IMAGE CONVOLUTION

CSE633: Parallel Algorithm

Zhi Wen Huang
Image Convolution

Input image

\[
\begin{array}{cccccccc}
10,30 & 11,00 & 12,20 & 13,00 & 14,00 & 15,00 & 16,00 \\
10,10 & 11,10 & 12,10 & 13,10 & 14,10 & 15,10 & 16,10 \\
10,30 & 11,30 & 12,30 & 13,30 & 14,30 & 15,30 & 16,30 \\
10,40 & 11,40 & 12,40 & 13,40 & 14,40 & 15,40 & 16,40 \\
10,50 & 11,50 & 12,50 & 13,50 & 14,50 & 15,50 & 16,50 \\
10,60 & 11,60 & 12,60 & 13,60 & 14,60 & 15,60 & 16,60 \\
\end{array}
\]

\[\times\]

Filter

\[
\begin{array}{ccc}
H(0,0) & H(1,0) & H(2,0) \\
H(0,1) & H(1,1) & H(2,1) \\
H(0,2) & H(1,2) & H(2,2) \\
\end{array}
\]

\[\Rightarrow\]

Output image

\[
\begin{array}{cccc}
C(0,0) & C(0,1) & C(0,2) \\
C(1,0) & C(1,1) & C(1,2) \\
C(2,0) & C(2,1) & C(2,2) \\
\end{array}
\]
Sequential Approach

- Matrix Size $2^{10} \times 2^{10}$
- Kernel Size 3x3
- Use zero-padding images
  - Add extra layers to do convolution.
- Medium filtering – This is use to reduce noisy image.
  - Our method is to find the medium base of the kernel size.
Recap

- Problem: How fast we can reduce the computation for depending on kernel and size.
Parallel Approach

- Matrix Size $2^{15} \times 2^{15}$
- Kernel Size 3x3
- Padding images
  - Each matrixes padded image received from adjacency node if necessarily.
  - Out layer of the matrix are added zero-padded images.
- Medium filtering – This is use to reduce noisy image.
  - The Kernel size is 3x3.
- Template matching – This is to find common in terms of pixels
  - Kernel size is 27x27
  - Uses Sum of Squared Difference algorithm
Things I do..

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th>N/S</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>...</td>
<td>N/S</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/S</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/s+1</td>
<td>+2</td>
<td>...</td>
<td>N/S</td>
</tr>
<tr>
<td>+2</td>
<td>3</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>N/S+1</th>
<th></th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N/S</td>
<td></td>
<td></td>
<td>N/S</td>
</tr>
<tr>
<td>N/s+1</td>
<td>+2</td>
<td>...</td>
<td>N</td>
</tr>
<tr>
<td>+2</td>
<td>3</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Continued..

- When Node0 needs the outside layer from Node1, Node1 send the first column to Node0.
- Node1 waits till Node0 complete the tasks then Node0 will send the completed column to Node1.
Corners

- Node3 Upper left corner need to wait Node1 and Node2 to send the lower row from Node1 and right column from Node3.
- Why Node3 does not need to get Node0 the corner value?
- Since Node1 and Node2 already received the corner value from Node0, Then when passing the data, we just need to take one of the value either from Node1 or Node2.
Parallel Approach cont.

- 1024x1024 Matrix
- 16 Nodes
- Each will have:
  - 256*256 matrix
  - Padded matrix will be 257*257
- 48 Message passing and receiving.
Parallel Approach cont.

- Mpi4py – Library for Python to work with MPI
- Python/A Anaconda
- What I change with my original idea:
  - Only using 3x3 kernel
  - Changing the image size to max for each node.
Results

- Matrix $2^{10} \times 2^{10}$ with $3 \times 3$ kernel

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Time(s)</th>
<th>Nodes</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>117.34</td>
<td>32</td>
<td>0.10</td>
</tr>
<tr>
<td>2</td>
<td>29.21</td>
<td>64</td>
<td>0.022</td>
</tr>
<tr>
<td>4</td>
<td>7.44</td>
<td>128</td>
<td>0.007</td>
</tr>
<tr>
<td>8</td>
<td>1.85</td>
<td>256</td>
<td>0.024</td>
</tr>
<tr>
<td>16</td>
<td>0.41</td>
<td>512</td>
<td>0.030</td>
</tr>
</tbody>
</table>
Results

* Matrix $2^{12} \times 2^{12}$ kernel 3x3

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Time(s)</th>
<th>Nodes</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2122.82</td>
<td>32</td>
<td>1.65</td>
</tr>
<tr>
<td>2</td>
<td>530.26</td>
<td>64</td>
<td>0.40</td>
</tr>
<tr>
<td>4</td>
<td>133.24</td>
<td>128</td>
<td>0.11</td>
</tr>
<tr>
<td>8</td>
<td>34.48</td>
<td>256</td>
<td>0.26</td>
</tr>
<tr>
<td>16</td>
<td>6.7</td>
<td>512</td>
<td>0.52</td>
</tr>
</tbody>
</table>
Results

