Communication-Avoiding Parallel Strassen: Implementation and Performance

Grey Ballard, James Demmel, Benjamin Lipshitz and Oded Schwartz

Sandia National Labs
UC Berkeley

Simons Institute Workshop
October 22, 2013

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 and by the National Science Foundation under agreement DMS-0635607.

The Plan

- I'll present a new parallel algorithm based on Strassen's matrix multiplication, called **C**ommunication **A**voiding **P**arallel **S**trassen
- The new Strassen-based parallel algorithm CAPS
  - is communication optimal
    - matches the lower bounds [B., Demmel, Holtz, Schwartz, '11]
  - is faster: in theory and in practice
- I'll also show performance results and talk about practical considerations for using Strassen and CAPS
- Strassen's algorithm is not just a theoretical idea: it can be practical in parallel and deserves further exploration

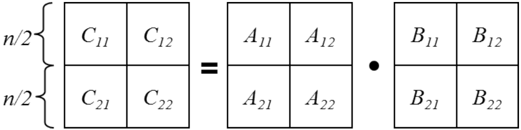
Outline

Motivation: Strassen's fast matrix multiplication (1969)

Strassen's original algorithm uses 7 multiplies and 18 adds for $n = 2$.
Most importantly, it can be applied recursively.

$$
\begin{aligned}
{\color{red}Q_1} &= (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \\
{\color{red}Q_2} &= (A_{21} + A_{22}) \cdot B_{11} \\
{\color{red}Q_3} &= A_{11} \cdot (B_{12} - B_{22}) \\
{\color{red}Q_4} &= A_{22} \cdot (B_{21} - B_{11}) \\
{\color{red}Q_5} &= (A_{11} + A_{12}) \cdot B_{22} \\
{\color{red}Q_6} &= (A_{21} - A_{11}) \cdot (B_{11} + B_{12}) \\
{\color{red}Q_7} &= (A_{12} - A_{22}) \cdot (B_{21} + B_{22})
\end{aligned}
$$

$$
\begin{aligned}
C_{11} &= {\color{red}Q_1} + {\color{red}Q_4} - {\color{red}Q_5} + {\color{red}Q_7} \\
C_{12} &= {\color{red}Q_3} + {\color{red}Q_5} \\
C_{21} &= {\color{red}Q_2} + {\color{red}Q_4} \\
C_{22} &= {\color{red}Q_1} - {\color{red}Q_2} + {\color{red}Q_3} + {\color{red}Q_6}
\end{aligned}
$$



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

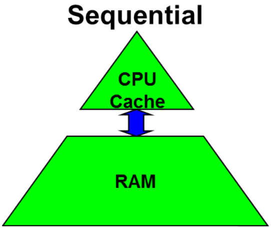
$$F(n) = \Theta\left(n^{\log_2 7}\right)$$

$$\log_2 7 \approx 2.81$$

Motivation: 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 will only get more expensive relative to arithmetic



Motivation: communication costs

$\gamma =$ time per FLOP

$\beta =$ time per word

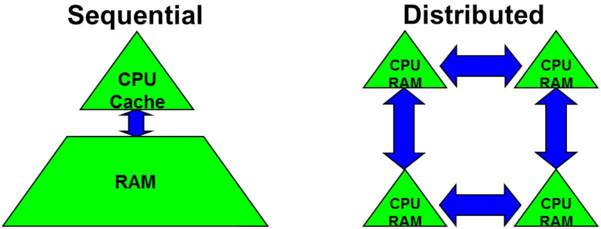
$\alpha =$ time per message

$F = \#$Flops

$BW = \#$Words

$L = \#$Messages

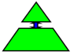
$$\text{Running time } = \gamma \cdot F + \beta \cdot BW + \alpha \cdot L$$



Outline

Communication lower bounds for matrix multiplication

Classical (cubic):

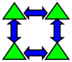


Sequential

[Hong & Kung 81]

- Combinatorial proof
- Sequential only

$$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} M\right)$$



Distributed

[Irony, Toledo, Tiskin 04]

- Geometric proof
- Sequential and parallel

$$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} \frac{M}{P}\right)$$

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

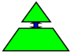
Communication lower bounds for matrix multiplication

[B., Demmel, Holtz, Schwartz 11]:

- Sequential and parallel
- Graph expansion proof

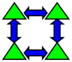
Strassen:

Classical (cubic):



Sequential

$$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right) \qquad \Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} M\right)$$



Distributed

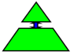
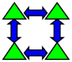
$$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \frac{M}{P}\right) \qquad \Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} \frac{M}{P}\right)$$

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

Communication lower bounds for matrix multiplication

[B., Demmel, Holtz, Schwartz 11]:

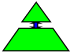
- Sequential and parallel
- Graph expansion proof

	Strassen:	Strassen-like:	Classical (cubic):
 Sequential	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\omega_0} M\right)$	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} M\right)$
 Distributed	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \frac{M}{P}\right)$	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\omega_0} \frac{M}{P}\right)$	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} \frac{M}{P}\right)$

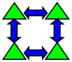
n = matrix dimension, M = fast/local memory size, P = number of processors

Communication lower bounds for matrix multiplication

Strassen:



$$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right)$$



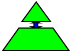
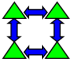
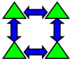
$$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \frac{M}{P}\right)$$

Classical (cubic):

$$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} M\right)$$

$$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} \frac{M}{P}\right)$$

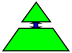
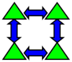
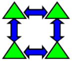
n = matrix dimension, M = fast/local memory size, P = number of processors

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

Memory independent bound [B., Demmel, Holtz, Lipshitz, Schwartz 12]

Communication lower bounds for matrix multiplication

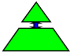
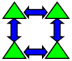
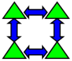
Algorithms attaining these bounds?

