Formal Methods for High-Performance Linear Algebra Libraries

John Gunnels
Robert van de Geijn

Department of Computer Sciences
The University of Texas at Austin
{gunnels,rvdg}@cs.utexas.edu

Oct. 3, 2000

Funding

- NASA (Remote Exploration and Experimentation Project)
- NASA (Earth and Space Sciences Project)
- NSF (PRISM project)
- The Intel Research Council

Overview

- Motivation
- A Formal Approach to Deriving Linear Algebra Algorithms
- Impact on Implementation of Linear Algebra Algorithms
- Impact on Library Development and Maintainability
- Impact on Performance
- Beyond Dense Linear Algebra
- Conclusion

Motivation

Consider Matrix-matrix multiplication

- Problem: Compute $C = AB + C$
  - ⋆ Sequential algorithm:

$$
\begin{array}{l}
\textbf{for } i = 1 : m \\
\quad \textbf{for } j = 1 : n \\
\quad\quad \textbf{for } p = 1 : k \\
\quad\quad\quad \gamma_{ij} = \alpha_{ip}\beta_{pj} + \gamma_{ij} \\
\quad\quad \textbf{endfor} \\
\quad \textbf{endfor} \\
\textbf{endfor}
\end{array}
$$

- Requirements:
  - ⋆ High performance
  - ⋆ Distributed memory parallel
  - ⋆ All cases: AB, A^TB, AB^T, A^TB^T
- Solution: Implemented as part of a well known package

Problem:

Unmanageable Complexity?

- For a complete dense linear algebra library we need a substantial library of routines:
  - ⋆ Functionality.
    - ◇ LAPACK and beyond.
  - ⋆ Accomodating multiple layers of memory hierarchy:
    - ◇ Sequential at each level of memory hierarchy
    - ◇ Shared memory parallel
    - ◇ Distributed memory parallel
    - ◇ Out-of-core (parallel)
  - ⋆ Variants
    - ◇ Three or more variants per operation per layer of hierarchy

For each operation we need as many as several dozen implementations.

Requirements

- Correctness
- Coverage
- Performance
  - ⋆ A variant for all occasions
  - ⋆ Parameterized cost estimates
  - ⋆ Strategy for picking variants
- Numerical Stability

Solution

- A formal approach to deriving all algorithms
  - ⋆ Correctness
  - ⋆ Coverage
- The algorithms are the code
  - ⋆ No need for debugging and testing
  - ⋆ No need for documentation
- Performance
  - ⋆ Some variants perform better than others depending on machine and problem parameters

A Formal Approach to Deriving Linear Algebra Algorithms

A Well-Understood Operation: LU Factorization without pivoting

$$A \rightarrow LU$$

where

- A, L and U are $n \times n$,
- L is unit lower triangular, and
- U is upper triangular.

A typical derivation of the right-looking algorithm:

Partition and equate

$$\left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right) = \left(\begin{array}{c|c} L_{11} & 0 \\ \hline L_{21} & L_{22} \end{array}\right) \left(\begin{array}{c|c} U_{11} & U_{12} \\ \hline 0 & U_{22} \end{array}\right) = \left(\begin{array}{c|c} L_{11}U_{11} & L_{11}U_{12} \\ \hline L_{21}U_{11} & L_{21}U_{12} + L_{22}U_{22} \end{array}\right)$$

where A_{11}, L_{11}, and U_{11} are $b \times b$. Then the following equalities must hold:

$$\begin{array}{rcl|rcl} A_{11} & = & L_{11}U_{11} & A_{12} & = & L_{11}U_{12} \\ \hline A_{21} & = & L_{21}U_{11} & A_{22} & = & L_{21}U_{12} + L_{22}U_{22} \end{array}$$

giving the algorithm

- $$\begin{array}{c||c|c} & & \\ \hline\hline & A_{11} \leftarrow \{L\backslash U\}_{11} = LU(A_{11}) & A_{12} \leftarrow U_{12} = L_{11}^{-1}A_{12} \\ \hline & A_{21} \leftarrow L_{21} = A_{21}U_{11}^{-1} & A_{22} \leftarrow A_{22} - L_{21}U_{12} \end{array}$$
- recursively factor $A_{22} \rightarrow L_{22}U_{22}$

A derivation of the left-looking algorithm:

Partition A, L, and U conformally

$$A = \left(\begin{array}{c||c|c} A_{00} & A_{01} & A_{02} \\ \hline\hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right), L = \left(\begin{array}{c||c|c} L_{00} & 0 & 0 \\ \hline\hline L_{10} & L_{11} & 0 \\ \hline L_{20} & L_{21} & L_{22} \end{array}\right), \text{ and } U = \left(\begin{array}{c||c|c} U_{00} & U_{01} & U_{02} \\ \hline\hline 0 & U_{11} & U_{12} \\ \hline 0 & 0 & U_{22} \end{array}\right)$$

where $\{A, L, U\}_{00}$ are $k \times k$ and $\{A, L, U\}_{11}$ are $b \times b$. Update current contents of A from $A^{(k)} = \left(\begin{array}{c||c|c} \{L\backslash U\}_{00} & A_{01} & A_{02} \\ \hline\hline L_{10} & A_{11} & A_{12} \\ \hline L_{20} & A_{21} & A_{22} \end{array}\right)$ to $A^{(k)} = \left(\begin{array}{c||c|c} \{L\backslash U\}_{00} & U_{01} & A_{02} \\ \hline\hline L_{10} & \{L\backslash U\}_{11} & A_{12} \\ \hline L_{20} & L_{21} & A_{22} \end{array}\right)$.

Algorithm:

- $$\begin{array}{c||c|c} & A_{01} \leftarrow U_{01} = L_{00}^{-1} A_{01} & \\ \hline\hline & A_{11} \leftarrow \{L\backslash U\}_{11} = LU(A_{11} - L_{10} U_{01}) & \\ \hline & A_{21} \leftarrow (A_{21} - L_{20} U_{01}) U_{11}^{-1} & \end{array}$$
- Repartition and continue

Observation

- The two derivations appear to be quite different.
- We now show how to unify the approach using formal methods.
- The key is to ask the question "What is in the matrix at a given stage?"
- Would like to derive all important variants

