Assignments

Operation	Comment	Call	Team
Example			
$B := LB$	Lower triangular matrix L	`Trmm_llnn( L, B )`	Robert vdG
Symmetric matrix-matrix multiplication			
$C := AB + C$	A symmetric, stored in lower-triangular part	`Symm_ll( A, B, C )`	Ben Nguyen, Andrew Dong-Tran, Jeffrey Leung
$C := AB + C$	A symmetric, stored in upper-triangular part	`Symm_lu( A, B, C )`	Wesley Chung, Hiep Vu, David Shi
$C := BA + C$	A symmetric, stored in lower-triangular part	`Symm_rl( A, B, C )`	Jeff Taube and Justin Salazar and Darya Mylius
$C := BA + C$	A symmetric, stored in upper-triangular part	`Symm_ru( A, B, C )`	Liangkun Zhao, Yajie Niu
Symmetric rank-k update			
$C := AA^T + C$	C symmetric, stored in lower-triangular part	`Syrk_ln( A, C )`	Jorge Munoz, Max Svetlik
$C := AA^T + C$	C symmetric, stored in upper-triangular part	`Syrk_ut( A, C )`	
$C := A^T A + C$	C symmetric, stored in lower-triangular part	`Syrk_ln( A, C )`	
$C := A^T A + C$	C symmetric, stored in upper-triangular part	`Syrk_ut( A, C )`	

Operation	Comment	Team	
Symmetric rank-2k update			
$C := AB^T + BA^T + C$	C symmetric, stored in lower-triangular part	`Syr2k_ln( A, B, C )`	Ryan Young and Benny Renard
$C := AB^T + BA^T + C$	C symmetric, stored in upper-triangular part	`Syr2k_un( A, B, C )`	Scott Munro and Matthew Chin
$C := A^T B + B^T A + C$	C symmetric, stored in lower-triangular part	`Syr2k_lt( A, B, C )`	Raeeca Narimani and Allison Wallpole
$C := A^T B + B^T A + C$	C symmetric, stored in upper-triangular part	`Syr2k_ut( A, B, C )`	Pushkar and Sarah
Triangular matrix-matrix multiplication			
$B := L^T B$	L stored in lower triangle	`Trmm_lltn( A, B )`	Sean Wang and Marco Guajardo, Tim, Vincent
$B := UB$	U stored in upper triangle	`Trmm_lunn( A, B )`	
$B := U^T B$	U stored in upper triangle	`Trmm_lutn( A, B )`	Chris Getz, Bradley Holloway, Rohit Pattanaik
$B := BL$	L stored in lower triangle	`Trmm_rlnn( A, B )`	Ben Yang, Andras Balogh, Eric Lee
$B := BL^T$	L stored in lower triangle	`Trmm_rltn( A, B )`	Amanda Nguyen, Tim Kwan
$B := BU$	U stored in upper triangle	`Trmm_runn( A, B )`	
$B := BU^T$	U stored in upper triangle	`Trmm_rutn( A, B )`	
Triangular-triangular matrix multiplication			
$B := LU$	L lower triangular, U upper triangular	`Trtrmm_lunn( L, U, B )`	
$B := UL$	L lower triangular, U upper triangular	`Trtrmm_ulnn( U, L, B )`	
$B := U^T L$	L lower triangular, U upper triangular	`Trtrmm_ultn( U, L, B )`	
$B := UL^T$	L lower triangular, U upper triangular	`Trtrmm_ulnt( U, L, B )`	

Part I

Triangular Matrix-matrix Multiplication

Chapter 1

Cases

Triangular matrix-matrix multiplication (TRMM) is matrix-matrix multiplication where one of the matrices is square and (lower or upper) triangular.

The full set of triangular matrix-matrix multiplication cases is denoted by TRMM_□□□□, where the letters in the four boxes denote whether the triangular matrix is on the Left or Right, is Lower or Upper triangular, is Not transposed or Transposed, and has a Nonunit or Unit diagonal:

	LL□□	LU□□	RL□□	RU□□
□□N□	$B = LB$	$B = UB$	$B = BL$	$B = BU$
□□T□	$B = L^T B$	$B = U^T B$	$B = BL^T$	$B = BU^T$

Chapter 2

$B := LB$ – Team: Robert van de Geijn

2.1 Operation

Consider the operation

$$B := LB$$

where L is a $m \times m$ lower triangular matrix and B is a $m \times n$ matrix. This is a special case of triangular matrix-matrix multiplication, with the LOWER triangular matrix on the LEFT, and the triangular matrix is NOT transposed. We will refer to this operation as TRMM_LLNN where the LLNN stands for left lower no-transpose nonunit diagonal. The nonunit diagonal means we will use the entries of the matrix that are stored on the diagonal.