	Strassen:	Classical (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	$\Omega\left(\frac{n^2}{P^{2/\log_2 7}}\right)$	$\Omega\left(\frac{n^2}{P^{2/\log_2 8}}\right)$

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

Communication lower bounds for matrix multiplication

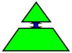
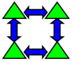
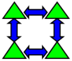
Algorithms attaining these bounds?

	Strassen:	Classical (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	$\Omega\left(\frac{n^2}{P^{2/\log_2 7}}\right)$	$\Omega\left(\frac{n^2}{P^{2/\log_2 8}}\right)$

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

Communication lower bounds for matrix multiplication

Algorithms attaining these bounds?

	Strassen:		Classical (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	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \frac{M}{P}\right)$	[B., Demmel, Holtz, Lipshitz, Schwartz 12] [McColl & Tiskin 99]	$\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} \frac{M}{P}\right)$ ✓
 Distributed	$\Omega\left(\frac{n^2}{P^{2/\log_2 7}}\right)$		$\Omega\left(\frac{n^2}{P^{2/\log_2 8}}\right)$ ✓

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

Lessons from lower bounds

1. Don't use a classical algorithm for the communication

   - Strassen can communicate less than classical

     Strassen: $\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \frac{M}{P}\right)$ Classical: $\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} \frac{M}{P}\right)$

Lessons from lower bounds

1. Don't use a classical algorithm for the communication

   - Strassen can communicate less than classical

   Strassen: $\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7}\frac{M}{P}\right)$ Classical: $\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8}\frac{M}{P}\right)$

2. Use all available memory

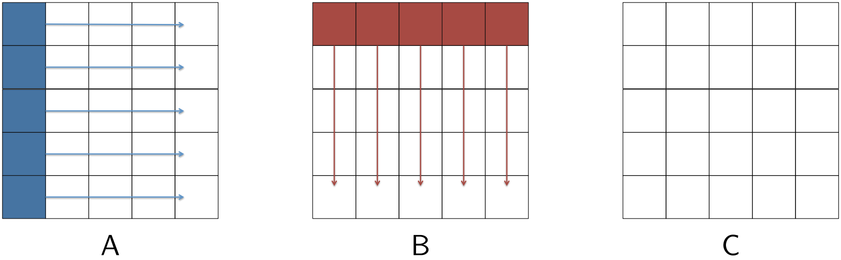
   - Communication bound decreases with increased memory
   - Up to a factor of $O(P^{1-2/\log_2 7})$ extra memory is useful

   Strassen: $\Omega\left(\max\left\{\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7}\frac{M}{P}, \frac{n^2}{P^{2/\log_2 7}}\right\}\right)$

Outline

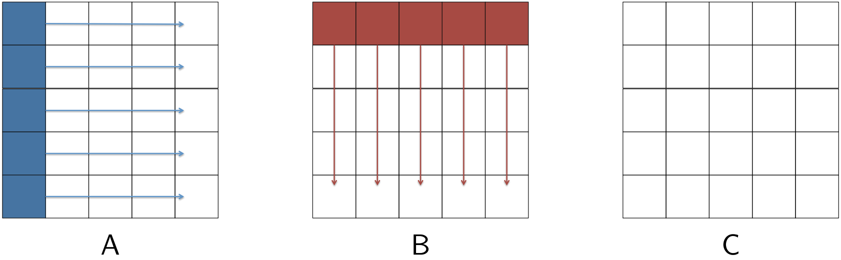
Simple "2D" Classical Algorithm

Here's the basic communication pattern for the classical "2D" algorithm:



Simple "2D" Classical Algorithm

Here's the basic communication pattern for the classical "2D" algorithm:



- 2D: think Cannon or SUMMA
  [Cannon 69, van de Geijn & Watts 97]
- 2.5D: think reduced communication by using more memory
  [Solomonik & Demmel 11]

Previous parallel Strassen-based algorithms

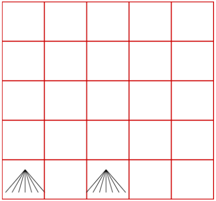
2D-Strassen: [Luo & Drake 95]

Run classical 2D inter-processors.

- Same communication costs as classical 2D.

Run Strassen locally.

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



Previous parallel Strassen-based algorithms

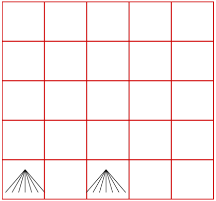
2D-Strassen: [Luo & Drake 95]

Run classical 2D inter-processors.

