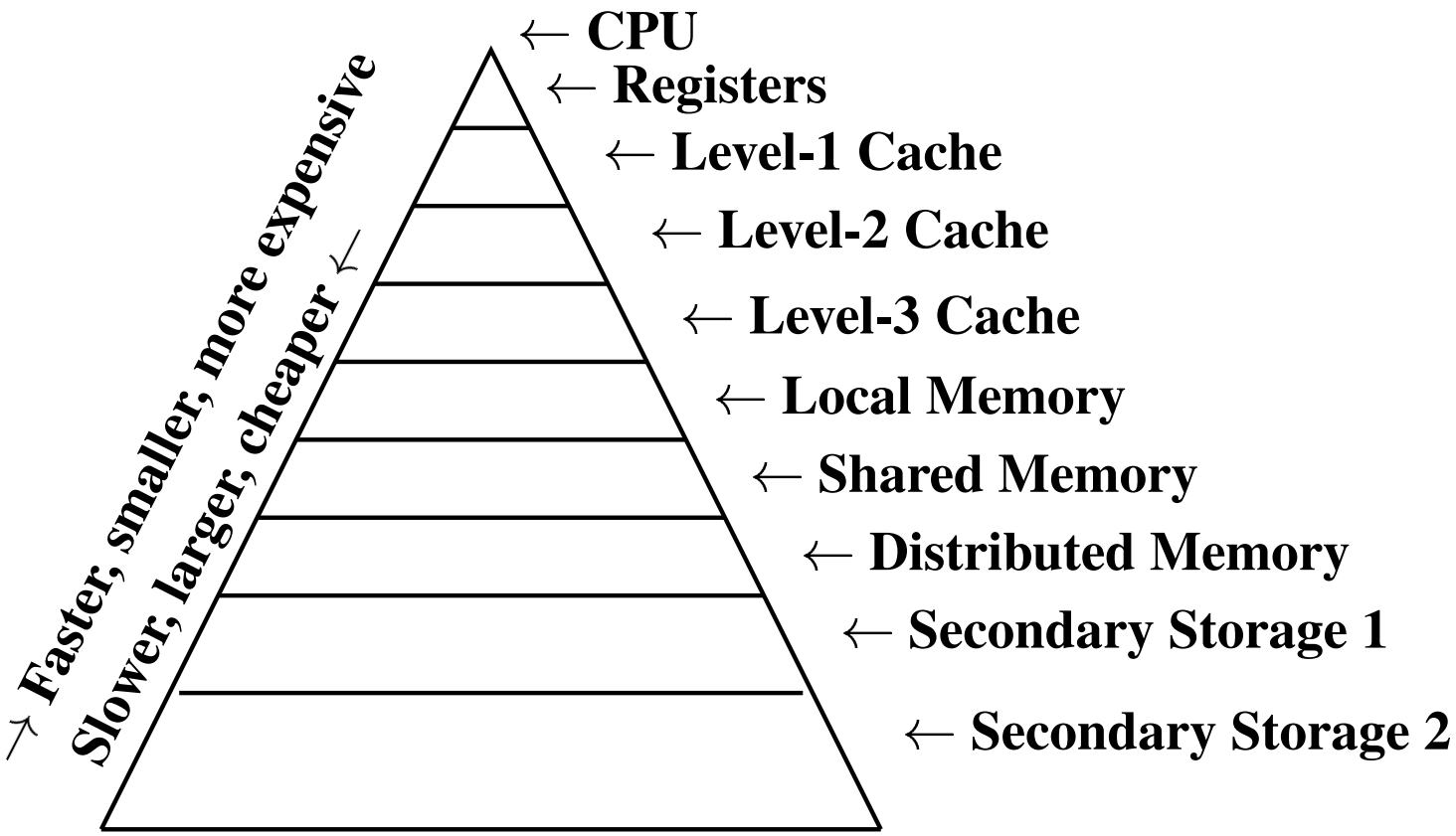


# New Algorithms

## Collaboration:

- **Bjarne S. Andersen**
  **Danish Meteorological Institute, Denmark**
- **John A. Gunnels and Fred Gustavson**
  **IBM Research Center, Yorktown Heights, NY**
- **J.K. Reid, Atlas Centre,**
  **Rutherford Appleton Laboratory, UK**
- **A. Karaivanov, M. Marinova, and**
  **P. Yalamov, Rousse University, Bulgaria**
- **Jack Dongarra (discussions)**
  **University of Tennessee, Knoxville, TN, USA**

# A computer memory hierarchy



# Symmetric/Hermitian Positive Definite Matrices

# Recursive Packed Storage Data Format

**Cholesky:** $A \approx LL^T$

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

$$A = \begin{pmatrix} L_{11} & \\ L_{21} & L_{22} \end{pmatrix} \times \begin{pmatrix} L_{11}^T & L_{21}^T \\ & L_{22}^T \end{pmatrix}$$

$$A = \begin{pmatrix} A_{11} & A_{21} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11}L_{11}^T & L_{11}L_{21}^T \\ L_{21}L_{11}^T & L_{21}L_{21}^T + L_{22}L_{22}^T \end{pmatrix}$$

$$A_{11} = L_{11}L_{11}^T, \quad L_{21}L_{11}^T = A_{21} \quad \text{and} \quad \hat{A}_{22} = L_{22}L_{22}^T$$

**where** $\hat{A}_{22} := A_{22} - L_{21}L_{21}^T$

**Cholesky:** $A \approx LL^T$

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

**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**
  - $L := \sqrt{A_{11}}$

**End recursion**

# Cholesky, Packed Storage

$$\mathbf{n} = 7, \quad \textbf{memory needed} = \mathbf{n} \times (\mathbf{n}+1)/2 = 28$$

$$\begin{pmatrix}
a_{1,1_1} & & & & & & \\
a_{2,1_2} & a_{2,2_8} & & & & & \\
a_{3,1_3} & a_{3,2_9} & a_{3,3_{14}} & & & & \\
a_{4,1_4} & a_{4,2_{10}} & a_{4,3_{15}} & a_{4,4_{19}} & & & \\
a_{5,1_5} & a_{5,2_{11}} & a_{5,3_{16}} & a_{5,4_{20}} & a_{5,5_{23}} & & \\
a_{6,1_6} & a_{6,2_{12}} & a_{6,3_{17}} & a_{6,4_{21}} & a_{6,5_{24}} & a_{6,6_{26}} & \\
a_{7,1_7} & a_{7,2_{13}} & a_{7,3_{18}} & a_{7,4_{22}} & a_{7,5_{25}} & a_{7,6_{27}} & a_{7,7_{28}}
\end{pmatrix}$$

**The mapping to array-subscript order of a $7 \times 7$ matrix for LAPACK Cholesky Algorithm using packed storage. Lower triangular case.**

# Cholesky, Recursive Packed Storage

$$\mathbf{n} = 7, \quad \textbf{memory needed} = \mathbf{n} \times (\mathbf{n}+1)/2 = 28$$

$$\left(\begin{array}{ccc|cccc}
a_{1,1_1} & & & & & & \\
a_{2,1_2} & a_{2,2_4} & & & & & \\
a_{3,1_3} & a_{3,2_5} & a_{3,3_6} & & & & \\
\hline
a_{4,1_7} & a_{4,2_{11}} & a_{4,3_{15}} & a_{4,4_{19}} & & & \\
a_{5,1_8} & a_{5,2_{12}} & a_{5,3_{16}} & a_{5,4_{20}} & a_{5,5_{23}} & & \\
a_{6,1_9} & a_{6,2_{13}} & a_{6,3_{17}} & a_{6,4_{21}} & a_{6,5_{24}} & a_{6,6_{26}} & \\
a_{7,1_{10}} & a_{7,2_{14}} & a_{7,3_{18}} & a_{7,4_{22}} & a_{7,5_{25}} & a_{7,6_{27}} & a_{7,7_{28}}
\end{array}\right)$$

**The mapping to array-subscript order of a $7 \times 7$ matrix for the Cholesky Algorithm using the recursive packed storage. The recursive block division is illustrated. Lower triangular case.**

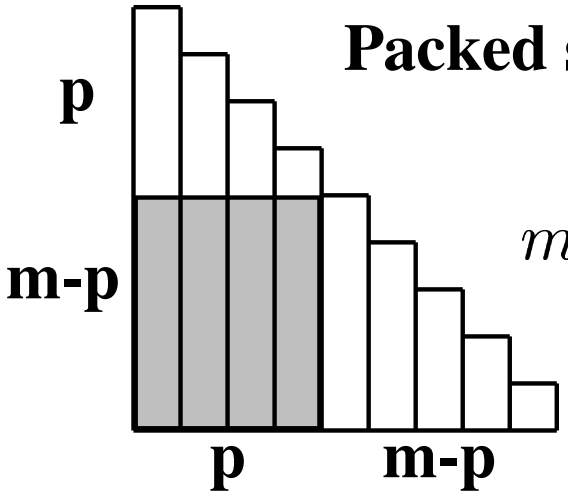
# Cholesky, Recursive Packed Storage

$$\mathbf{n} = 7, \quad \mathbf{memory\ needed} = \mathbf{n} \times (\mathbf{n}+1)/2 = 28$$

$$\left(\begin{array}{ccc|ccc|cc|c}
a_{1,1_1} & & & & & & & & \\
\hline
a_{2,1_2} & a_{2,2_4} & & & & & & & \\
\hline
a_{3,1_3} & a_{3,2_5} & a_{3,3_6} & & & & & & \\
\hline
a_{4,1_7} & a_{4,2_{11}} & a_{4,3_{15}} & a_{4,4_{19}} & & & & & \\
\hline
a_{5,1_8} & a_{5,2_{12}} & a_{5,3_{16}} & a_{5,4_{20}} & a_{5,5_{21}} & & & & \\
\hline
a_{6,1_9} & a_{6,2_{13}} & a_{6,3_{17}} & a_{6,4_{22}} & a_{6,5_{24}} & a_{6,6_{26}} & & & \\
\hline
a_{7,1_{10}} & a_{7,2_{14}} & a_{7,3_{18}} & a_{7,4_{23}} & a_{7,5_{25}} & a_{7,6_{27}} & a_{7,7_{28}} & &
\end{array}\right)$$

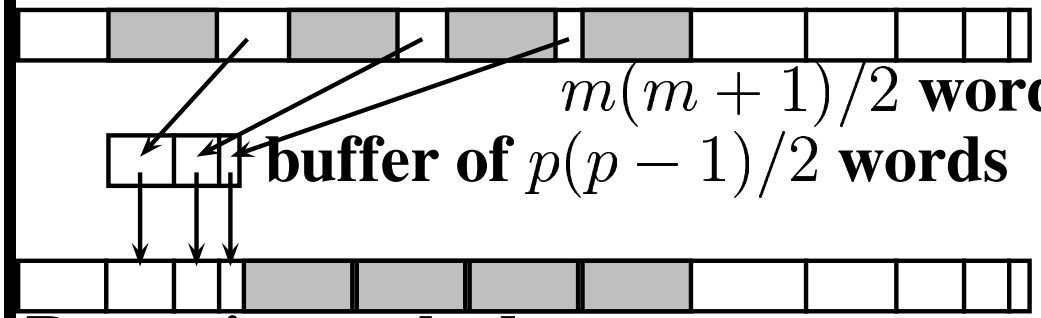
**The mapping to array-subscript order of a $7 \times 7$ matrix for the Cholesky Algorithm using the recursive packed storage. The recursive block division is illustrated. Lower triangular case.**

# Cholesky, Recursive Packed Storage




$m = 9, \quad p = [m/2] = 4$

**LAPACK packed storage memory map**



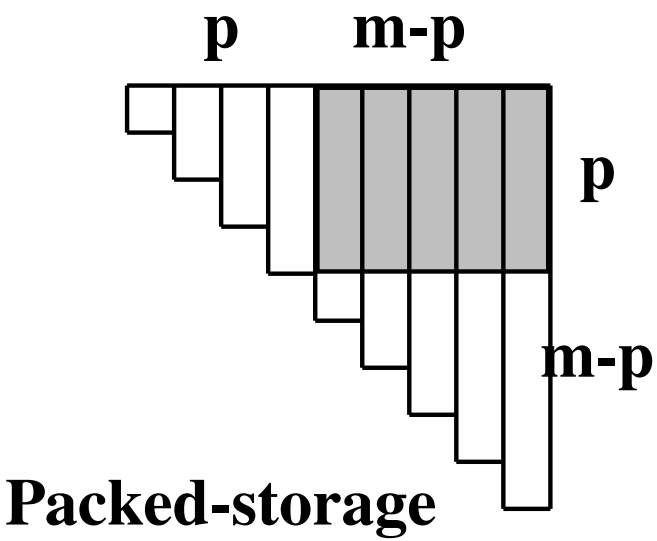
$m(m+1)/2$ **words**

**buffer of** $p(p-1)/2$ **words**

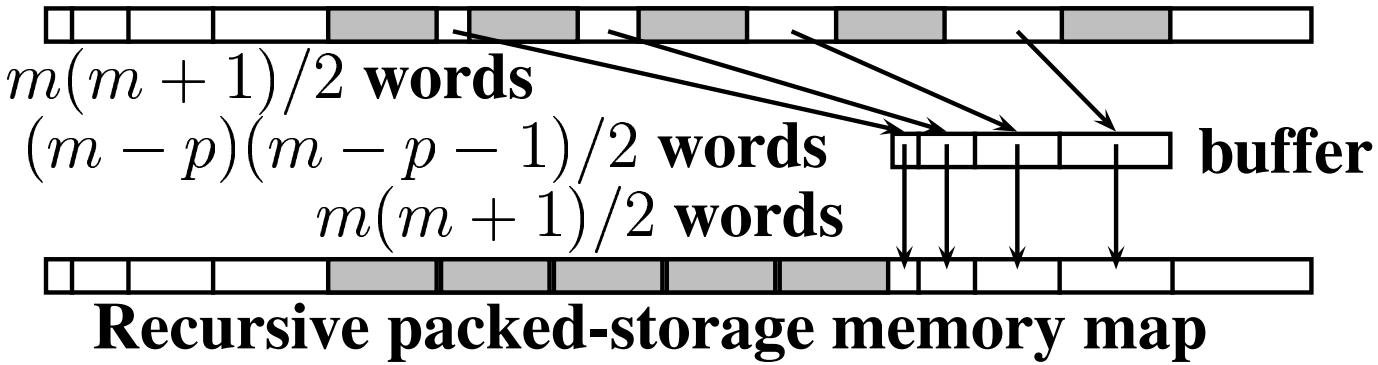
**Recursive packed storage memory map**

$m(m+1)/2$ **words**

## Cholesky, Recursive Packed Storage




**LAPACK packed-storage memory map**




**Recursive packed-storage memory map**

**Cholesky:** $A \approx LL^T$

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

**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**
  - $L := \sqrt{A_{11}}$

**End recursion**

## The Recursive RTRSM and RSYRK

**By simple algebraic manipulations**

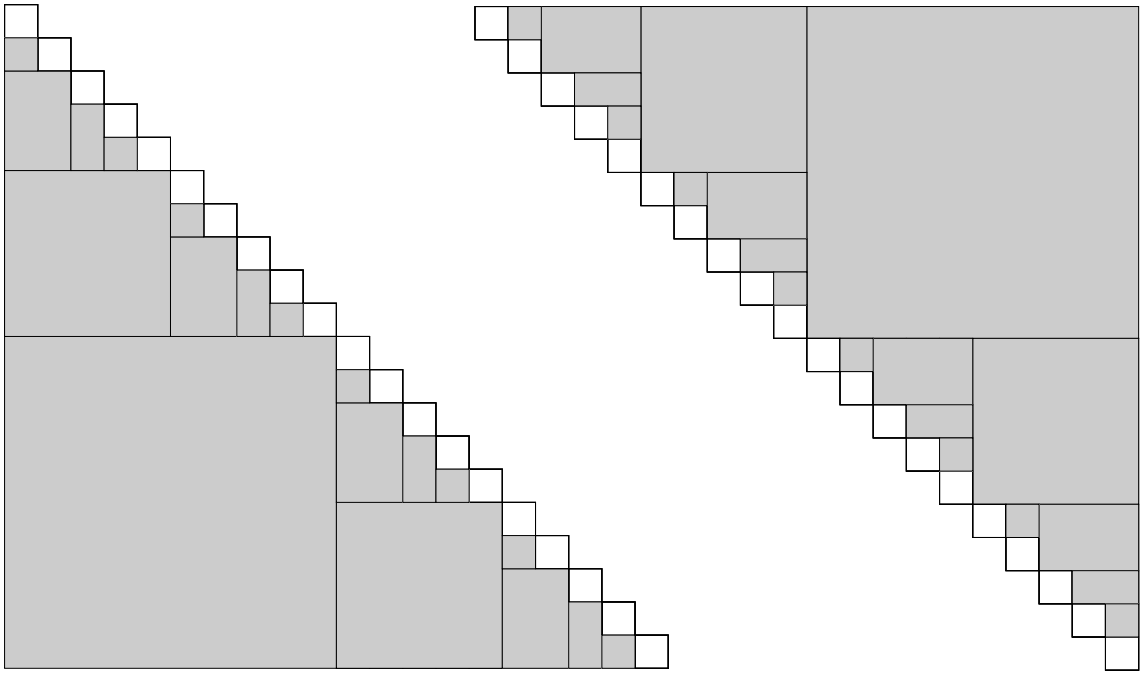
# RTRSM:

$$
\begin{aligned}
X_{11}A_{11}^T &= \alpha B_{11} && \textbf{RTRSM} \\
\hat{B}_{12} &= B_{12} - \alpha^{-1} X_{11} A_{21}^T && \textbf{GEMM} \\
X_{12}A_{22}^T &= \alpha \hat{B}_{12} && \textbf{RTRSM}
\end{aligned}
$$