A Formal Approach to LU Factorization

Partition

$$A = \left(\begin{array}{c||c} A_{TL}^{(k)} & A_{TR}^{(k)} \\ \hline\hline A_{BL}^{(k)} & A_{BR}^{(k)} \end{array}\right), L = \left(\begin{array}{c||c} L_{TL}^{(k)} & 0 \\ \hline\hline L_{BL}^{(k)} & L_{BR}^{(k)} \end{array}\right), \text{ and } U = \left(\begin{array}{c||c} U_{TL}^{(k)} & U_{TR}^{(k)} \\ \hline\hline 0 & U_{BR}^{(k)} \end{array}\right) \qquad (1)$$

where $A_{TL}^{(k)}$, $L_{TL}^{(k)}$, and $U_{TL}^{(k)}$ are all $k \times k$ matrices. Then

$$\left(\begin{array}{c||c} A_{TL}^{(k)} & A_{TR}^{(k)} \\ \hline\hline A_{BL}^{(k)} & A_{BR}^{(k)} \end{array}\right) = \left(\begin{array}{c||c} L_{TL}^{(k)} & 0 \\ \hline\hline L_{BL}^{(k)} & L_{BR}^{(k)} \end{array}\right) \left(\begin{array}{c||c} U_{TL}^{(k)} & U_{TR}^{(k)} \\ \hline\hline 0 & U_{BR}^{(k)} \end{array}\right) = \left(\begin{array}{c||c} L_{TL}^{(k)} U_{TL}^{(k)} & L_{TL}^{(k)} U_{TR}^{(k)} \\ \hline\hline L_{BL}^{(k)} U_{TL}^{(k)} & L_{BL}^{(k)} U_{TR}^{(k)} + L_{BR}^{(k)} U_{BR}^{(k)} \end{array}\right)$$

and thus the following equalities must hold:

$$\begin{array}{c||c} A_{TL}^{(k)} = L_{TL}^{(k)} U_{TL}^{(k)} & A_{TR}^{(k)} = L_{TL}^{(k)} U_{TR}^{(k)} \\ \hline\hline A_{BL}^{(k)} = L_{BL}^{(k)} U_{TL}^{(k)} & A_{BR}^{(k)} = L_{BL}^{(k)} U_{TR}^{(k)} + L_{BR}^{(k)} U_{BR}^{(k)} \end{array}$$

Here "TL", "TR", "BL" and "BR" represent where we are in the matrix.
Question: What can be in the matrix at an intermediate step?

$$
\left(\begin{array}{c||c}
A_{TL}^{(k)} = L_{TL}^{(k)} U_{TL}^{(k)} & A_{TR}^{(k)} = L_{TL}^{(k)} U_{TR}^{(k)} \\
\hline
A_{BL}^{(k)} = L_{BL}^{(k)} U_{TL}^{(k)} & A_{BR}^{(k)} = L_{BL}^{(k)} U_{TR}^{(k)} + L_{BR}^{(k)} U_{BR}^{(k)}
\end{array}\right)
$$

Possible conditions (loop-invariants):

Eager

$\{L \backslash U\}_{TL}^{(k)}$	$U_{TR}^{(k)}$
$L_{BL}^{(k)}$	$A_{BR}^{(k)} - L_{BL}^{(k)} U_{TR}^{(k)}$

Lazy

$\{L \backslash U\}_{TL}^{(k)}$	$A_{TR}^{(k)}$
$A_{BL}^{(k)}$	$A_{BR}^{(k)}$

Row-lazy

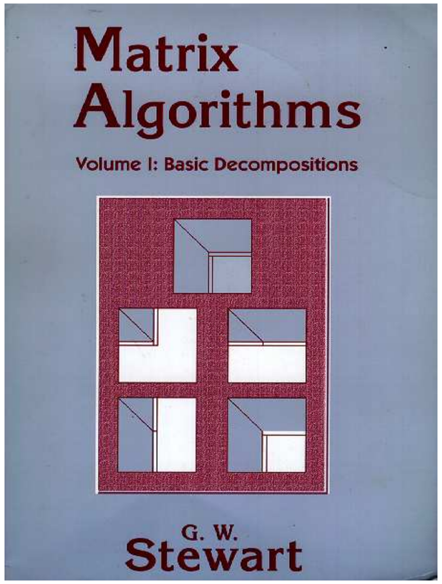
$\{L \backslash U\}_{TL}^{(k)}$	$U_{TR}^{(k)}$
$A_{BL}^{(k)}$	$A_{BR}^{(k)}$

Column-lazy

$\{L \backslash U\}_{TL}^{(k)}$	$A_{TR}^{(k)}$
$L_{BL}^{(k)}$	$A_{BR}^{(k)}$

Row-Column-lazy

$\{L \backslash U\}_{TL}^{(k)}$	$U_{TR}^{(k)}$
$L_{BL}^{(k)}$	$A_{BR}^{(k)}$



How to move the computation ahead

- Consider the "Lazy" condition where currently the matrix holds

$$\left(\begin{array}{c||c} \{L\backslash U\}_{TL}^{(k)} & A_{TR}^{(k)} \\ \hline A_{BL}^{(k)} & A_{BR}^{(k)} \end{array}\right)$$

- Identify the parts to be update: Repartition

$$\left(\begin{array}{c||c} \{L\backslash U\}_{TL}^{(k)} & A_{TR}^{(k)} \\ \hline A_{BL}^{(k)} & A_{BR}^{(k)} \end{array}\right) = \left(\begin{array}{c||c|c} \{L\backslash U\}_{00}^{(k)} & A_{01}^{(k)} & A_{02}^{(k)} \\ \hline A_{10}^{(k)} & A_{11}^{(k)} & A_{12}^{(k)} \\ A_{20}^{(k)} & A_{21}^{(k)} & A_{22}^{(k)} \end{array}\right)$$

where $A_{11}^{(k)}$ is $b \times b$.

- Update the appropriate parts

$$\left(\begin{array}{c||c|c} \{L\backslash U\}_{00}^{(k)} & A_{01}^{(k)} & A_{02}^{(k)} \\ \hline A_{10}^{(k)} & A_{11}^{(k)} & A_{12}^{(k)} \\ A_{20}^{(k)} & A_{21}^{(k)} & A_{22}^{(k)} \end{array}\right) \leftarrow \left(\begin{array}{c||c|c} \{L\backslash U\}_{00}^{(k)} & U_{01}^{(k)} & A_{02}^{(k)} \\ \hline L_{10}^{(k)} & \{L\backslash U\}_{11}^{(k)} & A_{12}^{(k)} \\ A_{20}^{(k)} & A_{21}^{(k)} & A_{22}^{(k)} \end{array}\right) = \left(\begin{array}{c||c|c} \{L\backslash U\}_{00}^{(k)} & L_{00}^{(k)\,-1} A_{01}^{(k)} & A_{02}^{(k)} \\ \hline A_{10}^{(k)} U_{00}^{(k)\,-1} & LU(A_{11}^{(k)} - L_{10}^{(k)} U_{01}^{(k)} & A_{12}^{(k)} \\ A_{20}^{(k)} & A_{21}^{(k)} & A_{22}^{(k)} \end{array}\right)$$

- Move the boundary

$$\left(\begin{array}{c||c} \{L\backslash U\}_{TL}^{(k+b)} & A_{TR}^{(k+b)} \\ \hline A_{BL}^{(k+b)} & A_{BR}^{(k+b)} \end{array}\right) = \left(\begin{array}{c|c||c} \{L\backslash U\}_{00}^{(k)} & U_{01}^{(k)} & A_{02}^{(k)} \\ L_{10}^{(k)} & \{L\backslash U\}_{11}^{(k)} & A_{12}^{(k)} \\ \hline A_{20}^{(k)} & A_{21}^{(k)} & A_{22}^{(k)} \end{array}\right)$$

Resulting Lazy Algorithm

Partition $A = \left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right)$
where A_{TL} is 0×0
do until A_{BR} is 0×0

Repartition
$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c||c|c} A_{00} & A_{01} & A_{02} \\ \hline\hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$
where A_{11} is $b \times b$

View A_{00} as $\{L\backslash U\}_{00}$
$A_{01} \leftarrow L_{01} = L_{00}^{-1} A_{01}$
$A_{10} \leftarrow L_{10} = A_{10} U_{00}^{-1}$
$A_{11} \leftarrow \{L\backslash U\}_{11} = LU(A_{11} - L_{10} U_{01})$

