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 **Communication Avoiding Parallel Strassen**

- 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

1. Motivation
2. Lower Bounds
3. Algorithms
4. Performance
5. Practical Considerations
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.

\[
\begin{align*}
Q_1 & = (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \\
Q_2 & = (A_{21} + A_{22}) \cdot B_{11} \\
Q_3 & = A_{11} \cdot (B_{12} - B_{22}) \\
Q_4 & = A_{22} \cdot (B_{21} - B_{11}) \\
Q_5 & = (A_{11} + A_{12}) \cdot B_{22} \\
Q_6 & = (A_{21} - A_{11}) \cdot (B_{11} + B_{12}) \\
Q_7 & = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})
\end{align*}
\]

\[
\begin{align*}
C_{11} & = Q_1 + Q_4 - Q_5 + Q_7 \\
C_{12} & = Q_3 + Q_5 \\
C_{21} & = Q_2 + Q_4 \\
C_{22} & = Q_1 - Q_2 + Q_3 + Q_6
\end{align*}
\]

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

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

\[
\log_2 7 \approx 2.81
\]
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

$\gamma = \text{time per FLOP}$

$\beta = \text{time per word}$

$\alpha = \text{time per message}$

$F = \#\text{Flops}$

$BW = \#\text{Words}$

$L = \#\text{Messages}$

Running time $= \gamma \cdot F + \beta \cdot BW + \alpha \cdot L$
Outline

1. Motivation
2. Lower Bounds
3. Algorithms
4. Performance
5. Practical Considerations
Communication lower bounds for matrix multiplication

Classical (cubic):

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

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

\[ n = \text{matrix dimension}, \ M = \text{fast/local memory size}, \ P = \text{number of processors} \]
Communication lower bounds for matrix multiplication

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

- Sequential and parallel
- Graph expansion proof

Strassen:

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

Classical (cubic):

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

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

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

\[ n = \text{matrix dimension}, \ M = \text{fast/local memory size}, \ P = \text{number of processors} \]
Communication lower bounds for matrix multiplication

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

- Sequential and parallel
- Graph expansion proof

Strassen:
\[ \Omega \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 7} M \right) \]

Strassen-like:
\[ \Omega \left( \left( \frac{n}{\sqrt{M}} \right)^{\omega_0} M \right) \]

Classical (cubic):
\[ \Omega \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 8} M \right) \]

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

\[ \Omega \left( \left( \frac{n}{\sqrt{M}} \right)^{\omega_0} \frac{M}{P} \right) \]

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

\[ n = \text{matrix dimension}, \quad M = \text{fast/local memory size}, \quad P = \text{number of processors} \]
Communication lower bounds for matrix multiplication

Strassen:

Sequential: \[ \Omega \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 7} M \right) \]

Distributed: \[ \Omega \left( \left( \frac{n}{\sqrt{M}} \right)^{\log_2 7} \frac{M}{P} \right) \]

Classical (cubic):

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

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

\[ n = \text{matrix dimension}, \quad M = \text{fast/local memory size}, \quad P = \text{number of processors} \]
Communication lower bounds for matrix multiplication

Strassen:

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

Classical (cubic):

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

Sequential

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

Distributed

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

Distributed

\[ \Omega \left( \frac{n^2}{P^2/\log_2 7} \right) \]

\[ \Omega \left( \frac{n^2}{P^2/\log_2 8} \right) \]

Memory independent bound [B., Demmel, Holtz, Lipshitz, Schwartz 12]
Communication lower bounds for matrix multiplication

Algorithms attaining these bounds?

Strassen:

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

Classical (cubic):

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

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

\[ \Omega \left( \frac{n^2}{P^2/\log_2 7} \right) \]

\[ \Omega \left( \frac{n^2}{P^2/\log_2 8} \right) \]

\[ n = \text{matrix dimension}, \ M = \text{fast/local memory size}, \ P = \text{number of processors} \]
Communication lower bounds for matrix multiplication

Algorithms attaining these bounds?

Strassen:

Sequential

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

Distributed

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

\[ \Omega \left( \frac{n^2}{P^2/\log_2 7} \right) \]

Classical (cubic):

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

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

\[ \Omega \left( \frac{n^2}{P^2/\log_2 8} \right) \]

\( n = \) matrix dimension, \( M = \) fast/local memory size, \( P = \) number of processors
Communication lower bounds for matrix multiplication

Algorithms attaining these bounds?

