How Practical is Fast Matrix Multiplication?

Grey Ballard

based on joint work with Austin Benson

December 10, 2014

Sandia National Laboratories

U.S. DEPARTMENT OF ENERGY

NNSA National Nuclear Security Administration

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

(SAND2014-18305 PE)

Outline

Strassen's algorithm (1969)

Strassen showed how to use 7 scalar multiplies for 2×2 matrix multiplication

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

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 algorithm (1969)

Strassen showed how to use 7 scalar multiplies for 2×2 matrix 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

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

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

Strassen's algorithm (1969)

Strassen showed how to use 7 scalar multiplies for 2×2 matrix 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

Flop count recurrence:

$$F(n) = 7 \cdot F(n/2) + O(n^2)$$
$$F(1) = 1$$
$$F(n) = O\left(n^{\log_2 7}\right)$$

$$\log_2 7 \approx 2.81$$

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 algorithm (1969)

Strassen showed how to use 7 scalar multiplies for 2×2 matrix 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

Word count recurrence:

$$W(n) = 7 \cdot W(n/2) + O(n^2)$$
$$W(\sqrt{M}) = O(M)$$
$$W(n) = O\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$$

(M is the cache size in words)

Strassen's Algorithm

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

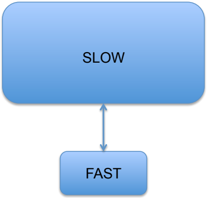
$$\begin{aligned}
C_{11} &= M_1 + M_4 - M_5 + M_7 \\
C_{12} &= M_3 + M_5 \\
C_{21} &= M_2 + M_4 \\
C_{22} &= M_1 - M_2 + M_3 + M_6
\end{aligned}$$

Memory models for communication costs

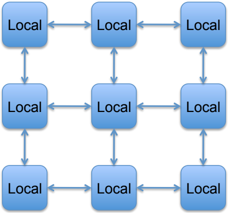
Algorithms have two kinds of costs: computation and *communication*

- moving data within memory hierarchy on a sequential computer
- moving data between processors on a parallel computer

For high-level analysis, we use these memory models:



Sequential



Parallel

Runtime Model

Strassen's algorithm (1969)

Strassen showed how to use 7 scalar multiplies for 2×2 matrix 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

Word count recurrence:

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

$$W(\sqrt{M}) = O(M)$$

$$W(n) = O\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$$

(M is the cache size in words)

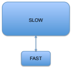
Strassen's Algorithm

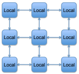
$$\begin{aligned}
M_1 &= (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \\
M_2 &= (A_{21} + A_{22}) \cdot B_{11} \\
M_3 &= A_{11} \cdot (B_{12} - B_{22}) \\
M_4 &= A_{22} \cdot (B_{21} - B_{11}) \\
M_5 &= (A_{11} + A_{12}) \cdot B_{22} \\
M_6 &= (A_{21} - A_{11}) \cdot (B_{11} + B_{12}) \\
M_7 &= (A_{12}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

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

Communication costs of matrix multiplication

	Classical	**Strassen's**
Seq	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} M\right)$	$O\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$
Par		





Units $=$ (max) words communicated

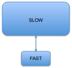
O $=$ algorithm exists, Ω $=$ lower bound exists, Θ $=$ both exist

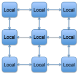
n $=$ matrix dimension, M $=$ fast/local memory size, P $=$ number of processors

References:

Communication costs of matrix multiplication

	Classical	**Strassen's**
Seq	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} M\right)$	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$
Par		





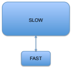
Units = (max) words communicated

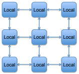
O = algorithm exists, Ω = lower bound exists, Θ = both exist

n = matrix dimension, M = fast/local memory size, P = number of processors

References: [BDHS11],

	Classical	Strassen's
Seq	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} M\right)$	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$
Par		$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \frac{M}{P}\right)$





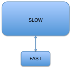
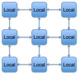
Units = (max) words communicated

O = algorithm exists, Ω = lower bound exists, Θ = both exist

n = matrix dimension, M = fast/local memory size, P = number of processors

References: [BDHS11],

Communication costs of matrix multiplication

	Classical	**Strassen's**
Seq 	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} M\right)$	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$
Par 	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} \frac{M}{P}\right)$	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \frac{M}{P}\right)$

Units = (max) words communicated

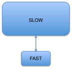
O = algorithm exists, Ω = lower bound exists, Θ = both exist

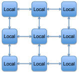
n = matrix dimension, M = fast/local memory size, P = number of processors