Continue with
$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c|c||c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline\hline A_{20} & A_{21} & A_{22} \end{array}\right)$$
enddo

Row-lazy Algorithm

Partition $A = \left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right)$
where A_{TL} is 0×0
do until A_{BR} is 0×0

Repartition

$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c||c|c} A_{00} & A_{01} & A_{02} \\ \hline\hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

where A_{11} is $b \times b$

View A_{00} as $\{L\backslash U\}_{00}$
$A_{10} \leftarrow L_{10} = A_{10} U_{00}^{-1}$
$A_{11} \leftarrow \{L\backslash U\}_{11} = LU(A_{11} - L_{10} U_{01})$
$A_{12} \leftarrow U_{12} = L_{11}^{-1}(A_{12} - L_{10} U_{02})$

Continue with

$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c|c||c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline\hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

enddo

Column-lazy Algorithm

Partition $A = \left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right)$

where A_{TL} is 0×0

do until A_{BR} is 0×0

Repartition

$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c||c|c} A_{00} & A_{01} & A_{02} \\ \hline\hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

where A_{11} is $b \times b$

View A_{00} as $\{L \backslash U\}_{00}$

$A_{01} \leftarrow U_{01} = U_{00}^{-1} A_{01}$

$A_{11} \leftarrow \{L \backslash U\}_{11} = LU(A_{11} - L_{10} U_{01})$

$A_{21} \leftarrow L_{21} = (A_{21} - L_{20} U_{01}) U_{11}^{-1}$

Continue with

$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c|c||c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline\hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

enddo

Row-column-lazy Algorithm

Partition $A = \left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right)$

where A_{TL} is 0×0

do until A_{BR} is 0×0

Repartition

$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c||c|c} A_{00} & A_{01} & A_{02} \\ \hline\hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

where A_{11} is $b \times b$

$A_{11} \leftarrow \{L \backslash U\}_{11} = LU(A_{11} - L_{10}U_{01})$

$A_{12} \leftarrow U_{12} = L_{11}^{-1}(A_{12} - L_{10}U_{02})$

$A_{21} \leftarrow L_{21} = (A_{21} - L_{20}U_{01})U_{11}^{-1}$

Continue with

$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c|c||c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline\hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

enddo

Eager Algorithm

Partition $A = \left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right)$

where A_{TL} is 0×0

do until A_{BR} is 0×0

Repartition

$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c||c|c} A_{00} & A_{01} & A_{02} \\ \hline\hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

where A_{11} is $b \times b$

$A_{11} \leftarrow \{L \backslash U\}_{11} = LU(A_{11})$
$A_{12} \leftarrow U_{12} = L_{11}^{-1} A_{12}$
$A_{21} \leftarrow L_{21} = A_{21} U_{11}^{-1}$
$A_{22} \leftarrow A_{22} - L_{21} U_{12}$

Continue with

$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array}\right) = \left(\begin{array}{c|c||c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline\hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

enddo

Impact on Implementation of Linear Algebra Algorithms

Capturing the Algorithm in Code

Partition $A = \left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array} \right)$

where A_{TL} is 0×0

do until A_{BR} is 0×0

Repartition

$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c||c|c} A_{00} & A_{01} & A_{02} \\ \hline\hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array} \right)$$

where A_{11} is $b \times b$

Continue with

$$\left(\begin{array}{c||c} A_{TL} & A_{TR} \\ \hline\hline A_{BL} & A_{BR} \end{array} \right) = \left(\begin{array}{c|c||c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline\hline A_{20} & A_{21} & A_{22} \end{array} \right)$$

enddo

C code

```
TX_Obj_partition_2x2( A,  &ATL, /**/ &ATR,
                        /* ************** */
                          &ABL, /**/ &ABR, /* with */ 0, /* by */ 0, /* submatrix */ TX_QUADRANT_TL );

while ( b = min( TX_Obj_length( ABR ), nb_alg ) ){

  TX_Obj_repartition_2x2_to_3x3( ATL, /**/ ATR,         &A00, /**/ &A01, &A02,
                              /* ************* */    /* ******************** */
                                      /**/              &A10, /**/ &A11, &A12,
                                 ABL, /**/ ABR,         &A20, /**/ &A21, &A22,
               /* with */ b, /* by */ b, /* A11 split from */ TX_QUADRANT_BR );

  /* **************************************************************************************************** */









  /* **************************************************************************************************** */

  TX_Obj_continue_with_3x3_to_2x2( &ATL, /**/ &ATR,         A00, A01, /**/ A02,
                                         /**/               A10, A11, /**/ A12,
                                 /* ************** */    /* ***************** */
                                   &ABL, /**/ &ABR,         A20, A21, /**/ A22,
               /* with A11 added to submatrix */ TX_QUADRANT_TL );

}
```

Eager LU in C

```
TX_Obj_partition_2x2( A,  &ATL, /**/ &ATR,
                        /* ************** */
                          &ABL, /**/ &ABR, /* with */ 0, /* by */ 0, /* submatrix */ TX_QUADRANT_TL );

while ( b = min( TX_Obj_length( ABR ), nb_alg ) ){

  TX_Obj_repartition_2x2_to_3x3( ATL, /**/ ATR,         &A00, /**/ &A01, &A02,
                              /* ************* */    /* ******************** */
                                      /**/              &A10, /**/ &A11, &A12,
                                 ABL, /**/ ABR,         &A20, /**/ &A21, &A22,
               /* with */ b, /* by */ b, /* A11 split from */ TX_QUADRANT_BR );

  /* ********************************************************************************************** */

  TX_LU_nopivot_level2( A11 );

  TX_Trsm( TX_SIDE_RIGHT, TX_UPPER_TRIANGULAR, TX_NO_TRANSPOSE, TX_NONUNIT_DIAG, ONE, A11, A21 );
  TX_Trsm( TX_SIDE_LEFT,  TX_LOWER_TRIANGULAR, TX_NO_TRANSPOSE, TX_UNIT_DIAG,    ONE, A11, A12 );
  TX_Gemm( TX_NO_TRANSPOSE, TX_NO_TRANSPOSE, MINUS_ONE, A21, A12, ONE, A22 );

  /* ********************************************************************************************** */

  TX_Obj_continue_with_3x3_to_2x2( &ATL, /**/ &ATR,         A00, A01, /**/ A02,
                                         /**/               A10, A11, /**/ A12,
                                  /* ************** */    /* ***************** */
                                    &ABL, /**/ &ABR,        A20, A21, /**/ A22,
               /* with A11 added to submatrix */ TX_QUADRANT_TL );

}
```