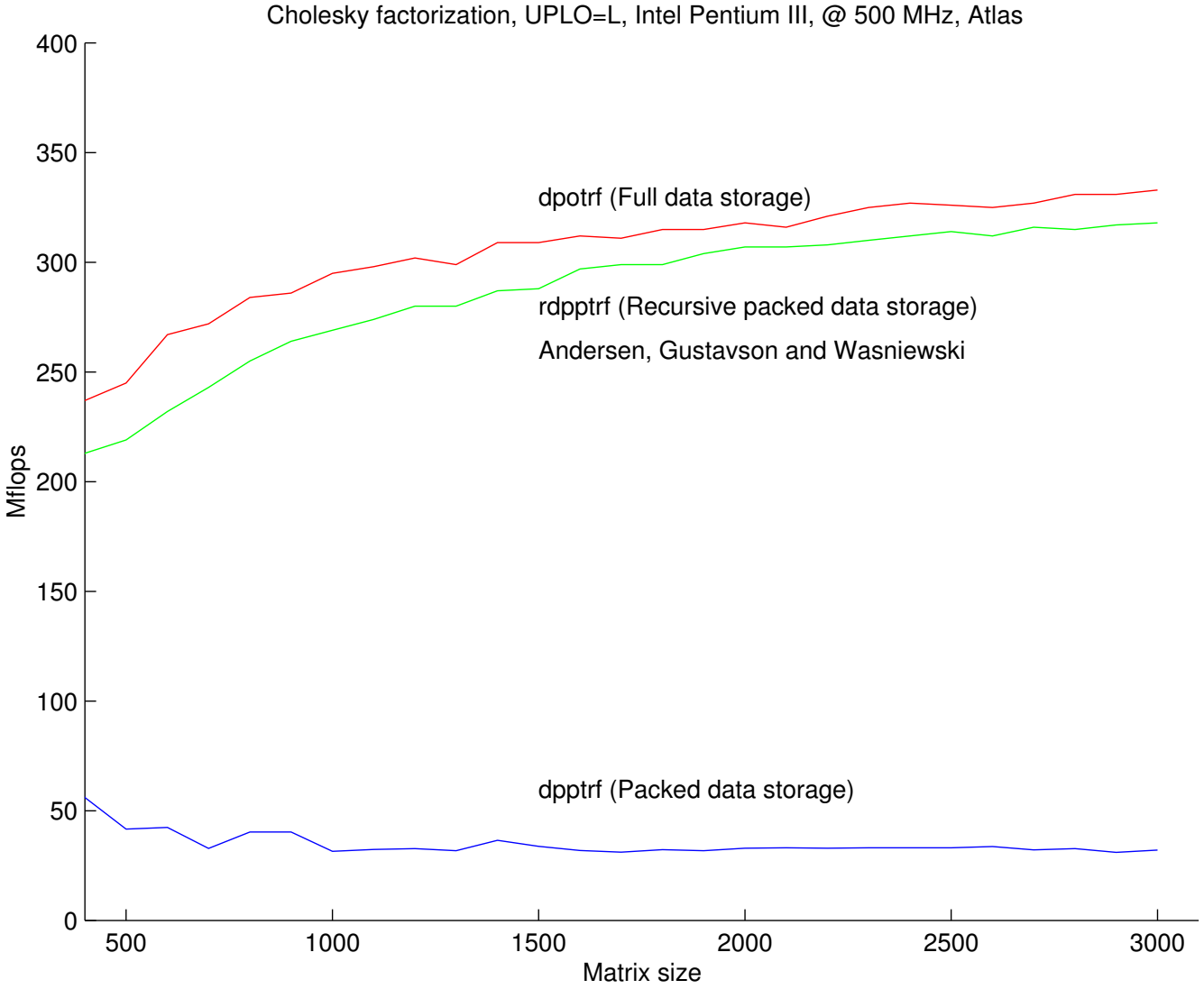
# RSYRK:

$$
\begin{aligned}
C_{11} &= \beta C_{11} + \alpha A_{11} A_{11}^T && \textbf{RSYRK} \\
C_{21} &= \beta C_{21} + \alpha A_{21} A_{11}^T && \textbf{GEMM} \\
C_{22} &= \beta C_{22} + \alpha A_{21} A_{21}^T && \textbf{RSYRK}
\end{aligned}
$$

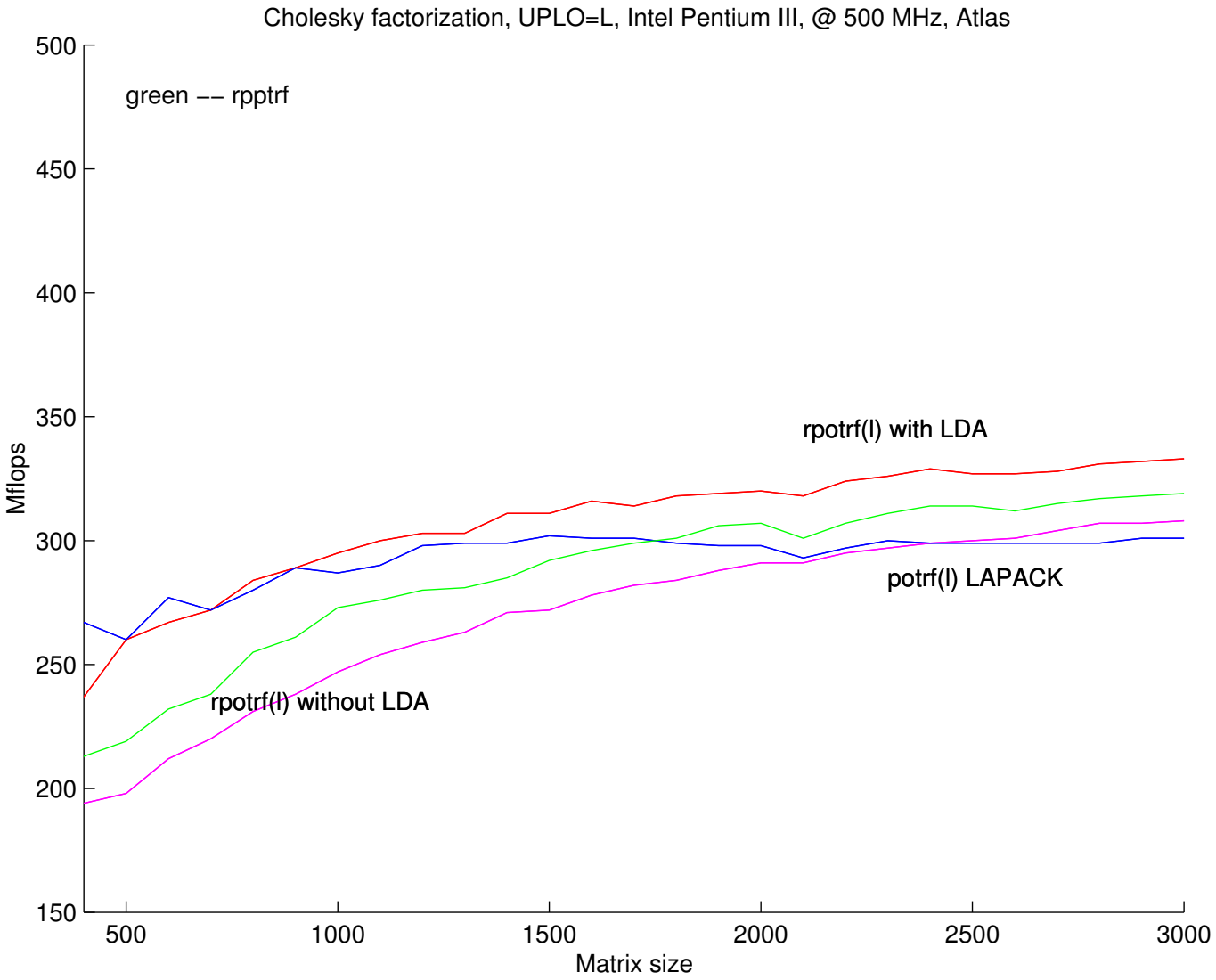
# Cholesky, Recursive Packed Storage



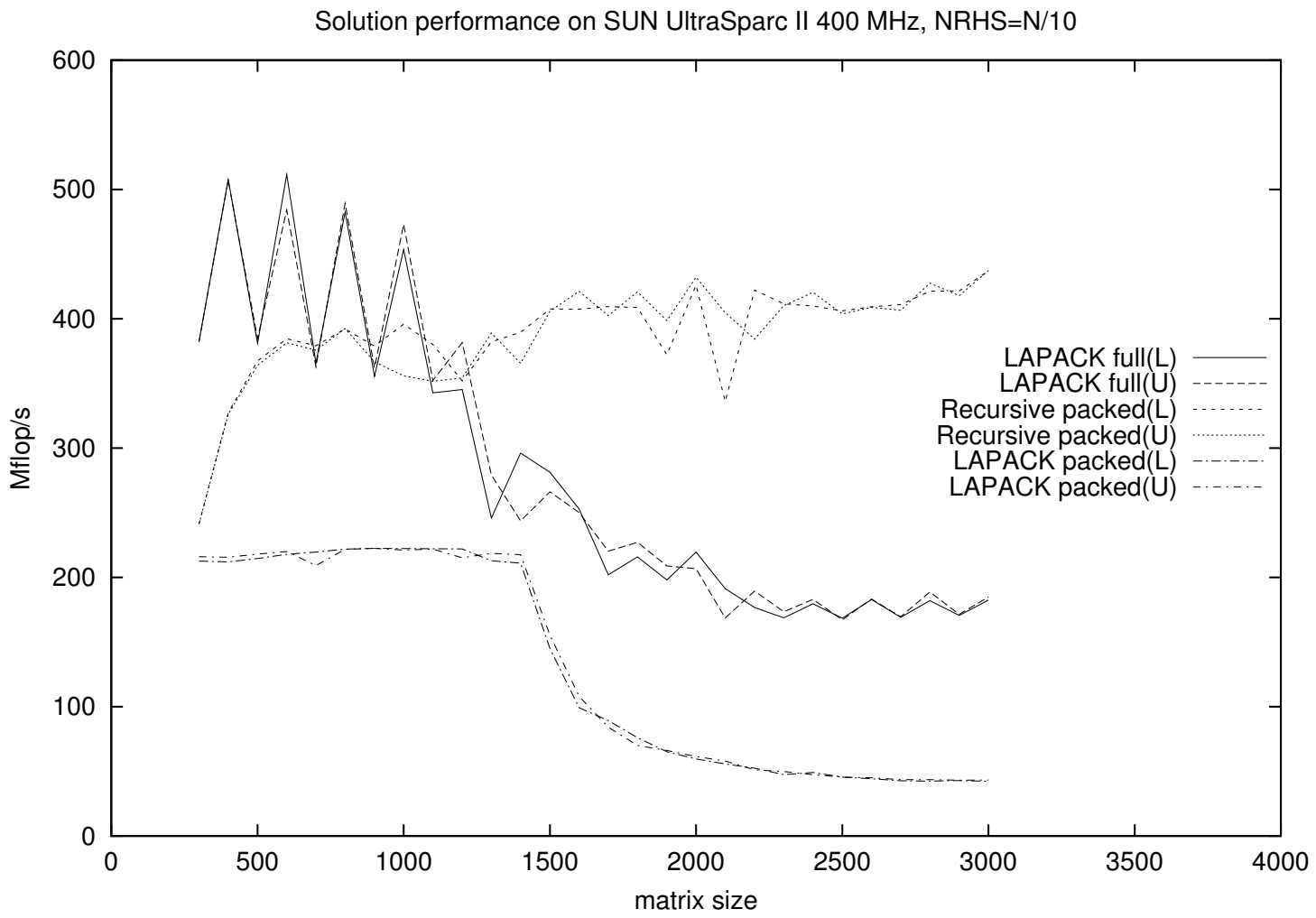
# Cholesky: packed vs. full data storage



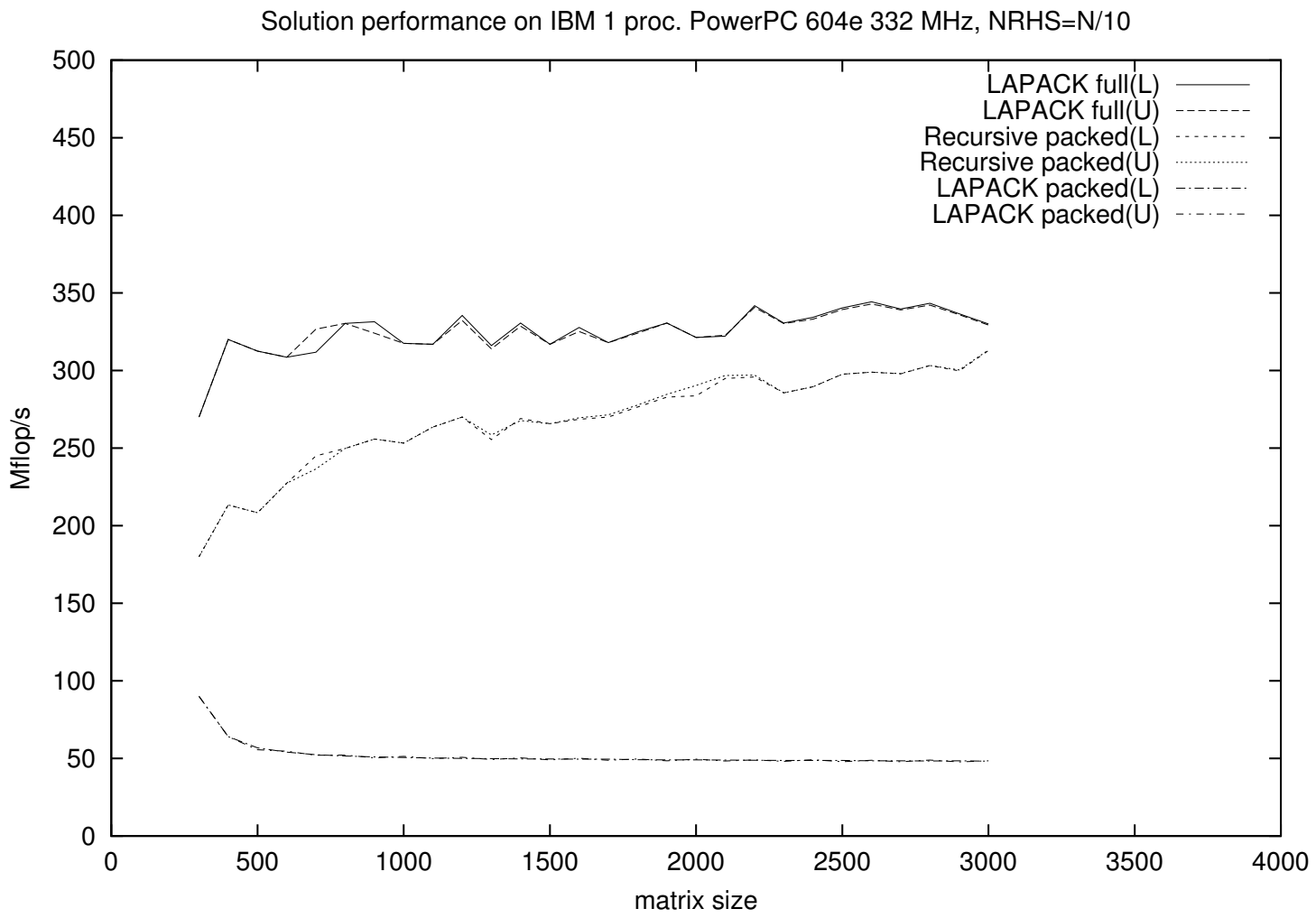
# Cholesky


Cholesky factorization, UPLO=L, Intel Pentium III, @ 500 MHz, Atlas
green –– rpptrf
rpotrf(l) with LDA
potrf(l) LAPACK
rpotrf(l) without LDA
Mflops
Matrix size
500
450
400
350
300
250
200
150
500
1000
1500
2000
2500
3000

# Cholesky, Recursive Packed Storage



# Cholesky, Recursive Packed Storage



## References


- **F.G. Gustavson. "Recursion Leads to Automatic Variable Blocking for Dense Linear-Algebra Algorithms". IBM Journal of Research and Development, 41(6), November 1997.**
- **Bjarne S. Andersen, Fred G. Gustavson, and Jerzy Waśniewski. "A recursive formulation of Cholesky factorization of a matrix in packed storage". ACM Transactions on Mathematical Software, 27(2):214–244, June 2001**

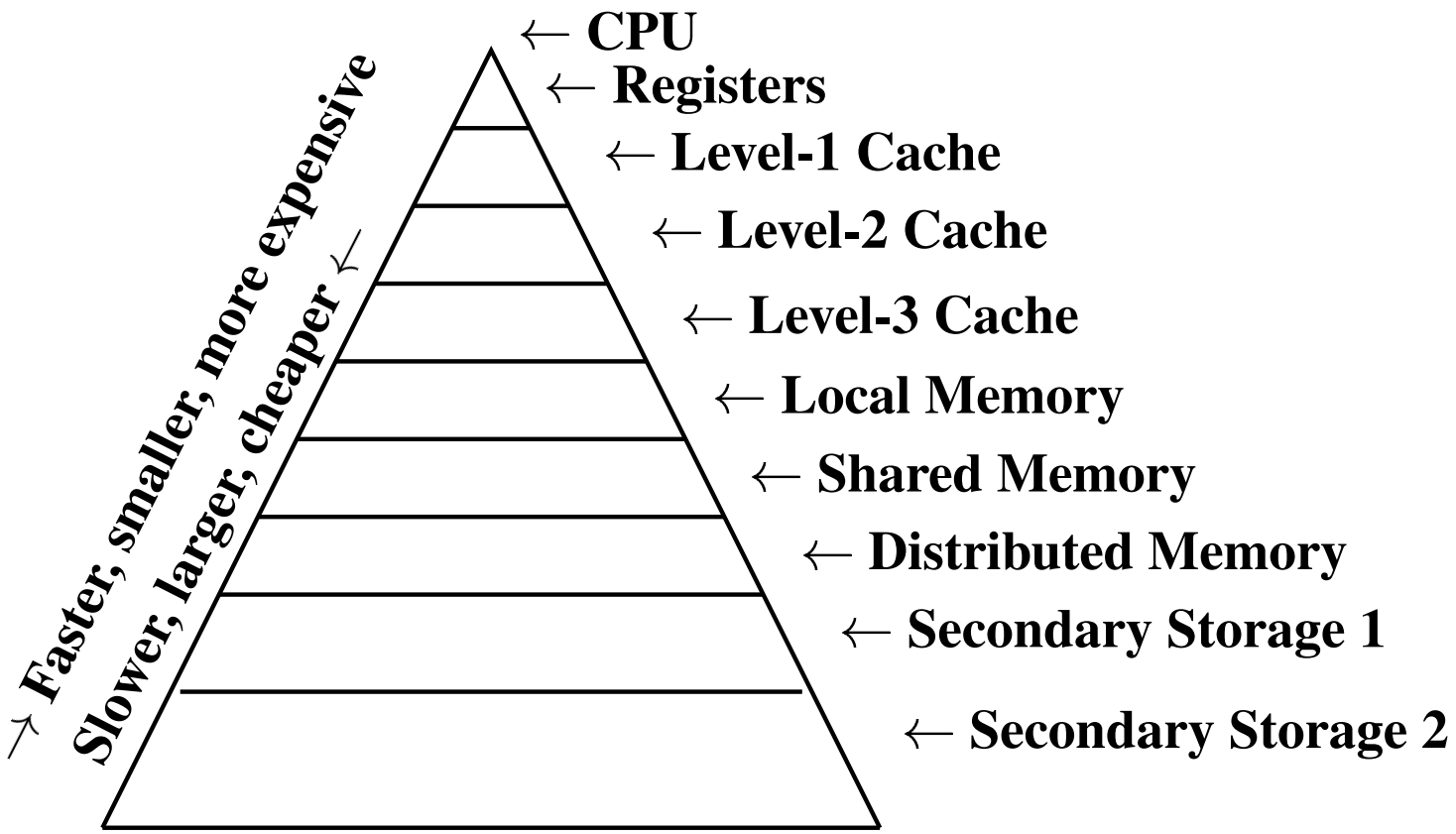
# Symmetric/Hermitian Positive Definite Matrices

## Block Packed Hybrid Storage Data Format

**Joint work with**

**Fred Gustavson, John Reid,**
**Bjarne Andersen and John Gunnels**

# A computer memory hierarchy



## Cholesky, Block Packed Hybrid Storage

**Block size nb chosen to fit comfortably in level-1 cache.**

**Note that _GEMM:**

$$C = aAB + bC$$

**is efficient if $A$ can be retained in cache while columns of $B$ and $C$ are 'streamed' in and columns of the modified $C$ are streamed out. Important that $B$ and $C$ be held by columns.**

