Discovering Fast Matrix Multiplication Algorithms via Tensor Decomposition

Grey Ballard

Wake Forest University

SIAM Conference on Computational Science & Engineering
March 1, 2017

Collaborators

This is joint work with...

- Austin Benson - Stanford University
- Christian Ikenmeyer - Max-Planck Institute for Informatics
- Tammy Kolda - Sandia National Laboratories
- JM Landsberg - Texas A&M University
- Kathryn Rouse - Wake Forest University
- Nick Ryder - University of California Berkeley

Fast Matrix Multiplication

Definition

Fast matrix multiplication algorithms require $o(n^3)$ arithmetic operations to multiply $n \times n$ matrices.

Examples

- Strassen [Str69]: $O(n^{\log_2 7}) = O(n^{2.81})$
- Coppersmith-Winograd [CW87]: $O(n^{2.376})$
- Le Gall [LG14]: $O(n^{2.373})$

Strassen's Algorithm

Strassen's algorithm uses 7 multiplies for 2 $\times$ 2 multiplication

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

Classical Algorithm

$$\begin{aligned}
M_1 &= A_{11} \cdot B_{11} \\
M_2 &= A_{12} \cdot B_{21} \\
M_3 &= A_{11} \cdot B_{12} \\
M_4 &= A_{12} \cdot B_{22} \\
M_5 &= A_{21} \cdot B_{11} \\
M_6 &= A_{22} \cdot B_{21} \\
M_7 &= A_{21} \cdot B_{12} \\
M_8 &= A_{22} \cdot B_{22} \\
C_{11} &= M_1 + M_2 \\
C_{12} &= M_3 + M_4 \\
C_{21} &= M_5 + M_6 \\
C_{22} &= M_7 + M_8
\end{aligned}$$

Strassen's Algorithm

$$\begin{aligned}
M_1 &= (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \\
M_2 &= (A_{21} + A_{22}) \cdot B_{11} \\
M_3 &= A_{11} \cdot (B_{12} - B_{22}) \\
M_4 &= A_{22} \cdot (B_{21} - B_{11}) \\
M_5 &= (A_{11} + A_{12}) \cdot B_{22} \\
M_6 &= (A_{21} - A_{11}) \cdot (B_{11} + B_{12}) \\
M_7 &= (A_{12} - A_{22}) \cdot (B_{21} + B_{22}) \\
C_{11} &= M_1 + M_4 - M_5 + M_7 \\
C_{12} &= M_3 + M_5 \\
C_{21} &= M_2 + M_4 \\
C_{22} &= M_1 - M_2 + M_3 + M_6
\end{aligned}$$

Strassen's algorithm uses 7 multiplies for 2×2 multiplication

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

For $n \times n$ matrices, we split into quadrants and use recursion

Strassen's Algorithm

$$\begin{aligned}
M_1 &= (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \\
M_2 &= (A_{21} + A_{22}) \cdot B_{11} \\
M_3 &= A_{11} \cdot (B_{12} - B_{22}) \\
M_4 &= A_{22} \cdot (B_{21} - B_{11}) \\
M_5 &= (A_{11} + A_{12}) \cdot B_{22} \\
M_6 &= (A_{21} - A_{11}) \cdot (B_{11} + B_{12}) \\
M_7 &= (A_{12} - A_{22}) \cdot (B_{21} + B_{22})
\end{aligned}$$

$$\begin{aligned}
C_{11} &= M_1 + M_4 - M_5 + M_7 \\
C_{12} &= M_3 + M_5 \\
C_{21} &= M_2 + M_4 \\
C_{22} &= M_1 - M_2 + M_3 + M_6
\end{aligned}$$

Flop count recurrence:

$$F(n) = 7 \cdot F(n/2) + O(n^2)$$

$$F(1) = 1$$

$$F(n) = O\left(n^{\log_2 7}\right)$$

$$\log_2 7 \approx 2.81$$

Recursion allows us to focus on base case

2×2

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

multiplies	6	7	8
flop count	$O\left(n^{2.58}\right)$	$O\left(n^{2.81}\right)$	$O\left(n^{3}\right)$

Recursion allows us to focus on base case

2×2

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

multiplies	$\cancel{6}$	7	8
flop count	$\cancel{O\left(n^{2.58}\right)}$	$O\left(n^{2.81}\right)$	$O\left(n^3\right)$