Lazy LU in C

```
TX_Obj_partition_2x2( A,  &ATL, /**/ &ATR,
                        /* ************** */
                          &ABL, /**/ &ABR, /* with */ 0, /* by */ 0, /* submatrix */ TX_QUADRANT_TL );

while ( b = min( TX_Obj_length( ABR ), nb_alg ) ){

  TX_Obj_repartition_2x2_to_3x3( ATL, /**/ ATR,         &A00, /**/ &A01, &A02,
                              /* ************* */    /* ********************* */
                                      /**/              &A10, /**/ &A11, &A12,
                                 ABL, /**/ ABR,         &A20, /**/ &A21, &A22,
               /* with */ b, /* by */ b, /* A11 split from */ TX_QUADRANT_BR );

  /* ************************************************************************************************** */

  TX_Trsm( TX_SIDE_RIGHT, TX_UPPER_TRIANGULAR, TX_NO_TRANSPOSE, TX_NONUNIT_DIAG, ONE, A00, A10 );
  TX_Trsm( TX_SIDE_LEFT,  TX_LOWER_TRIANGULAR, TX_NO_TRANSPOSE, TX_UNIT_DIAG,    ONE, A00, A01 );
  TX_Gemm( TX_NO_TRANSPOSE, TX_NO_TRANSPOSE, MINUS_ONE, A10, A01, ONE, A11 );

  TX_LU_nopivot_level2( A11 );

  /* ************************************************************************************************** */

  TX_Obj_continue_with_3x3_to_2x2( &ATL, /**/ &ATR,         A00, A01, /**/ A02,
                                         /**/               A10, A11, /**/ A12,
                                 /* ************** */    /* ****************** */
                                   &ABL, /**/ &ABR,         A20, A21, /**/ A22,
               /* with A11 added to submatrix */ TX_QUADRANT_TL );

}
```

Column-lazy LU in C

```
TX_Obj_partition_2x2( A,  &ATL, /**/ &ATR,
                        /* ************** */
                          &ABL, /**/ &ABR, /* with */ 0, /* by */ 0, /* submatrix */ TX_QUADRANT_TL );

while ( b = min( TX_Obj_length( ABR ), nb_alg ) ){

  TX_Obj_repartition_2x2_to_3x3( ATL, /**/ ATR,         &A00, /**/ &A01, &A02,
                              /* ************* */    /* ******************** */
                                      /**/              &A10, /**/ &A11, &A12,
                                 ABL, /**/ ABR,         &A20, /**/ &A21, &A22,
               /* with */ b, /* by */ b, /* A11 split from */ TX_QUADRANT_BR );

  /* **************************************************************************************************** */

  TX_Trsm( TX_SIDE_LEFT,  TX_LOWER_TRIANGULAR, TX_NO_TRANSPOSE, TX_UNIT_DIAG,    ONE, A00, A01 );
  TX_Gemm( TX_NO_TRANSPOSE, TX_NO_TRANSPOSE, MINUS_ONE, A10, A10, ONE, A11 );
  TX_LU_nopivot_level2( A11 );
  TX_Gemm( TX_NO_TRANSPOSE, TX_NO_TRANSPOSE, MINUS_ONE, A20, A10, ONE, A21 );
  TX_Trsm( TX_SIDE_RIGHT, TX_UPPER_TRIANGULAR, TX_NO_TRANSPOSE, TX_NONUNIT_DIAG, ONE, A11, A21 );

  /* **************************************************************************************************** */

  TX_Obj_continue_with_3x3_to_2x2( &ATL, /**/ &ATR,         A00, A01, /**/ A02,
                                         /**/               A10, A11, /**/ A12,
                                 /* ************** */    /* ***************** */
                                   &ABL, /**/ &ABR,         A20, A21, /**/ A22,
               /* with A11 added to submatrix */ TX_QUADRANT_TL );

}
```

Row-lazy LU in C

```
TX_Obj_partition_2x2( A,  &ATL, /**/ &ATR,
                        /* ************** */
                          &ABL, /**/ &ABR, /* with */ 0, /* by */ 0, /* submatrix */ TX_QUADRANT_TL );

while ( b = min( TX_Obj_length( ABR ), nb_alg ) ){

  TX_Obj_repartition_2x2_to_3x3( ATL, /**/ ATR,         &A00, /**/ &A01, &A02,
                              /* ************* */    /* ******************** */
                                      /**/              &A10, /**/ &A11, &A12,
                                 ABL, /**/ ABR,         &A20, /**/ &A21, &A22,
               /* with */ b, /* by */ b, /* A11 split from */ TX_QUADRANT_BR );

  /* ************************************************************************************************** */

  TX_Trsm( TX_SIDE_RIGHT, TX_UPPER_TRIANGULAR, TX_NO_TRANSPOSE, TX_NONUNIT_DIAG, ONE, A00, A10 );
  TX_Gemm( TX_NO_TRANSPOSE, TX_NO_TRANSPOSE, MINUS_ONE, A10, A10, ONE, A11 );
  TX_LU_nopivot_level2( A11 );
  TX_Gemm( TX_NO_TRANSPOSE, TX_NO_TRANSPOSE, MINUS_ONE, A10, A02, ONE, A12 );
  TX_Trsm( TX_SIDE_LEFT,  TX_LOWER_TRIANGULAR, TX_NO_TRANSPOSE, TX_UNIT_DIAG,    ONE, A11, A12 );

  /* ************************************************************************************************** */

  TX_Obj_continue_with_3x3_to_2x2( &ATL, /**/ &ATR,         A00, A01, /**/ A02,
                                         /**/               A10, A11, /**/ A12,
                                 /* ************** */    /* ***************** */
                                   &ABL, /**/ &ABR,         A20, A21, /**/ A22,
               /* with A11 added to submatrix */ TX_QUADRANT_TL );

}
```

Row-column lazy LU in C

```
TX_Obj_partition_2x2( A,  &ATL, /**/ &ATR,
                        /* ************** */
                          &ABL, /**/ &ABR, /* with */ 0, /* by */ 0, /* submatrix */ TX_QUADRANT_TL );

while ( b = min( TX_Obj_length( ABR ), nb_alg ) ){

  TX_Obj_repartition_2x2_to_3x3( ATL, /**/ ATR,         &A00, /**/ &A01, &A02,
                              /* ************* */    /* ******************** */
                                      /**/              &A10, /**/ &A11, &A12,
                                 ABL, /**/ ABR,         &A20, /**/ &A21, &A22,
               /* with */ b, /* by */ b, /* A11 split from */ TX_QUADRANT_BR );

  /* ***************************************************************************************************** */
  TX_Gemm( TX_NO_TRANSPOSE, TX_NO_TRANSPOSE, MINUS_ONE, A10, A01, ONE, A11 );
  TX_Gemm( TX_NO_TRANSPOSE, TX_NO_TRANSPOSE, MINUS_ONE, A20, A01, ONE, A21 );
  TX_Gemm( TX_NO_TRANSPOSE, TX_NO_TRANSPOSE, MINUS_ONE, A10, A02, ONE, A12 );
  TX_LU_nopivot_level2( A11 );
  TX_Trsm( TX_SIDE_RIGHT, TX_UPPER_TRIANGULAR, TX_NO_TRANSPOSE, TX_NONUNIT_DIAG, ONE, A11, A21 );
  TX_Trsm( TX_SIDE_LEFT,  TX_LOWER_TRIANGULAR, TX_NO_TRANSPOSE, TX_UNIT_DIAG,    ONE, A11, A12 );

  /* ***************************************************************************************************** */

  TX_Obj_continue_with_3x3_to_2x2( &ATL, /**/ &ATR,         A00, A01, /**/ A02,
                                         /**/               A10, A11, /**/ A12,
                                 /* ************** */    /* ***************** */
                                   &ABL, /**/ &ABR,         A20, A21, /**/ A22,
               /* with A11 added to submatrix */ TX_QUADRANT_TL );

}
```

But what about pivoting?