- Same communication costs as classical 2D.

Run Strassen locally.

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



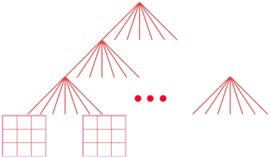
Strassen-2D: [Luo & Drake 95; Grayson, Shah, van de Geijn 95]

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.



Previous parallel Strassen-based algorithms

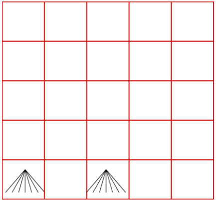
2D-Strassen: [Luo & Drake 95]

Run classical 2D inter-processors.

- Same communication costs as classical 2D.

Run Strassen locally.

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



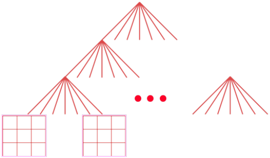
Strassen-2D: [Luo & Drake 95; Grayson, Shah, van de Geijn 95]

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.

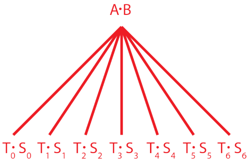


Neither is communication optimal, even if you use 2.5D

Main idea of CAPS algorithm

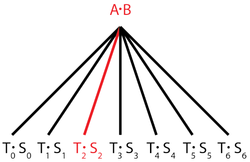
At each level of recursion tree, choose either breadth-first or depth-first traversal of the recursion tree

Breadth-First-Search (BFS)



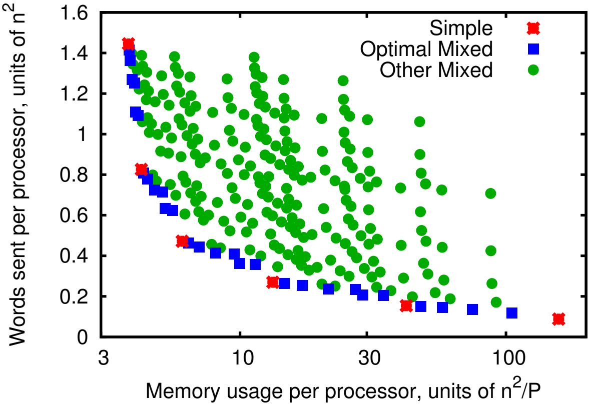
- Runs all 7 multiplies in parallel
  - each uses $P/7$ processors
- Requires $7/4$ as much extra memory
- Requires communication, but
- All BFS minimizes communication if possible

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

Tuning the choices of BFS and DFS Steps



The memory and communication costs of all $\binom{10}{5} = 252$ possible interleavings of BFS and DFS steps for multiplying matrices of size $n = 351{,}232$ on $P = 7^5 = 16{,}807$ processors using 10 Strassen steps.

Asymptotic costs analysis

		Flops	Bandwidth Cost
Strassen	Lower Bound	$\frac{n^{\log_2 7}}{P}$	$\max\left\{\frac{n^{\log_2 7}}{PM^{(\log_2 7)/2-1}}, \frac{n^2}{P^{2/\log_2 7}}\right\}$
	2D-Strassen	$\frac{n^{\log_2 7}}{P^{(\log_2 7-1)/2}}$	$\frac{n^2}{P^{1/2}}$
	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	$\frac{n^{\log_2 7}}{P}$	$\max\left\{\frac{n^{\log_2 7}}{PM^{(\log_2 7)/2-1}}, \frac{n^2}{P^{2/\log_2 7}}\right\}$
Classical			

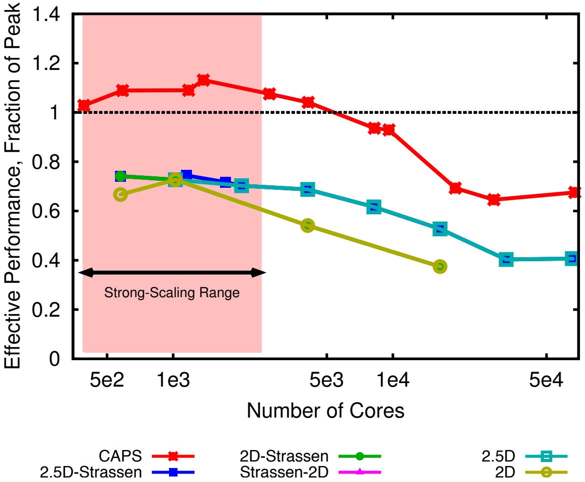
Asymptotic costs analysis

		Flops	Bandwidth Cost
Strassen	Lower Bound	$\frac{n^{\log_2 7}}{P}$	$\max\left\{\frac{n^{\log_2 7}}{PM^{(\log_2 7)/2-1}}, \frac{n^2}{P^{2/\log_2 7}}\right\}$
	2D-Strassen	$\frac{n^{\log_2 7}}{P^{(\log_2 7-1)/2}}$	$\frac{n^2}{P^{1/2}}$
	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	$\frac{n^{\log_2 7}}{P}$	$\max\left\{\frac{n^{\log_2 7}}{PM^{(\log_2 7)/2-1}}, \frac{n^2}{P^{2/\log_2 7}}\right\}$
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	$\frac{n^3}{P}$	$\max\left\{\frac{n^3}{PM^{1/2}}, \frac{n^2}{P^{2/3}}\right\}$