Recursion allows us to focus on base case

2×2

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}$$

multiplies	$\cancel{6}$	7	8
flop count	$\cancel{O\left(n^{2.58}\right)}$	$O\left(n^{2.81}\right)$	$O\left(n^{3}\right)$

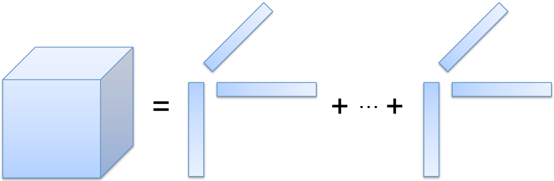
3×3

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

multiplies	19	21	23	27
flop count	$O\left(n^{2.68}\right)$	$O\left(n^{2.77}\right)$	$O\left(n^{2.85}\right)$	$O\left(n^{3}\right)$

Searching for a base case algorithm

Finding a base case algorithm corresponds to computing an exact CP decomposition of a particular 3D tensor



$$\mathcal{T} = \sum_{r=1}^{R} \mathbf{u}_r \circ \mathbf{v}_r \circ \mathbf{w}_r$$

*Note the $=$ sign: we're not looking for an approximation

2×2 matrix multiplication as a tensor operation

$$\mathbf{A} \cdot \mathbf{B} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \mathbf{C}$$

is equivalent to

$$\mathcal{T} \times_1 \mathbf{a} \times_2 \mathbf{b} = \mathcal{T} \times_1 \begin{pmatrix} a_{11} \\ a_{12} \\ a_{21} \\ a_{22} \end{pmatrix} \times_2 \begin{pmatrix} b_{11} \\ b_{12} \\ b_{21} \\ b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} \\ c_{21} \\ c_{12} \\ c_{22} \end{pmatrix} = \mathbf{c}$$

where $\mathcal{T}$ is a $4 \times 4 \times 4$ tensor with the following slices:

$$\mathbf{T}_1 = \begin{pmatrix} 1 & & & \\ & & 1 & \\ & & & \\ & & & \end{pmatrix} \quad \mathbf{T}_2 = \begin{pmatrix} & & & \\ & & & \\ 1 & & & \\ & & 1 & \end{pmatrix} \quad \mathbf{T}_3 = \begin{pmatrix} & 1 & & \\ & & & 1 \\ & & & \\ & & & \end{pmatrix} \quad \mathbf{T}_4 = \begin{pmatrix} & & & \\ & & & \\ & 1 & & \\ & & & 1 \end{pmatrix}$$

2×2 matrix multiplication as a tensor operation

$$\mathbf{A} \cdot \mathbf{B} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \mathbf{C}$$

is equivalent to

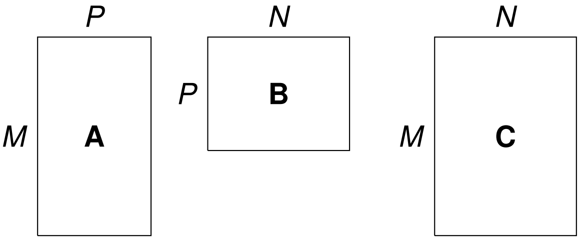
$$\mathcal{T} \times_1 \mathbf{a} \times_2 \mathbf{b} = \mathcal{T} \times_1 \begin{pmatrix} a_{11} \\ a_{12} \\ a_{21} \\ a_{22} \end{pmatrix} \times_2 \begin{pmatrix} b_{11} \\ b_{12} \\ b_{21} \\ b_{22} \end{pmatrix} = \begin{pmatrix} c_{11} \\ c_{21} \\ c_{12} \\ c_{22} \end{pmatrix} = \mathbf{c}$$

For example:

$$\mathbf{T}_2 \times_1 \begin{pmatrix} a_{11} \\ a_{12} \\ a_{21} \\ a_{22} \end{pmatrix} \times_2 \begin{pmatrix} b_{11} \\ b_{12} \\ b_{21} \\ b_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} & & & \\ & & & \\ 1 & & & \\ & & 1 & \end{pmatrix} \begin{pmatrix} b_{11} \\ b_{12} \\ b_{21} \\ b_{22} \end{pmatrix} = a_{21} b_{11} + a_{22} b_{21} = c_{21}$$

