Algorithms: Matrix-Matrix Multiplication
Simple Algorithm

• A X B => C, matrices of size NxN, using p = q^2 procs
• Start with a 2D (block) decomposition of A, B and C
  • Each process gets a (N/q)x(N/q) block
Simple Algorithm

- A $\times$ B $\Rightarrow$ C, matrices of size $N\times N$, using $P = q^2$ procs
- Start with a 2D (block) decomposition of A, B and C
  - Each process gets a $(N/q) \times (N/q)$ block
- Each processsor broadcasts it’s A piece along its row, and its B piece along its column
- Use sub-communicators for this purpose, as we learned
Simple Algorithm: Analysis

• Isoefficiency
  • Using communication volume (i.e. # of bytes) as the communication cost
    • Assume, for now, broadcasting M bytes takes $O(M)$ time
  • Communication: $2\sqrt{p} \frac{N^2}{p}$
  • Computation: $\frac{N^3}{p}$
  • $\frac{\text{communication}}{\text{computation}} = \frac{2\sqrt{p}N^2}{N^3} = \frac{2\sqrt{p}}{N} = k$ \hspace{1cm} (1)
  • $W = N^3$; i.e $N = W^{1/3}$
  • Substituting for $N$ in (1): $\frac{2\sqrt{p}}{W^{1/3}} = k$
  • So, $W = \frac{8}{k^3} p^{1.5}$ So, i.e. $W \propto p^{1.5}$
Simple Algorithm: Analysis

• Isoefficiency
  • Using communication volume (i.e. # of bytes) as the communication cost
    • Assume, for now, broadcasting M bytes takes O(M) time
  • O(p^{1.5}), which is ok

• What is the problem with this algorithm?

• Memory on each processor increases to (q-1) times its original value.
  • q-1 blocks of A and q-1 blocks of B!
Cannon’s Matrix Multiplication Algorithm

• Idea is to keep only one tile of B and C on every processor at any step
• Tile movements are like a well-choreographed dance
• Recall : $q = \sqrt{p}$

• Phase 1:
  • Shift each tile of $A, A_{i,j}$, leftwards by $i$ steps (i.e. send it to $P_i, (j-i)\%q$)
  • Shift each tile of $B, B_{i,j}$, upwards by $j$ steps

• Phase 2:
  • Repeat $q$ times:
    • Multiply available tiles and add to the local $C$ tile
    • Shift $A$ tile leftwards and $B$ tile upwards
C[0,0] =
\[ C_{0,0} = A_{0,0} \times B_{0,0} \]
\[
\begin{align*}
C(0,0) &= A(0,0) \times B(0,0) + A(0,1) \times B(1,0) \\
\end{align*}
\]
C[0,0] = A[0,0]*B[0,0] + A[0,1]*B[1,0] + A[0,2]*B[2,0]
Cannon’s Algorithm
<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A(0,0)</td>
<td>A(0,1)</td>
<td>A(0,2)</td>
</tr>
<tr>
<td>1</td>
<td>A(1,0)</td>
<td>A(1,1)</td>
<td>A(1,2)</td>
</tr>
<tr>
<td>2</td>
<td>A(2,0)</td>
<td>A(2,1)</td>
<td>A(2,2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>B(0,0)</td>
<td>B(0,1)</td>
<td>B(0,2)</td>
</tr>
<tr>
<td>1</td>
<td>B(1,0)</td>
<td>B(1,1)</td>
<td>B(1,2)</td>
</tr>
<tr>
<td>2</td>
<td>B(2,0)</td>
<td>B(2,1)</td>
<td>B(2,2)</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>---</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>0</td>
<td>A(0,0)</td>
<td>A(0,1)</td>
<td>A(0,2)</td>
</tr>
<tr>
<td>1</td>
<td>A(1,0)</td>
<td>A(1,1)</td>
<td>A(1,2)</td>
</tr>
<tr>
<td>2</td>
<td>A(2,0)</td>
<td>A(2,1)</td>
<td>A(2,2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>B(0,0)</td>
<td>B(0,1)</td>
<td>B(0,2)</td>
</tr>
<tr>
<td>1</td>
<td>B(1,0)</td>
<td>B(1,1)</td>
<td>B(1,2)</td>
</tr>
<tr>
<td>2</td>
<td>B(2,0)</td>
<td>B(2,1)</td>
<td>B(2,2)</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>0</td>
<td>A(0,0)</td>
<td>A(0,1)</td>
<td>A(0,2)</td>
</tr>
<tr>
<td>1</td>
<td>A(1,0)</td>
<td>A(1,2)</td>
<td>A(1,0)</td>
</tr>
<tr>
<td>2</td>
<td>A(2,0)</td>
<td>A(2,1)</td>
<td>A(2,2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>B(0,0)</td>
<td>B(0,1)</td>
<td>B(0,2)</td>
</tr>
<tr>
<td>1</td>
<td>B(1,0)</td>
<td>B(1,1)</td>
<td>B(1,2)</td>
</tr>
<tr>
<td>2</td>
<td>B(2,0)</td>
<td>B(2,1)</td>
<td>B(2,2)</td>
</tr>
</tbody>
</table>
A(0,0)  A(0,1)  A(0,2)
A(1,1)  A(1,2)  A(1,0)
A(2,2)  A(2,0)  A(2,1)