References: [BDHS11],

Communication costs of matrix multiplication

	Classical	**Strassen's**
Seq	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} M\right)$	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$
Par	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} \frac{M}{P}\right)$	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \frac{M}{P}\right)$





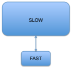
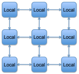
Units = (max) words communicated

O = algorithm exists, Ω = lower bound exists, Θ = both exist

n = matrix dimension, M = fast/local memory size, P = number of processors

References: [BDHS11], [BDH$^+$12a],

Communication costs of matrix multiplication

	Classical	**Strassen's**
Seq 	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} M\right)$	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$
Par 	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} \frac{M}{P}\right)$	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \frac{M}{P}\right) + O\left(\frac{n^2}{P^{2/\log_2 7}}\right)$

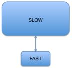
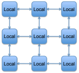
Units = (max) words communicated

O = algorithm exists, Ω = lower bound exists, Θ = both exist

n = matrix dimension, M = fast/local memory size, P = number of processors

References: [BDHS11], [BDH+12a],

Communication costs of matrix multiplication

	Classical	**Strassen's**
Seq 	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} M\right)$	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$
Par 	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} \frac{M}{P}\right) + \Theta\left(\frac{n^2}{P^{2/\log_2 8}}\right)$	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \frac{M}{P}\right) + O\left(\frac{n^2}{P^{2/\log_2 7}}\right)$

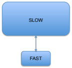
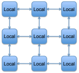
Units = (max) words communicated

O = algorithm exists, Ω = lower bound exists, Θ = both exist

n = matrix dimension, M = fast/local memory size, P = number of processors

References: [BDHS11], [BDH$^+$12a],

Communication costs of matrix multiplication

	Classical	**Strassen's**
Seq 	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} M\right)$	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$
Par 	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} \frac{M}{P}\right) + \Theta\left(\frac{n^2}{P^{2/\log_2 8}}\right)$	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \frac{M}{P}\right) + \Theta\left(\frac{n^2}{P^{2/\log_2 7}}\right)$

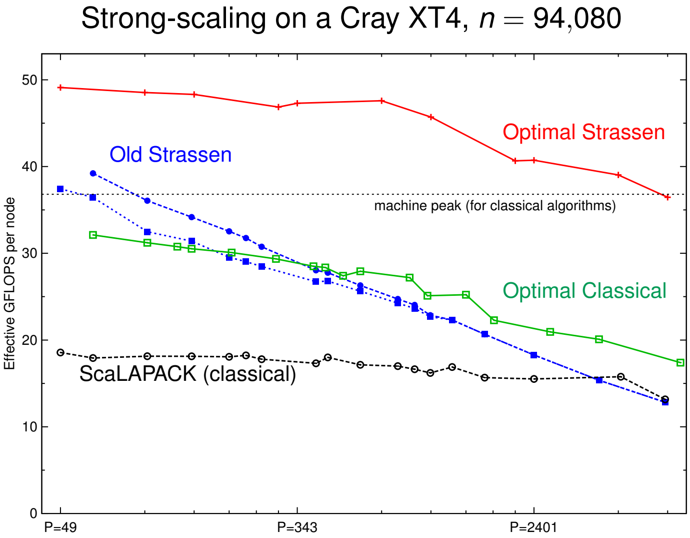
Units = (max) words communicated

O = algorithm exists, Ω = lower bound exists, Θ = both exist

n = matrix dimension, M = fast/local memory size, P = number of processors

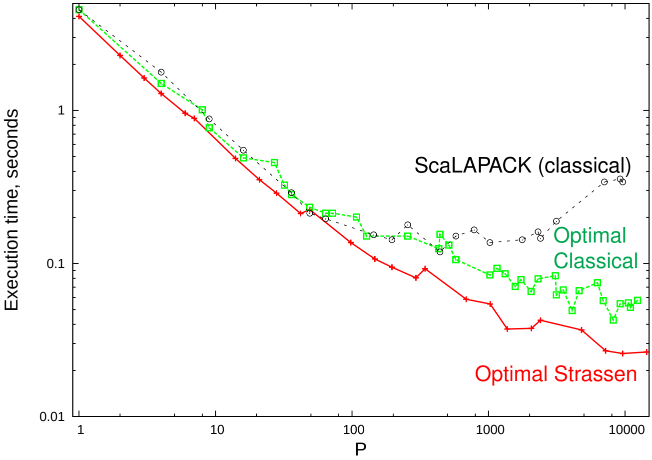
References: [BDHS11], [BDH$^+$12a], [BDH$^+$12b],

Performance of optimal algorithms on large problem

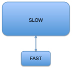
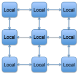


