The LTL^T factorization: A glimpse at the future

Robert van de Geijn

The Science of High-Performance Computing Group

with Maggie Myers, Devangi Parikh, Devin Matthews, RuQing Xu, Tze Meng Low, Ishna Satyarth, Chao Yin, and others

What is FLAME?

- A notation for representing algorithms
- A systematic method for deriving families of algorithms
- APIs for representing algorithms in code
- A library (libflame) with LAPACK-like functionality
- Educational materials
- A community
- …

Decades of progress (FLAME)

- PLAPACK, targeting distributed memory architectures

  R. van de Geijn. Using PLAPACK--parallel linear algebra package. MIT Press. 1997.

  Many papers

  Already proposed an API that inspired the BLIS object API.

- FLAME notation

  E.S. Quintana, G. Quintana, X. Sun, R. van de Geijn. A note on parallel matrix inversion. SISC. 2001.

  Many other papers

- FLAME Methodology and APIs

  J..A Gunnels, F.G. Gustavson, G.M. Henry, R.A. van de Geijn. FLAME: Formal linear algebra methods environment. TOMS. 2001.

  Many other papers

Decades of progress (FLAME)

- libflame

  F.G. Van Zee, E. Chan, R.A. van de Geijn, E.S. Quintana-Orti, G. Quintana-Orti. The libflame library for dense matrix computations. CISE 2009.

  Many other papers

- SuperMatrix

  G. Quintana-Ortí, E.S. Quintana-Ortí, R.A. van de Geijn, F.G. Van Zee, E. Chan. Programming matrix algorithms-by-blocks for thread-level parallelism. TOMS 2009.

  Many other papers

- LAFF-On Programming for Correctness (MOOC)

  Maggie Myers and Robert van de Geijn. LAFF-On Programming for Correctness. edX.

  .

Decades of progress (BLIS)

- Goto's algorithm

  K. Goto, R. van de Geijn. On reducing TLB misses in matrix multiplication. Technical Report TR02-55. 2002.

  K. Goto, R.A. van de Geijn. Anatomy of high-performance matrix multiplication. TOMS. 2008.

- BLIS

  F.G. Van Zee, R.A. van de Geijn. BLIS: A framework for rapidly instantiating BLAS functionality. TOMS 2015.

  Many other papers

- TBLIS

  D.A. Matthews. High-Performance Tensor Contraction without Transposition. SISC 2018.

- LAFF-On Programming for High Performance (MOOC)

  Robert van de Geijn, Maggie Myers, Devangi Parikh. LAFF-On Programming for High Performance. edX.

.

Back to the future

- Two decades ago, FLAME/libflame was the future.
- It still is.
- BLIS is our key to that future.

