DSLs for DLA: Past the BLAS

Robert van de Geijn
Department of Computer Science
Institute for Computation Engineering and Sciences

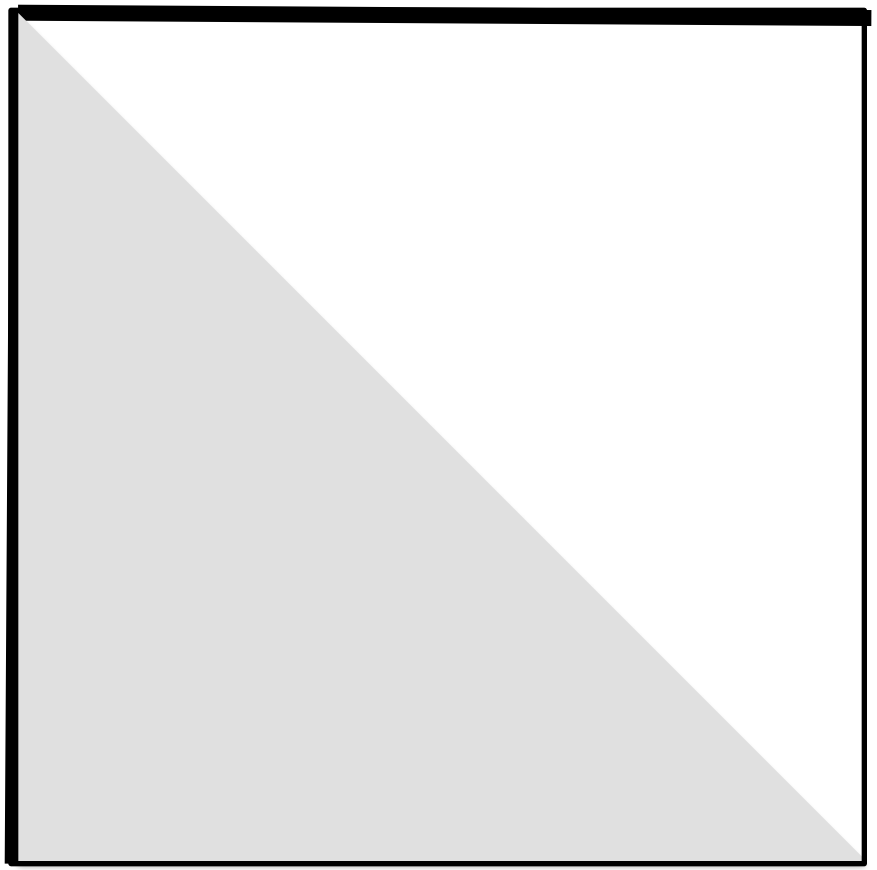
Motivation

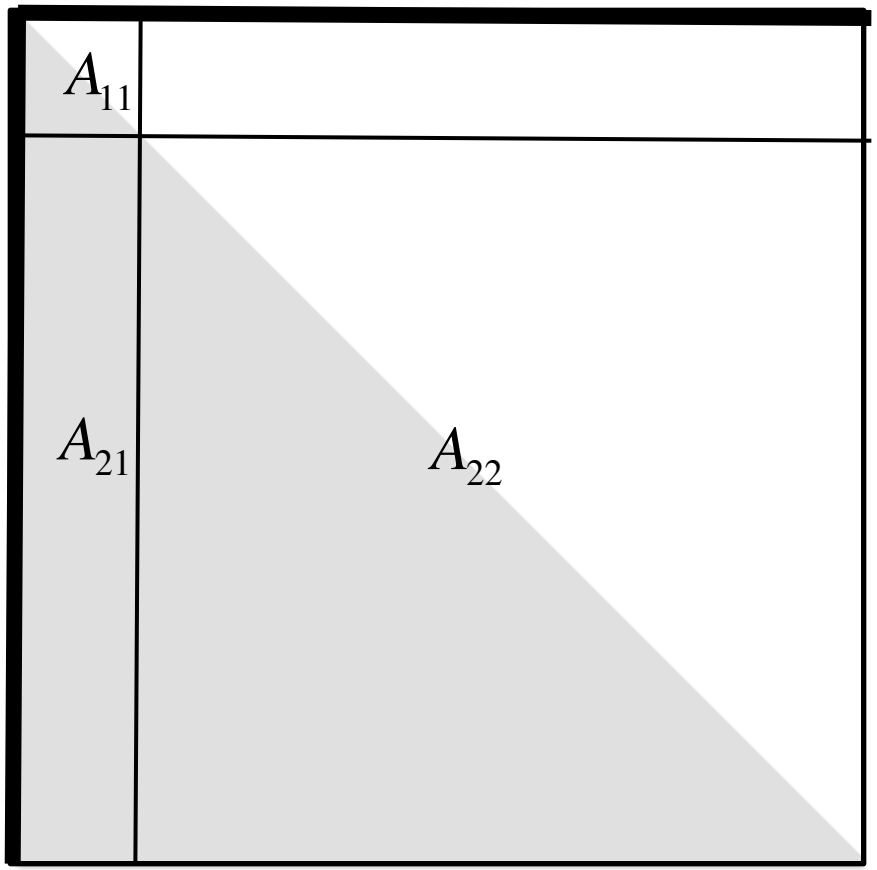
- Throughout we will use Cholesky factorization as a motivating example:
  - Given Symmetric Positive Definite A compute L such that

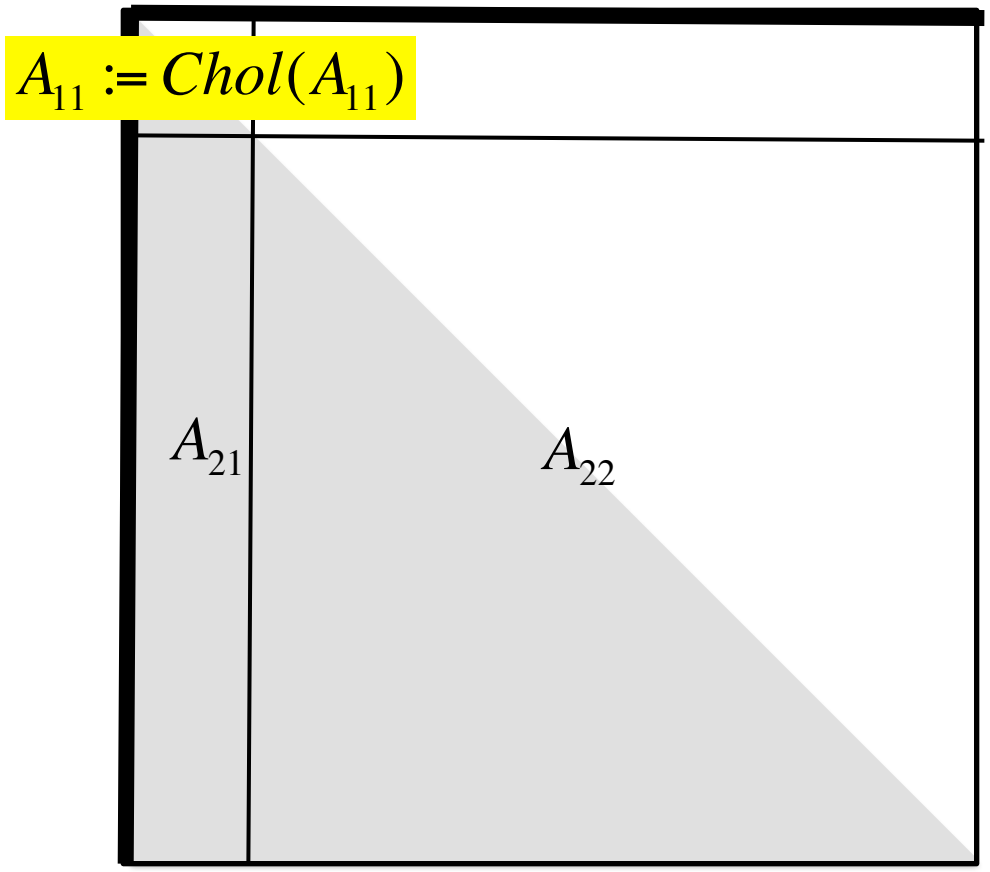
$$A = L\,L^T$$

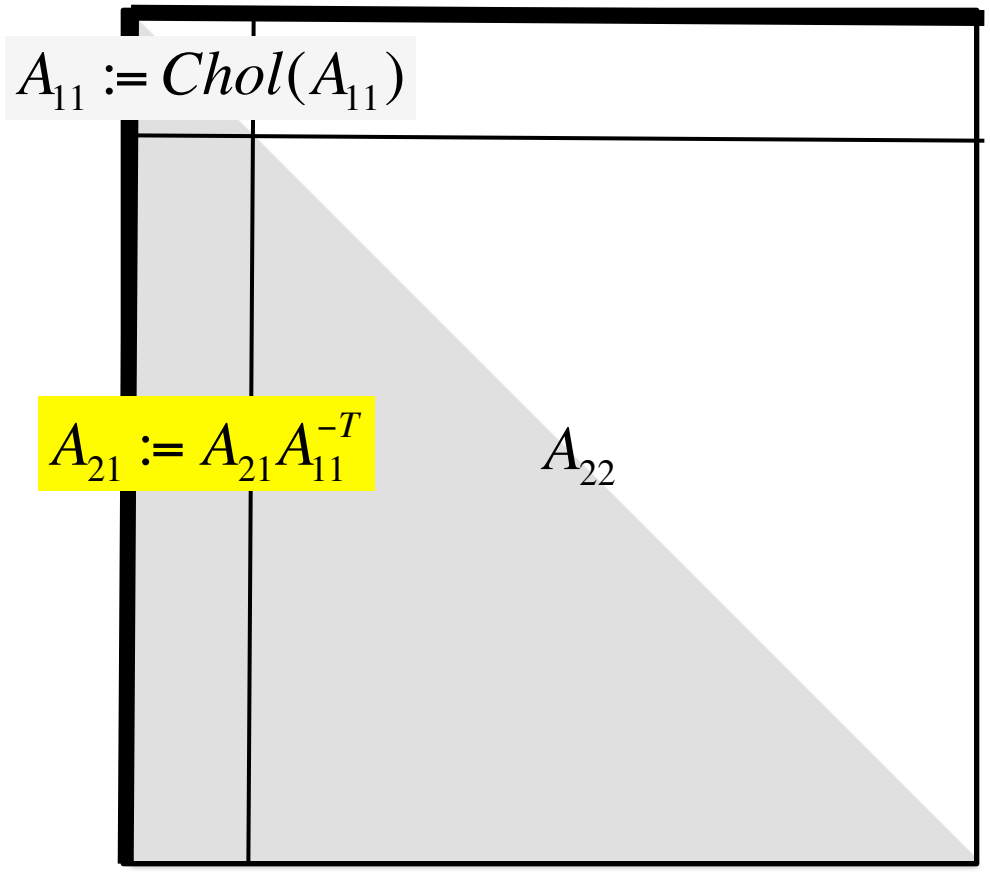
  - L typically overwrites A.

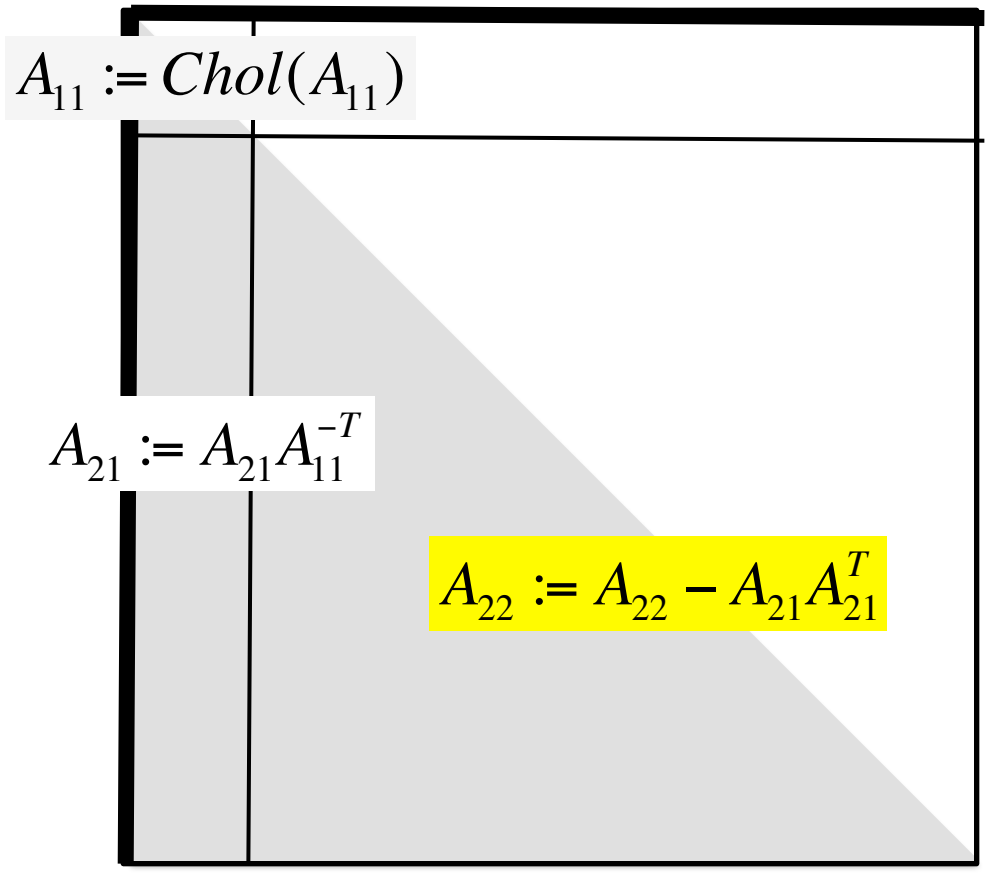
Blocked Right-looking Algorithm

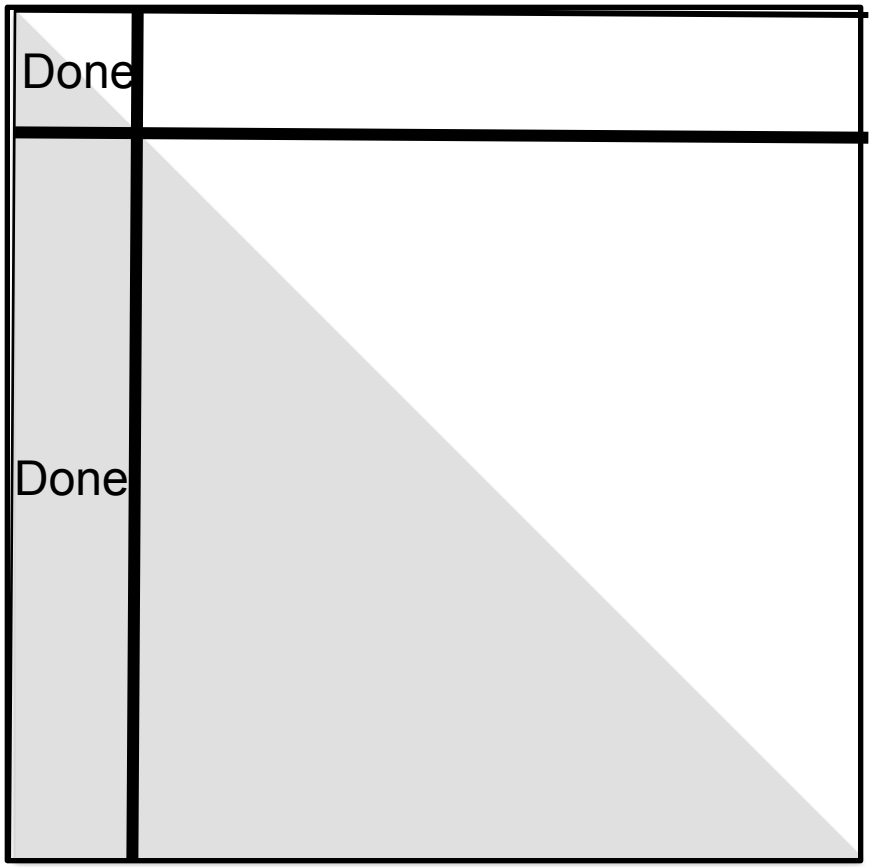



A_{11}
A_{21}
A_{22}


$A_{11} := Chol(A_{11})$
A_{21}
A_{22}


$A_{11} := Chol(A_{11})$
$A_{21} := A_{21}A_{11}^{-T}$
A_{22}


$A_{11} := Chol(A_{11})$
$A_{21} := A_{21}A_{11}^{-T}$
$A_{22} := A_{22} - A_{21}A_{21}^{T}$


Done
Done

A Brief and Incomplete History

State-of-the-art in 1980

- LINPACK:

```
dchdc(a,lda,n,work,jpvt,job,info)
```

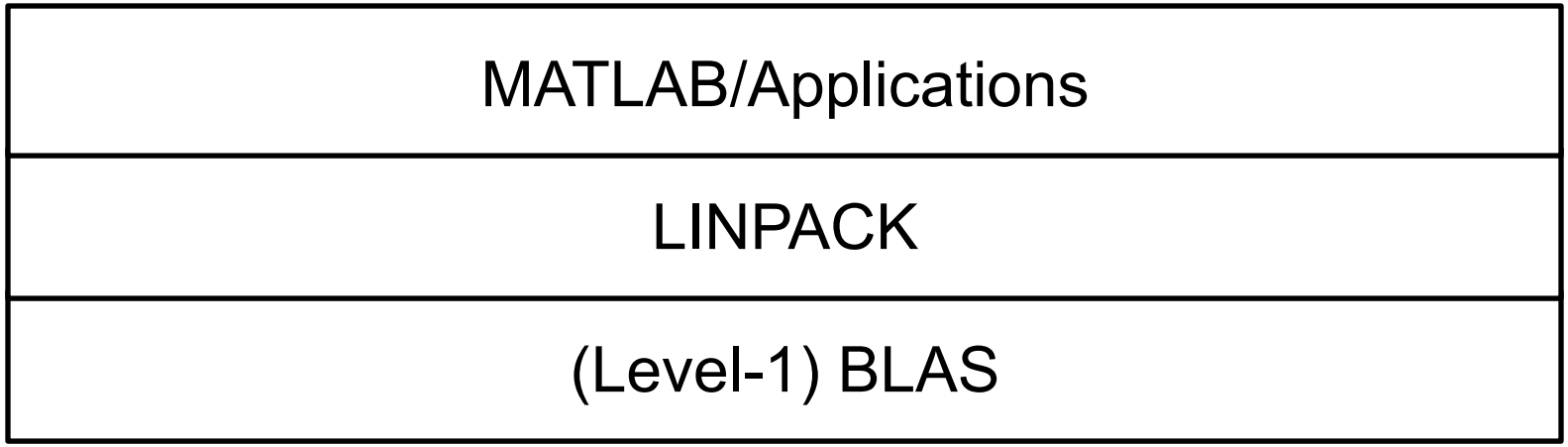
J. J. Dongarra, C. B. Moler, J. R. Bunch, and G. W. Stewart. LINPACK Users' Guide, 1979.

- MATLAB:

```
L = chol( A )
```

Cleve Moler. MATLAB-an interactive matrix laboratory. 1980.

Layering in 1980



(Level-1) BLAS

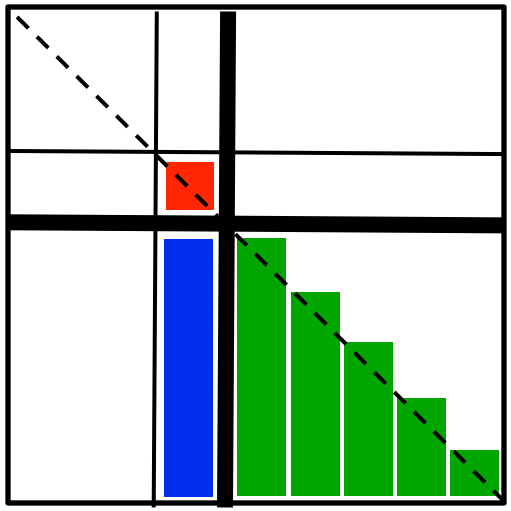
- C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic Linear Algebra Subprograms for Fortran Usage. *ACM TOMS, 1979.*
  - Provide portable high performance
  - Improve readability

```
sscal, dscal, cscal, zscal
scopy, dcopy, ccopy, zcopy
sdot,  ddot,  cdot,  zdot
saxpy, daxpy, caxpy, zaxpy
 ...
```

  - **Early example of a Domain Specific Language**

LINPACK-like Cholesky factorization

```
do j=1, n
  A( j,j ) = sqrt( A( j,j ) )
  call dscal( n-j, 1.0d00 / A( j,j ), A( j+1, j ), 1 )
  do k=j+1,n
    call daxpy( n-k+1, -A( k,j ), A(k,j), 1, A( k, k ), 1 );
  enddo
enddo
```



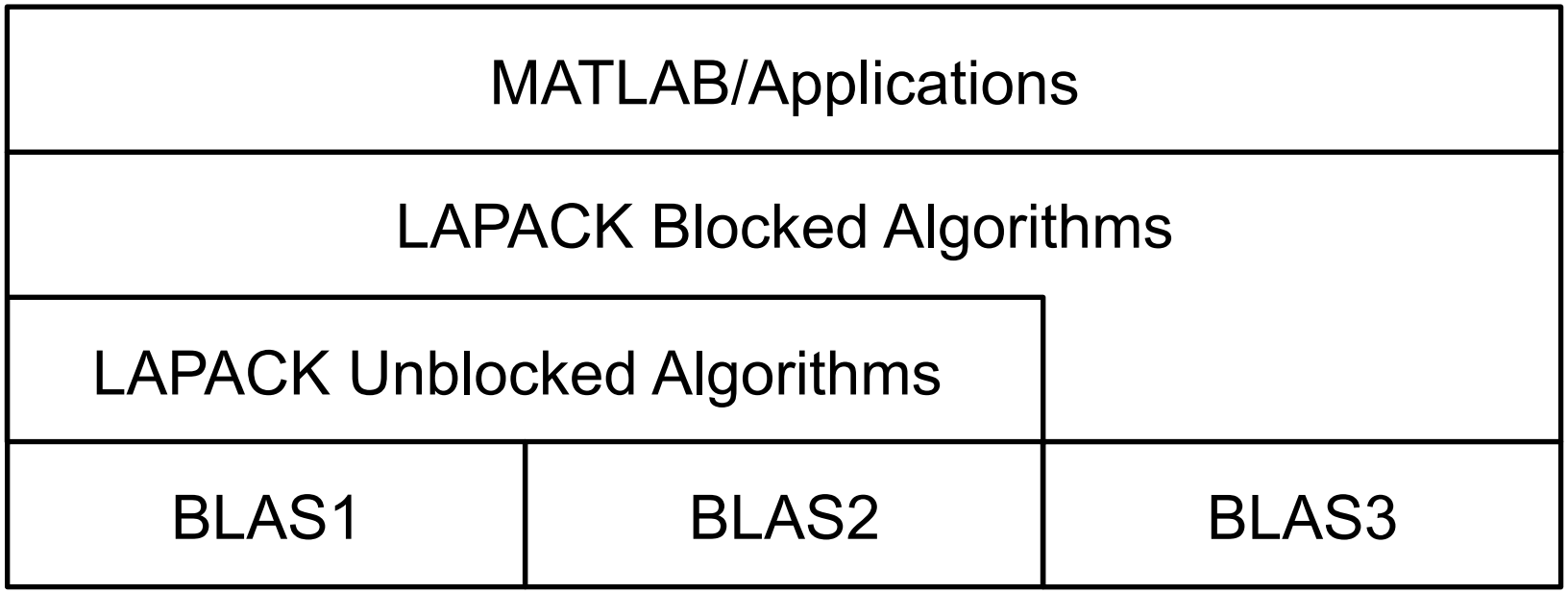
The State-of-the-Art in 1990

- LAPACK:

```
DPOTRF( UPLO, N, A, LDA, INFO )
```

Anderson et al. LAPACK: A portable linear algebra library for high-performance computers. Supercomputing 1990.