Outline

Performance of CAPS on large problems

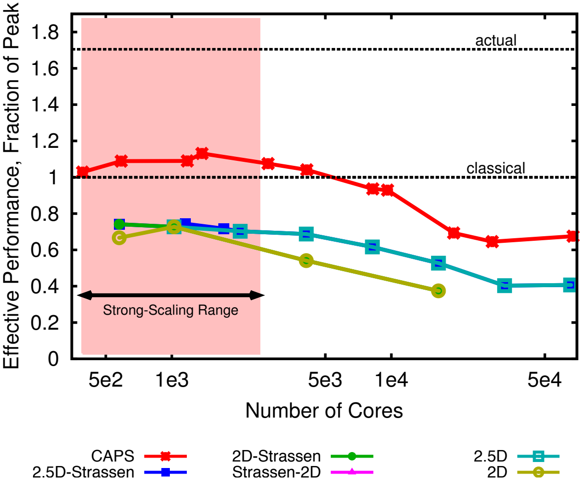
Strong-scaling on Intrepid (IBM BG/P), $n = 65{,}856$.



Strassen-Winograd peak

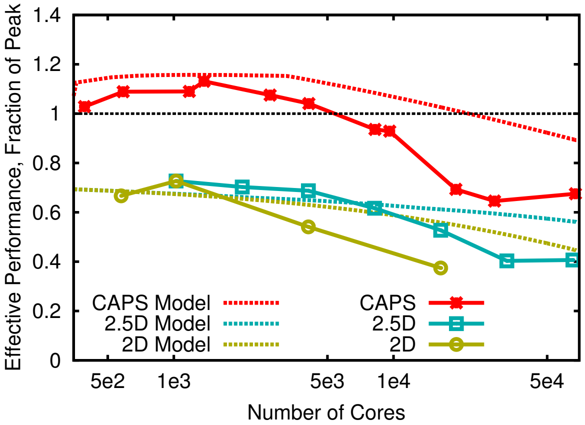
Performance of CAPS on large problems

Strong-scaling on Intrepid (IBM BG/P), $n = 65{,}856$.



Strassen-Winograd peak

Performance: Model vs Actual

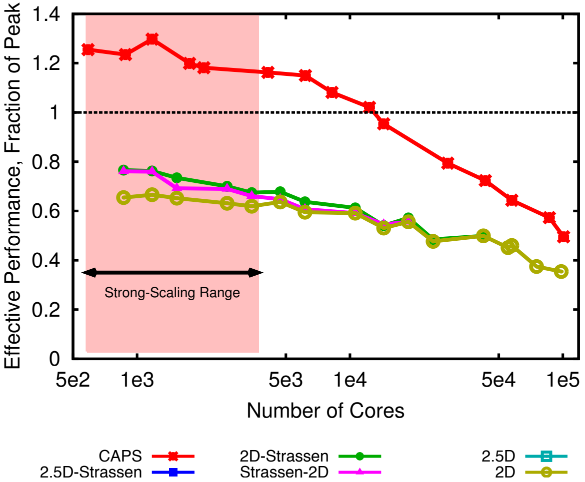


Comparison of the parallel models with the algorithms in strong scaling of matrix dimension $n = 65{,}856$ on Intrepid.

▸ No Contention

Performance of CAPS on large problems

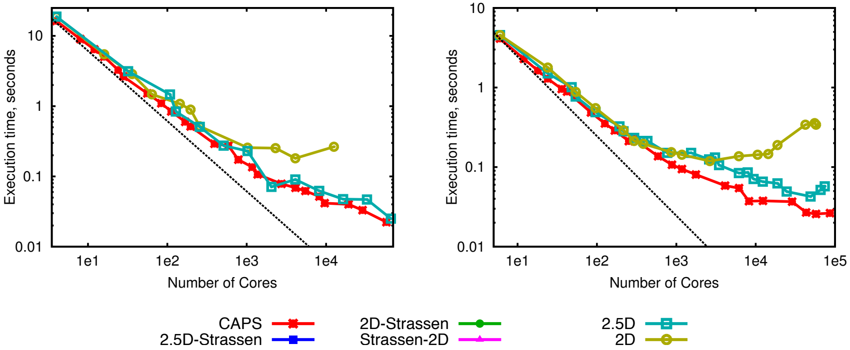
Strong-scaling on Hopper (Cray XE6), $n = 131{,}712$.



Franklin

Performance of CAPS on small (comm-bound) problems

Strong-scaling on Intrepid (left) and Hopper (right), $n = 4704$.