**The consequence for the lower blocked format is that each block should be held by rows.**

**For efficient rearrangement want no element to be moved out of its block column.**

# Cholesky, Packed Storage

$$\mathbf{n} = 7, \quad \textbf{memory needed} = \mathbf{n} \times (\mathbf{n}+1)/2 = 28$$

$$\begin{pmatrix}
a_{1,1_1} & & & & & & \\
a_{2,1_2} & a_{2,2_8} & & & & & \\
a_{3,1_3} & a_{3,2_9} & a_{3,3_{14}} & & & & \\
a_{4,1_4} & a_{4,2_{10}} & a_{4,3_{15}} & a_{4,4_{19}} & & & \\
a_{5,1_5} & a_{5,2_{11}} & a_{5,3_{16}} & a_{5,4_{20}} & a_{5,5_{23}} & & \\
a_{6,1_6} & a_{6,2_{12}} & a_{6,3_{17}} & a_{6,4_{21}} & a_{6,5_{24}} & a_{6,6_{26}} & \\
a_{7,1_7} & a_{7,2_{13}} & a_{7,3_{18}} & a_{7,4_{22}} & a_{7,5_{25}} & a_{7,6_{27}} & a_{7,7_{28}}
\end{pmatrix}$$

**The mapping to array-subscript order of a $7 \times 7$ matrix for LAPACK Cholesky Algorithm using packed storage. Lower triangular case.**

# Cholesky, Block Packed Hybrid Storage

$$\mathbf{n} = 7, \quad \textbf{memory needed} = \mathbf{n} \times (\mathbf{n}+1)/2 = 28$$

$$\left(\begin{array}{ccc|ccc|c}
a_{1,1_1} & & & & & & \\
a_{2,1_2} & a_{3,1_3} & & & & & \\
a_{4,1_4} & a_{5,1_5} & a_{6,1_6} & & & & \\
\hline
a_{7,1_7} & a_{2,2_8} & a_{3,2_9} & a_{4,4_{19}} & & & \\
a_{2,4_{10}} & a_{2,5_{11}} & a_{2,6_{12}} & a_{5,4_{20}} & a_{6,4_{21}} & & \\
a_{2,7_{13}} & a_{3,3_{14}} & a_{3,4_{15}} & a_{7,4_{22}} & a_{5,5_{23}} & a_{5,6_{24}} & \\
\hline
a_{3,5_{16}} & a_{3,6_{17}} & a_{3,7_{18}} & a_{7,5_{25}} & a_{6,6_{26}} & a_{7,6_{27}} & a_{7,7_{28}}
\end{array}\right)$$

**The mapping to array-subscript order of a $7 \times 7$ matrix for LAPACK Cholesky Algorithm using block packed hybrid storage. Lower triangular case. Each block held by rows. Sort can be done using a buffer of size $n \times nb$. Here nb = 3.**

## Cholesky code, lower block hybrid format

**do j = 1,** $\lceil n/nb \rceil$

  **do k = 1, j-1**

    $A_{jj} = A_{jj} - L_{jk}L_{jk}^T$ **! Call of level-3 BLAS _SYRK**

    **do i = j+1,** $\lceil n/nb \rceil$

      $A_{ij} = A_{ij} - L_{ik}L_{jk}^T$ **! Call of level-3 BLAS _GEMM**

    **end do**

  **end do**

  $L_{jj}L_{jj}^T = A_{jj}$ **! Call of LAPACK _POTRF**

  **do i = j+1,** $\lceil n/nb \rceil$

    $L_{ij}L_{jj}^T = A_{ij}$ **! Call of level-3 BLAS _TRSM**

  **end do**

**end do**

## Cholesky code, lower block hybrid format

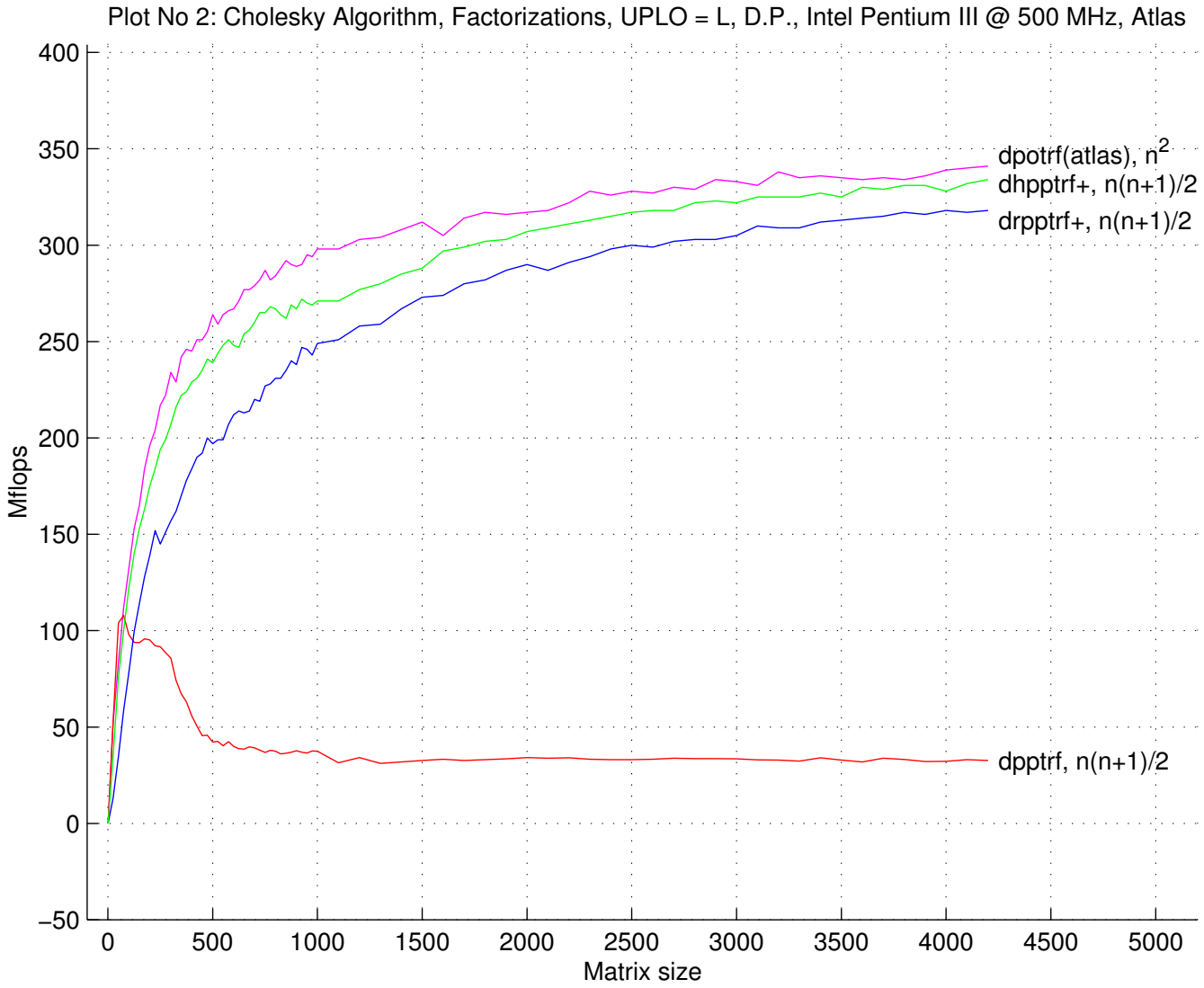
**Can implement the inner loops (index i) by a single call of _GEMM and _TRSM.**

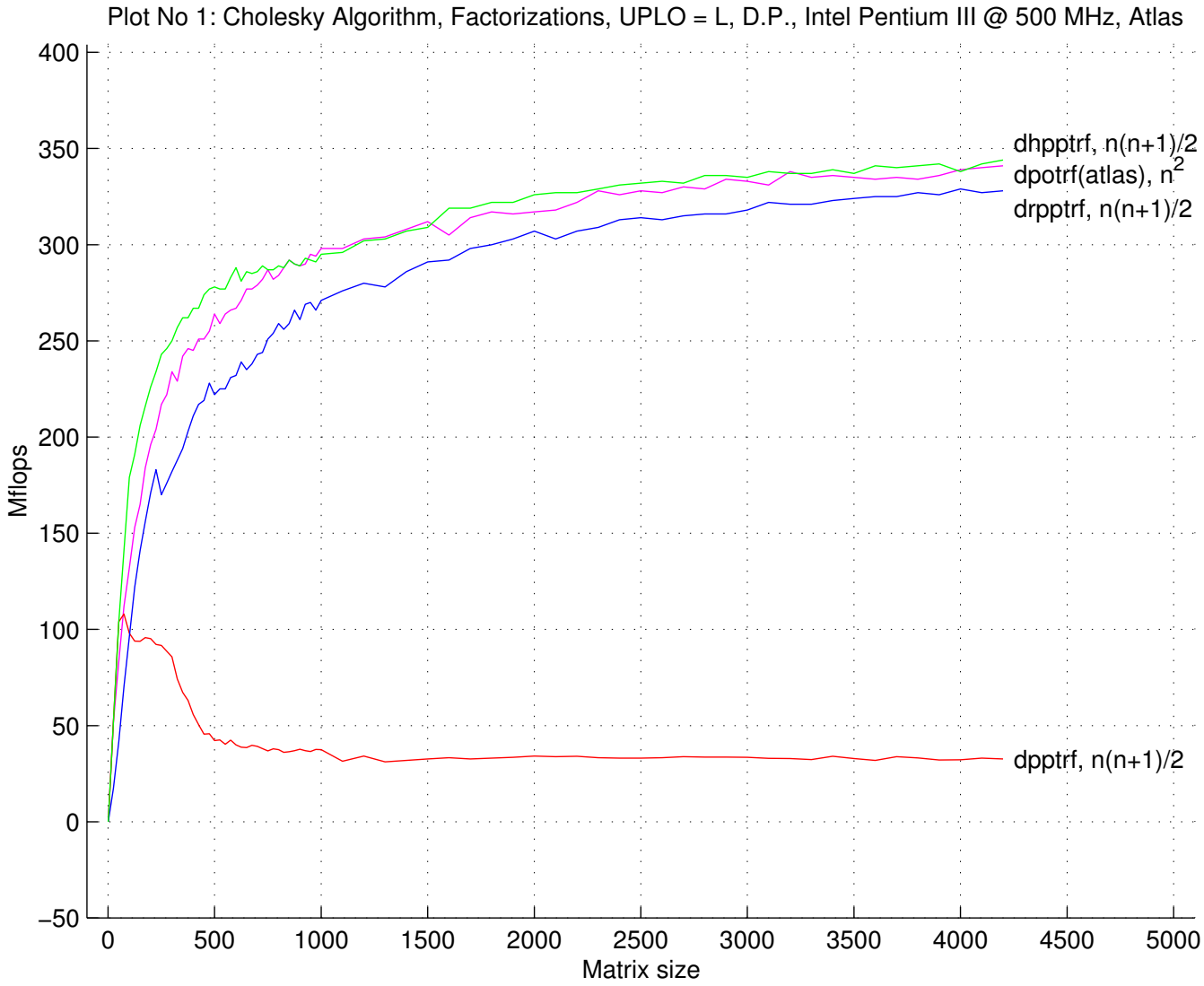
**$A_{jj}$ then $L_{jj}$ held in buffer in full format and used for _POTRF and _TRSM.**

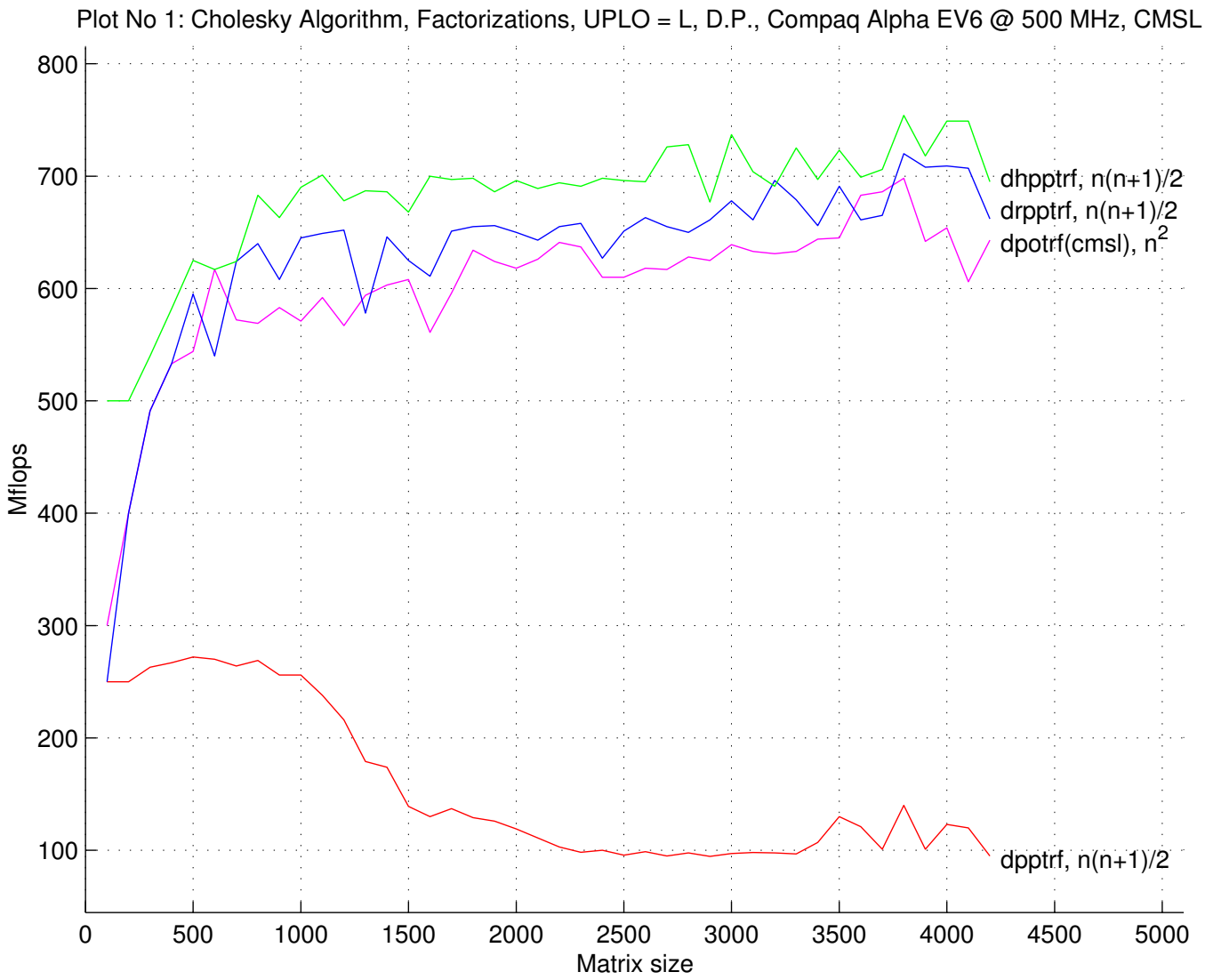
**For large problems, most of the work is in _GEMM. This is performed in contigous memory and is actually called for the transpose:**

$$A_{ij}^T = A_{ij}^T - L_{jk} L_{ik}^T$$

**Note that $L_{jk}$ is held by rows and $A_{ij}^T$ and $L_{ik}^T$ are held by columns. The operation can therefore be applied efficiently with streaming.**


Plot No 2: Cholesky Algorithm, Factorizations, UPLO = L, D.P., Intel Pentium III @ 500 MHz, Atlas
Mflops
Matrix size
400
350
300
250
200
150
100
50
0
−50
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
dpotrf(atlas), n2
dhpptrf+, n(n+1)/2
drpptrf+, n(n+1)/2
dpptrf, n(n+1)/2


Plot No 1: Cholesky Algorithm, Factorizations, UPLO = L, D.P., Intel Pentium III @ 500 MHz, Atlas
Mflops
Matrix size
400
350
300
250
200
150
100
50
0
−50
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
dhpptrf, n(n+1)/2
dpotrf(atlas), n2
drpptrf, n(n+1)/2
dpptrf, n(n+1)/2


Plot No 1: Cholesky Algorithm, Factorizations, UPLO = L, D.P., Compaq Alpha EV6 @ 500 MHz, CMSL
Mflops
Matrix size
800
700
600
500
400
300
200
100
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
dhpptrf, n(n+1)/2
drpptrf, n(n+1)/2
dpotrf(cmsl), n2
dpptrf, n(n+1)/2

## Kernel subroutines

**For factorizing the diagonal blocks, LAPACK's _POTRF is unsatisfactory since it uses the level-2 BLAS _POTF2.**

**We can expect the matrix to be held in level-1 cache, so it is register usage that is important.**

**Fred proposed writing a block code in Fortran with very small block size.**

**All the BLAS are replaced by in-line code with inner loops unrolled.**

**We have found block size $kp = 2$ to be adequate.**

## Kernel code

**The code takes the form:**

**do i = 1,** $\lceil n/kb \rceil$

$A_{ii} = A_{ii} - \sum_{k=1}^{i-1} (U_{ki}^T U_{ki})$ **! Like level-3 BLAS _SYRK**

$U_i i^T U_{ii} = A_{ii}$ **! Cholesky factorization of block**

**do j = i+1, nb**

$A_{ij} = A_{ij} - \sum_{k=1}^{i-1} (U_{ki}^T U_{kj})$ **! Like level-3 BLAS _GEMM**

$U_{ii}^T U_{ij} = A_{ij}$ **! Like level-3 BLAS _TRSM**

**end do**

**end do**

**The key loop is the one that corresponds to _GEMM. For this, the following code is suitable**

```
do k = 1, ii - 1
   aki = a(k,ii)
   akj = a(k,jj)
   t11 = t11 - aki*akj
   aki1 = a(k,ii+1)
   t21 = t21 - aki1*akj
   akj1 = a(k,jj+1)
   t12 = t12 - aki*akj1
   t22 = t22 - aki1*akj1
end do
```

**8 local variables, hopefully in registers, 4 memory accesses, 8 flops. Also tried inner loop that updates $A_{ij}$ and $A_{i,j+1}$. $aki$ and $aki1$ need to be loaded only once, so get 14 local variables, 6 memory accesses, 8 flops.**

**Performance in Mflops of the Kernel Cholesky Algorithm. Comparison between different computers and different versions of subroutines.**

| Order | LAPACK | | Recur- | Mini-block | |
|---|---|---|---|---|---|
| | Vendor | Comp. | sive | 2×2 | 2×2 & 2×4 |
| **IBM Power4, 1700 MHz, ESSL Library:** | | | | | |
| 40 | 1658 | 1503 | 707 | 1999 | 1999 |
| 72 | 2653 | 2303 | 1447 | 2751 | 2753 |
| 100 | 3037 | 2481 | 1930 | 2957 | 2945 |
| **SUN Ultra III, 900 MHz, Sunperf BLAS Library:** | | | | | |
| 40 | 392 | 427 | 251 | 803 | 941 |
| 72 | 598 | 664 | 417 | 1191 | 1012 |
| 100 | 619 | 830 | 589 | 1143 | 1506 |
| **HP Itanium 2, 1000 MHz, HP MLIB BLAS Library:** | | | | | |
| 40 | 449 | 448 | 153 | 1125 | 1133 |
| 72 | 597 | 595 | 266 | 1711 | 1722 |
| 100 | 567 | 559 | 364 | 2103 | 2103 |