2.2 Precondition and postcondition

In the precondition

$$B = \widehat{B}$$

$\widehat{B}$ denotes the original contents of B. This allows us to express the state upon completion, the postcondition, as

$$B = L\widehat{B}.$$

2.3 Partitioned Matrix Expressions and loop invariants

There are two PMEs for this operation.

2.3.1 PME 1

To derive the second PME, partition

$$L \rightarrow \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array} \right), \quad \text{and} \quad B \rightarrow \left(\begin{array}{c} B_T \\ \hline B_B \end{array} \right).$$

Substituting these into the postcondition yields

$$\left(\begin{array}{c} B_T \\ \hline B_B \end{array}\right) = \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \left(\begin{array}{c} \widehat{B}_T \\ \hline \widehat{B}_B \end{array}\right)$$

or, equivalently,

$$\left(\begin{array}{c} B_T \\ \hline B_B \end{array}\right) = \left(\begin{array}{c} L_{TL}\widehat{B}_T \\ \hline L_{BL}\widehat{B}_T + L_{BR}\widehat{B}_B \end{array}\right)$$

so that, upon completion

$$\begin{array}{c} B_T = L_{TL}\widehat{B}_T \\ \hline B_B = L_{BL}\widehat{B}_T + L_{BR}\widehat{B}_B \end{array}$$

From this, we can choose two loop invariants:

Invariant 1: $\left(\begin{array}{c} B_T = \widehat{B}_T \\ \hline B_B = L_{BR}\widehat{B}_B \end{array}\right)$. (The top part has been left alone and the bottom part has been partially computed).

Invariant 2: $\left(\begin{array}{c} B_T = \widehat{B}_T \\ \hline B_B = L_{BL}\widehat{B}_T + L_{BR}\widehat{B}_B \end{array}\right)$. (The top part has been left alone and the bottom part has been completely computed).

2.3.2 PME 2

To derive the second PME, partition

$$B \rightarrow \left(\begin{array}{c|c} B_L & B_R \end{array}\right)$$

and does not partition L. Substituting these into the postcondition yields

$$\left(\begin{array}{c|c} B_L & B_R \end{array}\right) = L\left(\begin{array}{c|c} \widehat{B}_L & \widehat{B}_R \end{array}\right)$$

or, equivalently,

$$\left(\begin{array}{c|c} B_L & B_R \end{array}\right) = \left(\begin{array}{c|c} L\widehat{B}_L & L\widehat{B}_R \end{array}\right)$$

so that, upon completion

$$\begin{array}{c|c} B_L = L\widehat{B}_L & B_R = L\widehat{B}_R \end{array}$$

From this, we can choose two more loop invariants:

Invariant 3: $\left(\begin{array}{c|c} B_L = L\widehat{B}_L & B_R = \widehat{B}_R \end{array}\right)$. (The left part has been completely finished and the right part has been left untouched).

Invariant 4: $\left(\begin{array}{c|c} B_L = \widehat{B}_L & B_R = L\widehat{B}_R \end{array}\right)$. (The left part has been completely finished and the right part has been left untouched).

2.3.3 Notes

How do I decide to partition the matrices in the postcondition?

- Pick a matrix (operand), any matrix.
- If that matrix has
  - a triangular structure (in storage), then you want to either partition is into four quadrants, or not at all. Symmetric matrices and triangular matrices have a triangular structure (in storage).
  - no particular structure, then you partition it vertically (left-right), horizontally (top-bottom), or not at all.
- Next, partition the other matrices similarly, but conformally (meaning the resulting multiplications with the parts are legal).

Take our problem here: $B := LB$. Start by partitioning L in to quadrants:

$$B = \left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \widehat{B}.$$

Now, the way partitioned matrix multiplication works, this doesn't make sense:

$$B = \underbrace{\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \widehat{B}.}_{\left(\begin{array}{c} L_{TL} \times \text{something} + 0 \times \text{something} \\ \hline L_{BL} \times \text{something} + L_{BR} \times \text{something} \end{array}\right)} \quad .$$

So, we need to also partition B into a top part and a bottom part:

$$\left(\begin{array}{c} B_T \\ \hline B_B \end{array}\right) = \underbrace{\left(\begin{array}{c|c} L_{TL} & 0 \\ \hline L_{BL} & L_{BR} \end{array}\right) \left(\begin{array}{c} \widehat{B}_T \\ \hline \widehat{B}_B \end{array}\right)}_{\left(\begin{array}{c} L_{TL}B_T + B_B \\ \hline L_{BL}B_T + L_{BR}B_B \end{array}\right)} . \; .$$

Alternatively, what if you don't partition L? You have to partition *something* so let's try partitioning B:

$$\left(\begin{array}{c} B_T \\ \hline B_B \end{array}\right) = L \left(\begin{array}{c} \widehat{B}_T \\ \hline \widehat{B}_B \end{array}\right)$$

But that doesn't work... Instead

$$\left(\begin{array}{c|c} B_L & B_R \end{array}\right) = L \left(\begin{array}{c|c} \widehat{B}_L & \widehat{B}_R \end{array}\right) = \left(\begin{array}{c|c} L\widehat{B}_L & L\widehat{B}_R \end{array}\right)$$

works just fine.