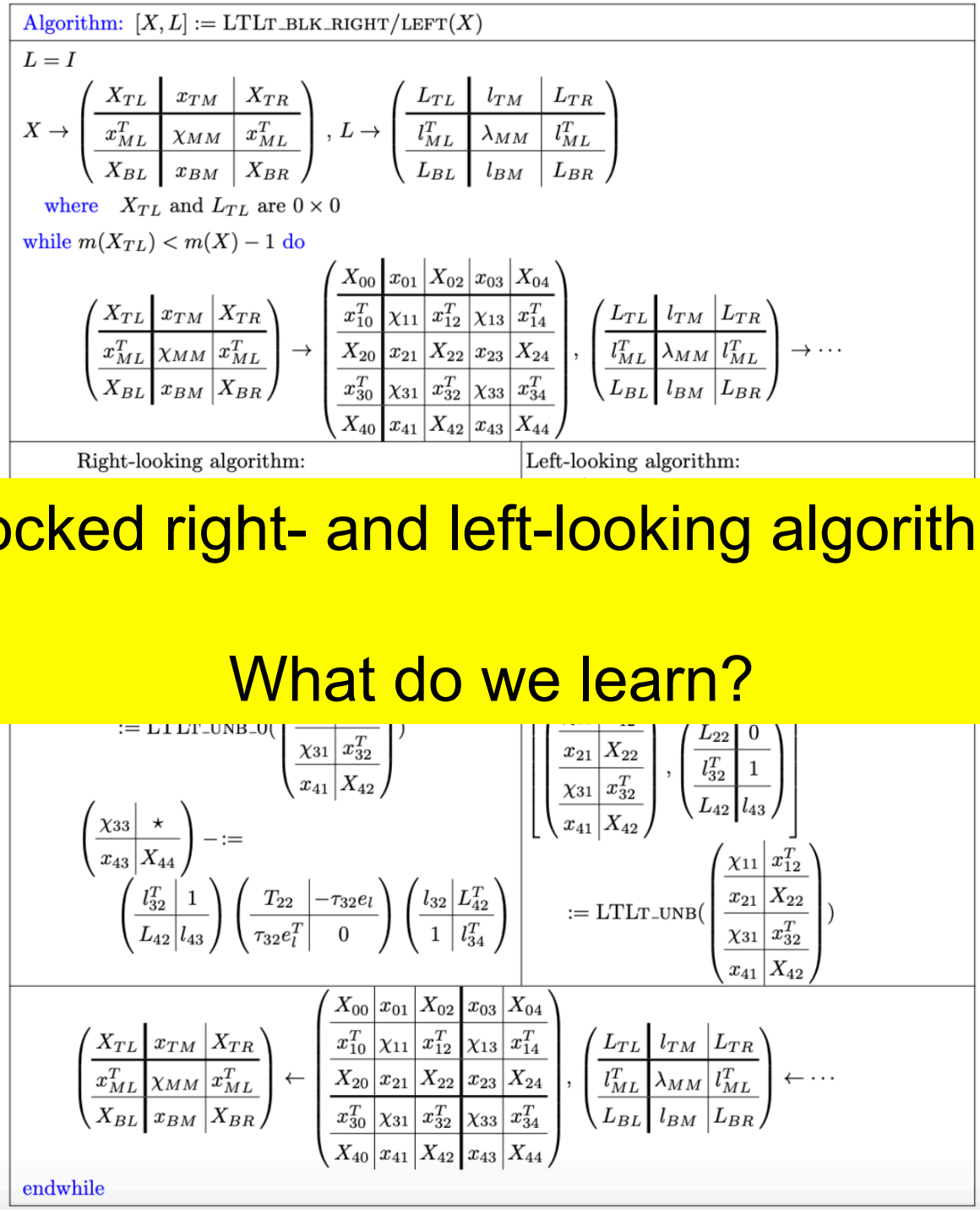
An example: LTL^T factorization

- Given: skew-symmetric matrix X
- Desired: $X = L\ T\ L^T$, where L is unit lower triangular, T is skew-symmetric tridiagonal.
- What if we apply the FLAME methodology?

