Practicality of Large Scale Fast Matrix Multiplication

Grey Ballard, James Demmel, Olga Holtz,
Benjamin Lipshitz and Oded Schwartz

UC Berkeley

IWASEP
June 5, 2012
Napa Valley, CA

Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from Par Lab affiliates National Instruments, NEC, Nokia, NVIDIA, and Samsung. Research is also supported by DOE grants DE-SC0003959, DE-SC0004938, and DE-AC02-05CH11231; the Sofja Kovalevskaja programme of Alexander von Humboldt Foundation; and by the National Science Foundation under agreement DMS-0635607.

Introduction

- Classical matrix multiplication is nearly ubiquitous, even though asymptotically faster algorithms have been know since 1969
- Concerns about fast matrix multiplication:
  - Practical speed
  - Stability
- This talk addresses both concerns

Outline

- Strassen's algorithm
- New parallel algorithm
  - Communication optimal
  - Faster in practice
- Stability of Strassen
  - Normwise error bound
  - Diagonal scaling, improved error bounds
  - Stability experiments

Recall: Strassen's fast matrix multiplication

Strassen's original algorithm uses 7 multiplies and 18 adds for $n = 2$.
It is applied recursively (blockwise).

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

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

$$
\begin{array}{r} n/2\,\{ \\ n/2\,\{ \end{array}
\begin{array}{|c|c|}\hline C_{11} & C_{12} \\ \hline C_{21} & C_{22} \\ \hline \end{array}
=
\begin{array}{|c|c|}\hline A_{11} & A_{12} \\ \hline A_{21} & A_{22} \\ \hline \end{array}
\bullet
\begin{array}{|c|c|}\hline B_{11} & B_{12} \\ \hline B_{21} & B_{22} \\ \hline \end{array}
$$

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

$$T(n) = \Theta\left(n^{\log_2 7}\right)$$

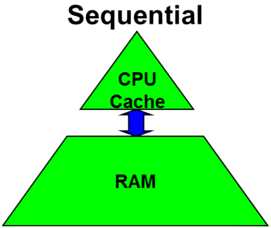
Improved by Winograd to 15 additions

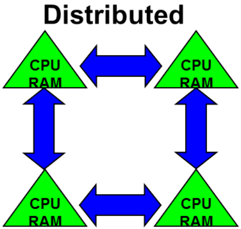
Communication costs

Two kinds of costs:

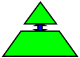
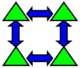
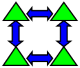
- Arithmetic (FLOPs)
- Communication: moving data between
  - levels of a memory hierarchy (sequential case)
  - over a network connecting processors (parallel case)

Communication is becoming more expensive relative to computation

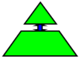
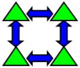
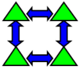




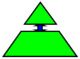
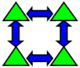
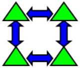
Communication lower bounds for matrix multiplication

	Strassen:	Classic (cubic):
 Sequential	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} M\right)$
 Distributed	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \frac{M}{P}\right)$	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} \frac{M}{P}\right)$
 Distributed	$\Omega\left(\frac{n^2}{P^{2/\log_2 7}}\right)$	$\Omega\left(\frac{n^2}{P^{2/\log_2 8}}\right)$

Algorithms attaining these bounds?

	Strassen:	Classic (cubic):
 Sequential	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} M\right)$
 Distributed	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \frac{M}{P}\right)$	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} \frac{M}{P}\right)$
 Distributed	$\Omega\left(\frac{n^2}{P^{2/\log_2 7}}\right)$	$\Omega\left(\frac{n^2}{P^{2/\log_2 8}}\right)$

Algorithms attaining these bounds?

