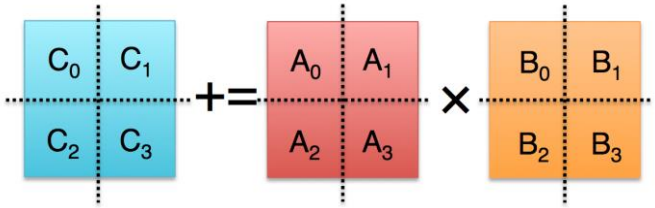
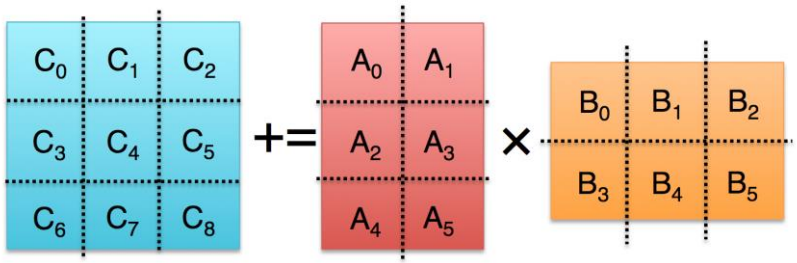




Practical Fast Matrix Multiplication Algorithms

Jianyu Huang

Committee members: Prof. Robert van de Geijn (Advisor),
Prof. Don Batory, Prof. George Biros, Prof. Keshav Pingali, and Dr. Greg Henry

August 22, 2017
UTCS PhD Dissertation Proposal
GDC 5.816, Austin, TX

Outline

- Motivation
- Background
- Practical Strassen’s Algorithm (SC16)
- Practical Fast Matrix Multiplication (IPDPS17)
- Proposed Work
- Conclusion

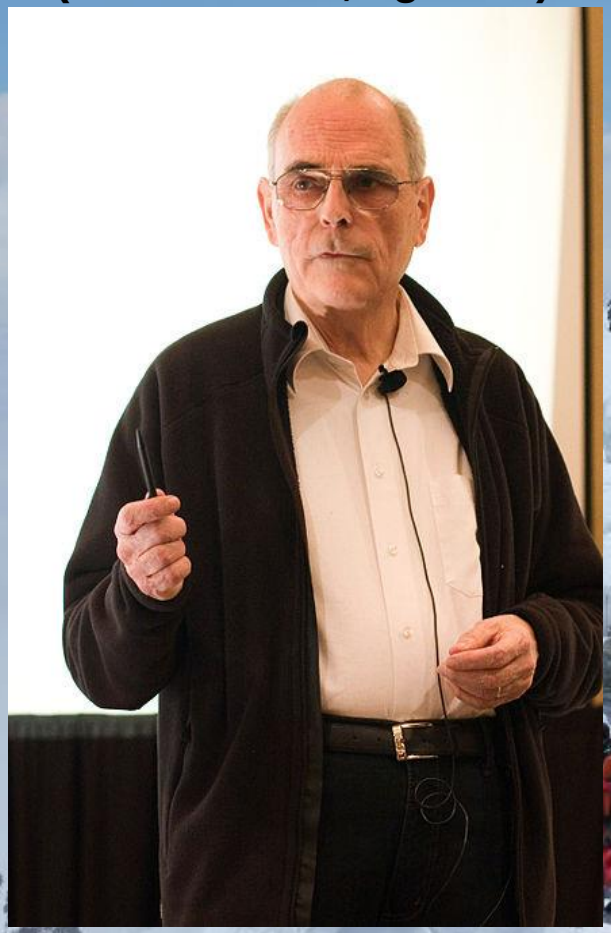
Outline

- Motivation
- Background
- Practical Strassen's Algorithm (SC16)
- Practical Fast Matrix Multiplication (IPDPS17)
- Proposed Work
- Conclusion

* Background Image: Mount Everest. From https://www.adventurealternative.com

STRASSEN, the FIRST Fast Matrix Multiplication (FMM)…

Volker Strassen
(Born in 1936, aged 81)



Original Strassen Paper (1969)

Numer. Math. 13, 354–356 (1969)

Gaussian Elimination is not Optimal

VOLKER STRASSEN*

Received December 12, 1968

1. Below we will give an algorithm which computes the coefficients of the product of two square matrices A and B of order n from the coefficients of A and B with less than $4.7 \cdot n^{\log 7}$ arithmetical operations (all logarithms in this paper are for base 2, thus $\log 7 \approx 2.8$; the usual method requires approximately $2n^3$ arithmetical operations). The algorithm induces algorithms for inverting a matrix of order n, solving a system of n linear equations in n unknowns, computing a determinant of order n etc. all requiring less than const $n^{\log 7}$ arithmetical operations.

This fact should be compared with the result of KLYUYEV and KOKOVKIN-SHCHERBAK [1] that Gaussian elimination for solving a system of linear equations is optimal if one restricts oneself to operations upon rows and columns as a whole. We also note that WINOGRAD [2] modifies the usual algorithms for matrix multiplication and inversion and for solving systems of linear equations, trading roughly half of the multiplications for additions and subtractions.

It is a pleasure to thank D. BRILLINGER for inspiring discussions about the present subject and ST. COOK and B. PARLETT for encouraging me to write this paper.

2. We define algorithms $\alpha_{m,k}$ which multiply matrices of order $m2^k$, by induction on k: $\alpha_{m,0}$ is the usual algorithm for matrix multiplication (requiring m^3 multiplications and $m^2(m-1)$ additions). $\alpha_{m,k}$ already being known, define $\alpha_{m,k+1}$ as follows:

If A, B are matrices of order $m2^{k+1}$ to be multiplied, write

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad AB = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix},$$

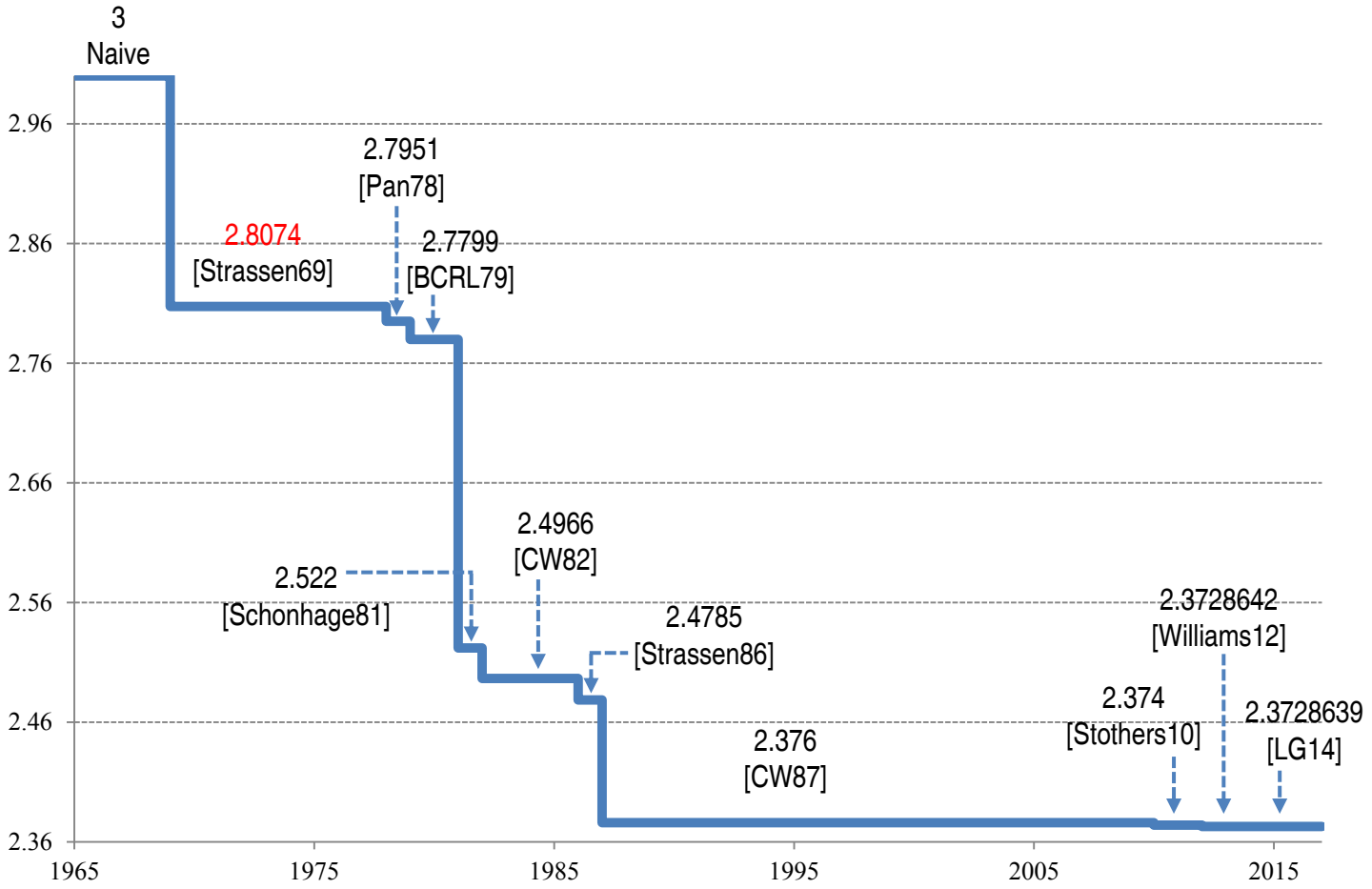
where the A_{ik}, B_{ik}, C_{ik} are matrices of order $m2^k$. Then compute

$$\begin{aligned} \text{I} &= (A_{11}+A_{22})(B_{11}+B_{22}), \\ \text{II} &= (A_{21}+A_{22})B_{11}, \\ \text{III} &= A_{11}(B_{12}-B_{22}), \\ \text{IV} &= A_{22}(-B_{11}+B_{21}), \\ \text{V} &= (A_{11}+A_{12})B_{22}, \\ \text{VI} &= (-A_{11}+A_{21})(B_{11}+B_{12}), \\ \text{VII} &= (A_{12}-A_{22})(B_{21}+B_{22}), \end{aligned}$$

* The results have been found while the author was at the Department of Statistics of the University of California, Berkeley. The author wishes to thank the National Science Foundation for their support (NSF GP-7454).

50 Years of History of Fast Matrix Multiplication (in theory)

FMM: Matrix multiplication algorithms with asymptotic complexity $< O(n^3)$



Exponent ω for the arithmetic complexity of fast matrix multiplication $O(O^{\omega})$

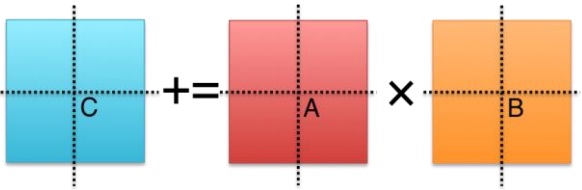
Jean-Guillaume Dumas, and Victor Pan. "Fast matrix multiplication and symbolic computation." *arXiv preprint arXiv:1612.05766* (2016).

40 Years of FMM implementations (in practice)

- Layering FMM upon standard GEMM calls
  - IBM ESSL, Huss-Lederman et al'96, Douglas et al'94
- Lower level implementation
  - D'Alberto & Nicolau'09
- Sequential
  - Pan'78, Bailey'88, Laderman et al'92, Douglas et al'94, Huss-Lederman et al'96
- Parallelization for shared memory
  - Kumar et al'95, D'Alberto & Nicolau'09, D'Alberto et al'11, Benson & Ballard'15
- Parallelization for distributed memory
  - Luo & Drake'95, Grayson & van de Geijn'96, Lipshitz et al'12
- Parallelization for GPUs
  - Li at el'11, Lai et al'13

Motivations

Problems with conventional FMM implementations



Conventional Implementations

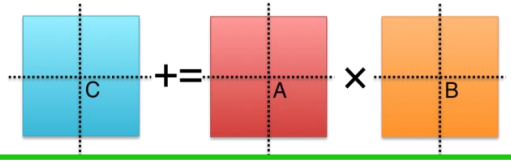
Matrix Size	**Must be large**	☹
Matrix Shape	**Must be square**	☹
No Additional Workspace	✗	☹
Parallelism	**Usually task parallelism**	☹

One-level Strassen's Algorithm (In theory)

Assume m, n, and k are all even. A, B, and C are $m \times k$, $k \times n$, $m \times n$ matrices, respectively. Letting

$$C = \begin{pmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{pmatrix}, A = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix}, B = \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix}$$

We can compute $C := C + AB$ by



Direct Computation

$$\begin{aligned} C_{00} &:= A_{00}B_{00} + A_{01}B_{10} + C_{00}; \\ C_{01} &:= A_{00}B_{01} + A_{01}B_{11} + C_{01}; \\ C_{10} &:= A_{10}B_{00} + A_{11}B_{10} + C_{10}; \\ C_{11} &:= A_{10}B_{01} + A_{11}B_{11} + C_{11}; \end{aligned}$$

8 multiplications, 8 additions

Strassen's Algorithm

$$\begin{aligned} M_0 &:= (A_{00} + A_{11})(B_{00} + B_{11}); \\ M_1 &:= (A_{10} + A_{11})B_{00}; \\ M_2 &:= A_{00}(B_{01} - B_{11}); \\ M_3 &:= A_{11}(B_{10} - B_{00}); \\ M_4 &:= (A_{00} + A_{01})B_{11}; \\ M_5 &:= (A_{10} - A_{00})(B_{00} + B_{01}); \\ M_6 &:= (A_{01} - A_{11})(B_{10} + B_{11}); \\ C_{00} &:= M_0 + M_3 - M_4 + M_7 + C_{00}; \\ C_{01} &:= M_2 + M_4 + C_{01}; \\ C_{10} &:= M_1 + M_3 + C_{10}; \\ C_{11} &:= M_0 - M_1 + M_2 + M_5 + C_{11}. \end{aligned}$$

7 multiplications, 22 additions

*Strassen, Volker. "Gaussian elimination is not optimal." *Numerische Mathematik* 13, no. 4 (1969): 354-356.

Multi-level Strassen's Algorithm (In theory)

$M_0 := (A_{00}+A_{11})(B_{00}+B_{11});$
$M_1 := (A_{10}+A_{11})B_{00};$
$M_2 := A_{00}(B_{01}-B_{11});$
$M_3 := A_{11}(B_{10}-B_{00});$
$M_4 := (A_{00}+A_{01})B_{11};$
$M_5 := (A_{10}-A_{00})(B_{00}+B_{01});$
$M_6 := (A_{01}-A_{11})(B_{10}+B_{11});$
$C_{00} \mathrel{+}= M_0 + M_3 - M_4 + M_6$
$C_{01} \mathrel{+}= M_2 + M_4$
$C_{10} \mathrel{+}= M_1 + M_3$
$C_{11} \mathrel{+}= M_0 - M_1 + M_2 + M_5$

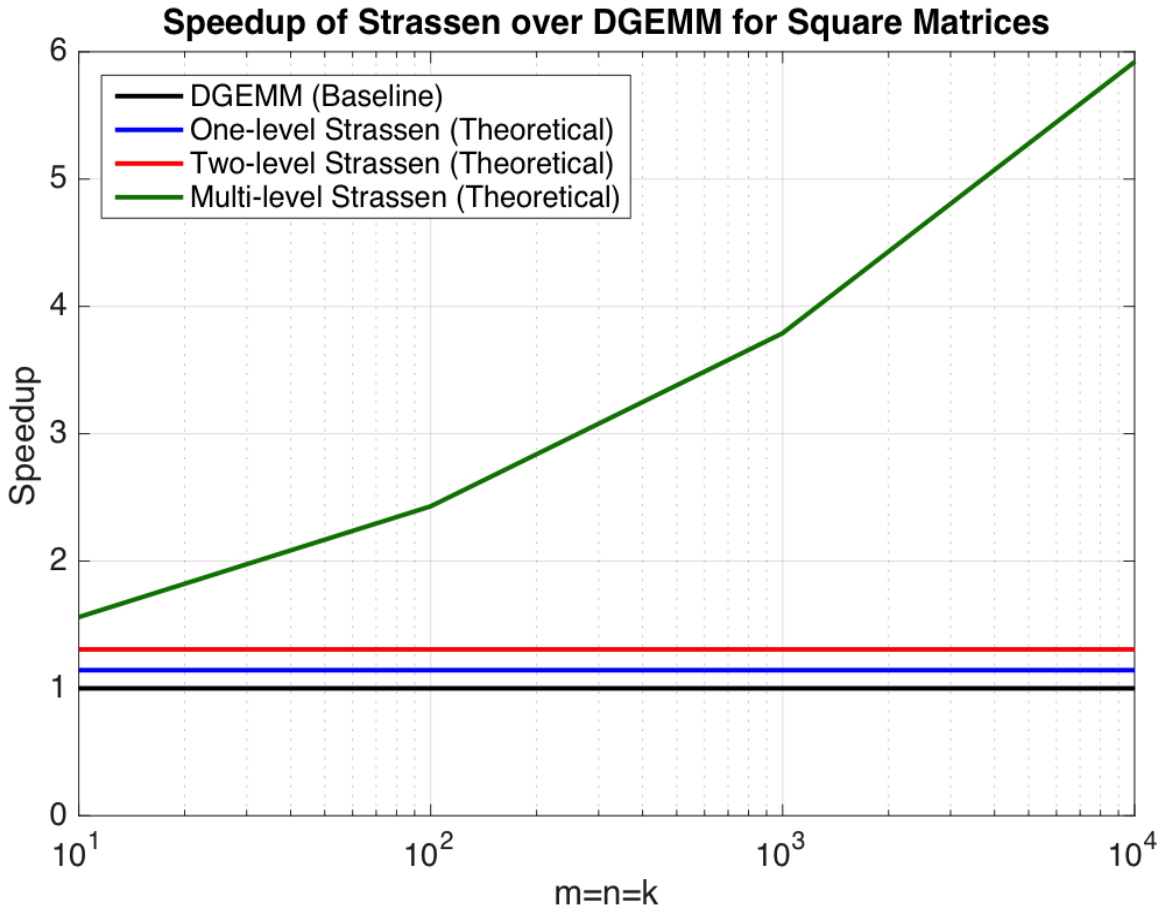
- One-level Strassen (1.14x speedup)
  - 8 multiplications → 7 multiplications ;
- Two-level Strassen (1.30x speedup)
  - 64 multiplications → 49 multiplications;
- *d*-level Strassen ($n^3/n^{2.803}$ speedup)
  - 8^d multiplications → 7^d multiplications;

If originally $m = n = k = 2^d$, where d is an integer, then the cost becomes
$(7/8)^{\log_2(n)}\, 2n^3 = n^{\log_2(7/8)} 2n^3 \approx 2n^{2.807}$ flops.

Multi-level Strassen's Algorithm (In theory)

$M_0 := (A_{00}+A_{11})(B_{00}+B_{11});$
$M_1 := (A_{10}+A_{11})B_{00};$
$M_2 := A_{00}(B_{01}-B_{11});$
$M_3 := A_{11}(B_{10}-B_{00});$
$M_4 := (A_{00}+A_{01})B_{11};$
$M_5 := (A_{10}-A_{00})(B_{00}+B_{01});$
$M_6 := (A_{01}-A_{11})(B_{10}+B_{11});$
$C_{00} \mathrel{+}= M_0 + M_3 - M_4 + M_6$
$C_{01} \mathrel{+}= M_2 + M_4$
$C_{10} \mathrel{+}= M_1 + M_3$
$C_{11} \mathrel{+}= M_0 - M_1 + M_2 + M_5$

- One-level Strassen (1.14x speedup)
  - 8 multiplications → 7 multiplications ;
- Two-level Strassen (1.30x speedup)
  - 64 multiplications → 49 multiplications;
- *d*-level Strassen ($n^3/n^{2.803}$ speedup)
  - 8^d multiplications → 7^d multiplications;



Strassen's Algorithm (In practice)

$M_0 := (A_{00}+A_{11})(B_{00}+B_{11});$

$M_1 := (\mathrm{A}_{10}+\mathrm{A}_{11})\mathrm{B}_{00};$

$M_2 := A_{00}(B_{01}-B_{11});$

$M_3 := A_{11}(B_{10}-B_{00});$

$M_4 := (\mathrm{A}_{00}+\mathrm{A}_{01})\mathrm{B}_{11};$

$M_5 := (A_{10}-A_{00})(B_{00}+B_{01});$

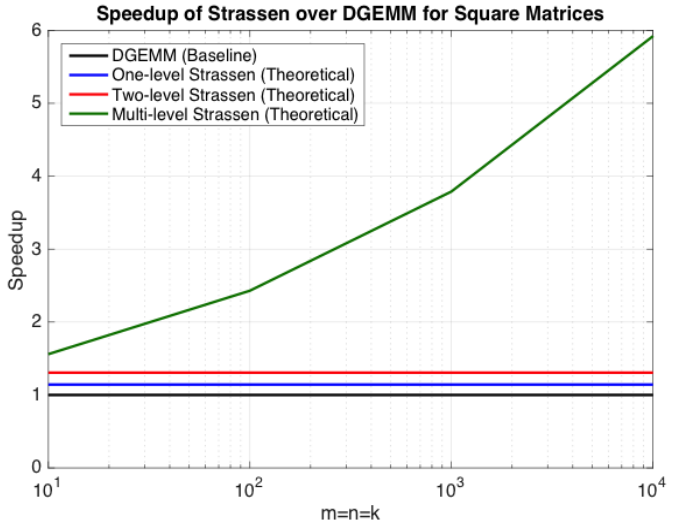
$M_6 := (A_{01}-A_{11})(B_{10}+B_{11});$

$C_{00} \mathrel{+}= M_0 + M_3 - M_4 + M_6$

$C_{01} \mathrel{+}= M_2 + M_4$

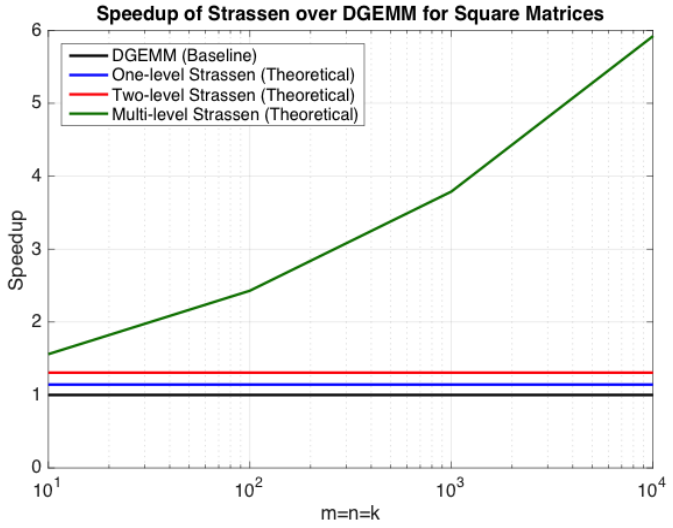
$C_{10} \mathrel{+}= M_1 + M_3$

$C_{11} \mathrel{+}= M_0 - M_1 + M_2 + M_5$



Strassen's Algorithm (In practice)

$$M_0 := (\overbrace{A_{00}+A_{11}}^{T_0})(\overbrace{B_{00}+B_{11}}^{T_1});$$
$$M_1 := (\overbrace{A_{10}+A_{11}}^{T_2})B_{00};$$
$$M_2 := A_{00}(\overbrace{B_{01}-B_{11}}^{T_3});$$
$$M_3 := A_{11}(\overbrace{B_{10}-B_{00}}^{T_4});$$
$$M_4 := (\overbrace{A_{00}+A_{01}}^{T_5})B_{11};$$
$$M_5 := (\overbrace{A_{10}-A_{00}}^{T_6})(\overbrace{B_{00}+B_{01}}^{T_7});$$
$$M_6 := (\overbrace{A_{01}-A_{11}}^{T_8})(\overbrace{B_{10}+B_{11}}^{T_9});$$
$$C_{00} \mathrel{+}= M_0 + M_3 - M_4 + M_6$$
$$C_{01} \mathrel{+}= M_2 + M_4$$
$$C_{10} \mathrel{+}= M_1 + M_3$$
$$C_{11} \mathrel{+}= M_0 - M_1 + M_2 + M_5$$



- One-level Strassen (1.14x speedup)
  - 7 multiplications + 22 additions;
- Two-level Strassen (1.30x speedup)
  - 49 multiplications + 334 additions;

Strassen's Algorithm (In practice)

$M_0 := (\overbrace{A_{00}+A_{11}}^{T_0})(\overbrace{B_{00}+B_{11}}^{T_1});$

$M_1 := (\overbrace{A_{10}+A_{11}}^{T_2})B_{00};$

$M_2 := A_{00}(\overbrace{B_{01}-B_{11}}^{T_3});$

$M_3 := A_{11}(\overbrace{B_{10}-B_{00}}^{T_4});$

$M_4 := (\overbrace{A_{00}+A_{01}}^{T_5})B_{11};$

$M_5 := (\overbrace{A_{10}-A_{00}}^{T_6})(\overbrace{B_{00}+B_{01}}^{T_7});$

$M_6 := (\overbrace{A_{01}-A_{11}}^{T_8})(\overbrace{B_{10}+B_{11}}^{T_9});$

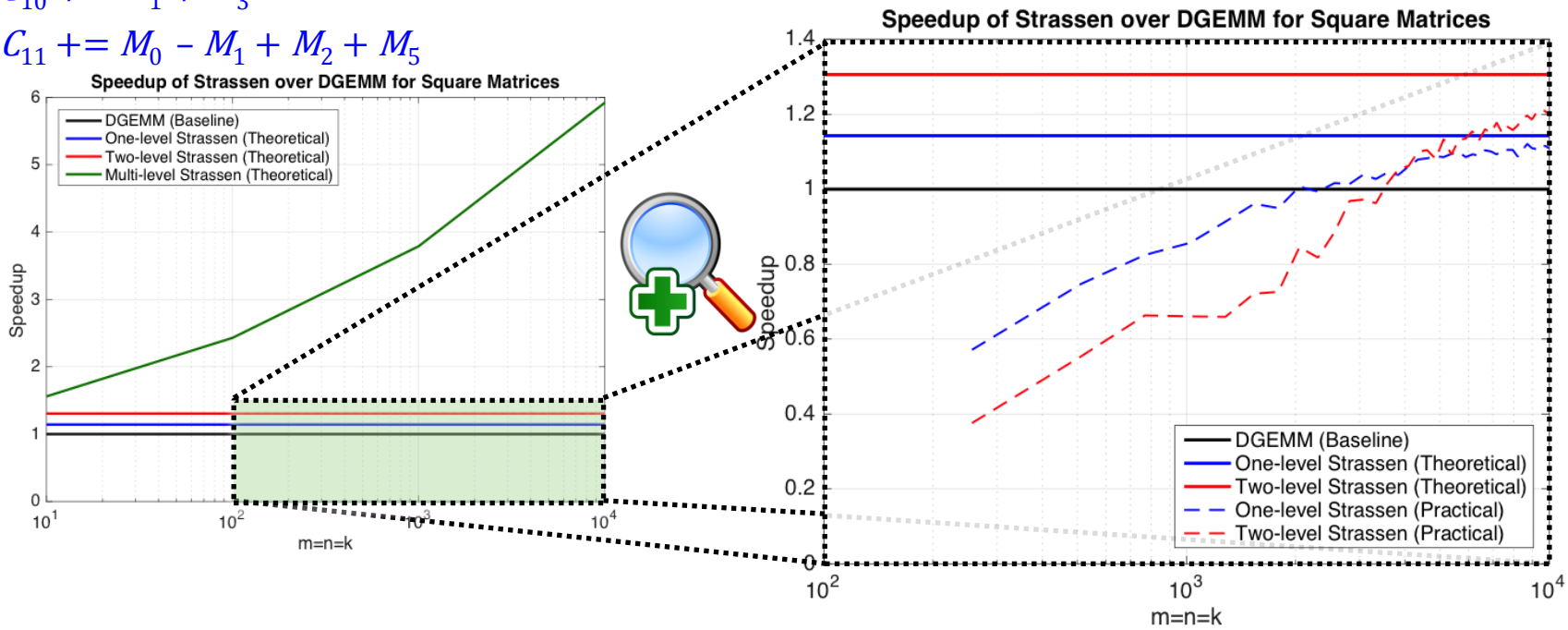
$C_{00} \mathrel{+}= M_0 + M_3 - M_4 + M_6$

$C_{01} \mathrel{+}= M_2 + M_4$

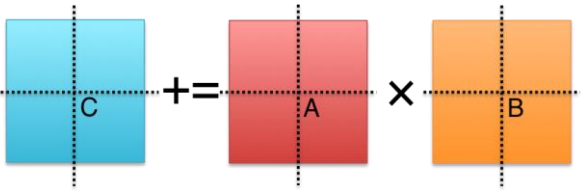
$C_{10} \mathrel{+}= M_1 + M_3$

$C_{11} \mathrel{+}= M_0 - M_1 + M_2 + M_5$

- One-level Strassen (1.14x speedup)
  - 7 multiplications + 22 additions;
- Two-level Strassen (1.30x speedup)
  - 49 multiplications + 334 additions;
- *d*-level Strassen ($n^3/n^{2.803}$ speedup)
  - Numerical unstable; Not achievable

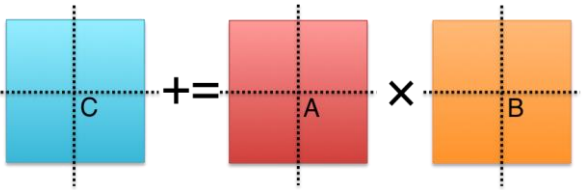


To achieve practical high performance of Strassen/FMM algorithms……



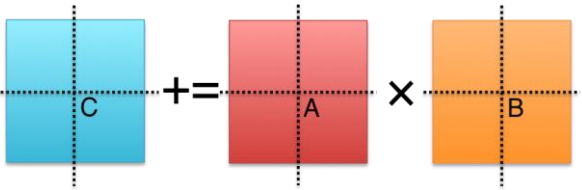
	Conventional Implementations	Our Solutions
Matrix Size	Must be large	
Matrix Shape	Must be square	
No Additional Workspace	✗	
Parallelism		

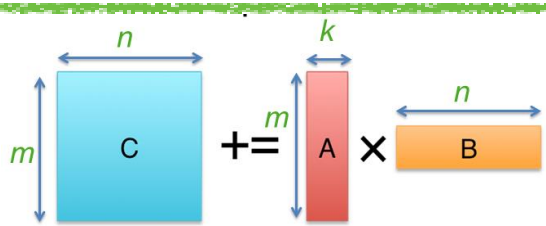
To achieve practical high performance of Strassen/FMM algorithms……



	Conventional Implementations	Our Solutions
Matrix Size	Must be large	
Matrix Shape	Must be square	
No Additional Workspace	✗	
Parallelism	Usually task parallelism	

To achieve practical high performance of Strassen/FMM algorithms……



	Conventional Implementations		Our Solutions	
Matrix Size	Must be large			
Matrix Shape	Must be square			
No Additional Workspace				
Parallelism	Usually task parallelism		Can be data parallelism	

Proposed Work

- Practical high-performance Strassen.
  - Utilize a performance model to predict the relative performance

❑ Jianyu Huang, Tyler Smith, Greg Henry, and Robert van de Geijn. "Strassen's Algorithm Reloaded." In ***SC'16***.

