GEMM3(): Constant-workspace high-performance multiplication of three matrices for matrix chaining

Krzysztof A. Drewniak

The University of Texas at Austin

Date TBD

Outline

Matrix chaining problem

- Problem: compute $A_1 A_2 \cdots A_n$ efficiently, A_i matrices
- Where do the parentheses go?
- $O(n \log n)$ algorithm, also $O(n^3)$ with dynamic programming
- Fewer flops $\rightarrow$ more performance?

Generalized matrix chaining

- In reality — transposes, inverses, properties
- Ex:
  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 automagically

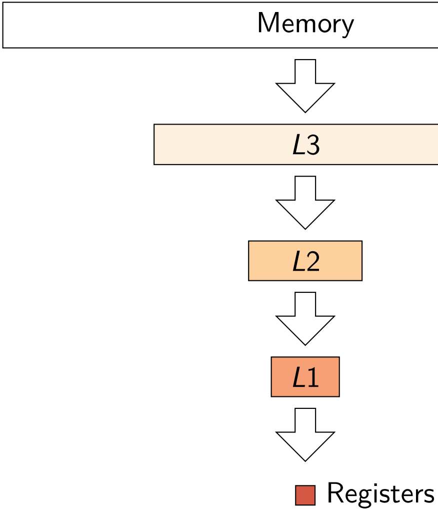
GEMM3() — Why bother?

- Examples again:
  - $\boldsymbol{X_i^b S_i (Y_i^b)^T} R_i^{-1}$
  - $\tau_u \tau_v \boldsymbol{vv^T A uu^T}$
  - $\boldsymbol{L^{-1} A (L^{-1})^H}$ (L lower triangular)
- All multiply three matrices as a subproblem
- (Notation: $D \mathrel{+}= ABC$ and GEMM3())

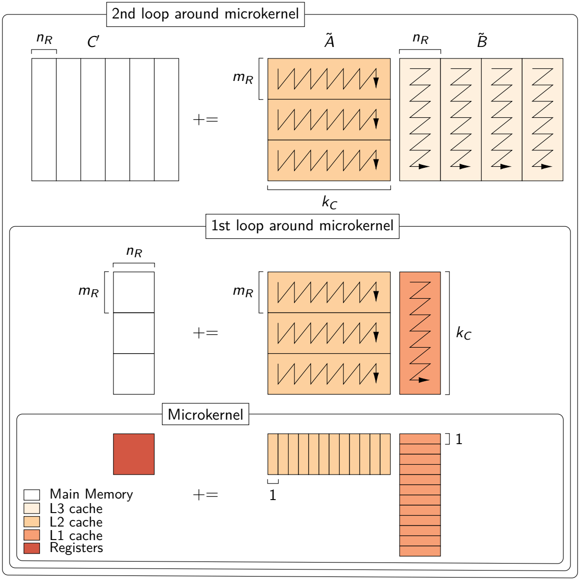
GEMM3() — Why a new algorithm?

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

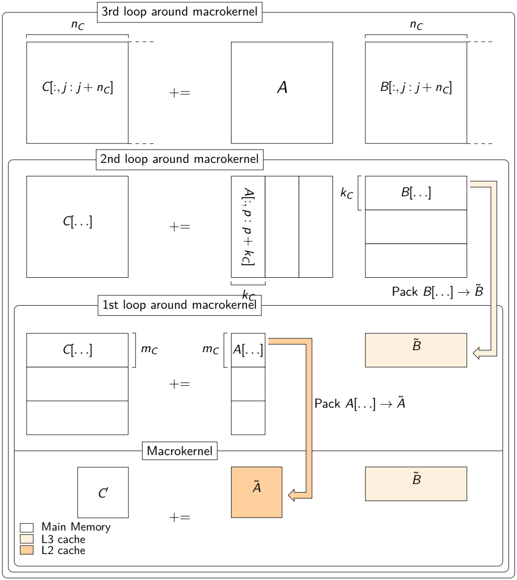
Memory hierarchy



GEMM(): The kernels



GEMM(): The algorithm



Picking constants: 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 constants: k_C

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

$$m_r k_c S_{elem} = C_A C_{L1} N_{L1} \qquad n_r k_C S_{elem} = C_B C_{L1} N_{L1}$$

Cache:

$$\begin{matrix} A & A & B & B \\ A & B & B & C \end{matrix} \quad \cdots$$

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

Maximizing k_C improves performance

$$\begin{aligned} C_B &= \left\lceil \frac{n_R k_C S_{elem}}{N_{L1} C_{L1}} \right\rceil \\ &= \left\lceil \frac{n_R}{m_R} C_A \right\rceil \\ C_A &\leq \left\lfloor \frac{W_{L1} - 1}{1 + \frac{n_R}{m_R}} \right\rfloor \end{aligned}$$