	Strassen:	Classic (cubic):
 Sequential	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} M\right)$
 Distributed	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \frac{M}{P}\right)$	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} \frac{M}{P}\right)$
 Distributed	$\Omega\left(\frac{n^2}{P^{2/\log_2 7}}\right)$	$\Omega\left(\frac{n^2}{P^{2/\log_2 8}}\right)$

Our new algorithm

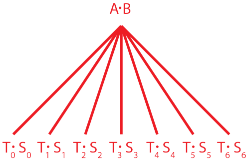
Lessons from lower bounds

- Don't use a classical algorithm for the communication
  - Strassen can communicate less than classical
- Make local multiplies as large as possible
- Use all available memory, up to $O(n^2/P^{2/\log_2 7})$
  - Communication bound decreases with increased memory
- Send memory size messages to minimize latency

Main Idea of CAPS algorithm

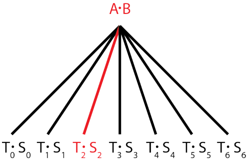
At each level of recursion tree, choose either breadth-first or depth-first traversal of the recursion tree

Breadth-First-Search (BFS)



- Runs all 7 multiplies in parallel
  - each uses $P/7$ processors
- Requires $7/4$ as much extra memory
- Requires communication, but
- All BFS minimizes communication if possible

Depth-First-Search (DFS)

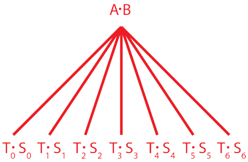


- Runs all 7 multiplies sequentially
  - each uses all P processors
- Requires $1/4$ as much extra memory
- No immediate communication
- Increases bandwidth by factor of $7/4$
- Increases latency by factor of 7

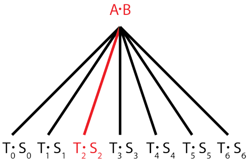
Main Idea of CAPS algorithm

At each level of recursion tree, choose either breadth-first or depth-first traversal of the recursion tree

Breadth-First-Search (BFS)



Depth-First-Search (DFS)



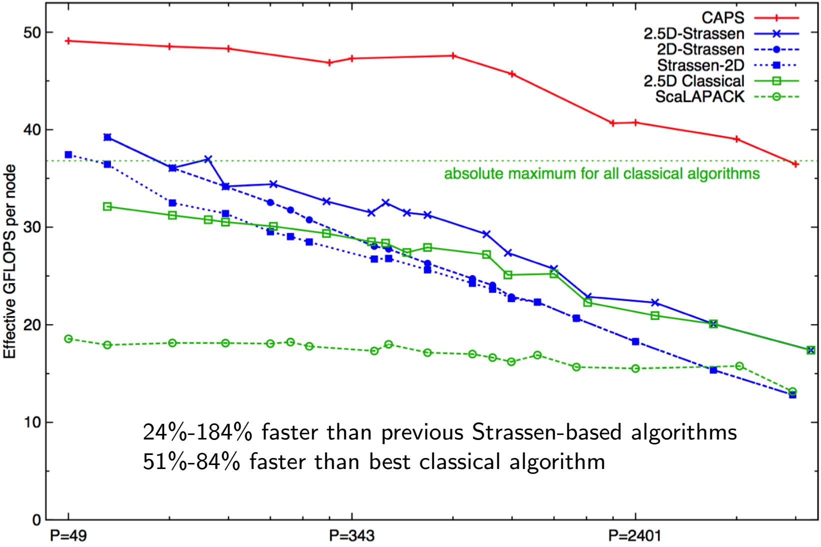
CAPS
    if enough memory **and** $P \geq 7$
        then BFS step
        else DFS step
    end if

Asymptotic costs analysis

		Flops	Bandwidth
Strassen	Lower Bound	$\frac{n^{\omega_0}}{P}$	$\max\left\{\frac{n^{\omega_0}}{PM^{\omega_0/2-1}}, \frac{n^2}{P^{2/\omega_0}}\right\}$
	2D-Strassen	$\frac{n^{\omega_0}}{P^{(\omega_0-1)/2}}$	$\frac{n^2}{P^{1/2}}$
	Strassen-2D	$\left(\frac{7}{8}\right)^{\ell}\frac{n^3}{P}$	$\left(\frac{7}{4}\right)^{\ell}\frac{n^2}{P^{1/2}}$
	CAPS	$\frac{n^{\omega_0}}{P}$	$\max\left\{\frac{n^{\omega_0}}{PM^{\omega_0/2-1}}, \frac{n^2}{P^{2/\omega_0}}\right\}$
Classical	Lower Bound	$\frac{n^3}{P}$	$\max\left\{\frac{n^3}{PM^{1/2}}, \frac{n^2}{P^{2/3}}\right\}$
	2D	$\frac{n^3}{P}$	$\frac{n^2}{P^{1/2}}$
	2.5D	$\frac{n^3}{P}$	$\max\left\{\frac{n^3}{PM^{1/2}}, \frac{n^2}{P^{2/3}}\right\}$

Performance of CAPS

Strong-scaling on Franklin (Cray XT4), $n = 94080$.