Outline

Practical Considerations for Strassen

1. Harder to reach actual peak performance
   - computation to communication ratio smaller than classical
2. Additions and multiplications are no longer balanced
3. Architectures are based on powers of 2 not 7
   - CAPS prefers $P = m \cdot 7^k$
   - Intrepid requires allocation of power of two number of nodes
4. Stability bounds are not as strong as for classical

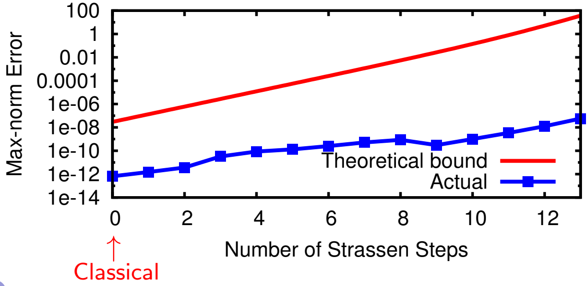
Stability - why you shouldn't worry

- 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 with techniques like diagonal scaling

Stability - why you shouldn't worry

- 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 with techniques like diagonal scaling
- Taking fewer Strassen steps improves the bound
- Theoretical bounds are pessimistic in the typical case

$$\frac{\|C - A \cdot B\|}{\|A\| \, \|B\|}$$



Diagonal Scaling

Summary

The CAPS matrix multiplication algorithm

1. is communication optimal
2. is faster: in theory and in practice
3. can be practical and should be used and improved

Communication-Avoiding Parallel Strassen: Implementation and Performance

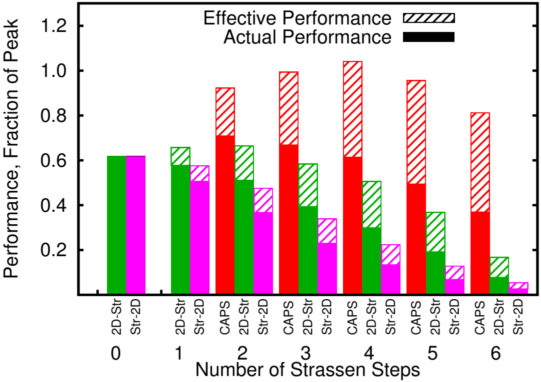
Grey Ballard, James Demmel, Benjamin Lipshitz and Oded Schwartz

Thank You!

www.eecs.berkeley.edu/~ballard
http://bebop.cs.berkeley.edu

Extra slides

Effective vs Actual Performance



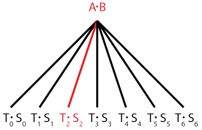
Efficiency at various numbers of Strassen steps, $n = 21952$, on 49 nodes (196 cores) of Intrepid.

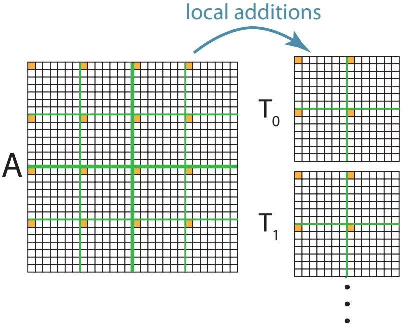
Extras

Communication-Free DFS

Possible if each processor owns corresponding entries of four submatrices of A, B, and C. [Luo & Drake 95; Grayson, Shah, van de Geijn 95]

- Additions of submatrices of A to form the T_i (no communication)
- Additions of submatrices of B to form the S_i (no communication)
- Recursive calls $Q_i = T_i \cdot S_i$ (communication deeper in recursion tree)
- Additions of the Q_i to form submatrices of C (no communication)



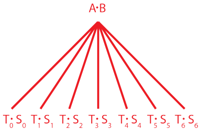


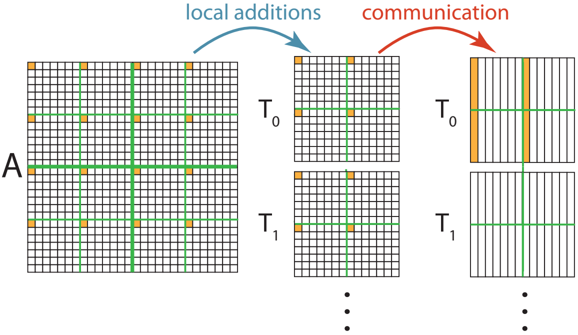
Extras

Communication Pattern of BFS

- Additions of submatrices of A, B to form T_i, S_i (no communication)
- Redistribution of the T_i, S_i (communication)
- Recursive calls $Q_i = T_i \cdot S_i$ (communication deeper in recursion tree)
- Redistribution of the Q_i (communication)
- Additions of the Q_i to form submatrices of C (no communication)

Redistributions are disjoint 7-way all-to-all communications.

