Communication Optimal Distributed Memory Strassen Implementation

Ben Lipshitz
CS 294
Fall 2011
Outline

• Background
  – Communication bounds for Strassen-like
  – Strassen-Winograd summary

• Implementation Details
  – Data layout
  – Communication pattern
  – Communication cost

• Performance
  – Actual performance
  – Performance model
Recall: Strassen’s Fast Matrix Multiplication

[Strassen 69]

- Compute 2 x 2 matrix multiplication using only 7 multiplications (instead of 8).
- Apply recursively (block-wise)

\[
\begin{array}{c|c}
C_{11} & C_{12} \\
\hline
C_{21} & C_{22} \\
\end{array}
\begin{array}{c|c}
A_{11} & A_{12} \\
\hline
A_{21} & A_{22} \\
\end{array}
\cdot
\begin{array}{c|c}
B_{11} & B_{12} \\
\hline
B_{21} & B_{22} \\
\end{array}
\]

\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*}

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

\[T(n) = \Theta(n^{\log_2 7})\]

(T is the number of flops)
Strassen-like algorithms

• Compute $n_0 \times n_0$ matrix multiplication using only $n_0^{\omega_0}$ multiplications (instead of $n_0^3$).

• Apply recursively (block-wise)

$\omega_0 \approx$

2.81 [Strassen 69] works fast in practice.

2.79 [Pan 78]

2.78 [Bini 79]

Algorithms below this line aren’t explicit

2.55 [Schönhage 81]

2.50 [Pan Romani, Coppersmith Winograd 84]

2.48 [Strassen 87]

2.38 [Coppersmith Winograd 90]

2.38 [Cohn Kleinberg Szegedy Umans 05] Group-theoretic approach

\[
T(n) = n_0^{\omega_0} \cdot T(n/n_0) + O(n^2)
\]

\[
T(n) = \Theta(n^{\omega_0})
\]
New lower bound for Strassen’s fast matrix multiplication

The communication bandwidth lower bound is

\[
\begin{align*}
\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} M\right) & \quad \Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\omega_0} M\right) & \quad \Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} M\right) \\
\Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 7} \frac{M}{P}\right) & \quad \Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\omega_0} \frac{M}{P}\right) & \quad \Omega\left(\left(\frac{n}{\sqrt{M}}\right)^{\log_2 8} \frac{M}{P}\right)
\end{align*}
\]

The parallel lower bounds applies to algorithms using:
Minimal required memory: \( M = \Theta(n^2/P) \)
Extra available memory: \( M = \Theta(c \cdot n^2/P) \)
Run Strassen’s algorithm recursively. When blocks are small enough, work in local memory. Assuming a message length is $M$, the count of messages (latency analysis) is:

$$L(n, M) = \begin{cases} 
7L \left( \frac{n}{2}, M \right) + \Theta \left( \frac{n^2}{M} \right) & \text{if } 3.5n^2 > M \\
1 & \text{Otherwise}
\end{cases}$$

Therefore

$$L(n, M) = \Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\omega_0} \right)$$

and

$$BW(n, M) = \Theta \left( \left( \frac{n}{\sqrt{M}} \right)^{\omega_0} M \right)$$