## Cholesky solution, single right-hand side

**do j = 1,** $\lceil n/nb \rceil$

$L_{jj}Y_j = B_j$ **! Call of level-2 BLAS _TPSV**

$B_i = B_i - L_{ij}Y_j, \forall i > j$ **! Single call of level-2 BLAS _GEMV**

**end do**

**do j =** $\lceil n/nb \rceil$**, 1, -1**

$Y_j = Y_j - \sum_{i>j}(L_{ij}^T X_i)$ **! Single call of level-2 B LAS _GEMV**

$L_{jj}X_j = Y_j$ **! Call of level-2 BLAS _TPSV**

**end do**

**Similar code could be applied for many rhs, with BLAS3 codes replacing BLAS2. However, the blocks** $B_i$ **,** $Y_i$ **, and** $X_i$ **would not occupy contiguous memory.**

## Cholesky solution, many right-hand sides

**If the number of columns $m$ is modest, we therefore make a copy of $B$ as a block matrix, with each block $B_i$ held contiguously by columns:**

$$\begin{array}{cccc} 0 & 3 & 6 & 9 \\ 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ \hline 12 & 15 & 18 & 21 \\ 13 & 16 & 19 & 22 \\ 14 & 17 & 20 & 23 \\ \hline 24 & 25 & 26 & 27 \end{array}$$

**Have a single level-3 BLAS call for each block, but each matrix is held contiguously, and streaming is available.**

## Upper packed format

**Essentially the same advantages are obtained by doing a $UU^T$ factorization, which corresponds to a backward pivot sequence.**

**Equally good numerically, but rejected since not compatible with what LAPACK does.**

**Forced us to abandon the property of contiguous blocks forming an array. No merging of BLAS3 calls.**

# Computers Used

| Processor | MHz | Peak Mflops | Cache sizes level-1 | level-2 | level-3 | TLB entries |
|---|---|---|---|---|---|---|
| IBM Power4 | 1700 | 6800 | 64K | 1.5M* | 32M* | 1024 |
| SUN UltSparc | 900 | 1800 | 64K | 8M | None | 512 |
| HP Itanium 2 | 1000 | 4000 | 32K | 256K | 1.5M | 128 |
| SGI R12000 | 300 | 600 | 32K | 8M | None | 64 |
| HP Alpha EV6 | 500 | 1000 | 64K | 4M | None | 128 |
| INTEL Pent. 3 | 500 | 500 | 16K | 512K | None | 32 |

*Shared with another processor.

**Mflops, Cholesky factorizations including rearrangement, different $nb$ values, IBM Power4.**

| $n$ | 40 | 64 | 100 | 160 | 250 | 400 | 640 | 1000 | 1600 | 2500 | 4000 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Lower Packed Hybrid** | | | | | | | | | | | |
| $nb$=40 | 956 | 1444 | 2055 | 2679 | 3156 | 3626 | 3861 | 3933 | 3968 | 3960 | 4040 |
| $nb$=72 | 957 | 1520 | 1972 | 2650 | 3100 | 3626 | 3971 | 4086 | 4162 | 4166 | 4292 |
| $nb$=100 | 943 | 1494 | 2103 | 2741 | 3193 | 3715 | 3971 | 4119 | 4162 | 4117 | 4258 |
| $nb$=200 | 947 | 1515 | 2116 | 2417 | 2895 | 3440 | 3786 | 4059 | 4213 | 4322 | 4500 |
| **Upper Packed Hybrid** | | | | | | | | | | | |
| $nb$=40 | 1117 | 1425 | 1916 | 2523 | 2911 | 3404 | 3598 | 3655 | 3710 | 3720 | 3716 |
| $nb$=72 | 1111 | 1743 | 2052 | 2590 | 3006 | 3489 | 3761 | 3947 | 4015 | 4084 | 4102 |
| $nb$=100 | 1102 | 1732 | 2376 | 2831 | 3159 | 3678 | 3832 | 4033 | 4112 | 4150 | 4191 |
| $nb$=200 | 1105 | 1732 | 2375 | 2584 | 2997 | 3447 | 3761 | 4033 | 4266 | 4340 | 4481 |

**Mflops, Cholesky factorizations including rearrangement, different $nb$ values, SUN Ultra III.**

| $n$ | 40 | 64 | 100 | 160 | 250 | 400 | 640 | 1000 | 1600 | 2500 | 4000 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Lower Packed Hybrid** | | | | | | | | | | | |
| $nb$=40 | 394 | 548 | 644 | 773 | 832 | 963 | 1046 | 1106 | 1110 | 949 | 842 |
| $nb$=72 | 391 | 558 | 660 | 760 | 857 | 993 | 1107 | 1182 | 1215 | 1120 | 1045 |
| $nb$=100 | 389 | 557 | 708 | 738 | 824 | 959 | 1095 | 1115 | 1137 | 1144 | 1083 |
| $nb$=200 | 390 | 557 | 708 | 782 | 867 | 965 | 1080 | 1180 | 1201 | 1206 | 1254 |
| **Upper Packed Hybrid** | | | | | | | | | | | |
| $nb$=40 | 526 | 608 | 680 | 804 | 830 | 943 | 1006 | 1040 | 1052 | 915 | 791 |
| $nb$=72 | 520 | 769 | 767 | 838 | 916 | 1032 | 1104 | 1182 | 1168 | 1123 | 1004 |
| $nb$=100 | 516 | 770 | 957 | 841 | 901 | 1026 | 1116 | 1177 | 1213 | 1145 | 1065 |
| $nb$=200 | 518 | 768 | 954 | 988 | 983 | 1032 | 1144 | 1181 | 1240 | 1280 | 1243 |

**Mflops, Cholesky factorizations, lower case, $nb$ = 100, IBM Power4.**

| $n$ | 40 | 64 | 100 | 160 | 250 | 400 | 640 | 1000 | 1600 | 2500 | 4000 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| p. lapack | 747 | 951 | 1043 | 1024 | 1059 | 1101 | 1037 | 709 | 638 | 621 | 635 |
| v. p. lapack | 1750 | 2359 | 2658 | 2346 | 3107 | 3560 | 3773 | 3870 | 3969 | 3815 | 3836 |
| f. lapack | 440 | 722 | 1390 | 2119 | 2562 | 3242 | 3495 | 3797 | 3901 | 3787 | 4010 |
| v. f. lapack | 1492 | 2165 | 2486 | 3194 | 3454 | 3677 | 3832 | 3921 | 4162 | 4037 | 4327 |
| p. recursive+ | 170 | 379 | 593 | 1024 | 1586 | 2077 | 2621 | 3030 | 3434 | 3555 | 3943 |
| p. recursive | 181 | 406 | 618 | 1060 | 1652 | 2133 | 2700 | 3111 | 3523 | 3604 | 3980 |
| p. hybrid+ | 878 | 1488 | 2085 | 2721 | 3211 | 3754 | 3974 | 4112 | 4188 | 4200 | 4275 |
| p. hybrid | 1006 | 1717 | 2334 | 2977 | 3441 | 3938 | 4149 | 4279 | 4266 | 4269 | 4309 |

**Mflops, Cholesky Factorizations, lower case, $nb$ = 200, SUN UltraSPARC III.**

| $n$ | 40 | 64 | 100 | 160 | 250 | 400 | 640 | 1000 | 1600 | 2500 | 4000 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| p. lapack | 183 | 247 | 296 | 312 | 321 | 325 | 328 | 331 | 315 | 200 | 169 |
| f. lapack | 299 | 436 | 637 | 842 | 933 | 1086 | 864 | 1173 | 1203 | 1169 | 1236 |
| p. recursive+ | 95 | 202 | 275 | 426 | 550 | 727 | 913 | 1010 | 1093 | 1118 | 1215 |
| p. recursive | 102 | 219 | 290 | 454 | 581 | 760 | 945 | 1043 | 1146 | 1162 | 1249 |
| p. hybrid+ | 390 | 557 | 708 | 782 | 867 | 965 | 1080 | 1180 | 1201 | 1206 | 1254 |
| p. hybrid | 529 | 778 | 959 | 973 | 1003 | 1077 | 1164 | 1267 | 1277 | 1257 | 1304 |

**Mflops, Cholesky Factorizations, lower case, $nb$ = 200, HP Itanium 2.**

| $n$ | 40 | 64 | 100 | 160 | 250 | 400 | 640 | 1000 | 1600 | 2500 | 4000 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| p. lapack | 218 | 328 | 463 | 644 | 825 | 952 | 1039 | 660 | 609 | 595 | 590 |
| f. lapack | 376 | 581 | 838 | 1307 | 1728 | 2189 | 2546 | 2777 | 2904 | 3028 | 3141 |
| p. recursive+ | 110 | 232 | 378 | 698 | 1079 | 1608 | 2170 | 2531 | 2758 | 2861 | 3013 |
| p. recursive | 121 | 256 | 411 | 742 | 1147 | 1675 | 2305 | 2666 | 2874 | 2942 | 3056 |
| p. hybrid+ | 657 | 1017 | 1689 | 1731 | 1279 | 1384 | 1552 | 1829 | 2089 | 2216 | 2402 |
| p. hybrid | 763 | 1139 | 1849 | 1815 | 1390 | 1504 | 1661 | 1910 | 2167 | 2274 | 2438 |

**Mflops, Solution, many right-hand sides, Notes: Results for Vendor Packed Lapack L and Vendor Full Lapack very similar to corresponding Lapack results. $nb$ = 100, $mb$ = 100, IBM Power4.**

| $n$ | 40 | 64 | 100 | 160 | 250 | 400 | 640 | 1000 | 1600 | 2500 | 4000 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| p. lapack L | 919 | 1298 | 1577 | 1822 | 2027 | 2182 | 2088 | 1733 | 1526 | 1460 | 1456 |
| p. lapack U | 912 | 1296 | 1572 | 1836 | 2044 | 2197 | 2088 | 1710 | 1517 | 1467 | 1469 |
| f. lapack L | 3195 | 3368 | 3750 | 4021 | 4210 | 4266 | 4259 | 4257 | 4441 | 4320 | 4368 |
| f. lapack U | 3205 | 3392 | 3740 | 4045 | 4175 | 4238 | 4259 | 4222 | 4495 | 4320 | 4353 |
| p. recursive L | 1240 | 1510 | 1956 | 2658 | 3075 | 3306 | 3524 | 3734 | 4147 | 4166 | 4571 |
| p. recursive U | 1225 | 1493 | 1935 | 2655 | 3091 | 3306 | 3510 | 3750 | 4147 | 4166 | 4555 |
| P. Hybrid L | 2359 | 2941 | 3411 | 3778 | 3991 | 4084 | 4231 | 4285 | 4362 | 4422 | 4620 |
| P. Hybrid U | 2374 | 2927 | 3389 | 3736 | 3965 | 4132 | 4231 | 4293 | 4415 | 4380 | 4555 |

**Mflops, Solution, many right-hand sides, $nb$ = 200, $mb$ = 100, SUN UltraSPARC III.**

| $n$ | 40 | 64 | 100 | 160 | 250 | 400 | 640 | 1000 | 1600 | 2500 | 4000 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| p. lapack L | 289 | 341 | 373 | 265 | 220 | 209 | 203 | 199 | 119 | 63 | 54 |
| p. lapack U | 276 | 326 | 357 | 294 | 271 | 277 | 281 | 270 | 179 | 99 | 87 |
| f. lapack L | 901 | 933 | 1045 | 1089 | 1103 | 1279 | 1238 | 1300 | 1268 | 1312 | 1299 |
| f. lapack U | 894 | 974 | 1042 | 1095 | 1103 | 1276 | 1244 | 1312 | 1261 | 1390 | 1290 |
| p. recursive L | 280 | 467 | 489 | 678 | 849 | 981 | 1081 | 1241 | 1296 | 1362 | 1436 |
| p. recursive U | 267 | 466 | 477 | 659 | 843 | 968 | 1071 | 1239 | 1291 | 1360 | 1434 |
| p. hybrid L | 766 | 852 | 959 | 1023 | 1083 | 1236 | 1317 | 1334 | 1373 | 1420 | 1436 |
| p. hybrid U | 767 | 858 | 959 | 1031 | 1074 | 1234 | 1316 | 1336 | 1401 | 1453 | 1435 |

**Mflops, Solution, many right-hand sides, $nb$ = 200, $mb$ = 100, HP Itanium 2.**

| $n$ | 40 | 64 | 100 | 160 | 250 | 400 | 640 | 1000 | 1600 | 2500 | 4000 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| p. lapack L | 223 | 305 | 385 | 471 | 538 | 576 | 608 | 628 | 640 | 649 | 655 |
| p. lapack U | 225 | 308 | 385 | 465 | 532 | 578 | 614 | 632 | 642 | 648 | 655 |
| f. lapack L | 272 | 411 | 593 | 844 | 1150 | 1525 | 1878 | 2222 | 2452 | 2637 | 2813 |
| f. lapack U | 271 | 406 | 593 | 835 | 1157 | 1538 | 1881 | 2222 | 2452 | 2626 | 2813 |
| p. recursive L | 712 | 958 | 1325 | 1641 | 2071 | 2440 | 2678 | 2765 | 3015 | 3094 | 3062 |
| p. recursive U | 695 | 958 | 1289 | 1617 | 2077 | 2392 | 2636 | 2784 | 3015 | 3094 | 3062 |
| p. hybrid L | 251 | 396 | 581 | 817 | 1103 | 1473 | 1853 | 2196 | 2560 | 2765 | 2976 |
| p. hybrid U | 251 | 396 | 577 | 808 | 1103 | 1463 | 1853 | 2193 | 2544 | 2777 | 2969 |

**Mflops, Solution, one right-hand side, $nb$ = 200, SUN UltraSPARC III.**

| $n$ | 40 | 64 | 100 | 160 | 250 | 400 | 640 | 1000 | 1600 | 2500 | 4000 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| p. lapack L | 285 | 339 | 383 | 268 | 221 | 208 | 204 | 199 | 84 | 56 | 51 |
| p. lapack U | 270 | 325 | 371 | 290 | 274 | 280 | 281 | 280 | 154 | 96 | 85 |
| f. lapack L | 363 | 479 | 575 | 657 | 665 | 798 | 798 | 495 | 370 | 285 | 286 |
| f. lapack U | 373 | 472 | 601 | 698 | 667 | 786 | 791 | 540 | 340 | 306 | 288 |
| p. recursive L | 71 | 113 | 155 | 229 | 299 | 404 | 504 | 560 | 318 | 258 | 252 |
| p. recursive U | 72 | 114 | 155 | 229 | 306 | 405 | 505 | 566 | 314 | 263 | 277 |
| p. hybrid L | 247 | 310 | 354 | 294 | 338 | 411 | 519 | 554 | 348 | 265 | 229 |
| p. hybrid U | 240 | 309 | 352 | 292 | 337 | 411 | 520 | 584 | 325 | 231 | 224 |

**Mflops, Solution, one right-hand side, $nb$ = 200, HP Itanium 2.**

| $n$ | 40 | 64 | 100 | 160 | 250 | 400 | 640 | 1000 | 1600 | 2500 | 4000 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| p. lapack L | 221 | 301 | 382 | 471 | 518 | 587 | 655 | 622 | 640 | 649 | 656 |
| p. lapack U | 225 | 301 | 385 | 462 | 542 | 584 | 738 | 663 | 657 | 653 | 652 |
| f. lapack L | 181 | 263 | 341 | 452 | 522 | 625 | 317 | 307 | 314 | 285 | 258 |
| f. lapack U | 183 | 261 | 344 | 438 | 554 | 603 | 344 | 312 | 317 | 287 | 255 |
| p. recursive L | 76 | 118 | 173 | 268 | 370 | 529 | 615 | 662 | 779 | 886 | 999 |
| p. recursive U | 72 | 118 | 169 | 258 | 382 | 527 | 591 | 658 | 771 | 888 | 1001 |
| p. hybrid L | 194 | 279 | 371 | 449 | 618 | 847 | 743 | 649 | 624 | 620 | 789 |
| p. hybrid U | 193 | 280 | 370 | 450 | 620 | 855 | 776 | 670 | 640 | 633 | 789 |

**Mflops, Solution, one right-hand side, $nb$ = 100, IBM Power4.**

| $n$ | 40 | 64 | 100 | 160 | 250 | 400 | 640 | 1000 | 1600 | 2500 | 4000 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| p. lapack L | 746 | 1145 | 1422 | 1732 | 1993 | 2117 | 1897 | 1619 | 1475 | 1432 | 1443 |
| p. lapack U | 746 | 1145 | 1446 | 1775 | 2009 | 2190 | 1793 | 1483 | 1415 | 1402 | 1417 |
| V. p. lapack L | 746 | 1185 | 1472 | 1746 | 1993 | 2182 | 2039 | 1695 | 1505 | 1448 | 1456 |
| f. lapack L | 706 | 1074 | 1422 | 1760 | 1907 | 1639 | 1437 | 1311 | 1399 | 1344 | 1141 |
| f. lapack U | 706 | 1108 | 1422 | 1775 | 2009 | 1890 | 1622 | 1251 | 1362 | 1330 | 1385 |
| V. f. lapack L | 746 | 1108 | 1422 | 1790 | 1993 | 1980 | 1675 | 1212 | 1400 | 1343 | 1292 |
| V. f. lapack U | 706 | 1074 | 1422 | 1790 | 1920 | 1586 | 1411 | 1267 | 1358 | 1314 | 1065 |
| p. recursive L | 268 | 404 | 552 | 804 | 880 | 1073 | 1054 | 1002 | 1150 | 1216 | 967 |
| p. recursive U | 268 | 414 | 559 | 817 | 895 | 1169 | 1188 | 997 | 1175 | 1247 | 1055 |
| p. hybrid L | 746 | 1074 | 1446 | 1579 | 1759 | 1451 | 1218 | 1085 | 1036 | 1012 | 743 |
| p. hybrid U | 746 | 1108 | 1446 | 1591 | 1771 | 1404 | 1202 | 1065 | 1008 | 985 | 720 |

## Conclusions

**Mini-blocked kernel remarkable successful.**

**It is possible to get good performance with packed formats, usually comparable with full Lapack, sometimes better.**

**Packed hybrid is better than the packed recursive for modest n and for solving one rhs.**

**Packed hybrid may be slower than the packed recursive for large n.**

## Reference


- **B.S. Andersen, J.A. Gunnels, F. Gustavson, J.K. Reid, and J. Waśniewski. "A Fully Portable High Performance Minimal Storage Hybrid Format Cholesky Algorithm". ACM Trans. of Math. Software, 31 (2005), 201-227.**

# Symmetric/Hermitian Indefinite Matrices

## Symmetric Indefinite Matrix

$A$ $\qquad\qquad\qquad\qquad\qquad\qquad$ $A$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