Yes, pivoting can be handled equally elegantly.

Parallel Eager LU in C

```
PLA_Obj_partition_2x2( A,  &ATL, /**/ &ATR,
                          /* ************** */
                            &ABL, /**/ &ABR, /* with */ 0, /* by */ 0, /* submatrix */ PLA_QUADRANT_TL );

while ( b = min( PLA_Obj_length( ABR ), nb_alg ) ){

  PLA_Obj_repartition_2x2_to_3x3( ATL, /**/ ATR,         &A00, /**/ &A01, &A02,
                               /* ************* */    /* ******************** */
                                       /**/              &A10, /**/ &A11, &A12,
                                  ABL, /**/ ABR,         &A20, /**/ &A21, &A22,
                /* with */ b, /* by */ b, /* A11 split from */ PLA_QUADRANT_BR );

  /* ************************************************************************************************ */

  PLA_LU_nopivot_level2( A11 );

  PLA_Trsm( PLA_SIDE_RIGHT, PLA_UPPER_TRIANGULAR, PLA_NO_TRANSPOSE, PLA_NONUNIT_DIAG, ONE, A11, A21 );
  PLA_Trsm( PLA_SIDE_LEFT,  PLA_LOWER_TRIANGULAR, PLA_NO_TRANSPOSE, PLA_UNIT_DIAG,    ONE, A11, A12 );
  PLA_Gemm( PLA_NO_TRANSPOSE, PLA_NO_TRANSPOSE, MINUS_ONE, A21, A12, ONE, A22 );

  /* ************************************************************************************************ */

  PLA_Obj_continue_with_3x3_to_2x2( &ATL, /**/ &ATR,         A00, A01, /**/ A02,
                                          /**/               A10, A11, /**/ A12,
                                  /* ************** */    /* ***************** */
                                    &ABL, /**/ &ABR,         A20, A21, /**/ A22,
                /* with A11 added to submatrix */ PLA_QUADRANT_TL );

}
```

Parallel Out-of-Core Eager LU in C

```
POOCLA_Obj_partition_2x2( A,  &ATL, /**/ &ATR,
                            /* ************** */
                              &ABL, /**/ &ABR, /* with */ 0, /* by */ 0, /* submatrix */ PLA_QUADRANT_TL );

while ( b = min( PLA_Obj_length( ABR ), nb_alg ) ){

  POOCLA_Obj_repartition_2x2_to_3x3( ATL, /**/ ATR,         &A00, /**/ &A01, &A02,
                                  /* ************* */    /* ******************** */
                                          /**/              &A10, /**/ &A11, &A12,
                                   ABL, /**/ ABR,           &A20, /**/ &A21, &A22,
                /* with */ b, /* by */ b, /* A11 split from */ PLA_QUADRANT_BR );

  /* ****************************************************************************************************** */

  POOCLA_LU_nopivot_level2( A11 );

  POOCLA_Trsm( PLA_SIDE_RIGHT, PLA_UPPER_TRIANGULAR, PLA_NO_TRANSPOSE, PLA_NONUNIT_DIAG, ONE, A11, A21 );
  POOCLA_Trsm( PLA_SIDE_LEFT,  PLA_LOWER_TRIANGULAR, PLA_NO_TRANSPOSE, PLA_UNIT_DIAG,    ONE, A11, A12 );
  POOCLA_Gemm( PLA_NO_TRANSPOSE, PLA_NO_TRANSPOSE, MINUS_ONE, A21, A12, ONE, A22 );

  /* ****************************************************************************************************** */

  POOCLA_Obj_continue_with_3x3_to_2x2( &ATL, /**/ &ATR,         A00, A01, /**/ A02,
                                             /**/               A10, A11, /**/ A12,
                                     /* ************** */    /* ***************** */
                                       &ABL, /**/ &ABR,         A20, A21, /**/ A22,
                /* with A11 added to submatrix */ PLA_QUADRANT_TL );

}
```

Impact on Library Development and Maintainability

Experience 1: Classification of Algorithms

- With John Gunnels
- Problem: Traditional classifications of algorithms are inconsistent and vague.
  - ⋆ Right-looking algorithm for LU factorization does not resemble a right-looking algorithm for UL factorization
- Solution: Our formal approach leads to a natural classification of algorithms as being lazy or eager.

Experience 2: Solving the Lyapunov Equation

- With Enrique Quintana (UJI, Spain).
- Problem: Update $A \leftarrow U^{-1}AU^{-T}$ where A is symmetric and U is uppertriangular.
- Requirements:
  - ⋆ Derive all possible variants for this operation
  - ⋆ Take advantage of symmetry in A
  - ⋆ Take advantage of upper triangular structure of U
  - ⋆ Perform an in-place update of A
  - ⋆ Peak performance should be within 10% of peak of matrix-matrix multiplication
- Solution: an algorithm that we claim would not have been discovered using conventional derivation

Experience 3: Implementation of level-3 BLAS.

- With John Gunnels and Greg Henry (Intel).
- Problem: Implement

$C \leftarrow \alpha AB + \beta C$ $C \leftarrow \alpha AB^T + \beta C$	$C \leftarrow \alpha A^T B + \beta C$ $C \leftarrow \alpha A^T B^T + \beta C$
$C \leftarrow \alpha (L + \hat{L}^T) B + \beta C$ $C \leftarrow \alpha B (L + \hat{L}^T) + \beta C$	$C \leftarrow \alpha (U + \hat{U}^T) B + \beta C$ $C \leftarrow \alpha B (U + \hat{U}^T) + \beta C$
$lo(C) \leftarrow \alpha \mathrm{lo}(AA^T) + \beta \mathrm{lo}(C)$ $lo(C) \leftarrow \alpha \mathrm{lo}(A^T A) + \beta \mathrm{lo}(C)$	$up(C) \leftarrow \alpha \mathrm{up}(AA^T) + \beta \mathrm{up}(C)$ $up(C) \leftarrow \alpha \mathrm{up}(A^T A) + \beta \mathrm{up}(C)$
$lo(C) \leftarrow \alpha \mathrm{lo}(AB^T + BA^T) + \beta \mathrm{lo}(C)$ $lo(C) \leftarrow \alpha \mathrm{lo}(A^T B + B^T A) + \beta \mathrm{lo}(C)$	$up(C) \leftarrow \alpha \mathrm{up}(AB^T + BA^T) + \beta \mathrm{up}(C)$ $up(C) \leftarrow \alpha \mathrm{up}(A^T B + B^T A) + \beta \mathrm{up}(C)$
$B \leftarrow \alpha LB$ $B \leftarrow \alpha UB$ $B \leftarrow \alpha BL$ $B \leftarrow \alpha BU$	$B \leftarrow \alpha L^T B$ $B \leftarrow \alpha U^T B$ $B \leftarrow \alpha BL^T$ $B \leftarrow \alpha BU^T$
$B \leftarrow \alpha L^{-1} B$ $B \leftarrow \alpha U^{-1} B$ $B \leftarrow \alpha BL^{-1}$ $B \leftarrow \alpha BU^{-1}$	$B \leftarrow \alpha L^{-T} B$ $B \leftarrow \alpha U^{-T} B$ $B \leftarrow \alpha BL^{-T}$ $B \leftarrow \alpha BU^{-T}$

- Requirements: Deal explicitly with L1 cache, L2 cache, and main memory.
- Solution: Time for implementation around 10 hours including testing.