CAPS Summary

The CAPS matrix multiplication algorithm

- is communication optimal
  - matches the communication lower bounds
  - moves asymptotically less data than all existing algorithms
- is faster: asymptotically and in practice
  - faster than any parallel classical algorithm can be
  - faster than any parallel Strassen-based algorithm we are aware of
- applies to other fast matrix multiplication algorithms
  - but there might not be any other practical ones

Stability

- CAPS has the same stability properties as any other Strassen (Strassen-Winograd) algorithm.
- Weaker stability guarantee than classical, but still norm-wise stable.
- This can be improved through diagonal scaling.
  - Two best scaling schemes give incomparable bounds
  - Can check which bound is better in $O(n^2)$ time
- The improved error bounds match those of matrix factorization such as classical LU and QR.

Diagonal Scaling

Outside scaling: $D^A C D^B = (D^A A)(B D^B)$

- Scale so each row of A and each column of B has unit norm.
- Explicitly:
  - Let $D^A_{ii} = (\|A(i,:)\|)^{-1}$, and $D^B_{jj} = (\|B(:,j)\|)^{-1}$.
  - Scale $A' = D^A A$, and $B' = B D^B$.
  - Use Strassen for the product $C' = A'B'$.
  - Unscale $C = \left(D^A\right)^{-1} C' \left(D^B\right)^{-1}$.

Diagonal Scaling

Outside scaling: $D^A C D^B = (D^A A)(B D^B)$

- Scale so each row of A and each column of B has unit norm.
- Explicitly:
  - Let $D^A_{ii} = (\|A(i,:)\|)^{-1}$, and $D^B_{jj} = (\|B(:,j)\|)^{-1}$.
  - Scale $A' = D^A A$, and $B' = B D^B$.
  - Use Strassen for the product $C' = A'B'$.
  - Unscale $C = \left(D^A\right)^{-1} C' \left(D^B\right)^{-1}$.

Inside scaling: $C = (AD)(D^{-1}B)$

- Scale so each column of A has the same norm as the corresponding row of B.
- Explicitly:
  - Let $D_{ii} = (\|A(:,i)\|/\|B(i,:)\|)^{-1/2}$.
  - Scale $A' = AD$, and $B' = D^{-1}B$.
  - Use Strassen for the product $C = A'B'$.

