

# GEMM3: Constant-Workspace High-Performance Multiplication of Three Matrices for Matrix Chaining

Krzysztof A. Drewniak

The University of Texas at Austin

April 13, 2018

# Matrix chaining problem

- ▶ Problem: compute  $A_1 A_2 \cdots A_n$  efficiently
- ▶  $O(n \log n)$  algorithm, also  $O(n^3)$  with dynamic programming
- ▶ Fewer flops  $\rightarrow$  more performance?

# Generalized matrix chaining

- ▶ In reality — transposes, inverses, properties
- ▶ Ensemble Kalman filter  $X_i^b S_i (Y_i^b)^T R_i^{-1}$   
 Tridiagonalization  $\tau_u \tau_v v v^T A u u^T$   
 Two-sided triangular solve  $L^{-1} A L^{-H}$  ( $L$  lower triangular)
- ▶ Performance with BLAS/LAPACK – must be expert
- ▶ Less performance with Matlab, numpy, etc. (left-to-right)
- ▶ Linnea: expression  $\rightarrow$  BLAS calls automatically

# GEMM3 — Why bother?

- ▶ ▶  $\mathbf{X}_i^b \mathbf{S}_i (\mathbf{Y}_i^b)^T \mathbf{R}_i^{-1}$
- ▶  $\tau_u \tau_v \mathbf{v} \mathbf{v}^T \mathbf{A} \mathbf{u} \mathbf{u}^T$
- ▶  $\mathbf{L}^{-1} \mathbf{A} (\mathbf{L}^{-1})^H$  ( $\mathbf{L}$  lower triangular)
- ▶ All multiply three matrices as a subproblem
- ▶ (Notation:  $G += DEF$  and GEMM3)
- ▶ Not everything divides well this way

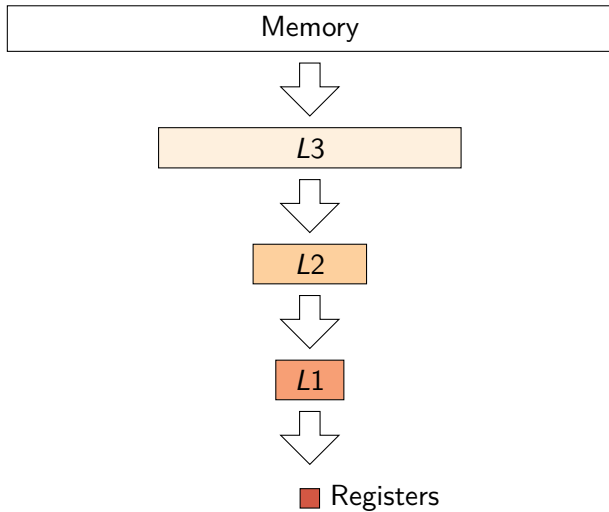
## GEMM3 — Why a new algorithm?

- ▶ Current approach: parentheses, multiply twice, store temporary  $T$
- ▶  $T$  often eats memory (& perf)
- ▶ We can do better!
- ▶ Use how GEMM works to nest computations
- ▶  $O(1)$  extra memory, maybe more performance

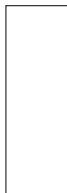
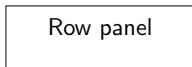
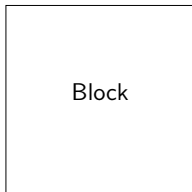
## Section 2