Execution time of optimal algorithms on small problem

Strong-scaling on a Cray XE6, $n = 4704$



Communication costs of matrix multiplication

	Classical	**Strassen's**
Seq 	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} M\right)$	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$
Par 	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} \frac{M}{P}\right) + \Theta\left(\frac{n^2}{P^{2/\log_2 8}}\right)$	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \frac{M}{P}\right) + \Theta\left(\frac{n^2}{P^{2/\log_2 7}}\right)$

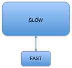
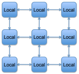
Units $=$ (max) words communicated

O $=$ algorithm exists, Ω $=$ lower bound exists, Θ $=$ both exist

n $=$ matrix dimension, M $=$ fast/local memory size, P $=$ number of processors

References: [BDHS11], [BDH+12a], [BDH+12b],

Communication costs of matrix multiplication

	Classical	**Strassen's**	**Fast**: $\Theta(n^{\omega_0})$ flops
Seq 	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} M\right)$	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\omega_0} M\right)$
Par 	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} \frac{M}{P}\right) + \Theta\left(\frac{n^2}{P^{2/\log_2 8}}\right)$	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \frac{M}{P}\right) + \Theta\left(\frac{n^2}{P^{2/\log_2 7}}\right)$	$\Theta\left(\left(\frac{n}{\sqrt{M}}\right)^{\omega_0} \frac{M}{P}\right) + \Theta\left(\frac{n^2}{P^{2/\omega_0}}\right)$

Units $=$ (max) words communicated

O $=$ algorithm exists, Ω $=$ lower bound exists, Θ $=$ both exist

n $=$ matrix dimension, M $=$ fast/local memory size, P $=$ number of processors

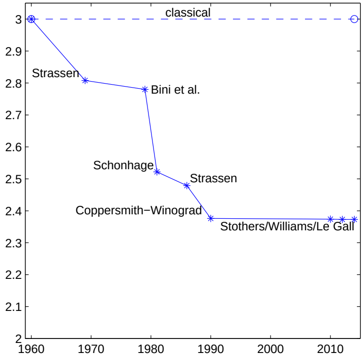
References: [BDHS11], [BDH$^+$12a], [BDH$^+$12b], [BDHS12, BDHS14]

How small can ω_0 get?

People have worked on this problem for decades!

- "fast" algorithms multiply matrices using $O(n^{\omega_0})$ flops, $\omega_0 < 3$

Exponent over time

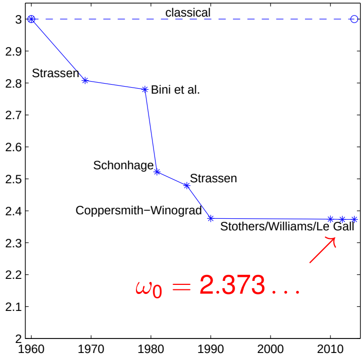


How small can ω_0 get?

People have worked on this problem for decades!

- "fast" algorithms multiply matrices using $O(n^{\omega_0})$ flops, $\omega_0 < 3$

Exponent over time



How small can ω_0 get?

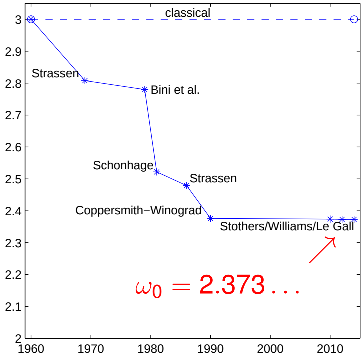
People have worked on this problem for decades!

- "fast" algorithms multiply matrices using $O(n^{\omega_0})$ flops, $\omega_0 < 3$

Most fast algorithms are only theoretical because they

- involve approximations
  - $A \cdot B = C + \lambda E$
- are not explicit
  - only proofs of existence
- have (possibly) large constants or log factors
  - most theoreticians care about only the exponent ω in $O(n^{\omega+\epsilon})$

Exponent over time



Practical Fast Algorithms

- Strassen's algorithm *is* practical
- Many algorithms are better in theory, are any better in practice?
- Can we find practical algorithms that have been overlooked?
- Can we implement and benchmark all known algorithms?