- Practical fast matrix multiplication algorithms.
  - Leverage a code generator to generate ~200 implementations

❑ Jianyu Huang, Leslie Rice, Devin A. Matthews, Robert A. van de Geijn. "Generating Families of Practical Fast Matrix Multiplication Algorithms." In ***IPDPS17***

- Acceleration of tensor contractions via Strassen.
  - First paper to apply Strassen's algorithm for tensor contraction.

❑ Jianyu Huang, Devin A. Matthews, and Robert A. van de Geijn. "Strassen's Algorithm for Tensor Contraction." arXiv:1704.03092 (2017). In Submission.

- Exploration of Strassen/FMM for distributed memory.
- Fusion with machine learning applications.
- Pedagogical outreach for Strassen.

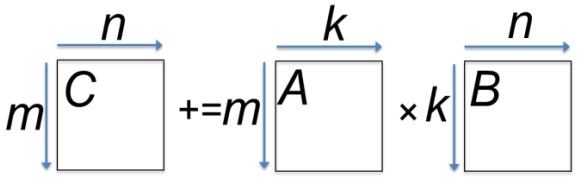
❑ Jianyu Huang, Robert A. van de Geijn. "BLISlab: A Sandbox for Optimizing GEMM." FLAME Working Note #80, The University of Texas at Austin, Department of Computer Science. Technical Report TR-16-13. August 31, 2016.

Outline

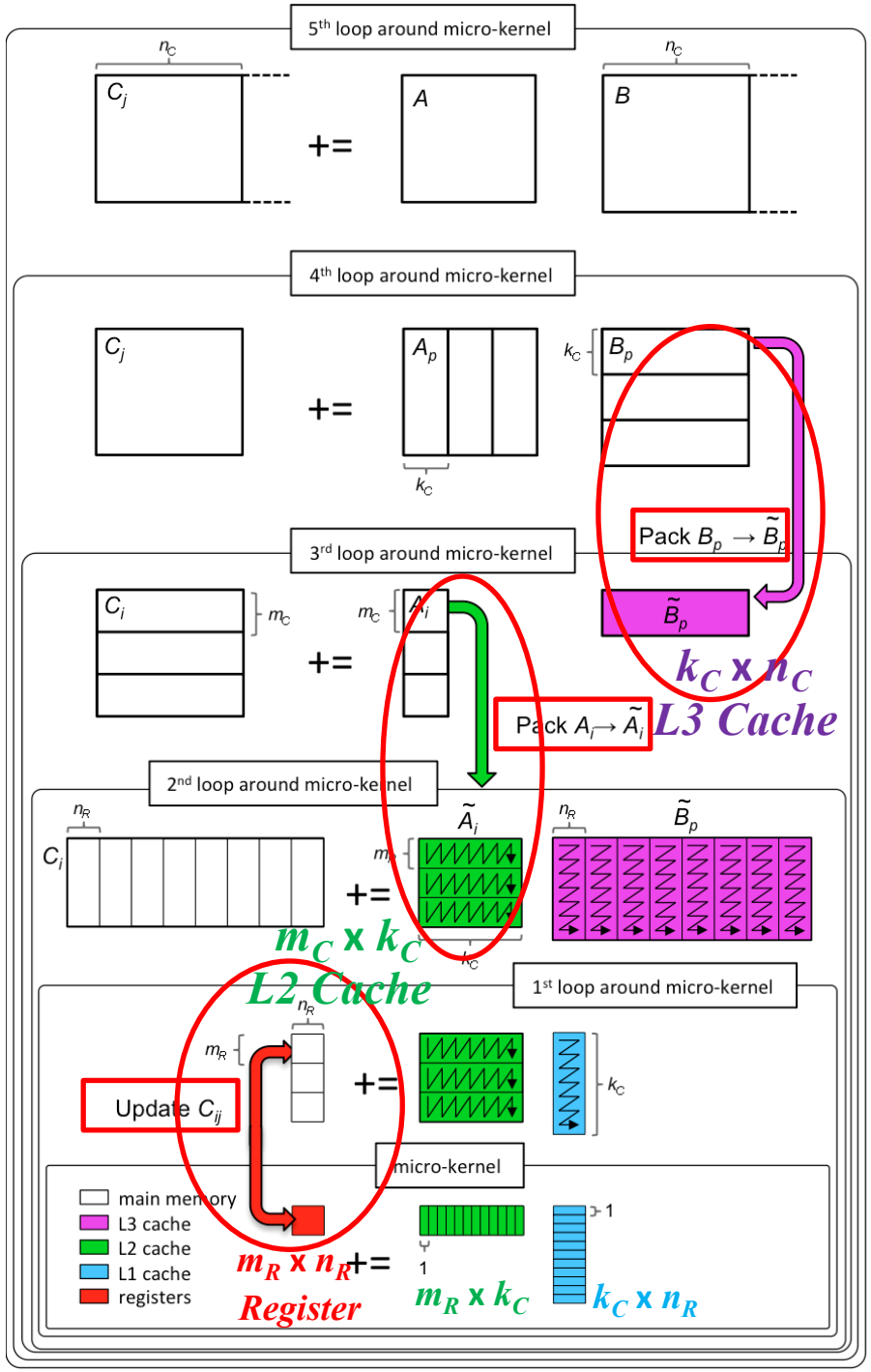
- Motivation
- Background
- Practical Strassen’s Algorithm (SC16)
- Practical Fast Matrix Multiplication (IPDPS17)
- Proposed Work
- Conclusion

High-performance matrix multiplication (GEMM)

State-of-the-art GEMM in BLIS



- BLAS-like Library Instantiation Software (BLIS) is a portable framework for instantiating BLAS-like dense linear algebra libraries.
  - ❑Field Van Zee, and Robert van de Geijn. "BLIS: A Framework for Rapidly Instantiating BLAS Functionality." *ACM TOMS* 41.3 (2015): 14.
- BLIS provides a refactoring of GotoBLAS algorithm (best-known approach on CPU) to implement GEMM.
  - ❑Kazushige Goto, and Robert van de Geijn. "High-performance implementation of the level-3 BLAS." *ACM TOMS* 35.1 (2008): 4.
  - ❑Kazushige Goto, and Robert van de Geijn. "Anatomy of high-performance matrix multiplication." *ACM TOMS* 34.3 (2008): 12.
- GEMM implementation in BLIS has 6-layers of loops. The outer 5 loops are written in C. The inner-most loop (micro-kernel) is written in assembly for high performance.



GotoBLAS algorithm for GEMM in BLIS

$$\underset{m \times n}{C} \mathrel{+}= \underset{m \times k}{A} \times \underset{k \times n}{B}$$

Loop 5 **for** $j_c = 0 : n-1$ **steps of** n_c
$\mathcal{J}_c = j_c : j_c + n_c - 1$
Loop 4 **for** $p_c = 0 : k-1$ **steps of** k_c
$\mathcal{P}_c = p_c : p_c + k_c - 1$
$B(\mathcal{P}_c, \mathcal{J}_c) \rightarrow \widetilde{B}_p$
Loop 3 **for** $i_c = 0 : m-1$ **steps of** m_c
$\mathcal{I}_c = i_c : i_c + m_c - 1$
$A(\mathcal{I}_c, \mathcal{P}_c) \rightarrow \widetilde{A}_i$
// macro-kernel
Loop 2 **for** $j_r = 0 : n_c - 1$ **steps of** n_r
$\mathcal{J}_r = j_r : j_r + n_r - 1$
Loop 1 **for** $i_r = 0 : m_c - 1$ **steps of** m_r
$\mathcal{I}_r = i_r : i_r + m_r - 1$
//micro-kernel
Loop 0 **for** $p_r = 0 : p_c - 1$ **steps of** 1
$C_c(\mathcal{I}_r, \mathcal{J}_r) \mathrel{+}= \widetilde{A}_i(\mathcal{I}_r, p_r)\, \widetilde{B}_p(p_r, \mathcal{J}_r)$
endfor
endfor
endfor
endfor
endfor
endfor

*Field G. Van Zee, and Tyler M. Smith. "Implementing high-performance complex matrix multiplication via the 3m and 4m methods." In *ACM Transactions on Mathematical Software (TOMS), accepted.*

Outline

- Motivation
- Background
- Practical Strassen’s Algorithm (SC16)
- Practical Fast Matrix Multiplication (IPDPS17)
- Proposed Work
- Conclusion

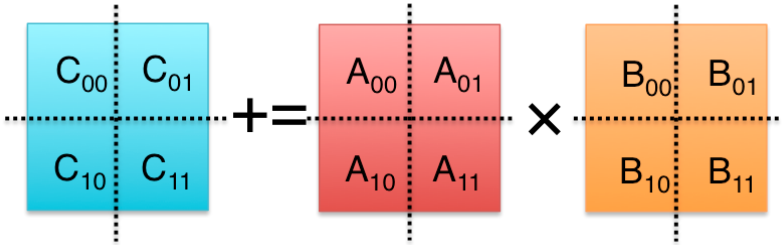
Practical Strassen's Algorithm

***Jianyu Huang**, Tyler Smith, Greg Henry, and Robert van de Geijn. "Strassen's Algorithm Reloaded." In *SC'16*.

Strassen's Algorithm Reloaded

$M_0 := (A_{00}+A_{11})(B_{00}+B_{11});$
$M_1 := (A_{10}+A_{11})B_{00};$
$M_2 := A_{00}(B_{01}-B_{11});$
$M_3 := A_{11}(B_{10}-B_{00});$
$M_4 := (A_{00}+A_{01})B_{11};$
$M_5 := (A_{10}-A_{00})(B_{00}+B_{01});$
$M_6 := (A_{01}-A_{11})(B_{10}+B_{11});$
$C_{00} \mathrel{+}= M_0 + M_3 - M_4 + M_6$
$C_{01} \mathrel{+}= M_2 + M_4$
$C_{10} \mathrel{+}= M_1 + M_3$
$C_{11} \mathrel{+}= M_0 - M_1 + M_2 + M_5$

$M_0 := (A_{00}+A_{11})(B_{00}+B_{11});$	$C_{00} \mathrel{+}= M_0;\ C_{11} \mathrel{+}= M_0;$
$M_1 := (A_{10}+A_{11})B_{00};$	$C_{10} \mathrel{+}= M_1;\ C_{11} \mathrel{-}= M_1;$
$M_2 := A_{00}(B_{01}-B_{11});$	$C_{01} \mathrel{+}= M_2;\ C_{11} \mathrel{+}= M_2;$
$M_3 := A_{11}(B_{10}-B_{00});$	$C_{00} \mathrel{+}= M_3;\ C_{10} \mathrel{+}= M_3;$
$M_4 := (A_{00}+A_{01})B_{11};$	$C_{01} \mathrel{+}= M_4;\ C_{00} \mathrel{-}= M_4;$
$M_5 := (A_{10}-A_{00})(B_{00}+B_{01});$	$C_{11} \mathrel{+}= M_5;$
$M_6 := (A_{01}-A_{11})(B_{10}+B_{11});$	$C_{00} \mathrel{+}= M_6;$

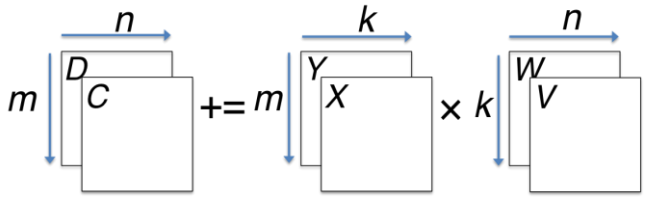


$M := (X+Y)(V+W);$ $C \mathrel{+}= M;$ $D \mathrel{+}= M;$

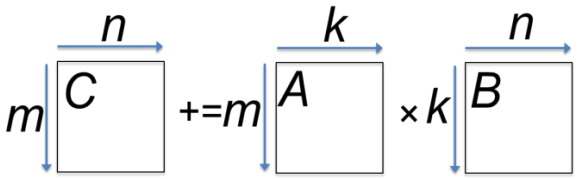
General operation for one-level Strassen:

$M := (X+\delta Y)(V+\varepsilon W);$ $C \mathrel{+}= \gamma_0 M;\ D \mathrel{+}= \gamma_1 M;$
$\gamma_0, \gamma_1, \delta, \varepsilon \in \{-1, 0, 1\}.$

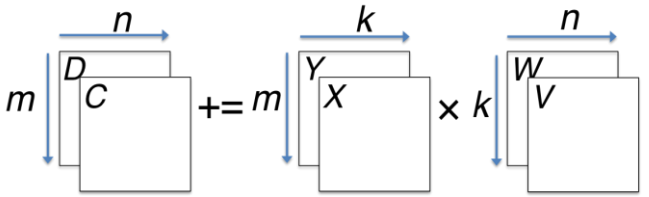
$M := (X + Y)(V + W);\quad C \mathrel{+}= M;\ D \mathrel{+}= M;$

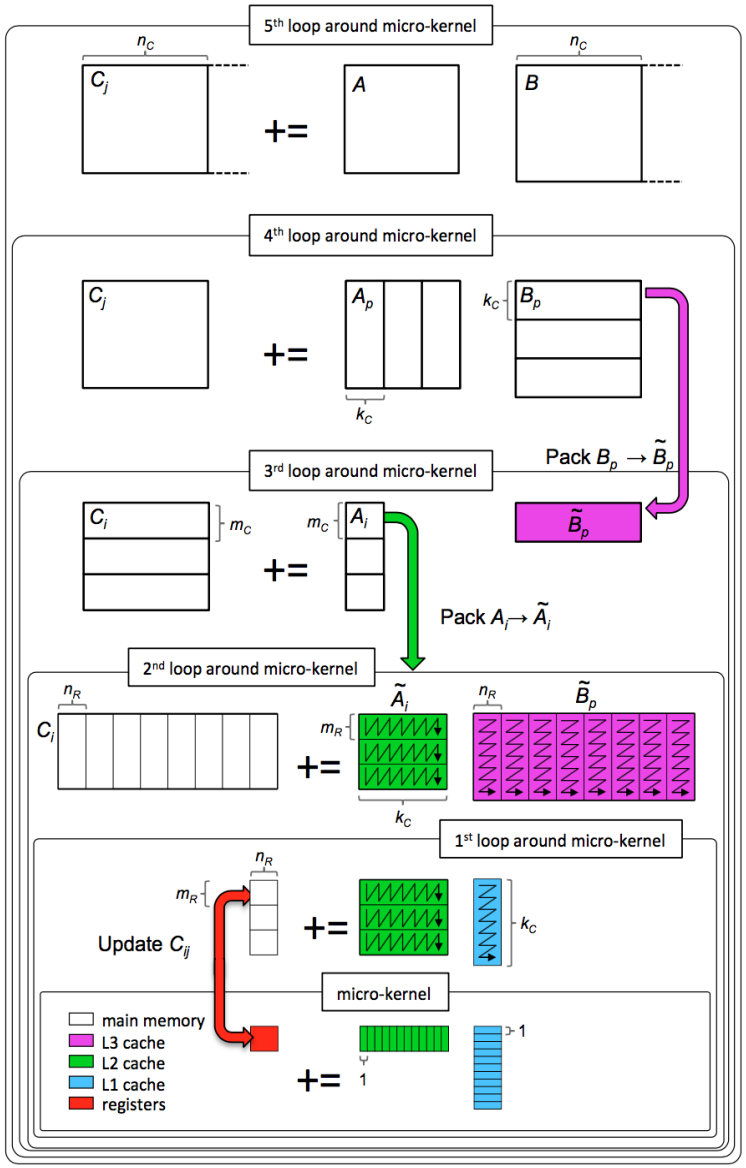


$C \mathrel{+}= AB;$

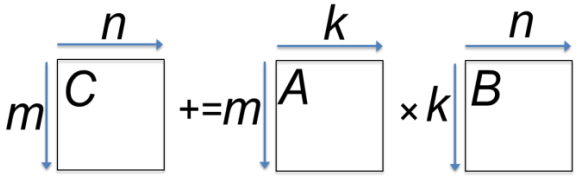


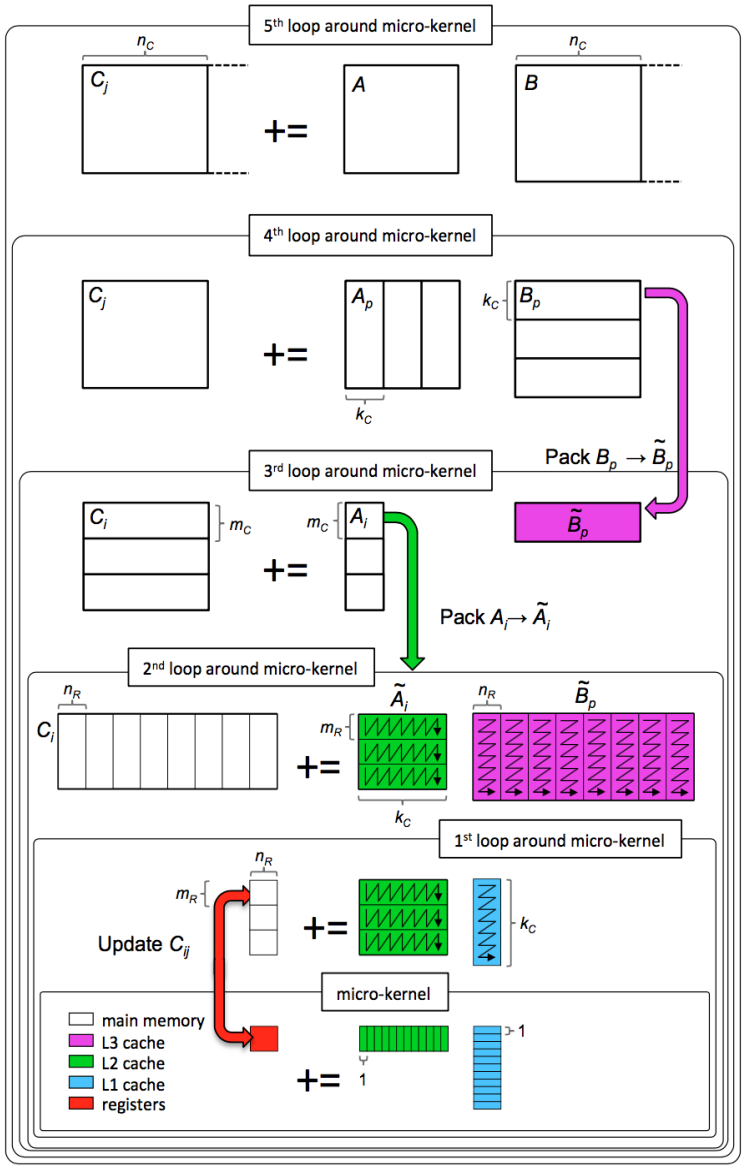
$M := (X+Y)(V+W);\quad C \mathrel{+}= M;\ D \mathrel{+}= M;$