**A diagonal pivoting method is used to factor $A$ as**

$$A = P^T U D U^T P, \quad \text{if} \quad \mathbf{UPLO} = \text{'U'} \quad \textbf{or}$$

$$A = P^T L D L^T P, \quad \text{if} \quad \mathbf{UPLO} = \text{'L'}$$

- **$P$ is a permutation matrix**
- **$U$ and $L$ are unit upper and lower triangular matrices, respectively**
- **$D$ is a symmetric block diagonal with $1 \times 1$ and $2 \times 2$ diagonal blocks.**

# Bunch-Kaufman Pivoting Strategy

$\alpha = (1+\sqrt{17})/8$; $\lambda = |a_{r1}| = \max\{|a_{21}|, \ldots, |a_{n1}|\}$

**if** $\lambda > 0$

  **if** $|a_{11}| \geq \alpha\lambda$ **then** $s = 1$; $P_1 = I$

  **else**

    $\sigma = |a_{pr}| = \max\{|a_{1r}, \ldots |a_{r-1,r}, |a_{r+1,r}|, \ldots, |a_{nr}|\}$

    **if** $\sigma|a_{11}| \geq \alpha\lambda^2$ **then** $s = 1$, $P_1 = I$

    **else if** $|a_{rr}| \geq \alpha\sigma$

      $s = 1$ **and choose** $P_1$ **so** $(P_1^T A P_1)_{11} = a_{rr}$

    **else**

      $s = 2$ **and choose** $P_1$ **so** $(P_1^T A P_1)_{21} = a_{r1}$

    **end**

  **end**

**end**

# Perturbation Approach

## Perturbation Approach

- **Perturbation approach with iterative refinement.**
- **Perturbation approach with the Sherman-Morrison-Woodbury formula.**

$$(A + UV^T)^{-1} = A^{-1} - A^{-1}U(I + V^T A^{-1}U)^{-1}V^T A^{-1},$$

**where $A$ is $n$-by-$n$, $U$ and $V$ are $n$-by-$k$ matrices ($k$ must be small).**

- **Mixed approach.**

## Perturbation Approach

### Recursive Perturbation-Based Algorithm

**We add a small number $\delta > 0$ to each divisor $|a| < \delta$:**

$$a = a + Sgn(a)\delta$$

**where**

$$Sgn(a) = \begin{cases} sign(a) & \textbf{if } a \neq 0 \\ 1 & \textbf{if } a = 0 \end{cases}$$

# Symmetric Indefinite Matrix

## Numerical Experiments

**We ran two kind of experiments:**

- **Experiments (denoted by Series 1), the random matrices are generated by the LAPACK subroutine DLAGGE.**
- **Experiments (denoted by Series 2), the matrices $A$ are of the following type[a]:**

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

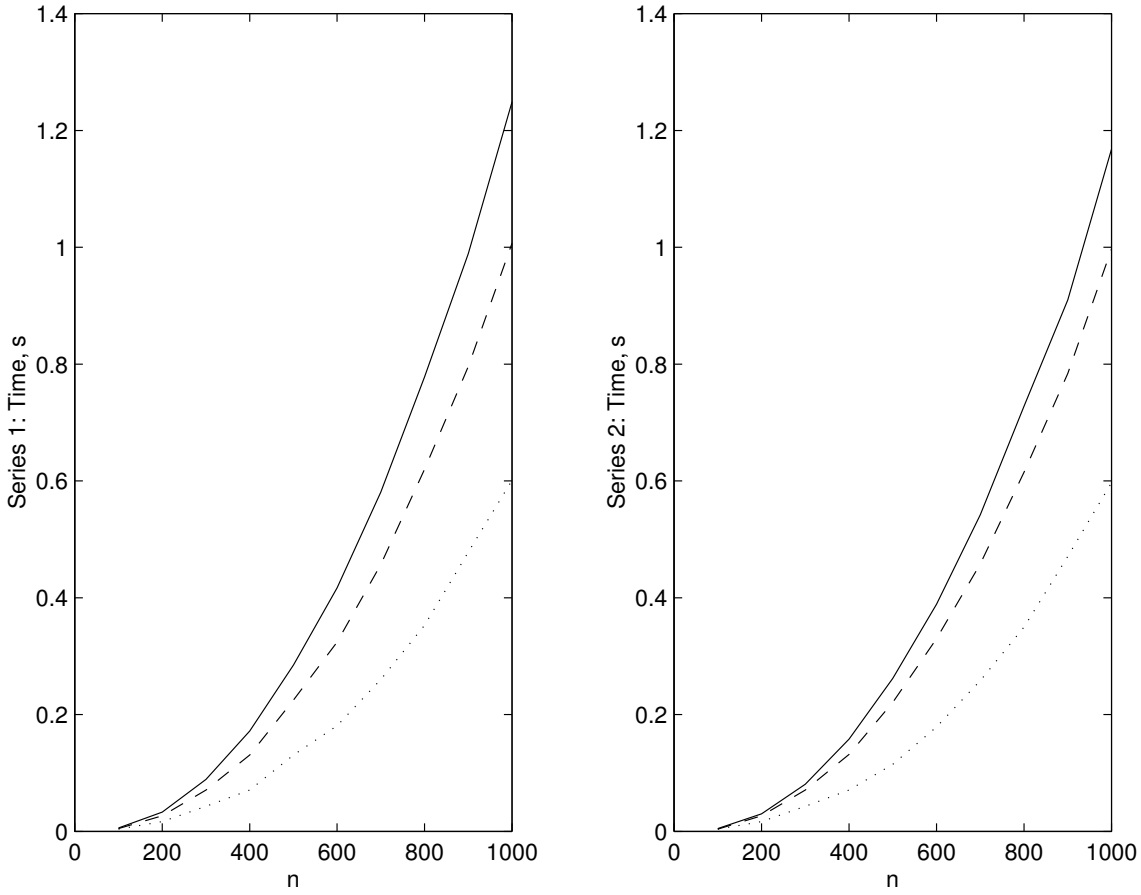
**where $\Delta$ is a diagonal matrix with small entries, $C$ is a random matrix with large entries, and $I$ is the identity. The entries of $\Delta$ are chosen to be less than $\delta$.**

[a] Suggested to us by John Reid.

# Symmetric Indefinite Matrix

## Perturbation Approach

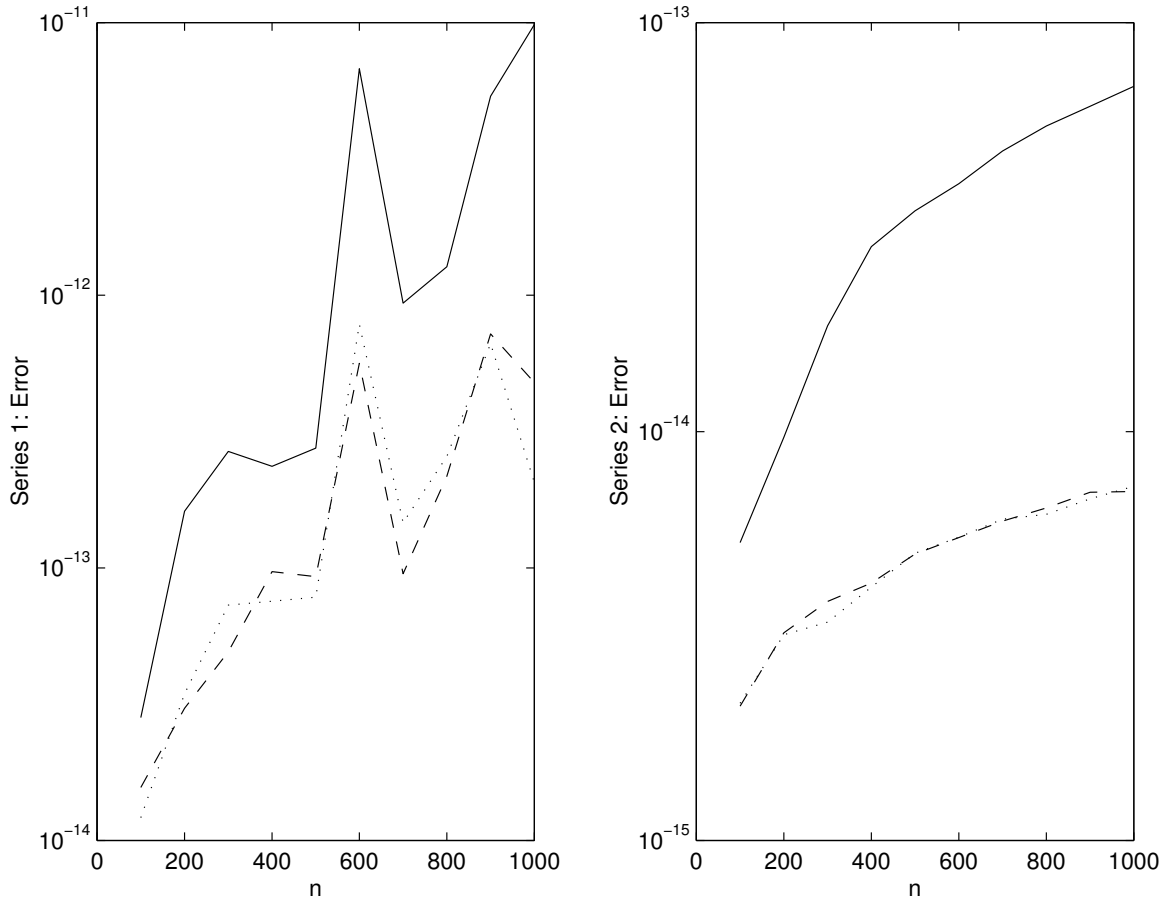
**CPU times on the IBM SMP (4xPPC604e/332Mhz) of DSYSV (—), P_DSYSV (- - -) and DPRSIV (. . .)**



# Symmetric Indefinite Matrix

## Perturbation Approach

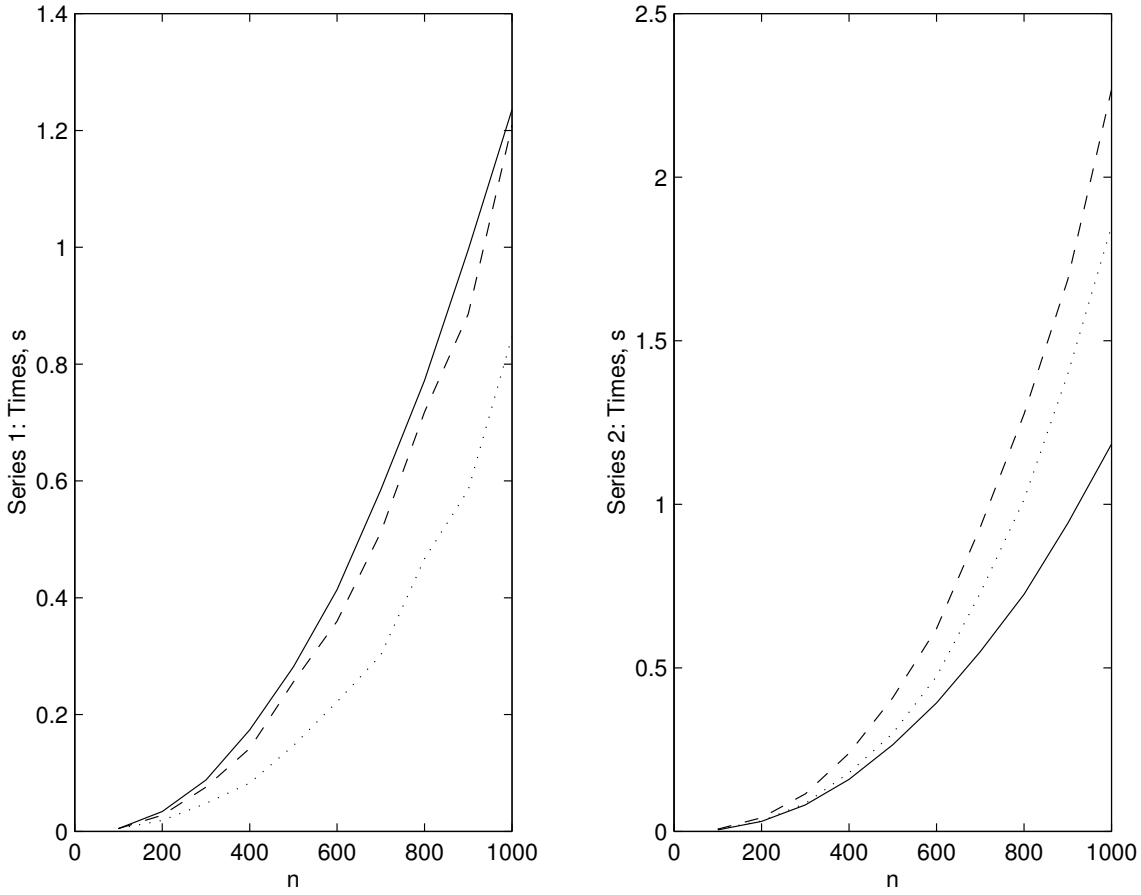
**Errors on the IBM SMP (4xPPC604e/332Mhz) of DSYSV (—), P_DSYSV (- - -) and DPRSIV (. . .)**



# Symmetric Indefinite Matrix

## Perturbation SMW Approach

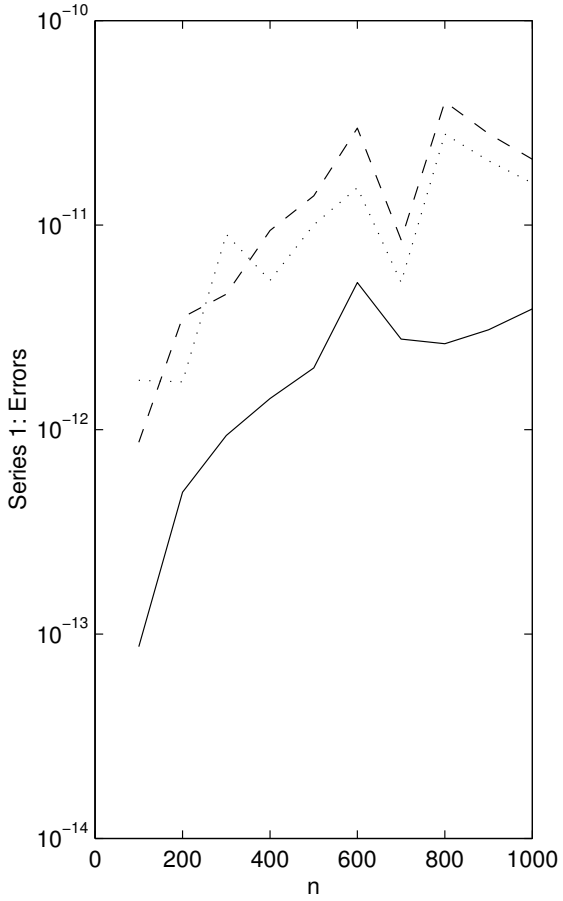
**CPU times on the IBM SMP (4xPPC604e/332Mhz) of DSYSV (—), P_DSYSV (- - -) and DPRSIV (. . .)**