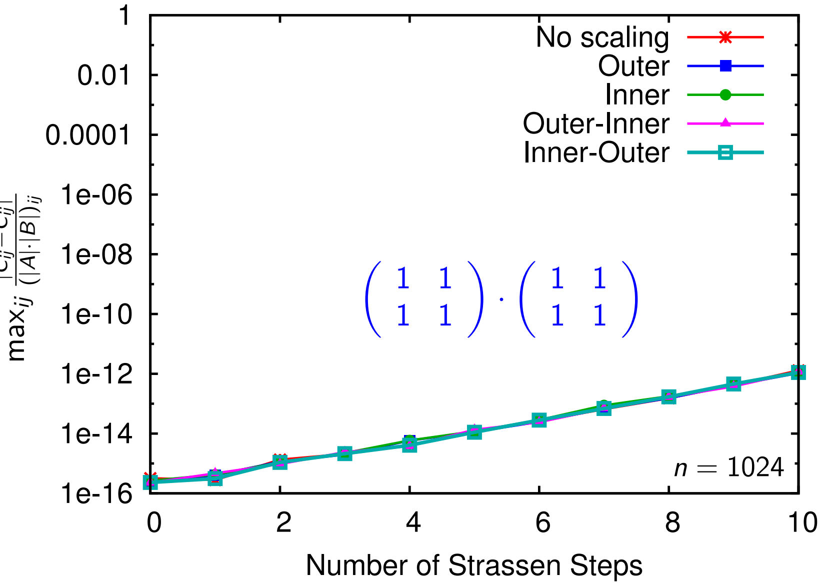
Error bounds

$$|C_{ij} - \hat{C}_{ij}| \leq O(\epsilon) f(n) \cdot \ldots$$

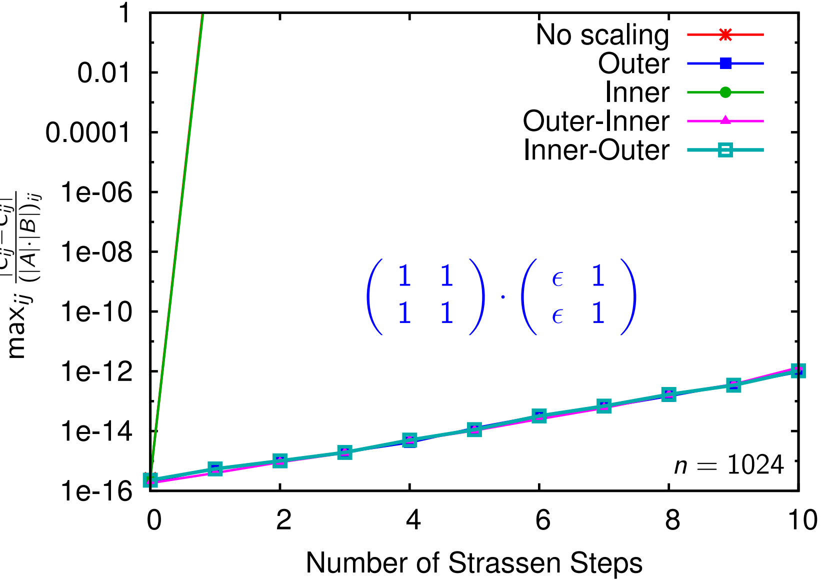
$$\begin{tikzcd}
 & \begin{array}{c}\text{Classical:}\\ (|A|\cdot|B|)_{ij}\end{array} \arrow[dl] \arrow[dr] & \\
\begin{array}{c}\text{Outside-Inside:}\\ \|A(i,:)\|\|B(:,j)\|\||D^A A|\cdot|BD^B|\|\end{array} \arrow[d] & & \begin{array}{c}\text{Inside-Outside:}\\ \|(AD)(i,:)\|\|(D^{-1}B)(:,j)\|\end{array} \arrow[d] \\
\begin{array}{c}\text{Outside:}\\ \|A(i,:)\|\|B(:,j)\|\end{array} \arrow[dr] & & \begin{array}{c}\text{Inside:}\\ \||A|\cdot|B|\|\end{array} \arrow[dl] \\
 & \begin{array}{c}\text{No Scaling:}\\ \|A\|\|B\|\end{array} &
\end{tikzcd}$$

$X \to Y$ means that bound X is stronger than bound Y.