Fast algorithms are based on recursion

Strassen showed how to use 7 multiplies instead of 8 for 2×2 multiplication

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

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

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

Recursion allows us to focus on base case

$2 \times 2 \times 2$

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

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

$2 \times 2 \times 2$

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

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

Recursion allows us to focus on base case

$2 \times 2 \times 2$

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

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

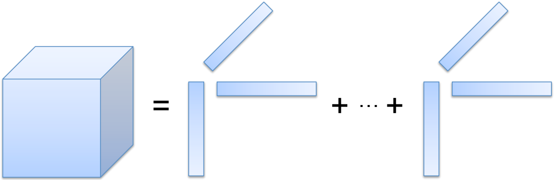
$3 \times 3 \times 3$

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

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 better base case corresponds to computing a low-rank decomposition of a particular 3D tensor



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

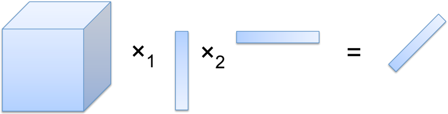
This is the main problem to solve

- various ways to attack it, but basically a search problem
- as base case gets bigger, tensor dimensions and rank get bigger

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

is equivalent to

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



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

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

▸ Operation with low-rank decomposition

Low-rank decomposition for Strassen

$$\mathfrak{T} = \sum_{r=1}^{7} \mathbf{u}_r \circ \mathbf{v}_r \circ \mathbf{w}_r$$

Strassen's decomposition is represented by these 3 factor matrices:

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

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

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

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:

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

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

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

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

Classical Algorithm

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:

		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_{12}			1		1		
	C_{21}		1		1			
	C_{22}	1	−1	1			1	

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

▸ Classical Algorithm

Main search problem

Given base case dimensions M, P, and N (multiplying $M \times P$ and $P \times N$ matrices), the tensor $\mathfrak{T} \in \{0, 1\}^{MP \times PN \times MN}$ is specified.

Main search problem

Given base case dimensions M, P, and N (multiplying $M \times P$ and $P \times N$ matrices), the tensor $\mathfrak{T} \in \{0,1\}^{MP \times PN \times MN}$ is specified.

Then for some desired rank $R < MNP$, find

$$\mathbf{U} \in \mathbb{F}^{MP \times R}, \ \mathbf{V} \in \mathbb{F}^{PN \times R}, \ \mathbf{W} \in \mathbb{F}^{MN \times R}$$

such that

$$t_{ijk} = \sum_{r=1}^{R} u_{ir} v_{jr} w_{kr} \quad \text{for all} \quad i, j, k$$

(these $(MNP)^2$ scalar constraints are equivalent to $\mathfrak{T} = \sum \mathbf{u}_r \circ \mathbf{v}_r \circ \mathbf{w}_r$)

Main search problem

Given base case dimensions M, P, and N (multiplying $M \times P$ and $P \times N$ matrices), the tensor $\mathfrak{T} \in \{0, 1\}^{MP \times PN \times MN}$ is specified.

Then for some desired rank $R < MNP$, find

$$\mathbf{U} \in \mathbb{F}^{MP \times R}, \ \mathbf{V} \in \mathbb{F}^{PN \times R}, \ \mathbf{W} \in \mathbb{F}^{MN \times R}$$

such that

$$t_{ijk} = \sum_{r=1}^{R} u_{ir} v_{jr} w_{kr} \quad \text{for all} \quad i, j, k$$

(these $(MNP)^2$ scalar constraints are equivalent to $\mathfrak{T} = \sum \mathbf{u}_r \circ \mathbf{v}_r \circ \mathbf{w}_r$)

- solution corresponds to algorithm with $\omega_0 = 3 \log_{MPN} R$

How do you solve it?

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

- the problem is NP-complete (for general $\mathfrak{T}$)
- 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

Algorithm base case	Multiplies (fast)	Multiplies (classical)	Speedup per recursive step	ω_0
$\langle 2,2,3 \rangle$	11	12	9%	2.89
$\langle 2,2,5 \rangle$	18	20	11%	2.89
$\langle 2,2,2 \rangle$ [Str69]	7	8	14%	2.81
$\langle 2,2,4 \rangle$	14	16	14%	2.85
$\langle 3,3,3 \rangle$	23	27	17%	2.85
$\langle 2,3,3 \rangle$	15	18	20%	2.81
$\langle 2,3,4 \rangle$	20	24	20%	2.83
$\langle 2,4,4 \rangle$	26	32	23%	2.82
$\langle 3,3,4 \rangle$	29	36	24%	2.82
$\langle 3,4,4 \rangle$	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