General matrix multiplication tensor

$\langle M, P, N\rangle$ means multiplying $M \times P$ by $P \times N$



Matrix multiplication tensor $\mathfrak{T}$ is $MP \times PN \times MN$
Number of 1's in $\mathfrak{T}$ is MPN

$$\langle M, P, N\rangle \equiv \langle N, M, P\rangle \equiv \langle P, N, M\rangle \equiv \langle P, M, N\rangle \equiv \langle M, N, P\rangle \equiv \langle N, P, M\rangle$$

Matrix multiplication using low-rank decomposition

Here's the matrix multiplication as tensor operation again:

$$\mathcal{T} \times_1 \mathbf{a} \times_2 \mathbf{b} = \mathbf{c}$$

Here's our low-rank decomposition:

$$\mathcal{T} = \sum_{r=1}^{R} \mathbf{u}_r \circ \mathbf{v}_r \circ \mathbf{w}_r$$

Here's an encoding of our matrix multiplication algorithm:

$$\mathcal{T} \times_1 \mathbf{a} \times_2 \mathbf{b} = \sum_{r=1}^{R} (\mathbf{u}_r \circ \mathbf{v}_r \circ \mathbf{w}_r) \times_1 \mathbf{a} \times_2 \mathbf{b} = \sum_{r=1}^{R} \left[(\mathbf{a}^T \mathbf{u}_r) \cdot (\mathbf{b}^T \mathbf{v}_r) \right] \mathbf{w}_r = \mathbf{c}$$

Connection between factor matrices and algorithm

Strassen's algorithm

$$
\begin{aligned}
M_1 &= (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \\
M_2 &= (A_{21} + A_{22}) \cdot B_{11} \\
M_3 &= A_{11} \cdot (B_{12} - B_{22}) \\
M_4 &= A_{22} \cdot (B_{21} - B_{11}) \\
M_5 &= (A_{11} + A_{12}) \cdot B_{22} \\
M_6 &= (A_{21} - A_{11}) \cdot (B_{11} + B_{12}) \\
M_7 &= (A_{12} - A_{22}) \cdot (B_{21} + B_{22})
\end{aligned}
$$

$$
\begin{aligned}
C_{11} &= M_1 + M_4 - M_5 + M_7 \\
C_{12} &= M_3 + M_5 \\
C_{21} &= M_2 + M_4 \\
C_{22} &= M_1 - M_2 + M_3 + M_6
\end{aligned}
$$

Strassen's factor matrices:

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

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

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

$\mathbf{U}, \mathbf{V}, \mathbf{W}$ matrices encode the algorithm

Strassen's algorithm

$$
\begin{aligned}
M_1 &= (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \\
M_2 &= (A_{21} + A_{22}) \cdot B_{11} \\
M_3 &= A_{11} \cdot (B_{12} - B_{22}) \\
M_4 &= A_{22} \cdot (B_{21} - B_{11}) \\
M_5 &= (A_{11} + A_{12}) \cdot B_{22} \\
M_6 &= (A_{21} - A_{11}) \cdot (B_{11} + B_{12}) \\
M_7 &= (A_{12} - A_{22}) \cdot (B_{21} + B_{22})
\end{aligned}
$$

$$
\begin{aligned}
C_{11} &= M_1 + M_4 - M_5 + M_7 \\
C_{12} &= M_3 + M_5 \\
C_{21} &= M_2 + M_4 \\
C_{22} &= M_1 - M_2 + M_3 + M_6
\end{aligned}
$$

Strassen's factor matrices:

		M_1	M_2	M_3	M_4	M_5	M_6	M_7
U	A_{11}	1		1		1	−1	
	A_{12}					1		1
	A_{21}		1				1	
	A_{22}	1	1		1			−1
V	B_{11}	1	1		−1		1	
	B_{12}			1			1	
	B_{21}				1			1
	B_{22}	1		−1		1		1
W	C_{11}	1			1	−1		1
	C_{21}		1		1			
	C_{12}			1		1		
	C_{22}	1	−1	1			1	

U, **V**, **W** matrices encode the algorithm

How do you solve it?