Layering in 1990

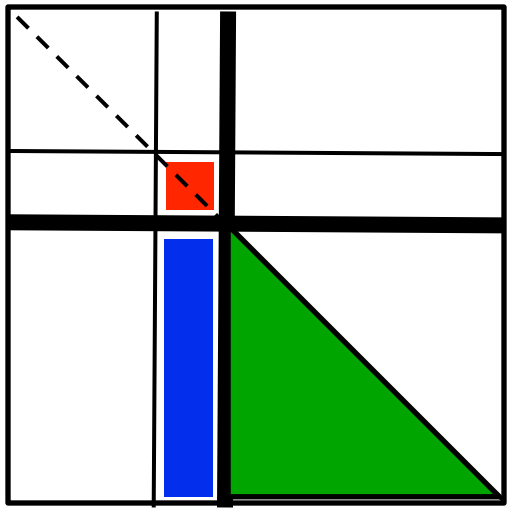


Level-2 BLAS

- Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM TOMS. 1988.*
  - **`sgemv, dgemv, cgemv, zgemv`**
    `sger, dger, cger, zger`
    `ssyr, dsyr, csyr, zsyr`
    `strsv, dtrsv, ctrsv, ztrsv`
    …

LAPACK-like Unblocked Cholesky

```
do j=1, n
   A( j,j ) = sqrt( A( j,j ) )
   call dscal( n-j, 1.0d00 / A( j,j ), A( j+1, j ), 1 )

   call dsyr( `Lower`, n-k+1, -1.0d00, A( j+1,j ), 1, A( j+1, j+1 ), lda );

enddo
```



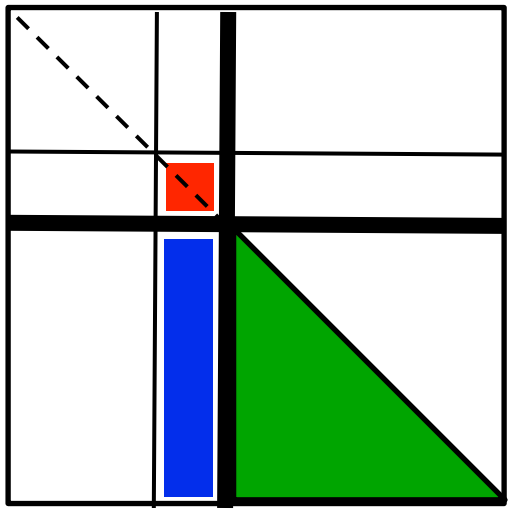
Improved performance and readability

Level-3 BLAS

- J. J. Dongarra, Jeremy Du Croz, Sven Hammarling, and I. S. Duff. A set of level 3 basic linear algebra subprograms. *ACM TOMS.* 1990.
  - **`sgemm, dgemm, cgemm, zgemm`**
    `ssymm, dsymm, csymm, zsymm`
    `ssyrk, dsyrk, csyrk, zsyrk`
    `strsm, dtrsm, ctrsm, ztrsm`
    …

LAPACK-like Blocked Cholesky

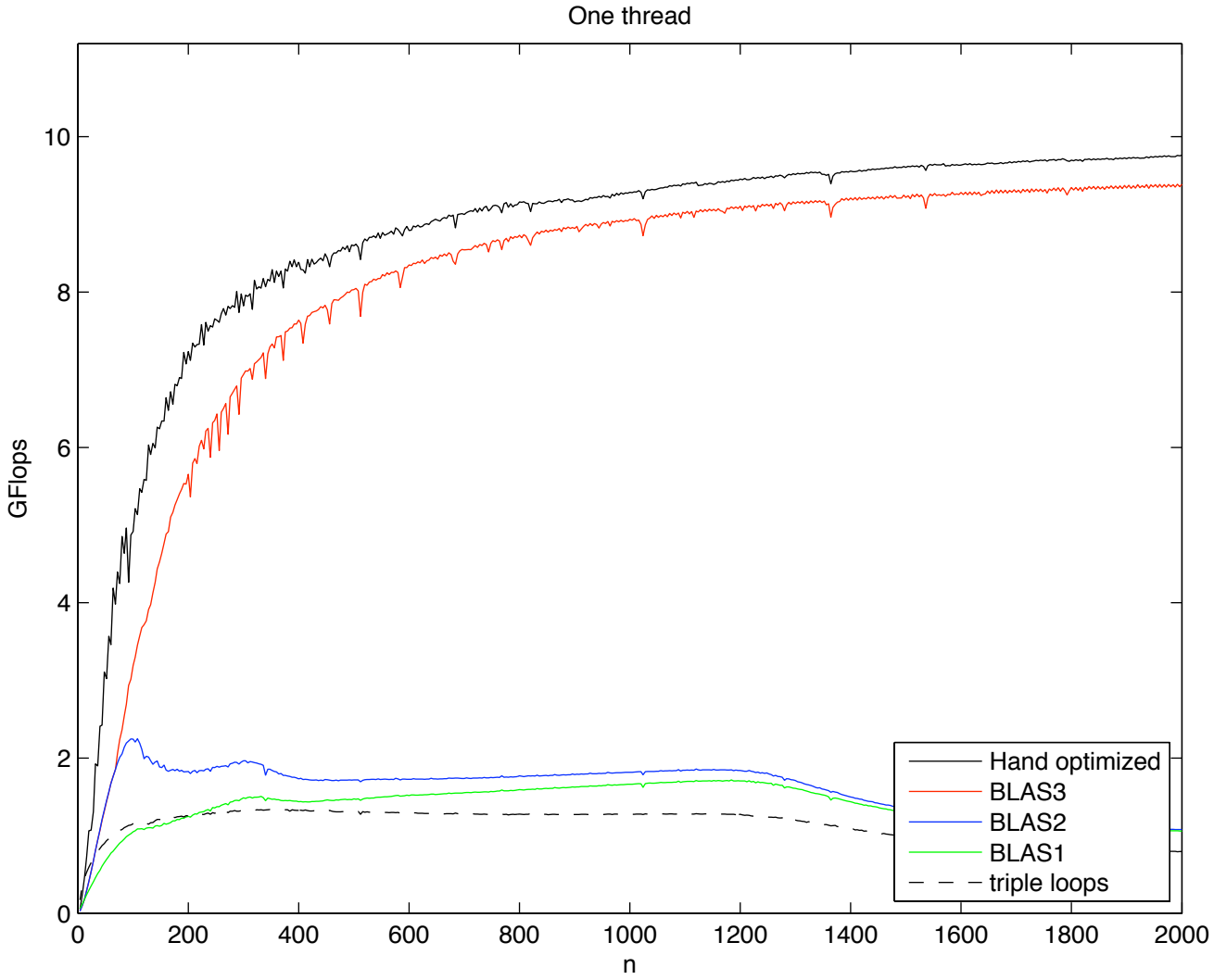
```
DO 20 J = 1, N, NB
  JB = MIN( NB, N-J+1 )
  CALL DPOTF2( 'Lower', JB, A( J, J ), LDA, INFO )
  CALL DTRSM( 'Right', 'Lower', 'Transpose', 'Non-unit',
$             N-J-JB+1, JB, ONE, A( J, J ), LDA,
$             A( J+JB, J ), LDA )
  CALL DSYRK( 'Lower', 'No transpose',
$       N-J-JB+1, JB, -ONE, A( J+JB, J ), LDA,
$       ONE, A( J+JB, J+JB ), LDA )
ENDDO
```



Improved performance

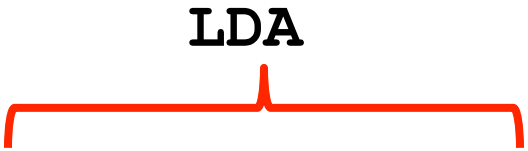
Improved readability???

Performance



The State-of-the-Art in 1996

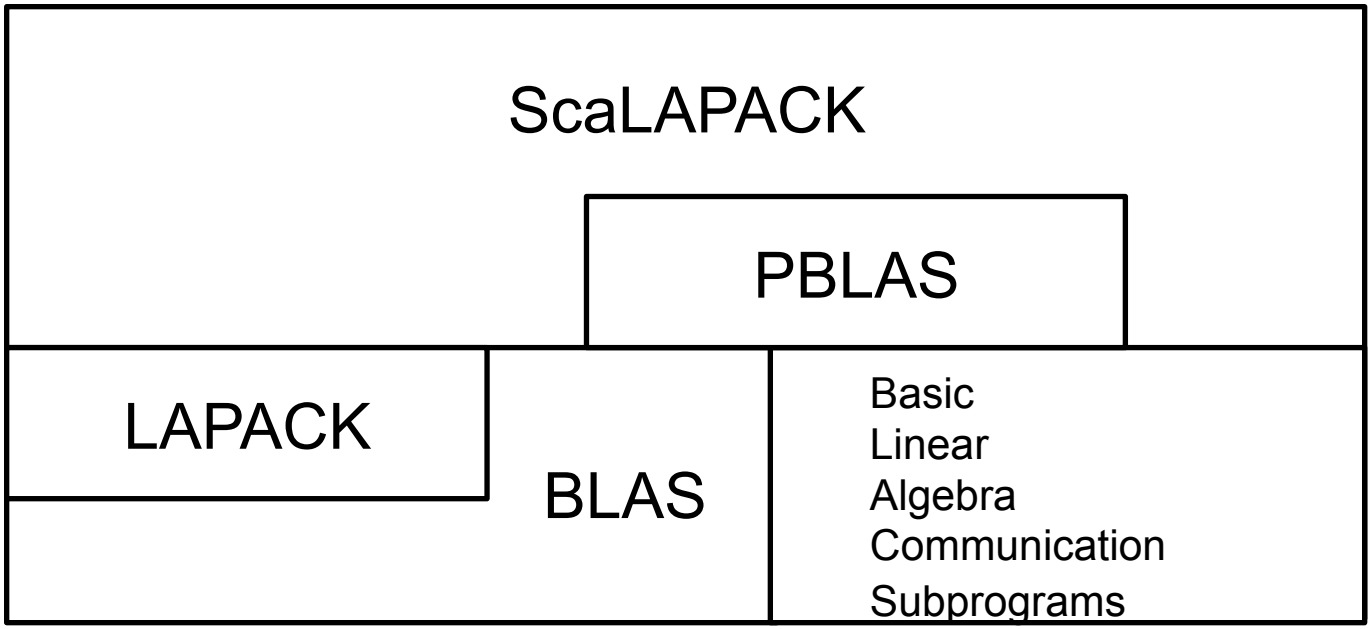
- ScaLAPACK:




```
PDPOTRF( UPLO, N, A, IA, JA, IDESC, INFO )
```

S. Blackford et al. ScaLAPACK Users' Guide. 1997.

DLA State-of-the-Art in 1996



Blackford et al. *ScaLAPACK Users' Guide.* SIAM Press. 1997.

PBLAS

```
psdot, etc.
psgemv, etc.
psgemm, etc.
  …
```

J. Choi, et. All. “ScaLAPACK: A portable linear algebra library for distributed memory computers --- Design issues and performance”, PARA '95, 1996.

DSL in the spirit of BLAS

ScaLAPACK Cholesky Factorization

```
 DO 20 J = JN+1, JA+N-1, DESCA( NB_ )
    JB = MIN( N-J+JA, DESCA( NB_ ) )
    I = IA + J - JA

    CALL PDPOTF2( UPLO, JB, A, I, J, DESCA, INFO )
    IF( INFO.NE.0 ) THEN
       INFO = INFO + J - JA
       GO TO 30
    END IF
*
    IF( J-JA+JB+1.LE.N ) THEN
      CALL PDTRSM( 'Right', 'Lower', 'Transpose', 'Non-Unit', N-J-JB+JA,
$                  JB, ONE, A, I, J, DESCA, A, I+JB, J, DESCA )

      CALL PDSYRK( UPLO, 'No Transpose', N-J-JB+JA, JB, -ONE,
     $             A, I+JB, J, DESCA, ONE, A, I+JB, J+JB,
     $                      DESCA )
*
           END IF
   20    CONTINUE
```

The “State-of-the-Art” in 1997

- PLAPACK:

```
PLA_Chol( uplo, A )
```

P. Alpatov, G. Baker, C. Edwards, J. Gunnels, G. Morrow, J. Overfelt, R. van de Geijn, Y.-J. Wu. PLAPACK: parallel linear algebra package design overview. Supercomputing, 1997.

Robert van de Geijn. Using PLAPACK. MIT Press. 1997.

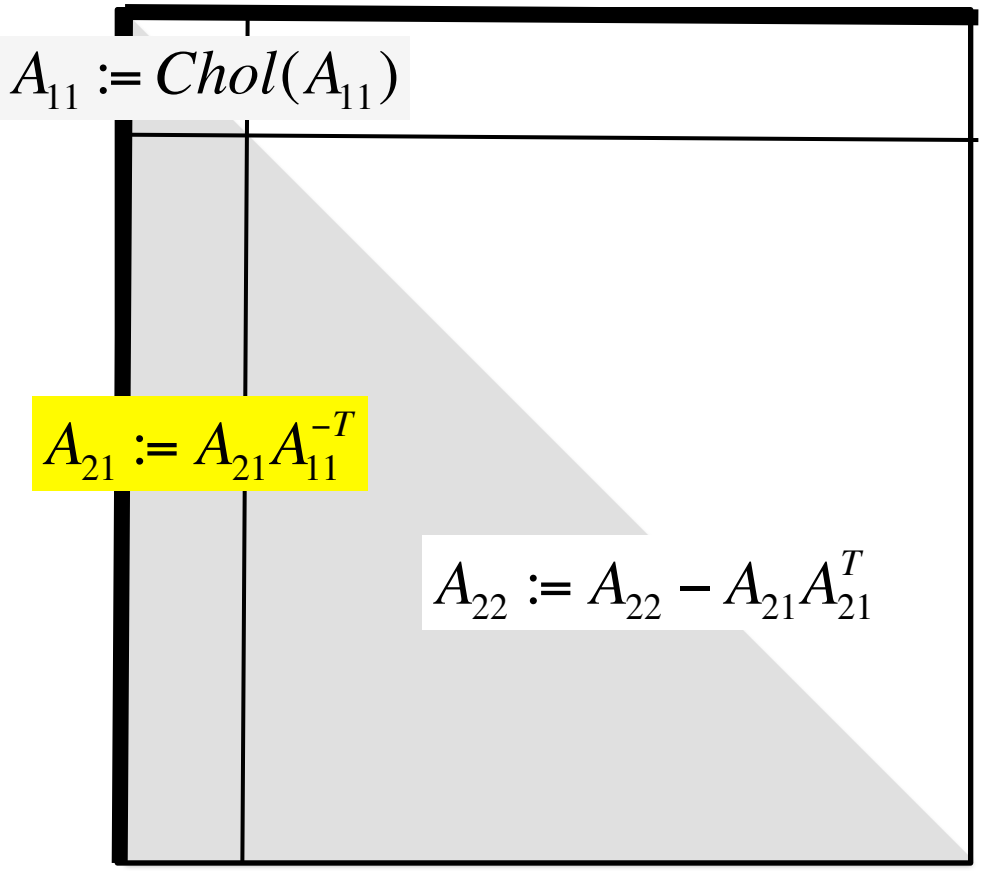
Parallel Linear Algebra Package (PLAPACK)

- Conceived in Spring 1996.
  - CS395T Parallel Computing at UT Austin
  - Users' manual for fictitious distributed memory parallel dense linear algebra (DLA) package (SL – Simple Library)
  - Class implemented parts of the package

PLAPACK Cholesky Factorization

```
PLA_Obj_view_all( A, &ABR );

while ( TRUE ) {
  PLA_Obj_split_size( ABR, PLA_SIDE_TOP,  &size_top,  &owner_top );
  PLA_Obj_split_size( ABR, PLA_SIDE_LEFT, &size_left, &owner_left )
  if ( 0 == ( size = min( min( size_top, size_left), nb_alg ) ) ) break;

  PLA_Obj_split_4( ABR, size, size,   &A11, PLA_DUMMY,
                                      &A21, &ABR );

  PLA_Local_chol( PLA_LOWER_TRIANGULAR, A11 );

  PLA_Trsm( PLA_SIDE_RIGHT, PLA_LOWER_TRIANGULAR,
            PLA_TRANSPOSE,  PLA_NONUNIT_DIAG,      one, A11, A21 );

  PLA_Syrk( PLA_LOWER_TRIANGULAR, minus_one, A21, one, ABR );
}
```

Code devoid of indexing.




```
 CALL PDTRSM( 'Left', 'Lower', 'Transpose', 'Non-Unit', JB, N-J-JB+JA,
&                     ONE, A, I, J, DESCA, A, I, J+JB, DESCA )
```

```
 PLA_Trsm( PLA_SIDE_RIGHT, PLA_LOWER_TRIANGULAR,
           PLA_TRANSPOSE,  PLA_NONUNIT_DIAG,       one, A11, A21 );
```

PLAPACK Innovations

- Object-based API (DSL) inspired by MPI.
- Describe data and its distribution
- Many new parallel DLA algorithms.
- Families of algorithms.
- Sensible user interface.

Great product, lousy marketing department…

PLAPACK's Legacy: 1996 - 2016

- FLAME
  - Notation
  - Methodology
  - APIs
    - FLAMEC
    - FLAME@lab
    - FLAMEPy
    - FLaTeX
  - libflame
- FLAME-it
- SuperMatrix
- FLAPACK
- Elemental
- BLIS
- LAC/LAP
- DxT
- LAFF
- ROTE
- TBLIS

What do we learn from our experience?

Developing a DSL for HPC

- **Reflect:** How do we reason about algorithms?
- **Abstract:** How can we express algorithms so the algorithm captures how we reason about them?
- **Derive:** How do we make reasoning about algorithms systematic?
- **Encode:** How do we represent algorithms in code?
- **(Re)layer:** How do we layer our solution to attain portable high performance?
- **Build:** How do we develop useable software?
- **Preach:** How do we get the word out? How do we build a user/developer community?

Overview

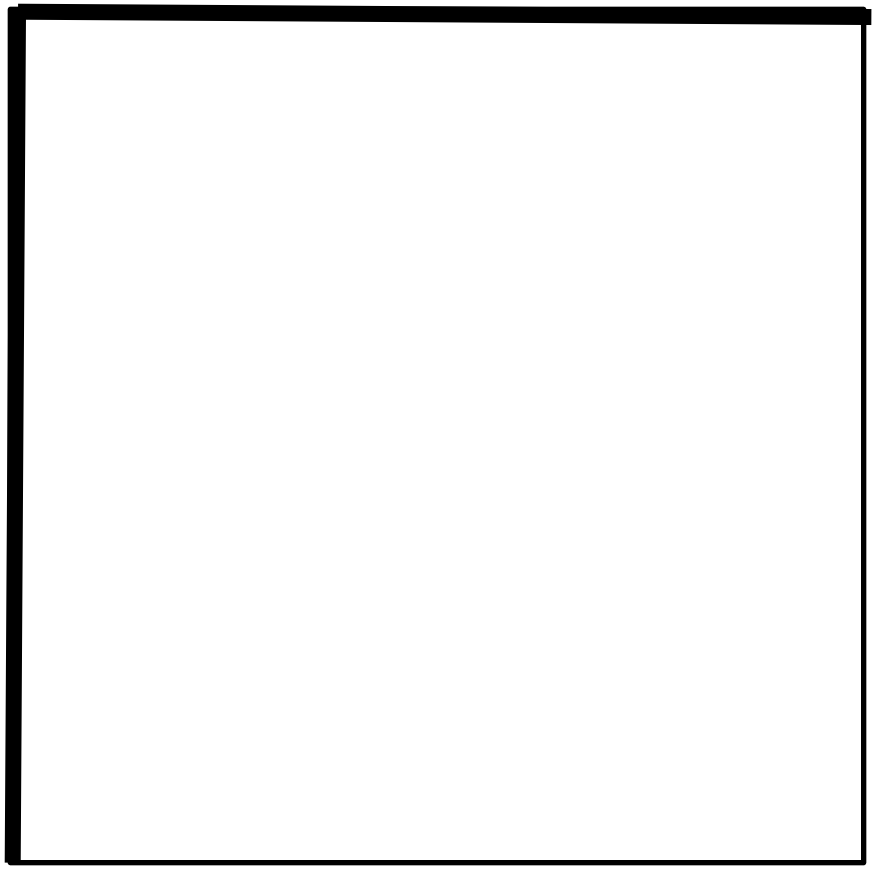
- **Reflect**
- Abstract
- Derive
- Encode
- (Re)layer
- Build
- Preach

Reflect

"The idea of knowledge as cumulative — a ladder, or a tower of stones, rising higher and higher — existed only as one possibility among many. For several hundred years, scholars of scholarship had considered that they might be like dwarves seeing farther by **standing on the shoulders of giants**, but they tended to believe more in rediscovery than in progress." - James Gleick