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

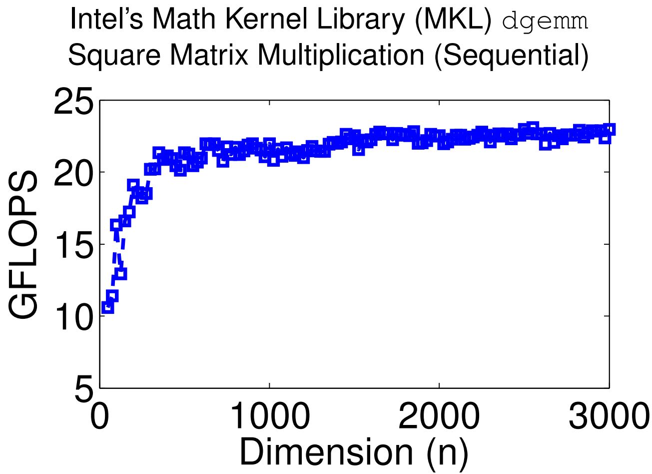
Discovered algorithms

Algorithm base case	Multiplies (fast)	Multiplies (classical)	Speedup per recursive step	ω_0
$\langle 2,2,3 \rangle$	11	12	9%	2.89
$\langle 2,2,5 \rangle$	18	20	11%	2.89
$\langle 2,2,2 \rangle$ [Str69]	7	8	14%	2.81
$\langle 2,2,4 \rangle$	14	16	14%	2.85
$\langle 3,3,3 \rangle$	23	27	17%	2.85
$\langle 2,3,3 \rangle$	15	18	20%	2.81
$\langle 2,3,4 \rangle$	20	24	20%	2.83
$\langle 2,4,4 \rangle$	26	32	23%	2.82
$\langle 3,3,4 \rangle$	29	36	24%	2.82
$\langle 3,4,4 \rangle$	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

How do these algorithms perform in practice?

- All these algorithms have the same structure:
  - perform additions according to **U**, **V**, **W**, and make recursive calls
- Code generator can translate **U**, **V**, **W** into an implementation
- Sequential performance is based on:
  - classical multiplication implementation performance (vendor library)
  - efficiency of additions
  - crossover point of fast to classical
- Parallel performance depends also on parallelization approach
  - and hardware parameters

Classical performance



- shape of `dgemm` curve gives rule of thumb for crossover point

Performing (and optimizing) additions

Additions are completely memory bandwidth bound

- time is proportional to communication (flops are free)

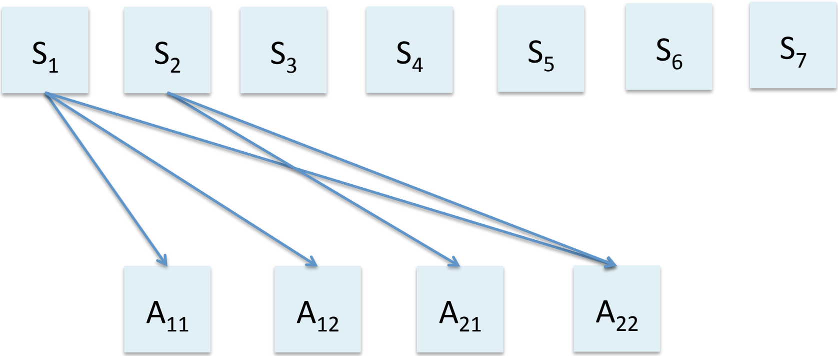
We micro-benchmarked three approaches:

1. Pairwise: most straightforward
2. Streaming: minimizes communication (in theory)
3. Write-once: best performance

We also considered common subexpression elimination

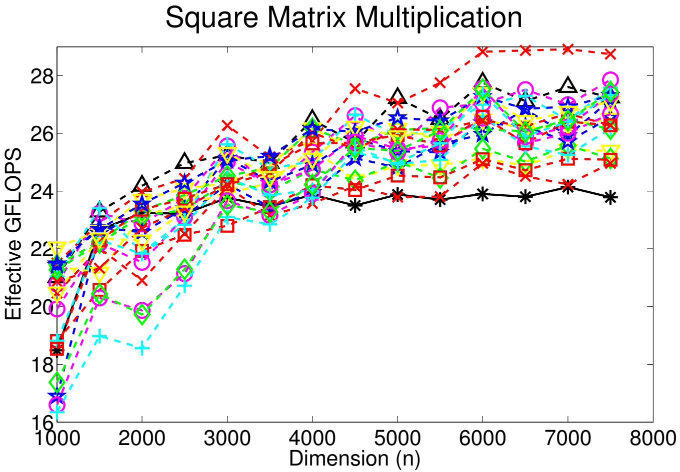
1. can help pairwise and streaming approaches
2. often hurts write-once approach