Step	Algorithm: $[X, L] := \text{LTLT\_UNB\_RIGHT}(X)$
1a	$\left\{ X = \widehat{X} \wedge (\exists L, T \mid \widehat{X} = LTL^T) \right\}$
4	$L = I$ $X \rightarrow \left(\begin{array}{c\|c\|c} X_{TL} & x_{TM} & X_{TR} \\ \hline x_{ML}^T & \chi_{MM} & x_{MR}^T \\ \hline X_{BL} & x_{BM} & X_{BR} \end{array}\right)$, $L \rightarrow \cdots$, $T \rightarrow \cdots$ where X_{TL} is 0×0, L_{TL} is 0×0, T_{TL} is 0×0
2	$\left\{ \left(\begin{array}{c\|c\|c} X_{TL} & \star & \star \\ \hline x_{ML}^T & \chi_{MM} & \star \\ \hline X_{BL} & x_{BM} & X_{BR} \end{array}\right) = \left(\begin{array}{c\|c\|c} T_{TL} & \star & \star \\ \hline \tau_{ML} e_i^T & 0 & \star \\ \hline 0 & \tau_{BM} L_{BR} e_f & L_{BR} T_{BR} L_{BR}^T \end{array}\right) \wedge \cdots \right\}$
3	while $m(X_{TL}) < m(X) - 1$ do
2,3	$\left\{ \left(\begin{array}{c\|c\|c} X_{TL} & \star & \star \\ \hline x_{ML}^T & \chi_{MM} & \star \\ \hline X_{BL} & x_{BM} & X_{BR} \end{array}\right) = \left(\begin{array}{c\|c\|c} T_{TL} & \star & \star \\ \hline \tau_{ML} e_i^T & 0 & \star \\ \hline 0 & \tau_{BM} L_{BR} e_f & L_{BR} T_{BR} L_{BR}^T \end{array}\right) \wedge \cdots \wedge m(X_{TL}) < m(X) - 1 \right\}$
5a	$\left(\begin{array}{c\|c\|c} X_{TL} & x_{TM} & X_{TR} \\ \hline x_{ML}^T & \chi_{MM} & x_{MR}^T \\ \hline X_{BL} & x_{BM} & X_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c\|c\|c\|c} X_{00} & x_{01} & x_{02} & X_{03} \\ \hline x_{10}^T & \chi_{11} & \chi_{12} & x_{13}^T \\ \hline x_{20}^T & \chi_{21} & \chi_{22} & x_{23}^T \\ \hline X_{30} & x_{31} & x_{32} & X_{33} \end{array}\right)$
6	$\left\{ \left(\begin{array}{c\|c\|c\|c} X_{00} & \star & \star & \star \\ \hline x_{10}^T & \chi_{11} & \star & \star \\ \hline x_{20}^T & \chi_{21} & \chi_{22} & \star \\ \hline X_{30} & x_{31} & x_{32} & X_{33} \end{array}\right) = \left(\begin{array}{c\|c\|c\|c} T_{00} & \star & \star & \star \\ \hline \tau_{10} e_i^T & 0 & \star & \star \\ \hline 0 & \tau_{21} \left(\begin{array}{c} 1 \\ \hline l_{32} \end{array}\right) & \left(\begin{array}{c\|c} 1 & 0 \\ \hline l_{32} & L_{33} \end{array}\right) & \left(\begin{array}{c\|c} 0 & \star \\ \hline \tau_{32} e_f & T_{33} \end{array}\right) \left(\begin{array}{c\|c} 1 & l_{32}^T \\ \hline 0 & L_{33}^T \end{array}\right) \\ \hline 0 & & & \end{array}\right) \wedge \cdots \right\}$
8	$l_{32} := x_{31} / \chi_{21}$ $x_{31} := 0$ $X_{33} := X_{33} + (l_{32} x_{32}^T - x_{32} l_{32}^T)$ (skew symmetric rank-2 update,
5b	$\left(\begin{array}{c\|c\|c} X_{TL} & x_{TM} & X_{TR} \\ \hline x_{ML}^T & \chi_{MM} & x_{MR}^T \\ \hline X_{BL} & x_{BM} & X_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c\|c\|c\|c} X_{00} & x_{01} & x_{02} & X_{03} \\ \hline x_{10}^T & \chi_{11} & \chi_{12} & x_{13}^T \\ \hline x_{20}^T & \chi_{21} & \chi_{22} & x_{23}^T \\ \hline X_{30} & x_{31} & x_{32} & X_{33} \end{array}\right)$
7	$\left\{ \left(\begin{array}{c\|c\|c\|c} X_{00} & \star & \star & \star \\ \hline x_{10}^T & \chi_{11} & \star & \star \\ \hline x_{20}^T & \chi_{21} & \chi_{22} & \star \\ \hline X_{30} & x_{31} & x_{32} & X_{33} \end{array}\right) = \left(\begin{array}{c\|c\|c\|c} T_{00} & \star & \star & \star \\ \hline \tau_{10} e_i^T & 0 & \star & \star \\ \hline 0 & \tau_{21} & 0 & \star \\ \hline 0 & 0 & \tau_{32} L_{33} e_f & L_{33} T_{33} L_{33}^T \end{array}\right) \right\}$
2	$\left\{ \left(\begin{array}{c\|c\|c} X_{TL} & \star & \star \\ \hline x_{ML}^T & \chi_{MM} & \star \\ \hline X_{BL} & x_{BM} & X_{BR} \end{array}\right) = \left(\begin{array}{c\|c\|c} T_{TL} & \star & \star \\ \hline \tau_{ML} e_i^T & 0 & \star \\ \hline 0 & \tau_{BM} L_{BR} e_f & L_{BR} T_{BR} L_{BR}^T \end{array}\right) \wedge \cdots \right\}$
	endwhile
2,3	$\left\{ \left(\begin{array}{c\|c\|c} X_{TL} & \star & \star \\ \hline x_{ML}^T & \chi_{MM} & \star \\ \hline X_{BL} & x_{BM} & X_{BR} \end{array}\right) = \left(\begin{array}{c\|c\|c} T_{TL} & \star & \star \\ \hline \tau_{ML} e_i^T & 0 & \star \\ \hline 0 & \tau_{BM} L_{BR} e_f & L_{BR} T_{BR} L_{BR}^T \end{array}\right) \wedge \cdots \wedge \neg(m(X_{TL}) < m(X) - 1) \right\}$
1b	$\left\{ X = T \wedge \widehat{X} = LTL^T \right\}$