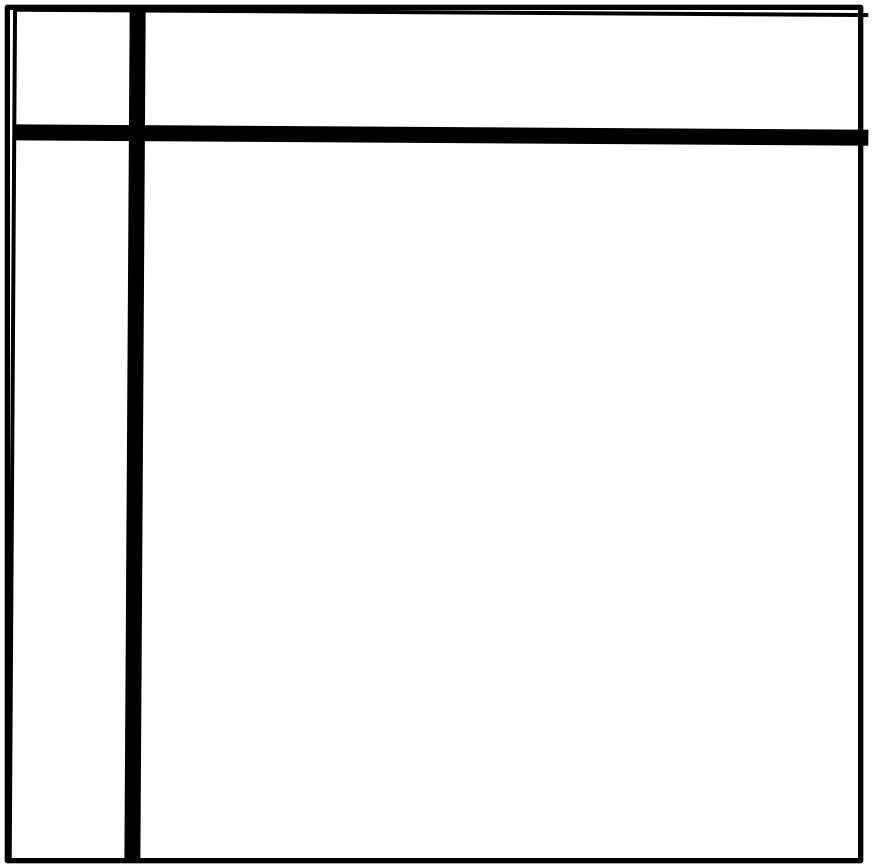
Reflect

Reflect

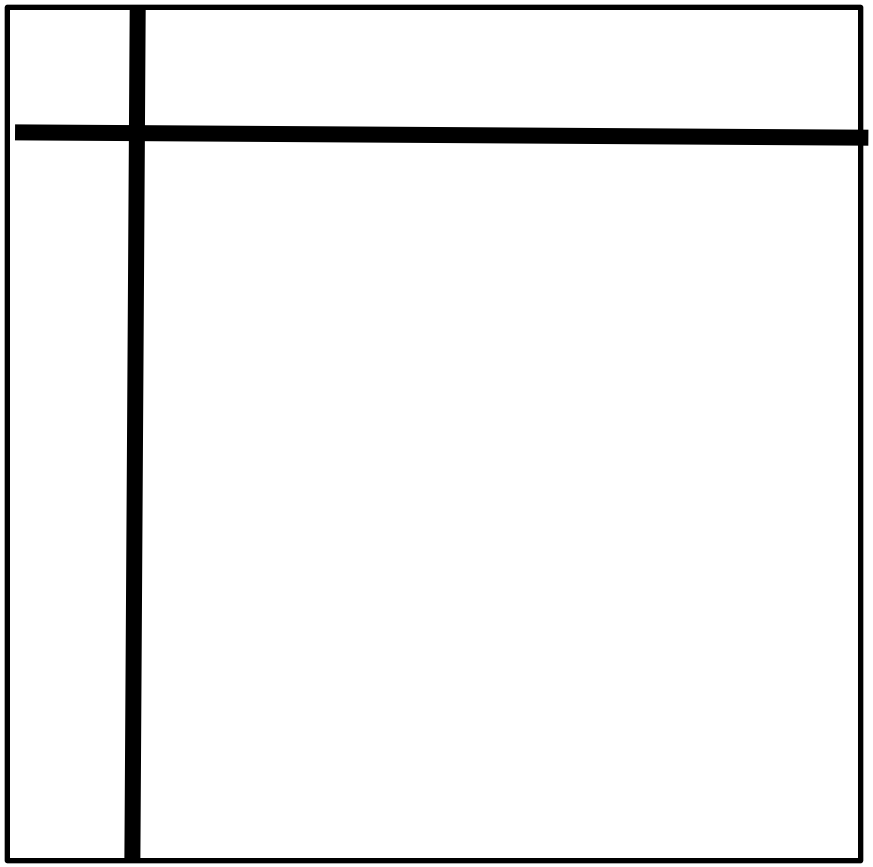


Reflect

Reflect

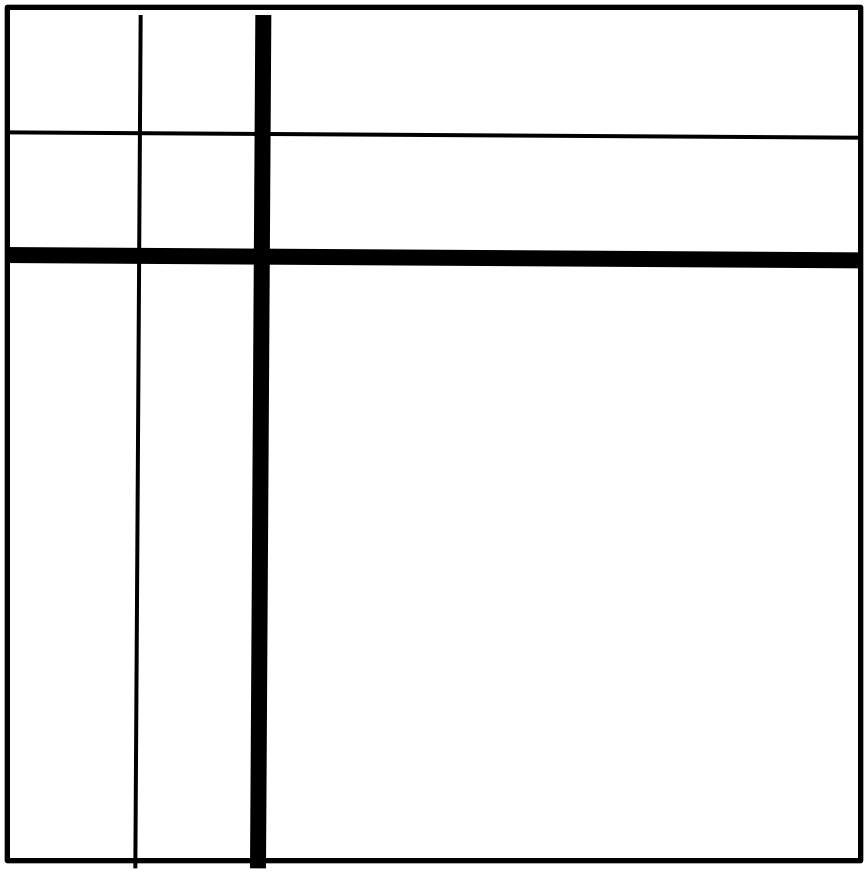


Reflect

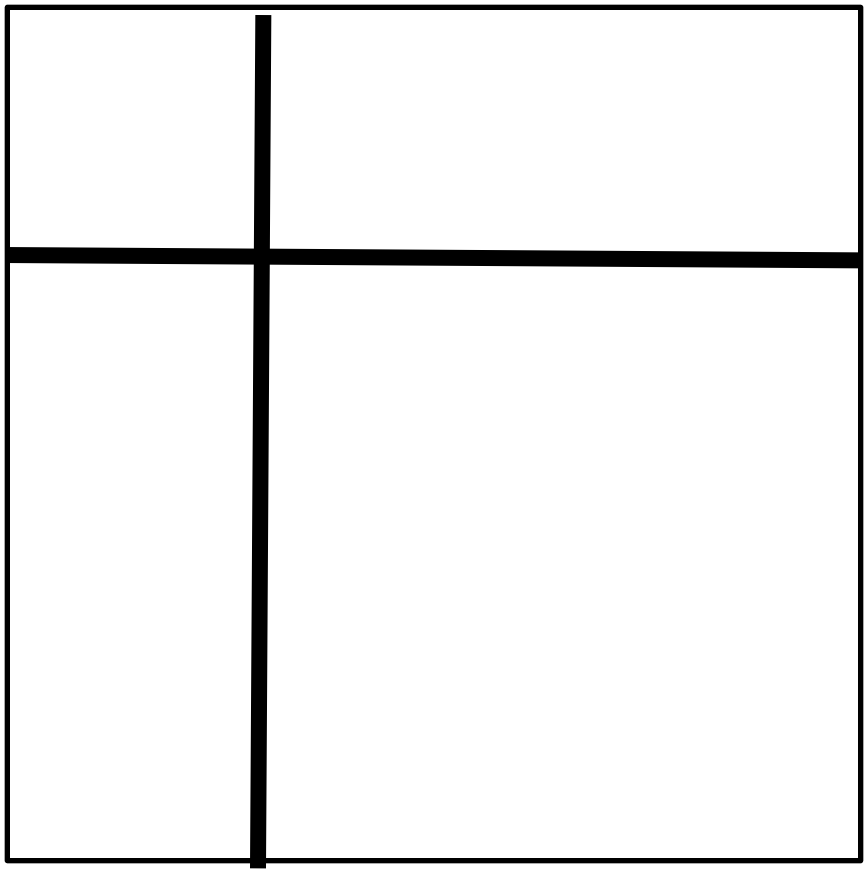


Reflect

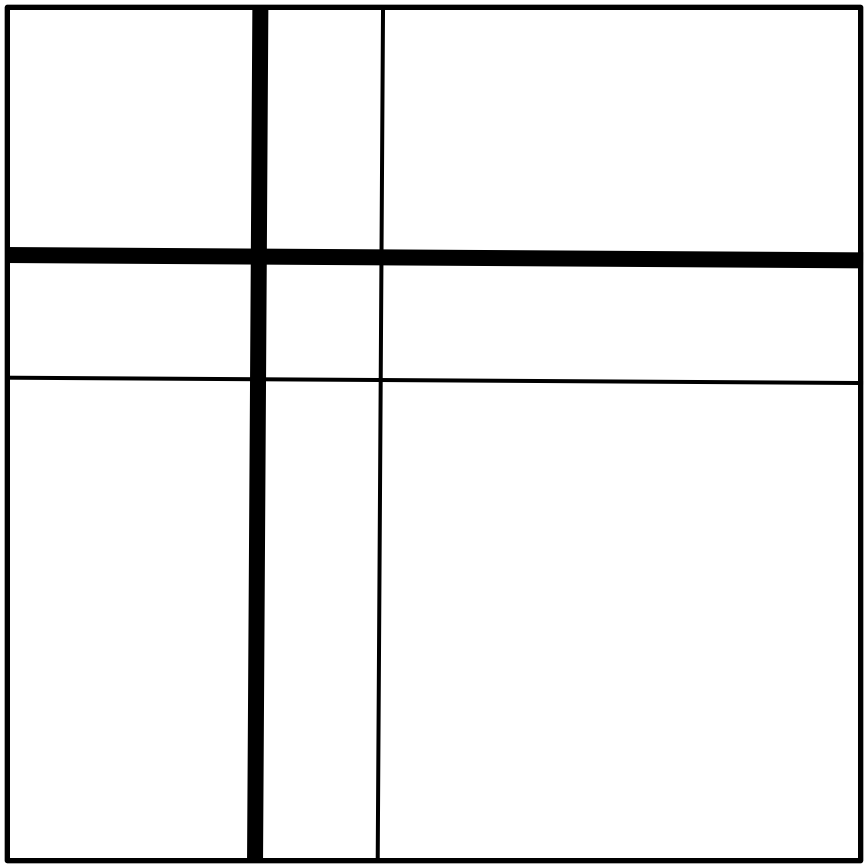
Reflect



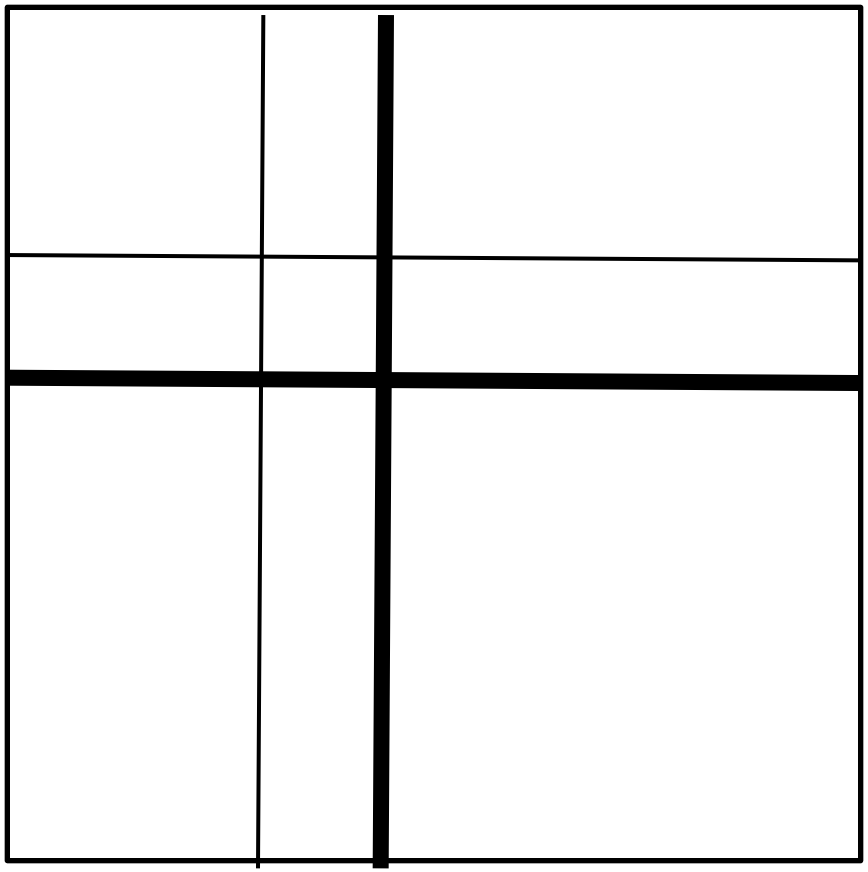
Reflect



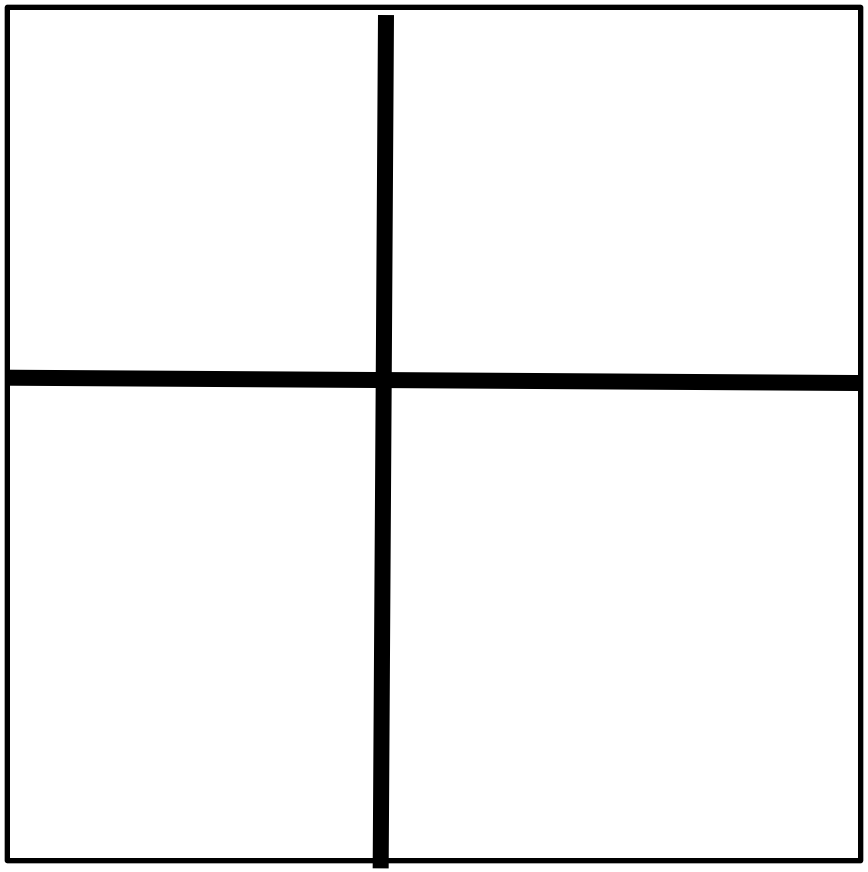
Reflect



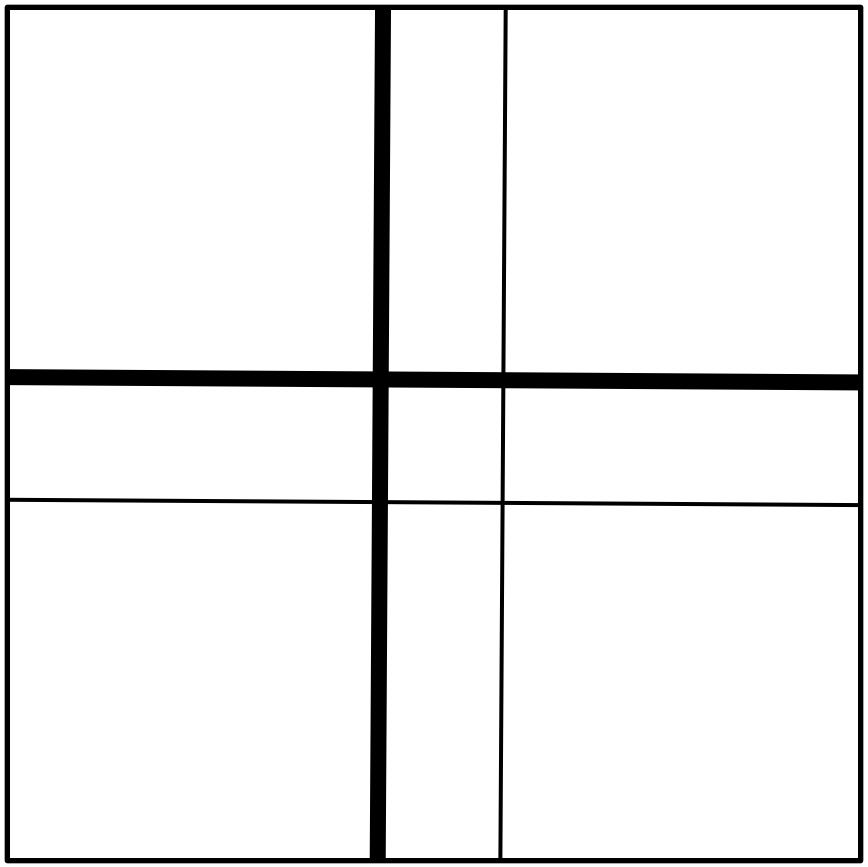
Reflect



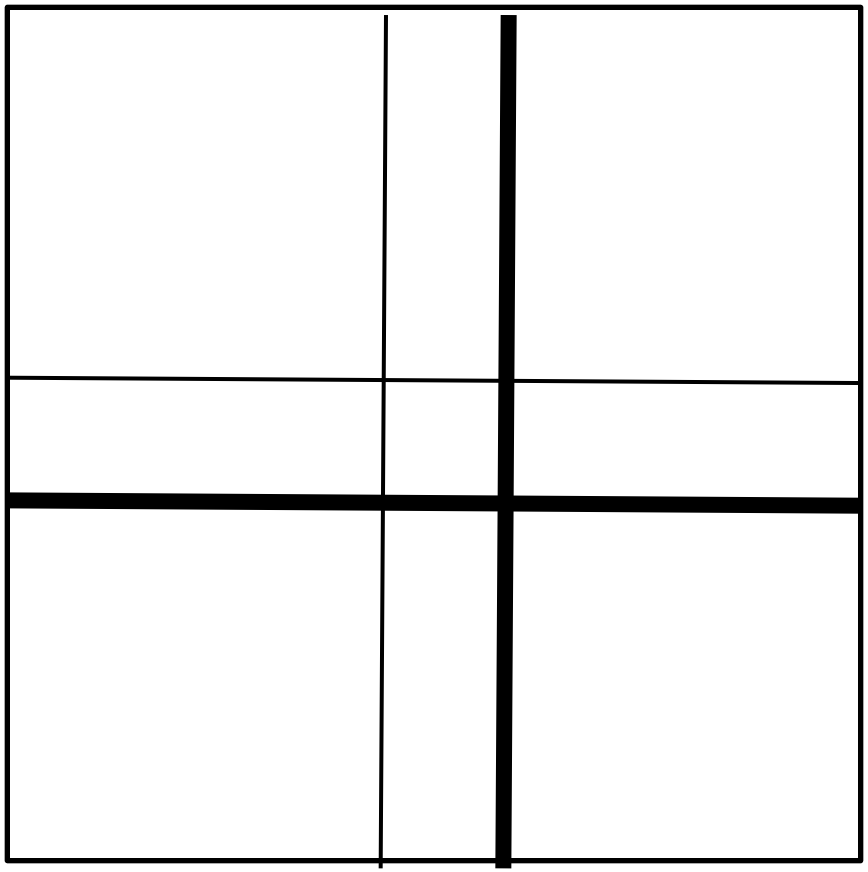
Reflect



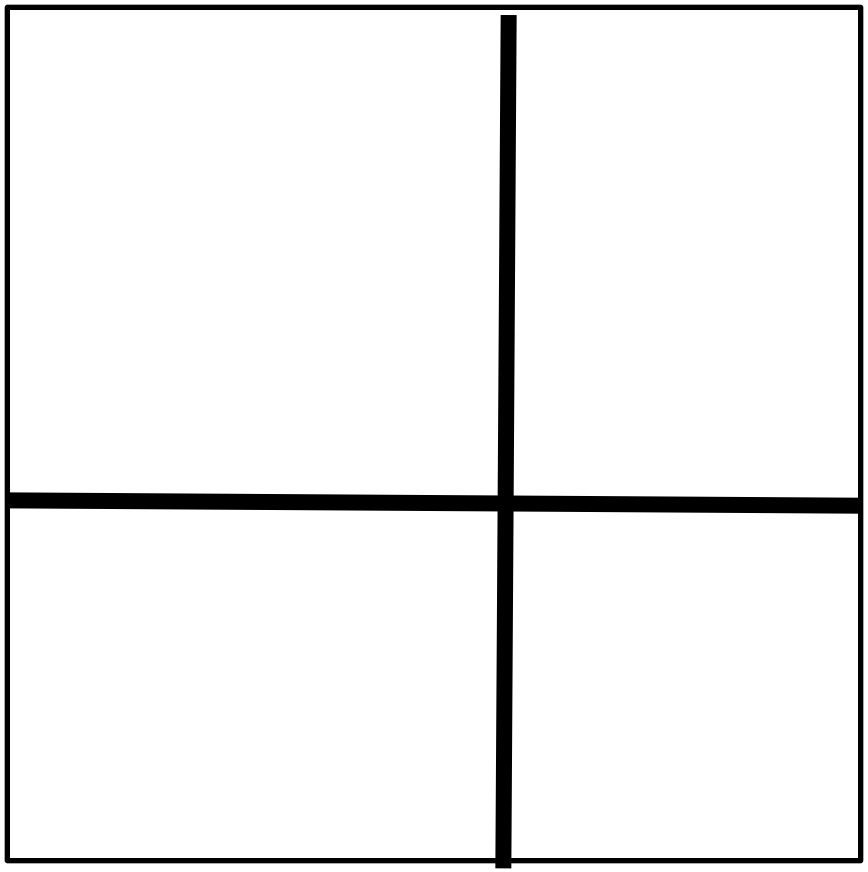
Reflect



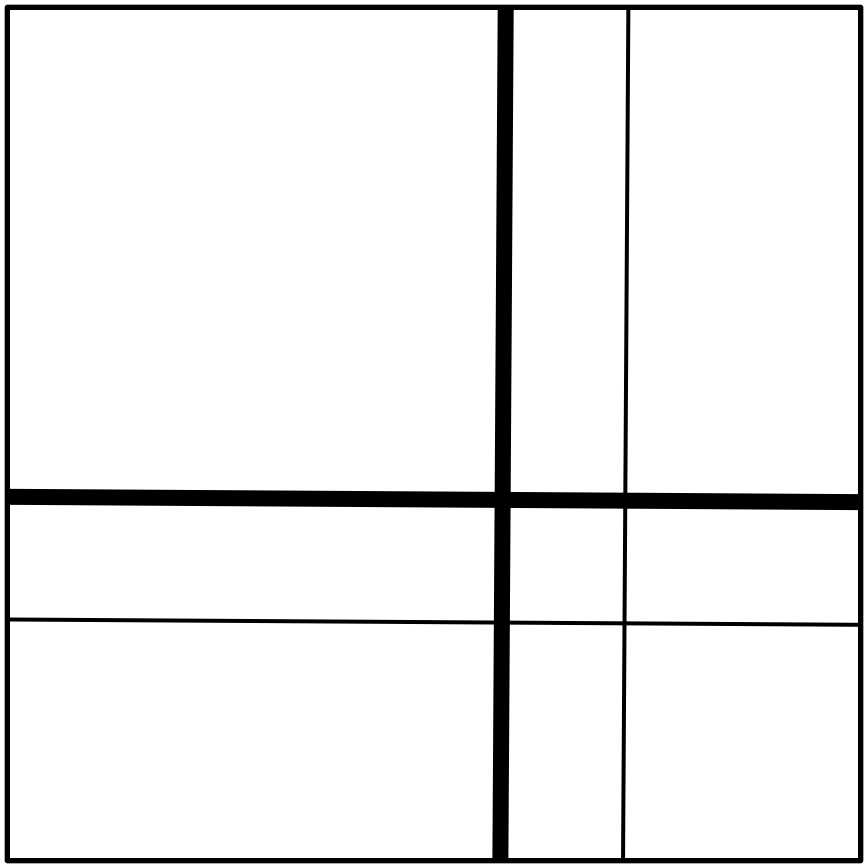
Reflect



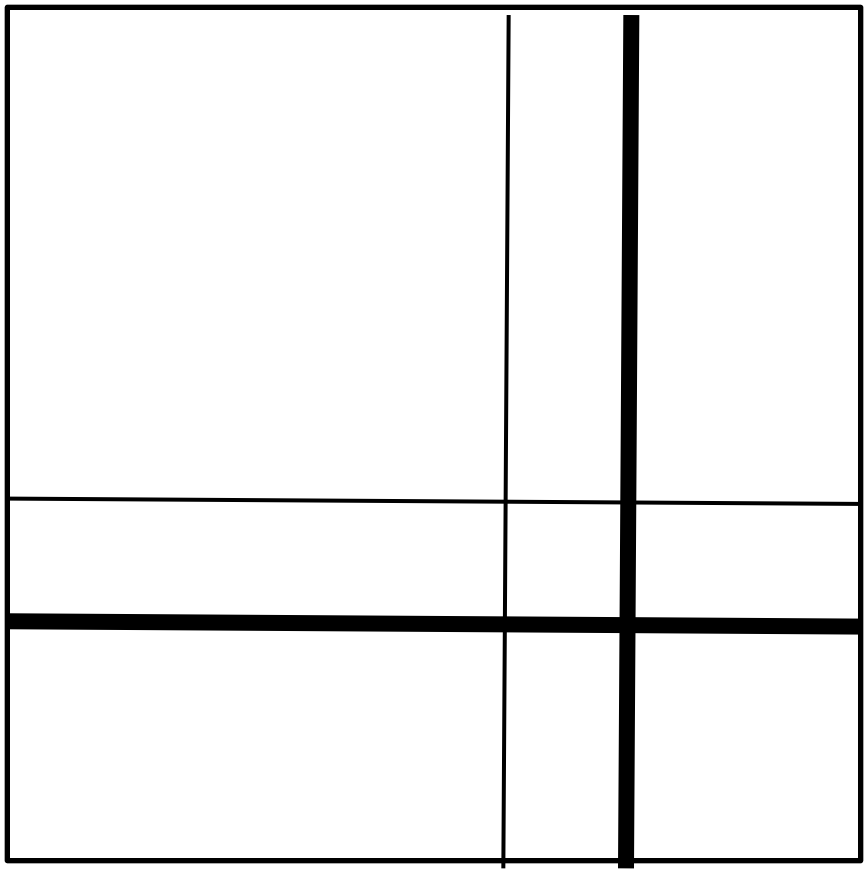
Reflect



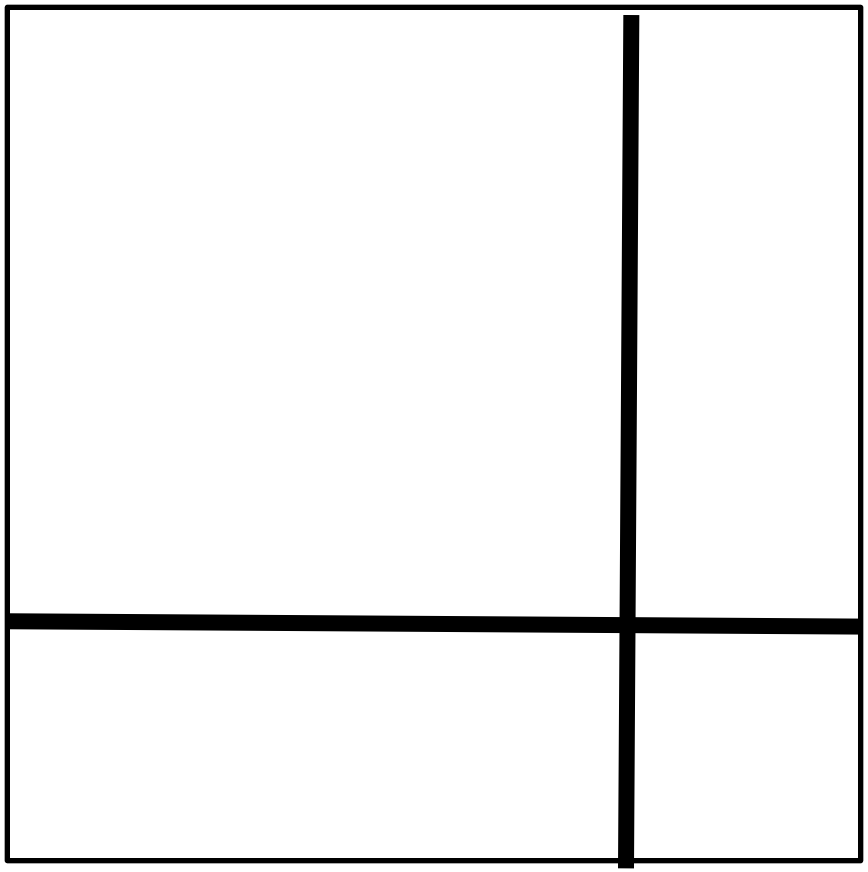
Reflect



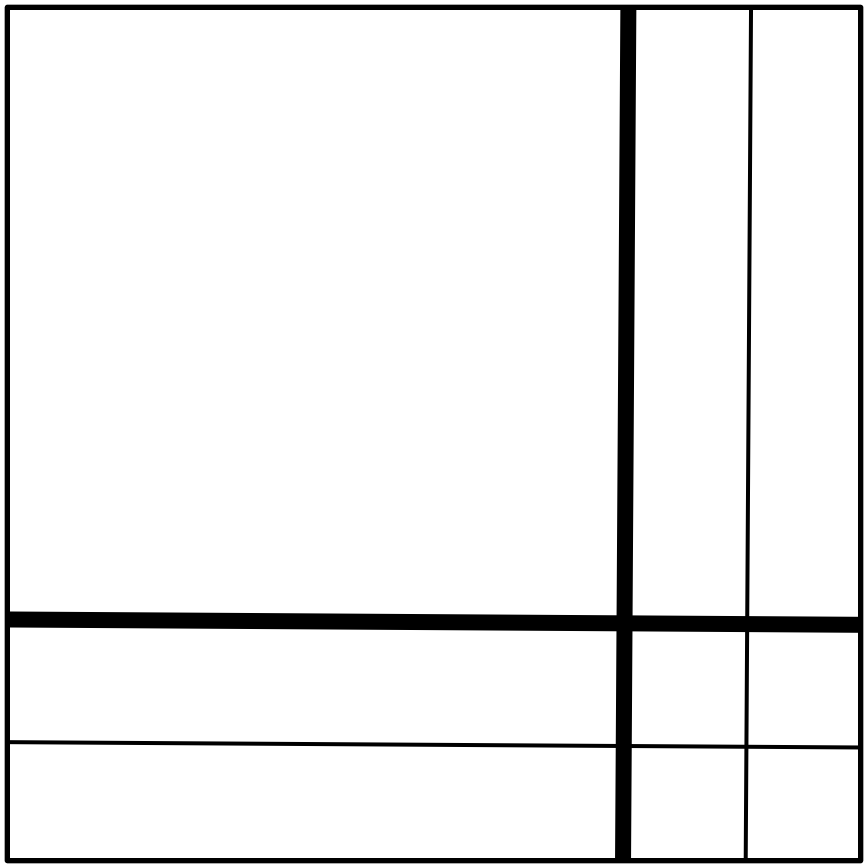
Reflect



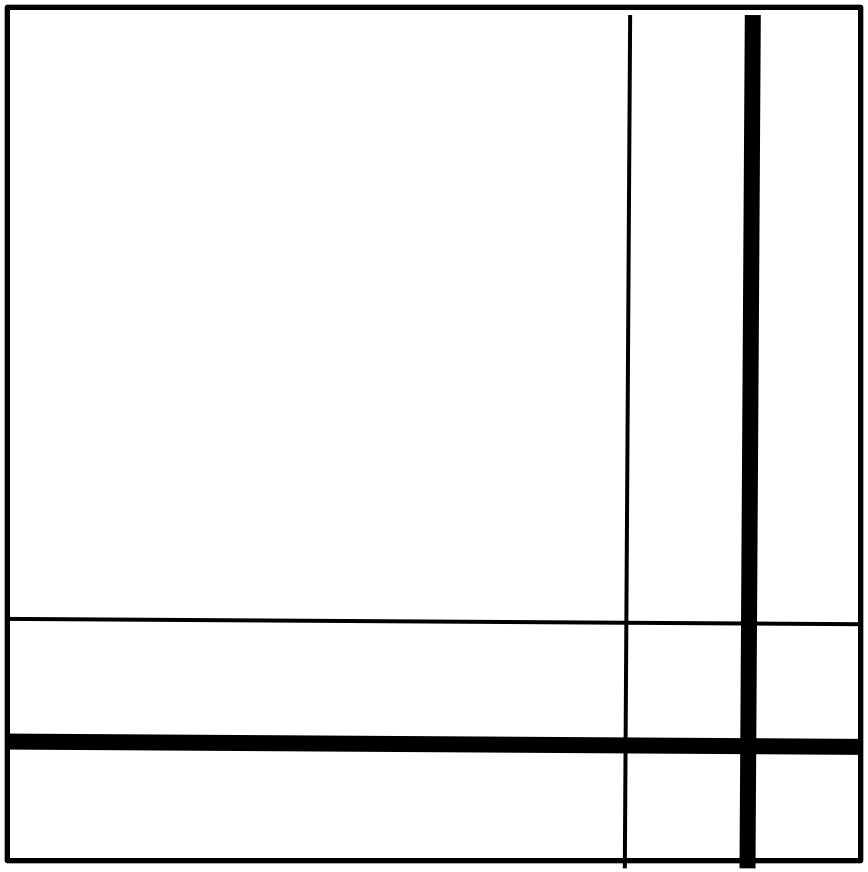
Reflect



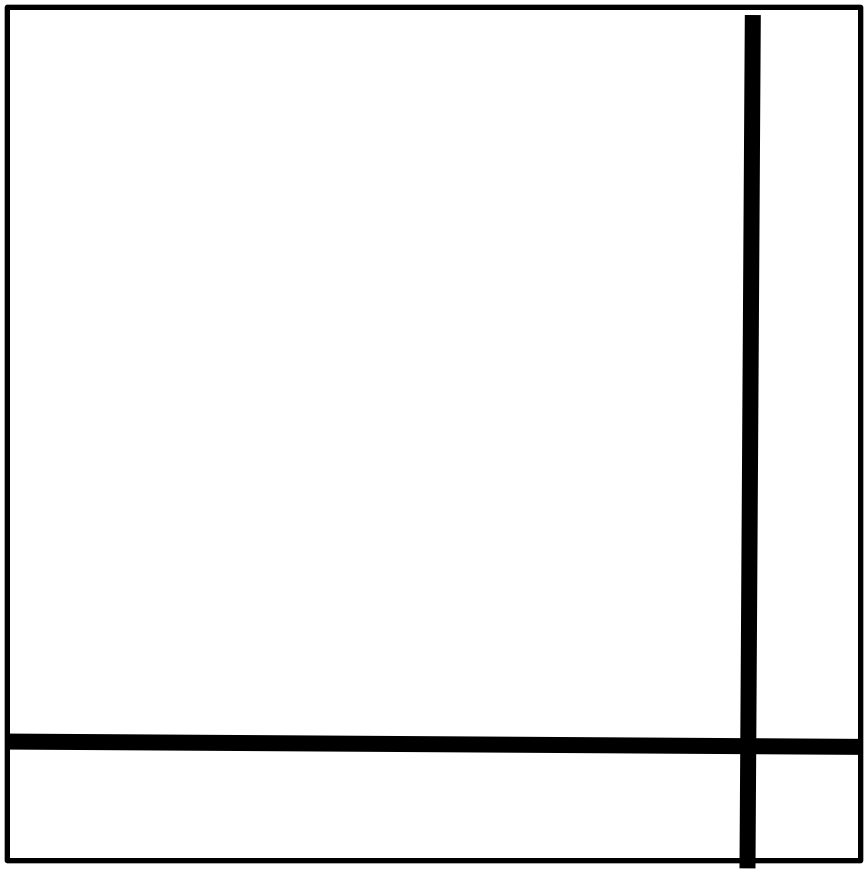
Reflect



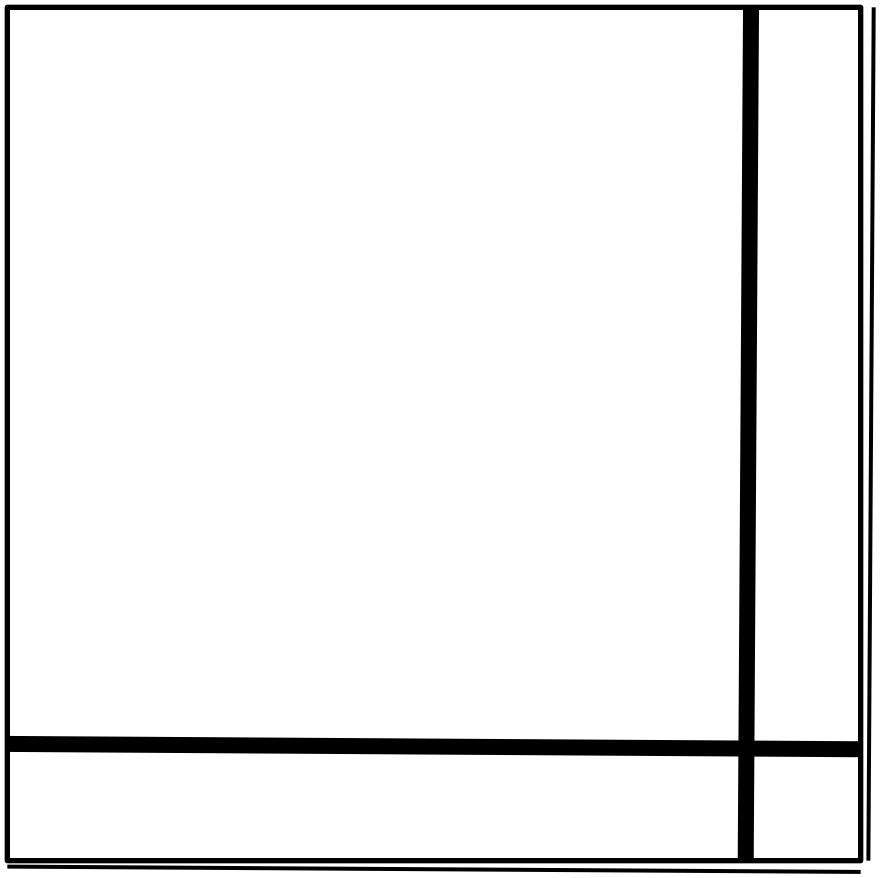
Reflect



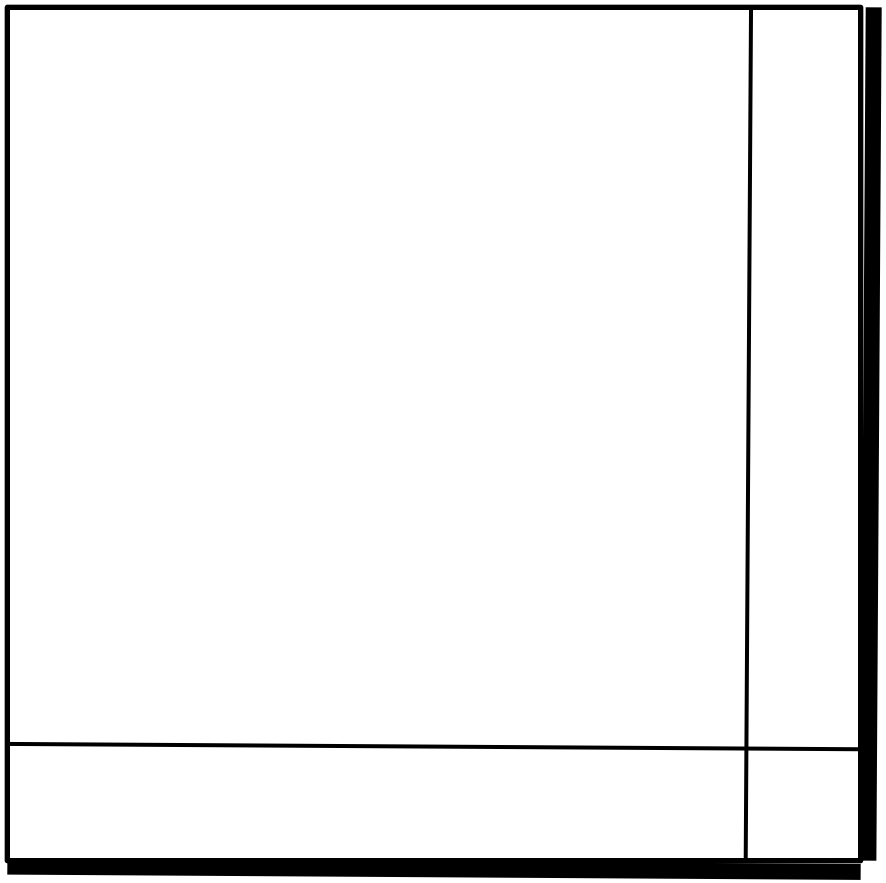
Reflect



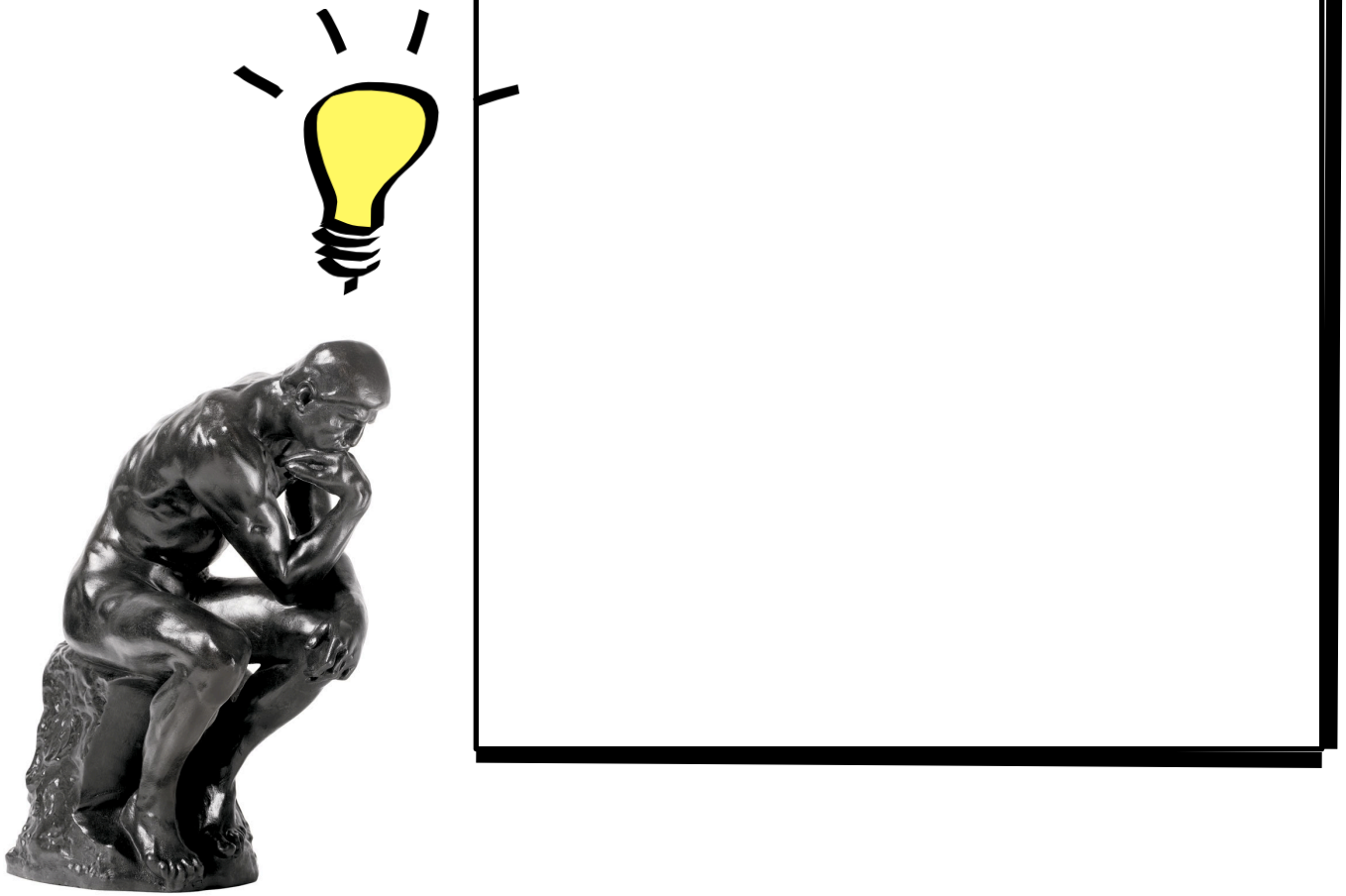
Reflect



Reflect



Reflect

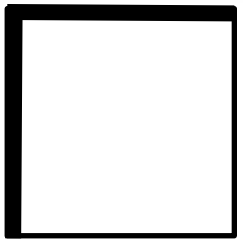


Overview

- Reflect
- **Abstract**
- Derive
- Encode
- (Re)layer
- Build
- Preach

Algorithm: $A := \text{CHOL\_UNB\_VAR3}(A)$

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



where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ do

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

where α_{11} is 1×1

$$\alpha_{11} := \sqrt{\alpha_{11}}$$

$$a_{21} := a_{21}/\alpha_{11}$$

$$A_{22} := A_{22} - a_{21}a_{21}^T \quad \text{(update only lower triangular part)}$$

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

endwhile

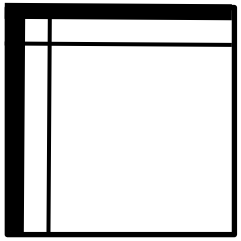
Algorithm: $A := \text{CHOL\_UNB\_VAR3}(A)$

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

where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ do

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



where α_{11} is 1×1

$$\alpha_{11} := \sqrt{\alpha_{11}}$$

$$a_{21} := a_{21}/\alpha_{11}$$

$$A_{22} := A_{22} - a_{21}a_{21}^T \quad \text{(update only lower triangular part)}$$

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

endwhile

Algorithm: $A := \text{CHOL\_UNB\_VAR3}(A)$

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

where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ do

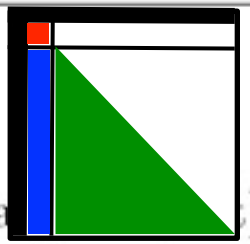
$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$

where α_{11} is 1×1

$\alpha_{11} := \sqrt{\alpha_{11}}$

$a_{21} := a_{21}/\alpha_{11}$

$A_{22} := A_{22} - a_{21}a_{21}^T$ (update only lower tria)



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

endwhile

Algorithm: $A := \text{CHOL\_UNB\_VAR3}(A)$

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

where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ do

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

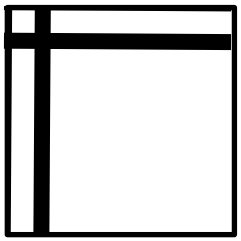
where α_{11} is 1×1

$$\alpha_{11} := \sqrt{\alpha_{11}}$$

$$a_{21} := a_{21}/\alpha_{11}$$

$A_{22} := A_{22} - a_{21}a_{21}^T$ (update only lower triangular part)

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



endwhile

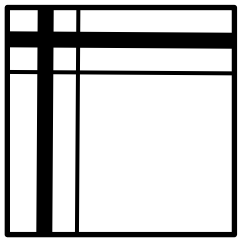
Algorithm: $A := \text{CHOL\_UNB\_VAR3}(A)$

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

where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ do

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



where α_{11} is 1×1

$$\alpha_{11} := \sqrt{\alpha_{11}}$$

$$a_{21} := a_{21}/\alpha_{11}$$