Write-once approach to additions

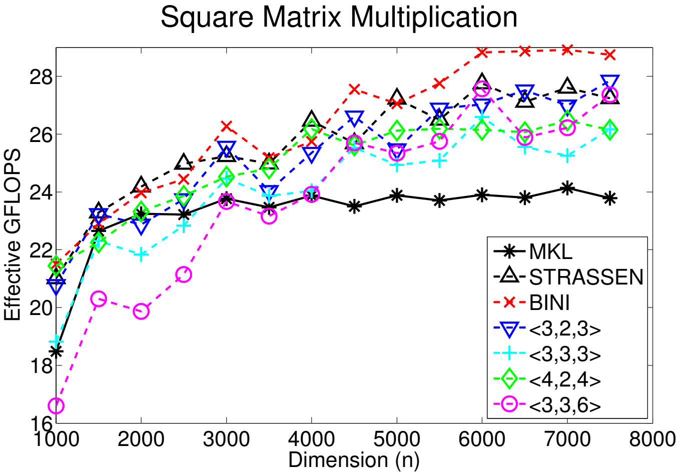


$$
\begin{aligned}
S_1 &= A_{11} - A_{12} \qquad\qquad + A_{22} \\
S_2 &= \qquad\qquad\qquad\quad A_{21} - A_{22} \\
&\vdots
\end{aligned}
$$

Sequential performance of fast algorithms

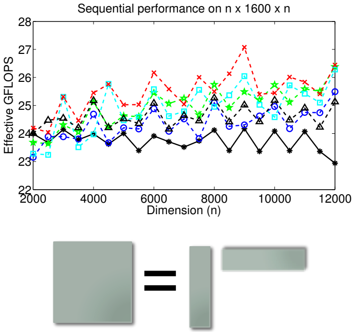


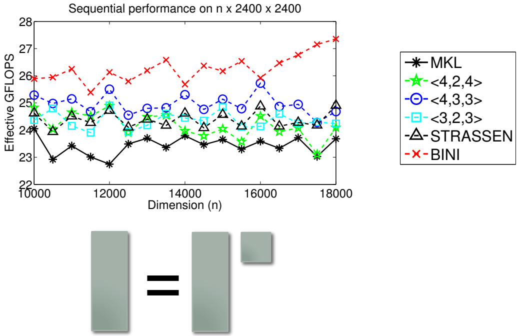
Sequential performance of fast algorithms



Sequential performance of fast algorithms

Rectangular Matrix Multiplication

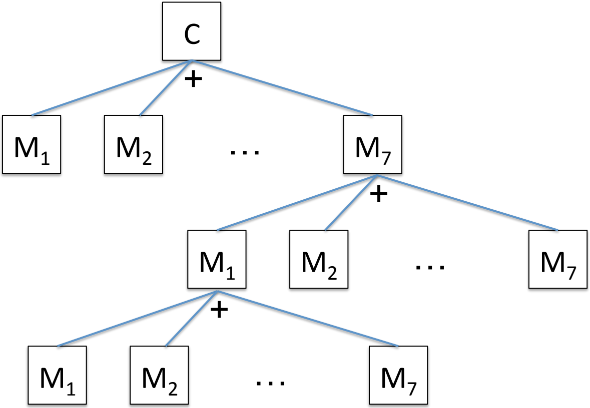




- best algorithms match “shape” of problem

Parallelization schemes: recursion tree traversal

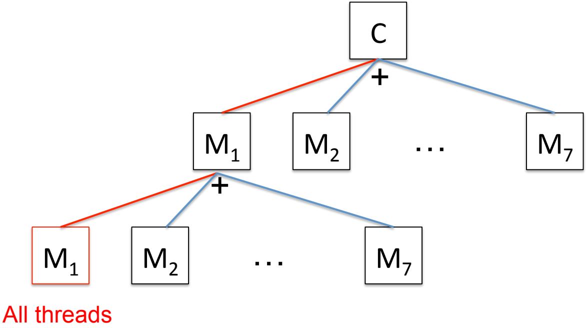
We consider 3 methods for shared-memory parallelization,
based on traversing recursion tree



Parallelization schemes: recursion tree traversal

DFS: depth first search is simplest scheme

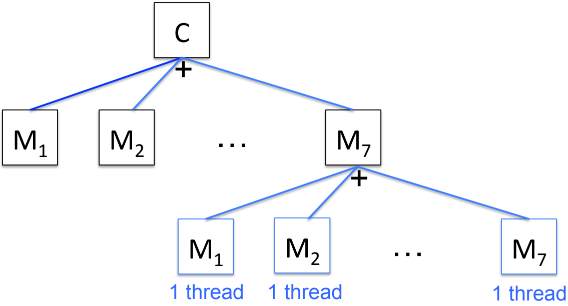
- all parallelism in calls to `dgemm`, always load balanced
- requires large subproblems for high performance



Parallelization schemes: recursion tree traversal

BFS: breadth first search relies on sequential `dgemm`

- maintains high performance for small subproblems
- load balancing of multiplies is no longer guaranteed