Experience 4: Parallel Linear Algebra Package (PLAPACK)

- With Philip Alpatov, Greg Baker, Carter Edwards, John Gunnels, Brian Gunter, Greg Morrow, Peter Nagel, Enrique Quintana, Beatrice Riviera, Jason Wu, ..., RvdG
- Problem: Implement Parallel (Dense) Linear Algebra Library
- Requirements:
  - ⋆ Functionality similar to ScaLAPACK.
  - ⋆ Performance similar to ScaLAPACK
  - ⋆ Sensible user interface.
- Solution:
  - ⋆ Complete rewrite of the library during 1999.
  - ⋆ About 40,000 lines of code.
  - ⋆ Performance exceeds ScaLAPACK.

Experience 5: One-Sweep Matrix Inversion Algorithm

- With Enrique Quintana (UJI, Spain), Gregorio Quintana (UJI, Spain) and Xiaobai Sun (Duke).
- Problem: Parallel Gauss-Jordan based blocked matrix inversion
- Requirements:
  - ⋆ One-sweep approach to improve load-balance and reduce communication.
  - ⋆ Stability similar to traditional three-sweep method
- Solution:
  - ⋆ PLAPACK implementation
  - ⋆ Stability established by showing one-sweep algorithm is equivalent to four-sweep algorithm with known stability properties.
  - ⋆ Convincing enough for referees for SJSC.

Experience 6: Out-of-Core implementation of Matrix Factorization

- With Wesley Reiley and Brian Gunter.
- Problem: Parallel Out-of-Core implementation of matrix factorization algorithms
- Requirements: High performance
- Solution:
  - ⋆ Parallel OOC Cholesky factorization implemented by bright undergraduate as an honors project (360 MFLOPS per processor on Cray T3E-600).
  - ⋆ New scalable tiled parallel OOC QR factorization

Experience 7: Fault Tolerant BLAS

- With John Gunnels, Dan Katz (JPL) and Enrique Quintana (UJI, Spain)
- Problem: Add algorithmic fault tolerance to BLAS.
- Requirements:
  - ⋆ Transparent to user.
  - ⋆ Expected rate of corruption: no more than one bit every 5 seconds
  - ⋆ Performance degradation of no more than 10-15%
- Solution:
  - ⋆ Minor modifications to TXBLAS
  - ⋆ Transparent to user.
  - ⋆ Performance degradation of no more than 15%
  - ⋆ Can detect and correct at a rate of one corruption every millisecond.

Experience 8: Broadway Compiler

- Sam Guyer and Calvin Lin
- Problem: Encode van de Geijn's brain
- Requirements: Automatically optimizations should be at least as fast as those handcoded by yours truely
- Solution: See Calvin's talk yesterday

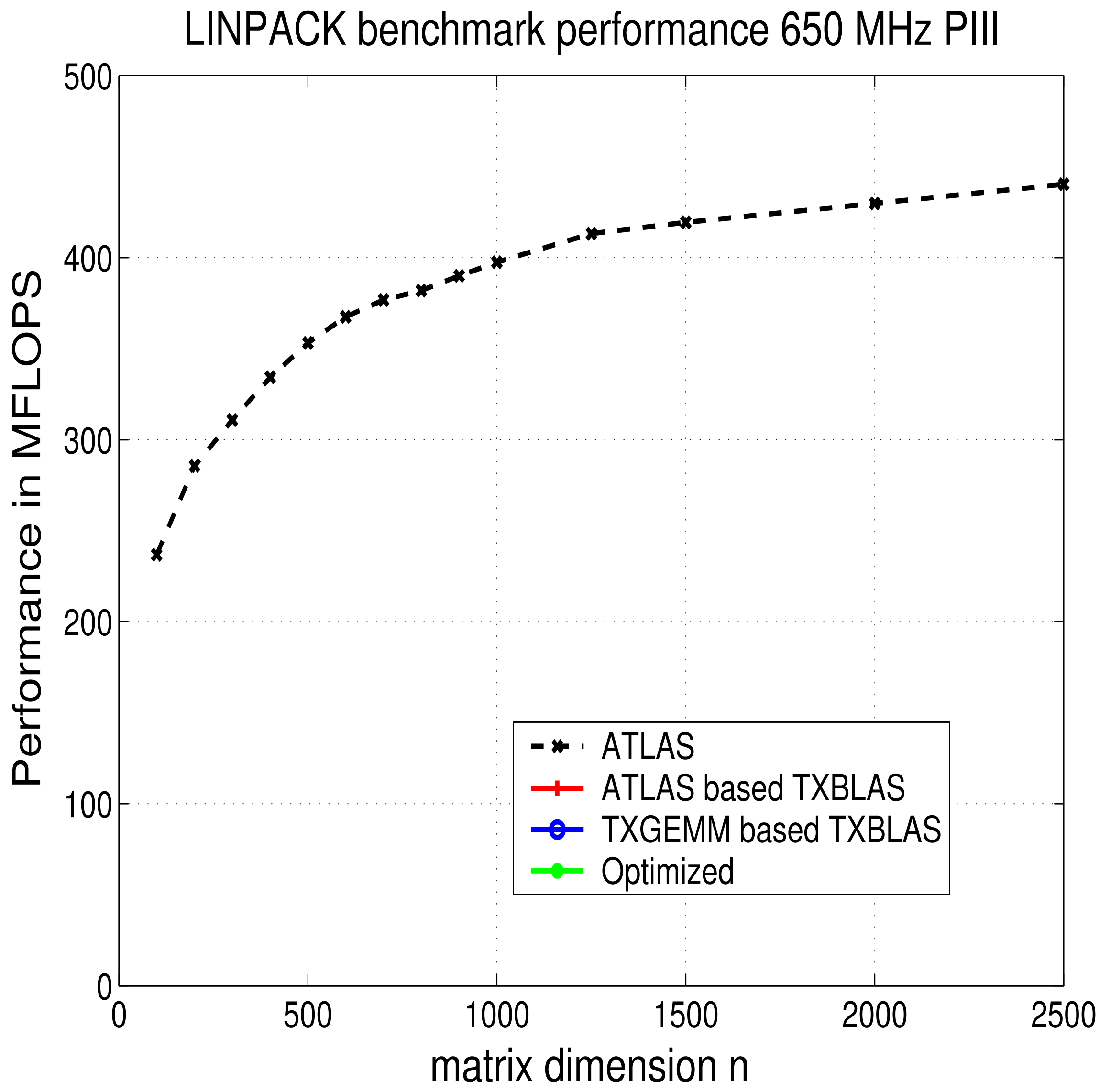
Experience 9: Reimplementation of LAPACK