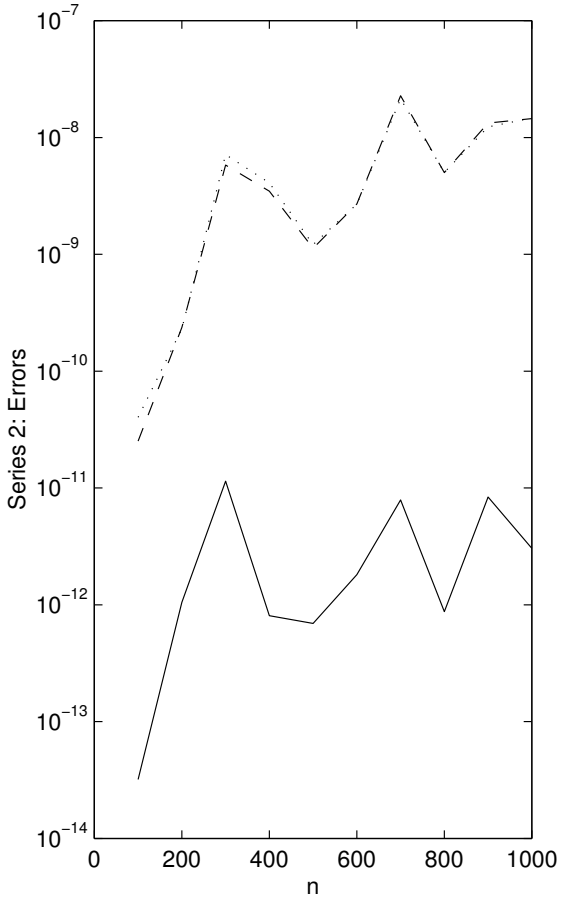


# Symmetric Indefinite Matrix

## Perturbation SMW Approach

**Errors on the IBM SMP (4xPPC604e/332Mhz) of DSYSV (—), P_DSYSV (- - -) and DPRSIV (. . .)**

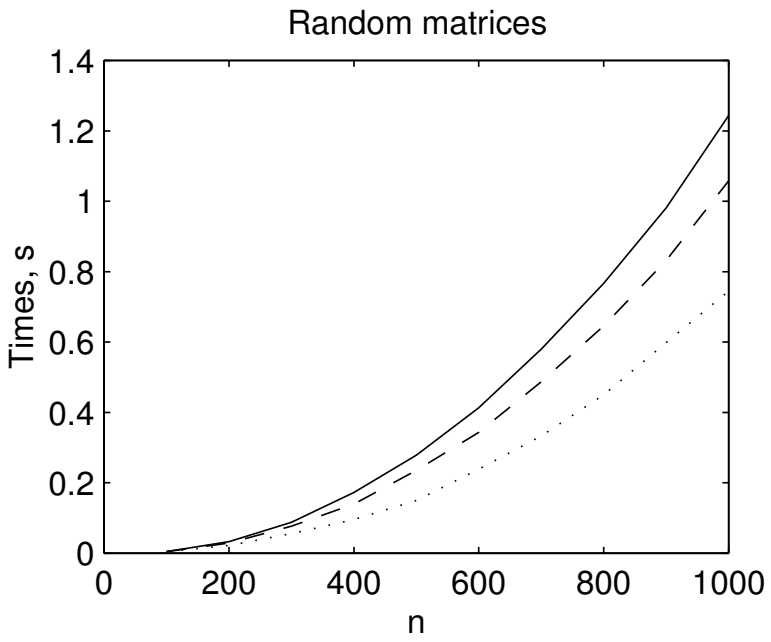


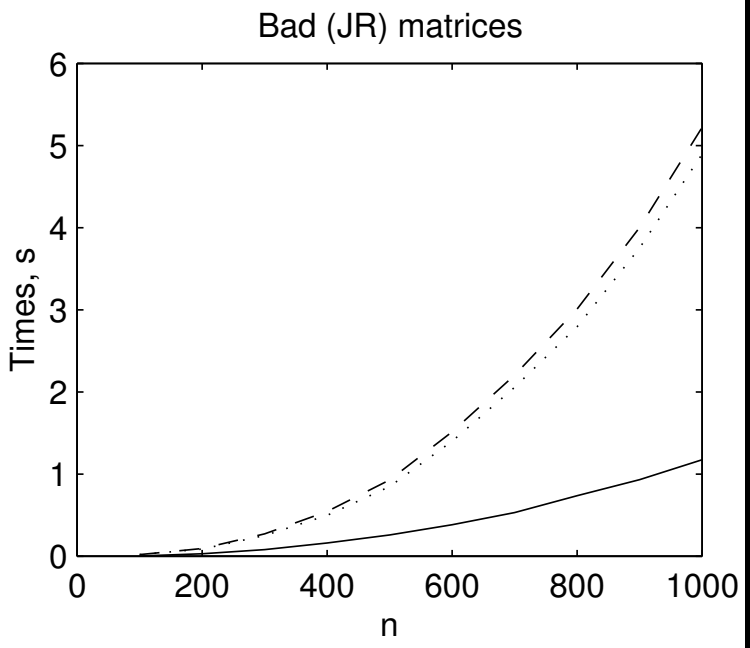



# Symmetric Indefinite Matrix

## Mixed Approach

**CPU times on the IBM SMP (4xPPC604e/332Mhz) of DSYSV (—), P_DSYSV (- - -) and DPRSIV (. . .)**

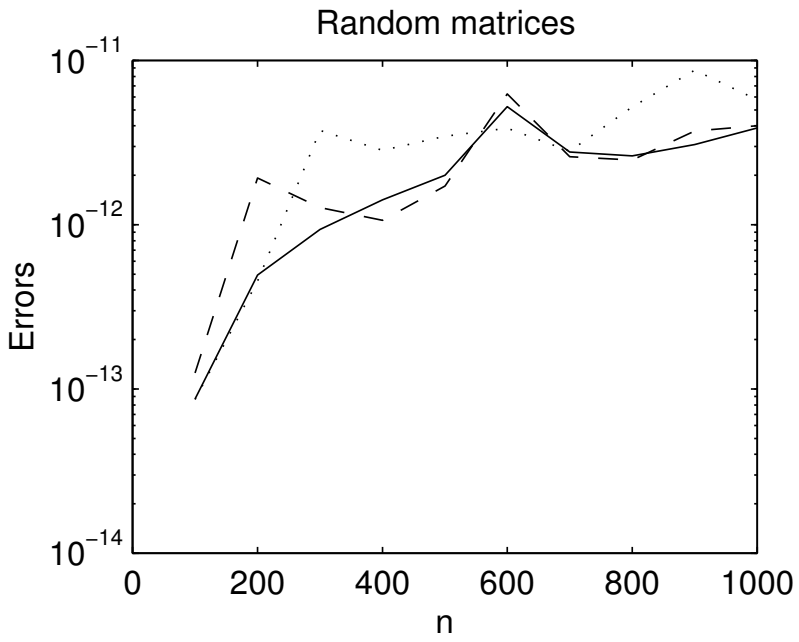


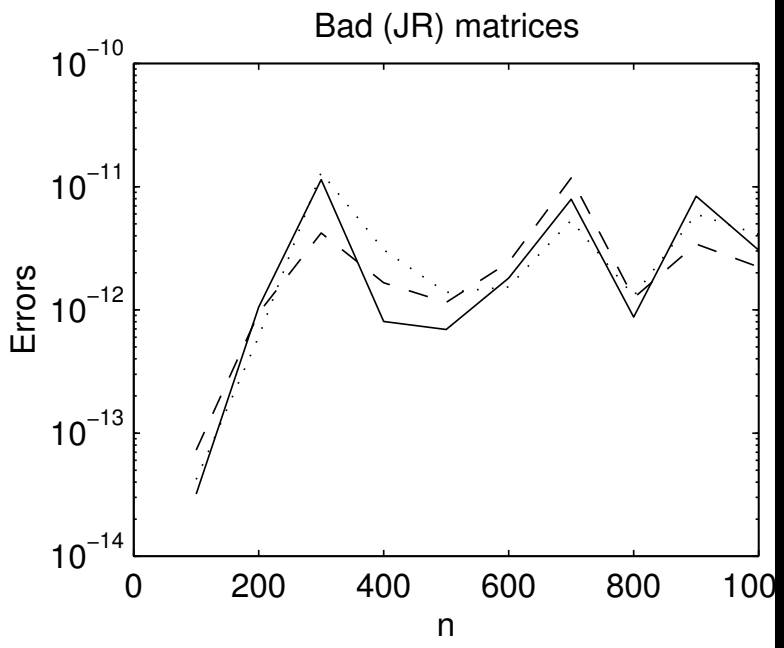



# Symmetric Indefinite Matrix

## Mixed Approach

**Errors on the IBM SMP (4xPPC604e/332Mhz) of DSYSV (—), P_DSYSV (- - -) and DPRSIV (. . .)**

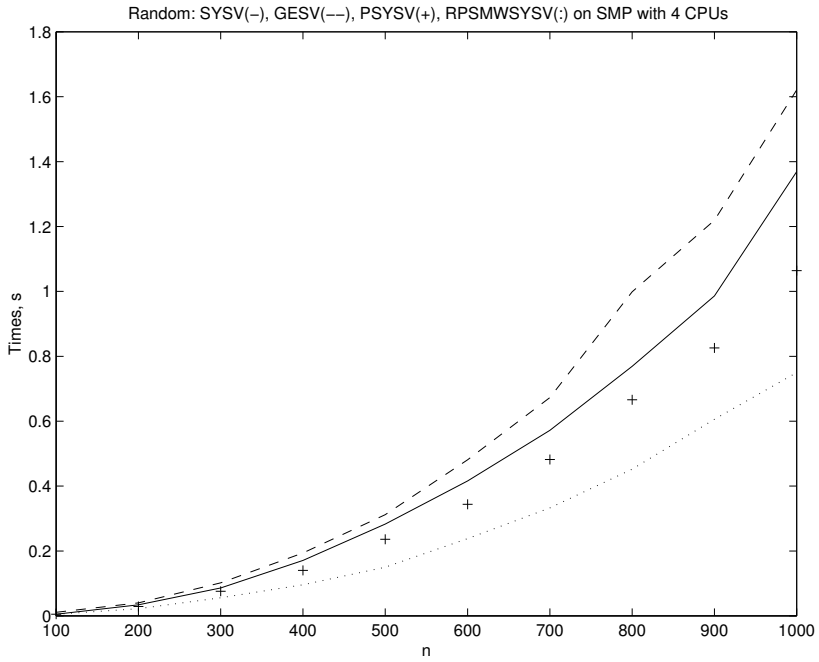


# Perturbation Approach, Numerical Results

## $LU$ and $LDL^T$

**CPU times on the IBM SMP (4xPPC604e/332Mhz) of DSYSV (—), DGESV (- - -) DPSYSV (+) and RPSMWSYSV (. . .)**



## Reference


- **F. Gustavson, A. Karaivanov, J. Waśniewski, and P. Yalamov. "A Recursive Formulation of Algorithms for Symmetric Indefinite Linear Systems". UNI•C report, 15 pages, Number UNIC-99-03, 1999. Submitted to SIAM J. Matrix Anal. Appl.**

**Symmetric/Hermitian Indefinite Matrices**

# Block Packed Algorithm

# Symmetric Indefinite Matrix

$A$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ & a_{22} & a_{23} & a_{24} \\ & & a_{33} & a_{34} \\ & & & a_{44} \end{pmatrix}$$

or

$A$

$$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

**A diagonal pivoting method is used to factor $A$ as**

$$A = P^T U D U^T P, \quad \text{if} \quad \mathbf{UPLO} = \text{'U'} \quad \textbf{or}$$

$$A = P^T L D L^T P, \quad \text{if} \quad \mathbf{UPLO} = \text{'L'}$$

- **$P$ is a permutation matrix**
- **$U$ and $L$ are unit upper and lower triangular matrices, respectively**
- **$D$ is a symmetric block diagonal with $1 \times 1$ and $2 \times 2$ diagonal blocks.**

# Bunch-Kaufman Pivoting Strategy

$\alpha = (1+\sqrt{17})/8$; $\lambda = |a_{r1}| = \max\{|a_{21}|, \ldots, |a_{n1}|\}$

**if** $\lambda > 0$

  **if** $|a_{11}| \geq \alpha\lambda$ **then** $s = 1; \quad P_1 = I$

  **else**

    $\sigma = |a_{pr}| = \max\{|a_{1r}, \ldots |a_{r-1,r}, |a_{r+1,r}|, \ldots, |a_{nr}|\}$

    **if** $\sigma|a_{11}| \geq \alpha\lambda^2$ **then** $s = 1, \; P_1 = I$

    **else if** $|a_{rr}| \geq \alpha\sigma$

      $s = 1$ **and choose** $P_1$ **so** $(P_1^T A\, P_1)_{11} = a_{rr}$

    **else**

      $s = 2$ **and choose** $P_1$ **so** $(P_1^T A\, P_1)_{21} = a_{r1}$

    **end**

  **end**

**end**

## Symmetric Indefinite Matrix

$$\mathbf{n} = 7, \quad \mathbf{memory\ needed} = \mathbf{n} \times (\mathbf{n}+1)/2 = 28$$

$$\begin{pmatrix}
a_{1,1_1} & & & & & & \\
a_{2,1_2} & a_{2,2_8} & & & & & \\
a_{3,1_3} & a_{3,2_9} & a_{3,3_{14}} & & & & \\
a_{4,1_4} & a_{4,2_{10}} & a_{4,3_{15}} & a_{4,4_{19}} & & & \\
a_{5,1_5} & a_{5,2_{11}} & a_{5,3_{16}} & a_{5,4_{20}} & a_{5,5_{23}} & & \\
a_{6,1_6} & a_{6,2_{12}} & a_{6,3_{17}} & a_{6,4_{21}} & a_{6,5_{24}} & a_{6,6_{26}} & \\
a_{7,1_7} & a_{7,2_{13}} & a_{7,3_{18}} & a_{7,4_{22}} & a_{7,5_{25}} & a_{7,6_{27}} & a_{7,7_{28}}
\end{pmatrix}$$

**The mapping of 7× 7 real symmetric, complex symmetric or complex Hermitian matrix for the LAPACK algorithm using the packed storage. Lower triangular case.**

## Symmetric Indefinite Matrix

**n** $= 7,$ **lda** $= 9,$ **nb** $= 2,$ **memory needed** $= 51$

$$
\left(\begin{array}{c|cc|ccc|cc}
a_{1,1_1} & \otimes & \odot & & & & & \\
a_{2,1_2} & a_{2,2_{11}} & \odot & & & & & \\
a_{3,1_3} & a_{3,2_{12}} & a_{3,3_{21}} & \otimes & \odot & & & \\
a_{4,1_4} & a_{4,2_{13}} & a_{4,3_{22}} & a_{4,4_{29}} & \odot & & & \\
a_{5,1_5} & a_{5,2_{14}} & a_{5,3_{23}} & a_{5,4_{30}} & a_{5,5_{37}} & \otimes & \odot & \\
a_{6,1_6} & a_{6,2_{15}} & a_{6,3_{24}} & a_{6,4_{31}} & a_{6,5_{38}} & a_{6,6_{43}} & \odot & \\
a_{7,1_7} & a_{7,2_{16}} & a_{7,3_{25}} & a_{7,4_{32}} & a_{7,5_{39}} & a_{7,6_{44}} & a_{7,7_{49}} & \\
\oslash & \oslash & \oslash & \oslash & \oslash & \oslash & \oslash & \\
\oslash & \oslash & \oslash & \oslash & \oslash & \oslash & \oslash &
\end{array}\right)
$$

**The mapping of 7$\times$ 7 real symmetric, complex symmetric or complex Hermitian matrix for the algorithm using the block packed overlapped storage. Lower triangular case.**

## Symmetric Indefinite Matrix

**n** $= 7,$ **lda** $= 9,$ **nb** $= 3,$ **memory needed** $= 54$

$$
\left(\begin{array}{ccc|ccc|cc}
a_{1,1_1} & \otimes & \otimes & \odot & & & & \\
a_{2,1_2} & a_{2,2_{11}} & \otimes & \odot & & & & \\
a_{3,1_3} & a_{3,2_{12}} & a_{3,3_{21}} & \odot & & & & \\
a_{4,1_4} & a_{4,2_{13}} & a_{4,3_{22}} & a_{4,4_{31}} & \otimes & \otimes & \odot & \\
a_{5,1_5} & a_{5,2_{14}} & a_{5,3_{23}} & a_{5,4_{32}} & a_{5,5_{38}} & \otimes & \odot & \\
a_{6,1_6} & a_{6,2_{15}} & a_{6,3_{24}} & a_{6,4_{33}} & a_{6,5_{39}} & a_{6,6_{45}} & \odot & \\
a_{7,1_7} & a_{7,2_{16}} & a_{7,3_{25}} & a_{7,4_{34}} & a_{7,5_{40}} & a_{7,6_{46}} & a_{7,7_{52}} & \\
\oslash & \oslash & \oslash & \oslash & \oslash & \oslash & \oslash & \\
\oslash & \oslash & \oslash & \oslash & \oslash & \oslash & \oslash_{54} &
\end{array}\right)
$$

**The mapping of 7$\times$ 7 real symmetric, complex symmetric or complex Hermitian matrix for the algorithm using the block packed overlapped storage. Lower triangular case.**

## Symmetric Indefinite Matrix

```
SUBROUTINE SSPTRF( UPLO, N, AP, &
      IPIV, INFO)
!
 CHARACTER UPLO
 INTEGER INFO, N
!
 INTEGER IPIV( * )
 REAL AP( * )
```

## Symmetric Indefinite Matrix

**uplo = 'l'**, **n** $= 7$, **lda** $= 9$, **nb** $= 2$, **length of AP** $= 28$

$$\mathbf{AP} = \left(\begin{array}{c|c|c|c} a_{1,1_1} & \otimes & \odot & \diamond \\ a_{2,1_2} & a_{2,2_{11}} & \odot & \\ a_{3,1_3} & a_{3,2_{12}} & a_{3,3_{21}} & \\ a_{4,1_4} & a_{4,2_{13}} & a_{4,3_{22}} & \\ a_{5,1_5} & a_{5,2_{14}} & a_{5,3_{23}} & \\ a_{6,1_6} & a_{6,2_{15}} & a_{6,3_{24}} & \\ a_{7,1_7} & a_{7,2_{16}} & a_{7,3_{25}} & \\ \oslash & \oslash & \oslash & \\ \oslash & \oslash & \oslash & \end{array}\right)$$

**length of BUFF** $= 31$

$$\mathbf{BUFF} = \left(\begin{array}{cc|cc|c} a_{3,3_1} & \otimes & \odot & & \\ a_{4,3_2} & a_{4,4_9} & \odot & & \\ a_{5,3_3} & a_{5,4_{10}} & a_{5,5_{17}} & \otimes & \odot \\ a_{6,3_4} & a_{6,4_{11}} & a_{6,5_{18}} & a_{6,6_{23}} & \odot \\ a_{7,3_5} & a_{7,4_{12}} & a_{7,5_{19}} & a_{7,6_{24}} & a_{7,7_{29}} \\ \oslash & \oslash & \oslash & \oslash & \oslash \\ \oslash & \oslash & \oslash & \oslash & \oslash \end{array}\right)$$

**Block packed overlapped storage. Lower triangular case.**

## Symmetric Indefinite Matrix

**n** $= 7,$ **lda** $= 9,$ **nb** $= 3,$ **length of BUFF** $= 36$

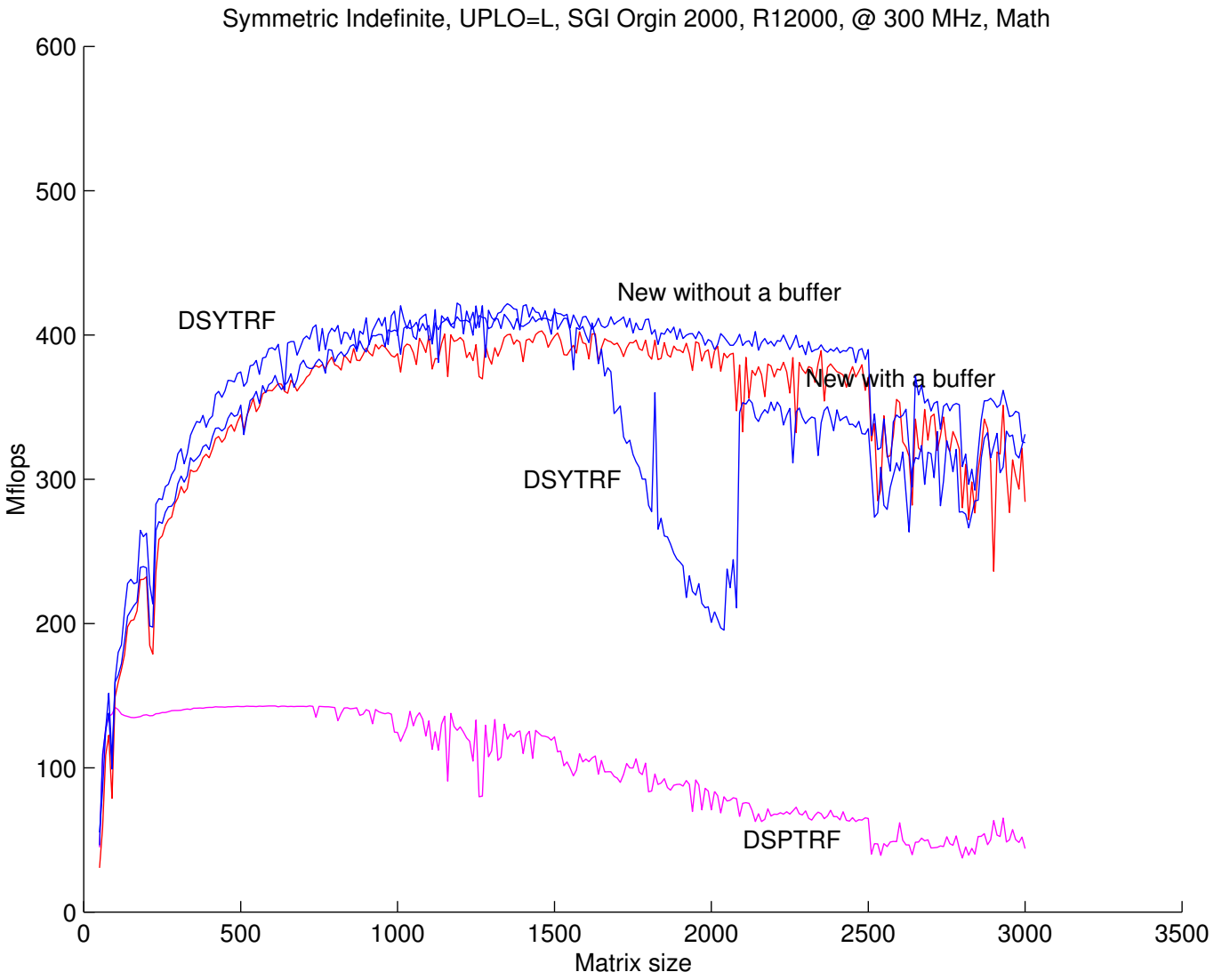
$$\mathbf{BUFF} = \left(\begin{array}{cccc} a_{1,1_1} & \otimes & \otimes & \odot \\ a_{2,1_2} & a_{2,2_{11}} & \otimes & \odot \\ a_{3,1_3} & a_{3,2_{12}} & a_{3,3_{21}} & \odot \\ a_{4,1_4} & a_{4,2_{13}} & a_{4,3_{22}} & a_{4,4_{31}} \\ a_{5,1_5} & a_{5,2_{14}} & a_{5,3_{23}} & a_{5,4_{32}} \\ a_{6,1_6} & a_{6,2_{15}} & a_{6,3_{24}} & a_{6,4_{33}} \\ a_{7,1_7} & a_{7,2_{16}} & a_{7,3_{25}} & a_{7,4_{34}} \\ \oslash & \oslash & \oslash & \oslash \\ \oslash & \oslash & \oslash & \oslash \end{array}\right)$$