$$A_{22} := A_{22} - a_{21}a_{21}^T \quad \text{(update only lower triangular part)}$$

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

endwhile

Algorithm: $A := \text{CHOL\_UNB\_VAR3}(A)$

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

where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ do

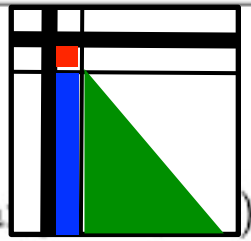
$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$

where α_{11} is 1×1

$\alpha_{11} := \sqrt{\alpha_{11}}$

$a_{21} := a_{21}/\alpha_{11}$

$A_{22} := A_{22} - a_{21}a_{21}^T$ (update only lower tria



)

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

endwhile

Algorithm: $A := \text{CHOL\_UNB\_VAR3}(A)$

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

where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ do

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

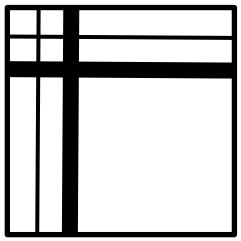
where α_{11} is 1×1

$$\alpha_{11} := \sqrt{\alpha_{11}}$$

$$a_{21} := a_{21}/\alpha_{11}$$

$$A_{22} := A_{22} - a_{21}a_{21}^T \quad \text{(update only lower triangular part)}$$

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



endwhile

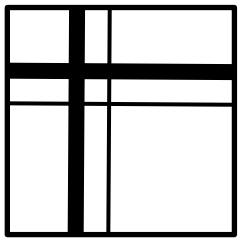
Algorithm: $A := \text{CHOL\_UNB\_VAR3}(A)$

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

where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ do

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



where α_{11} is 1×1

$$\alpha_{11} := \sqrt{\alpha_{11}}$$

$$a_{21} := a_{21}/\alpha_{11}$$

$$A_{22} := A_{22} - a_{21}a_{21}^T \quad \text{(update only lower triangular part)}$$

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

endwhile

Algorithm: $A := \text{CHOL\_UNB\_VAR3}(A)$

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

where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ do

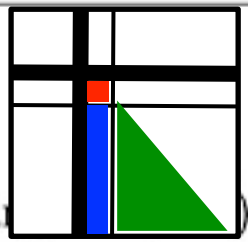
$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$$

where α_{11} is 1×1

$\alpha_{11} := \sqrt{\alpha_{11}}$

$a_{21} := a_{21}/\alpha_{11}$

$A_{22} := A_{22} - a_{21}a_{21}^T$ (update only lower tria



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

endwhile

Algorithm: $A := \text{CHOL\_UNB\_VAR3}(A)$

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

where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ do

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

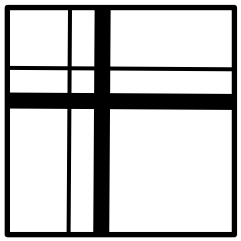
where α_{11} is 1×1

$\alpha_{11} := \sqrt{\alpha_{11}}$

$a_{21} := a_{21}/\alpha_{11}$

$A_{22} := A_{22} - a_{21}a_{21}^T$ (update only lower triangular part)

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



endwhile

FLAME Notation

X. Sun, E. S. Quintana, G. Quintana, and R. van de Geijn. A Note on Parallel Matrix Inversion. *SISC*, 2001.

Factor $A \leftarrow L\backslash U = \mathrm{LU}(A)$:

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

Determine block size b

View

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

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

$$\begin{aligned} A_{11} &\leftarrow L\backslash U_{11} = \mathrm{LU}(A_{11}) \\ A_{21} &\leftarrow L_{21} = A_{21}U_{11}^{-1} \\ A_{12} &\leftarrow U_{12} = L_{11}^{-1}A_{12} \\ A_{22} &\leftarrow \hat{A}_{22} = A_{22} - L_{21}U_{12} \\ &\boxed{(= A_{22} - (A_{21}U_{11}^{-1})(L_{11}^{-1}A_{12}))} \end{aligned}$$

Continue with

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

enddo

Insight

- It starts with the right notation:

  Express algorithms at the level of abstraction at which we reason.