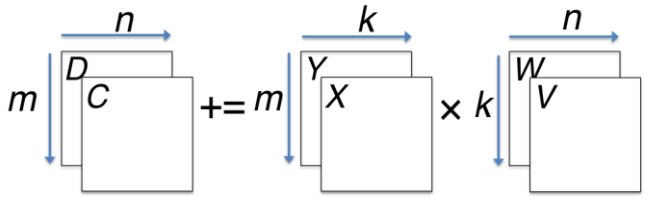


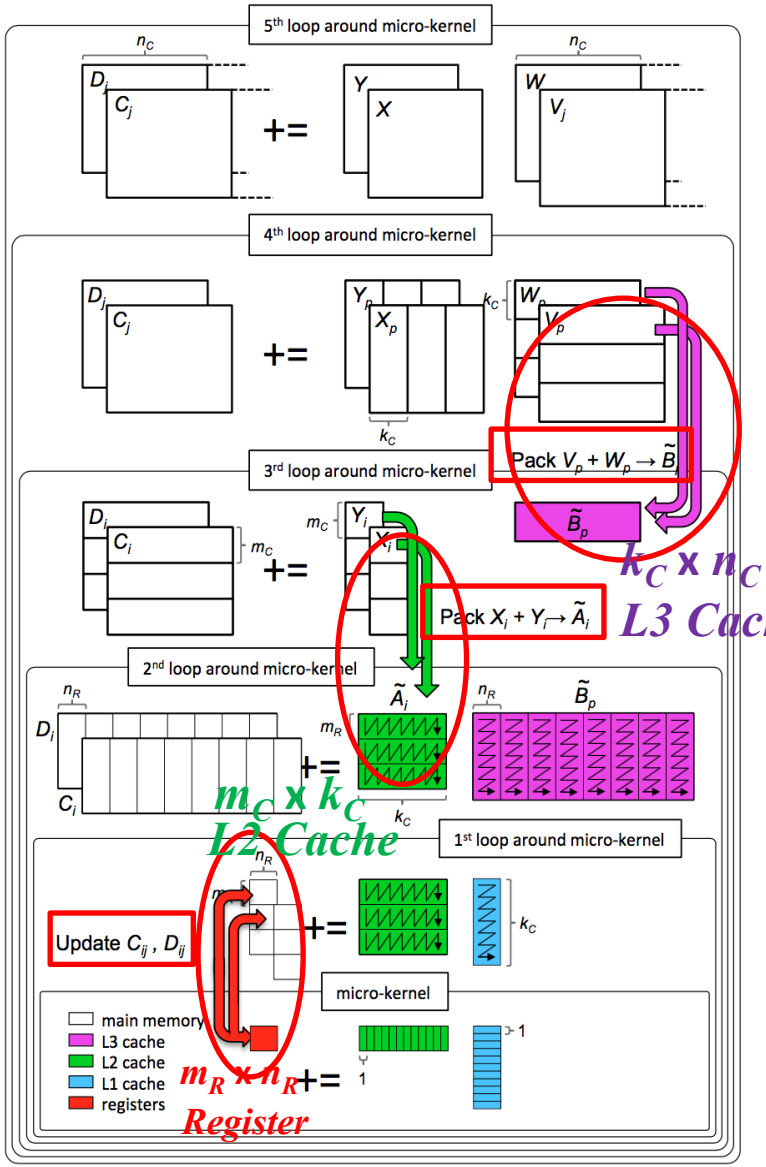
$C \mathrel{+}= AB;$

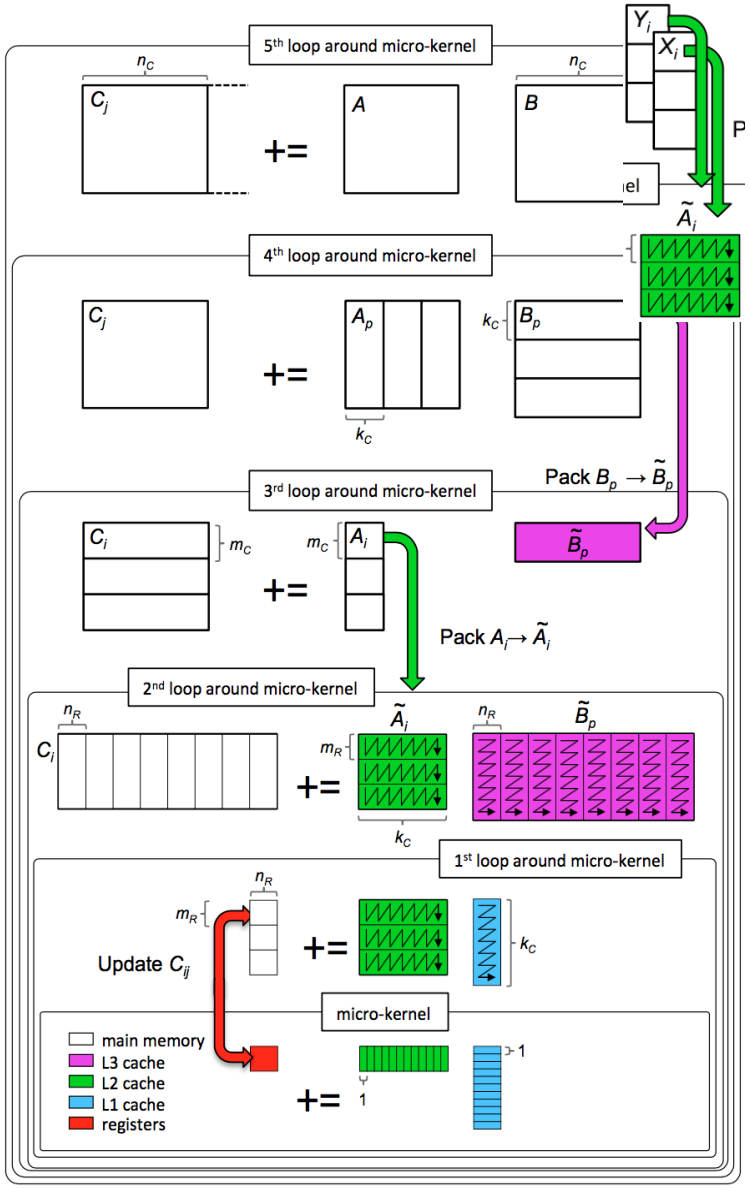


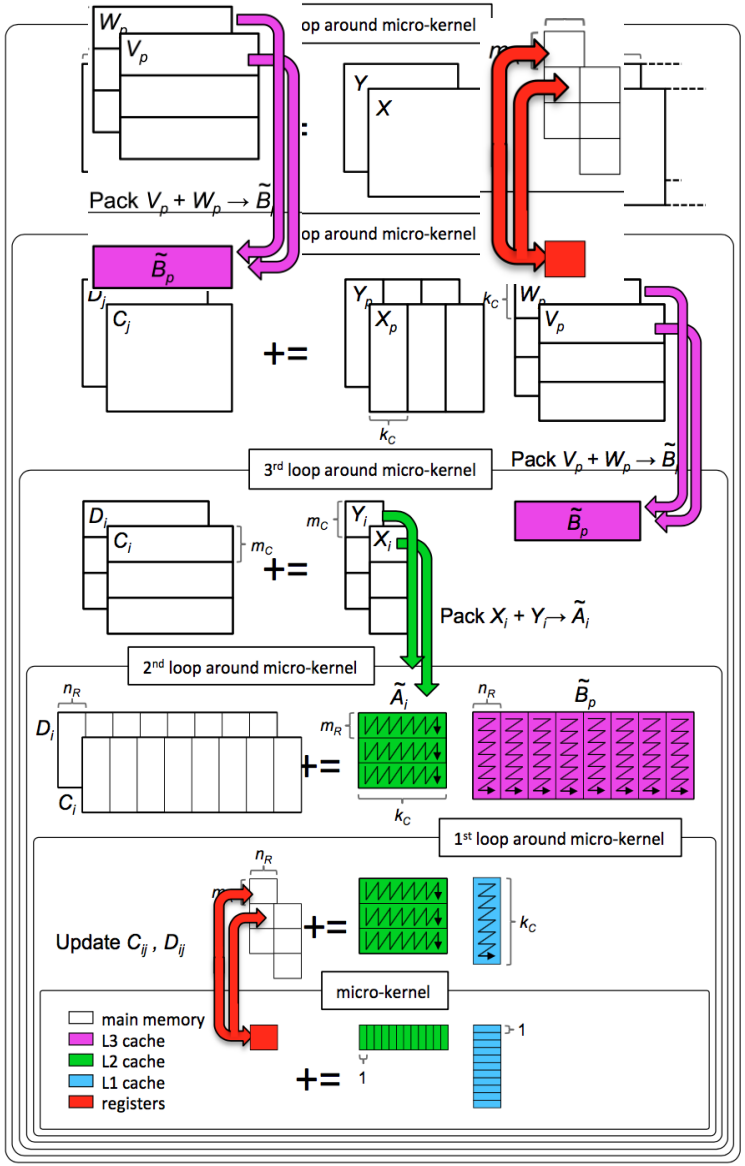


$M := (X+Y)(V+W);\ C \mathrel{+}= M;\ D \mathrel{+}= M;$

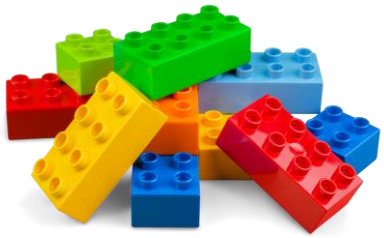


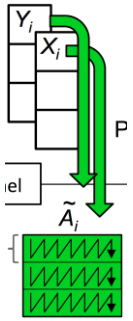


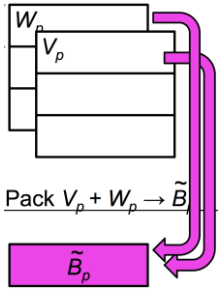

5th loop around micro-kernel
4th loop around micro-kernel
Pack $B_p \rightarrow \tilde{B}_p$
3rd loop around micro-kernel
Pack $A_i \rightarrow \tilde{A}_i$
2nd loop around micro-kernel
1st loop around micro-kernel
Update C_{ij}
micro-kernel
main memory
L3 cache
L2 cache
L1 cache
registers

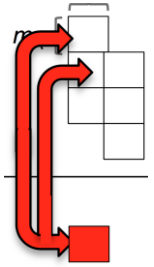

Pack $V_p + W_p \rightarrow \tilde{B}_p$
3rd loop around micro-kernel
Pack $X_i + Y_i \rightarrow \tilde{A}_i$
2nd loop around micro-kernel
1st loop around micro-kernel
Update C_{ij} , D_{ij}
micro-kernel
main memory
L3 cache
L2 cache
L1 cache
registers

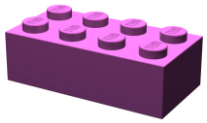
Variations on a theme











- Naïve Strassen ✘ ✘ ✘
- AB Strassen ✔ ✔ ✘
- ABC Strassen ✔ ✔ ✔

Performance Model

- Performance Metric

$$\textit{Effective}\ \text{GFLOPS} = \frac{2 \cdot m \cdot n \cdot k}{\text{time (in seconds)}} \cdot 10^{-9}$$

- Total Time Breakdown

$$T = T_a + T_m$$

Arithmetic Operations

Memory Operations

Arithmetic Operations

$$T_a = T_a^{\times} + T_a^{A+} + T_a^{B+} + T_a^{C+}$$

Submatrix multiplication

Extra additions with submatrices of A, B, C, respectively

- DGEMM
  - No extra additions

$$T_a = 2mnk \cdot \tau_a$$

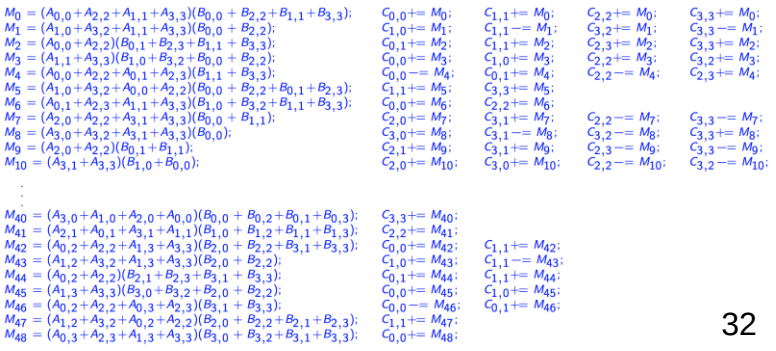
- One-level Strassen (ABC, AB, Naïve)
  - 7 submatrix multiplications
  - 5 extra additions of submatrices of A and B
  - 12 extra additions of submatrices of C

$$T_a = (7 \times 2\frac{m}{2}\frac{n}{2}\frac{k}{2} + 5 \times 2\frac{m}{2}\frac{k}{2} + 5 \times 2\frac{k}{2}\frac{n}{2} + 12 \times 2\frac{m}{2}\frac{n}{2}) \cdot \tau_a$$

- Two-level Strassen (ABC, AB, Naïve)
  - 49 submatrix multiplications
  - 95 extra additions of submatrices of A and B
  - 144 extra additions of submatrices of C

$$T_a = (49 \times 2\frac{m}{4}\frac{n}{4}\frac{k}{4} + 95 \times 2\frac{m}{4}\frac{k}{4} + 95 \times 2\frac{k}{4}\frac{n}{4} + 144 \times 2\frac{m}{4}\frac{n}{4}) \cdot \tau_a$$

$M_0 := (A_{00}+A_{11})(B_{00}+B_{11})$; $C_{00} \mathrel{+}= M_0$; $C_{11} \mathrel{+}= M_0$;
$M_1 := (A_{10}+A_{11})B_{00}$; $C_{10} \mathrel{+}= M_1$; $C_{11} \mathrel{-}= M_1$;
$M_2 := A_{00}(B_{01}-B_{11})$; $C_{01} \mathrel{+}= M_2$; $C_{11} \mathrel{+}= M_2$;
$M_3 := A_{11}(B_{10}-B_{00})$; $C_{00} \mathrel{+}= M_3$; $C_{10} \mathrel{+}= M_3$;
$M_4 := (A_{00}+A_{01})B_{11}$; $C_{01} \mathrel{+}= M_4$; $C_{00} \mathrel{-}= M_4$;
$M_5 := (A_{10}-A_{00})(B_{00}+B_{01})$; $C_{11} \mathrel{+}= M_5$;
$M_6 := (A_{01}-A_{11})(B_{10}+B_{11})$; $C_{00} \mathrel{+}= M_6$;



Memory Operations

$$T_m = N_m^{A\times}\cdot T_m^{A\times} + N_m^{B\times}\cdot T_m^{B\times} + N_m^{C\times}\cdot T_m^{C\times} + N_m^{A+}\cdot T_m^{A+} + N_m^{B+}\cdot T_m^{B+} + N_m^{C+}\cdot T_m^{C+}$$

- DGEMM

$$T_m = (1\cdot mk\lceil\frac{n}{n_c}\rceil + 1\cdot nk + 1\cdot 2\lambda mn\lceil\frac{k}{k_c}\rceil)\cdot\tau_b$$

- One-level

➢ ABC Strassen

$$T_m = (12\cdot\frac{m}{2}\frac{k}{2}\lceil\frac{n/2}{n_c}\rceil + 12\cdot\frac{n}{2}\frac{k}{2} + 12\cdot 2\lambda\frac{m}{2}\frac{n}{2}\lceil\frac{k/2}{k_c}\rceil)\cdot\tau_b$$

➢ AB Strassen

$$T_m = (12\cdot\frac{m}{2}\frac{k}{2}\lceil\frac{n/2}{n_c}\rceil + 12\cdot\frac{n}{2}\frac{k}{2} + 7\cdot 2\lambda\frac{m}{2}\frac{n}{2}\lceil\frac{k/2}{k_c}\rceil + 36\cdot\frac{m}{2}\frac{n}{2})\cdot\tau_b$$

➢ Naïve Strassen

$$T_m = (7\cdot\frac{m}{2}\frac{k}{2}\lceil\frac{n/2}{n_c}\rceil + 7\cdot\frac{n}{2}\frac{k}{2} + 7\cdot 2\lambda\frac{m}{2}\frac{n}{2}\lceil\frac{k/2}{k_c}\rceil + 19\cdot\frac{m}{2}\frac{k}{2} + 19\cdot\frac{n}{2}\frac{k}{2} + 36\cdot\frac{m}{2}\frac{n}{2})\cdot\tau_b$$

- Two-level

➢ ABC Strassen

$$T_m = (194\cdot\frac{m}{4}\frac{k}{4}\lceil\frac{n/4}{n_c}\rceil + 194\cdot\frac{n}{4}\frac{k}{4} + 144\cdot 2\lambda\frac{m}{4}\frac{n}{4}\lceil\frac{k/4}{k_c}\rceil)\cdot\tau_b$$

➢ AB Strassen

$$T_m = (194\cdot\frac{m}{4}\frac{k}{4}\lceil\frac{n/4}{n_c}\rceil + 194\cdot\frac{n}{4}\frac{k}{4} + 49\cdot 2\lambda\frac{m}{4}\frac{n}{4}\lceil\frac{k/4}{k_c}\rceil + 432\cdot\frac{m}{4}\frac{n}{4})\cdot\tau_b$$

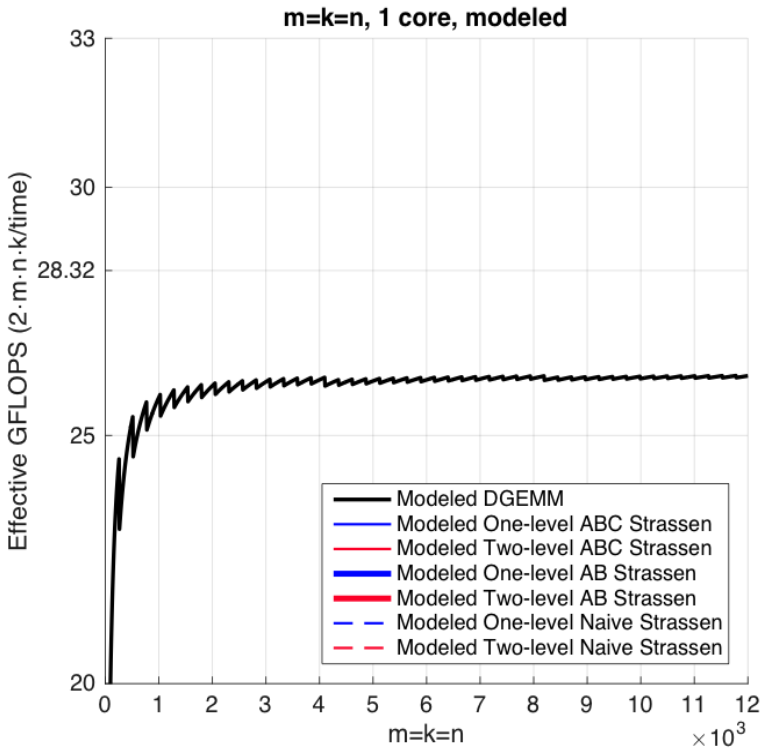
➢ Naïve Strassen

$$T_m = (49\cdot\frac{m}{4}\frac{k}{4}\lceil\frac{n/4}{n_c}\rceil + 49\cdot\frac{n}{4}\frac{k}{4} + 49\cdot 2\lambda\frac{m}{4}\frac{n}{4}\lceil\frac{k/4}{k_c}\rceil + 293\cdot\frac{m}{4}\frac{k}{4} + 293\cdot\frac{n}{4}\frac{k}{4} + 432\cdot\frac{m}{4}\frac{n}{4})\cdot\tau_b$$

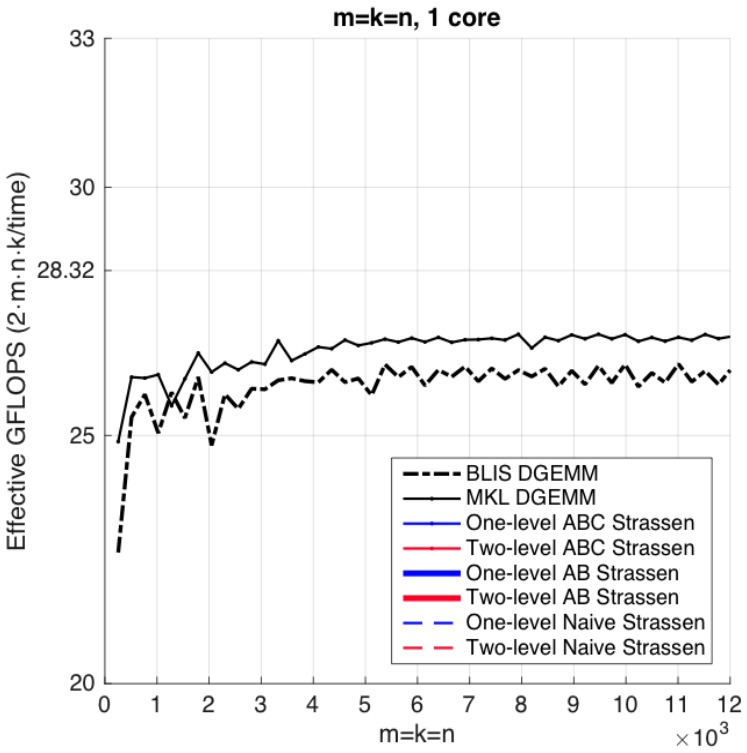
Modeled and Actual Performance on Single Core

Observation (Square Matrices)

Modeled Performance

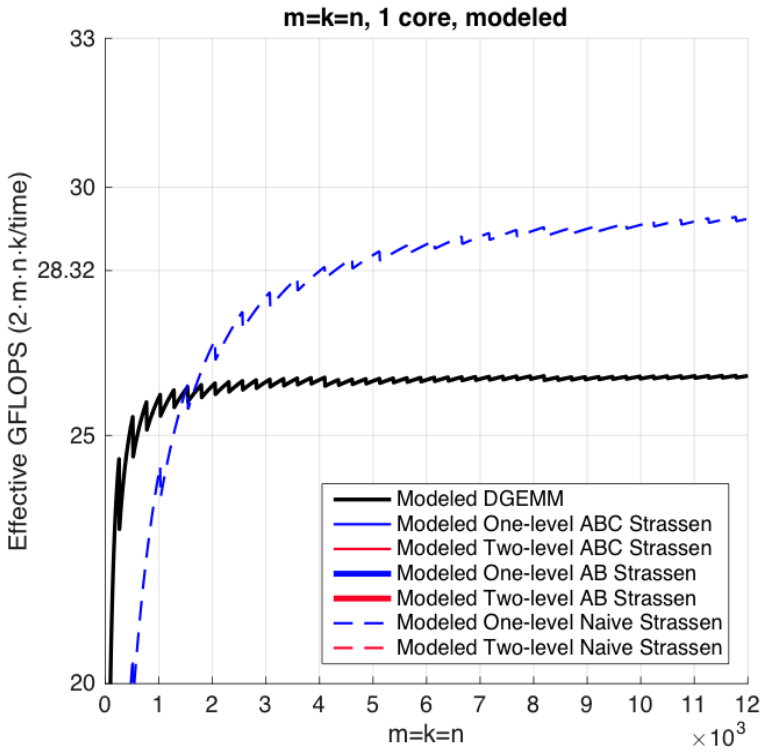


Actual Performance

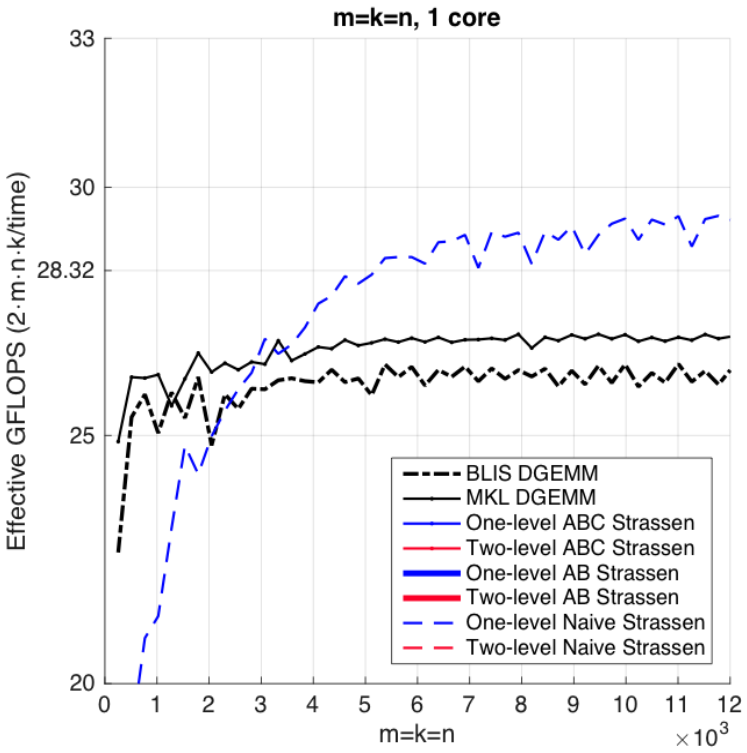


Observation (Square Matrices)

Modeled Performance

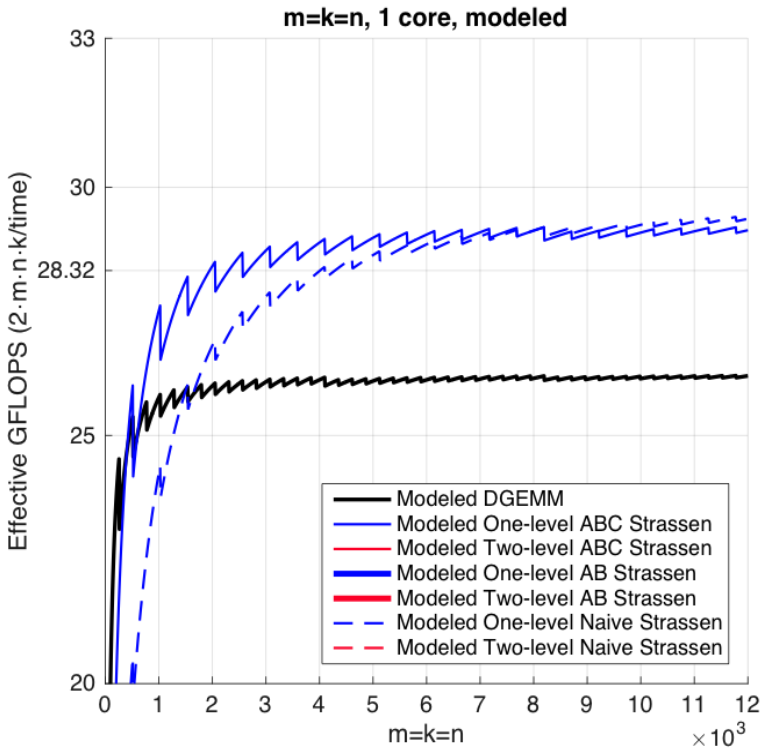


Actual Performance

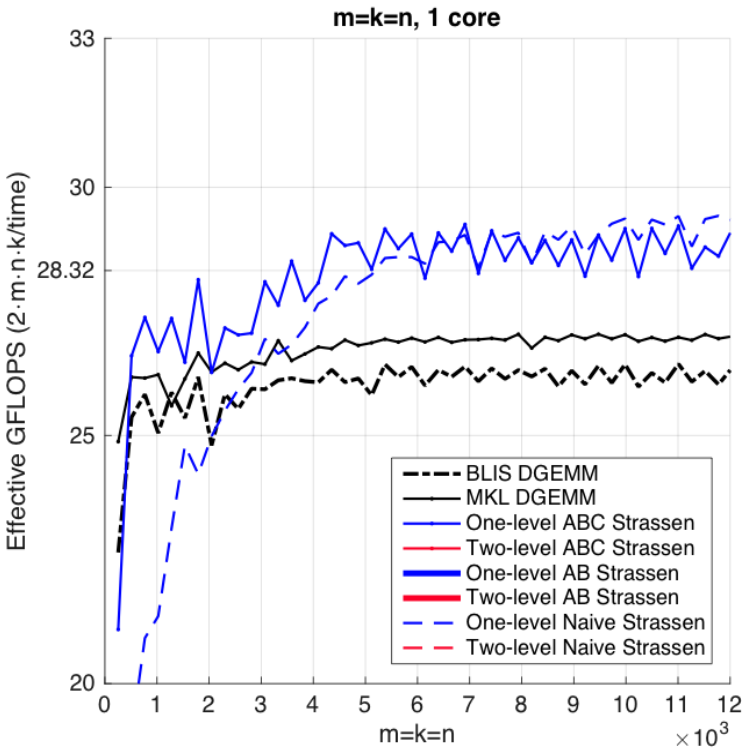


Observation (Square Matrices)

Modeled Performance

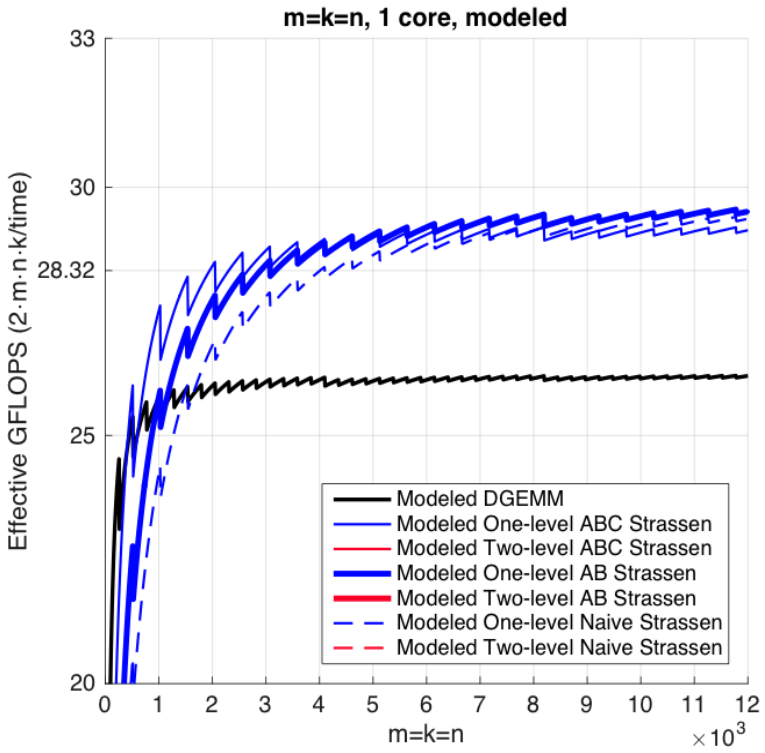


Actual Performance

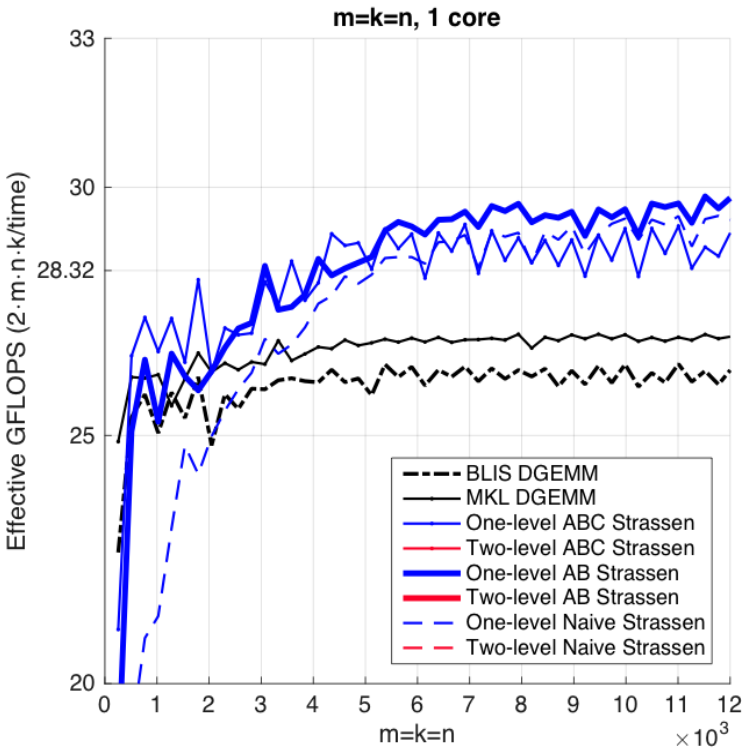


Observation (Square Matrices)