Picking constants: 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

Data reuse

- Every loop reads *something* repeatedly
- Relevant things: packed blocks — making them takes time
- Packed block reuse problems:
  - m small — low time between remakes of $\tilde{B}$
  - n small — same for $\tilde{A}$
  - k tiny — microkernel doesn't do much, small caches

Key concept of the algorithm

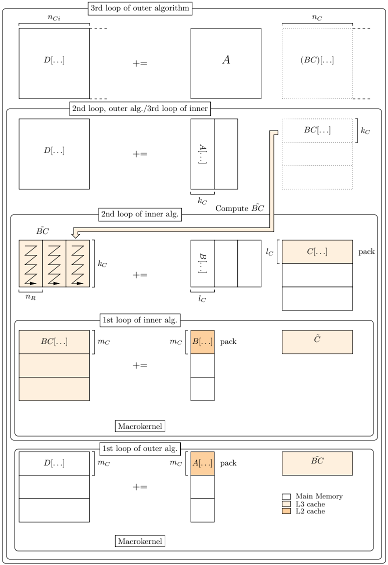
- We want $D \mathrel{+}= ABC$, (dimensions: m, k, l, n in order)
- BC first needed in packing step
- Compute a block then
- Have $\textsc{gemm}()$ algorithm, but...

The tricky bits

Problem	Solution
Redundant loop over n ($n \leq n_C$)	Remove it
Packing output wastes space/time	Tweak microkernel params
$\tilde{C}$ fights $\tilde{BC}$ in $L3$	Halve n_C
Low $\tilde{C}$ reuse	Low impact in practice
$m_R \nmid k_C$, leaving fringe	Shrink k_C slightly

Table: Tweaks needed to make GEMM() fusion work

The algorithm



$D \mathrel{+}= (AB)C$

- Putting parentheses there sometimes better
- Deriving directly doesn't work
  - Multiple recomputations of $\tilde{A}B$
- However, $D \mathrel{+}= (AB)C \Leftrightarrow D^T \mathrel{+}= C^T(B^T A^T)$

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
- Experiments on Haswell machine from UT lab

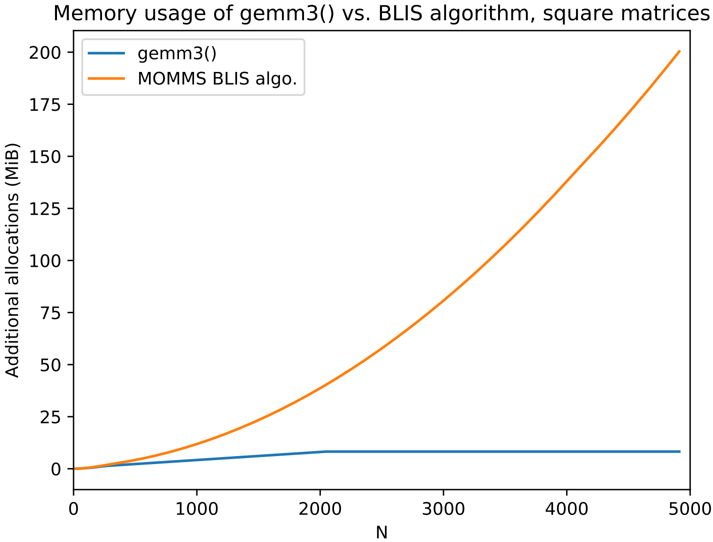
	GEMM3()	BLIS algorithm
m_C	72	72
k_C	252	256
l_C	256	
n_C	2040	4080

Table: Constants for Haswell CPUs

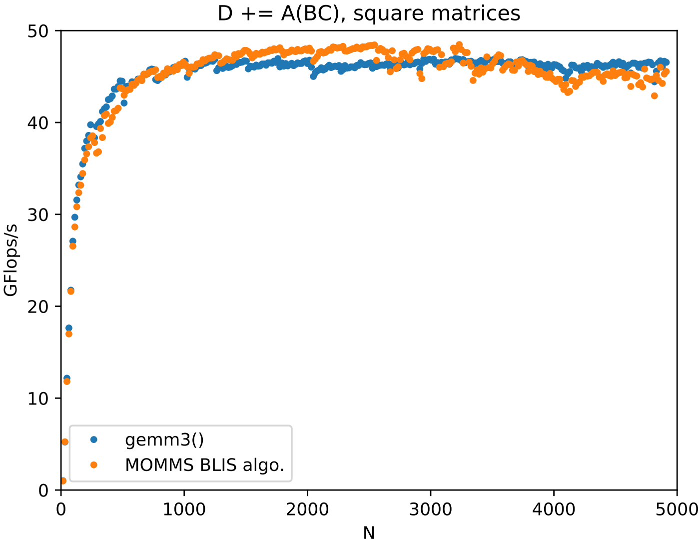
Experiments

1. $D \mathrel{+}= A(BC)$, square matrices
   - Inputs column-major, outputs row-major for fairness
2. $D^T \mathrel{+}= C^T(B^T A^T)$, square matrices
   - After transpose, all row major
3. $D \mathrel{+}= A(BC)$, rectangles (one dimension small)