Overview

- Reflect
- Abstract
- **Derive**
- Encode
- Layer
- Build
- Preach

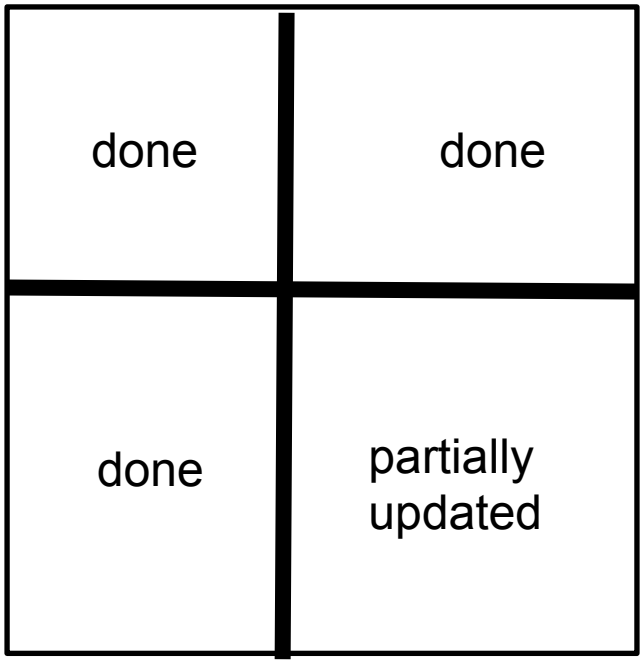
"The only effective way to raise the confidence level of a program significantly is to give a convincing proof of its correctness. But one should not first make the program and then prove its correctness, because then the requirement of providing the proof would only increase the poor programmers burden. On the contrary: the programmer should let correctness proof and program grow hand in hand."

– Dijkstra

Step	Algorithm: $A := \text{CHOL\_UNB\_VAR3}(A)$
1a	$A = \widehat{A}$
4	where
2	
3	while do
2,3	$\wedge$
5a	where
6	
8	
5b	
7	
2	
	endwhile
2,3	$\wedge \neg($ $)$
1b	$A = L \wedge LL^T = \widehat{A}$

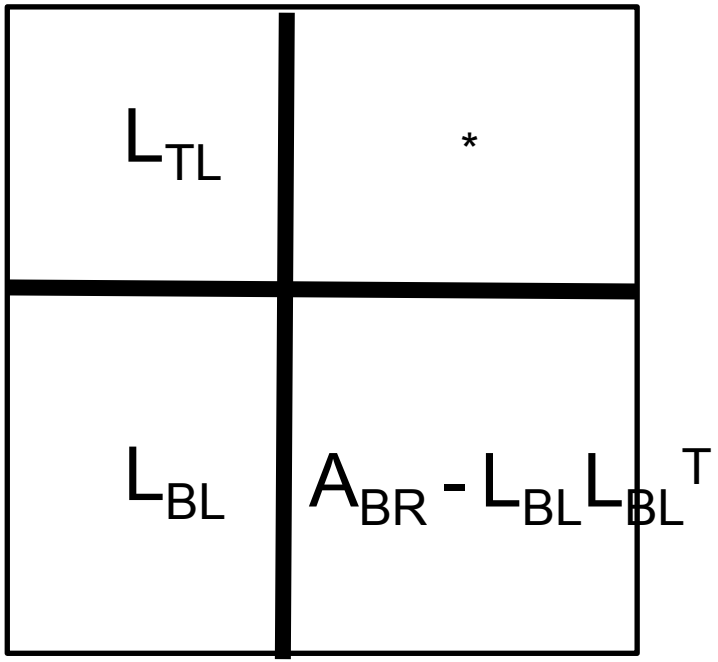
Loop invariant

- A loop invariant is a predicate that is *true* before and after each iteration of the loop.



Loop invariant

- A loop invariant is a predicate that is *true* before and after each iteration of the loop.



Important: We can determine the loop invariant a priori

The FLAME Methodology:

Our weapon of
Math induction
for the war on error

Step	Algorithm: $A := \text{CHOL\_UNB\_VAR3}(A)$
	$A \rightarrow \left(\begin{array}{c\|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$, $L \rightarrow \left(\begin{array}{c\|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array}\right)$ where A_{TL} and L_{TL} are 0×0
	while $m(A_{TL}) < m(A)$ do
	$\left(\begin{array}{c\|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$, $\left(\begin{array}{c\|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|c} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right)$ where α_{11} and λ_{11} are 1×1
	$\alpha_{11} := \sqrt{\alpha_{11}}$ $a_{21} := a_{21}/\alpha_{11}$ $A_{22} := A_{22} - a_{21}a_{21}^T$ (update only lower triangular part)
	$\left(\begin{array}{c\|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c\|c\|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$, $\left(\begin{array}{c\|c} L_{TL} & L_{TR} \\ \hline L_{BL} & L_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c\|c\|c} L_{00} & l_{01} & L_{02} \\ \hline l_{10}^T & \lambda_{11} & l_{12}^T \\ \hline L_{20} & l_{21} & L_{22} \end{array}\right)$
	endwhile

Algorithm: $A := \text{CHOL\_UNB\_VAR3}(A)$

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

where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ do

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

where α_{11} is 1×1

$\alpha_{11} := \sqrt{\alpha_{11}}$

$a_{21} := a_{21}/\alpha_{11}$

$A_{22} := A_{22} - a_{21}a_{21}^T$ (update only lower triangular part)

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

endwhile

A Systematic Methodology

- But what about more complex matrix computations?
  - They may involve many matrices.
- Can the method be automated?
- Diego Fabregat-Traver and Paolo Bientinesi.
  - Cl1ck

Related Publications

- **John A. Gunnels. A Systematic Approach to the Design and Analysis of Linear Algebra Algorithms. Ph.D. Dissertation. Nov. 2001**
- John Gunnels, Fred G. Gustavson, Greg M. Henry, Robert A. van de Geijn. FLAME: Formal Linear Algebra Methods Environment, ACM Transactions on Mathematical Software (TOMS), 2001
- Paolo Bientinesi, John A. Gunnels, Margaret E. Myers, Enrique S. Quintana-Orti, Robert A. van de Geijn. The science of deriving dense linear algebra algorithms. ACM Transactions on Mathematical Software (TOMS), 2005
- **Paolo Bientinesi. Mechanical Derivation and Systematic Analysis of Correct Linear Algebra Algorithms. Ph.D. Dissertation. Aug. 2006**
- Robert A. van de Geijn and Enrique S. Quintana-Orti. The Science of Programming Matrix Computations. www.lulu.com , 2008
- Paolo Bientinesi, John Gunnels, Maggie Myers, Enrique Quintana-Orti, Tyler Rhodes, Robert van de Geijn, and Field Van Zee. Deriving Linear Algebra Libraries. Formal Aspects of Computing. 2012
- **Tze Meng Low. A Calculus of Loop Invariants for Dense Linear Algebra Optimization. Computer Science. December 2013.**
- **…**

Insight

- Notation facilitates systematic reasoning.
- In our case: systematic to the point where it has been automated.
- In our case: we derive families of algorithms for each operation
- Pick the best algorithm for the situation

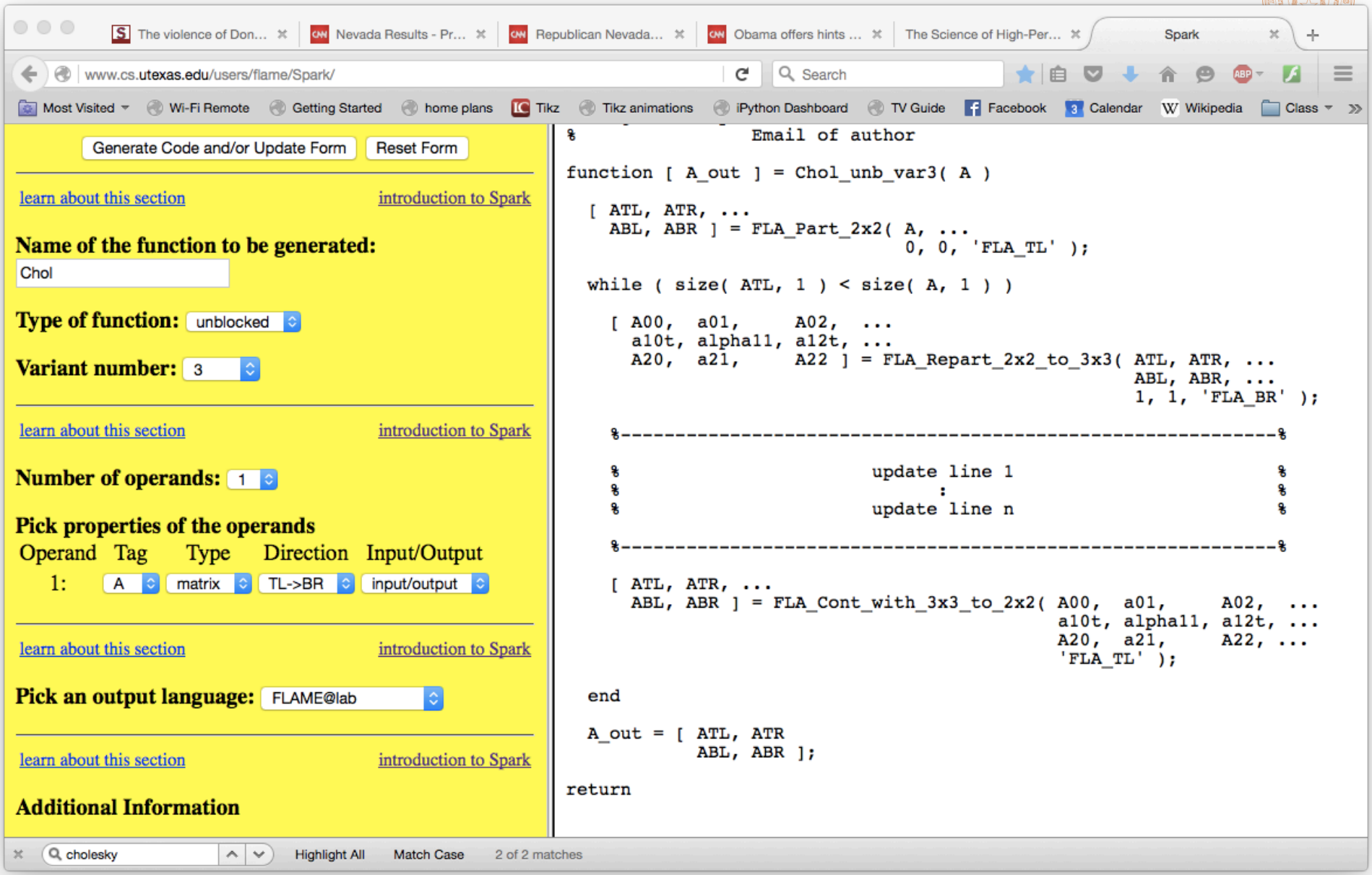
Overview

- Reflect
- Abstract
- Derive
- **Encode**
- (Re)layer
- Build
- Preach

Encoding Algorithms

- How do we translate correct algorithms to correct code?

FLAME APIs


The violence of Don...
Nevada Results - Pr...
Republican Nevada...
Obama offers hints ...
The Science of High-Per...
Spark
www.cs.utexas.edu/users/flame/Spark/
Search
Most Visited
Wi-Fi Remote
Getting Started
home plans
Tikz
Tikz animations
iPython Dashboard
TV Guide
Facebook
Calendar
Wikipedia
Class
Generate Code and/or Update Form
Reset Form
learn about this section
introduction to Spark
Name of the function to be generated:
Chol
Type of function: unblocked
Variant number: 3
learn about this section
introduction to Spark
Number of operands: 1
Pick properties of the operands
Operand Tag Type Direction Input/Output
1: A matrix TL->BR input/output
learn about this section
introduction to Spark
Pick an output language: FLAME@lab
learn about this section
introduction to Spark
Additional Information
% Email of author
function [A_out] = Chol_unb_var3(A)
[ATL, ATR, ...
ABL, ABR] = FLA_Part_2x2(A, ...
0, 0, 'FLA_TL');
while (size(ATL, 1) < size(A, 1))
[A00, a01, A02, ...
a10t, alpha11, a12t, ...
A20, a21, A22] = FLA_Repart_2x2_to_3x3(ATL, ATR, ...
ABL, ABR, ...
1, 1, 'FLA_BR');
%---%
% update line 1 %
% : %
% update line n %
%---%
[ATL, ATR, ...
ABL, ABR] = FLA_Cont_with_3x3_to_2x2(A00, a01, A02, ...
a10t, alpha11, a12t, ...
A20, a21, A22, ...
'FLA_TL');
end
A_out = [ATL, ATR
ABL, ABR];
return
cholesky
Highlight All
Match Case
2 of 2 matches

```
function [ A_out ] = Chol_unb_var3( A )

  [ ATL, ATR, ...
    ABL, ABR ] = FLA_Part_2x2( A, ...
                               0, 0, 'FLA_TL' );

  while ( size( ATL, 1 ) < size( A, 1 ) )

    [ A00,  a01,     A02,  ...
      a10t, alpha11, a12t, ...
      A20,  a21,     A22 ] = FLA_Repart_2x2_to_3x3( ATL, ATR, ...
                                                    ABL, ABR, ...
                                                    1, 1, 'FLA_BR'

    %-------------------------------------------------------------%

    %                         update line 1                       %
    %                               :                             %
    %                         update line n                       %

    %-------------------------------------------------------------%

    [ ATL, ATR, ...
      ABL, ABR ] = FLA_Cont_with_3x3_to_2x2( A00,  a01,     A02,  .
                                             a10t, alpha11, a12t, .
                                             A20,  a21,     A22, ..
                                             'FLA_TL' );

  end
```

Algorithm: $A := \text{CHOL\_UNB\_VAR3}(A)$

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

where A_{TL} is 0×0

while $m(A_{TL}) < m(A)$ do

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

where α_{11} is 1×1

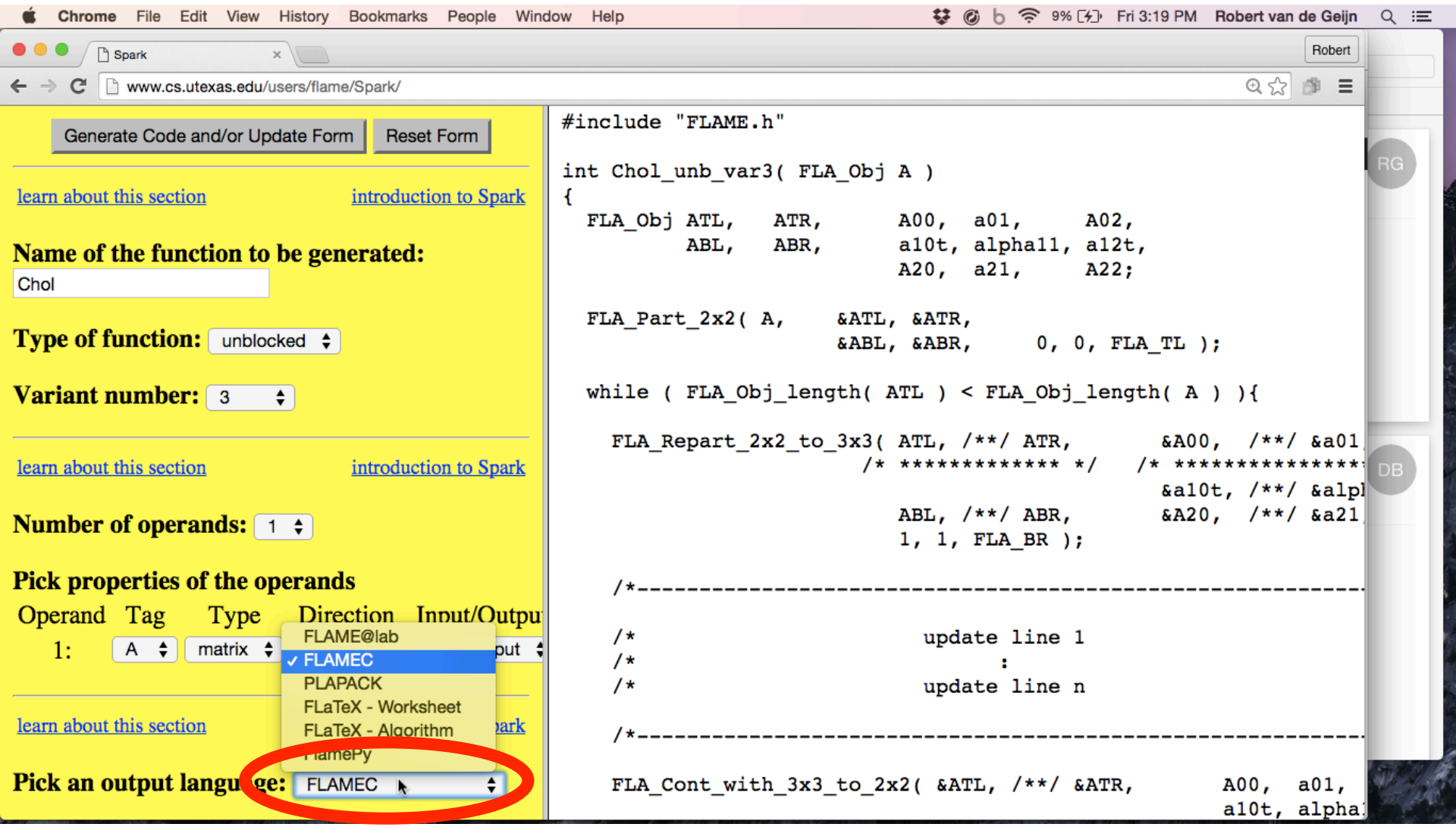
$\alpha_{11} := \sqrt{\alpha_{11}}$

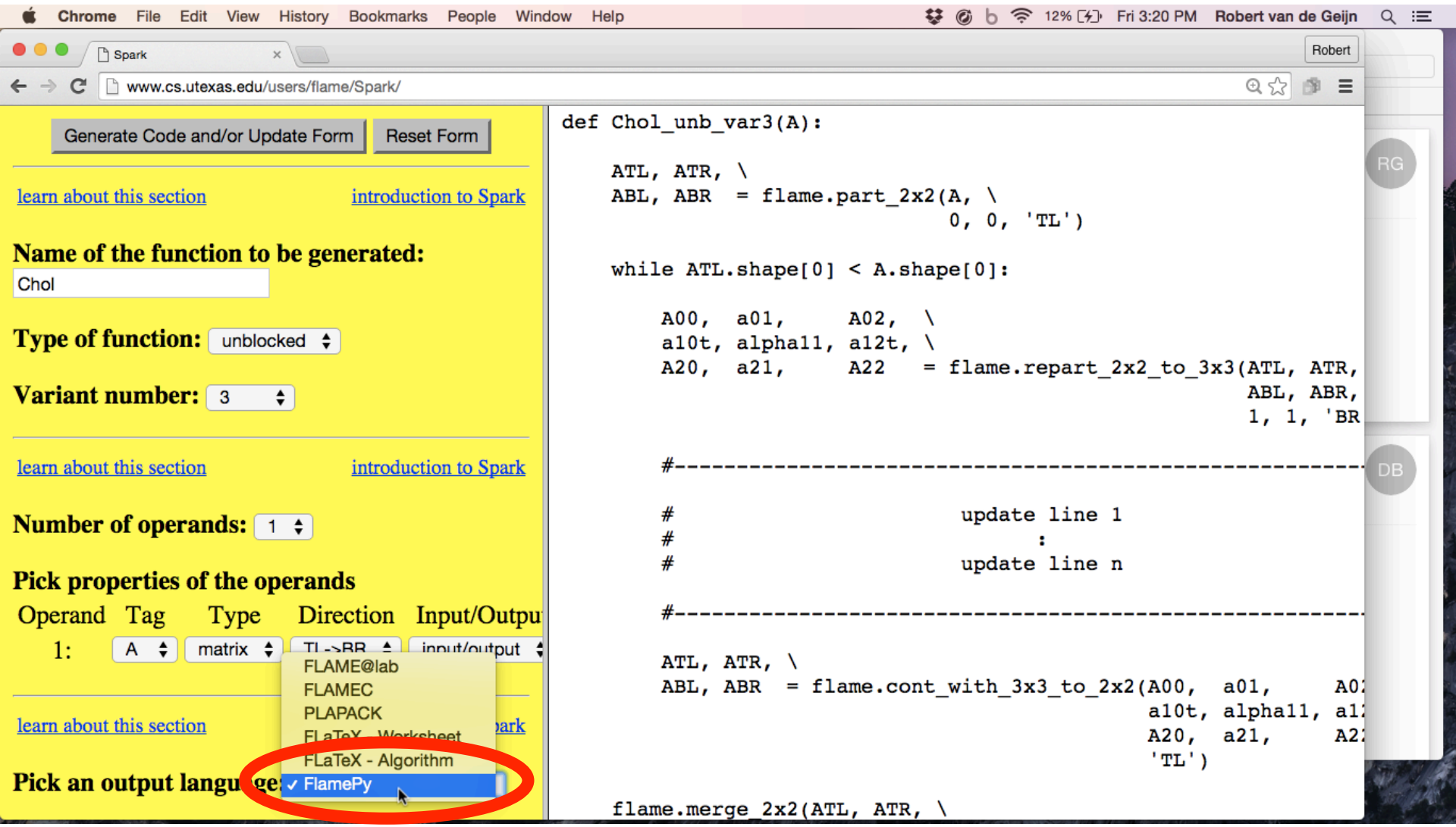
$a_{21} := a_{21}/\alpha_{11}$

$A_{22} := A_{22} - a_{21}a_{21}^T$ (update only lo

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

endwhile


Chrome File Edit View History Bookmarks People Window Help
9% Fri 3:19 PM Robert van de Geijn
Spark
Robert
www.cs.utexas.edu/users/flame/Spark/
Generate Code and/or Update Form
Reset Form
learn about this section
introduction to Spark
Name of the function to be generated:
Chol
Type of function: unblocked
Variant number: 3
learn about this section
introduction to Spark
Number of operands: 1
Pick properties of the operands
Operand Tag Type Direction Input/Outpu
1: A matrix
FLAME@lab
FLAMEC
PLAPACK
FLaTeX - Worksheet
FLaTeX - Algorithm
FlamePy
learn about this section
Pick an output langu ge: FLAMEC
#include "FLAME.h"
int Chol_unb_var3(FLA_Obj A)
{
FLA_Obj ATL, ATR, A00, a01, A02,
ABL, ABR, a10t, alpha11, a12t,
A20, a21, A22;
FLA_Part_2x2(A, &ATL, &ATR,
&ABL, &ABR, 0, 0, FLA_TL);
while (FLA_Obj_length(ATL) < FLA_Obj_length(A)){
FLA_Repart_2x2_to_3x3(ATL, /**/ ATR, &A00, /**/ &a01,
/* ************* */ /* ***************
&a10t, /**/ &alpl
ABL, /**/ ABR, &A20, /**/ &a21,
1, 1, FLA_BR);
/*--
/* update line 1
/* :
/* update line n
/*--
FLA_Cont_with_3x3_to_2x2(&ATL, /**/ &ATR, A00, a01,
a10t, alpha1


