PARALLEL IMAGE FILTERING

Presented by John Hunter
Problem

Apply an image filter using parallel processing

• Averaging Filter
• Modifying Sequential Strategy
• Performing Parallel Matrix Multiplication

https://studynewyork.us/schools/university-at-buffalo-suny/
Typical Sequential Strategy

1. Apply Padding
2. Creating a kernel
3. Begin at row 0 and column 0
4. Apply the kernel
5. Divide by the kernel size
6. Set the pixel
7. Move to the next location
8. Repeat

\[(a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4 + a_5x_5 + a_6x_6 + a_7x_7 + a_8x_8 + a_9x_9) \times \frac{1}{9} = 171\]
Modified Sequential Strategy

1. Apply Padding
2. Slightly Alter the kernel
3. Begin at row 0 and column 0
4. Perform matrix multiplication at the point.
5. Sum one column in the resulting matrix.
6. Set the pixel
7. Move to the next location
8. Repeat
Sequential Matrix Multiplication

Input: 2 N×N matrices A, B
Outputs: 1 N×N matrix C

For(x=0; x < N; x++)
    For(y=0; j < N; y++)
        For (i=0; i < N; i++)
            C[x][y] = C[x][y] + A[x][i] × B[i][y]
        EndFor
    EndFor
EndFor
Parallel Strategy

1. Apply Padding
2. Slightly Alter the kernel
3. Begin at row 0 and column 0
4. Perform matrix multiplication at the point in parallel.
5. Sum one column in the resulting matrix.
6. Set the pixel
7. Move to the next location
8. Repeat
Parallel Matrix Multiplication

Utilize Cannon’s Algorithm

Input: 2 N×N matrices A, B

Outputs: 1 N×N matrix C

1. Align P processors in a mesh of size sqrt(P) × sqrt(P)
2. Divide A and B into P sub-matrices
3. Assign each submatrices A_i and B_i to processor p_i
4. All processors p_i performs sequential matrix multiplication on A_i and B_i to create C_i
5. All processors p_i pass A_i horizontally to their neighboring processor p
6. All processors p_i pass B_i vertically to their neighboring processor p
7. All processors p_i performs sequential matrix multiplication on the newly acquired A_i and B_i and add the value to C_i
8. Repeat steps 5-7 until every processor p has seen all the data
Parallel Implementation Results
Kernel Size = 1,024 x 1,024

<table>
<thead>
<tr>
<th># of PEs</th>
<th>Data Per Processor</th>
<th>Single Matrix Multiply</th>
<th>Overall Image Filter Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>262,144</td>
<td>12.75s</td>
<td>336.21s</td>
</tr>
<tr>
<td>16</td>
<td>65,536</td>
<td>6.36s</td>
<td>176.33s</td>
</tr>
<tr>
<td>25</td>
<td>41,943</td>
<td>5.13s</td>
<td>145.4s</td>
</tr>
<tr>
<td>36</td>
<td>29,127</td>
<td>4.3s</td>
<td>122.43s</td>
</tr>
<tr>
<td>49</td>
<td>21,399</td>
<td>2.52s</td>
<td>80.32s</td>
</tr>
<tr>
<td>64</td>
<td>16,384</td>
<td>7.37s</td>
<td>201.25s</td>
</tr>
<tr>
<td>81</td>
<td>12,945</td>
<td>7.01s</td>
<td>191.25s</td>
</tr>
<tr>
<td>121</td>
<td>8,665</td>
<td>5.74s</td>
<td>158.5s</td>
</tr>
</tbody>
</table>

Tests run on a 5 x 5 size image

Parallel Processing Results
Parallel Implementation Results
Kernel Size = 4,096 x 4,096

<table>
<thead>
<tr>
<th># of PEs</th>
<th>Data Per Processor</th>
<th>Single Matrix Multiply</th>
<th>Overall Image Filter Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>4,194,303</td>
<td>447.47s</td>
<td>11,205s</td>
</tr>
<tr>
<td>16</td>
<td>1,048,575</td>
<td>144.69s</td>
<td>3,632s</td>
</tr>
<tr>
<td>25</td>
<td>671,088</td>
<td>110.91s</td>
<td>2,785s</td>
</tr>
<tr>
<td>36</td>
<td>466,033</td>
<td>85.84s</td>
<td>2,162s</td>
</tr>
<tr>
<td>49</td>
<td>342,392</td>
<td>69.08s</td>
<td>1,743s</td>
</tr>
<tr>
<td>64</td>
<td>262,143</td>
<td>47.39s</td>
<td>1,202s</td>
</tr>
<tr>
<td>81</td>
<td>207,126</td>
<td>42.05s</td>
<td>1,067s</td>
</tr>
<tr>
<td>121</td>
<td>138,654</td>
<td>28.9s</td>
<td>740s</td>
</tr>
</tbody>
</table>

Tests run on a 5 x 5 size image

Parallel Processing Results

![Graph showing the relationship between the number of PEs and the overall image filter time.](chart.png)
### Parallel Implementation Results

**Kernel Size = 8,192 x 8,192**

<table>
<thead>
<tr>
<th># of PEs</th>
<th>Data Per Processor</th>
<th>Single Matrix Multiply</th>
<th>Overall Image Filter Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>16,777,216</td>
<td>3.590s</td>
<td>89.766s*</td>
</tr>
<tr>
<td>16</td>
<td>4,194,304</td>
<td>1.105s</td>
<td>27.642s</td>
</tr>
<tr>
<td>25</td>
<td>2,684,354</td>
<td>774s</td>
<td>19.366s</td>
</tr>
<tr>
<td>36</td>
<td>1,864,135</td>
<td>557s</td>
<td>13.941s</td>
</tr>
<tr>
<td>49</td>
<td>1,369,568</td>
<td>449s</td>
<td>11.241s</td>
</tr>
<tr>
<td>64</td>
<td>1,048,576</td>
<td>318s</td>
<td>7.967s</td>
</tr>
<tr>
<td>81</td>
<td>828,504</td>
<td>289s</td>
<td>7.242s</td>
</tr>
<tr>
<td>121</td>
<td>554,618</td>
<td>211s</td>
<td>5.275s</td>
</tr>
</tbody>
</table>

Tests run on a 5 x 5 size image

* Estimated due to long runtime

---

**Parallel Processing Results**

![Graph showing the relationship between number of PEs and time](image-url)
Overall Parallel Implementation Results

Parallel Processing Results On Number of PEs
Challenges

• Organizing processors into a mesh and assigning matrices
• Message passing and getting correct neighbors
• Determining the correct Slurm parameters
Citations

• [1] https://www.researchgate.net/figure/512-512-grayscale-image-Cameraman_fig1_326140507