### High-Performance GEMM

# Memory hierarchy



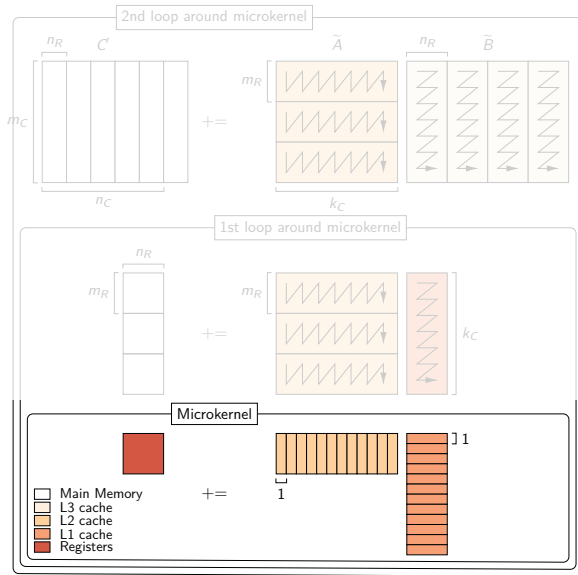
# Important matrix shapes



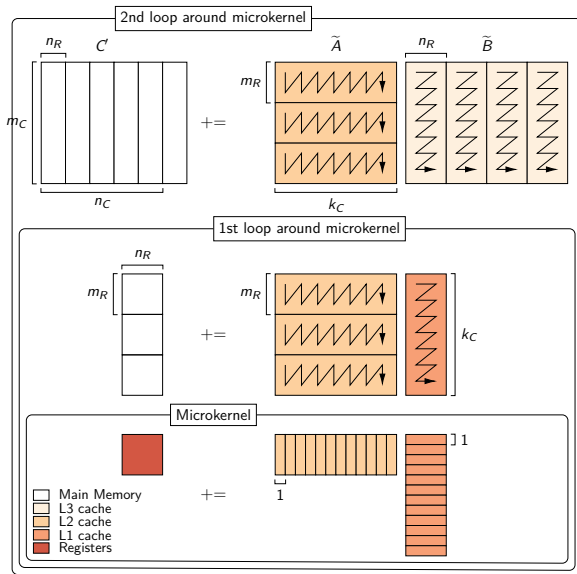
Column panel



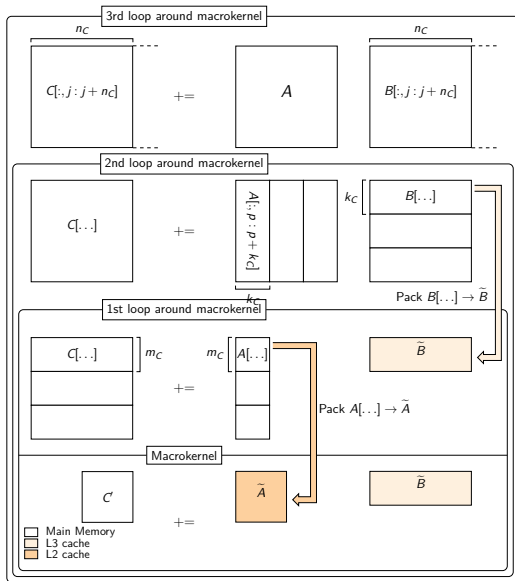
# GEMM: The kernels



# GEMM: The kernels



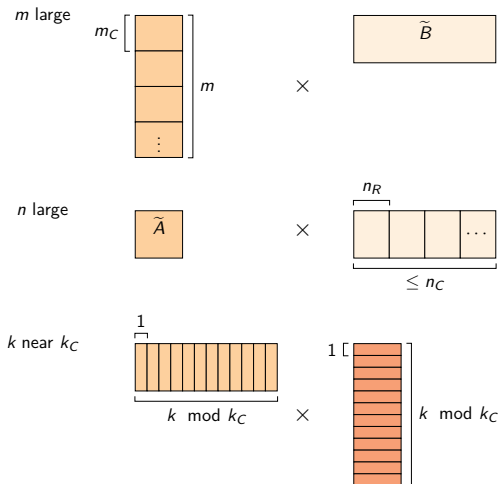
# GEMM: The algorithm



# Data reuse

- Every loop reads *something* repeatedly

Want:



## Section 3

# The GEMM3 algorithm

# Key concept of the algorithm

- ▶ We want  $G \mathrel{+}= DEF$ , (dimensions:  $m, k, l, n$  in order)
- ▶  $EF$  first needed in packing step
- ▶ Compute a row panel then

# Deriving GEMM3: Partitionings

$$G \quad += \quad D \quad [(EF) \quad \leftrightarrow \quad E \quad F]$$

1.  $m \times n \quad += \quad m \times k \quad \left[ \begin{array}{c} k \times n \end{array} \right] \leftrightarrow \left[ \begin{array}{cc} k \times l & l \times n \end{array} \right]$

2.  $m \times n_C \quad += \quad m \times k \quad \left[ \begin{array}{c} k \times n_C \end{array} \right] \leftrightarrow \left[ \begin{array}{cc} k \times l & l \times n_C \end{array} \right]$