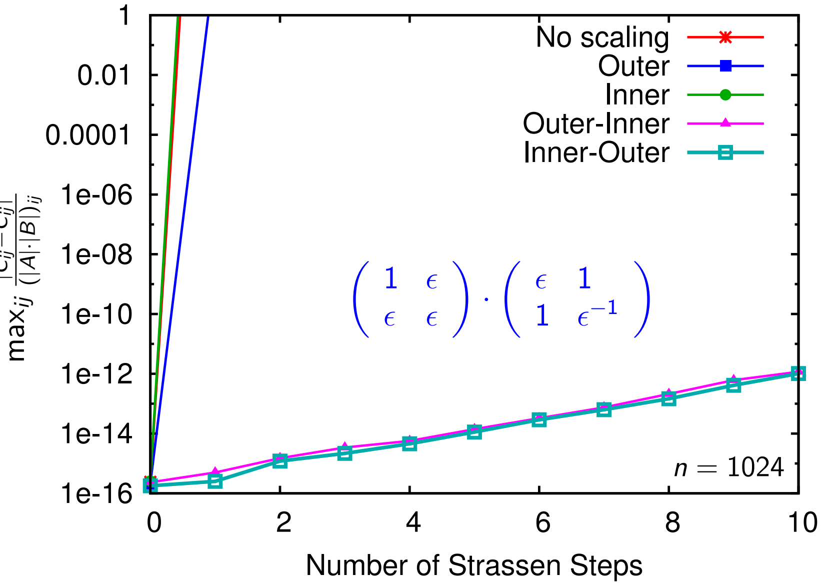
Scaling example: easy case



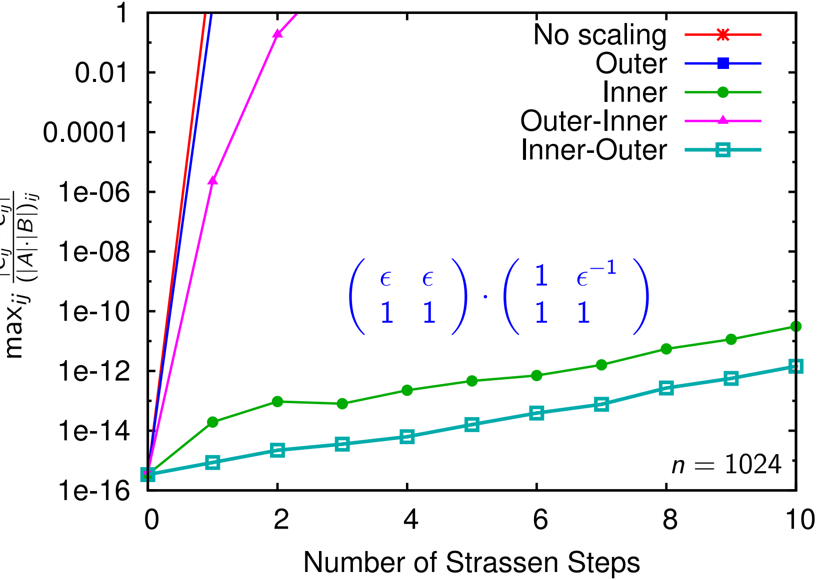
Scaling example: needs outer scaling



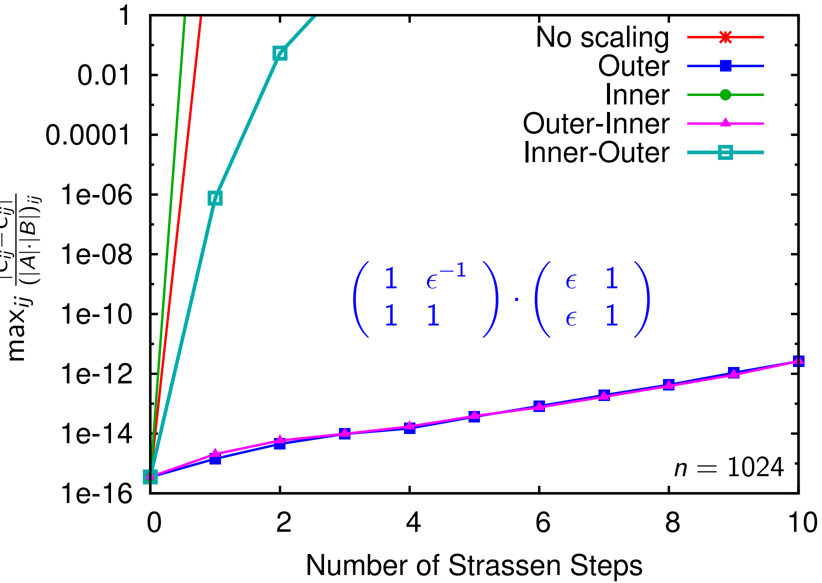
Scaling example: needs inner and outer



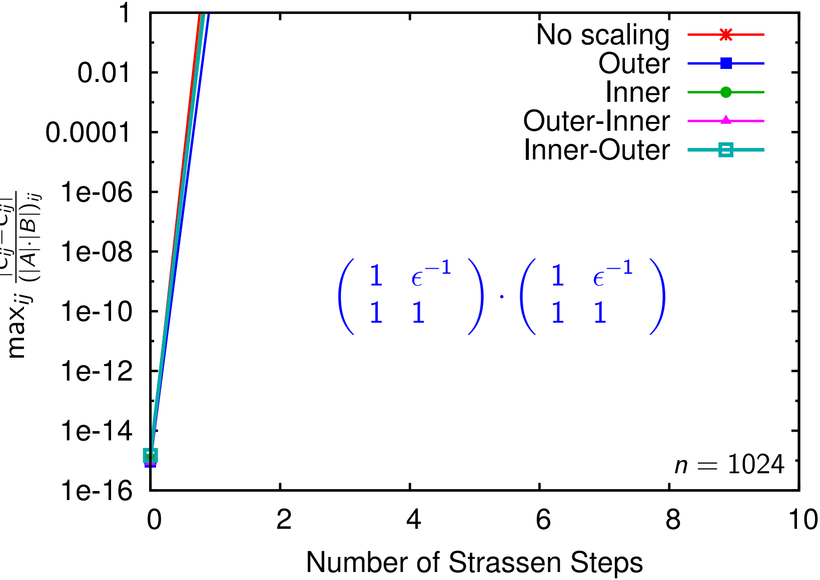
Scaling example: inner-outer better than outer-inner



Scaling example: outer-inner better than inner-outer



Scaling example: problems scaling can't fix



Error bounds

$$|C_{ij} - \hat{C}_{ij}| \leq O(\epsilon) f(n) \cdot \ldots$$

$$\begin{tikzcd}
 & \begin{array}{c}\text{Classical:}\\ (|A|\cdot|B|)_{ij}\end{array} \arrow[dl] \arrow[dr] & \\
\begin{array}{c}\text{Outside-Inside:}\\ \|A(i,:)\|\|B(:,j)\|\||D^A A|\cdot|BD^B|\|\end{array} \arrow[d] & & \begin{array}{c}\text{Inside-Outside:}\\ \|(AD)(i,:)\|\|(D^{-1}B)(:,j)\|\end{array} \arrow[d] \\
\begin{array}{c}\text{Outside:}\\ \|A(i,:)\|\|B(:,j)\|\end{array} \arrow[dr] & & \begin{array}{c}\text{Inside:}\\ \||A|\cdot|B|\|\end{array} \arrow[dl] \\
 & \begin{array}{c}\text{No Scaling:}\\ \|A\|\|B\|\end{array} &
\end{tikzcd}$$

$X \rightarrow Y$ means that bound X is stronger than bound Y.

Stability summary

- Scaling improves error bound of Strassen
  - To be comparable to many other algorithms
  - But still not a good as classical algorithm
- Applies to other fast matrix multiplication algorithms
- Inner-then-outer or outer-then-inner scaling are best
- Can choose between them by evaluating their bounds
- Open problem: simultaneously attain inner and outer scaling bounds?