Problem: Find $\mathbf{U}$, $\mathbf{V}$, $\mathbf{W}$ such that $\mathcal{T} = \sum \mathbf{u}_r \circ \mathbf{v}_r \circ \mathbf{w}_r$

- the problem is NP-complete (for general $\mathcal{T}$)
- many combinatorial formulations of the problem
- efficient numerical methods can compute low-rank approximations
  - typical approach is "alternating least squares" (ALS)
  - pitfall: getting stuck at local minima > 0
  - pitfall: facing ill-conditioned linear least squares problems
  - pitfall: numerical solution is good only to machine precision
- we seek exact, discrete, and sparse solutions

Alternating least squares with regularization

Most successful scheme due to Smirnov [Smi13]

Repeat

1. $$\mathbf{U} = \arg\min_{\mathbf{U}} \left\|\mathbf{T}_{(U)} - \mathbf{U}(\mathbf{W} \odot \mathbf{V})^{\mathsf{T}}\right\|_F^2 + \lambda \left\|\mathbf{U} - \tilde{\mathbf{U}}\right\|_F^2$$

2. $$\mathbf{V} = \arg\min_{\mathbf{V}} \left\|\mathbf{T}_{(V)} - \mathbf{V}(\mathbf{W} \odot \mathbf{U})^{\mathsf{T}}\right\|_F^2 + \lambda \left\|\mathbf{V} - \tilde{\mathbf{V}}\right\|_F^2$$

3. $$\mathbf{W} = \arg\min_{\mathbf{W}} \left\|\mathbf{T}_{(W)} - \mathbf{W}(\mathbf{V} \odot \mathbf{U})^{\mathsf{T}}\right\|_F^2 + \lambda \left\|\mathbf{W} - \tilde{\mathbf{W}}\right\|_F^2$$

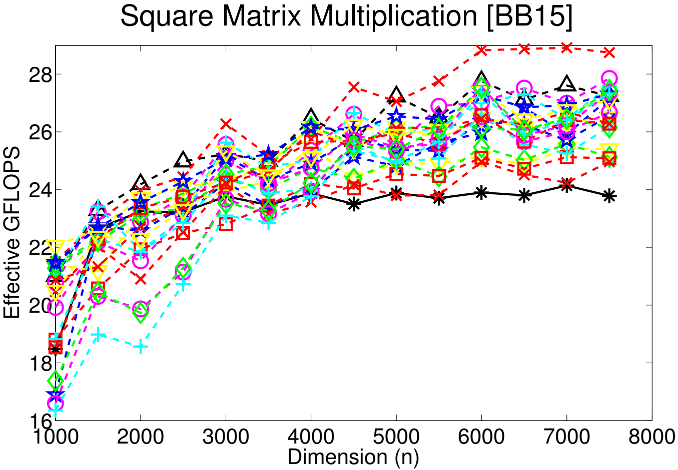
Until convergence

Art of optimization scheme in tinkering with λ, $\tilde{\mathbf{U}}$, $\tilde{\mathbf{V}}$, $\tilde{\mathbf{W}}$ (each iteration)

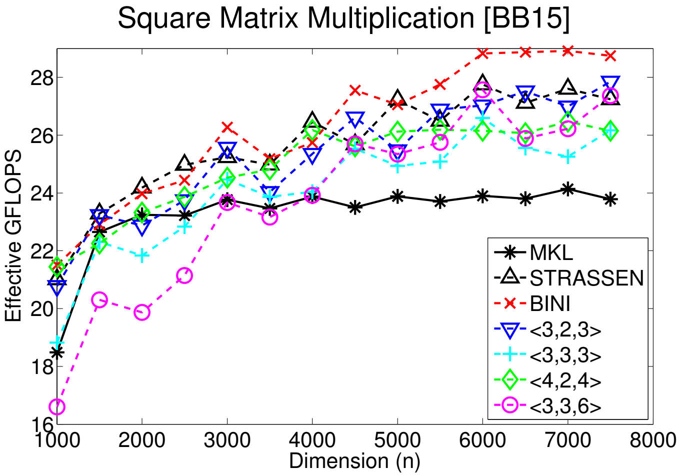
Discovered algorithms (a subset)

Algorithm base case	Multiplies (fast)	Multiplies (classical)	Speedup per recursive step	ω_0
$\langle 2,2,3\rangle$ [BB15]	11	12	9%	2.89
$\langle 2,2,5\rangle$ [BB15]	18	20	11%	2.89
$\langle 2,2,2\rangle$ [Str69]	7	8	14%	2.81
$\langle 2,2,4\rangle$ [BB15]	14	16	14%	2.85
$\langle 3,3,3\rangle$ [BB15]	23	27	17%	2.85
$\langle 2,3,3\rangle$ [BB15]	15	18	20%	2.81
$\langle 2,3,4\rangle$ [BB15]	20	24	20%	2.83
$\langle 2,4,4\rangle$ [BB15]	26	32	23%	2.82
$\langle 3,3,4\rangle$ [BB15]	29	36	24%	2.82
$\langle 3,4,4\rangle$ [Smi17]	38	48	26%	2.82
$\langle 3,3,6\rangle$ [Smi13]	40	54	35%	2.77
$\langle 2,2,3\rangle^*$ [BCRL79]	10	12	20%	2.78
$\langle 3,3,3\rangle^*$ [Sch81]	21	27	29%	2.77