**length of AP** $= 28$

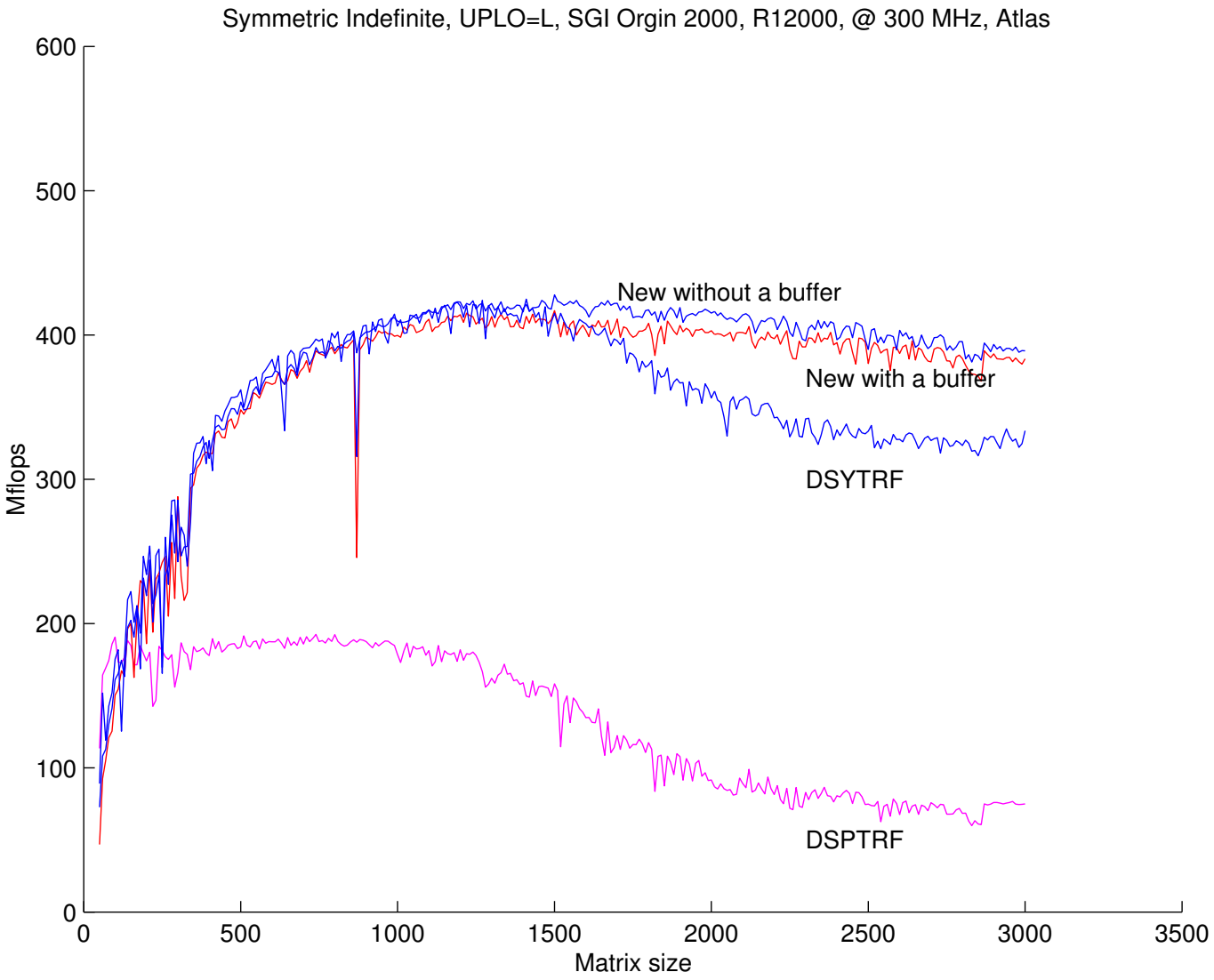
$$\mathbf{AP} = \left(\begin{array}{ccccc} a_{4,4_1} & \otimes & \otimes & \odot & \diamond \\ a_{5,4_2} & a_{5,5_8} & \otimes & \odot & \diamond \\ a_{6,4_3} & a_{6,5_9} & a_{6,6_{15}} & \odot & \diamond \\ a_{7,4_4} & a_{7,5_{10}} & a_{7,6_{16}} & a_{7,7_{22}} & \diamond \\ \oslash & \oslash & \oslash & \oslash & \\ \oslash & \oslash & \oslash & \oslash & \end{array}\right)$$

**Block packed overlapped storage. Lower triangular case.**

# $LDL^T$, Four Factorization Algorithms, SGI Lib



# $LDL^T$, Four Factorization Algorithms, Atlas Lib



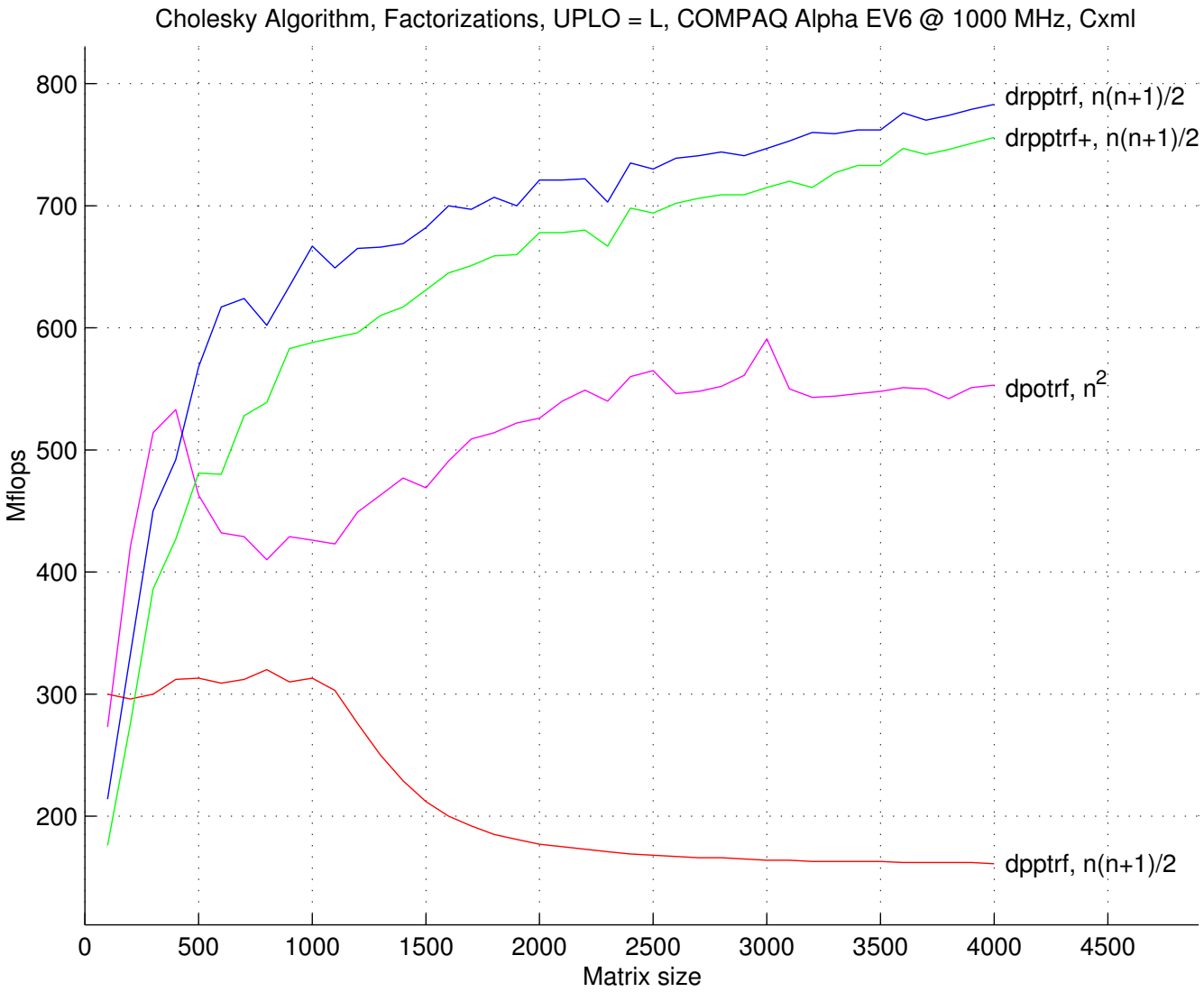
## Reference


- **F. Gustavson, A. Karaivanov, M. Marinova, J. Wasniewski, and P. Yalamov. "A Fast Minimal Storage Symmetric Indefinite Solver". In Para'2000 Conference Proceedings, Bergen, Norway, 2000.**

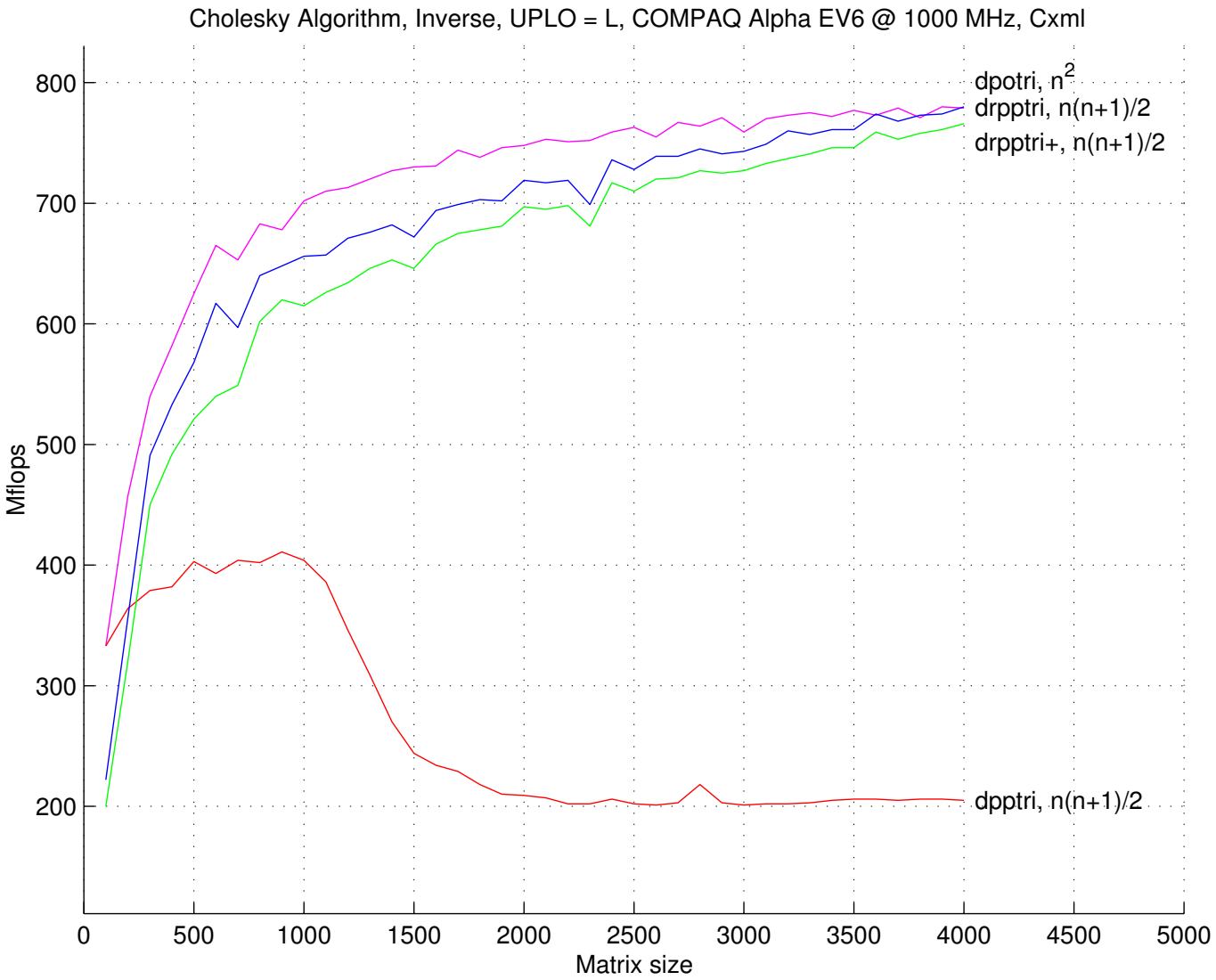
**Symmetric Matrices**

# More Results

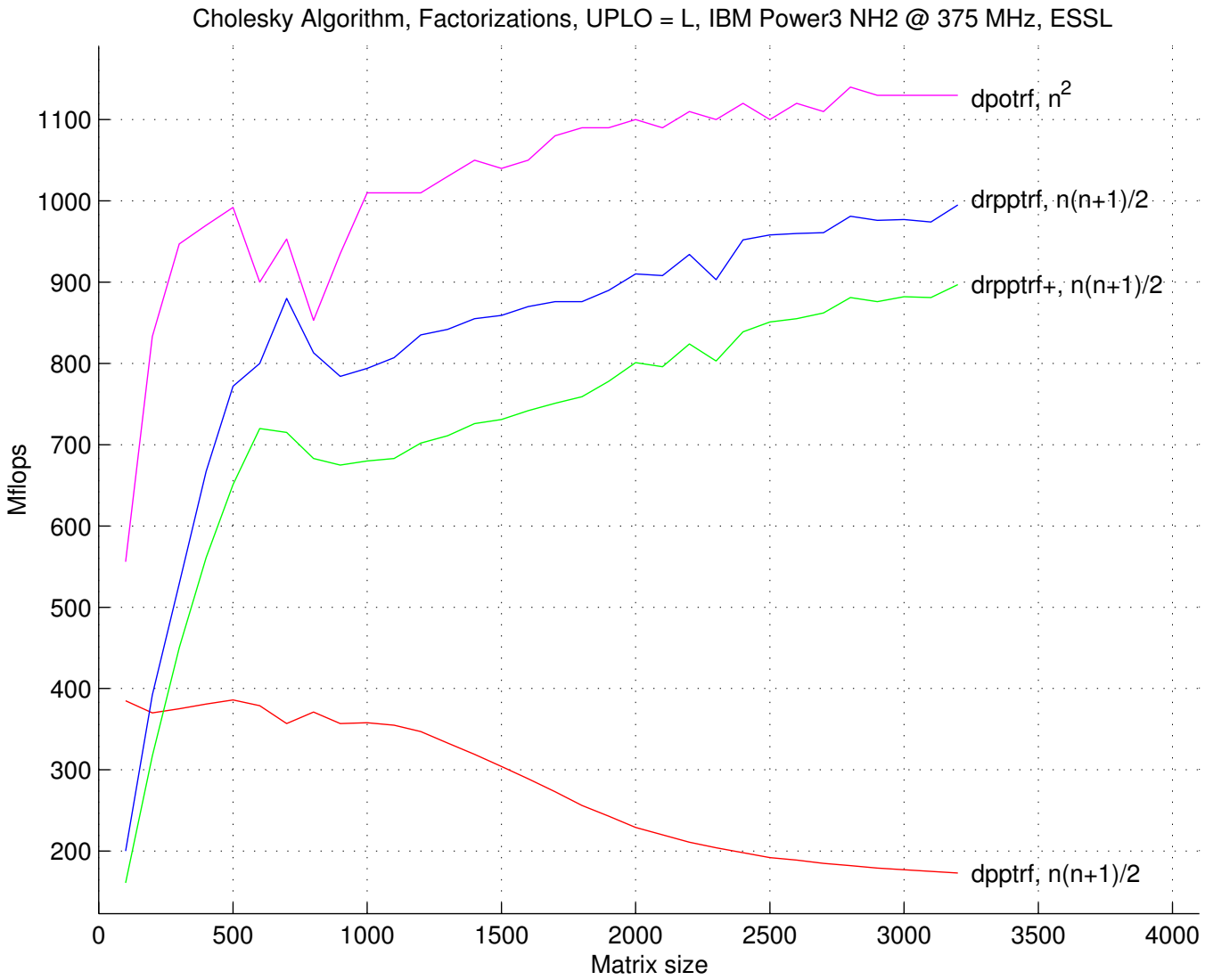
# Factorization



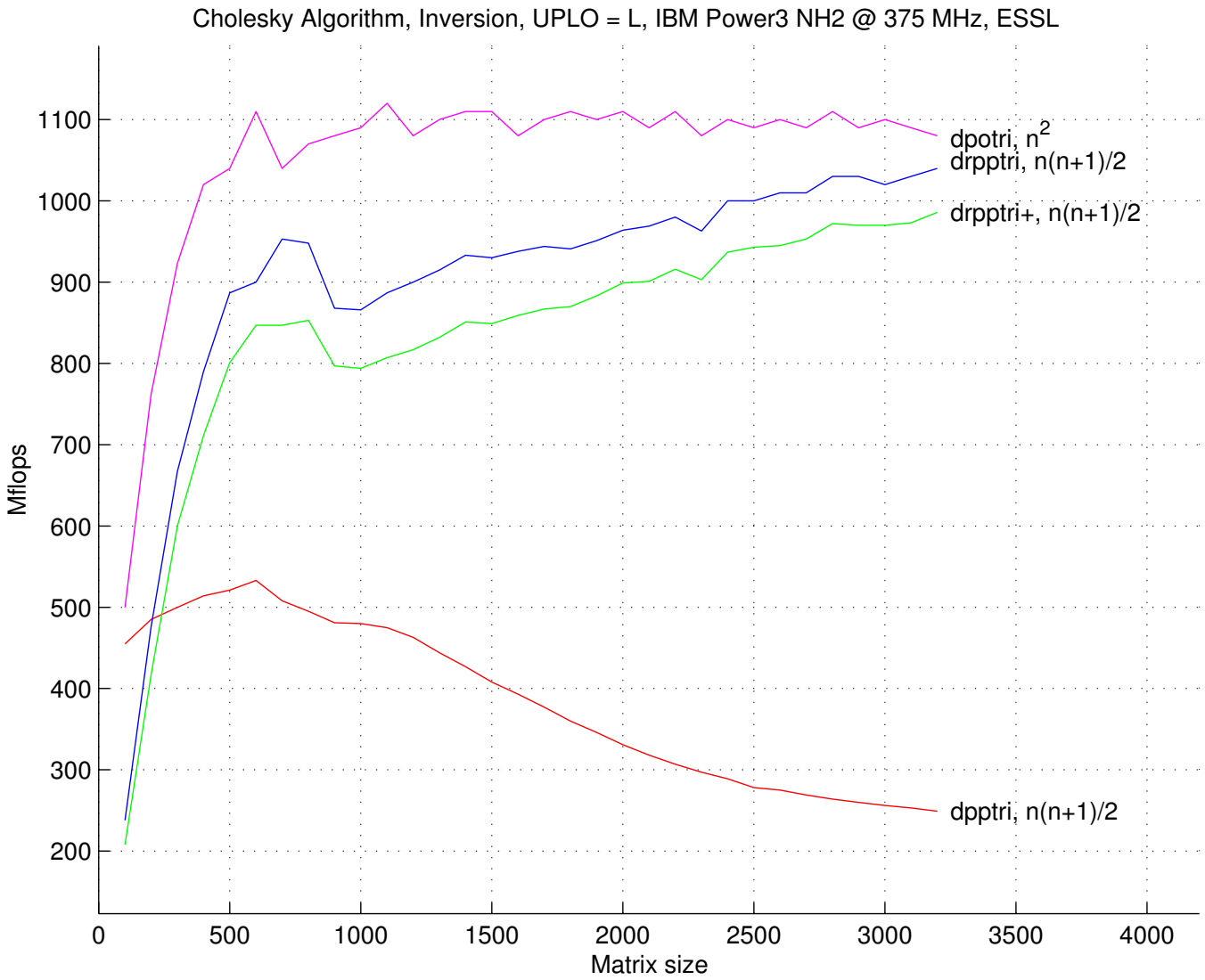
# Inversion


Cholesky Algorithm, Inverse, UPLO = L, COMPAQ Alpha EV6 @ 1000 MHz, Cxml
Mflops
Matrix size
800
700
600
500
400
300
200
0
500
1000
1500
2000
2500
3000
3500
4000
4500
5000
dpotri, n2
drpptri, n(n+1)/2
drpptri+, n(n+1)/2
dpptri, n(n+1)/2