3.  $m \times n_C \quad += \quad \begin{array}{c} m \times \\ k_C \end{array} \quad \left[ \begin{array}{c} k_C \times n_C \end{array} \right] \leftrightarrow \left[ \begin{array}{cc} k_C \times l & l \times n_C \end{array} \right]$

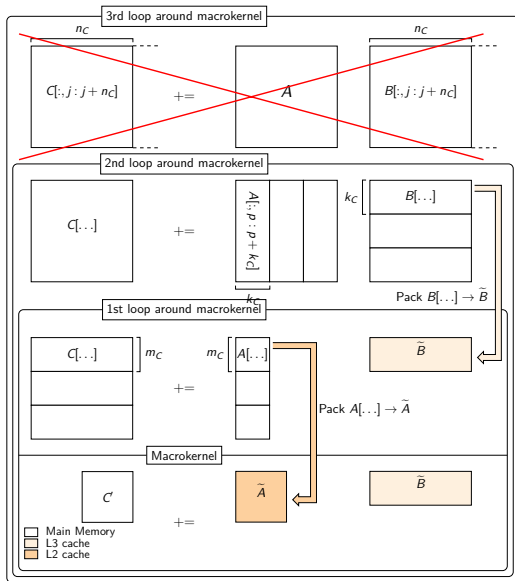
## Deriving GEMM3: Inner algorithm

$$\boxed{EF: k_C \times n_C} = \boxed{E: k_C \times l} \boxed{F: l \times n_C}$$

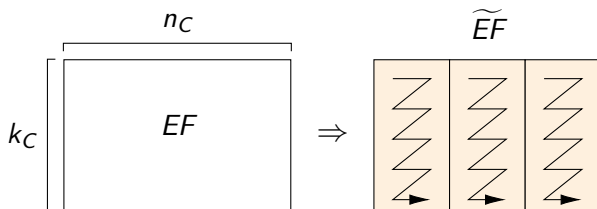
- ▶ Only point to compute  $EF$  in constant memory
- ▶ GEMM algorithm needs tweaks



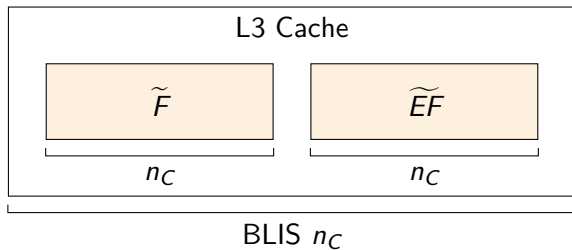
# Inner algorithm tweaks: Removing the outer loop



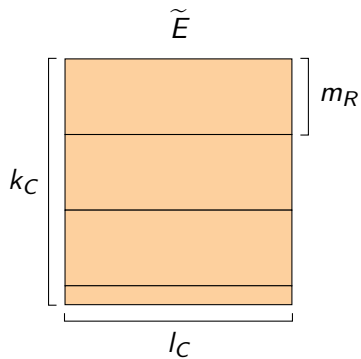
# Inner alg. tweaks: Microkernel packed writes



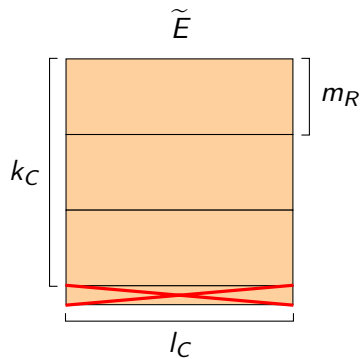
# Inner alg. tweaks: Halving $n_C$



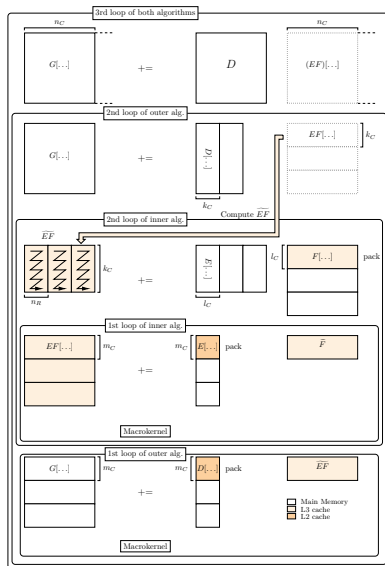
# Inner algtweaks: Small $k_C$ reduction