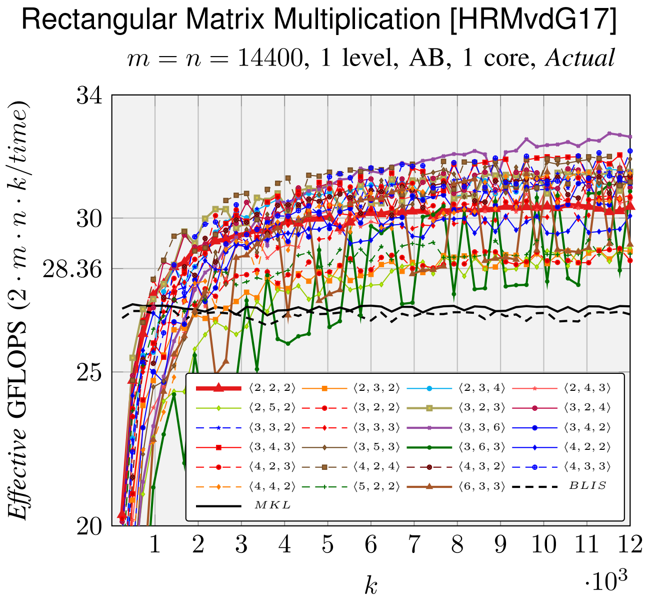
Are these algorithms faster in practice?



Are these algorithms faster in practice?



Are these algorithms faster in practice?



How big can we go?

- Current numerical techniques are hitting their limits
  - tensor size grows like N^6 if $M = P = N$
  - number of variables grows faster than $3N^4$ if $M = P = N$
- Nothing new has been found for $\langle 4, 4, 4 \rangle$
  - Strassen's $\langle 2, 2, 2 \rangle$ algorithm can be used twice
- Can we exploit properties particular to matrix multiplication?

Cyclic symmetry of square matrix multiplication

Let $\mathcal{M}$ be the matrix multiplication tensor for $M = P = N$

$\mathcal{M}$ has cyclic symmetry:

$$m_{ijk} = m_{kij} = m_{jki}$$

Cyclic symmetry of square matrix multiplication

Let $\mathcal{M}$ be the matrix multiplication tensor for $M = P = N$

$\mathcal{M}$ has cyclic symmetry:

$$m_{ijk} = m_{kij} = m_{jki}$$

This means if $\mathbf{U}, \mathbf{V}, \mathbf{W}$ is a solution,
then so are $\mathbf{W}, \mathbf{U}, \mathbf{V}$ and $\mathbf{V}, \mathbf{W}, \mathbf{U}$

Cyclic symmetry of square matrix multiplication

Let $\mathcal{M}$ be the matrix multiplication tensor for $M = P = N$

$\mathcal{M}$ has cyclic symmetry:

$$m_{ijk} = m_{kij} = m_{jki}$$

This means if $\mathbf{U}, \mathbf{V}, \mathbf{W}$ is a solution,
then so are $\mathbf{W}, \mathbf{U}, \mathbf{V}$ and $\mathbf{V}, \mathbf{W}, \mathbf{U}$

Is this property reflected in the low-rank decomposition?

$$\sum_r \mathbf{u}_r \circ \mathbf{v}_r \circ \mathbf{w}_r \equiv \sum_r \mathbf{w}_r \circ \mathbf{u}_r \circ \mathbf{v}_r \equiv \sum_r \mathbf{v}_r \circ \mathbf{w}_r \circ \mathbf{u}_r ?$$