These match the lower bounds.
# Parallel Algorithms and Implementations

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Flops</th>
<th>Local memory requirement</th>
<th>BW</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D classic (minimal memory) Cannon/SUMMA</td>
<td>( c = 1 )</td>
<td>( \frac{n^3}{P} )</td>
<td>( \frac{n^2}{P} )</td>
<td>( \frac{n^2}{\sqrt{P}} )</td>
</tr>
<tr>
<td>3D classic (maximal memory)</td>
<td>( c = P^{1/3} )</td>
<td>( \frac{n^3}{P} )</td>
<td>( \frac{n^2}{P^{2/3}} )</td>
<td>( \frac{n^2}{P^{2/3}} )</td>
</tr>
<tr>
<td>2.5D classic (extra memory) [Demmel Solomonik 11] [Grayson Shah van de Geijn 95] [Nguyen Lavalallee Bui Trung 05]</td>
<td>( 1 \leq c \leq P^{1/3} = P^{1-2/3} )</td>
<td>( \frac{n^3}{P} )</td>
<td>( c \cdot \frac{n^2}{P} )</td>
<td>( \frac{n^2}{\sqrt{cP}} )</td>
</tr>
<tr>
<td>New algorithms</td>
<td>( c = 1 )</td>
<td>( \frac{n^\omega_0}{P} )</td>
<td>( \frac{n^2}{P} )</td>
<td>( \frac{n^2}{P^{2-\omega_0/2}} )</td>
</tr>
<tr>
<td></td>
<td>( c = P^{1-2/\omega_0} )</td>
<td>( \frac{n^\omega_0}{P} )</td>
<td>( \frac{n^2}{P^{2/\omega_0}} )</td>
<td>( \frac{n^2}{P^{2/\omega_0}} )</td>
</tr>
<tr>
<td></td>
<td>( 1 \leq c \leq P^{1-2/\omega_0} )</td>
<td>( \frac{n^\omega_0}{P} )</td>
<td>( c \cdot \frac{n^2}{P} )</td>
<td>( c^{1-\omega_0/2} \cdot \frac{n^2}{P^{2-\omega_0/2}} )</td>
</tr>
</tbody>
</table>

BW and L: stated up to a polylog \( P \) factor. Other: up to a constant factor
Green means optimal (up to polylog \( P \) factor). Red means not optimal
\( \omega_0 = \log_2 7 \) for Strassen’s algorithm.
The upper bound on \( c \) to be derived shortly. It is stricter for faster algorithms.
Strassen-Winograd Algorithm

C = A•B

A =

\[
\begin{array}{cc}
A_{11} & A_{12} \\
A_{21} & A_{22}
\end{array}
\]

Q_i = T_i • S_i

U_1 = Q_1 + Q_4
U_2 = U_1 + Q_5
U_3 = U_1 + Q_3
C_{11} = Q_1 + Q_2
C_{12} = U_3 + Q_6
C_{21} = U_2 - Q_7
C_{22} = U_2 + Q_3

T_0 = A_{11}
T_1 = A_{12}
T_2 = A_{21} + A_{22}
T_3 = T_2 - A_{12}
T_4 = A_{11} - A_{21}
T_5 = A_{12} + T_3
T_6 = A_{22}

S_0 = B_{11}
S_1 = B_{21}
S_2 = B_{12} + B_{11}
S_3 = B_{22} - S_2
S_4 = B_{22} - B_{12}
S_5 = B_{22}
S_6 = S_3 - B_{21}
BFS vs. DFS

**BFS**

- Runs all 7 multiplies in parallel
- Each on \( \frac{p}{7} \) processors
- Each needs \( \frac{7}{4} \) as much mem

**DFS**

- Runs all 7 multiplies sequentially
- Each on all \( p \) processors
- Each needs \( \frac{1}{4} \) as much mem

\[
\text{If enoughMemory and } \log_7 p \geq 1 \\
\text{then BFS step} \\
\text{else DFS step} \\
\text{end if}
\]
Perfect Strong Scaling Range

• Classical 2.5d [Demmel Solomonik 2011] scales perfectly:
  – From $c=1$ to $c=p^{1/3}$
  – From $p_{\text{min}}$ to $p_{\text{max}}=p_{\text{min}}^{1.5}$ processors because each processor has memory $M$.
  – Above $c=p^{1/3}$, no useful use of extra memory.
Perfect Strong Scaling Range

• Assume $p$ is a power of 7

• Strassen “2.5d”, maximum memory case:
  – Want to get down to 1 processor as quickly as possible to maximize memory usage.
  – That means taking $k = \log_7 p_{\text{max}}$ BFS steps