FLAME Family Values

- Parlett-Reid algorithm
  - Right-looking algorithm
  - Performs a syr2 update per iteration
- Wimmer's two-step
  - Right-looking algorithm
  - Performs a syr2 update ever other iteration
- Aasen's algorithm
  - Left-looking algorithm
  - Performs gemv per iteration
- New blocked right-looking algorithm!
  - Similar to an algorithm by Rozloznik et al. but better (no temp storage, more elegant)
- New (?) blocked left-looking algorithm



Blocked right- and left-looking algorithms

What do we learn?

Algorithm: $[X, L] := \text{LTLT\_BLK\_RIGHT/LEFT}(X)$

$L = I$

$$X \rightarrow \left(\begin{array}{c|c|c} X_{TL} & x_{TM} & X_{TR} \\ \hline x_{ML}^T & \chi_{MM} & x_{ML}^T \\ \hline X_{BL} & x_{BM} & X_{BR} \end{array}\right), \quad L \rightarrow \left(\begin{array}{c|c|c} L_{TL} & l_{TM} & L_{TR} \\ \hline l_{ML}^T & \lambda_{MM} & l_{ML}^T \\ \hline L_{BL} & l_{BM} & L_{BR} \end{array}\right)$$

where X_{TL} and L_{TL} are 0×0

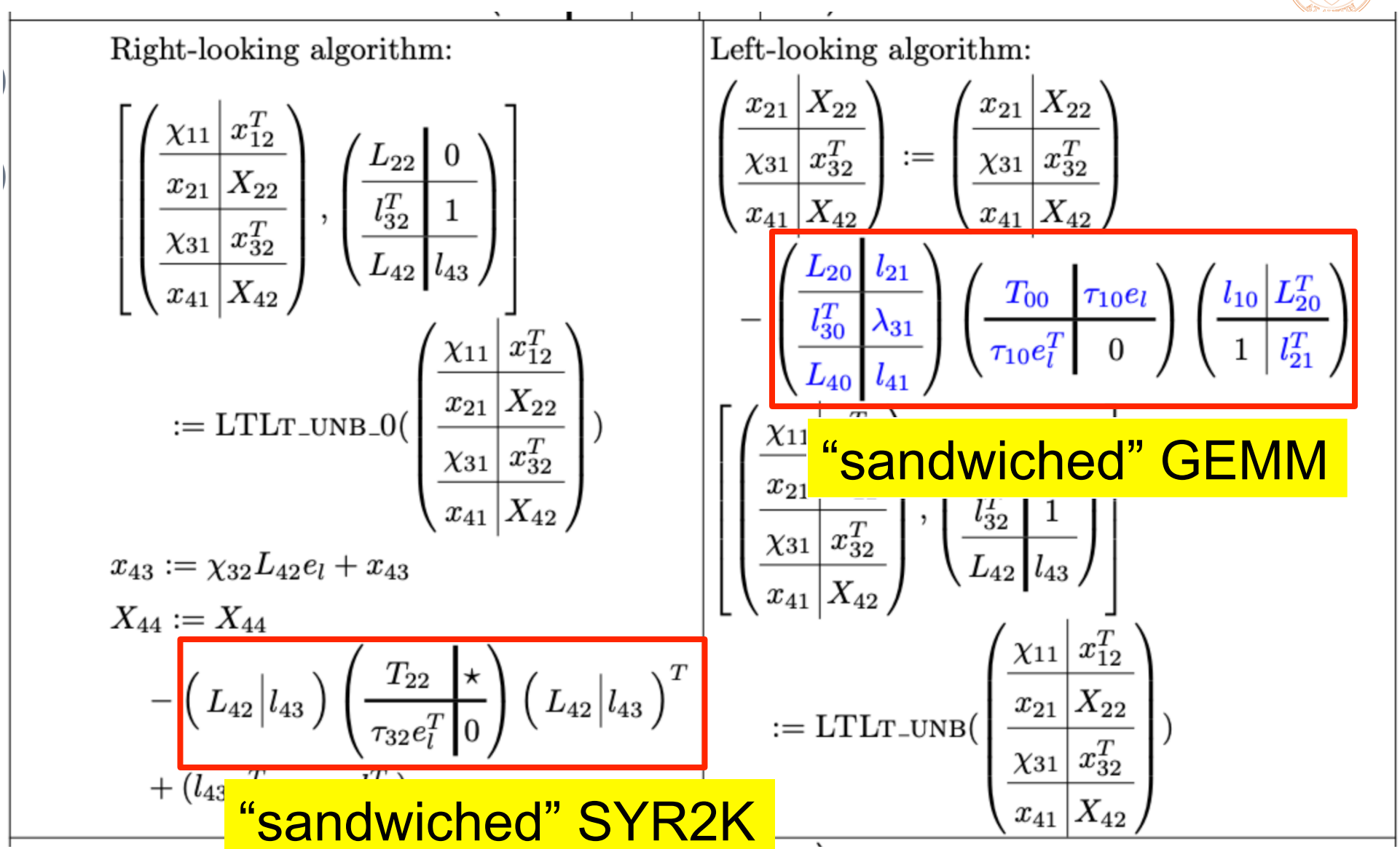
while $m(X_{TL}) < m(X) - 1$ do