Cyclic invariance of Strassen's algorithm

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

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

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

Cyclic invariance of Strassen's algorithm

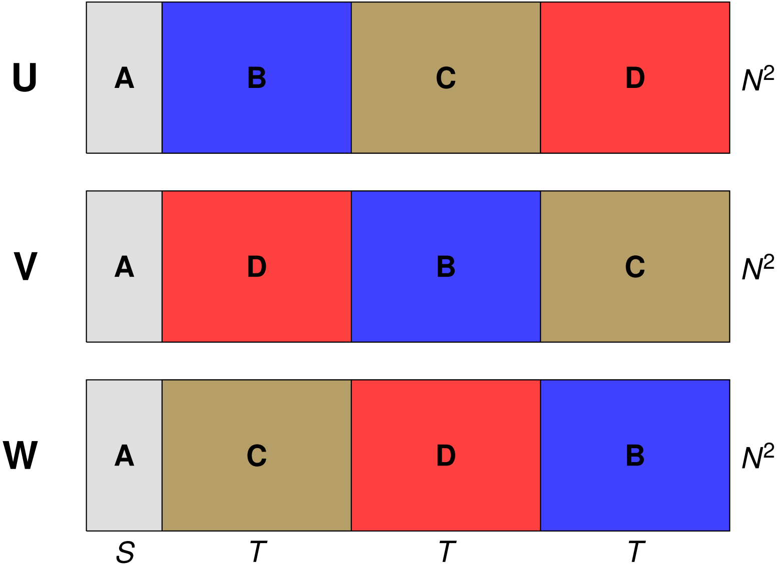
$$\mathbf{U} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & -1 \end{pmatrix}$$

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

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

If you cyclically permute **U**, **V**, **W**,
you get the same rank-one components in a different order

Searching for cyclic-invariant solutions



$$R = S + 3T$$

Cyclic-invariant decompositions

Decomposition with no constraints

$$\mathcal{T} = \sum_{r=1}^{R} \mathbf{u}_r \circ \mathbf{v}_r \circ \mathbf{w}_r$$

Decomposition with cyclic-invariant constraint

$$\mathcal{T} = \sum_{s=1}^{S} \mathbf{a}_s \circ \mathbf{a}_s \circ \mathbf{a}_s + \sum_{t=1}^{T} (\mathbf{b}_t \circ \mathbf{c}_t \circ \mathbf{d}_t + \mathbf{d}_t \circ \mathbf{b}_t \circ \mathbf{c}_t + \mathbf{c}_t \circ \mathbf{d}_t \circ \mathbf{b}_t)$$

Number of variables reduced by factor of 3,
but expression is no longer multilinear (not linear in **A**)

New rank-23 cyclic-invariant solutions for $\langle 3,3,3\rangle$

A					B						C						D					
	1										-1	-1		1		-1						
										-1	1	1		-1					-1		-1	
					1			-1			1	1	-1	-1					-1			
									1				1				-1			1		1
1													-1		-1		1	1		-1		
		-1		1								1	-1				1	1		-1		
							1		-1		-1	-1				-1				-1		
			1	-1	-1	1				-1			1		1		-1	-1		1		
		1	1	-1													-1	-1		1		

$$\mathbf{U} = \begin{pmatrix} \mathbf{A} & \mathbf{B} & \mathbf{C} & \mathbf{D} \end{pmatrix}$$

$$\mathbf{V} = \begin{pmatrix} \mathbf{A} & \mathbf{D} & \mathbf{B} & \mathbf{C} \end{pmatrix}$$

$$\mathbf{W} = \begin{pmatrix} \mathbf{A} & \mathbf{C} & \mathbf{D} & \mathbf{B} \end{pmatrix}$$

Rank-23 is the best known exact rank for $\langle 3,3,3\rangle$;
many previous solutions exist but none are cyclic invariant

We computed cyclic-invariant solutions with $S = 2, 5, 11$

What about $\langle 4, 4, 4 \rangle$?

- Performing two steps of Strassen's algorithm yields rank-49 cyclic-invariant solution

- No known exact decomposition of rank < 49
  - cyclic invariant or otherwise

What about $\langle 4, 4, 4 \rangle$?

- Performing two steps of Strassen's algorithm yields rank-49 cyclic-invariant solution
- No known exact decomposition of rank < 49
  - cyclic invariant or otherwise
- No success yet in computing cyclic-invariant solutions
  - but the truth is out there

Summary

- Discovering fast matrix multiplication algorithms corresponds to computing an exact CP decomposition
- Any new algorithm can be implemented efficiently
- Exploiting symmetry of matrix multiplication can reduce the size of the problem, want to tackle $\langle 4, 4, 4 \rangle$
- Still exploring how to use more structure to scale up to larger dimensions

Thank You!