Practicality of Large Scale Fast Matrix Multiplication

Grey Ballard, James Demmel, Olga Holtz,
Benjamin Lipshitz and Oded Schwartz

Thank You!

Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from Par Lab affiliates National Instruments, NEC, Nokia, NVIDIA, and Samsung. Research is also supported by DOE grants DE-SC0003959, DE-SC0004938, and DE-AC02-05CH11231; the Sofja Kovalevskaja programme of Alexander von Humboldt Foundation; and by the National Science Foundation under agreement DMS-0635607.

References

G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz.
Communication-optimal parallel algorithm for Strassen's matrix multiplication.
Technical Report EECS-2012-32, UC Berkeley, March 2012.
To appear in SPAA 2012.

G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz.
Strong scaling of matrix multiplication algorithms and memory-independent communication lower bounds.
Technical Report EECS-2012-31, UC Berkeley, March 2012.
To appear in SPAA 2012.

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, New York, NY, USA, 2011. ACM.

B. Lipshitz, G. Ballard, O. Schwartz, and J. Demmel.
Communication-avoiding parallel Strassen: Implementation and performance.
Technical Report EECS-2012-90, UC Berkeley, May 2012.
Submitted to SC 2012.

Extra slides

1. Previous Parallel Strassen
2. Data Layout
3. Strassen-Winograd Algorithm
4. Hardware scaling
5. Time breakdown
6. Small problems

Previous parallel Strassen-based algorithms

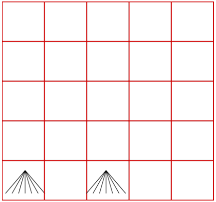
2D-Strassen: [Luo, Drake 95]

Run classical 2D inter-processors.