$$\left(\begin{array}{c|c|c} X_{TL} & x_{TM} & X_{TR} \\ \hline x_{ML}^T & \chi_{MM} & x_{ML}^T \\ \hline X_{BL} & x_{BM} & X_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c|c|c} X_{00} & x_{01} & X_{02} & x_{03} & X_{04} \\ \hline x_{10}^T & \chi_{11} & x_{12}^T & \chi_{13} & x_{14}^T \\ \hline X_{20} & x_{21} & X_{22} & x_{23} & X_{24} \\ \hline x_{30}^T & \chi_{31} & x_{32}^T & \chi_{33} & x_{34}^T \\ \hline X_{40} & x_{41} & X_{42} & x_{43} & X_{44} \end{array}\right), \left(\begin{array}{c|c|c} L_{TL} & l_{TM} & L_{TR} \\ \hline l_{ML}^T & \lambda_{MM} & l_{ML}^T \\ \hline L_{BL} & l_{BM} & L_{BR} \end{array}\right) \rightarrow \cdots$$

3 x 3 → 5 x 5 → 3 x 3

$$\left(\begin{array}{c|c|c} X_{TL} & x_{TM} & X_{TR} \\ \hline x_{ML}^T & \chi_{MM} & x_{ML}^T \\ \hline X_{BL} & x_{BM} & X_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c|c|c} X_{00} & x_{01} & X_{02} & x_{03} & X_{04} \\ \hline x_{10}^T & \chi_{11} & x_{12}^T & \chi_{13} & x_{14}^T \\ \hline X_{20} & x_{21} & X_{22} & x_{23} & X_{24} \\ \hline x_{30}^T & \chi_{31} & x_{32}^T & \chi_{33} & x_{34}^T \\ \hline X_{40} & x_{41} & X_{42} & x_{43} & X_{44} \end{array}\right), \left(\begin{array}{c|c|c} L_{TL} & l_{TM} & L_{TR} \\ \hline l_{ML}^T & \lambda_{MM} & l_{ML}^T \\ \hline L_{BL} & l_{BM} & L_{BR} \end{array}\right) \leftarrow \cdots$$