Discovering Fast Matrix Multiplication Algorithms via Tensor Decomposition

Grey Ballard

WAKE FOREST UNIVERSITY

ballard@wfu.edu

More structure...

Transposition symmetry

$$t_{jik} = t'_{ijk}$$

where

$$\mathcal{T}' = \mathcal{T} \times_1 \mathbf{P} \times_2 \mathbf{P} \times_3 \mathbf{P}$$

and **P** is the "vec-permutation" matrix

Transposition symmetry

$$t_{jik} = t'_{ijk}$$

where

$$\mathcal{T}' = \mathcal{T} \times_1 \mathbf{P} \times_2 \mathbf{P} \times_3 \mathbf{P}$$

and **P** is the “vec-permutation” matrix

This is derived from the fact that

$$\mathbf{AB} = \mathbf{C}$$

implies

$$\mathbf{B}^T\mathbf{A}^T = \mathbf{C}^T$$

More structure...

Multiplication invariance

$$\mathfrak{T} = \mathfrak{T} \times_1 \left(\mathbf{Y}^{-T} \otimes \mathbf{X}\right) \times_2 \left(\mathbf{Z}^{-T} \otimes \mathbf{Y}\right) \times_3 \left(\mathbf{Z}^{-T} \otimes \mathbf{X}\right)$$

where $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ are nonsingular matrices

Multiplication invariance

$$\mathfrak{T} = \mathfrak{T} \times_1 \left(\mathbf{Y}^{-T} \otimes \mathbf{X}\right) \times_2 \left(\mathbf{Z}^{-T} \otimes \mathbf{Y}\right) \times_3 \left(\mathbf{Z}^{-T} \otimes \mathbf{X}\right)$$

where $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ are nonsingular matrices

This is derived from the fact that

$$\mathbf{AB} = \mathbf{C}$$

implies

$$\left(\mathbf{XAY}^{-1}\right)\left(\mathbf{YBZ}^{-1}\right) = \left(\mathbf{XCZ}^{-1}\right)$$

Example algorithm: $\langle 4, 2, 4 \rangle$

Partition matrices like this:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \\ A_{41} & A_{42} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} & B_{13} & B_{14} \\ B_{21} & B_{22} & B_{23} & B_{24} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \\ C_{41} & C_{42} & C_{43} & C_{44} \end{bmatrix}$$

1. Take 26 linear combos of A_{ij}'s according to **U** (68 adds)
2. Take 26 linear combos of B_{ij}'s according to **V** (52 adds)
3. Perform 26 multiplies (recursively)
4. Take linear combos of outputs to form C_{ij}'s acc. to **W** (69 adds)

Classical algorithm performs 32 multiplies yielding a possible speedup of 23% per step

References I

Austin R. Benson and Grey Ballard.
A framework for practical parallel fast matrix multiplication.
In Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2015, pages 42–53, New York, NY, USA, February 2015. ACM.

D. Bini, M. Capovani, F. Romani, and G. Lotti.
$O(n^{2.7799})$ complexity for $n \times n$ approximate matrix multiplication.
Information Processing Letters, 8(5):234–235, 1979.

D. Coppersmith and S. Winograd.
Matrix multiplication via arithmetic progressions.
In Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, STOC '87, pages 1–6, New York, NY, USA, 1987. ACM.

Jianyu Huang, Leslie Rice, Devin A. Matthews, and Robert van de Geijn.
Generating families of practical fast matrix multiplication algorithms.
In Proceedings of the 31st IEEE International Parallel and Distributed Processing Symposium, 2017. To appear.

François Le Gall.
Powers of tensors and fast matrix multiplication.
In Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation, ISSAC '14, pages 296–303, New York, NY, USA, 2014. ACM.

A. Schönhage.
Partial and total matrix multiplication.
SIAM Journal on Computing, 10(3):434–455, 1981.

A.V. Smirnov.
The bilinear complexity and practical algorithms for matrix multiplication.
Computational Mathematics and Mathematical Physics, 53(12):1781–1795, 2013.

Alexey Smirnov.
Several bilinear algorithms for matrix multiplication.
Technical report, ResearchGate, 2017.

V. Strassen.
Gaussian elimination is not optimal.
Numerische Mathematik, 13:354–356, 1969.
10.1007/BF02165411.