Part II

Symmetric Matrix-matrix Multiplication

Chapter 3

Cases

Symmetric matrix-matrix multiplication (SYMM) is matrix-matrix multiplication where one of the matrices is square and symmetric, stored in the lower or upper triangular part of the matrix.

The full set of symmetric matrix-matrix multiplication cases is denoted by SYMM_□□, where the letters in the two boxes denote whether the symmetric matrix is on the Left or Right, and is stored in the Lower or Upper triangular part of that matrix:

	L□	R□	
□L	$C := AB + C$	$C := BA + C$	A symmetric, stored in lower triangle
□U	$C := AB + C$	$C := BA + C$	A symmetric, stored in upper triangle

Part III

Symmetric Rank-k Update

Chapter 4

Cases

Symmetric rank-k update (SYRK) is matrix-matrix multiplication of a matrix with its transpose, where the matrix C being updated is symmetric, stored in the lower or upper triangular part of the matrix.

The full set of symmetric rank-k cases is denoted by SYRK_□□, where the letters in the two boxes denote whether the matrices begin multiplied are Not transposed or Trankposed, and whether the matrix being updated is stored in the Lower or Upper triangular part:

	N□	T□	
□L	$C := AA^T + C$	$C := A^TA + C$	C symmetric, stored in lower triangle
□U	$C := AA^T + C$	$C := A^TA + C$	C symmetric, stored in upper triangle

Part IV

Symmetric Rank-2k Update

Chapter 5

Cases

Symmetric rank-2k update (SYR2K) is matrix-matrix multiplication of two matrices where both the result and the transpose of that result are added to a symmetric matrix C, stored in the lower or upper triangular part of the matrix.

The full set of symmetric rank-2k cases is denoted by SYR2K_□□, where the letters in the two boxes denote whether the matrices begin multiplied are Not transposed or Transposed, and whether the matrix being updated is stored in the Lower or Upper triangular part:

	N□	T□	
□L	$C := AB^T + BA^T + C$	$C := A^T B + B^T A + C$	C symmetric, stored in lower triangle
□U	$C := AB^T BA^T + C$	$C := A^T B + B^T A + C$	C symmetric, stored in upper triangle

Part V

Triangular Solve with Multiple Right-hand Sides

Chapter 6

Cases

Triangular solve with multiple right-hand sides (TRSM) is a matrix-matrix multiplication in disguise: Not only is one of the matrices square and (lower or upper) triangular, but in addition, one multiplies with the inverse of the matrix.

The full set of TRSM cases is denoted by TRSM_□□□□, where the letters in the four boxes denote whether the triangular matrix is on the Left or Right, is Lower or Upper triangular, is Not transposed or Transposed, and has a Nonunit or Unit diagonal:

	LL□□	LU□□	RL□□	RU□□
□□N□	$B := L^{-1}B$	$B := U-1B$	$B := BL^{-1}$	$B := BU^{-1}$
□□T□	$B := L^{-T}B$	$B :: U^{-T}B$	$B := BL^{-T}$	$B := BU^{-T}$

We will only consider the case where we compute with the diagonal.

Where does the name *triangular solve with multiple right-hand sides* come from? If one wants to compute $B := A^{-1}B$ one can also think of this as solving $AX = B$, overwriting the matrix A with the solution matrix X. Now, partition X and B by columns. Then

$$\underbrace{A\left(\begin{array}{c|c|c|c} x_0 & x_1 & \cdots & x_{n-1}\end{array}\right)}_{\left(\begin{array}{c|c|c|c} Ax_0 & Ax_1 & \cdots & Ax_{n-1}\end{array}\right)} = \left(\begin{array}{c|c|c|c} b_0 & b_1 & \cdots & b_{n-1}\end{array}\right)$$

and hence, for each column j, $Ax_j = b_j$. This means that one can instead solve with matrix A, and because B has many columns, this becomes a *solve with multiple right-hand sides*. If A is triangular, then it is a *triangular solve with multiple right-hand sides*. Importantly, the inverse of the matrix is never computed.