- Same communication costs as classical 2D.

Run Strassen locally.

- Can't use Strassen on the full matrix size.



Extras

Previous parallel Strassen-based algorithms

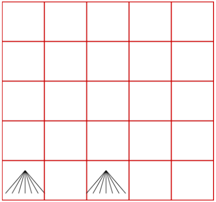
2D-Strassen: [Luo, Drake 95]

Run classical 2D inter-processors.

- Same communication costs as classical 2D.

Run Strassen locally.

- Can't use Strassen on the full matrix size.



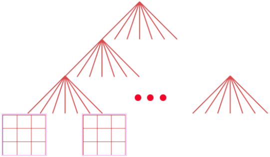
Strassen-2D: [Luo, Drake 95; Grayson, Shah, van de Geijn 95]

Run Strassen inter-processors

- This part can be done without communication.

Then run classical 2D.

- Communication costs grow exponentially with the number of Strassen steps.



▸ Extras

Previous parallel Strassen-based algorithms

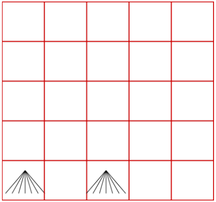
2D-Strassen: [Luo, Drake 95]

Run classical 2D inter-processors.

- Same communication costs as classical 2D.

Run Strassen locally.

- Can't use Strassen on the full matrix size.



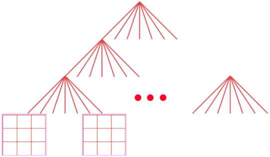
Strassen-2D: [Luo, Drake 95; Grayson, Shah, van de Geijn 95]

Run Strassen inter-processors

- This part can be done without communication.

Then run classical 2D.

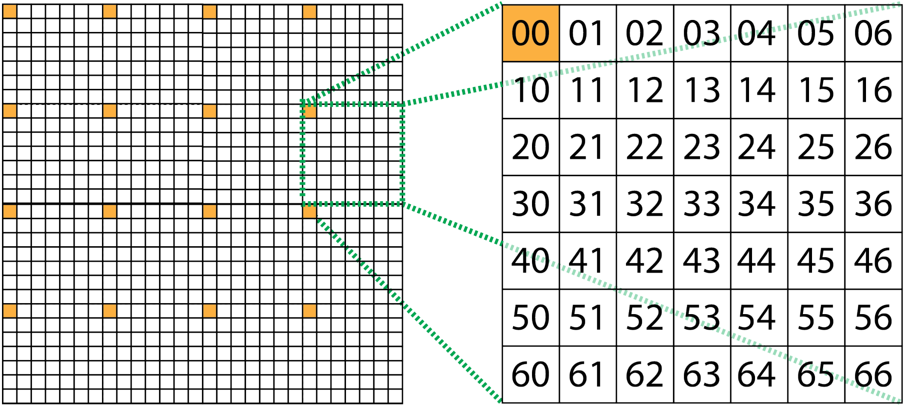
- Communication costs grow exponentially with the number of Strassen steps.



Neither is communication optimal

Extras

Data Layout



Extras

Strassen-Winograd Algorithm

$$\left(\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right) = C = A \cdot B = \left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right) \cdot \left(\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right)$$

$$\begin{array}{lll} & & Q_i = S_i \cdot T_i \\ S_0 = A_{11} & T_0 = B_{11} & U_1 = Q_i + Q_4 \\ S_1 = A_{12} & T_1 = B_{21} & U_2 = U_1 + Q_5 \\ S_2 = A_{21} + A_{22} & T_2 = B_{12} + B_{11} & U_3 = U_1 + Q_5 \\ S_3 = S_2 - A_{12} & T_3 = B_{22} - T_2 & C_{11} = Q_1 + Q_2 \\ S_4 = A_{11} - A_{21} & T_4 = B_{22} - B_{12} & C_{12} = U_3 + Q_6 \\ S_5 = A_{12} + S_3 & T_5 = B_{22} & C_{21} = U_2 - Q_7 \\ S_6 = A_{22} & T_6 = T_3 - B_{21} & C_{22} = U_2 + Q_3 \end{array}$$