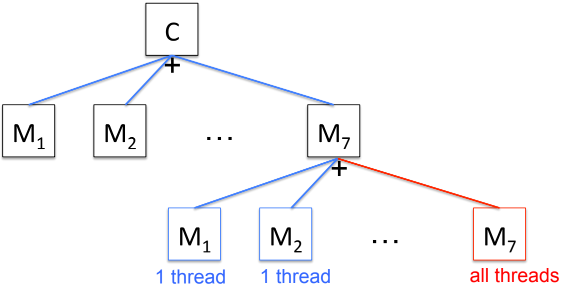


- 2 steps of Strassen creates 49 subproblems; we have 24 cores

Parallelization schemes: recursion tree traversal

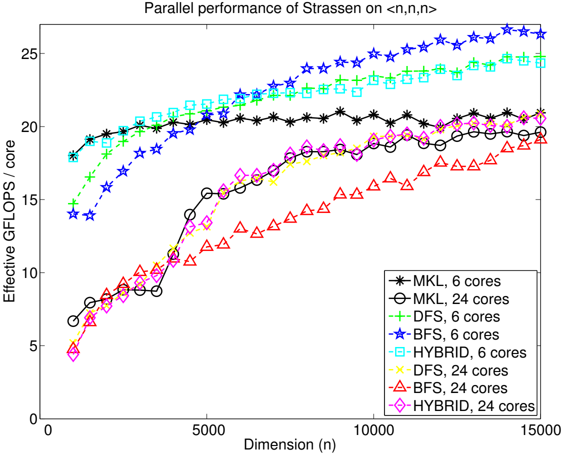
HYBRID

- use BFS as much as possible
- use DFS to load balance leftovers

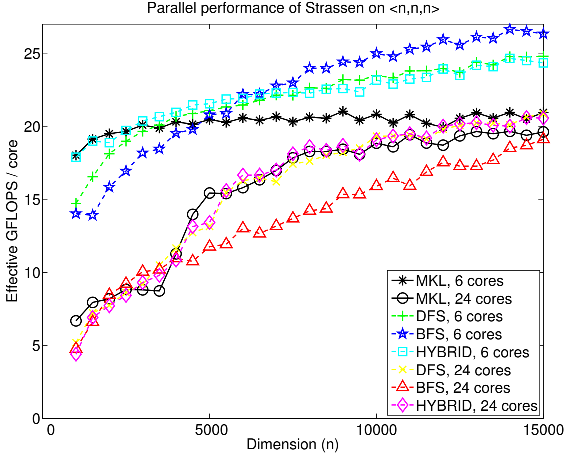


- 2 steps of Strassen creates 49 subproblems; we have 24 cores

Parallel performance of fast algorithms (square case)

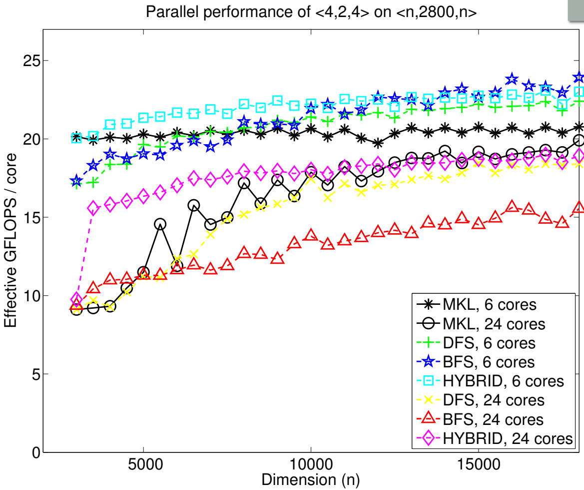


Parallel performance of fast algorithms (square case)

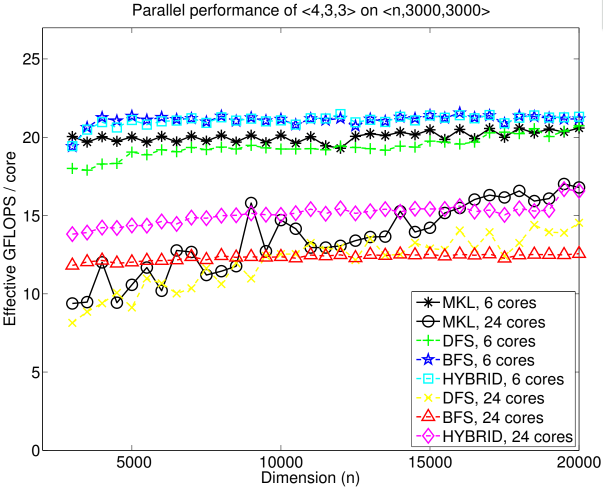


- at 24 threads, not only are the additions bandwidth bound, but they don't scale as well as the multiplies (bandwidth scaling is $< 6\times$)