Modeled Performance

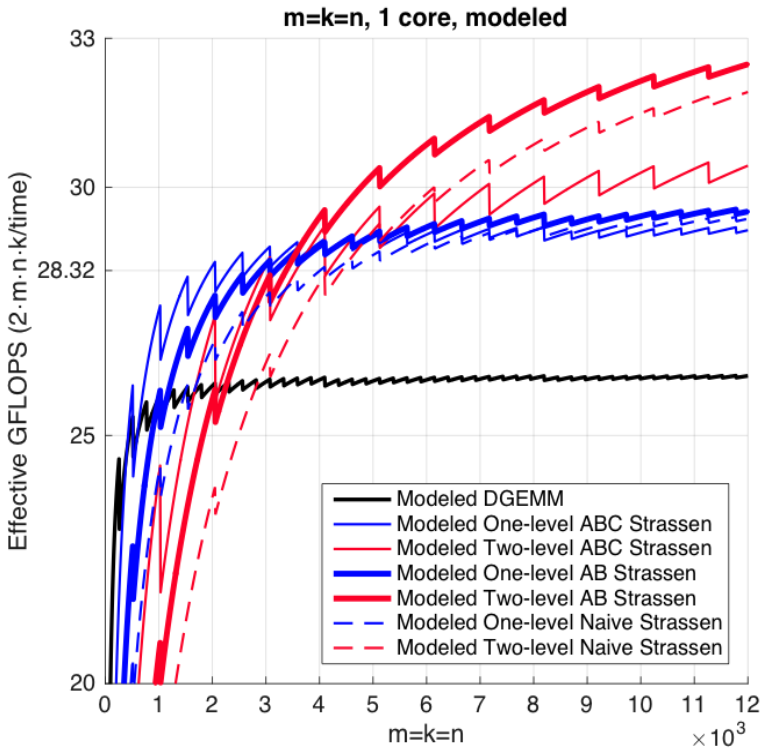


Actual Performance

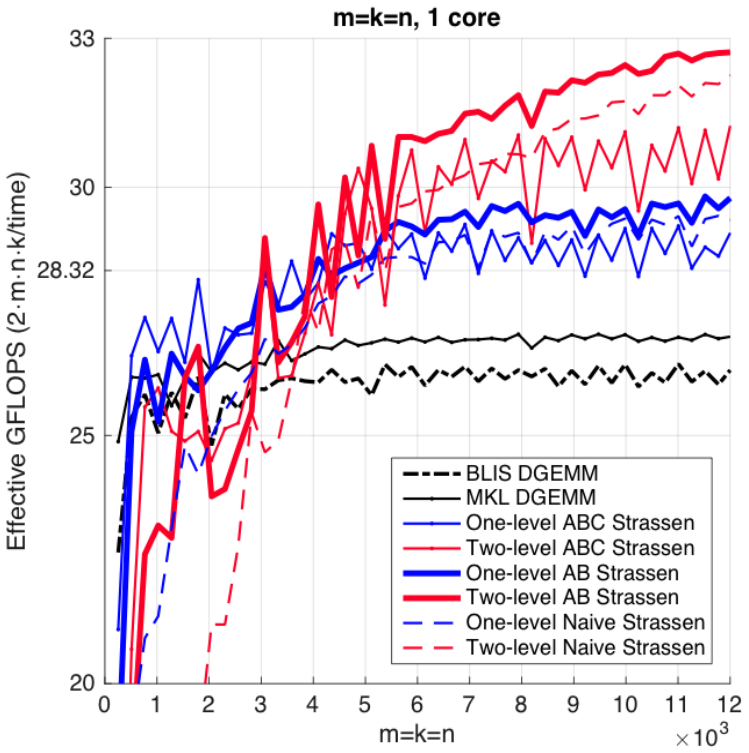


Observation (Square Matrices)

Modeled Performance

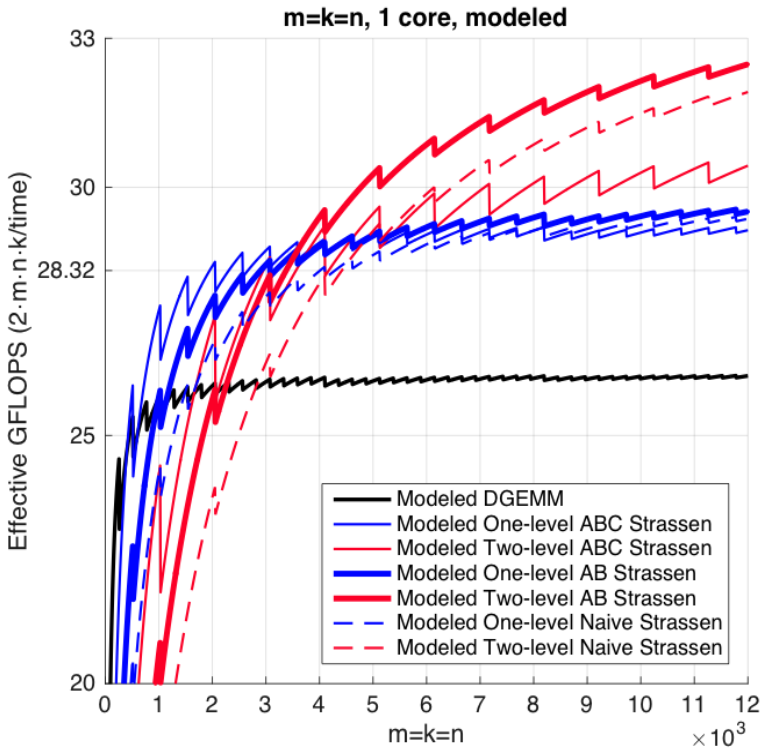


Actual Performance

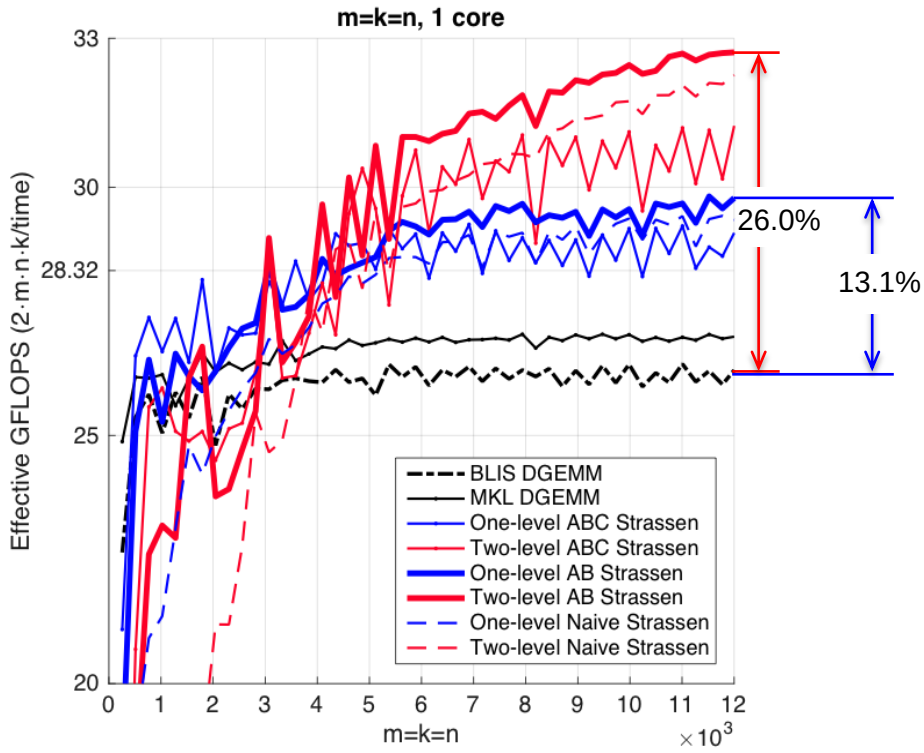


Observation (Square Matrices)

Modeled Performance



Actual Performance

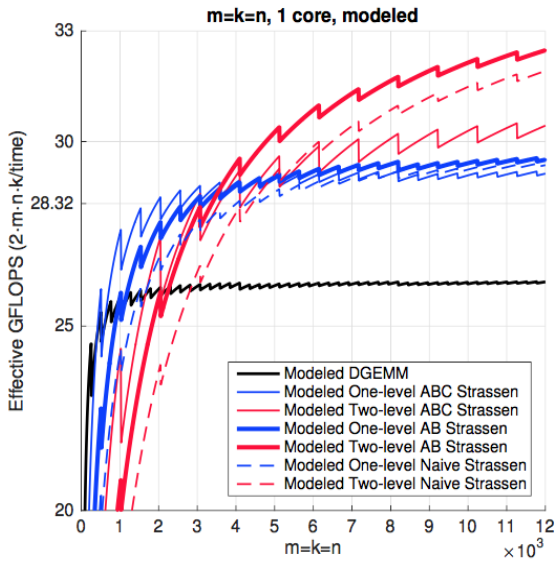


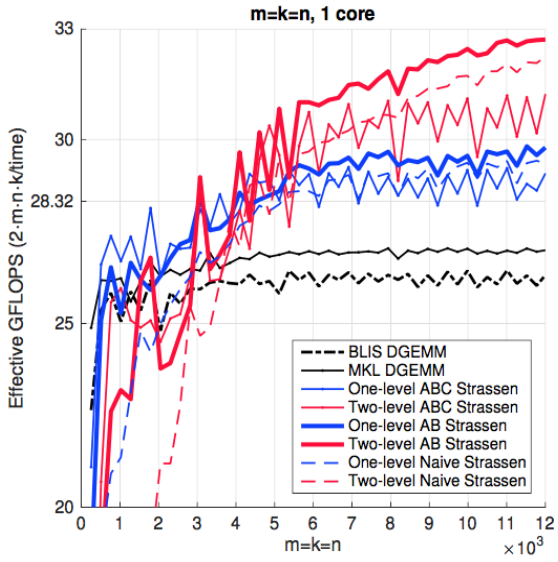
Theoretical Speedup over DGEMM

- One-level Strassen (1+14.3% speedup)
  - 8 multiplications → 7 multiplications;
- Two-level Strassen (1+30.6% speedup)
  - 64 multiplications → 49 multiplications;

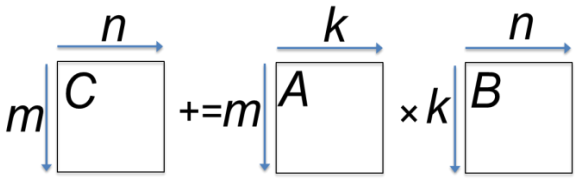
Observation (Square Matrices)

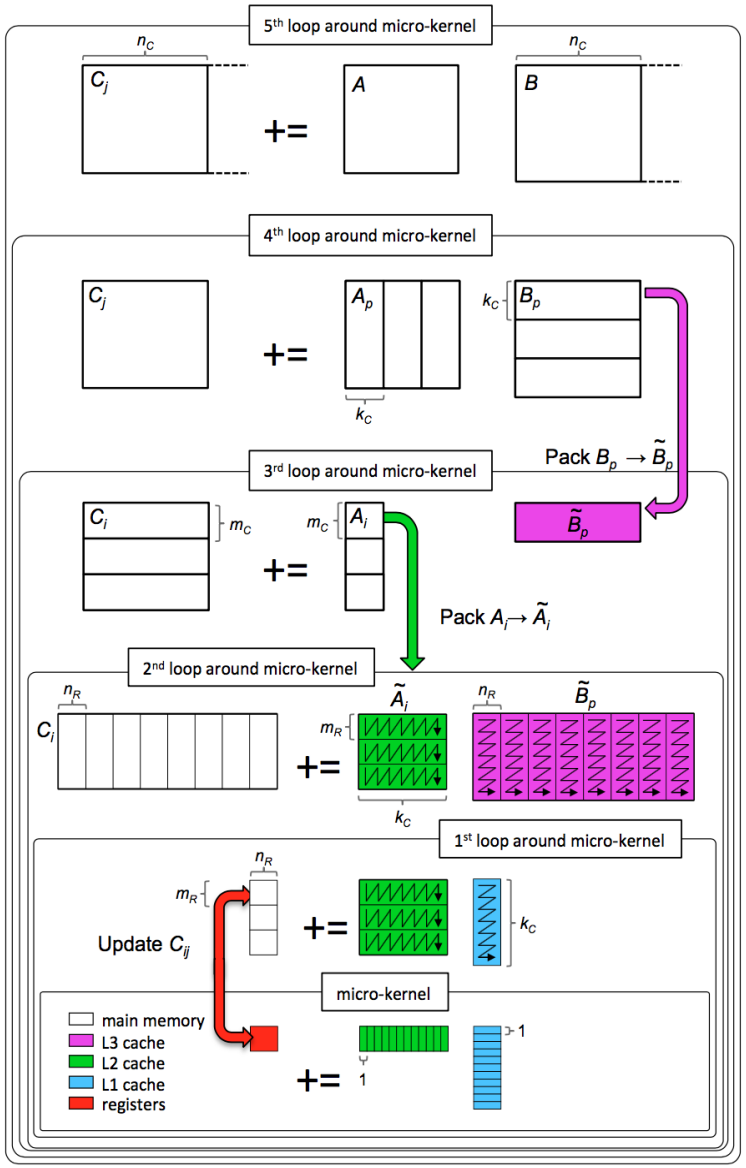
- Both one-level and two-level
  - For small square matrices, **ABC Strassen** outperforms **AB Strassen**
  - For larger square matrices, this trend reverses
- Reason
  - **ABC Strassen** avoids storing M (M resides in the register)
  - **ABC Strassen** increases the number of times for updating submatrices of *C*



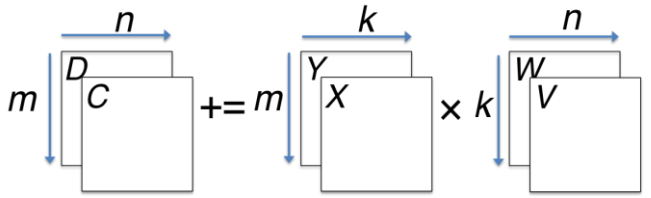


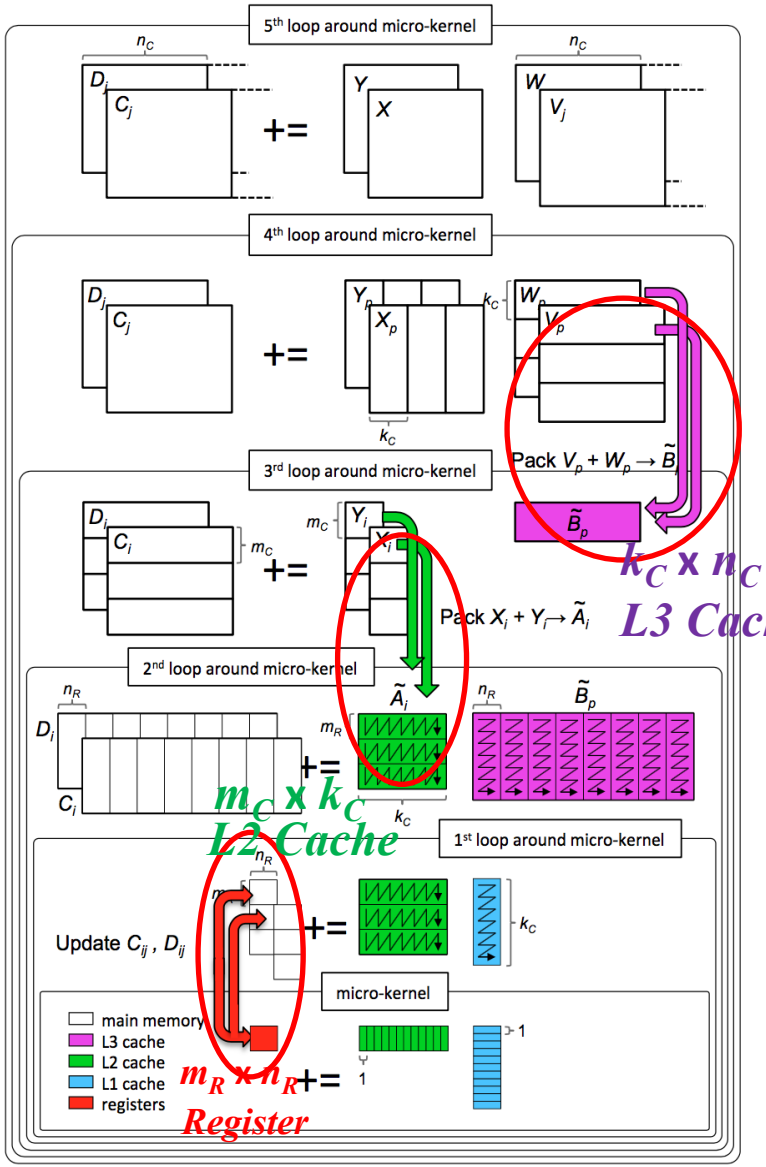
$C \mathrel{+}= AB;$





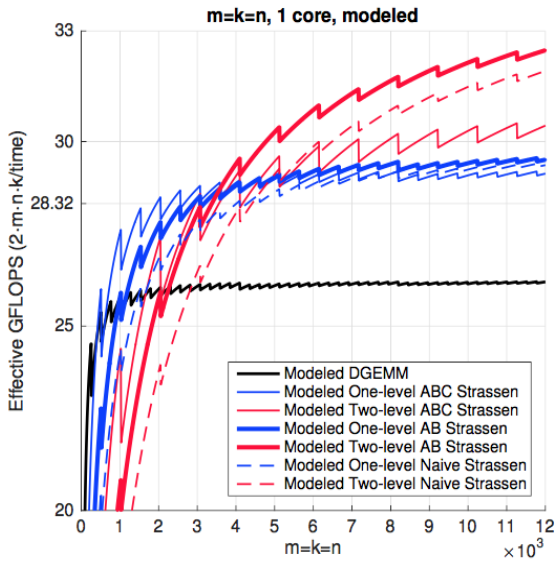
$M := (X+Y)(V+W);\ C \mathrel{+}= M;\ D \mathrel{+}= M;$

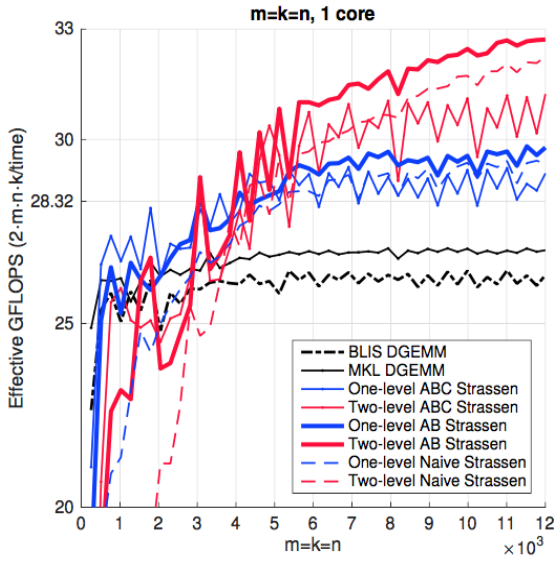




Observation (Square Matrices)

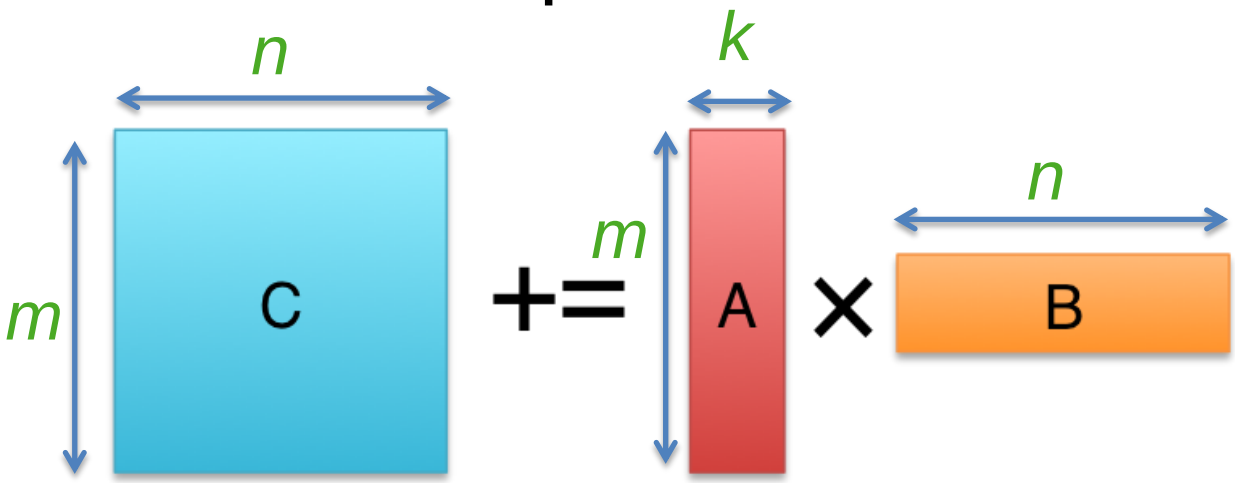
- Both one-level and two-level
  - For small square matrices, **ABC Strassen** outperforms **AB Strassen**
  - For larger square matrices, this trend reverses
- Reason
  - **ABC Strassen** avoids storing M (M resides in the register)
  - **ABC Strassen** increases the number of times for updating submatrices of *C*





Observation (Rank-k Update)

- What is Rank-k update?



Observation (Rank-k Update)

- Importance of Rank-k update

Numer. Math. 13, 354–356 (1969)

Gaussian Elimination is not Optimal

VOLKER STRASSEN*

Received December 12, 1968

1. Below we will give an algorithm which computes the coefficients of the product of two square matrices A and B of order n from the coefficients of A and B with less than $4.7 \cdot n^{\log 7}$ arithmetical operations (all logarithms in this paper are for base 2, thus $\log 7 \approx 2.8$; the usual method requires approximately $2n^3$ arithmetical operations). The algorithm induces algorithms for inverting a matrix of order n, solving a system of n linear equations in n unknowns, computing a determinant of order n etc. all requiring less than const $n^{\log 7}$ arithmetical operations.

This fact should be compared with the result of KLYUYEV and KOKOVKIN-SHCHERBAK [1] that Gaussian elimination for solving a system of linear equations is optimal if one restricts oneself to operations upon rows and columns as a whole. We also note that WINOGRAD [2] modifies the usual algorithms for matrix multiplication and inversion and for solving systems of linear equations, trading roughly half of the multiplications for additions and subtractions.

It is a pleasure to thank D. BRILLINGER for inspiring discussions about the present subject and ST. COOK and B. PARLETT for encouraging me to write this paper.

2. We define algorithms $\alpha_{m,k}$ which multiply matrices of order $m2^k$, by induction on k: $\alpha_{m,0}$ is the usual algorithm for matrix multiplication (requiring m^3 multiplications and $m^2(m-1)$ additions). $\alpha_{m,k}$ already being known, define $\alpha_{m,k+1}$ as follows:

If A, B are matrices of order $m2^{k+1}$ to be multiplied, write

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad AB = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix},$$

where the A_{ik}, B_{ik}, C_{ik} are matrices of order $m2^k$. Then compute

$$\begin{aligned} \text{I} &= (A_{11}+A_{22})(B_{11}+B_{22}), \\ \text{II} &= (A_{21}+A_{22})B_{11}, \\ \text{III} &= A_{11}(B_{12}-B_{22}), \\ \text{IV} &= A_{22}(-B_{11}+B_{21}), \\ \text{V} &= (A_{11}+A_{12})B_{22}, \\ \text{VI} &= (-A_{11}+A_{21})(B_{11}+B_{12}), \\ \text{VII} &= (A_{12}-A_{22})(B_{21}+B_{22}), \end{aligned}$$

* The results have been found while the author was at the Department of Statistics of the University of California, Berkeley. The author wishes to thank the National Science Foundation for their support (NSF GP-7454).