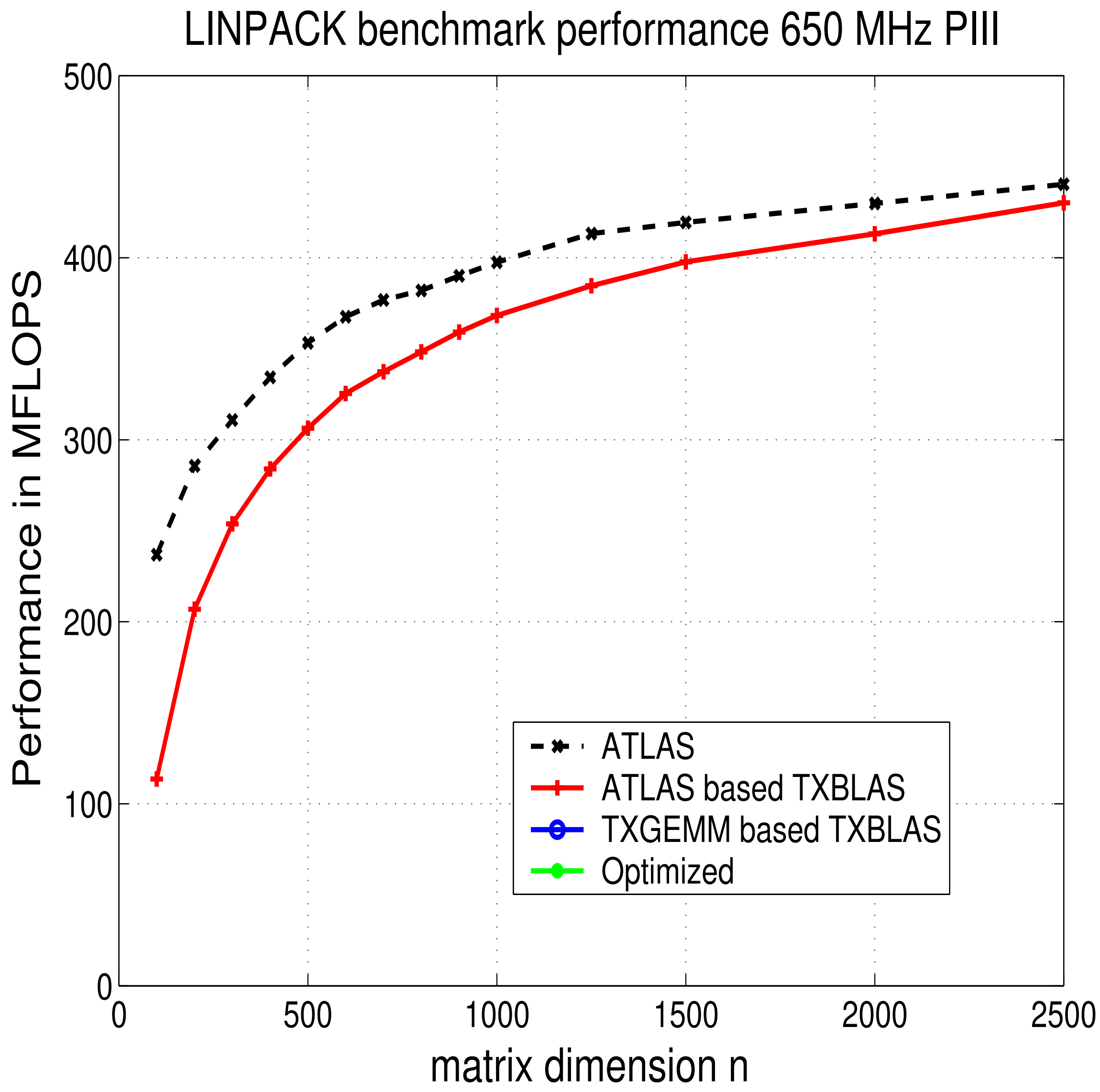
- Problem: Make LAPACK more readable
- Requirements: Must be simple enough to be done by first year graduate students
- Solution: Class project for introductory graduate numerical linear algebra class.

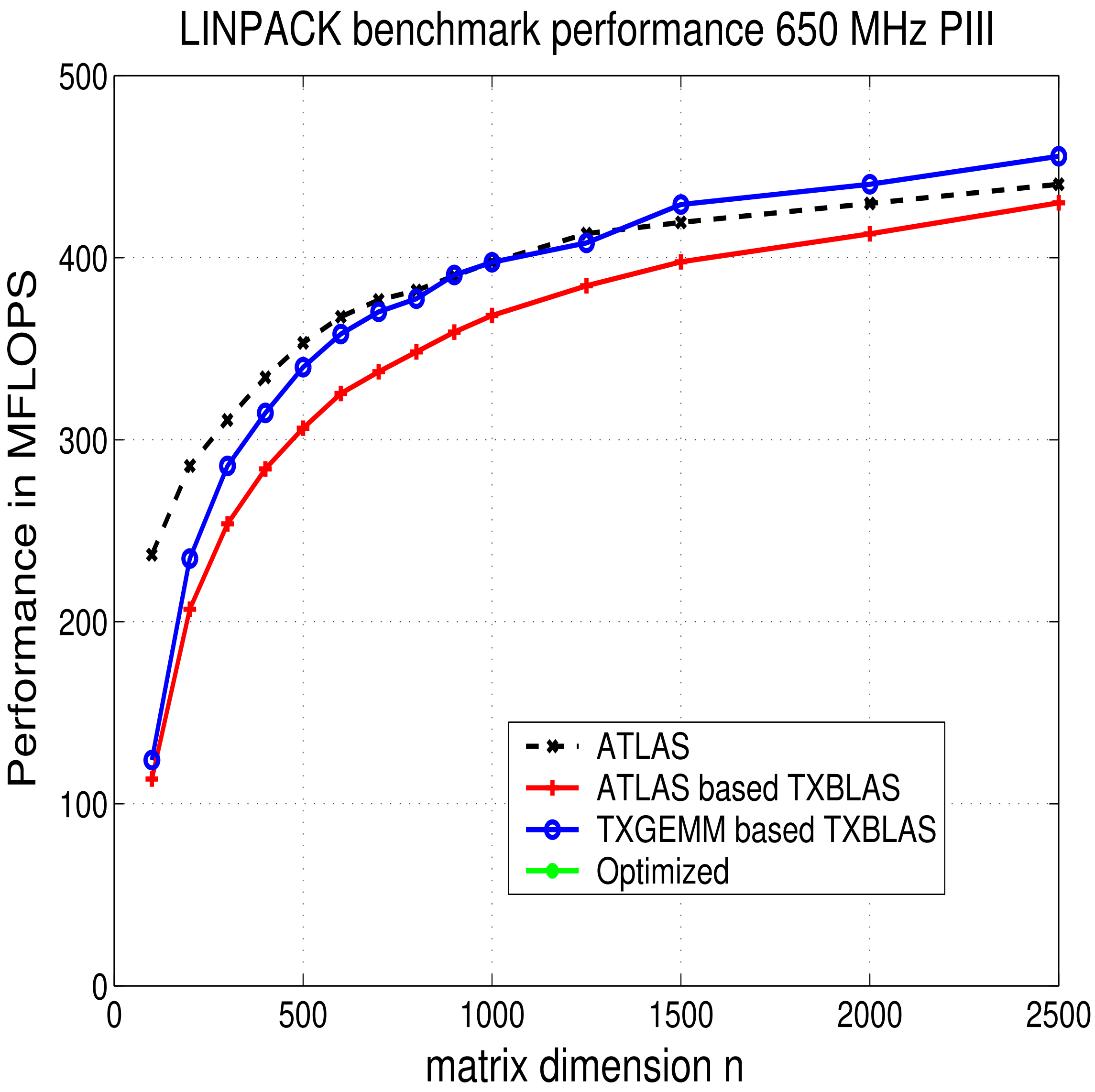
$\boxed{\text{Impact on Performance}}$

LINPACK Benchmark (unofficial)