Condition to fill the memory:

$$ \frac{4n^2}{p_{\text{min}}} = M = \left( \frac{7}{4} \right)^k \frac{4n^2}{p_{\text{max}}} $$

Solution:

From $p=p_{\min}$ to $p_{\max} = p_{\min}^{\omega_0/2} \approx p_{\min}^{1.4}$

From $c=1$ to $c_{\max} = p^{1-2/\omega_0} \approx p^{0.29}$
Outline

• Background
  – Communication bounds for Strassen-like
  – Strassen-Winograd summary

• Implementation Details
  – Data layout
  – Communication pattern
  – Communication cost

• Performance
  – Actual performance
  – Performance model
Data Layout

Top level: І-Morton recursive layout

Bottom level: block cyclic layout

Orange squares are what processor 0 owns
Data Layout

- \( s \) (=number of Strassen steps) levels of Η-Morton, followed by block cyclic
- Blocks are square when \( p \) is on even power of 7 and rectangular (portrait) when \( p \) is an odd power of 7.
- Requires \( n \) to be a multiple of \( 2^s 7^{\lfloor \log_7 p \rfloor} \)
- Works with any block size; best performance when block size is 1
- Easily convertible to/from block cyclic with a multiple of \( 2^s 7^{\lfloor \log_7 p \rfloor} \) blocks
Our data layout

Block cyclic layout with a multiple of $2^{s_7 \left\lceil \log_7 p \right\rceil}$ blocks

local rearrangement
Local Data Layout

• $s$ (=number of Strassen steps) levels of $\Pi$-Morton.

• Then blocks in column-major order.

• Entries in each block in column-major order.
Strassen-Winograd Algorithm

\[ C = A \cdot B \]

\[ A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \]

\[ Q_i = T_i \cdot S_i \]

\[ U_1 = Q_1 + Q_4 \]
\[ U_2 = U_1 + Q_5 \]
\[ U_3 = U_1 + Q_3 \]
\[ C_{11} = Q_1 + Q_2 \]
\[ C_{12} = U_3 + Q_6 \]
\[ C_{21} = U_2 - Q_7 \]
\[ C_{22} = U_2 + Q_3 \]

\[ T_0 = A_{11} \]
\[ T_1 = A_{12} \]
\[ T_2 = A_{21} + A_{22} \]
\[ T_3 = T_2 - A_{12} \]
\[ T_4 = A_{11} - A_{21} \]
\[ T_5 = A_{12} + T_3 \]
\[ T_6 = A_{22} \]

\[ S_0 = B_{11} \]
\[ S_1 = B_{21} \]
\[ S_2 = B_{12} + B_{11} \]
\[ S_3 = B_{22} - S_2 \]
\[ S_4 = B_{22} - B_{12} \]
\[ S_5 = B_{22} \]
\[ S_6 = S_3 - B_{21} \]
BFS Step $p=7 \rightarrow p=1$, $s=1$
BFS $p=49 \rightarrow p=7, s=2$

local additions

communication

parallel multiplies on 7 processors
DFS, $p=7$, $s=2$

local additions

sequential multiplies on 7 processors
Analyzing communication cost

- Cost of a DFS step: 0
- Cost of a BFS step from \( n \times n \) on \( p \) processors to \( (n/2) \times (n/2) \) on \( p/7 \) processors:
  - 7-way all-to-all, each processor is involved in one
  - 6 messages sent and received
  - \( n^2/(4p) \) doubles in each
  - \( 12n^2/p \) bytes sent+received. \( 12=1/4*(8\text{bytes/double})*(3 \text{ for 2 encodings and 1 decoding})*(2 \text{ for send+receive}) \)
- Total \( k=\log_7 p \) BFS steps, choose order
Maximum Memory Case ("3d")