**Strassen:**

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

**Classical (cubic):**

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

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

Our new algorithm

\[
\Omega \left( \frac{n^2}{P^2/\log_2 8} \right)
\]

\(n = \) 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

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

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

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

1 Motivation

2 Lower Bounds

3 Algorithms

4 Performance

5 Practical Considerations
Simple “2D” Classical Algorithm

Here’s the basic communication pattern for the classical “2D” algorithm:

A

B

C
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

<table>
<thead>
<tr>
<th></th>
<th>Flops</th>
<th>Bandwidth Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Strassen</strong></td>
<td><strong>Lower Bound</strong></td>
<td>$\frac{n^{\log_2 7}}{P}$</td>
</tr>
<tr>
<td></td>
<td>2D-Strassen</td>
<td>$\frac{n^{\log_2 7}}{P^{(\log_2 7-1)/2}}$</td>
</tr>
<tr>
<td></td>
<td>Strassen-2D</td>
<td>$\left( \frac{7}{8} \right)^{\ell} \frac{n^3}{P}$</td>
</tr>
<tr>
<td></td>
<td>CAPS</td>
<td>$\frac{n^{\log_2 7}}{P}$</td>
</tr>
<tr>
<td><strong>Classical</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Asymptotic costs analysis

<table>
<thead>
<tr>
<th>Method</th>
<th>Flops</th>
<th>Bandwidth Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lower Bound</strong></td>
<td><strong>Flops</strong></td>
<td><strong>Bandwidth Cost</strong></td>
</tr>
<tr>
<td>Strassen</td>
<td>$\frac{n \log_2 7}{P}$</td>
<td>$\max \left{ \frac{n \log_2 7}{PM^{(\log_2 7)/2-1}}, \frac{n^2}{P^{2/\log_2 7}} \right}$</td>
</tr>
<tr>
<td>2D-Strassen</td>
<td>$\frac{n \log_2 7}{P(\log_2 7-1)/2}$</td>
<td>$\frac{n^2}{P^{1/2}}$</td>
</tr>
<tr>
<td>Strassen-2D</td>
<td>$(\frac{7}{8})^\ell \frac{n^3}{P}$</td>
<td>$(\frac{7}{4})^\ell \frac{n^2}{P^{1/2}}$</td>
</tr>
<tr>
<td>CAPS</td>
<td>$\frac{n \log_2 7}{P}$</td>
<td>$\max \left{ \frac{n \log_2 7}{PM^{(\log_2 7)/2-1}}, \frac{n^2}{P^{2/\log_2 7}} \right}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Classical</strong></th>
<th><strong>Lower Bound</strong></th>
<th><strong>Bandwidth Cost</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>2D</td>
<td>$\frac{n^3}{P}$</td>
<td>$\max \left{ \frac{n^3}{PM^{1/2}}, \frac{n^2}{P^{2/3}} \right}$</td>
</tr>
<tr>
<td>2.5D</td>
<td>$\frac{n^3}{P}$</td>
<td>$\max \left{ \frac{n^3}{PM^{1/2}}, \frac{n^2}{P^{2/3}} \right}$</td>
</tr>
</tbody>
</table>
Outline

1. Motivation
2. Lower Bounds
3. Algorithms
4. Performance
5. Practical Considerations
Performance of CAPS on large problems

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

![Performance Graph](chart.png)

- **CAPS**
- **2.5D-Strassen**
- **2D-Strassen**
- **Strassen-2D**
- **Strassen-Winograd peak**

Grey Ballard
Performance of CAPS on large problems

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

![Graph showing strong-scaling performance](image)

- **CAPS**
- 2.5D-Strassen
- 2D-Strassen
- Strassen-2D

Grey Ballard

Strassen-Winograd peak
Comparison of the parallel models with the algorithms in strong scaling of matrix dimension $n = 65,856$ on Intrepid.
Performance of CAPS on large problems

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

![Graph showing strong-scaling results for CAPS, 2.5D-Strassen, 2D-Strassen, Strassen-2D, and 2D methods. The graph plots effective performance as a fraction of peak against the number of cores. The strong-scaling range is indicated by a red shaded area.]
Performance of CAPS on small (comm-bound) problems

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

[Graphs showing strong-scaling results for Intrepid and Hopper with different algorithms: CAPS, 2.5D-Strassen, 2D-Strassen, and Strassen-2D.]
Outline

1 Motivation
2 Lower Bounds
3 Algorithms
4 Performance
5 Practical Considerations
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

![Graph showing max-norm error vs. number of Strassen steps with classical and actual lines.]

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

- 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
Thank You!