# Inner algtweaks: Small $k_C$ reduction



# The algorithm



# The small drawback

Problem shape:

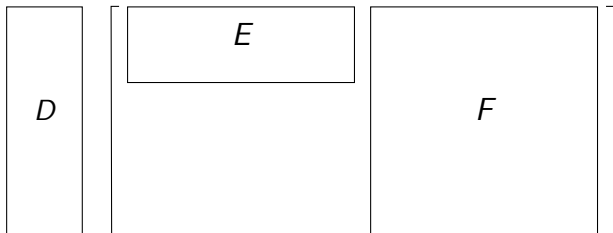
$$\boxed{\widetilde{EF}: k_C \times n_C} = \boxed{E: k_C \times l} \boxed{F: l \times n_C}$$

Reuse problem:  $m$  small

$$\begin{matrix} m_R \\ \left[ \begin{array}{c} \text{orange box} \\ \text{orange box} \\ \text{orange box with red X} \\ \text{orange box with red X} \end{array} \right] k_C \end{matrix} \times \boxed{\widetilde{F}}$$

$$G += (DE)F$$

Ex: Don't



- ▶ Putting parentheses there sometimes better
- ▶ Deriving directly doesn't work — bad shape
- ▶ However,  $G += (DE)F \Leftrightarrow G^T += F^T(E^T D^T)$



## Section 4

# Experiments and Results

## Implementation details

- ▶ Multilevel Optimization of Matrix Multiply Sandbox (MOMMS)
- ▶ Extended to support three matrices
- ▶ Implement both GEMM3 and BLIS algorithm
- ▶ BLIS algorithm port performs like BLIS
- ▶ Machine: 3.5 GHz (one core used), 15 GB RAM, 32 KB  $L1$  cache, 256 KB  $L2$ , 8 MB  $L3$ . Peak perf 56 GFLOPS/s.

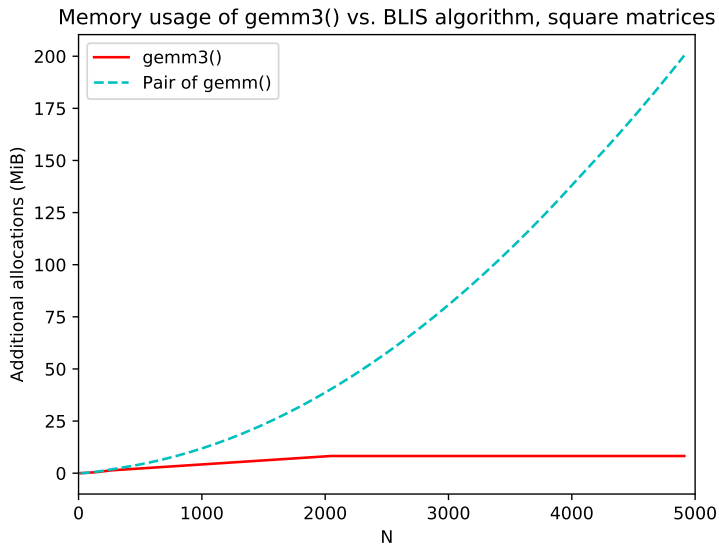
	GEMM3	BLIS algorithm
$m_R$	6	6
$n_R$	8	8
$m_C$	72	72
$k_C$	252	256
$l_C$	256	
$n_C$	2040	4080