- Take k BFS steps followed by s-k local DFS steps
- Messages: $36 \log_7 p$, matches lower bound up to $k = \log_7 p$ factor
- Bytes: 
  \[
  \frac{12n^2}{p} \sum_{i=0}^{k-1} \left( \frac{7}{4} \right)^i < \frac{12n^2}{p} \left( \frac{7}{4} \right)^k \frac{4}{3} = \frac{16n^2}{p^2/\omega_0}
  \]
  matches lower bound up to constant factor. (for $k=4$, $16 \rightarrow 14.3$)
- Imperfect strong scaling beyond
Simple Interleaved Case ("2.5d")

• Have factor c extra memory. Plan
  - Do $s_1 = \log_4(c_{\text{max}}/c)$ DFS steps (no communication)
  - Do $k = \log_7 p$ BFS steps
  - Finish with local DFS steps (no communication)

\[
\text{Words} = 7^{s_1} \text{Words} \left( \frac{n}{2^{s_1}}, 3d \right)
\]

\[
7^{s_1} = \left( \frac{c_{\text{max}}}{c} \right)^{\omega_0/2}, \quad 2^{s_1} = \sqrt{\frac{c_{\text{max}}}{2}}, \quad c_{\text{max}} = p^{1-2/\omega_0}
\]

\[
\text{Words} < \frac{16n^2}{p^2-\omega_0/2} c^{1-\omega_0/2}
\]

Matches lower bound up to a constant factor
General Interleaved Case

- The code does dynamic interleaving based on available memory.
- This should do at least as well as the simple interleaving model, which is one of its options.

width signifies communication cost
Communication vs memory usage

Possible runs on $p=2401, n=175616$
Communication vs memory usage

Good choices only

Bytes sent/received vs Memory usage MB
Communication vs memory usage

Good choices only, memory not too full

Possible executions

C M^\left(1-\frac{w}{2}\right)
Outline

• Background
  – Communication bounds for Strassen-like
  – Strassen-Winograd summary

• Implementation Details
  – Data layout
  – Communication pattern
  – Communication cost

• Performance
  – Actual performance
  – Performance model
Performance on $7^k$ Processes

Strong Scaling of CA Strassen on Franklin (Cray XT4)
Performance on “arbitrary” # of processors

Strong Scaling of CA Strassen on Franklin (Cray XT4)
Roofline model for Strassen-Winograd

- Recurrence relation:
  - \( \text{flops}(2n) = 7 \text{flops}(n) + 15an^2 \)
  - There are \( 15n^2 \) additions, but they have a different cost than the multiplies, hence the parameter \( a \)
  - Ignore cache performance: \( a=2 \)
  - Experimentally: \( a=127 \) for \( C=A+B \) all in different memory locations on Franklin.
  - Really only 33, not 45, matrices read/written for the 15 adds, so let’s use \( a=93 \).
Roofline model for Strassen-Winograd

• Recurrence relation with a=93:
  – $\text{flops}(2n) = 7 \text{flops}(n) + 1395n^2$ (strassen step)
  – $\text{flops}(2n) = 2(2n)^3 = 8 \text{flops}(n)$ (base case)

• Crossover point: $2n=1395$. Conveniently close to the threshold for getting peak on dgemm.

• For $n=40320$, we should take 5 strassen steps, expect effective performance of $1.66*\text{peak}$, or $61.2 \text{ Gflop/s}$ for Franklin.
Open Problems / Future Work

• Scheduling Strassen BFS steps for
  – p not a power of 7
  – Shared memory Strassen
  – Heterogeneous machine
Open Problems / Future Work

• Better heuristic for choosing BFS/DFS interleaving order
• Optimal embedding into network topology to minimize hops / contention
• Use dag structure to lower constant in communication. New data layout?
• Reduce communication with redundant additions
• Related: Strassen vs. Strassen-Winograd
• Overlap communication with additions
• Incorporate Strassen into ScaLAPACK
• Consider other fast algorithms, eg. Pan and Bini
• Apply to the rest of dense linear algebra, and beyond.