Blocked LU with partial pivoting (**getrf**)

Algorithm: $[A, p] := \text{LUPIV\_BLK}(A)$

Partition

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

where A_{TL} is 0×0, p_T has 0 elements

while $n(A_{TL}) < n(A)$ **do**

Determine block size b

Repartition

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

$$\left(\begin{array}{c} p_T \\ \hline p_B \end{array}\right) \rightarrow \left(\begin{array}{c} p_0 \\ \hline p_1 \\ \hline p_2 \end{array}\right)$$

where A_{11} **is** $b \times b$, p_1 **is** $b \times 1$

$$\left[\left(\begin{array}{c} A_{11} \\ \hline A_{21} \end{array}\right), p_1\right] := \text{LUPIV\_UNB}\left(\begin{array}{c} A_{11} \\ \hline A_{21} \end{array}\right)$$

$$\left(\begin{array}{c|c} A_{10} & A_{12} \\ \hline A_{20} & A_{22} \end{array}\right) := \text{PIV}\left(p_1, \left(\begin{array}{c|c} A_{10} & A_{12} \\ \hline A_{20} & A_{22} \end{array}\right)\right)$$

$$A_{12} := L_{11}^{-1} A_{12}$$

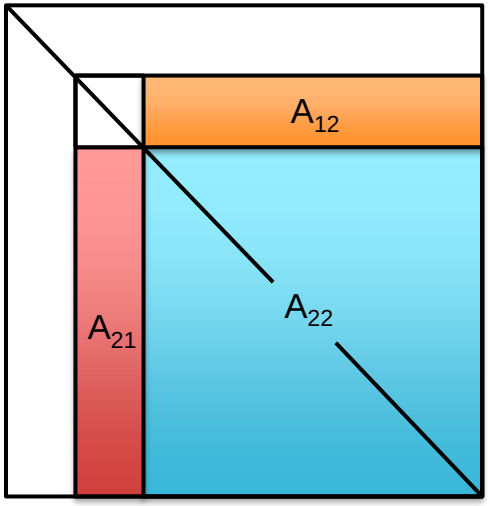
$$A_{22} := A_{22} - A_{21}A_{12}$$

Continue with

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

$$\left(\begin{array}{c} p_T \\ \hline p_B \end{array}\right) \leftarrow \left(\begin{array}{c} p_0 \\ \hline p_1 \\ \hline p_2 \end{array}\right)$$

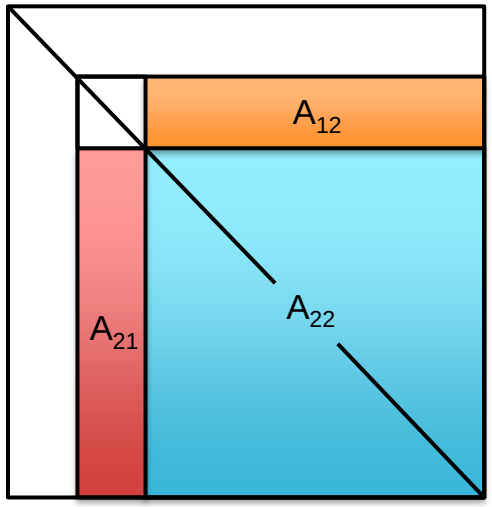
endwhile



Observation (Rank-k Update)

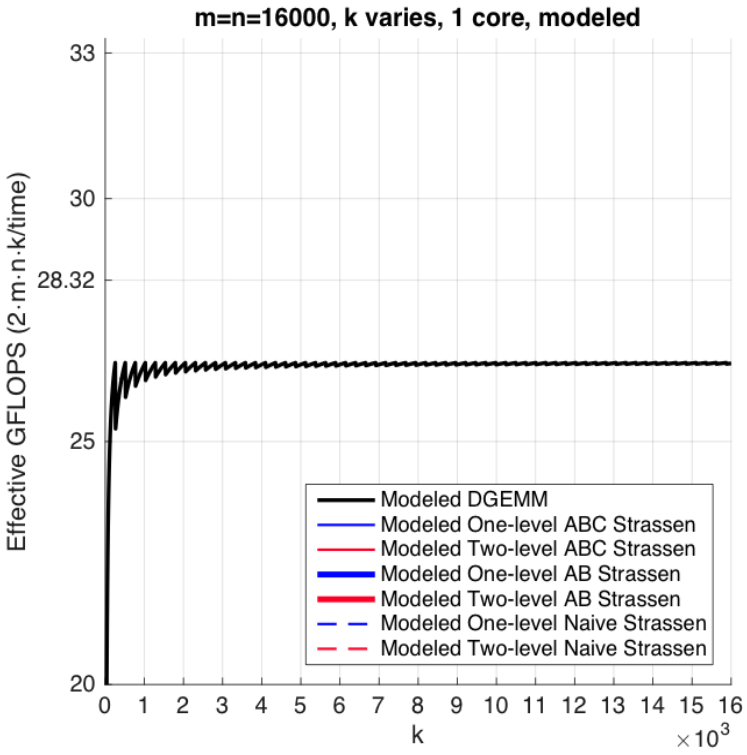
- Importance of Rank-k update

+= ×

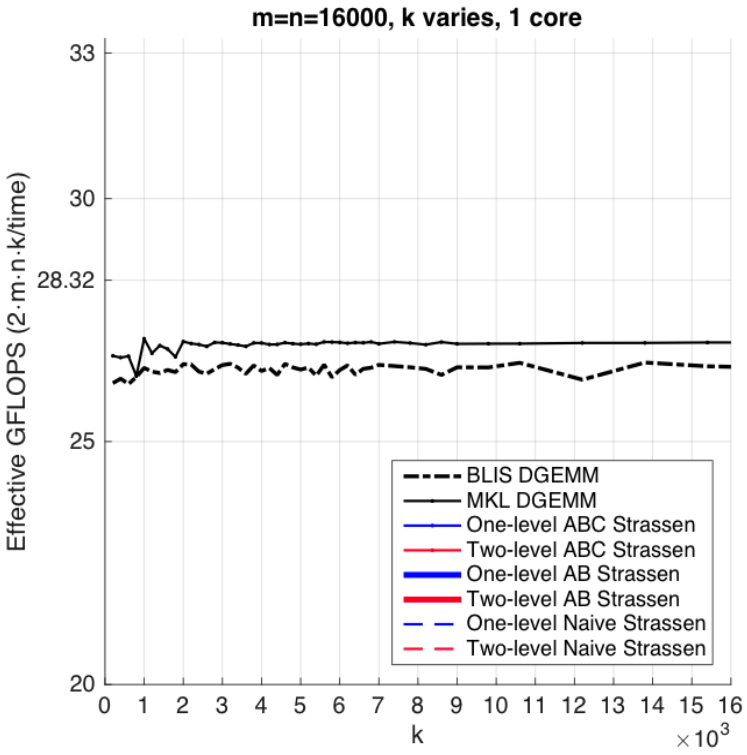


Observation (Rank-k Update)

Modeled Performance

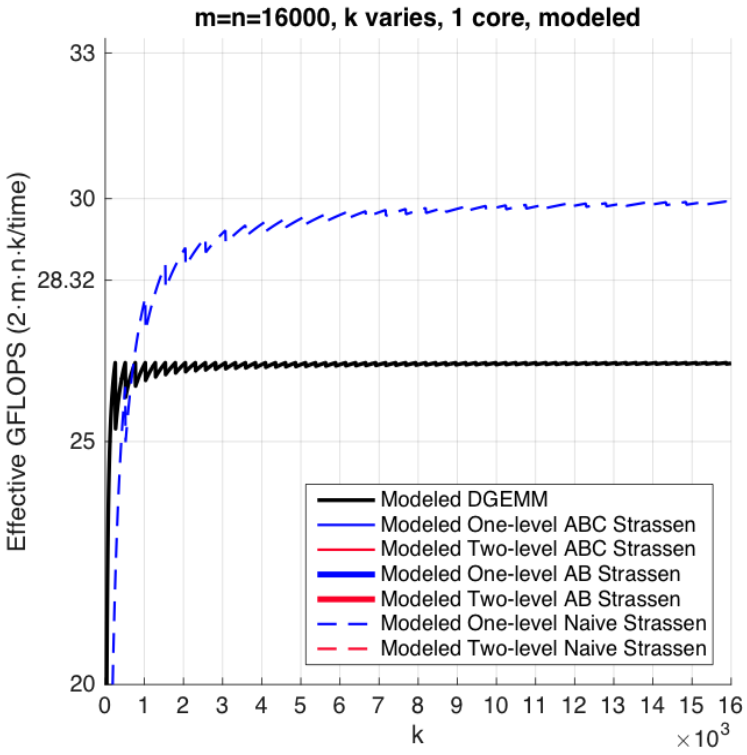


Actual Performance

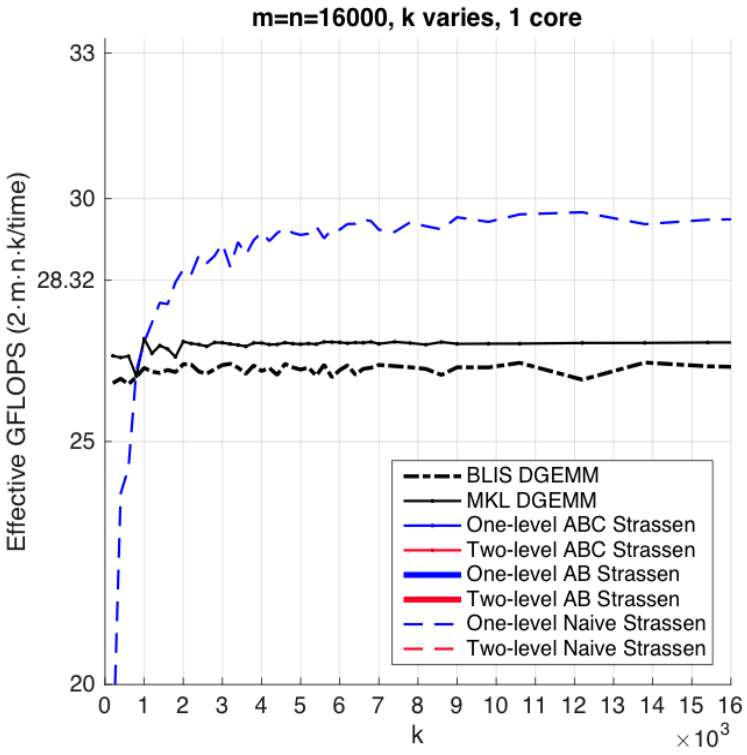


Observation (Rank-k Update)

Modeled Performance

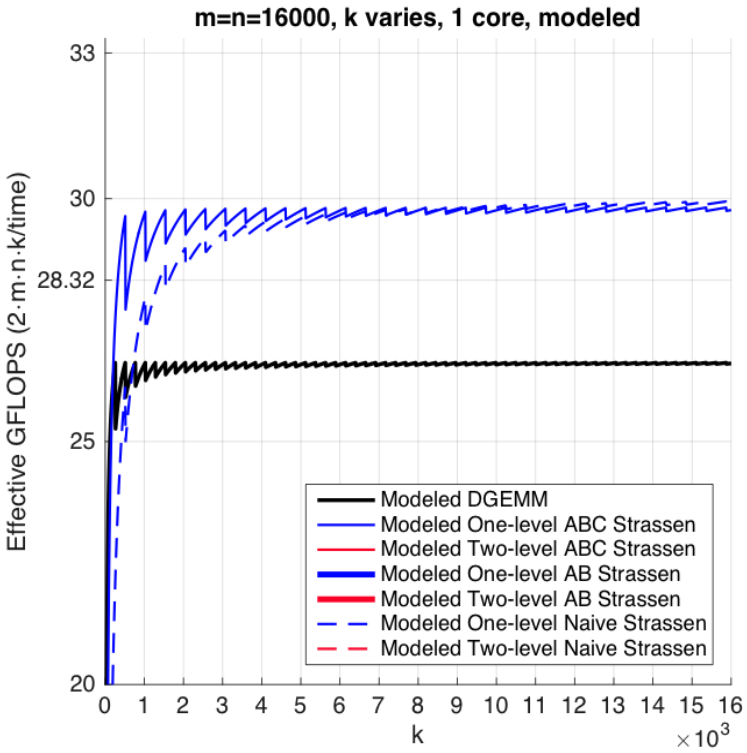


Actual Performance

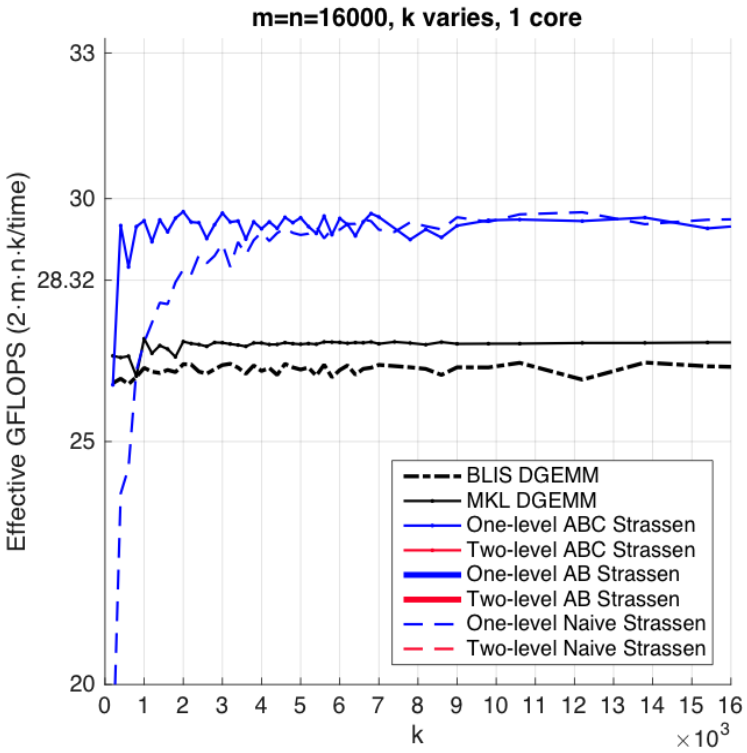


Observation (Rank-k Update)

Modeled Performance

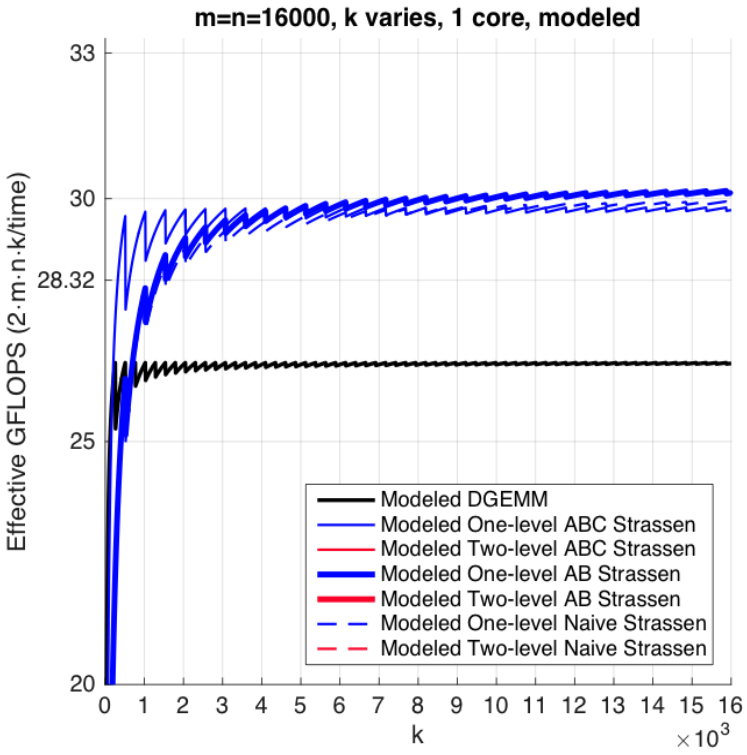


Actual Performance

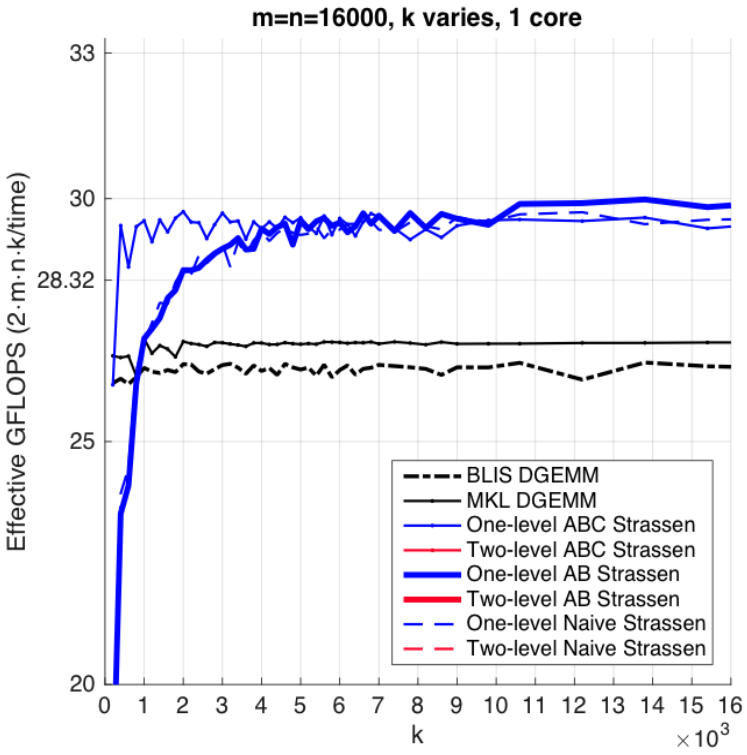


Observation (Rank-k Update)

Modeled Performance

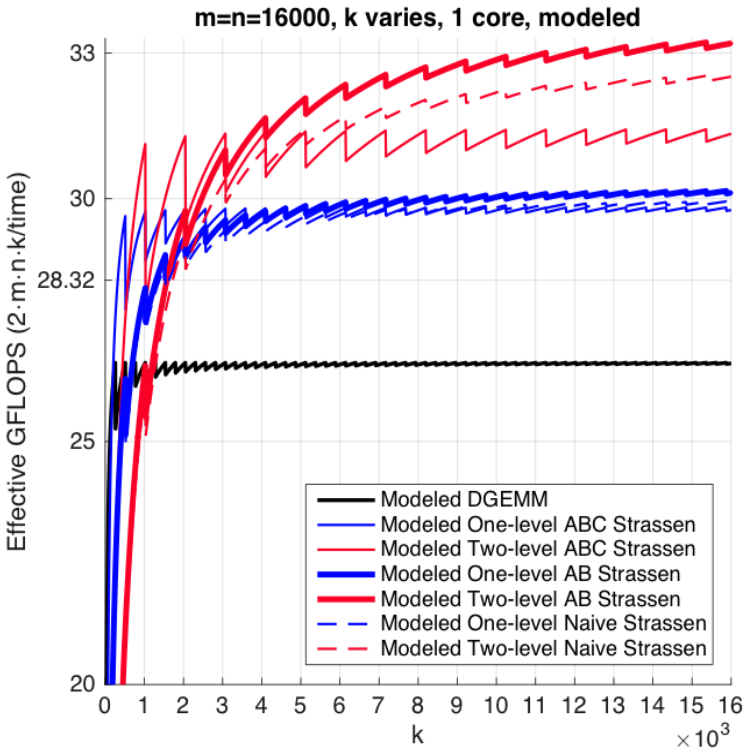


Actual Performance

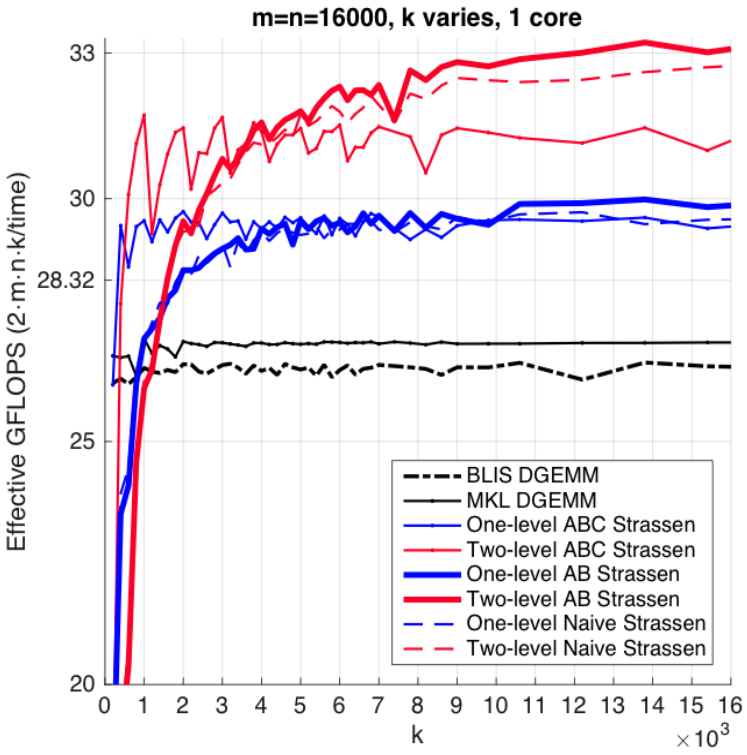


Observation (Rank-k Update)

Modeled Performance

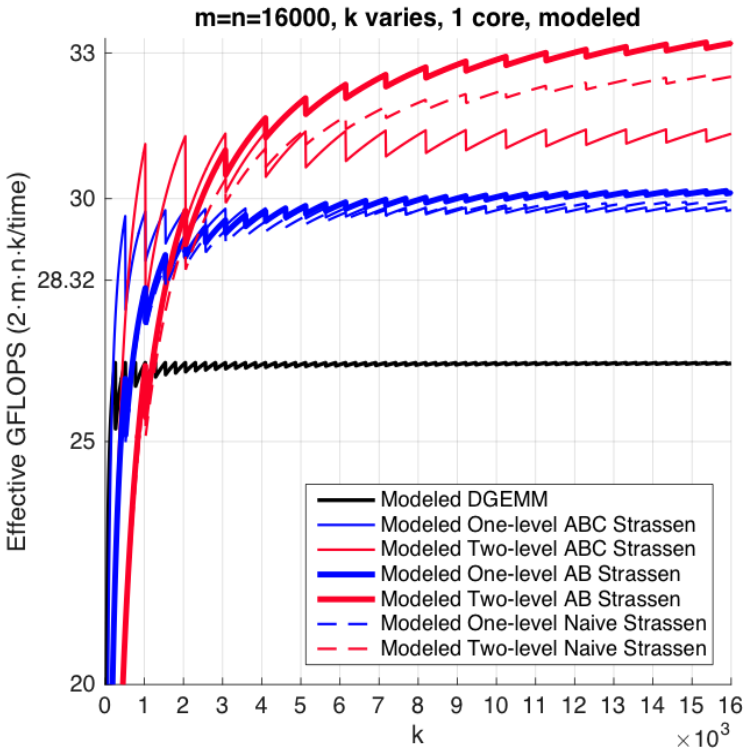


Actual Performance

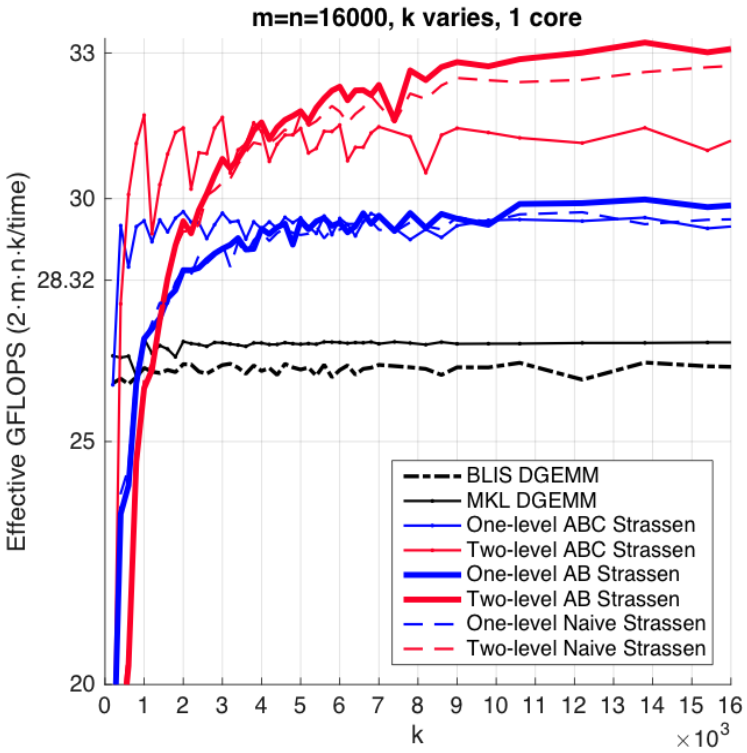


Observation (Rank-k Update)

Modeled Performance



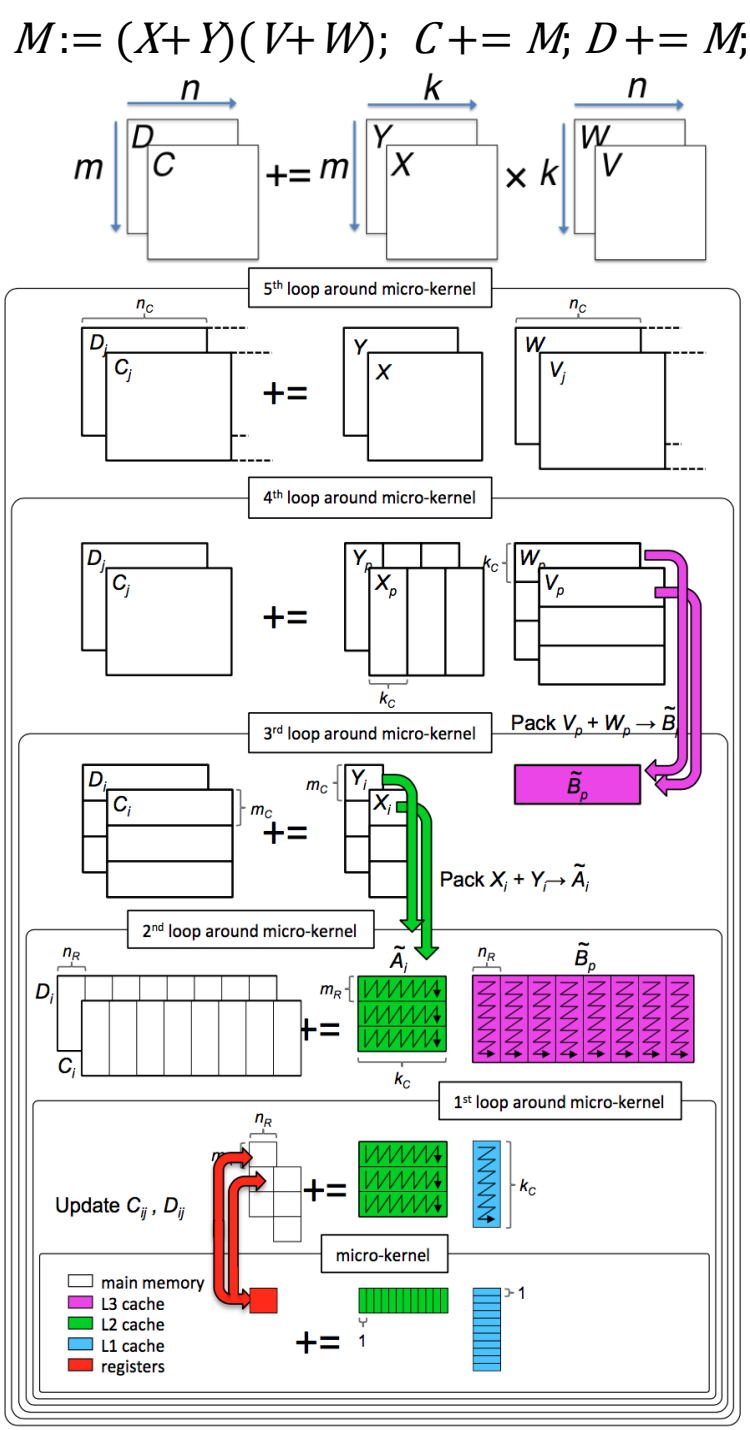
Actual Performance



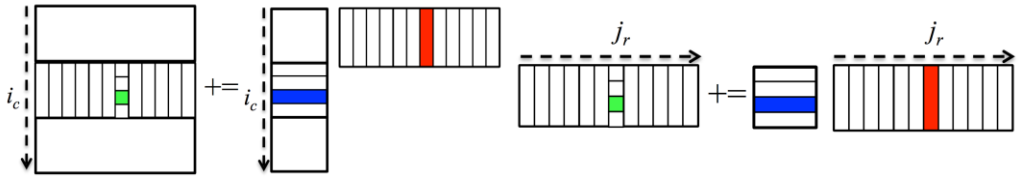
- Reason:

  ABC Strassen avoids forming the temporary matrix M explicitly in the memory (M resides in register), especially important when m, n >> k.

Parallelization



- Use the same parallel scheme as BLIS: Parallelize 3rd or 2nd loop



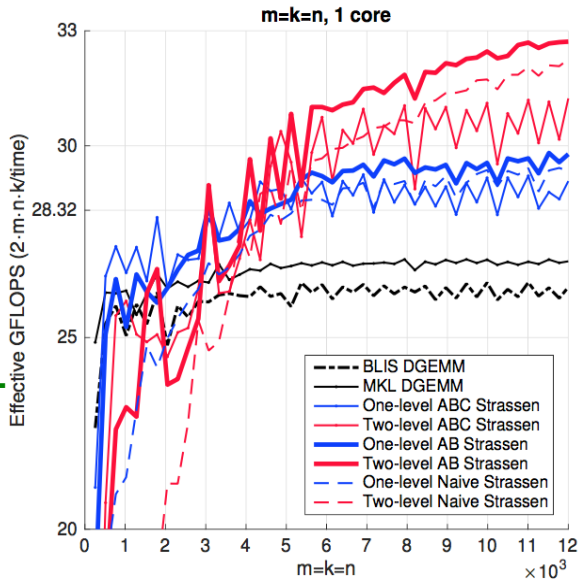
- Comparing with Task Parallelism:
  - Pro: data parallelism; load balance
  - Con: synchronization between instructions

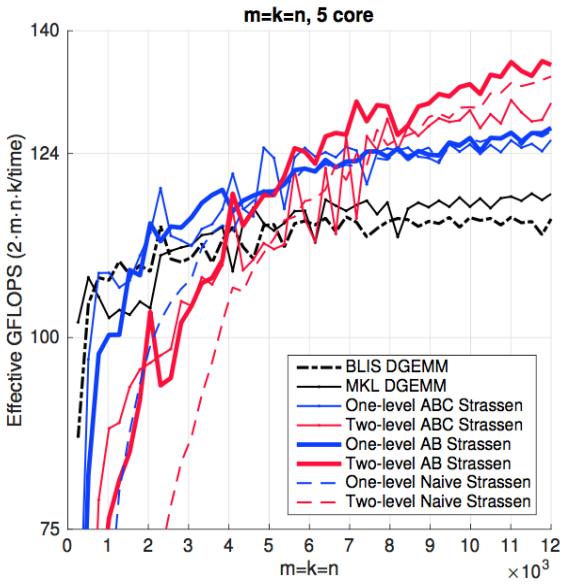
*Tyler M. Smith, Robert Van De Geijn, Mikhail Smelyanskiy, Jeff R. Hammond, and Field G. Van Zee. "Anatomy of high-performance many-threaded matrix multiplication." In *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*, pp. 1049-1059. IEEE, 2014.