endwhile


Right-looking algorithm:
$\left[\left(\begin{array}{c|c} \chi_{11} & x_{12}^T \\ \hline x_{21} & X_{22} \\ \hline \chi_{31} & x_{32}^T \\ \hline x_{41} & X_{42} \end{array}\right), \left(\begin{array}{c|c} L_{22} & 0 \\ \hline l_{32}^T & 1 \\ \hline L_{42} & l_{43} \end{array}\right)\right]$
$:= \text{LTLT\_UNB\_0}\left(\begin{array}{c|c} \chi_{11} & x_{12}^T \\ \hline x_{21} & X_{22} \\ \hline \chi_{31} & x_{32}^T \\ \hline x_{41} & X_{42} \end{array}\right)$
$x_{43} := \chi_{32} L_{42} e_l + x_{43}$
$X_{44} := X_{44}$
$-\left(\begin{array}{c|c} L_{42} & l_{43} \end{array}\right)\left(\begin{array}{c|c} T_{22} & \star \\ \hline \tau_{32} e_l^T & 0 \end{array}\right)\left(\begin{array}{c|c} L_{42} & l_{43} \end{array}\right)^T$
$+ (l_{43}$
"sandwiched" SYR2K
Left-looking algorithm:
$\left(\begin{array}{c|c} x_{21} & X_{22} \\ \hline \chi_{31} & x_{32}^T \\ \hline x_{41} & X_{42} \end{array}\right) := \left(\begin{array}{c|c} x_{21} & X_{22} \\ \hline \chi_{31} & x_{32}^T \\ \hline x_{41} & X_{42} \end{array}\right)$
$-\left(\begin{array}{c|c} L_{20} & l_{21} \\ \hline l_{30}^T & \lambda_{31} \\ \hline L_{40} & l_{41} \end{array}\right)\left(\begin{array}{c|c} T_{00} & \tau_{10} e_l \\ \hline \tau_{10} e_l^T & 0 \end{array}\right)\left(\begin{array}{c|c} l_{10} & L_{20}^T \\ \hline 1 & l_{21}^T \end{array}\right)$
"sandwiched" GEMM
$:= \text{LTLT\_UNB}\left(\begin{array}{c|c} \chi_{11} & x_{12}^T \\ \hline x_{21} & X_{22} \\ \hline \chi_{31} & x_{32}^T \\ \hline x_{41} & X_{42} \end{array}\right)$

Algorithm: $[X, L] := \text{LTLT\_UNB\_WIMMER}(X)$

$L = I$

```
auto [T, m, B] = partition_rows<DYNAMIC,1,DYNAMIC>(X);
```

while $m(X_{TL}) < m(X) - 1$ do

```
auto [R0,r1,r2,r3,R4] = repartition<2>(T, m, B);

L(r3,r2) = X(r3,r1) / X(r2,r1);

blas::skr2(1.0, L(R4,r3), X(R4,r3), 1.0, X(R4,R4));
X(R4,r3) -= X(r3,r2)*L(r3,r2)*L(R4,r3);

tie(T, m, B) = continue_with(R0, r1, r2, r3, R4);
```

endwhile

How do we translate this into code?

- Retaining pedagogical value
- Attaining high performance

Algorithm: $[X, L] := \text{LTLT\_UNB\_WIMMER}(X)$

$L = I$

```
auto [T, m, B] = partition_rows<DYNAMIC,1,DYNAMIC>(X);
```

while $m(X_{TL}) < m(X) - 1$ do

```
auto [R0,r1,r2,r3,R4] = repartition<2>(T, m, B);

    L(r3,r2) = X(r3,r1) / X(r2,r1);
    L(R4,r2) = X(R4,r1) / X(r2,r1);
    X(r3,r1) = 0;
    X(R4,r1) = 0;

    L(R4,r3) = X(R4,r2) / X(r3,r2);
    X(R4,r2) = 0;

    X(R4,r3) += X(r3,r2)*L(R4,r2);

    blas::skr2(1.0, L(R4,r3), X(R4,r3), 1.0, X(R4,R4));
    X(R4,r3) -= X(r3,r2)*L(r3,r2)*L(R4,r3);

tie(T, m, B) = continue_with(R0, r1, r2, r3, R4);
```