- Matrix $2^{15} \times 2^{15}$ Kernel 3x3

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Time(s)</th>
<th>Nodes</th>
<th>Time(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>NAN</td>
<td>32</td>
<td>26.3</td>
</tr>
<tr>
<td>2</td>
<td>NAN</td>
<td>64</td>
<td>6.89</td>
</tr>
<tr>
<td>4</td>
<td>1810.27</td>
<td>128</td>
<td>1.78</td>
</tr>
<tr>
<td>8</td>
<td>454.05</td>
<td>256</td>
<td>0.41</td>
</tr>
<tr>
<td>16</td>
<td>105.24</td>
<td>512</td>
<td>0.96</td>
</tr>
</tbody>
</table>
Computations in Numbers

- Trial 1: $2^{10} \times 2^{10} \times 3 \times 3 = 9437184$ computations
- Trial 2: $2^{12} \times 2^{12} \times 3 \times 3 = 150994944$ computations
- Trial 3: $2^{15} \times 2^{15} \times 3 \times 3 = 9663676416$ computations

- However this does not include how long each processor waits.
Template Matching

- In this application, we want to find the correlation between in pixel given the template and the image.
- Treat it as “Where’s Waldo” book where we have he picture of Waldo’s head as the template and image is the background.
Normalized Cross-Correlation

- Normalized Cross-Correlation algorithm is used to detect similarities between the template kernel and the image.

\[
NCC = \frac{1}{mm'} \sum_{i=p}^{p+m-1} \sum_{i'=p'}^{p'+m'-1} (f_{i,i'} - \bar{f}) \cdot (g_{i+q-p,i'+q'-p'} - \bar{g}) \]

\[
= \frac{1}{mm'} \left( \sum_{i=p}^{p+m-1} \sum_{i'=p'}^{p'+m'-1} |f_{i,i'} - \bar{f}|^2 \right) \cdot \frac{1}{mm'} \left( \sum_{i=q}^{q+m-1} \sum_{i'=q'}^{q'+m'-1} |g_{i,i'} - \bar{g}|^2 \right)
\]

\[
= \frac{\bar{f}^* \bar{g} - (f)^* \bar{g}}{\sqrt{|f|^2 - |\bar{f}|^2} \sqrt{|g|^2 - |\bar{g}|^2}}
\]
Sequential Approach

- Given a template 5x5 and an Image
- Calculate the NCC value for each image patch and return the max NCC value.
- Image patch size is the same as the template size.
Results

• Template is located at $X = 55$, $Y = 455$
• Our guess is located at $X = 55$, $Y = 455$
• Ncc-value = 0.99 which is near 1 means that good correlation.

```
[0] MPI startup(): 0 18866 cpn-d07-16-01.cbls.ccr.buffalo.edu +1
('Template: (', 440, ',', 407, ')')
Template:
[[ 89. 115. 252. 120. 110.]]
[ 88. 127. 68. 211. 77.]
[129. 0. 79. 42. 75.]
[140. 77. 13. 138. 250.]
[133. 42. 85. 21. 129.]]
Guess Template:
('Job finish in ', 33.74525713920593)
```
Parallel Approach

• Each processor gets part of the input image.
• Each processor will have the same template to work on.
• Individually, compute the NCC algorithm to find the max correlation on the given image
• Detects borders and corner pixels and receive data from neighbor nodes to complete the algorithm
• Return the highest NCC value and check if the return axis matches with the template.
Results: 2 Nodes

[0] MPI startup(): 0 13978 cpn-d07-16-01.cbls.ccr.buffalo.edu
[0] MPI startup(): 1 13979 cpn-d07-16-01.cbls.ccr.buffalo.edu
('Template: ('', 261, ',', 239, ',')')
Template:
[[ 22. 227.  92. 162.   8.]
 [ 49.   5.  89.  68. 201.]
 [ 56.  95.  55.  84. 105.]
[196. 245.   3. 172.  30.]
 [ 40. 229. 131. 243.  17.]]
Hello-----118
Hello-----124
Hello-----127
Hello-----129
Guess Template:
('Rank: ', 1, 'X: ', 261, 'Y: ', 239, 'NCC-value: ', 1.0)
('Job finish in ', 16.52583122253418)
Results: 4 Nodes

[0] MPI startup(): 3  14550  cpn-d07-16-01.cbls.ccr.buffalo.edu
('Template: (', 183, ',', '', 295, ')')

Template:
[[183.  94.  218.  56.  228.]
[129.  97.  234.  76.  195.]
[195.  162.  244.  204.  143.]
[ 25.  83.  136.  255.  28.]
[101.   8.  241.  205.  150.]]

Guess Template:
('Job finish in ', 8.053385019302368)
Results: 16 Nodes

('Template: (', 167, ',', 492, ')')
Template:
[[253. 85. 89. 29. 30.]
 [226. 98. 177. 142. 56.]
 [ 87. 55.  6. 214.  90.]
[203. 248.  48. 134. 232.]
 [ 2. 171. 223. 32. 185.]]
Guess Template:
('Job finish in ', 1.6863749027252197)
Challenges

- Input data. When memory is not sufficient, creates many bugs and error.
- Running a large file, sometimes the scheduler doesn’t stop, and it will continued running on the time you set on.
- Coding with MPI library, very hard to keep track each nodes and their data. Ex. When to stop executing each node, so that they receive all the data they needed.
- Currently working on template matching, where I have a 1024x1024 matrix size, and a 27x27 template. Got stuck when, the template is on more than 4 nodes.
References

• UB CCR Support Home page.
• https://pypi.org/project/mpi4py/
Thank you!