Table: Parameters for Haswell CPUs

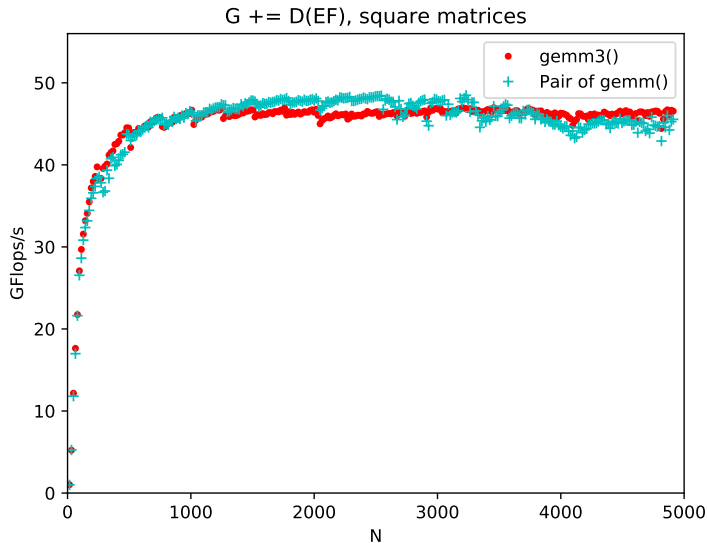
# Experiments

1.  $G += D(EF)$ , square matrices
  - ▶ Inputs column-major, outputs row-major for fairness
2.  $G^T += F^T(E^T D^T)$ , square matrices
  - ▶ After transpose, all row major
3.  $G += D(EF)$ , rectangles (one dimension small)