Chrome File Edit View History Bookmarks People Window Help
12% Fri 3:20 PM Robert van de Geijn
Spark
Robert
www.cs.utexas.edu/users/flame/Spark/
Generate Code and/or Update Form
Reset Form
learn about this section
introduction to Spark
Name of the function to be generated:
Chol
Type of function: unblocked
Variant number: 3
learn about this section
introduction to Spark
Number of operands: 1
Pick properties of the operands
Operand Tag Type Direction Input/Outpu
1: A matrix TL->BR input/output
learn about this section
Pick an output langu ge:
FLAME@lab
FLAMEC
PLAPACK
FLaTeX - Worksheet
FLaTeX - Algorithm
FlamePy
def Chol_unb_var3(A):
ATL, ATR, \
ABL, ABR = flame.part_2x2(A, \
0, 0, 'TL')
while ATL.shape[0] < A.shape[0]:
A00, a01, A02, \
a10t, alpha11, a12t, \
A20, a21, A22 = flame.repart_2x2_to_3x3(ATL, ATR,
ABL, ABR,
1, 1, 'BR
#
update line 1
:
update line n
#
ATL, ATR, \
ABL, ABR = flame.cont_with_3x3_to_2x2(A00, a01, A0
a10t, alpha11, a1
A20, a21, A2
'TL')
flame.merge_2x2(ATL, ATR, \

Related Publications

- John Gunnels, Fred G. Gustavson, Greg M. Henry, Robert A. van de Geijn. FLAME: Formal Linear Algebra Methods Environment, ACM Transactions on Mathematical Software (TOMS), 2001
- Paolo Bientinesi, Enrique S. Quintana-Orti, Robert A. van de Geijn. Representing linear algebra algorithms in code: the FLAME application program interfaces. ACM Transactions on Mathematical Software (TOMS), 2005
- Margaret Myers, Pierce van de Geijn, Robert van de Geijn. Linear Algebra: Foundations to Frontiers - Notes to LAFF. (MOOC) offered by edX
- Field G. Van Zee. libflame: The Complete Reference. www.lulu.com, 2009

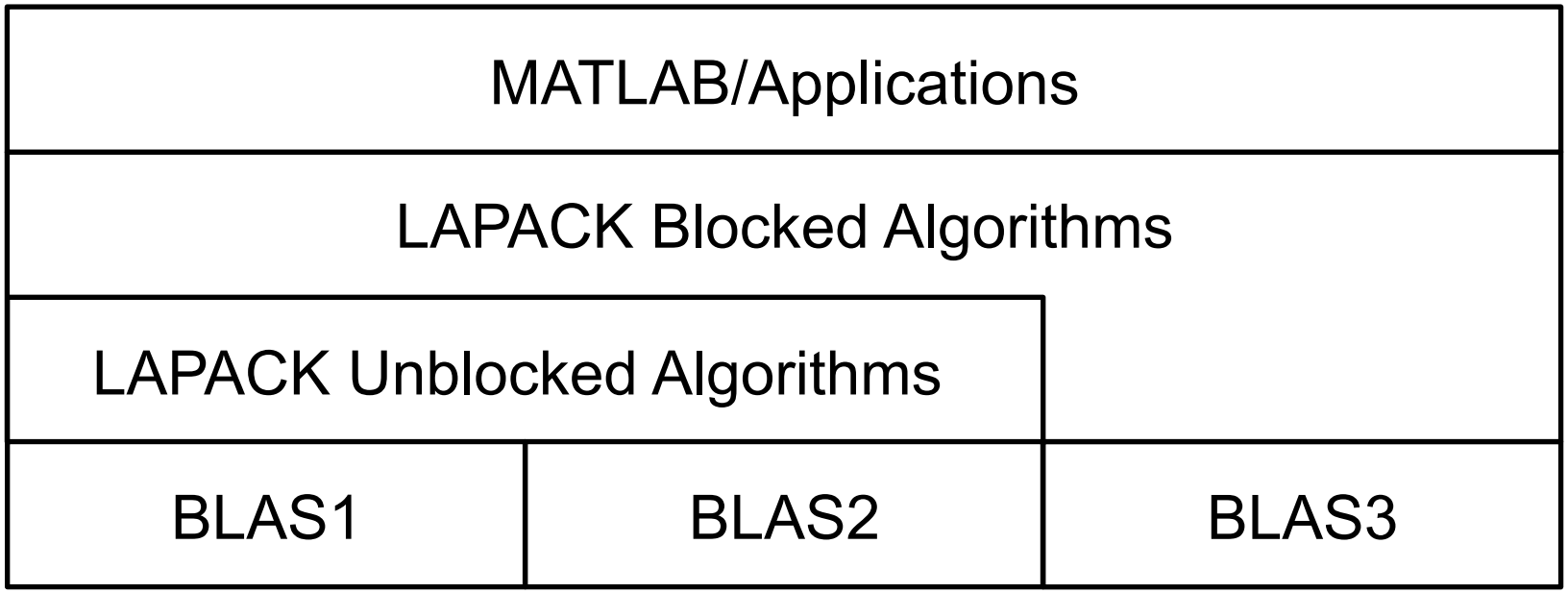
Insights

- APIs as a Domain Specific Languages (DSL) .
- Notation for algorithms should mirror how we reason.
- Implementations in code should closely resemble the algorithm.
- Our approach is language agnostic: API can be created for most (any?) language.

Overview

- Reflect
- Abstract
- Derive
- Encode
- **(Re)layer**
- Build
- Preach

Traditional Layering

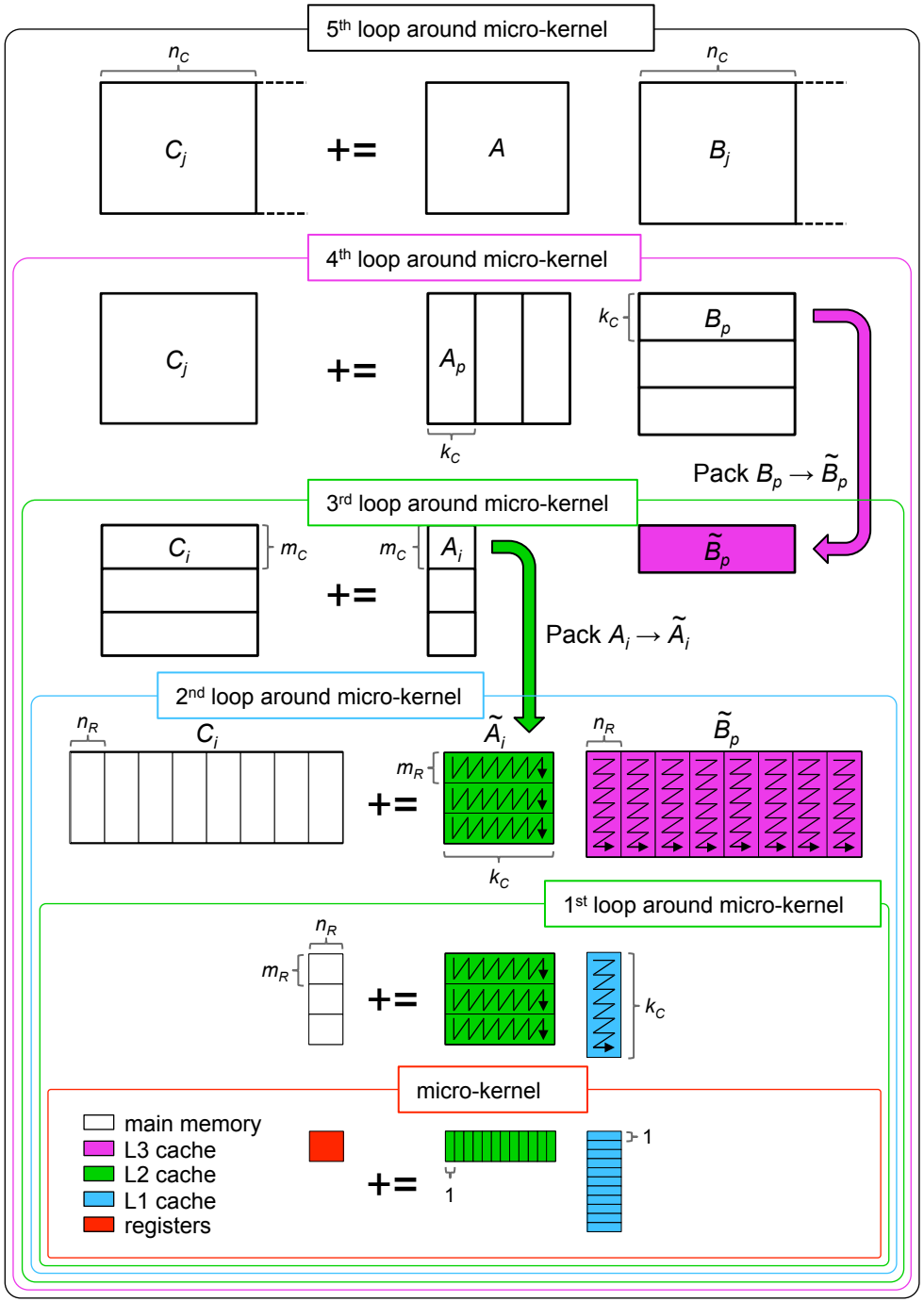


BLAS-Like Library Instantiation Framework

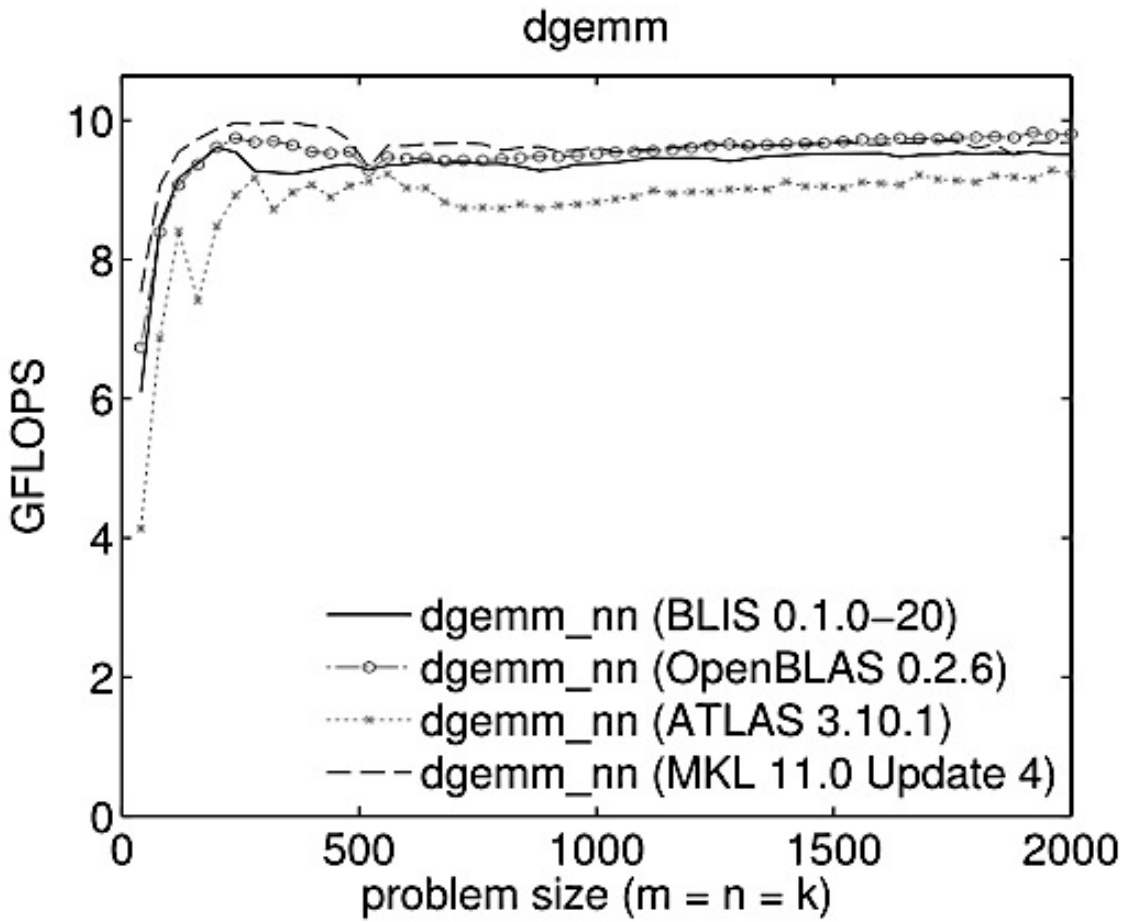
- Framework for rapid instantiation of the BLAS
  - Level 1, 2, and 3 BLAS
- Alternative APIs
- Set of building blocks for BLAS-like operations

Field G. Van Zee, Robert A. van de Geijn. BLIS: A Framework for Rapidly Instantiating BLAS Functionality. ACM Transactions on Mathematical Software (TOMS), 2015

$$C = AB + C$$

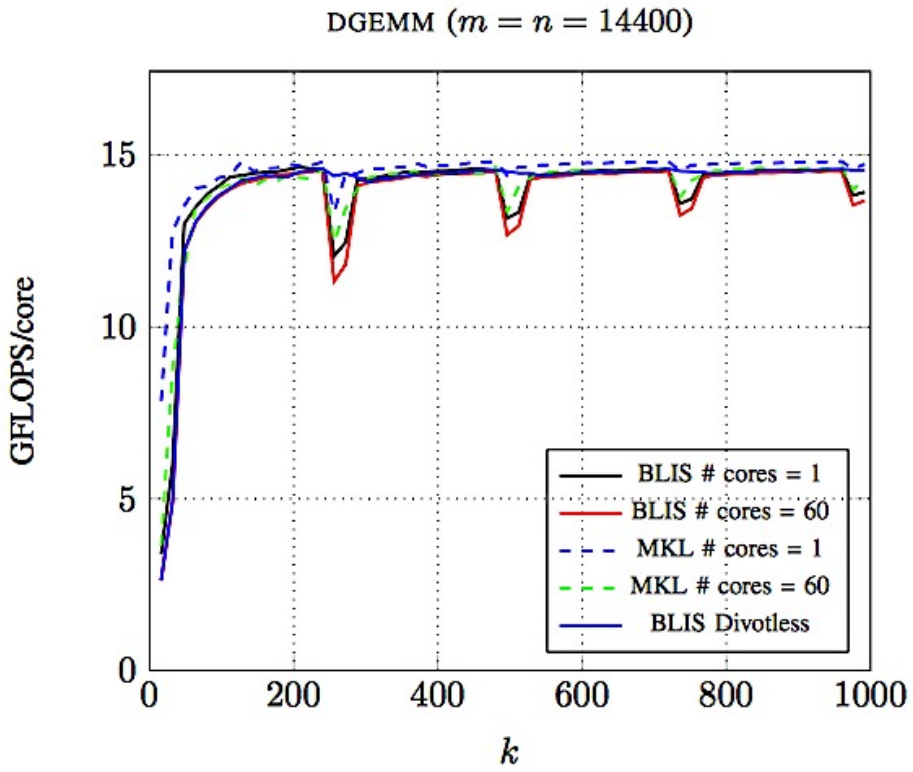

5th loop around micro-kernel
nC
Cj
+=
A
Bj
4th loop around micro-kernel
Ap
kC
Bp
Pack Bp → B̃p
3rd loop around micro-kernel
Ci
mC
Ai
Pack Ai → Ãi
2nd loop around micro-kernel
nR
mR
1st loop around micro-kernel
micro-kernel
main memory
L3 cache
L2 cache
L1 cache
registers
1

Intel Xeon “Dunnington” (one core)



Intel Xeon Phi

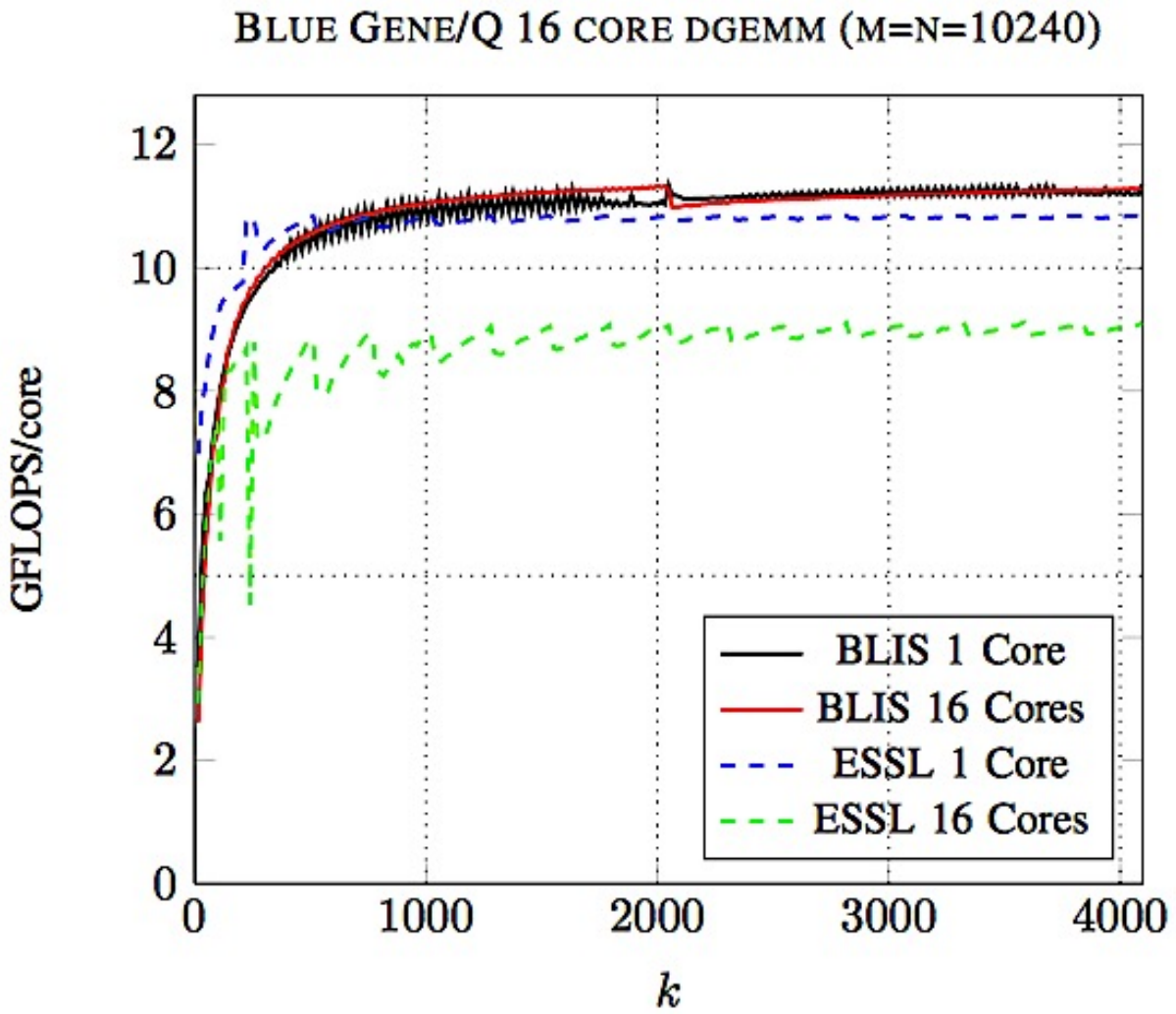
240 threads



84% of peak

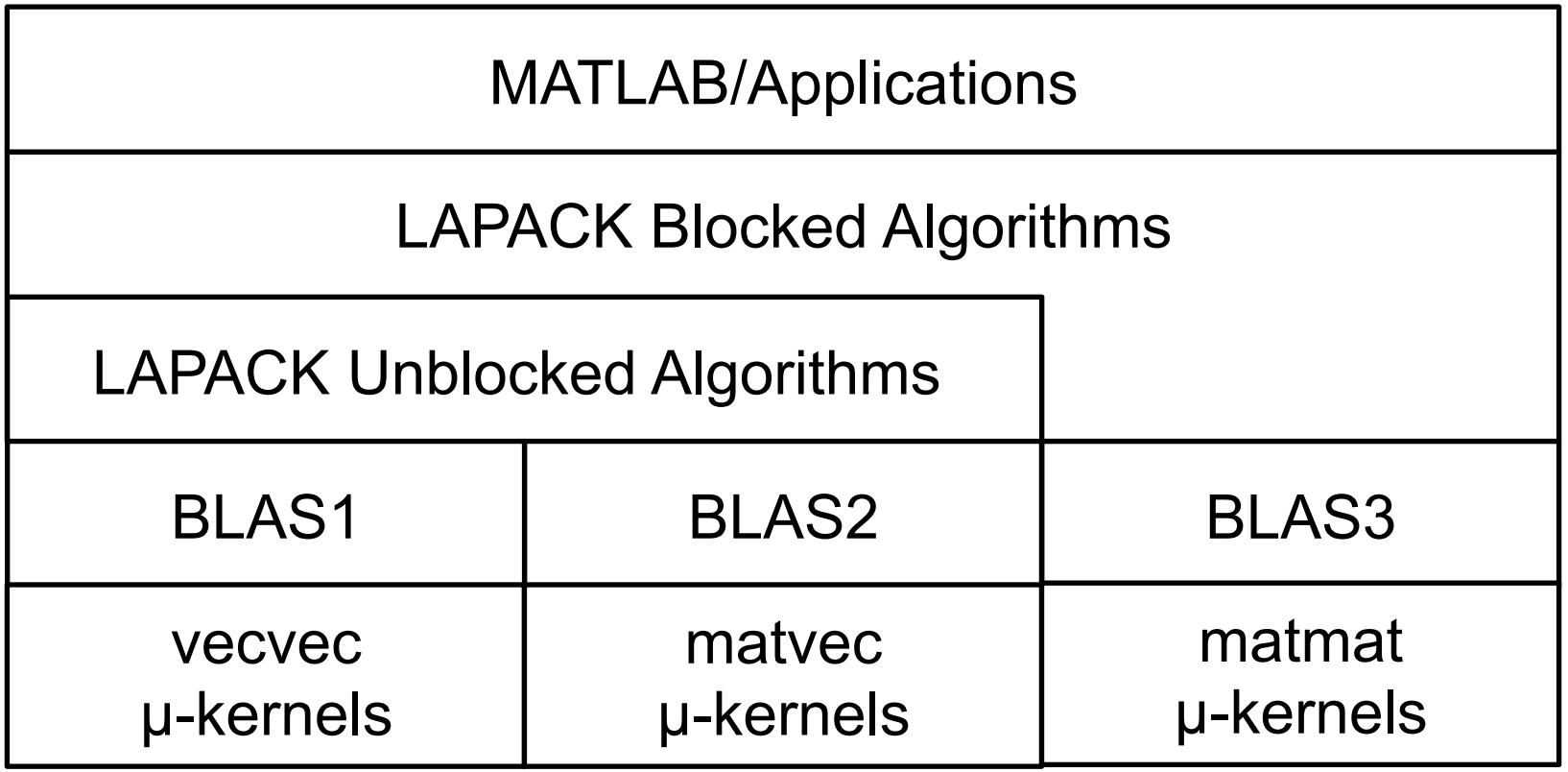
T. M. Smith, R. van de Geijn, M. Smelyanskiy, J. R. Hammond, and F. G. Van Zee. Anatomy of High-Performance Many-Threaded Matrix Multiplication. IPDPS 2014.