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

1. Performance: Model vs Actual
2. Time breakdown
3. DFS vs BFS
4. BFS on 7 Processors
5. Sequential Performance
6. Data Layout
7. Strassen-Winograd Algorithm
8. Actual vs Effective Performance
9. Small problem on Franklin
10. Big problem on Franklin
11. Diagonal Scaling
12. Open Problems
Efficiency at various numbers of Strassen steps, $n = 21952$, on 49 nodes (196 cores) of Intrepid.
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)
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.
BFS on 7 Processors

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

Grey Ballard
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.
<table>
<thead>
<tr>
<th>Column 0</th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
<th>Column 5</th>
<th>Column 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>05</td>
<td>06</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
</tr>
<tr>
<td>20</td>
<td>21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td>26</td>
</tr>
<tr>
<td>30</td>
<td>31</td>
<td>32</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
</tr>
<tr>
<td>40</td>
<td>41</td>
<td>42</td>
<td>43</td>
<td>44</td>
<td>45</td>
<td>46</td>
</tr>
<tr>
<td>50</td>
<td>51</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
<td>56</td>
</tr>
<tr>
<td>60</td>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
<td>65</td>
<td>66</td>
</tr>
</tbody>
</table>
\[
\begin{pmatrix}
C_{11} & C_{12} \\
C_{21} & C_{22}
\end{pmatrix} = C = A \cdot B = \begin{pmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{pmatrix} \cdot \begin{pmatrix}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{pmatrix}
\]

\[
S_0 = A_{11} \quad T_0 = B_{11} \quad Q_i = S_i \cdot T_i
S_1 = A_{12} \quad T_1 = B_{21} \quad U_1 = Q_i + Q_4
S_2 = A_{21} + A_{22} \quad T_2 = B_{12} + B_{11} \quad U_2 = U_1 + Q_5
S_3 = S_2 - A_{12} \quad T_3 = B_{22} - T_2 \quad U_3 = U_1 + Q_5
S_4 = A_{11} - A_{21} \quad T_4 = B_{22} - B_{12} \quad C_{11} = Q_1 + Q_2
S_5 = A_{12} + S_3 \quad T_5 = B_{22} \quad C_{12} = U_3 + Q_6
S_6 = A_{22} \quad T_6 = T_3 - B_{21} \quad C_{21} = U_2 - Q_7
C_{22} = U_2 + Q_3
\]
Time breakdown comparison between the parallel model and data on Intrepid. In each case the entire modeled execution time is normalized to 1.
Performance on Franklin for small problem

$n = 3136$ on Franklin
Performance of CAPS on large problem

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

![Graph showing the performance of various algorithms on different numbers of cores. The x-axis represents the number of cores, ranging from $2e2$ to $2e4$. The y-axis represents the effective performance, fraction of peak. The graph compares CAPS, 2.5D-Strassen, 2D-Strassen, Strassen-2D, and 2D algorithms. The strong-scaling range is indicated by a shaded area.]
Sequential recursive Strassen is communication optimal

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

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

- Solution is

\[
W(n, M) = O\left(\frac{n^{\omega_0}}{M^{\omega_0/2-1}}\right)
\]
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 $B' = B D^B$.
  - Use Strassen for the product $C' = A' B'$.
  - Unscale $C = (D^A)^{-1} C' (D^B)^{-1}$.
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 $C = (D^A)^{-1} C' (D^B)^{-1}$.

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 $C = A'B'$.
Stability: easy case

\[
\frac{\max_j |\hat{C}_{ij} - C_{ij}|}{(|A| \cdot |B|)_{ij}}
\]

\[
\begin{pmatrix}
1 & 1 \\
1 & 1 \\
\end{pmatrix}
\cdot
\begin{pmatrix}
1 & 1 \\
1 & 1 \\
\end{pmatrix}
\]
Stability: more interesting case

\[
\max_{ij} \left| \hat{C}_{ij} - C_{ij} \right| \frac{1}{(|A| \cdot |B|)_{ij}}
\]

\[
\left( \begin{array}{cc} \epsilon & \epsilon \\ 1 & 1 \end{array} \right) \cdot \left( \begin{array}{cc} 1 & \epsilon^{-1} \\ 1 & 1 \end{array} \right)
\]
Stability: problems scaling can’t fix

\[ \left\| \hat{C}_{ij} - C_{ij} \right\|_{ij} \leq \epsilon \cdot \max_{ij} \left( |A||B| \right)_{ij} \]

\[ \left( \begin{array}{cc} 1 & \epsilon^{-1} \\ \epsilon^{-1} & 1 \end{array} \right) \cdot \left( \begin{array}{cc} 1 & \epsilon^{-1} \\ 1 & 1 \end{array} \right) \]
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), $n = 65,856$. 

![Graph showing strong-scaling performance](image)
Comparison of the parallel models with the algorithms in strong scaling of matrix dimension $n = 65,856$ on Intrepid.
Extra slides

1. Performance: Model vs Actual
2. Time breakdown
3. DFS vs BFS
4. BFS on 7 Processors
5. Sequential Performance
6. Data Layout
7. Strassen-Winograd Algorithm
8. Actual vs Effective Performance
9. Small problem on Franklin
10. Big problem on Franklin
11. Diagonal Scaling
12. Open Problems