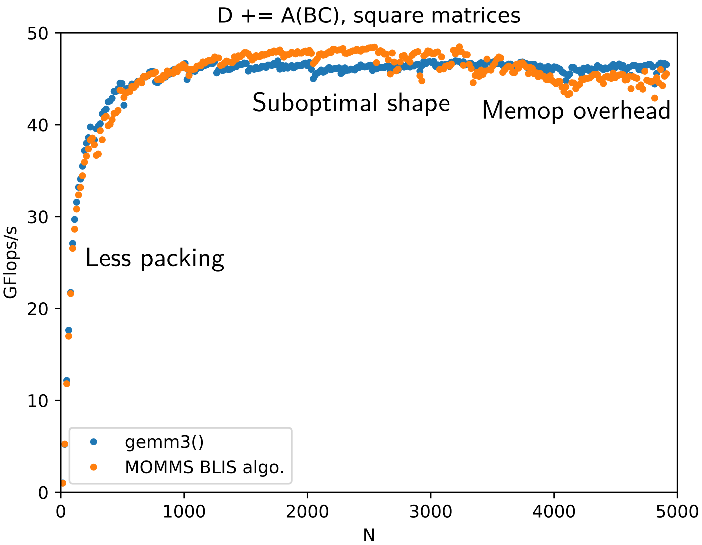
Workspace usage, square matrices



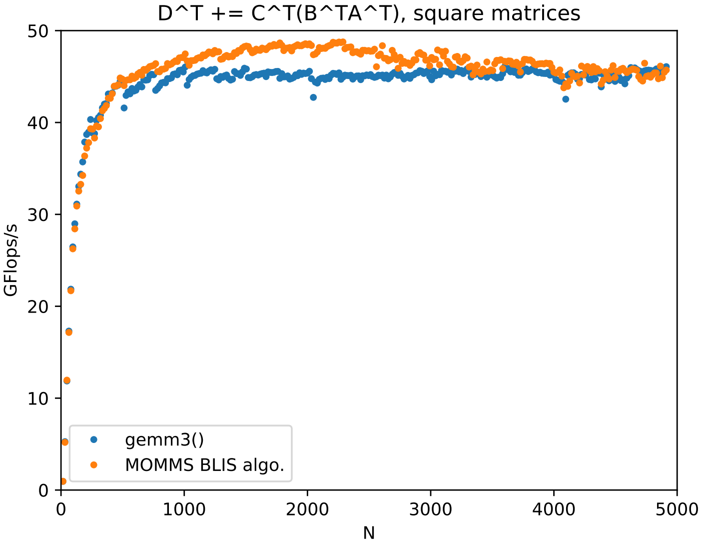
$D \mathrel{+}= A(BC)$, square matrices



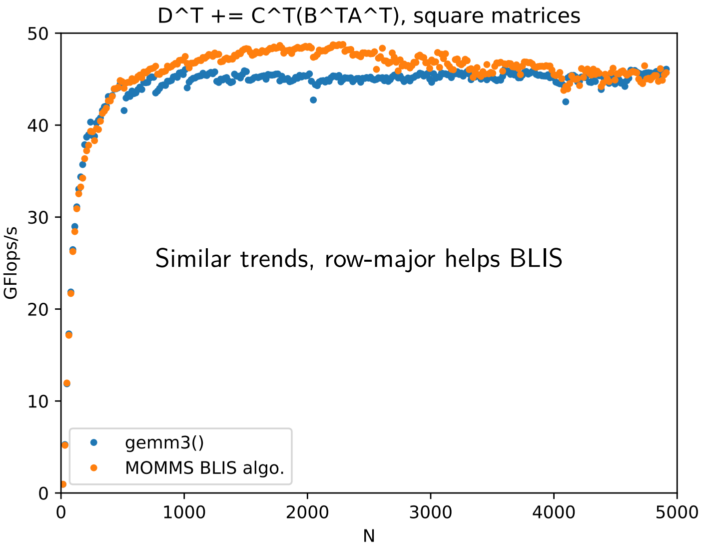
$D += A(BC)$, square matrices



$D \mathrel{+}= (AB)C$, square matrices



$D += (AB)C$, square matrices



$D += A(BC)$, rectangular matrices