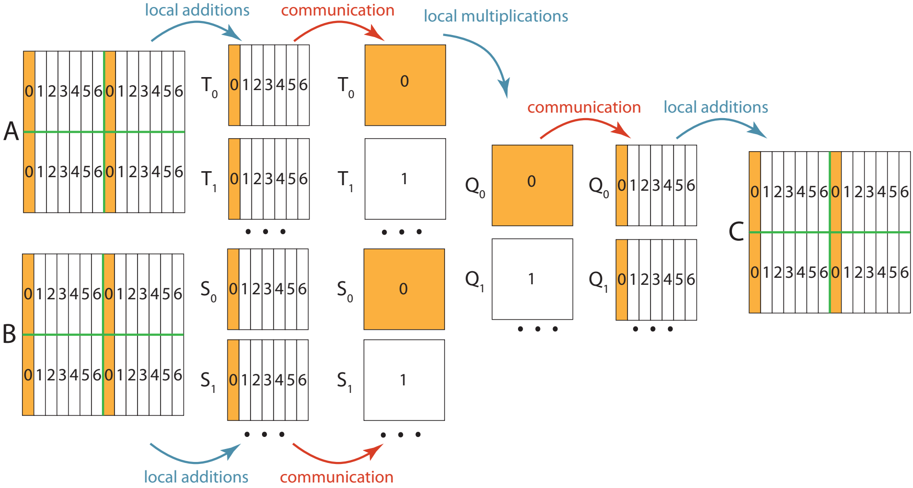




Extras

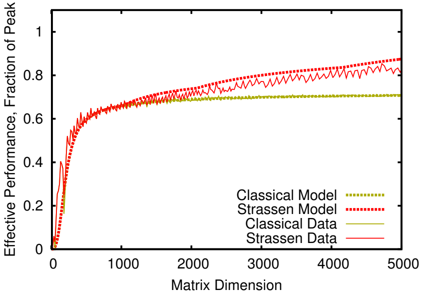
BFS on 7 Processors

Requires 3 all-to-all communications, one for each of A, B, C

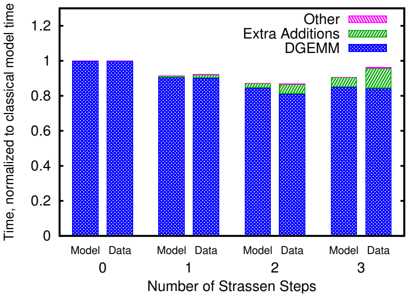


▸ Extras

Sequential Performance

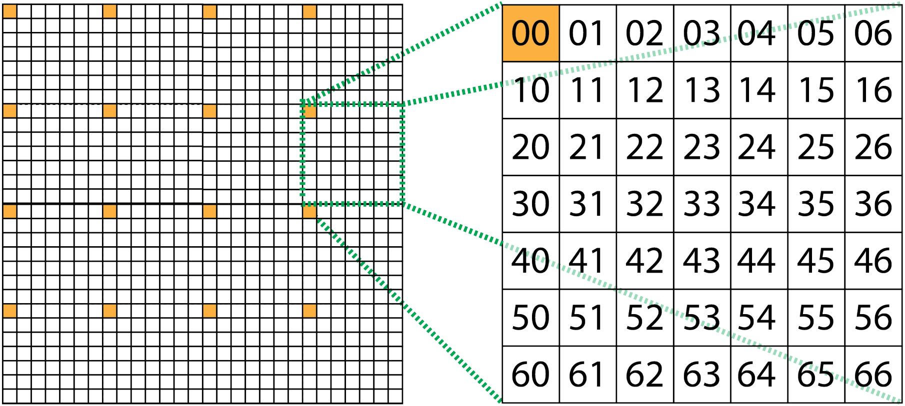


Comparison of the sequential model to the actual performance of classical and Strassen matrix multiplication on four cores (one node) of Intrepid.



Time breakdown comparison between the sequential model and the data for $n = 4097$. Both model and data times are normalized to the modeled classical algorithm time.

Extras


00 01 02 03 04 05 06
10 11 12 13 14 15 16
20 21 22 23 24 25 26
30 31 32 33 34 35 36
40 41 42 43 44 45 46
50 51 52 53 54 55 56
60 61 62 63 64 65 66

▸ Extras

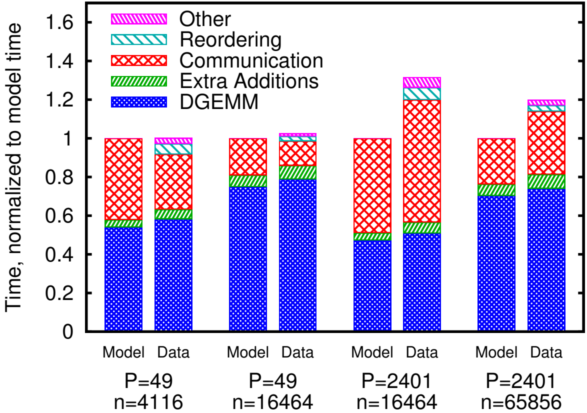
Strassen-Winograd Algorithm

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

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

Extras

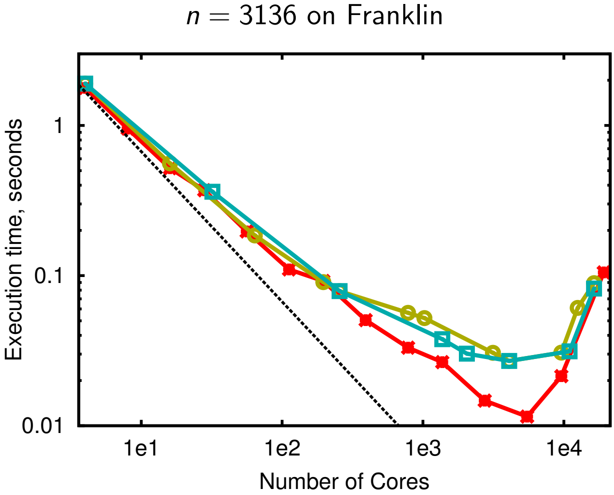
Performance Breakdown: Model vs Actual



Time breakdown comparison between the parallel model and data on Intrepid. In each case the entire modeled execution time is normalized to 1.