IBM

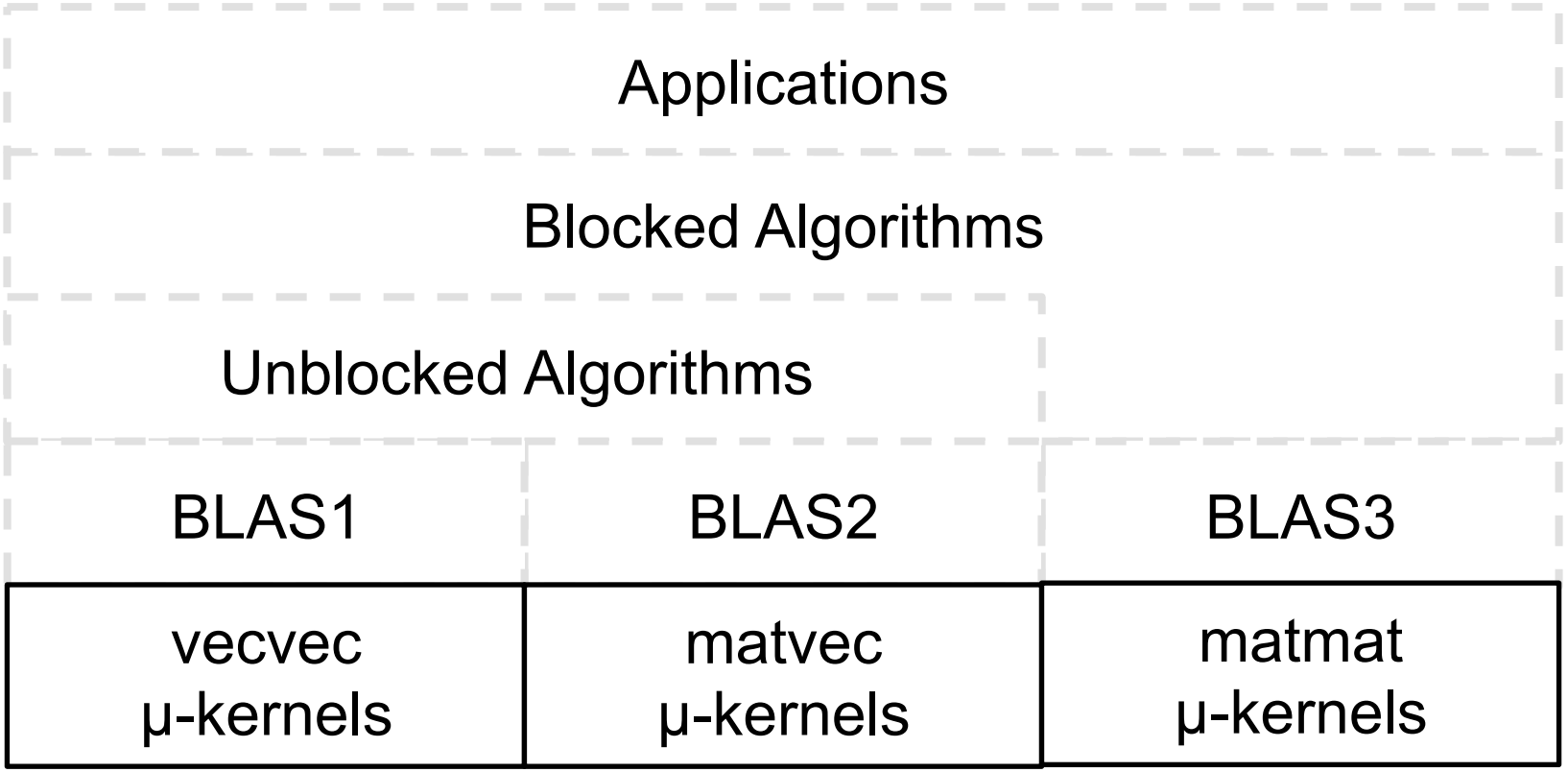


T. M. Smith, R. van de Geijn, M. Smelyanskiy, J. R. Hammond, and F. G. Van Zee. Anatomy of High-Performance Many-Threaded Matrix Multiplication. IPDPS 2014.

Relayering


MATLAB/Applications
LAPACK Blocked Algorithms
LAPACK Unblocked Algorithms
BLAS1
BLAS2
BLAS3
vecvec
µ-kernels
matvec
µ-kernels
matmat
µ-kernels

Relayering


Applications
Blocked Algorithms
Unblocked Algorithms
BLAS1
BLAS2
BLAS3
vecvec
µ-kernels
matvec
µ-kernels
matmat
µ-kernels

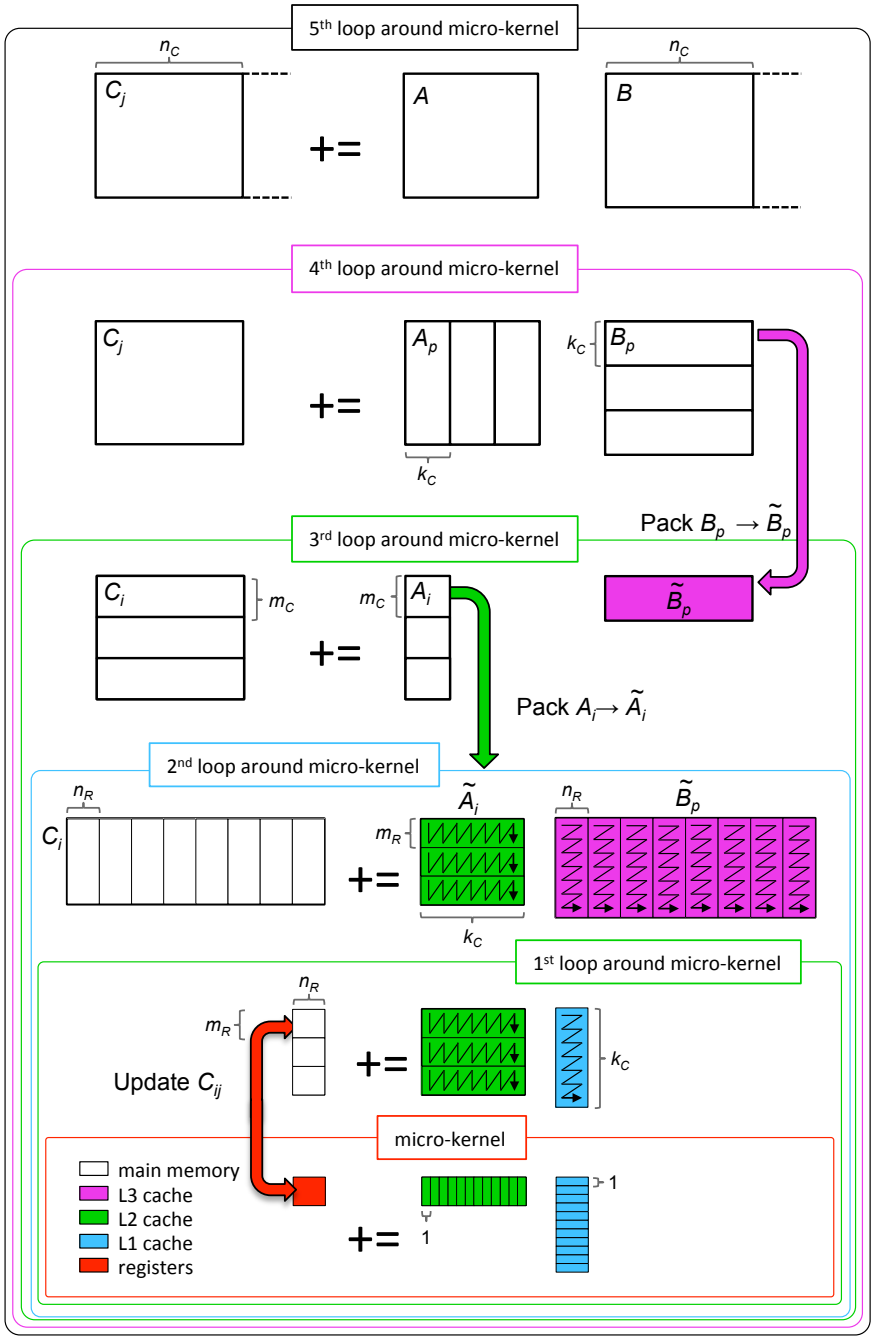
BLIS primitives as building blocks

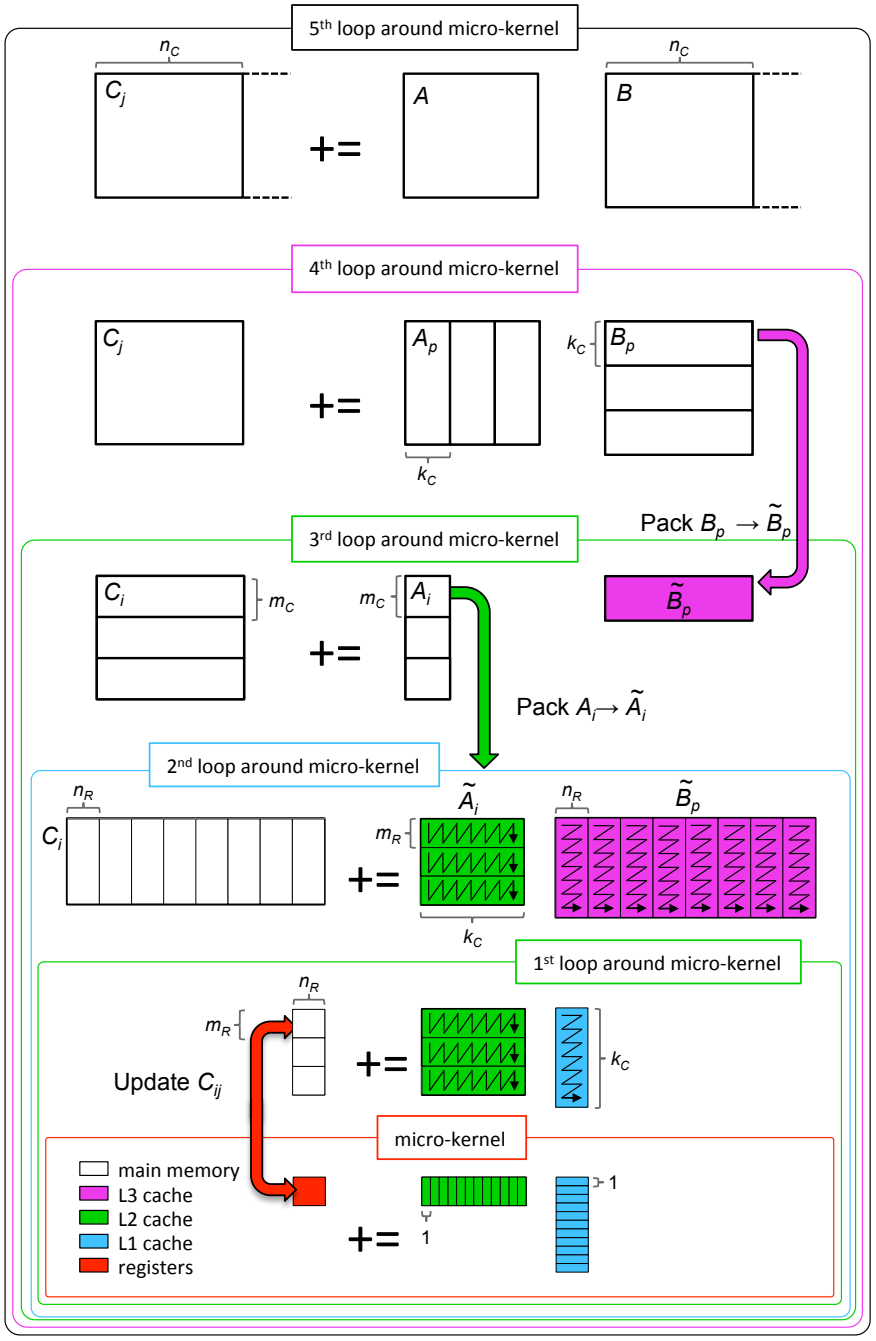
- Strassen

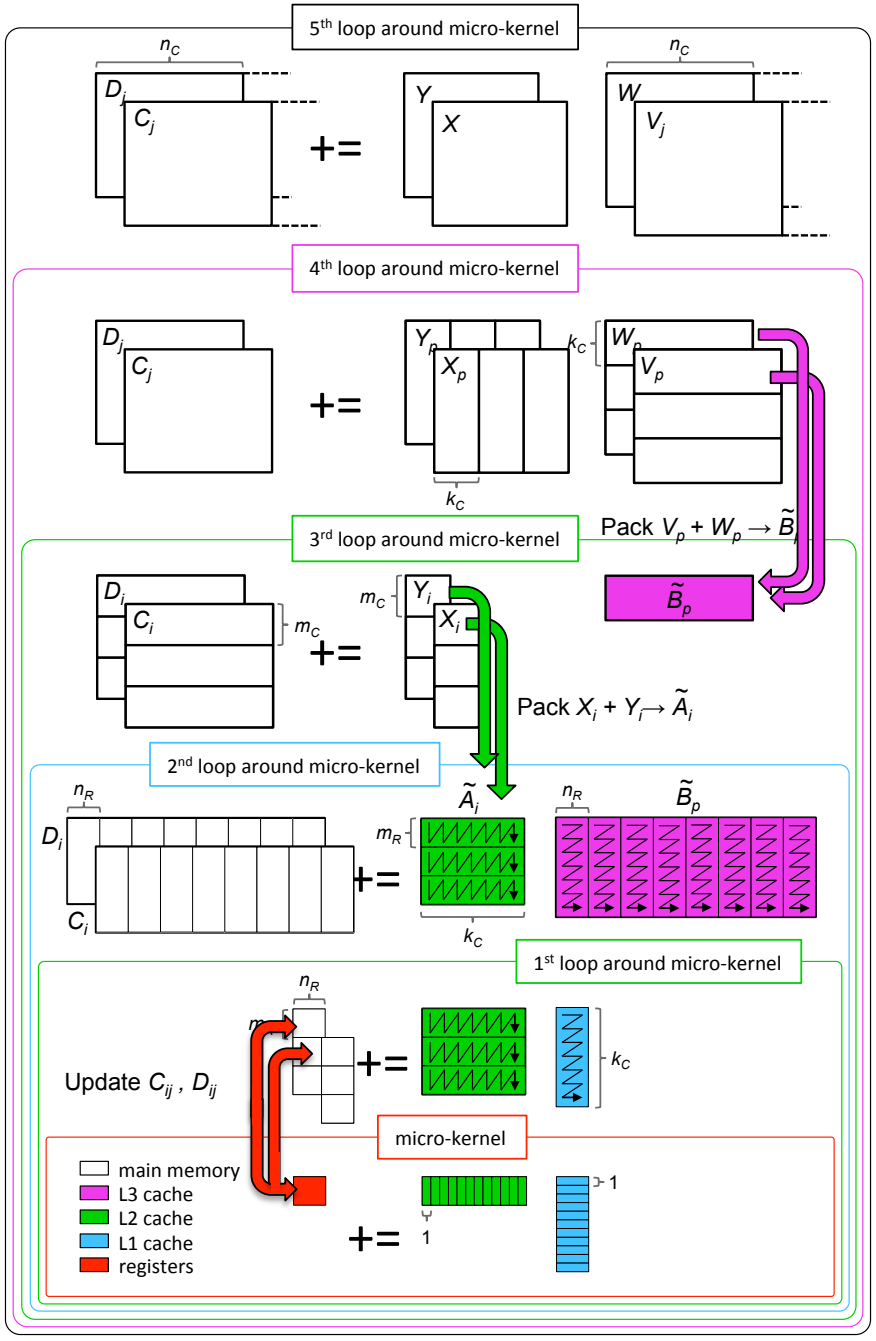
$$C = \begin{pmatrix} C_{TL} & C_{TR} \\ C_{BL} & C_{BR} \end{pmatrix}, A = \begin{pmatrix} A_{TL} & A_{TR} \\ A_{BL} & A_{BR} \end{pmatrix}, \text{ and } B = \begin{pmatrix} B_{TL} & B_{TR} \\ B_{BL} & B_{BR} \end{pmatrix},$$

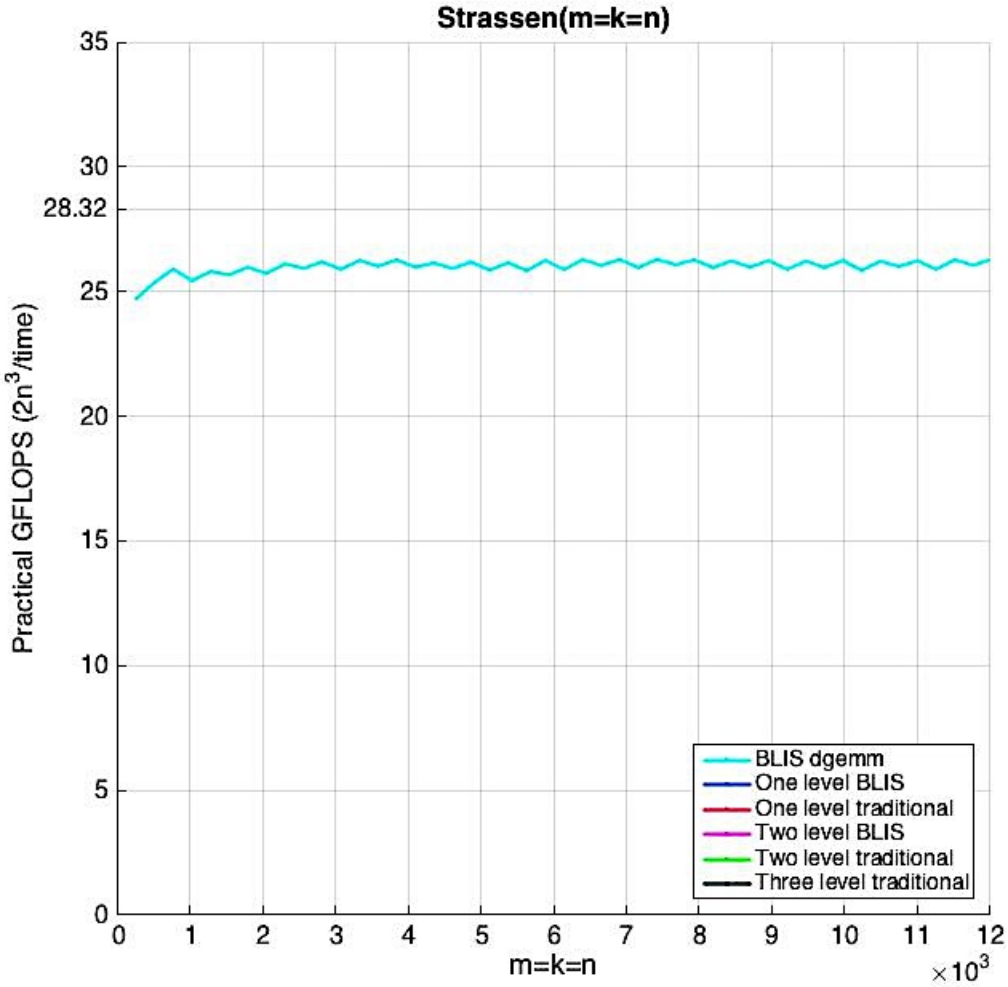
where C_{TL} is $\frac{m}{2} \times \frac{n}{2}$, A_{TL} is $\frac{m}{2} \times \frac{k}{2}$, and B_{TL} $\frac{k}{2} \times \frac{n}{2}$. Then it can be verified that

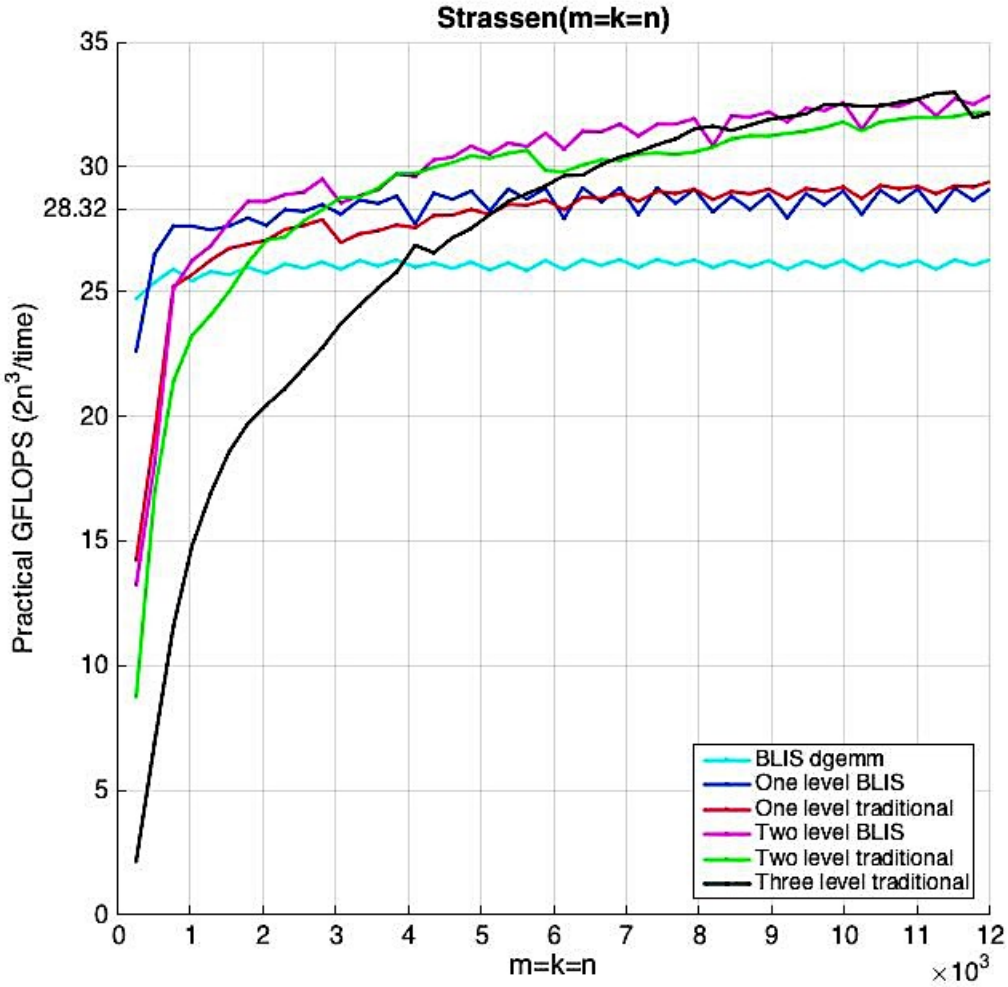
$$\begin{array}{lllll}
M_0 := (A_{TL} + A_{BR})(B_{TL} + B_{BR}); & C_{TL} := & \alpha M_0 + C_{TL}; & C_{BR} := & \alpha M_0 + C_{BR} \\
M_1 = (A_{BL} + A_{BR})B_{TL}; & C_{BL} := & \alpha M_1 + C_{BL}; & C_{BR} := & -\alpha M_1 + C_{BR} \\
M_2 = A_{TL}(B_{TR} - B_{BR}); & C_{TR} := & \alpha M_2 + C_{TR}; & C_{BR} := & \alpha M_2 + C_{BR} \\
M_3 = A_{BR}(B_{BL} - B_{TL}); & C_{TL} := & \alpha M_3 + C_{TL}; & C_{BL} := & \alpha M_3 + C_{BL} \\
M_4 = (A_{TL} + A_{TR})B_{BR}; & C_{TR} := & \alpha M_4 + C_{TR}; & C_{TL} := & -\alpha M_4 + C_{TL} \\
M_5 = (A_{BL} - A_{TL})(B_{TL} + B_{TR}); & & & C_{BR} := & \alpha M_5 + C_{BR} \\
M_6 = (A_{TR} - A_{BR})(B_{BL} + B_{BR}); & & & C_{TL} := & \alpha M_6 + C_{TL}
\end{array}$$