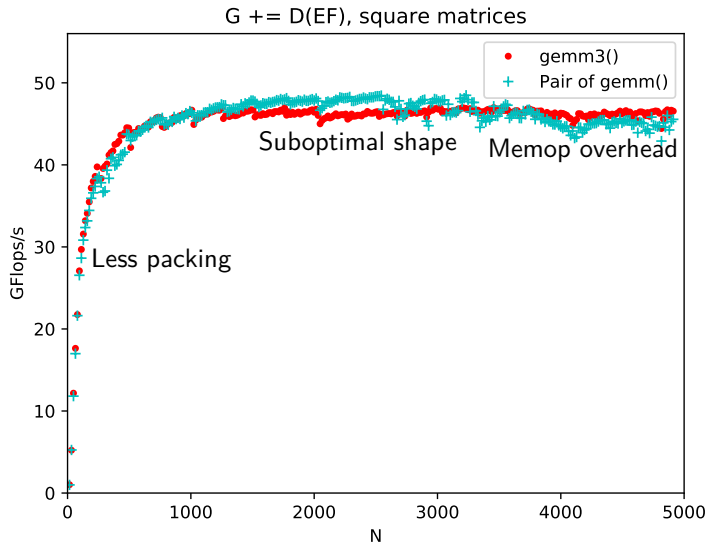
# Workspace usage, square matrices



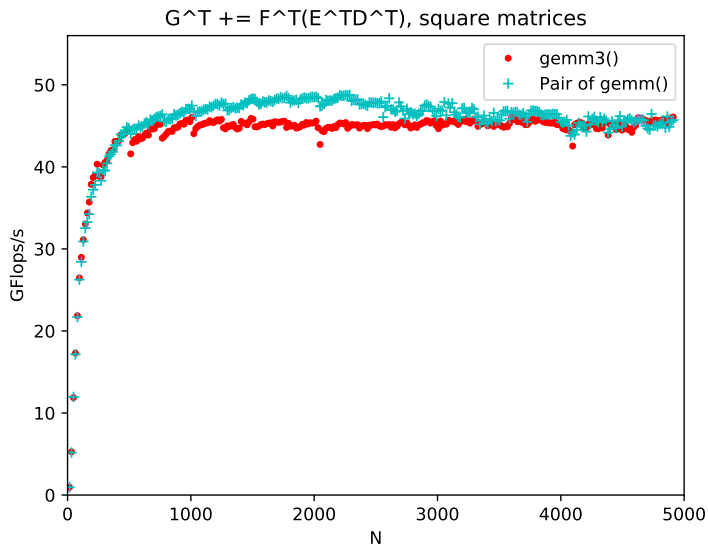
# $G += D(EF)$ , square matrices



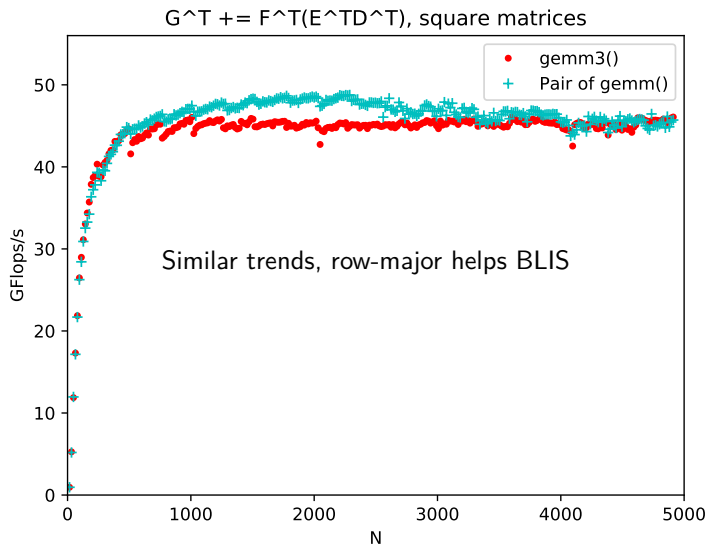
# $G += D(EF)$ , square matrices



# $G += (DE)F$ , square matrices



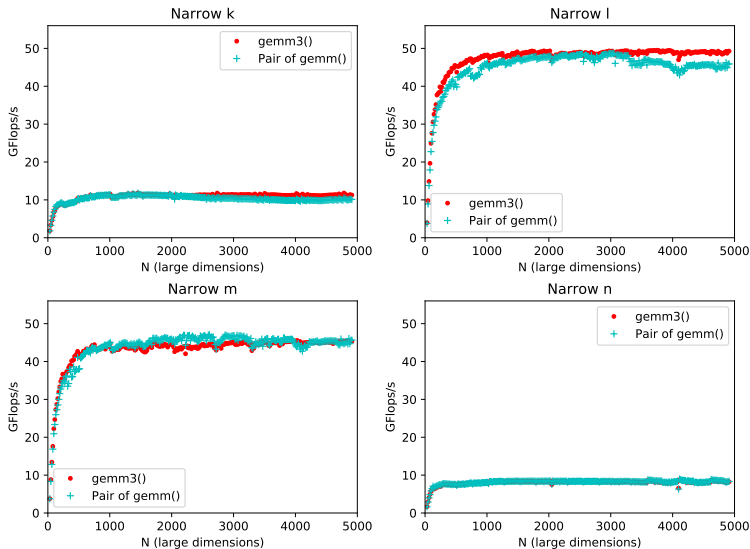
# $G += (DE)F$ , square matrices





# $G += D(EF)$ , rectangular matrices

$G += D(EF)$ , narrow dimension = 9



# Conclusions

- ▶ GEMM structure lets us make GEMM3
- ▶ Constant memory
- ▶ Comprable performance
- ▶ Cleaner API

# Future Work

- ▶ Parallel case
- ▶ More architectures

# Acknowledgments

- ▶ Prof. Robert van de Geijn — advising and providing inspiration
- ▶ Dr. Tyler Smith — writing MOMMS and algorithm design
- ▶ Prof. Tze Meng Low — performance fixes
- ▶ NSF awards CCF-1714091 and ACI-1550493 — funding

# Questions?

## Picking parameters: $m_R, n_R$

- ▶ Determine microkernel
- ▶ Based on microarchitecture — register width, FMA properties
- ▶ We're reusing BLIS's work
- ▶ Can swap  $m_R$  and  $n_R$

Picking parameters:  $k_C$ 

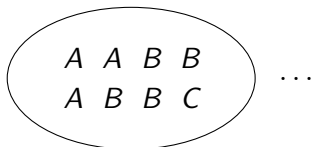
Placing memory in cache: [tag][set #][offset in line]

$$m_r k_C S_{elem} = C_A C_{L1} N_{L1}$$

$$n_r k_C S_{elem} = C_B C_{L1} N_{L1}$$

L1

Cache:



$$C_A + C_B + 1 \leq W_{L1}$$

Maximizing  $k_C$  improves performance

$$C_B = \left\lceil \frac{n_R k_C S_{elem}}{N_{L1} C_{L1}} \right\rceil$$

$$= \left\lceil \frac{n_R}{m_R} C_A \right\rceil$$

$$C_A \leq \left\lfloor \frac{W_{L1} - 1}{1 + \frac{n_R}{m_R}} \right\rfloor$$

GEMM3

## Picking parameters: $m_C$ and $n_C$

- ▶ For  $m_C$ : reserve ways for  $B$  and  $C$
- ▶ Then take all you can
- ▶  $n_C$ , leave out what architecture requires, then divide
- ▶ L3 is very big, tuning is much less needed