Partition ranges rather than matrices

endwhile

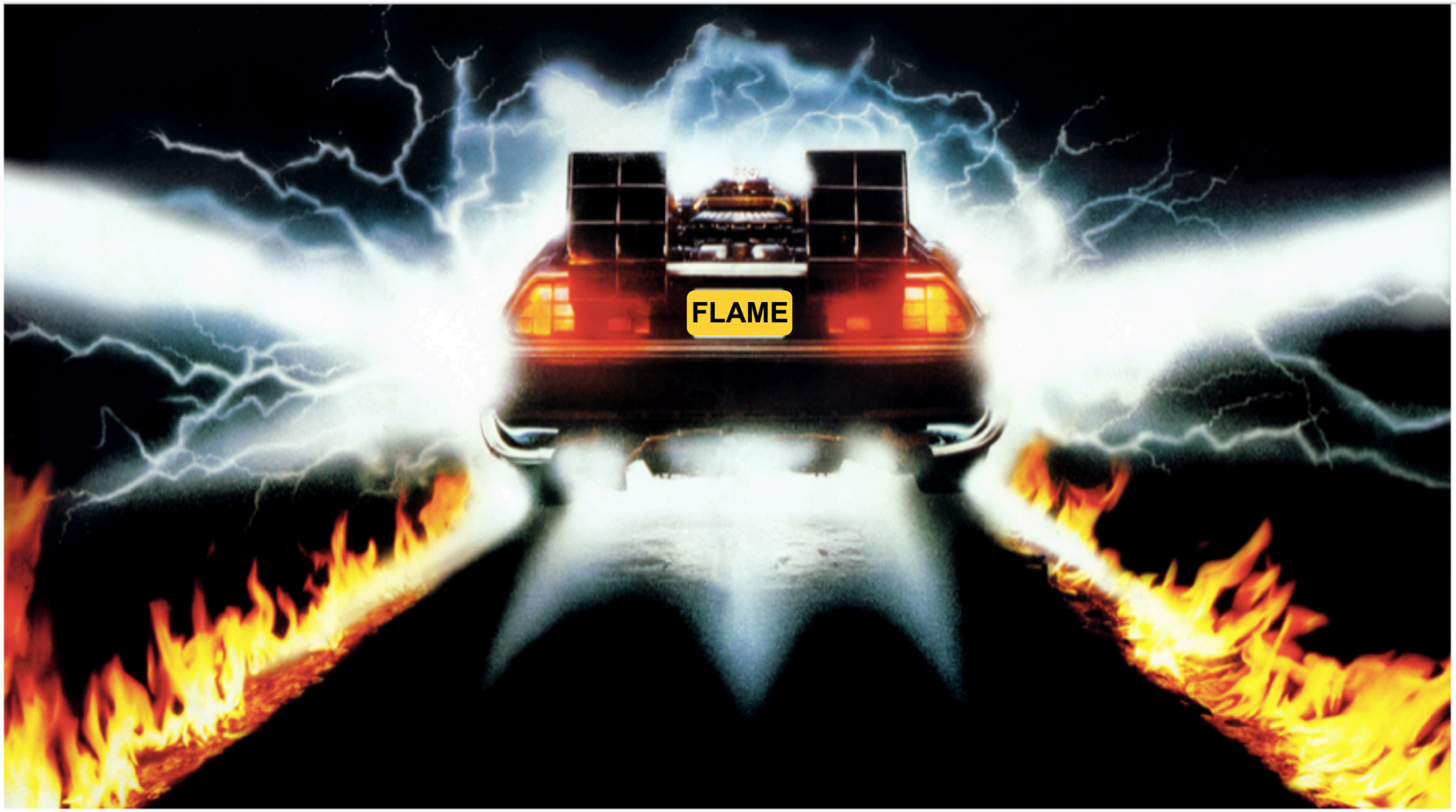
```
using namespace MArray;

matrix<double> LTLt_unb_Wimmer(const matrix_view<double>& X)
{
    using std::tie;
    const auto n = X.length(0);
    MARRAY_ASSERT(n == X.length(1)); //The matrix must be square
    MARRAY_ASSERT(n % 2 == 0); //The matrix must have an even number of rows

    matrix<double> L{n, n}; //Default is fill with 0

    auto [T, m, B] = partition_rows<DYNAMIC,1,DYNAMIC>(X);

    while (L.size() > 1)
    {
        auto [R0,r1,r2,r3,R4] = repartition<2>(T, m, B);

        L(r3,r2) = X(r3,r1) / X(r2,r1);
        L(R4,r2) = X(R4,r1) / X(r2,r1);
        X(r3,r1) = 0;
        X(R4,r1) = 0;

        L(R4,r3) = X(R4,r2) / X(r3,r2);
        X(R4,r2) = 0;

        X(R4,r3) += X(r3,r2)*L(R4,r2);

        blas::skr2(1.0, L(R4,r3), X(R4,r3), 1.0, X(R4,R4));
        X(R4,r3) -= X(r3,r2)*L(r3,r2)*L(R4,r3);

        tie(T, m, B) = continue_with(R0, r1, r2, r3, R4);
    }

    return L;
}
```

Benefits of splitting ranges

- Splitting is just fancy indexing
- Splitting ranges ties rows and/or columns of different matrix operands
- Devin claims the resulting code can be as fast as hand coded
- Extends beyond matrices to higher order tensors

Linear and Multi-linear Libraries: The Next Generation

- Time to get Back to the Future!


FLAME