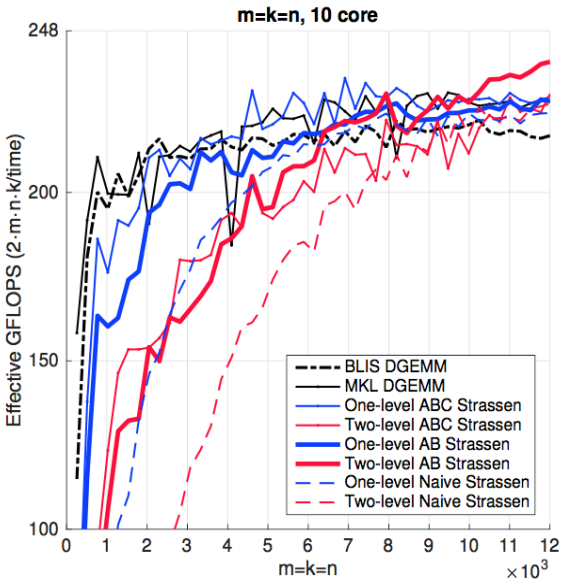
Intel Xeon E5-2680 v2 (Ivy Bridge, 10 core/socket)

1 core | **5 core** | **10 core**

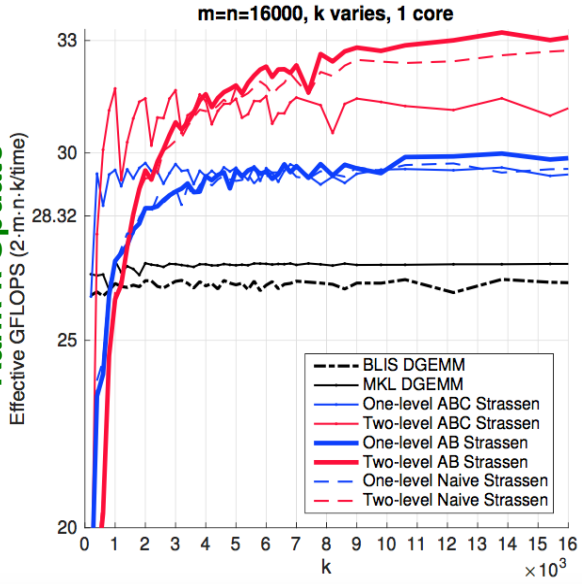
Square Matrices

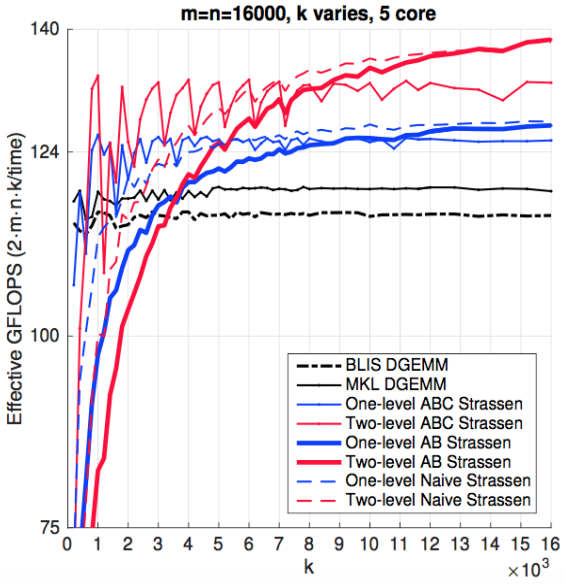


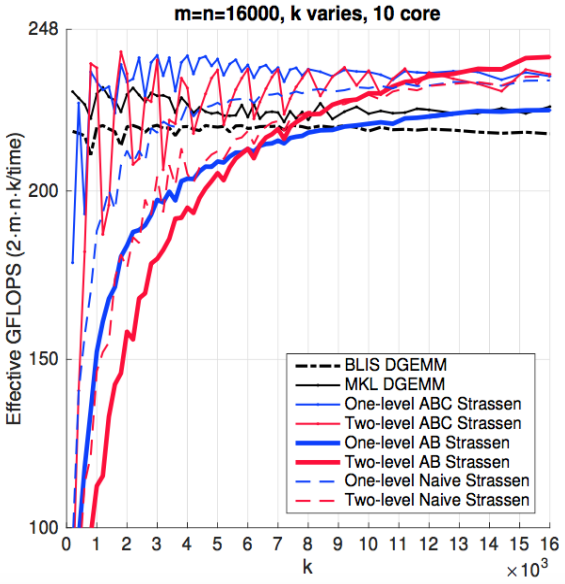




Rank-k Update







Outline

- Motivation
- Background
- Practical Strassen’s Algorithm (SC16)
- Practical Fast Matrix Multiplication (IPDPS17)
- Proposed Work
- Conclusion

Practical Fast Matrix Multiplication

Jianyu Huang**, Leslie Rice, Devin A. Matthews, Robert A. van de Geijn. “Generating Families of Practical Fast Matrix Multiplication Algorithms.” In ***IPDPS17

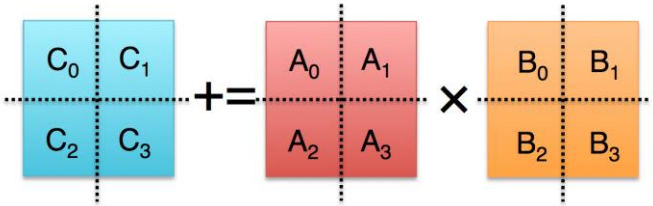
<2,2,2> Strassen's Algorithm

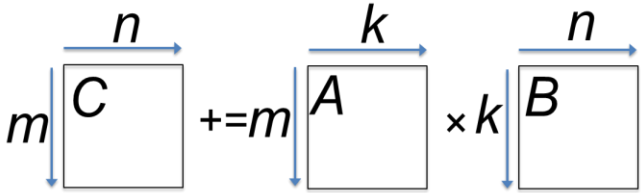
$M_0 := (A_0+A_3)(B_0+B_3)$; $C_0 \mathrel{+}= M_0$; $C_3 \mathrel{+}= M_0$;
$M_1 := (A_2+A_3)B_0$; $C_2 \mathrel{+}= M_1$; $C_3 \mathrel{-}= M_1$;
$M_2 := A_0(B_1-B_3)$; $C_1 \mathrel{+}= M_2$; $C_3 \mathrel{+}= M_2$;
$M_3 := A_3(B_2-B_0)$; $C_0 \mathrel{+}= M_3$; $C_2 \mathrel{+}= M_3$;
$M_4 := (A_0+A_1)B_3$; $C_1 \mathrel{+}= M_4$; $C_0 \mathrel{-}= M_4$;
$M_5 := (A_2-A_0)(B_0+B_1)$; $C_3 \mathrel{+}= M_5$;
$M_6 := (A_1-A_3)(B_2+B_3)$; $C_0 \mathrel{+}= M_6$;

$$\mathbf{U} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{bmatrix}$$

$$\mathbf{V} = \begin{bmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$





Dario Bini, and Grazia Lotti. "Stability of fast algorithms for matrix multiplication." *Numerische Mathematik* 36.1 (1980): 63-72.

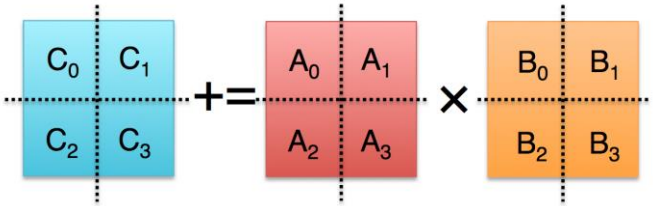
$\langle \tilde{m}, \tilde{k}, \tilde{n} \rangle$, Fast Strassen's Algorithm (FFMM)

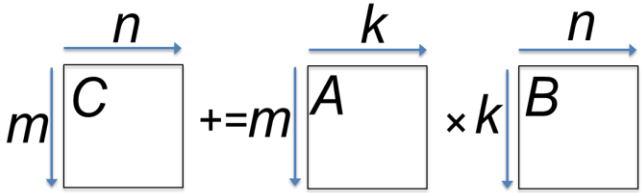
$$
\begin{aligned}
M_0 &:= (A_0 + A_3)(B_0 + B_3); & C_0 &+= M_0;\ C_3 += M_0;\\
M_1 &:= (A_2 + A_3)B_0; & C_2 &+= M_1;\ C_3 -= M_1;\\
M_2 &:= A_0(B_1 - B_3); & C_1 &+= M_2;\ C_3 += M_2;\\
M_3 &:= A_3(B_2 - B_0); & C_0 &+= M_3;\ C_2 += M_3;\\
M_4 &:= (A_0 + A_1)B_3; & C_1 &+= M_4;\ C_0 -= M_4;\\
M_5 &:= (A_2 - A_0)(B_0 + B_1); & C_3 &+= M_5;\\
M_6 &:= (A_1 - A_3)(B_2 + B_3); & C_0 &+= M_6;
\end{aligned}
$$

$$
\mathbf{U} = \begin{bmatrix}
1 & 0 & 1 & 0 & 1 & -1 & 0\\
0 & 0 & 0 & 0 & 1 & 0 & 1\\
0 & 1 & 0 & 0 & 0 & 1 & 0\\
1 & 1 & 0 & 1 & 0 & 0 & -1
\end{bmatrix}
$$

$$
\mathbf{V} = \begin{bmatrix}
1 & 1 & 0 & -1 & 0 & 1 & 0\\
0 & 0 & 1 & 0 & 0 & 1 & 0\\
0 & 0 & 0 & 1 & 0 & 0 & 1\\
1 & 0 & -1 & 0 & 1 & 0 & 1
\end{bmatrix}
$$

$$
\mathbf{W} = \begin{bmatrix}
1 & 0 & 0 & 1 & -1 & 0 & 1\\
0 & 0 & 1 & 0 & 1 & 0 & 0\\
0 & 1 & 0 & 1 & 0 & 0 & 0\\
1 & -1 & 1 & 0 & 0 & 1 & 0
\end{bmatrix}
$$





$\langle \widetilde{m}, \widetilde{k}, \widetilde{n} \rangle$ Fast Matrix Multiplication (FMM)

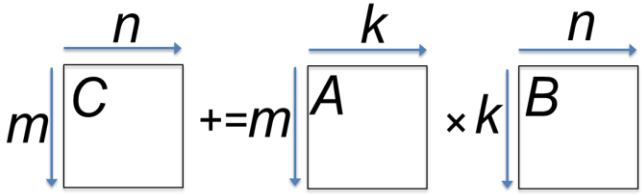
$$
\begin{array}{ll}
M_0 := (A_0+A_3)(B_0+B_3); & C_0 \mathrel{+}= M_0;\ C_3 \mathrel{+}= M_0; \\
M_1 := (A_2+A_3)B_0; & C_2 \mathrel{+}= M_1;\ C_3 \mathrel{-}= M_1; \\
M_2 := A_0(B_1-B_3); & C_1 \mathrel{+}= M_2;\ C_3 \mathrel{+}= M_2; \\
M_3 := A_3(B_2-B_0); & C_0 \mathrel{+}= M_3;\ C_2 \mathrel{+}= M_3; \\
M_4 := (A_0+A_1)B_3; & C_1 \mathrel{+}= M_4;\ C_0 \mathrel{-}= M_4; \\
M_5 := (A_2-A_0)(B_0+B_1); & C_3 \mathrel{+}= M_5; \\
M_6 := (A_1-A_3)(B_2+B_3); & C_0 \mathrel{+}= M_6;
\end{array}
$$

$$
\mathbf{U} = \begin{bmatrix}
1 & 0 & 1 & 0 & 1 & -1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & -1
\end{bmatrix}
$$

$$
\mathbf{V} = \begin{bmatrix}
1 & 1 & 0 & -1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 0 & -1 & 0 & 1 & 0 & 1
\end{bmatrix}
$$

$$
\mathbf{W} = \begin{bmatrix}
1 & 0 & 0 & 1 & -1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 1 & 0 & 0 & 0 \\
1 & -1 & 1 & 0 & 0 & 1 & 0
\end{bmatrix}
$$

$$
C = \begin{pmatrix} C_0 & \cdots & C_{\widetilde{n-1}} \\ \vdots & & \vdots \\ C_{\widetilde{(m-1)n}} & \cdots & C_{\widetilde{mn-1}} \end{pmatrix},\ A = \begin{pmatrix} A_0 & \cdots & A_{\widetilde{k-1}} \\ \vdots & & \vdots \\ A_{\widetilde{(m-1)k}} & \cdots & A_{\widetilde{mk-1}} \end{pmatrix},\ B = \begin{pmatrix} B_0 & \cdots & B_{\widetilde{n-1}} \\ \vdots & & \vdots \\ B_{\widetilde{(k-1)n}} & \cdots & B_{\widetilde{kn-1}} \end{pmatrix}
$$



$\langle \tilde{m}, \tilde{k}, \tilde{n} \rangle$ Fast Matrix Multiplication (FMM)

for $r = 0 \ldots R$

$M_0 := (A_0 + A_3)(B_0 + B_3)$; $C_0 \mathrel{+}= M_0$; $C_3 \mathrel{+}= M_0$;
$M_1 := (A_2 + A_3)B_0$; $C_2 \mathrel{+}= M_1$; $C_3 \mathrel{-}= M_1$;
$M_2 := A_0(B_1 - B_3)$; $C_1 \mathrel{+}= M_2$; $C_3 \mathrel{+}= M_2$;
$M_3 := A_3(B_2 - B_0)$; $C_0 \mathrel{+}= M_3$; $C_2 \mathrel{+}= M_3$;
$M_4 := (A_0 + A_1)B_3$; $C_1 \mathrel{+}= M_4$; $C_0 \mathrel{-}= M_4$;
$M_5 := (A_2 - A_0)(B_0 + B_1)$; $C_3 \mathrel{+}= M_5$;
$M_6 := (A_1 - A_3)(B_2 + B_3)$; $C_0 \mathrel{+}= M_6$;

$$M_r := \left(\sum_{i=0}^{\tilde{m}\tilde{k}-1} u_{ir} A_i \right) \times \left(\sum_{j=0}^{\tilde{k}\tilde{n}-1} v_{jr} B_j \right);$$

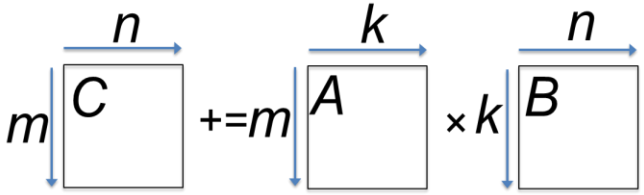
$$C_p \mathrel{+}= w_{pr} M_r \quad (p = 0 \ldots \tilde{m}\tilde{n} - 1)$$

$$\mathbf{U} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{bmatrix}$$

$$\mathbf{V} = \begin{bmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$C = \begin{pmatrix} C_0 & \cdots & C_{\tilde{n}-1} \\ \vdots & & \vdots \\ C_{(\tilde{m}-1)\tilde{n}} & \cdots & C_{\tilde{m}\tilde{n}-1} \end{pmatrix}, A = \begin{pmatrix} A_0 & \cdots & A_{\tilde{k}-1} \\ \vdots & & \vdots \\ A_{(\tilde{m}-1)\tilde{k}} & \cdots & A_{\tilde{m}\tilde{k}-1} \end{pmatrix}, B = \begin{pmatrix} B_0 & \cdots & B_{\tilde{n}-1} \\ \vdots & & \vdots \\ B_{(\tilde{k}-1)\tilde{n}} & \cdots & B_{\tilde{k}\tilde{n}-1} \end{pmatrix}$$



$\langle \widetilde{m}, \widetilde{k}, \widetilde{n} \rangle$ Fast Matrix Multiplication (FMM)

for $r = 0, \ldots, R-1$,

$$M_r := \left(\sum_{i=0}^{\widetilde{mk}-1} u_{ir} A_i \right) \times \left(\sum_{j=0}^{\widetilde{kn}-1} v_{jr} B_j \right);$$

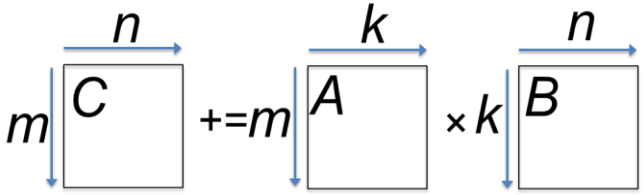
$$C_p \mathrel{+}= w_{pr} M_r \quad (p = 0, \ldots, \widetilde{mn}-1)$$

$$\mathbf{U} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{bmatrix}$$

$$\mathbf{V} = \begin{bmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$C = \begin{pmatrix} C_0 & \cdots & C_{\widetilde{n}-1} \\ \vdots & & \vdots \\ C_{\widetilde{(m-1)}\widetilde{n}} & \cdots & C_{\widetilde{mn}-1} \end{pmatrix}, A = \begin{pmatrix} A_0 & \cdots & A_{\widetilde{k}-1} \\ \vdots & & \vdots \\ A_{\widetilde{(m-1)}\widetilde{k}} & \cdots & A_{\widetilde{mk}-1} \end{pmatrix}, B = \begin{pmatrix} B_0 & \cdots & B_{\widetilde{n}-1} \\ \vdots & & \vdots \\ B_{\widetilde{(k-1)}\widetilde{n}} & \cdots & B_{\widetilde{kn}-1} \end{pmatrix}$$



The set of coefficients that determine the $\langle \widetilde{m}, \widetilde{k}, \widetilde{n} \rangle$ algorithm is denoted as $[\![U, V, W]\!]$.

$\langle \tilde{3}, \tilde{2}, \tilde{3} \rangle$ Fast Matrix Multiplication (FMM)

for $r = 0, \ldots, R-1$,

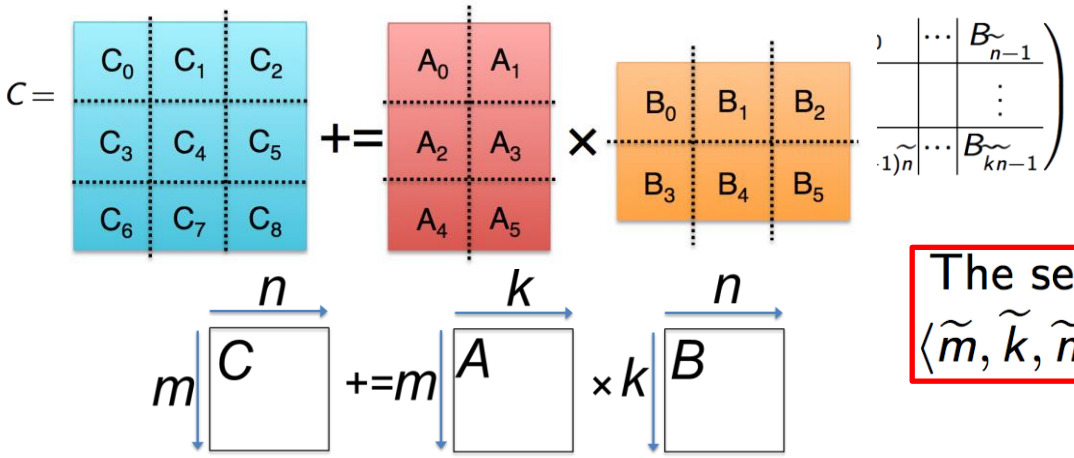
$$M_r := \left(\sum_{i=0}^{\tilde{m}\tilde{k}-1} u_{ir} A_i \right) \times \left(\sum_{j=0}^{\tilde{k}\tilde{n}-1} v_{jr} B_j \right);$$

$$C_p \mathrel{+}= w_{pr} M_r \quad (p = 0, \ldots, \tilde{m}\tilde{n}-1)$$

$$\mathbf{U} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{bmatrix}$$

$$\mathbf{V} = \begin{bmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$



The set of coefficients that determine the $\langle \tilde{m}, \tilde{k}, \tilde{n} \rangle$ algorithm is denoted as $[\![U, V, W]\!]$.

<3,2,3> Fast Matrix Multiplication (FMM)

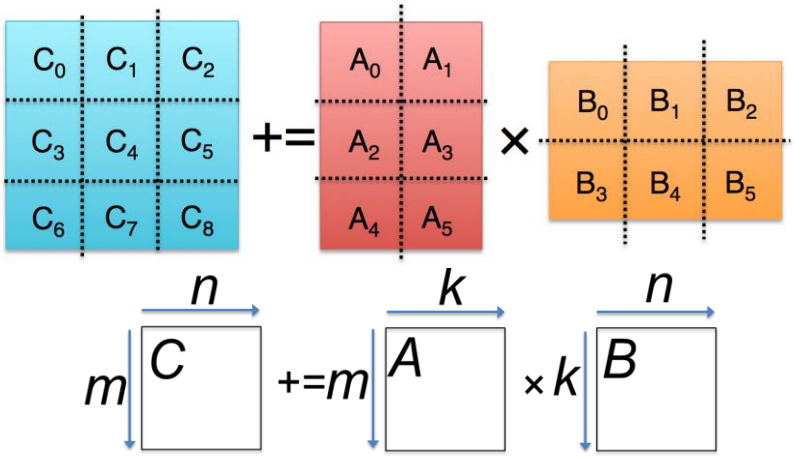
$M_0 := (-A_1+A_3+A_5)(B_4+B_5);$ $C_2 -= M_0;$
$M_1 := (-A_0+A_2+A_5)(B_0-B_5);$ $C_2 -= M_1;\ C_7 -= M_1;$

for $r = 0, \ldots, R-1,$

$$M_r := \left(\sum_{i=0}^{\widetilde{mk}-1} u_{ir} A_i\right) \times \left(\sum_{j=0}^{\widetilde{kn}-1} v_{jr} B_j\right);$$

$$C_p += w_{pr} M_r \ (p = 0, \ldots, \widetilde{mn}-1)$$

$M_{13} := (A_2-A_4)(B_0-B_2+B_3-B_5);$ $C_6 -= M_{13};$
$M_{14} := (-A_0+A_1+A_2-A_3)(B_5);$ $C_2 -= M_{14}; C_4 -= M_{14};$



$$\mathbf{U} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{bmatrix}$$

$$\mathbf{V} = \begin{bmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

The set of coefficients that determine the $\langle \widetilde{m}, \widetilde{k}, \widetilde{n} \rangle$ algorithm is denoted as $[\![U, V, W]\!]$.

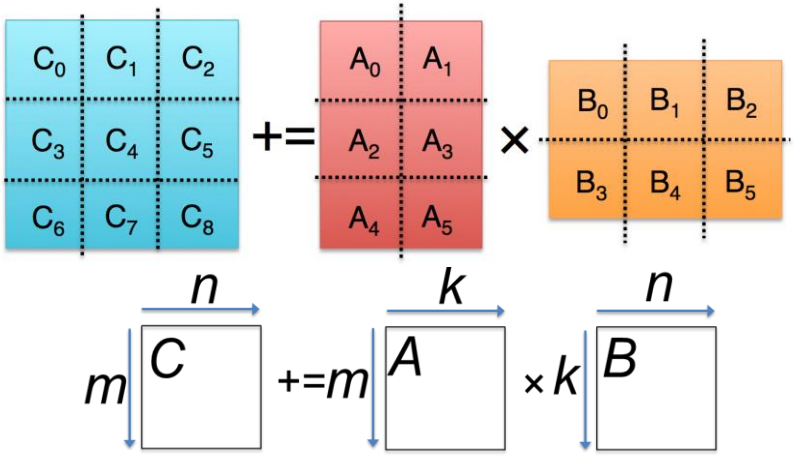
<3,2,3> Fast Matrix Multiplication (FMM)

$M_0 := (-A_1+A_3+A_5)(B_4+B_5)$; $C_2 -= M_0$;
$M_1 := (-A_0+A_2+A_5)(B_0-B_5)$; $C_2 -= M_1$; $C_7 -= M_1$;
$M_2 := (-A_0+A_2)(-B_1-B_4)$; $C_0-=M_2; C_1+=M_2; C_7+=M_2$;
$M_3 := (A_2-A_4+A_5)(B_2-B_3+B_5)$; $C_2+=M_3; C_5+=M_3; C_6-=M_3$;
$M_4 := (-A_0+A_1+A_2)(B_0+B_1+B_4)$; $C_0-=M_4; C_1+=M_4; C_4+=M_4$;
$M_5 := (A_2)(B_0)$; $C_0+=M_5; C_1-=M_5; C_3+=M_5; C_4-=M_5; C_6+=M$
$M_6 := (-A_2+A_3+A_4-A_5)(B_3-B_5)$; $C_2-=M_6; C_5-=M_6$;
$M_7 := (A_3)(B_3)$; $C_2+=M_7; C_3+=M_7; C_5+=M_7$;
$M_8 := (-A_0+A_2+A_4)(B_1+B_2)$; $C_7+=M_8$;
$M_9 := (-A_0+A_1)(-B_0-B_1)$; $C_1+=M_9; C_4+=M_9$;
$M_{10} := (A_5)(B_2+B_5)$; $C_2+=M_{10}; C_6+=M_{10}; C_8+=M_{10}$;
$M_{11} := (A_4-A_5)(B_2)$; $C_2+=M_{11}; C_5+=M_{11}; C_7-=M_{11}; C_8+=M_{11}$;
$M_{12} := (A_1)(B_0+B_1+B_3+B_4)$; $C_0+=M_{12}$;
$M_{13} := (A_2-A_4)(B_0-B_2+B_3-B_5)$; $C_6-=M_{13}$;
$M_{14} := (-A_0+A_1+A_2-A_3)(B_5)$; $C_2-=M_{14}; C_4-=M_{14}$;

$$\mathbf{U} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{bmatrix}$$

$$\mathbf{V} = \begin{bmatrix} 1 & 1 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & -1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$



The set of coefficients that determine the $\langle \widetilde{m}, \widetilde{k}, \widetilde{n} \rangle$ algorithm is denoted as $[\![U, V, W]\!]$.