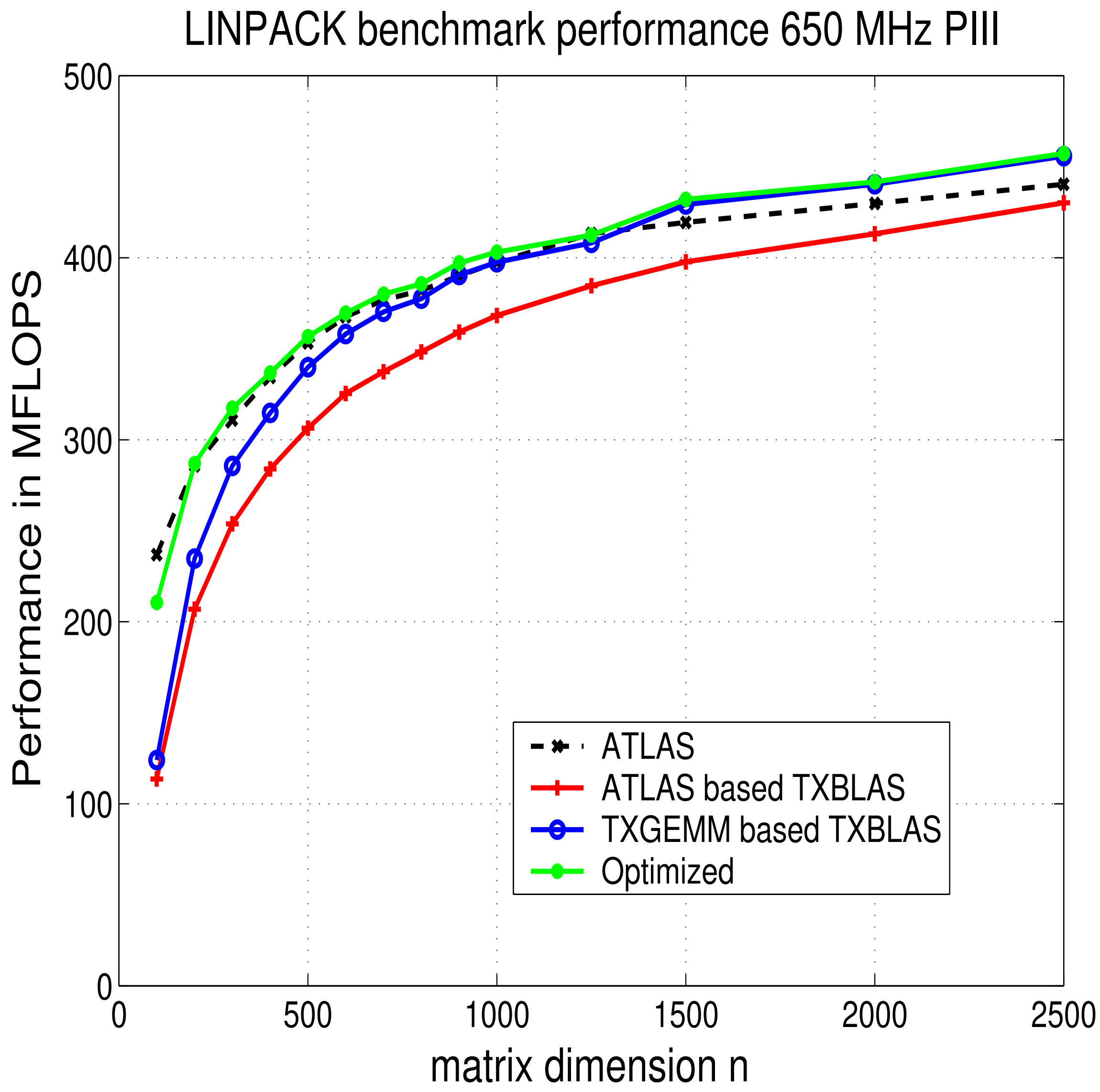
- LU factorization with pivoting followed by forward and backward substitution.
- Double precision real
- Pentium III (650 MHz)
- Linux (Redhat 6.2)
- Reference performance: LU from ATLAS R3.1
- Peak achieved for matrix-matrix multiply:
  - ⋆ Around 490 MFLOPS for ATLAS matrix-matrix multiply


LINPACK benchmark performance 650 MHz PIII
Performance in MFLOPS
matrix dimension n
0
100
200
300
400
500
0
500
1000
1500
2000
2500
ATLAS
ATLAS based TXBLAS
TXGEMM based TXBLAS
Optimized


LINPACK benchmark performance 650 MHz PIII
Performance in MFLOPS
0
100
200
300
400
500
0
500
1000
1500
2000
2500
matrix dimension n
ATLAS
ATLAS based TXBLAS
TXGEMM based TXBLAS
Optimized


LINPACK benchmark performance 650 MHz PIII
Performance in MFLOPS
matrix dimension n
0
100
200
300
400
500
0
500
1000
1500
2000
2500
ATLAS
ATLAS based TXBLAS
TXGEMM based TXBLAS
Optimized

Impact on Performance



Beyond Dense Linear Algebra

Closing the Book on Dense Linear Algebra

- John Gunnels
- Problem: How to best convince people that dense linear algebra is done (at least from a software engineering point of view)
- Requirements:
  - ⋆ Consistent classification of algorithms (done)
  - ⋆ Consistent language for expressing algorithms
  - ⋆ Systematic method for developing all algorithms
    - ◇ Better yet: automated method for developing all algorithms
  - ⋆ Systematic method for deriving a cost estimate for all algorithms
    - ◇ Better yet: automated method for deriving a cost estimate for all
  - ⋆ Heuristic for choosing between algorithms
- Solution: Progress toward this goal in John's dissertation as well as the Broadway compiler work by Sam Guyer and Calvin Lin

Sparse Iterative Methods

- Manipulation of Krylov vectors as part of iterative methods is really a dense linear algebra operation
- Our approach works very nicely in that domain

Sparse Direct Matrix Operations

- Problem: Solution of large sparse linear systems
- Requirements:
  - ⋆ A direct method may be preferred for various reasons
- Solution: By viewing sparse matrices with structure as hierarchical matrices our approach to deriving algorithms applies

$\boxed{\text{Conclusion}}$

Summary

- There is a place for formal methods in scientific computing
- Direct benefits:
  - ⋆ Coverage
  - ⋆ Correctness
  - ⋆ Productivity
- Indirect benefits:
  - ⋆ A consistent classification of linear algebra algorithms
  - ⋆ A consistent template for expressing algorithms
  - ⋆ A consistent template for coding algorithms
  - ⋆ Key to high performance (Surprise!)

More information

- http://www.cs.utexas.edu/users/plapack/
- http://www.cs.utexas.edu/users/plapack/TXBLAS/