Extras

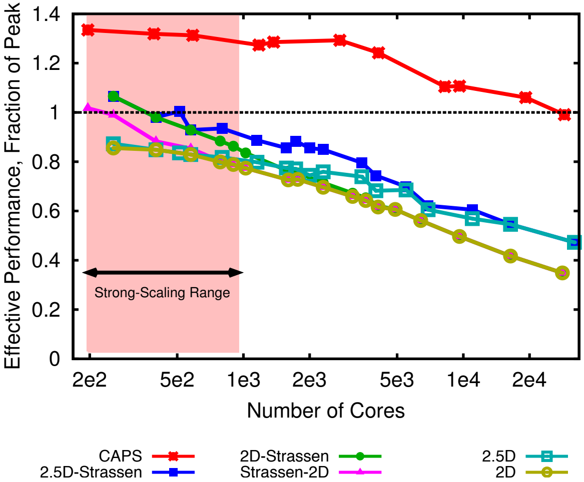
Performance on Franklin for small problem



▸ Extras

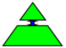
Performance of CAPS on large problem

Strong-scaling on Franklin (Cray XT4), $n = 94{,}080$.



Extras Hopper

Sequential recursive Strassen is communication optimal



Sequential

- Run Strassen algorithm recursively.
- When blocks are small enough, work in local 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

Diagonal Scaling

Outside scaling:

- 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 $B' = BD^B$.
  - 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:

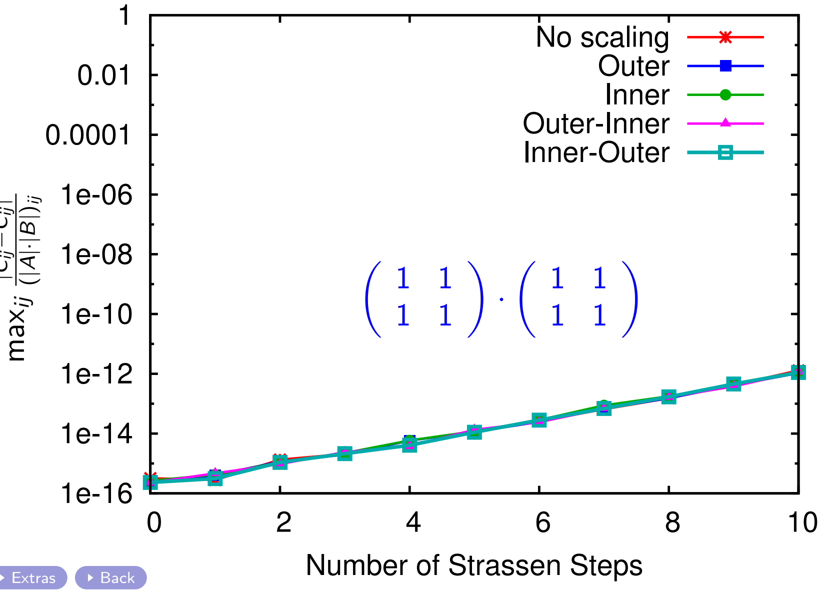
- Scale so each row of A and each column of B has unit norm.
- Explicitly:
  - Let $D^A_{ii} = (\|A(i,:)\|)^{-1}$, and $D^B_{jj} = (\|B(:,j)\|)^{-1}$.
  - Scale $A' = D^A A$, and $B' = BD^B$.
  - Use Strassen for the product $C' = A'B'$.
  - Unscale $C = \left(D^A\right)^{-1} C' \left(D^B\right)^{-1}$.

Inside scaling:

- 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 $C = A'B'$.