$\langle \widetilde{m}, \widetilde{k}, \widetilde{n} \rangle$	Ref. $\widetilde{m}\widetilde{k}\widetilde{n}$ R	Speedup (%)				
		Theory	Practical #1		Practical #2	
			Ours	[1]	Ours	[1]
$\langle 2, 2, 2 \rangle$						
$\langle 2, 3, 2 \rangle$						
$\langle 2, 3, 4 \rangle$						
$\langle 2, 4, 3 \rangle$						
$\langle 2, 5, 2 \rangle$						
$\langle 3, 2, 2 \rangle$						
$\langle 3, 2, 3 \rangle$						
$\langle 3, 2, 4 \rangle$						
$\langle 3, 3, 2 \rangle$						
$\langle 3, 3, 3 \rangle$						
$\langle 3, 3, 6 \rangle$						
$\langle 3, 4, 2 \rangle$						
$\langle 3, 4, 3 \rangle$						
$\langle 3, 5, 3 \rangle$						
$\langle 3, 6, 3 \rangle$						
$\langle 4, 2, 2 \rangle$						
$\langle 4, 2, 3 \rangle$						
$\langle 4, 2, 4 \rangle$						
$\langle 4, 3, 2 \rangle$						
$\langle 4, 3, 3 \rangle$						
$\langle 4, 4, 2 \rangle$						
$\langle 5, 2, 2 \rangle$						
$\langle 6, 3, 3 \rangle$						

$\langle \widetilde{m}, \widetilde{k}, \widetilde{n} \rangle$	Ref.	$\widetilde{m}\widetilde{k}\widetilde{n}$	R	Speedup (%)				
				Theory	Practical #1		Practical #2	
					Ours	[1]	Ours	[1]
$\langle 2, 2, 2 \rangle$	[13]	8	7					
$\langle 2, 3, 2 \rangle$	[1]							
$\langle 2, 3, 4 \rangle$	[1]							
$\langle 2, 4, 3 \rangle$	[10]							
$\langle 2, 5, 2 \rangle$	[10]							
$\langle 3, 2, 2 \rangle$	[10]							
$\langle 3, 2, 3 \rangle$	[10]	18	15					
$\langle 3, 2, 4 \rangle$	[10]							
$\langle 3, 3, 2 \rangle$	[10]							
$\langle 3, 3, 3 \rangle$	[14]							
$\langle 3, 3, 6 \rangle$	[14]							
$\langle 3, 4, 2 \rangle$	[1]							
$\langle 3, 4, 3 \rangle$	[14]							
$\langle 3, 5, 3 \rangle$	[14]							
$\langle 3, 6, 3 \rangle$	[14]							
$\langle 4, 2, 2 \rangle$	[10]							
$\langle 4, 2, 3 \rangle$	[1]							
$\langle 4, 2, 4 \rangle$	[10]							
$\langle 4, 3, 2 \rangle$	[10]							
$\langle 4, 3, 3 \rangle$	[10]							
$\langle 4, 4, 2 \rangle$	[10]							
$\langle 5, 2, 2 \rangle$	[10]							
$\langle 6, 3, 3 \rangle$	[14]							

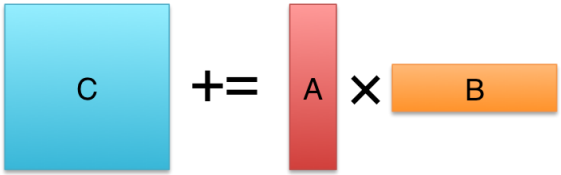
Speedup (%): FMM vs. GEMM

$\langle \widetilde{m}, \widetilde{k}, \widetilde{n} \rangle$	Ref.	$\widetilde{m}\widetilde{k}\widetilde{n}$	R	Speedup (%)				
				Theory	Practical #1		Practical #2	
					Ours	[1]	Ours	[1]
$\langle 2,2,2 \rangle$	[13]	8	7	14.3				
$\langle 2,3,2 \rangle$	[1]							
$\langle 2,3,4 \rangle$	[1]							
$\langle 2,4,3 \rangle$	[10]							
$\langle 2,5,2 \rangle$	[10]							
$\langle 3,2,2 \rangle$	[10]							
$\langle 3,2,3 \rangle$	[10]	18	15	20.0				
$\langle 3,2,4 \rangle$	[10]							
$\langle 3,3,2 \rangle$	[10]							
$\langle 3,3,3 \rangle$	[14]							
$\langle 3,3,6 \rangle$	[14]							
$\langle 3,4,2 \rangle$	[1]							
$\langle 3,4,3 \rangle$	[14]							
$\langle 3,5,3 \rangle$	[14]							
$\langle 3,6,3 \rangle$	[14]							
$\langle 4,2,2 \rangle$	[10]							
$\langle 4,2,3 \rangle$	[1]							
$\langle 4,2,4 \rangle$	[10]							
$\langle 4,3,2 \rangle$	[10]							
$\langle 4,3,3 \rangle$	[10]							
$\langle 4,4,2 \rangle$	[10]							
$\langle 5,2,2 \rangle$	[10]							
$\langle 6,3,3 \rangle$	[14]							

Speedup (%): FMM vs. GEMM

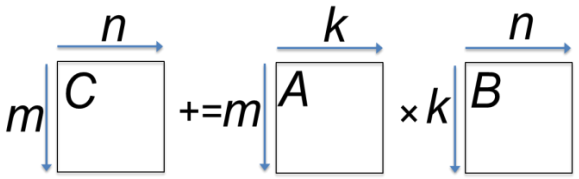
$\langle \tilde{m}, \tilde{k}, \tilde{n} \rangle$	Ref.	$\tilde{m}\tilde{k}\tilde{n}$	R	Speedup (%)				
				Theory	Practical #1		Practical #2	
					Ours	[1]	Ours	[1]
$\langle 2,2,2 \rangle$	[13]	8	7	14.3	+11.9	-3.0		
$\langle 2,3,2 \rangle$	[1]							
$\langle 2,3,4 \rangle$	[1]							
$\langle 2,4,3 \rangle$	[10]							
$\langle 2,5,2 \rangle$	[10]							
$\langle 3,2,2 \rangle$	[10]							
$\langle 3,2,3 \rangle$	[10]	18	15	20.0	+14.1	-0.7		
$\langle 3,2,4 \rangle$	[10]							
$\langle 3,3,2 \rangle$	[10]							
$\langle 3,3,3 \rangle$	[14]							
$\langle 3,3,6 \rangle$	[14]							
$\langle 3,4,2 \rangle$	[1]							
$\langle 3,4,3 \rangle$	[14]							
$\langle 3,5,3 \rangle$	[14]							
$\langle 3,6,3 \rangle$	[14]							
$\langle 4,2,2 \rangle$	[10]							
$\langle 4,2,3 \rangle$	[1]							
$\langle 4,2,4 \rangle$	[10]							
$\langle 4,3,2 \rangle$	[10]							
$\langle 4,3,3 \rangle$	[10]							
$\langle 4,4,2 \rangle$	[10]							
$\langle 5,2,2 \rangle$	[10]							
$\langle 6,3,3 \rangle$	[14]							

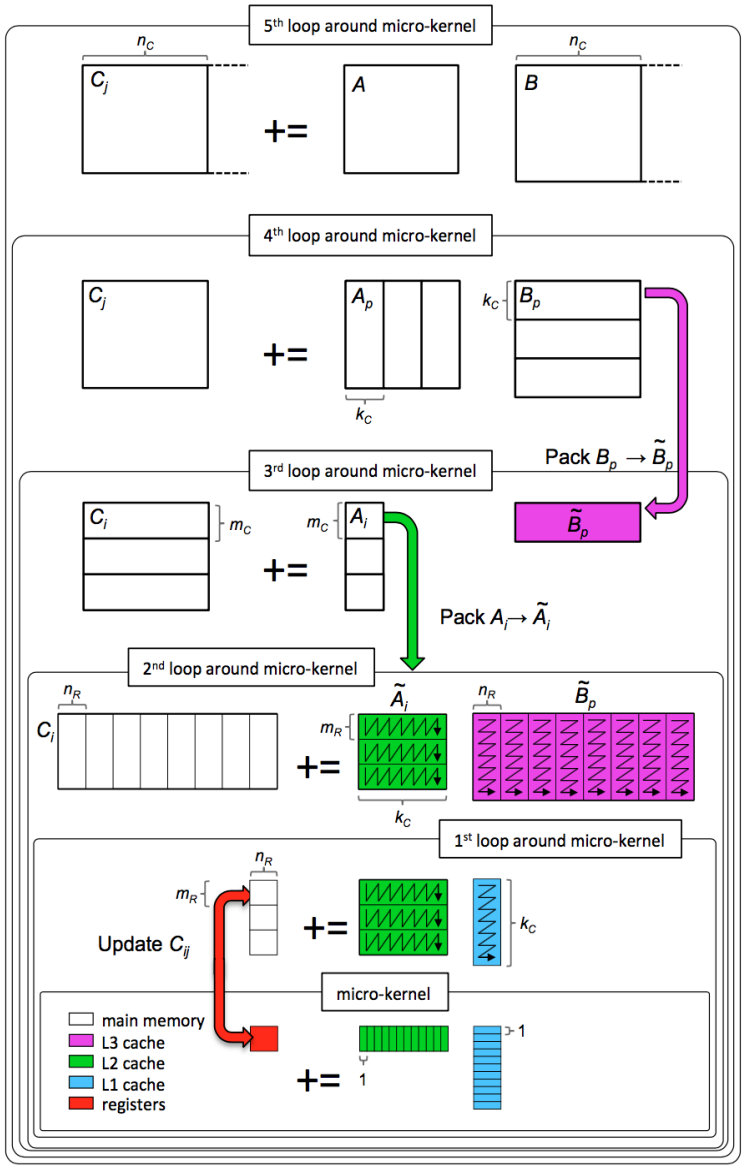
Practical #1: m=n=14400, k=480



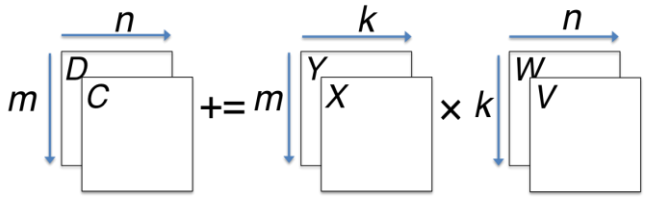
* [1] Austin R. Benson, Grey Ballard. "A framework for practical parallel fast matrix multiplication." *PPoPP*15.

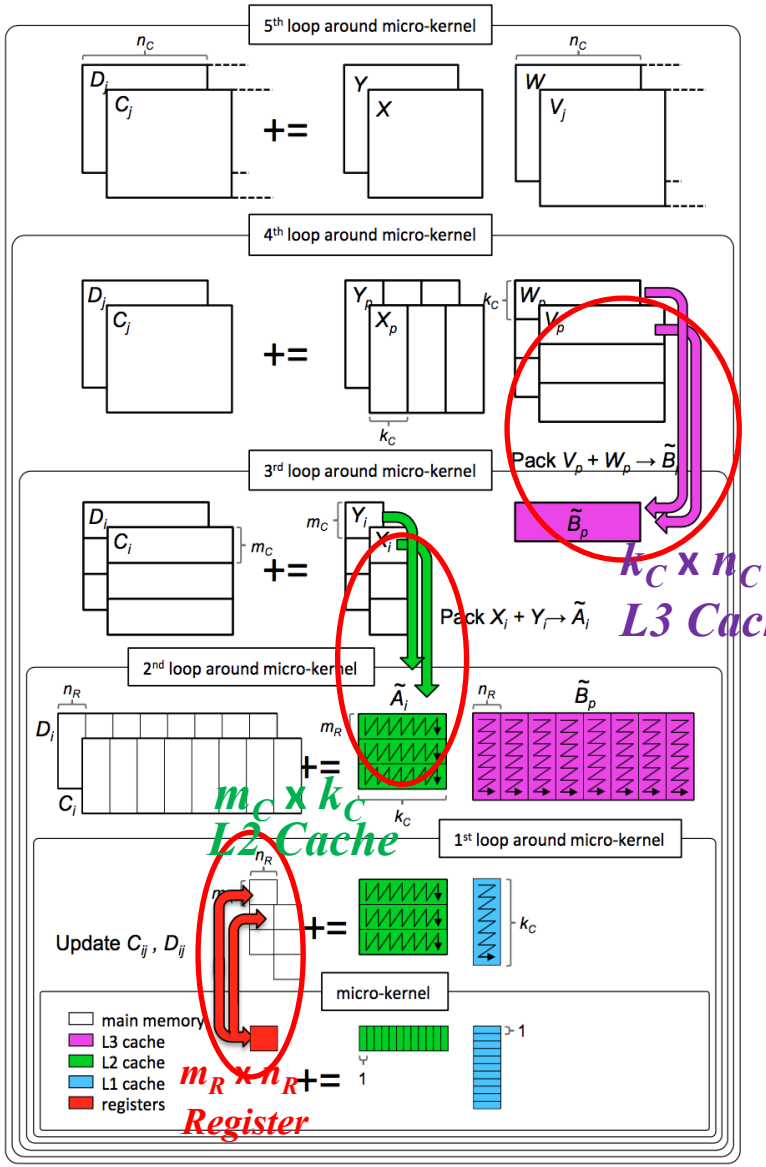
$C \mathrel{+}= AB;$





$M := (X+Y)(V+W);\ C \mathrel{+}= M;\ D \mathrel{+}= M;$

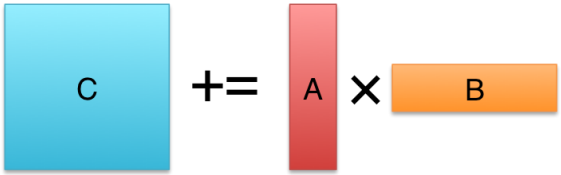




Speedup (%): FMM vs. GEMM

$\langle \tilde{m}, \tilde{k}, \tilde{n} \rangle$	Ref.	$\tilde{m}\tilde{k}\tilde{n}$	R	Speedup (%)				
				Theory	Practical #1		Practical #2	
					Ours	[1]	Ours	[1]
$\langle 2,2,2 \rangle$	[13]	8	7	14.3	+11.9	-3.0		
$\langle 2,3,2 \rangle$	[1]							
$\langle 2,3,4 \rangle$	[1]							
$\langle 2,4,3 \rangle$	[10]							
$\langle 2,5,2 \rangle$	[10]							
$\langle 3,2,2 \rangle$	[10]							
$\langle 3,2,3 \rangle$	[10]	18	15	20.0	+14.1	-0.7		
$\langle 3,2,4 \rangle$	[10]							
$\langle 3,3,2 \rangle$	[10]							
$\langle 3,3,3 \rangle$	[14]							
$\langle 3,3,6 \rangle$	[14]							
$\langle 3,4,2 \rangle$	[1]							
$\langle 3,4,3 \rangle$	[14]							
$\langle 3,5,3 \rangle$	[14]							
$\langle 3,6,3 \rangle$	[14]							
$\langle 4,2,2 \rangle$	[10]							
$\langle 4,2,3 \rangle$	[1]							
$\langle 4,2,4 \rangle$	[10]							
$\langle 4,3,2 \rangle$	[10]							
$\langle 4,3,3 \rangle$	[10]							
$\langle 4,4,2 \rangle$	[10]							
$\langle 5,2,2 \rangle$	[10]							
$\langle 6,3,3 \rangle$	[14]							

Practical #1: m=n=14400, k=480

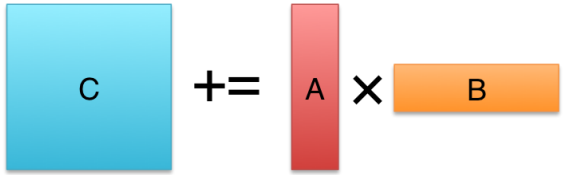


* [1] Austin R. Benson, Grey Ballard. "A framework for practical parallel fast matrix multiplication." *PPoPP*15.

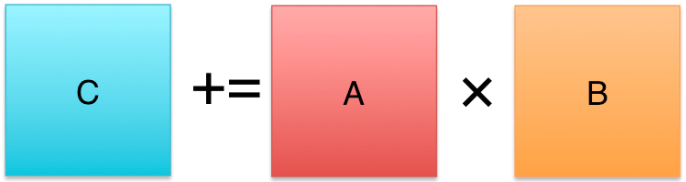
$\langle \tilde{m}, \tilde{k}, \tilde{n} \rangle$	Ref.	$\tilde{m}\tilde{k}\tilde{n}$	R	Speedup (%)				
				Theory	Practical #1		Practical #2	
					Ours	[1]	Ours	[1]
$\langle 2, 2, 2 \rangle$	[13]	8	7	14.3	+11.9	-3.0	+13.1	+13.1
$\langle 2, 3, 2 \rangle$	[1]							
$\langle 2, 3, 4 \rangle$	[1]							
$\langle 2, 4, 3 \rangle$	[10]							
$\langle 2, 5, 2 \rangle$	[10]							
$\langle 3, 2, 2 \rangle$	[10]							
$\langle 3, 2, 3 \rangle$	[10]	18	15	20.0	+14.1	-0.7	+17.2	+16.8
$\langle 3, 2, 4 \rangle$	[10]							
$\langle 3, 3, 2 \rangle$	[10]							
$\langle 3, 3, 3 \rangle$	[14]							
$\langle 3, 3, 6 \rangle$	[14]							
$\langle 3, 4, 2 \rangle$	[1]							
$\langle 3, 4, 3 \rangle$	[14]							
$\langle 3, 5, 3 \rangle$	[14]							
$\langle 3, 6, 3 \rangle$	[14]							
$\langle 4, 2, 2 \rangle$	[10]							
$\langle 4, 2, 3 \rangle$	[1]							
$\langle 4, 2, 4 \rangle$	[10]							
$\langle 4, 3, 2 \rangle$	[10]							
$\langle 4, 3, 3 \rangle$	[10]							
$\langle 4, 4, 2 \rangle$	[10]							
$\langle 5, 2, 2 \rangle$	[10]							
$\langle 6, 3, 3 \rangle$	[14]							

Speedup (%): FMM vs. GEMM

Practical #1: m=n=14400, k=480



Practical #2: m=n=14400, k=12000

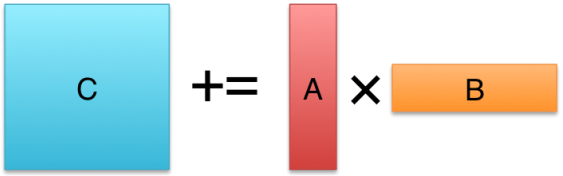


* [1] Austin R. Benson, Grey Ballard. "A framework for practical parallel fast matrix multiplication." *PPoPP*15.

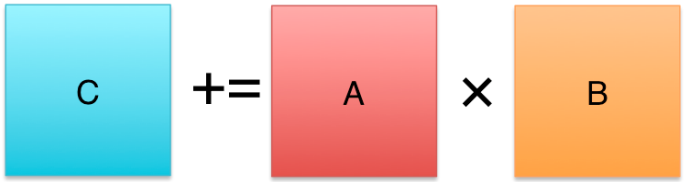
$\langle \tilde{m}, \tilde{k}, \tilde{n} \rangle$	Ref.	$\tilde{m}\tilde{k}\tilde{n}$	R	Speedup (%)				
				Theory	Practical #1		Practical #2	
					Ours	[1]	Ours	[1]
$\langle 2,2,2 \rangle$	[13]	8	7	14.3	+11.9	-3.0	+13.1	+13.1
$\langle 2,3,2 \rangle$	[1]	12	11	9.1	+5.5	-13.1	+7.7	+7.7
$\langle 2,3,4 \rangle$	[1]	24	20	20.0	+11.9	-8.0	+16.3	+17.0
$\langle 2,4,3 \rangle$	[10]	24	20	20.0	+4.8	-15.3	+14.9	+16.6
$\langle 2,5,2 \rangle$	[10]	20	18	11.1	+1.5	-23.1	+8.6	+8.3
$\langle 3,2,2 \rangle$	[10]	12	11	9.1	+7.1	-6.6	+7.2	+7.5
$\langle 3,2,3 \rangle$	[10]	18	15	20.0	+14.1	-0.7	+17.2	+16.8
$\langle 3,2,4 \rangle$	[10]	24	20	20.0	+11.9	-1.8	+16.1	+17.0
$\langle 3,3,2 \rangle$	[10]	18	15	20.0	+11.4	-8.1	+17.3	+16.5
$\langle 3,3,3 \rangle$	[14]	27	23	17.4	+8.6	-9.3	+14.4	+14.7
$\langle 3,3,6 \rangle$	[14]	54	40	35.0	-34.0	-41.6	+24.2	+20.1
$\langle 3,4,2 \rangle$	[1]	24	20	20.0	+4.9	-15.7	+16.0	+16.8
$\langle 3,4,3 \rangle$	[14]	36	29	24.1	+8.4	-12.6	+18.1	+20.1
$\langle 3,5,3 \rangle$	[14]	45	36	25.0	+5.2	-20.6	+19.1	+18.9
$\langle 3,6,3 \rangle$	[14]	54	40	35.0	-21.6	-64.5	+19.5	+17.8
$\langle 4,2,2 \rangle$	[10]	16	14	14.3	+9.4	-4.7	+11.9	+12.2
$\langle 4,2,3 \rangle$	[1]	24	20	20.0	+12.1	-2.3	+15.9	+17.3
$\langle 4,2,4 \rangle$	[10]	32	26	23.1	+10.4	-2.7	+18.4	+19.1
$\langle 4,3,2 \rangle$	[10]	24	20	20.0	+11.3	-7.8	+16.8	+15.7
$\langle 4,3,3 \rangle$	[10]	36	29	24.1	+8.1	-8.4	+19.8	+20.0
$\langle 4,4,2 \rangle$	[10]	32	26	23.1	-4.2	-18.4	+17.1	+18.5
$\langle 5,2,2 \rangle$	[10]	20	18	11.1	+7.0	-6.7	+8.2	+8.5
$\langle 6,3,3 \rangle$	[14]	54	40	35.0	-33.4	-42.2	+24.0	+20.2

Speedup (%): FMM vs. GEMM

Practical #1: m=n=14400, k=480



Practical #2: m=n=14400, k=12000



* [1] Austin R. Benson, Grey Ballard. "A framework for practical parallel fast matrix multiplication." *PPoPP*15.

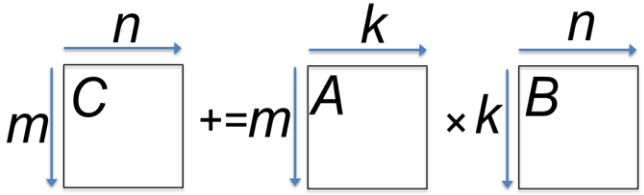
One-level Fast Matrix Multiplication (FMM)

for $r = 0, \ldots, R-1$,

$$M_r := \left(\sum_{i=0}^{\widetilde{m}\widetilde{k}-1} u_{ir} A_i \right) \times \left(\sum_{j=0}^{\widetilde{k}\widetilde{n}-1} v_{jr} B_j \right);$$

$$C_p \mathrel{+}= w_{pr} M_r \ (p = 0, \ldots, \widetilde{m}\widetilde{n}-1)$$

$$C = \left(\begin{array}{c|c|c} C_0 & \cdots & C_{\widetilde{n}-1} \\ \hline \vdots & & \vdots \\ \hline C_{(\widetilde{m}-1)\widetilde{n}} & \cdots & C_{\widetilde{m}\widetilde{n}-1} \end{array} \right), A = \left(\begin{array}{c|c|c} A_0 & \cdots & A_{\widetilde{k}-1} \\ \hline \vdots & & \vdots \\ \hline A_{(\widetilde{m}-1)\widetilde{k}} & \cdots & A_{\widetilde{m}\widetilde{k}-1} \end{array} \right), B = \left(\begin{array}{c|c|c} B_0 & \cdots & B_{\widetilde{n}-1} \\ \hline \vdots & & \vdots \\ \hline B_{(\widetilde{k}-1)\widetilde{n}} & \cdots & B_{\widetilde{k}\widetilde{n}-1} \end{array} \right)$$



The set of coefficients that determine the $\langle \widetilde{m}, \widetilde{k}, \widetilde{n} \rangle$ algorithm is denoted as $[\![U, V, W]\!]$.

Two-level Fast Matrix Multiplication (FMM)

for $r = 0, \ldots, R-1$,

$$M_r := \left(\sum_{i=0}^{\widetilde{m}\widetilde{k}-1} u_{ir} A_i \right) \times \left(\sum_{j=0}^{\widetilde{k}\widetilde{n}-1} v_{jr} B_j \right);$$

$$C_p \mathrel{+}= w_{pr} M_r \ (p = 0, \ldots, \widetilde{m}\widetilde{n}-1)$$

$$C = \left(\begin{array}{c|c|c} C_0 & \cdots & C_{\widetilde{n}-1} \\ \hline \vdots & & \vdots \\ \hline C_{(\widetilde{m}-1)\widetilde{n}} & \cdots & C_{\widetilde{m}\widetilde{n}-1} \end{array} \right), A = \left(\begin{array}{c|c|c} A_0 & \cdots & A_{\widetilde{k}-1} \\ \hline \vdots & & \vdots \\ \hline A_{(\widetilde{m}-1)\widetilde{k}} & \cdots & A_{\widetilde{m}\widetilde{k}-1} \end{array} \right), B = \left(\begin{array}{c|c|c} B_0 & \cdots & B_{\widetilde{n}-1} \\ \hline \vdots & & \vdots \\ \hline B_{(\widetilde{k}-1)\widetilde{n}} & \cdots & B_{\widetilde{k}\widetilde{n}-1} \end{array} \right)$$

1st Level:
Partition: $\langle \widetilde{m}, \widetilde{k}, \widetilde{n} \rangle$
Coefficient matrices: $[\![U, V, W]\!]$

$$C = \left(\begin{array}{c|c|c} C_0 & \cdots & C_{\widetilde{n}'-1} \\ \hline \vdots & & \vdots \\ \hline C_{(\widetilde{m}'-1)\widetilde{n}'} & \cdots & C_{\widetilde{m}'\widetilde{n}'-1} \end{array} \right), A = \left(\begin{array}{c|c|c} A_0 & \cdots & A_{\widetilde{k}'-1} \\ \hline \vdots & & \vdots \\ \hline A_{(\widetilde{m}'-1)\widetilde{k}'} & \cdots & A_{\widetilde{m}'\widetilde{k}'-1} \end{array} \right), B = \left(\begin{array}{c|c|c} B_0 & \cdots & B_{\widetilde{n}'-1} \\ \hline \vdots & & \vdots \\ \hline B_{(\widetilde{k}'-1)\widetilde{n}'} & \cdots & B_{\widetilde{k}'\widetilde{n}'-1} \end{array} \right)$$

2nd Level:
Partition: $\langle \widetilde{m}', \widetilde{k}', \widetilde{n}' \rangle$
Coefficient matrices: $[\![U', V', W']\!]$

m | $+=m$ | $\times k$

of coefficients determine the algorithm is denoted as $[\![\widetilde{U}', \widetilde{V}', \widetilde{W}']\!]'$.

The set of coefficients of a two-level $\langle \widetilde{m}, \widetilde{k}, \widetilde{n} \rangle$ and $\langle \widetilde{m}', \widetilde{k}', \widetilde{n}' \rangle$ FMM algorithm can be denoted as .