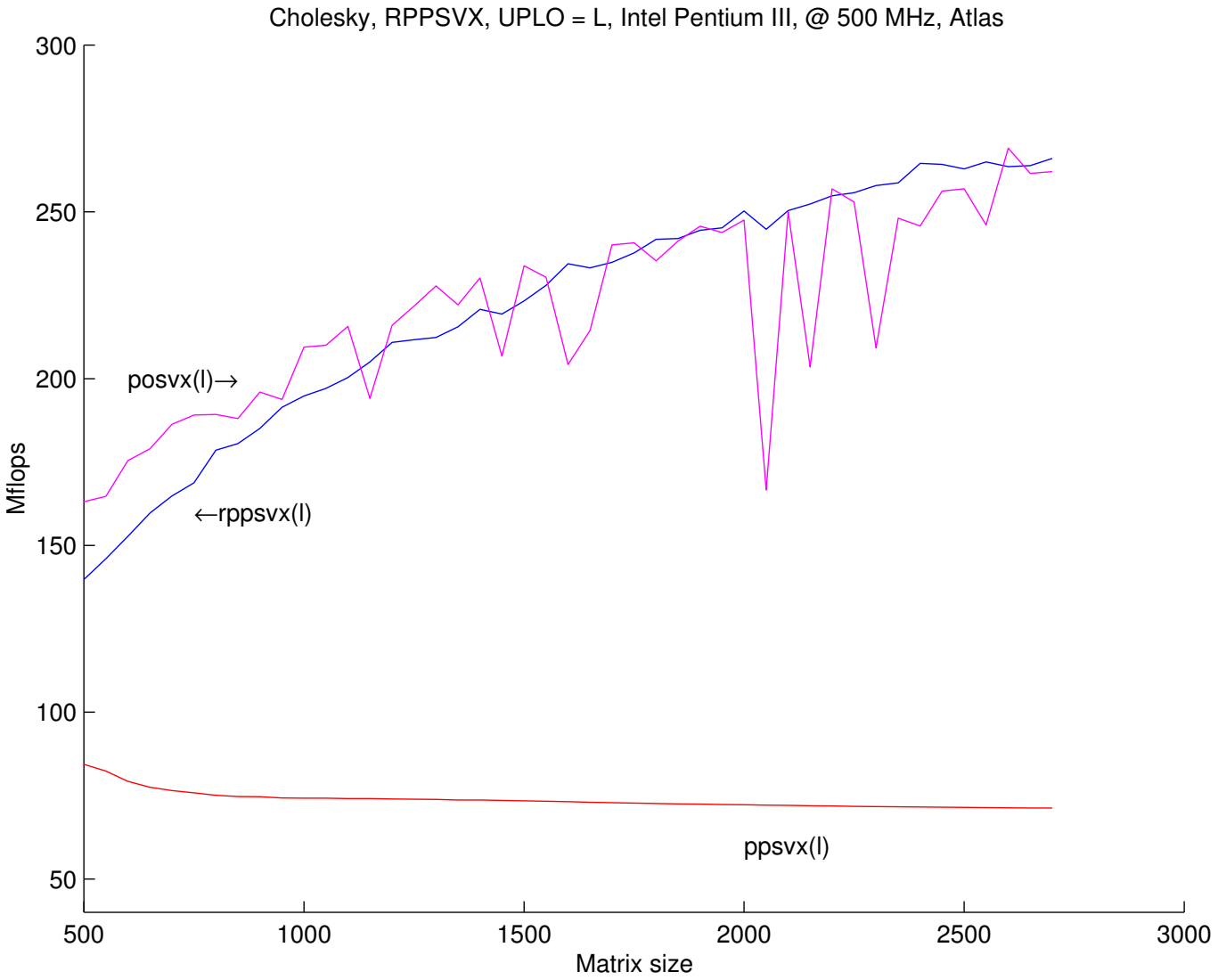
# Factorization



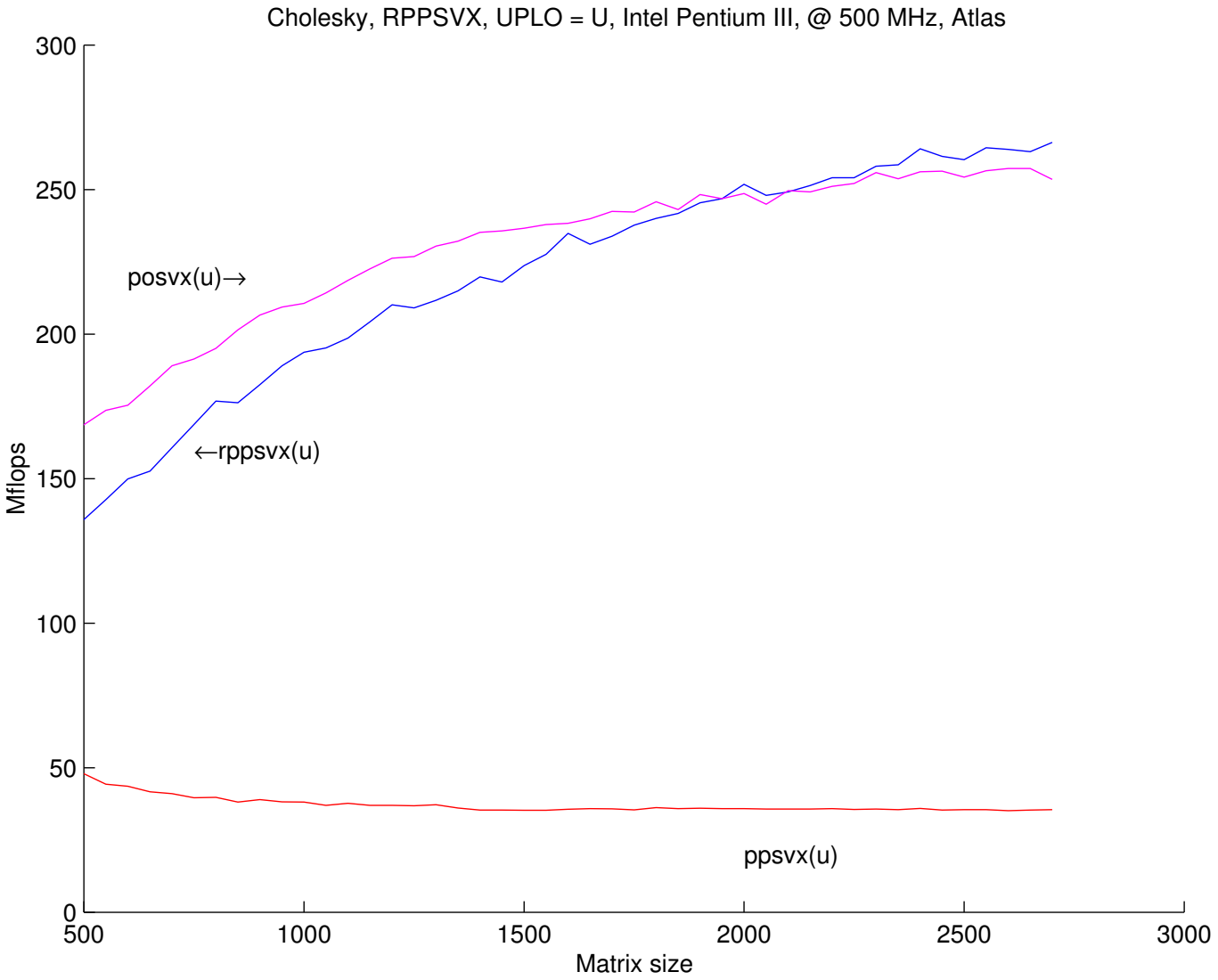
# Inversion


Cholesky Algorithm, Inversion, UPLO = L, IBM Power3 NH2 @ 375 MHz, ESSL
Mflops
Matrix size
dpotri, n^2
drpptri, n(n+1)/2
drpptri+, n(n+1)/2
dpptri, n(n+1)/2
1100
1000
900
800
700
600
500
400
300
200
0
500
1000
1500
2000
2500
3000
3500
4000

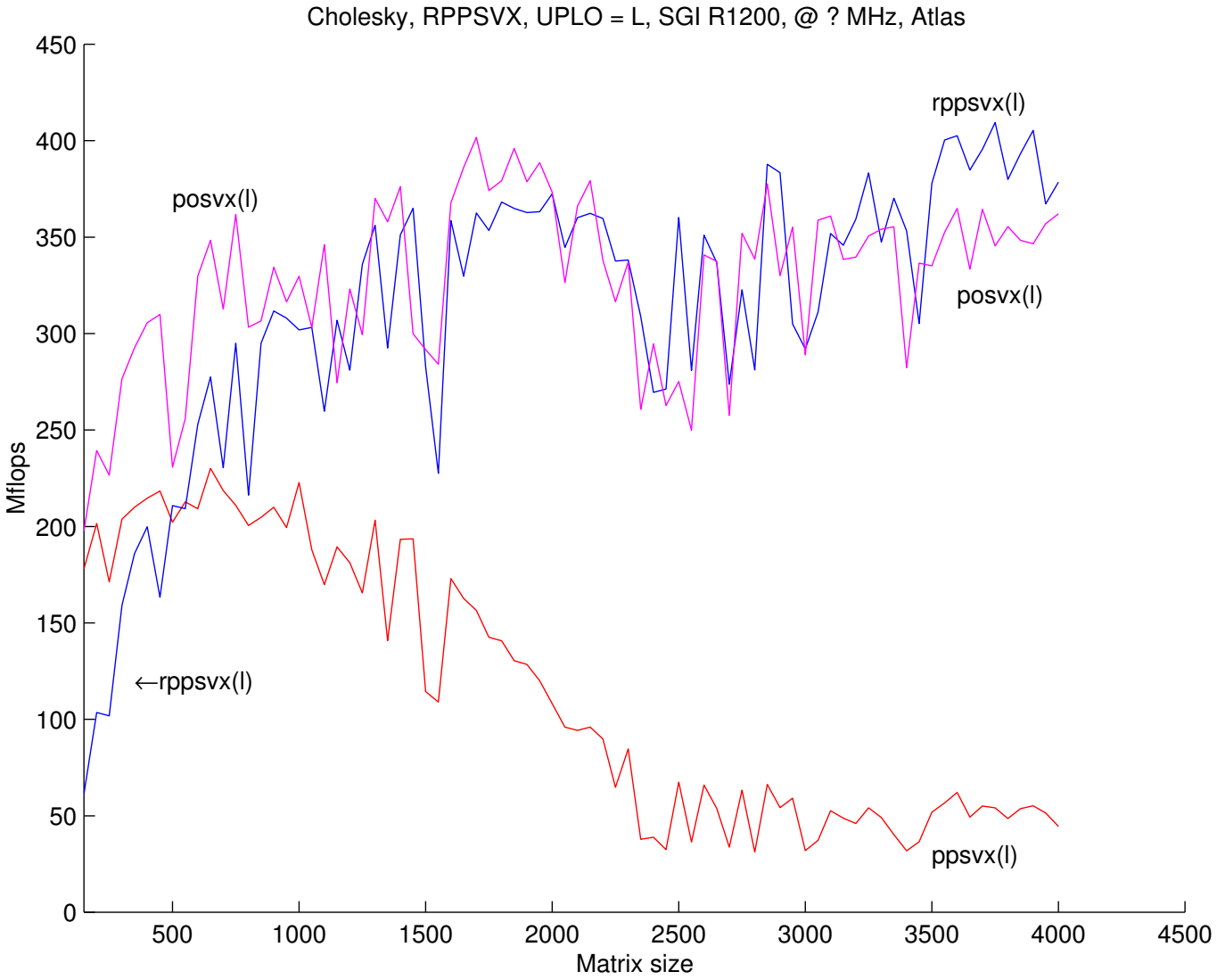
# Cholesky


Cholesky, RPPSVX, UPLO = L, Intel Pentium III, @ 500 MHz, Atlas
Mflops
Matrix size
300
250
200
150
100
50
500
1000
1500
2000
2500
3000
posvx(l)→
←rppsvx(l)
ppsvx(l)

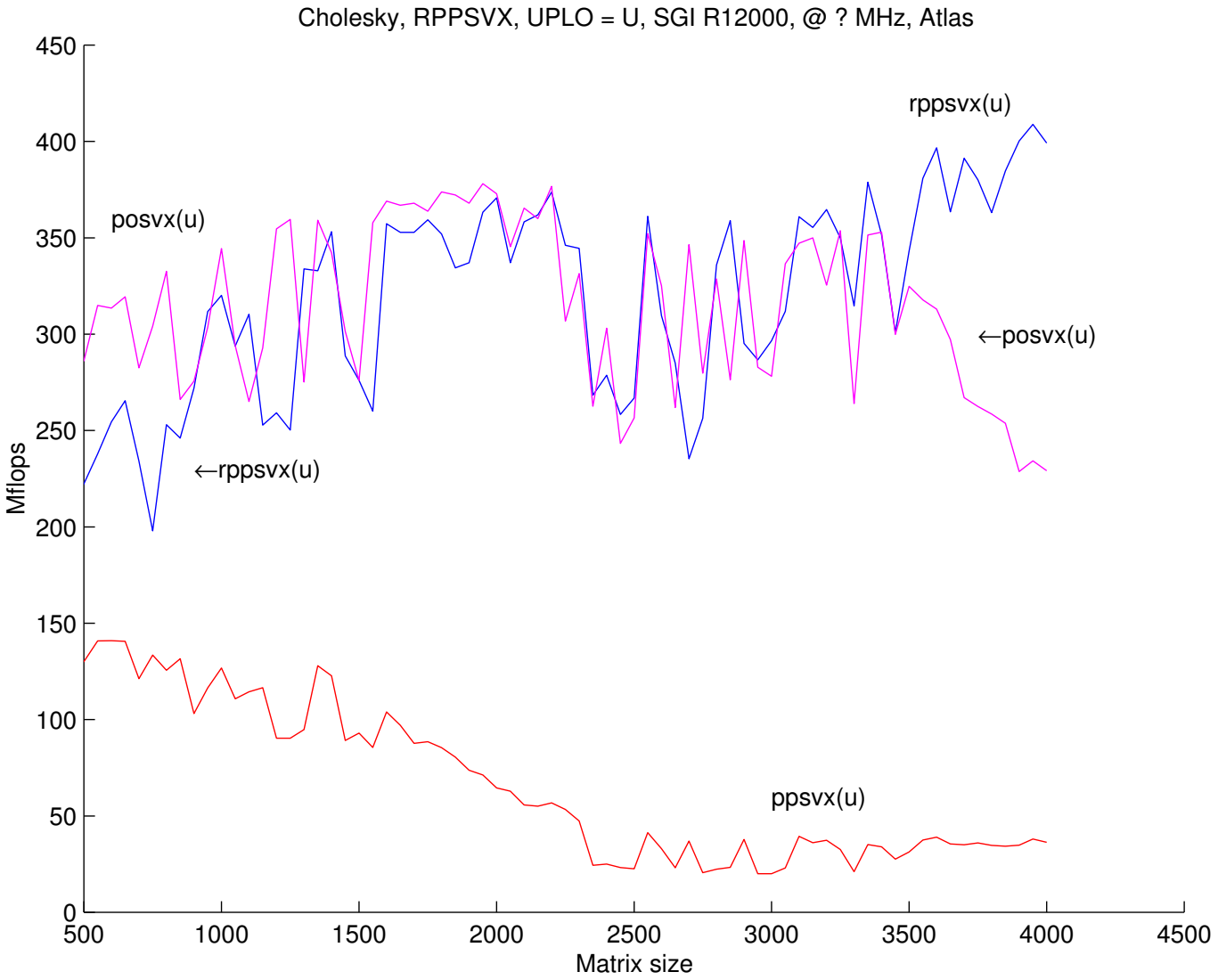
# Cholesky


Cholesky, RPPSVX, UPLO = U, Intel Pentium III, @ 500 MHz, Atlas
Mflops
Matrix size
posvx(u)→
←rppsvx(u)
ppsvx(u)
0
50
100
150
200
250
300
500
1000
1500
2000
2500
3000

# Cholesky


Cholesky, RPPSVX, UPLO = L, SGI R1200, @ ? MHz, Atlas
Mflops
Matrix size
rppsvx(l)
posvx(l)
posvx(l)
←rppsvx(l)
ppsvx(l)
0
50
100
150
200
250
300
350
400
450
500
1000
1500
2000
2500
3000
3500
4000
4500

# Cholesky


Cholesky, RPPSVX, UPLO = U, SGI R12000, @ ? MHz, Atlas
Mflops
Matrix size
rppsvx(u)
posvx(u)
←posvx(u)
←rppsvx(u)
ppsvx(u)
0
50
100
150
200
250
300
350
400
450
500
1000
1500
2000
2500
3000
3500
4000
4500

# New Data Storage Formats for Dense Matrices Lead to Variety of High-Performance Algorithms (Solving Linear Systems of Equations)

**Jerzy Waśniewski**

**Emeritus Senior Research Professor**

**Department of Informatics & Mathematical Modeling**

**Technical University of Denmark**

**DTU, Bldg. 305**

**DK - 2800 Lyngby, Denmark**

**e-mail: jw@imm.dtu.dk**

**http://www.imm.dtu.dk/~jw/Lectures/ppam05.pdf**

**August 14, 2005**