5th loop around micro-kernel
n_C
C_j
+=
A
B
n_C
4th loop around micro-kernel
C_j
+=
A_p
k_C
B_p
k_C
Pack $B_p \rightarrow \tilde{B}_p$
3rd loop around micro-kernel
C_i
m_C
+=
m_C
A_i
$\tilde{B}_p$
Pack $A_i \rightarrow \tilde{A}_i$
2nd loop around micro-kernel
n_R
C_i
+=
$\tilde{A}_j$
m_R
k_C
n_R
$\tilde{B}_p$
1st loop around micro-kernel
n_R
m_R
Update C_{ij}
+=
k_C
micro-kernel
main memory
L3 cache
L2 cache
L1 cache
registers
+=
1
1

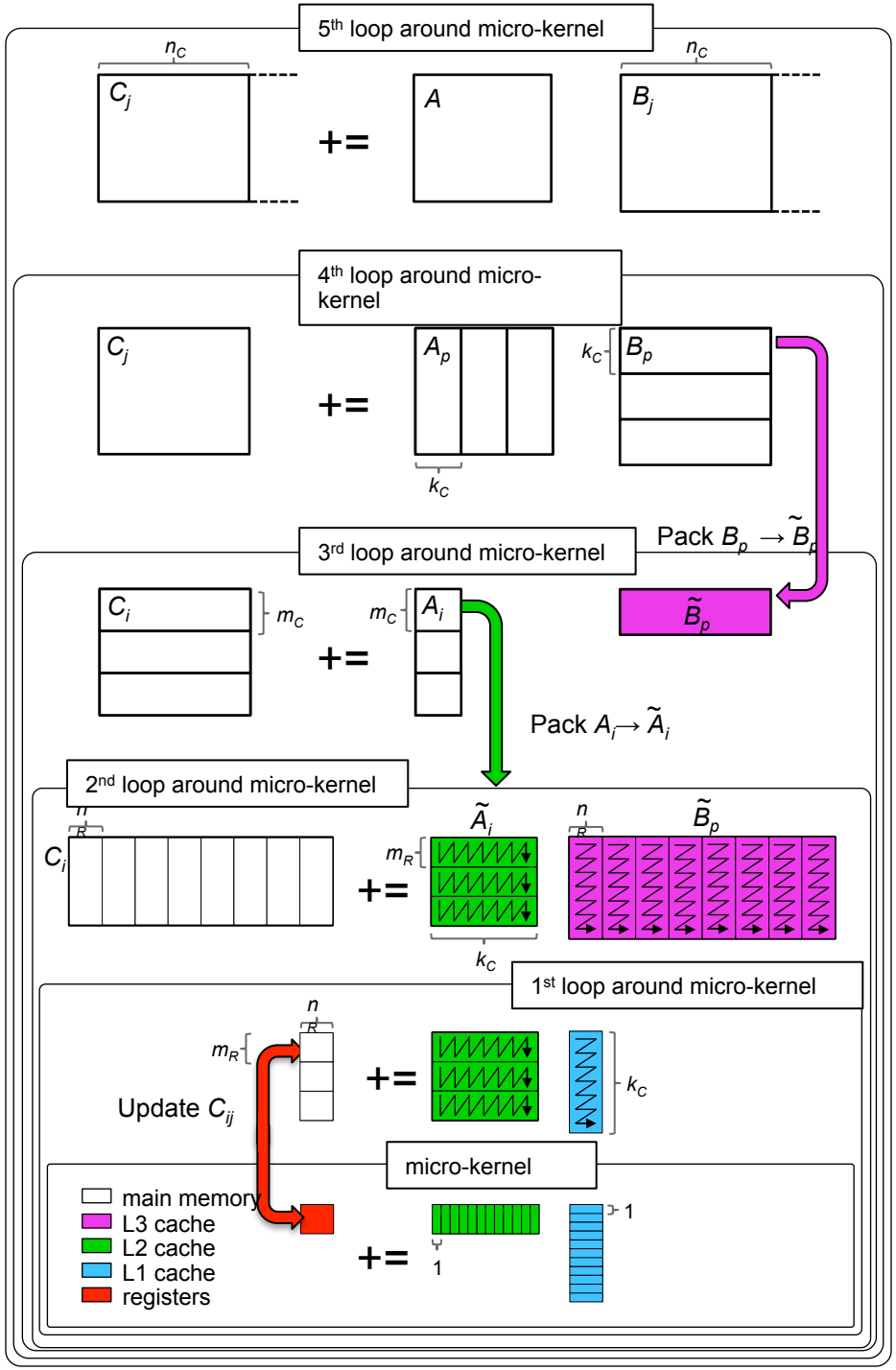

5th loop around micro-kernel
4th loop around micro-kernel
3rd loop around micro-kernel
2nd loop around micro-kernel
1st loop around micro-kernel
micro-kernel
Pack $B_p \rightarrow \tilde{B}_p$
Pack $A_i \rightarrow \tilde{A}_i$
Update C_{ij}
main memory
L3 cache
L2 cache
L1 cache
registers

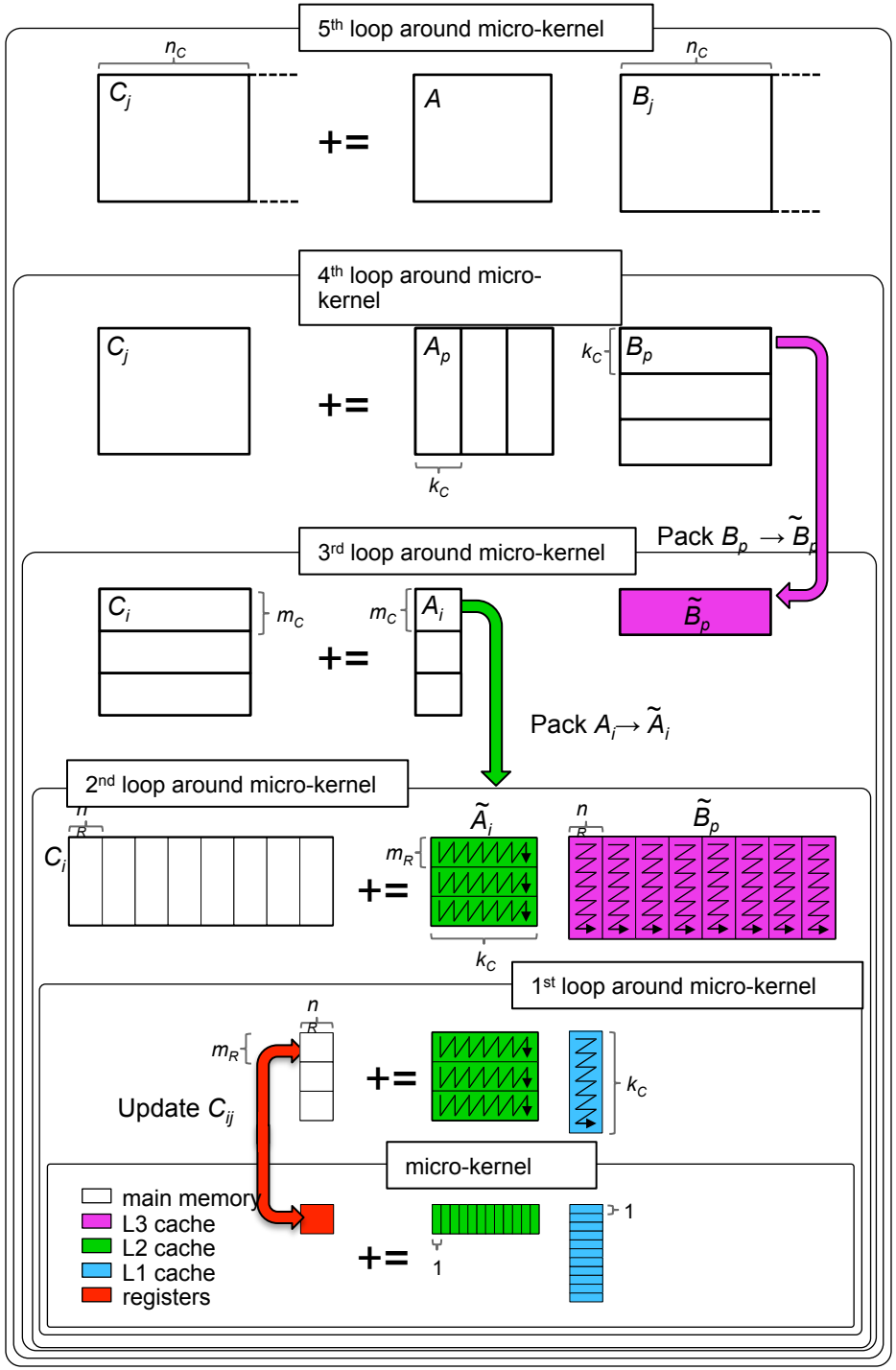

5th loop around micro-kernel
4th loop around micro-kernel
3rd loop around micro-kernel
2nd loop around micro-kernel
1st loop around micro-kernel
micro-kernel
Pack $V_p + W_p \rightarrow \tilde{B}_p$
Pack $X_i + Y_i \rightarrow \tilde{A}_i$
Update C_{ij} , D_{ij}
main memory
L3 cache
L2 cache
L1 cache
registers


Strassen(m=k=n)
Practical GFLOPS (2n^3/time)
35
30
28.32
25
20
15
10
5
0
0
1
2
3
4
5
6
7
8
9
10
11
12
m=k=n
×10^3
BLIS dgemm
One level BLIS
One level traditional
Two level BLIS
Two level traditional
Three level traditional



J. Huang, T. Smith, G. Henry, R. van de Geijn. Strassen's Algorithm Restarted SC'16. Numerical Algorithms, Part II. Wednesday 3:30pm-5pm. Room: 355-E


5th loop around micro-kernel
n_C
C_j
+=
A
B_j
n_C
4th loop around micro-kernel
C_j
+=
A_p
k_C
k_C
B_p
Pack $B_p \rightarrow \tilde{B}_p$
3rd loop around micro-kernel
C_i
m_C
+=
m_C
A_i
$\tilde{B}_p$
Pack $A_i \rightarrow \tilde{A}_i$
2nd loop around micro-kernel
n_R
C_i
+=
m_R
$\tilde{A}_i$
k_C
n_R
$\tilde{B}_p$
1st loop around micro-kernel
n_R
m_R
Update C_{ij}
+=
k_C
micro-kernel
main memory
L3 cache
L2 cache
L1 cache
registers
+=
1
1




```
using GotoGEMM =
partition_gemm_nc<
  partition_gemm_kc<
    pack_b<BuffersForB,
      partition_gemm_mc<
        pack_a<BuffersForA,
          partition_gemm_nr<
            partition_gemm_mr<
              gemm_micro_kernel
>>>>>>>;
```

```
using GotoGEMM =
partition_gemm_nc<

 partition_gemm_kc<

   pack_b<BuffersForB,

     partition_gemm_mc<

       pack_a<BuffersForA,

         partition_gemm_nr<

           partition_gemm_mr<

             gemm_micro_kernel

>>>>>>>;
```

Tensor contractions are matrix-matrix multiplications

```
using GotoGEMM =
partition_gemm_nc<
 partition_gemm_kc<
   pack_b<BuffersForB,
     partition_gemm_mc<
       pack_a<BuffersForA,
         partition_gemm_nr<
           partition_gemm_mr<
             gemm_micro_kernel
>>>>>>>;
```

```
Using TensorGEMM =
partition_gemm_nc<
 partition_gemm_kc<
   pack_b<BuffersForB,
     partition_gemm_mc<
       pack_a<BuffersForA,
         partition_gemm_nr<
           partition_gemm_mr<
             gemm_micro_kernel
>>>>>>>;
```

Devin Matthews,
"High-Performance Tensor Contraction without BLAS",
Research Poster #35

```
template <typename T, int Variant, uplo_t Uplo>

void cholesky(matrix<T>& A)
{
    partition_2x2<T, TL_TO_BR    > A_2x2(A);
    partition_3x3<T, TL_TO_BR, BS> A_3x3(A);

    while (A_2x2.m_TL < A.m)
    {
        A_3x3.repartition_down(A_2x2);

        auto& A11 = A_3x3.A11();
        auto& A21 = A_3x3.A21();
        auto& A22 = A_3x3.A22();

        // A11 = chol( A11 )
        cholesky<T, 3, LOWER>(A11);

        // A21 = A21 * inv( tril( A11 )' )
        trsm<T, RIGHT, LOWER, ADJOINT, NONUNIT>(T(1), A11, A21);

        // A22 = A22 - A21 * A21'
        herk<T, LOWER, NORMAL>(T(-1), A21, T(1), A22);

        A_2x2.continue_with(A_3x3);
    }
}
```

Related Publications

- J. A. Gunnels, G. M. Henry, and R. A. van de Geijn. A Family of High-Performance Matrix Algorithms. In *Computational Science - 2001*.
- Kazushige Goto, Robert A. van de Geijn. Anatomy of high-performance matrix multiplication. ACM Transactions on Mathematical Software (TOMS), 2008.
- Kazushige Goto, Robert van de Geijn. High-performance implementation of the level-3 BLAS. ACM Transactions on Mathematical Software (TOMS), 2008
- Field G. Van Zee, Robert A. van de Geijn. BLIS: A Framework for Rapidly Instantiating BLAS Functionality. ACM Transactions on Mathematical Software (TOMS), 2015
- Tyler M. Smith, Robert van de Geijn, Mikhail Smelyanskiy, Jeff R. Hammond, and Field G. Van Zee. Anatomy of High-Performance Many-Threaded Matrix Multiplication. IPDPS 2014.
- Field G. Van Zee, Tyler Smith, Bryan Marker, Tze Meng Low, Robert A. van de Geijn, Francisco D. Igual, Mikhail Smelyanskiy, Xianyi Zhang, Michael Kistler, Vernon Austel, John Gunnels, Lee Killough. The BLIS Framework: Experiments in Portability. ACM Transactions on Mathematical Software. 2016
- Devin A. Matthews. High-Performance Tensor Contraction without BLAS. Submitted to SISC.
- J. Huang, T. Smith, G. Henry, R. van de Geijn. Strassen's Algorithm Restarted. SC'16.
- …

Insights

- BLIS: Careful refactoring of GotoBLAS
- Less code to optimize
- More loops to parallelize
- Provides building blocks for innovative optimization
- **Enables more flexible DSL for DLA than BLAS**

Overview

- Reflect
- Abstract
- Derive
- Encode
- Layer
- **Build**
- Preach

New DLA Software Stack

- Funded by NSF's Software Infrastructure for Sustained Innovation program and gifts from industry.
  - Functionality: BLAS and LAPACK
  - DSL being designed

Software

High performance dense linear algebra libraries, each addressing a layer in the linear algebra software stack, have been developed by the team and our collaborators from both academia and industry.

BLIS

BLIS is a software framework for instantiating high-performance BLAS-like dense linear algebra libraries. BLIS is written in Standard C (mostly ISO C90 with a few C99 extensions) and available under a new/modified/3-clause BSD license.

Get it here

libFLAME

libFLAME is a high performance dense linaer algebra library that is the result of the FLAME methodology for systematically developing dense linear algebra libraries. The FLAME methodology is radically different from the LINPACK/LAPACK approach that dates back to the 1970s.

libFLAME is LAPACK compatible and includes all of LAPACK's functionality.

Get it here

Elemental

Elemental is a distributed-memory library for dense and sparse-direct linear algebra and optimization.

Get it here

ROTE

ROTE is a library for tensor contractions targeting distributed memory architectures. ROTE is written in ... and available under a new/modified/3-clause BSD license.

Get it here

CFOUR

Get it here

DxTer

DxTer is a domain-agnostic tool for generating high-performance code for an input specification using a knowledge base that encodes information about a particular software domain. DxTer has been used to generate code for BLIS, Elemental, and ROTE.

Get it here

Overview

- Reflect
- Abstract
- Derive
- Encode
- Layer
- Build
- **Preach**

Building a user/developer community

- Usual means:
  - Software/Publications/Talks/Workshops
- Pedagogical outreach
  - **BLISlab:** Classroom activity based on BLIS matrix-matrix multiplication
  - **Linear Algebra: Foundations to Frontiers (LAFF)**
    - MOOC on edX
  - **LAFF-On Programming for Correctness**
    - MOOC on edX (April 2017)
  - **LAFF-On Programming for Performance**
    - MOOC based on BLISlab (maybe…)

LAFF–On Programming for Correctness

https://www.youtube.com/watch?v=1B0PPMNQsP4

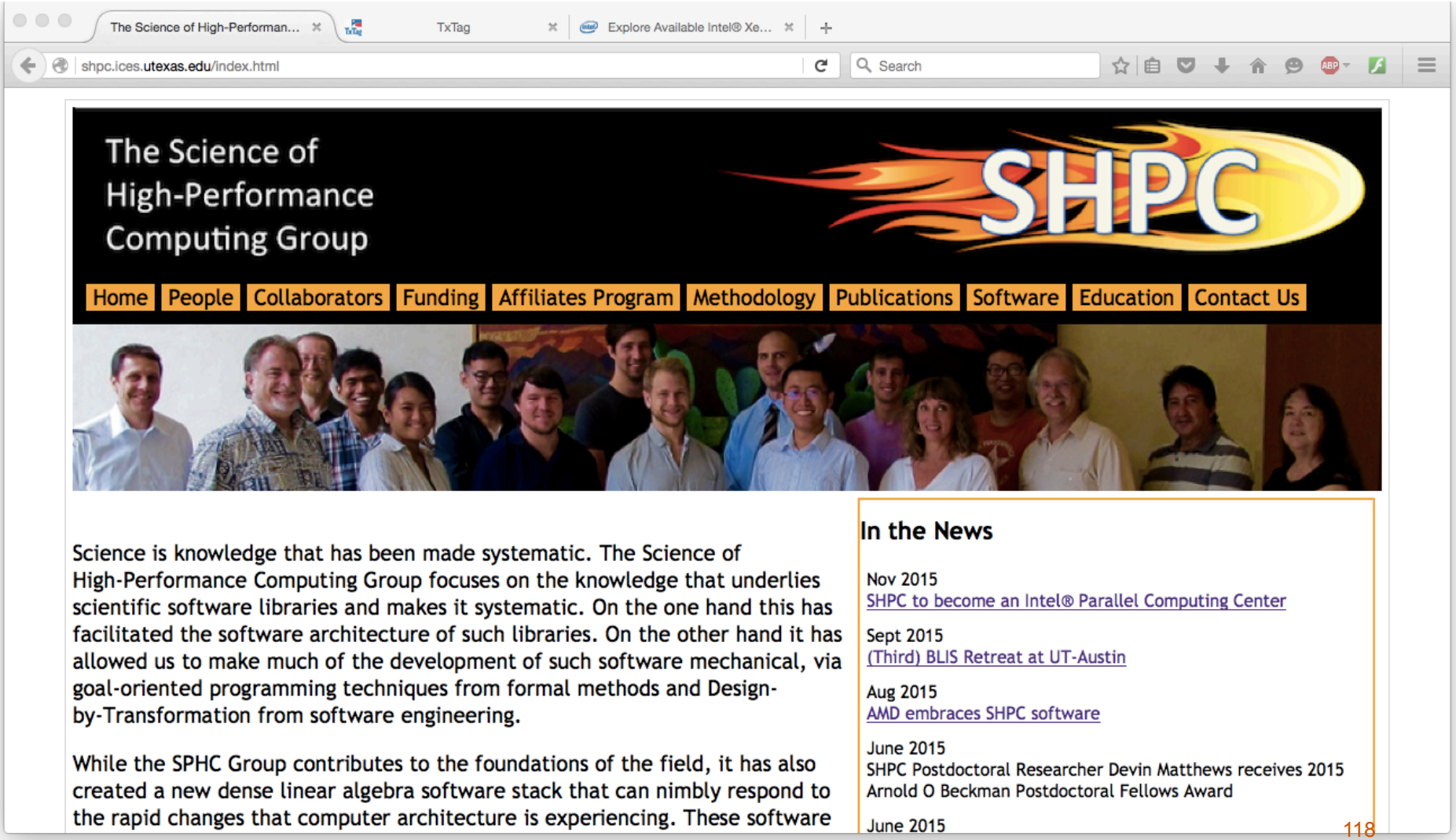
Coming in Spring 2017 to edX.org

Conclusion

- A Domain Specific Language starts with Domain Specific Notation.
- Notation supports systematic reasoning.
- The DSL facilitates code that closely resembles algorithms.
- Proper layering enables optimization across layers.

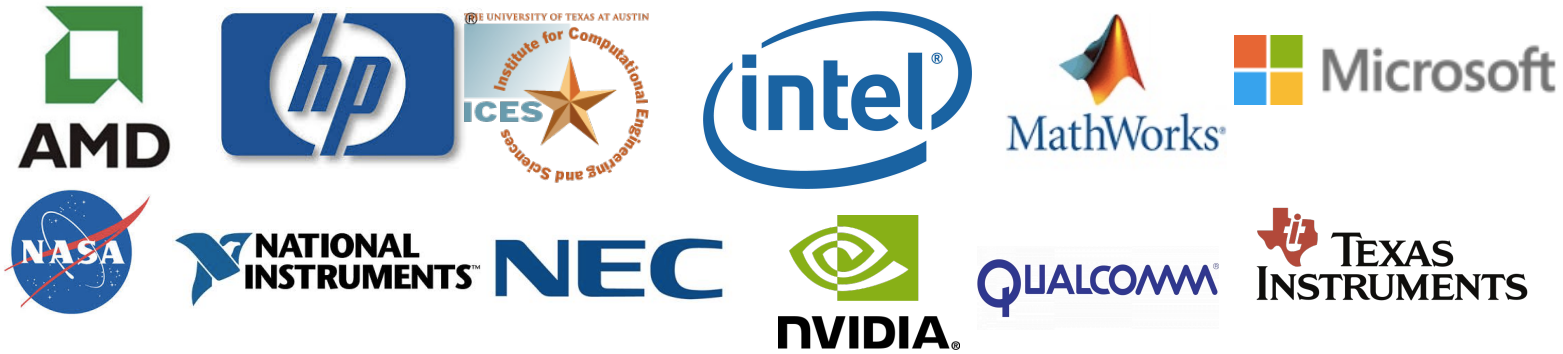
“Elegance is not a dispensable luxury but a quality that decides between success and failure.” -- Edsger W. Dijkstra

The Team


The Science of High-Performan...
TxTag
Explore Available Intel® Xe...
shpc.ices.utexas.edu/index.html
Search
The Science of High-Performance Computing Group
SHPC
Home
People
Collaborators
Funding
Affiliates Program
Methodology
Publications
Software
Education
Contact Us
Science is knowledge that has been made systematic. The Science of High-Performance Computing Group focuses on the knowledge that underlies scientific software libraries and makes it systematic. On the one hand this has facilitated the software architecture of such libraries. On the other hand it has allowed us to make much of the development of such software mechanical, via goal-oriented programming techniques from formal methods and Design-by-Transformation from software engineering.
While the SPHC Group contributes to the foundations of the field, it has also created a new dense linear algebra software stack that can nimbly respond to the rapid changes that computer architecture is experiencing. These software
In the News
Nov 2015
SHPC to become an Intel® Parallel Computing Center
Sept 2015
(Third) BLIS Retreat at UT-Austin
Aug 2015
AMD embraces SHPC software
June 2015
SHPC Postdoctoral Researcher Devin Matthews receives 2015 Arnold O Beckman Postdoctoral Fellows Award
June 2015

Funding

- NSF
  - Award ACI-1550493: **Sustaining Innovation in the Linear Algebra Software Stack for Computational Chemistry and other Sciences.** 2016 - 2018
  - Award ACI-1148125: **A Linear Algebra Software Infrastructure for Sustained Innovation in Computational Chemistry and other Sciences.** 2012 - 2016
  - Many other NSF grants 2002 - present



More information

- http://shpc.ices.utexas.edu
- https://github.com/flame/
- https://github.com/flame/blis/
- https://github.com/flame/libflame/
- https://github.com/devinamatthews/tblis