B(0,0)  B(1,1)  B(2,2)
B(1,0)  B(2,1)  B(0,2)
B(2,0)  B(0,1)  B(1,2)

C[1,1] = 

C(0,0)  C(0,1)  C(0,2)
C(1,0)  C(1,1)  C(1,2)
C(2,0)  C(2,1)  C(2,2)
\[
C_{1,1} = A_{1,2} \times B_{2,1} +
\]

\[
\begin{array}{ccc}
A_{0,0} & A_{0,1} & A_{0,2} \\
A_{1,1} & A_{1,2} & A_{1,0} \\
A_{2,2} & A_{2,0} & A_{2,1} \\
\end{array}
\begin{array}{ccc}
B_{0,0} & B_{1,1} & B_{2,2} \\
B_{1,0} & B_{2,1} & B_{0,2} \\
B_{2,0} & B_{0,1} & B_{1,2} \\
\end{array}
\]

\[
\begin{array}{ccc}
C_{0,0} & C_{0,1} & C_{0,2} \\
C_{1,0} & C_{1,1} & C_{1,2} \\
C_{2,0} & C_{2,1} & C_{2,2} \\
\end{array}
\]

L.V.Kale
$C[1,1] = A[1,2]*B[2,1] + C(0,0)$
\[ C_{1,1} = A_{1,2}B_{2,1} + A_{1,0}B_{0,1} + \]

\[
\begin{align*}
\begin{array}{ccc}
A_{0,1} & A_{0,2} & A_{0,0} \\
A_{1,2} & A_{1,0} & A_{1,1} \\
A_{2,0} & A_{2,1} & A_{2,2} \\
\end{array}
\end{align*}
\begin{align*}
\begin{array}{ccc}
B_{1,0} & B_{1,1} & B_{1,2} \\
B_{2,0} & B_{0,1} & B_{2,2} \\
B_{0,0} & B_{1,1} & B_{2,2} \\
\end{array}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{ccc}
C_{0,0} & C_{0,1} & C_{0,2} \\
C_{1,0} & C_{1,1} & C_{1,2} \\
C_{2,0} & C_{2,1} & C_{2,2} \\
\end{array}
\end{align*}
\]

L.V.Kale
\[ C_{1,1} = A_{1,2}B_{2,1} + A_{1,0}B_{0,1} + A_{1,1}B_{1,1} \]
Cannon’s Algorithm: analysis

• Same amount of communication
  • Think about what data comes in to a processor
  • So, same isoefﬁciency: \( O(p^{1.5}) \)
Johnson’s 3D Matrix Multiplication

• How can we reduce communication?
• The matrix multiplication (sequential) has 3 nested loops, and we tiled only the outer 2.
  • What if we tile the 3\textsuperscript{rd} (k) loop as well?
• What does that mean in distributed memory context?
  • Note that the k loop is involves a reduction

• Basic idea:
  • organize processes in a 3D cube of cubes,
  • distribute A on one face of the process cube
  • Distribute B on another phase of the process cube
  • Collect C on the 3rd phase of the process cube
3D Matrix Multiplication

Tile size = \( \frac{N}{p^{1/3}} \times \frac{N}{p^{1/3}} \)
3D Matrix Multiplication: Analysis

- Tile size = $\frac{N}{P^{1/3}} \times \frac{N}{P^{1/3}}$

- Assuming pipelined broadcast, and ignoring per-message cost (because messages are large), each process receives 2 tiles:
  - Communication volume proportional to $\frac{N \times N}{P^{2/3}}$
  - Computation, as always: $\frac{N^3}{p}$

- Exercise: calculate isoefficiency

- Memory pressure:
  - More than Cannon’s but less than the original (multicast based) version
  - Each tile is duplicated in $P^{1/3}$ places (as opposed to $\sqrt{P}$ places in
Another new Algorithm: 2.5D matrix mpy

• Optional reading for interested students

• Edgar Solomonic (now at UIUC), and J. Demmel (Berkeley)
  • Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms, 2011