Extras

Implications for hardware scaling

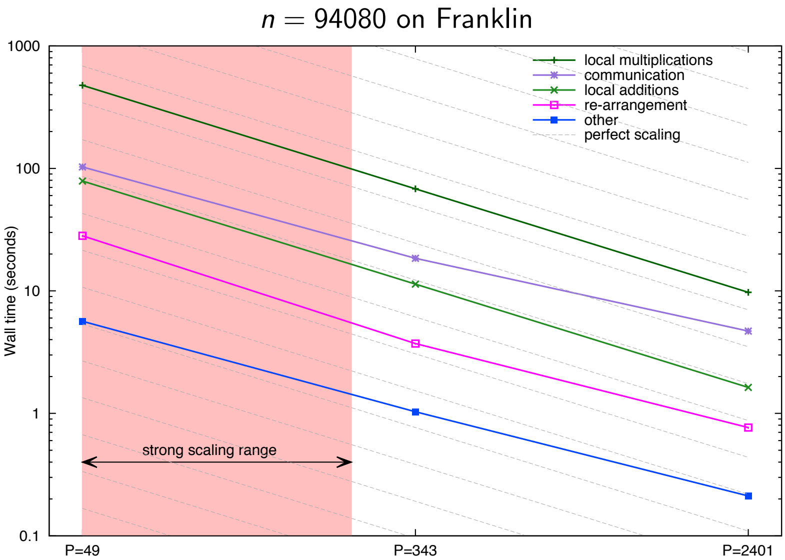
Requirements so that dense matrix multiplication is computation bound.

	Bandwidth Requirement	Latency Requirement
Classic	$\gamma M^{1/2} \gtrsim \beta$	$\gamma M^{3/2} \gtrsim \alpha$
Strassen	$\gamma M^{\omega_0/2-1} \gtrsim \beta$	$\gamma M^{\omega_0/2} \gtrsim \alpha$

Strassen performs fewer flops and less communication, but is more demanding on the hardware.

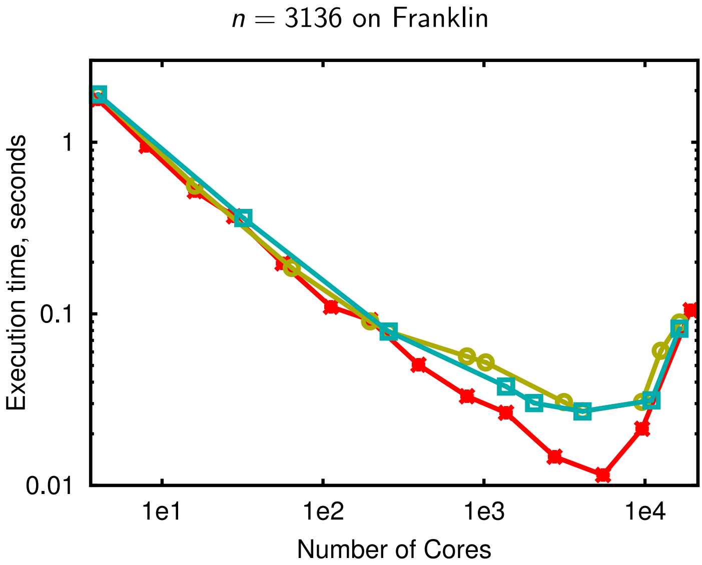
Extras

CAPS time breakdown



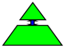
Extras

Performance on small problems



▸ Extras

Sequential recursive Strassen is communication optimal



Sequential

- Run Strassen algorithm recursively.
- When blocks are small enough, work in localy memory, so no further bandwidth cost

$$W(n,M) = \begin{cases} 7W(\frac{n}{2}, M) + O(n^2) & \text{if } 3n^2 > M \\ O(n^2) & \text{otherwise} \end{cases}$$

- Solution is

$$W(n,M) = O\left(\frac{n^{\omega_0}}{M^{\omega_0/2-1}}\right)$$

▸ Extras

Extra slides

1. Previous Parallel Strassen
2. Data Layout
3. Strassen-Winograd Algorithm
4. Hardware scaling
5. Time breakdown
6. Small problems