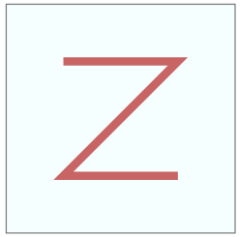
Two-level Fast Matrix Multiplication (FMM)

for $r = 0, \ldots, R \cdot R' - 1$,

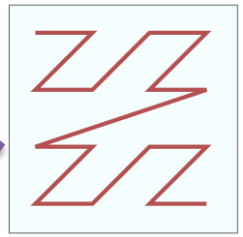
Kronecker Product!

$$M_r := \left(\sum_{i=0}^{\widetilde{mk}\cdot\widetilde{m'k'}-1} (U \otimes U')_{i,r} A_i \right) \times \left(\sum_{j=0}^{\widetilde{kn}\cdot\widetilde{k'n'}-1} (V \otimes V')_{j,r} B_j \right);$$

$$C_p \mathrel{+}= (W \otimes W')_{p,r} M_r \quad (p = 0, \ldots, \widetilde{mn} \cdot \widetilde{m'n'} - 1)$$



1-level Morton-like ordering index



2-level Morton-like ordering index

2-level Morton-like ordering index

$$C = \begin{pmatrix} C_0 & \cdots & C_{\widetilde{n}-1} \\ \hline \vdots & & \vdots \\ \hline C_{\widetilde{(m-1)}\widetilde{n}} & \cdots & C_{\widetilde{mn}-1} \end{pmatrix}, A = \begin{pmatrix} A_0 & \cdots & A_{\widetilde{k}-1} \\ \hline \vdots & & \vdots \\ \hline A_{\widetilde{(m-1)}\widetilde{k}} & \cdots & A_{\widetilde{mk}-1} \end{pmatrix}, B = \begin{pmatrix} B_0 & \cdots & B_{\widetilde{n}-1} \\ \hline \vdots & & \vdots \\ \hline B_{\widetilde{(k-1)}\widetilde{n}} & \cdots & B_{\widetilde{kn}-1} \end{pmatrix}$$

1st Level:
Partition: $\langle \widetilde{m}, \widetilde{k}, \widetilde{n} \rangle$
Coefficient matrices: $[\![U, V, W]\!]$

$$C = \begin{pmatrix} C_0 & \cdots & C_{\widetilde{n'}-1} \\ \hline \vdots & & \vdots \\ \hline C_{\widetilde{(m'-1)}\widetilde{n'}} & \cdots & C_{\widetilde{m'n'}-1} \end{pmatrix}, A = \begin{pmatrix} A_0 & \cdots & A_{\widetilde{k'}-1} \\ \hline \vdots & & \vdots \\ \hline A_{\widetilde{(m'-1)}\widetilde{k'}} & \cdots & A_{\widetilde{m'k'}-1} \end{pmatrix}, B = \begin{pmatrix} B_0 & \cdots & B_{\widetilde{n'}-1} \\ \hline \vdots & & \vdots \\ \hline B_{\widetilde{(k'-1)}\widetilde{n'}} & \cdots & B_{\widetilde{k'n'}-1} \end{pmatrix}$$

2nd Level:
Partition: $\langle \widetilde{m'}, \widetilde{k'}, \widetilde{n'} \rangle$
Coefficient matrices: $[\![U', V', W']\!]$

The set of coefficients of a two-level $\langle \widetilde{m}, \widetilde{k}, \widetilde{n} \rangle$ and $\langle \widetilde{m'}, \widetilde{k'}, \widetilde{n'} \rangle$ FMM algorithm can be denoted as $[\![U ? U', V ? V', W ? W']\!]$.

Code Generator

- FMM implementation Generator
  - Generate ~200 implementations
- Model Generator
  - Generate the performance model for each implementation
  - Predict the relative performance accurately

Performance Model

$$T = T_a + T_m$$

$$T_a = N_a^{\times} \cdot T_a^{\times} + N_a^{A+} \cdot T_a^{A+} + N_a^{B+} \cdot T_a^{B+} + N_a^{C+} \cdot T_a^{C+}$$

$$T_m = N_m^{A\times} \cdot T_m^{A\times} + N_m^{B\times} \cdot T_m^{B\times} + N_m^{C\times} \cdot T_m^{C\times} + N_m^{A+} \cdot T_m^{A+} + N_m^{B+} \cdot T_m^{B+} + N_m^{C+} \cdot T_m^{C+}$$

	type	τ	GEMM	L-level
$T_a^{\times}$	-	τ_a	$2mnk$	$2\frac{m}{\widetilde{M_L}}\frac{n}{\widetilde{N_L}}\frac{k}{\widetilde{K_L}}$
T_a^{A+}	-	τ_a	-	$2\frac{m}{\widetilde{M_L}}\frac{k}{\widetilde{K_L}}$
T_a^{B+}	-	τ_a	-	$2\frac{k}{\widetilde{K_L}}\frac{n}{\widetilde{N_L}}$
T_a^{C+}	-	τ_a	-	$2\frac{m}{\widetilde{M_L}}\frac{n}{\widetilde{N_L}}$
$T_m^{A\times}$	r	τ_b	$mk\lceil\frac{n}{n_c}\rceil$	$\frac{m}{\widetilde{M_L}}\frac{k}{\widetilde{K_L}}\lceil\frac{n/\widetilde{N_L}}{n_c}\rceil$
$\widetilde{T_m^{A\times}}$	w	τ_b	$mk\lceil\frac{n}{n_c}\rceil$	$\frac{m}{\widetilde{M_L}}\frac{k}{\widetilde{K_L}}\lceil\frac{n/\widetilde{N_L}}{n_c}\rceil$
$T_m^{B\times}$	r	τ_b	nk	$\frac{n}{\widetilde{N_L}}\frac{k}{\widetilde{K_L}}$
$\widetilde{T_m^{B\times}}$	w	τ_b	nk	$\frac{n}{\widetilde{N_L}}\frac{k}{\widetilde{K_L}}$
$T_m^{C\times}$	r/w	τ_b	$2\lambda mn\lceil\frac{k}{k_c}\rceil$	$2\lambda\frac{m}{\widetilde{M_L}}\frac{n}{\widetilde{N_L}}\lceil\frac{k/\widetilde{K_L}}{k_c}\rceil$
T_m^{A+}	r/w	τ_b	mk	$\frac{m}{\widetilde{M_L}}\frac{k}{\widetilde{K_L}}$
T_m^{B+}	r/w	τ_b	nk	$\frac{n}{\widetilde{N_L}}\frac{k}{\widetilde{K_L}}$
T_m^{C+}	r/w	τ_b	mn	$\frac{m}{\widetilde{M_L}}\frac{n}{\widetilde{N_L}}$

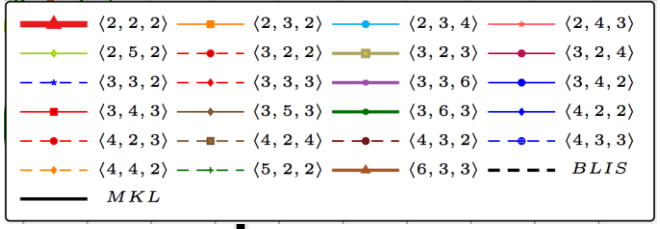
	GEMM	L-level		
		ABC	AB	Naive
$N_a^{\times}$	1	R_L	R_L	R_L
N_a^{A+}	-	$nnz(\bigotimes U)$-R_L	$nnz(\bigotimes U)$-R_L	$nnz(\bigotimes U)$-R_L
N_a^{B+}	-	$nnz(\bigotimes V)$-R_L	$nnz(\bigotimes V)$-R_L	$nnz(\bigotimes V)$-R_L
N_a^{C+}	-	$nnz(\bigotimes W)$	$nnz(\bigotimes W)$	$nnz(\bigotimes W)$
$N_m^{A\times}$	1	$nnz(\bigotimes U)$	$nnz(\bigotimes U)$	R_L
$\widetilde{N_m^{A\times}}$	-	-	-	-
$N_m^{B\times}$	1	$nnz(\bigotimes V)$	$nnz(\bigotimes V)$	R_L
$\widetilde{N_m^{B\times}}$	-	-	-	-
$N_m^{C\times}$	1	$nnz(\bigotimes W)$	R_L	R_L
N_m^{A+}	-	-	-	$nnz(\bigotimes U)$+R_L
N_m^{B+}	-	-	-	$nnz(\bigotimes V)$+R_L
N_m^{C+}	-	-	$3nnz(\bigotimes W)$	$3nnz(\bigotimes W)$

$\widetilde{M_L} = \prod_{l=0}^{L-1} \widetilde{m_l}$, $\widetilde{K_L} = \prod_{l=0}^{L-1} \widetilde{k_l}$, $\widetilde{N_L} = \prod_{l=0}^{L-1} \widetilde{n_l}$, $R_L = \prod_{l=0}^{L-1} R_l$.

$\bigotimes U = \bigotimes_{l=0}^{L-1} U_l$, $\bigotimes V = \bigotimes_{l=0}^{L-1} V_l$, $\bigotimes W = \bigotimes_{l=0}^{L-1} W_l$.

$m = n = 14400$, 1 level

2 level, ABC

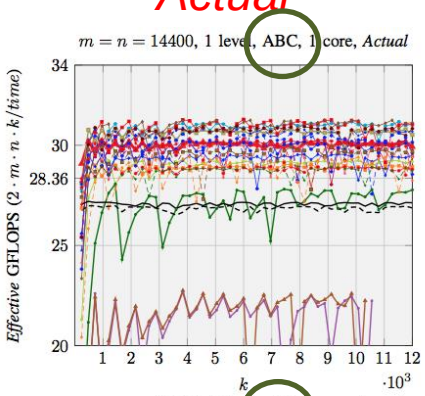


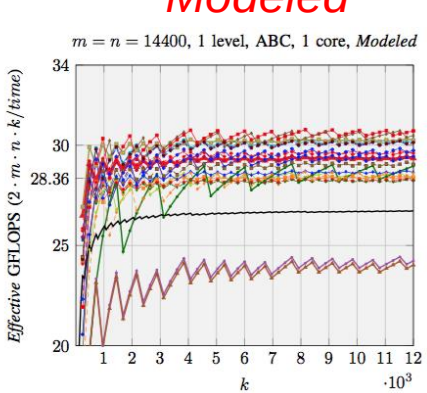
Actual

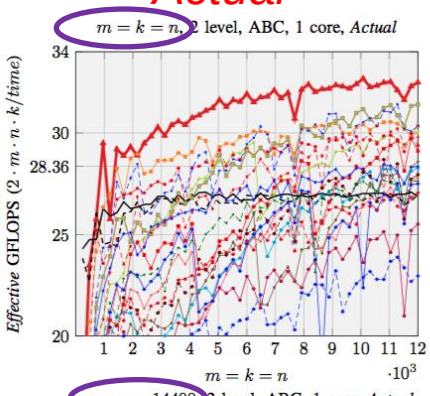
Modeled

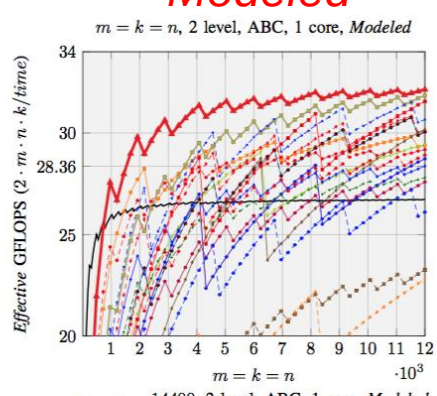
Actual

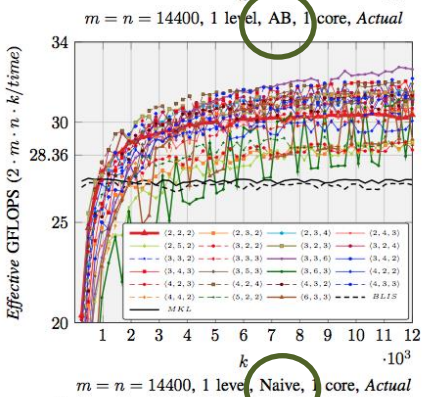
Modeled

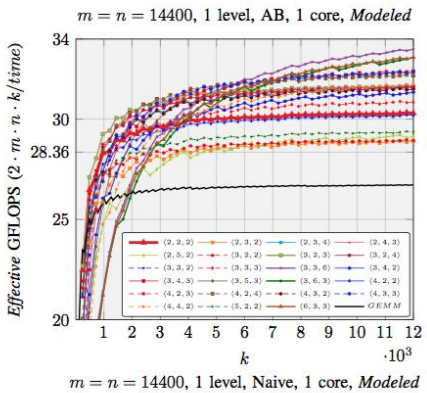


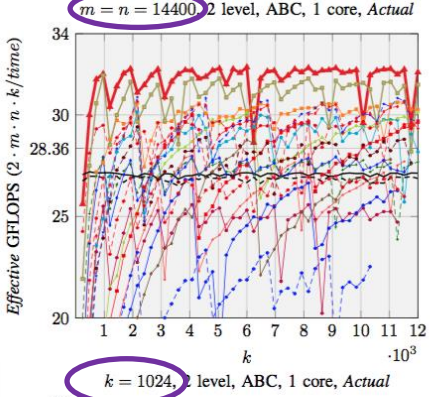


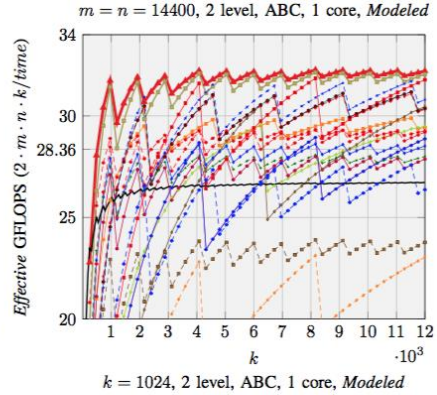


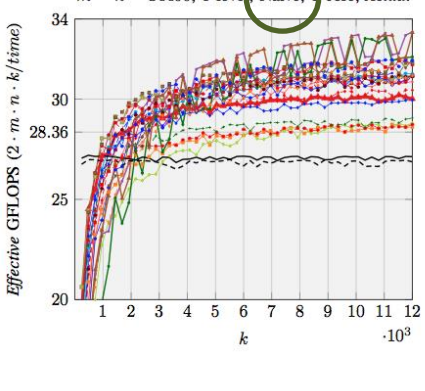


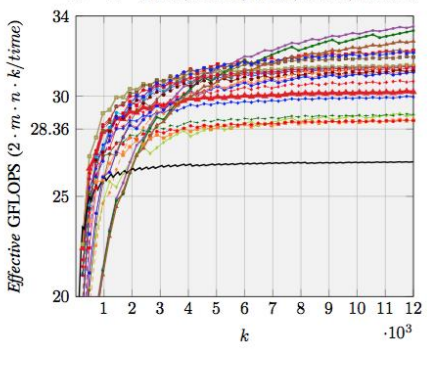


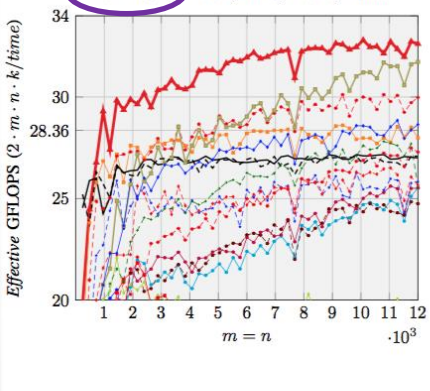


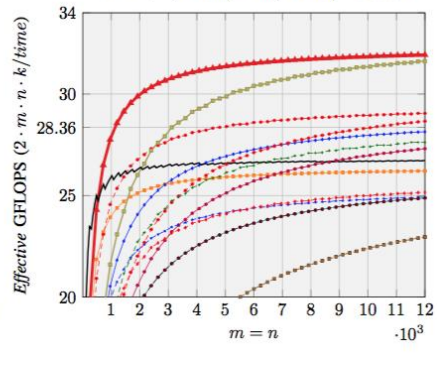












Outline

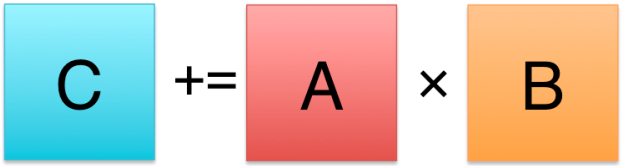
- Motivation
- Background
- Practical Strassen’s Algorithm (SC16)
- Practical Fast Matrix Multiplication (IPDPS17)
- Proposed Work
- Conclusion

1. Tensor Contraction

Jianyu Huang, Devin A. Matthews, and Robert A. van de Geijn. “Strassen’s Algorithm for Tensor Contraction.” arXiv:1704.03092 (2017). In Submission.

Matrix vs. Tensor

Matrix Multiplication

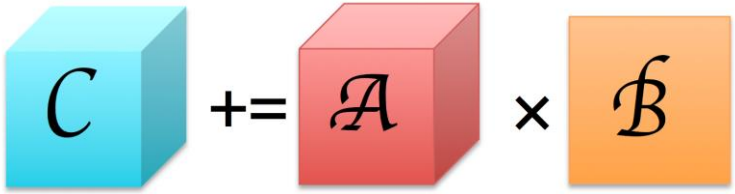


$$C += AB$$

$$C_{i,j} += \sum_k A_{i,k} B_{k,j}$$

BLAS/BLIS!

Tensor Contraction



$$\mathcal{C} += \mathcal{A}\mathcal{B}$$

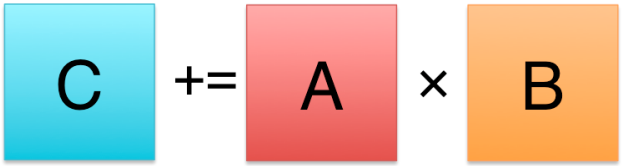
$$\mathcal{C}_{a,b,c} += \sum_d \mathcal{A}_{d,c,a} \mathcal{B}_{d,b}$$

TBLIS!

Devin A. Matthews. “High-Performance Tensor Contraction without Transposition.” Accepted in *SISC*.

Matrix vs. Tensor

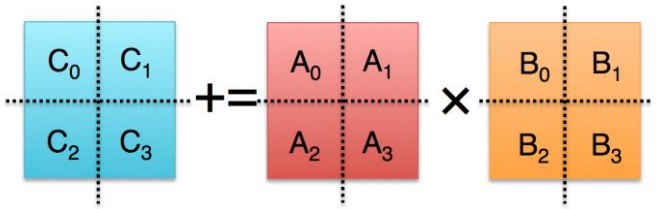
Matrix Multiplication



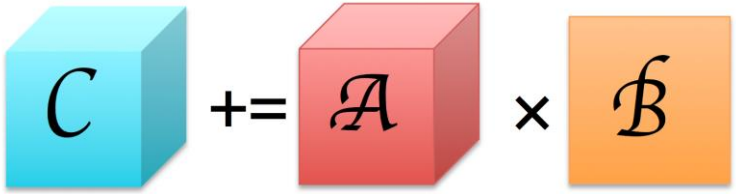
$$C \mathrel{+}= AB$$

$$C_{i,j} \mathrel{+}= \sum_k A_{i,k} B_{k,j}$$

BLAS/BLIS!



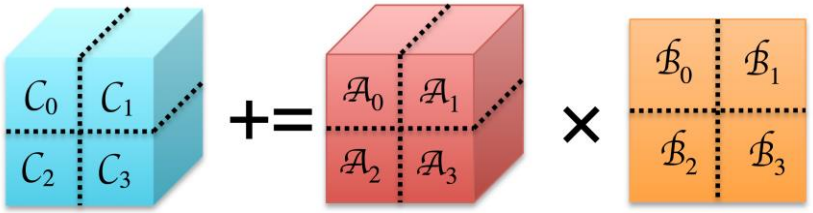
Tensor Contraction



$$\mathcal{C} \mathrel{+}= \mathcal{A}\mathcal{B}$$

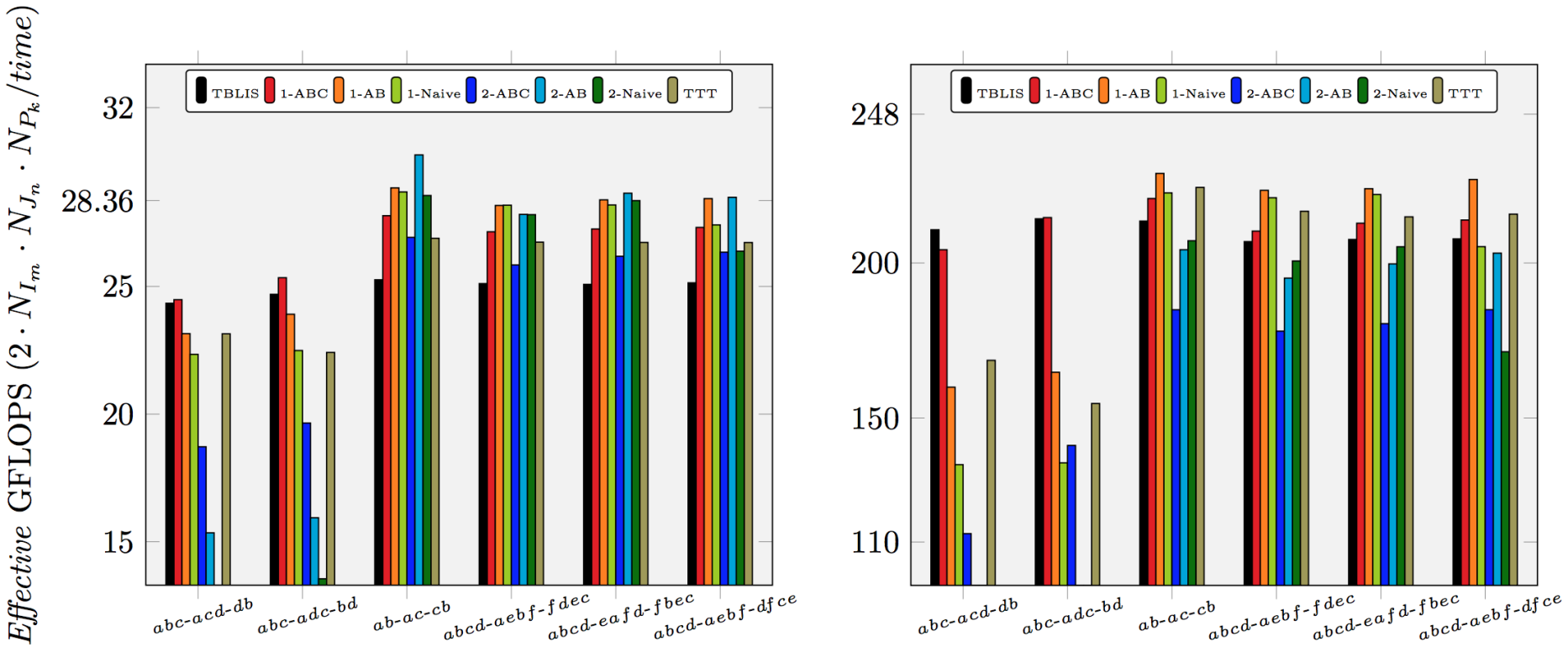
$$\mathcal{C}_{a,b,c} \mathrel{+}= \sum_d \mathcal{A}_{d,c,a} \mathcal{B}_{d,b}$$

TBLIS!



Devin A. Matthews. "High-Performance Tensor Contraction without Transposition." Accepted in *SISC*.

Performance for Tensor Strassen



$C_{abcd} += A_{aebf} B_{dfce}$ is denoted as *abcd-aebf-dfce*

Jianyu Huang, Devin A. Matthews, and Robert A. van de Geijn. "Strassen's Algorithm for Tensor Contraction." arXiv:1704.03092 (2017). In Submission.

2. Other extensions

- Distributed Memory:
  - Run Strassen on the inter-processor level:
    utilize larger submatrix size
  - Reduce the communication overhead:
    fuse packing with collective communications
- Machine Learning Primitives:
  - GraphBLAS primitive operation:

$$\mathcal{C} = \bigoplus_{k} \mathcal{A}_{ik} \bigotimes \mathcal{B}_{kj}$$

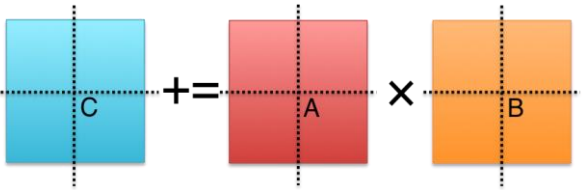
  - Generaralized GEMM-like ML operation:

$$\theta(\mathcal{C})_{i:} = \bigodot_{j} \mathcal{K}(\bigoplus_{k} \alpha(\mathcal{A})_{ik} \bigotimes \beta(\mathcal{B})_{kj})$$

Outline

- Motivation
- Background
- Practical Strassen’s Algorithm (SC16)
- Practical Fast Matrix Multiplication (IPDPS17)
- Proposed Work
- Conclusion

To achieve practical high performance of Strassen/FMM algorithms……



	Conventional Implementations		Our Solutions	
Matrix Size	Must be large	☹	C += A × B	☺
Matrix Shape	Must be square	☹	n, k, m, m, C += A × B, n	☺
No Additional Workspace	✗	☹	✓	☺
Parallelism	Usually task parallelism	☹	Can be data parallelism	☺

Proposed Work

- Practical high-performance Strassen.
  - Utilize a performance model to predict the relative performance

❑ Jianyu Huang, Tyler Smith, Greg Henry, and Robert van de Geijn. "Strassen's Algorithm Reloaded." In ***SC'16***.

- Practical fast matrix multiplication algorithms.
  - Leverage a code generator to generate ~200 implmentations

❑ Jianyu Huang, Leslie Rice, Devin A. Matthews, Robert A. van de Geijn. "Generating Families of Practical Fast Matrix Multiplication Algorithms." In ***IPDPS17***

- Acceleration of tensor contractions via Strassen.
  - First paper to apply Strassen's algorithm for tensor contraction.

❑ Jianyu Huang, Devin A. Matthews, and Robert A. van de Geijn. "Strassen's Algorithm for Tensor Contraction." arXiv:1704.03092 (2017). In Submission.

- Exploration of Strassen/FMM for distributed memory.
- Fusion with machine learning applications.
- Pedagogical outreach for Strassen.

❑ Jianyu Huang, Robert A. van de Geijn. "BLISlab: A Sandbox for Optimizing GEMM." FLAME Working Note #80, The University of Texas at Austin, Department of Computer Science. Technical Report TR-16-13. August 31, 2016.

Publications

- Papers
  - **Jianyu Huang**, Devin A. Matthews, and Robert A. van de Geijn. “Strassen’s Algorithm for Tensor Contraction.” arXiv:1704.03092 (2017). In Submission.
  - **Jianyu Huang**, Leslie Rice, Devin A. Matthews, Robert A. van de Geijn. “Generating Families of Practical Fast Matrix Multiplication Algorithms.” In ***IPDPS17***
  - **Jianyu Huang**, Tyler M. Smith, Greg H. Henry, and Robert A. van de Geijn. “Strassen’s Algorithm Reloaded.” In ***SC’16***.
  - Chenhan D. Yu, **Jianyu Huang**, Woody Austin, Bo Xiao, George Biros. “Performance Optimization for the K-Nearest Neighbors Kernel on x86 Architectures.” In ***SC’15***
- Software
  - Code Generator for Practical Fast Matrix Multiplication Algorithms:

    https://github.com/flame/fmm-gen
  - Strassen’s Algorithm for Tensor Contraction:

    https://github.com/flame/tblis-strassen
- Pedagogical Effort
  - **Jianyu Huang**, Robert A. van de Geijn. “BLISlab: A Sandbox for Optimizing GEMM.” FLAME Working Note #80, The University of Texas at Austin, Department of Computer Science. Technical Report TR-16-13. August 31, 2016.

    https://github.com/flame/blislab

Thank you!

Acknowledgement