Objective

- To learn more about the analysis of tiled convolution/stencil algorithms

A Small 1D Example

- output and input tiles for block 1
- For MASK_WIDTH of 5, each block loads $8 + (5 – 1) = 12$ elements (12 memory loads)

Each Output Uses MASK_WIDTH Inputs

- $P[8]$ uses $N[6], N[7], N[8], N[9], N[10]$
- $P[15]$ uses $N[13], N[14], N[15], N[16], N[17]$

Total of $8 * 5$ values from tile used for the output.
A simple way to calculate tiling benefit

- \((8+5-1) = 12\) elements loaded
- \(8 \times 5\) global memory accesses replaced by shared memory accesses
- Bandwidth reduction of \(40/12 = 3.3\)

In General, for 1D

- Load \(TILE\_WIDTH + MASK\_WIDTH - 1\) elements.
- Replace \(TILE\_WIDTH \times MASK\_WIDTH\) global memory accesses with shared memory.
- Bandwidth reduction of \((TILE\_SIZE \times MASK\_WIDTH) / (TILE\_SIZE + MASK\_WIDTH - 1)\)

Another Way to Look at Reuse

- Each access \(tile\) replaces an access to \(N\).
- The total number of global memory accesses (to the \((8+5-1) = 12\) \(N\) elements) replaced by shared memory accesses is
  \[
  1 + 2 + 3 + 4 + 5 \times (8-5+1) + 4 + 3 + 2 + 1
  = 10 + 20 + 10
  = 40
  
  There are 12 \(N\) elements, so the average reduction is
  \(40/12 = 3.3\)
Ghost elements change ratios

- For a boundary tile, we load
  \[ \text{TILE}_\text{WIDTH} + (\text{MASK}_\text{WIDTH}-1)/2 \text{ elements} \]
  - 10 in our example of \( \text{TILE}_\text{WIDTH} \) of 8 and \( \text{MASK}_\text{WIDTH} \) of 5

- Computing boundary elements do not access global memory for ghost cells
  - Total accesses is \( 6 \times 5 + 4 + 3 = 37 \) accesses
    (when computing the \( P \) elements)

  The reduction is \( 37/10 = 3.7 \)

Bandwidth Reduction for 1D

- The reduction is
  \[
  \frac{\text{TILE}_\text{SIZE} \times \text{MASK}_\text{WIDTH}}{\text{TILE}_\text{SIZE} + \text{MASK}_\text{WIDTH} - 1}
  \]

<table>
<thead>
<tr>
<th>TILE_WIDTH</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>128</th>
<th>256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reduction</td>
<td>4.0</td>
<td>4.4</td>
<td>4.7</td>
<td>4.9</td>
<td>4.9</td>
</tr>
<tr>
<td>Mask_Width = 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reduction</td>
<td>6.0</td>
<td>7.2</td>
<td>8.0</td>
<td>8.5</td>
<td>8.7</td>
</tr>
<tr>
<td>Mask_Width = 9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Review: Parallelization of Tile Load

- Loading input tile requires \((8+5-1)^2 = 144\) reads.
- Calculation of each output requires \(5^2 = 25\) input elements.
- \(8 \times 8 \times 25 = 1,600\) global memory accesses for computing output tile converted to shared memory accesses.
- Bandwidth reduction of \(1,600/144 = 11.1 \times\)
In General

- \((TILE\_WIDTH+MASK\_WIDTH-1)^2\) elements need to be loaded from \(N\) into shared memory
- The calculation of each \(P\) element needs to access \(MASK\_WIDTH^2\) elements of \(N\)
- \(TILE\_WIDTH^2 \times MASK\_WIDTH^2\) global memory accesses converted into shared memory accesses

- Bandwidth reduction of \(TILE\_WIDTH^2 \times MASK\_WIDTH^2 / (TILE\_WIDTH+MASK\_WIDTH-1)^2\)

Bandwidth Reduction for 2D

- The reduction is \(TILE\_WIDTH^2 \times MASK\_WIDTH^2 / (TILE\_WIDTH+MASK\_WIDTH-1)^2\)

<table>
<thead>
<tr>
<th>TILE_WIDTH</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reduction</td>
<td>Mask_Width = 5</td>
<td>11.1</td>
<td>16</td>
<td>19.7</td>
</tr>
<tr>
<td>Reduction</td>
<td>Mask_Width = 9</td>
<td>20.3</td>
<td>36</td>
<td>51.8</td>
</tr>
</tbody>
</table>

2B/FLOP for Untiled Convolution

How much global memory per FLOP in untiled convolution?

In untiled convolution,
- each value from \(N\) (4B from global memory)
- is multiplied by a value from \(M\) (4B from constant cache, 1 FLOP),
- then added to a running sum (1 FLOP).
That gives 2B / FLOP.

Full Use of Compute Requires 13.3× Reuse

Recall our reuse discussion from matrix multiply:
- 1,000 GFLOP/s for GPU from ~2010, and
- 150 GB/s memory bandwidth.

Dividing memory bandwidth by \(2B/FLOP\),
\[
\frac{150 \, \text{GB/s}}{2 \, \text{B/FLOP}} = 75 \, \text{GFLOP/s} = 7.50\% \text{ of peak.}
\]

Need at least \(100/7.50 = 13.3\times\) reuse to make full use of compute resources.
In 2020, Need 52.1× Reuse

That was 2010.

In 2020, the GRID K520 (remember MP0?) offers
• nearly 5,000 GFLOP/s, but only
• 192 GB/s memory bandwidth.

Dividing memory bandwidth by 2B/FLOP,
\[
\frac{192 \text{ GB/s}}{2 \text{ B/FLOP}} = 96 \text{ GFLOP/s} = 1.92\% \text{ of peak.}
\]

Need at least 100/1.92 = 52.1× reuse
to make full use of compute resources.

Need Really Big Mask to Balance Resources

Let’s make another table: % of peak compute
• for 1D tiled convolution,
• with TILE_WIDTH 1024.

<table>
<thead>
<tr>
<th>MASK_WIDTH</th>
<th>2010</th>
<th>2020</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>37%</td>
<td>9.6%</td>
</tr>
<tr>
<td>9</td>
<td>67%</td>
<td>17%</td>
</tr>
<tr>
<td>15</td>
<td>100%</td>
<td>28%</td>
</tr>
<tr>
<td>55</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Food for Thought

• Ratios are different for tiles on boundaries.
• More importantly,
  – Each thread loads 4B to shared memory.
  – 2,048 threads load only 8kB.
  – Shared memory is usually 64kB or larger.
  – What can one do with the rest?

Improved approach left as homework.
(For example, can raise MW=7 from 67% to 81%).
ANY MORE QUESTIONS?
READ CHAPTER 7