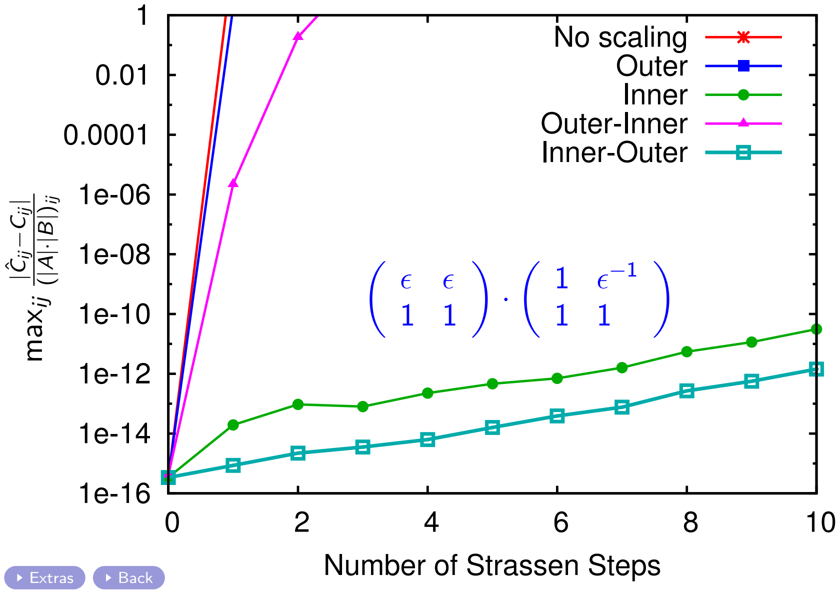
Extras Back

Stability: easy case



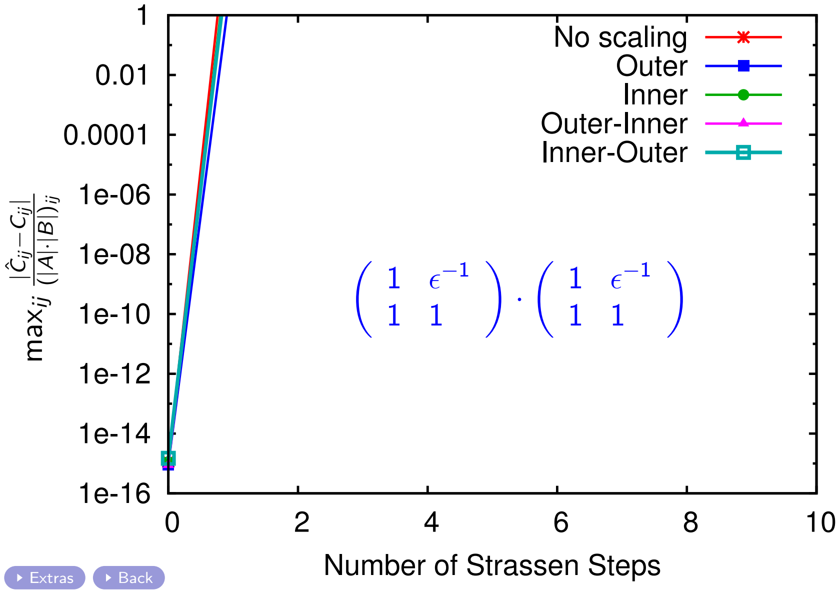
▸ Extras ▸ Back

Stability: more interesting case



Extras Back

Stability: problems scaling can't fix



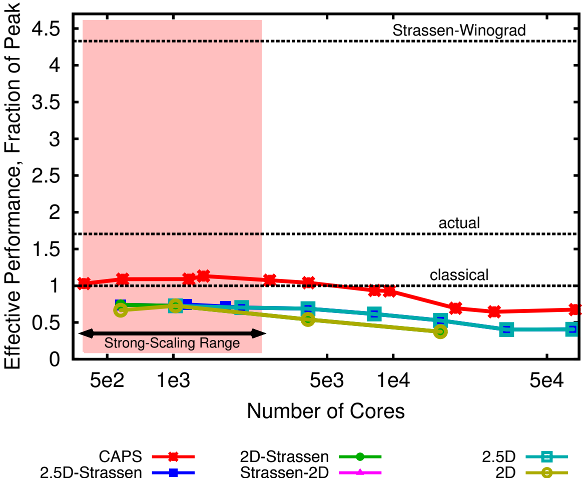
▸ Extras ▸ Back

Discussion / open problems

- Our parallelization approach extends to other matrix multiplication algorithms:
    - classical matrix multiplication (matching the 2.5D algorithm)
    - other fast matrix multiplication algorithms
- And to other algorithms with recursive formulations?
- Make use of CAPS within other linear algebra algorithms

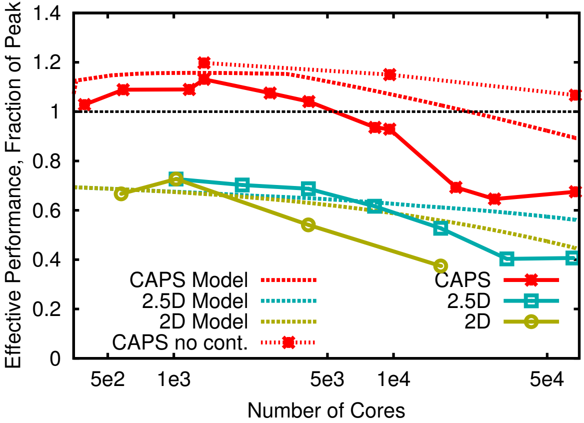
Performance of CAPS on large problems

Strong-scaling on Intrepid (IBM BG/P), $n = 65{,}856$.



Back

Performance: Model vs Actual



Comparison of the parallel models with the algorithms in strong scaling of matrix dimension $n = 65{,}856$ on Intrepid.

Back

Extra slides