Parallel performance of fast algorithms (rect case)



Parallel performance of fast algorithms (rect case)



Conclusions

- in theory, fast algorithms reduce both computation and communication
- in practice, fast algorithms like Strassen's can outperform `dgemm`
- for square matrices, Strassen's algorithm is hard to beat
- for rectangular matrices, algorithm should match the shape
- shared-memory parallelization faces bandwidth bottleneck

Open questions

1. What are the numerical properties of all these algorithms?
2. How will they perform on distributed-memory parallel architectures?
3. Are there applications that can benefit from approximate algorithms?
4. Have we exhausted the possibilities of practical fast algorithms?
5. Can we use fast algorithms in the context of linear algebra and other applications?

How Practical is Fast Matrix Multiplication?

Grey Ballard

For more details, see tech report:
http://arxiv.org/abs/1409.2908

gmballa@sandia.gov
www.sandia.gov/~gmballa

Extra slides

Runtime Model

Measure computation in terms of # *flops* performed

Measure communication in terms of # *words* communicated

Time per flop: γ

Time per word: β

Total running time of an algorithm (ignoring overlap):

$$\gamma \cdot (\text{\# flops}) + \beta \cdot (\text{\# words})$$

$\beta \gg \gamma$ as measured in time *and* energy, and the relative cost of communication is increasing

Extras Back

Matmul-as-tensor-operation 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 new matrix multiplication algorithm:

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

Extras Back

Connection between factor matrices and algorithm

Classical algorithm:

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

Classical factor matrices:

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

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

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

Factor matrices for an approximate algorithm (Bini's)

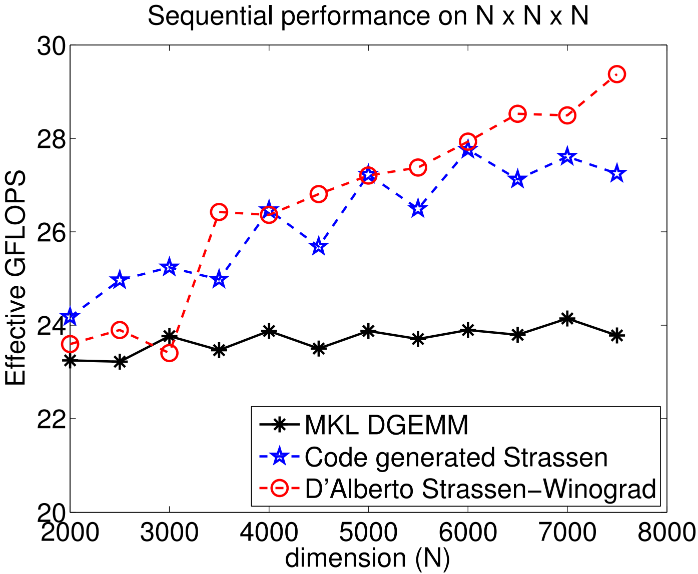
$$\mathbf{U} = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \lambda & \lambda & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda & \lambda \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

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

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

Extras

Code generated vs tuned performance



Extras

Tuned code: [DBN11]

References I

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.

G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz.
Brief announcement: strong scaling of matrix multiplication algorithms and memory-independent communication lower bounds.
In *Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '12, pages 77–79, New York, NY, USA, 2012. ACM.

G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz.
Communication-optimal parallel algorithm for Strassen's matrix multiplication.
In *Proceedings of the 24th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '12, pages 193–204, New York, NY, USA, 2012. ACM.

G. Ballard, J. Demmel, O. Holtz, and O. Schwartz.
Graph expansion and communication costs of fast matrix multiplication.
In *Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '11, pages 1–12. ACM, 2011.

G. Ballard, J. Demmel, O. Holtz, and O. Schwartz.
Graph expansion and communication costs of fast matrix multiplication.
Journal of the ACM, 59(6):32:1–32:23, December 2012.

Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz.
Communication costs of strassen's matrix multiplication.
Commun. ACM, 57(2):107–114, February 2014.

References II

Paolo D'Alberto, Marco Bodrato, and Alexandru Nicolau.
Exploiting parallelism in matrix-computation kernels for symmetric multiprocessor systems: Matrix-multiplication and matrix-addition algorithm optimizations by software pipelining and threads allocation.
ACM Trans. Math. Softw., 38(1):2:1–2:30, December 2011.

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.

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